intel®

Multimedia and Supercomputing Processors

intel®

Intel Corporation is a leading supplier of microcomputer components, modules and systems. When Intel first introduced the microprocessor in 1971, it created the era of the microcomputer. Today, Intel architectures are considered world standards. Intel products are used in a wide variety of applications including, embedded systems such as automobiles, avionics systems and telecommunications equipment, and as the CPU in personal computers, network servers and supercomputers. Others bring enhanced capabilities to systems and networks. Intel's mission is to deliver quality products through leading-edge technology.

MULTIMEDIA AND SUPERCOMPUTING PROCESSORS

1992

intel

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and may only be used to identify Intel products:

376, Above, ActionMedia, BITBUS, Code Builder, DeskWare, Digital Studio, DVI, EtherExpress, ETOX, FaxBACK, Grand Challenge, i, i287, i386, i387, i486, i487, i750, i860, i960, ICE, iLBX, Inboard, Intel, Intel287, Intel386, Intel387, Intel486, Intel487, intel inside., Intellec, iPSC, iRMX, iSBC, iSBX, iWARP, LANPrint, LANSelect, LANShell, LANSight, LANSpace, LANSpool, MAPNET, Matched, MCS, Media Mail, NetPort, NetSentry, OpenNET, PRO750, ProSolver, READY-LAN, Reference Point, RMX/80, SatisFAXtion, SnapIn 386, Storage Broker, SugarCube, The Computer Inside., TokenExpress, Visual Edge, and WYPIWYF.

MDS is an ordering code only and is not used as a product name or trademark. MDS is a registered trademark of Mohawk Data Sciences Corporation.

CHMOS and HMOS are patented processes of Intel Corp.

Intel Corporation and Intel's FASTPATH are not affiliated with Kinetics, a division of Excelan, Inc. or its FASTPATH trademark or products.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation
Literature Sales
P.O. Box 7641
Mt. Prospect, IL 60056-7641

intel

INTEL SERVICE

INTEL'S COMPLETE SUPPORT SOLUTION WORLDWIDE

Intel Service is a complete support program that provides Intel customers with hardware support, software support, customer training, and consulting services. For detailed information contact your local sales offices.

Service and support are major factors in determining the success of a product or program. For Intel this support includes an international service organization and a breadth of service programs to meet a variety of customer needs. As you might expect, Intel service is extensive. It can start with On-Site Installation and Maintenance for Intel and non-Intel systems and peripherals, Repair Services for Intel OEM Modules and Platforms, Network Operating System support for Novell NetWare and Banyan VINES software, Custom Integration Services for Intel Platforms, Customer Training, and System Engineering Consulting Services. Intel maintains service locations worldwide. So wherever you're using Intel technology, our professional staff is within close reach.

ON-SITE INSTALLATION AND MAINTENANCE

Intel's installation and maintenance services are designed to get Intel and Intel-based systems and the networks they use up and running—fast. Intel's service centers are staffed by trained and certified Customer Engineers throughout the world. Once installed, Intel is dedicated to keeping them running at maximum efficiency, while controlling costs.

REPAIR SERVICES FOR INTEL OEM MODULES AND PLATFORMS

Intel offers customers of its OEM Modules and Platforms a comprehensive set of repair services that reduce the costs of system warranty, maintenance, and ownership. Repair services include module or system testing and repair, module exchange, and spare part sales.

NETWORK OPERATING SYSTEM SUPPORT

An Intel software support contract for Novell NetWare or Banyan VINES software means unlimited access to troubleshooting expertise any time during contract hours—up to seven days per week, twenty-four hours per day. To keep networks current and compatible with the latest software versions, support services include access to minor releases and "patches" as made available by Novell and Banyan.

CUSTOM SYSTEM INTEGRATION SERVICES

Intel Custom System Integration Services enable resellers to order completely integrated systems assembled from a list of Intel386™ and Intel486™ microcomputers and validated hardware and software options. These services are designed to complement the reseller's own integration capabilities. Resellers can increase business opportunities, while controlling overhead and support costs.

CUSTOMER TRAINING

Intel offers a wide range of instructional programs covering various aspects of system design and implementation. In just three to five days a limited number of individuals learn more in a single workshop than in weeks of self-study. Covering a wide variety of topics, Intel's major course categories include: architecture and assembly language, programming and operating systems, BITBUS™, and LAN applications.

SYSTEM ENGINEERING CONSULTING

Intel provides field system engineering consulting services for any phase of your development or application effort. You can use our system engineers in a variety of ways ranging from assistance in using a new product, developing an application, personalizing training and customizing an Intel product to providing technical and management consulting. Working together, we can help you get a successful product to market in the least possible time.

intel

DATA SHEET DESIGNATIONS

Intel uses various data sheet markings to designate each phase of the document as it relates to the product. The marking appears in the upper, right-hand corner of the data sheet. The following is the definition of these markings:

Data Sheet Marking	Description
Product Preview	Contains information on products in the design phase of development. Do not finalize a design with this information. Revised information will be published when the product becomes available.
Advanced Information	Contains information on products being sampled or in the initial production phase of development.*
Preliminary	Contains preliminary information on new products in production.*
No Marking	Contains information on products in full production.*

*Specifications within these data sheets are subject to change without notice. Verify with your local Intel sales office that you have the latest data sheet before finalizing a design.

Table of Contents

Alphanumeric Index

i750™ Microprocessor Family 1

82750DB
DISPLAY PROCESSOR

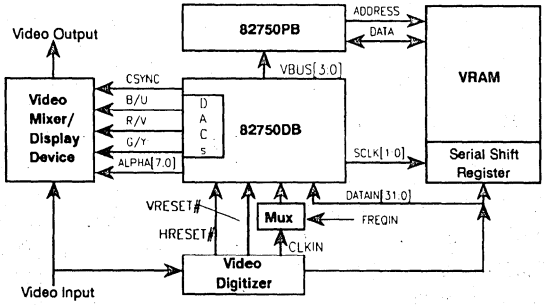
- **Programmable Video Timing**
  - 28 MHz and 45MHz Operating Frequency
  - Pixel/Line Address Range to 4096
  - Fully Programmable Sync, Equalization, and Serration Components
  - Fully Programmable Blanking and Active Display Start and Stop Times
  - Genlocking Capability
- **Flexible Display Characteristics**
  - 8-, Pseudo 16-, 16-, and 32-Bit/Pixel Modes
  - Selectable Pixel Widths of 1.0, 1.5, 2.0, 2.5, through 14 Periods of the Input Frequency
  - Support Popular Display Resolutions: VGA, XGA, NTSC, PAL, and SECAM
  - On-Chip Triple DAC for Analog RGB/YUV Output
  - Mix Graphics and Video Images on a Pixel by Pixel Basis
  - Real Time Expansion of the Reduced Sample Density Video Color Components (U, V) to Full Resolution
  - Three Independently Addressable Color Palettes
  - Programmable 2X Horizontal Interpolation of Y Channel
  - 16 x 16 x 2-Bit Cursor Map with Independently Programmable 2X Expansion Factors in X and Y Dimensions
  - YUV to RGB Color Space Conversion
  - 2X Vertical Replication of Y, U, and V Data for Displaying Full Motion Video on VGA Monitor
  - Register and Function Compatible with the 82750DA

1

Intel's 82750DB is a custom designed VLSI chip used for processing and displaying video graphic information. It is register and function compatible with the 82750DA.

Reset inputs allow the 82750DB to be genlocked to an external sync source. By programming internal control registers, this sync can be modified to accommodate a wide variety of scanning frequencies. A large selection of bits/pixel, pixels/line, and pixel widths are programmable, allowing a wide latitude in trading-off image quality vs update rate and VRAM requirements.

The 82750DB can operate in a digitizing mode, wherein it generates timing and control signals to the 82750PB and VRAM, but does not output display information. Besides digitizer support signals and video synchronization, the 82750DB outputs digital and analog RGB or YUV information and an 8-bit digital word of alpha data. This alpha channel data may be used to obtain a fractional mix of 82750DB outputs with another video source.



240855-1

82750DB Subsystem Diagram

Order Number: 240855-003

82750DB Display Processor

CONTENTS

CONTENTS

FIGURES

CONTENTS

CONTENTS

PAGE

TABLES

1.0 82750DB PIN DESCRIPTION

Pinout

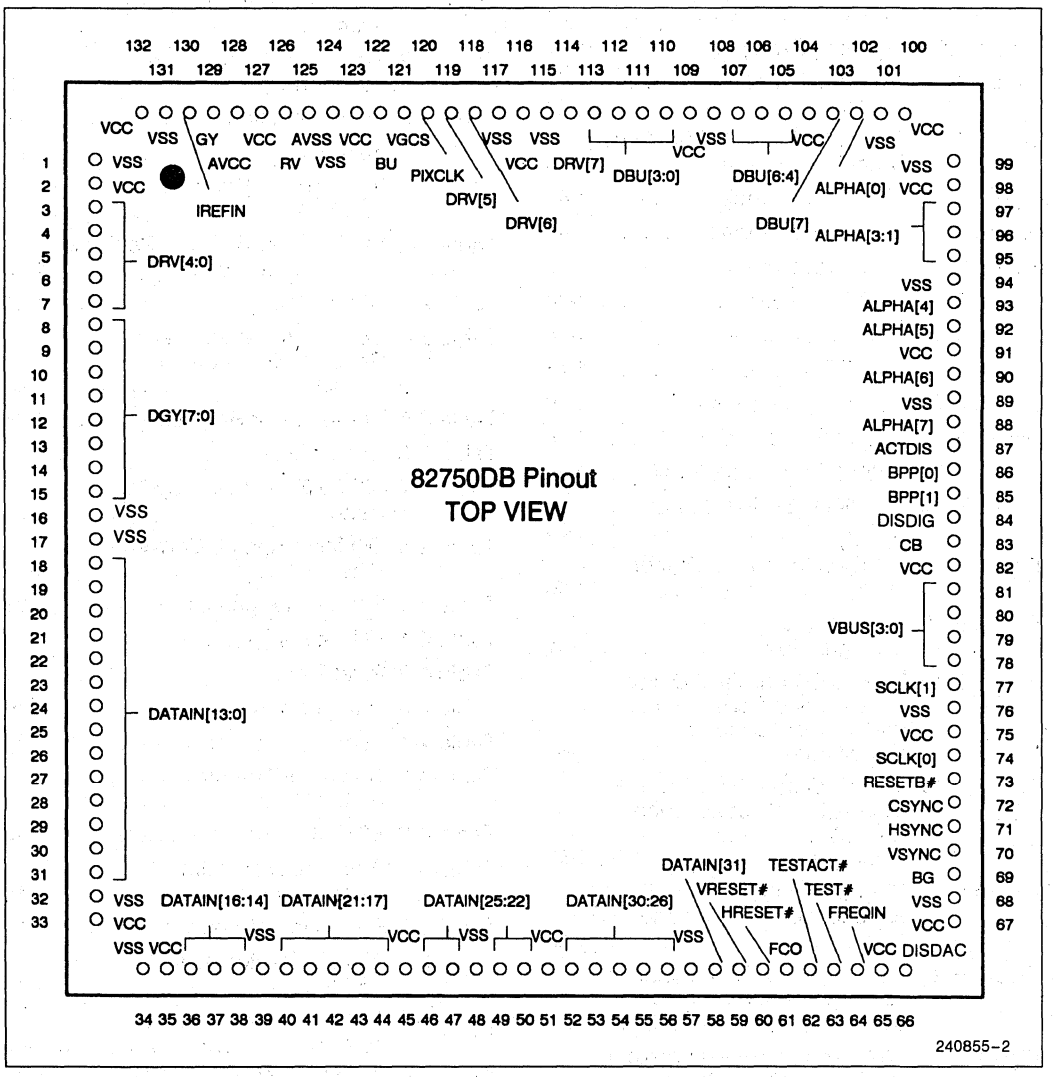


Figure 1-1. 82750DB Pinout

Table 1-1. Pin Cross Reference by Pin Name

Pin Name	Location
ACTDIS	87
ALPHA[7]	88
ALPHA[6]	90
ALPHA[5]	92
ALPHA[4]	93
ALPHA[3]	95
ALPHA[2]	96
ALPHA[1]	97
ALPHA[0]	102
AVCC	128
AVSS	125
BG	69
BPP[1]	85
BPP[0]	86
BU	122
CB	83
CSYNC	72
DATAIN[31]	58
DATAIN[30]	56
DATAIN[29]	55
DATAIN[28]	54
DATAIN[27]	53
DATAIN[26]	52
DATAIN[25]	50
DATAIN[24]	49
DATAIN[23]	47
DATAIN[22]	46
DATAIN[21]	44
DATAIN[20]	43
DATAIN[19]	42
DATAIN[18]	41
DATAIN[17]	40
DATAIN[16]	38

Pin Name	Location
DATAIN[15]	37
DATAIN[14]	36
DATAIN[13]	31
DATAIN[12]	30
DATAIN[11]	29
DATAIN[10]	28
DATAIN[9]	27
DATAIN[8]	26
DATAIN[7]	25
DATAIN[6]	24
DATAIN[5]	23
DATAIN[4]	22
DATAIN[3]	21
DATAIN[2]	20
DATAIN[1]	19
DATAIN[0]	18
DBU[7]	103
DBU[6]	105
DBU[5]	106
DBU[4]	107
DBU[3]	110
DBU[2]	111
DBU[1]	112
DBU[0]	113
DGY[7]	8
DGY[6]	9
DGY[5]	10
DGY[4]	11
DGY[3]	12
DGY[2]	13
DGY[1]	14
DGY[0]	15
DISDAC	66
DISDIG	84

Pin Name	Location
DRV[7]	114
DRV[6]	118
DRV[5]	119
DRV[4]	3
DRV[3]	4
DRV[2]	5
DRV[1]	6
DRV[0]	7
FCO	61
FREQIN	64
GY	129
HRESET#	60
HYSNC	71
IREFIN	130
PIXCLK	120
RESETB#	73
RV	126
SCLK[1]	77
SCLK[0]	74
TEST#	63
TESTACT#	62
VBUS[3]	81
VBUS[2]	80
VBUS[1]	79
VBUS[0]	78
V_{CC}	2
V_{CC}	33
V_{CC}	35
V_{CC}	45
V_{CC}	51
V_{CC}	65
V_{CC}	67
V_{CC}	75

Pin Name	Location
V_{CC}	82
V_{CC}	91
V_{CC}	98
V_{CC}	100
V_{CC}	104
V_{CC}	109
V_{CC}	116
V_{CC}	123
V_{CC}	127
V_{CC}	132
VGCS	121
VRESET#	59
V_{SS}	1
V_{SS}	16
V_{SS}	17
V_{SS}	32
V_{SS}	34
V_{SS}	39
V_{SS}	48
V_{SS}	57
V_{SS}	66
V_{SS}	68
V_{SS}	76
V_{SS}	89
V_{SS}	94
V_{SS}	99
V_{SS}	101
V_{SS}	108
V_{SS}	115
V_{SS}	117
V_{SS}	124
V_{SS}	131
VSYNC	70

Table 1-2. Pin Cross Reference by Location

Location	Pin Name	Location	Pin Name	Location	Pin Name	Location	Pin Name
1	V_{SS}	34	V_{SS}	67	V_{CC}	100	V_{CC}
2	V_{CC}	35	V_{CC}	68	V_{SS}	101	V_{SS}
3	DRV[4]	36	DATAIN[14]	69	BG	102	ALPHA[0]
4	DRV[3]	37	DATAIN[15]	70	VSYNC	103	DBU[7]
5	DRV[2]	38	DATAIN[16]	71	HSYNC	104	V_{CC}
6	DRV[1]	39	V_{SS}	72	CSYNC	105	DBU[6]
7	DRV[0]	40	DATAIN[17]	73	RESETB#	106	DBU[5]
8	DGY[7]	41	DATAIN[18]	74	SCLK[0]	107	DBU[4]
9	DGY[6]	42	DATAIN[19]	75	V_{CC}	108	V_{SS}
10	DGY[5]	43	DATAIN[20]	76	V_{SS}	109	V_{CC}
11	DGY[4]	44	DATAIN[21]	77	SCLK[1]	110	DBU[3]
12	DGY[3]	45	V_{CC}	78	VBUS[0]	111	DBU[2]
13	DGY[2]	46	DATAIN[22]	79	VBUS[1]	112	DBU[1]
14	DGY[1]	47	DATAIN[23]	80	VBUS[2]	113	DBU[0]
15	DGY[0]	48	V_{SS}	81	VBUS[3]	114	DRV[7]
16	V_{SS}	49	DATAIN[24]	82	V_{CC}	115	V_{SS}
17	V_{SS}	50	DATAIN[25]	83	CB	116	V_{CC}
18	DATAIN[0]	51	V_{CC}	84	DISDIG	117	V_{SS}
19	DATAIN[1]	52	DATAIN[26]	85	BPP[1]	118	DRV[6]
20	DATAIN[2]	53	DATAIN[27]	86	BPP[0]	119	DRV[5]
21	DATAIN[3]	54	DATAIN[28]	87	ACTDIS	120	PIXCLK
22	DATAIN[4]	55	DATAIN[29]	88	ALPHA[7]	121	VGCS
23	DATAIN[5]	56	DATAIN[30]	89	V_{SS}	122	BU
24	DATAIN[6]	57	V_{SS}	90	ALPHA[6]	123	V_{CC}
25	DATAIN[7]	58	DATAIN[31]	91	V_{CC}	124	V_{SS}
26	DATAIN[8]	59	VRESET#	92	ALPHA[5]	125	AV_{SS}
27	DATAIN[9]	60	HRESET#	93	ALPHA[4]	126	RV
28	DATAIN[10]	61	FCO	94	V_{SS}	127	V_{CC}
29	DATAIN[11]	62	TESTACT#	95	ALPHA[3]	128	AV_{CC}
30	DATAIN[12]	63	TEST#	96	ALPHA[2]	129	GY
31	DATAIN[13]	64	FREQIN	97	ALPHA[1]	130	IREFIN
32	V_{SS}	65	V_{CC}	98	V_{CC}	131	V_{SS}
33	V_{CC}	66	DISDAC	99	V_{SS}	132	V_{CC}

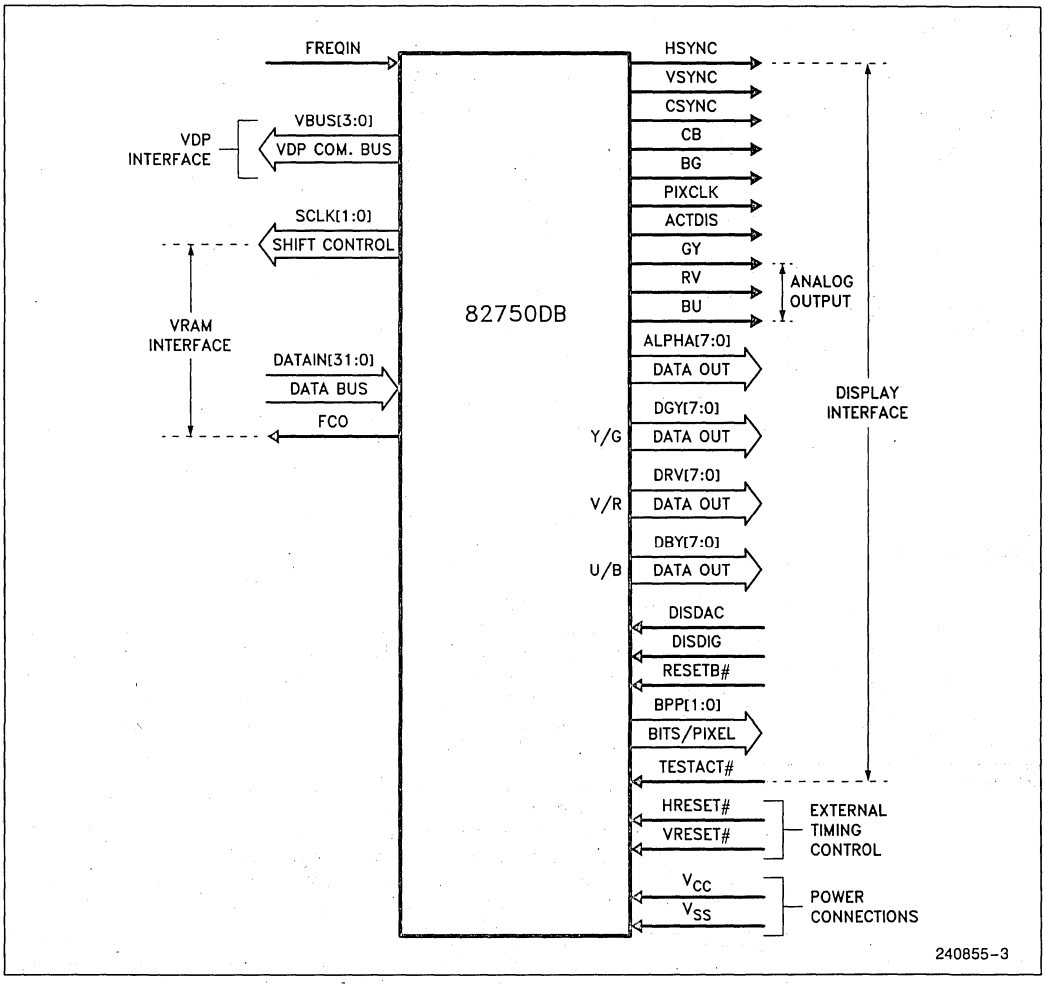


Figure 1-2. 82750DB Functional Signal Groupings

Quick Pin Reference

Table 1-3. Pin Descriptions

Symbol	Type	Name and Function
FREQIN	I	**FREQUENCY INPUT CLOCK:** In normal use, the 82750DB supplies refresh timing for an associated VRAM through the 82750PB. This places a lower limit on the line frequency, which is a programmed multiple of FREQIN. It must generate enough refresh cycles, so a minimum line rate of 4 kHz is required. Furthermore, the 82750PB may run no less than 1/8 the speed of the 82750DB, since the 82750PB samples the timing and control signals generated by the 82750DB. The period of FREQIN is known as a "T" cycle.
RESETB#	I	**EXTERNAL RESET:** Input signal which places all units in the 82750DB into an initialized state, and sets the transfer rate to a default value of 1/3X the operating frequency. It is an edge sensitive iniput which must be held low for a minimum of ten T-cycles. The slowest transfer rate is selected to ensure that the 82750DB will read the register information correctly during the first register transfer, independent of the speed of the VRAMs. During the reset state, the analog video outputs and digital outputs are set to the black level. This will occur a maximum of four cycles after RESETB# is set to a zero. This signal is also used in conjunction with the TESTACT# input to disable outputs.
VRESET#	I	**VERTICAL RESET:** By programming a bit in an internal register, the 82750DB may be placed in the Genlock mode. If this mode is selected, assertion of VRESET# resets all vertical timing to the first line of the next field. It does not affect the horizontal timing, but does generate the on-chip end of field signals. It is an edge sensitive input that is sampled in the 82750DB at the internal time corresponding to the rising edge of FREQIN. If the Genlock mode has not been enabled, this signal will have no effect on the sync timing. The 82750DB will then operate in a free-running mode. Refer to Chapter 3 for a detailed description of genlocking the 82750DB.
HRESET#	I	**HORIZONTAL RESET:** When in the Genlock mode, this input will reset all of the horizontal timing to the start of the line (beginning of horizontal sync). HRESET# does not affect vertical timing (except for an up-to one-line delay) or any other 82750DB registers. This signal is an edge sensitive input that is sampled in the 82750DB at that internal time corresponding to the rising edge of FREQIN. As was the case with the VRESET# signal, this input will be ignored when not in the Genlock mode.
VBUS[3:0]	O	**VDP COMMUNICATION BUS:** The 82750DB outputs status and VRAM transfer requests over these lines to the 82750PB, for 2 to 16 T-cycles (as programmed by the user). Transfer requests can tie up the 82750DB/VRAM, 82750PB/VRAM, or 82750PB/82750DB (VBUS) interfaces for a longer period due to VRAM arbitration. When signals are not being sent out, the VBUS has value 1111, the "null command."
SCLK[1:0]	O	**VRAM SHIFT CLOCKS:** Transfer requests to the 82750PB cause a VRAM address to be set up, and the VRAM serial registers loaded (in the case of displaying) or unloaded (in the case of digitizing). These signals are used to shift data out of and into the VRAMs. Both signals are identical, and run at a maximum rate of 1X of the pixel frequency, except during transfer requests, at which time they run at 1X, 1/2X, or 1/3X of the operating frequency of the 82750DB, as programmed by the user.
DATAIN[31:0]	I	**DATA INPUT BUS:** This is the input data clocked in from VRAM by the SCLK[1:0] signals. The format of the input data is a function of the programmed number of bits/pixel and of the type of transfer cycle being executed. Data will be sampled internally on the rising edge of FREQIN.

Table 1-3. Pin Descriptions (Continued)

Symbol	Type	Name and Function
FCO	O	**FRAME CAPTURE ON:** This is the output signal which indicates to the digitizer that the VRAM serial port has been turned from read mode to write mode. The digitizer may then drive the (common) VRAM serial register data I/O pins. FCO will be asserted after the programmer specifies digitization, five lines after the start of the active vertical display, at the time of HSYNC. This gives the external logic time to switch directions of the VRAM serial data bus. This signal will end four lines after vertical active stops, at the next HSYNC, to make sure the digitizer is off before the next beginning-of-field register transfer.
HSYNC	O	**HORIZONTAL SYNCHRONIZATION:** Video synchronization signal which is asserted at the beginning of every line and ends a programmed time later. (The duration of this signal is specified in T-cycles.)
VSYNC	O	**VERTICAL SYNCHRONIZATION:** Video synchronization signal which can be programmed to start (once) and end (once) in every field. (The start and stop position may be specified in half-line units.)
CSYNC	O	**COMPOSITE SYNCHRONIZATION PULSE:** This contains the programmed vertical serration and equalization information, as well as horizonal synchronization pulses.
CB	O	**COMPOSITE BLANKING:** This signal can be programmed to end once and start once in each line, and end once and start once every field.
BG	O	**BURST GATE:** This signal starts and stops at user-programmable horizontal positions in each line, in a programmable vertical group of lines. The primary use of this signal is to provide a "window" during which the BURST output should be inserted to generate a baseband NTSC signal. The output frequency is set by an integer divisor (0–31) and the rate of the FREQIN clock input. To use this effectively, the 82750DB must operate at an integer multiple of the NTSC 3.58 MHz color subcarrier. The number is programmed in two's complement form in the General Control register.
PIXCLK	O	**PIXEL CLOCK:** This output signals valid data on the DGY, DRV, DBU, GY, RV, and BU lines. PIXCLK becomes active one-half of a T-cycle after valid data appears on DGY, DRV, or DBU, and coincident with GY, RV, and BU. During active display time it is issued at a steady rate of 1/(T-cycles/pixel) times per T-cycle, and otherwise at a steady rate of once per T-cycle. Its duration is one-half of a T-cycle, and its rising edge may synchronize with either rising or falling edges of FREQIN depending on the pixel frequency. This signal may be used to synchronize off-chip processing of the pixel data outputs.
GY, RV, BU	O	**ANALOG PIXEL OUTPUTS:** These signals are the processed pixel data from the 82750DB in analog form. During the display, these signals may be programmed to output pixel data in either YUV or RGB format.

Output Format	DGY	DRV	DBU
YUV	Y	V	U
RGB	G	R	B

DGY[7:0], DRV[7:0], DBU[7:0]	O	**DIGITAL VIDEO OUTPUTS:** These are the digital outputs of the GY, RV, and BU channels, respectively. They are valid with respect to the rising edge of PIXCLK.
ALPHA[7:0]	O	**ALPHA CHANNEL:** These 8 bits are used to output a digital value for mixing the 82750DB output with another video signal off-chip. The alpha channel information may be included in the pixel data, or may be output based on a comparison of the pixel data with user-programmed values.
ACTDIS	O	**ACTIVE DISPLAY:** This is the active portion of the display as programmed by the user. It is delayed by the pipeline through the 82750DB, which is 5 lines vertically and a variable number horizontally, depending on the display mode.

Table 1-3. Pin Descriptions (Continued)

<table>
<tr><th>Symbol</th><th>Type</th><th>Name and Function</th></tr>
<tr><td>BPP[1:0]</td><td>O</td><td>BITS PER PIXEL: During the nonactive display, the user programmed bits/pixel is encoded on these lines. During active display, the BPP[0] signal is multiplexed with a signal, Cursor Active, which indicates if the cursor data is currently active (non-transparent). When the Cursor Active output signal is asserted, this indicates that cursor overlay data is currently being output. Also during the active display, the BPP[1] signal is multiplexed with a signal, VUGR, which indicates whether the 82750DB is operating in a graphics or video mode. When the VUGR output signal is asserted, this indicates the G, R, and B outputs are derived from the subsampled VU data. These pins allow users to latch the BPP[1:0] signals during nonactive display time (as indicated by ACTDIS being zero) for post-processing of the 82750DB output. The active cursor window on BPP[0] can be used during active display, to multiplex in other video streams into the output display. The following table illustrates the encoding on the BPP signals.
<table>
<tr><th>Bits/Pixel</th><th>ACTDIS</th><th>BPP[0]</th><th>BPP[1]</th></tr>
<tr><td>8</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>16</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>32</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>pseudo 16</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>8</td><td>1</td><td>Cursor Active</td><td>VUGR</td></tr>
<tr><td>16</td><td>1</td><td>Cursor Active</td><td>VUGR</td></tr>
<tr><td>32</td><td>1</td><td>Cursor Active</td><td>VUGR</td></tr>
<tr><td>pseudo 16</td><td>1</td><td>Cursor Active</td><td>VUGR</td></tr>
</table></td></tr>
<tr><td>DISDAC</td><td>I</td><td>DISABLE ANALOG OUTPUTS: When this input is active, the Analog Pixel Outputs are set to a high-impedance state.</td></tr>
<tr><td>DISDIG</td><td>I</td><td>DISABLE DIGITAL OUTPUTS: When this input is active, the digital outputs of the 82750DB will be set to zero. In applications that use only the analog outputs of the 82750DB, the digital outputs must be disabled.</td></tr>
<tr><td>TESTACT#</td><td>I</td><td>TEST ACTIVE: Active low signal that is used in conjunction with the RESETB# signal to allow the chip to perform one of the following functions:
<table>
<tr><th>RESETB#</th><th>TESTACT#</th><th>82750DB State</th></tr>
<tr><td>0</td><td>1</td><td>Enter Reset State</td></tr>
<tr><td>0</td><td>0</td><td>Enter Reset State
Tristate All Outputs
Analog Outputs are Zero</td></tr>
<tr><td>1</td><td>1</td><td>Normal Operation</td></tr>
<tr><td>1</td><td>0</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>TEST#</td><td>I</td><td>TEST INPUT: This signal must be set to VCC to guarantee correct chip operation.</td></tr>
<tr><td>VGCS</td><td>O</td><td>INTERNAL VOLTAGE REFERENCE: This signal must be decoupled to AVCC.</td></tr>
<tr><td>IREFIN</td><td>I</td><td>ANALOG CURRENT REFERENCE: Under normal operation, this signal should be tied to a temperature compensated current reference to AVSS. This signal must be decoupled to AVCC.</td></tr>
<tr><td>AV_{CC}</td><td>I</td><td>ANALOG POWER pin provides +5 V_{DC} supply to the Digital to Analog Converter.</td></tr>
<tr><td>AV_{SS}</td><td>I</td><td>ANALOG GROUND pin provides the 0V connection to which the analog outputs are referenced. This must be connected to VSS.</td></tr>
<tr><td>V_{CC}</td><td>I</td><td>POWER pins provide +5 V_{DC} supply input.</td></tr>
<tr><td>V_{SS}</td><td>I</td><td>GROUND pins provide the 0V connection to which all inputs and outputs are referenced.</td></tr>
</table>

Table 1-4. Input Pins

Name	Active Level	Synchronous/ Asynchronous
FREQIN	HIGH	Synchronous
RESETB#	LOW	Asynchronous
VRESET#	LOW	Asynchronous
HRESET#	LOW	Asynchronous
DISDIG	HIGH	Asynchronous
TESTACT#	LOW	Asynchronous
TEST#	LOW	ASynchronous

All output pins have an active level of HIGH, and are floated when RESETB# and TESTACT# are set to a zero. The exceptions are GY, RV, and BU which will be forced to a zero level.

2.0 ARCHITECTURE

Overview

There are 10 units in the 82750DB. Each of the units operates independently at the maximum clock rate input to the chip. The control information for each block is distributed in programmable registers throughout the chip. These registers are loaded on user-specified lines during the horizontal and vertical blanking intervals of the field. The register data that was read in from VRAM is passed from block to block during the blanking intervals of the display, on the same lines that the pixel information is passed during the active display. The Functional Block Diagram is shown in Figure 2-1.

In order to maximize speed and compensate for processing delays, the chip is heavily pipelined. All inter-block information is delay-equalized to accommodate the different pipeline lengths in each module. As a result, the total pipeline delay is dependent on the number of processing units that are used to generate the display. Chapter 4 describes how the user programming is affected by these pipeline delays.

Each of the units are described in more detail in the following sections of this chapter.

Sync Generation and Timing

The sync generation and timing block generates all of the internal timing and control signals, as well as the video synchronization signals. Sync and timing information may be derived from two sources: from the master clock, in which case the control registers on the 82750DB are programmed to provide the desired display frequency in terms of periods of the master clock (T-cycles), or from the horizontal and vertical external reset signals. (The latter is known as the genlock mode.) Characteristics such as line rate, blanking and border intervals, and composite synchronization parameters can be independently set. Since the 82750DB can be reprogrammed once each line, horizontal strips of different resolutions can be supported on the same display. However, the horizontal strips that can be supported are limited by the host processor's response to redefining the bitmap pointers resident on the 82750PB.

The horizontal and vertical display parameters are fully programmable. Figure 2-2 illustrates the horizontal programming parameters. The line starts at the programmed start position, with the length of half of a line programmed in T-cycles. The length of the total line is twice the half-line length. Parameters such as horizontal sync start, horizontal sync width, horizontal blanking start and stop, and horizontal active start and stop are all specified by the user. Note that the border time is not explicitly programmed, but is defined as the region of the display line where neither active display nor blanking is programmed to occur. In order for the 82750DB to function correctly, the width of the horizontal active display should be programmed such that the end of the horizontal active display coincides with the end of the last displayed pixel.

Figure 2-3 shows the vertical programming parameters. The basic unit for vertical programming is in units of half lines, with the half-line count for each field starting at zero. Where appropriate for a parameter, the count is programmed in units of full lines. The length of the complete field is programmed in half lines, which makes it convenient for distinguishing between interlaced and non-interlaced displays. (For interlaced displays, the number of half lines is odd, for non-interlaced displays, it is even.) The vertical active and blanking regions may be independently programmed, with the border time defined as the region where blanking and active display is not on.

NOTE:

Sync parameters are completely independent of the display parameters. This allows the sync signals to be positioned anywhere in the field (even during active display).

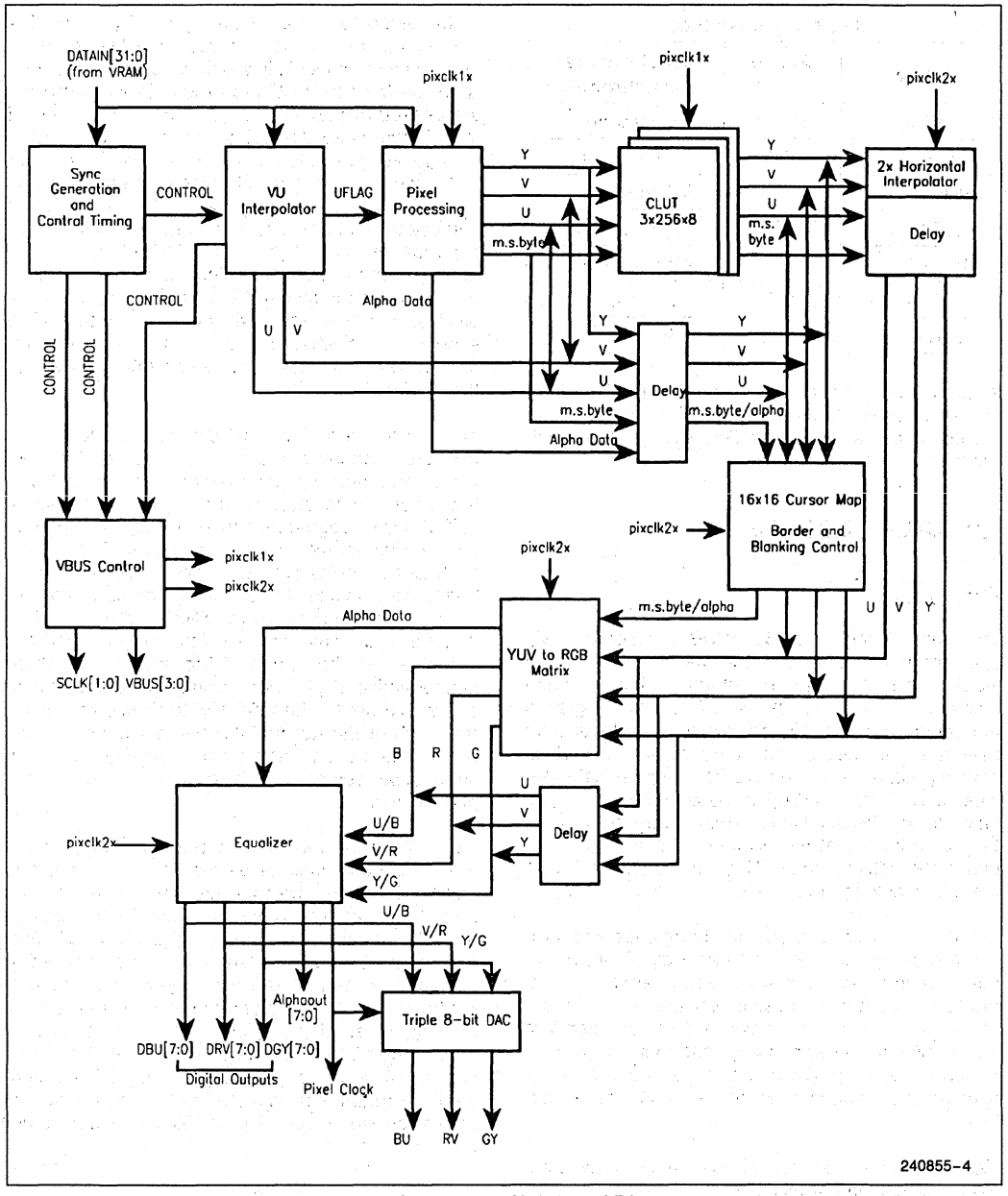
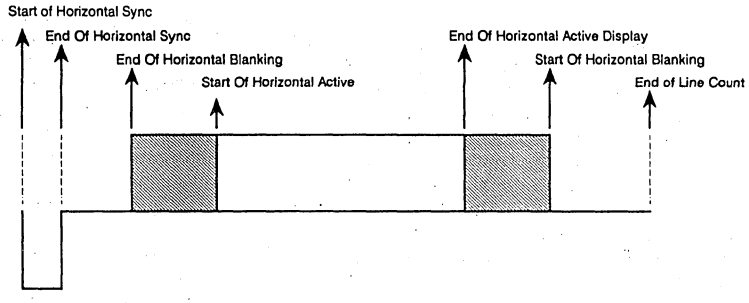


Figure 2-1. 82750DB Unit Level Diagram

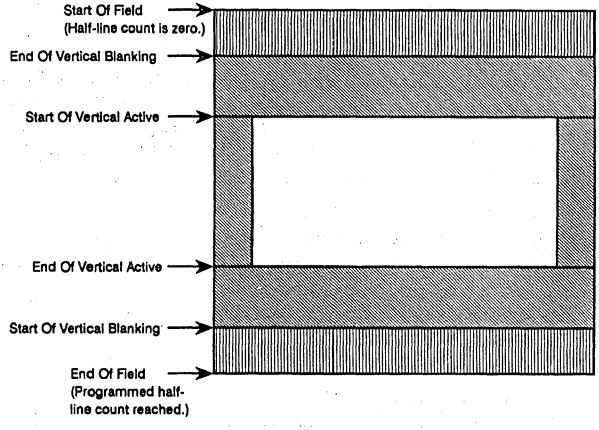


- All horizontal programming parameters are in periods of the master clock.
- Border may be eliminated by programming the blanking time to abut the active display.
- Pixel widths must be an integer divisor of horizontal active width.

Border Time Active Display

240855-5

Figure 2-2. Horizontal Programming Parameters



- All vertical programming parameters shown are in half lines.
- Vertical border may be eliminated by abutting active display and blanking times.
- Positioning of vertical blanking and sync are fully independent.

Programming Blanking Border Time Active Display

240855-6

Figure 2-3. Vertical Programming Parameters

VBUS Control

The VBUS controller sends all 82750DB requests for display bitmaps, VRAM refresh, and synchronization information to the 82750PB, at programmable times during a field. Transfer requests are scheduled to occur on a line basis, so only their vertical position (or line) is specified by the user. Other commands, like refresh requests, occur every line, and their horizontal position (or dot position) in the line must be specified by the user. Transfer requests are given the highest priority by the VBUS control circuit and are performed first during a blanking interval. The programmer has the responsibility of scheduling the line oriented codes, like refresh, so that they do not collide with the transfer requests.

Besides arbitrating the scheduled transfer requests, the VBUS controller also reads the data from the VRAM shift registers using the two shift clock outputs (SCLK[1:0]). The code corresponding to the type of data to be read is asserted for a programmable number of cycles on the 4-bit VBUS. The 82750DB then waits a programmable delay before reading the data from the VRAM. This delay should be long enough to guarantee that the 82750PB has completed loading the information into the serial shift register of the VRAM. Both signals are off while the code causing the transfer cycle is active on the VBUS, as well as during the read delay time. Figure 2-4 illustrates this communication between the 82750PB and the 82750DB.

When the delay wait is over, the shift clock outputs are activated. The SCLK[1:0] signals' behavior is dependent on the transfer rate that the user has selected—either 1X, 1/2X, or 1/3X the operating frequency. Note that if the RESETB# signal is applied, the transfer rate is automatically set to 1/3X during the first automatic register transfer, regardless of the state of the transfer rate selection. The transfer rate may be changed in the first register transfer after RESETB# is set to a logic one value.

Figure 2-5 illustrates how the SCLKs operate in the 1X mode in a system. SCLK[1:0] signals will toggle between zero and one on the rising edge of FREQIN, after an internal logic delay. The data is read into the 82750DB on the rising edge of the internal clock, one 82750DB clock cycle after the SCLK outputs are asserted. Since there are 32 data input pins, each SCLK can read in the serial data from eight 256 x 4 VRAM memory devices. Adding external buffering to the SCLKs (to drive more memory) will also add delay to the memory access. The delay increase may require more than one T-cycle before the VRAM data is valid. In this case, the time between the rising edge of the internal 82750DB clock that generates the SCLKs and the edge that latches the data must be increased.

There are two solutions, the operating frequency of 82750DB can be lowered to accommodate a longer T-cycle, or the 1/2X SCLK mode may be selected (as shown in Figure 2-6). When using the 1/2X transfer rate, the data is read into the 82750DB on the rising edge of the internal clock, two 82750DB clock cycles after the SCLK outputs are asserted.

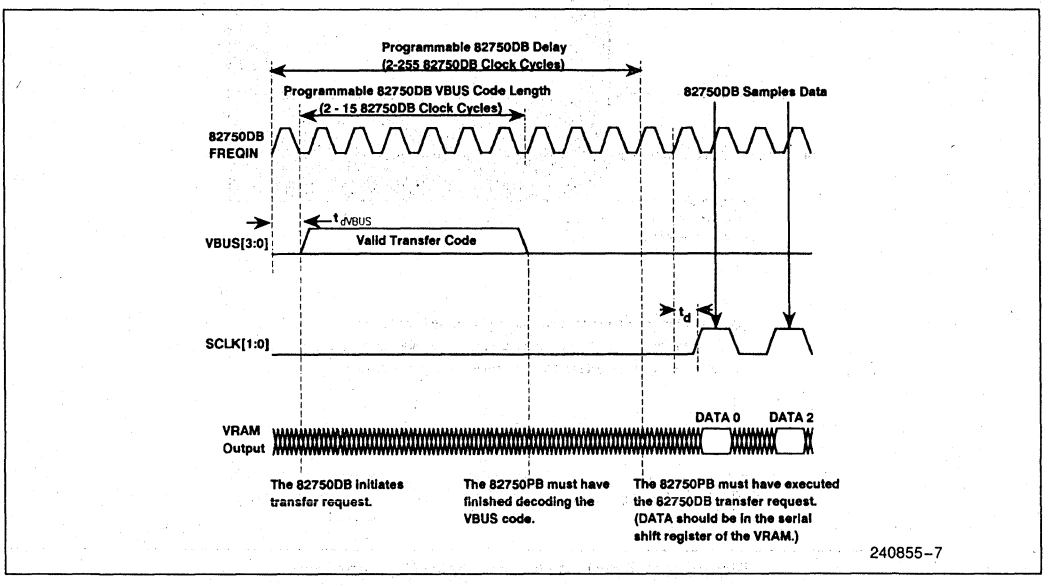


Figure 2-4. 82750PB/82750DB Communication

Figure 2-7 illustrates 1/3X (default) shift clock operation that is used during the RESET mode or may be programmed by the user. The first word of data is latched by the 82750DB on the rising ede of the FREQIN that is three T-cycles after the SCLK outputs were asserted. This allows three full 82750DB cycles for the VRAMs to output valid data, which gives extra margin for applications that need longer shift read cycles (due to slower memories or external logic delays) and do not wish to operate the 82750DB at a slower speed.

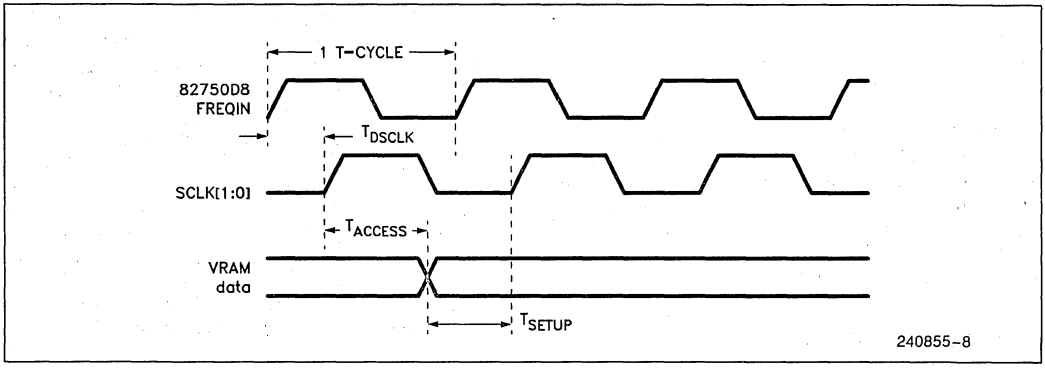


Figure 2-5. 82750DB 1X Shift Clock Operation

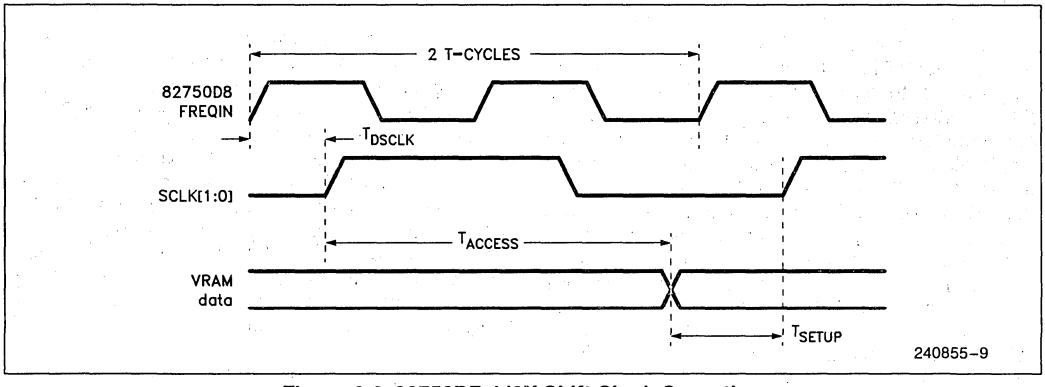


Figure 2-6. 82750DB 1/2X Shift Clock Operation

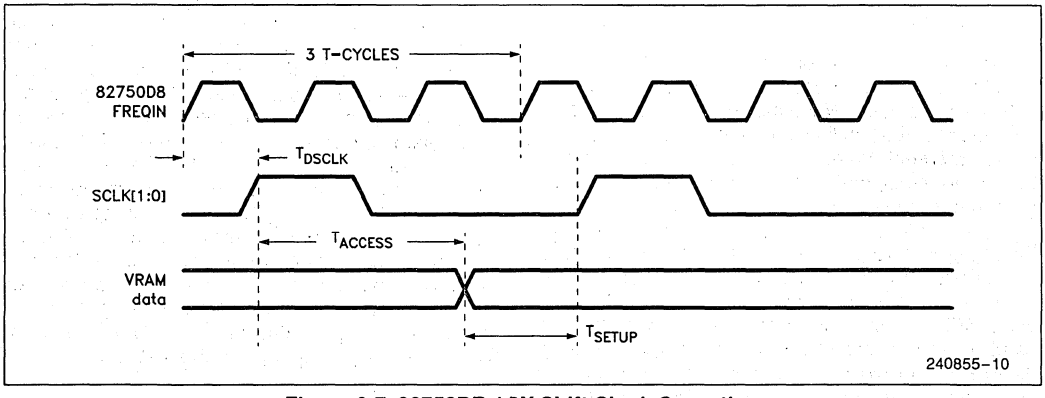


Figure 2-7. 82750DB 1/3X Shift Clock Operation

When reading data from memory during active display, the SCLK[1:0] outputs operate at a rate required to support the programmed display rate. This rate is determined from the following equation:

$$\text{RATE} = \frac{(\#\text{ of bits/pixel})}{(\text{32-bit/word}) * (\#\text{ word/fetch}) * (\#\text{T-cycle/pixel})}$$

where: # bits/pixel and # T-cycles/pixel are user-programmed

word/fetch is: 1

The SCLK[1:0] outputs will be the same frequency as the input clock in the 1X shift clock mode, and one half the input clock frequency when using the 1/2X mode. The frequency will be one third in the input clock when using the 1/3X mode. In the 1/3X mode the SCLK[1:0] outputs will be high for one T-cycle, and low for 2 T-cycles.

VBUS CODE DESCRIPTION

When the 82750DB is actively fetching and displaying pixels, VUXFER, BMX/YBMNPX, and REGX are typically sent over the VBUS. Of the three codes, REGX has top priority, followed by VUXFER, and last by BMX/YBMNPX. These commands may be programmed to occur each active line during the blanking interval for the line just completed. If a register transfer has been programmed for an active line, it takes priority and is executed first. Otherwise, immediately after the register transfer, any scheduled VUXFER and BMX/YBMNPX commands are executed. The programmer has the responsibility for verifying that the sum of times required by these commands does not exceed horizontal non-active display time. The 82750DB will commence fetching pixels at the subsequent start of active display. A detailed explanation of the different types of VBUS commands and their corresponding codes follows.

Transfer Requests

The following commands request the 82750PB to transfer information from the VRAM array into the VRAM shift register. When multiple requests are programmed for a given line, they are listed in the priority they are sent. When asserting a transfer request, the programmer must be aware of two other programmed parameters, VBLEN and SCLK delay.

The VBLEN parameter is a user programmed value whose bits lie in the General Control Register. It is the length of time, in 82750DB T-cycles, that a particular VBUS code will be held at the outputs. It is used to ensure that the asynchronously operating 82750PB chip will have enough time to recognize and begin operating on an 82750DB transfer request.

The other parameter the programmer needs to set is the SCLK delay. This can be found in the Pixel Control Register. It is the number of 82750DB clock cycles that the DB will wait before clocking in data, out of the VRAM, after the initiation of a transfer request on the VBUS outputs.

REGX (0010) This command requests that the 82750PB transfer 82750DB register information into the VRAM shift registers. Besides the automatic 82750DB register transfer that occurs on the second line (line 2) of each field, the programmer can specify the next horizontal line on which another register transfer is to take place. The transfers may be scheduled many times during the field. On the first transfer, the 82750PB uses the contents of its 82750DBc register as the starting address of the 82750DB register data. On each subsequent access, the programmed pitch value in 82750PB's 82750DBc-PITCH register is added to the accumulated start address. The programmer must ensure that the data is stored in VRAM at the correct address. Since the pitch remains constant, the longest register load will determine the pitch value.

The VBUS unit performs a vertical checksum on all the register information. Each bit in the register word undergoes an exclusive-OR with the corresponding bit in the previous data word. The 82750DB compares this information with the user generated checksum, which is the last 32-bit data word read into the 82750DB during a register transfer. If the values do not match, the 82750DB will disable all of its digital sync and data outputs, enter the reset state, and send a SHUTDOWN code (82750DBSD) to the 82750PB over the VBUS[3:0] outputs. If the new checksum is correct, the new register values will take effect immediately.

VUXFER (0001) This code is used to request VU data, providing new VU data is required by the 82750DB. This command is issued only on vertically active lines (as programmed in the register, not as seen on the screen) and possibly the four lines after. On each line, a row of V and/or U samples are loaded into the VU interpolator line stores. The pattern of requests depends upon the mode in which the VU interpolator is operating. In the interlaced VU mode, one line of samples for both the V and U components are fetched during each transfer; in the non-interlaced VU mode, only one line of samples for either the V or U components is fetched. Table 2-1 illustrates the pattern of requests. M is the programmed first vertical active line, and N the last active line. The modes listed have VU transfer requests following the end of horizontal active of the lines specified, stopping with the last line, N + 4.

Table 2-1. VU Transfer Request Patterns

Mode	Active Line	Request VU Data
2x Non-Interlaced	M	Fetch 1st Line of V
	M + 1	Fetch 1st Line of U
	M + 4	Fetch 2nd Line of V
	M + 5	Fetch 2nd Line of U
	N + 4	Fetch Last Line of V
2x Interlaced (Odd and Even Fields)	M	Fetch 1st Line of V and U
	M + 4	Fetch 2nd Line of V and U
	M + 5	Fetch 3rd Line of V and U
	N + 4	Fetch Last Line of V and U
4x Non-Interlaced	M	Fetch 1st Line of V
	M + 1	Fetch 1st Line of U
	M + 4	Fetch 2nd Line of V
	M + 5	Fetch 2nd Line of U
	M + 8	Fetch 3rd Line of V
	N + 4	Fetch Last Line of V
4x Interlaced (Odd and Even Fields)	M	Fetch 1st Line of V and U
	M + 4	Fetch 2nd Line of V and U
	M + 6	Fetch 3rd Line of V and U
	N + 4	Fetch Last Line of V and U

The 82750PB uses another internal pointer to cause the VRAM to load the desired VU data into its shift registers (incrementing the pointer by a pitch value). This command is asserted for a programmable number of T-cycles (m), as specified in the Miscellaneous Control register. Then, the 82750DB fetches them, tying up the 82750DB/VRAM interface for (n + 2) cycles, where n is ¼ the programmable total number of 8-bit samples of V and U fetched. Note that one extra word, which may overlap the next VBUS command, is fetched.

By setting a bit in the Miscellaneous Control register, it is possible to replicate lines of V and U generated by the interpolator for the entire field. Since each line of VU data is displayed twice, the rate that the VU sample map has to be fetched from VRAM is reduced by ½. Table 2-2 lists the sequence of VU loads.

In some cases, the VU interpolator may cover only a portion of the display. In those instances, M in the above examples would be the first line that VU interpolation is enabled. N would be the last line that VU interpolation is enabled. Regardless of the state of the Line Replicate bit, there would be no vertical pipeline delay between the loading of the first line of samples and the second line of samples. The first line of samples would be loaded at M-1, and the second line at M. This reduces the delay between switching interpolation modes during a single display.

Table 2-2. VU Transfer Request Patterns with Line Replicate

Mode	Active Line	Request
2x Non-Interlaced	M	Fetch 1st Line of V
	M + 1	Fetch 1st Line of U
	M + 4	Fetch 2nd Line of V
	M + 5	Fetch 2nd Line of U
	M + 8	Fetch 3rd Line of V
	M + 9	Fetch 3rd Line of U
	N + 4	Fetch Last Line of V
2x Interlaced (Odd and Even Fields)	M	Fetch 1st Line of V and U
	M + 4	Fetch 2nd Line of V and U
	M + 6	Fetch 3rd Line of V and U
	N + 4	Fetch Last Line of V and U
4x Non-Interlaced	M	Fetch 1st Line of V
	M + 1	Fetch 1st Line of U
	M + 4	Fetch 2nd Line of V
	M + 5	Fetch 2nd Line of U
	M + 12	Fetch 3rd Line of V
	M + 13	Fetch 3rd Line of U
	N + 4	Fetch Last Line of V
4x Interlaced (Odd and Even Fields)	M	Fetch 1st Line of V and U
	M + 4	Fetch 2nd Line of V and U
	M + 8	Fetch 3rd Line of V and U
	N + 4	Fetch Last Line of V and U

BMX (0000) This command requests a bitmap. BMX (0000) is sent after horizontal active stops, beginning on the fifth line after vertical active starts, and continuing until the fifth line after vertical active stops. (There is a vertical pipeline delay of five lines through the 82750DB, due to internal timing requirements.) A line programmed to start at line M, wil have its first active line displayed at line M + 5. The 82750PB uses an internal pointer to cause the VRAM shift registers to be loaded with pixel values. The 82750DB subsequently fetches them as required for display. This command is asserted on the VBUS for the user-programmed number of T-cycles and must be completed before active display begins.

YBMNPX (0100) This command performs a Y bitmap transfer without performing a pitch calculation. When the line replicate mode is selected by Bit 22 in the Miscellaneous Control register, this code is asserted every other display line so that the same line of information can be used twice.

Digitizer Commands

When in the line replicate mode, and digitizing an NTSC source (for example, when genlocking an NTSC source to a system that uses only a VGA monitor), each line of captured data is effectively output at twice the rate. Since each line need only be stored once in memory (it is duplicted automatically in the display mode) only one WRDIGI code, followed by a WRDIGINP, is sent every other line. On alternate lines, two WRDIGINP are sent and will select the last address that was written, without incrementing the 82750PB bitmap address pointer. This is described in detail in Chapter 3.

WRDIGI (0011) This command requests a write of digitized data. The operation of this command is dependent upon the external hardware and is discussed in the section on genlocking (page 29). If digitizing is enabled, this command is asserted on the VBUS for a programmable number of T-cycles. The pointer is then incremented by a pitch value. Since each horizontal line is stored in a single row of memory, this pitch value is equal to the horizontal resolution, in bytes, for non-interlaced bitmaps. For interlaced bitmaps, the pitch value is equal to twice the horizontal resolution, in bytes. This allows alternate lines of data to be skipped over in successive fields.

WRDIGINP (0111) This command allows access to digitized data without performing a pitch calculation. WRDIGINP (0111) requests that the 82750PB perform a transfer request at the last calculated address. Note that only a memory transfer cycle is performed—the pitch value is not added to this address. This will always ensure that the digitized data is written into the last selected memory address, in case a physical memory boundary has been crossed. This command is asserted after the WRDIGI transfer has completed.

Refresh and Control Commands

The following signals are used to pass refresh requests and control information to the 82750PB.

DFL (1000) The Display Format Load command is a maskable host processor interrupt that can be programmed to occur at any time during the display. This is used by the 82750PB to transfer the shadow register contents into the working register set in the VRAM interface. This is useful in supporting split-screen-type applications, where it is desirable to change the bitmap pointers at some point before the end of the display.

82750DBSD (1001) This command is the 82750DB Shut Down code. During every register transfer, the 82750DB keeps an internal vertical exclusive-or checksum of the register data as it is read onto the chip. The last word of data that is read during the register transfer is the user-generated checksum. If the two checksums match, operation proceeds as normal. If they do not match, the 82750DB enters the reset state and sends this code to the 82750PB. The 82750DB will remain reset until the reset pin is asserted and negated by the host processor.

REFRESH (1010) This command asks the 82750PB to generate up to 15 refresh cycles every horizontal line. The 82750DB transfer cycles have a higher priority than refresh requests in the 82750PB. REFRESH will not be asserted if programmed to occur at the same time as a transfer request code.

Video Synchronization Information

The following codes are used to pass the video line and field information from 82750DB to the pixel processor.

VEVEN (1101) This code indicates the start of an even (i.e. second) field of a frame. This command is sent coincident with line one of each even field. When genlocking to an external source (see pg. 29), the occurence of a vreset signal during programmed horizontal active time will cause the 82750DB to output a VEVEN code on the VBUS.

VODD (1100) This code indicates the start of an odd (i.e. first or only) field of a frame. This command is always sent immediately after RESETB# is negated, and coincident with line one of the odd field. Similarly, when genlocking, the occurence of a vreset signal during any time other than horizontal active time will cause the 82750DB to output a VODD code on the VBUS.

HLIN (1110) This code marks every horizontal line at a programmable point in the line. HLIN is used by the 82750PB to increment its horizontal line counter.

Pixel Processing Path

This logic accepts the 32-bit word from the input latch and divides the word into the programmed pixel format. This will result in either four 8-bit pixels, two 16-bit pixels, one 32-bit pixel, or an 8-bit pixel with an 8-bit alpha value (pseudo 16-bit mode). The pixels act as addresses to the color table, or may bypass the table completely as described below.

Pixel information may be mixed with the output of the VU interpolator, which outputs interpolated samples derived from a reduced sample bitmap. The least significant bit of Y or LSB of U can be programmed to act as a switch between using the explicit pixel value of YUV or using the luminance portion of the pixel with the VU portion obtained from the interpolator. If the value of the LSB of Y (or U, whichever is selected) is zero, the pixel data is used. If the LSB of Y (or U) is one, the output of the VU interpolator is used. Note that if the LSB of Y is used as the switch flag, the luminance portion of the word will be only 7 bits wide.

The alpha information is also processed in this block. The alpha data may come from one of two sources: it may be explicitly coded in the pixel word, as is the case in the 32-bit/pixel and pseudo 16-bit/pixel mode, or it may be obtained by comparing the Y portion of the pixel with a preprogrammed value and outputting one preprogrammed value if they match and a different value if they do not match. This latter capability is known as Alpha Trap.

VU Interpolation

When VU interpolation is enabled by the programmer, and when the display is in the active region, "VU data" will be fetched, as required by the interpolator (by the mechanisms discussed previously in the section titled "VBUS Code Description"). This data has the format V, V, . . . , V, U, U, . . . , U where each V or U is 8 bits, and the bytes are grouped into 32-bit double-words with earliest in lowest order. The number, "N", of V bytes and U bytes is the same; N is programmed to be either 256 samples, or one of 32 to 192 samples in 32-byte increments.

The first V data and the first U data fetched on the first line of VU interpolation supplies the VU value for the first active pixel on that line. All the other VU pairs that are fetched define values for the grid of pixels defined below and to the right of this one by the VU expansion factor every other or every fourth horizontally and vertically. Most other VU values are filled in recursively by interpolation. Wherever there is a pixel which lies between two pixels with known values, it is given the value of the weighted average of the known values. Values are understood to be non-negative integers. When the final value is outputted, any fractions are truncated or rounded to the closest odd integer according to the programmed value of the interpolation round flag. This process is iterated until all pixels have assigned color values. If the number of VU data samples loaded into the 82750DB is not enough to cover the active display area, then the last data sample will be replicated horizontally across the active display window.

As mentioned previously in the VBUS Control discussion, each line of VU data can be used twice by setting the Line Replicate bit in the Miscellaneous Control register. Also, each horizontal VU sample can be replicated by setting the VU Replicate bit in the Pixel Control register. This will cause the V and U pixels generated by the VU interpolator every pixel time to be used twice. This can result in an effective 8X horizontal expansion, which is useful when horizontal blanking time is at a premium. This bit affects the horizontal interpolation algorithm only, and will not affect the line loading sequence for VU during the active display.

When interpolation is turned on by the programmer (by specifying a non-zero number of samples to be fetched), VU interpolation may nevertheless be disabled for each pixel if the following conditions are met:

1. Conditional interpolation has been selected by the programmer,

   AND

   Either of the two user-programmed conditions:

   a. Switching on the LSB of the U bit has been selected, and the lowest-order bit of the U value fetched for the upper left pixel in the block has value zero. This allows switching to occur on a 2 x 2-pixel or 4 x 4-pixel grid, depending on the expansion mode the user has selected. The full 8 bits of Y and V are used, but the usable space of U has been decreased to 7 bits.

   b. Switching on the LSB of the Y bit has been selected, and the low order bit of the Y value for the current pixel has a value of zero.

2. Display of fetched and interpolated VU values may also be suppressed by setting the Interpolation Output Enable bit (in the miscellaneous control register) to zero. This will allow VU data to be loaded into the VU line stores without displaying VU data. This is useful when a mid-screen transition is made between two interpolation modes, to compensate for the vertical latency of the interpolation process.

Colormap Lookup Table (CLUT) Operation

The 82750DB contains three 256 x 8-bit color lookup tables. The color maps can be accessed separately, or may act as one large 256 x 24-bit table. The manner in which the tables are addressed is determined by the programmed bits/pixel and depends on whether the pixel is a graphics or video pixel. Also each Y, U, and V color table address can be masked. The masks can be used in all the bit/pixel modes, but are most useful with the 16-bit/pixel mode. In this mode, the mask allows the YUV values to be mapped to 8-bit values instead of 6-5-5.

Each channel (Y, U, V) has a MASK SET register and a MASK DATA register that selects the color lookup address bit to be changed and the new value of the bit, respectively. A simple mask operation on one channel is illustated in Figure 2-8.

The CLUT address mask operation is determined by a logical equation given by:

$$\text{Result} = (\text{mask set and mask data}) \mid (\overline{\text{mask set}} \text{ and data byte})$$

Each bit of the Result byte is determined individually by this equation. The Result byte is then further processed in order to produce the CLUT RAM address.

For modes that require both video and graphics to pass through the color table, the table can be split into two halves: one half for graphics and the other for video pixels. By using the SPLITCLUT bit in the Miscellaneous Control register in conjunction with the LSB of Y or U, the color table address is forced to either the video table or graphics table automatically. In this case, the masking operation is still used, but the address is forced to either an even or odd entry, regardless of the results of the masking operation. The flag bit that decides between the two types of pixels automatically selects the correct portion of the CLUT table for a single channel. Note the LSB of Y or U selects the proper half of the CLUT for that single component. The SPLIT CLUT mode assures the proper half of the CLUT is used for all three components.

The color table can be bypassed completely when displaying either graphics or video, independent of the programmed bits/pixel. This is programmed by the user via the VIDEO PASS and GRAPHICS PASS bits in the Miscellaneous Control register. Table 2-3 summarizes the various modes when using the CLUT.

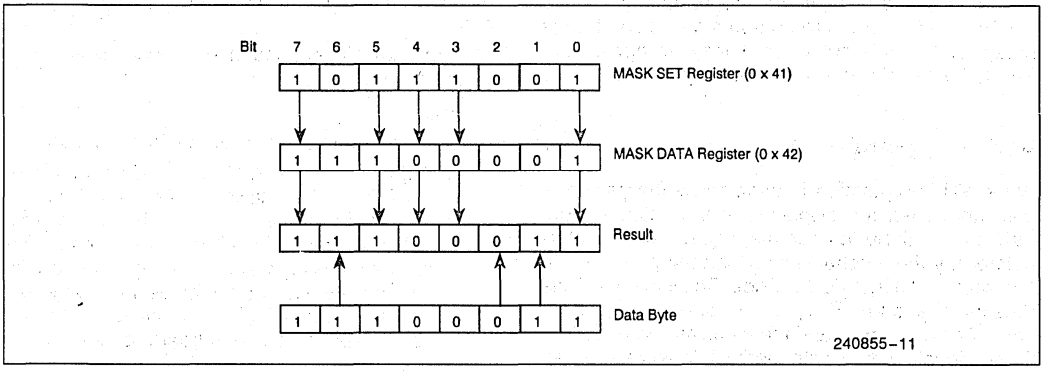


Figure 2-8. Mask Operation on CLUT Address

Table 2-3. CLUT Modes

Graphics Pass	Video Pass	LSB Y or U	SPLITCLUT	Colormap Address
0	X	0	0	Masked Graphics Data
1	X	0	X	Graphics Pixels Bypass CLUT
X	0	1	0	Masked Video Data
X	1	1	X	Video Pixels Bypass CLUT
0	X	0	1	Even Address Only (Graphics)
X	0	1	1	Odd Address Only (Video)
1	1	X	X	CLUT Not Used at All

When writing to the CLUT, the most significant byte of the data word corresponds to the address, and the least significant 24 bits are the YUV data (least significant to most significant, respectively). An index register is used to allow the 6-bit address to be mapped to an 8-bit number. (Refer to Chapter 4 for more information.) By resetting the 82750DA Disable bit, it is possible to make the CLUT look like the reduced entry color lookup table on the 82750DA.

The following paragraphs summarizes the possible bit/pixel modes, using the LSB of Y or U switching ability and the various graphics and video bypass modes. Note that there are modes where the LSB of Y or U are not used to switch between graphics and video.

8-BIT/PIXEL GRAPHICS MODE

This is the graphics-only mode, in which the 8 bits are used as inputs to all three color tables. This makes the color maps look like a single, 256 x 24-bit CLUT and allows 256 unique colors from a palette of 16 million to be available at any given time. If the Graphics Pass bit is asserted, the CLUT will be bypassed and the 8-bit values of the Y, U, and V channels will be input to each channel of the converter matrix.

8-BIT/PIXEL VIDEO MODE

When used with subsampled VU information from the interpolator, the 8 bits are actually a luminance value. The Y portion addresses the Y color table, V the V color table, and U the U color table. By using the color table, a one-to-one mapping exists, allowing non-linear transformations to be applied to the pixel data to enhance the quality of the reconstructed image. By asserting the VIDEOPASS bit in the Miscellaneous Control register, the color table can be bypassed.

8-BIT/PIXEL MIXED MODE

In the 8-bit/pixel mixed mode the LSB of Y or U is used as a switch flag to change the index to the color tables. When the switch flag is set to a one, the Y value corresponds to a luminance value, and the VU values are the chrominance information obtained from the VU interpolator. In this case each video component is used as an address to its corresponding CLUT as described above. When the switch flag is set to a zero, the VU values are not used and the Y value is used as the address to all color tables. These pixels are treated the same as in the 8-bit/pixel graphics mode.

In this mode the applications programmer must ensure that the proper information has been loaded into specific areas of the color maps. For example, all the video pixels will use the odd address values. By restricting the address used in the graphics and video mode, two unique maps may coexist in the tables. One map is used for non-linear transformations on video data, and the other for graphics color lookup table applications.

As illustrated above, the CLUT can be bypassed by asserting either or both of the bypass controls.

PSEUDO 16-BIT/PIXEL GRAPHICS MODE

In the pseudo 16-bit/pixel graphics mode each 32-bit data word is made up of two, 16-bit pixel words. The 82750DB processes each 16-bit pixel word, so that the least significant 8 bits correspond to pixel information, and the most significant 8 bits are used as alpha information. The 82750DB uses the lower 8 bits as inputs to all three color tables. This makes the color maps look like a single, 256 x 24-bit color table. If the Graphics Pass bit is asserted, the CLUT will be bypassed and the 8-bit values of the Y, U, and V channels will be input to each channel of the converter matrix.

PSEUDO 16-BIT/PIXEL VIDEO MODE

When used with subsampled VU information, the least significant 8 bits of the pixel word are actually a luminance value. The most significant 8 bits are used as alpha information. The VU information is generated by the 82750DB interpolator. Each of the color maps uses the corresponding 8-bit video component as an addess. By asserting the Video Pass bit in the Miscellaneous Control register, the color table can be bypassed.

PSEUDO 16-BIT/PIXEL MIXED MODE

In this mode the LSB of Y or U is used as switch flag to change the index to the color tables. When the LSB of Y or U is set to a one, the lower 8-bit value corresponds to a luminance value, and the V and U values are the chrominance information. In this case, each video component of the 82750DB is used as a colormap address as described above. When the LSB of Y or U is set to zero, the V and U values from the interpolator are not used, and the Y value is used as the address to all color tables.

16-BIT/PIXEL GRAPHICS MODE

The 16-bit pixel word is broken up on the 82750DB to yield 6 bits of Y, and 5 bits each of V and U. The Y bits are the least significant, and the U bits are the most significant. These values are then padded with zeros in the lower order bits, to obtain an 8-bit word for each pixel component. Each component addresses its respective CLUT. However, the Y channel may access only 64 unique locations, and 5-bit resolution for VU restricts them to 32 unique locations each. The address range may be extended by using the colormap mask registers to add 2 bits of precision in the least significant bits for Y and 3 least significant bits each for VU channels. This allows the programmer to access all the entries in the color table by reprogramming the MASK DATA and MASK SET registers during the blanking interval.

16-BIT/PIXEL VIDEO MODE

This mode works like the 8-bit/pixel video mode described above, except that the 82750DB has processed the information so that the Y channel contains the least significant 8 bits of the 16-bit data word. The V and U information is generated by the VU interpolator. If the SPLITCLUT mode is selected, the LSB of the address is forced to an odd entry in the three color tables.

16-BIT/PIXEL MIXED MODE

When the switch flag is zero, the graphics mode is selected and the inputs to the CLUT are the respective YUV data in the 6-5-5 format. These pixel values are extended by using the colormap masking registers. When the switch flag indicates the video mode, the lower 8 bits of the 16-bit pixel word and the VU values obtained from the interpolar are input to their respective CLUTs. If the SPLITCLUT mode is selected, the LSB of the address is forced to either an odd or even entry in the three color tables, depending on whether the data is video or graphics information.

32-BIT/PIXEL GRAPHICS MODE

Eight bits each of Y, U, and V are used as addresses to each segment of the color table. Since the size of the addressable color space is not increased, the advantage of using the color map is for special effects or gamma correction. The most significant 8 bits of the 32-bit data word are used for the alpha channel data. If the Graphics Pass bit is asserted, the CLUT will be bypassed and the 8-bit values of the Y, V, and U will be input to each channel of the converter matrix.

32-BIT/PIXEL VIDEO MODE

The Y channel contains the least significant 8 bits of the 32-bit data word. The U and V information is generated by the VU interpolator. The YUV channels are input to their respective color tables. The size of the addressable color space is not increased, but this can be used to take advantage of a non-linear transformation, which may aid in the decompression process. The most significant 8 bits of the data word are used for the alpha channel data.

32-BIT/PIXEL MIXED MODE

When the switch flag is zero, the graphics mode is selected, and the inputs to the CLUT are the respective 8 bits each of YUV data. These pixel values may be masked by using the colormap mask data and mask set registers. When the switch flag indicates the video mode, the lower 8 bits of the pixel word and the VU values obtained from the interpolator are input to their respective CLUTs. If the SPLITCLUT mode is selected, the LSB of the address is set to either an odd or even entry in the three color tables, depending on whether the data is video or graphics information. The most significant 8 bits of the data word are used for the alpha channel data.

Y Interpolator

The Y Interpolator performs a 2X horizontal linear interpolation on each line of Y values. When Y interpolation is enabled, the internal pixel clock is twice the frequency of PIXCLK output.

NOTE:

If Y interpolation is enabled, then only the integer values of pixel times greater than 1X may be used.

The interpolation may be separately controlled for both video and graphics pixels, via the Viden and Gren bits (bits 12 and 11) of the General Control register. A video pixel is defined as one generated using VU interpolated values. A graphics pixel does not use the VU interpolator. The effects of setting the control bits, the 82750DB enable flag, and video/graphics pixel switch (V/G Switch) on the output of the interpolator are summarized in Table 2-4.

Because of the asymmetric nature of the internal pixel clock used on 82750DB, the number of T-cycles between successive Y pixels varies depending on the programmed pixel width. When enabled, there is a pipeline delay through the Y Interpolator equal to the number of T-cycles between each internal pixel clock.

When the interpolator is bypassed as described above, there is a fixed delay through this block. The V and U data are delayed by one pixel clock to allow the chroma data to line up with the luminance data. Other control signals, such as the register address byte (most significant byte of the 32-bit data word read from VRAM), the pixel clock, horizontal and vertical active displays, composite blanking, and register load enable signals are also delayed by one pixel clock in order to line up with the YUV data. The programmer must ensure that the active display timing is programmed to take the appropriate delay through the Y Interpolator into account.

Table 2-4. Control Bit Settings and Resulting Interpolator Output

82750DB Enable	Viden	Gren	V/G Switch	Result
0	X	X	X	Interpolator Bypassed
1	0	0	X	Interpolator Bypassed
1	0	1	0	Interpolate Graphics Pixel
1	0	1	1	Do Not Interpolate Video Pixel
1	1	0	1	Interpolate Video Pixel
1	1	0	0	Do Not Interpolate Graphics Pixel
1	1	1	X	Interpolate Both Video and Graphics Pixels

Cursor

Hardware support for a 16 x 16-pixel cursor has been included on the 82750DB. The cursor is capable of providing sharp color transitions, when using subsampled VU bitmaps. Software intervention is minimized, leaving the host with more processing cycles to perform other operations.

Under normal operation, the XY starting display position of the cursor is loaded into the Cursor Control register during a 82750DB register load. On the display line corresponding to the Y start position, the

cursor is displayed when the X starting position (specified in T-cycles) is reached. On the following 15 lines, the cursor will be displayed at this X position every line, for both interlaced and non-interlaced displays.

A normal 82750DB register transfer is used to load the entire 16 x 16 x 2 bits (16 words of 32 bits each) of cursor data. During this register transfer, the cursor data is distinguished from normal register data by placing the Cursor Control register immediately before the 16 words of cursor data. When the 82750DB loads the Cursor Control register, it will interpret the next sixteen 32-bit words of register data as the cursor bitmap, and will disable the other registers on the 82750DB from decoding the address field of the 32-bit data word. (The checksum of the 82750DB register data is not performed during the loading of the cursor bitmap data.) The cursor bitmap will be loaded a line at a time, starting at line zero and continuing in sequential order to line 15. Each line in the cursor map actually contains sixteen 2-bit cursor pixels, with the two least significant bits corresponding to the first cursor pixel in that line, and the two most significant bits corresponding to the 16th cursor pixel on that line. Each 2-bit pixel may select one of the three Cursor Color registers or transparency, according to the format indicated in Table 2-5.

Table 2-5. Cursor Color Registers

Cursor Pixel	Output
00	Transparency (Cursor Pixel Not Displayed)
01	Cursor Color Register 1
10	Cursor Color Register 2
11	Cursor Color Register 3

Three 24-bit color registers that hold the color information for the cursor may be written to at any time during the register load. The cursor may be loaded any time during the blanking intervals of the display. For displays that do not program the cursor during the display, the cursor bitmap may be loaded during the vertical blanking interval.

When the T-cycle count equals the value programmed into the X start position of the Cursor Control register, the first cursor pixel can be displayed. Each 2-bit cursor pixel will select one of the three Cursor Color registers or transparency. The 24-bit output of one of the three color registers (or the actual display pixel data if transparency is used) is input to the YUV converter.

The cursor bitmap length is 16 lines, and the width is 16 pixels. Although the length of the cursor may be changed dynamically by chaining register loads to update the cursor map, the size of the cursor is dependent on the type of display. For interlaced displays, each line of cursor data will appear on the same line of each field. This results in a cursor of 16 x 32 pixels. For non-interlaced displays, the same line of cursor information will appear on the same line every field. The cursor in this case will be 16 x 16 pixels. The size of the cursor may be doubled independently in the horizontal and/or vertical direction by setting the 2X Horizontal Cursor or 2X Vertical Cursor bit in the General Control register. In this case, no new data is loaded into the cursor map; the data is just replicated in the corresponding dimension. Table 2-6 summarizes some of the possible cursor sizes. Note that by loading the cursor bitmap with different data at the start of every field, cursor sizes not listed below may be achieved.

Table 2-6. Cursor Sizes

2X Horz. Cursor	2X Vert. Cursor	Display	Cursor Size (in Pixels)
Off	Off	Interlaced	16 x 32
On	Off	Interlaced	32 x 32
Off	On	Interlaced	16 x 64
On	On	Interlaced	32 x 64
Off	Off	Non-Interlaced	16 x 16
On	Off	Non-Interlaced	32 x 16
Off	On	Non-Interlaced	16 x 32
On	On	Non-Interlaced	32 x 32

There is a complex relationship between the cursor and the pixel data especially when using non-integral divisors of the pixel clocks. Since the pixel data output from the 82750DB pixel path always changes coincident with the rising edge of the clock, the cursor start position must be positioned on the rising edge of any period of the pixel clock. The programmer must enforce the corresponding restrictions on the start and stop position of the cursor.

YUV to RGB Converter

The following equations give the theoretical relationship between analog RGB components, R, G, B, and analog YUV components, Y, U, V.

$$Y = 0.298822\,R + 0.586816\,G + 0.114363\,B \quad (1a)$$

$$V = R - Y = 0.701178\,R - 0.586816\,G - 0.114363\,B \quad (1b)$$

$$U = B - Y = -0.298822\,R - 0.586816\,G + 0.885637\,B \quad (1c)$$

where: $0.0 < G, R, B < 1.0$

$0.0 < Y < 1.0$

$-0.701 < V < +0.701$

$-0.886 < U < -0.886$

Solving for G, R, B, we can obtain the inverse relationship:

$$G = Y - 0.509228\,V - 0.194888\,U \quad (2a)$$

$$R = Y + V \quad (2b)$$

$$B = Y + U \quad (2c)$$

where: $0.0 < G, R, B < 1.0$

$0.0 < Y < 1.0$

$-0.701 < V < +0.701$

$-0.886 < U < +0.886$

The luminance channel for the YUV inputs is presumed to swing between 0.0V and 1.0V. However, the chroma components do not and need to be normalized to a 0V to 1V range. The offset binary encoding used to obtain unsigned numbers must also be accounted for. This encoding should center the V and U inputs at the midpoint of the voltage range. The equations for the normalized version of Y, V, and U (Y′, V′, and U′ respectively) are:

$$Y' = Y \quad (3a)$$

$$V' = \frac{0.5V}{0.701} + 0.5 \quad (3b)$$

$$U' = \frac{0.5U}{0.886} + 0.5 \quad (3c)$$

where: $0.0 < Y', V' U' < 1.0$

$0.0 < Y < 1.0$

$-0.701 < V < +0.701$

$-0.886 < U < +0.886$

When converting the normalized analog values Y′, V′, U′ to digital y, v, u values, the D.C. offset and conversion ranges are compatible with the CCIR 601 standard for digital video. The ranges for the components and the corresponding Digital to Analog equivalent equations are given below:

$$y = (235 - 16)\,Y' + 16 \quad (4a)$$

where: $16 < y < 235$

$$v = (240 - 16)\,V' + 16 \quad (4b)$$

where: $16 < v < 240$

$$u = (240 - 16)\,U' + 16 \quad (4c)$$

where: $16 < u < 240$

Substituting the normalized analog voltages of Equation 3 into Equation 4, we obtain the digital version of the input data, used in the DVI™ Technology system:

$$y = (219)\,Y + 16 \quad (5a)$$

$$v = \frac{112V}{0.701} + 128 \quad (5b)$$

$$u = \frac{112U}{0.886} + 128 \quad (5c)$$

where: $0.0 < Y < 1.0$

$-0.886 < U < 0.886$

$-0.701 < V < 0.701$

$16 < y < 235$

$16 < v, u < 240$

By solving equations 5 for Y, U, V, and substituting into Equation 2, we get the relationship between analog R, G, B and the digital DVI y, u, v data:

$$G = 0.004566\,y - 0.003187\,v - 0.001541\,u + 0.532242 \quad (6a)$$

$$R = 0.004566\,y + 0.006259\,v - 0.874202 \quad (6b)$$

$$B = 0.004566\,y + 0.007911\,u - 1.085631 \quad (6c)$$

where: $0.0 < R, G, B < 1.0$

$16 < y < 235$

$16 < v, u < 240$

If the inputs of the Digital to Analog Converter are scaled to accommodate the nominal input range of 0 to 219, we obtain the following relationship between the inputs to the DVI Technology system, (y, v, u) and inputs to the Digital to Analog Converters (r, g, b). Note that all out of range RGB values (> 255 or < 0 due to excursions in the inputs) are clipped to 255 or 0.

$$g = y - 0.698001\,v - 0.337633\,u + 116.56116 \quad (7a)$$

$$r = y + 1.370705\,v - 191.45029 \quad (7b)$$

$$b = y + 1.732446\,u - 237.75314 \quad (7c)$$

where: $16 < y < 235$

$16 < v, u < 240$

$0 < g, r, b < 255$

By substitution of Equation 5 into Equation 1, and by converting G, R, and B to digital values, we can obtain the inverse relationship of Equation 7:

$$y = +0.298822\,r + 0.586816\,g + 0.114363\,b + 16 \quad (8a)$$

$$u = -0.172486\,r - 0.338721\,g + 0.511206\,b + 128 \quad (8b)$$

$$jpv = +0.511545\,r - 0.428112\,g - 0.083434\,b + 128 \quad (8c)$$

where: $16 < y < 235$

$16 < v, u < 240$

$0 < g, r, b < 255$

Output Equalization

The units on the 82750DB process the pixel information at the operating frequency of the chip. If the output pixel rate is not equal to the maximum frequency, the units have null states during which processing is suspended. This type of operation is necessary on the 82750DB because of the large amount of pipelining. Table 2-7 gives the pattern of T-cycles on the 82750DB during which processing is active, according to the programming shown in Table 4-2.

The pixel information must be output at a rate that is some sub-multiple of the operating frequency. The divisor is programmed by the user, and may be from 1 to 12 times slower than the period of FREQIN, in increments of ½. Divisors of 13 and 14 are also programmable. Because non-integral divisors are used, it is necessary for the 82750DB to output different information on both phases of FREQIN. This is illustrated in Figure 2-9, which uses a 2.5 divisor for the clock. Notice that the pixel clock output (PIXCLK) transitions fall alternately on the active and inactive phase of the input frequency, while the internal pixel clock transitions always occur on the active phase. Also note that PIXCLK does not have a 50% duty cycle.

The equalizing logic derives a clock that has a period equal to the programmed pixel rate, providing an edge to sample the output information. This allows the Digital to Analog Converter to directly sample the output of the pixel data path before performing the analog conversion.

Table 2-7. 82750DB Active T-Cycle Patterns

Pixel Time (T-Cycles)	Pattern Of Internal Pixel Clock
1	Always On
1.5	1 On/1 On/1 Off
2	1 On/1 Off
2.5	1 On/1 Off/1 On/2 Off
3	1 On/2 Off
3.5	1 On/2 Off/1 On/3 Off
4	1 On/3 Off
4.5	1 On/3 Off/1 On/4 Off
5	1 On/4 Off
5.5	1 On/4 Off/1 On/5 Off
6	1 On/5 Off
6.5	1 On/5 Off/1 On/6 Off
7	1 On/6 Off
7.5	1 On/6 Off/1 On/7 Off
8	1 On/7 Off
8.5	1 On/7 Off/1 On/8 Off
9	1 On/8 Off
9.5	1 On/8 Off/1 On/9 Off
10	1 On/9 Off
10.5	1 On/9 Off/1 On/10 Off
11	1 On/10 Off
11.5	1 On/10 Off/1 On/11 Off
12	1 On/11 Off
13	1 On/12 Off
14	1 On/13 Off

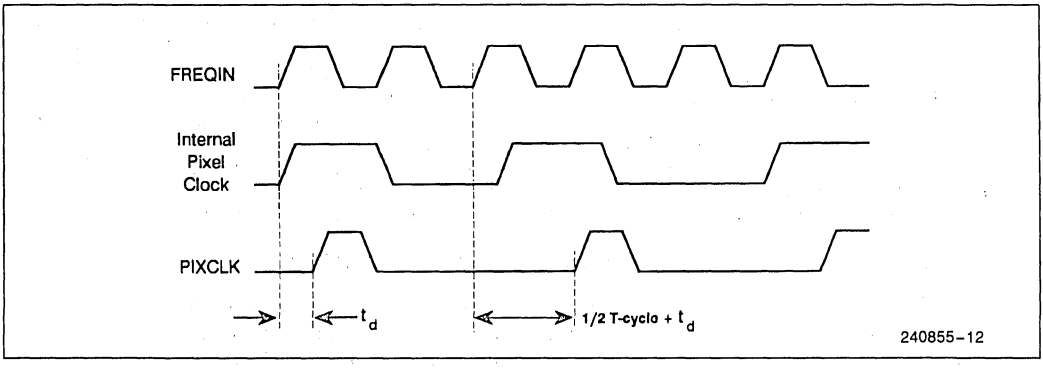


Figure 2-9 Divide by 2.5 Pixel Clock

Digital to Analog Converters

The Digital to Analog Converters (DACs) take three channels of video information output from the pixel data path, converting it from 8-bit digital values to analog voltage levels typically between 0V and 1V. The conversion is monotonic, and a pixel clock is used to derive a two-phase clock internal to the DAC. The data is sampled from the output of either the pixel path, or the YUV to RGB matrix on the rising edge of the internal active phase of this clock. The DISDAC input pin can be asserted to disable the analog outputs and place them into a high-impedance state.

The analog outputs of the triple DAC are referenced to an external current source, which must be connected to the IREFIN pin. All the analog outputs are scaled by this current reference. The value of the analog output full scale is as follows:

$$Ifs = Iref * \frac{255}{18.5}$$

where: Iref is the magnitude of the reference current.

The output voltage generated at full scale is:

$$Vfs = Ifs * Rext$$

Rext is the load resistance value.

A typical output load for the analog outputs (RV, BU, GY) is 75Ω. The speed of the DAC analog output rise and fall times is determined by the time constant:

$$Rext * (Cext + Cout)$$

where: Cext is the external capacitance applied and Cout is the intrinsic capacitance of an analog output.

For high performance the objective would be to minimize Rext and Cext. The voltage Voutfs can be determined by any combination of Ifs and Rext, but must not exceed 1.5V. In addition Ifs must not exceed 22 mA. The analog outputs must go through an external buffer to drive doubly-terminated 75Ω coax line.

Table 2-8 lists pins which are used to configure the triple DAC.

Table 2-8. Digital To Analog Converter Pins

Signal	Description
IREFIN	Analog Current Reference. Must Be Decoupled to AVCC.
VGCS	Internal Voltage Reference. Must Be Decoupled to AVCC.
AV_{CC}	Analog Power
AV_{SS}	Analog Ground
GY, RV, BU	Analog Pixel Outputs
DISDIG	Disable Digital Outputs
DISDAC	Disable Analog Outputs

NOTE:

The digital video outputs must be disabled by setting DISDIG high whenever the analog outputs are used. Otherwise the A.C. and D.C. characteristics of the DAC are not guaranteed.

3.0 HARDWARE INTERFACE

82750DB Reset Operation

Upon power-up, the 82750DB is in an indeterminate state and must be reset. The RESETB# signal asserted by the host processor is sampled on the rising edge of FREQIN. The 82750DB will enter the reset state a maximum of four cycles after RESETB# is sampled. The 82750DB will request the 82750PB to generate VRAM refresh cycles by asserting a REFRESH code on the VBUS for 16 T-cycles. This code is repeated every 256 T-cycles, until RESETB# is negated.

NOTE:

The RESETB# input is an edge-triggered input. After power-up, the host processor must set the RESETB# input low for a minimum of ten T-cycles in order to reset the 82750DB. The host must then set the RESETB# input high to start normal operation.

When the RESETB# input is released, a Start of Vertical Field command (VODD) is sent for 16 T-cycles to the 82750PB via the VBUS. This code is immediately followed by a Register Transfer Request command (REGX) that is held for 256 T-cycles. This 256 T-cycle wait assures that the 82750PB has ample time to honor the 82750DB register transfer request. The register data is then read into the 82750DB from the serial port of the VRAMs at a rate that is equal to ⅓ of the operating frequency. If the register transfer does not terminate after 256 T-cycles, the 82750DB will automatically stop the transfer, send an 82750DBSD code to the 82750PB, and re-enter the reset state.

During this register transfer, and on all subsequent register transfers (programmed or automatic), the 82750DB performs a vertical checksum on the register data. The last 32-bit word read in during a register transfer is the user-generated checksum of that register data. If the 82750DB-generated checksum error does not match the user-generated checksum, the 82750DB sends a SHUTDOWN code to the 82750PB via the VBUS, and will automatically re-enter the reset state. The 82750DB will remain in the reset state until the RESETB# input is toggled by the host processor. Any VRAM requests or control signals programmed to occur during this time will be ignored.

Normal programmed operations start after the first successful register load. Frame timing will start at the beginning of a horizontal line and at the beginning of the first field sometimes referred to as line 1 of field 1. There will not be a horizontal sync pulse on the first line after reset, but HSYNC will be generated on every line thereafter. All horizontal and vertical programming parameters as well as scheduling of any transfer requests and control information to be sent on the VBUS must be set up by the user during the first register load. Included in the control information are parameters for the 82750PB to refresh the VRAM. Refresh must occur on every line. This requires that the line rate of the 82750DB must be at least 4 kHz to guarantee that enough refresh cycles are generated. Additional register transfers (up to one per line) may be programmed to occur on any line during the field. As a result of this transfer display characteristics and programming parameters may be changed.

After the first field, automatic register transfers will occur on the second line of each subsequent field. Note that all register transfers will occur at 1/3 of the operating frequency of the 82750DB, unless the 1X or 1/2X SCLK mode has been programmed by the user.

Throughout the reset process, the states of all outputs become valid at various times. Specifically, after being held low for at least 10 T-cycles, RESETB# must transition to a high state in order to initiate normal operation. By the time RESETB# reaches this low to high transition, the states of SCLK[1:0], VBUS[3:0], HSYNC, VSYNC, CSYNC, and FCO are valid. 10 T-cycles following RESETB#'s transition from low to high, the states of BG, CB, ACTDIS, PIXCLK, DGY[7:0], DRV[7:0], and DBU[7:0] become valid. ALPHA[7:0] and BPP[1:0] signals reach a valid state 10 T-cycles following the completion of the first register load following reset.

Input/Output Transformation

In general, the control outputs, including the sync signals, are delayed by pipelining effects from their corresponding inputs. If the output sync signals are taken as the time base, the first pixel in a line is actually fetched by an SCLK that is up to 19 T-cycles before its corresponding PIXCLK. Some later pixels may be delayed by an additional number of T-cycles, depending upon bits/pixels, pixel timing, and whether Y interpolation is enabled.

Outside of the active display region and before the blanking output is asserted, border pixels are output. Where the blanking region has been entered and the display is not active, the output is the value contained in the Blanking Color register.

Pixel handling in the active region is defined by three parameters:

1. The bits/pixel parameter.
2. Whether VU interpolation is in effect or not.
3. If the 82750DB Enable bit has been selected.

VU interpolation is in effect for a given pixel if:

1. The VU interpolator is turned on (VU sample load set to non-zero load value),

   AND

2. VU interpolation display is permitted (VU interpolation display operations bit equals 1),

   AND

3. One of the two following conditions is met:

   a. Either the interpolation is unconditional,

      OR

   b. The controlling Y or the controlling U sample for this pixel has a least significant bit of 1.

The value of the alpha output may come from one of the following three sources:

1. It may be explicitly coded into the pixel data (32-bit/pixel and pseudo 16-bit/pixel with Alpha modes only).
2. It may be output from one of two programmable registers, Alpha0 and Alpha1.
3. During the portion of the display when the border is active, the 8 most significant bits of the Border Alpha register may be output.

Table 3-1 illustrates how the Alpha outputs are selected.

Table 3-1. Selecting Alpha Outputs

Alpha Enable	Alpha Trap Select	Alpha Output
0	X	Alpha0 Register
1	0	Alpha0 Register (8, 16 bpp)
1	0	MS Byte of Pixel (32, Pseudo 16 bpp)
1	1	Trap Match = 0, Alpha0 Register
1	1	Trap Match = 1, Alpha1 Register

Genlocking on the 82750DB

The genlocking algorithm on the 82750DB uses horizontal and vertical resets, HRESET# and VRESET#, obtained from an external device. When the Genlock bit in the Miscellaneous Control register is off, the 82750DB will ignore all signals present on it's HRESET# and VRESET# inputs. The 82750DB will resync itself when the programmed end of line count is received. This allows the user to turn off genlock without having to worry about the state of the input video.

When the Genlock bit is set to one, the 82750DB will use the external resets to reset its internal horizontal and vertical sync counters. In this case, the width of the active line is determined by the HRESET# signal, and the length of the field is governed by VRESET#. The programmed values for these registers will be ignored. As shown in Figure 3-1, when asserted VRESET# and HRESET# are effected just after the third falling edge of FREQIN. VRESET# has no effect on the 82750DB if the first half of the first line of an odd field or the second (and only) half of the first line of an even field is already in progress. HRESET# has no effect on the 82750DB if it occurs during the programmed first half of the line. The user may decrease the effect of jitter by reducing the "window" during which the vertical reset signal is supposed to occur. This can be done by scheduling a register load to occur after the vertical active display time has ended, thereby decreasing the programmable horizontal active window to a size acceptable for the video source. When VRESET# is received during this reduced, programmed horizontal active window, the 82750DB is reset to an even vertical field. When VRESET# occurs at any other time in the horizontal scan line, the 82750DB is set to an odd field.

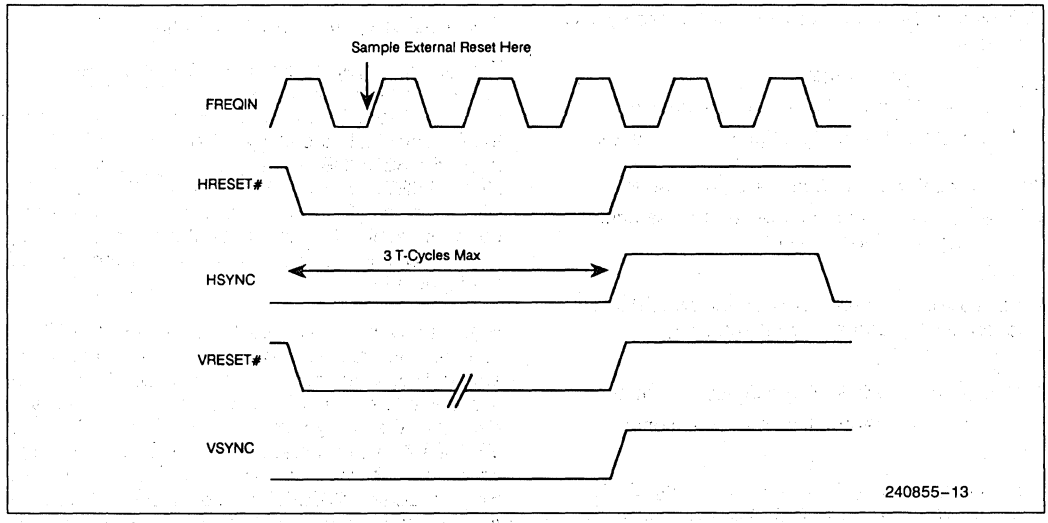


Figure 3-1. Horizontal and Vertical Reset Timing

Digitizing Images with the 82750DB

Digitizing is enabled by setting the Digitize Enable bit in the Miscellaneous Control register. Note that enabling the digitize mode does not automatically enable genlocking. The Genlock bit must be set separately, if it is required. When digitizing, the 82750DB is used to shift digitized data into the VRAM shift registers, and then transfer this data into the VRAM array.

The 82750DB also provides an external "digitizer window" signal, FCO. This signal defines the vertical active region that the digitizer enabled. Typically, the user sets up the display parameters to reflect the "window" of the display to be digitized. The horizontal and vertical active window size can be selected by programming the Active Start and Stop registers. FCO is derived from the Vertical Start and Stop registers, and is used to enable the digitizer to drive the VRAM bus. During the programmed vertical blanking interval the FCO signal will be negated, and therefore, the digitizer is prohibited from driving the VRAM bus. This will allow data to be read from the VRAM serial data bus during the automatic register transfer that is performed at the start of the field. Note that it will still be possible to program the 82750DB to digitize during the vertical blanking interval, in order, for example, to capture time codes from a VCR.

When capturing and displaying NTSC data during the horizontal blanking interval of the first display line, a WRDIGINP command is sent on the VBUS to the 82750PB. (Refer to Figure 3-2.) Recall that there is a 5-line vertical pipeline delay through the 82750DB. If the first display line is programmed to be n, the first display line will occur at n + 5. Similarly, if the last line is programmed to be m, then the last display will be line m + 5. The WRDIGINP VBUS code causes a dummy write transfer cycle that places the VRAMs in the write mode. The 82750PB then sets the bitmap pointers to the first line's address (L0). This code is immediately followed by another WRDIGINP command that causes the 82750PB to perform a write transfer cycle at the L0 address. Since no digitized data has been read in, invalid data is loaded into row L0 of the VRAM array.

During the active display of the first display line, the 82750DB provides shift clocks at the programmed pixel rate. The digitized data is shifted into the VRAMs while the user-programmed horizontal active window is active. During the horizontal blanking interval of the next line, the 82750DB sends a WRDIGI code to the 82750PB, thereby transferring the L0 data from the shift register to the VRAM array at the L0 address. The 82750PB performs a pitch calculation, pointing it to the L1 row. After the WRDIGI

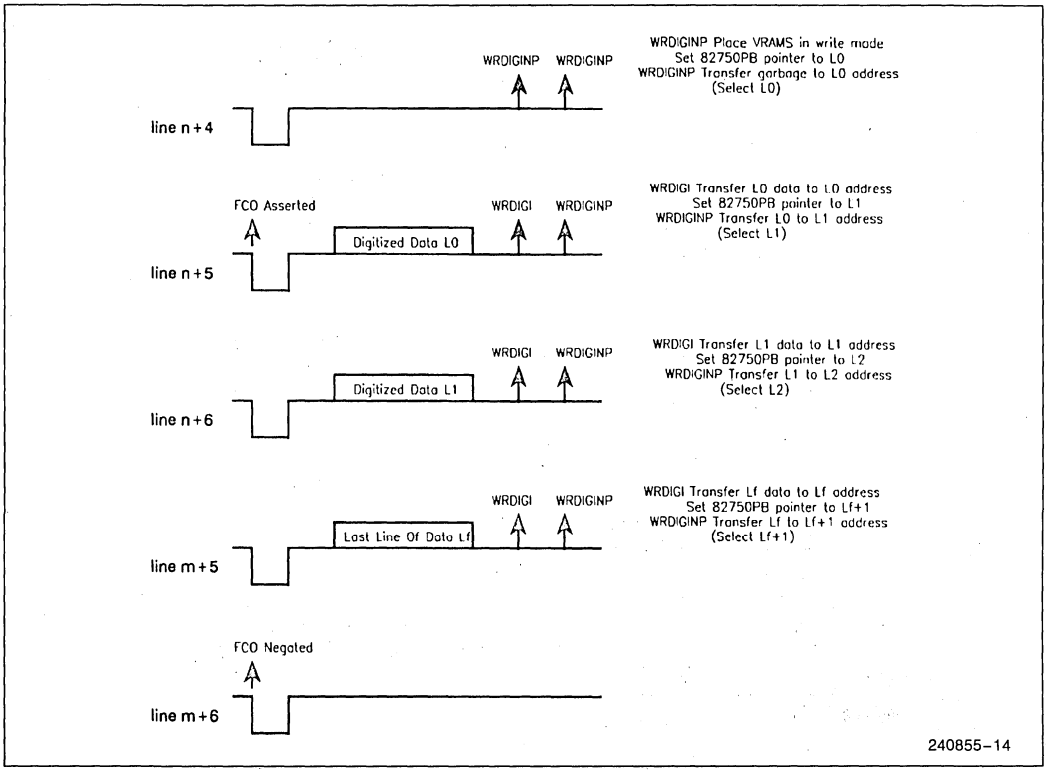


Figure 3-2. Digitizing Example

transfer has finished, the 82750DB issues a WDIGINP command to the 82750PB that performs a write transfer cycle at L1 address. This will write the L0 data into the L1 address. The next line the L1 row will be written over with L1 data. This same procedure continues for the entire active display, until the last active line is reached (m + 5). A final pair of WRDIGI and WRDIGINP codes are sent to the 82750PB to load in the last line of data. At the start of horizontal sync of the next line, the FCO signal will be negated.

The purpose of the WDIGINP may not be apparent at first glance. This signal ensures that the correct data is written into the last selected VRAM address. This is necessary when crossing the physical boundaries of VRAM memory.

When the 82750DB is genlocked, the digitizing device must also provide the HRESET# and VRESET# signals. The device must ensure that VRESET# is never asserted during the start of the line. This allows a register transfer (which shortens the active display and is required for digitizing) to complete before the start of a field register transfer.

The vertical sync pulses are buffered, so the start of the field transfer request can be honored immediately after the previous transfer request is finished.

Also, captured NTSC data may be displayed on a VGA-type monitor. This requires the 82750DB to operate at a VGA frequency (approximately 31.5 kHz), which is twice that of NTSC. Each line of captured NTSC data is read into the 82750DB twice. Setting the line replicate bit makes doubling of memory unnecessary. Figure 3-3 illustrates how the 82750DB operates in such a mode. The Line Replicate, Digitizer, and Genlock bits in the Miscellaneous Control register are assumed to be set to one. During the HBI of the first display line, a dummy write transfer cycle (WRDIGINP) places the VRAMs in the write mode. The 82750PB then sets the bitmap pointers to the first line's address (L0). This code is immediately followed by a WDIGINP command, causing the 82750PB to perform a write transfer cycle at the L0 address. Since no digitized data has been read in, unknown values are loaded into row L0 of the VRAM array.

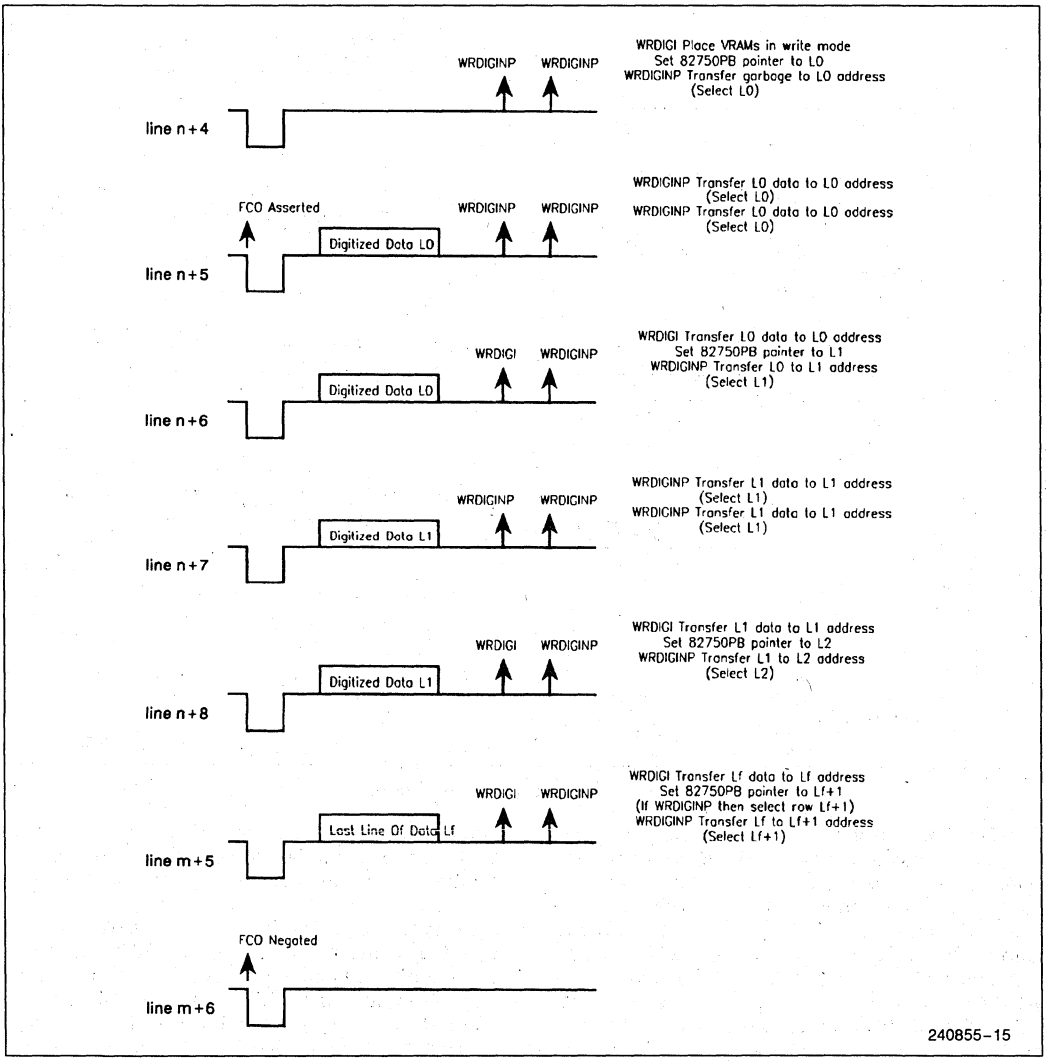


Figure 3-3. Digitizing Example with Line Replicate

At the end of the first line the 82750DB sends two WRDIGINP codes to the 82750PB, thereby transferring the L0 data from the shift register to the VRAM array at the L0 address. The 82750PB does not perform a pitch calculation, so the pointer remains at the address for L0. After the second display line (which has the same data as the first line), a WRDIGI code is sent to the 82750PB that writes the L0 data to the L0 address and updates the bitmap pointer to L1. The WRDIGINP signal immediately following this selects the L1 address. After the third line of data, two WRDIGINP codes that select the L1 address are sent. After the fourth line, (which has the same data as the third line) a write operation is performed to load L1 data into the L1 address, and the 82750PB pointer is updated to address L2. A WRDIGINP code is sent to select the L2 address. This same procedure continues for the entire active display, until the last active line is reached (m + 5). A final pair of WRDIGI and WRDIGINP or two WRDIGINP codes are set to the 82750PB to load in the last line of data. At the start of horizontal sync of the next line, the FCO signal will be negated.

4.0 PROGRAMMING THE 82750DB

Overview

All registers are loaded by the issuance of a REGX command from the 82750DB to the 82750PB over the VBUS. This causes the 82750PB to load a sequence of register values into the VRAM serial output registers from an address designated by a 82750DB register pointer. After the request is granted, a new 82750DB register word is read in with each SCLK. Each 32-bit word consists of a register address in the high byte and register values in the rest of the word. The sequence is terminated by a stop code that corresponds to the address byte being equal to 0xff. A variable number of 32-bit words can be loaded. During reset, if a stop bit is not found within 256 T-cycles, the register transfer is terminated, a SHUTDOWN code is asserted on the VBUS, and the 82750DB returns to the reset state. All transfer requests are terminated at the start of a new field. This ensures that non-terminating register transfers caused by bad register data will be halted.

During this register transfer, and on all subsequent register transfers (programmed or automatic), the 82750DB performs a vertical checksum on the register data. The last 32-bit word read in during a register transfer is the user-generated checksum of that register data. If the 82750DB-generated checksum error does not match the user-generated checksum, the 82750DB sends out a SHUTDOWN code to the 82750PB via the VBUS, and will automatically re-enter the reset state.

Pipeline Delay through the 82750DB

The actual horizontal pipeline delay through the 82750DB is dependent on processing elements used to generate the output. If Y interpolation is not used, the pipeline delay is:

Horiz. Active Pipeline Delay = 16 cycles + SCLK Transfer Timing Delay

Here the SCLK Transfer Timing Delay is 1 for 1X, 2 for 1/2X, and 3 for 1/3X.

If Y interpolation is used, the pipeline delay is:

Horiz. Pipeline Delay = 16 cycles + SCLK Transfer Timing Delay + Integer (Pixel Time)

The integer (Pixel Time) is simply the integer value of the programmed pixel time. The horizontal pipeline delay for blanking differs from that of active. When y-interpoloation is on or off, the pipeline delay for horizontal blanking is:

Horiz. Blanking Pipeline Delay = 10 cycles + SCLK Transfer Timing Delay

The horizontal sync pipeline delay is always equal to 0 cycles.

Thus all horizontal parameters, (e.g. horizontal blanking start, active stop) must be programmed to account for the total horizontal pipeline delay. The vertical pipeline delay. The vertical blanking and vertical sync pipeline delay are always equal to 0 lines. All vertical parameters must be programmed so that this delay is taken into account.

PROGRAMMING CONSIDERATIONS

The user must ensure that the 82750DB is programmed correctly. Illegal or illogical combinations of display parameters are not corrected in hardware, and may cause the 82750DB to output erroneous display or timing information. The following list highlights some basic guidelines to follow when programming the 82750DB.

1. The maximum rate that data may be read into the 82750DB is determined by the type of memory used. This in turn effects the maximum rate and depth of data that can be displayed. If 32 bits of data can only be read into the 82750DB every two clock cycles, only 16 bits of data may be displayed every clock cycle. The programmer should match the transfer rate (1X, 1/2X, or 1/3X) with the memory speed, and the display pixel rate with the pixel depth and memory bandwidth.
2. Blanking intervals of the display are defined by the non-active programmed time. During this portion of the display, programmed transfers take place. If a transfer does not complete before the start of the active display, it is terminated, and active display data is shifted into the 82750DB at the programmed rate. During horizontal blanking intervals, the user should allow enough time for all programmed register, colormap, and VU data transfers to complete.
3. When digitizing (capturing) images, no other bitmap transfers (e.g., REGX,VU) should be scheduled to occur during the active portion of the field.
4. Active start and stop times should not be programmed to overlap the blanking stop and start times, taking the pipeline delay through the 82750DB into account.
5. Programming the Y interpolation to occur in a non-integral pixel width will cause the Y channel to output incorrect data.

CURSOR REGISTERS

The following registers are used to program the characteristics of the on-chip cursor.

Cursor Position Update Register 0x5b

31 24	23 12	11 0
01011011	Vertical Position	Horizontal Position

— Horizontal Position in units of T-cycles
— Vertical Position in units of full lines

This register gives the horizontal and vertical position of the cursor. The cursor will extend 16-pixel periods, starting at the prescribed horizontal position, for the next 16 lines. (Or 32-pixel periods for 32 lines if the 2X Cursor Mode bits in the General Control register are set to one.

Cursor Control Register 0x5a

31 24	23 12	11 0
01011010	Vertical Position	Horizontal Position

— Horizontal Position in units of T-cycles
— Vertical Position in units of full lines

This register also gives the horizontal and vertical position of the cursor. The cursor will extend 16-pixel periods, starting at the prescribed horizontal position, for the next 16 lines. (Or 32-pixel periods for 32 lines if the 2X Cursor Mode bits in the General Control register are set to one.) Receipt of this address also causes the 82750DB to interpret the next sixteen 32-bit words of register data as the 16 x 16 x 2-bit cursor map. This will cause the register address decoding logic internal to the 82750DB to be disabled, and the next 16 words of information will be loaded into the Cursor table. Each 32-bit word will be interpreted as a line (16 pixels) of cursor data, with the two least significant bits corresponding to the first cursor pixel to be displayed.

Cursor Color 3 0x59

31 24	23 16	15 8	7 0
01011001	Blue/U Color	Red/V Color	Green/Y Color

If the cursor is enabled and the 24 bits of data in this register are selected, the data will be sent directly to the YUV conversion matrix during active display. The bits should be programmed as RGB values when the YUV to RGB matrix is not being used.

Cursor Color 2 0x58

31 24	23 16	15 8	7 0
01011000	Blue/U Color	Red/V Color	Green/Y Color

If the cursor is enabled and the 24 bits of data in this register are selected, the data will be sent directly to the YUV conversion matrix during active display. The bits should be programmed as RGB values when the YUV to RGB matrix is not being used.

Cursor Color 1 0x57

31 24	23 16	15 8	7 0
01010111	Blue/U Color	Red/V Color	Green/Y Color

If the cursor is enabled and the 24 bits of data in this register are selected, the data will be sent directly to the YUV conversion matrix during active display. The bits should be programmed as RGB values when the YUV to RGB matrix is not being used.

DISPLAY TIMING REGISTERS

Each register has two, 12-bit components, listed with least significant bits first, followed by the 12 most significant bits. Horizontal timing is measured in units of T-cycles (periods of the master clock) from the start of horizontal sync. The register content defines the number of T-cycles that elapse before the event controlled by this register takes place. The exception to this rule is the base counter, which specifies the number of T-cycles/half line. Zero is not an allowable value; use the total number of T-cycles per half line or full line instead. Unused bits should be zero. Sync signals are RESET to initial values as specified for each; "start" means to set to 1, and "stop" means to be reset to zero.

Base Counter 0x56

31 24	23 12	11 0
01010110	# of Lines/Field	#of T-Cycles/Half Lines

— T-cycles/Hal Line in units of T-cycles (Periods of the master Clock)
— Half Lines/Field in units of half lines

As defined by NTSC standards, vertical timing can be measured from the start of a field in one of two ways: either in units of half lines, or in units of full lines. When programmed for an interlaced display, (i.e. an odd number of half lines per field) the start of a field coincides with the start of a line on odd fields and with the midpoint of a line on even fields. In the latter case, for an event that is programmed in full lines, the first half line is ignored, and counting begins with the first full line. With this interpretation, the register content defines the number of half or full lines that elapse before the event controlled by this register takes place. The same may be said for the horizontal component, which is defined by the number of T-cycles/half line. The hardware does not look for nor correct illogical combinations of register settings. The monitor should be protected from damage with external circuitry when debugging is in progress.

All of the internal timing is derived from comparing the programmed values with the values of this register. The horizontal base counter is programmed using the least significant 12 bits. In this case the values loaded into this register should be one less than the desired value. Bits 23 through 12 are used to specify the number of half lines per field.

Sync Stops 0x55

31 24	23 12	11 0
01010101	VSYNC Stop	HSYNC Stop

— HSYNC Stop in units of T-cycles
— VSYNC Stop in units of half lines

Sync Starts 0x54

31 24	23 12	11 0
01010100	VSYNC Start	HSYNC Start

— HSYNC Start in units of T-cycles
— VSYNC Start in units of half lines

The Sync Stops and Sync Starts registers are used in conjunction with one another to specify the start and stop locations of the horizontal sync, HSYNC, and vertical sync, VSYNC, output signals. VSYNC may be programmed to start and stop at any time during a given field as defined on a half-line interval. Bits 23 through 12 in the Sync Starts and Sync Stops registers are used to define the start and stop times for VSYNC, respectively. Similarly, HSYNC may be programmed to start and stop at any line position as defined in units of T-cycles. Bits 11 through 0 in the Sync Starts and Sync Stops registers are used to define the start and stop positions for HSYNC, respectively.

The horizontal component of the Sync Stops register also affects the composite sync, of CSYNC output. In this case, the CSYNC output will be the same as the HSYNC output, except during the vertical sync and equalization interval. In the latter case, the CSYNC output is determined by the Serration and Equalization registers.

Blanking Stops 0x53

31 24	23 12	11 0
01010011	Vertical Blank Stop	Horizontal Blank Stop

— HB Stop in units of T-cycles
— VB Stop in units of half lines

The Blanking Start and Stop registers control the composite blanking output (CB). The horizontal blanking start and stop position, in units of T-cycles, can be specified to occur at any time during the line. By the same token, the vertical blanking start and stop positions can be programmed to occur at any half-line interval.

The CB output combines both the horizontal and vertical blanking pulses programmed using these two registers. This information is independent from the HSYNC, VSYNC, and CSYNC outputs, so the user must specify the proper blanking intervals for the monitor that is being used. If the programmer specifies the blanking period to end before the active line starts, or start after the active line has ended, the border color is output. Due to internal pipeline delays on the 82750DB, the values should be one less than desired for VB Start and Stop. For HB Start and Stop subtract the total horizontal pipeline delay.

Blanking Starts 0x52

31 24	23 12	11 0
01010010	Vertical Blank Start	Horizontal Blank Start

— HB Start in units of T-cycles Resets to 1
— VB Start in units of half lines Resets to 1

Program values one less than desired for VB Start and Stop. For horizontal blanking start, load numbers less than the total horizontal pipeline delay.

Serration Start 0x51

31 24	23 12	11 0
01010001	Not Used	Serration Start

— SER Start in units of T-cycles Resets to 0
— (not used)

The vertical component of the CSYNC (composite sync) signal is made up of two types of pulses: equalization and serration pulses. The window during which the serration pulses are active, is determined by the VSYNC start and stop positions, as shown in Figure 4-1. When vertical sync (VSYNC) is active, in this case on line 3, the first serration pulse is output on the CSYNC signal. This pulse will start at the T-cycle count specified in Bits 11 to 0 of the Serration Start register. The pulse will end when the half-line count specified in the Base Counter register has been reached. This pulse will be repeated for every half line that the VSYNC output is programmed to be active, regardless of the position in the field. In Figure 4-1, this continues until half line 12, or line 6.

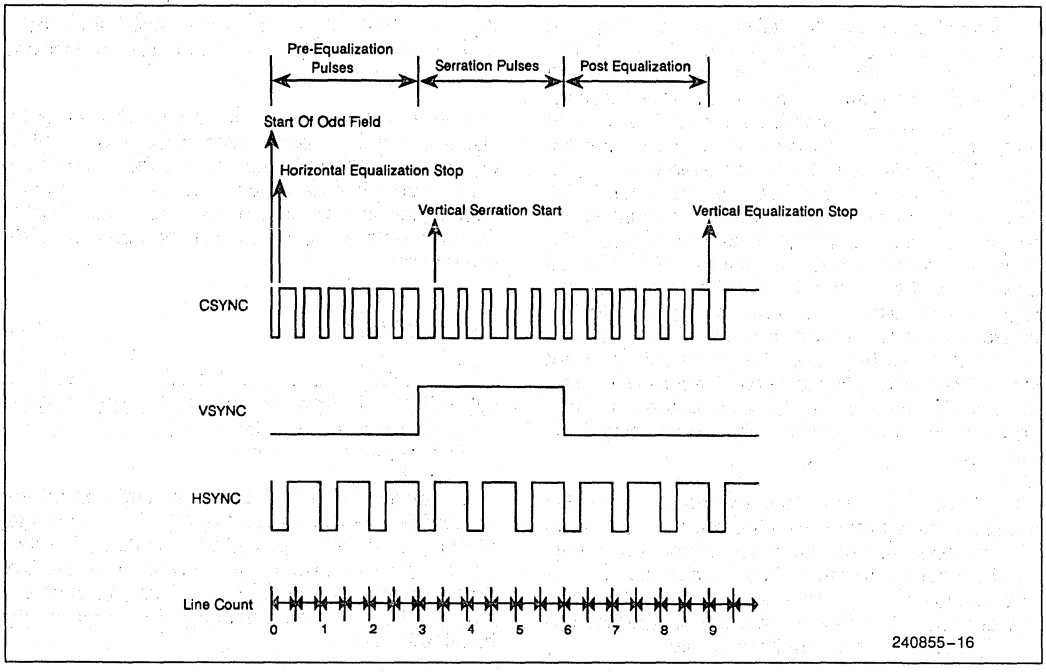


Figure 4-1. Programming the Video Sync Outputs

Equalization Parameters 0x50

31 24	23 12	11 0
0 1 0 1 0 0 0 0	Vertical Equalization Stop	Horizontal Equalization Stop

— EQH Stop in units of T-cycles Resets to 1
— EQV Stop in units of half lines Resets to 1

During the vertical equalizing period, which starts at field-beginning, an equalization pulse is output on the CSYNC signal at the beginning of each half line, as shown in Figure 4-1. The width of this equalization pulse is determined by the value in bits 11 to 0 of this register. The half line on which these pulses are to stop is programmed in bits 23 through 12 of this register. If VSYNC is programmed to occur during the equalization interval (as it is for NTSC type displays), the serration pulses are output on the CSYNC signal.

Active Region Stops 0x4f

31 24	23 12	11 0
0 1 0 0 1 1 1 1	Vertical Active Stop	Horizontal Active Stop

— Actdis Stop in units of T-cycles
— Vertical Stop in units of full lines

The active region window, during which pixels to be displayed are fetched from VRAM, is defined by the Active Region Start and Stop registers. The first display line is actually five lines after the line indicated in the vertical region of the Active Region Start register. The position of the active region on a horizontal line is determined by the horizontal component of the Active Region Start register. Pixels will be fetched from VRAM at a rate determined by the number of bits/pixel and pixel widths. In order for the 82750DB to operate properly, the horizontal width of the active region window must be an integral number of display pixel widths, taking into account the horizontal pipeline delay. Also, the Active Region Start and Stop must fall within a single line boundary, as dictated by the Base Counter register. When the first pixel actually appears at the output of the 82750DB, the output is a function of the processing elements used as discussed above.

When the active region is over, the border color is output until the programmed blanking time is reached. Both the border and blanking information is output at the transfer rate programmed by the user.

Active Region Starts 0x4e

31 24	23 12	11 0
0 1 0 0 1 1 1 0	Vertical Active Start	Horizontal Active Start

— Actdis Start in units of T-cycles
— Vertical Start in units of full lines

Burst Gate Stop 0x4d

31 24	23 12	11 0
0 1 0 0 1 1 0 1	Vertical BG Stop	Horizontal BG Stop

— Horizontal Stop Position in units of T-cycles
— Vertical Stop Position in units of full lines

The Burst Gate Horizontal and Vertical Start and Stop registers allow the user to program a window into which burst can be added. This is useful when modulating the outputs of the 82750DB.

Burst Gate Start 0x4c

31 24	23 12	11 0
0 1 0 0 1 1 0 0	Vertical BG Start	Horizontal BG Start

— Horizontal Start Position in units of T-cycles
— Vertical Start Position in units of full lines

VBUS CODE REGISTERS

The following group of registers are used by the programmer to schedule when VBUS transfer or control codes are to be sent to the 82750PB by the 82750DB.

Display Format Load Interrupt 0x4b

31 24	23 12	11 0
0 1 0 0 1 0 1 1	Vertical DFL Position	Horizontal DFL Position

— Horizontal Position in units of T-cycles
— Vertical Position in units of full lines

This is the programmable XY interrupt, used by the 82750PB to perform a load of the Shadow Copy registers. This interrupt is sent on the VBUS when the bits 23 to 12 match the current display line position, and bits 11 to 0 match the T-cycle count.

Line Notification Timing 0x4a

31 24	23 12	11 0
01001010	Not Used	Horizontal HLIN Position

— HLIN timing in units of T-cycles
— Not Used

This indicates the position on each line to send a HLINE code on the VBUS. The 82750PB requires this information to keep track of the current display line when drawing graphics.

Refresh and Register Transfer 0x49

31 24	23 12	11 0
01001001	REGX Line Number	Refresh Horizontal Position

— REFRESH horizontal timing in units of T-cycles
— Register Transfer Line number in units of full lines

When the T-cycle count matches the value programmed into bit 11 to 0 of this register, a refresh code is sent to the 82750PB. Since these codes tie up the 82750PB for at least eight 82750PB cycles, the programmer must ensure that no transfer requests are scheduled to occur during this time.

The line number for the next register transfer is specified in bits 23 to 12 of this register. If programmed to occur, REGX will always be the first transfer request sent to the 82750PB, immediately after the end of active display.

COLOR REGISTERS

The following registers specify the state of DBU, DRV, DGY, and ALPHA signals during the field.

Border Color 0x48

31 24	23 16	15 8	7 0
01001000	Blue/U Color	Red/V Color	Green/Y Color

The 24 bits of data in this register are sent directly to the YUV conversion matrix during border time. Border time is defined as the region in which neither active display nor blanking is programmed to occur. The bits should be programmed as RGB values when the YUV to RGB matrix is not being used.

Alpha Register 0x47

31 24	23 16	15 8	7 0
01000111	Border Alpha	Alpha1 Register	Alpha0 Register

The least significant 8 bits are for the ALPHA0 register and are used during blanking and if the alpha trap value is not matched. The next 8 bits are for the ALPHA1 register when the alpha trap value is matched. The most significant 8 bits provide the alpha channel value during the border time.

Blanking Color 0x46

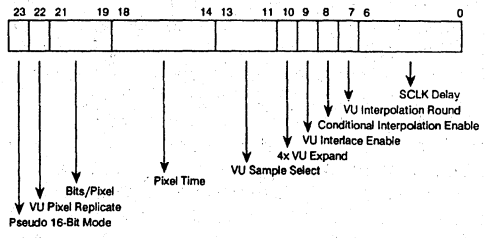
31 24	23 16	15 8	7 0
01000110	Blue/U Color	Red/V Color	Green/Y Color

The 24 bits of data in this register are sent directly through the YUV conversion matrix during the programmed blanking time.

CONTROL REGISTERS

The following registers are used to define the operating modes of the 82750DB.

Pixel Control 0x45



240855-17

Bits 6:0—SCLK Delay

The number "m" of T-cycles from initiation of a transfer request on the VBUS until the first SCLK is asserted by the 82750DB.

Bit 7—VU Interpolation Round

When equal to 0, this bit means truncate during interpolation. When set to one, this bit means round to odd during interpolation.

Bit 8—Conditional Interpolation Enable

When reset to zero, this bit means all values of Y and U are a full 8 bits of precision. When set to one, this bit means the least bit of the Y sample or the U sample controls the switching between VU interpolation and graphics mode.

Bit 9—VU Interlace Enable

Setting this bit to a one causes the interpolator to output different data on the odd and even fields. During the odd field, the odd lines of the interpolation sequence will be output. During the even field, the even lines of the interpolation sequence will be output. Full lines of the programmed number of samples of both the V and U data will be read in during each VU transfer. Setting this bit to a zero will cause horizontally and vertically interpolated data to be output on both fields. Only a full line of either V or U samples will be read in during each transfer request in this mode.

Bit 10—4X VU Expand

When this bit is set to a zero, a 2X expansion in both directions is performed. By setting this bit to a one, a 4X expansion is performed.

Bits 13:11—VU Sample Select

Table 4-1 provides the code and number of V and U samples for bits 13:11.

Table 4-1. VU Sampling

Code	Number of V And U Samples
000	0 Samples for Each V and U
111	32 Samples for Each V and U
110	64 Samples for Each of V and U
101	96 Samples for Each of V and U
100	128 Samples for Each of V and U
011	160 Samples for Each of V and U
010	192 Samples for Each of V and U
001	256 Samples for Each of V and U

Bits 18:14—Pixel Time

Table 4-2 lists the codes and pixel duration for bits 18:14.

Table 4-2. Pixel Times

Code	Duration of Pixel
00001	1.0 T-cycle
00010	1.5 T-cycles
00100	2.0 T-cycles
01000	2.5 T-cycles
10000	3.0 T-cycles
10001	3.5 T-cycles
10010	4.0 T-cycles
10100	4.5 T-cycles
11000	5.0 T-cycles
11001	5.5 T-cycles
11010	6.0 T-cycles
11100	6.5 T-cycles
11101	7.0 T-cycles
11110	7.5 T-cycles
00011	8.0 T-cycles
00101	8.5 T-cycles
00110	9.0 T-cycles
00111	9.5 T-cycles
01001	10.0 T-cycles
01010	10.5 T-cycles
01011	11.0 T-cycles
01100	11.5 T-cycles
01101	12.0 T-cycles
01110	13.0 T-cycles
01111	14.0 T-cycles

Bits 21:19—Bits/Pixel

Table 4-3 provides the code and number of bits/pixel for bits 21:19.

Table 4-3. Number of Bits/Pixel

Code	Number of Bits/Pixel
001	8
010	16
100	32

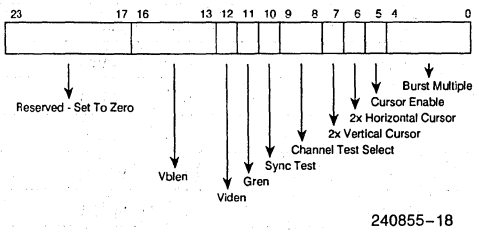
Bit 22—VU Pixel Replicate

When set to one, each pixel generated by the VU Interpolator is held for 2-pixel times. This allows an effective 8X expansion of VU data. This is useful for high resolution applications where the blanking time is not sufficient to support higher VU sample loads.

Bit 23—Pseudo 16-Bit Mode

When set to one and 16 bits per pixel is chosen (bits 21:19), the 82750DB is in the 16-bit with Alpha mode. Setting this signal to zero while in the 16-bit/pixel mode puts the 82750DB into the 16-bit (655) mode. This bit represents a "don't care" input for all other values of bit/pixel.

General Control **0x44**



Bits 4:0—Burst Multiple

These bits are used to program a divisor of the FREQIN clock input in order to recover the 3.58 MHz NTSC color subcarrier. The programmed value is the two's complement of the desired divisor. The allowed range of values is 00001 through 11111 which corresponds to divisions of 31 through 1. Note that the 82750DB must be operating at an integer multiple of 3.58 MHz for this to work effectively.

Bit 5—Cursor Enable

When set to one, the hardware cursor will output the cursor data at prescribed intervals if programmed to do so.

Bit 6—2X Horizontal Cursor

When this bit is set to one, and the Cursor Enable bit is set to one, every pixel on each line of the cursor will be replicated once. Thus a cursor that was 16 x 16 pixels will become 32 x 16 pixels.

Bit 7—2X Vertical Cursor

When this bit is set to one, and the Cursor Enable bit is set to one, each line of the cursor will be replicated once. Thus a cursor that was 16 x 16 pixels will become a 16 x 32-pixel cursor.

Bit 9:8—Channel Select

These two bits control which output channel is muxed onto the alpha digital outputs. It allows Y, U, or V data to be available at the alpha channel. The coding is provided in Table 4-4.

Table 4-4. Test Mode Select Coding

Code	Alpha Channel Output
00	Alpha Channel
01	Y Channel
10	V Channel
11	U Channel

Bit 10—Sync Test

This bit must be set to zero for proper operation.

Bit 11—Gren

This is the Graphics Enable bit for the Y Interpolator. When this bit is set to one and the pixel is a graphics pixel, switch is zero, a 2X interpolation will be performed on the pixel.

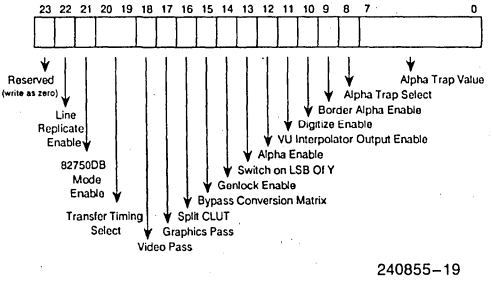
Bit 12—Viden

This is the Video Enable bit of the Y Interpolator. When this bit is set to one and the pixel is a video pixel, switch is one, a 2X interpolation will be performed on the pixel.

Bit 16:13—Vblen

These bits program the T-cycle length of each VBUS code. The VBUS code length will be one T-cycle longer than the programmed value. These bits must have a minimum value of 2, and a maximum value of 15.

Miscellaneous Control **0x43**



240855-19

Bits 7:0—Alpha Trap

Bits 7:0 are 8-bit values used for comparison with the current pixel's Y value, to select one of two programmable alpha values.

Bit 8—Alpha Trap Select

A value of one enables the Y value of the current pixel to be compared with the value in the Alpha Trap register. If the two values match and Alpha has been enabled via the Alpha Enable bit, the contents of the ALPHA1 register are output on ALPHA[7:0]. If the two values don't match and Alpha Enable has been set to one, the content of the ALPHA0 register is output. When Alpha Trap Select is set to a zero in the pseudo 16- or 32-bit mode, the most significant byte of the pixel word is output. When Alpha Trap Select is set to zero in all other modes, the value of the ALPHA0 register is output.

Bit 9—Border Alpha Enable

A value of one enables the eight most significant bits in the ALPHA register to be output. When set to a zero, the ALPHA0 register is output during border time.

Bit 10—Digitize Enable

When this bit is set to a one, the FCO signal will be set to a one, and the transfer codes for bitmaps will indicate that write operations should occur.

Bit 11—VU Interpolator Output Enable

This bit enables VU interpolation data to be displayed. When set to a zero, all pixels are treated as graphic pixels.

Bit 12—Alpha Enable

When set to one, the alpha output is governed by the alpha trap value, as described above. When reset to zero, the contents of the ALPHA0 register is the alpha output in the 8- and 16-bit modes, and the explicit ALPHA data encoded in the pseudo 16- and 32-bit modes.

Bit 13—Switch on LS Bit of Y

When set to one, the least significant bit of Y is used as a Video/Graphics switch in all modes. When reset to zero, the least significant bit of U from the interpolator acts as a switch.

Bit 14—Genlock Enable

This bit enables the genlock mode of the 82750DB. In this mode, receipt of the external HRESET# signal during the second half of a scan line will cause the termination of that scan line. Similarly, receipt of the externally produced VRESET# signal will terminate the field. In both cases, terminate denotes that the proper on-chip signals are produced to signify end of the line and end of the field.

Bit 15—Bypass Conversion Matrix

When this bit is set to a one the YUV to RGB matrix will be bypassed, and the Y, U, and V data will feed directly into the Digital to Analog Converters.

Bit 16—Split CLUT

This bit divides the CLUT into an odd and an even half, depending on the polarity of the Video/Graphics switch. This switch is selectable and may be either the LSB of U from the interpolator or Y from the pixel word. The LSB of the CLUT address is set to one (odd address) if the Video/Graphics switch is one; the LSB of the CLUT address is set to zero (even address) if the Video/Graphics switch is zero.

Bit 17—Graphics Pass

Setting this bit to a one bypasses the CLUT for graphics pixels, even in non-mixed modes.

Bit 18—Video Pass

When set to a one, all video pixels (luminance values associated with sub-sampled UV values) will bypass the color table. For mixed modes, this corresponds to the switch flag having a value of one.

Bit 20:19—Transfer Timing Select

These bits are two-bit codes that select one of three possible transfer shift clock rates. This allows the operating speed of the 82750DB to be tailored to the external memory access time. After RESET, the transfer rate is set to the slowest possible clock rate (1/3X). The programmed rate is used during all non-active display times for transferring data from VRAMs. It also defines the rate that the border and blanking data is output. During active display, the data is read as needed from VRAM using the programmed timing. The coding of these bits is listed in Table 4-5.

Table 4-5. Coding of Transfer Timing Select Bits

Bit 20	Bit 19	Result
0	0	1/3X Transfer (Default)
0	1	1/2X Transfer
1	0	1X Transfer

Bit 21—82750DB Enable

When set to zero, the 82750DB will be the register equivalent of a 82750DA. When set to a one all the features of the 82750DB will be enabled.

Bit 22—Line Replicate Enable

When this bit is set to one, every line in the active display is generated twice. Each new bitmap transfer occurs at half the line rate, with a new VBUS code being used to indicate that a transfer is to take place without the pitch calculation. The VU Interpolator will also duplicate the lines it generates, yielding more time between transfer cycles. This mode is useful for obtaining a 2X increase in vertical resolution without the need for increasing the VRAM transfer bandwidth.

COLOR MAP REGISTERS

The following registers are used to access and control the three 256 x 8-bit Color Lookup Tables.

Mask Data Registers 0x42

31 24	23 16	15 8	7 0
01000010	Blue/U Mask Data	Red/V Mask Data	Green/Y Mask Data

Each of the three 8-bit registers contains the bit pattern used when the corresponding bit in the Mask Set register is asserted.

Mask Set Registers 0X41

31 24	23 16	15 8	7 0
01000001	Blue/U Color	Red/V Color	Green/Y Color

This is a 24-bit register that contains the mask bit pattern for the RGB/YUV color map addresses. When a bit in this register is asserted, the corresponding bit in the address is set to the value defined in the Mask Data registers.

CLUT Index Register 0x40

31 24	23 16	15 8	7 0
01000000	Not Used	Not Used	YUV CLUT Index

The CLUT Index register is an 8-bit register used for loading the color tables. This register maps the user-specified 6-bit color map address into an 8-bit address. A logical OR operation is performed between the 6-bit address and the 8-bit index word to obtain the new CLUT address.

Color Lookup Table Addresses 0x00–0x3f

If the 82750DB Enable mode bit in the Miscellaneous Control register is set to zero, the CLUT addresses are decoded to appear as addresses to the reduced-size 82750DA color table. The least significant four bits of the address are used for the Y color table address, and the upper nibble is used to address the V and U color table simultaneously. This is a compatibility mode for the 82750DA, which has a reduced-size color table.

31 28	27 24	23 16	15 8	7 0
UV Address	Y Address	U Data	V Data	Y Data

If the 82750DB Enable mode bit is set to one, the full color table is used. In this case, the most significant byte of the 32-bit data word is used as an address to the color table. The address is ORed with the most recently loaded CLUT Index register.

31 30	29 24	23 16	15 8	7 0
0 0	YUV Address	U Data	V Data	Y Data

82750DB Register Summary

The following table illustrates the register space of the 82750DB.

Table 4-6. 82750DB Register Space

Address	82750DB Register
0x00–0x0f	CLUT Locations 0–15
0x10–0x30	CLUT Locations 16–48
0x31	CLUT Location 49
0x32	CLUT Location 50
0x33	CLUT Location 51
0x34	CLUT Location 52
0x35–0x37	CLUT Location 53–55
0x38	CLUT Location 56
0x39–0x3f	CLUT Location 57–63
0x40	CLUT Index Register
0x41	CLUT Mask Set Register
0x42	CLUT Mask Data Register
0x43	Miscellaneous Control
0x44	General Control
0x45	Pixel Control
0x46	Blanking Color
0x47	Alpha Register
0x48	Border Color
0x49	Register Transfer
0x4a	Line Notification and Timing
0x4b	DFL Load
0x4c	Burst Gate Start
0x4d	Burst Gate Stop
0x4e	Active Region Start
0x4f	Active Region Stop
0x50	Equalization Parameters
0x51	Serration Start
0x52	Blanking Start

Address	82750DB Register
0x53	Blanking Stop
0x54	Sync Start
0x55	Sync Stop
0x56	Base Counters
0x57	Cursor Color 1
0x58	Cursor Color 2
0x59	Cursor Color 3
0x5a	Cursor Control
0x5b	Not Used
0x5c	Not Used
0x5d	Not Used
0x5e	Not Used
0x5f	Not Used
0x60	Not Used
0x61	Not Used
0x62	Not Used
0x63	Not Used
0x64	Not Used
0x65	Not Used
0x66	Not Used
0x67	Not Used
0x68	Not Used
0x69–0x6e	Not Used
0x6f	Not Used
0x70	Not Used
0x71–0x7f	Not Used
0x80–0xfe	Not Used
0xff	Stop Code

5.0 ELECTRICAL DATA

Maximum Ratings

Table 5-1 is a stress rating only, and functional operation at the maximums is not guaranteed. Functional operating conditions are given in the DC and AC Characteristics (Tables 5-2, 5-3, 5-4, and 5-5).

Exposure to the Maximum Ratings may affect device reliability. Furthermore, although the 82750DB contains protective circuitry to resist damage from static electrical discharge, always take precautions to avoid high static voltages or electric fields.

Table 5-1. Absolute Maximum Requirements

Condition	Maximum Requirement
Case Temperature under Bias	−65°C to 110°C
Storage Temperature	−65°C to 110°C
Voltage on Any Pin with Respect to Ground	−0.5V to V_{CC} + 0.5V
Supply Voltage with Respect to V_{SS}	−0.5V to +6.5V

DC Characteristics

Table 5-2. DC Characteristics V_{CC} = 5V ±10%, T_{CASE} = 0°C to 95°C

Symbol	Parameter	Min	Typ	Max	Unit	Notes
V_{IL}	Input LOW Voltage	−0.3		0.8	V	
V_{IH}	Input HIGH Voltage	2.0		V_{CC} + 0.3	V	
V_{OL}	Output LOW Voltage		0.2	0.4	V	I_{OL} = 4.0 mA(1)
V_{OH}	Output HIGH Voltage	2.4	3.0		V	I_{OH} = −1.0 mA(1)
I_{IL}	Input Leakage Current	−10		+10	µA	V_{SS}<V_{IN}<V_{CC}
I_{OZ}	Output Leakage Current	−10		+10	µA	V_{SS}<V_{IN}<V_{CC}
I_{CCT}	Power Supply Current		185	250	mA	28 MHz(2)
I_{CCNT}	Power Supply Current		140	190	mA	28 MHz(3)
I_{CCT}	Power Supply Current		280	375	mA	45 MHz(2)
I_{CCNT}	Power Supply Current		215	285	mA	45 MHz(3)
C_{IN}	Input Capacitance			10.0	pF	F_C = 1 MHz(4)
C_{OUT}	Output Capacitance			12.0	pF	F_C = 1 MHz(4)
C_{FREQIN}	FREQIN Input Capacitance			20.0	pF	F_C = 1 MHz(4)

NOTES:
1. Measured with FREQIN = 7 MHz.
2. Typical current value measured under typical conditions with the Digital Outputs (DGY, DRV, and DBU) toggling. Maximum current value guaranteed with 50 pF maximum output loading. Analog Outputs disabled.
3. Typical current value measured under typical conditions with the Digital Outputs (DGY, DRV, and DBU) not toggling. Maximum current value guaranteed with 50 pF maximum output loading. Analog Supply Current IACC not included.
4. Not 100% tested.

AC Characteristics

Table 5-3. AC Characteristics at 28 MHz V_{CC} = 5V ±10%, T_{CASE} = 0°C to 95°C, C_L = 50 pF

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	7	28	MHz		1XClock
t_1	FREQIN Period	35	140	ns	5-1	
t_2	FREQIN High Time	12	23	ns	5-1	(Note 1)
t_3	FREQIN Low Time	12	23	ns	5-1	(Note 1)
t_4	FREQIN Fall Time		4	ns	5-1	
t_5	FREQIN Rise Time		4	ns	5-1	
t_{6a}	HSYNC, VSYNC, CSYNC, BG, FCO Valid Delay		24	ns	5-2	
t_{6b}	VBUS[3:0] Valid Delay		26	ns	5-2	
t_7	RESETB#, VRESET#, HRESET#, DISDIG, TESTACT Setup	0		ns	5-3	
t_8	RESET #, VRESET#, HRESET#, DISDIG, TESTACT Hold	13		ns	5-3	
t_9	SCLK[1:0] Valid Delay High		14	ns	5-4	1X Mode
t_{10}	SCLK[1:0] Valid Delay Low		$1/2t_1 + 14$	ns	5-4	1X Mode
t_{11}	SCLK[1:0] Valid Delay		14	ns	5-5, 5-6	1/2X, 1/3X Mode
t_{12}	DATAIN[31:0] Setup	5		ns	5-4, 5-5, 5-6	
t_{13}	DATAIN[31:0] Hold	5		ns	5-4, 5-5, 5-6	
t_{14}	PIXCLK Valid Delay		$1/2t_1 + 20$	ns	5-7	(Note 2)
t_{15}	PIXCLK Valid Delay		20	ns	5-7	(Note 3)
t_{16}	DRV[7:0], DGY[7:0], DBU[7:0], ALPHA[7:0], ACTDIS, CB, BPP[0], BPP[1] Output Setup	3		ns	5-8	
t_{17}	DRV[7:0], DGY[7:0], DBU[7:0], ALPHA[7:0], ACTDIS, CB, BPP[0], BPP[1] Output Hold	15		ns	5-8	
t_{18}	VBUS[3.0], SCLK[1.0], FCO, HSYNC, VSYNC, CSYNC, CB, BG, PIXCLK, DRV[7:0], DGY[7:0], DBU[7:0], ALPHA[7:0], ACTDIS, BPP[0], BPP[1] Float Delay		30	ns	5-9	(Note 4)
t_{19}	DISDIG, DRV[7:0], DGY[7:0], DBU[7:0], Digital Output Disable Delay	$3t_1$		ns	5-10	
t_{20}	DISDIG, DRV[7:0], DGY[7:0], DBU[7:0], Digital Output Enable Delay	$3t_1$		ns	5-10	
t_{21}	DISDAC, RV, GY, BU Analog Output Disable Delay		19	ns	5-11	(Note 6)
t_{22}	DISDAC, RV, GY, BU Analog Output Enable Delay		19	ns	5-11	(Note 6)

NOTES:
1. This assumes a 35 ns period. For other speeds, the FREQIN High and Low Times should fall within a 40% to 60% duty cycle.
2. For integer pixel times t_{14} is the Valid Delay on all assertions of PIXCLK during active display time.
3. For non-integer pixel times t_{15} is the Valid Delay on alternating assertions of PIXCLK during active display time.
4. Not 100% tested.
5. All A.C. specifications are measured at the 1.5V crossing point with a 50 pF load.
6. Analog output delay is measured at the 50% level of the full scale transition with $R_L = 75\Omega$ and $C_L = 25$ pF.

AC Characteristics

Table 5-4. AC Characteristics at 45 MHz $V_{CC} = 5V \pm 10\%$, $T_{CASE} = 0°C$ to $95°C$, $C_L = 50$ pF

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	7	45	MHz		1XClock
t_1	FREQIN Period	22	140	ns	5-1	
t_2	FREQIN High Time	7	15	ns	5-1	(Note 1)
t_3	FREQIN Low Time	7	15	ns	5-1	(Note 1)
t_4	FREQIN Fall Time		4	ns	5-1	
t_5	FREQIN Rise Time		4	ns	5-1	
t_{6a}	HSYNC, VSYNC, CSYNC, BG, FCO Valid Delay		20	ns	5-2	
t_{6b}	VBUS[3:0] Valid Delay		26	ns	5-2	
t_7	RESETB#, VRESET#, HRESET#, DISDIG, TESTACT Setup	0		ns	5-3	
t_8	RESET B#, VRESET#, HRESET#, DISDIG, TESTACT Hold	13		ns	5-3	
t_9	SCLK[1:0] Valid Delay High		12	ns	5-4	1X Mode
t_{10}	SCLK[1:0] Valid Delay Low		$1/2t_1 + 12$	ns	5-4	1X Mode
t_{11}	SCLK[1:0] Valid Delay		12	ns	5-5, 5-6	1/2X, 1/3X Mode
t_{12}	DATAIN[31:0] Setup	3		ns	5-4, 5-5, 5-6	
t_{13}	DATAIN[31:0] Hold	3		ns	5-4, 5-5, 5-6	
t_{14}	PIXCLK Valid Delay		$1/2t_1 + 20$	ns	5-7	(Note 2)
t_{15}	PIXCLK Valid Delay		20	ns	5-7	(Note 3)
t_{16}	DRV[7:0], DGY[7:0], DBU[7:0], ALPHA[7:0], ACTDIS, CB, BPP[0], BPP[1]/VUGR Output Setup	0		ns	5-8	
t_{17}	DRV[7:0], DGY[7:0], DBU[7:0], ALPHA[7:0], ACTDIS, CB, BPP[0], BPP[1]/VUGR Output Hold	10		ns	5-8	
t_{18}	VBUS[3.0], SCLK[1.0], FCO, HSYNC, VSYNC, DRV[7:0], DGY[7:0], ALPHA[7:0], ACTDIS, BPP[0], BPP[1]/VUGR Float Delay		30	ns	5-9	(Note 4)
t_{19}	DISDIG, DRV[7:0], DGY[7:0], DBU[7:0], Digital Output Disable Delay	$3t_1$		ns	5-10	

AC Characteristics (Continued)

Table 5-4. AC Characteristics at 45 MHz V_{CC} = 5V ±10%, T_{CASE} = 0°C to 95°C, C_L = 50 pF

Symbol	Parameter	Min	Max	Unit	Figure	Notes
t_{20}	DISDIG, DRV[7:0], DGY[7:0], DBU[7:0], Digital Output Enable Delay	$3t_1$		ns	5-10	
t_{21}	DISDAC, RV, GY, BU Analog Output Disable Delay		19	ns	5-11	(Note 6)
t_{22}	DISDAC, RV, GY, BU Analog Output Enable Delay		19	ns	5-11	(Note 6)

NOTES:
1. This assumes a 22 ns period. For other speeds, the FREQIN High and Low Times should fall within a 40% to 60% duty cycle.
2. For integer pixel times t_{14} is the Valid Delay on all assertions of PIXCLK during active display time.
3. For non-integer pixel times t_{15} is the Valid Delay on alternating assertions of PIXCLK during active display time.
4. Not 100% tested.
5. All A.C. specifications are measured at the 1.5V crossing point with a 50 pF load.
6. Analog output delay is measured at the 50% level of the full scale transition with R_L = 75Ω and C_L = 25 pF.

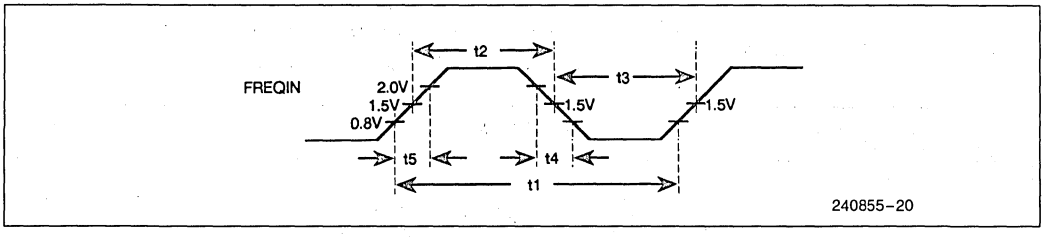


Figure 5-1. Clock Waveforms

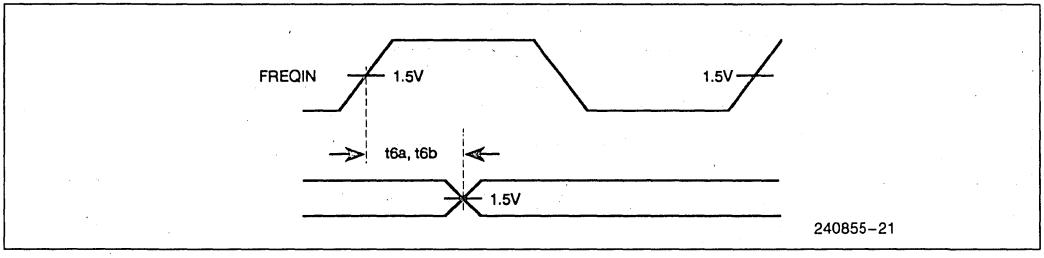


Figure 5-2. Output Waveforms

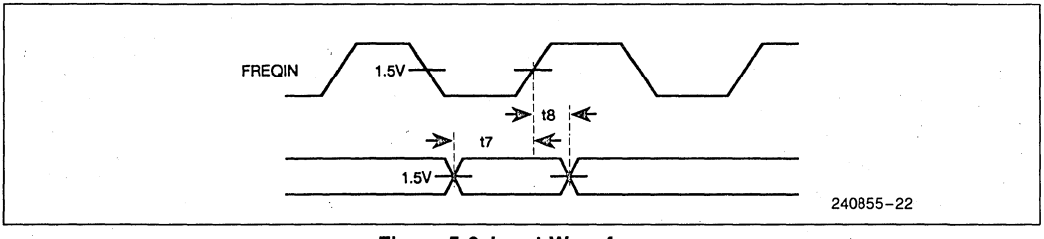


Figure 5-3. Input Waveforms

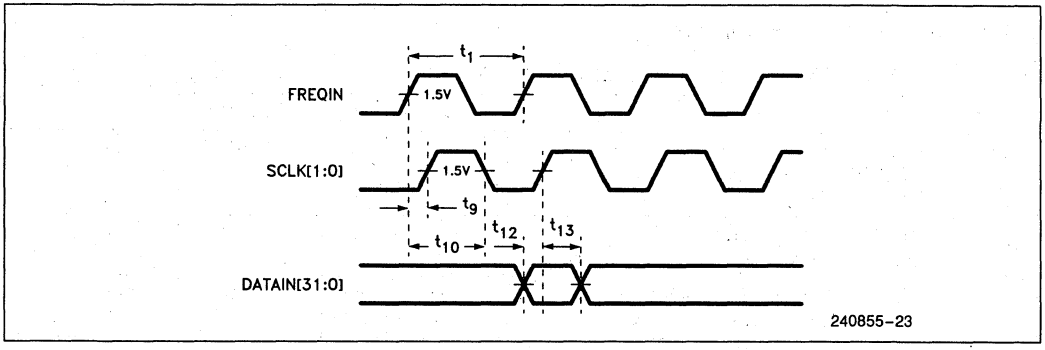


Figure 5-4. 1X SCLK Mode

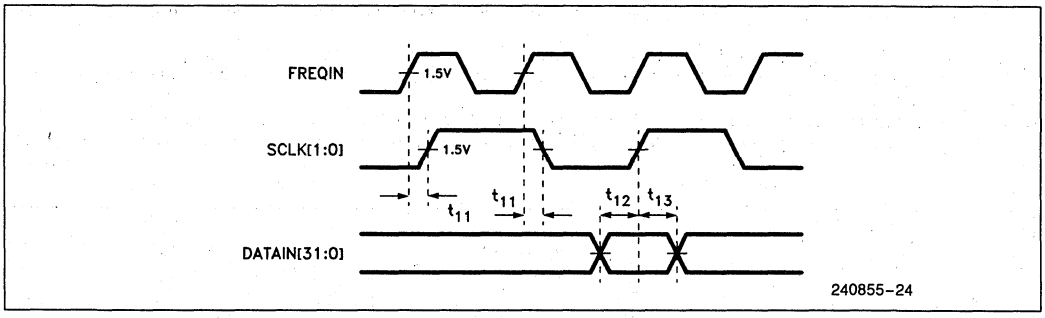


Figure 5-5. 1/2X SCLK Mode

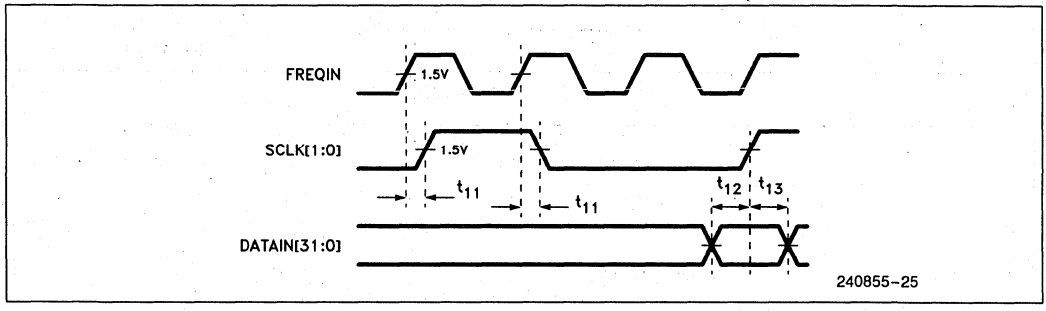


Figure 5-6. 1/3X SCLK Mode

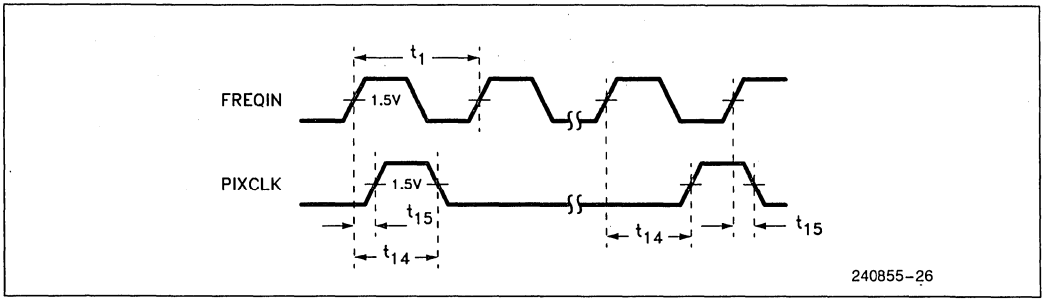


Figure 5-7. PIXCLK Waveforms

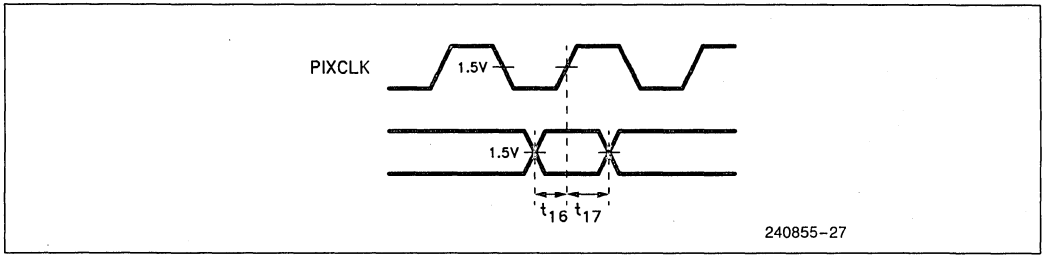


Figure 5-8. Output Setup and Hold

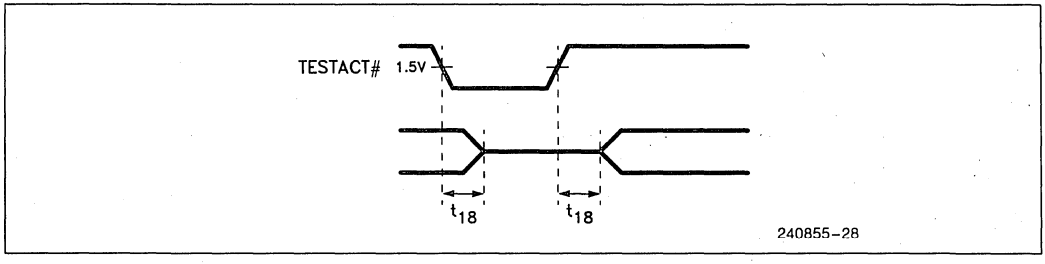


Figure 5-9. TESTACT # Float Delay

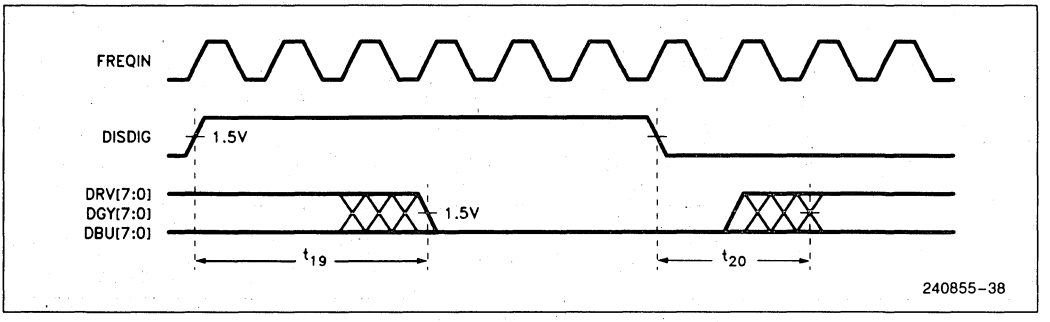


Figure 5-10. DISDIG to Digital Output Delay

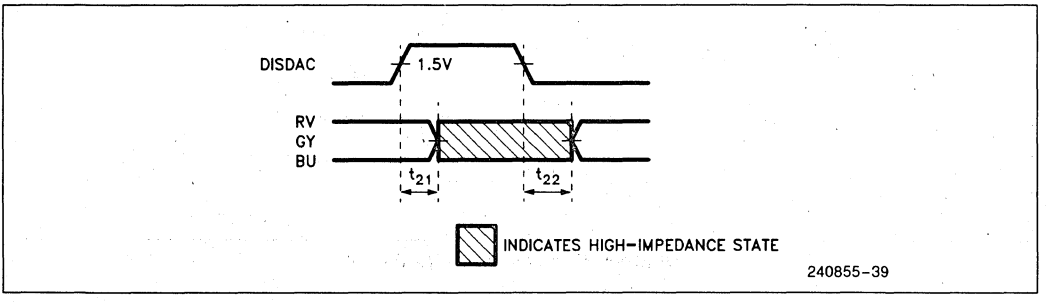


Figure 5-11. DISDAC to Analog Output Delay

Digital to Analog Converter Electrical Characteristics

Table 5-5. DAC D.C. Characteristics AV_{CC} = 5V ±10%; T_{CASE} = 0°C to +95°C

Symbol	Parameter	Min	Typ	Max	Unit	Notes
Iref	Reference Current			1500	µA	
Ifs	Output Current* (Full Scale)	0.93 * (255/18.5) * Iref		1.07 * (255/18.5) * Iref	mA	(Note 1)
Vfs	Output Voltage (Full Scale)		1.0	1.5	V	
INL	Integral Nonlinearity		1.0	±3	LSB	
DNL	Differential Nonlinearity			±1	LSB	
IACC	Analog Supply Current			3 * Ifs + 8	mA	(Note 2)
DDTR	DAC to DAC Tracking at Full Scale		2.0	5.0	%	(Note 3)
Cout	Output Capacitance			12	pF	(Note 4)

NOTES:
1. Maximum Ifs allowed = 22 mA.
2. Maximum IACC allowed = 74 mA. Typical value of IACC = 3 * Ifs + 6
3. Maximum deviation between RV, GY and BU outputs at fullscale output voltage.
4. Not 100% tested.
5. All DAC testing done with Iref = 1500 µA.

Table 5-6. DAC A.C. Characteristics

Symbol	Parameter	Min	Typ	Max	Unit	Notes
tr, tf	Rise/Fall Time			10	ns	(Note 1)
ClkF	Clock Feedthrough		–28		dB	(Note 2)
GlEn	Glitch Energy		100		pV-sec	(Notes 2, 3)
Skew	Output Skew			3	ns	
Xtlk	Crosstalk		200		pV-sec	(Note 2)

NOTES:
1. Maximum value is for R_L = 75Ω and C_L = 25 pF. Defined as 10% to 90% fullscale transmission.
2. Assumes an 80 MHz filter on output.
3. Glitch energy generated from the influence that 2 active outputs have on an idle output.
4. DISDIG must be tied high.
5. Assumes the use of 0.1 µF capacitor between VGCS and AV_{CC} and 0.1 µF and 10 µF capacitors between IREFIN and AV_{CC}.

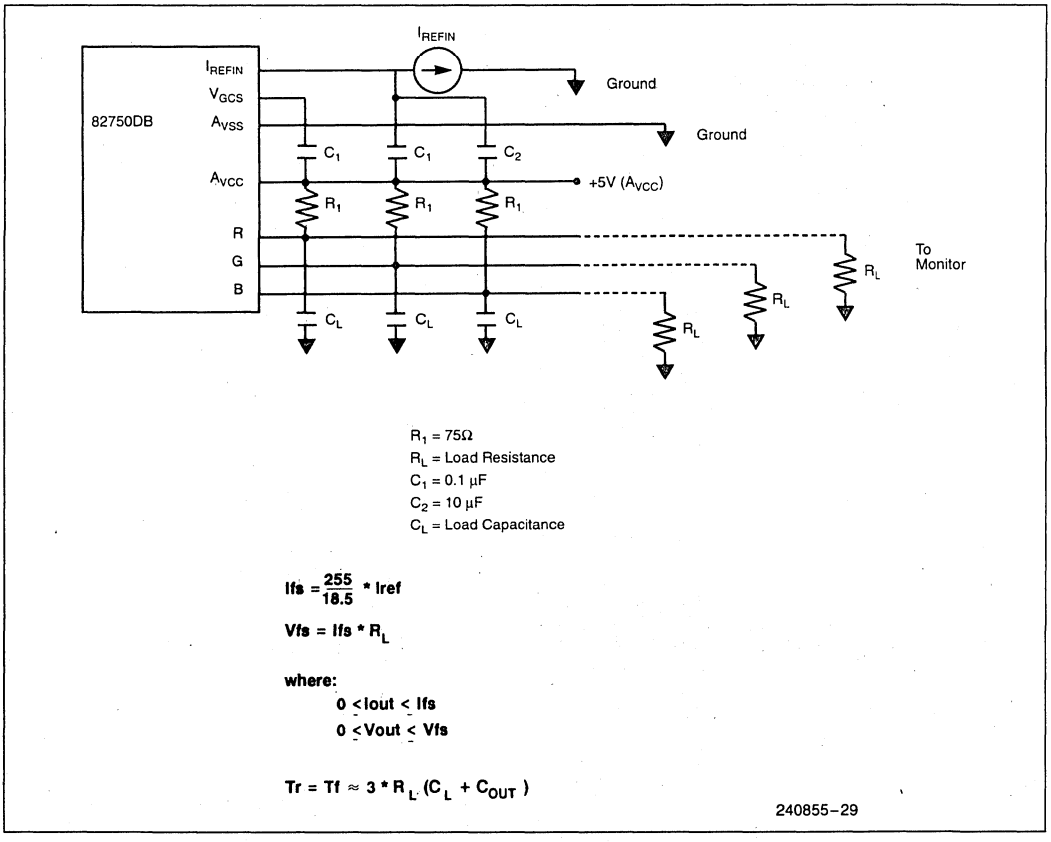
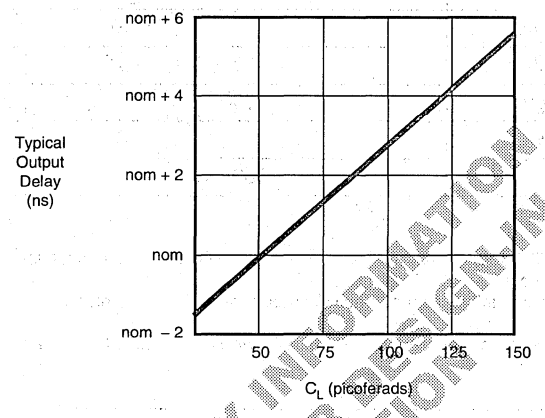


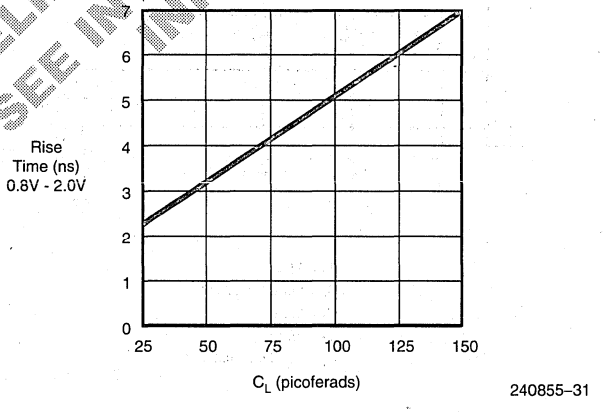
Figure 5-12. Typical Output Configuration

Output Delay and Rise Time versus Load Capacitance



NOTE:
This graph will not be linear outside of the C_L range shown. nom = nominal value given in A.C. Characteristics table.

Figure 5-13. Typical Output Valid Delay versus Load Capacitance under Worst Case Conditions



NOTE:
This graph will not be linear outside of the C_L range shown.

Figure 5-14. Typical Output Rise Time versus Load Capacitance under Worst Case Conditions

6.0 MECHANICAL DATA

Packaging Outlines and Dimensions

Intel packages the 82750DB in a Plastic Quad Flat Pack (PQFP). Table 6-1 gives the symbol list for the PQFP.

Table 6-1. PQFP Symbol List

Letter or Symbol	Description of Dimensions
A	Package Height: Distance from Seating Plane to Highest Point of Body
A_1	Standoff: Distance from Seating Plane to Base Plane
D/E	Overall Package Dimension: Lead Tip to Lead Tip
D_1/E_1	Plastic Body Dimension
D_2/E_2	Bumper Distance
D_3/E_3	Footprint
D_4/E_4	Foot Radius Location
L_1	Foot Length
N	Total Number of Leads

The PQFP has the following specifications:

1. All dimensions and tolerances conform to ANSI Y14.5M-1982.
2. Datum plane-H-is located at the mold parting line and coincident with the bottom of the lead where lead exits plastic body.
3. Datums A-B and -D- are to be determined where center leads exit plastic body at datum plane -H-.
4. Controlling dimension is the inch.
5. Dimensions D_1, D_2, E_1, and E_2 are measured at the mold parting line and do not include mold protrusion. Allowable mold protrusion is 0.18 mm (0.007 in.) per side.
6. Pin 1 identifier is located within one of the two zones indicated.
7. Measured at datum plane -H-.
8. Measured at seating plane datum -C-.

Table 6-2 provides outline characteristics for 0.025-in. pitch.

Table 6-2. Intel Case Outline Drawings for PQFP at 0.025 Inch Pitch

Symbol	Description	Min	Max
N	Leadcount	132	132
A	Package Height	0.160	0.180
A_1	Standoff	0.020	0.040
D,E	Terminal Dimension	1.070	1.090
D_1, E_1	Package Body	0.947	0.953
D_2, E_2	Bumper Distance	1.097	1.103
D_3, E_3	Lead Dimension	0.800 REF	0.800 REF
D_4, E_4	Foot Radius Location	1.023	1.037
L_1	Foot Length	0.020	0.030

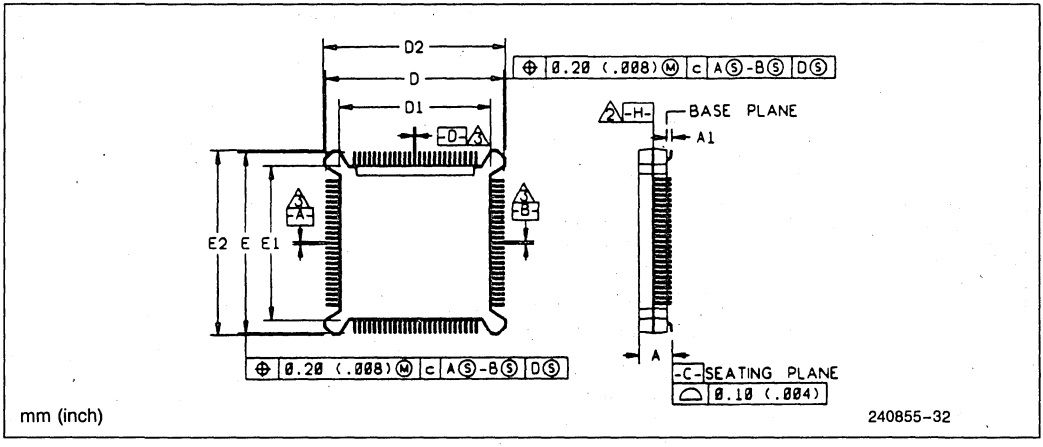


Figure 6-1. Principal Dimensions of the 82750DB in the 132-Lead PQFP Package

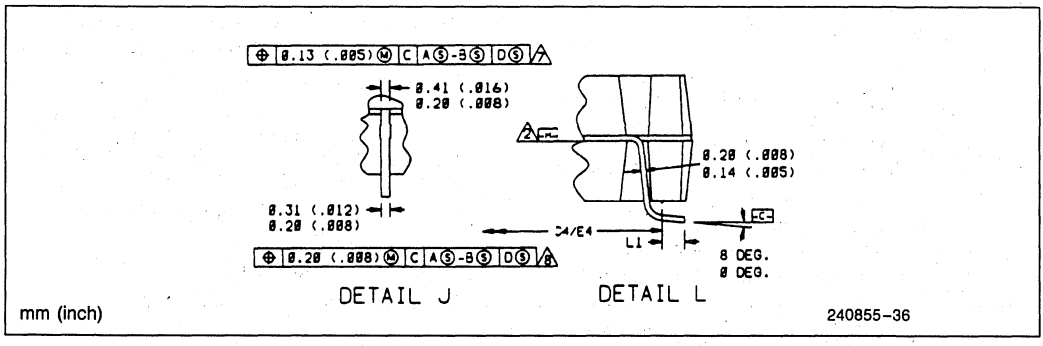


Figure 6-2. 132-Lead PQFP Mechanical Package Detail—Typical Lead

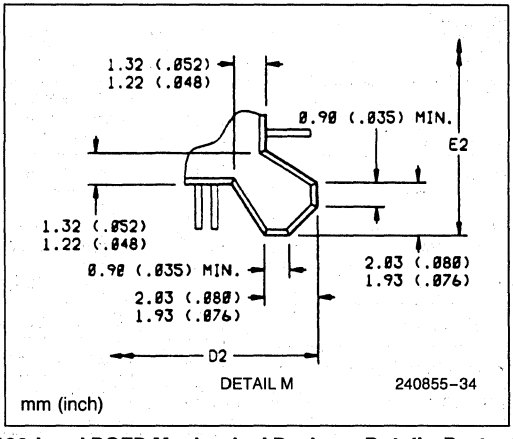


Figure 6-3. 132-Lead PQFP Mechanical Package Detail—Protective Bumper

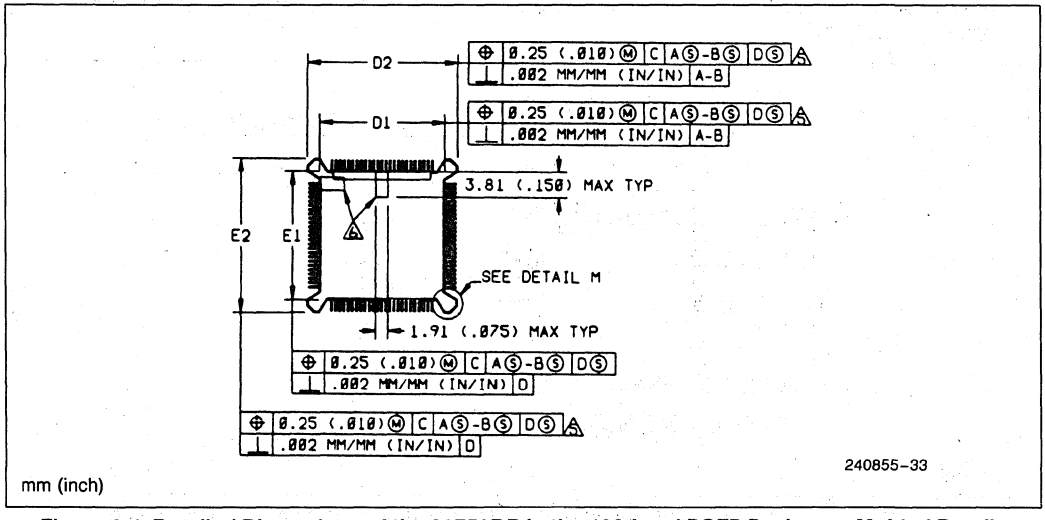
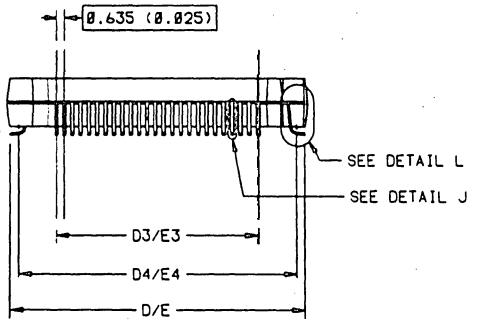


Figure 6-4. Detailed Dimensions of the 82750DB in the 132-Lead PQFP Package—Molded Details



240855-35

mm (inch)

Figure 6-5. Detailed Dimensions of the 82750DB in the 132-Lead PQFP Package—Terminal Details

NOTES:

1. ALL DIMENSIONS AND TOLERANCES CONFORM TO ANSI Y14.5M-1982
2. DATUM PLANE -H- LOCATED AT THE MOLD PARTING LINE AND COINCIDENT WITH THE BOTTOM OF THE LEAD WHERE LEAD EXITS PLASTIC BODY
3. DATUMS A-B AND -D- TO BE DETERMINED WHERE CENTER LEADS EXIT PLASTIC BODY AT DATUM PLANE -H-
4. CONTROLLING DIMENSION, INCH
5. DIMENSIONS D1, D2, E1 AND E2 ARE MEASURED AT THE MOLD PARTING LINE. D1 AND E1 DO NOT INCLUDE AN ALLOWABLE MOLD PROTRUSION OF 0.18 MM (.007 IN) PER SIDE. D2 AND E2 DO NOT INCLUDE A TOTAL ALLOWABLE MOLD PROTRUSION OF 0.18 MM (.007 IN) AT MAXIMUM PACKAGE SIZE.
6. PIN 1 IDENTIFIER IS LOCATED WITHIN ONE OF THE TWO ZONES INDICATED
7. MEASURED AT DATUM PLANE -H-
8. MEASURED AT SEATING PLANE DATUM -C-

240855-37

Package Thermal Specifications

The 82750DB is specified for operation when T_C (the case temperature) is within the range of 0°C to 95°. T_C may be measured in any environment to determine whether the 82750DB is within specified operating range. The case temperature should be measured at the center of the top surface.

T_A (the ambient temperature) can be calculated from θ_{CA} (thermal resistance from case to ambient) with the following equation:

$$T_A = T_C - P * \theta_{CA}$$

Typical values for θ_{CA} at various airflows are given in Table 6-3 for the 132-lead PQFP package. When using the digital outputs, Table 6-4 shows the maximum T_A allowable (without exceeding T_C) at various airflows. The power dissipation (P) is calculated by using the typical supply currents at 5V as shown in Table 5-2.

Similarly, when using the analog outputs, the maximum T_A allowed is a function of Ifs. The equation for calculating the power is given in the following equation which can then be used in calculating the maximum T_A.

$$P = 5V * (I_{CCNT} + (3 * I_{fs} + 6))$$

Table 6-3. Therman Resistances (°C/W)

	θ_{CA} Versus Airflow—ft/min (m/sec)					
Package	**0 (0)**	**200 (1.01)**	**400 (2.02)**	**600 (3.04)**	**800 (4.06)**	**1000 (5.07)**
132-Lead PQFP	26.0	17.5	14.0	11.5	9.5	8.5

Table 6-4. Maximum T_A at Various Airflows (°C)

		T_A Versus Airflow—ft/min (m/sec)					
Package	**Frequency (MHz)**	**0 (0)**	**200 (1.01)**	**400 (2.03)**	**600 (3.04)**	**800 (4.06)**	**1000 (5.07)**
132-Lead PQFP	28	71	79	82	84	86	87
	45	59	71	75	79	82	83

82750PB
PIXEL PROCESSOR

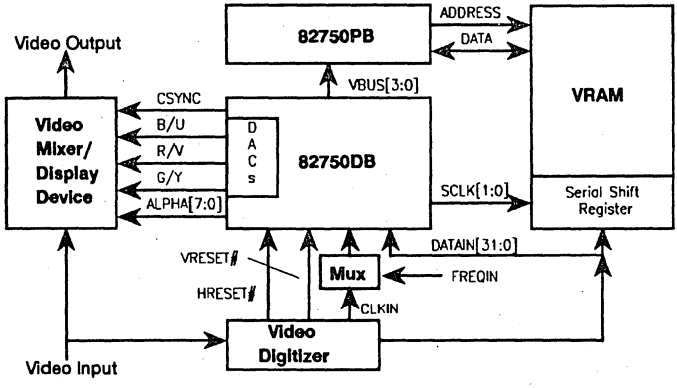
- 25 MHz Clock with Single Cycle Execution
- Zero Branch Delay
- Wide Instruction Word Processor
- 512 x 48-Bit Instruction RAM
- 512 x 16-Bit Data RAM
- Two Internal 16-Bit Buses
- ALU with Dual-Add-With-Saturation Mode
- Variable Length Sequence Decoder
- Pixel Interpolator
- High Performance Memory Interface
  - 32-Bit Memory Data Bus
  - 50 MBytes per Second Maximum
  - 25 MBytes per Second with Standard VRAMs or DRAMs
- 16 General-Purpose Registers
- 4 Gbyte Linear Address Space
- 132-Pin PQFP
- Compatible with the 82750PA

1

Intel's 82750PB is a 25 MHz wide instruction processor that generates and manipulates pixels. When paired with its companion chip, the 82750DB, and used to implement a DVI Technology video subsystem, the 82750PB provides real time (30 images/sec) pixel processing, real time video compression, interactive motion video playback and real time video effects.

Real time pixel manipulations, including 30 images/sec video compression, are supported by the 25 MHz instruction rate. On-chip instruction RAM provides programmability for execution of a wide range of algorithms that support motion video decompression, text, and 2D and 3D graphics. Inner loops are optimized with the integration of sixteen 16-bit quad ported registers, on-chip DRAM, and two loop counters that provide zero delay two-way branching "free" in any instruction. Two, 16-bit internal buses enable two parallel register transfers on each 82750PB instruction, contributing to the real time performance of the video processing. Another feature that adds to the processing power of the 82750PB is the 16-bit ALU, which includes an 8-bit dual-add-with-saturate operation critical for pixel arithmetic. Other specialized features for pixel processing include a 2D pixel interpolator for image processing functions and a variable length sequence decoder for decoding compressed data.

The 82750PB is implemented using Intel's low-power CHMOS IV Technology and is packaged in a 132-lead space-saving, plastic quad flat pack (PQFP) package.



240854-1

82750PB Subsystem Diagram

February 1991
Order Number: 240854-003

82750PB Pixel Processor

CONTENTS

CONTENTS

82750PB Pixel Processor

CONTENTS

CONTENTS

82750PB Pixel Processor

CONTENTS

CONTENTS

1.0 82750PB PIN DESCRIPTION

Pinout

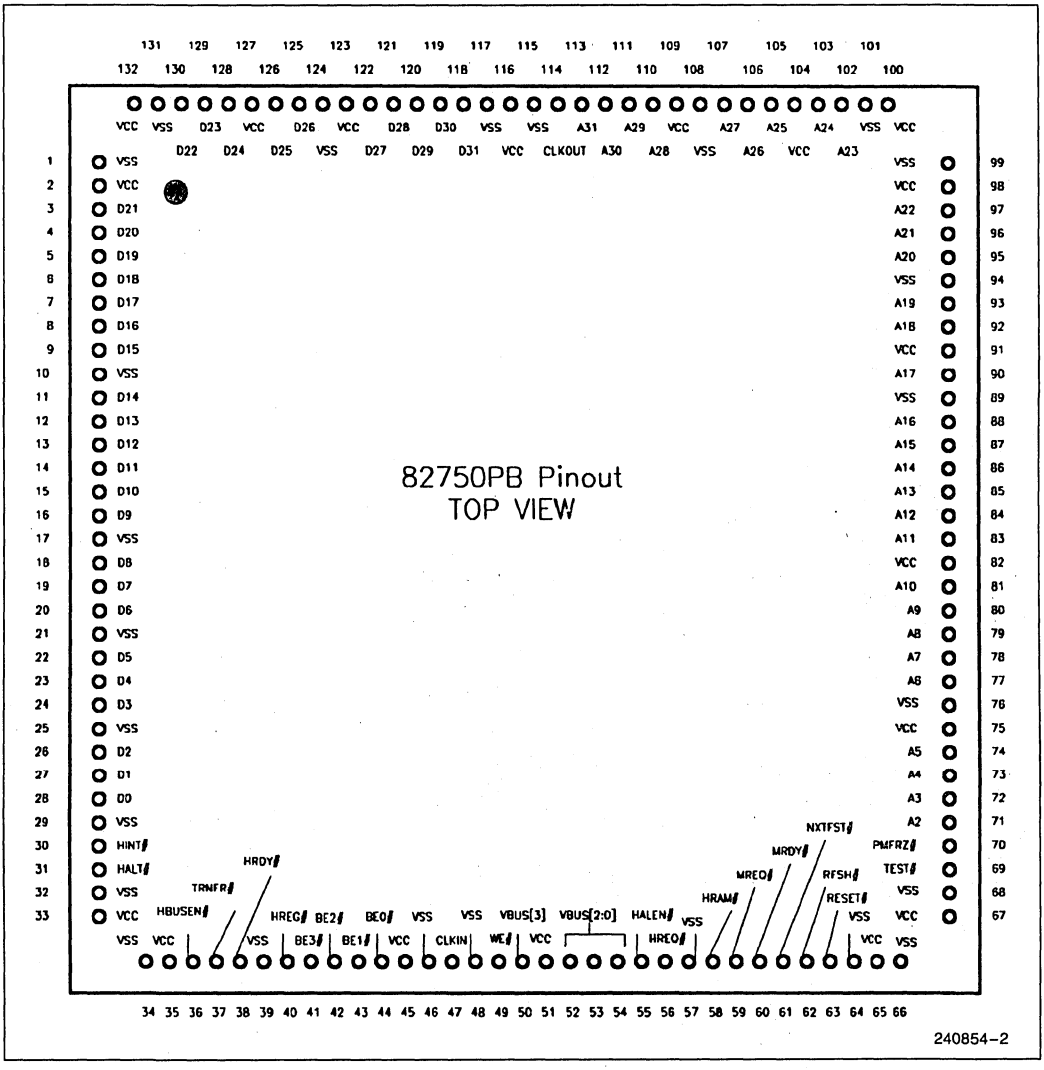


Figure 1-1. 82750PB Pinout

Table 1-1. Pin Cross Reference by Pin Name

Pin Name	Location
A2	71
A3	72
A4	73
A5	74
A6	77
A7	78
A8	79
A9	80
A10	81
A11	83
A12	84
A13	85
A14	86
A15	87
A16	88
A17	90
A18	92
A19	93
A20	95
A21	96
A22	97
A23	102
A24	103
A25	105
A26	106
A27	107
A28	110
A29	111
A30	112
A31	113
BE0#	44
BE1#	43
BE2#	42

Pin Name	Location
BE3#	41
CLKIN	47
CLKOUT	114
D0	28
D1	27
D2	26
D3	24
D4	23
D5	22
D6	20
D7	19
D8	18
D9	16
D10	15
D11	14
D12	13
D13	12
D14	11
D15	9
D16	8
D17	7
D18	6
D19	5
D20	4
D21	3
D22	130
D23	129
D24	128
D25	126
D26	125
D27	122
D28	121
D29	120

Pin Name	Location
D30	119
D31	118
HALEN#	55
HALT#	31
HBUSEN#	36
HINT#	30
HRAM#	58
HRDY#	38
HREG#	40
HREQ#	56
MRDY#	60
MREQ#	59
NXTFST#	61
PMFRZ#	70
RESET#	63
RFSH#	62
TEST#	69
TRNFR#	37
VBUS[0]	54
VBUS[1]	53
VBUS[2]	52
VBUS[3]	50
V_{CC}	2
V_{CC}	33
V_{CC}	35
V_{CC}	45
V_{CC}	51
V_{CC}	65
V_{CC}	67
V_{CC}	75
V_{CC}	82
V_{CC}	91
V_{CC}	98

Pin Name	Location
V_{CC}	100
V_{CC}	104
V_{SS}	94
V_{CC}	109
V_{CC}	116
V_{CC}	123
V_{CC}	127
V_{CC}	132
V_{SS}	1
V_{SS}	32
V_{SS}	34
V_{SS}	39
V_{SS}	48
V_{SS}	57
V_{SS}	66
V_{SS}	68
V_{SS}	76
V_{SS}	89
V_{SS}	99
V_{SS}	101
V_{SS}	108
V_{SS}	115
V_{SS}	117
V_{SS}	124
V_{SS}	131
V_{SS}	10
V_{SS}	17
V_{SS}	21
V_{SS}	25
V_{SS}	29
V_{SS}	46
V_{SS}	64
WE#	49

Table 1-2. Pin Cross Reference by Location

Location	Pin Name	Location	Pin Name	Location	Pin Name	Location	Pin Name
1	V_{SS}	34	V_{SS}	67	V_{CC}	100	V_{CC}
2	V_{CC}	35	V_{CC}	68	V_{SS}	101	V_{SS}
3	D21	36	HBUSEN#	69	TEST#	102	A23
4	D20	37	TRNFR#	70	PMFRZ#	103	A24
5	D19	38	HRDY#	71	A2	104	V_{CC}
6	D18	39	V_{SS}	72	A3	105	A25
7	D17	40	HREG#	73	A4	106	A26
8	D16	41	BE3#	74	A5	107	A27
9	D15	42	BE2#	75	V_{CC}	108	V_{SS}
10	V_{SS}	43	BE1#	76	V_{SS}	109	V_{CC}
11	D14	44	BE0#	77	A6	110	A28
12	D13	45	V_{CC}	78	A7	111	A29
13	D12	46	V_{SS}	79	A8	112	A30
14	D11	47	CLKIN	80	A9	113	A31
15	D10	48	V_{SS}	81	A10	114	CLKOUT
16	D9	49	WE#	82	V_{CC}	115	V_{SS}
17	V_{SS}	50	VBUS[3]	83	A11	116	V_{CC}
18	D8	51	V_{CC}	84	A12	117	V_{SS}
19	D7	52	VBUS[2]	85	A13	118	D31
20	D6	53	VBUS[1]	86	A14	119	D30
21	V_{SS}	54	VBUS[0]	87	A15	120	D29
22	D5	55	HALEN#	88	A16	121	D28
23	D4	56	HREQ#	89	V_{SS}	122	D27
24	D3	57	V_{SS}	90	A17	123	V_{CC}
25	V_{SS}	58	HRAM#	91	V_{CC}	124	V_{SS}
26	D2	59	MREQ#	92	A18	125	D26
27	D1	60	MRDY#	93	A19	126	D25
28	D0	61	NXTFST#	94	V_{SS}	127	V_{CC}
29	V_{SS}	62	RFSH#	95	A20	128	D24
30	HINT#	63	RESET#	96	A21	129	D23
31	HALT#	64	V_{SS}	97	A22	130	D22
32	V_{SS}	65	V_{CC}	98	V_{CC}	131	V_{SS}
33	V_{CC}	66	V_{SS}	99	V_{SS}	132	V_{CC}

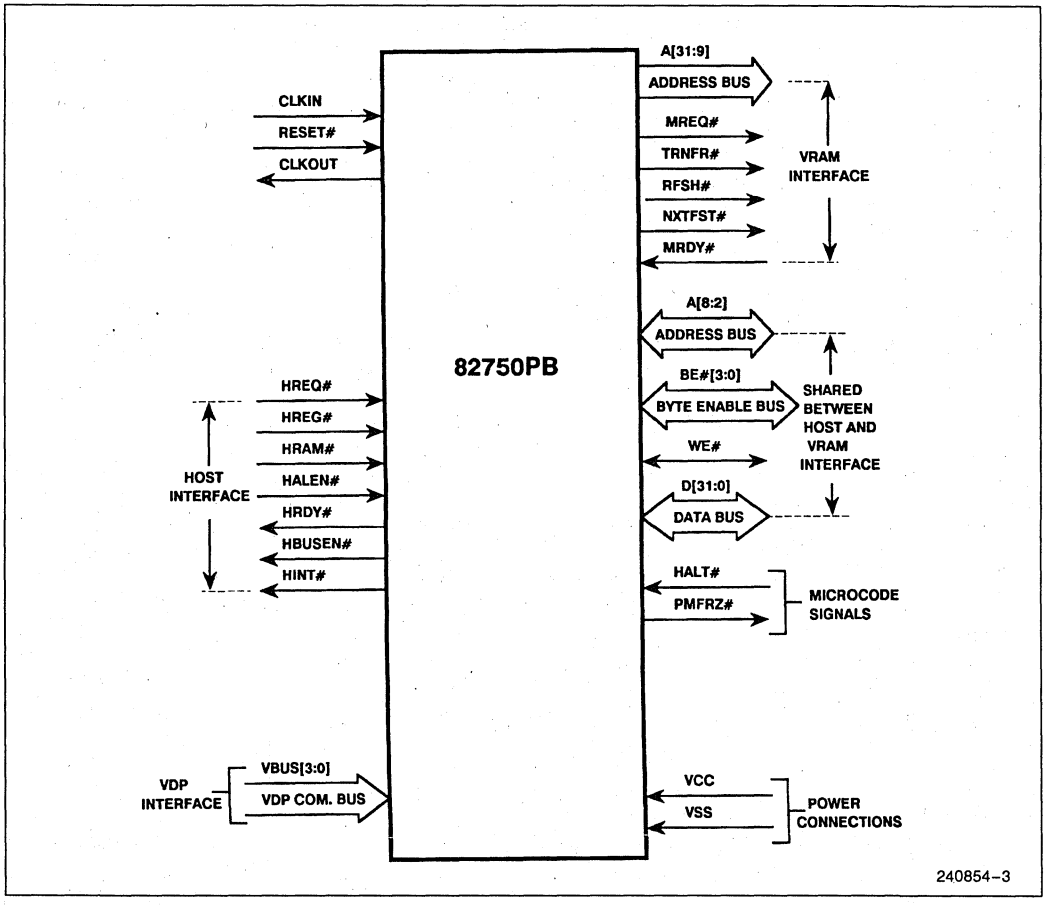


Figure 1-2. 82750PB Functional Signal Groupings

Quick Pin Reference

Table 1-3 provides descriptions of 82750PB pins.

Table 1-3. Pin Descriptions

Symbol	Type	Name and Function
CLKIN	I	CLKIN is a **1X CLOCK INPUT** that provides the fundamental timing for the 82750PB. One cycle of CLKIN is denoted as one T-cycle.
RESET#	I	The 82750PB is reset and initialized by holding this signal active for at least ten T-cycles. Refer to Initializing the 82750PB Section in Chapter 3.
HREQ#	I	The **HOST REQUEST** signal is a request from the host CPU to perform a read or write access to either registers on the 82750PB, an external device, or to VRAM shared by the 82750PB and the host. The type of access that is requested is determined by the host access definition signals: HREG#, HRAM#, and WE#.
HREG,# HRAM#	I	The **HOST REGISTER** and **HOST RAM** signals, when validated by HREQ#, are used to define three host access cycles. HRAM# active indicates the host is requesting a VRAM read or write cycle. HREG# active indicates that the host is requesting a 82750PB register read or register write cycle. When both signals are inactive, a host external cycle is requested.
HBUSEN#	O	**HOST BUS ENABLE** is asserted by the 82750PB at the start of a host access to indicate that the 82750PB Address and Data buses (A[31:2], BE#[3:0], and D[31:0]) have been tri-stated. This allows the host to drive the same buses either for accessing shared VRAM or the 82750PB internal registers.
HALEN#	I	The **HOST ADDRESS LATCH ENABLE** signal is used to indicate to the 82750PB that the host has asserted a valid address (A[31:2], BE#[3:0]) and write enable (WE#).
HRDY#	O	**HOST READY** is asserted by the 82750PB at the end of a host access to indicate that the access cycle is ready for data transfer. For a host write cycle, HRDY# indicates that the 82750PB is ready to accept data from the host. For a host VRAM write cycle, HRDY# indicates that the VRAM has latched the data from the host. For a host read cycle, HRDY# indicates that output data from the 82750PB or VRAM is ready to be latched by the host.
HINT#	O	**HOST INTERRUPT:** This output is asserted when an interrupt condition is detected by the 82750PB, and the enable bit in the PROCESSOR CONTROL register corresponding to that interrupt condition is set to a ONE. HINT# stays active until the host CPU reads the INTERRUPT STATUS register. If an interrupt condition that is enabled occurs during the same cycle that the INTERRUPT STATUS register is being read, HINT# remains active.
D[31:0]	I/O	The **DATA BUS** is used to transfer data between: 1. The 82750PB and VRAM, and 2. The Host CPU and internal 82750PB registers. During host VRAM accesses, this bus is tri-stated to allow the host to share the same VRAM data bus. During host accesses to internal 82750PB registers all 32 bits are used for data transfer.
A[31:9] A[8:2]	O I/O	The **ADDRESS BUS** is shared between the 82750PB and the host for addressing VRAM. This 30-pin bus addresses 32-bit double words in VRAM. Byte Enable signals are used to address individual bytes or words within a double word in VRAM. In addition, the address for host accesses to internal 82750PB registers are communicated to the 82750PB using the lower seven pins, A[8:2], and the BE# pins. During host access cycles to either VRAM or 82750PB internal registers, A[31:2] are tri-stated. For internal register accesses, as indicated by HREG# being low, the lower seven bits, A[8:2], are used as the host address input.
CLKOUT	O	The **CLOCK OUTPUT** signal is one of the two internal clocks and is synchronized with CLKIN. It is always driven and will have a 50% duty cycle.

Table 1-3. Pin Descriptions (Continued)

Symbol	Type	Name and Function
BE#[3:0]	I/O	The **BYTE ENABLE BUS** is shared by the 82750PB and the host for addressing VRAM down to the byte level. The correspondence between the four Byte Enable pins and the D[31:0] pins is: BE#[3]–D[31:24], BE#[2]–D[23:16], BE#[1]–D[15:8], and BE#[0]–D[7:0]. During VRAM read cycles, the 82750PB enables all four bytes. During write cycles the 82750PB only enables those bytes that are to be written. Bytes that are not enabled are not to be altered in VRAM. During host accesses to 82750PB on-chip registers, the BE#[0] pin is used as an input to select whether the even or odd word is being accessed; the double word address is provided by the host on the A[8:2] pins. BE#[0] = 0 indicates that data is transferred on D[15:0]. BE#[0] = 1 indicates that data is transferred on D[31:16].
MREQ#	O	**MEMORY REQUEST** is asserted for the first cycle, T1, of each VRAM cycle.
TRNFR#, RFSH#	O	The **MEMORY CYCLE DEFINITION SIGNALS:** Transfer, Refresh and Write Enable are asserted at the same time as MREQ#, but stay active for the entire VRAM cycle. TRNFR# active indicates a VRAM transfer cycle. RFSH# active indicates a VRAM refresh cycle. If neither TRNFR# nor RFSH# are active, a VRAM data read or write cycle is requested.
WE#	I/O	The **WRITE ENABLE** pin is used as an output during a 82750PB/VRAM cycle to drive the WE# signal, which defines the access as a VRAM read cycle (when inactive) or write cycle (when active). During Host/VRAM and Host External cycles, the 82750PB tri-states this pin to allow the host to drive the VRAM write enable signals directly. During Host/register cycles, this pin is used as an input for the Host Write Enable signal to determine whether the host is reading or writing the 82750PB register.
NXTFST#	O	The **NEXT FAST** signal indicates that the following vram cycle can be performed with a page-mode or bank-interleaved access. This signal is asserted during the first of a pair of VRAM cycles that is guaranteed to be within the same VRAM page and in opposite banks—a pair of accesses to two sequential double words in VRAM at addresses Even Address and Even Address + 1. In other words, A[2] is a zero for the first cycle and a one for the second cycle.
MRDY#	I	The **MEMORY READY** input indicates that the VRAM cycle has progressed to the point where it is ready to perform the data transfer. For a VRAM read cycle, the VRAM data can be latched by the transition of MRDY# to an active state. For a VRAM write cycle, MRDY# indicates that the data has been latched into the VRAMs.
VBUS[3:0]	I	The **VDP COMMUNICATION BUS** is used to communicate from the 82750DB to the 82750PB. Codes sent over this bus indicate interrupt requests, transfer requests, and status information. Since the 82750DB and 82750PB run asynchronously, the VBUS signals are sampled on the falling edge of CLKIN and compared with the previous sample. For a VBUS code to be detected by the 82750PB, it must be valid for two successive samples.
HALT#	I	The **HALT** signal causes the microcode processor on the 82750PB to halt prior to executing the next instruction. This signal does not halt the VRAM interface. The Halt signal will allow the design of a hardware emulator for the 82750PB based on an 82750PB chip.
TEST#	I	The **TEST** signal is used for test purposes only and must remain high for normal operation.

Table 1-3. Pin Descriptions (Continued)

Symbol	Type	Name and Function
PMFRZ#	O	The **PERFORMANCE MONITORING AND FREEZE** signal is toggled by specific microcode instructions and can be used to determine the time required to execute certain sections of microcode.
V_{CC}	I	**POWER** pins provide the +5V D.C. supply input.
V_{SS}	I	**GROUND** pins provide the 0V connection to which all inputs and outputs are referenced.

Table 1-4. Output Pins

Name	Active Level	When Floated
CLKOUT	High	Always Driven
A[31:9]	High	Reset*, Host Cycle
HBUSEN#	Low	Reset*
HRDY#	Low	Reset*
HINT#	Low	Reset*
MREQ#	Low	Reset*
TRNFR#, RFSH#	Low	Reset*
NXTFST#	Low	Reset*
PMFRZ#	Low	Reset*

*The reset state is caused by RESET# being active low.

Table 1-5. Input Pins

Name	Active Level	Synchronous/ Asynchronous
CLKIN	High	Synchronous
RESET#	Low	Asynchronous
HREQ#	Low	Asynchronous*
HREG#	Low	Synchronous
HRAM#	Low	Synchronous
MRDY#	Low	Synchronous
VBUS[3:0]	High	Asynchronous
HALT#	Low	Synchronous
HALEN#	Low	Asynchronous*

*Can be programmed to accept synchronous inputs.

Table 1-6. Input/Output Pins

Name	Active Level	When Floated	Synch/Async
D[31:0]	High	Reset*, Host Cycle	Synchronous
A[8:2]	High	Reset*, Host Cycle	Synchronous
BE#[3:0]	Low	Reset*, Host Cycle	Synchronous
WE#	Low	Reset*, Host Cycle	Synchronous

*The reset state is caused by RESET# being active low.

All output pins are floated when RESET is active low.

2.0 ARCHITECTURE

Overview

The 82750PB includes a wide instruction word processor that comprises a number of processing, storage, and input/output elements. The wide instruction word architecture allows a number of these elements to operate in parallel. The 82750PB executes one instruction every internal clock cycle or T-cycle. The various elements are connected via two 16-bit buses, the A bus and B bus, as shown in Figure 2-1. During each instruction execution cycle, data can be transferred from a bus source to a bus destination element on both buses.

Registers

{*rN; N = 0–15*}

There are 16 general-purpose data registers, each 16 bits wide, that are connected to both the A bus and B bus as both sources and destinations. These registers are designated *r0–r15*. All the registers are functionally identical except *r0*, which also includes logic for bit shifting and byte swapping. A register can source both the A bus and the B bus in the same cycle. A register cannot be the destination of both the A bus and the B bus in a single instruction. Because the registers are doubly latched, the same register may be both a source and destination in the same cycle. The result is that the data in the register prior to the current cycle will be driven on the source bus, and the data on the destination bus will be latched into the register at the end of the cycle.

Register *r0* has additional logic to allow bit shifting and byte swapping. The value in *r0* can be shifted left or right one bit position per instruction cycle. For a right shift, the new MSB is equal to the old MSB; in other words, the value is sign-extended. For left shifting, the new LSB is equal to zero. *R0* can not be shifted and loaded in the same instruction. Byte swapping, on the other hand, only occurs when *r0* is being loaded with a value from the A bus or B bus. Byte swapping causes the most significant byte and the least significant byte of the 16-bit value being loaded into *r0* to be interchanged. Refer to Chapter 4 for a description of the SHFT microcode field that controls the shifting and swapping operations in *r0*.

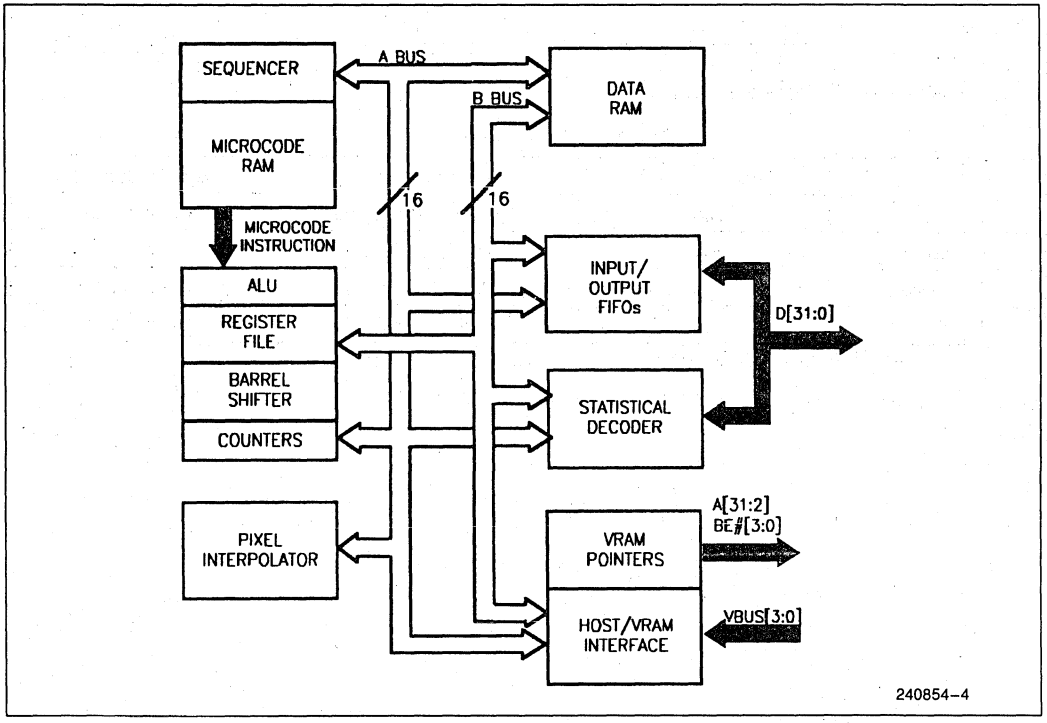


Figure 2-1. 82750PB Block Diagram

ALU

{*alu, cc*}

The ALU performs 16-bit arithmetic and logic operations, and can also be operated as two independent 8-bit ALUs for the Dual-Add-with-Saturate operation. There are two fields in the microcode instruction that affect the operation of the ALU: the ALUOP field specifies the operation to be performed, and the ALUSS field specifies the source of the two ALU inputs. Refer to Chapter 4 for further information on these fields.

The two ALU operands either come from values held in the ALU input latches or from "eavesdropping" on the A or B buses. The result of any ALU operation is latched in the ALU output register, *alu*. In a subsequent instruction this result can be transferred to any A or B destination.

The ALU has four condition flag outputs: CarryOut, Sign, Overflow, and Zero. CarryOut is the carry out of the most significant bit position. Sign is equal to the value of the most significant bit of the result. Overflow is the exclusive-OR of CarryOut and the CarryIn to the most significant bit position of the result. Zero is true (a value of 1) if all 16 bits of the ALU result are equal to zero. CarryOut and Overflow are defined as equal to zero for all logical operations. For most ALU operations, the state of these four condition flags are latched when the operation is complete. There are eight operations (nop, a*, b*, +], -], 0*, prof and int) that are exceptions. These operations are performed without disturbing the condition state of the previous ALU operation.

Microcode routines can read and write the ALU condition flag register, *cc*. This can be used to save and restore the state of these flags. The bit ordering of the ALU condition flags within *cc* are given in Table 2-1.A complete list of ALU opcodes is given in Table 2-2.

Table 2-1. Bit Assignments for *cc* Register

Bit	Condition
Bit 0	False (This bit of the *cc* is always read as a zero.)*
Bit 1	ALU Carry Out
Bit 2	ALU Overflow
Bit 3	ALU Sign
Bit 4	ALU Zero
Bit 5	Loop Counter Zero*
Bit 6	R0 LSB*
Bit 7	R0 MSB*
Bit 15:8	RESERVED. The state of these bits is undefined when read; write as zeros.

*These are read-only values and are not affected by writes to the cc register.

Table 2-2. ALU Opcodes

Operation	Mnemonic
No Operation	nop
pass a	a
pass b	b
1's compliment of a	~a
1's compliment of b	~b
a AND b	&
(NOT a) AND b	~&
a AND (NOT b)	&~
a OR b	|
a XOR b	^
a + b	+
a + b + 1	++
a − b	−
−a + b	−+
2's compliment of a	−a
2's compliment of b	−b
Increment a	a++
Increment b	b++
Decrement a	a−−
Decrement b	b−−
Dual Add with Sat.	+]
a + b + (Prev. Carry)	+<
a − b − (Prev. Borrow)	−<
−a + b − (Prev. Borrow)	−+<
Interrupt Host	int
Zero	0*
Pass a, Don't Latch Flags	a*
Pass b, Don't Latch Flags	b*
(NOT a) OR b	~|
a OR (NOT b)	|~
Dual Sub. with Sat.	−]
Perform. Monitor/Profile	prof

The Dual-Add-with-Saturate operation performs independent 8-bit ADDs on the upper and lower bytes of the two ALU operands. The two bytes of the A operand are treated as unsigned binary numbers ($00{:}FF_{16}$ corresponds to $0{:}255_{10}$). The two bytes of the B operand are treated as offset binary numbers

with an offset of +128 (00:FF_{16} corresponds to -128_{10}:127_{10}). The upper and lower byte results are treated as 9-bit offset binary, including the carry output of each byte, with a +128 offset (000:$1FF_{16}$ corresponds to -128_{10}:383_{10}) and are saturated to a range of 0–255_{10}. A result that is less than zero is set equal to zero or 00_{16} and a result that is greater than +255 is set equal to +255 or FF_{16}.

In fact, this operation is symmetric. Either the A operand or the B operand can be defined as the unsigned binary value, and the other operand will be treated as the offset signed binary value.

Dual-subtract-with-saturate is similar to dual-add-with-saturate. It calculates A − B + 128 on each 8-bit half of the two 16-bit inputs, and clamps the results to 0 and 255. This can be viewed as subtracting an offset-binary signed byte (−128 to 127) from an unsigned byte (0 to 255).

The ALU opcode 'int' generates the MCINT (microcode interrupt) condition. When this condition is detected by interrupt logic in the host CPU interface, and if the Enable MCINT bit in the PROCESSOR CONTROL register is set to a ONE, the host interrupt output, HINT#, will be asserted. Refer to Chapter 3 for further information on host interface.

The 'prof' opcode activates the PMFRZ# pin, and is primarily used for performance monitoring and/or debugging.

Barrel Shifter

{*shift, shift-r, shift-rl, shift-l*}

The barrel shifter performs a single cycle, n-bit left or right shift. The barrel shifter operates independent of the ALU. The three barrel shifter operations are: *Shift-r* for a right shift with sign extend; *Shift-rl* for right shift with zero fill; and *Shift-l* for a left shift with zero fill. The shift operation is invoked by writing a 4-bit value (the shift amount) to one of three A bus registers, depending on which of the three operations is to be performed. The operand is taken from the B bus, and the result is stored in the barrel shifter output register, *Shift*. Like the ALU result register, the value in *Shift* can be read onto the A bus or B bus in the following instruction cycle.

A barrel shifter operation does not affect any of the condition flags.

Data RAM

{*dramN, *dramN, ++, −−; N = 1–4*}

The Data RAM holds 512, 16-bit words that are accessed using four pointers. To access a value in a particular location, the microcode routine must first load a pointer with the address to be accessed, and then perform a read or write using the same pointer. In parallel with the data RAM access, the pointer can optionally be post-incremented or post-decremented. The four pointers, referred to as *dram1–dram4*, can be written and read via the A bus. When a dram pointer, which is only 9 bits wide, is read onto the A bus, its upper seven bits are set to zeros.

NOTE:

The width of the dram pointers may change in later versions of the 82750PB. Software should not rely on the width of a pointer to, for example, mask the upper seven bits of a value to zero.

All four pointers can be used to read or write the Data RAM from either the A or B bus. Only one Data RAM access can be performed in any cycle. A Data RAM access is referred to, using C language syntax, as **dram1*. The * means "the value pointed to by". As another example, **dram3*++ means access the Data RAM using the pointer *dram3* and increment *dram3*. The symbol −− in place of the ++ would indicate autodecrement.

Loop Counters

{*cnt,cnt2*}

Two 16-bit loop counters are available to microcode programs for automatically counting iterations of a microcode loop. In parallel with other operations performed in an instruction, either loop counter can be decremented, and a conditional branch can be made based on the loop counter value being equal or not equal to zero. Since the two loop counters can be written and read on the A bus, as *cnt* and *cnt2* respectively, they can also be used for variable storage when not being used as loop counters. The loop counters can be written to and decremented during the same instruction cycle. The value in the counter at the start of the next cycle will be the value written to the counter minus one.

The LC microcode bit determines the loop counter that is selected for decrementing and/or branching in an instruction. The LC microcode bit does not affect the loop counter that is written or read over the A bus, since each loop counter is separately addressable as a A bus source or destination. Refer to

Chapter 4 for a description of the CNT −− microcode bit that causes the select loop counter to be decremented, and for a description of the CFSEL microcode field that is used to perform a conditional branch based on the selected loop counter's value.

Microcode RAM

{*mcode1-3, maddr, pc*}

The 82750PB executes instructions stored in an on-chip microcode RAM. This RAM holds 512 instructions and each instruction is 48 bits wide. Normally, to start the microcode processor, the host CPU will load a microcode program into the microcode RAM, point the program counter, *pc*, to the start of the program, and then release the HALT bit to start executing the microcode program. The microcode processor can also load its own microcode RAM to overlay new routines and therefore, does not require constant intervention by the host to perform multiple operations.

Writing an instruction into Microcode RAM is done by first loading the three registers *mcode3, mcode2*, and *mcode1* with the three 16-bit words of the instruction (the most significant word goes into *mcode1*), and then loading the address where the instruction should be written into *maddr*.

The host CPU can also read the Microcode RAM by first loading the *pc* with the address of the instruction to be read and then reading the three 16-bit words of the instruction from the *mcode1–mcode3* registers. Normally, this would be done by the Host CPU while the 82750PB is halted. Since *mcode1–mcode3* hold the instruction pointed to by the *pc* (i.e. the instruction that is about to be executed), normally reading these three registers from a microcode routine is not useful.

The read registers named mcode1–mcode3 and the write registers also named mcode1–mcode3 are in fact different registers. Writing values into mcode1-mcode3 and then reading the values of mcode1-mcode3 will not read back the same values just written. The read registers hold the instruction stored ir the instruction latch (the instruction to be executed). The write registers hold an instruction that is about to be written into microcode RAM.

After writing to *maddr* to load an instruction into microcode RAM, a one cycle freeze occurs and during the freeze a write to the microcode RAM takes place. The instruction following the write to *maddr* can either jump to the address just loaded or start loading the *mcode1–mcode3* registers with the next instruction to be written.

Here are two examples that illustrate the fact that the 82750PB requires at least one instruction between the write to maddr and the execution of the instruction that is loaded by the write to maddr.

Example 1:

```
maddr = ADDR1          /* load instruction */
jmp addr1              /* jump to it, this is the extra inst. required between */
                       /* writing to maddr and executing the loaded inst. */
. . .
ADDR1:
???????????            /* here's where new instruction gets loaded */
```

Example 2:

```
maddr = INST
nop                    /* extra instruction */
INST:
???????????            /* instruction gets loaded here */
```

When a microcode routine writes to *pc*, one more instruction is executed before the jump to the new address takes effect. For example:

```
pc = ADDR1
r0 = r1   jmp ADDR2    /* this instruction gets executed but */
                       /* its jump to ADDR2 is ignored. */
. . .
ADDR1:
r3 = r0                /* after this instruction executes r3 = r0 = r1 */
```

When the host CPU writes to the *pc*, the instruction at the address that was written is loaded into the *mcode1–mcode3* registers and, when the microcode processor is released from its Halt condition, this is the first instruction that will be executed.

When the host CPU reads the *pc*, the result returned is the address of the instruction that will be executed when Halt is released, that is, the address of the instruction held in the *mcode1–mcode3* registers.

Horizontal Line Counter

{*lcnt*}

The 12-bit Horizontal Line Counter is updated by VBUS codes from the 82750DB to track the horizontal display line that is currently being scanned by the 82750DB. The counter is reset by a VODD code and incremented each time an HLINE code is received. A value can also be written into a Horizontal Line Counter but this is used primarily for testing the 82750PB. The upper four bits will always read zeros.

Field Counter

{*fcnt*}

The 4-bit field counter is updated by VBUS codes from the display processor to keep track of the field count being displayed by 82750DB. The counter is incremented each time a V_{ODD} code or V_{EVEN} code is received. When reading the field counter, the upper 12 bits will read zeros. This counter will not be initialized upon reset.

Input FIFOs

{*inN-lo, inN-hi, inN-c, *inN; N = 1, 2*}

There are two input channels, referred to as input FIFOs, through which the processor can read pixels or data from VRAM. Each channel automatically fetches 64-bit quad words from VRAM and breaks them into 8-bit bytes or 16-bit words that are read by microcode. Each input FIFO operates independently and can be programmed to automatically increment or decrement through bytes or words in VRAM. The FIFOs are double buffered so that while values are being extracted from one quad word (64 bits), the next quad word is being prefetched from VRAM.

The mode control register for each input FIFO, designated *in1-c* or *in2-c*, contains four mode bits as seen in Figure 2-2. The WORD/BYTE bit (bit 0) determines whether the input FIFO is in word mode (WORD/BYTE = 0) or byte mode (WORD/BYTE = 1). In byte mode, the FIFO can start reading on any byte boundary and in word mode on any word boundary.

The INC/DEC bit (bit 1) determines the order that bytes or words are read from VRAM. In INCREMENT mode, with INC/DEC = 0, the FIFO reads from the least significant byte or word to the most significant byte or word of each double word and increments through double words in VRAM. In DECREMENT mode, with INC/DEC = 1, the FIFO reads from most significant byte or word to least significant byte or word within a double word and decrements through double words in VRAM.

The AHOLD bit (Bit 2) is used by the address hold mode. When asserted, (bit 2 = 1) the automatic address increment/decrement function will be disabled and input FIFOs will not double buffer VRAM data. In other words, at the end of a VRAM cycle, when the FIFO has been updated with 64 bits of VRAM data, the input FIFO will not issue another MREQ# until there is a write to the address-lo registers OR a roll-over/roll-under read access of the input FIFO. If a roll-over/roll-under occurs, then a memory request will be issued to fetch data from the same VRAM location. If there is a write to the address-lo register, the FIFO will then fetch data from the new location.

The PREFETCH OFF bit (bit 3) specifies whether the FIFO will automatically prefetch successive quad words from VRAM or will only fetch a new quad word when a value from that quad word is requested. In PREFETCH-ON mode, bit 3 = 0, the input FIFO prefetches successive quad words from VRAM as necessary to keep its buffer full (either from ascending or descending addresses, depending on the state of the INC/DEC bit). In PREFETCH-OFF mode, the FIFO will still prefetch the first two quad words to fill its buffer (when started at a new address location), but will only fetch a new quad word when a read request is made to the FIFO for a value in the next unfetched quad word.

The CB bit (bit 4) allows circular buffers of sizes 64 Kbytes, 128 Kbytes, or 256 Kbytes to be created in VRAM memory. The choice of different sizes of buffers are determined by programming the least signficant 3 bits of the circular buffer register (cir-

bits:	15 . . . 6	5	4	3	2	1	0
	Set to Zeros	BY-32 MODE	CB	PF OFF	AHOLD	INC/DEC	WORD/BYTE

Figure 2-2. Input FIFO Control Register

cbuf). To enable this feature, the CB bit has to be set to a 1, then depending on the buffer size selected, the appropriate address pin that goes off chip will be forced to a 0 (register pointers remain unchanged). Table 2-3 shows the programming combinations of the circular buffer register.

It is important to note that the internal address counters themselves are not affected by the circbuf function. Only the selected external address pin is forced to '0'.

Table 2-3. Circular Buffer Register (circbuf)

Bits [2:0]	Buffer Size	Effect on PB Address Bus (If Function Enabled)
000	Disabled	None
100	256 Kbytes	Address Pin 18 Forced to 0
010	128 Kbytes	Address Pin 17 Forced to 0
001	64 Kbytes	Address Pin 16 Forced to 0

In "BY-32" MODE (bit 3), the pointer increments or decrements by 32 bits, independent of whether the FIFO is in 8-bit pixel mode or 16-bit pixel mode. This mode was added to facilitate microcode that operates on one component of a 32-bit per pixel image.

The standard sequence for initializing an input FIFO is to write to the control register *(in-c)*, the high address *(in-hi)*, and then the low address *(in-lo)* of the appropriate FIFO. Refer to the access state diagram in Chapter 3. The write to *in-lo* causes the FIFO to start reading from VRAM. A byte or word is then read from **in*. Successive reads from **in* will read sequential bytes or words from VRAM. Writing to the control register each time the FIFO is started at a new address is not necessary, except to change the FIFO's mode. Also, if the new address is within the same 64 kByte page of VRAM, only the lo-address needs to be written in order to start the FIFO reading from the new address.

If microcode attempts to read a value from an empty input FIFO, the processor is frozen prior to the execution of the instruction, until the FIFO's control logic has fetched another double word from VRAM and extracted the next value. At this point, the processor is released from the frozen state, and the instruction that reads the value is executed. When the processor is frozen waiting for a particular FIFO that isn't yet ready, that FIFO's VRAM access priority is raised above all other FIFOs.

Output FIFOs

{*outN-lo, outN-hi, outN-c, *outN, outN+ +; N = 1, 2*}

There are two output channels, referred to as output FIFOs, through which the graphics processor writes pixels or data to VRAM. Each channel automatically collects bytes or words into 64-bit quad words and writes the quad words to VRAM. Each output FIFO operates independently and can be programmed to write bytes or words into sequential addresses in VRAM (either incrementing or decrementing). The FIFOs are double buffered so that while one quad word is waiting to be written to VRAM, the next quad word can be assembled from individual bytes or words.

The mode control register for each output FIFO, designated *out1-c* or *out2-c*, contains six mode bits as shown in the Figure 2-3. The WORD/BYTE bit (bit 0) determines whether the output FIFO is in word mode (WORD/BYTE = 0) or byte mode (WORD/BYTE = 1). In byte mode the FIFO can start writing on any byte boundary in VRAM and in word mode on any word boundary.

The INC/DEC bit (bit 1) determines the order that bytes or words are written to VRAM. In INCREMENT mode, with INC/DEC = 0, the FIFO writes from the least significant byte or word to the most significant byte or word in a double word and increments through double words in VRAM. In DECREMENT mode, with INC/DEC = 1, the FIFO writes from most significant byte or word to least significant byte or word within a double word and decrements through double words in VRAM.

When the AHOLD bit (bit 2) is set, the output FIFO quad word address is not incremented or decremented. In this mode, the FIFO continues to output to a single quad word in VRAM.

The FORCE-LSB bits (bits 3 and 4) are used to force the least significant bit of each byte written to VRAM to either a zero or a one. This can be used, for example, to force the LSB to the correct polarity when writing to the U bitmap during motion video decompression. In certain display modes for the 82750DB, the LSB of the 8-bit samples in the U or Y bitmap are used to select VIDEO or GRAPHICS display mode for the n x n group of display pixels corresponding to the particular U or Y sample. A one in the FORCE-

bits:	15–6	5	4	3	2	1	0
	Set to Zeros	BY-32 MODE	FORCE-LSB ENABLE	FORCE-LSB VALUE	AHOLD	INC/DEC	WORD/BYTE

Figure 2-3. Output FIFO Control Register

LSB ENABLE bit (bit 4) enables the forcing; a zero results in normal operation. The FORCE-LSB VALUE bit (bit 3) is used as the value to which the LSB is forced. Whether in byte mode or word mode, the LSB of *each byte* is forced to the FORCE-LSB value.

In "BY-32" MODE (bit 5), the pointer increments or decrements by 32 bits, independent of whether the FIFO is in 8-bit pixel mode or 16-bit pixel mode. This mode is used to facilitate microcode that operates on one component of a 32-bit per pixel image. The bytes or words that are skipped over will be unchanged in VRAM.

The standard sequence for initializing an output FIFO is to write to the control register *(out-c)*, the low address *(out-lo)*, and then the high address *(out-hi)* of the appropriate FIFO. A series of bytes or words is then written to **out*. Refer to the access state diagram in Chapter 3 (Figure 3-1).

In order to flush any remaining data in an output FIFO before changing its VRAM pointer, it is necessary to write to the control register. When pointing to a new location in VRAM, if the new address is within the same 64 kByte page of VRAM, only the lo-address needs to be written.

There must be one instruction between the write to the output FIFOs low address and the first write to *outN. Therefore, it is recommended that outN-lo be written before outN-hi. The write to outN-hi insures that this requirement is met. If only the outN-lo value is being changed, it is still necessary to have one additional instruction before the first write to *outN.

When writing bytes or words to VRAM through an output FIFO, a byte or word can be skipped over by writing to *outN++* instead of **outN*. When the values are written to VRAM, any byte or word that was skipped will retain its original value in VRAM, and its value is not altered by the VRAM write. This can be used when writing a series of pixels, some of which are "transparent", allowing whatever was behind them to show through.

If the microcode routine attempts to write a value to a full output FIFO, the processor is frozen prior to the execution of the instruction. The processor remains frozen until the FIFO has a chance to write one of the buffered quad words to VRAM. At that point, the processor is released from the frozen state, and the instruction that writes the value is executed. When the processor is frozen, waiting for a particular FIFO that isn't yet ready, that FIFO's VRAM access priority is raised above all other FIFOs.

Statistical Decoder

{*stat-lo, stat-hi, stat-c, stat-ram, *stat, *stat#*}

The Statistical Decoder (also referred to as the Huffman Decoder) is a specialized input channel that can read a variable-length bit sequence from VRAM and convert it into a fixed-length bit sequence that is read by the microcode processor. In image compression, as well as in other applications such as text compression, certain values occur more frequently than others. A means of compressing this data is to use fewer bits to encode more frequently occurring values and more bits to encode less frequently occurring values. This type of encoding results in a variable-length sequence in which the length of a symbol (the group of bits used to encode a single value) can range for example, from one bit to sixteen bits.

The statistical code that the statistical decoder can decode is of either of the two forms:

0x		1x
10x		01x
110xxx		001xxx
1110xxxxx		0001xxxxx
...	or	...
11111110xxxxxx		00000001xxxxxx
111111110xxxxxx		000000001xxxxxx
...		...

Each symbol of a given length (one per line as shown here) consists of a run-in sequence followed by some number of x-bits. The run-in sequence is defined as a series of zero or more ONEs followed by a ZERO or, as in the code on the right above, zero or more ZEROs followed by a ONE. The remainder of this description will use examples of the code on the left. A bit in the decoder's control register determines the polarity of the run-in sequence bits.

In the example on the left, there would be two symbols of length two: 00 and 01. Each x-bit can take on a ZERO or ONE value. The number of x-bits following a run-in sequence can range from zero to six. Since the goal, in general, is to have a few short codes and a larger number of long codes, typically, codes with fewer run-in bits will have fewer x's following. However, this is not a hardware constraint. A code of this form is completely described by a code description table indicating: for each length of run-in sequence, R = the number of ONEs in the run-in, and how many x-bits follow the ZERO. The value of R is used as an index into the code description table. Due to the hardware implementation, the number actually stored in the table is 2^x, where x is the number of x-bits.

For the example above, the corresponding code description values are given in Table 2-4.

Table 2-4. Sample Code Description Table

R	X	2^X(dec.)	2^X(bin.)
0	1	2	000 0010
1	1	2	000 0010
2	3	8	000 1000
3	5	32	010 0000
...			
7	6	64	100 0000

Note that the table only goes up to symbols with seven ONEs in the run-in. For symbols with more than seven ONEs, the value of X and 2^X for seven ONEs is used for all symbols having seven or more ONEs in the run-in sequence. For example, in the code above a symbol with eight or more ONEs in the run-in sequence has six x-bits following the ZERO, which is the same as symbols having seven ONEs.

For each different symbol, including all symbols of the same run-in length with different x-bit values, the decoder generates a unique fixed-length, 16-bit value. Some of the decoded values for the sample code given above are provided in Table 2-5.

Table 2-5. Decoded Values

Symbol*	Decoded Value
00	0
01	1
100	2
101	3
110000	4
110001	5
110010	6
...	...
110111	11
111000000	12
...	...
111011111	43
...	...

*The x-bits of the symbol are in **boldface** for clarity.

The algorithm for generating a decoded value from a symbol is as follows: all symbols of a given run-in length are assigned a base value, B; the value corresponding to a particular symbol is equal to B plus the binary value of the x-bits in the symbol. The base valule B for a symbol with a run-in length of R is calculated by:

$$B(R) = SUM[2^{X(r)}] \text{ with } r = 0 \text{ to } R - 1,$$

where X(r) corresponds to the X value in the table entry corresponding to R = r.

For example, in the above code:
B(0) = 0, B (0) is always zero
B(1) = 0 + 2 = 2
B(2) = 0 + 2 + 2 = 4
B(3) = 0 + 2 + 2 + 8 = 12
B(4) = 0 + 2 + 2 + 8 + 32 = 44

This is one of the reasons that the table holds 2^X instead of X. The calculation of B(R) are easier to implement in logic.

There are two enhancements that are made to this coding scheme in the implementation on the 82750PB. These two modes are referred to as END mode and SHORT mode. If neither END nor SHORT mode are enabled, the decoding is performed as described above. SHORT mode allows the decoder to be switched easily to a simpler code format without having to reload the code description table. In the SHORT form, all symbols have the same number of x-bits, as though all entries in the table had been filled with the same value of 2^X. When SHORT mode is invoked, this value of 2^X is obtained from a field in the statistical decoder's CONTROL word, instead of from the individual table entries.

END mode is added in recognition of the fact that, for codes with few symbols, some increase in efficiency is possible by not having to place a zero at the end of the longest run-in sequence. For example, consider the code:

0
10x
110x

The END mode allows us to shorten the last symbol to 11x instead of 110x. The trailing ZERO is not required because the decoder has been told that the maximum length of a run-in is two ONEs. The resulting symbol set and corresponding decoded values are given in Table 2-6.

Table 2-6. END Mode Decoded Values

Symbol	Decoded Value
0	0
100	1
101	2
110	3
111	4

The number of x-bits must be constant for all symbols of the same run-in length. Therefore, a code such as:

0
10xx
11xxx ← NOT CORRECT! . . . Must be 11xx.

is not allowed. The last symbol (11xxx, in this case) uses the same table entry for 2^X as the next to last symbol (10xx) and, therefore, the last symbol will be 11xx.

The maximum length of the run-in sequence in END mode is specified by placing an END flag in the code description table. For example, a code and the corresponding table is shown in Table 2-7.

Table 2-7. END Flag Decoded Values

Code	Table Entries		
	Index	END Bit	2^X
0	0	0	0
10xx	1	0	4
110xxx	2	1	8
111xxx	3	-	-
	4	-	-
	5	-	-
	6	-	-
	7	-	-

The hyphens indicate that those table entries aren't used to decode this code. Note that the symbol 111xxx has three x-bits because of the value of 2^X in Index 2; it is not based on the 2^X value in Index 3.

The SHORTED and END modes can be invoked simultaneously, resulting in a code such as:

0x
10x
110x
111x

with a SHORT -2^X value = 2 (for 1 x-bit in each symbol) and the END bit set in Index 2.

Packed binary fields with one to seven bits per field can be read using the statistical decoder by setting the END bit in Index 0 and by programming the X value to be N − 1, where N is the number of bits per field. For example, packed three-bit fields could be decoded as shown in Table 2-8.

Table 2-8. Packed 3-Bit Field Decoded Values

Code	Table Entries		
	Index	END Bit	2^X
0xx	0	1	4{N = 3, so X = 2}
1xx	1	-	-
	2	-	-
	3	-	-
	4	-	-
	5	-	-
	6	-	-
	7	-	-

The unpacked bits are in reverse order relative to how they are stored in VRAM. For example, if three-bit values are packed in VRAM, the pattern 110 in VRAM is read from right to left and gives an unpacked or decoded value of 3.

The CONTROL register for the statistical decoder *(stat-c)* is used to specify the mode to use for decoding, as well as to invoke certain modes for writing and reading the code description table. Refer to the bit assignments for this register below. To write to the code description table, the WRITE bit (bit 4) is set to a ONE; the starting table index is reset to zero. Each write to the table causes the index to increment by one. This index will wrap around from seven back to zero. For example, to write all eight table entries the user would write a value of 0x10 to *stat-c* register and then write eight 8-bit values to the register *stat-ram*. The most significant bit of each 8-bit value is the END bit, and the lower seven bits are the values of 2^X. To read the code description table, the TEST bit (bit 5) of the CONTROL register is set to a one. The table entries are then read from the decoder's data register *(*stat)*. Reads and writes always start at table entry zero.

NOTE:

*When reading the code description table, it is necessary to wait one instruction time between the write to stat-c and the first read from *stat. An access diagram showing all legal sequences for read and write FIFO registers is shown in Chapter 3 (Figure 3-1).*

The code for reading the eight table entries into the first eight locations of data RAM would be:

```
dram3 = 0          stat-c = 0x20          /test mode to read the stat-ram (the table)
cnt = 8                                   /wait one inst. before first read
LOOP:
                   *dram3++ = *stat cnt--
                   jcp loop               /two inst. loop necessary to wait one inst.
                                          /between each read from *stat.
```

Bits	15	14	13	12:8	7	6	5	4	3	2:0
	POL	RSVD*	CB	SVAL	SHORT	END	TEST	WRITE	RSVD*	Starting *Stat-ram* ADDRESS

* Reserved: write zeros to these bits.

Figure 2-4. Statistical Decode CONTROL Register

END mode is enabled by setting the END bit (bit 6) in the CONTROL register to a ONE. The SHORT mode is enabled by setting the SHORT bit (bit 7) in the CONTROL register to a ONE. When in SHORT mode, the five SVAL bits (bits 12:8) in the CONTROL register are used as the SHORT -2^X value.

The POL bit (bit 15) determines the polarity of the run-in sequence bits. If bit 15 = 0, then ONEs ending in ZERO (e.g., 1110xxx) sequence is selected. If bit 15 = 1, the ZEROs ending in ONE (e.g., 0001xxx) sequence is selected.

The CB bit (bit 13) allows circular buffers of sizes 64 Kbytes, 128 Kbytes, or 256 Kbytes to be created in memory, as in the case of the input FIFO. The choice of different sizes of buffers are determined by programming the least significant 3 bits of the circular buffer register (circbuf). To enable this feature, the CB bit has to be set to a 1, then depending on the buffer size selected, the appropriate address pin that goes off chip will be forced to a 0 (register pointers remain unchanged). Table 2-3 shows the programming combination of the circular buffer register.

The decoding parameters may be changed between symbols by writing to the CONTROL register and, if necessary, writing new values into the code description table. The correct procedure for changing the code type or decode mode is to read the last value from the decoder prior to the change, using **stat#* instead of **stat*. This keeps the decoder from automatically starting to decode the next symbol. At this point, the code description table and the SHORT and END mode bits can be changed as desired. The next time the CONTROL register is written with both TEST = 0 and WRITE = 0, the decoder will begin to decode the next symbol using the new parameters.

The statistical decoder buffers one quad word read from VRAM so that the decoding of bits in one 32-bit word and the fetch of the next 32-bit word may overlap. As with the input and output FIFOs, the decoder has a VRAM pointer associated with it that points to the location in VRAM from which it is reading data. This pointer increments twice each time a new quad word is read; there is no decrement mode. When the least significant word of the decoder's pointer *(stat-lo)* is written, any data that had previously been prefetched from VRAM is ignored, and the decoder fetches one quad word starting from this new location.

The 82750PB assumes that the statistically encoded bitstream in VRAM starts with the least significant bit of a *double* word. That is, the two LSBs of the address written to start-lo are ignored.

The statistical decoder decodes data at a rate of one bit per T-cycle. To a first approximation, the decode time for an N-bit symbol is:

$$\text{decode time (in T-cycles)} = N + 1$$

Since it takes at least 64 T-cycles to decode data from one quad word, which is the time required fo eight quad word reads from VRAM, the decoder should rarely run out of data. Therefore, the above estimate should very accurately model the actual decoding rate of the statistical decoder.

The statistical decoder always begins to read the bitstream from the least significant bit of the double word found at the starting location in VRAM. That is, the decoder does not start on a byte or word boundary as an input FIFO or output FIFO does, but only on double word boundaries. The bitstream moves from the least significant bit to the most significant bit of a double word and then to the least significant bit of the next double word (at the next higher ad-

dress location). For the x-bits, the first x-bit read from the bitstream becomes the most significant bit of the x-bit field when it is interpreted as a binary number. The example below shows a code definition, a bitstream stored in VRAM, and the resulting decoded values.

The code definition and range of values for each symbol length are indicated in Table 2-9.

Table 2-9. VRAM Bitstream Decode Values

Symbol	Values	Comments
0	0	
10x	1, 2	100 = 1, 101 = 2
110xx	3–6	11000 = 3, . . . , 11011 = 6
1110xxx	7–14	1110000 = 7, . . . , 1110111 = 14

Decoding starts at address 0 in this example. The two double words at addresses 0 and 1 are:

0: 0xAC98E14D

1: 0x372E74CB

The bitstream in VRAM, with colons dividing the symbols (read from right to left starting at LSB of address 0) is shown in Figure 2-5.

Table 2-10 lists the symbols, in the order they are encountered in the bitstream, and the corresponding decoded values.

Table 2-10. Decoding Symbols

Symbol	Value	Comments
101	2	Starts at LSB, Address 0, Scanning Left
100	1	
101	2	
0	0	
0	0	
0	0	
0	0	
1110001	8	
100	1	
100	1	
11010	5	
1110100	11	Spans First and Second Double Word
11001	4	
0	0	
1110011	10	
101	2	
0	0	
0	0	
1110110	13	
. . .	. . .	

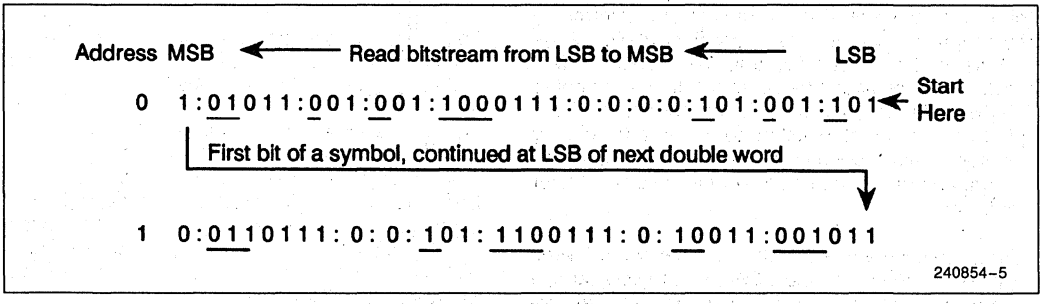


Figure 2-5. VRAM Bitstream Decoding Addresses

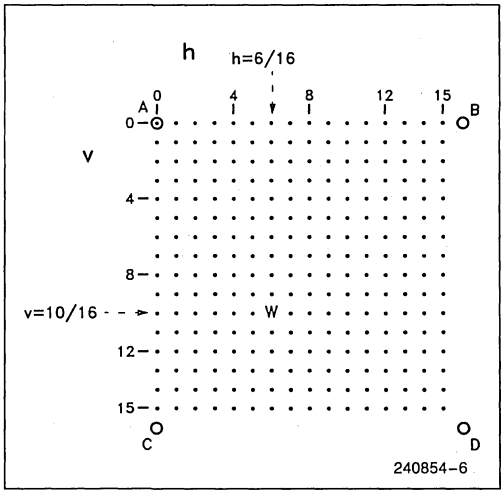


Figure 2-6. Pixel Interpolation

Pixel Interpolator

{*Pixint-c, Pixint*}

The pixel interpolator performs bilinear interpolation on four 8-bit pixels to generate, in effect, a pixel shifted by a fraction of a pixel position. See Figure 2-6. If the four pixels have values of A, B, C, and D; and the horizontal weight and vertical weight are h and v, respectively, the interpolated value W, ignoring any quantization effects, is given by:

$$W = A*(1-h)(1-v)+B*h(1-v)+C*(1-h)v+D*hv$$

The values of h and v are even multiples of 1/16. Figure 2-6 illustrates pixel interpolation with an h weight of 6/16 or 3/8 and a v weight of 10/16 or 5/8.

The pixel interpolar can operate in two modes: sequential-2D and random-2D. Sequential-2D mode is used for motion video decoding and when an array of pixels are interpolated with a common weighting. Random-2D mode is used either when the pixel arrays to be interpolated are not adjacent pixels in two rows or when the weight is changed for each interpolation. (The word random is used here to mean non-sequential.)

The example in Figure 2-7 shows a single row of pixels being interpolated in Sequential-2D mode using two rows from the original (source) bitmap. The h and v weighting are constant for all the interpolated pixels. In this case, the weights appear to be approximately h = 10/16 and v = 6/16.

```
A   B   E   F   I  ...  —First Input Row
  W   X   Y   Z    ...  —Interpolated Row
C   D   G   H   K  ...  —Second Input Row
```

Figure 2-7. Sequential-2D Pixel Interpolation

The pixel interpolator is pipelined and requires some startup sequence to fill the pipeline. Once filled, the pixel interpolator generates a new interpolated pixel every two T-cycles when in Sequential-2D mode. Source pixels are written into the interpolator as pixel pairs. In the case above, the pixel pair BA would be written first, followed by the pixel pair DC. It would seem more natural to refer to the pixel pair as AB, but because of the way 8-bit pixels are arranged in 16-bit words in VRAM, the left-most pixel on the screen is the least significant byte position. For example, if pixel A had a hex value of 0xAA and B had a value of 0xBB, the 16-bit word containing pixels A and B would have a value of 0xBBAA.

Then, two pixels are read from the interpolator. Because the pipeline isn't full yet, these pixels are read and discarded. This loop of writing two pixel pairs and reading two output pixels continues four times. The two pixels that are read this fourth time are the first two valid output pixels: W and X. The interpolator may also collect output (interpolated) pixels into pixel pairs. For exmple, pixels W and X, instead of being output separately, would be combined into a 16-bit pixel pair XW. Since there are two possible phase relationships between the input pixel pairs and output pixel pairs, the desired phasing (either X and W paired or Y and X paired) can be specified.

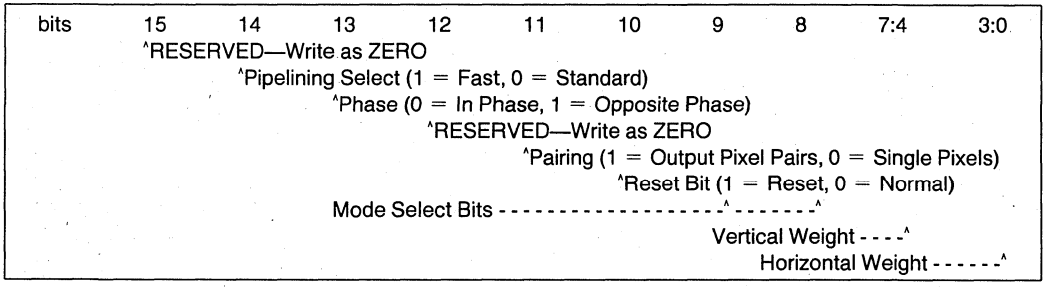


Figure 2-8. Pixel Interpolator Control Register

Random-2D interpolation is used either when the pixels to be interpolated are not in horizontal rows or when the weight is changed for each interpolated pixel. Examples for this are smooth warping or smooth scaling operations. In the case of Random-2D, the processing for successive interpolated pixels can not take advantage of pipelining; each pixel is considered to be the first pixel of a Sequential mode interpolation. The weight and the two input pixel-pairs are written into the interpolator. After waiting at least 10 T-cycles, the one interpolated pixel can be read. (The delay is 10 cycles when in the standard mode (bit 14 = 0) and 6 T-cycles when in the fast mode (bit 14 = 1).) Then, the next two input pixel-pairs and if necessary, the new weight value, are written, and 10 cycles later the next interpolated pixel can be read.

The h and v weight values, the mode selection, and other control bits are written to the pixel interpolator control register *(avg-c)*. The bit assignment for this register is in Figure 2-8. The least significant byte holds the 4-bit v value (bits 7:4) and the 4-bit h value (bits 3:0).

NOTE:

The values used for h and v here are numerators of the fraction where the implied denominator is 16.

MODE SELECT

Bits 8 and 9 are used to select on of four operating modes, of which only two are presently defined. These modes are given in Table 2-11.

Table 2-11. Mode Select Operating Modes

Bits 9:8	Mode
00	RANDOM-2D
01	Sequential-2D
10	RESERVED
11	RESERVED

RESET

Writing a ONE to bit 10 resets the pixel interpolator. The pixel interpolator must be reset prior to changing modes.

PAIRING

A ZERO in bit 11 causes the pixel interpolator to output individual pixels. A ONE causes the interpolator to collect adjacent pixels (in Sequential-2D mode) into 16-bit pixel pairs. This feature assists in motion video decoding, when combined with the ALU's dual-add-with-saturate operation, by allowing two pixels to be processed each cycle. The phasing used in collecting the pixel pairs is determined by the Phase bit described below.

PHASE

When output pixels are collected into pixel pairs, there are two possible alignments of the input pixel pairs to the output pixel pairs. The Phase bit (bit 13) selects the alignment to be used, based on the relative word alignment of the source and destination bitmaps in VRAM. When the Phase bit is set to a ZERO, this indicates that the bitmaps are in-phase. In this case, the first two output pixels are grouped into one 16-bit pixel pair (with the first pixel in the least significant byte). When the Phase bit is set to a ONE, the bitmaps are out-of-phase. In this case, the first pixel is placed in the most significant byte of the first pixel pair, with invalid data in the least significant byte, and the second and third output pixels are collected into the second pixel pair. This is illustrated in Figure 2-9.

PIPELINING

A ZERO in bit 14 causes the pixel interpolator to use the standard amount of pipeline delay. A ONE in this field will select the fast mode that has less pipeline delay. Table 2-12 shows the pipelining delay for both modes. Note that the effect of the phase bit is to add an extra pixel delay.

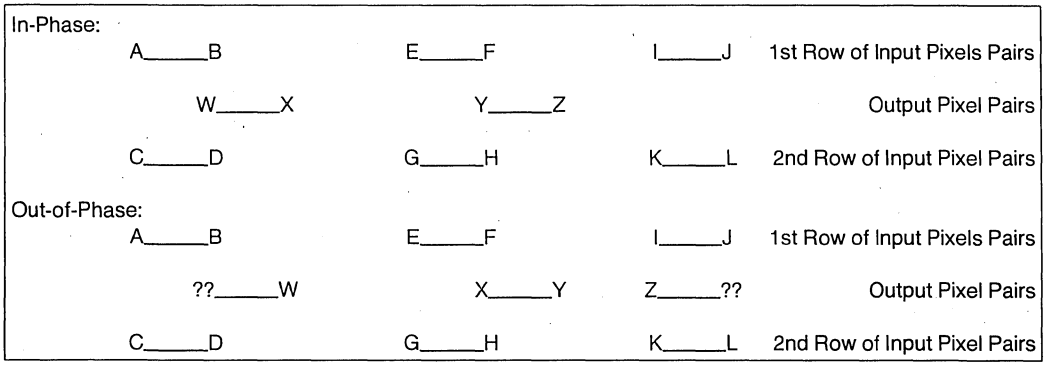


Figure 2-9. Pixel Pair Phases

Table 2-12. Pipelining Delay for Sequential-2D NON-PAIR Mode

Pipelining Bit (Bit 14)	Phase Bit (Bit 13)	Pipeline Delay in Output Pixels
0	0	6
0	1	7
1	0	2
1	1	3

When in PAIR mode (with bit 11 = one), the amount of pixel delay does not change, but half as many reads and writes are required to fill the pipeline because each read or write of the averager transfers two pixels. For example, when in the standard mode (bit 14 = 0), with zero phase (bit 13 = 0) and pair mode (bit 11 = 1), three indeterminate pixel pairs must be read before the first good pixel pair is read. In the same case but with the phase bit = 1, the fourth pixel pair read contains one good pixel and one indeterminate pixel, and the fifth pixel pair read contains two good pixels.

RESERVED

Bits 15 and 12 are reserved for future use. Write ZEROs into these bit positions.

Signature Register

{*hwid*}

The signature register can be read either by the host CPU or by microcode to determine the version of the 82750PB. The value of the signature register can be used to distinguish between the 82750PB in the 82750PA emulation mode, and the 82750PB in native mode. The currently defined signature values given in Table 2-13.

Table 2-13. Signature Values

Value	Definition
0xFFFE	The 82750PB Emulating the 82750PA
0XFFFC	The 82750PB in Native Mode

All other signature values are presently undefined but may be used in the future to denote other versions of the 82750 architecture.

Display Format Registers

{*yeven, yodd, vu, vptr*}

The 82750PB's processor can write to the display registers in the VRAM interface. These registers are pointers and pitch values that address display bitmaps and 82750DB register loads in VRAM. Pointers are 32-bit values that specify the specify the starting byte address of a bitmap or register load within a 4 GByte address space. The bottom two address bits are ignored since display bitmaps and register loads must start on a double word boundary. Therefore, the internal representation of a pointer is a 30-bit value. The pitch value associated with each pointer indicates the number of bytes between the start of two lines of a display bitmap or between the start of two register loads. The pitch is a single 16-bit value with its two least significant bits ignored, since the pitch must be an integer number of double words. Currently, there is also a restriction in the 82750DB limiting all display bitmap pitches to powers of two; so, the maximum display bitmap pitch is $\pm 2^{14}$ Bytes = ± 16 kBytes. The display registers are described in Table 2-14.

Table 2-14. Display Registers

Register	Description
yeven-lo, hi	This register pair points to the start of the Y bitmap or main bitmap that is to be displayed during an even field scan.
yodd-lo, hi	This register pair points to the start of the Y bitmap or main bitmap that is to be displayed during the odd field scan.
ypitch	The value in this register is added to the current Y bitmap pointer value each time a Y transfer is performed.
vu-lo, hi	This register pair points to the start of the VU bitmap. This bitmap is read to generate the VU values for both odd and even field scans.
vupitch	This value is added to the current VU bitmap pointer value each time a VU transfer is performed.
vptr-lo, hi	This register pair points to the start of a series of 82750DB register loads stored in VRAM.
vpitch	This value is added to the current 82750DB register load pointer each time a 82750DB register load is performed. The pitch is equal to the number of bytes from the start of one register load to the start of the next register load.

3.0 HARDWARE INTERFACE

VRAM Interface

The VRAM interface performs the following operations:

- Maintains VRAM pointers for the two input FIFOs, the two output FIFOs, the statistical decoder, the Y (main) bitmap, the VU bitmap, and the 82750DB register load.
- Decodes VBUS codes and takes appropriate actions such as generating a transfer cycle, scheduling refresh cycles, or generating interrupt conditions.
- Arbitrates VRAM accesses between the two input FIFOs, the two output FIFOs, the statistical decoder, the transfer request logic, the VRAM refresh logic, and the external VRAM access logic.
- During a memory cycle, performs appropriate address arithmetic on the VRAM pointer used for that memory cycle.
- As a result of certain VBUS codes, performs a shadow copy that consists of copying display-related VRAM pointer values from shadow registers (that are loaded by the host CPU or the microcode processor) to working registers where the various pointers are used for transfer cycles when the 82750DB is refreshing the display screen.

Table 3-1. VRAM Interface Signals

Signal	Description
MREQ#	**MEMORY REQUEST** is asserted during the first cycle of a VRAM memory access.
TRNFR#	The **TRANSFER** output indicates the current memory cycle is a result of a 82750DB transfer request.
RFSM#	The **REFRESH** output indicates the current memory cycle is a result of a 82750DB refresh request.
NXTFST#	The **NEXT FAST** output indicates the next memory access will use the same row address as the current memory access. This facilitates the use of page mode memory accesses.
MRDY#	The **MEMORY READY** input indicates the availability of valid data on the D[31:0] pins.

VRAM ACCESSES

The 82750PB can initiate five different types of memory accesses: FIFO read, FIFO write, transfer read, transfer write, and refresh. In addition, the 82750PB supports VRAM accesses by external logic. During an external access VRAM cycle, the 82750PB tri-states its VRAM address and data buses and performs a host VRAM read or host VRAM write cycle. There is another operation performed by the 82750PB, a shadow copy, that is not a VRAM cycle but is arbitrated as though it were, since no VRAM cycles can take place during a shadow copy.

The seven types of VRAM cycles initiated by the 82750PB, including host VRAM read and host VRAM write, begin with the 82750PB asserting a combination of its three VRAM cycle definition outputs: TRNFR#, RFSH#, and WE#. External logic detects the state of these signals, validated by MREQ#, and produces the appropriate sequence of VRAM control signals (RAS, CAS, etc.) to perform the type of memory cycle the 82750PB has requested. The 82750PB requires that each of these VRAM cycles take a minimum of two T-cycles, or T-states, denoted T1 and T2. External logic can insert additional T2 states in order to stretch the VRAM cycle to more than two T-cycles. The start of a new VRAM access cycle is signaled by the assertion of MREQ# for the first T-cycle, T1. The VRAM access cycle definition signals, TRNFR#, RFSH#, and WE#, are asserted at the start of T1 and remain asserted until the end of the last T2. Other VRAM operations can be described similarly by sequences of T-states. Refer to Figure 3-4 and 3-5 on page 42 for timing diagrams.

Table 3-2 defines the states used for all VRAM access operations. A state diagram for the VRAM/Host Interface is provided in Figure 3-1. This diagram includes the FIFO access states

Table 3-2. 82750PB VRAM Access States

State	Description
Ti	Idle State, No VRAM Activity
T1, TF1	First State of a VRAM FIFO Cycle
T2, TF2	Last State of a VRAM FIFO Cycle
TSC	The T-State required to perform a shadow copy
TTX1	First State of a VRAM Transfer Cycle
TTX2	Last State of a VRAM Transfer Cycle
TRF1	First State of a VRAM Refresh Cycle
TRF2	Last State of a VRAM Refresh Cycle

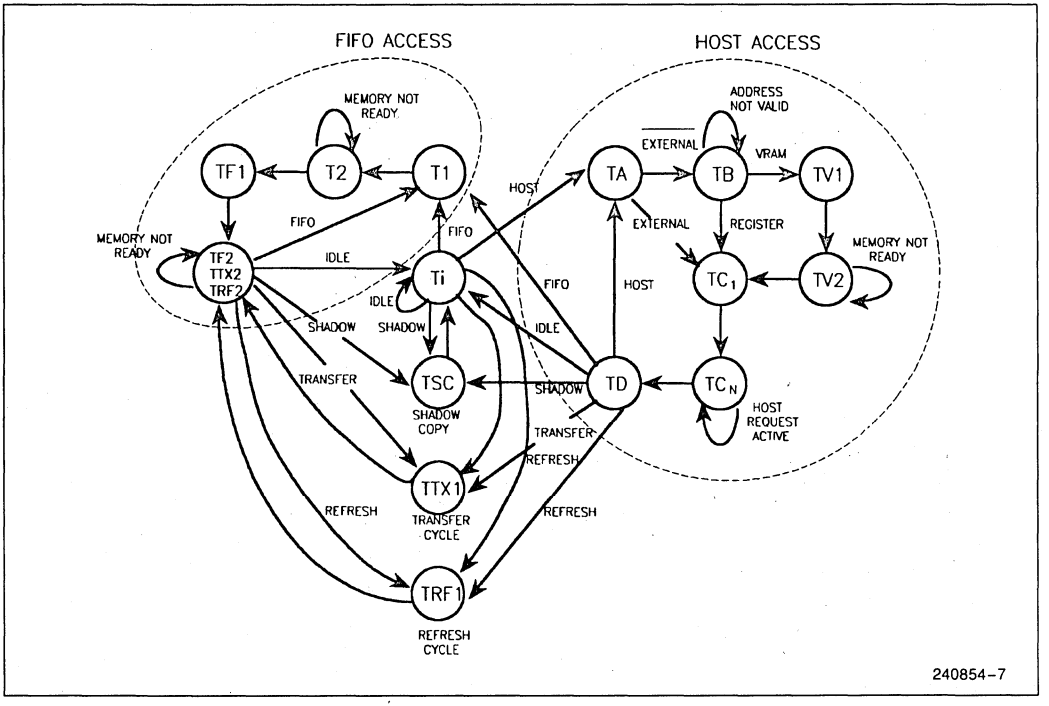


Figure 3-1. Access State Diagram

Note that during successive VRAM cycles it is not necessary to go back to the idle state, Ti, between each cycle; the T_{F2} state can be followed directly by a T1 state, starting at the next VRAM cycle. This results in efficient utilization of the 82750PB/VRAM bandwidth by allowing a VRAM cycle time of 2 T-states.

FAST VRAM CYCLES

When the 82750PB performs Data Read or Data Write VRAM cycles for the input or output FIFOs, it performs two 32-bit accesses to read or write one 64-bit value. These accesses are always performed in a sequence of EvenAddress followed by EvenAddress + 1, which guarantees both that the two sequential accesses will be in opposite banks and that the two accesses will be within the same VRAM page. This allows external logic to use either bank-interleaving or a page-mode access to complete the second access of the sequence and improve the VRAM bandwidth. However, the second access does not need to be handled differently from the first. Except for the assertion of the NXTFST# signal, both accesses are treated as standard VRAM accesses. External logic can ignore the NXTFST# signal, though, and treat the two accesses as two normal data read or data write cycles. Note that NXTFST# is not asserted for transfer, refresh, or host memory accesses.

The NXTFST# output signal is provided for cases when external logic can generate a faster access for the second access of the two sequential accesses. During such a pair of accesses, NXTFST# is asserted during the first of the two accesses in order to provide sufficient time for the external logic to generate the appropriate fast memory cycle for the second access. Refer to the timing diagrams in Figures 3-4 and 3-5 (page 42) for examples illustrating the use of the NXTFST# signal.

VBUS CODES

Transfer request, interrupt, and synchronization codes are sent over the BUS from the 82750DB to the 82750PB. The codes recognized by the 82750PB are listed in Table 3-3, along with the actions taken by the 82750PB as a result of receiving each code. Codes that cause TRANSFER cycles must be asserted for at least two clock cycles of the 82750PB to insure that, in the worst case, the 82750PB completes the transfer cycle before the code is released and the 82750DB starts shifting data from the VRAM shift registers. Other codes must also be asserted for a minimum of two 82750PB clock cycles. Only the codes given in the Table 3-3 are valid codes for the VBUS. Other codes are reserved for future use and should not be used. Once a transfer cycle code is sent to the 82750PB, any non-transfer code may be sent immediately. A subsequent transfer cycle code should be sent only after the current transfer cycle is completed.

Table 3-3. VBUS Codes

Binary	Name	Action
0000	YBMX	TXRD Cycle Using Yc; Yc = Yc + Yp*
0001	VUBMX	TXRD Cycle Using VUc; VUc = VUc + VUp
0010	REGX	TXRD Cycle Using Vc; Vc = Vc + Vp
0011	WRDIGX	TXWR Cycle Using Yc; Yc = Yc + Yp
0100	YNPBMX	TXRD Cycle Using Yc; Yc = Yc
0101	Reserved	Reserved
0110	Reserved	Reserved
0111	WRDIGNPX	TXWR Cycle Using Yc; Yc = Yc
1000	DFL	DFL Int; Shadow Copy**
1001	82750DBSD	82750DB Shutdown Interrupt
1010	REFRESH	Schedule N Refresh Cycles
1011	Reserved	Reserved
1100	VODD	VBI Int; OF Int; Shadow Copy Odd; Hline = 0***
1101	VEVEN	VBI Int; EF Int; Shadow Copy Even
1110	HLINE	Icnt++ (Increment Line Counter)
1111	NULL	No Action

NOTES:
*Yc—Y bitmap pointer, current; Yp—Y bitmap pitch; VU—VU bitmap; V—82750DB register load.
**Shadow Copy with Yc = Y-start-odd in odd field; Yc = Y-start-even in even field.
***Hline—Horizontal Line Counter.

PRIORITY

Each time the VRAM state machine completes a VRAM operation and returns to the Ti state, it examines all pending VRAM access requests and selects the highest priority request for the next VRAM operation. The priority ordering of these requests are listed in Table 3-4.

Table 3-4. Priority of VRAM Operations

Request Type	Priority
Transfer Cycle	Highest
Shadow Copy	•
Host Access	•
VRAM Refresh	•
FIFO Read/Write	Lowest

NOTE:
The shadow copy is treated as a VRAM operation even though it does not result in an access to VRAM.

The VRAM refresh operation is placed low on the priority list to reduce the latency in servicing transfer requests and external VRAM requests. Since a single REFRESH code from the 82750DB schedules a number of refresh cycles, a higher priority for refresh would cause all the refresh cycles to occur in a burst that would lock out all lower priority requests until all refresh cycles completed. Instead, the following restriction applies to all request types with higher priority than refresh: high priority requests, such as transfer cycles, shadow copies, and external VRAM access must occur infrequently enough to allow proper refresh of the VRAM chips. Transfer cycles and shadow copies, by their nature, occur infrequently so they are not generally a problem.

There is a separate priority scheme for the five FIFO channels. The scheme used is rotating priority with automatic override and single cycle arbitration. Rotating priority means that the priority is assigned in a fixed cyclic order with the lowest priority given to the FIFO channel that "won" the last FIFO access. There is only one level of memory , so the order that requests arrive is not a factor in the arbitration. The cyclic order is given in Figure 3-2.

As an example, if input FIFO 0 (abbreviated if0) was the last channel to perform a cycle, the priority order for the next FIFO access (from highest to lowest) would be: if1, sd, of0, of1, and if0.

Automatic override that the rotating cyclic priority can be bypassed if there is an URGENT condition for one of the channels. A channel is urgent if the microcode processor is frozen because the processor is waiting for that channel to be ready. The channel can be either an input channel that is empty or an output channel that is full. In this case, the urgent channel gets the next available cycle. However, the priority will still be lower than non-FIFO requests, such as refresh cycles.

Single clock cycle arbitration means that the selection of the next channel that will get an access occurs in a single T-cycle or T-state, either in a Ti state or during the last T2 state of the previous VRAM cycle.

VRAM POINTERS

The VRAM interface maintains VRAM pointers for the FIFOs, as well as display-related pointers for the 82750DB. Internally each pointer or address is stored as a 30-bit value addressing a double word in VRAM. The pointer values are read and written as two 16-bit words representing a 32-bit byte address (refer to the Figure 3-3). With a 30-bit double word address, the 82750PB can decode a VRAM address space of 1G double words or 4 GBytes.

Input and output FIFOs can address down to a single word or byte in VRAM. A FIFO's pointer is post-incremented or post-decremented in parallel with its VRAM read or write cycle.

The statistical decoder can only start decoding bitstreams on double word boundaries in VRAM and can only increment through VRAM. The decoder's pointer is post-incremented in parallel with each of its VRAM read cycles.

Display-related pointers are updated by adding a pitch value to the current value during the corresponding transfer cycle.

If a VRAM pointer appears on the B-Bus as source or as a destination then the following rules apply:

Rule 1

If a B-Bus destination refers to an address that is both Even and >0x1f, then the source is restricted to "-lo" pointers if the source refers to a pointer.

Rule 2

If a B-Bus destination refers to an address that is both Odd and >0x1f, then the source is restricted to "-hi" pointers if the source refers to a pointer.

SHADOW COPY

When a VODD, VEVEN, or DFL code is received from the 82750DB over the VBUS, a shadow copy is scheduled. The actual shadow copy will occur as soon as the priority logic allows. Any VRAM access in progress must complete and a pending transfer cycle, if any, must be performed before the shadow copy can start. During the operation, shadow registers for the Y-START, Y-PITCH, VU-START, VU-PITCH, 82750DB-START, and 82750DB-PITCH are copied into the corresponding working registers. During display refresh, the address arithmetic is performed on the working registers. The shadow registers can be loaded by the host CPU or by a microcode routine with less critical timing constraints, and then copied instantly by a shadow copy with it is time to update the registers, either prior to the next field or during the active display for split screen effects.

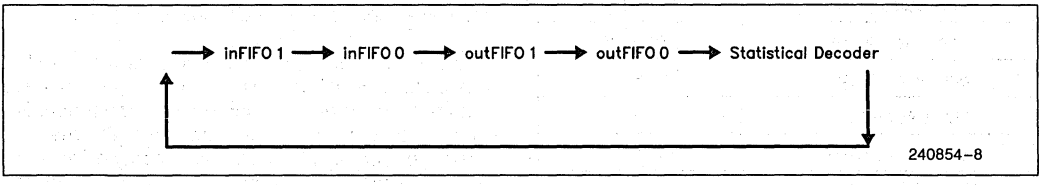


Figure 3-2. Cyclic Ordering of FIFOs

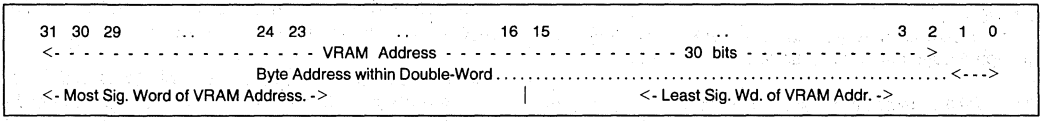


Figure 3-3. VRAM Addressing

There are actually two shadow registers for Y-START. One for start of odd fields and one for start of even fields. A VODD code causes Y-START-ODD to be copied into the working register Y-CURRENT. Similarly, a VEVEN code causes the Y-START-EVEN to be copied into Y-CURRENT. A DFL code causes the Y-START-ODD value to be copied if the most recent start of field code received is a VODD, or a Y-START-EVEN value if the most recent start of field code was a VEVEN. This allows a simple interlaced or non-interleaced display to be refreshed with no host CPU intervention. For more complex displays, such as split screens, the host CPU must update the shadow registers prior to each shadow copy. A shadow copy operation requires 2 T-cycles.

Host Interface

The Host Interface provides the following functions:

- Arbitrates host CPU and 82750PB access to VRAM.
- Provides the host access to external devices.
- Provides the host access to 82750PB internal registers and memories.

Signals specific to the Host Interface are listed in Table 3-5.

1

Table 3-5. Host Interface Signals

Signal	Description
HREQ#	**HOST REQUEST:** Asynchronous request from the host for all types of host access. Used both to request and release system buses.
HREG#	**HOST REGISTER:** Single-ranked control to request host access to 82750PB internal registers in concert with HRAM#.
HRAM#	**HOST VRAM:** Single-ranked control to request host access to VRAM in concert with HREG#.
HALEN#	**HOST ADDRESS LATCH ENABLE:** Asynchronous status from the host indicating the presence of valid address, write enable (transaction direction control), and the byte enables at the interface of the 82750PB.
HBUSEN#	**HOST BUS ENABLE:** 82750PB synchronous status granting the host access to the address, write enable, data bus, and byte enables at the interface of the 82750PB.
HRDY#	**HOST READY:** 82750PB synchronous status to the host indicating the presence of valid data appearing at the 82750PB's databus for VRAM and register accesses and optionally for external accesses.
HINT#	**HOST INTERRUPT:** 82750PB synchronous interrupt to the host, set under direct or indirect microprogram control.

Signals common to the host, VRAM, and external device interfaces are listed in Table 3-6.

Table 3-6. Host, VRAM, and External Device Interfaces

Signal	Description
A[31:2]	**ADDRESS BUS:** System address bus used to select unique VRAM, the 82750PB register, and external device locations that will be accessed under host control. The lower seven bits A[8:2] are bidirectional and are used during register accesses
D[31:0]	**DATA BUS:** Bidirectional system data bus used to transfer data to and from all sources and destinations. When transferring 16-bit host register values, the data bus MSH and LSH will both carry identical values.
WE#	**WRITE ENABLE:** Bidirectional, single-ranked signal used to determine the data transfer direction. When active during host register cycles, data flows from the host to an 82750PB destination. During host VRAM cycles, WE# active will define the data direction to be from the host to VRAM.
BE[3:0]#	**BYTE ENABLE:** Bidirectional signals used to select the bytes that will be modified during data transactions. All host register transactions are performed 16 bits at a time, while VRAM may be modified 8 bits at a time.

As with VRAM operations, host operations are described through a sequence of T-states. Table 3-7 defines the T-states used to implement all host transactions with VRAM, external devices, and the 82750PB.

The master execution state diagram that defines the VRAM/Host transactions is provided in Figure 3-1.

Table 3-7. 82750PB Host Transaction States

State	Description
TA	First state of any host transaction. Entry into TA will be granted after HREQ# has been asserted. During this state, the 82750PB will tri-state its address, data bus, write enable, and byte enable signals to provide a full cycle of "dead-band" before the assertion of HBUSEN#. In the state immediately following TA HBUSEN# will assert, allowing the host to drive the host buses.
TB	First cycle in which the host is granted bus access for register or VRAM transactions. The sequencer will remain in TB until HALEN# is received, indicating that the address write enable and byte enable signals are stable at the 82750PB pins.
TC1	First cycle that output data is valid.
TCn	This state is entered to wait for the completion of the current host cycle. The cycle is defined as complete when HREQ# deasserts. HRDY# is asserted along with valid data until the transition to state TD occurs.
TD	The last cycle of a host transaction. HBUSEN# is deasserted allowing one dead-band cycle to allow control of the address, data, write enable, and byte enable signals to be returned to the 82750PB.
TV1	First cycle of a Host VRAM transaction. Memory is requested and is followed by a transition to TV2.
TV2	Last cycle of a Host VRAM transaction. The sequencer will remain in TV2 until MRDY# is received.

A single stage of input synchronization is employed for HREG#, HRAM#, WE#, and BE[0]#, while HREQ# and HALEN# are programmable to have one or two stages by bit 12 of the Microcode Processor Control Register. See Table 3-10. T-state transitions are caused by the synchronized versions of these signals.

The synchronized versions of HREG# and HRAM# must be stable before entry into T-state TA. The synchronized versions of WE#, BE[0]#, and HALEN# should be stable before exiting T-State TB. Once asserted, all of the above signals should remain stable until the deassertion of HBUSEN#.

The type of host cycle to perform is determined by the states of HREG# and HRAM# as indicated in Table 3-8.

Table 3-8. Host Cycle Types

HREG#	HRAM#	Host Cycle Type
1	1	External
0	1	Register
1	0	VRAM
0	0	Reserved

HOST REGISTER ACCESS

The host has access to the 82750PB's internal registers and memories to monitor and control the operation of the microcode processor, provide a means of debugging microprogram routines, and to function as the primary test port for production testing.

Register access is initiated by the host asserting HREQ#, HREG#, and HRAM# as shown in Table 3-8 and in the timing diagrams on pages 42 through 45. After the host has been granted bus access by an active HBUSEN# in state TB, the address, write enable, and byte enables may be driven. After these signals have stabilized HALEN# is asserted, enabling a read or a write operation to occur.

In the case of a register read, state TC1 is entered and the data bus is driven with the internal value. One cycle later, a transition to state TC occurs, and HRDY# activates, signaling the presence of stabilized data at the 82750PB data pins. This state (TC) will be maintained until the host deasserts HREQ#, signaling the completion of the cycle that caused a transition to state TD.

In the case of a register write, TC1 is again entered (from TB), but the data bus may now be driven by the host. (During host cycles, data bus drive activity is indirectly controlled by WE# and an additional dead-band is provided by entry into state TC1 to allow for internal WE# stabilization.) Stable data at the 82750PB interface, as well as the completion of the write cycle, is signaled by the deassertion of HREQ#. As with reads, the deactivation of HRDY# signals the transition to state TD.

As state TD is entered, HRDY# and HBUSEN# deassert, the address data, write enable, and byte enables tri-state, and bus control is returned to the 82750PB in the following cycle.

HOST VRAM ACCESS

Because the 82750PB is so closely coupled with VRAM, host accesses to VRAM are arbitrated and controlled by the 82750PB. VRAM access is initiated by the host asserting HREQ#, HREG#, and HRAM# as shown in the Host Cycle Table above and in the timing diagrams on pages 42 through 45. After the host has been granted bus access by an active HBUSEN#, the address, write enable, and byte enables may then be driven. After these signals have stabilized at the memory devices (or longest relevant propagation path), HALEN# is asserted, enabling a read or a write operation to occur.

Because VRAM will not drive the data bus until after a memory request, a transition into state TC1 to allow for data bus direction stabilization is not required. Instead, a transition to state TV1 occurs, which asserts MREQ# for a single cycle and is followed by a transition to TV2. TV2 will remain the current state until the reception of an active MRDY#.

In the case of a VRAM read, the memory data bus will be driven during TV1, and valid data will appear in state TV2. Data will be guaranteed valid coincident with the deassertion of MRDY# from memory.

In the case of a VRAM write, the memory data bus is driven with valid data during TV1. Again the reception of MRDY# will serve to indicate the completion of the memory operation.

NOTE:
The host device must be able to transmit or receive memory data in order to be valid at the trailing edge of MRDY# at the data's destination (memory or host).

After MRDY# becomes active, a transition from TV2 into TC1 is accomplished to allow time to propagate data to the host. TC is then entered to await the deassertion of HREQ# (if it has not already occurred). TD is then entered, duplicating the dead-banding previously described.

HOST EXTERNAL ACCESS

In addition to VRAM and register host access, an external device access mechanism is provided. During this access, upon the receipt of HREQ# with HREQ# and HRAM# inactive, the 82750PB releases the address, data, write enable, and byte enables in state TA.

The difference here is that state TC1 is directly entered from TA, thereby ignoring any transitions of HALEN#. Since the 82750PB also ignores the data bus direction control (write enable) the host and an external device may communicate unencumbered by the 82750PB.

Entry into state TC directly follows TC1 in the expected sequence and remains there until HREQ# is released. This is followed by entry into TD. HBUSEN# is asserted during the timing that TC1 and TCN are active.

During an external access, HRDY# is not asserted unless the external logic asserts MRDY# as shown in Figure 3-7.

HOST REGISTER ADDRESS MAPPING

Table 3-9 shows the host address mapping of the on-chip registers and memories, in terms of the offset in bytes, from the base address for 82750PB accesses. Note that the 82750PB only supports word accesses to these registers. Therefore, the least significant bit of the byte offset should be set to zero. The 82750PB forms the register address from inputs on the A[31:2] pins and BE#[3:0] pins. The A[31:2] specify the double word address of the register, and combinations of the BE# pins determine which of the two words with the double word is being addressed. BE#[3:0] = 1100_2 selects the least significant word within a double word, and BE#[3:0] = 0011_2 selects the most significant word within a double word. These are the only two valid patterns for BE# inputs during a host register access cycle.

Table 3-9. Host Address Mapping

Byte Address	Description
0x000–0x07E	(a) A source and destination registers
0x080–0x0FE	(b) B source and destination registers
0x100–0x17E	(c) Microcode processor control and status registers
0x180–0x1FE	(d) VRAM pointer RAM

NOTE:

The host should only perform 16-bit word reads or writes to 82750PB registers. The 82750PB does not support byte reads or writes or double word reads or writes to on-chip registers.

When the host CPU reads or writes to areas (a, b, or d) and the 82750PB is not already in a HALT state, the microcode processor is automatically HALTED for the one T-cycle actually required to complete the data transfer, and then the processor is restarted after the transfer is complete. If the 82750PB is in a HALT state when the host access is initiated, it will remain in the HALT state following the completion of the access. This is transparent to both the host CPU and the microcode processor.

During an access to areas (a) or (b), bits 6:1 of the byte offset should be set to the source or destination code for the register that will be read or written. The coding is the same as used in the microcode instruction word. Bit 0 is always set to a zero. Refer to the 82750PB Source and Destination Coding Table found in Chapter 4.

Area (c) contains one write-only register, the CONTROL register, and two read-only registers, the INTERRUPT FLAG register and the microcode PROCESSOR STATUS register. The CONTROL register is used to halt or single-step the microcode processor, which enables or masks interrupts to the host CPU, selects the signal that is output via the PMON/FRZ pin, and enables or disables the 82750PA emulation mode. The bit assignments for the CONTROL register are given in Table 3-10.

During reset of the 82750PB, the HALT bit is set to a one, the six Interrupt Enable bits are reset to zero, the Disable SYNC bit is set to zero, the PMON/FRZ bit is set to zero (so that the FRZ signal is output), and the Enable 82750PB bit is reset to zero (so that on reset, the 82750PB starts in a 82750PA emulation mode).

Table 3-10. Bit Assignments for Microcode Processor CONTROL Register (Write-Only, Byte Offset = 0x100)

Bit	Name	Description
Bit 0	HALT	1 = Microcode Processor Halt 0 = Microcode Processor Run
Bit 1	SINGLE-STEP	1 = Execute One Instruction and then Halt (Only when Already Halted, Bit 0 = 1) 0 = No Action
Bit 2	Enable MCINT	1 = Enable Microcode Interrupts to Host CPU 0 = Mask Microcode Interrupts
Bit 3	Enable VBI	1 = Enable Vertical Blanking Interrupt to Host CPU 0 = Mask Vertical Blanking Interrupt
Bit 4	Enable DFL	1 = Enable DFL Interrupt to Host CPU 0 = Mask DFL Interrupt
Bit 5	Enable SD	1 = Enable 82750DB Shutdown Interrupt to Host 0 = Mask SD Interrupt
Bit 6	Enable OFI	1 = Enable Odd Field Interrupt 0 = Mask OF Interrupt
Bit 7	Enable EFI	1 = Enable Even Field Interrupt 0 = Mask EF Interrupt
Bits 8–11*		1 = RESERVED; Write as Zeros
Bit 12	Disable SYNC	1 = Disable Synchronizers for HREQ#/HALEN# 0 = Enable Synchronizers for HREQ#/HALEN#

Bit 13	PMON/FRZ	1 = Output FRZ # Signal on PMFRZ # Pin
		0 = Output PMON # Signal on PMFRZ # Pin

Bit 14		1 = RESERVED; Write as Zero
Bit 15	Enable 82750PB	1 = Enable 82750PB Mode 0 = Enable 82750PA Emulation Mode

*All other bits are reserved for future use, and should be written as zeros.

The INTERRUPT FLAG register holds a flag for each of the six interrupt sources. A flag bit is set to a one when the interrupt condition is detected (independent of the state of the corresponding Interrupt Enable/Mask bit in the CONTROL register), and all flags are cleared to zero each time the INTERRUPT FLAG register is read. If this register is read during the same cycle that an interrupt condition is detected, the flag bit corresponding to that interrupt condition will remain at a one. This new interrupt condition will then be seen by the host processor when it next reads the INTERRUPT FLAG register. The flag insures that an interrupt is not lost if it occurs at the same cycle that the INTERRUPT FLAG register is read (and reset). In addition, the Microcode Interrupt source has an overflow flag that indicates if more than one Microcode Interrupt has occurred since the Interrupt Flag register was last read. The bit assignments for the INTERRUPT FLAG register are listed in Table 3-11.

The PROCESSOR STATUS register holds four status bits: HALT, FREEZE, PMON, and SYNC status. HALT indicates that the processor is HALTED due to a HALT bit in the CONTROL register being set to a ONE or due to the HALT# pin being asserted. FREEZE indicates that the processor is waiting for one of the VRAM channels to become ready or is waiting for an access to the VRAM pointer RAM. PMON is a signal that can be toggled by a special ALU opcode or a special B source code. This signal can be used for performance monitoring of microcode. SYNC status bit indicates the presence or absence of the internal synchronizers for HREQ# and HALEN# inputs. In addition, the Interrupt Mask bits that are written into the PROCESSOR CONTROL register can be read from this register. These mask bits are read in the same polarity that they are written, but note that the bit positions and bit ordering are not consistent with the PROCESSOR CONTROL register. The bit assignments for this register are given in Table 3-12.

Address mapping for areas (a), (b), and (d) are given in Tables 3-13 to 3-15.

Table 3-11. Bit Assignments for INTERRUPT FLAG Register (Read-Only, Byte Offset = 0x100)

Bit	Description
Bit 8:0	Not Used, the State of These Bits Are Not Specified
Bit 9	EF Interrupt Flag
Bit 10	OF Interrupt Flag
Bit 11	MCINT Overflow Flag
Bit 12	82750DB Shutdown Interrupt
Bit 13	MCINT Microcode Interrupt
Bit 14	VBI Vertical Blanking Interrupt
Bit 15	DFL Display Format Load Interrupt

Table 3-12. Bit Assignments for PROCESSOR STATUS Register (Read-Only, Byte Offset = 0x102)

Bit	Description
Bit 0	HALT (1 = Halted, 0 = Running)
Bit 1	FREEZE (1 = Frozen, 0 = Running)
Bit 2	PMON (1 = Active, 0 = Inactive)
Bit 3	Synchronizers on HREQ#/HALEN# (0 = Enabled, 1 = Disabled)
Bit 9:4	Not Used, the State of These Bits is Not Specified
Bit 10	MCINT Microcode Interrupt Mask
Bit 11	VBI Vertical Blanking Interrupt Mask
Bit 12	DFL Display Format Load Interrupt Mask
Bit 13	82750DB Shutdown Interrupt Mask
Bit 14	OF Interrupt Mask
Bit 15	EF Interrupt Mask

Table 3-13. 82750PB A Bus Source/Destination Address Mapping

Address (Hex)	ADST	ASRC
0x000	Null	Null
0x002		hwid
0x004		cc
0x006	maddr	
0x008		alu
0x00A	cnt	cnt
0x00C	cnt2	cnt2
0x00E	lcnt	lcnt
0x010	r0	r0
0x012	r1	r1
0x014	r2	r2
0x016	r3	r3
0x018	r4	r4
0x01A	r5	r5
0x01C	r6	r6
0x01E	r7	r7
0x020	mcode3	mcode3
0x022	mcode2	mcode2
0x024	mcode1	mcode1
0x026	pc	pc
0x028	pixint-c	
0x02A	pixint	pixint
0x02C	*dram1	*dram1
0x02E	*dram2	*dram2
0x030	*dram1++	*dram1++
0x032	*dram2++	*dram2++
0x034	*dram1--	*dram1--
0x036	*dram2--	*dram2--
0x038	dram1	dram1
0x03A	dram2	dram2
0x03C	dram3	dram3
0x03E	dram4	dram4
0x040	*out1	*in1

Address (Hex)	ADST	ASRC
0x042	out1++	*in2
0x044	shift-hi	*stat
0x046	out1-hi	*stat#
0x048	*out2	
0x04A	out2++	
0x04C	shift-r	
0x04E	out2-hi	
0x050	out1-c	
0x052	in1-c	
0x054	shift-l	
0x056	in1-hi	
0x058	out2-c	
0x05A	in2-c	
0x05C		
0x05E	in2-hi	
0x060	r8	r8
0x062	r9	r9
0x064	r10	r10
0x066	r11	r11
0x068	r12	r12
0x06A	r13	r13
0x06C	r14	r14
0x06E	r15	r15
0x070	cc	shift
0x072	fcnt	fcnt
0x074	*dram3	*dram3
0x076	*dram4	*dram4
0x078	*dram3++	*dram3++
0x07A	*dram4++	*dram4++
0x07C	*dram3--	*dram3--
0x07E	*dram4--	*dram4--

Table 3-14. 82750PB B Bus Source/Destination Address Mapping

Address (Hex)	BDST	BSRC
0x080	Null	Null
0x082		alu
0x084	*dram3	*dram3
0x086	*dram4	*dram4
0x088	*dram3++	*dram3++
0x08A	*dram4++	*dram4++
0x08C	*dram3--	*dram3--
0x08E	*dram4--	*dram4--
0x090	r0	r0
0x092	r1	r1
0x094	r2	r2
0x096	r3	r3
0x098	r4	r4
0x09A	r5	r5
0x09C	r6	r6
0x09E	r7	r7
0x0A0	r8	*in1
0x0A2	r9	*in2
0x0A4	r10	*stat
0x0A6	r11	*stat#
0x0A8	r12	circbuf
0x0AA	r13	
0x0AC	r14	
0x0AE	r15	
0x0B0	circbuf	literal 0
0x0B2		literal 1
0x0B4	*dram1	literal 2
0x0B6	*dram2	literal 3
0x0B8	*dram1++	literal 4
0x0BA	*dram2++	literal 5
0x0BC	*dram1--	literal 6
0x0BE	*dram2--	literal 7
0x0C0	*out1	prof

Address (Hex)	BDST	BSRC
0x0C2	out1++	
0x0C4	out1-lo	out1-lo
0x0C6	out1-hi	out1-hi
0x0C8	*out2	stat-lo
0x0CA	out2++	stat-hi
0x0CC	out2-lo	out2-lo
0x0CE	out2-hi	out2-hi
0x0DO	out1-c	out1-c
0x0D2	in1-c	in1-c
0x0D4	in1-lo	in1-lo
0x0D6	in1-hi	in1-hi
0x0D8	out2-c	out2-c
0x0DA	in2-c	in2-c
0x0DC	in2-lo	in2-lo
0x0DE	in2-hi	in2-hi
0x0E0	stat-ram	r8
0x0E2	stat-c	r9
0x0E4	stat-lo	r10
0x0E6	stat-hi	r11
0x0E8	yeven-lo	r12
0x0EA	yeven-hi	r13
0x0EC	yodd-lo	r14
0x0EE	yodd-hi	r15
0x0F0	ypitch	shift
0x0F2		stat-c
0x0F4	vu-lo	*dram1
0x0F6	vu-hi	*dram2
0x0F8	vupitch	*dram1++
0x0FA	vpitch	*dram2++
0x0FC	vptr-lo	*dram1--
0x0FE	vptr-hi	*dram2--

Table 3-15. VRAM Pointer RAM Mapping

Byte Address	Name	Description
0x180 0x182	Yw-lo Yw-hi	Working Copy of Y Pointer
0x184 0x186	out1-lo out1-hi	Output FIFO 1 Pointer
0x188	Yw-pitch	Working Copy of Y Pitch
0x18A		RESERVED
0x18C 0x18E	out2-lo out2-hi	Output FIFO 2 Pointer
0x190 0x192	VUw-lo VUw-hi	Working Copy of VU Pointer
0x194 0x196	in1-lo in1-hi	Input FIFO 1 Pointer
0x198	VUpitchw	Working Copy of VU Pitch
0x19A	vpitchw	Working Copy of 82750DB Pitch
0x19C 0x19E	in2-lo in2-hi	Input FIFO 2 Pointer
0x1A0 0x1A2	vptrw-lo vptrw-hi	Working Copy of 82750DB Pointer
0x1A4 0x1A6	stat-lo stat-hi	Working Copy of Statistical Decoder Pointer
0x1A8 0x1AA	Yeven-lo Yeven-hi	Shadow Copy of Y Start Even Pointer
0x1AC 0x1AE	Yodd-lo Yodd-hi	Shadow Copy of Y Start Odd Pointer
0x1B0	Ypitch	Shadow Copy of Y Pitch
0x1B2	rfcnt	RFSH Cycles per RFSH Code from 82750DB
0x1B4 0x1B6	VU-lo VU-hi	Shadow Copy of VU Start Pointer
0x1B8	VUpitch	Shadow Copy of VU Pitch
0x1BA	vpitch	Shadow Copy of 82750DB Pitch
0x1BC 0x1BE	vptr-lo vptr-hi	Shadow Copy of 82750DB Pointer

NOTE: Register rfont write only register and should never be read.

Initializing the 82750PB

The 82750PB is placed in a RESET state by asserting RESET# for at least ten T-cycles. In the RESET state, which continues until RESET# is released, all of the 82750PB's outputs are tri-stated for compatibility with board test requirements.

Proper initialization of the 82750PB requires that the 82750PB is held in a RESET state by keeping RESET# active for at least 10 T-cycles, and then releasing RESET#. This is referred to as the INITIAL state. In the INITIAL state:

- The microcode processor is halted.
- All six interrupts are masked, and the interrupt latches are cleared.
- The 82750PA/82750PB instruction format select bit is set to the 82750PA.
- The VRAM interface is ready to service VRAM requests; however, none of the VRAM pointers are valid.

- The number of refresh cycles that will be generated each time a RFSH code is received from the 82750DB is set to 14 cycles.
- All bidirectional I/O pins are tristated.

After the 82750PB has been initialized, i.e., placed in the INITIAL state, but prior to releasing the 82750DB's reset signal, the following operations must be performed:

- Load the REFRESH-CYCLES-PER-LINE register with the appropriate value (the equation for the value is: VALUE = $(2^N - 1)$, where N is the number of cycles; for example, 5 refresh cycles would result in VALUE = $2^5 - 1 = 31_{10} = 001F_{16}$. The refresh register is 14 bits wide and the way it works is to generate one refresh everytime a right shift results in a '1' bit. It continues the right sifting until it finds a '0' bit and halts. Hence from programming point of view: $001F_{16} = FFDF_{16} = 5$ refresh cycles per line.
- Load the shadow copies of Y, VU, and 82750DB pointers and pitches.
- Load the appropriate 82750DB Register Load list into VRAM starting at the address pointed to by the 82750DB pointer.

Prior to releasing the microcode processor from its HALTed state to run a microcode program, the following operations must be performed:

- If 82750PB code is to be executed, bit 15 of the 82750PB CONTROL register must be set to a one.
- Load a microcode program into microcode RAM on the 82750PB by writing to the three instruction word registers (*mcode1* – the most significant word of the instruction, *mcode2,* and *mcode3* – the least significant word of the instruction, the one containing the next address field) and then writing to *maddr,* the address in microcode RAM where the instruction will be loaded.
- Load the PC with the address in microcode RAM of the first instruction to be executed.
- Write to the 82750PB CONTROL register with the HALT bit (bit 0) set to zero, causing the processor to start executing an instruction sequence, or with the SINGLE-STEP bit (bit 1) set to a one (keeping HALT also set to one), causing the processor to execute a single instruction.

Performance Monitoring

Two signals, FRZ# and PMON#, which are useful for microcode performance monitoring, are available both as external signals, multiplexed on a single output pin, and as bits in the Processor Status register. FRZ# is active for each T-cycle when the microcode processor is frozen, waiting for access to VRAM or to the VRAM Pointer RAM. PMON# can be toggled by a special ALU opcode or a special B bus source code. This allows PMON# to be used to indicate what particular segment of microcode is being execute. The PMON/FRZ bit in the Processor Control register selects the signal that is being output.

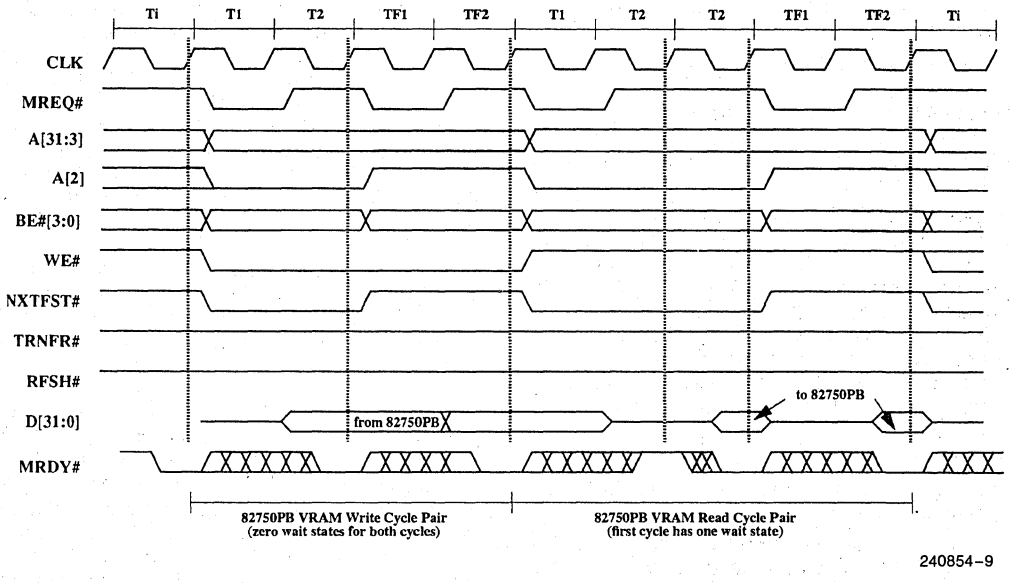
Freezes may indicate that the microcode routine is not making the most efficient use of the input and output FIFO buffering. This is particularly important for the inner loops of graphics and video routines that are memory-bandwidth limited. Ideally, inner loops should be balanced so that the rate pixels are processed is equal to the rate that they can be read from and written to VRAM with no freezes. The buffering in the input and output FIFOs serve to make sequential reads and writes to VRAM more efficient by performing full 64-bit reads and writes, instead of individual 8-bit or 16-bit accesses. This has the effect of averaging the VRAM read/write rate over a number of instruction times. For example, if the 82750PB is performing a 64-bit read or write every 8 T-cycles, for an average of 8 bits per T-cycle, a two instruction inner loop could read one 8-bit pixel and write one 8-bit pixel without any freezes occurring (assuming the source pixels and the destination pixels are each sequential).

The PMON# provides a more standard performance monitoring capability by indicating when a particular segment of microcode, bracketed by special instructions that toggle the PMON# signal, is being executed. This allows either absolute execution-time measurement or measurement of the fraction of the total execution time that is required by the segment. Either the ALU opcode 'prof' or the B bus source code 'prof' will toggle the PMON signal.

An external HALT pin is provided on the 82750PB to allow external debugging hardware to immediately halt the microcode processor. Activating this input causes the microcode processor to halt prior to executing the next instruction. When the processor is halted, the VRAM interface portion of the 82750PB continues to operate normally, performing transfer cycles, refresh cycles, and shadow copies as requested by the 82750DB.

Host/VRAM Timing Diagrams

Figures 3-4 through 3-8 are Host/VRAM Timing Diagrams.



NOTES:
1. Address pin A[2] is always ZERO for the first cycle of a cycle pair and ONE for the second cycle.
2. The two cycles of a cycle pair are both writes or both reads.

Figure 3-4 VRAM Read and Write Cycles

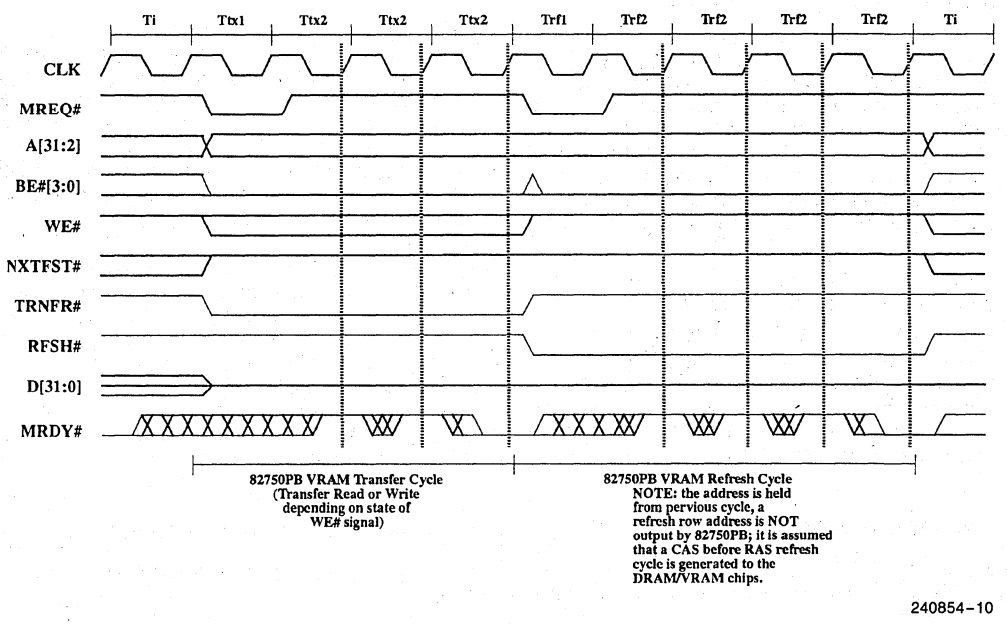
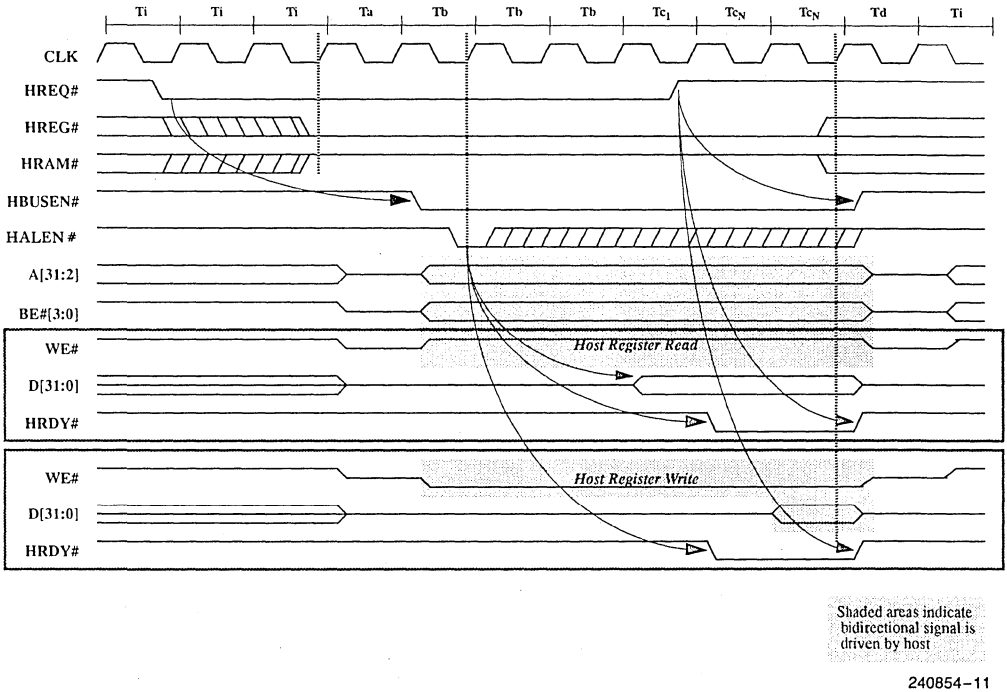


Figure 3-5. VRAM Transfer and Refresh Cycles



NOTES:
1. MREQ#, RFSH#, TRNFR#, and NXTFST# remain inactive during Host Register Read and Write cycles.
2. If HALEN#/HREQ# synchronizers are disabled then the second Ti and Tb states will be missing.

Figure 3-6. Host Register Read and Write Cycles

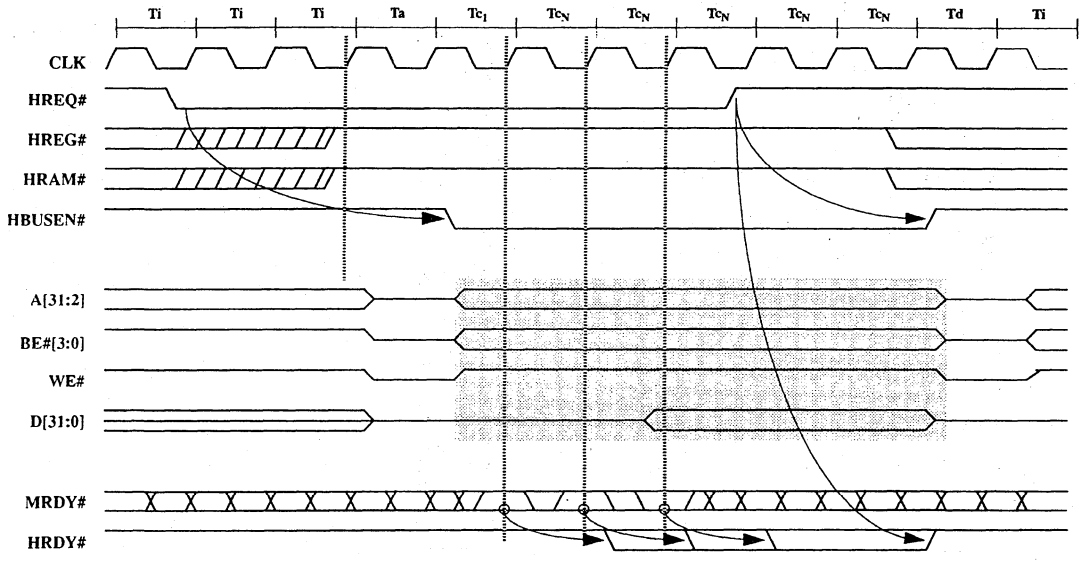
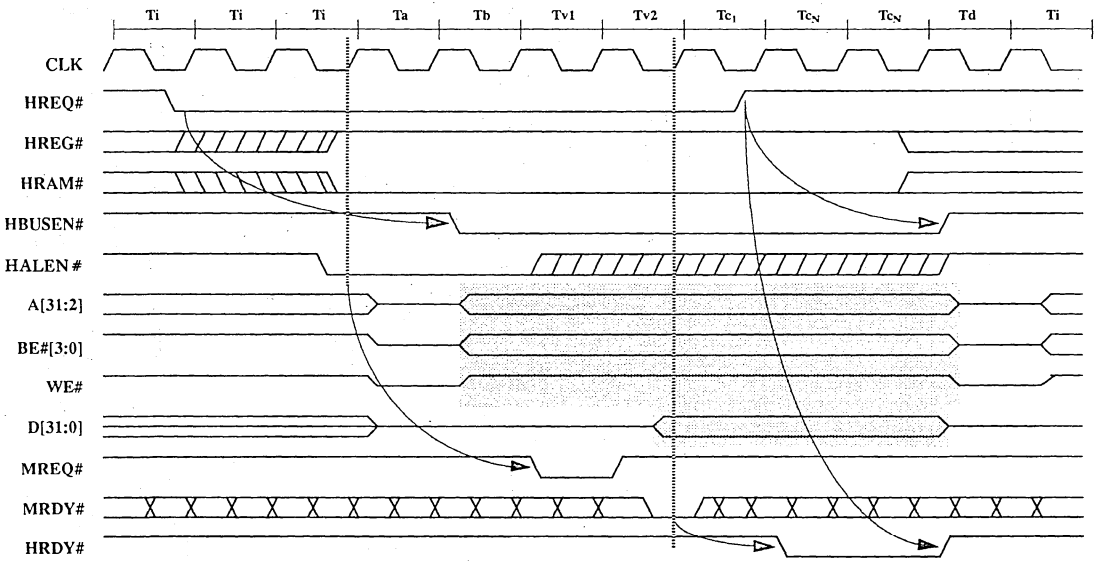


Figure 3-7. Host External Read and Write Cycles

NOTES:
1. MREQ#, RFSH# TRNFR#, and NXTFST# remain inactive during Host External Read and Write cycles.
2. If the Synchronizer on HREQ# is disabled, then the second Ti state will be missing.



Note: 82750PB will stay in Tb for the maximum of:

1) one T-state, OR

2) two T-states after VALEN# goes low.

Shaded areas indicate bidirectional signal is driven by host

240854-13

Figure 3-8. Host VRAM Read and Write Cycles

NOTES:
1. RFSH#, TRNFR#, and NXTFST# remain inactive during Host VRAM Read and Write cycles.
2. If the Synchronizers on HREQ#/HALEN# is disabled, then the second Ti state will be missing.

4.0 MICROCODE INSTRUCTION FORMAT

Overview

The 82750PB executes two slightly different instruction formats: one that is backward compatible with the 82750PA and another that allows full access to the microcode resources of the 82750PB. The 82750PA/82750PB bit in the 82750PB processor control register determines which instruction format is in effect (see Chapter 3). On reset, the 82750PB is placed in 82750PA instruction format mode. In this mode the 82750PB will execute binary microcode originally assembled for the 82750PA in a manner that is functionally equivalent to the 82750PA.

The following description applies to the 82750PB instruction format. Exact definitions of 82750PB instruction formats and field codings are shown in Figure 4-2 and Table 4-5.

Instruction Sequencing

The instruction word for 82750PB's microcode processor is 48 bits wide. The Microcode RAM holds 512 instructions. Nine bits of each instruction specify the address of the next instruction to be executed. Each instruction fetch reads two instructions (of odd address and even address pair) using the upper eight bits of the 9-bit instruction address. Both the LSB of the instruction address and a Condition Flag bit, selected from eight possible branching conditions, are used to determine whether the next instruction to be executed is the even address instruction or odd address instruction, according to the logic table shown as Table 4-1.

Table 4-1. Microcode Next Instruction Selection

LSB of Address	Condition Flag State	Next Instruction
0	0 (FALSE)	EVEN
0	1 (TRUE)	EVEN
1	0 (FALSE)	ODD
1	1 (TRUE)	EVEN

For an unconditional branch, the condition flag FALSE (which is always zero) is selected; this causes the LSB of the address to be passed through to select the next instruction: LSB = 0 selects EVEN and LSB = 1 selects ODD. This allows unconditional branching to any of the 512 instructions in the RAM. For a conditional branch, the LSB of the address is set to a one; this causes the state of the condition flag to select the next instruction: FALSE selects the ODD instruction and TRUE selects the EVEN instruction. Therefore, a conditional branch jumps to either the odd or even instruction of an odd/even pair depending on the state of the condition.

Instruction Word Field Descriptions

Each field of the microcode instruction format is described in the following sections.

NADDR—NEXT INSTRUCTION ADDRESS FIELD

This field holds the address of the next instruction to be executed. Taking advantage of the fact that the microcode RAM is physically organized as 256 deep by 96 wide (two instructions are fetched per read cycle), a zero delay two-way branch can be achieved. The only case in which this field is not used to determine the address of the next instruction to be executed is when an instruction writes to the PC. (The term PC refers to the register that holds the address of the next instruction to be executed.) When an instruction loads the PC a one instruction delay occurs before the load takes effect. Therefore, the instruction pointed to by the next instruction field of the instruction that loads the PC is executed before the jump to the new address occurs. This is shown in Table 4-2.

There are no restrictions on the instruction following a PC load; it will always be executed, even while single stepping the processor or if the processor is frozen on that instruction.

CFSEL—CONDITION FLAG SELECT FIELD

This field selects which condition flag will be used with the LSB of NADDR to select the next instruction from the odd/even pair. The condition flag assignment is given in Table 4-3.

Table 4-2. PC Load Example

Addr	Instruction	NADDR	Comments
10	pc = 0	55	Load PC with zero.
55	r0 = 1	X	This instruction is executed but its next address field is ignored.
0	r1 = r0	25	PC load takes effect after a one instructon delay, the result is that r1 = r0 = 1.

Table 4-3. Condition Flag Select Field Assignments

Value	Flag	Description
000	FALSE	Select for Unconditional Branch
001	CARRY	Carry Out from ALU Condition Flag Latch
010	OVF	Overflow from ALU Condition Flag Latch
011	SIGN	Sign from ALU Condition Flag Latch
100	ZERO	Zero from ALU Condition Flag Latch
101	LCNTZ	TRUE if Selected Loop Counter = 0
110	LSB	LSB of Data Register r0
111	MSB	MSB of Data Register r0

NOTE:
The ALU condition flags (CARRY, OVF, SIGN, and ZERO) are latched in the ALU Condition Flag register. This register is updated for most—but not all—ALU operations. The remaining flags (LCNTZ, LSB, and MSB) are updated and latched each cycle.

ASRC—A BUS SOURCE SELECT FIELD

This field selects the element that should drive its data onto the A bus during the execution of this instruction. The mapping for this and the following three fields is provided in Chapter 6.

ADST—A BUS DESTINATION SELECT FIELD

This field selects which element should latch data from the A bus during the execution of this instruction. See ASRC above.

BSRC—B BUS SOURCE SELECT FIELD

Same as ASRC, but for B bus. See ASRC above.

BDST—B BUS DESTINATION SELECT FIELD

Same as ADST, but for B bus. See ADST above.

CNT—DECREMENT LOOP COUNTER BIT

A one in this bit position causes the selected Loop Counter (selected by LC, the loop counter select bit) to be decremented. The new value of the loop counter and the updated LCNTZ condition flag are not ready until the next instruction cycle. Therefore, in a loop where the loop counter is decremented and tested for zero in the same instruction (typically in a one instruction loop), the start value for the loop counter should be one less than the number of times the loop should be executed.

LIT—LITERAL SELECT BIT

When this bit is a one, the ASRC and CFSEL fields are replaced with a 9-bit literal value that is driven as a source in the least significant 9 bits of the A bus. In this case, the upper 7 bits of the A bus are forced to zeros. The mapping of bits from the literal field to the A bus is shown in Figure 4-1.

NOTE

A conditional branch and a literal on the A bus are not allowed in the same instruction. A 3-bit literal can be placed on the B bus in any instruction.

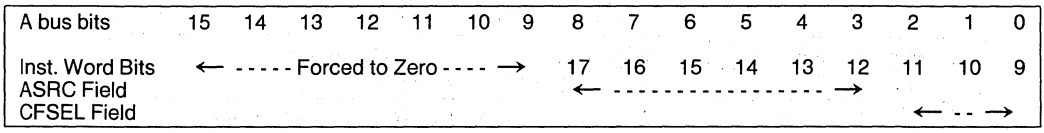


Figure 4-1. Literal Field Mapping onto a Bus

SHFT—SHIFT CONTROL FIELD

This field controls the bit shifting and byte swapping logic associated with register *r0*. The encoding of this field is given in Table 4-4.

Table 4-4. SHIFT Control Field Coding

SHFT	Operation
00	No Shift or Swap Operation
01	Shift r0 Right One Bit Position, Sign Extend
10	Shift r0 Left One Bit Position, Zero Fill
11	Byte Swap the Value Being Loaded into r0*

*Byte swapping only works when r0 is the destination on the A bus or the B bus. It does not swap data held in r0, only data being loaded. In order to byte swap data in register r0, r0 must be both a source and destination for either the A or B bus.

ALUSS—ALU SOURCE SELECT BITS

These two bits are used as enables for the two ALU input latches. Bit 39 enables the latch that connects to the A bus; bit 38 enables the latch connected to the B bus. A one in either bit position causes the corresponding input latch to latch the value on the bus to which it is connected (the A or B bus). A zero on either bit causes the corresponding latch to hold its current content. This allows the ALU operands either to come from "eavesdropping" on the A or B bus transfers occurring in the current instruction cycle or to be held for multiple instruction cycles in either the A or B input latch.

ALUOP—ALU OPERATION CODE FIELD

This field specifies the ALU instruction to be performed during the current instruction cycle. The encoding of this field is given in Figure 4-2. Normally, at the end of the instruction execution, the result of the ALU operation is latched in the ALU output latch that can be a source on either the A or B buses. However, if a NOP is selected for the ALU operation, the ALU output latch is not latched. The data is held from the previous instruction. In addition to NOP, certain other ALU opcodes do not actually perform ALU operations and therefore, do not latch the ALU results. They are INT (microcode interrupt) and the PROF instruction.

LC—LOOP COUNTER SELECT BIT

This bit selects which of the two loop counters is to be used for decrementing or Loop-Counter-Zero conditional branching in the current instruction. A zero selects loop counter zero and a one selects loop counter one.

Refer to the Intel *82750PB Microcode Programming Guide* for more information on microcode programming.

Table 4-5. 82750PB Source/Destination Coding

Address (Hex)	BDST	BSRC	ADST	ASRC
0x0	Null	Null	Null	Null
0x1		alu		hwid
0x2	*dram3	*dram3		cc
0x3	*dram4	*dram4	maddr	
0x4	*dram3++	*dram3++		alu
0x5	*dram4++	*dram4++	cnt	cnt
0x6	*dram3--	*dram3--	cnt2	cnt2
0x7	*dram4--	*dram4--	lcnt	lcnt
0x8	r0	r0	r0	r0
0x9	r1	r1	r1	r1
0xA	r2	r2	r2	r2
0xB	r3	r3	r3	r3
0xC	r4	r4	r4	r4
0xD	r5	r5	r5	r5
0xE	r6	r6	r6	r6
0xF	r7	r7	r7	r7
0x10	r8	*in1	mcode3	mcode3
0x11	r9	*in2	mcode2	mcode2
0x12	r10	*stat	mcode1	mcode1
0x13	r11	*stat#	pc	pc
0x14	r12	circbuf	pixint-c	
0x15	r13		pixint	pixint
0x16	r14		*dram1	*dram1
0x17	r15		*dram2	*dram2
0x18	circbuf	literal 0	*dram1++	*dram1++
0x19		literal 1	*dram2++	*dram2++
0x1A	*dram1	literal 2	*dram1--	*dram1--
0x1B	*dram2	literal 3	*dram2--	*dram2--
0x1C	*dram1++	literal 4	dram1	dram1
0x1D	*dram2++	literal 5	dram2	dram2
0x1E	*dram1--	literal 6	dram3	dram3
0x1F	*dram2--	literal 7	dram4	dram4
0x20	*out1	prof	*out1	*in1

Table 4-5. 82750PB Source/Destination Coding (Continued)

Address (Hex)	BDST	BSRC	ADST	ASRC
0x21	out1 + +		out1 + +	*in2
0x22	out1-lo	out1-lo	shift-rl	*stat
0x23	out1-hi	out1-hi	out1-hi	*stat#
0x24	*out2	stat-lo	*out2	
0x25	out2 + +	stat-hi	out2 + +	
0x26	out2-lo	out2-lo	shift-r	
0x27	out2-hi	out2-hi	out2-hi	
0x28	out1-c	out1-c	out1-c	
0x29	in1-c	in1-c	in1-c	
0x2A	in1-lo	in1-lo	shift-l	
0x2B	in1-hi	in1-hi	in1-hi	
0x2C	out2-c	out2-c	out2-c	
0x2D	in2-c	in2-c	in2-c	
0x2E	in2-lo	in2-lo		
0x2F	in2-hi	in2-hi	in2-hi	
0x30	stat-ram	r8	r8	r8
0x31	stat-c	r9	r9	r9
0x32	stat-lo	r10	r10	r10
0x33	stat-hi	r11	r11	r11
0x34	yeven-lo	r12	r12	r12
0x35	yeven-hi	r13	r13	r13
0x36	yodd-lo	r14	r14	r14
0x37	yodd-hi	r15	r15	r15
0x38	ypitch	shift	cc	shift
0x39		stat-c	fcnt	fcnt
0x3A	vu-lo	*dram1	*dram3	*dram3
0x3B	vu-hi	*dram2	*dram4	*dram4
0x3C	vupitch	*dram1 + +	*dram3 + +	*dram3 + +
0x3D	vpitch	*dram2 + +	*dram4 + +	*dram4 + +
0x3E	vptr-lo	*dram1 − −	*dram3 − −	*dram3 − −
0x3F	vptr-hi	*dram2 − −	*dram4 − −	*dram4 − −

	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24
	mcode 1																mcode2							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8
bit coding	LC SEL	SHFT CNTL		ALU OPCODE					ALU SS		LIT	CNT	B Bus Destination				B Bus Source							
	1	2		5					2		1	1	6				6							
0x0	cnt	nop		NOP					hold		nop	nop	null				null							
0x1	cnt2	shft r		ZERO					lat b		lit	dec					alu							
0x2		shft l		a					lat a				*dram3				*dram3							
0x3		swap		b					both				*dram4				*dram4							
0x4				~a									*dram3 + +				*dram3 + +							
0x5				~b									*dram4 + +				*dram4 + +							
0x6				&									*dram3 − −				*dram3 − −							
0x7				~&									*dram4 − −				*dram4 − −							
0x8				&~									r0				r0							
0x9				+ +									r1				r1							
0xA				|								r2				r2							
0xB				~ |								r3				r3							
0xC				| ~								r4				r4							
0xD				− <									r5				r5							
0xE				−									r6				r6							
0xF				− + <									r7				r7							
0x10				+									r8				*in1							
0x11				−									r9				*in2							
0x12				− +									r10				*stat							
0x13				− a									r11				*stat#							
0x14				− b									r12				circbuf							
0x15				a + +									r13											
0x16				b + +									r14											
0x17				a − −									r15											
0x18				b − −									circbuf				literal 0							
0x19				int													literal 1							
0x1A				prof									*dram1				literal 2							
0x1B				a*									*dram2				literal 3							
0x1C				b*									*dram1 + +				literal 4							
0x1D				+ <									*dram2 + +				literal 5							
0x1E				+]									*dram1 − −				literal 6							
0x1F				−]									*dram2 − −				literal 7							
0x20													*out1				prof							
0x21													out1 + +											
0x22													out1 − lo				out1 − lo							
0x23													out1 − hi				out1 − hi							
0x24													*out2				stat-lo							
0x25													out2 + +				stat-hi							
0x26													out2 − lo				out2 − lo							
0x27													out2 − hi				out2 − hi							
0x28													out1 − c				out1 − c							
0x29													in1 − c				in1 − c							
0x2A													in1 − lo				in1 − lo							
0x2B													in1 − hi				in1 − hi							
0x2C													out2 − c				out2 − c							
0x2D													in2 − c				in2 − c							
0x2E													in2 − lo				in2 − lo							
0x2F													in2 − hi				in2 − hi							
0x30													stat − ram				r8							
0x31													stat − c				r9							
0x32													stat − lo				r10							
0x33													stat − hi				r11							
0x34													yeven − lo				r12							
0x35													yeven − hi				r13							
0x36													yodd − lo				r14							
0x37													yodd − hi				r15							
0x38													ypitch				shift							
0x39																	stat − c							
0x3A													vu − lo				*dram1							
0x3B													vu − hi				*dram2							
0x3C													vupitch				*dram1 + +							
0x3D													vpitch				*dram2 + +							
0x3E													vptr − lo				*dram1 − −							
0x3F													vptr − hi				*dram2 − −							

Figure 4-2. 82750PB Instruction Word Format

	23 22 21 20 19 18 17 16	15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
	mcode 2	mcode 3		
	7 6 5 4 3 2 1 0	15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
bit coding	A Bus Destination	A Bus Source	Cond Flag Select	Next Address
	6	6	3	9
0x0	null	null	FALSE	
0x1		hwid	CARRY	
0x2		cc	OVERFLOW	
0x3	moddr		SIGN	
0x4		alu	ZERO	
0x5	cnt	cnt	CNT0	
0x6	cnt2	cnt2	LSB r0	
0x7	lcnt	lcnt	MSB r0	
0x8	r0	r0		
0x9	r1	r1		
0xA	r2	r2		
0xB	r3	r3		
0xC	r4	r4		
0xD	r5	r5		
0xE	r6	r6		
0xF	r7	r7		
0x10	mcode3	mcode3		
0x11	mcode2	mcode2		
0x12	mcode1	mcode1		
0x13	pc	pc		
0x14	pixint – c			
0x15	pixint	pixint		
0x16	*dram1	*dram1		
0x17	*dram2	*dram2		
0x18	*dram1 + +	*dram1 + +		
0x19	*dram2 + +	*dram2 + +		
0x1A	*dram1 – –	*dram1 – –		
0x1B	*dram2 – –	*dram2 – –		
0x1C	dram1	dram1		
0x1D	dram2	dram2		
0x1E	dram3	dram3		
0x1F	dram4	dram4		
0x20	*out1	*in1		
0x21	out1 + +	*in2		
0x22	shift – rl	*stat		
0x23	out1 – hi	*stat#		
0x24	*out2			
0x25	out2 + +			
0x26	shift – r			
0x27	out2 – hi			
0x28	out1 – c			
0x29	in1 – c			
0x2A	shift – l			
0x2B	in1 – hi			
0x2C	out2 – c			
0x2D	in2 – c			
0x2E				
0x2F	in2 – hi			
0x30	r8	r8		
0x31	r9	r9		
0x32	r10	r10		
0x33	r11	r11		
0x34	r12	r12		
0x35	r13	r13		
0x36	r14	r14		
0x37	r15	r15		
0x38	cc	shift		
0x39	fcnt	fcnt		
0x3A	*dram3	*dram3		
0x3B	*dram4	*dram4		
0x3C	*dram3 + +	*dram3 + +		
0x3D	*dram4 + +	*dram4 + +		
0x3E	*dram3 – –	*dram3 – –		
0x3F	*dram4 – –	*dram4 – –		

Figure 4-2. 82750PB Instruction Word Format (Continued)

5.0 ELECTRICAL DATA

Maximum Ratings

Table 5-1 is a stress rating only, and functional operation at the maximums is not guaranteed. Functional operating conditions are given in the DC and AC Characteristics (Tables 5-2, 5-3, 5-4, and 5-5).

Exposure to Maximum Ratings may affect device reliability. Furthermore, although the 82750PB contains protective circuitry to resist damage from static electrical discharge, always take precautions to avoid high static voltages or electric fields.

DC Characteristics

Table 5-1. Absolute Maximum Requirements

Condition	Maximum Requirement
Case Temparature under Bias	−65°C to 110°C
Storage Temperature	−65°C to 150°C
Voltage on Any Pin with Respect to Ground	−0.5V to V_{CC} + 0.5V
Supply Voltage with Respect to V_{SS}	−0.5V to + 6.5V

Table 5-2. DC Characteristics $V_{CC} = 5V \pm 10\%$, T_{CASE} = 0°C to 90°C

Symbol	Parameter	Min	Typ	Max	Unit	Notes
V_{IL}	Input LOW Voltage	−0.3		0.8	V	(Note 1)
V_{IH}	Input HIGH Voltage	2.0		V_{CC} + 0.3	V	(Note 1)
V_{OL}	Output LOW Voltage		0.2	0.4	V	I_{OL} = 4.0 mA(1)
V_{OH}	Output HIGH Voltage	2.4	3.0		V	I_{OH} = −1.0 mA(1)
I_{IL}	Input Leakage Current	−10		+10	µA	$V_{SS}<V_{IN}<V_{CC}$
I_{OZ}	Output Leakage Current	−10		+10	µA	$V_{SS}<V_{IN}<V_{CC}$
I_{CC}	Power Supply Current		150	200	mA	25 MHz(2)
C_{IN}	Input Capacitance			10.0	pF	F_C = 1 MHz(3)
C_{OUT}	Output Capacitance			12.0	pF	F_C = 1 MHz(3)
C_{CLKIN}	CLKIN Input Capacitance			20.0	pF	F_C = 1 MHz(3)

NOTES:
1. Measured with CLKIN = 8 MHz.
2. Typical current value measured under typical conditions. Maximum current value guaranteed with 50 pF maximum output loading.
3. Not 100% tested.

AC Characteristics

Table 5-3. AC Characteristics at 25 MHz V_{CC} = 5V ±10%, T_{CASE} = 0°C to +90°C, C_L = 50 pF

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	8	25	MHz		1xClock
t_1	CLKIN Period	40	125	ns	5-1	
t_2	CLKIN High Time	14	26	ns	5-1	(Note 1)
t_3	CLKIN Low Time	14	26	ns	5-1	(Note 1)
t_4	CLKIN Fall Time		4	ns	5-1	
t_5	CLKIN Rise Time		4	ns	5-1	
t_{6a}	A[31:2], BE # [3:0], WE #, D[31:0], HINT #, PMFRZ # Valid Delay	3	25	ns	5-2	
t_{6b}	MREQ #, TRNFR #, RFSH #, NXTFST #, HBUSEN #, HRDY #, Valid Delay	3	18	ns	5-2	
t_7	A[31:2], BE # {3:0], WE#, D[31:0] Float Delay		30	ns	5-2	(Note 2)
t_8	MRDY # Setup	10		ns	5-3	
t_9	MRDY # Hold	6		ns	5-3	
t_{10}	HREQ #, VBUS[3:0], RESET #, HALEN #, HALT # Setup	8		ns	5-3	
t_{11}	HREQ #, VBUS[3:0], RESET #, HALEN #, HALT # Hold	6		ns	5-3	
t_{12}	A[8:2], BE # [3:0], WE #, D[31:0] Setup	4		ns	5-3	(Note 3)
t_{13}	A[8:2], BE # [3:0], WE #, D[31:0] Hold	6		ns	5-3	(Note 3)
t_{14}	HREG #, HRAM # Setup	10		ns	5-3	
t_{15}	HREG #, HRAM # Hold	6		ns	5-3	
t_{16}	CLKOUT Valid Delay		18	ns	5-4	
t_{17}	CLKOUT High Time	$1/2t_1 - 6$	$1/2t_1 + 6$	ns	5-4	

NOTES:
1. This assumes 40 ns period. For other speeds these values should fall between 40% to 60% duty cycle.
2. Not 100% tested. Guaranteed by design characterization.
3. Inputs must remain valid throughout all cycles of host accesses. See Figures 3-6 through 3-8.
4. All A.C. specifications are measured at the 1.5V crossing point with a 50 pF load.

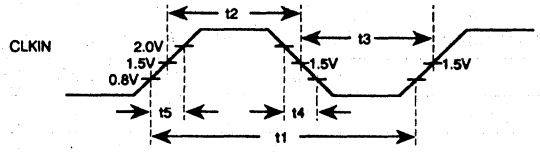


Figure 5-1. Clock Waveforms

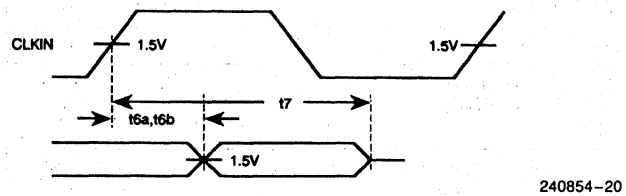
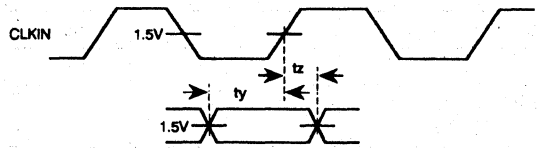


Figure 5-2. Output Waveforms



NOTES:
ty = t8, t10, t12, t14 (setup times)
tz = t9, t11, t13, t15 (hold times)

Figure 5-3. Input Waveforms

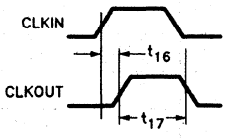
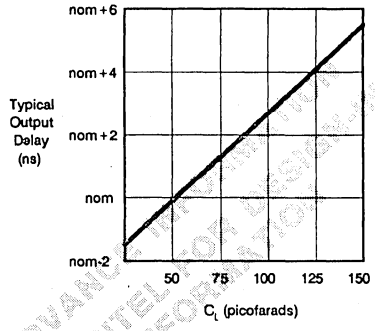


Figure 5-4. CLKOUT Waveforms

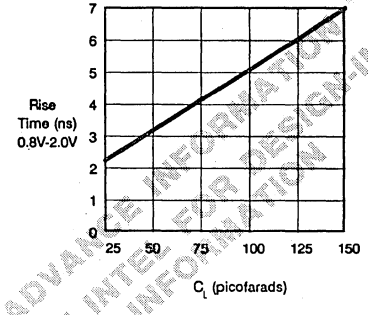
Output Delay and Rise Time Versus Load Capacitance



240854-22

NOTE:
This graph will not be linear outside of the C_L range shown.
nom = nominal value given in A.C. Characteristics table.

Figure 5-5. Typical Output Valid Delay Versus Load Capacitance under Worst Case Conditions



240854-23

NOTE:
This graph will not be linear outside of the C_L range shown.

Figure 5-6. Typical Output Rise Time Versus Load Capacitance under Worst Case Conditions

6.0 MECHANICAL DATA

Packaging Outlines and Dimensions

Intel packages the 82750PB in a Plastic Quad Flat Pack (PQFP). Table 6-1 gives the symbol list for the PQFP.

Table 6-1. PQFP Symbol List

Letter or Symbol	Description of Dimensions
A	Package Height: Distance from Seating Plane to Highest Point of Body
A_1	Standoff: Distance from Seating Plane to Base Plane
D/E	Overall Package Dimension: Lead Tip to Lead Tip
D_1/E_1	Plastic Body Dimension
D_2/E_2	Bumper Distance
D_3/E_3	Footprint
L_1	Foot Length
N	Total Number of Leads

The PQFP has the following specifications:

1. All dimensions and tolerances conform to ANSI Y14.5M-1982.
2. Datum plane —H— is located at the mold parting line and coincident with the bottom of the lead where lead exits plastic body.
3. Datums A-B and —D— are to be determined where center leads exit plastic body at datum plane —H—.
4. Controlling dimension is the inch.
5. Dimensions D_1, D_2, E_1, and E_2 are measured at the mold parting line and do not include mold protrusion. Allowable mold protrusion is 0.18 mm (0.007 in.) per side.
6. Pin 1 identifier is located within one of the two zones indicated.
7. Measured at datum plane —H—.
8. Measured at seating plane datum —C—.

Table 6-2 provides outline characteristics for 0.025 in. pitch.

Table 6-2. Intel Case Outline Drawings for PQFP at 0.025 inch Pitch

Symbol	Description	Min	Max
N	Leadcount	132	132
A	Package Height	0.160	0.170
A_1	Standoff	0.020	0.030
D, E	Terminal Dimiension	1.075	1.085
D_1, E_1	Package Body	0.0947	0.953
D_2, E_2	Bumper Distance	1.097	1.103
D_3, E_3	Lead Dimension	0.800 REF	0.800 REF
L_1	Foot Length	0.020	0.030

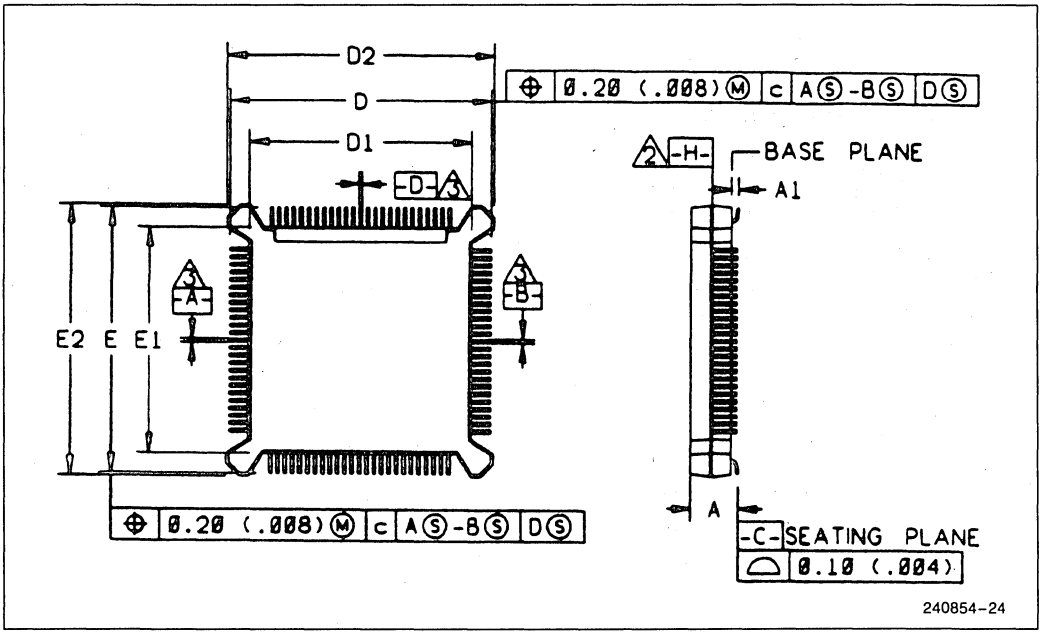


Figure 6-1. Principal Dimensions of the 82750PB in the 132-Lead PQFP Package

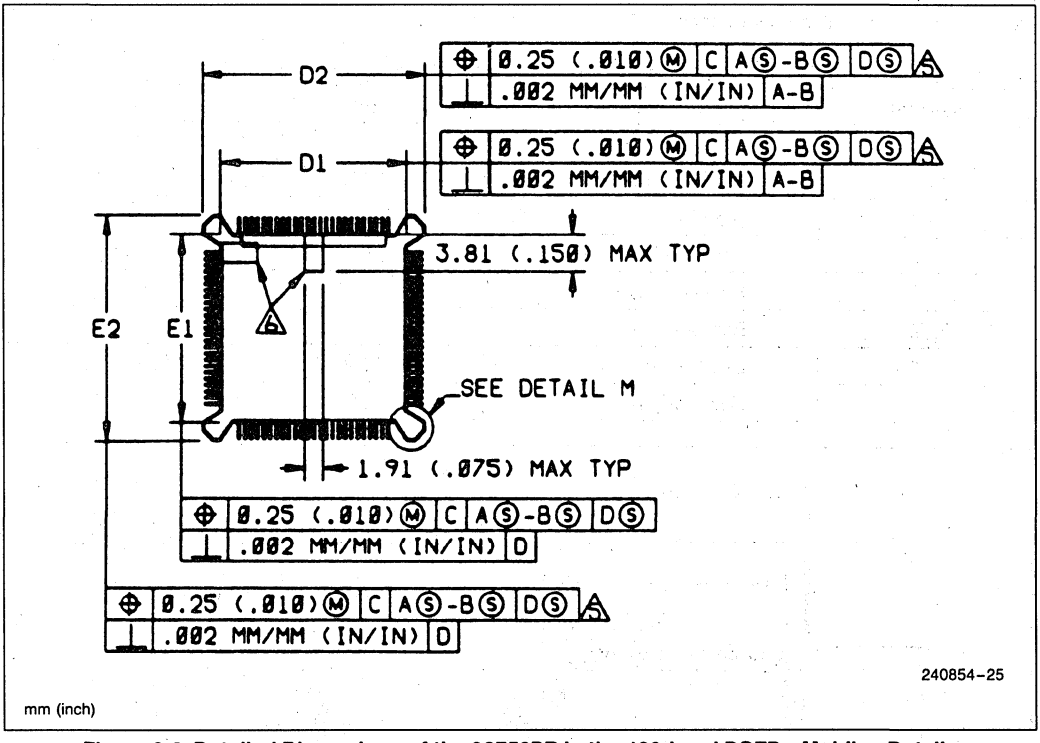


Figure 6-2. Detailed Dimensions of the 82750PB in the 132-Lead PQFP—Molding Details

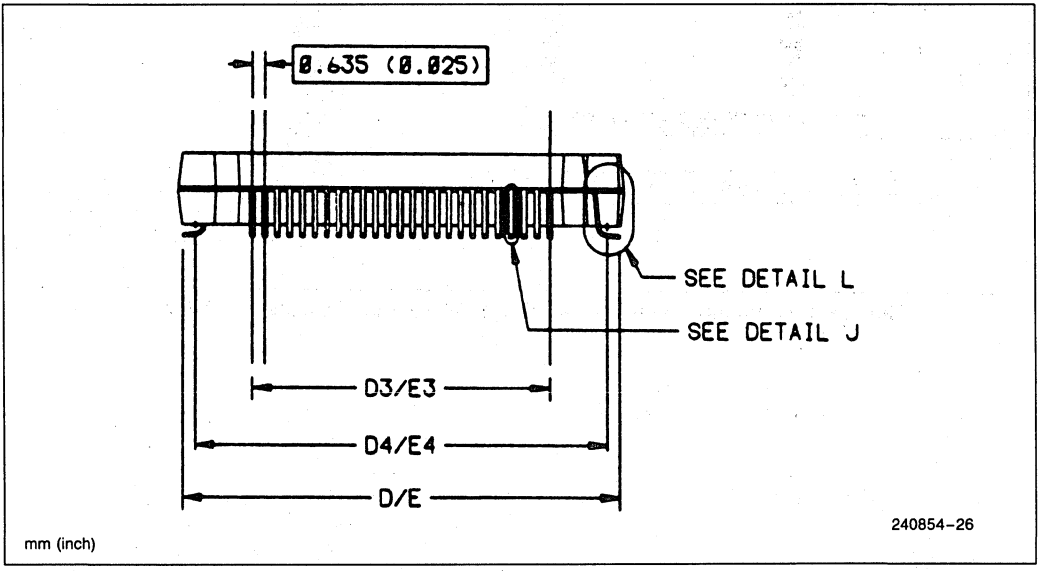
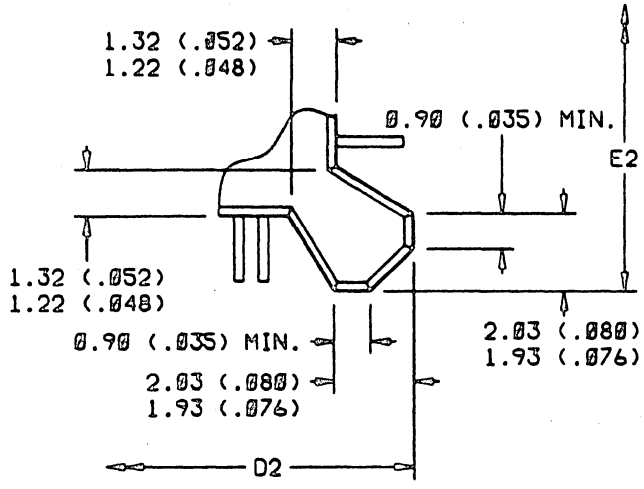
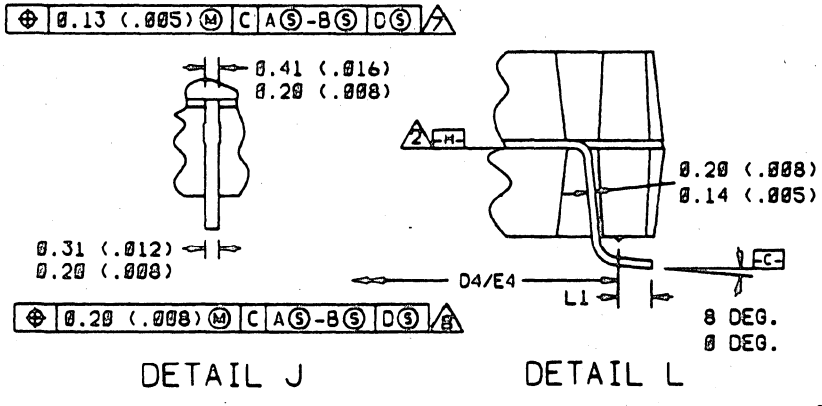


Figure 6-3. Detailed Dimensions of the 82750PB in the 132-Lead PQFP—Terminal Details



mm (inch)

Figure 6-4. 132-Lead PQFP Mechanical Package Detail—Protective Bumper



mm (inch)

Figure 6-5. 132-Lead PQFP Mechanical Package Detail—Typical Lead

NOTES:

1. ALL DIMENSIONS AND TOLERANCES CONFORM TO ANSI Y14.5M-1982
2. DATUM PLANE -H- LOCATED AT THE MOLD PARTING LINE AND COINCIDENT WITH THE BOTTOM OF THE LEAD WHERE LEAD EXITS PLASTIC BODY
3. DATUMS A-B AND -D- TO BE DETERMINED WHERE CENTER LEADS EXIT PLASTIC BODY AT DATUM PLANE -H-
4. CONTROLLING DIMENSION, INCH
5. DIMENSIONS D1, D2, E1 AND E2 ARE MEASURED AT THE MOLD PARTING LINE. D1 AND E1 DO NOT INCLUDE AN ALLOWABLE MOLD PROTRUSION OF 0.18 MM (.007 IN) PER SIDE. D2 AND E2 DO NOT INCLUDE A TOTAL ALLOWABLE MOLD PROTRUSION OF 0.18 MM (.007 IN) AT MAXIMUM PACKAGE SIZE.
6. PIN 1 IDENTIFIER IS LOCATED WITHIN ONE OF THE TWO ZONES INDICATED
7. MEASURED AT DATUM PLANE -H-
8. MEASURED AT SEATING PLANE DATUM -C-

240854-29

Package Thermal Specifications

The 82750PB is specified for operation when T_C (the case temperature) is within the range of 0°C to 90°C. T_C may be measured in any environment to determine whether the 82750PB is within specified operation range. The case temperature should be measured at the center of the top surface.

T_A (the ambient temperature) can be calculated from θ_{CA} (thermal resistance from case to ambient) with the following equation:

$$T_A = T_C - P * \theta_{CA}$$

Typical values for θ_{CA} at various airflows are given in Table 6-3 for the 132-lead PQFP package. Table 6-4 shows the maximum T_A allowable (wihout exceeding T_C) at various airflows. The power dissipation (P) is calculated by using the typical supply current at 5V as shown in Table 5-2.

Table 6-3. Thermal Resistance (°C/W)

	θ_{CA} Versus Airflow—ft/min (m/sec)					
Package	**0 (0)**	**200 (1.01)**	**400 (2.03)**	**600 (3.04)**	**800 (4.06)**	**1000 (5.07)**
132-Lead PQFP	26.0	17.5	14.0	11.5	9.5	8.5

Table 6-4. Maximum T_A at Various Airflows (°C)

		T_A Versus Airflow—ft/min (m/sec)					
Package	**Frequency (MHz)**	**0 (0)**	**200 (1.01)**	**400 (2.03)**	**600 (3.04)**	**800 (4.06)**	**1000 (5.07)**
132-Lead PQFP	25	70	76	80	81	83	84

i860™ Microprocessor Family 2

i860™ XP MICROPROCESSOR

- **Parallel Architecture that Supports Up to Three Operations per Clock**
  - **One Integer or Control Instruction**
  - **Up to Two Floating-Point Results**
- **High Performance Design**
  - **40/50 MHz Clock Rate**
  - **100 Peak Single Precision MFLOPS**
  - **75 Peak Double Precision MFLOPS**
  - **64-Bit External Data Bus**
  - **64-Bit Internal Code Bus**
  - **128-Bit Internal Data Bus**
- **High Integration on One Chip**
  - **32-Bit Integer and Control Unit**
  - **32/64-Bit Pipelined Floating-Point**
  - **64-Bit 3-D Graphics Unit**
  - **Paging Unit with 64 Four-Kbyte and 16 Four-Mbyte Pages**
  - **16 Kbyte Code Cache**
  - **16 Kbyte Data Cache**
- **Fast, Multiprocessor-Oriented Bus**
  - **Burst Cycles Move 400 Mbyte/Sec**
  - **Hardware Cache Snooping**
  - **MESI Cache Consistency Protocol**
  - **Supports Second-Level Cache**
  - **Supports DRAM**
- **Compatible with Industry Standards**
  - **ANSI/IEEE Standard 754-1985 for Binary Floating-Point Arithmetic**
  - **Intel 386™/Intel 486™/i860™ Data Formats and Page Table Entries**
  - **Binary Compatible with i860™ XR Applications Instruction Set**
  - **Detached Concurrency Control Unit (CCU) Supports Parallel Architecture Extensions (PAX)**
  - **JEDEC 262-pin Ceramic Pin Grid Array Package**
  - **IEEE Standard 1149.1/D6 Boundary-Scan Architecture**
- **Easy to Use**
  - **On-Chip Debug Register**
  - **UNIX*/860**
  - **APX Attached Processor Executive**
  - **Assembler, Linker, Simulator, Debugger, C and FORTRAN Compilers, FORTRAN Vectorizer, Scalar and Vector Math Libraries**
  - **Graphics Libraries**

2

The Intel i860 XP Microprocessor (order code A80860XP) delivers supercomputing performance in a single VLSI component. The 32/64-bit architecture of the i860 XP microprocessor balances integer, floating point, and graphics performance for applications such as engineering workstations, scientific computing, 3-D graphics workstations, and multiuser systems. Its parallel architecture achieves high throughput with RISC design techniques, multiprocessor support, pipelined processing units, wide data paths, large on-chip caches, 2.5 million transistor design, and fast 0.8-micron silicon technology.

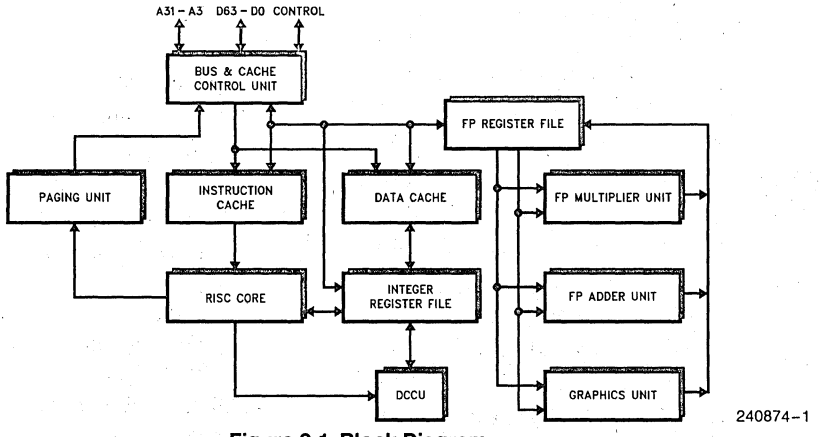


Figure 0.1. Block Diagram

*UNIX is a registered trademark of UNIX System Laboratories, Inc.
Intel, i860, Intel386 and Intel486 are trademarks of Intel Corporation.

November 1991
Order Number: 240874-002

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

PAGE

CONTENTS

PAGE

CONTENTS

PAGE

FIGURES

CONTENTS

PAGE

CONTENTS

PAGE

CONTENTS

PAGE

TABLES

CONTENTS

PAGE

1.0 FUNCTIONAL DESCRIPTION

As shown by the block diagram on the front page, the i860 XP Microprocessor consists of the following units:

1. Integer Registers and Core Execution Unit
2. Floating-Point Registers and Control Unit
3. Floating-Point Adder Unit
4. Floating-Point Multiplier Unit
5. Graphics Unit
6. Paging Unit
7. Instruction Cache
8. Data Cache
9. Bus and Cache Control Unit
10. Detached Concurrency Control Unit

The core execution unit controls overall operation of the i860 XP microprocessor. It executes load, store, integer, bit, I/O, and control-transfer operations, and fetches instructions for the floating-point unit as well. A set of 32 × 32-bit general-purpose registers are provided for the manipulation of integer data. Load and store instructions move 8-, 16-, and 32-bit data to and from these registers. Its full set of integer, logical, and control-transfer instructions give the core unit the ability to execute complete systems software and applications programs. A trap mechanism provides rapid response to exceptions and external interrupts. Debugging is supported by the ability to trap on data or instruction reference.

The floating-point hardware is connected to a separate set of floating-point registers, which can be accessed as 16 × 64-bit registers or as 32 × 32-bit registers. Load and store instructions can also access these same registers as 8 × 128-bit registers. All floating-point and graphics instructions use these registers as their source and destination operands.

The floating-point control unit controls both the floating-point adder and the floating-point multiplier, issuing instructions, handling all source and result exceptions, and updating status bits in the floating-point status register. The adder and multiplier can operate in parallel, producing up to two results per clock. The floating-point data types, floating-point instructions, and exception handling all support the IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985).

The floating-point adder performs addition, subtraction, comparison, and conversions on 64- and 32-bit floating-point values. An adder instruction executes in three clocks; however, in pipelined mode, a new result is generated every clock.

The floating-point multiplier performs floating-point and integer multiply as well as floating-point reciprocal operations on 64- and 32-bit floating-point values. A multiplier instruction executes in three to four clocks; however, in pipelined mode, a new result can be generated every clock for single-precision and every other clock for double precision.

The graphics unit supports three-dimensional drawing in a graphics frame buffer, with color intensity shading and hidden surface elimination via the Z-buffer algorithm. The graphics unit recognizes the pixel as an 8-, 16-, or 32-bit integer data type. It can compute individual red, blue, and green color intensity values within a pixel; but it does so with parallel operations that take advantage of the 64-bit internal word size and 64-bit external bus. The graphics features of the i860 XP microprocessor assume that the surface of a solid object is drawn with polygon patches which, like the pieces of a puzzle, collectively approximate the shape of the original object. The color intensities of the vertices of the polygon and their distances from the viewer are known, but the distances and intensities of the other points must be calculated by interpolation. The graphics instructions of the i860 XP microprocessor directly aid such interpolation.

The paging unit implements protected, paged, virtual memory. The paging unit uses two four-way set-associative cache memories called TLBs (Translation Lookaside Buffers) to perform the translation of logical address to physical address, and to check for access violations. The access protection scheme employs two levels of privilege: user and supervisor. One TLB supports 4 Kbyte pages, and has 64 entries; the other supports 4 Mbyte pages, and has 16 entries.

The instruction cache is a four-way set-associative memory of 16 Kbytes, with 32-byte lines. It transfers up to 64 bits per clock (400 Mbyte/sec at 50 MHz).

The data cache is a four-way set-associative memory of 16 Kbytes, with 32-byte lines. It transfers up to 128 bits per clock (800 Mbyte/sec at 50 MHz). The i860 XP microprocessor normally uses write-back caching, i.e. memory writes update the cache (if applicable) without necessarily updating memory immediately; however, under both software and hardware control, write-through and write-once policies can be implemented, or caching can be inhibited. The caches are transparent to applications software.

The bus and cache control unit performs data and instruction accesses for the core unit. It receives cycle requests and specifications from the core unit, performs the data-cache or instruction-cache miss processing, controls TLB translation, and provides

the interface to the external bus. Its pipelined structure supports up to three outstanding bus cycles. Its burst mode transfers data at up to 400 Mbyte/sec at 50 MHz. In multiprocessor systems, it maintains cache consistency by monitoring bus activity in parallel with other CPU functions.

The DCCU (detached concurrency control unit) is a compatible subset of the external CCU that expedites loop-level parallelism and synchronization in multiprocessor systems. The DCCU consists of registers and a counter that allow a single i860 XP microprocessor to run binary code compiled for a multiprocessor system adhering to the PAX parallel applications binary interface (ABI).

The i860 XP microprocessor may to be used with or without an external, secondary cache built from 82495XP and 82490XP cache components. An 82495XP and 82490XP cache provides up to 512 Kbytes of high-speed storage for data and instruction combined. In most cases, an 82495XP and 82490XP cache can provide data to the CPU with zero wait states. The larger size of an external cache can provide an increased hit rate when the size or number of data structures and programs exceeds the size of the internal caches. In multiprocessor systems, the external cache serves as local memory, and can reduce bus traffic. An external cache also hides the processor from rest of system, which is a double advantage:

1. The processor can be upgraded without affecting design of the memory and other subsystems.
2. Slower and less expensive memory and I/O subsystem designs can be employed without unduly lowering overall system performance.

Refer to the *82495XP Cache Controller/82490XP Cache RAM Data Sheet* (Intel Order #240956) for more information.

2.0 PROGRAMMING INTERFACE

The programmer-visible aspects of the architecture of the i860 XP microprocessor include data types, registers, instructions, and traps.

2.1 Data Types

The i860 XP microprocessor provides operations for integer and floating-point data. Integer operations are performed on 32-bit operands with some support also for 64-bit operands. Load and store instructions can reference 8-bit, 16-bit, 32-bit, 64-bit, and 128-bit operands. Floating-point operations are performed on IEEE-standard 32- and 64-bit formats. Graphics instructions operate on arrays of 8-, 16-, or 32-bit pixels.

2.1.1 INTEGER

An integer is a 32-bit signed value in standard two's complement form. A 32-bit integer can represent a value in the range $-2{,}147{,}483{,}648$ (-2^{31}) to $2{,}147{,}483{,}647$ ($+2^{31} - 1$). Arithmetic operations on 8- and 16-bit integers can be performed by sign-extending the 8- or 16-bit values to 32 bits, then using the 32-bit operations.

There are also add and subtract instructions that operate on 64-bit long integers.

Load and store instructions may also reference (in addition to the 32- and 64-bit formats previously mentioned) 8- and 16-bit items in memory. When an 8- or 16-bit item is loaded into a register, it is converted to an integer by sign-extending the value to 32 bits. When an 8- or 16-bit item is stored from a register, the corresponding number of low-order bits of the register are used.

2.1.2 ORDINAL

Arithmetic operations are available for 32-bit ordinals. An ordinal is an unsigned integer. An ordinal can represent values in the range 0 to $4{,}294{,}967{,}295$ ($+2^{32} - 1$).

Also, there are add and subtract instructions that operate on 64-bit ordinals.

2.1.3 SINGLE- AND DOUBLE-PRECISION REAL

Figure 2.1 shows the real number formats. A single-precision real (also called "single real") data type is a 32-bit binary floating-point number. Bit 31 is the sign bit; bits 30..23 are the exponent; and bits 22..0 are the fraction. In accordance with ANSI/IEEE standard 754, the value of a single-precision real is defined as follows:

1. If **e** = 0 and **f** ≠ 0 or **e** = 255 then generate a floating-point source-exception trap when encountered in a floating-point operation.
2. If $0 < \mathbf{e} \leq 255$, then the value is $(-1)^{s} \times 1.\mathbf{f} \times 2^{e-127}$.
3. If **e** = 0 and **f** = 0, then the value is signed zero.

A double-precision real (also called "double real") data type is a 64-bit binary floating-point number. Bit 63 is the sign bit; bits 62..52 are the exponent; and bits 51..0 are the fraction. In accordance with ANSI/IEEE standard 754, the value of a double-precision real is defined as follows:

1. If **e** = 0 and **f** ≠ 0 or **e** = 2047, then generate a floating-point source-exception trap when encountered in a floating-point operation.

2. If $0 < \mathbf{e} < 2047$, then the value is $(-1)^{s} \times 1.\mathbf{f} \times 2^{e-1023}$.
3. If $\mathbf{e} = 0$ and $\mathbf{f} = 0$, then the value is signed zero.

The special values infinity, NaN ("Not a Number"), indefinite, and denormal generate a trap when encountered. The trap handler implements IEEE-standard results.

A double real value occupies an even/odd pair of floating-point registers. Bits 31..0 are stored in the even-numbered floating-point register; bits 63..32 are stored in the next higher odd-numbered floating-point register.

2.1.4 PIXEL

A pixel may be 8-, 16-, or 32-bits long, depending on color and intensity resolution requirements. Regardless of the pixel size, the i860 XP microprocessor always operates on 64 bits of pixel data at a time. The pixel data type is used by two kinds of instructions:

- The selective pixel-store instruction that helps implement hidden surface elimination.
- The pixel add instruction that helps implement 3-D color intensity shading.

To perform color intensity shading efficiently in a variety of applications, the i860 XP microprocessor defines three pixel formats according to Table 2.1.

Figure 2.2 illustrates one way of assigning meaning to the fields of pixels. These assignments are for illustration purposes only. The i860 XP microprocessor defines only the field sizes, not the specific use of each field. Other ways of using the fields of pixels are possible.

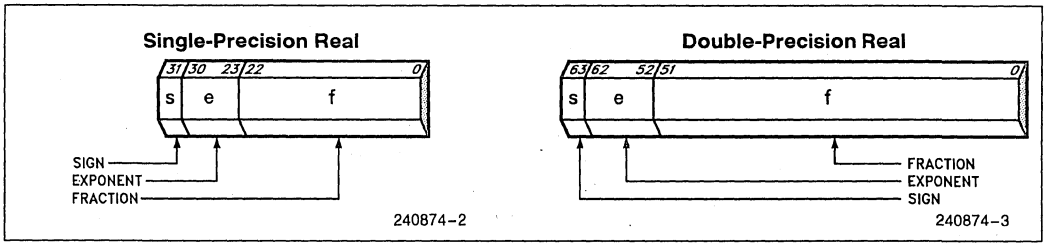
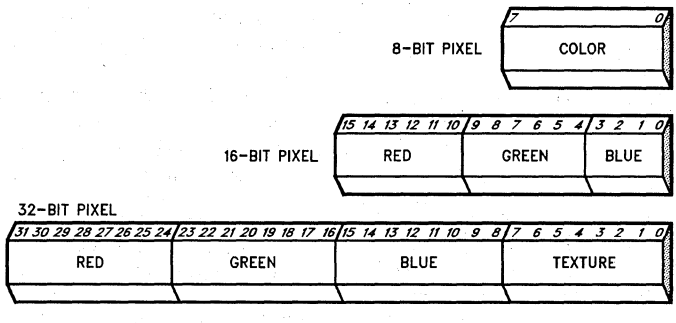


Figure 2.1. Real Number Formats

Table 2.1. Pixel Formats

Pixel Size (in bits)	Bits of Color 1 Intensity(1)	Bits of Color 2 Intensity(1)	Bits of Color 3 Intensity(1)	Bits of Other Attribute (Texture, Color)
8	N (≤8) bits of intensity(2)			8–N
16M	6	6	4	0
32	8	8	8	8

NOTES:
1. The intensity attribute fields may be assigned to colors in any order convenient to the application.
2. With 8-bit pixels, up to 8 bits can be used for intensity; the remaining bits can be used for any other attribute, such as color or texture. Bits that require interpolation (shading), such as those for intensity, must be the low-order bits of the pixel.



NOTE:
These assignments of specific meanings to the fields of pixels are for illustration only. Only the field sizes are defined, not the specific use of each field.

Figure 2.2. Pixel Format Example

2.2 Register Set

As Figure 2.3 shows, the i860 XP microprocessor has the following registers:

- An integer register file
- A floating-point register file
- Control registers **psr**, **epsr**, **db**, **dirbase**, **fir**, **fsr**, **bear**, **ccr**, **p3**, **p2**, **p1**, **p0**
- Special-purpose registers KR, KI, T, MERGE, STAT, and NEWCURR

The control registers are accessible only by load and store control-register instructions; the integer and floating-point registers are accessed by arithmetic operations and load and store instructions. The special-purpose registers KR, KI, and T are used by floating-point instructions; MERGE is used by graphics instructions. NEWCURR and STAT are used for concurrency control; they are accessed by memory load and store instructions.

2.2.1 INTEGER REGISTER FILE

There are 32 integer registers, each 32 bits wide, referred to as **r0** through **r31**, which are used for address computation and scalar integer computations. Register **r0** always returns zero when read.

2.2.2 FLOATING-POINT REGISTER FILE

There are 32 floating-point registers, each 32-bits wide, referred to as **f0** through **f31**, which are used for floating-point computations. Registers **f0** and **f1** always return zero when read. The floating-point registers are also used by a set of integer operations, primarily for graphics computations.

When accessing 64-bit floating-point or integer values, the i860 XP microprocessor uses an even/odd pair of registers. When accessing 128-bit values, it uses an aligned set of four registers (**f0, f4, f8, f12, f16, f20, f24,** or **f28**). The instruction must designate the lowest register number of the set of registers containing 64- or 128-bit values. Misaligned register numbers produce undefined results. The register with the lowest number contains the least significant part of the value. For 128-bit values, the register pair with the lower number contains the value from the lower memory address; the register pair with the higher number contains the value from the higher address.

The 128-bit load and store instructions, along with the 128-bit data path between the floating-point registers and the data cache, help to sustain an extraordinarily high rate of computation.

2.2.3 PROCESSOR STATUS REGISTER

The processor status register (**psr**) contains miscellaneous state information for the current process. Figure 2.4 shows the format of the **psr**.

- BR (Break Read) and BW (Break Write) enable a data access trap when the operand address matches the address in the **db** register and a read or write (respectively) occurs.
- Various instructions set CC (Condition Code) according to tests they perform. The branch-on-condition-code instructions test its value. The **bla** instruction sets and tests LCC (Loop Condition Code).
- IM (Interrupt Mode), if set, enables external interrupts on the INT pin; disables interrupts on INT if clear. IM does not affect parity error interrupts or interrupts on the BERR pin.

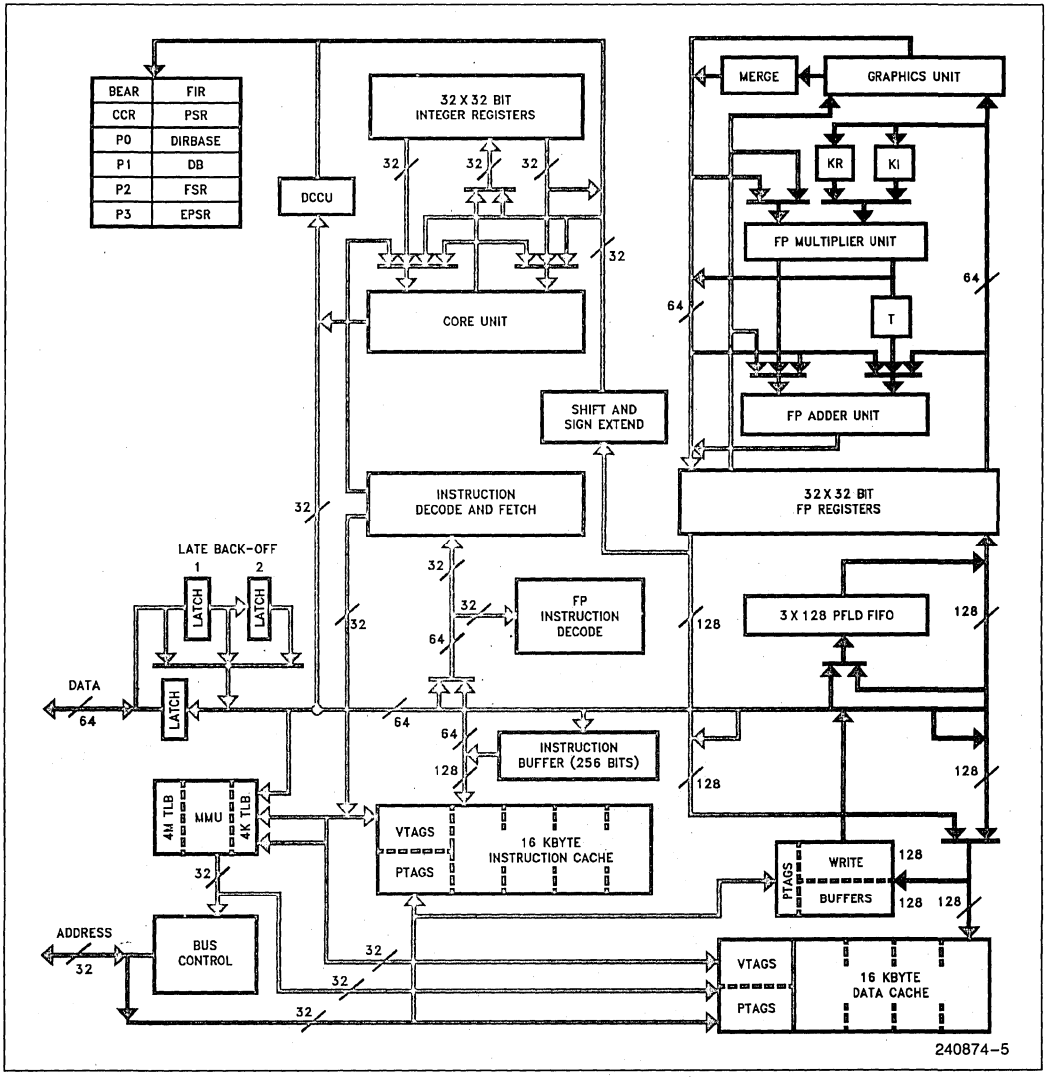


Figure 2.3. Registers and Data Paths

- U (User Mode) is set when the i860 XP microprocessor is executing in user mode; it is clear when the i860 XP microprocessor is executing in supervisor mode. In user mode, writes to some control registers are inhibited. This bit also controls the memory protection mechanism.
- PIM (Previous Interrupt Mode) and PU (Previous User Mode) save the corresponding status bits (IM and U) on a trap, because those status bits are changed when a trap occurs. They are restored into their corresponding status bits when returning from a trap handler with a branch indirect instruction when a trap flag is set in the **psr**.
- FT (Floating-Point Trap), DAT (Data Access Trap), IAT (Instruction Access Trap), IN (Interrupt), and IT (Instruction Trap) are trap flags. They are set when the corresponding trap condition occurs. IN is set on INT, bus error and parity error. The trap handler examines these bits (and other trap bits in the **epsr**) to determine which condition or conditions have caused the trap.
- DS (Delayed Switch) is set if a trap occurs during the instruction before dual-instruction mode is entered or exited. If DS is set and DIM (Dual Instruction Mode) is clear, the i860 XP microprocessor switches to dual-instruction mode one instruction

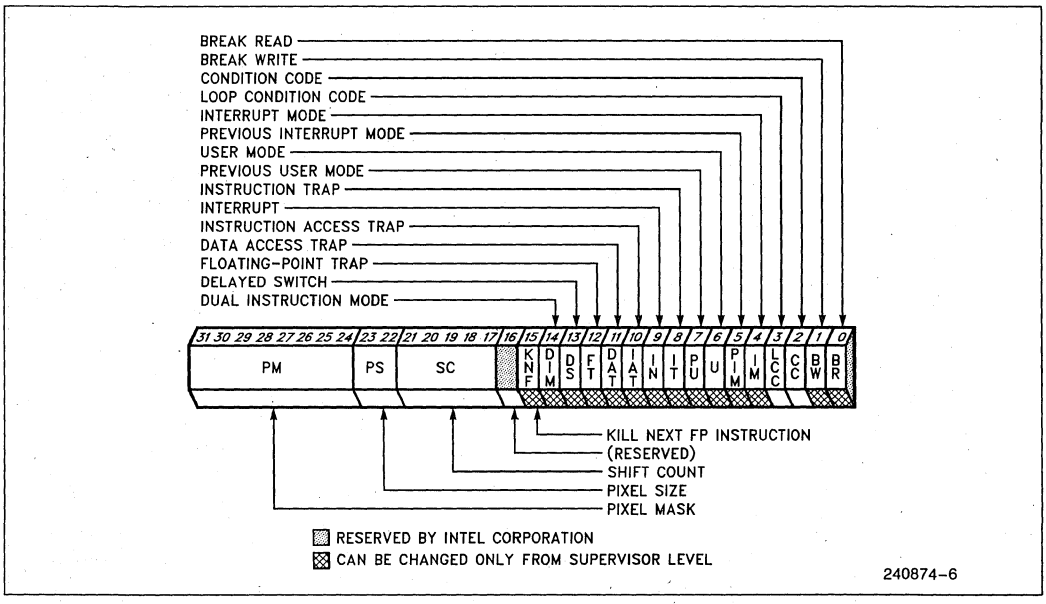


Figure 2.4. Processor Status Register

after returning from the trap handler. If DS and DIM are both set, the i860 XP microprocessor switches to single-instruction mode one instruction after returning from the trap handler.

- When a trap occurs, the i860 XP microprocessor sets DIM if it is executing in dual-instruction mode; it clears DIM if it is executing in single-instruction mode. If DIM is set after returning from a trap handler, the i860 XP microprocessor resumes execution in dual-instruction mode.
- When KNF (Kill Next Floating-Point Instruction) is set, the next floating-point instruction is suppressed (except that its dual-instruction mode bit is interpreted). A trap handler sets KNF if the trapped floating-point instruction should not be reexecuted.
- SC (Shift Count) stores the shift count used by the last right-shift instruction. It controls the number of shifts executed by the double-shift instruction.
- PS (Pixel Size) and PM (Pixel Mask) are used by the pixel-store and other graphics instructions. The values of PS control pixel size as defined by Table 2.2. The bits in PM correspond to pixels to be updated by the pixel-store instruction **pst.d**. The low-order bit of PM corresponds to the low-order pixel of the 64-bit source operand of **pst.d**. The number of low-order bits of PM that are actually used is the number of pixels that fit into 64-bits, which depends upon PS. If a bit of PM is set, then **pst.d** stores the corresponding pixel. Refer also to the **pst.d** instruction in section 10.

Table 2.2. Values of PS

Value	Pixel Size in Bits	Pixel Size in Bytes
00	8	1
01	16	2
10	32	4
11	(undefined)	(undefined)

2.2.4 EXTENDED PROCESSOR STATUS REGISTER

The extended processor status register (**epsr**) contains additional state information for the current process beyond that stored in the **psr**. Figure 2.5 shows the format of the **epsr**.

- The processor type is 2 for the i860 XP microprocessor.
- The stepping number has a unique value that distinguishes among different revisions of the processor.
- IL (Interlock) is set if a trap occurs after a **lock** instruction but before the last BRDY# of the load or store following the subsequent **unlock** instruction. IL indicates to the trap handler that a locked sequence has been interrupted. When the trap handler finds IL set, it should scan backwards for the **lock** instruction and restart at that point. The absence of a **lock** instruction within 30–33 instructions of the trap indicates a programming error.

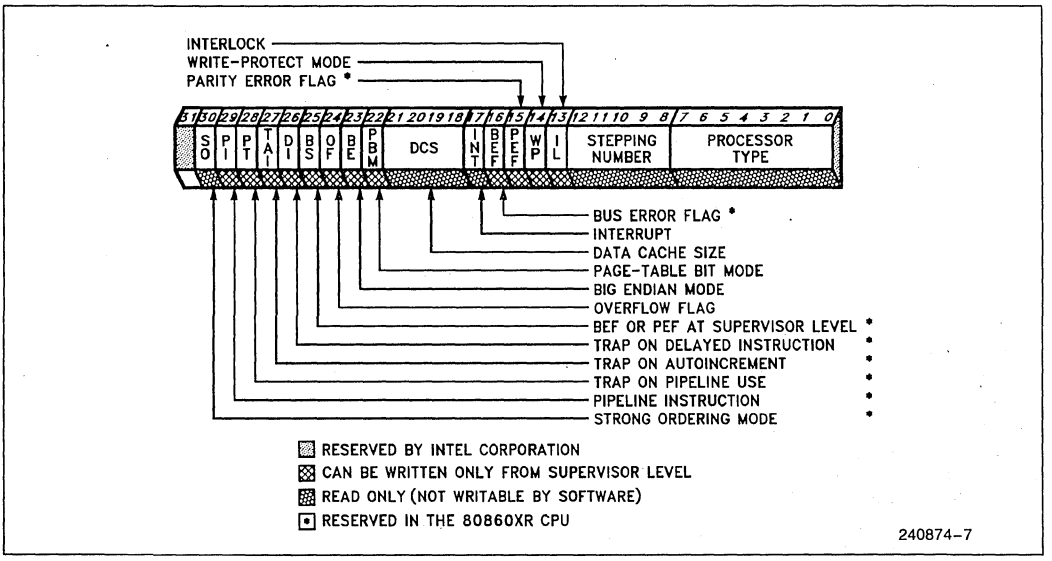


Figure 2.5. Extended Processor Status Register

- WP (write protect) controls the semantics of the W bit of page table entries. A clear W bit in either the directory or the page table entry causes writes to be trapped. When WP is clear, writes are trapped in user mode, but not in supervisor mode. When WP is set, writes are trapped in both user and supervisor modes.
- PEF (parity error flag) is set by the i860 XP microprocessor when a parity error trap occurs. As soon as PEF is set, further parity error and bus error traps are masked. Software must clear PEF to reenable such traps. PEF is set at RESET.
- BEF (bus error flag) is set by the i860 XP microprocessor when the BERR pin is asserted, indicating a bus error. As soon as BEF is set, further parity error and bus error traps are masked. Software must clear BEF to reenable such traps. BEF is set at RESET.
- INT (Interrupt) is the value of the INT input pin.
- DCS (Data Cache Size) is a read-only field that tells the size of the on-chip data cache. The number of bytes actually available is 2^{12} + DCS; therefore, a value of zero indicates 4 Kbytes, one indicates 8 Kbytes, etc. The value of DCS for the i860 XP microprocessor is two, which indicates 16 Kbytes.
- PBM (Page-Table Bit Mode) has no effect in the i860 XP microprocessor. PBM is used by the i860 XR microprocessor.
- BE (Big Endian) controls the ordering of bytes within a data item in memory. Normally (i.e. when BE is clear) the i860 XP microprocessor operates in little endian mode, in which the addressed byte is the low-order byte. When BE is set (big endian mode), the low-order three bits of all 32-bit data load and store addresses are complemented, then masked to the appropriate boundary for alignment. This causes the addressed byte to be the most significant byte. Big endian mode affects not only the memory load and store instructions but also the **ldio, stio, ldint,** and **scyc** instructions.
- OF (Overflow Flag) is set by **adds**, **addu**, **subs**, and **subu** when integer overflow occurs. For **adds** and **subs**, OF is set if the carry from bit 31 is different than the carry from bit 30. For **addu**, OF is set if there is a carry from bit 31. For **subu**, OF is set if there is no carry from bit 31. Under all other conditions, it is cleared by these instructions. OF may be changed by arithmetic instructions in either user or supervisor mode. It may be changed by the **st.c** instruction in supervisor mode only. OF controls the function of the **intovr** instruction. Inside the trap handler, OF may not be valid for traps other than one caused by **intovr**.
- BS (bus or parity error trap in supervisor mode) is set by the i860 XP microprocessor when a bus or parity error occurs during a supervisor mode memory access cycle. This is true even though the processor may have switched to user mode by the time these errors are reported. The BS bit contains valid information only if BERR is asserted in the same clock as BRDY# or one clock after that. In all other conditions the contents of the BS bit are undefined. The operating system can use this bit to decide, for example, whether to abort the process (user mode) or reboot the system (supervisor mode).

- DI (trap on delayed instruction) is set by the i860 XP microprocessor when a trap occurs on a delayed instruction (the instruction located after a delayed branch instruction). When DI is set, the trap handler must restart the interrupted procedure from the branch instruction rather than at the address in **fir**.
- TAI (trap on autoincrement instruction) is set by the i860 XP microprocessor when a trap occurs on an instruction with autoincrement. When TAI is set, the trap handler should undo the autoincrement (that is, restore *src2* to its original value).
- PT (trap on pipeline use) indicates to the i860 XP microprocessor that a trap should be generated and PI should be set when it executes an instruction that uses the floating-point or graphics unit. Such instructions include all the instructions designated "Floating-Point Unit" in Table 2.9, plus the **pfld** instruction. PT is set and cleared only by software. It can be used by the trap handler to avoid unnecessary saving and restoring of the pipelines (refer to section 2.8). When a trap due to PT occurs, the floating-point operation has not started, and the pipelines have not been advanced. Such a trap also sets the IT bit of **psr**.
- The behavior of PI (pipeline instruction) depends on the setting of PT. If PT = 0, the i860 XP microprocessor sets PI when any pipelined instruction or **pfld** is executed. If PT = 1, the processor sets PI and traps when it decodes any instruction that uses the pipes, whether scalar or pipelined. PI may be set even if KNF is set and the next floating point instruction is suppressed. Refer to section 2.8.
- SO (strong ordering) indicates whether the processor is in strong ordering mode (SO = 1) or weak ordering mode (SO = 0). SO is set if the EWBE# pin is active (LOW) at RESET. (Refer to the paragraphs on write cycle reordering in section 5.)

2.2.5 DATA BREAKPOINT REGISTER

The data breakpoint register (**db**) is used to generate a trap when the i860 XP microprocessor accesses an operand at the virtual address stored in this register. The trap is enabled by BR and BW in **psr**. When comparing, a number of low order bits of the address are ignored, depending on the size of the operand. For example, a 16-bit access ignores the low-order bit of the address when comparing to **db**; a 32-bit access ignores the low-order two bits. This ensures that any access that overlaps the address contained in the register will generate a trap. The trap occurs *before* the register or memory update by the load or store instruction.

2.2.6 DIRECTORY BASE REGISTER

The directory base register **dirbase** (shown in Figure 2.6) controls address translation, caching, and bus options.

- ATE (Address Translation Enable), when set, enables the virtual-address translation algorithm.
- DPS (DRAM Page Size) controls how many bits to ignore when comparing the current bus-cycle address with the previous bus-cycle address to generate the NENE# signal. This feature allows for higher speeds when using static column or page-mode DRAMs and consecutive reads and writes access the same column or page. The comparison ignores the low-order 12 + DPS bits. A value of zero is appropriate for one bank of 256K × *n* RAMs, 1 for 1M × *n* RAMS, etc. For interleaved memory, increase DPS by one for each power of interleaving—add one for 2-way, two for 4-way, etc.

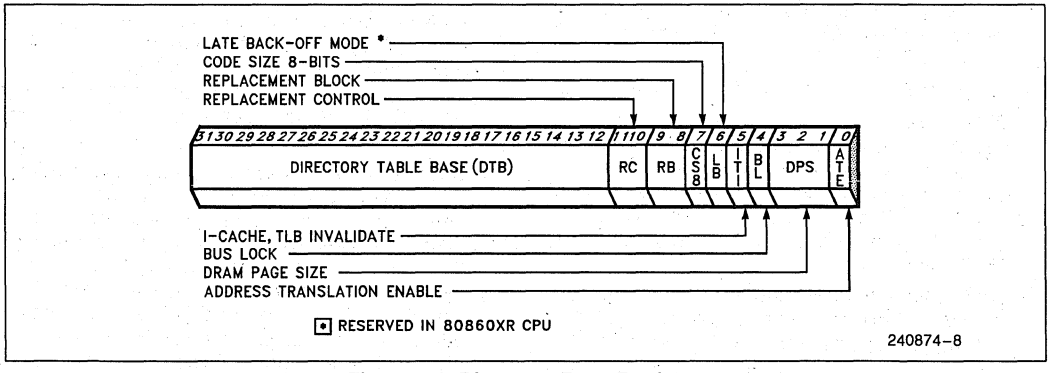


Figure 2.6. Directory Base Register

- When BL (Bus Lock) is set, external bus accesses are locked. The LOCK# signal is asserted with the next bus cycle (excluding instruction fetch and write-back cycles) whose internal bus request is generated after BL is set. It remains set on every subsequent bus cycle as long as BL remains set. The LOCK# signal is deasserted on the next load or store instruction after BL is cleared. Traps immediately clear BL. The **lock** and **unlock** instructions control the BL bit. The result of modifying BL with the **st.c** instruction is not defined.
- ITI (Cache and TLB Invalidate), when set in the value that is loaded into **dirbase**, causes all entries in the instruction cache and virtual tags in the address-translation cache (TLB) to be invalidated. Also invalidates all virtual tags in the data cache. The ITI bit does not remain set in **dirbase**. ITI always appears as zero when reading **dirbase**.
- When software sets the LB bit, the i860 XP microprocessor enters two-clock late back-off mode. This mode gives two additional clock periods of decision time to the external logic that may need to use the BOFF# signal to cancel a bus cycle or data transfer. If the processor enters one-clock late back-off mode during RESET via configuration pin strapping, the LB bit has no effect, and it is impossible to enter two-clock late back-off mode. Furthermore, software cannot exit two-clock late back-off mode once it is activated; the LB bit cannot be cleared except by resetting the processor.
- When CS8 (Code Size 8-Bit) is set, instruction cache misses are processed as 8-bit bus cycles. When this bit is clear, instruction cache misses are processed as 64-bit bus cycles. This bit can not be set by software; hardware sets this bit at initialization time. It can be cleared by software (one time only) to allow the system to execute out of 64-bit memory after bootstrapping from 8-bit EPROM. A nondelayed branch to code in 64-bit memory should directly follow the **st.c** (store control register) instruction that clears CS8, in order to make the transition from 8-bit to 64-bit memory occur at the correct time. The branch instruction must be aligned on a 64-bit boundary.
- RB (Replacement Block) identifies the cache line (block) to be replaced by cache replacement algorithms. RB conditions the cache flush instruction **flush**, which is discussed in Section 10. Table 2.3 explains the values of RB.
- RC (Replacement Control) controls cache replacement algorithms. Table 2.4 explains the significance of the values of RC.
- DTB (Directory Table Base) contains the high-order 20 bits of the physical address of the page directory when address translation is enabled (i.e. ATE = 1). The low-order 12 bits of the address are zeros.

Table 2.3. Values of RB

Value	Replace TLB Block	Replace Instruction and Data Cache Block
0 0	0	0
0 1	1	1
1 0	2	2
1 1	3	3

Table 2.4. Values of RC

Value	Meaning
00	Selects the normal (random) replacement algorithm where any block in the set may be replaced on cache misses in all caches.
01	Instruction, data, and TLB cache misses replace the block selected by RB. This mode is used for cache and TLB testing.
10	Data cache misses replace the block selected by RB. Instruction and TLB caches use random replacement. This mode is used when flushing the data cache with the **flush** instruction.
11	Disables data and TLB caches replacement. Instruction cache uses random replacement.

2.2.7 FAULT INSTRUCTION REGISTER

When a trap occurs, this register contains the address of the trapping instruction (not necessarily the instruction that created the conditions that required the trap). The **fir** is a read-only register. In single-instruction mode, using a **ld.c** instruction to read the **fir** anytime except the first time after a trap saves in *idest* the address of the **ld.c** instruction; in dual-instruction mode, the address of its floating-point companion (address of the **ld.c** − 4) is saved.

2.2.8 FLOATING-POINT STATUS REGISTER

The floating-point status register (**fsr**) contains the floating-point trap and rounding-mode status for the current process. Figure 2.7 shows its format.

- If FZ (Flush Zero) is clear and underflow occurs, a result-exception trap is generated. When FZ is set and underflow occurs, the result is set to zero, and no trap due to underflow occurs.
- If TI (Trap Inexact) is clear, inexact results do not cause a trap. If TI is set, inexact results cause a trap. The sticky inexact flag (SI) is set whenever an inexact result is produced, regardless of the setting of TI.
- RM (Rounding Mode) specifies one of the four rounding modes defined by the IEEE standard. Given a true result *b* that cannot be represented by the target data type, the i860 XP microprocessor determines the two representable numbers *a* and *c* that most closely bracket *b* in value ($a < b < c$). The i860 XP microprocessor then rounds (changes) *b* to *a* or *c* according to the mode selected by RM as defined in Table 2.5. Rounding introduces an error in the result that is less than one least-significant bit.

Table 2.5. Values of RM

Value	Rounding Mode	Rounding Action
00	Round to nearest or even	Closer to *b* of *a* or *c*; if equally close, select even number (the one whose least significant bit is zero).
01	Round down (toward $-\infty$)	*a*
10	Round up (toward $+\infty$)	*c*
11	Chop (toward zero)	Smaller in magnitude of *a* or *c*.

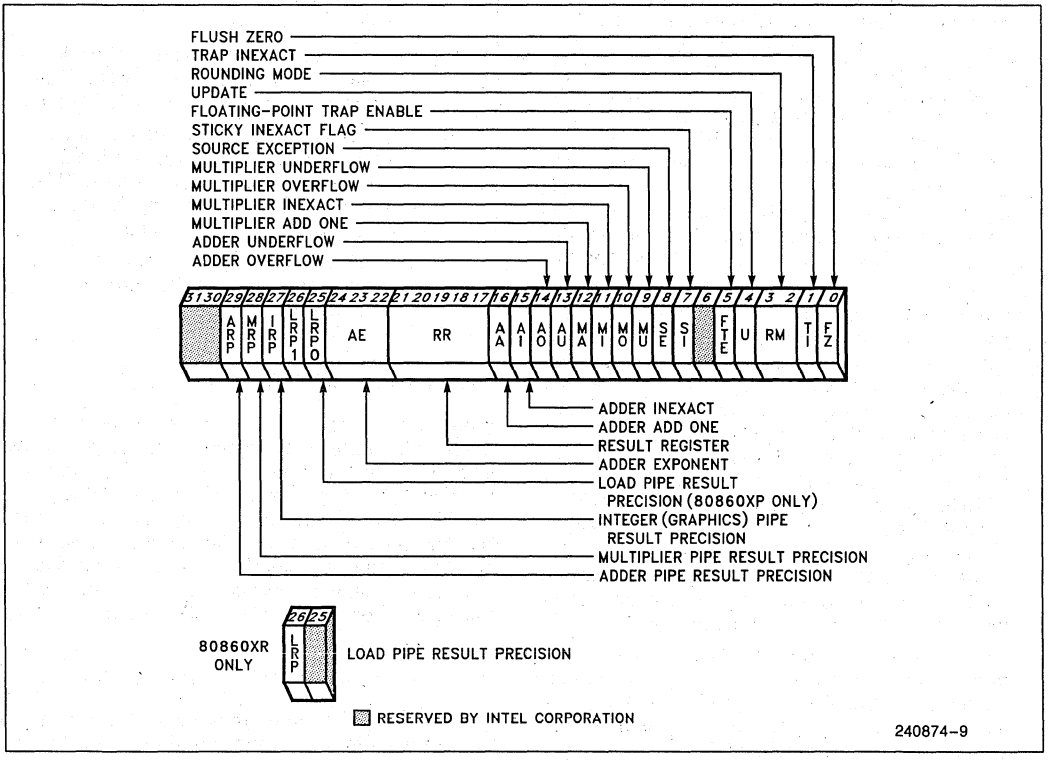


Figure 2.7. Floating-Point Status Register

- The U-bit (Update Bit), if set in the value that is loaded into **fsr** by a **st.c** instruction, enables updating of the result-status bits (AE, AA, AI, AO, AU, MA, MI, MO, and MU) in the first-stage of the floating-point adder and multiplier pipelines. If this bit is clear, the result-status bits are unaffected by a **st.c** instruction; **st.c** ignores the corresponding bits in the value that is being loaded. An **st.c** always updates **fsr** bits 21..17 and 8..0 directly. The U-bit does not remain set; it always appears as zero when read.
- The FTE (Floating-Point Trap Enable) bit, if clear, disables all floating-point traps (invalid input operand, overflow, underflow, and inexact result).
- SI (Sticky Inexact) is set when the last-stage result of either the multiplier or adder is inexact (i.e. when either AI or MI is set). SI is "sticky" in the sense that it remains set until reset by software. AI and MI, on the other hand, can by changed by the subsequent floating-point instruction.
- SE (Source Exception) is set when one of the source operands of a floating-point operation is invalid; it is cleared when all the input operands are valid. Invalid input operands include denormals, infinities, and all NaNs (both quiet and signaling).
- When read from the **fsr**, the result-status bits MA, MI, MO, and MU (Multiplier Add-One, Inexact, Overflow, and Underflow, respectively) describe the last-stage result of the multiplier.

  When read from the **fsr**, the result-status bits AA, AI, AO, AU, and AE (Adder Add-One, Inexact, Overflow, Underflow, and Exponent, respectively) describe the last-stage result of the adder. The high-order three bits of the 11-bit exponent of the adder result are stored in the AE field.

  The Adder Add-One and Multiplier Add-One bits indicate that the absolute value of the result fraction grew by one least-significant bit due to rounding. AA and MA are not influenced by the sign of the result.

  After a floating-point operation in a given unit (adder or multiplier), the result-status bits of that unit are undefined until the point at which result exceptions are reported.

  When written to the **fsr** with the U-bit set, the result-status bits are placed into the first stage of the adder and multiplier pipelines. When the processor executes pipelined operations, it propagates the result-status bits of a particular unit (multiplier or adder) one stage for each pipelined floating-point operation for that unit. When they reach the last stage, they replace the normal result-status bits in the **fsr** and generate traps, if enabled. When the U-bit is not set, result-status bits in the word being written to the **fsr** are ignored.

  In a floating-point dual-operation instruction (e.g. add- and-multiply or subtract-and-multiply), both the multiplier and the adder may set exception bits. The result-status bits for a particular unit remain set until the next operation that uses that unit.
- RR (Result Register) specifies which floating-point register (**f0-f31**) was the destination register when a result-exception trap occurs due to a scalar operation.
- IRP (Integer (Graphics) Pipe Result Precision), MRP (Multiplier Pipe Result Precision), and ARP (Adder Pipe Result Precision) aid in restoring pipeline state after a trap or process switch. Each defines the precision of the last-stage result in the corresponding pipeline. One of these bits is set when the result in the last stage of the corresponding pipeline is double precision; it is cleared if the result is single precision.
- LRP1 and LRP0 (Load Pipe Result Precision) together define the size of the last-stage result of the load pipeline. They are encoded as Table 2.6 shows.

Table 2.6. Values of LRP1 and LRP0

LRP1	LRP0	pfld Length
0	0	*(reserved)*
0	1	4 Bytes
1	0	8 Bytes
1	1	16 Bytes

2.2.9 KR, KI, T, AND MERGE REGISTERS

The KR, KI, and T registers are special-purpose registers used by the dual-operation floating-point instructions **pfam**, **pfsm**, **pfmam**, and **pfmsm**, which initiate both an adder operation and a multiplier operation. The KR, KI, and T registers can store values from one dual-operation instruction and supply them as inputs to subsequent dual-operation instructions. (Refer to Figure 2.16.)

The MERGE register is used only by the graphics instructions. The purpose of the MERGE register is to accumulate (or merge) the results of multiple-addition operations that use as operands the color-intensity values from pixels or distance values from a Z-buffer. The accumulated results can then be stored in one 64-bit operation.

Two multiple-addition instructions and an OR instruction use the MERGE register. The addition instructions are designed to add interpolation values to each color-intensity field in an array of pixels or to each distance value in a Z-buffer.

Refer to the instruction descriptions in section 10 for more information about these registers.

2.2.10 BUS ERROR ADDRESS REGISTER

The **bear** helps the trap handler determine faulty memory locations. The i860 XP microprocessor loads a valid address into **bear** under these conditions:

- For bus errors, the **bear** receives the address of the cycle for which the BERR signal is asserted, if external hardware asserts BERR in the same clock as it asserts BRDY# or one clock later.
- For parity errors on a read, the **bear** receives the address of the cycle during which the processor detects the error, if external hardware asserts PEN# with BRDY# for that cycle.

If external hardware does not meet these conditions, the contents of the **bear** are undefined.

A valid address in **bear** is accurate to 29 bits; that is, address signals A31–A3 are latched in the high-order 29 bits of **bear**. At RESET and after every parity and bus error trap, software must read the **bear** before further parity and bus error traps can occur. The **bear** is a read-only register.

2.2.11 PRIVILEGED REGISTERS

The registers **p0, p1 p2,** and **p3** are provided for the operating system to use. They do not affect processor operation. They can be accessed by the **ld.c** and **st.c** instructions, but they can be written only in supervisor mode. They may be used to store information such as the interrupt stack pointer, current user stack pointer at the beginning of the trap handler, register values during trap handling, processor ID in a multiprocessor system, or for any other purpose.

2.2.12 CONCURRENCY CONTROL REGISTER

The concurrency control register (**ccr**) controls the operation of the internal Concurrency Control Unit (CCU), which is described in section 2.5. The **ccr** can be written in supervisor mode only, but can be read in user or supervisor mode. Figure 2.8 shows the format of the **ccr**.

DO (Detached Only) bit and CO (CCU On) bit together specify the CCU configuration. DO, when set, indicates that there is no external CCU. CO (CCU On) bit, when set, indicates that the Concurrency Control Architecture is enabled. Table 2.7 summarizes the modes defined by CO and DO bits. The reserved combinations should not be used by software.

If the DCCU is on (CO = DO = 1), the processor intercepts and interprets all memory loads and stores which are to the CCU address space, which is the two pages defined by CCUBASE. Loads and stores to that address range do not go to memory, but to the DCCU.

Table 2.7. Values of CO and DO

CO	DO	Mode
0	0	External CCU, or no CCU
0	1	*reserved*
1	0	*reserved*
1	1	Internal CCU (DCCU) only

CCUBASE is the virtual address of the memory area into which the CCU registers are mapped. Software must set bit 12 to zero, because the CCUBASE must be aligned on a two page (8 Kbyte) boundary. This is because an external CCU contains supervisor registers mapped to the second page.

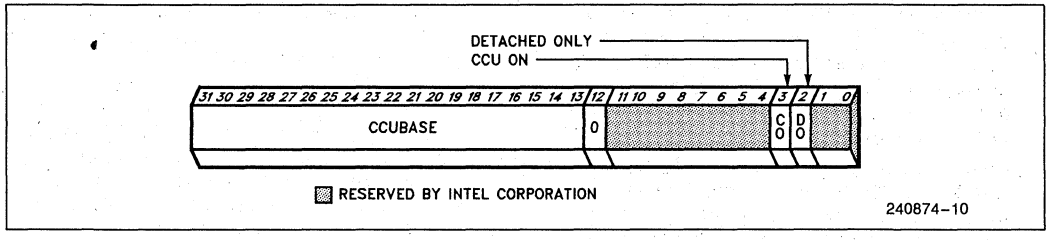


Figure 2.8. Concurrency Control Register

2.2.13 NEWCURR REGISTER

The NEWCURR register is part of the detached CCU (concurrency control unit). It a 32-bit counter that supplies an iteration count for loop execution. (Refer to section 2.5.)

NEWCURR is architecturally a 64-bit register, but only the low-order 32 bits are provided in this implementation. Compiler and operating-system data structures should provide for a 64-bit size for future implementation.

2.2.14 STAT REGISTER

The STAT register is part of the detached CCU (concurrency control unit). As Figure 2.9 shows, it contains the following bits:

InLoop Indicates that the processor is currently executing a concurrent loop. This bit is set when a processor starts a concurrent, non-nested loop, and it is cleared when the processor enters serial code when not nested or idle. It can also be read or written directly.

Nested Indicates whether the processor is in the nested state. InLoop is copied into this bit when starting a nested loop. Otherwise, it can be read or written directly.

Detached Always contains the value of **ccr** bit DO.

STAT is architecturally a 64-bit register. Compiler and operating-system data structures should provide for a 64-bit size for future implementation.

2.3 Addressing

Memory is addressed in byte units with a paged virtual-address space of 2^{32} bytes. Data and instructions can be located anywhere in this address space. Address arithmetic uses 32-bit input values and produces 32-bit results. The low-order 32 bits of the result are used in case of overflow.

Normally, multibyte data values are stored in memory in little endian format, i.e. with the least significant byte at the lowest memory address. As an option, the ordering can be dynamically selected by software in supervisor mode. The i860 XP microprocessor also offers big endian mode, in which the most significant byte of a data item is at the lowest address. Figure 2.10 defines by example how data is transferred from memory over the bus into a register in both modes. Big endian and little endian data areas should not be mixed within a 64-bit data word. Illustrations of data structures in this data sheet show data stored in little endian mode, i.e. the rightmost (low-order) byte is at the lowest memory address.

Code accesses are always done with little endian addressing. This implies that instructions appear differently than documented here when accessed as big endian data. Intel Corporation recommends that disassemblers running in a big endian system convert instructions that have been read as data back to little endian form and present them in the format documented here.

Page directories and page tables are also accessed in little endian mode, regardless of the value of the BE bit.

Big endian mode affects not only the memory load and store instructions but also the **ldio, stio, ldint,** and **scyc** instructions.

Alignment requirements are as follows (any violation results in a data-access trap):

- 128-bit values are aligned on 16-byte boundaries when referenced in memory (i.e. the four least significant address bits must be zero).
- 64-bit values are aligned on 8-byte boundaries when referenced in memory (i.e. the three least significant address bits must be zero).
- 32-bit values are aligned on 4-byte boundaries when referenced in memory (i.e. the two least significant address bits must be zero).

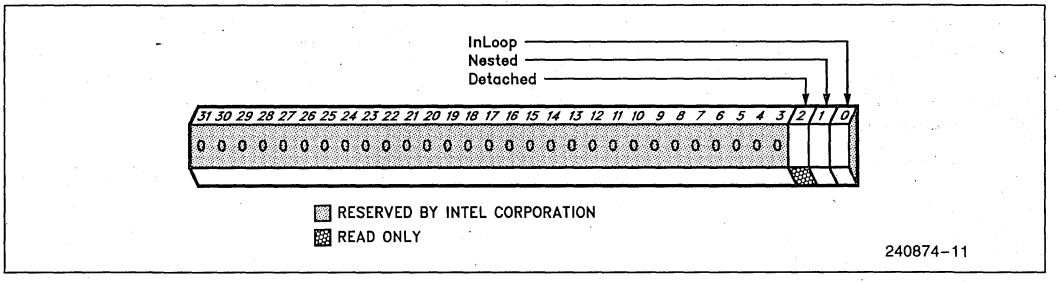
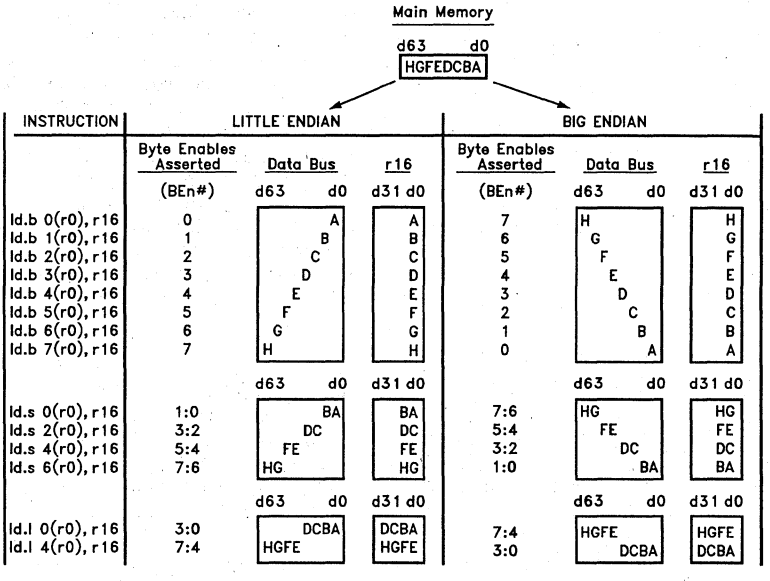


Figure 2.9. Concurrency Status Register



NOTE:
64- and 128-bit big endian accesses are treated the same as little endian accesses

Figure 2.10. Little and Big Endian Memory Transfers

- 16-bit values are aligned on 2-byte boundaries when referenced in memory (i.e. the least significant address bit must be zero).

2.4 Virtual Addressing

When address translation is enabled, the processor maps instruction and data virtual addresses into physical addresses before referencing memory. This address transformation is compatible with that of the Intel386 and Intel486 microprocessors and implements the basic features needed for page-oriented virtual-memory systems and page-level protection.

The address translation is optional. Address translation is disabled when the processor is reset. It is enabled when a store (**st.c**) to **dirbase** sets the ATE bit. The operating system typically does this during software initialization. Address translation is disabled again when **st.c** clears the ATE bit. The ATE bit must be set if the operating system is to implement page-oriented protection or page-oriented virtual memory.

2.4.1 PAGE FRAME

A *page frame* is a unit of contiguous addresses of physical main memory. A *page* is the collection of data that occupies a page frame when that data is present in main memory or occupies some location in secondary storage when there is not sufficient space in main memory.

The i860 XP microprocessor architecture supports two sizes of pages and page frames: four Mbytes and four Kbytes. Four Kbyte page frames begin on four Kbyte boundaries and are fixed in size. Four Mbyte page frames begin on four Mbyte boundaries and are fixed in size. The four Kbyte address transformation is compatible with that of the Intel 486 microprocessor.

2.4.2 VIRTUAL ADDRESS

A virtual address refers indirectly to a physical address by specifying a page and an offset within that page. Figure 2.11 shows the formats of virtual addressess. The format for virtual addresses that refer to four Mbyte pages is different from that of four Kbyte pages.

Figure 2.12 shows how the i860 XP microprocessor converts a virtual address into the physical address by consulting page tables. The addressing mechanism uses the DIR field as an index into a page directory. For 4K pages, it uses the PAGE field as an index into the page table determined by the page directory and uses the OFFSET field to address a byte within the page determined by the page table. For 4M pages, the page directory entry determines the page address, and the OFFSET field addresses a byte within that page table.

2.4.3 PAGE TABLES

A page table is simply an array of 32-bit page specifiers. A page table is itself a page, and contains 4 Kbytes of data or at most 1K 32-bit entries.

At the highest level is a page directory. The page directory holds up to 1K entries that address either page tables of the second level or 4-Mbyte pages.

A page table of the second level addresses up to 1K 4-Kbyte pages. All the tables addressed by one page directory, therefore, can address 1M 4-Kbyte pages.

Whether 4-Mbyte pages, 4-Kbyte pages, or some combination of the two are used, one page directory can cover the entire four gigabyte physical address space of the i860 XP microprocessor (1K page directory entries × 4M page or 1K page directory entries × 1K page table entries × 4K page).

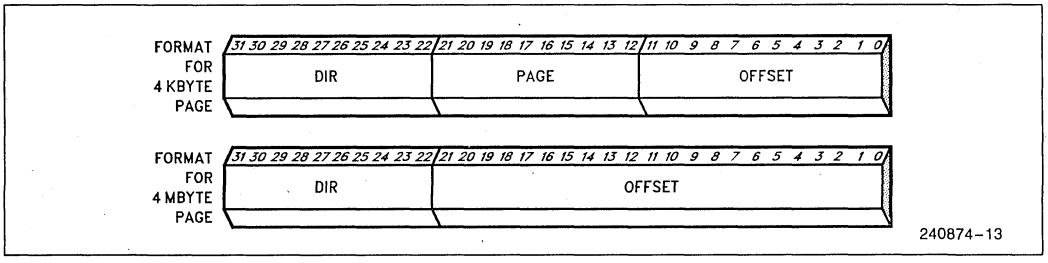


Figure 2.11. Formats of Virtual Addresses

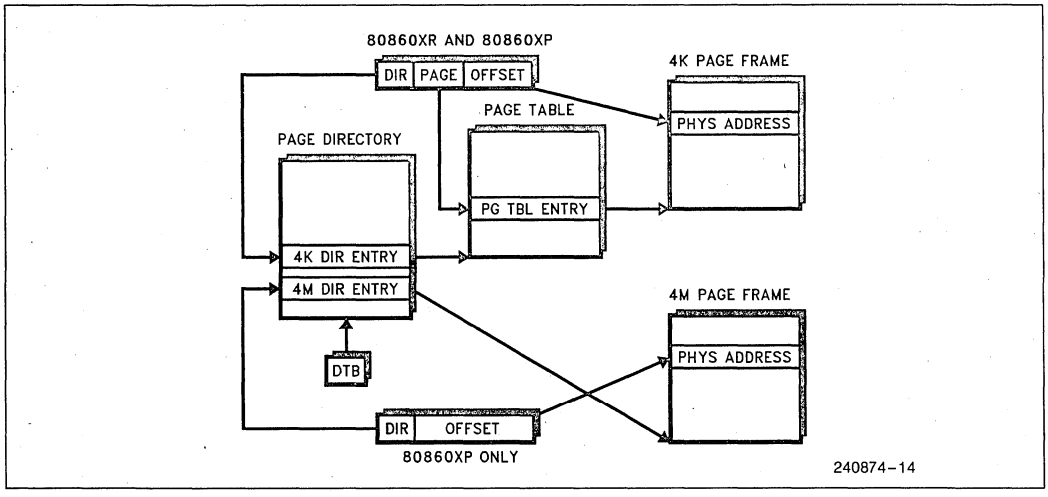


Figure 2.12. Address Translation

The physical address of the current page directory is stored in the DTB field of the **dirbase** register. Memory management software has the option of using one page directory for all processes, one page directory for each process, or some combination of the two.

2.4.4 PAGE-TABLE ENTRIES

Page-table entries (PTEs) have one of the formats shown by Figure 2.13.

2.4.4.1 Page Frame Address

The page frame address specifies the physical starting address of a page. In a page directory, the page frame address is either the address of a page table or the address of the four Mbyte page frame that contains the desired memory operand. In a second-level page table, the page frame address is the address of the 4-Kbyte page frame that contains the desired memory operand.

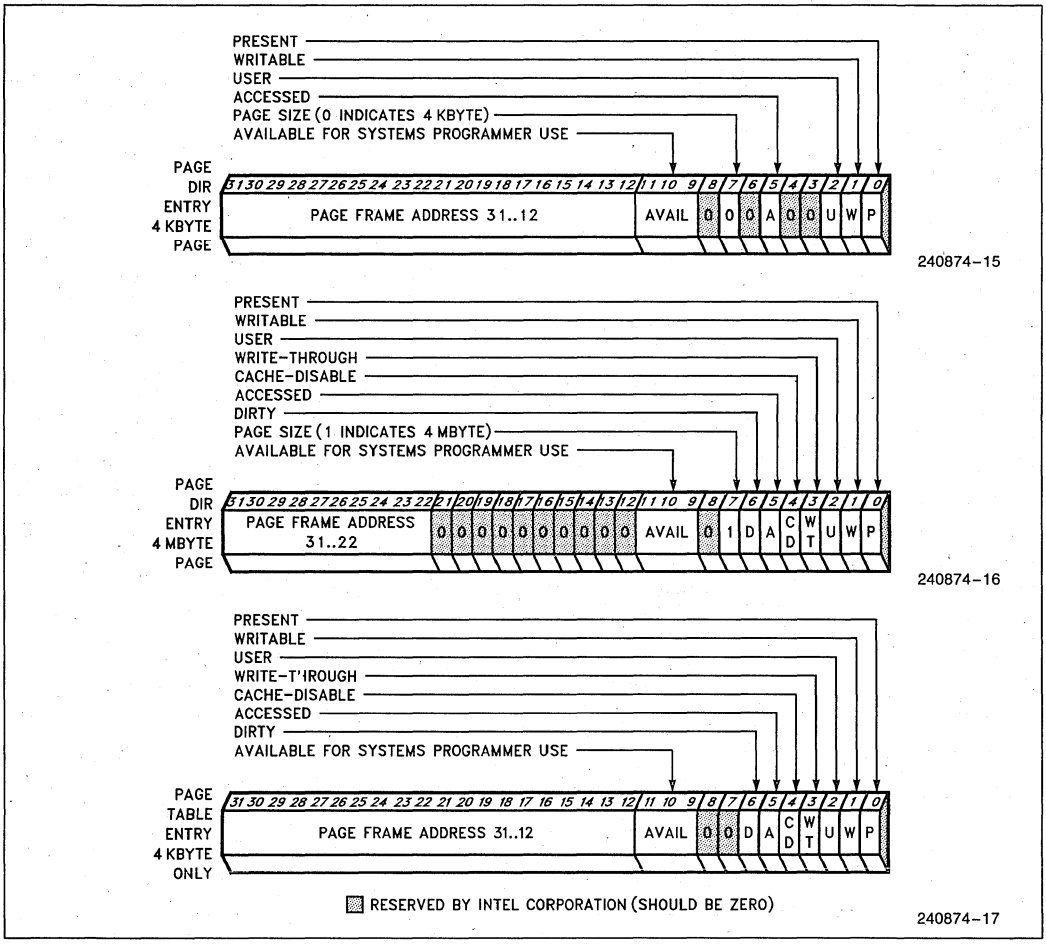


Figure 2.13. Formats of Page Table Entries

2.4.4.2 Present Bit

The P (present) bit indicates whether a page table entry can be used in address translation. P = 1 indicates that the entry can be used. When P = 0 in either level of page tables, the entry is not valid for address translation, and the rest of the entry is available for software use; none of the other bits in the entry is tested by the hardware. If P = 0 in either level of page tables when an attempt is made to use a page-table entry for address translation, the processor signals either a data-access fault or an instruction-access fault. In software systems that support paged virtual memory, the trap handler can bring the required page into physical memory.

Note that there is no P bit for the page directory itself. The page directory may be not-present while the associated process is suspended, but the operating system must ensure that the page directory indicated by the **dirbase** image associated with the process is present in physical memory before the process is dispatched.

2.4.4.3 Writable and User Bits

The W (writable) and U (user) bits are used for page-level protection, which the i860 XP microprocessor performs at the same time as address translation. The concept of privilege for pages is implemented by assigning each page to one of two levels:

Supervisor level (U = 0) — For the operating system and other systems software and related data.

User level (U = 1) — For applications procedures and data.

The U bit of the **psr** indicates whether the i860 XP microprocessor is executing at user or supervisor level. The i860 XP microprocessor maintains the U bit of **psr** as follows:

- The i860 XP microprocessor clears the **psr** U bit to indicate supervisor level when a trap occurs (including when the **trap** instruction causes the trap). The prior value of U is copied into PU.
- The i860 XP microprocessor copies the **psr** PU bit into the U bit when an indirect branch is executed and one of the trap bits is set. If PU was one, the i860 XP microprocessor enters user level.

With the U bit of **psr** and the W and U bits of the page table entries, the i860 XP microprocessor implements the following protection rules:

- When at user level, a read or write of a supervisor-level page causes a trap.
- When at user level, a write to a page whose W bit is not set causes a trap.
- When at user level, a store (**st.c**) to certain control registers is ignored.
- When at user level, privileged instructions (**ldio, stio, scyc, ldint**) have no effect.

When the i860 XP microprocessor is executing at supervisor level, all pages are addressable, but, when it is executing at user level, only pages that belong to the user level are addressable.

When the i860 XP microprocessor is executing at supervisor level, all pages are readable. Whether a page is writable depends upon the write-protection mode controlled by WP of **epsr**:

WP = 0 All pages are writable.

WP = 1 A write to page whose W bit is not set causes a trap.

When the i860 XP microprocessor is executing at user level, only pages that belong to user level and are marked writable are actually writable; pages that belong to supervisor level are neither readable nor writable from user level.

2.4.4.4 Write-Through Bit

The i860 XP microprocessor implement both write-back and write-through caching policies for the on-chip instruction and data caches. If WT is set, the write-through policy is applied to data from the corresponding page. If WT is clear, the normal write-back policy is applied to data from the page.

For four-Mbyte pages, the WT bit of the page directory entry is used. For four-Kbyte pages, only the WT bit of the second-level page table entry is used; the WT bit of the page directory entry is not referenced by the processor, but is *reserved*.

The value of the WT bit is driven externally on the PWT pin, so that external caches can employ the same policy used internally.

2.4.4.5 Cache Disable Bit

If a page's CD (cache disable) bit is set, data from the page is not placed in the internal instruction or data caches (regardless of the value of the WT bit). Clearing CD permits the processor to place data from the associated page into internal caches.

For four-Mbyte pages, the CD bit of the page directory entry is used. For four-Kbyte pages, only the CD bit of the second-level page table entry is used; the CD bit of the page directory entry is not referenced by the processor, but is *reserved*.

The value of the CD bit is driven externally on the PCD pin, so that cacheability can be the same in both internal and external caches.

2.4.4.6 Accessed and Dirty Bits

The A (accessed) and D (dirty) bits provide data about page usage in both levels of the page tables.

The i860 XP microprocessor sets the A-bit before a read or write operation to a page. For four-Kbyte pages, it sets the A-bit of both levels of page tables.

The processor tests the dirty bit before a write, and, under certain conditions, causes traps. The trap handler then has the opportunity to maintain appropriate values in the dirty bits. For four-Mbyte pages, the D bit of the page directory entry is used. For four-Kbyte pages, only the D bit of the second-level page table entry is used; the D bit of the page directory entry is not referenced by the processor, but is *reserved*. The precise algorithm for using these bits is specified in section 2.4.5.

An operating system that supports paged virtual memory can use the D and A bits to determine what pages to eliminate from physical memory when the demand for memory exceeds the physical memory available. The D and A bits are normally initialized to zero by the operating system. The processor sets the A bit when a page is accessed either by a read or write operation. When a data-access fault occurs, the trap handler sets the D bit if an allowable write is being performed, then reexecutes the instruction.

The operating system is responsible for coordinating its updates to the accessed and dirty bits with updates by the CPU and by other processors that may share the page tables. The i860 XP microprocessor automatically asserts the LOCK# signal while testing and setting the A bit.

2.4.4.7 Page Tables for Trap Handlers

When paging is enabled (ATE = 1), software that creates page tables and directories must assure that A = 1 always in the PTEs and PDEs for the code pages of the trap handler and the first data page accessed by the handler. Preallocation of these pages is required in case a trap occurs during a lock sequence. Otherwise, recursive traps would be generated, as the A-bit would need to be set by the translation hardware, which is a trapping situation in itself.

2.4.4.8 Combining Protection of Both Levels of Page Tables

For any four-Kbyte page, the protection attributes of its page directory entry may differ from those of its page table entry. The i860 XP microprocessor computes the effective protection attributes for a page by examining the protection attributes in both the directory and the page table and choosing the more restrictive of the two.

2.4.5 ADDRESS TRANSLATION ALGORITHM

The following algorithm defines the translation of each virtual address to a physical address. Let DIR, PAGE, and OFFSET be the fields of the virtual address; let PFA1 and PFA2 be the page frame address fields of the first and second level page tables respectively; DTB is the page directory table base address stored in the **dirbase** register.

1. Read the PDE (Page Directory Entry) at the physical address formed by DTB:DIR:00.
2. If P in the PDE is zero, generate a data- or instruction-access fault.
3. If W in the PDE is zero, the operation is write, and either the U bit of the PSR is set or WP = 1, generate a data-access fault.
4. If the U bit in the PDE is zero and U bit in the **psr** is set, generate a data- or instruction-access fault.
5. If A in the PDE is zero and the TLB miss occurred inside a locked sequence, generate a data or instruction access fault. (The trap allows software to set A to one and restart the sequence. This helps external bus hardware determine unambiguously what address corresponds to a locked semaphore.)
6. If bit 7 of the PDE is one (four Mbyte page), and the operation is write, and D = 0 in the PDE, generate a data-access fault.
7. If A = 1 in the PDE, continue at step 11. Otherwise, assert LOCK#.
8. Perform the PDE read as in step 1 and the P, W and U bit checks as in steps 2 through 4.
9. Write the PDE with A bit set.
10. Deassert LOCK#.
11. If bit 7 of the PDE is one (four Mbyte page), form the physical address as PFA1:OFFSET, and exit address translation. In this case, PFA1 is 10 bits and OFFSET is 22 bits.
12. The remaining steps are for four Kbyte pages. If the A-bit in the PDE was zero before translation began, assert LOCK#.

13. Fetch the PTE at the physical address formed by PFA1:PAGE:00.
14. Perform the P-, W-, U-, and A-bit checks as in steps 2 through 5 with the second-level PTE. If A = zero in the PTE, and the TLB miss occurred inside a locked sequence, generate a data or instruction access fault. LOCK# remains active.
15. If the operation is write, and D in the PTE is zero, generate a data access fault.
16. If the A-bit in the PDE was already active before translation began, and the A-bit in the PTE is already active, go to step 20.
17. If LOCK# is not already active, assert it and refetch the PTE.
18. Perform the U-, W-, and P-bit checks and A-bit setting in the PTE as in steps 8 through 9. Do the locked write update of the PTE to unlock the bus, even if the A-bit in the PTE is already one.
19. Deassert LOCK#.
20. Form the physical address as PFA2:OFFSET. In this case, PFA2 is 20 bits and OFFSET is 12 bits.

During translation, the i860 XP microprocessor looks only in external memory for page directories and page tables. The data cache is not searched. Therefore, any code that modifies page directories or page tables must keep them out of the cache. The tables should either be kept in noncacheable memory or in write-through pages or should be flushed from the cache.

The i860 XP microprocessor expects page directories and page tables to be in little endian format. The operating system must maintain these tables in little endian format either by setting BE to zero when manipulating the tables or by complementing bit two of the 32-bit address when loading or storing entries.

2.4.6 ADDRESS TRANSLATION FAULTS

The address translation fault can be signalled as either an instruction access fault or a data-access fault. The instruction causing the fault can be reexecuted upon returning from the trap handler.

2.5 Detached CCU

The i860 XP microprocessor supports parallel processing, where multiple processors work simultaneously on different parts of the same problem. The Concurrency Control Unit (CCU) controls work sharing among CPUs, in multiprocessor systems. The CCU is a VLSI chip that allows multiple processors to work together to execute portions of a single program in parallel. The CCU performs the iteration assignment for loop parallelization. Accesses to the CCU for synchronization are much faster than accesses to shared memory semaphores. The CCU is memory mapped, and its internal registers are accessed via memory load and store operations.

To take advantage of the parallel architecture, software must be compiled by parallelizing compilers that generate instructions to access the CCU. However, such instructions cannot run on a system that does not include a CCU. To allow an application compiled for parallel execution to run on any system based on the i860 XP microprocessor, a "Detached Only" CCU (DCCU, also referred to as "internal CCU") is implemented in the i860 XP microprocessor. The DCCU is a compatible subset of the external CCU, consisting of the minimal set of features required for a single CPU. The DCCU alone neither increases performance nor concurrency, but does allow software designed for parallel processing to run unmodified on a single CPU.

2.5.1 DCCU INITIALIZATION

After reset, the i860 XP microprocessor DCCU is disabled (CO and DO bits in **ccr** are cleared). To enable the DCCU, the CO and DO bits in **ccr** must be set by software. Before turning on the CCU, the operating system must invalidate the TLB and flush the data cache to make sure that they do not contain data from the CCU pages. The TLB is invalidated by setting ITI = 1 in the **dirbase** register. Also, the **flush** instruction must be used once per each line of the data cache to invalidate the physical address of the cache entry, if the two pages at the CCUBASE address may have been cached. The flush is unneeded if page tables or external hardware have prohibited caching of the CCUBASE pages.

Neither the external CCU nor the DCCU can be accessed within four instructions after **ccr** is modified.

2.5.2 DCCU ADDRESSING

The CCU facilities are memory-mapped, manipulated by normal load and store instructions. The DCCU is memory-mapped to a single 4 Kbyte user page. When the DCCU is active, all accesses to this page are satisfied by the DCCU, and no external bus cycle is generated. The address space of two adjacent pages beginning on an 8 Kbyte boundary is reserved for the CCU. The first (lower address) page contains

locations accessible in user mode (which includes the DCCU registers), and the second page contains locations accessible in supervisor mode (used for external CCU only). The base address of these pages is specified by the CCUBASE field in **ccr**. Accesses to the second page in DCCU-only mode have no effect on the DCCU, and are treated as normal memory accesses.

When the DCCU is active, accesses to its address page use only the virtual address, and no translation is done on the DCCU access. However, the accesses to an external CCU go through normal address translation. The operating system should make sure that the page table entries for the CCU pages are set so that no fault occurs during address translation. If an external CCU is used, the two PTEs for the CCU should have CD = 1 (caching disabled) and page frame addresses that match the external hardware addresses of the CCU. Accesses to the DCCU that cause a TLB miss do not cause the PTE to be loaded into the TLB.

If the external CCU is used when address translation is disabled (ATE=0), external hardware must deactivate KEN# for such accesses, to avoid caching external CCU accesses.

2.5.3 DCCU INTERNALS

The DCCU consists of an address decoder, a 32-bit counter (NEWCURR), and three bits of state information (InLoop, Nested, and Detached). InLoop, Nested and Detached correspond to bits 0, 1, and 2 respectively of the external CCU STAT register. The Detached bit always reflects the value of the DO bit in **ccr**.

Several addresses within the DCCU memory page are decoded to cause actions to NEWCURR, InLoop, and Nested state bits. The CCU register to be accessed is specified by address bits 11–3. The valid CCU addresses are shown in Table 2.8 with their mnemonics. Accesses to these address may also have side effects within the DCCU. Refer to the *i860™ Microprocessor Family Programmer's Reference Manual* for programming information. Loads from any other addresses within the DCCU memory page return zero; stores to any other addresses have no effect. Access to the DCCU by any load or store instructions other than **ld.x** and **st.x** produce undefined results.

Assemblers should encode address bits 2–0 as zero for accesses in little-endian mode. However, in big-endian mode (**epsr** BE bit = 1), DCCU accesses should have address bit 2 active. Thus, software for big-endian access to the DCCU must differ from little-endian software. That allows an external CCU to be accessed in both big and little endian modes.

When reading from the DCCU, the access latency is the same as reading data from the data cache—the data is ready for use as a source by the second instruction after the load. The first instruction after the load may use the data, but that instruction will experience a one-clock freeze before the data becomes available.

2.6 Instruction Set

Table 2.9 shows the complete set of instructions for the i860 XP microprocessor, grouped by function within processing unit. Refer to Section 10 for an algorithmic definition of each instruction. The instruction set of the i860 XP microprocessor is fully upward compatible with that of the i860 XR microprocessor, extended in a few ways to better serve certain application domains. User-level software applications written for the i860 XR microprocessor will run unmodified on the i860 XP microprocessor, but some supervisor code (for example, trap handlers) may need minor modifications. The i860 XR microprocessor instruction set has been extended with the following instructions:

- **ldio, stio**: I/O load and store instructions
- **ldint**: Load interrupt instruction to perform an interrupt acknowledge cycle and read the interrupt vector. Used to emulate the Intel 486 interrupt acknowledge sequence.
- **scyc**: A special-cycle instruction, used to generate bus cycles that signal invalidation and synchronization of an external cache.
- **pfld.q**: A pipelined, floating-point load of 128 bits.

Table 2.8. CCU Addresses

Mnemonic	A11–A8	A7–A4	Little Endian A3–A0	Big Endian A3–A0
cbr_*i*	0000	0**abc**	**b**000	**d**100
cget	1111	0110	0000	0100
cnewcurr	1111	1100	0000	0100
cstat	1111	1100	1000	1100
cstatci	1111	1101	0000	0100
cstatn	1111	1101	1000	1100
cclm	1111	1110	1000	1100
cver	1111	1111	1000	1100

NOTE:
Variable **i** is a 4-bit index formed by A6–A3. Let its binary form be represented by the symbols **abcd**.

Table 2.9. Instruction Set (1 of 2)

Core Unit	
Mnemonic	**Description**
Load and Store Instructions	
ld.x	Load integer
st.x	Store integer
fld.y	F-P load
fst.y	F-P store
pfld.y	Pipelined F-P load
pst.d	Pixel store
Register to Register Move	
ixfr	Transfer integer to F-P register
Integer Arithmetic Instructions	
addu	Add unsigned
adds	Add signed
subu	Subtract unsigned
subs	Subtract signed
Shift Instructions	
shl	Shift left
shr	Shift right
shra	Shift right arithmetic
shrd	Shift right double
Logical Instructions	
and	Logical AND
andh	Logical AND high
andnot	Logical AND NOT
andnoth	Logical AND NOT high
or	Logical OR
orh	Logical OR high
xor	Logical exclusive OR
xorh	Logical exclusive OR high
Control-Transfer Instructions	
br	Branch direct
bri	Branch indirect
bc	Branch on CC
bc.t	Branch on CC taken
bnc	Branch on not CC
bnc.t	Branch on not CC taken
bte	Branch if equal
btne	Branch if not equal
bla	Branch on LCC and add
call	Subroutine call
calli	Indirect subroutine call
intovr	Software trap on integer overflow
trap	Software trap

Floating-Point Unit	
Mnemonic	**Description**
Register to Register Move	
fxfr	Transfer F-P to integer register
F-P Multiplier Instructions	
fmul.p	F-P multiply
pfmul.p	Pipelined F-P multiply
pfmul3.dd	3-Stage pipelined F-P multiply
fmlow.p	F-P multiply low
frcp.p	F-P reciprocal
fsqr.p	F-P reciprocal square root
F-P Adder Instructions	
fadd.p	F-P add
pfadd.p	Pipelined F-P add
famov.r	F-P adder move
pfamov.r	Pipelined F-P adder move
fsub.p	F-P subtract
pfsub.p	Pipelined F-P subtract
pfgt.p	Pipelined greater-than compare
pfeq.p	Pipelined equal compare
fix.v	F-P to integer conversion
pfix.v	Pipelined F-P to integer conversion
ftrunc.v	F-P to integer truncation
Dual-Operation Instructions	
pfam.p	Pipelined F-P add and multiply
pfsm.p	Pipelined F-P subtract and multiply
pfmam.p	Pipelined F-P multiply with add
pfmsm.p	Pipelined F-P multiply with subtract
Long Integer Instructions	
fisub.z	Long-integer subtract
pfisub.z	Pipelined long-integer subtract
fiadd.z	Long-integer add
pfiadd.z	Pipelined long-integer add
Graphics Instructions	
fzchks	16-bit Z-buffer check
pfzchds	Pipelined 16-bit Z-buffer check
fzchkl	32-bit Z-buffer check
pfzchkl	Pipelined 32-bit Z-buffer check
faddp	Add with pixel merge
pfaddp	Pipelined add with pixel merge
faddz	Add with Z merge
pfaddz	Pipelined add with Z merge
form	OR with MERGE register
pform	Pipelined OR with MERGE register

Table 2.9. Instruction Set (2 of 2)

Core Unit	
Mnemonic	**Description**
I/O Instructions	
ldio.x	Load I/O
stio.x	Store I/O
ldint.x	Load interrupt vector
System Control Instructions	
flush	Cache flush
ld.c	Load from control register
st.c	Store to control register
lock	Begin interlocked sequence
unlock	End interlocked sequence
scyc.x	Special bus cycles
Assembler Pseudo-Operations	
Register to Register Move	
mov	Integer move
fmov.r	F-P reg-reg move
pfmov.r	Pipelined F-P reg-reg move
nop	Core no-operation
fnop	F-P no-operation
pfle.p	Pipelined F-P less-than or equal

The architecture of the i860 XP microprocessor uses parallelism to increase the rate at which operations may be introduced into the unit. Parallelism in the i860 XP microprocessor is **not** transparent; rather, programmers have complete control over parallelism and therefore can achieve maximum performance for a variety of computational problems.

2.6.1 PIPELINED AND SCALAR OPERATIONS

One type of parallelism used within the floating-point unit is "pipelining". The pipelined architecture treats each operation as a series of more primitive operations (called "stages") that can be executed in parallel. Consider just the floating-point adder as an example. Let **A** represent the operation of the adder. Let the stages be represented by $\mathbf{A}_1$, $\mathbf{A}_2$, and $\mathbf{A}_3$. The stages are designed such that $\mathbf{A}_{i+1}$ for one adder instruction can execute in parallel with $\mathbf{A}_i$ for the next adder instruction. Furthermore, each $\mathbf{A}_i$ can be executed in just one clock. The pipelining within the multiplier and graphics units can be described similarly, except that the number of stages may be different.

Figure 2.14 illustrates three-stage pipelining as found in the floating-point adder (also in the floating-point multiplier when single-precision input operands are employed). The central columns of the table represent the three stages of the pipeline. Each stage holds intermediate results and also (when introduced into the first stage by software) holds status information pertaining to those results. The table assumes that the instruction stream consists of a series of consecutive floating-point instructions, all of one type (i.e. all adder instructions or all single-precision multiplier instructions). The instructions are represented as **A**, **B**, etc. The rows of the table represent the states of the unit at successive clock cycles. Each time a pipelined operation is performed, the result of the last stage of the pipeline is stored in the destination register *fdest*, the pipeline is advanced one stage, and the input operands of the operation are transferred to the first stage of the pipeline.

Clock	Instruction	Pipeline			Result
		Stage 1	Stage 2	Stage 3	
1	A	A			
2	B	B	A		
3	C	C	B	A	
4	D	D	C	B	A → *fdest* of D
5	E	E	D	C	B → *fdest* of E
6	F	F	E	D	C → *fdest* of F

Figure 2.14. Pipelined Instruction Execution

In the i860 XP microprocessor, the number of pipeline stages ranges from one to three. A pipelined operation with a three-stage pipeline stores the result of the third prior operation. A pipelined operation with a two-stage pipeline stores the result of the second prior operation. A pipelined operation with a one-stage pipeline stores the result of the prior operation.

There are four floating-point pipelines: one for the multiplier, one for the adder, one for the graphics unit, and one for floating-point loads. The adder pipeline has three stages. The number of stages in the multiplier pipeline depends on the precision of the source operands in the pipeline; it may have two or three stages. The graphics unit has one stage for all precisions. The load pipeline has three stages for all precisions.

Changing the FZ (flush zero), RM (rounding mode), or RR (result register) bits of **fsr** while there are results in either the multiplier or adder pipeline produces effects that are not defined.

2.6.1.1 Scalar Mode

In addition to the pipelined execution mode, the i860 XP microprocessor also can execute floating-point instructions in "scalar" mode. Most floating-point instructions have both pipelined and scalar variants, distinguished by a bit in the instruction encoding. In scalar mode, the floating-point unit does not start a new operation until the previous floating-point operation is completed. The scalar operation passes through all stages of its pipeline before a new operation is introduced, and the result is stored automatically. Scalar mode is used when the next operation depends on results from the previous few floating-point operations (or when the compiler or programmer does not want to deal with pipelining).

2.6.1.2 Pipelining Status Information

Result status information in the **fsr** consists of the AA, AI, AO, AU, and AE bits, in the case of the adder, and the MA, MI, MO, and MU bits, in the case of the multiplier. This information arrives at the **fsr** via the pipeline in one of two ways:

1. It is calculated by the last stage of the pipeline. This is the normal case.
2. It is propagated from the first stage of the pipeline. This method is used when restoring the state of the pipeline after a preemption. When a store instruction updates the **fsr** and the the U bit being written into the **fsr** is set, the store updates the result status bits in the first stage of both the adder and multiplier pipelines. When software changes the result-status bits of the first stage of a particular unit (multiplier or adder), the updated result-status bits are propagated one stage for each pipelined floating-point operation for that unit. In this case, each stage of the adder and multiplier pipelines holds its own copy of the relevant bits of the **fsr**. When they reach the last stage, they override the normal result-status bits computed from the last-stage result.

At the next floating-point instruction (or at certain core instructions), after the result reaches the last stage, the i860 XP microprocessor traps if any of the status bits of the **fsr** indicate exceptions. Note that the instruction that creates the exceptional condition is not the instruction at which the trap occurs.

2.6.1.3 Precision in the Pipelines

In pipelined mode, when a floating-point operation is initiated, the result of an earlier pipelined floating-point operation is returned. The result precision of the current instruction applies to the operation being initiated. The precision of the value stored in *fdest* is that which was specified by the instruction that initiated that operation.

If *fdest* is the same as *fsrc1* or *fsrc2*, the value being stored in *fdest* is used as the input operand. In this case, the precision of *fdest* must be the same as the source precision.

The multiplier pipeline has two stages when the source operands are double-precision and three stages when they are single. This means that a pipelined multiplier operation stores the result of the second previous multiplier operation for double-precision inputs and third previous for single-precision inputs (except when changing precisions).

2.6.1.4 Transition between Scalar and Pipelined Operations

When a scalar operation is executed, it passes through all stages of the pipeline; therefore, any unstored results in the affected pipeline are lost. To avoid losing information, the last pipelined operations before a scalar operation should be dummy pipelined operations that unload unstored results from the affected pipeline.

After a scalar operation, the values of all pipeline stages of the affected unit (except the last) are undefined. No spurious result-exception traps result when the undefined values are subsequently stored by pipelined operations; however, the values should not be referenced as source operands.

For best performance a scalar operation should not immediately precede a pipelined operation whose *fdest* is nonzero.

2.6.1.5 Pipelined Loads

The **pfld** instruction is optimized for accesses that miss the data cache and transfer directly from memory. Therefore, even when there is a data cache hit, a **pfld** may generate a bus cycle. The data from the internal cache is used only if it was modified. Otherwise, data is taken from the external bus, even if it resides in the on-board cache.

The **pfld** FIFO can be extended externally, due to the facts that a **pfld** always generates a bus cycle and that such a cycle can be identified externally by the value on the CTYP pin. Software written for an externally-extended **pfld** pipeline must ensure that it does not **pfld** from a location that was modified in the data cache. When a **pfld** cache hit to a modified line occurs, the **pfld** pipeline length used by the i860 XP microprocessor is three stages. The modified data from the cache is put into the internal three-stage data FIFO, and the third **pfld** instruction after the data cache hit will update its *fdest* register with the modified data.

2.6.2 DUAL-INSTRUCTION MODE

Another form of parallelism results from the fact that the i860 XP microprocessor can execute both a floating-point and a core instruction simultaneously. Such parallel execution is called *dual-instruction mode*. When executing in dual-instruction mode, the instruction sequence consists of 64-bit aligned instruction pairs, with a floating-point instruction in the lower 32 bits and a core instruction in the upper 32 bits. Table 2.9 identifies which instructions are executed by the core unit and which by the floating-point unit.

Programmers specify dual-instruction mode either by including in the mnemonic of a floating-point instruction a **d.** prefix or by using the Assembler directives **.dual** . . . **.enddual**. Both of the specifications cause the D-bit of floating-point instructions to be set. If the i860 XP microprocessor is executing in single-instruction mode and encounters a floating-point instruction with the D-bit set, one more 32-bit instruction is executed before dual-mode execution begins. If the i860 XP microprocessor is executing in dual-instruction mode and a floating-point instruction is encountered with a clear D-bit, then one more pair of instructions is executed before resuming single-instruction mode. Figure 2.15 illustrates two variations of this sequence of events: one for extended sequences of dual-instructions and one for a single instruction pair.

Note that **d.fnop** cannot be used to initiate dual instruction mode.

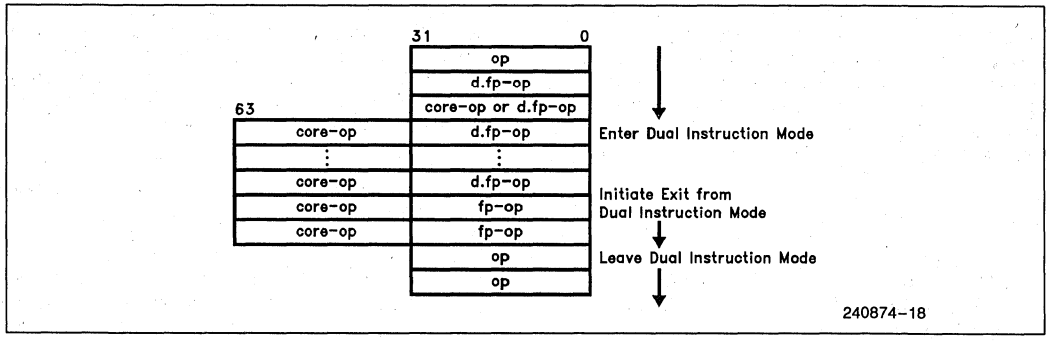


Figure 2.15. Dual-Instruction Mode Transitions (1 of 2)

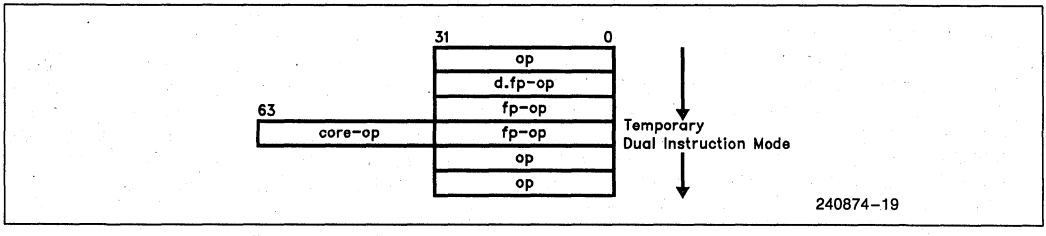


Figure 2.15. Dual-Instruction Mode Transitions (2 of 2)

When a 64-bit dual-instruction pair sequentially follows a delayed branch instruction in dual-instruction mode, both 32-bit instructions are executed.

2.6.3 DUAL-OPERATION INSTRUCTIONS

Special dual-operation floating-point instructions (add-and-multiply, subtract-and-multiply) use both the multiplier and adder units within the floating-point unit in parallel to efficiently execute such common tasks as evaluating systems of linear equations, performing the Fast Fourier Transform (FFT), and performing graphics transformations.

The instruction classes **pfam** *fsrc1, fsrc2, fdest*, **pfmam** *fsrc1, fsrc2, fdest* (add and multiply), **pfsm** *fsrc1, fsrc2, fdest*, and **pfmsm** *fsrc1, fsrc2, fdest* (subtract and multiply) initiate both an adder operation and a multiplier operation. Six operands are required, but the instruction format specifies only three operands; therefore, there are special provisions for specifying the operands. These special provisions consist of:

- Three special registers (KR, KI, and T) that can store values from one dual-operation instruction and supply them as inputs to subsequent dual-operation instructions.
  - — The constant registers KR and KI can store the value of *fsrc1* and subsequently supply that value to the multiplier pipeline in place of *fsrc1*.
  - — The transfer register T can store the last-stage result of the multiplier pipeline and subsequently supply that value to the adder pipeline in place of *fsrc1*.
- A four-bit data-path control field in the opcode (DPC) that specifies the operands and loading of the special registers.
  1. Operand-1 of the multiplier can be KR, KI, or *fsrc1*.
  2. Operand-2 of the multiplier can be *fsrc2*, the last-stage result of the multiplier pipeline, or the last-stage result of the adder pipeline.
  3. Operand-1 of the adder can be *fsrc1*, the T-register, the last-stage result of the multiplier pipeline, or the last-stage result of the adder pipeline.
  4. Operand-2 of the adder can be *fsrc2*, the last-stage result of the multiplier pipeline, or the last-stage result of the adder pipeline.

Figure 2.16 shows all the possible data paths surrounding the adder and multiplier. The DPC field in these instructions selects different data paths. Section 10 shows the various encodings of the DPC field.

Note that the mnemonics **pfam.p**, **pfsm.p**, **pfmam.p**, and **pfmsm.p** are never used as such in the assembly language; these mnemonics are used here to designate classes of related instructions. Each value of DPC has a unique mnemonic associated with it.

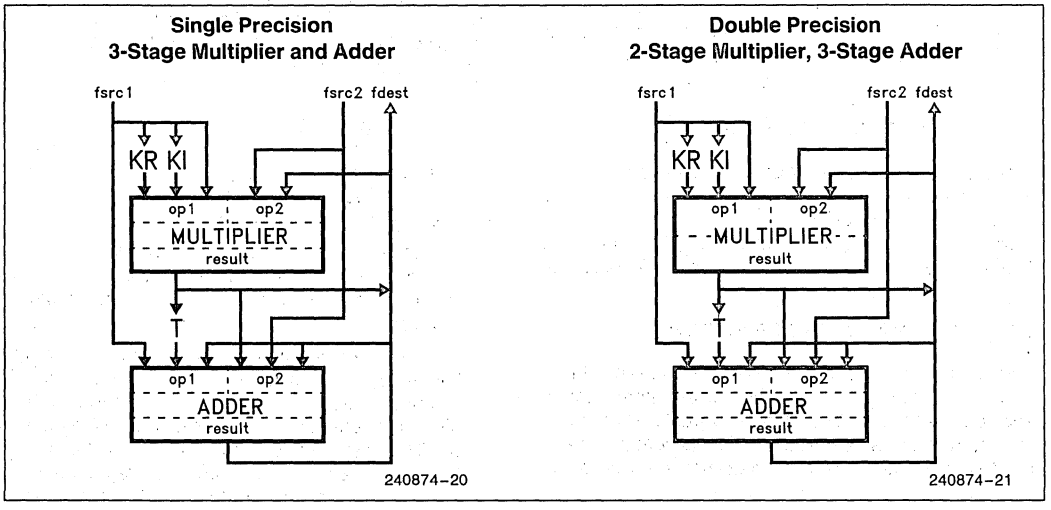


Figure 2.16. Dual-Operation Data Paths

2.7 Addressing Modes

Data access is limited to load and store instructions. Memory addresses are computed from two fields of load and store instructions: *isrc1* and *isrc2*.

1. *isrc1* either contains the identifier of a 32-bit integer register or contains an immediate 16-bit address offset.
2. *isrc2* always specifies a register.

Because either *isrc1* or *isrc2* may be null (zero), a variety of useful addressing modes result:

offset + ***register***	Useful for accessing fields within a record, where *register* points to the beginning of the record. Useful for accessing items in a stack frame, where *register* is **r3**, the register used for pointing to the beginning of the stack frame.
register + ***register***	Useful for two-dimensional arrays or for array access within the stack frame.
register	Useful as the end result of any arbitraryaddress calculation.
offset	Absolute address into the first or last 32K of the logical address space.

In addition, the floating-point load and store instructions may select autoincrement addressing. In this mode *isrc2* is replaced by the sum of *isrc1* and *isrc2* after performing the load or store. This mode makes stepping through arrays more efficient, because it eliminates one address-calculation instruction.

2.8 Traps and Interrupts

Traps are caused by exceptional conditions detected in programs or by external interrupts. Traps cause interruption of normal program flow to execute a special program known as a trap handler. Traps are divided into the types shown in Table 2.10.

2.8.1 TRAP HANDLER INVOCATION

This section applies to traps other than reset. When a trap occurs, execution of the current instruction is aborted. Except for bus error and parity error traps, the instruction is restartable. The processor takes the following steps while transferring control to the trap handler:

1. Copies U (user mode) of the **psr** into PU (previous U).
2. Copies IM (interrupt mode) into PIM (previous IM).
3. Sets U to zero (supervisor mode).
4. Sets IM to zero (interrupts disabled).
5. If the processor is in dual instruction mode, it sets DIM; otherwise it clears DIM.
6. If the processor is in single-instruction mode and the next instruction will be executed in dual-instruction mode or if the processor is in dual-instruction mode and the next instruction will be executed in single-instruction mode, DS is set; otherwise, it is cleared.
7. The appropriate trap type bits in **psr** and **epsr** are set (IT, IN, IAT, DAT, FT, OF, IL, PI, PT, BEF, PEF). Several bits may be set if the corresponding trap conditions occur simultaneously.
8. An address is placed in the fault instruction register (**fir**) to help locate the trapped instruction. In single-instruction mode, the address in **fir** is the address of the trapped instruction itself. In dual-instruction mode, the address in **fir** is that of the floating-point half of the dual instruction. If an instruction or data access fault occurred, the associated core instruction is the high-order half of the dual instruction (**fir** + 4). In dual-instruction mode, when a data access fault occurs in the absence of other trap conditions, the floating-point half of the dual instruction will already have been executed (except in the case of the **fxfr** instruction).

The processor begins executing the trap handler by transferring execution to virtual address 0xFFFFFF00. The trap handler begins execution in single-instruction mode. The trap handler must examine the trap-type bits in **psr** (IT, IN, IAT, DAT, FT) and **epsr** (OF, IL, PT, PI, BEF, PEF) to determine the cause or causes of the trap.

2.8.2 INSTRUCTION FAULT

This fault is caused by any of the following conditions. In all cases the processor sets the IT bit before entering the trap handler.

1. By the **trap** instruction. When **trap** is executed in dual-instruction mode, the floating-point companion of the **trap** instruction is not executed before the trap is taken.
2. By the **intovr** instruction. The trap occurs only if OF in **epsr** is set when **intovr** is executed. To distinguish between cases 1 and 2, the trap handler must examine the instruction addressed by **fir**. The trap handler should clear OF before returning. When **intovr** causes a trap in dual-instruction mode, the floating-point companion of the **intovr** instruction is completely executed before the trap is taken.

Table 2.10. Types of Traps

Type	Indication			Caused by	
	psr	epsr	fsr	Condition	Instruction
Instruction Fault	IT	OF		Software traps	**trap** **intovr**
		IL		Missing **unlock**	Any
		PT & PI		Pipeline usage	Any scalar or pipelined instruction that uses a pipeline
Floating Point Fault	FT		SE	Floating-point source exception	Any M- or A-unit except **fmlow**
				Floating-point result exception	Any M- or A-unit except **fmlow**, **pfgt**, and **pfeq**. Reported on any F-P instruction, **pst, fst,** and sometimes **fld, pfld,** and **ixfr**
			AO, MO	overflow	
			AU, MU	underflow	
			AI, MI	inexact result	
Instruction Access Fault	IAT			Address translation exception during instruction fetch	Any
Data Access Fault	DAT			Load/store address translation exception	Any load/store
				Misaligned operand address	Any load/store
				Operand address matches **db** register	Any load/store
Parity Error Fault	IN	PEF		Parity error on data pins during bus read operation when PEN# pin active	
Bus Error Fault	IN	BEF		External interrupt signal on BERR pin	
Interrupt	IN	INT		External interrupt signal on INT pin	
Reset	None	PEF, BEF		Hardware RESET signal	

3. By violation of lock/unlock protocol, explained below. (Note that **trap** and **intovr** should not be used within a locked sequence; otherwise, it would be difficult to distinguish between this and the prior cases.)
4. By execution of an instruction that uses a pipeline when the PT bit of **epsr** is set. (Refer to section 2.8.2.2.)

2.8.2.1 Lock Protocol

The lock protocol requires the following sequence of activities:

1. **lock**
2. Any load or store instruction. For compatibility with future processor generations, this should be a load.
3. **unlock**
4. Any load or store instruction. For compatibility with future processor generations, this should be a store.

There may be other instructions between any of these steps. The bus is locked after step 2, and remains locked until step 4. Step 4 must follow step 1 by 30 instructions or less; otherwise, an instruction trap occurs. In case of a trap, IL is also set. If the load or store instruction of step 2 accesses a previously unaccessed page (A=0), the bus is locked briefly while the A bit is set, unlocked, then locked again to satisfy the **lock** instruction and start the locked sequence.

2.8.2.2 Using PT and PI Bits

The PI and PT bits are provided to help the trap handler avoid unnecessarily saving and restoring the pipelines (refer to the section "Pipeline Preemption" in the *i860 Microprocessor Family Programmer's Reference Manual*).

Trap handlers that use PI or PT must initially examine **fsr**. If a pending trap exists—that is, if the FTE (floating-point trap enable) bit is set and any of the

floating-point exception bits (AI, AO, AU, MI, MO, MU) is active—the trap handler must save the pipelines. The i860 XP microprocessor, like the i860 XR microprocessor, may set an **fsr** exception bit before the floating-point trap is generated, and this pending trap relies on information in the pipeline. For example, an external interrupt might invoke the trap handler between the scalar floating-point instruction that produces an overflow and the next floating-point operation—the one that would cause a branch to the trap handler for the floating-point trap.

If no pending trap exists, the handler can follow either of the following two methods:

- **Using both PT and PI:** Upon invocation, the trap handler saves the state of PI and PT (in **epsr**), but does not save the pipes. If PI is found set (which means that the interrupted code needs the state information currently in the floating-point pipelines), the handler sets PT and clears PI (with a single **st.c** to **epsr** instruction), then continues with trap processing. If the pipes are used during trap handling (even by a scalar instruction), a trap will be generated with IT and PI set by hardware. The trap handler may then check PI and PT, and if both are set, clear PT, PI, and IT, save the pipes, set an indication that they were saved, and restart execution from the instruction that caused the trap. At the end of trap handling, the trap handler restores the pipes if they were saved, and restores PI and PT to their values before the trap. This method avoids both saving and restoring the pipes, assuming that most trap handling sequences do not alter the pipes, and therefore a trap for PT = 1 will not happen very often.
- **Using only PI:** Another approach is to leave PT = 0, using only the PI bit, which the processor sets each time a pipelined instruction or **pfld** is encountered (even if the floating point instruction is suppressed due to KNF = 1). The trap handler saves PI, saves the pipes if PI is set, sets an indication that they were saved, and clears PI. At the end of trap handling, the trap handler restores the pipes if they were saved, and restores PI to its value before the trap. With this method, the pipes are sometimes saved and restored unnecessarily if the trap handler code does not use the pipes. This method is advised when it is known that the trap handler uses the pipes.

2.8.3 FLOATING-POINT FAULT

The floating-point fault is reported on floating-point instructions, **pst**, **fst**, and sometimes **fld**, **pfld**, and **ixfr**. The floating-point faults of the i860 XP microprocessor support the floating-point exceptions defined by the IEEE standard as well as some other useful classes of exceptions. The i860 XP microprocessor divides these into two classes: source exceptions and result exceptions. The numerics library supplied by Intel provides the IEEE standard default handling for all these exceptions.

2.8.3.1 Source Exception Faults

All exceptional operands, including infinities, denormalized numbers and NaNs, cause a floating-point fault and set SE in the **fsr**. Source exceptions are reported on the instruction that initiates the operation. For pipelined operations, the pipeline is not advanced.

SE is undefined for faults on **fld**, **pfld**, **fst**, **pst**, and **ixfr** instructions under these conditions:

- In single-instruction mode, always.
- In dual-instruction mode, when the companion instruction is not a multiplier or adder operation.

2.8.3.2 Result Exception Faults

The result exceptions include:

- Overflow. The absolute value of the rounded true result would exceed the largest positive finite number in the destination format.
- Underflow (when FZ is clear). The absolute value of the rounded true result would be smaller than the smallest positive finite number in the destination format.
- Inexact result (when TI is set). The result is not exactly representable in the destination format. For example, the fraction 1/3 cannot be precisely represented in binary form. This exception occurs frequently and indicates that some (generally acceptable) accuracy has been lost.

The point at which a result exception is reported depends upon whether pipelined operations are being used:

- **Scalar (nonpipelined) operations.** Result exceptions are reported on the next floating-point, **fst.x**, or **pst.x** (and sometimes **fld**, **pfld**, **ixfr**) instruction after the scalar operation. When a trap occurs, the last-stage of the affected unit contains the result of the scalar operation.
- **Pipelined operations.** Result exceptions are reported when the result is in the last stage and the next floating-point (and sometimes **fld**, **pfld**, **ixfr**) instruction is executed. When a trap occurs, the pipeline is not advanced, and the last-stage results (that caused the trap) remain unchanged.

When no trap occurs (either because FTE is clear or because no exception occurred), the pipeline is ad-

vanced normally by the new floating-point operation. The result-status bits of the affected unit are undefined until the point that result exceptions are reported. At this point, the last-stage result-status bits (bits 29..22 and 16..9 of the **fsr**) reflect the values in the last stages of both the adder and multiplier. For example, if the last-stage result in the multiplier has overflowed and a **pfadd** is started, a trap occurs and MO is set.

For scalar operations, the RR bits of **fsr** report in which register the result was stored. RR is updated when the scalar instruction is initiated. The result exception trap, however, occurs on a subsequent instruction. Programmers must prevent intervening stores to **fsr** from modifying the RR bits. Prevention may take one of the following forms:

- Before any store to **fsr** when a result exception may be pending, execute a dummy floating-point operation to trigger the result-exception trap.
- Always read from **fsr** before storing to it, and mask updates so that the RR bits are not changed.

For pipelined operations, RR is cleared; the result is in the last stage of the pipeline of the appropriate unit. The trap handler must flush the pipeline, saving the results and the status bits.

In either pipelined or scalar mode, the trap handler must compute the result to be returned. In either case, the result delivered by the CPU has the same significand as the true result and has an exponent that is the low-order bits of the true result. The trap handler can inspect the delivered result, compute the result appropriate for that instruction (a NaN or an infinity, for example), and store the computed result. If RR is nonzero, the trap handler must store the computed result in the register specified by RR; if RR is zero, it must load the last stage of the pipeline with the computed result instead of the saved result.

Result exceptions may be reported for both the adder and multiplier at the same time. In this case, the trap handler should fix up the last stage of both pipelines.

2.8.4 INSTRUCTION ACCESS FAULT

This trap occurs during address translation for instruction fetches in any of these cases:

- The address fetched is in a page whose P (present) bit in the page table is clear (not present).
- The address fetched is in a supervisor mode page, but the processor is in user mode.
- The address fetched is in a page whose PTE has A = 0, and the access occurs during a locked sequence (i.e. between **lock** and **unlock**).

Note that several instructions are fetched at one time, either due to instruction prefetching or to instruction caching. Therefore, a trap handler can change from supervisor to user mode and continue to execute instructions fetched from a supervisor page. An instruction access trap occurs only when the next group of instructions is fetched from a supervisor page (up to eight instructions later). If, in the meantime, the handler branches to a user page, no instruction access trap occurs. No protection violation results, because the processor does not permit data accesses to supervisor pages while running in user mode.

2.8.5 DATA ACCESS FAULT

This trap results from an abnormal condition detected during data operand fetch or store. Such an exception can be due only to one of the following causes:

- An attempt is being made to write to a page whose D (dirty) bit is clear.
- A memory operand is misaligned (is not located at an address that is a multiple of the length of the data).
- The address stored in the debug register is equal to one of the addresses spanned by the operand.
- The operand is in a not-present page.
- An attempt is being made from user level to write to a read-only page or to access a supervisor-level page.
- The operand is in a page whose PTE has A = 0, and the access occurs during a locked sequence (i.e. between **lock** and **unlock**).
- Write protection (determined by **epsr** bit WP = 1) is violated in supervisor mode.

When a data access trap is taken on a pipelined floating-point instruction that occurs immediately after the load or store instruction that causes the trap, the destination register of the pipelined floating-point instruction may be partially updated. Correct execution will occur when the trap handler resumes execution after handling the DAT, because the pipelined floating-point instruction will then correctly update its destination register.

2.8.6 PARITY ERROR TRAP

If the PEN# pin is active and the bus unit detects a parity error during a bus read operation, the processor sets PEF and IN, then generates a trap. Further parity error traps are masked as soon as PEF is set. To reenable such traps, software must clear PEF and unfreeze BEAR by executing **ld.c bear,** *rdest*.

The interrupted program is not restartable. BS (bus or parity error trap in supervisor mode) is set by the i860 XP microprocessor when a parity error occurs while the processor is in supervisor mode. The operating system can use this bit to decide, for example, whether to abort the process (user mode) or reboot the system (supervisor mode).

2.8.7 BUS ERROR TRAP

When external hardware asserts the BERR pin, the processor sets BEF (bus error flag) and IN (interrupt), and then traps. Further BERR traps are masked as soon as BEF is set by hardware. To reenable such traps, software must clear BEF and unfreeze BEAR by executing **ld.c bear,** *rdest*.

BS (bus or parity error trap in supervisor mode) is set by the i860 XP microprocessor when a bus error occurs while the processor is in supervisor mode. The operating system can use this bit to decide, for example, whether to abort the process (user mode) or reboot the system (supervisor mode).

2.8.8 INTERRUPT TRAP

An interrupt is an event that is signaled from an external source. If the processor is executing with interrupts enabled (IM set in the **psr**), the processor sets the interrupt bit IN in the **psr** and INT in the **epsr**, then generates an interrupt trap.

Vectored interrupts are implemented by interrupt controllers and software. Software can use the **ldint** instruction to generate an interrupt acknowledge (INTA) cycle. This instruction generates a bus cycle with INTA cycle specifications, and places the data returned from the bus to the destination register. Tags are not checked in the data cache for hit, and the cycle is not burstable.

The Intel 486 microprocessor generates two INTA cycles as a response to an interrupt and inserts four idle clocks in between. To generate an interrupt acknowledge sequence that is compatible with the Intel 486 microprocessor, the **ldint** instruction sequence documented in section 5.1.4 should be executed.

2.8.9 RESET TRAP

When the i860 XP microprocessor is reset, execution begins in single-instruction mode at virtual address 0xFFFFFF00. This is the same address as for other traps. The reset trap can be distinguished from other traps by the fact that no trap bits are set. The instruction cache is flushed. The bits DPS, BL, and ATE in **dirbase** are cleared. CS8 is initialized by the value at the INT pin at the end of reset. The read-only fields of the **epsr** are set to identify the processor, while the IL, WP, and PBM bits are cleared. The bits U, IM, BR, and BW in **psr** are cleared, as are the trap bits FT, DAT, IAT, IN, and IT. All other bits of **psr** and all other register contents are **undefined**. Refer to Table 2.11 for a summary of these initial settings.

The software must ensure that the control registers are properly initialized before performing operations that depend on the values of those registers.

Reset code must initialize the floating-point pipeline state to zero with floating-point traps disabled to ensure that no spurious floating-point traps are generated.

After a RESET the i860 XP microprocessor starts execution at supervisor level (U=0). Before branching to the first user-level instruction, the RESET trap handler or subsequent initialization code has to set PU and a trap bit so that an indirect branch instruction will copy PU to U, thereby changing to user level.

2.9 Debugging

The i860 XP microprocessor supports debugging with both data and instruction breakpoints. The features of the i860 XP microprocessor architecture that support debugging include:

- **db** (data breakpoint register), which permits specification of a data address that the i860 XP microprocessor will monitor.
- BR (break read) and BW (break write) bits of the **psr**, which enable trapping of either reads or writes (respectively) to the address in **db**.
- DAT (data access trap) bit of the **psr**, which allows the trap handler to determine when a data breakpoint was the cause of the trap.
- **trap** instruction that can be used to set breakpoints in code. Any number of code breakpoints can be set. The values of the *isrc1* and *isrc2* fields help identify which breakpoint has occurred.
- IT (instruction trap) bit of the **psr**, which allows the trap handler to determine when a **trap** instruction was the cause of the trap.

Table 2.11. Register and Cache Values after Reset

Registers	Initial Vaule
Integer Registers	*Undefined*
Floating-Point Registers	*Undefined*
psr	U, IM, BR, BW, FT, DAT, IAT, IN, IT = 0; others are *undefined*
epsr	IL, WP, PBM, BE, PT = 0; BEF, PEF = 1; Processor Type, Stepping Number, DCS, SO are read only; others are *undefined*
db	*Undefined*
dirbase	DPS, BL, LB, ATE = 0; others are *undefined*
fir	*Undefined*
fsr	*Undefined*
bear	*Undefined*
p3–p0	*Undefined*
ccr	CO, DO = 0; others are *undefined*
KR, KI, T, MERGE	*Undefined*
NEWCURR	*Undefined*
STATUS	InLoop, Nested, Detached = 0
Caches	**Initial Value**
Instruction Cache	All entries invalid
Data Cache	All entries invalid
TLB	All entries invalid

3.0 ON-CHIP CACHES

By holding data, instructions, and address translation on-chip, the caches of the i860 XP microprocessor provide the following advantages:

1. Low chip count for the CPU subsystem.
2. Wide processor-to-cache path: 16 bytes for data, 8 bytes for instructions.
3. Fast access without requiring much additional high-speed design in the system. The fast (50 MHz) cache-access circuitry is hidden on chip; the external bus can respond more slowly without significantly degrading performance.

3.1 Address Translation Caches

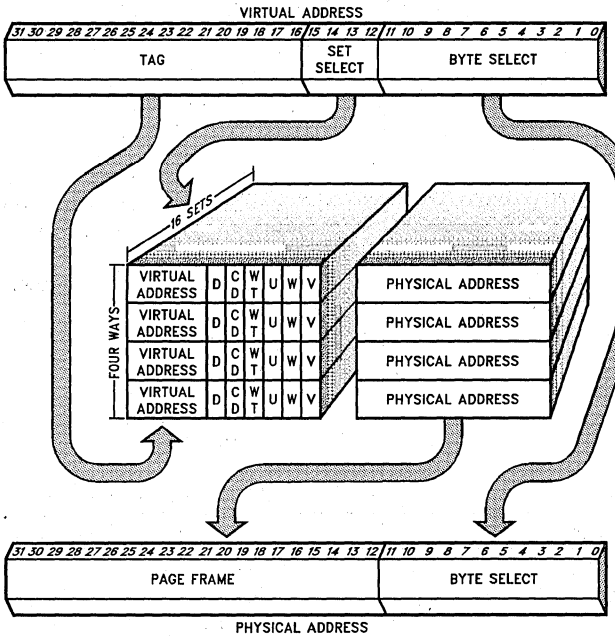
The i860 XP microprocessor allows both four Kbyte and four Mbyte page sizes, and a separate translation look-aside buffer (TLB) is used to cache address translation information for each page size. The TLB for four-Kbyte pages (Figure 3.1) has 64 entries, and the TLB for four-Mbyte pages (Figure 3.2) has 16 entries. Both are four-way set associative. The TLBs function when paging is enabled. When a page is first accessed, its translation information is saved in the appropriate TLB along with other page attributes, such as access rights and cacheability. Every address translation operation looks up the virtual address simultaneously in both TLBs. Only if the necessary paging information is not in either of the caches must the paging tables in memory be referenced. Both TLBs employ a random replacement algorithm to choose which of the four ways to replace.

If an instruction's virtual address is found in the instruction cache, the virtual address is not translated, and code access rights are not verified. However, when an instruction's virtual address is not found in the cache, address translation does occur, and all access rights are verified. The virtual addresses of data are always translated, and access rights are always verified.

The i860 XP microprocessor requires simultaneous access to data and instruction caches, but the TLBs can service only one address translation at a time. Data address translation has higher priority in the TLBs than instruction address translation, if both are required at the same time.

Any data or instruction access fault halts address translation at once, and the TLB is not updated. If a directory read causes an access fault, the page table is not read at all.

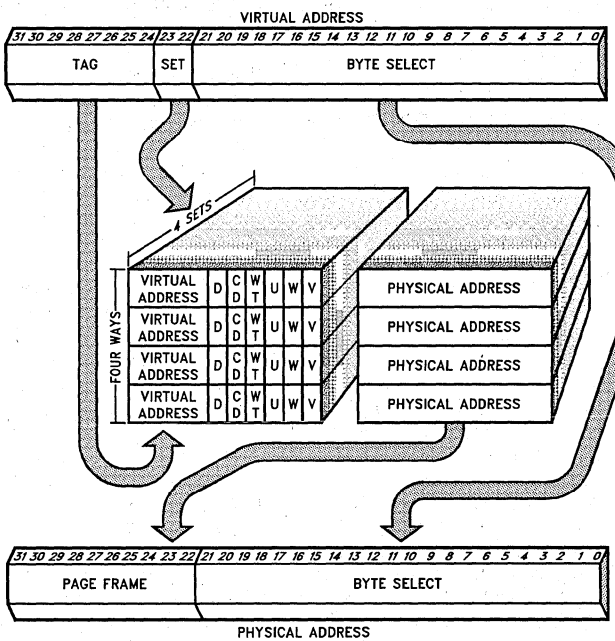
If the paging unit generates a fault (in setting the D bit for the first write to a nondirty page, for example), the corresponding entry is deleted from the TLB. Therefore, software does not need to invalidate the TLB entry in response to DAT or IAT faults.



NOTES:
D Dirty
CD Cache Disable
WT Write-Through
U User Mode
W Writable
V Validity

240874-22

Figure 3.1. 4K TLB Organization



NOTES:
D Dirty
CD Cache Disable
WT Write-Through
U User Mode
W Writable
V Validity

240874-23

Figure 3.2. 4M TLB Organization

If TLB replacement is initiated during a locked sequence generated by the **lock** instruction and if another locked sequence has to be executed to set the A-bit, the paging unit generates an access fault. This helps external hardware implement "locking by address" by preventing generation of nested lock sequences.

3.2 Internal Instruction and Data Caches

The i860 XP microprocessor has separate data and instruction caches on-chip. Having separate caches for instructions and data allows simultaneous cache look-up. Up to two instructions and 128 bits of data can be accessed simultaneously from these caches. The data and instruction caches hold 16 Kbytes each. A line can be filled from memory with a four-transfer burst.

The caches are fully transparent to applications software. Snooping (address monitoring) is designed into both instruction and data caches, to maintain cache consistency in multiprocessor systems.

Each cache has two sets of tags: *virtual* tags used for internal access, and *physical* tags used for snooping. Figure 3.3 shows how the bits of both virtual and physical addresses are mapped for caching. The presence of both virtual and physical tags supports *aliasing*, a situation in which the TLBs associate a single physical address with two or more virtual addresses.

Any area of memory can be cached, although both software and hardware can disallow certain areas from being cached—software by setting the CD bit in their page table entries; hardware by deasserting the KEN# signal for bus cycles with addresses that fall in those areas. (Data reads from the two four-Kbyte pages pointed to by the CCUBASE field of **ccr** are not cached (and the CACHE# signal is inactive), if the DCCU is activated by setting CO of the **ccr** register. This is independent of the value of KEN#.) When both software and hardware agree that a requested datum is cacheable, the i860 XP microprocessor fetches an entire 32-byte line and places it into the appropriate cache. Cache line fills are generated only for read misses, not for write misses. A store that misses the cache does not copy the missed line into cache from memory, but rather posts the datum in a write buffer, then sends it to the external bus when the bus is available.

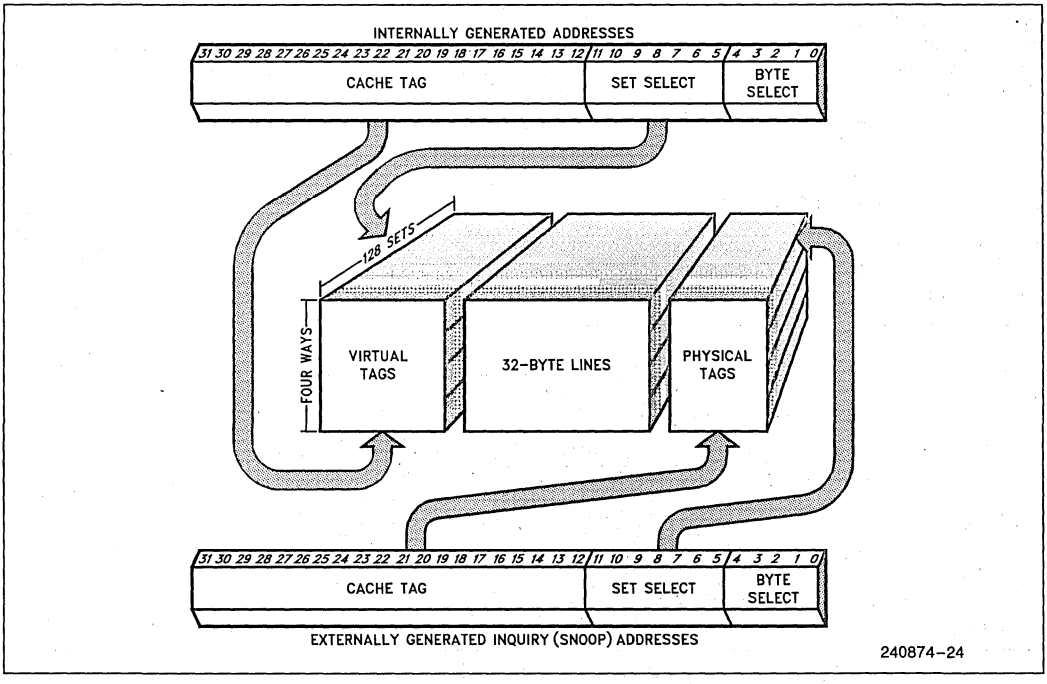


Figure 3.3. Cache Address Usage

3.2.1 DATA CACHE

Figure 3.4 shows the organization of the data cache. The data cache has two status bits per physical tag and one validity status bit for the virtual tag. A virtual tag hit is possible only when the validity bit of the virtual tag is set and the state of the physical tag is M, E, or S.

Aliasing support is built into the cache look-up algorithm. Even though a physical line may be aliased, the processor never enters the line twice in the data cache. If a virtual address is not found among the virtual tags in the data cache, a bus cycle is initiated (except a read is not issued at this time if the bus pipeline is full) and, at the same time, the physical tags are searched for the physical address (which by this time has been retrieved from the paging unit). For reads, if the physical address is found, the data returned from the bus is ignored, on-chip data is used, and the virtual tag is replaced with the new one. For writes, if a virtual address is not found, the write is issued on the bus and memory is updated. If the physical address is found, the line in cache is updated, and the virtual tag is replaced with the new one. However, the cache state (M, E, or S) of the physical-address tag does not change when the virtual tag is overwritten.

Note that the BE (big endian) bit of **epsr** has no influence on data cache behavior. Data items are kept in cache in exactly the same ordering as in external memory. Byte-shifting operations invoked by the BE bit upon loads and stores occur at the input to the register files only.

3.2.1.1 Data Cache Update Policies

To minimize bus traffic, a *write-back* policy is normally used. The write-back policy (also called *copy-back* and *deferred-write*) reduces bus traffic by eliminating many unnecessary writes. Writes to a line in the cache are not immediately forwarded to main memory; instead, they are accumulated in the cache. The modified cache line is written to main memory only when its cache space is needed for other data, when the modified data is needed by another processor, or when a flush procedure is executed.

Under the write-back policy, a write that hits the cache utilizes it for two cycles (one to check the virtual tags for hit, another to update the cache line). However, the cache pipeline allows successive store hits to operate at one per cycle. The processor's internal write buffers can hold two successive stores, preventing a freeze upon store miss.

Under a *write-through* policy, a write request to a line in the cache triggers updates to both cache and main memory. An address decoder, for example, can select the write-through policy for writes to video RAM, where it is necessary that writes be seen on the video display. Software, by setting the WT page-table bit, can select the write-through policy for specific areas of memory—those that are used for interprocessor message queues, for example.

A *write-once* policy combines write-through with write-back. Write-through is employed for the first write to a cache line, while subsequent writes to the same line follow the write-back policy. Write-once is valuable in multiprocessor systems to maintain cache consistency with the least possible bus traffic. The first write broadcasts to other processor nodes the fact that a line has been modified. Write-once is also used if a second-level cache is attached to the i860 XP microprocessor to maintain consistency between the first- and second-level caches.

The external system can dynamically change the update policy (write-back, write-through, write-once) of the i860 XP microprocessor with each cache line.

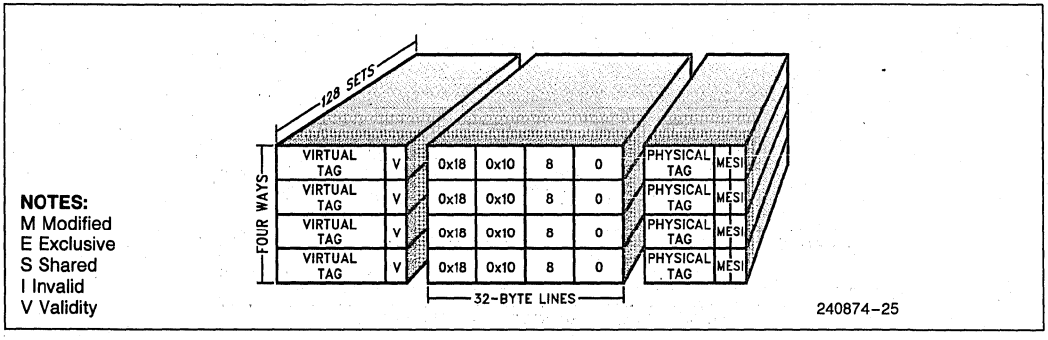


Figure 3.4. Data Cache Organization

3.2.2 INSTRUCTION CACHE

Figure 3.5 shows the organization of the instruction cache. The instruction cache has one validity bit that is common to both virtual and physical tags. Aliasing support for instructions consists not simply of changing the virtual tag, but rather fetching a line whenever a virtual tag miss occurs. If the physical address already exists in the instruction cache, its line and its tags are overwritten. So, even though a physical line may be aliased, the processor never enters the line twice in the instruction cache.

3.2.3 CACHE REPLACEMENT ALGORITHM

The data, instruction, and address-translation caches all use similar algorithms to choose which of the four cache blocks will be overwritten when a miss causes a line fetch.

First, the first invalid line (if any) in a set of four is replaced (in the order 0, 1, 2, 3). When there are no more invalid lines in a set, a pseudorandom replacement algorithm chooses which valid lines to replace. The algorithm is controlled by counters inside the chip. RESET initializes these counters to zero, so that the "randomness" is deterministic and two i860 XP CPUs executing the same code on identical boards have exactly the same series of cache hits, misses, and replacements.

Setting ITI to invalidate the caches and TLBs also resets the counters used to select the set used for cache line replacement. This brings the i860 XP microprocessor cache-replacement mechanism to a known state without resetting the whole chip.

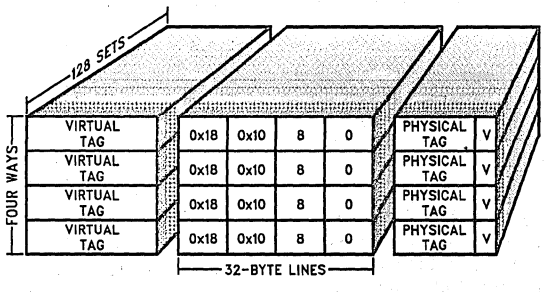
When the **flush** instruction is used to write back modified lines in the data cache, the flush routine must alter the RC (replacement control) field of **dirbase**. Therefore, replacement is not random. Instead, the block (or "way") replaced is the one selected by the RB (replacement block) field of **dirbase**.

3.2.4 CACHE CONSISTENCY PROTOCOL

The i860™ XP Microprocessor implements cache consistency via its use of a MESI (Modified, Exclusive, Shared, Invalid) protocol.

3.2.4.1 Data Cache States

Each line of the data cache of the i860 XP microprocessor can be in one of the states defined in Table 3.1. Note that the instruction cache of the i860 XP only implements the "SI" part of the MESI protocol, because the instruction cache is not writable.



NOTE:
V Validity

240874-26

Figure 3.5. Instruction Cache Organization

Table 3.1. MESI Cache Line States

Cache Line State:	M Modified	E Exclusive	S Shared	I Invalid
This cache line is valid?	Yes	Yes	Yes	No
The memory copy is . . .	. . . out of date	. . . valid	. . . valid	—
Copies exist in other caches?	No	No	Maybe	Maybe
A write to this line . . .	. . . does not go to bus	. . . does not go to bus	. . . goes to bus and updates the cache	. . . goes directly to bus

Table 3.2. Internally Initiated Cache State Transitions

State	Next State after Read	Next State after Write*
I	If WB/WT# = 1; E; else S Line fill	Write-through I
S	S	Write-through If WB/WT# = 1, E; else S
E	E	M
M	M	M

NOTE:
* "Write" does not include write-backs due to replacement. Those can only cause an M to I transition.

The state of a cache line can change as the result of either internal or external activity related to that line. Table 3.2 presents the line state transitions that result from internal activity of the i860 XP microprocessor in the data cache.

External cache-consistency support is provided through *inquiry cycles*. Inquiry cycles are initiated by other processors in a multiprocessor system to check whether an address is cached in the internal cache of the i860 XP microprocessor. Table 3.3 shows the line state transitions initiated by inquiry cycles.

Table 3.3. Inquiry-Initiated Cache State Transitions

State	INV = 0	INV = 1
I	I	I
S	S	I
E	S	I
M	S; write back the line	I; write back the line

3.2.4.2 Write-Once Policy

A write-once cache policy can be implemented through use of the WB/WT# input pin. The signal on this pin is sampled in both read and write cycles. A read miss causes a line to enter either S or E after the line fill. If WB/WT# is sampled LOW at the time of NA# or the first BRDY# activation, the line enters S state, forcing the next write hit to this line to show up on the bus. If WB/WT# is sampled HIGH, the line enters E state. In write-through cycles, the state of a line is changed from S to E when WB/WT# is sampled HIGH, so that subsequent writes will not be written through to the bus. Thus, if this signal is driven LOW on read cycles and HIGH on write cycles, a write-once cache policy is implemented. The easiest way to implement write-once (in systems not using the 82495XP cache controller) is to tie this pin to the W/R# output of the processor.

If the WT bit in the page table entry is set, the i860 XP microprocessor ignores the WB/WT# signal for the cycles that hit that page and always performs a write-through. In other words, hardware cannot override software's selection of the write-through policy.

3.2.4.3 Locked Access

Locked accesses are those data loads and stores that occur after a **lock** instruction up to and including the first load or store after the corresponding **unlock** instruction.

State transitions for locked accesses differ from those in Table 3.2 in ways that guarantee that locked accesses are seen by all processors in the system. Any locked load or store generates both a cache look-up and an external bus cycle, regardless of cache hit or miss.

1. In a locked read:
   a. If the required data is not found in the cache, the data from the bus is used. The data is placed in the cache if it is cacheable and KEN# is also asserted.
   b. If the required data is found in an unmodified (E or S) state, the data from the bus is used.
   c. If the data is found in the cache in a modified (M) state, the cached data is used, and the bus data is ignored, as long as no inquiry write-back occurs before the BRDY# of the bus cycle. If, however, an intervening inquiry write-back changes the line to S or I state, the bus data is used.
2. A locked store is forced through the cache and issued on the bus. No more data accesses occur until the last BRDY# for the store. If the store hits the internal cache, the cache update is done after the last BRDY# from the bus. Note that the line written by a locked store remains in M state in spite of the write-through to the bus, because the length of the write-through is less than the line size of 32 bytes.

Locked accesses are totally *serializing* in the sense that:

1. All loads and stores that precede the **lock** instruction are issued on the bus (if they miss the cache) before the first locked access is issued. The locked access can be issued before the last BRDY# of the prior cycle if NA# is activated in response to the prior cycle.
2. No load or store after the last locked access is issued internally or on the bus until the final BRDY# for all locked accesses.

To maximize performance, instruction fetches during the locked sequence are *not* serializing. When NA# invokes pipelining, instruction fetches may be issued while locked data fetches or stores remain on the bus.

3.3 Internal Cache Consistency

Both the instruction and the data caches can be snooped by externally generated inquiry cycles, and the result of the look-up is presented on the HIT# and HITM# output pins. These inquiry cycles help maintain consistency with caches of other processors. However, software must take care not to create inconsistencies such as the following among the internal caches (including the TLBs):

1. Changing the address space while leaving virtual-address tags from the prior space in the instruction or data cache.
2. Changing instructions in memory (or in the data cache) without changing them in the instruction cache.
3. Changing page table information in memory (or in the data cache) without changing the same information in the TLBs.

Under certain circumstances, such as I/O references, self-modifying code, page-table updates, or shared data in a multiprocessing system, it is necessary to bypass, to invalidate, or to flush the caches. The i860 XP microprocessor provides the following methods for doing this:

- **Bypassing Instruction and Data Caches.**
  1. If deasserted during cache-miss processing, the KEN# pin disables instruction and data caching of the referenced data.
  2. If the CD bit of the associated page table is set, caching of a page is disabled. The value of the CD bit is output on the PCD pin for use by external caches.
  3. If the WT bit of the associated page table is set, caching is not disabled, but writes pass through the cache. The value of the WT bit is output on the PWT pin for use by external caches. (Note that WT does not affect policy for the instruction cache, because the instruction cache is not writable. However, when an instruction from a page having the WT bit of the PTE set is placed in the data cache, the write-through policy applies just as for a data page.)
- **Invalidating Cache Entries.** Storing to the **dirbase** register with the ITI bit set invalidates each line of the instruction and address-translation caches. In the data cache, it invalidates the virtual tags, but not the physical tags.
- **Flushing the Data Cache.** The data cache is flushed by a software routine that uses the **flush** instruction. The **flush** instruction speeds up write-backs. The same effect (writing back modified lines) can be achieved with the load instruction **ld.l**, but this would be more than twice as slow—the load must first do four bus transfers to get new data, then write back the modified line. The **flush** instruction causes the write-backs without requiring a read from external memory to replace the modified line.

3.3.1 ADDRESS SPACE CONSISTENCY

In a multitasking virtual-address system, the operating system may intentionally employ aliasing, where several processes use the same physical memory while accessing it with different virtual addresses. When the operating system switches control from one process to the next, it changes the DTB field of the **dirbase** to point to a different page directory that defines the new address space. When this happens, all caches must be invalidated: the TLBs, so that the new page directory is read into the TLBs; the data and instruction caches, so that virtual addresses from the new space don't accidently match cached virtual addresses from the old space.

The caches are invalidated by setting the ITI bit when writing to **dirbase**. Invalidating the instruction cache invalidates both the physical and the virtual tags, because the instruction cache has one status (valid) bit, which is common to both physical and virtual tags. In the data cache, setting ITI does not invalidate physical tags. However, any modified lines will eventually be written back when their space is required for lines from the new address space or when external agents on the bus express a need for the modified data via inquiry cycles.

The caches are invalidated by setting the ITI bit when writing to **dirbase**. Note, however, that the operating system code that flushes the caches must be present during the flushing. Typically this code has the same virtual address for all processes.

NOTE:
The mapping of the page(s) containing the currently executing instruction, the next six instructions, and any data referenced by these instructions should not be different in the new page tables when the DTB is changed.

Enabling or disabling address translation (via the ATE bit) is similar to changing the DTB, in that the address mapping is changed. The virtual tags in the data and instruction cache must be invalidated prior to changing ATE.

3.3.2 INSTRUCTION CACHE CONSISTENCY

When software modifies a page containing instructions (as when a debugger replaces an instruction with the **trap** instruction to set a breakpoint), the instruction cache can become inconsistent for any of the following reasons:

- Because the data cache uses a write-back policy, changes to cached instruction pages do not immediately update memory.
- Changes to instructions do not automatically update the instruction cache.
- Instruction cache misses are not checked in the data cache.

Software must ensure that modified lines containing instructions are written to main memory before the instruction cache tries to read them. There are two methods for this:

1. Flush the data cache using the **flush** instruction. Note that to make the instruction cache consistent with the data cache, the data cache must be flushed *before* invalidating the instruction cache.
2. Mark all instruction pages as WT (write through) so that modifications to instructions are immediately written to memory. This is the better alternative.

In either case, the instruction cache must be invalidated (by a store to **dirbase** with ITI set) after a code page has been modified, so that the updated instructions will be read from memory.

3.3.3 PAGE TABLE CONSISTENCY

When the operating system modifies page tables or directories, the TLBs can become inconsistent with the modifications for any of the following reasons:

- Because the data cache uses a write-back policy, updates to cached page tables do not immediately update memory.
- Changes to page tables do not automatically update the TLB.
- The i860 XP microprocessor searches only external memory for page directories and page tables in the translation process. The data cache is not searched. (Data is not transferred from the data cache to the TLBs during TLB replacement cycles.)

Software must ensure that modified lines containing page table entries are written to main memory before the paging unit tries to read them. There are two methods for this:

1. Keep page tables and directories in noncacheable memory or write-through pages.
2. Flush the data cache using the **flush** instruction.

The processor itself invalidates the affected TLB entry, when a trap is triggered by the need to set the A or D bit. In other cases, after a page table or directory has been modified, software must invalidate the TLBs (by a store to **dirbase** with ITI set) so that the updated entries will be read from memory.

The data cache does not need flushing if the program is modifying only the P, U, W, A, or D bits of a PTE (as long as the page frame address is not changed and the PTE itself is not in the data cache.) The i860 XP CPU does not use the TLB for cache line write-backs; it writes to the address in the physical tag.

Thus, a trap handler can service a data access trap for D-bit zero merely by setting D = 1. When setting the P or A bits, there is no need to invalidate or flush any caches, because the processor does not load entries into the TLB that have P = 0 or A = 0.

Two potential TLB inconsistencies are avoided automatically by the i860 XP microprocessor.

1. If the paging unit issues a write cycle (to set the A bit, for example), this cycle is snooped by the data cache for invalidation.
2. Any TLB entry that causes a DAT or IAT is automatically invalidated.

3.3.4 CONSISTENCY OF CACHEABILITY

Normally, an operating system ensures that the page attributes (CD and WT) of a memory access are consistent with the cache contents. However, the operating system can fail to maintain consistency by the following actions:

- Changing the CD or WT bits while related lines are in the cache.
- Aliasing a physical address with virtual addresses that have differing CD or WT bits.

In these situations, the i860 XP microprocessor gives priority to cache state. For example:

1. If a read or write request is to a noncacheable page (CD = 1), but the data (or code) is found in cache, the request is satisfied by the cache, and no external cycle is issued.
2. If the physical address of a read or write request hits in the cache but the virtual address misses, the virtual tag is overwritten by the new virtual address, but the CD bit of the new virtual address is ignored.
3. If a store to a write-through page (WT = 1) hits a cache line in E or M state, no write-through cycle is issued; only the cache is updated.

3.3.5 LOAD PIPE CONSISTENCY

The **pfld** (pipelined floating-point load) instruction facilitates transfer of data from memory to registers, and avoids placing data in the data cache. When large amounts of data are used, **pfld** allows the programmer to keep rarely-used data out of the cache. The i860 XP microprocessor ensures consistency between cached data and **pfld** references. It checks the data cache and, upon a data cache hit to a modified line, forwards data from cache into the three-stage **pfld** pipeline.

3.3.6 SUMMARY

Table 3.4 summarizes flush and invalidation requirements, assuming that WT is set in the PTEs of instruction and page-table pages:

Table 3.4. Summary of Cache Flushing And Invalidation

Action	Flush Data Cache	Invalidate Caches (ITI)
Setting A	No	No
Setting P	No	No
Clearing P	No	Yes
Setting D	No	No
Changing protection (U,W)	No	Yes
Setting CD or WT	Yes	Yes
Changing PFA in a used(1) PTE	No	Yes
Changing **dirbase** DTB	No	Yes
Changing **dirbase** ATE	No	Yes
Changing **epsr** WP	No	No
Setting **ccr** DO and CO	Yes(2)	Yes(2)
Modifying code	No(3)	Yes

NOTES:
1. "Used" means a PTE that at some past time had P set.
2. If data from either of the CCU pages could have been cached.
3. Assuming all instructions and their page directories and page tables are in write-through or noncacheable pages.

4.0 HARDWARE INTERFACE

In the following description of hardware interface, the # symbol at the end of a signal name indicates that the active or asserted state occurs when the signal is at a low voltage. When no # is present after the signal name, the signal is asserted when at the high voltage level.

4.1 Pins Overview

Figure 4.1 identifies functional groupings of the pins. Table 4.1 lists every pin by its identifier, gives a brief description of its function, and lists some of its characteristics. All output pins are tristate, except BREQ, HIT#, HITM#, HLDA, LOCK#, and PCHK#.

Table 4.1. Pin Summary

Pin ID	Name	Active Level	When Floated Synch/Asynch	Internal Resistor
		Output Pins		
ADS#	Address Status	LOW	HLDA, clock after BOFF#	
BE7#–BE0#	Byte Enable	LOW	HLDA, BOFF#	
BREQ	Bus Request	HIGH		
CACHE#	Cache	LOW	HLDA, BOFF#	
CTYP	Cycle Type	HIGH	HLDA, BOFF#	
D/C#	Data/Code		HLDA, BOFF#	
HIT#	Snoop Hit Cache	LOW		
HITM#	Snoop Hit Modified Line	LOW		
HLDA	Hold Acknowledge	HIGH		
KB0,KB1	Cache Block	HIGH	HLDA, BOFF#	
LEN	Length	HIGH	HLDA, BOFF#	
LOCK#	Address Lock	LOW		
M/IO#	Memory/IO		HLDA, BOFF#	
NENE#	Next Near	LOW	HLDA, BOFF#	
PCD	Page Cache Disable	HIGH	HLDA, BOFF#	
PCHK#	Parity Check	LOW		
PCYC	Page Cycle	HIGH	HLDA, BOFF#	
PWT	Page Write-Through	HIGH	HLDA, BOFF#	
TDO	Test Output		Nonscan Mode	
W/R#	Write/Read		HLDA, BOFF#	
		Input/Output Pins		
A31–A3	Address	HIGH	AHOLD, HLDA, BOFF#	
D63–D0	Data	HIGH	HLDA, BOFF#	
DP7–DP0	Data Parity	HIGH	HLDA, BOFF#	
		Input Pins		
AHOLD	Address Hold	HIGH	Synch	
BERR	Bus Error	HIGH	Synch	
BOFF#	Back-Off	LOW	Synch	
RSRVD#	*Intel Reserved*			
BRDY#	Burst Ready	LOW	Synch	
BYPASS#	*Intel Reserved*	LOW		
CLK	Clock			
RESET	Reset	HIGH	Asynch	
EADS#	External Address Status	LOW	Synch	
EWBE#	External Write Buffer Empty	LOW	Synch	
FLINE#	Flush Line	LOW	Synch	
HOLD	Bus Hold	HIGH	Synch	
INT/CS8	Interrupt/Code-Size 8	HIGH	Asynch	
INV	Invalidate	HIGH	Synch	
KEN#	Cache Enable	LOW	Synch	
NA#	Next Address	LOW	Synch	
PEN#	Parity Enable	LOW	Synch	
TCK	Test Clock			
TDI	Test Data Input		Synch	Pull-up
TMS	Test Mode Select		Synch	Pull-up
TRST#	Test Reset	LOW	Asynch	Pull-up
WB/WT#	Write-Back/Write-Through		Synch	
SPARE	*Intel Reserved*			

The pins D/C#, W/R#, and M/IO# define bus cycle types. They are summarized in Table 4.2. For data transfers to or from memory, two additional pins, CTYP and PCYC, provide further information regarding the type of transfer, as shown in Table 4.3. Table 4.4 shows how the LEN and CACHE# pins determine cycle length.

Table 4.2. ADS# Initiated Bus Cycle Definitions

M/IO#	D/C#	W/R#	Bus Cycle Initiated
0	0	0	Interrupt Acknowledge
0	0	1	Special Cycle
0	1	0	I/O Read
0	1	1	I/O Write
1	0	0	Code Read
1	0	1	Reserved
1	1	0	Memory Read
1	1	1	Memory Write

Table 4.3. Memory Data Transfer Cycle Types

PCYC	CTYP	W/R#	Data Transfer Type
0	0	0	Normal read
0	1	0	Pipelined load (**pfld** instruction)
1	0	0	Page directory read
1	1	0	Page table read
0	0	1	Write-through (S-state hit)
0	1	1	Store miss or write-back
1	0	1	Page directory update
1	1	1	Page table update

NOTE:
PCYC and CTYP are defined only for memory data transfer cycles (D/C# = 1, M/IO# = 1)

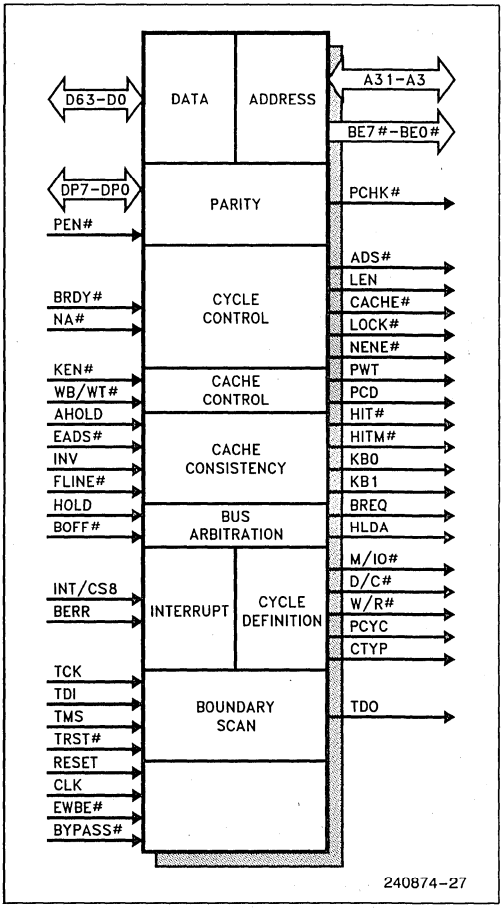


Figure 4.1. Signal Grouping

Table 4.4. Cycle Length Definition

W/R#	LEN	CACHE#	KEN#	Cycle Description	Burst Length
0	0	1	—	Noncacheable** 64-bit (or less) read	1
0	0	—	1	Noncacheable 64-bit (or less) read	1
1	0	1	—	64-bit (or less) write	1
—	0	1	—	I/O and Special Cycles	1
0	1	1	—	Noncacheable 128-bit read **(p)fld.q**	2
0	1	—	1	Noncacheable 128-bit read **(p)fld.q**	2
1	1	1	—	128-bit write **fst.q**	2
0	—	0	0	Cache line fill	4
1	—	0	—	Cache write-back	4

NOTE:
** Includes CS8-mode code fetches, which may be cached by the processor.
—Indicates "don't care" values.

4.2 Signal Description

In this section descriptions of all pins are presented in alphabetical order.

4.2.1 A31–A3 (ADDRESS PINS)

The 29-bit address bus (A31–A3) identifies addresses to a 64-bit location. Separate byte-enable signals (BE7#–BE0#) identify which bytes should be accessed within the 64-bit location.

The address lines are bidirectional. The i860 XP microprocessor drives the address lines unless it is in a hold state. The system drives address lines A31–A5 to perform cache line inquiries (refer to the EADS# signal description).

4.2.2 ADS# (ADDRESS STATUS)

The i860 XP microprocessor asserts ADS# to identify the first clock period of each bus cycle, the clock period during which new values become valid on the address bus and cycle-definition pins. This signal is held active for one clock.

If BOFF# is asserted, the processor floats ADS# two clocks after sampling BOFF# (and not, like all other pins, on the next clock). This is to ensure that ADS# is deasserted before it floats, and therefore is never left floating active.

ADS# can be asserted while AHOLD is active to initiate a cache write-back cycle.

4.2.3 AHOLD (ADDRESS HOLD)

The external system asserts AHOLD to perform a cache inquiry. In response to assertion of AHOLD, the i860 XP microprocessor immediately (in the next clock) stops driving the address bus (A31–A3 lines). The other buses remain active, and data can be transferred for previously issued read or write bus cycles during address hold. AHOLD is recognized even during RESET and LOCK#. The earliest that AHOLD can be deasserted is the clock after EADS# is asserted to start the inquiry.

If HITM# has activated due to an inquiry, the i860 XP microprocessor asserts ADS# while AHOLD is active to start the write-back of the modified line that was the target of the inquiry.

4.2.4 BE7#–BE0# (BYTE ENABLES)

The byte-enable pins are driven with the address. BE7# applies to D63–D56, BE0# applies to D7–D0.

In write cycles (noncacheable writes as well as cache line write-backs), the BE*n*# signals determine which bytes must be written into external memory for the current cycle.

In read cycles, the BEn# values indicate which byte the load instruction has requested. In all noncacheable read cycles (CACHE# or KEN# deasserted), the byte enables match the length and address of the requested data. Cacheable read cycles (KEN# asserted), however, result in four 64-bit memory transfers to fill an entire 32-byte cache line. The BE*n*# pins activated are those that represent the operand of the load instruction that caused the line fill, and these same BE*n*# pins remain activated for as long as A31–A5. All 64 bits must be returned for each cacheable cycle without regard for the BE*n*# signals.

While in CS8 mode, BE2#–BE0# serve as (active-high) lower-order address bits for instruction fetches (from the ROM). Data fetches and stores are not affected by CS8 mode, and BE2#–BE0# retain their normal byte-enable function for data.

4.2.5 BERR (BUS ERROR)

This is a nonmaskable interrupt input, which supports bus error handling or other urgent circumstances. BERR is not masked by the IM bit of the **psr** nor by lock cycles. When BERR is activated, the i860 XP microprocessor vectors to the trap handler and sets the bus error flag (BEF) in the **epsr**. BERR causes the physical address of the current bus cycle to be latched into the BEAR control register; thus, if asserted the clock of BRDY# or the clock after BRDY#, it causes the bus address to be latched for software to examine. BERR is rising-edge sensitive. Once the trap has occurred, further BEF traps cannot occur until software has cleared BEF and read BEAR.

BERR does not terminate outstanding bus cycles. Therefore, the system must still activate BRDY# a sufficient number of times or activate BOFF# for those cycles. Even though activating BOFF# temporarily halts the erring cycles, the i860 XP microprocessor will retry them when BOFF# is deasserted, in spite of BERR.

Timing of BERR is not influenced by late back-off mode.

4.2.6 BOFF# (BACK-OFF)

The system can assert this signal to abort all outstanding bus cycles that have not yet completed. In response to BOFF#, the i860 XP microprocessor

immediately (in the next clock) floats its bus, except for ADS#, which is floated one clock later. The processor floats all the same pins normally floated during bus hold; however, unlike a bus hold, HLDA is not asserted. (HLDA is asserted only in response to HOLD; no acknowledgment is required for BOFF#.). Any data and BRDY# returned to the processor while BOFF# is asserted are ignored. The processor remains in bus hold until BOFF# is deasserted, at which time it restarts the bus cycles by driving the address and cycle definition pins and asserting ADS#. When BOFF# deactivates, ADS# may be asserted the following clock. Thus a BOFF# duration of one clock results in not floating ADS# at all. BOFF# cannot be used to force the pins to float during RESET; use HOLD for that purpose.

4.2.7 BRDY# (BURST READY)

The input BRDY# indicates either that the external system has driven valid data on the data pins in response to a read request or that the external system has latched the data in response to a write request. The CPU ignores this signal when no bus requests are outstanding. During a bus cycle, BRDY# is sampled at each clock, starting with the clock after assertion of ADS# and continuing until all data for the cycle has been transferred. When BRDY# is sampled active in a read cycle, the data present on the pins is sampled.

4.2.8 BREQ (BUS REQUEST)

BREQ allows the i860 XP microprocessor to share the local bus with other bus masters. An external bus arbiter can use BREQ to implement an "on demand only" policy for granting the bus to the i860 XP microprocessor. The i860 XP microprocessor asserts BREQ the clock after it realizes an internal request for the bus. The system should sample this pin only when the i860 XP microprocessor is not in control of the bus (that is, when HLDA, BOFF#, or AHOLD is active). BREQ is undefined when the i860 XP microprocessor is driving the bus. BREQ may be deasserted between assertions of ADS#, but this does not imply that the CPU does not need the bus.

4.2.9 BYPASS# (BYPASS)

This pin is reserved by Intel Corporation and should be tied HIGH to V_{CC} through a resistor. When LOW, the phase-locked loop that generates the internal clock is unused. In this case, the internal clock has more skew relative to the external CLK, and the A.C. timing parameters are not guaranteed.

4.2.10 CACHE# (CACHEABILITY)

This output signal indicates internal cacheability of a bus request. Its timing follows that of the address bus.

The i860 XP microprocessor asserts CACHE# for cacheable reads and code fetches to announce its intention to cache the data. If CACHE# is asserted on a read cycle and if the KEN# input is active, the cycle is a burst line fill. If CACHE# is inactive in a read cycle, the i860 XP microprocessor does not cache the returned data, regardless of the KEN# pin. CACHE# is also asserted for cache line write-backs.

CACHE# is inactive for noncacheable reads (for example, **pfld, ldio, ldint**), TLB replacements, and store misses.

Table 4.4 shows how cacheability determines the number of data transfers in a cycle.

Note that the CACHE# output is always inactive for CS8 (Code-Size 8 bits) mode instruction fetches so that the instructions are fetched with single-transfer cycles. However, the code fetched may then be placed in the instruction cache, unless KEN# was inactive.

4.2.11 CLK (CLOCK)

The CLK input determines execution rate and timing of the i860 XP microprocessor. External timing parameters are specified relative to the rising edge of this signal. The i860 XP microprocessor can utilize a clock rate of 50 Mhz. The internal operating frequency is the same as the external clock. This signal requires TTL levels.

4.2.12 CTYP (CYCLE TYPE)

CTYP is one of the bus cycle definition signals. Tables 4.2 and 4.3 show the types of bus cycle generated. CTYP is defined only for data write and read requests. The value of this pin changes only when ADS# is asserted.

4.2.13 D/C# (DATA/CODE)

D/C# specifies whether the current request is for data or instructions. The data/code line is one of the bus cycle definition pins. Tables 4.2 and 4.3 show the types of bus cycle generated. The value of this pin changes only when ADS# is asserted.

4.2.14 D63–D0 (DATA PINS)

The bus interface has 64 bidirectional data pins (D63–D0) to transfer data in eight- to 64-bit quantities. Pins D7–D0 transfer the least significant byte; pins D63–D56 transfer the most significant byte. In read cycles, all 64 bits of the data bus are latched, even in CS8-mode instruction fetches when only the low-order eight bits are used. In write cycles, the i860 XP microprocessor does not drive D63–D0 in the clock of ADS#, but in the following clock.

4.2.15 DP7–DP0 (DATA PARITY)

There is one parity signal for each byte of the data bus. They are driven by the i860 XP microprocessor with even parity information on writes with the same timing as write data. Likewise, if parity checking is enabled by PEN#, the system must drive even parity information on these pins with the same timing as read information to ensure that the correct parity check status is indicated by the i860 XP microprocessor. "Even parity" means that the total number of set bits in a byte, including the parity bit, is even. Refer also to the PCHK# signal.

4.2.16 EADS# (EXTERNAL ADDRESS STATUS)

This signal indicates that a valid external address has been driven onto address pins A31–A5 of the i860 XP microprocessor to be used for a cache inquiry. This signal is recognized while the processor is in hold (HLDA is driven active), while forced off the bus with BOFF# input, or while AHOLD is asserted. The i860 XP microprocessor ignores EADS# at all other times. EADS# is not recognized if HITM# is active, nor during the clock after ADS#, nor during the clock after a valid assertion of EADS#. Table 4.5 shows when EADS is first sampled. It is then sampled in every clock as long as the hold remains active and HITM# remains inactive.

Table 4.5. EADS# Sample Time

Trigger	EADS# First Sampled
AHOLD	Second clock after AHOLD asserted
HOLD	First clock after HLDA asserted
BOFF#	Second clock after BOFF# asserted

INV and FLINE# are sampled in the same clock period that EADS# is validly asserted. HIT# and HITM# may be asserted as the results of a cache inquiry.

4.2.17 EWBE# (EXTERNAL WRITE BUFFER EMPTY)

At RESET, the value on EWBE# determines the ordering mode. The processor enters strong ordering mode if EWBE# is sampled active for at least the last three clocks before RESET deactivates; otherwise, it enters weak ordering mode.

In weak ordering mode, the value of EWBE# after reset does not affect processor operation.

In strong ordering mode, the external system asserts EWBE# as long as all external write buffers are empty. If an external write buffer is not empty (EWBE# deasserted) or the internal write buffer is not empty, the processor delays data cache updates so as to keep the external order of writes the same as the programmed order.

In systems that do not have external write buffers, EWBE# can be tied to V_{SS}, if strong ordering is desired, or to V_{CC}, if weak ordering is acceptable. Refer to sections 5.3.3 and 5.3.4 for more explanation and for other ways to control write ordering.

4.2.18 FLINE# (FLUSH LINE)

The system asserts FLINE# to request that the i860 XP microprocessor write back a modified cache line before other outstanding bus cycles are completed, if the line is hit by an external inquiry. If this pin is active in the same clock that EADS# is asserted, the write-back cycle is initiated, and the i860 XP microprocessor expects BRDY#s for the write-back before outstanding cycles (if any) are returned. If data transfer for another cycle is currently in progress when FLINE# is asserted (i.e. first BRDY# returned before HITM# asserted), the i860 XP microprocessor waits until the data transfers for that burst have completed, and only then does it assert the ADS# for the write-back. If the first BRDY# has not yet occurred for an outstanding cycle, NA# must be activated to trigger ADS# for the write-back.

At RESET, the value on FLINE# determines configuration. The processor enters one-clock late backoff mode if FLINE# is sampled active for at least the last three clocks before RESET deactivates.

4.2.19 HIT# (CACHE INQUIRY HIT)

This pin is one output of inquiry cycles. If an inquiry cycle hits a valid line in the caches of the i860 XP microprocessor (either data or instruction), HIT# is asserted two clocks after EADS# is activated. If the inquiry cycle misses the caches, this pin is negated two clocks after EADS# activation.

This pin changes its value only as a result of EADS# activation during AHOLD, HOLD, or BOFF# and retains its value until two clocks after the next valid activation of EADS#.

HIT# can be used to control the WB/WT# pin of other processors in a multiprocessor system. Activation of HIT# indicates that the inquiring processors should cache the line as S-state, not E-state.

4.2.20 HITM# (HIT MODIFIED LINE)

This pin is an output of inquiry cycles. When an inquiry hits a modified line in the internal data cache, the i860 XP microprocessor asserts HITM# two clocks after EADS# is activated. (Refer also to the EADS# signal.) The HITM# signal stays active until the last BRDY# for the corresponding write-back cycle. At all other times, HITM# is inactive. HIT# is also asserted when HITM# is asserted (except for the special case of an inquiry after the ADS# of a write-back).

4.2.21 HLDA (BUS HOLD ACKNOWLEDGE)

The i860 XP microprocessor activates HLDA in response to a hold request presented on the HOLD pin. Assertion of HLDA indicates that the i860 XP microprocessor has given the bus to another local bus master. It is driven active in the same clock that the i860 XP microprocessor floats its bus. All output pins are floated except LOCK#, BREQ, HLDA, PCHK#, HIT#, and HITM#.

The time required to acknowledge a hold request is one clock plus the number of clocks needed to finish any outstanding bus cycles (maximum of four outstanding cycles of four burst transfers each for total of 16 transfers). If this hold latency is too long for a given application, BOFF# can be used instead.

When leaving a bus hold, the i860 XP microprocessor deactivates HLDA and, in the same clock period, initiates a pending bus cycle, if any.

4.2.22 HOLD (BUS HOLD)

This pin, along with the output signal HLDA, is used for local bus arbitration. At some time after the HOLD signal is asserted, the i860 XP microprocessor releases control of the local bus and puts most bus interface outputs in floating state, then asserts HLDA—all during the same clock period. It maintains this state until HOLD is deasserted. Instruction execution stops only if required instructions or data cannot be read from the on-chip instruction and data caches. The i860 XP microprocessor ignores HOLD until all outstanding bus cycles are complete (until the last BRDY#). The i860 XP microprocessor recognizes HOLD even during RESET and LOCK#. HOLD cannot be used when the 82495XP cache controller is attached.

4.2.23 INV (INVALIDATE)

The external system asserts this signal to invalidate the cache-line state in the case of an inquiry cycle hit. It is sampled together with A31–A5 in the clock EADS# is active.

4.2.24 INT/CS8 (INTERRUPT/CODE-SIZE EIGHT BITS)

This input, like the BERR input, allows interruption of the current instruction stream. The processor samples INT as instruction boundaries. If interrupts are enabled (IM set in **psr**) when INT is sampled active, the i860 XP microprocessor fetches the next instruction from virtual address 0xFFFFFF00. INT is level triggered. To assure that an interrupt is recognized, INT should remain asserted until the software acknowledges the interrupt (by executing an interrupt-acknowledge cycle, for example). The interrupt may be ignored by the processor if the INT signal does not remain active.

Interrupt latency (the maximum time between assertion of INT and execution of the first instruction of the trap handler) depends both on the internal context and on the external system. After INT is asserted, the i860 XP microprocessor finishes all instructions currently being executed, including any outstanding bus cycles, before starting the trap handler. The following instruction sequence is an example of the worst case:

```
pfld.q
pfld.q
ld.l
br
ld.l
st.l
```

If INT is asserted during the execution stage of the last **ld.l** instruction, the execution of the trap handler may have to wait for:

- Two 2-transfer bursts (the **pfld** instructions)
- Two data cache line fills (misses by the **ld.l** instructions)
- Two data cache line write-backs (eliminating modified lines to open space for the fills)
- Two instruction cache line fills (the target of the **br** and the first instruction of the trap handler)
- Three TLB miss sequences of up to six nonpipelined accesses each (the **br**, the last **ld.l**, and the trap handler)

The time to finish the above bus activities can be extended by inquiry cycles and associated write-backs initiated by an external cache or bus controller.

Besides the bus-related delays, the i860 XP microprocessor has internal freeze conditions that can delay interrupt response by up to 10 additional clocks.

During a locked sequence, the INT pin is ignored, and the INT bit of **epsr** reflects the value on the INT pin. To limit the time that INT is ignored, the **lock** instruction can assert LOCK# for only 30–33 instructions before trapping.

This input is asynchronous, but appropriate setup and hold times must be met to insure recognition on any specific clock.

If INT is asserted for at least the last three clock periods before the falling edge of RESET, the i860 XP microprocessor enters eight-bit code-size (CS8) mode.

4.2.25 KB0, KB1 (CACHE BLOCK)

For reads, these output signals define which cache block (line) is going to receive the data. For write-backs, these lines specify which block is being flushed. They are driven together with cycle definition for cacheable data reads, TLB replacement, code fetch cycles, and write-backs. External hardware can use these signals to observe changes to cache blocks.

4.2.26 KEN# (CACHE ENABLE)

The i860 XP microprocessor samples KEN# to determine whether the data being read for the current cache-miss cycle is to be cached. When the i860 XP microprocessor generates a read cycle that can be cached (CACHE# output active) and KEN# is active, the cycle is transformed into a burst line fill. By activating KEN#, the memory system commits to a four-transfer burst. The entire 64 bits of the data bus are used for the read, regardless of the state of the byte-enable pins.

If KEN# is sampled inactive, code fetches are not transferred in bursts, but 128-bit data items may still be transferred with a burst length of two.

KEN# is sampled together with NA# or BRDY#, whichever comes first. It is sampled only with the *first* BRDY# of a burst; its value at any other time has no effect.

4.2.27 LEN (DATA LENGTH)

The LEN output pin specifies the number of burst transfers for each cycle. This pin and the CACHE# output pin are used by the system to determine the burst length for each cycle (refer to Table 4.4). The i860 XP microprocessor can generate 1, 2, or 4-transfer bursts for reads and writes.

LEN is inactive if the internal request is for 64 bits or less. If LEN is active, the internal request is for 128 bits or more, and the cycle should be returned as a two- or four-transfer burst. LEN is always active for 128-bit data accesses. LEN is always inactive for code accesses.

A cacheable read (CACHE# active) can be automatically converted to a four-transfer burst regardless of LEN by assertion of KEN#.

Table 4.4 summarizes different cycle lengths as they are calculated from the LEN and CACHE# signals. LEN has the same timing as the address.

4.2.28 LOCK# (ADDRESS LOCK)

This signal is used to provide atomic (indivisible) read-modify-write sequences in multiprocessor systems. The address to be locked is the one being driven on A31–A3 when LOCK# is activated. A multiprocessor bus arbiter must permit only one processor a locked read, locked write, or unlocked write to that address and must maintain the lock of that location across cycle boundaries until LOCK# deactivates. The simplest arbitration hardware can just lock the entire bus against all other accesses during LOCK# assertion; however, software must never assume that this implementation is being used.

The i860 XP microprocessor coordinates the external LOCK# signal with the **lock** and **unlock** instructions. Programmers do not have to be concerned about the fact that bus activity is not always synchronous with instruction execution. LOCK# is asserted with ADS# for the address operand of the first load or store instruction executed after the **lock** instruction.

After an **unlock** instruction, LOCK# is deasserted with the next load or store. The i860 XP microprocessor deactivates LOCK# one clock after ADS# for the last locked bus cycle. Unlike the i860 XR microprocessor, the i860 XP microprocessor does not deassert LOCK# immediately when a trap occurs. Instead, the trap handler must execute a load or store instruction to deassert LOCK#. (The handler does not have to execute an **unlock** instruction, however. The unlocking function is performed by the processor's trap logic.)

The i860 XP microprocessor also asserts LOCK# during TLB miss processing for updates of the accessed bit in page-directory and page-table entries. The maximum time that LOCK# can be asserted in this case is the time required to perform a nonpipelined, four-byte, read-modify-write sequence.

Between locked sequences, at least one cycle of no LOCK# is guaranteed by the behavior of the **unlock** instruction.

Between **lock** and **unlock** instructions, the INT pin is ignored.

Instruction fetches do not alter the LOCK# signal.

4.2.29 M/IO# (MEMORY-I/O)

M/IO# specifies whether the current cycle is for the memory address space or for the I/O address space. M/IO# is one of the bus cycle definition pins. Tables 4.2 and 4.3 show the types of bus cycle generated. The value of this pin changes only when ADS# is asserted.

4.2.30 NA# (NEXT ADDRESS REQUEST)

NA# makes address pipelining possible. The system asserts NA# for at least one clock to indicate that it is ready to accept the next address from the i860 XP microprocessor. (If the system does not implement pipelining, NA# must not be activated.) The i860 XP microprocessor samples NA# every clock, starting one clock after the activation of ADS#. If the i860 XP microprocessor has a new cycle pending internally when NA# is activated, it initiates that cycle in the clock after NA# is asserted. Up to three bus cycles can be outstanding simultaneously.

NA# is latched internally; the i860 XP microprocessor remembers that NA# was asserted until it has an internal request to send to the bus; so, assertion of NA# for a single clock can trigger an ADS# several clocks later. NA# is ignored in the clock of ADS#.

KEN# and WB/WT# inputs for the current cycle are sampled with NA#, if NA# is asserted before the first BRDY# of the current cycle.

NA# is also used in conjunction with FLINE# to invoke write-back of a modified line during outstanding bus cycles.

4.2.31 NENE# (NEXT NEAR)

The i860 XP microprocessor asserts NENE# when the current address is in the same DRAM page as the previous bus cycle. This signal allows higher-speed reads and writes in the case of consecutive accesses to static column or page-mode DRAMs. The i860 XP microprocessor determines the DRAM page size by inspecting the software-controlled DPS field in the **dirbase** register. The page size can range from 2^9 to 2^{16} 64-bit words, supporting DRAM sizes from 256K $\times$ 1 to 4G $\times$ *n*. The value of this pin changes only when ADS# is asserted. NENE# is never asserted for the next bus cycle after the address bus has been floating (after AHOLD, BOFF#, or HLDA is deasserted).

4.2.32 PCD (PAGE CACHE DISABLE)

PCD provides a cacheability indication on a page by page basis. This signal, together with PWT, is set to an attribute bit in the page table entry for the current cycle. When paging is enabled, PCD corresponds to the CD bit (bit 4) of the page table entry. The i860 XP microprocessor does not perform a cache fill to any page for which CD of the page table entry is set. When paging is disabled, or for any cycle that is not paged (**ldio, stio, ldint, scyc**), the i860 XP microprocessor drives PCD inactive.

During TLB miss processing, PCD is inactive while the address translation hardware is accessing the first level page directory. During accesses to the second-level page-table entry, PCD reflects the CD values taken from the first level page-table entry.

The value of this pin changes only when ADS# is asserted.

4.2.33 PCHK# (PARITY CHECK)

This output shows the result of the parity check on data pins in the previous clock of a read cycle. It is

asserted for one clock when incorrect parity has been detected. It reflects the parity status for the entire data bus.

PCHK# does not terminate outstanding bus cycles, so the system must still activate BRDY# a sufficient number of times or activate BOFF# for those cycles. PCHK# is always inactive after any code fetch in CS8 mode.

4.2.34 PCYC (PAGE CYCLE)

The page cycle line is active during memory read or write cycles to distinguish page-table accesses from other accesses. The types of bus cycle generated are indicated in Tables 4.2 and 4.3. The value of this pin changes only when ADS# is asserted.

4.2.35 PEN# (PARITY ENABLE)

The i860 XP microprocessor samples this signal for read cycles on the same clock edge at which BRDY# is found asserted. If sampled active, the i860 XP microprocessor feeds the parity check result into the interrupt logic. If a parity error is encountered, the i860 XP microprocessor vectors to the trap handler. The BEAR register latches the offending address, as described with the BERR signal. This interrupt is not masked by the IM bit of the PSR, nor is it masked during lock cycles.

The system should deassert PEN# any time the DP7–DP0 pins are known not to reflect the parity of the full eight-byte bus (for example, reads from I/O devices or ROMs that are not parity protected).

The system should deassert PEN# during code fetches in CS8 mode.

At RESET, the value of PEN# determines the output buffers configuration for ADS#, A21–A3, BE7#–BE0#, W/R#, HITM#. These pins are configured as normal (small output buffers) mode if PEN# is sampled active for at least the last three clocks before RESET deactivates. Otherwise, these pins are configured as high-current mode (large output buffers).

4.2.36 PWT (PAGE WRITE-THROUGH)

PWT provides a write-back/write-through indication on a page by page basis. This signal, together with PCD, is set to an attribute bit in the page table entry for the current cycle. When paging is enabled, PWT corresponds to the WT bit (bit 3), and write-back caching is implemented for this page only if WT is clear. When paging is disabled, or for any cycle that is not paged **(ldio, stio, ldint, scyc)**, the i860 XP microprocessor drives PWT inactive.

During TLB miss processing, PWT is inactive while the address translation hardware is accessing the first level page directory. During accesses to the second-level page-table entry, PWT reflects the WT value taken from the first level page-table entry.

The value of this pin changes only when ADS# is asserted.

4.2.37 RESET (SYSTEM RESET)

Asserting RESET for at least ten CLK periods causes initialization of the i860 XP microprocessor. On power up, RESET should remain active at least one millisecond after V_{CC} and CLK have reached their proper DC and AC specs. RESET is synchronous with CLK.

After the RESET signal goes inactive the processor remains in the RESET state for three more clocks. Applications that use the HOLD signal to float the bus during RESET should keep HOLD active for three more clocks after the RESET signal is deactivated.

4.2.38 RSRVD, SPARE

The RSRVD input is reserved by Intel Corporation and must be tied HIGH to V_{CC} through a resistor (5 KΩ). The spare input should be left unconnected.

4.2.39 TCK (TEST CLOCK)

This is the clock input for the TAP (test access port). If the TAP is to be used, this signal must be connected to a clock synchronous to CLK. If the TAP is not used, TCK can be tied low. TCK does not need to be kept running when boundary scan is not active.

The rising edge of TCK must be externally synchronized to CLK. The boundary scan latches retain their state when TCK is stopped at either logic zero or one.

4.2.40 TDI (TEST DATA INPUT)

TDI is the input for test instructions and data to the TAP. TDI is sampled on the rising edge of TCK. It is provided with an internal pull-up resistor, so that an open circuit at TDI produces a result equivalent to driving continuous HIGH signals.

4.2.41 TDO (TEST DATA OUTPUT)

This is the serial output of the TAP. The contents of TAP registers are shifted out through TDO on the falling edge of TCK. The data is moved from TDI to TDO without inversion, which allows easy serial cascading of different components for scanning.

TDO is held in high-impedance state, except while scanning is in progress. This allows parallel connection of these outputs for several components.

4.2.42 TMS (TEST MODE SELECT)

This input is decoded by the TAP to select the operation of the TAP. It is sampled at the rising edge of TCK. It is provided with an internal pull-up resistor to assure deterministic behavior for open-circuit failure at this pin. If boundary scan is not used, TMS can be tied high or left unconnected.

4.2.43 TRST# (TEST RESET)

This input resets the TAP. If the TAP is not used, TRST# should be tied LOW. To ensure deterministic behavior of the test logic, TMS should be held HIGH while TRST# changes from LOW to HIGH.

4.2.44 V_{CC} (SYSTEM POWER) AND V_{SS} (GROUND)

The i860 XP microprocessor has 54 pins for power and 56 for ground. All pins must be connected to the appropriate low-inductance power and ground signals in the system.

4.2.45 V_{CC}CLK (CLOCK POWER)

This is the power supply for the internal CLK buffer. It should be connected to the same V_{CC} plane as the other V_{CC} pins.

4.2.46 WB/WT# (WRITE-BACK/WRITE-THROUGH)

This input signal defines cache policy for the line being accessed in the current bus cycle. The processor samples WB/WT# for both reads and writes on the same clock edge at which it finds NA# or the first BRDY# asserted, whichever comes first. If this signal is sampled low, the write-through policy is applied to the cache line—if an internal write hits this line, it causes a write-through cycle. If this signal is sampled high, the write-back policy is applied—future write hits to this line do not show up on the bus.

4.2.47 W/R# (WRITE/READ)

This pin specifies whether a bus cycle is a read (LOW) or write (HIGH) cycle. Tables 4.2 and 4.3 show the types of bus cycle generated. The value of this pin changes only when ADS# is asserted.

5.0 BUS OPERATION

The interaction among signals is illustrated by timing diagrams. Figure 5.1 shows the conventions used in the timing diagrams.

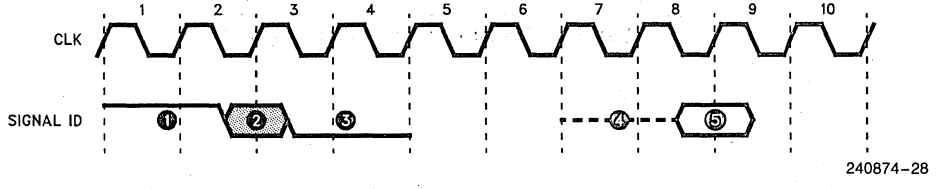
2

5.1 Bus Cycles

A bus cycle begins when the i860 XP microprocessor activates ADS# and ends when the system activates the last of a predetermined number of BRDY# signals. Figure 4.4 shows how the i860 XP microprocessor and the external system cooperate to determine the number of BRDY# activations in each cycle. The processor starts sampling BRDY# one clock after assertion of ADS# and continues sampling in every clock until the last BRDY# becomes active.

The i860 XP microprocessor supports several different types of bus cycle. These are introduced in order of complexity:

1. Single-transfer cycles
2. Multiple-transfer (burst) cycles
3. Pipelined cycles
4. Cache inquiry cycles



NOTES:
1. HIGH (high voltage)
2. Don't care or undefined
3. LOW (low voltage)
4. High-impedance (floating)
5. Either HIGH or LOW

Figure 5.1. Timing Diagram Conventions

5.1.1 SINGLE-TRANSFER CYCLE

The simplest bus cycle is the single-transfer, non-cacheable, 64-bit cycle either with or without wait states. The shortest bus cycle is two clock periods long. Read and write cycles of this type are shown in Figure 5.2.

A *wait state* is any clock in which the i860 XP microprocessor samples BRDY# but the system does not assert it. The system can add wait states to any cycle. Figure 5.3 shows cycles with two wait states added. Any number of wait states can be added to i860 XP microprocessor bus cycles by maintaining BRDY# inactive.

5.1.2 BURST CYCLES

When a bus request requires more than a single data transfer (refer to Table 4.4), the i860 XP microprocessor requires that the memory system perform a burst data transfer. Burst cycles allow the maximum bus transfer rate by eliminating unnecessary driving of the address bus. The addresses of the data items in burst cycles all fall within the same 32-byte aligned area (corresponding to an internal i860 XP microprocessor cache line). Given the address of the first transfer, external hardware can calculate the addresses of subsequent transfers. With these addresses eliminated from the bus, a new data item can be sampled into the i860 XP microprocessor every clock period.

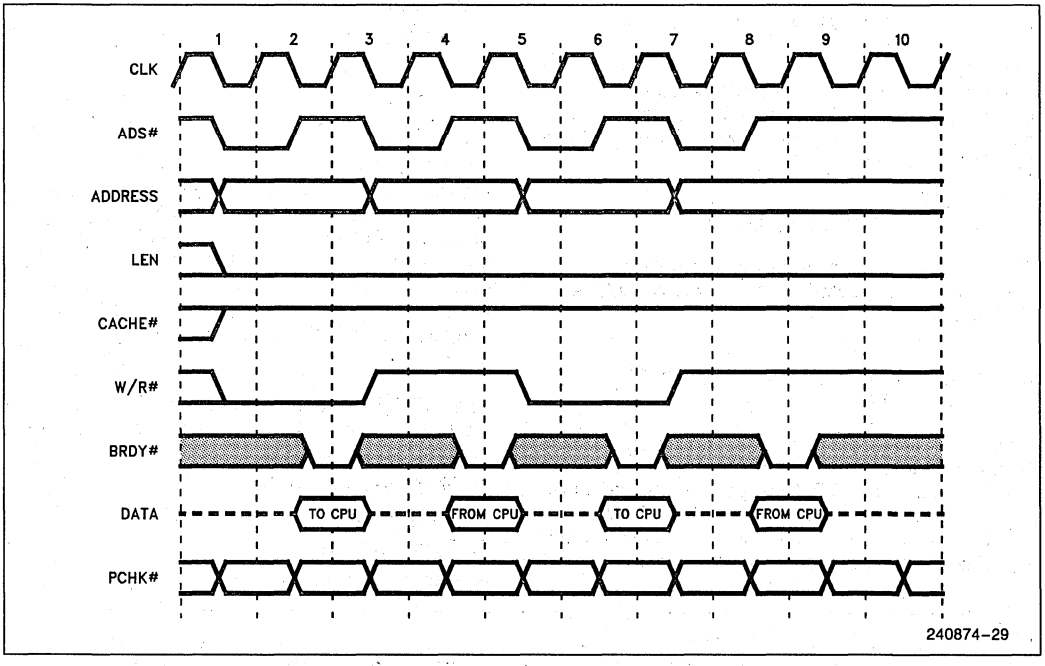


Figure 5.2. Fastest Single-Transfer Cycles

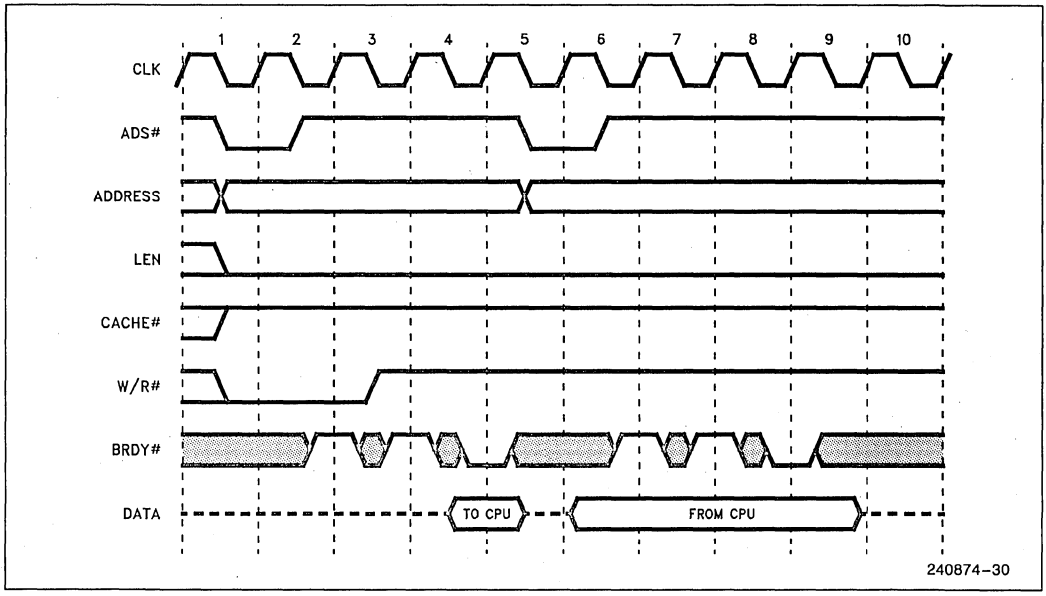
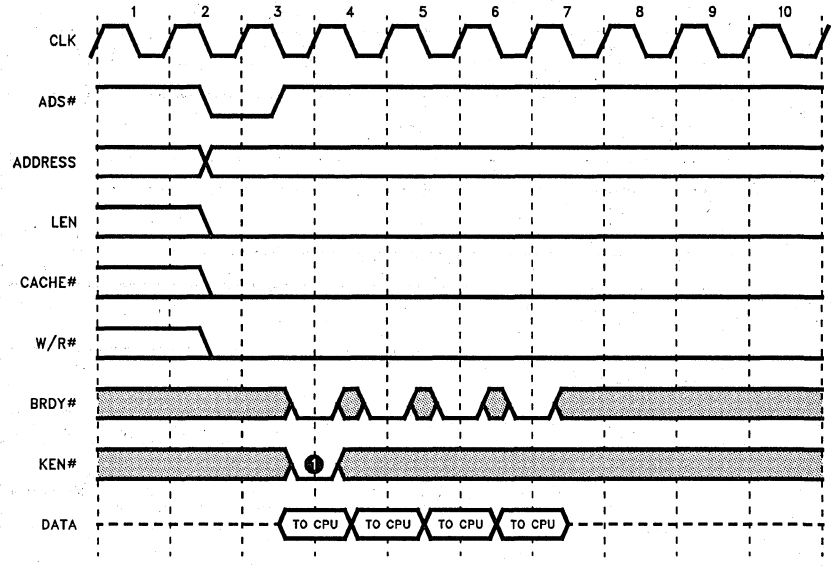


Figure 5.3. Single-Transfer Cycles with Wait States

The fastest possible burst cycle requires two clock periods for the first data item: one clock for ADS# and one clock for BRDY#; subsequent data items are transferred every clock period. One such bus cycle is shown in Figure 5.4. Note that, in this case, the initial cycle generated by the i860 XP microprocessor could be satisfied by a single data transfer, but the system transforms it into a multiple-transfer cache line fill by activating KEN# in the clock period of the first BRDY#. KEN# has this effect only if the CACHE# pin is active, which means the cycle is internally cacheable in the i860 XP microprocessor.

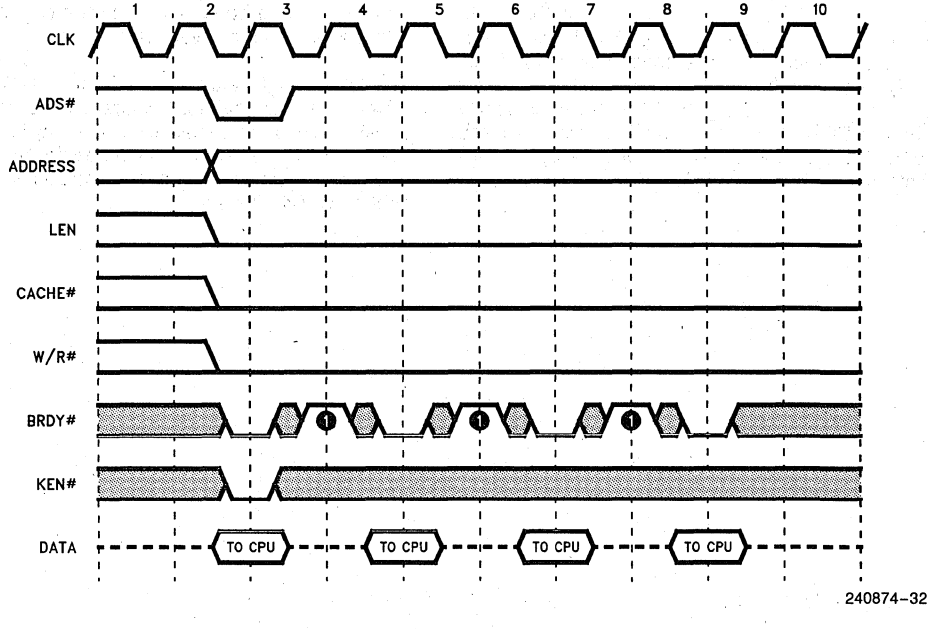
Read data is sampled only in the clock period in which BRDY# is returned, which means that data need not be sent to the i860 XP microprocessor every clock period in the burst cycle. Figure 5.5 shows an example of a burst cycle in which two clock periods are required for every burst item.

The burst length attributes LEN and CACHE# are driven with the address. Figure 5.6 illustrates two consecutive burst cycles with differing length attributes: the first one is a noncacheable 128-bit read, and the second one is a cache line fill initiated by a cacheable 64-bit read.



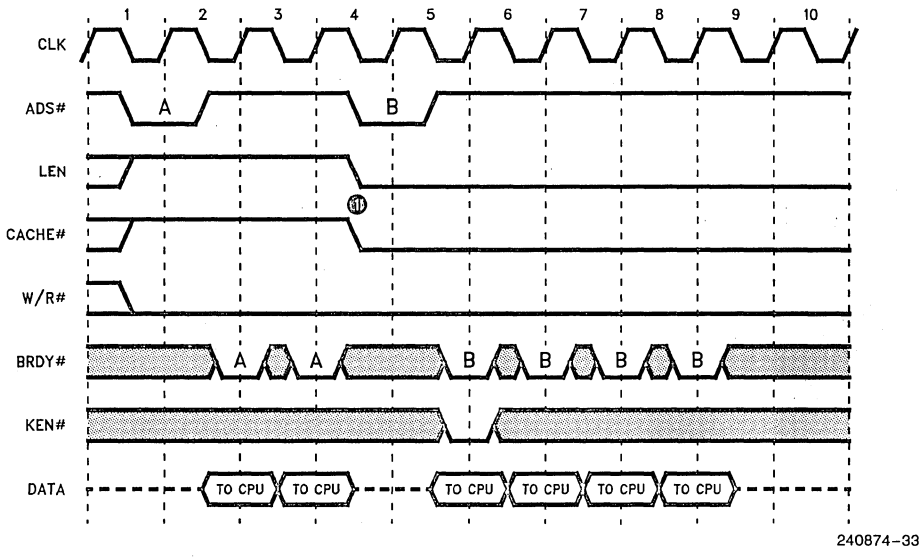
NOTE:
1. KEN# driven with first assertion of BRDY#

Figure 5.4. Basic Burst Cycle



NOTE:
1. Wait states added by delaying assertion of BRDY#

Figure 5.5. Slow Burst Cycle



NOTE:
1. Length attributes driven with ADS#

Figure 5.6. Different Lengths of Burst Cycles

The timing of write bursts is similar to that of read bursts. The i860 XP microprocessor does not put data on D63–D0 for writes until the clock period after ADS#.

When initiating any read, the i860 XP microprocessor presents the address for the data item requested. When the cycle is converted into a cache fill, the first data item returned corresponds to the address sent out by the i860 XP microprocessor. The remaining items must be returned in the order shown in Table 5.1. This ordering is optimized for two-bank memories, but works equally well with noninterleaved memories.

In i860 XP microprocessor systems, memory must support the burst order as defined in Table 5.1 for reads. For writes, the burst addresses are always increasing, so writes with four transfers match the first line of the table. In CS8 (code-size 8 bits) mode, instructions are not fetched in bursts.

Note that the i860 XP microprocessor drives only the first address of a burst cycle; the memory system is responsible for calculating subsequent addresses as shown in the table. The addresses can be derived by complementing A3 after every transfer, and complementing A4 after two transfers.

Table 5.1. Burst Order for Cache Line Transfers

1st Address	2nd Address	3rd Address	4th Address
0	8	0x10	0x18
8	0	0x18	0x10
0x10	0x18	0	8
0x18	0x10	8	0

5.1.3 PIPELINED CYCLES

A *pipelined* cycle is one that starts while one or two other bus cycles are outstanding. A cycle is considered *outstanding* until the last BRDY# is asserted to terminate that cycle. A *nonpipelined* cycle is one that starts when no other bus cycles are outstanding. Both types of cycle can be either read or write cycles. To allow high transfer rates in large memory systems, the i860 XP microprocessor supports two-level pipelining. New cycles can start as often as every other clock until three cycles are outstanding.

The system asserts NA# to indicate that the i860 XP microprocessor can start another cycle before the current one is completed. (NA# can even

be asserted while BRDY# is active.) The i860 XP microprocessor begins sampling NA# in the next clock after ADS# is asserted. If the following conditions are met, a new (pipelined) cycle begins:

1. NA# having been active
2. An internal request pending
3. Compatibility between the pending request and the outstanding requests (refer to Table 5.2)
4. HOLD, BOFF#, and AHOLD not active
5. Fewer than three cycles outstanding

The following "compatibility" rules determine when the processor does not issue a pipelined ADS# (they are the source of Table 5.2):

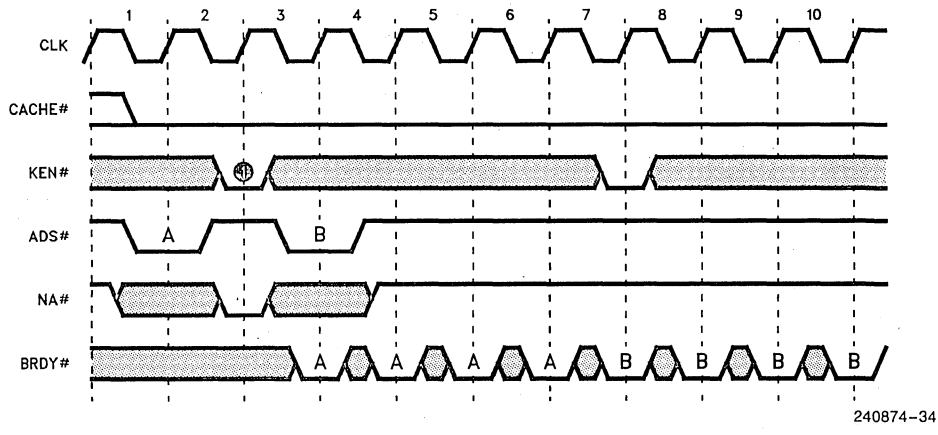
- Data cache line fills are pipelined into each other only in the case of an aliasing virtual tag miss with a physical tag hit.
- Reads can be pipelined into TLB miss writes. TLB misses for instructions can be pipelined into data accesses, and *vice versa*.
- No data cycle is ever pipelined while LOCK# is active.
- I/O cycles, special cycles, and **Idint** cycles never begin when any cycle is outstanding.

NA# may be asserted before, simultaneously with, or after the first BRDY# of the current cycle. If NA# is asserted before the first BRDY#, the cacheability (KEN#) and cache policy (WB/WT#) indicators for the current cycle are sampled during the same clock period as NA# is sampled active; otherwise, they are sampled with the first BRDY#. Figure 5.7 shows an example of four-transfer, pipelined, back-to-back reads. Note the timing of KEN#. Because NA# is asserted before the first BRDY# of the cycle A, KEN# is sampled with the NA# for cycle B.

Table 5.2. Pipeline Cycle Compatibility

	B	If A is Outstanding, can B be Pipelined into It?								
	A	Data Cache Line Fill	Data Cache Store Miss, Write-Thru	Data Cache Read Miss KEN# = 1	Write-Back**	Instruction Fetch	pfld	TLB Miss	Idio, stio, Idint, scyc	LOCK# Active
PREVIOUS CYCLE	Data Cache Line Fill	YES*	YES*	YES*	YES	YES	YES*	YES	NO	YES
	Data Cache Store Miss, Write-Thru	YES	YES	YES	YES	YES	YES	YES	NO	YES
	Data Cache Read Miss KEN# = 1	YES*	YES*	YES*	YES*	YES	YES*	YES	NO	YES*
	Write-Back	YES	YES	YES	NO	YES	YES	YES	NO	YES
	Instruction Fetch	YES	YES	YES	YES	YES	YES	YES	NO	YES
	pfld	YES	YES	YES	YES	YES	YES	YES	NO	YES
	TLB Miss	YES	YES	YES	YES	YES	YES	YES	NO	YES
	stio scyc	YES	YES	YES	YES	YES	YES	YES	NO	YES
	idio idint	NO	NO	NO	NO	YES	NO	YES	NO	NO
	LOCK# Active	NO	NO	NO	NO	YES	NO	YES	NO	NO

NOTE:
* Pipelining can occur if the first ADS# is for an aliasing virtual tag miss with a physical tag hit.
**Inquiry write-backs are not pipelined into prior cycle unless FLINE# is asserted.



NOTES:
A Four-transfer, cache line fill cycle
B Four-transfer, cache line fill cycle
1. KEN# for A simultaneous with NA#

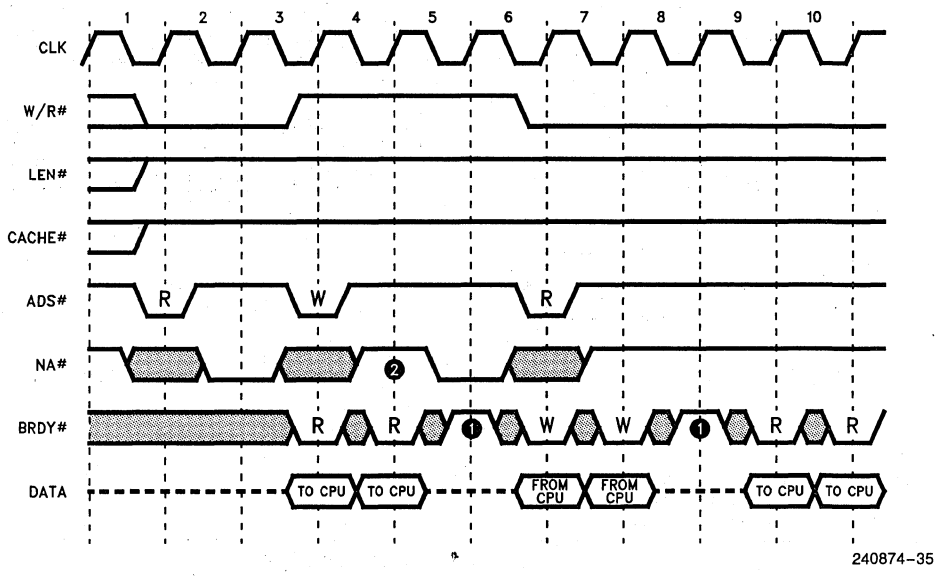
Figure 5.7. Pipelined Cache Line Fills

Write cycles can be pipelined into read cycles and *vice versa*, but, in both cases, the processor will leave one clock between bursts to allow bus turnover, and will ignore any BRDY# given to it at that time. Pipelined back-to-back read and write cycles are shown in Figure 5.8. On writes, assertion of NA# does not cause the values on the data bus to change; it just enables new address and cycle specification outputs.

5.1.4 INTERRUPT ACKNOWLEDGE CYCLES

In response to a trap caused by assertion of the INT pin, trap-handling software can generate interrupt acknowledge cycles by executing a procedure similar to the following.

```
//The following lock instruction must be on a 32-byte boundary:
  lock                        // Lock the bus
  ldint.b src2,  rdest        // First INTA cycle. Src2 contains 8.
  or      rdest, r0, rdest    // Won't proceed until rdest loaded.
  unlock                      // Unlock the bus after the next ldint
//nop                         // Insert 4 + <number of NOPs> idle
//nop                         //   clocks for 8259A recovery.
  ldint.b r0,    rdest        // Second INTA cycle
```



NOTES:
R Two-transfer, noncacheable read cycle
W Two-transfer, noncacheable write cycle
1. Idle clock for bus turnaround
2. Second assertion of NA# could be here

Figure 5.8. Pipelined Back-to-Back Read and Write Cycles

Figure 5.9 shows the interrupt acknowledge cycles generated by the code sequence. Interrupt acknowledge cycles are generated in locked pairs. The interrupt vector is returned during the second cycle. Each of the interrupt acknowledge cycles is terminated when the external system responds by asserting BRDY#. Wait states can be added by withholding BRDY#. There must be a number of idle clocks between the first and second cycles to allow for 8259A recovery time. The software controls the number of intervening clocks via the number of **nop** instructions in the interrupt acknowledge routine.

5.1.5 SPECIAL BUS CYCLES

The i860 XP microprocessor provides a special cycle to indicate to the external system that certain internal conditions have occurred. The special bus cycle (indicated by M/IO# = 0, D/C# = 0, and W/R# = 1) is generated by the i860 XP microprocessor as a response to **scyc** instruction execution. This cycle (defined in Table 5.3) is used to flush or invalidate a secondary cache. The defined value of byte enables can be generated by using an appropriate address operand in the **scyc** instruction. The **scyc** instruction does not have any effect on the internal caches. External hardware must acknowledge a special bus cycle by asserting BRDY# once. The data driven on the data bus with BRDY# is *undefined*. The effect of **scyc** is determined by decoders in external hardware.

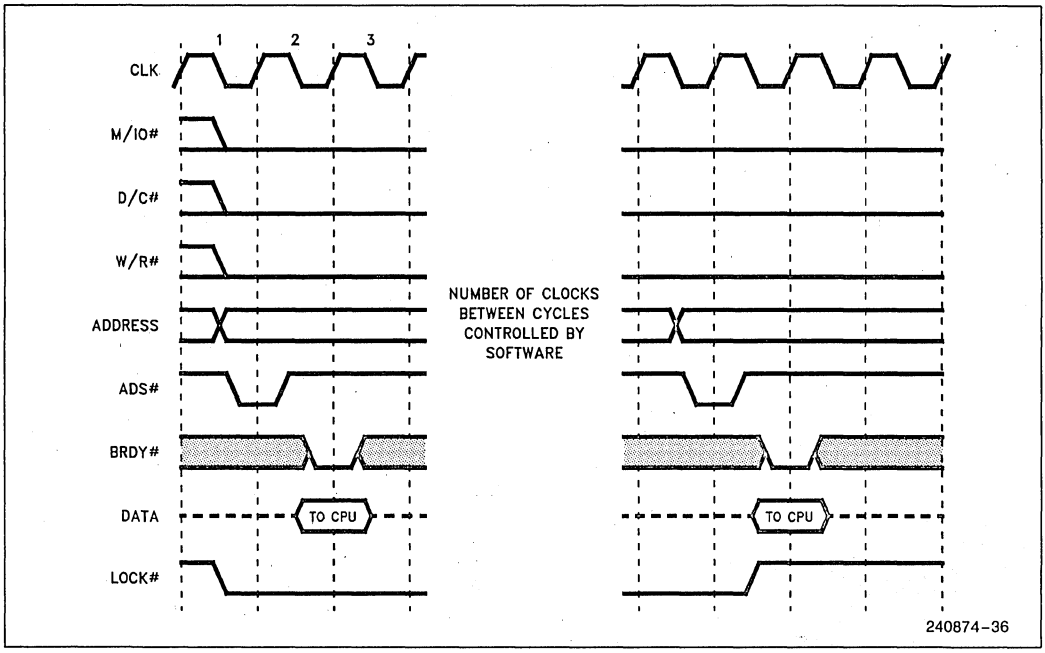


Figure 5.9. Example Interrupt Acknowledge Sequence

Table 5.3. Encoding of Special Bus Cycles

BE7#–BE0#	Special Bus Cycle
11110111	Write Back External Cache and Invalidate
11111011	Halt
11111101	Invalidate External Cache
11111110	Shut Down

All other encodings are *reserved.*

5.2 Bus Arbitration

The i860 XP microprocessor responds to three different signals that tell it to stop driving the bus:

HOLD Finishes outstanding cycles before giving up the bus.

BOFF# Aborts outstanding cycles and gives up bus immediately.

AHOLD Stops driving address bus and permits a cache inquiry.

AHOLD results in a partial hold state, which is covered in Section 5.3. The present section concentrates on HOLD and BOFF#.

When in a hold state (due either to HOLD or BOFF#), the i860 XP microprocessor uses BREQ to request control of the bus. If holding due to HOLD, AHOLD, or BOFF#, the processor activates BREQ in the clock after an internal bus request is generated. (In the case of HOLD, BREQ is asserted even though HLDA is asserted.) If holding due to BOFF# and cycles need to be restarted or there is a new internal request, it asserts the BREQ signal within four clock periods after the assertion of BOFF#. In all cases, BREQ remains active at least until the clock after ADS# is activated for the requested cycle.

5.2.1 HOLD AND HLDA ARBITRATION

HOLD indicates to the i860 XP microprocessor that another bus master needs control of the bus. When HOLD is asserted, the i860 XP microprocessor keeps control of the bus until all outstanding cycles are completed. Then it floats the output signals (except BREQ, HLDA, LOCK#, PCHK#, HIT#, and HITM#) and asserts HLDA. These outputs remain at the high-impedance state until HOLD is deasserted.

HLDA may be asserted as soon as the clock period after the one in which HOLD is asserted. HLDA may be deasserted as soon as the clock after the one in which HOLD is deasserted.

An example HOLD/HLDA transaction is shown in Figure 5.10. The i860 XP microprocessor recognizes HOLD even while RESET is asserted, and it drives HLDA in this case as well.

HOLD is recognized even when BOFF# is active, and the i860 XP microprocessor responds with HLDA the same as when the bus is idle.

5.2.2 BUS CYCLE BACK-OFF AND RESTART

The i860 XP microprocessor provides the ability to abort bus cycles and restart them again. It is necessary to abort cycles for reasons such as the following:

1. Retry after an error is detected by ECC or parity logic.
2. Escape from a deadlock; for example, when the i860 XP microprocessor is using A31–A3 to load a new cache line, but the 82495XP cache controller needs A31–A5 to invalidate a line in the CPU cache which the 82495XP cache controller is replacing in its cache in order to satisfy the CPU's line-fill request.
3. Maintain cache consistency; for example, the i860 XP microprocessor is attempting to read or write to a line that has been modified in the cache of another CPU.
4. Prevent illegal access to an address already locked by another CPU in a multiprocessor system.

5.2.2.1 Cycle Back-Off

Bus cycles are aborted when the system asserts BOFF#. The i860 XP microprocessor samples this pin in every clock period that it is driving the bus. When BOFF# is asserted, the i860 XP microprocessor immediately (in the next clock period) floats the bus. It floats the ADS# pin one clock period later, thereby giving time for ADS# to be deasserted so that it is not left floating active. The i860 XP microprocessor floats the same pins as for HOLD, but HLDA is not asserted. If a bus cycle is in progress at the time BOFF# is asserted, the cycle is aborted, and, in a read cycle, any data returned to the processor while BOFF# is active is ignored. BOFF# overrides BRDY#; so, if both are sampled active in the same clock, BRDY# is ignored. BOFF# aborts

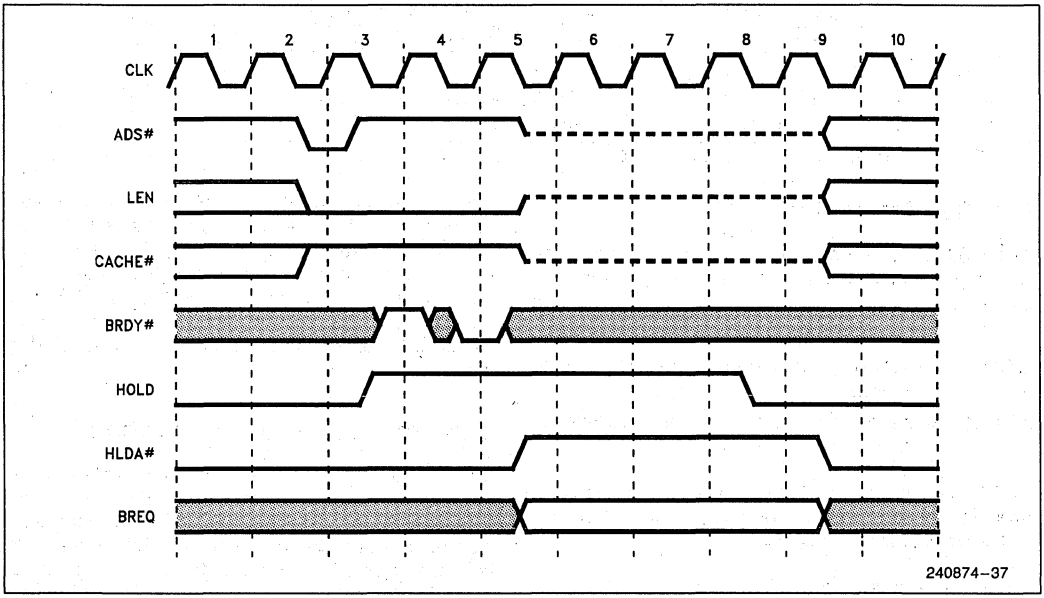


Figure 5.10. HOLD/HLDA Handshake

a burst cycle even if it arrives with the last BRDY# of the cycle. However, for read bursts, data transfers completed before assertion of BOFF# are used by the processor if they satisfy an internal request. Cacheable data is cached in spite of BOFF#; however, the cached data is overwritten when the cycle is restarted.

The bus remains in the high-impedance state until BOFF# is deasserted. If cycles need to be restarted or if a new internal request has been generated, the BREQ signal is asserted within four clock periods after the assertion of BOFF#.

5.2.2.2 Cycle Restart

When the system deasserts BOFF#, the i860 XP microprocessor restarts aborted bus cycles from the beginning by driving the address and status (A31–A3, W/R#, D/C#, etc.) and asserting ADS#. If more than one cycle was outstanding when BOFF# was asserted, the i860 XP microprocessor restarts all outstanding cycles in the same order. If HITM# is active due to an inquiry, the write-back for it will be the first cycle after deassertion of BOFF#. BOFF# restarts all aborted cycles except:

- The stale cycles mentioned in section 5.3.5.
- The read that may have been generated by an alias hit (virtual tag miss, but physical tag hit).
- The read that may have been generated by a **pfld** that hit the data cache.

If the processor's KEN# pin was active (with NA# or first BRDY#) before the cycle was aborted, external hardware must activate it again after the cycle is restarted. In other words, the system cannot use BOFF# to change the cacheability of a cycle via KEN#.

The LOCK# signal is not affected by restarted cycles; it retains its state in spite of BOFF# assertion.

5.2.2.3 Late Back-Off Modes

In some cases the logic that needs to assert BOFF# cannot make the necessary decision in time to cancel the relevant cycle or data transfer. For example:

1. The result of checking ECC or parity may not be available until one or two cycles after the BRDY# to which it corresponds.
2. When the i860 XP microprocessor is attempting to read or write to a line that might be modified in the cache of another processor on the same bus, it may be advantageous to let part of a burst run in parallel with inquiries to the other processors, rather than delay the entire burst until the inquiries are finished.

For such situations, the i860 XP microprocessor provides *late back-off mode*. For a read cycle in this mode, the processor employs a buffer to internally delay data and BRDY#, which allows BOFF# assertion to be delayed relative to the external BRDY#. Likewise, for a write cycle in this mode, BOFF# assertion can be delayed relative to BRDY#. However, data for a write cycle is not delayed.

Two flavors of late back-off mode are provided:

1. One allows BOFF# to be delayed by one clock period relative to the data transfer. The processor enters one-clock late back-off mode when the FLINE# pin has been sampled active for at least three clock periods when RESET deactivates.
2. The other allows BOFF# to be delayed by up to two clock periods relative to the data transfer. The i860 XP microprocessor enters this mode when software sets the LB bit of the **dirbase** register.

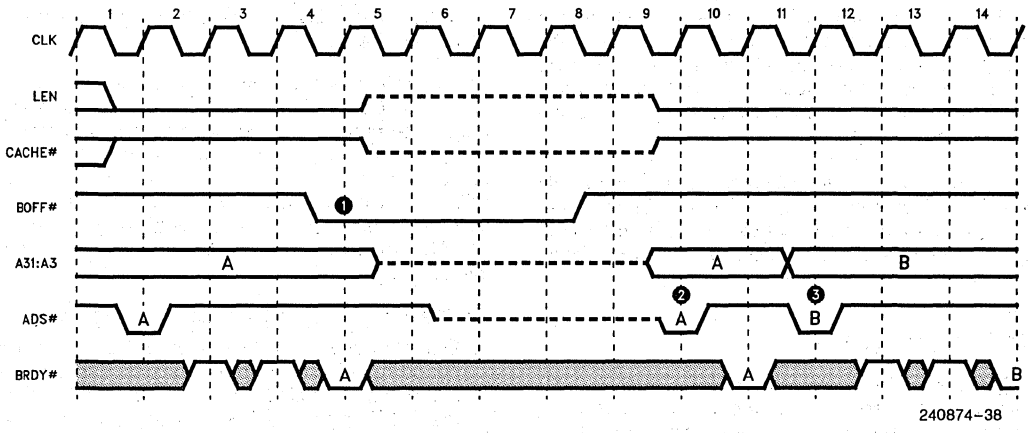
If the processor enters one-clock late back-off mode during RESET, it is impossible to enter two-clock late back-off mode. The LB bit has no effect. Furthermore, software cannot exit two-clock late back-off mode once it is activated, and the LB bit cannot be cleared except by resetting the processor.

Figures 5.12–5.17 illustrate variations on late back-off mode cycles. BOFF# can be (and usually is) asserted longer than one clock period, as Figure 5.11 shows; the remaining figures show an active time of only one clock.

5.2.2.4 One-Clock Late Back-Off Mode

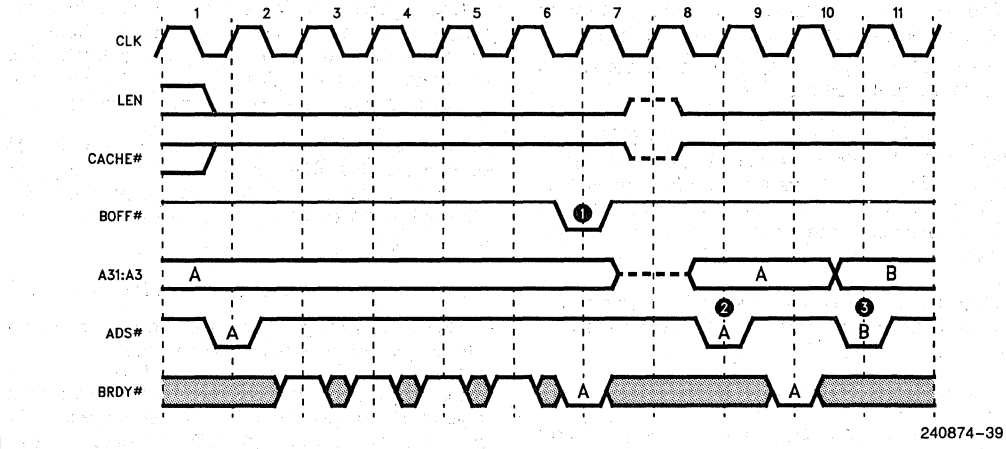
In *one-clock late back-off mode* the data is delayed internally by one clock before it is used.

In this mode, data and BRDY# are seen by internal logic one clock period later than they appear on the bus, which is equivalent to adding an extra wait state to reads on the external bus (Figure 5.13). All responses to BRDY# (assertion of the ADS# for the next cycle, assertion of HLDA in response to a HOLD request, and deassertion of HITM#) are delayed by one clock period compared to the normal mode of operation. Not delayed, however, are write data on D63–D0 and sampling of KEN# and WB/WT#. KEN# and WB/WT# must be valid with the first BRDY# assertion. Also, the response to NA# (assertion of ADS#) is not delayed if fewer than three pipelined cycles are outstanding.



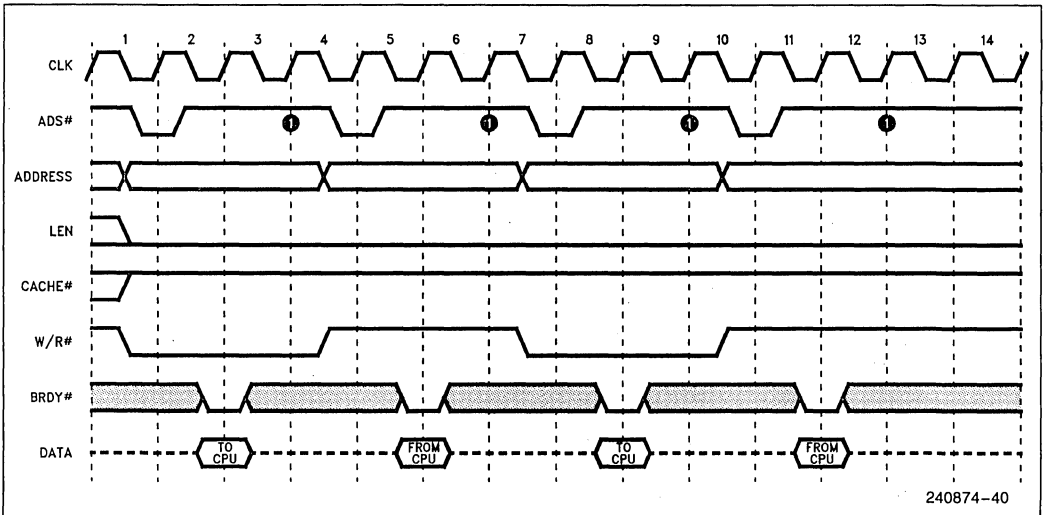
NOTES:
A Noncacheable, 64-bit cycle (one transfer)
B Next cycle (any type)
1. BOFF# cancels cycle and data transfer
2. Cycle A restarts one clock after BOFF# is deasserted
3. Earliest ADS# assertion for next cycle

Figure 5.11. Normal Back-Off



NOTES:
A Noncacheable, 64-bit cycle (one transfer)
B Next cycle (any type)
1. BOFF# cancels cycle and data transfer
2. Cycle A restarts one clock after BOFF# is deasserted
3. Earliest ADS# assertion for next cycle

Figure 5.12. One-Clock Normal Back-Off



NOTE:
1. Idle clock due to internal delay of BRDY#

Figure 5.13. Fastest Nonpipelined Cycles in One-Clock Late Back-Off Mode

If BOFF# is asserted as late as the second BRDY# (Figure 5.14), it cancels the entire cycle, ignores data latched with the first BRDY#, and ignores the data being driven with the second BRDY#. This is true of a two-transfer burst (shown) as well as a four-transfer burst (not shown).

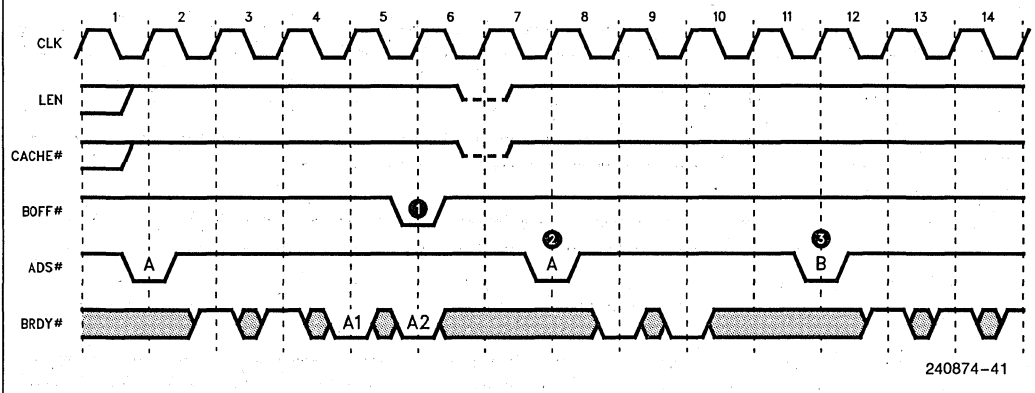
In a two-transfer burst, if BOFF# is asserted in the clock after the second BRDY# (Figure 5.15), it still cancels the cycle.

In a four-transfer burst, if BOFF# is asserted within one clock after the last BRDY# (Figure 5.16), it still forces a retry of the cycle, but previously transferred read data is used by the processor if it satisfies the read request.

5.2.2.5 Two-Clock Late Back-Off Mode

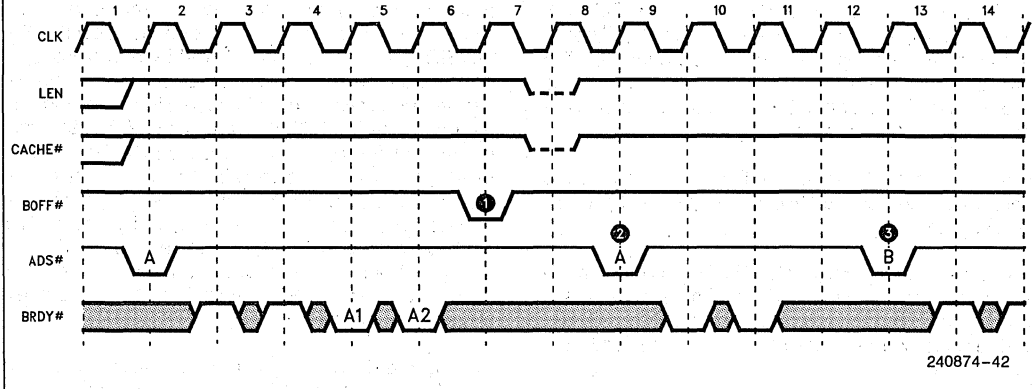
Two-clock late back-off mode gives external logic even more time to decide to use BOFF#. In this mode, data delivery is delayed by either one or two clock periods, depending on external activity. For any BRDY#, the data is delayed by one clock period. If in the next clock period BRDY# is again asserted, the previous data is used. However, if in that next clock period BRDY# remains inactive, the data is delayed for one extra clock period before it is used. The responses to BRDY# (assertion of the ADS# for the next cycle, assertion of HLDA, and deassertion of HITM#) are delayed by one or two clock periods, depending on the value of BRDY# in the next clock. The response to NA# (assertion of ADS#) is not delayed if fewer than three pipelined cycles are outstanding.

The **st.c dirbase** instruction that sets the LB bit must be aligned on a 32-byte boundary and must be followed by seven **nop** instructions. Software must not enable late back-off mode when the processor is used with the 82495XP external cache controller.



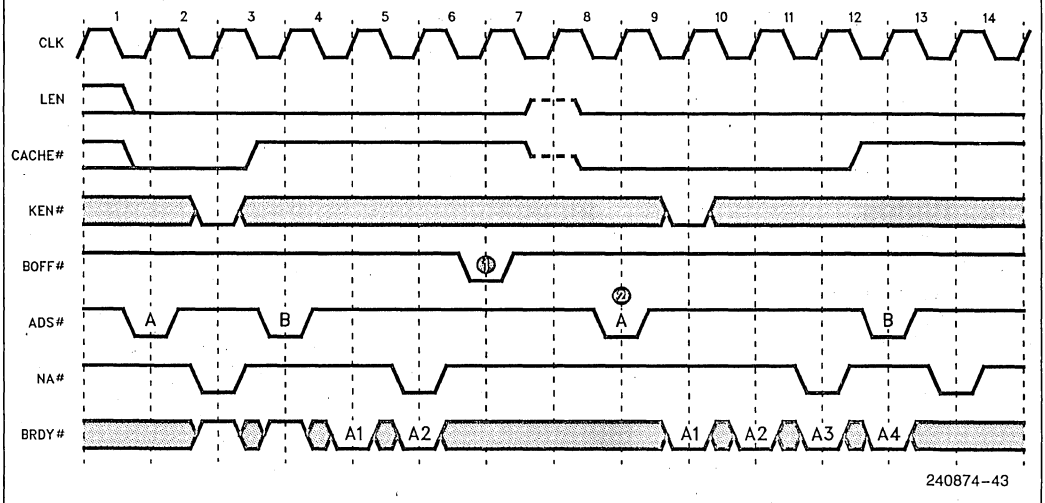
NOTES:
A Noncacheable, 128-bit cycle (two transfers)
B Next cycle (any type)
1. BOFF# cancels both transfers (A1 in buffer, A2 on D63–D0)
2. Cycle A restarts one clock after BOFF# is deasserted
3. Earliest ADS# assertion for next cycle

Figure 5.14. One-Clock Late Back-Off Mode (Case 1)



NOTES:
A Noncacheable, 128-bit cycle (two transfers)
B Next cycle (noncacheable)
1. BOFF# cancels both transfers (A2 in buffer is needed to satisfy request)
2. Cycle A restarts one clock after BOFF# is deasserted
3. Earliest ADS# assertion for next cycle

Figure 5.15. One-Clock Late Back-Off Mode (Case 2)



NOTES:
A Cacheable 64-bit (or less) cycle (four transfers)
B Next cycle (any type)
1. BOFF# cancels A2 and A3 transfers, but A1 transfer has already satisfied request
2. Cycle A restarts one clock after BOFF# is deasserted
3. Earliest ADS# assertion for next cycle

Figure 5.16. One-Clock Late Back-Off Mode (Case 3)

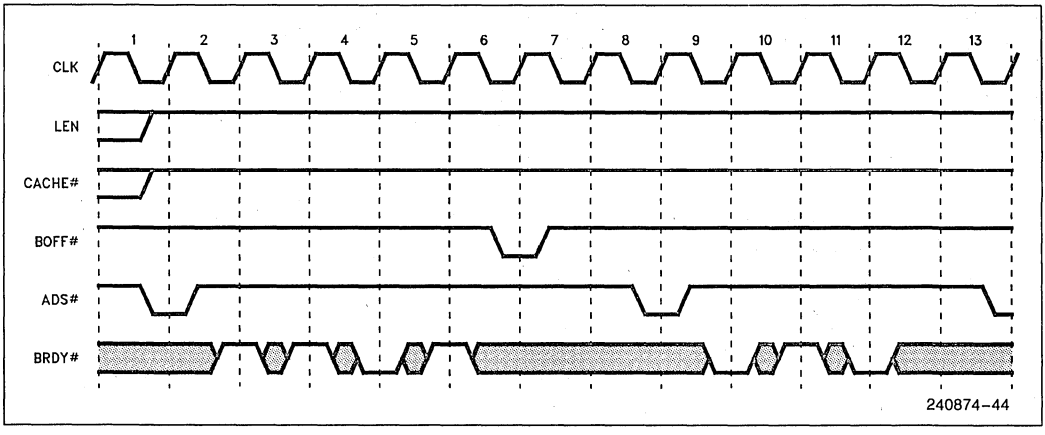


Figure 5.17. Two-Clock Late Back-Off Mode

5.3 Cache Inquiry Cycles (Snooping)

Another processor initiates an inquiry cycle to check whether an address is cached in the internal data or instruction cache of the i860 XP microprocessor. An inquiry cycle differs from any other cycle in that it is initiated externally to the i860 XP microprocessor, and the signal for beginning the cycle is EADS# (External Address Status) instead of ADS#. The address bus of the i860 XP microprocessor is bidirectional in order to allow the address of inquiry to be driven by the system. An inquiry cycle can begin during any hold state:

1. While HOLD and HLDA are asserted.
2. While BOFF# is asserted.
3. While AHOLD (address hold) is asserted.

If neither a HOLD nor a BOFF# is in effect, the system can assert AHOLD to interrupt the current bus activity.

EADS# is first sampled two clocks after BOFF# or AHOLD assertion, or one clock after HLDA. This allows time for the processor to float A31–A5 and for the system to stabilize the inquiry address there.

In the clock in which EADS# is asserted, the i860 XP microprocessor samples these inputs, which qualify the type of inquiry:

INV Specifies whether the line (if found) must be invalidated (that is, changed to I-state).

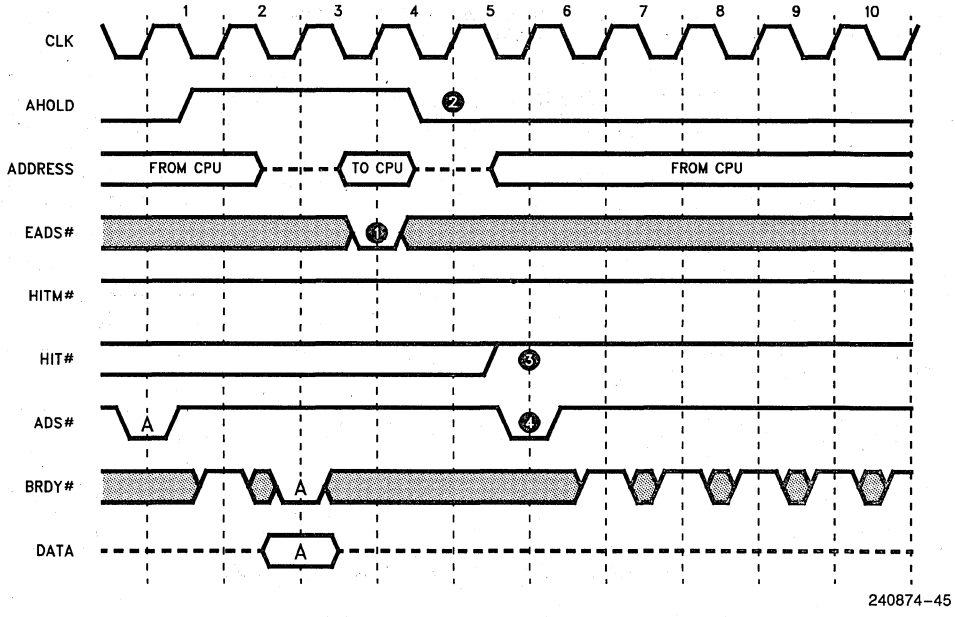
FLINE# Specifies whether the line (if found in M-state) must be written back immediately or after outstanding bus cycles are completed.

The i860 XP microprocessor compares the address of the inquiry request with addresses of lines in cache and of any line in the write-back buffer waiting to be transferred on the bus. It does not, however, compare with the address of write-miss data in the write buffers. Two clock periods after sampling EADS#, the i860 XP drives the results of the inquiry look-up on these output pins:

HIT# Specifies whether the address was found (active) or not found (inactive).

HITM# If active, the line found was in the M-state; if inactive, the line was in E- or S-state, or was not found.

Figure 5.18 shows an inquiry with AHOLD that misses the cache. When the system asserts AHOLD, the i860 XP microprocessor floats A31–A3 in the next clock period. It does not, however, assert HLDA; no acknowledge is required. Once the address pins are floating, external logic drives the address for the inquiry on A31–A5 and starts the inquiry cycle by activating EADS#. The i860 XP microprocessor does not begin sampling EADS# until the second clock after AHOLD is activated. EADS# activation may be delayed any number of clocks.



NOTES:
A Outstanding cycle (for example, a single-transfer read) finishes during the inquiry
1. Earliest assertion of EADS# is two clocks after assertion of AHOLD
2. Earliest deassertion of AHOLD is one clock after assertion of EADS#
3. HIT# is valid two clocks after assertion of EADS#
4. Earliest assertion of ADS# for next cycle is one clock after deassertion of AHOLD

Figure 5.18. Inquiry Miss Cycle

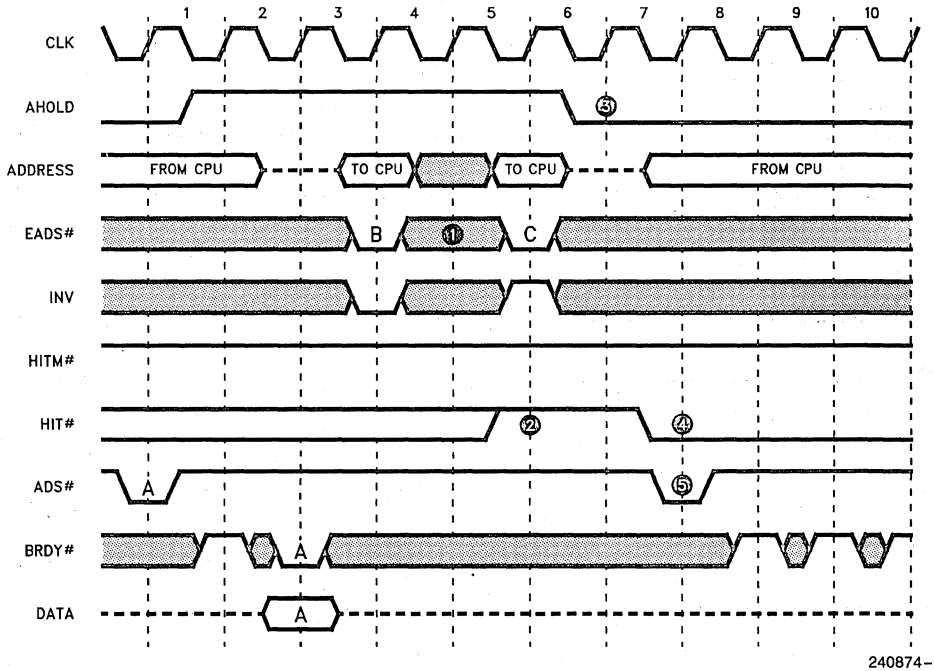
The earliest that AHOLD can be deasserted is the clock after EADS# assertion. However, by maintaining AHOLD active, multiple inquiry cycles can be executed in one AHOLD session (Figure 5.19). The i860 XP microprocessor can accept inquiry cycles at a rate of one every other clock period, unless a write-back is required. The earliest that ADS# can be asserted for the next cycle is the clock after AHOLD deassertion.

The second inquiry in Figure 5.19 hits an unmodified line in the cache. When a cache line with matching address is found and the INV input signal is asserted (as in this case), that line is invalidated (changed to I-state). If the INV signal is inactive, the line enters S-state.

5.3.1 INQUIRY WRITE-BACK CYCLES

If an inquiry cycle hits a dirty (M-state) line in the i860 XP microprocessor cache, the i860 XP microprocessor asserts the HITM# signal to indicate that the line will be written on the bus. The HITM# output becomes valid in the same clock period as HIT#. In this case the modified line is written out, and the cache entry is changed to either I or S state according to INV. The HITM# signal stays active through the last BRDY# for the corresponding write-back cycle.

An inquiry write-back cycle is similar to ordinary write-back cycles. It is initiated by assertion of ADS#. ADS# is asserted even when the AHOLD



NOTES:
A Outstanding cycle (for example, a single-transfer read) finishes during the inquiry
B Earliest inquiry, no invalidation
C Earliest successive inquiry, with invalidation
1. EADS# is not sampled in the clock after its assertion
2. Inquiry B misses cache
3. Earliest deassertion of AHOLD is one clock after last assertion of EADS#
4. Inquiry C hits cache, invalidates line
5. Earliest assertion of ADS# for next cycle is one clock after deassertion of AHOLD

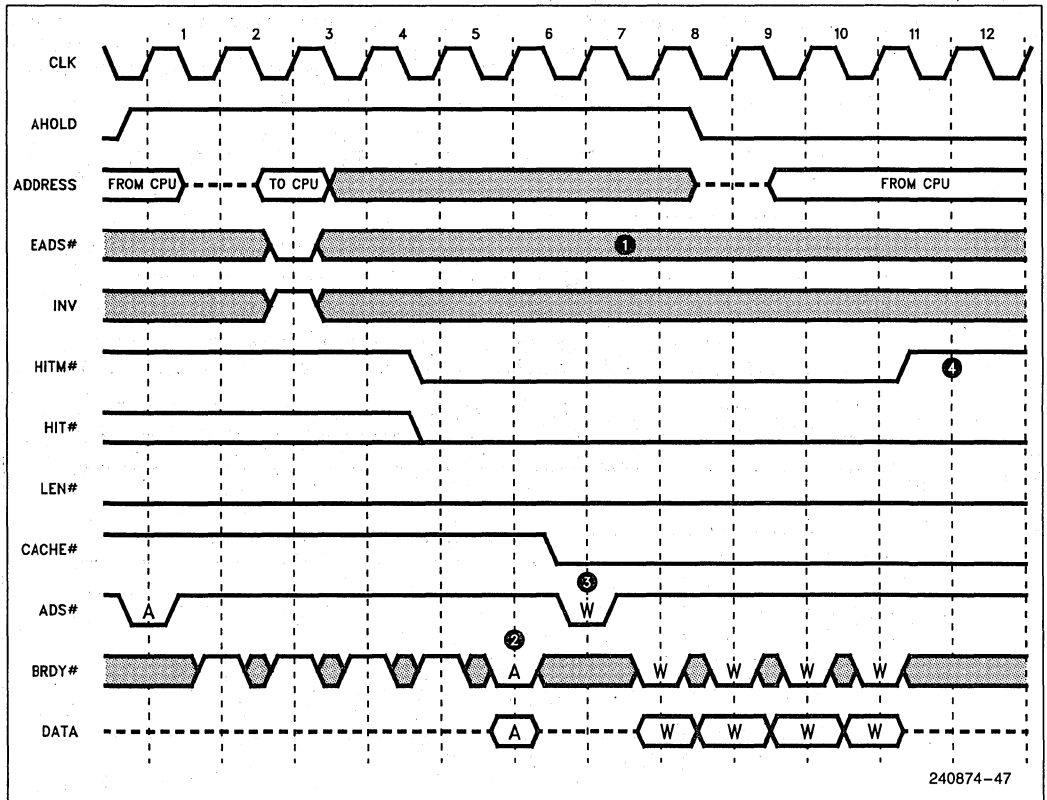
Figure 5.19. Fastest Inquiry Cycles (Miss and Hit)

signal is active. The cycle definition signals are driven properly by the processor, however, the address pins are not driven, because activation of AHOLD forces the i860 XP microprocessor off the address bus. If, however, AHOLD is deasserted before or during the write-back cycle, the i860 XP microprocessor drives the correct address for the write-back.

For all types of inquiry, the write-backs are not pipelined into an outstanding cycle, except when the FLINE# pin is used (refer to section 5.3.5). ADS# for the inquiry write-back is asserted from one to four clock periods after the HITM# pin is driven active or after the last BRDY# is returned for any outstanding cycle, whichever occurs later.

Bursts for a HITM# write-back, as for any write-back, are in the order 0, 8, 0x10, 0x18, because the i860 XP microprocessor ignores A4–A3 of the inquiry address.

Figure 5.20 shows an inquiry cycle that hits an M-state line.



NOTES:
A Outstanding cycle (for example, a single-transfer read)
W Write-back cycle
1. EADS# is not sampled while HITM# is active
2. Earliest ADS# assertion if not delayed by outstanding cycle
3. ADS# for write-back delayed by outstanding cycle
4. HITM# deactivates after last BRDY# of write-back

Figure 5.20. Inquiry Hit Cycle with Write-Back

The fact that a write-back cycle is initiated while address lines are floating supports multiple inquiries (with write-backs) during a single AHOLD session. This is especially useful during secondary cache replacement processing, when the secondary-cache line is larger than that of the i860 XP microprocessor.

Note that EADS# is ignored as long as HITM# is active. If the system is executing a series of inquiries, it might happen that the HITM# assertion for one inquiry masks the EADS# for a subsequent inquiry. In that case the system must reassert EADS# to restart the masked inquiry.

Inquiries can occur during a hold due to HOLD/HLDA or BOFF#. However, in these cases, the cycle definition pins and ADS# are floating. If an inquiry requires a write-back, the HOLD or BOFF# must be deasserted so that the cycle definition pins and ADS# can be driven to start the write-back cycle. If HITM# is active at the time of ADS#, the first ADS# issued after HOLD is deasserted corresponds to the write-back of the modified line which was snooped.

5.3.2 SNOOPING RESPONSIBILITY LIMITS

The i860 XP microprocessor takes responsibility for responding to inquiry cycles for a cache line only during the time that the line is actually in the cache or in a write-back buffer. There are times during the cache line fill cycle and during the cache replacement cycle when the line is "in transit", and inquiry (snooping) responsibility must be taken by other system components.

Systems designers should consider the possibility that an inquiry cycle may arrive at the same time as a cache line fill or replacement for the same address. This situation can occur:

- In multiprocessor systems that have external (secondary) caches with separate CPU and memory busses, thereby allowing concurrent activity on the two busses. In such systems, it is desirable to run invalidation cycles concurrently with other i860 XP microprocessor bus activity. It can happen that writes on the memory bus cause invalidation requests to the i860 XP microprocessor at the same time that the i860 XP microprocessor fetches data from the secondary cache. Such events can occur at any time relative to each other.
- In multiprocessor systems with no secondary cache, if memory is dual-ported. In such systems, two processors can simultaneously read the same line, each sending an inquiry to the other.

The simultaneous activities considered here may be for different data items in the same cache line. Unless the inquiry request is timed carefully with respect to the cache fill cycle, the cache-consistency mechanism may be subverted, and data inconsistencies may result (for example, both CPUs may get the line in E-state on a read). If the 82495XP and 82490XP cache is being used, the timing with respect to the i860 XP microprocessor is handled correctly by the cache controller; however, the same problem may arise between the memory system and the secondary cache.

There are two cases to consider:

1. Inquiry for a line that is being cached.
2. Inquiry for a line that is being replaced.

5.3.2.1 Inquiry for a Line Being Cached

The i860 XP microprocessor accepts an inquiry cycle at any time, even if it hits the line being cached at that time. Regardless of the timing of the cycle, the i860 XP microprocessor delivers the read data to the load instruction that initiated the read request. However, the timing of the invalidation cycle determines whether the line is placed in the cache and what value the i860 XP microprocessor drives on HIT#. Table 5.4 summarizes the different cases.

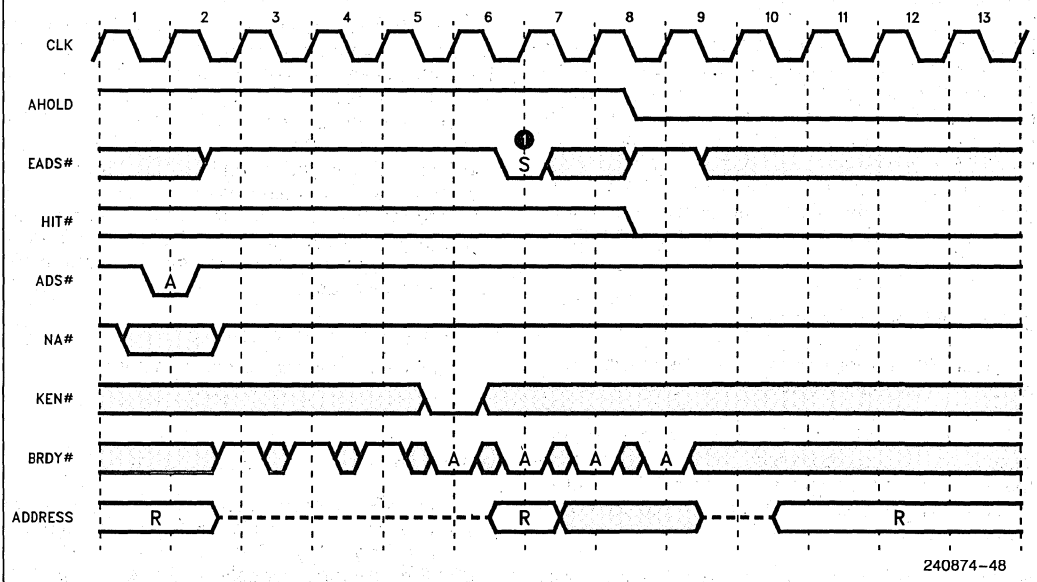
Table 5.4. Inquiry for a Line being Cached

	EADS# before or with NA# or 1st BRDY#	EADS# after NA# or 1st BRDY#
Line is cached?	YES	NO
HIT# =	Inactive	Active
Data/Instruction used by CPU?	YES	YES

If EADS# is asserted before or with the sampling of KEN#, the processor cannot match the address of the line being cached with an invalidation request. Thus, the processor does not assert HIT#. The external system must satisfy the inquiry with the correct data and WB/WT# status. If invalidation of that line is required, the system must do one of the following:

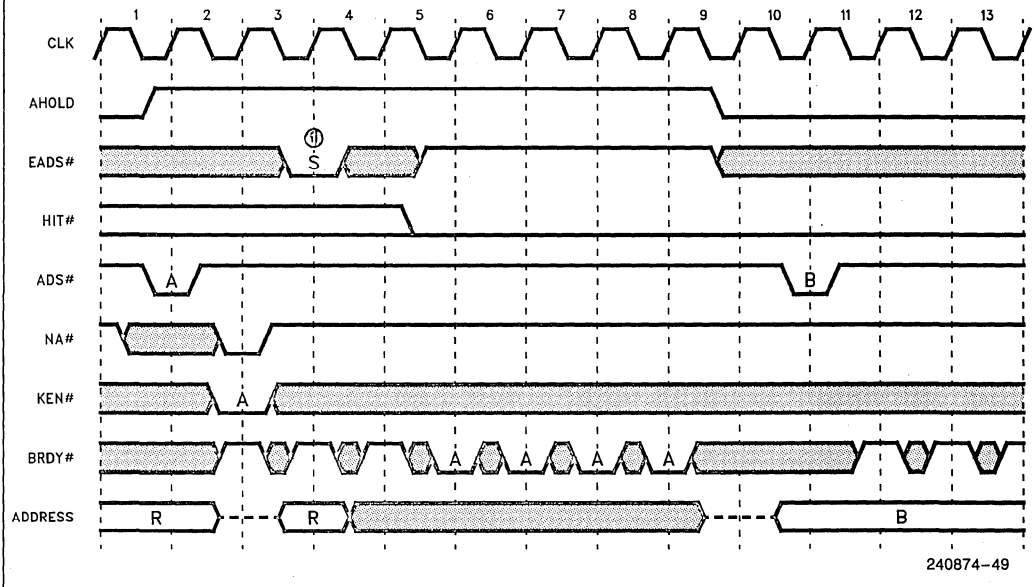
- Delay assertion of EADS# until one clock after assertion of KEN#.
- Reassert EADS# after KEN#.
- Make KEN# inactive at the first BRDY# or NA#, thereby preventing the line from being cached.

Figures 5.21 and 5.22 show when the i860 XP microprocessor picks up responsibility for inquiries for a line that it is caching. Figure 5.21 shows the earliest EADS# assertion that invalidates the line being cached relative to the first BRDY# for nonpipelined cycles. Figure 5.22 shows the earliest EADS# assertion that invalidates the line being cached relative to the first NA# for pipelined cycles. These timings hold for normal and late back-off modes.



NOTES:
A Cache line fill cycle
S Snoop (inquiry) cycle
R Addresses of cache line fill and snoop are the same
1. Earliest EADS# assertion that can invalidate line being filled

Figure 5.21. Snoop Responsibility Pickup (Nonpipelined Cycle)



NOTES:
A Cache line fill cycle
B Next cycle (any type)
S Snoop (inquiry) cycle
R Addresses of cycles A and S are the same
1. Earliest EADS# assertion that can invalidate line being filled

Figure 5.22. Snoop Responsibility Pickup (Pipelined Cycle)

5.3.2.2 Inquiry for a Line Being Replaced

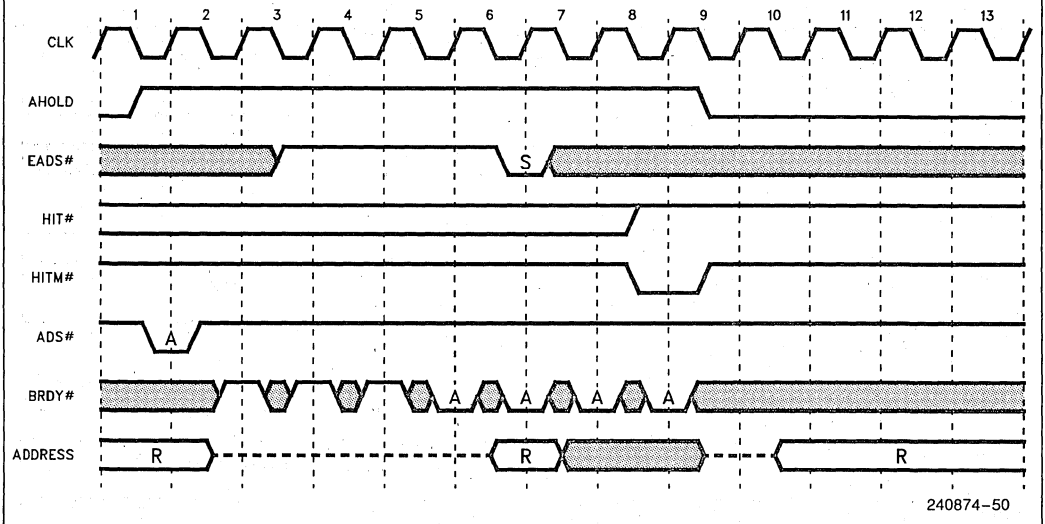
When the i860 XP microprocessor is replacing a line, there are two cases:

1. If the replacement does not require write-back, the address being replaced can be matched by an inquiry until assertion of NA# or first BRDY# of the line-fill cycle. From that point on, the inquiry has no effect.
2. If the replacement requires a write-back, the address being replaced can be matched by an inquiry until assertion of the last BRDY# for the write-back. An EADS# as late as two clocks before the last BRDY# can cause HITM# to be asserted.

Figures 5.23 through 5.25 show when the i860 XP microprocessor drops responsibility for recognizing inquiries for a line that it is writing back. They show the latest EADS# assertion that can cause HITM# assertion. In late back-off mode, EADS# can be asserted later, because BRDY# is internally delayed (Figures 5.24 and 5.25).

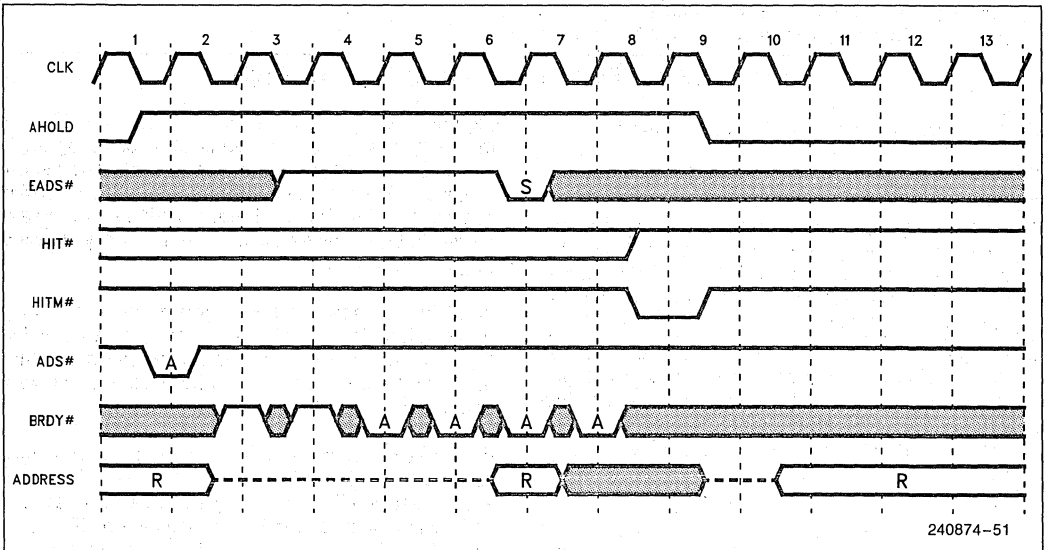
In all these cases, HITM# remains active for only one clock period. HITM#, as always, remains active through the last BRDY# of the corresponding write-back; in these cases the write-back has already completed.

If an inquiry cycle hits the write-back address after its ADS# has been issued, the i860 XP microprocessor asserts HITM#; however, HIT# is deasserted. This unique combination of values on HIT# and HITM# indicates that the write-back cycle corresponding to the HITM# has already been issued.



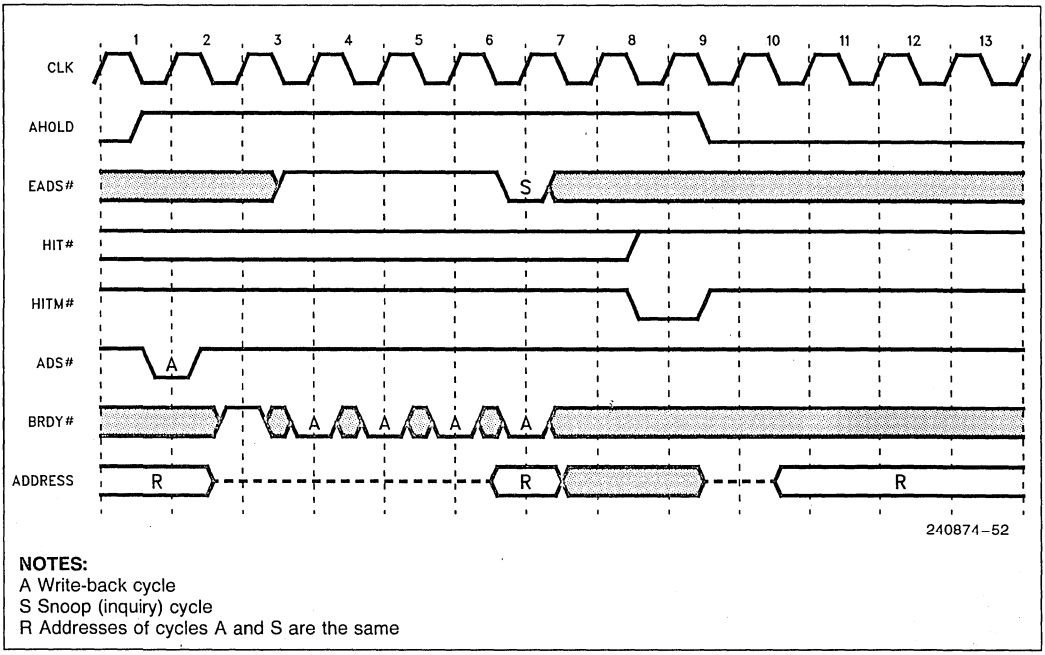
NOTES:
A Write-back cycle
S Snoop (inquiry) cycle
R Addresses of cycles A and S are the same

Figure 5.23. Latest Snooping of Write-Back (Not Late Back-Off Mode)



NOTES:
A Write-back cycle
S Snoop (inquiry) cycle
R Addresses of cycles A and S are the same

Figure 5.24. Latest Snooping of Write-Back (One-Clock Late Back-Off Mode)



NOTES:
A Write-back cycle
S Snoop (inquiry) cycle
R Addresses of cycles A and S are the same

Figure 5.25. Latest Snooping of Write-Back (Two-Clock Late Back-Off Mode)

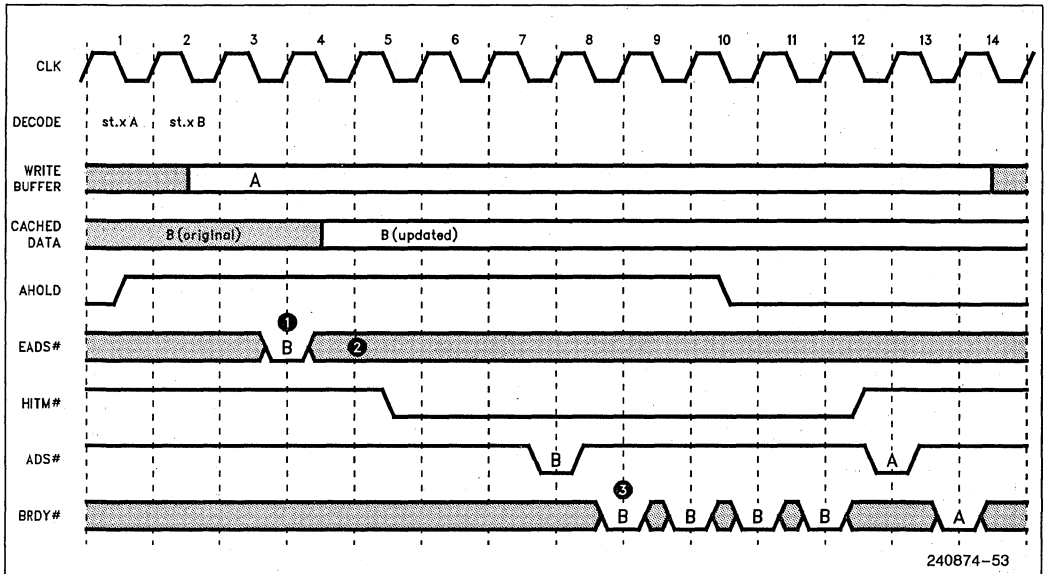
5.3.3 WRITE CYCLE REORDERING DUE TO BUFFERING

The MESI cache protocol and the ability to perform and respond to inquiry cycles guarantee that writes to the cache are logically equivalent to writes that go to memory. In particular, the *order* of read and write operations on cached data is the same as if the operations were on data in memory. Even uncached memory read and write requests usually occur on the external bus in the same order that they are issued in the program. For example, when a write miss is followed by a read miss, the write data goes onto the bus before the read request is put on the bus. However, the posting of writes in write buffers coupled with inquiry cycles may cause the order of writes seen on the external bus to differ from the order they appear in the program. Consider the following example, which is illustrated in Figure 5.26:

1. Three bus cycles are outstanding.
2. Processor 1 executes a store to address A, which misses the cache. This store is posted; that is, the data is latched in the write buffer while the processor continues execution without waiting for the store to be completed on the bus. In this case the store is not even put on the bus because there are already three outstanding cycles.
3. Processor 1 executes a store to address B, which hits the cache.
4. Processor 2 executes an inquiry for address B. Processor 1 looks in its cache, finds the modified line, asserts HIT# and HITM#, and executes a write-back cycle to address B, while the data for address A is still in the write buffer.
5. Processor 1 issues the write to address A on the bus.

In this example, the original order of the writes has been changed. In most cases it is not necessary that the ordering of writes be strictly maintained. But there are cases (for example, semaphore updates in a multiprocessor system) that require stores to be observed externally in the same order as programmed. There are several ways to ensure serialization of stores:

1. Bracket one of the stores with the **lock** and **unlock** instructions. That forces serialization of the stores (refer to section 5.4). In the above example of a store-miss followed by store-hit, locking either store would ensure that the internal store-hit does not update the cache until the miss gets to the external bus.
2. Apply the write-through policy to the critical data, by setting WT = 1 in the page table entries or by driving the WB/WT# pin low.



NOTES:
A Data written by **st.x A** instruction
B Data written by **st.x B** instruction
1. Snoop for address of B
2. Snoop look-up in tag array occurs here; finds B modified
3. Write-back of line containing B occurs before write of A

Figure 5.26. Write Reordering due to Buffering

3. Configure the processor for Strong Ordering Mode by asserting EWBE# during RESET.

Option 1 is implementable by user-level programs, while option 2 is an operating-system level solution, not directly implementable by user-level code. Option 3, the hardware solution, is discussed in greater detail in section 5.3.4.

5.3.4 STRONG ORDERING MODE

In strong ordering mode, the processor delays updates to its internal data cache in either of these conditions:

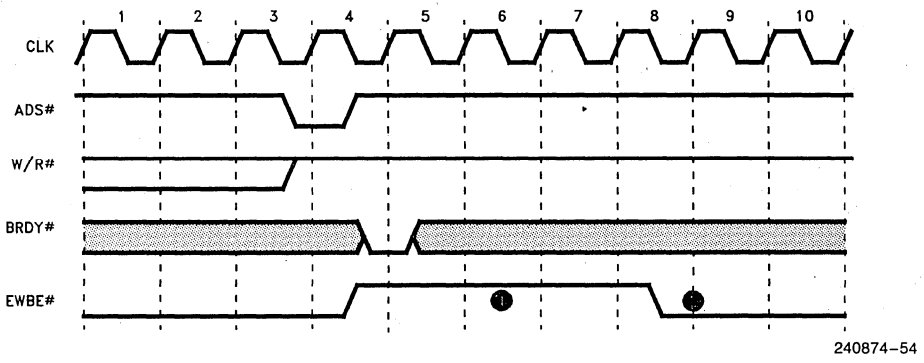
1. The internal write buffer is not empty.
2. An external write buffer is not empty (the external system signals this condition by deactivating the EWBE# signal).

By delaying the cache update until all write buffers are empty, the i860 XP microprocessor avoids the out-of-order sequence shown in section 5.3.3.

In strong ordering mode, EWBE# can be deasserted only between the ADS# and the last BRDY# of a store. The earliest deassertion is the clock after ADS#; the latest deassertion is together with the last BRDY#. EWBE# can be reasserted at any time, except when the processor is performing an inquiry write-back. In other words, EWBE# must not activate while HITM# is active. When EWBE# goes active, the processor completes any cache update that may have been delayed by its deassertion.

Figure 5.27 shows how an external cache can use EWBE# when a store miss in the i860 XP microprocessor is also a miss in the external cache.

An external cache controller should also refrain from updating the external cache while EWBE# is active.



NOTE:
1. Assumes the external cache needs five cycles to write the data to memory.
2. Pending internal data cache updates are delayed until the clock in which EWBE# is sampled LOW.

Figure 5.27. Timing of EWBE#

5.3.5 SCHEDULING INQUIRY WRITE-BACK CYCLES

In order to preserve system-wide ordering of memory transactions in multiprocessor systems that have a pipelined or split-transaction memory bus, it may be necessary to get the data corresponding to an inquiry hit before outstanding bus cycles are completed. Another bus master can always request an inquiry while the i860 XP microprocessor has cycles outstanding on the bus. However, when AHOLD is asserted, the i860 XP microprocessor normally completes outstanding cycles before it performs any write-back that may be required. The i860 XP microprocessor provides two methods for causing the inquiry write-back before outstanding cycles are completed:

FLINE# When FLINE# is asserted during the EADS# of an inquiry that hits an M-state line, the i860 XP microprocessor issues a write-back cycle and writes the dirty line to memory before the outstanding bus cycles are completed.

BOFF# If there are outstanding cycles on the bus, asserting BOFF# clears the bus pipeline. If an inquiry causes HITM# to be asserted, then the first cycle issued by the i860 XP microprocessor after deassertion of BOFF# is the inquiry write-back cycle. After the inquiry write-back, it reissues the aborted cycles.

5.3.5.1 Choosing between FLINE# and BOFF#

FLINE#, although the more efficient choice, cannot handle all situations. Under certain circumstances, it can happen that outstanding stores on the bus correspond to data that is obsolete relative to the data in the cache, because a subsequent store has updated the cache after the ADS# for the outstanding store has occurred. For example:

- An *aliasing store hit*, in which a cache virtual-tag miss occurs and the ADS# is issued at the same time as a physical-tag hit. Then the cached data would be updated before external memory, and a subsequent store to the new virtual address could also update cache before the outstanding bus store completed.
- *Back-to-back writes* to the same line can also update the cache more recently than the bus when the write-once update policy is employed. The first write updates the cache and generates a bus write request, but the second write only updates the cache.

In both of these examples the outstanding stores on the bus are obsolete relative to the data in the cache line. If an inquiry cycle hits a line and this line is written back out of order (that is, before outstanding stores are completed), special care should be taken to discard the outstanding stores.

The easiest way to avoid this situation is not to assert FLINE# when stores are outstanding, but use BOFF# instead. If out-of-order write-back is implemented with BOFF#, the i860 XP microprocessor does not restart the outstanding store to that line if such a store has been obsoleted by a later cache hit store. That is, the i860 XP microprocessor detects this condition and kills the obsolete data. However, lock-bracketed stores (including the last store in the lock sequence) are restarted by the i860 XP microprocessor, because lock-bracketed stores update the cache only after BRDY# is returned.

If, on the other hand, out-of-order write-back is implemented by using only the FLINE# pin, the external system must return BRDY#s for outstanding stores, but the data must be ignored if it has already been written out by an inquiry write-back.

Note that if a replacement write-back is in progress (ADS# has been issued, but last BRDY# has not occurred) and an inquiry hits the same line that is being written back, the FLINE# pin is ignored. The system can recognize this special case by the fact that HITM# is asserted while HIT# is deasserted. If other cycles are outstanding and it is necessary to write the line back before the other cycles, BOFF# can be used.

5.3.5.2 Reordering Write-Backs with FLINE#

FLINE# must be active during the EADS# that initiates an inquiry. BRDY# must not be asserted forthe previously issued cycles while HITM# is active. If HITM# is asserted while the data transfer of the outstanding cycle is in progress (i.e. first BRDY# has been asserted, but the entire transfer has not yet been completed), the i860 XP microprocessor waits for the current cycle to complete, and only then issues the write-back. After the last BRDY# for the ongoing burst (if any), BRDY# is ignored until the clock period after ADS# is asserted for the write-back.

From the viewpoint of the i860 XP microprocessor, an inquiry write-back cycle is just another bus cycle; so, if there is an outstanding cycle at the time of FLINE# and HITM# activation, the system must assert NA# to initiate the write-back.

Figure 5.28 illustrates simple cycle reordering, when FLINE# is not asserted during the data transfer of another cycle. The outstanding request could be either a read or write.

Figure 5.29 shows the case in which FLINE# is asserted after data transfer for the outstanding cycle has already started. In this case, the i860 XP microprocessor does not issue a write-back until the outstanding transfer is completed. NA# is needed in this example only if other outstanding cycles remain.

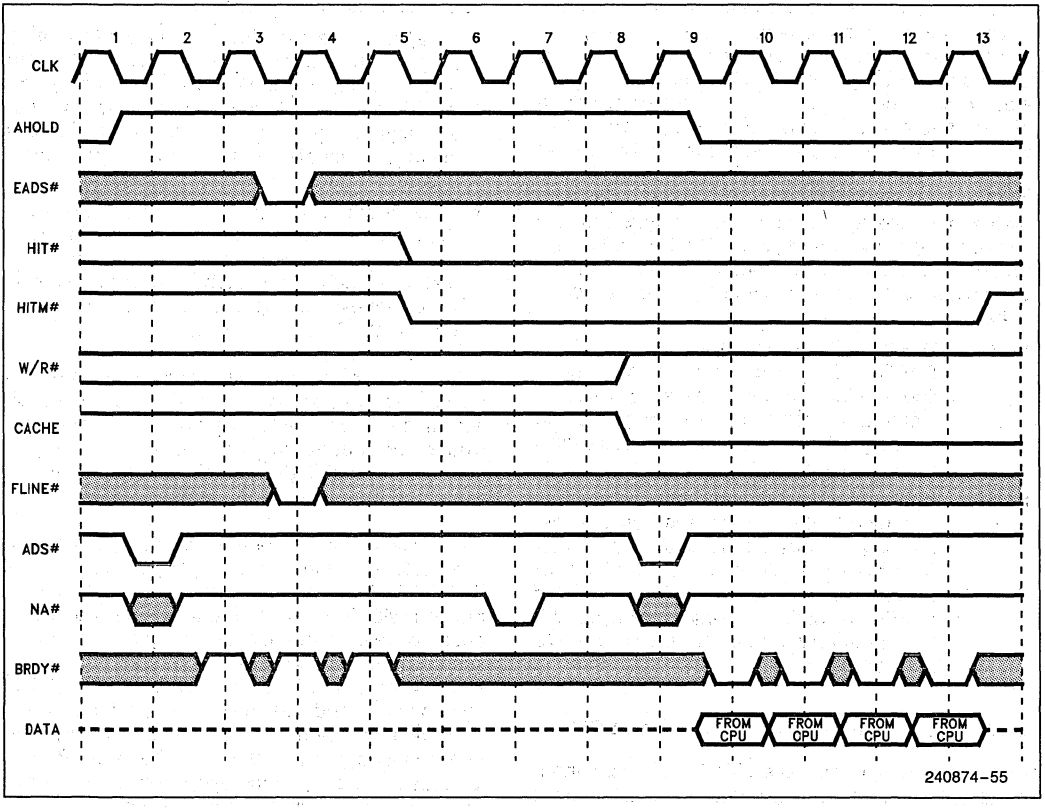
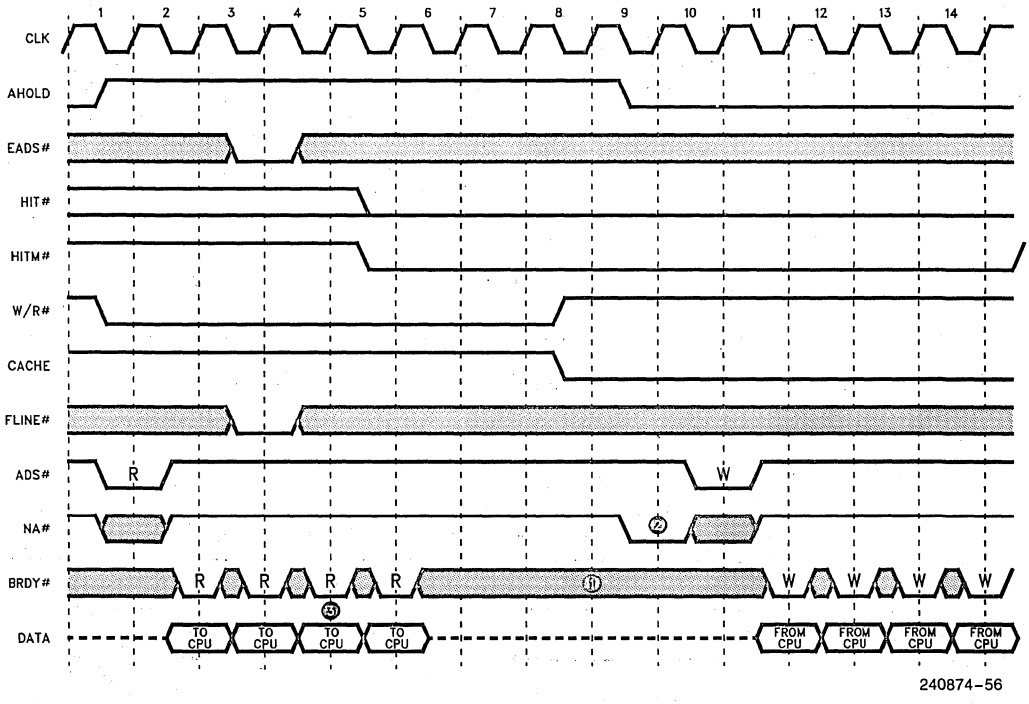


Figure 5.28. Cycle Reordering via FLINE# (No Ongoing Burst)



NOTES:
1. BRDY# is ignored by CPU from end of ongoing burst through ADS# of write-back, even if other cycles remain outstanding
2. NA# required only if another cycle is outstanding
3. If the first BRDY# is asserted here or sooner (relative to HITM#), the outstanding cycle completes before the FLINE# write-back.

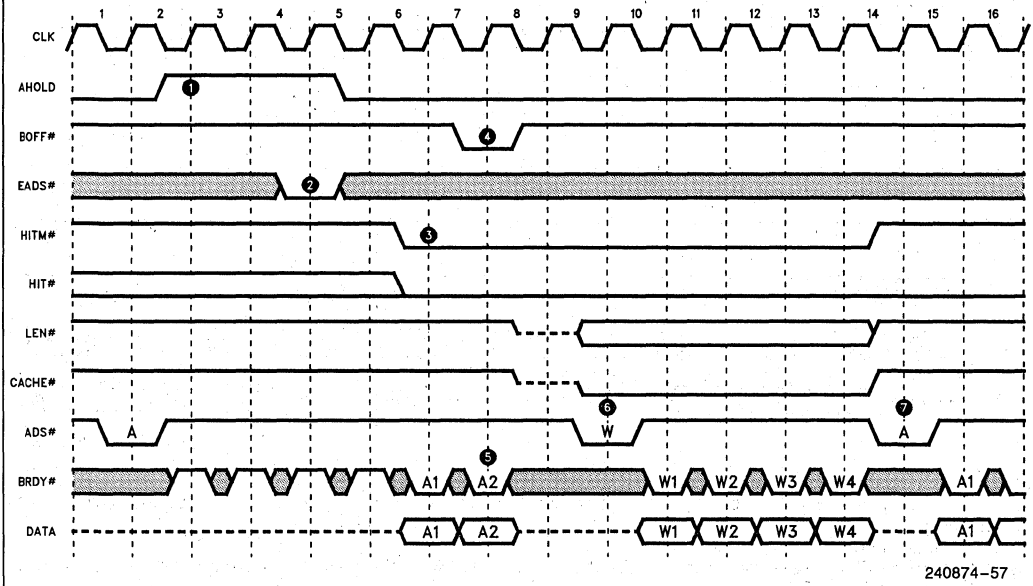
Figure 5.29. Cycle Reordering via FLINE# (Ongoing Burst)

5.3.5.3 Reordering Write-Backs with BOFF#

Back-off cycles are discussed in general in Section 5.2.2. Figure 5.30 shows how BOFF# can be used to cancel outstanding cycles so that an inquiry write-back can take place immediately.

5.4 The LOCK# Cycle Attribute

The processor asserts the LOCK# signal when several accesses to a single memory location must be effectively uninterruptible. By causing LOCK# to be asserted, a programmer can, for example, increment the contents of a memory variable and be assured that the variable will not be accessed between the read and the update of that variable.



NOTES:
A Outstanding cycle (for example, noncacheable 128-bit read) W Write-back cycle
1. AHOLD begins an inquiry while one cycle is outstanding.
2. Earliest assertion of EADS# is two clocks after assertion of AHOLD
3. Inquiry hits modified line.
4. Assertion of BOFF# aborts the outstanding cycle.
5. BRDY# asserted during BOFF# is ignored by CPU.
6. Write-back begins after deassertion of BOFF#.
7. Earliest assertion of ADS# for restart of cycle A (assuming no pipelining).

Figure 5.30. Cycle Reordering via BOFF# (Ongoing Burst)

The memory location to be locked is the one whose address is driven during the cycle in which LOCK# is first activated. In multiprocessor systems, external hardware should guarantee that no other processor is granted a locked read, locked write, or unlocked write to the same location until LOCK# is deasserted. The i860 XP microprocessor has no hardware provision to prevent another master from also locking the variable; this responsibility falls on the bus arbiter. In the simplest implementation, the arbiter can globally prevent other masters from accessing the bus.

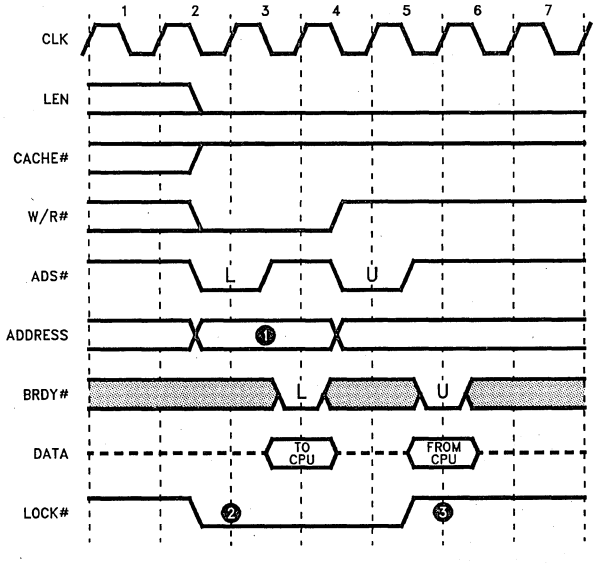
Not all cycles affect the value of LOCK#. Code fetches, write-backs due to replacement or inquiry, and cycles restarted due to BOFF# do not affect LOCK#. Any other type of cycle can be used to initiate or terminate LOCK#, including cache line fills, interrupt acknowledge, I/O, and special cycles.

Data accesses with LOCK# asserted are not pipelined, and other data cycles are not pipelined while a LOCK# cycle remains outstanding. Instruction fetches, however, may be pipelined during lock.

The i860 XP microprocessor can run very long lock sequences; therefore, to guarantee reasonable bus turnover latency in multimaster systems, the i860 XP microprocessor recognizes bus hold (HOLD), address hold (AHOLD), and back-off (BOFF#) while the LOCK# signal is active. In spite of such intervening conditions, the arbiter should prevent any other bus master from also locking or updating the variable the i860 XP microprocessor locked. In simple systems the HOLD input can be masked by the LOCK# output (that is, the external logic that generates HOLD can AND the LOCK# signal with other hold conditions). More sophisticated systems, however, may allow the bus to be turned over while LOCK# is asserted.

Whatever the lock implementation, arbiter design must, in one case, allow another processor to write the locked variable. That case is when another i860 XP microprocessor or master asserts HITM# in response to the inquiry generated by the locking processor's initial read. That other master must write back the locked variable before the i860 XP microprocessor can read it. This HITM# write-back must always be allowed.

The timing of LOCK# is shown in Figure 5.31. Note that LOCK# is asserted in the same clock period as ADS# for the locked address, but is deasserted in the clock period *after* ADS# for the unlocking load or store.



NOTES:
L Locking access
U Unlocking access
1. This address is to be locked
2. LOCK# is asserted with ADS#
3. LOCK# is deasserted one clock after ADS#

Figure 5.31. LOCK# Timing

5.5 RESET Initialization

Initialization of the i860 XP microprocessor is caused when the system asserts the RESET signal for at least ten clocks. Table 5.5 shows the status of output pins during the time that RESET is asserted. Note that the bidirectional data pins (D63–D0 and DP7–DP0) are floated during RESET, though the bidirectional A31–A3 pins are not. If the i860 XP microprocessor is used with 82495XP and 82496XP cache, however, the latter do float the bidirectional pins they share with i860 XP microprocessor during RESET. Note that HOLD requests are honored during RESET and that the HLDA output signal may also become active. The status of output pins depends on whether a HOLD request is being acknowledged. Note also that the test logic may be active during RESET and that the EXTEST instruction may drive other values on the output pins.

After the RESET signal goes inactive the processor remains in the RESET state for three more clocks. Applications that use the HOLD signal to float the bus during RESET should keep HOLD active for three more clocks after the RESET signal is deactivated.

Some aspects of processor configuration are determined by asserting input signals during RESET. To select a given option, the corresponding input must be asserted for at least the last three clocks before the falling edge of RESET; to deselect, the corresponding input must be deasserted for at least the last three clocks before the falling edge of RESET:

EWBE# Enter strong ordering mode.

FLINE# Enter one clock late back-off mode.

INT/CS8 Enter eight-bit code-size mode.

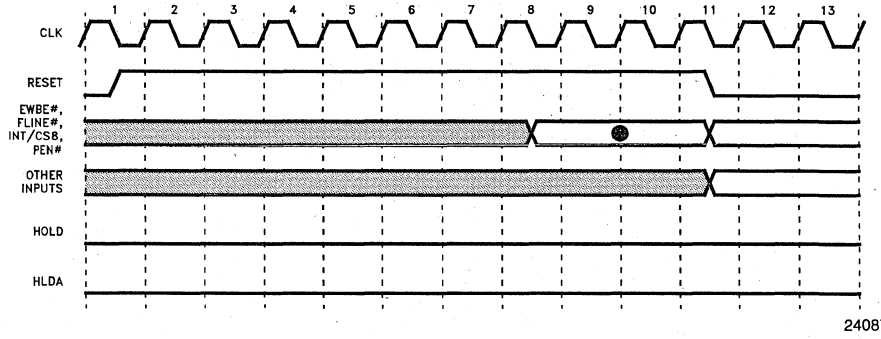
PEN# Enter normal (small output buffers) current mode.

Figure 5.32 shows how configuration pins are sampled during the three clock periods just before the falling edge of RESET. No inputs besides EWBE#, HOLD, FLINE#, INT/CS8, and PEN# are sampled during RESET.

Table 5.5. Output Pin Status during Reset

Pin Name	Pin Value	
	HOLD Not Acknowledged	HOLD Acknowledged
BREQ	LOW	LOW
HLDA	LOW	HIGH
W/R#, PWT, PCD	LOW	Tristate OFF
ADS#	HIGH	Tristate OFF
D63–D0, DP7–DP0	Tristate OFF	Tristate OFF
A31–A3, BE7#0–BE0#, NENE# CACHE#, CTYP, D/C#, KB0, KB1, LEN, M/IO#, PCYC	Undefined	Tristate OFF
PCHK#, HIT#	Undefined	Undefined
HITM#, LOCK#	HIGH	HIGH

NOTE:
This table does not apply if the test logic is running the EXTEST instruction.



NOTE:
1. The CPU samples these inputs in the clocks preceding the falling edge of reset.

Figure 5.32. Reset Activities

While in eight-bit code-size mode, instruction cache misses are one-byte reads (transferred on D7–D0 of the data bus) instead of eight-byte reads. This allows the i860 XP microprocessor to be bootstrapped from an eight-bit ROM. For these code reads, byte enables BE2#–BE0# are redefined to be the low order three bits of the address, so that a complete byte address is available. The entire eight-byte data bus continues to be parity-checked by the i860 XP microprocessor during CS8-mode instruction fetches; therefore, external hardware must either generate good parity on all eight bytes or disable parity traps by deasserting PEN# during CS8 mode.

While in this mode, instructions must reside in an eight-bit wide memory, while data must reside in a separate 64-bit wide memory. After the code has been loaded into 64-bit memory, initialization code can initiate 64-bit code fetches by clearing the CS8 bit of the **dirbase** register (refer to section 2). Once eight-bit code-size mode is disabled by software, it cannot be reenabled except by resetting the i860 XP microprocessor.

Instruction fetches in CS8 mode update the instruction cache if KEN# is asserted during NA# or all of the first eight BRDY#s (refer to section 4.2.26). They are pipelined if NA# is asserted. When used with the 82495XP and 82496XP cache, CS8 mode works only if the ROM locations are made non-cacheable.

6.0 TESTABILITY

The i860 XP microprocessor provides testability features compatible with the proposed *Standard Test Access Port and Boundary-Scan Architecture* (IEEE Std. P1149.1/D6). The subset of the standard test logic implemented in the i860 XP microprocessor provides for testing the interconnections between the i860 XP microprocessor and other integrated circuits once they have been assembled onto a printed circuit board.

The test logic consists of a boundary-scan register and other building blocks that are accessed through a test access port (TAP). The TAP provides a simple serial interface that makes it possible to test all signal traces with only a few probes.

The TAP can be controlled by a bus master. The bus master can be either automatic test equipment or a component that interfaces to a four-pin test bus.

6.1 Test Architecture

The test logic contains the following elements:

- Test access port (TAP), which consists of input pins TMS, TCK, TDI, and TRST#; and output pin TDO.
- TAP controller, which receives the dedicated test clock (TCK) and interprets the signals on the test mode select (TMS) line. The TAP controller generates clock and control signals for the instruction and test data registers and for other parts of the test logic.
- Instruction register (IR), which allows instruction codes to be shifted into the test logic. The instruction codes are used to select the test to be performed or the test data register to be accessed.
- Test data registers: Bypass Register (BPR), Device Identification Register (DID), and Boundary-Scan Register (BSR).

The instruction and test data registers are separate shift-register paths connected in parallel and having a common serial data input and a common serial data output connected to the TAP TDI and TDO signals respectively.

6.2 Test Data Registers

The test logic contains the following data registers:

- **Bypass Register (BPR)**: BPR is a one-bit shift register that provides a minimum-length path between TDI and TDO when no test operation of the component is required. This allows more rapid movement of test data to and from other board components that are required to perform test operations. While running through BPR, the data is transferred without inversion from TDI to TDO.
- **Device Identification Register (DID)**: This register contains the manufacturer's identification code, part number code, and version code in the format shown by Figure 6.1. The values are: manufacturer's identification code (9), part number code (61A0), version code (8), entire 32-bit value (0x861A0013).
- **Boundary Scan Register (BSR)**: The BSR is a single shift-register path containing 150 cells that are connected to all input and output pins of the i860 XP microprocessor. Figure 6.2 shows the logical structure of the BSR. Input cells only capture data; they do not affect operation of the i860 XP microprocessor. Data is transferred without inversion from TDI to TDO through the BSR during scanning. The BSR can be operated by the EXTEST and SAMPLE instructions.

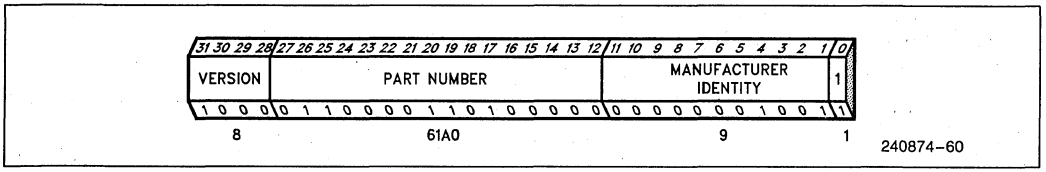


Figure 6.1. Format of DID Register

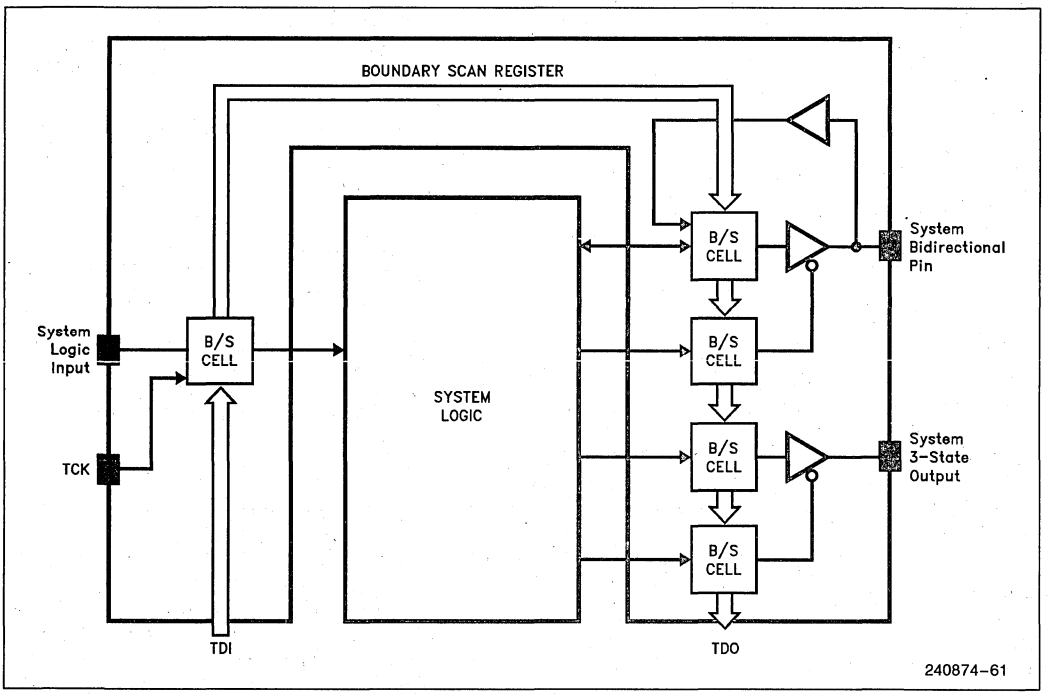


Figure 6.2. Logical Structure of BSR Register

6.3 Instruction Register

The Instruction Register (IR) selects the test to be performed and the test data register to be accessed. It is four bits wide, with no parity bit. Table 6.1 shows the encoding of the instructions supported by the TAP controller of the i860 XP microprocessor. The rightmost bit is the least significant and is the first shifted out on TDO.

Table 6.1. TAP Instruction Encoding

Instruction Code	Instruction
0000	EXTEST Boundary Scan
0001	SAMPLE Boundary Scan
0010	IDCODE
0011 . . . 1110	*Intel reserved* **CAUTION***
1111	BYPASS

* **CAUTION:** Operation of these private instructions may cause damage to the component.

EXTEST The BSR cells associated with output pins drive the output pins of the i860 XP microprocessor. Values scanned into the BSR cells become the output values. The BSR cells associated with input pins sample the inputs of the i860 XP microprocessor. Note that I/O pins can be input or output for this test, depending on their control setting. The values shifted to the input latches are not used by the internal logic of the i860 XP microprocessor. After use of the EXTEST command, the i860 XP microprocessor must be reset (with the RESET signal) before normal use.

SAMPLE The BSR cells associated with output pins sample the value driven by the i860 XP microprocessor. BSR cells associated with input pins sample on the rising edge of TCK the values driven to the i860 XP

microprocessor. BSR cells associated with I/O pins sample the value on the respective pin. The I/O pin can be driven by the i860 XP microprocessor or by external hardware. The values shifted to the input latches are not used by the internal logic of the i860 XP microprocessor.

IDCODE The identification code of the i860 XP microprocessor from the DID register is passed to TDO. The DID register is not altered by data shifted in on TDI.

BYPASS Test data is passed from TDI to TDO via the single-bit BPR, effectively bypassing the test logic of the i860 XP microprocessor. Because of its special encoding, this instruction can be entered by holding TDI HIGH while completing an instruction-scan cycle. This reduces the demands on the host test system in cases where access is required, for example, only to chip 57 on a 100-chip board.

Note that an open circuit fault in the board-level test data path causes the BPR register to be selected following an instruction-scan cycle, because the TDI input has a pull-up resistor. Therefore, no unwanted interference with the operation of the on-chip system logic can occur.

Table 6.2 defines which registers are active during execution of each instruction.

6.4 TAP Controller

The TAP Controller is a synchronous, finite state machine. It controls the sequence of operations of the test logic. The TAP Controller changes state only in response to the following events:

1. A rising edge of TCK.
2. A transition to logic zero at the TRST# input.
3. Power-up.

The value of the TMS input signal at a rising edge of TCK controls the sequence of state changes. The state diagram for the TAP controller is shown in Figure 6.3. Test designers must consider the operation of the state machine in order to design the correct sequence of values to drive on TMS.

6.4.1 TEST-LOGIC-RESET STATE

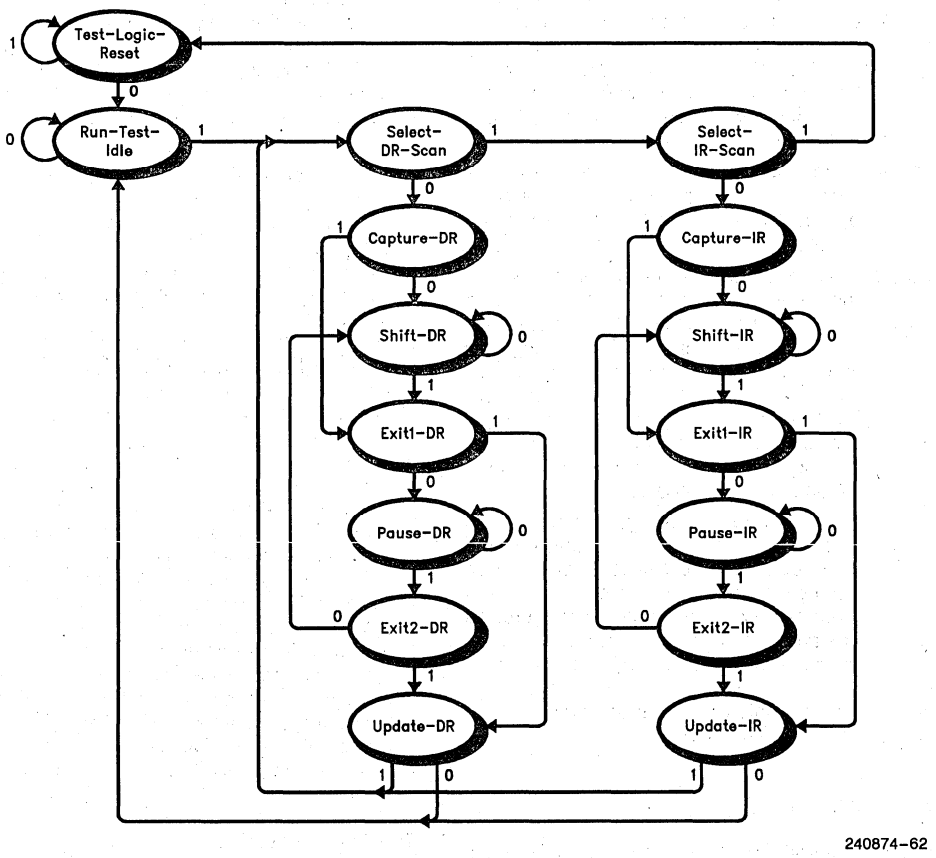
In this state, the test logic is disabled so that normal operation of the i860 XP microprocessor can continue unhindered. This is achieved by initializing the instruction register such that the IDCODE instruction is loaded. No matter what the original state of the controller, the controller enters *Test-Logic-Reset* when the TMS input is held HIGH for at least five rising edges of TCK. The controller remains in this state while TMS is HIGH.

If the controller leaves the *Test-Logic-Reset* state as a result of an erroneous LOW signal on the TMS line at the time of a rising edge of TCK (for example, a glitch due to external interference), it returns to the *Test-Logic-Reset* state following three rising edges of TCK while the TMS signal at the intended HIGH logic level. The operation of the test logic is such that no disturbance is caused to on-chip system logic operation as the result of such an error. On leaving the *Test-Logic-Reset* state, the controller moves into the *Run-Test/Idle* state, where no action occurs because the current instruction has been set to select operation of the DID register. The test logic is also inactive in the *Select-DR-Scan* and *Select-IR-Scan* states.

The TAP controller is also forced to the *Test-Logic-Reset* state by applying a LOW logic level to the TRST# input and at power-up.

Table 6.2. Registers Active by Instruction

Mode	Register		
	BSR	DID	BPR
EXTEST	TDI → BSR → TDO	Inactive	Inactive
SAMPLE	TDI → BSR → TDO	Inactive	Inactive
IDCODE	Inactive	DID → TDO	Inactive
BYPASS	Inactive	Inactive	TDI → BPR → TDO



NOTE:
0,1 The values present on TMS at the time of a rising edge on TCK.

Figure 6.3. TAP Controller State Diagram

6.4.2 RUN-TEST/IDLE STATE

The controller enters this state between scan operations. Once in this state, the controller remains in this state as long as TMS is held LOW. No activity occurs in the test logic. The instruction register and all test data registers retain their previous state. When TMS is HIGH and a rising edge is applied to TCK, the controller moves to the *Select-DR-Scan* state.

6.4.3 SELECT-DR-SCAN STATE

This is a temporary controller state. The test data register selected by the current instruction retains its previous state. If TMS is held LOW and a rising edge is applied to TCK when in this state, the controller moves into the *Capture-DR* state, and a scan sequence for the selected test data register is initiated. If TMS is held HIGH and a rising edge is applied to TCK, the controller moves to the *Select-IR-Scan* state.

The instruction does not change in this state.

6.4.4 SELECT-IR-SCAN STATE

This is a temporary controller state. The test data register selected by the current instruction retains its previous state. If TMS is held LOW and a rising edge is applied to TCK when in this state, the controller moves into the *Capture-IR* state, and a scan sequence for the instruction register is initiated. If TMS is held HIGH and a rising edge is applied to TCK, the controller moves to the *Test-Logic-Reset* state.

The instruction does not change in this state.

6.4.5 CAPTURE-DR STATE

In this state, the BSR captures input pin data if the current instruction is EXTEST or SAMPLE. The other test data registers, which do not have parallel input, are not changed.

The instruction does not change in this state.

When the TAP controller is in this state and a rising edge is applied to TCK, the controller enters the *Exit1-DR* state if TMS is HIGH or the *Shift-DR* state if TMS is LOW.

6.4.6 SHIFT-DR STATE

In this controller state, the test data register connected between TDI and TDO as a result of the current instruction shifts data one stage toward its serial output on each rising edge of TCK.

The instruction does not change in this state.

When the TAP controller is in this state and a rising edge is applied to TCK, the controller enters the *Exit1-DR* state if TMS is HIGH or remains in the *Shift-DR* state if TMS is LOW.

6.4.7 EXIT1-DR STATE

This is a temporary state. If TMS is held HIGH, a rising edge applied to TCK while in this state causes the controller to enter the *Update-DR* state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the *Pause-DR* state.

The test data register selected by the current instruction retains its previous state unchanged. The instruction does not change in this state.

6.4.8 PAUSE-DR STATE

The pause state allows the test controller to temporarily halt the shifting of data through the test data register in the serial path between TDI and TDO. This might be necessary, for example, to allow the tester to reload its pin memory from disk during application of a long test sequence.

The test data register selected by the current instruction retains its previous state. The instruction does not change in this state.

The controller remains in this state as long as TMS is LOW. When TMS goes HIGH and a rising edge is applied to TCK, the controller moves to the *Exit2-DR* state.

6.4.9 EXIT2-DR STATE

This is a temporary state. If TMS is held HIGH and a rising edge is applied to TCK, the scanning process terminates, and the TAP controller enters the *Update-DR* state. If TMS is held LOW and a rising edge is applied to TCK, the controller enters the *Shift-DR* state.

The test data register selected by the current instruction retains its previous state unchanged. The instruction does not change in this state.

6.4.10 UPDATE-DR STATE

The BSR register is provided with a latched parallel output to prevent changes at the parallel output while data is shifted in response to the EXTEST and SAMPLE instructions. When the TAP controller is in this state and the BSR register is selected, data is latched onto the parallel output of this register from the shift-register path on the falling edge of TCK. The data held at the latched parallel output does not change other than in this state.

All shift-register stages in test data registers selected by the current instruction retain their previous state unchanged. The instruction does not change in this state.

When the TAP controller is in this state and a rising edge is applied to TCK, the controller enters the *Select-DR-Scan* state if TMS is held HIGH or the *Run-Test/Idle* state if TMS is held LOW.

6.4.11 CAPTURE-IR STATE

In this controller state the shift register contained in the instruction register loads the fixed value 0001 on the rising edge of TCK.

The test data register selected by the current instruction retains its previous state. The instruction does not change in this state.

When the controller is in this state and a rising edge is applied to TCK, the controller enters the *Exit1-IR* state if TMS is held HIGH or the *Shift-IR* state if TMS is held LOW.

6.4.12 SHIFT-IR STATE

In this state, the shift register contained in the instruction register is connected between TDI and TDO and shifts data one stage towards its serial output on each rising edge of TCK.

The test data register selected by the current instruction retains its previous state. The instruction does not change in this state.

When the controller is in this state and a rising edge is applied to TCK, the controller enters the *Exit1-IR* state if TMS is held HIGH or remains in the *Shift-IR* state if TMS is held LOW.

6.4.13 EXIT1-IR STATE

This is a temporary state. If TMS is held HIGH, a rising edge applied to TCK while in this state causes the controller to enter the *Update-IR* state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the *Pause-IR* state.

The test data register selected by the current instruction retains its previous state unchanged. The instruction does not change in this state, and the instruction register retains its state.

6.4.14 PAUSE-IR STATE

This state allows the shifting of the instruction register to be temporarily halted.

The test data register selected by the current instruction retains its previous state. The instruction does not change in this state, and the instruction register retains its state.

The controller remains in this state as long as TMS is LOW. When TMS goes HIGH and a rising edge is applied to TCK, the controller moves to the *Exit2-IR* state.

6.4.15 EXIT2-IR STATE

This is a temporary state. If TMS is held HIGH and a rising edge is applied to TCK, the scanning process terminates, and the TAP controller enters the *Update-IR* state. If TMS is held LOW and a rising edge is applied to TCK, the controller enters the *Shift-IR* state.

The test data register selected by the current instruction retains its previous state unchanged. The instruction does not change in this state, and the instruction register retains its state.

6.4.16 UPDATE-IR STATE

The instruction shifted into the instruction register is latched onto the parallel output from the shift-register path on the falling edge of TCK. Once the new instruction has been latched, it becomes the current instruction.

Test data registers selected by the current instruction retain the previous state.

6.5 Boundary Scan Register Cell Ordering

Figure 6.4 shows the order of cells in the BSR. There are 150 cells including TDO. TDI is *not* a BSR cell.

The DCTL, ACTL, TCTL, and OCTL cells do not correspond to pins of the i860 XP microprocessor; rather, they control the bidirectional and tristate pins:

DCTL D63–D0, DP7–DP0

ACTL A31–A3

TCTL Tristate outputs: ADS#, BE7#–BE0#, CACHE#, CTYP, D/C#, KB0, KB1, LEN, M/IO#, NENE#, PCD, PCYC, PWT, W/R#

OCTL Outputs not floated in normal operation: BREQ, HIT#, HITM#, HLDA, LOCK#, PCHK#

If a value of one is loaded into any of these control latches, the associated pins will not drive the external bus while running EXTEST.

The values of DCTL, ACTL, TCTL, and OCTL are *undefined* during the SAMPLE instruction.

The values and direction of I/O and outputs do not change during the scanning process (that is, during *Shift-DR* states). They only change after scanning is completed (in the *Update-DR* state).

The decision table, Table 6.3, defines how the boundary scan instructions EXTEST and SAMPLE/PRELOAD utilize BSR.

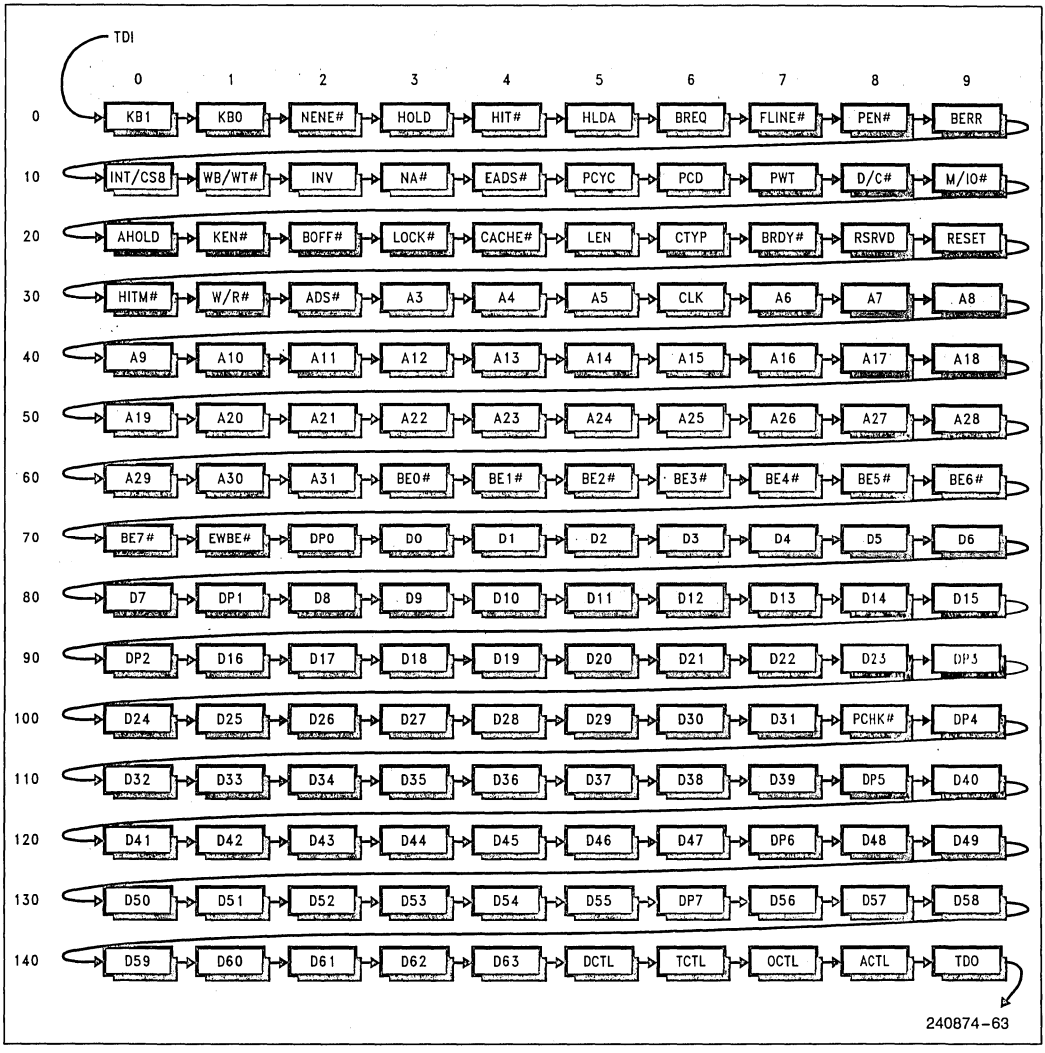


Figure 6.4. Boundary Scan Register Ordering

6.6 TAP Controller Initialization

TAP can be initialized by applying a high signal level on the TMS input for five periods of TCK or by activating the TRST# input pin. TCK does not have to be running in order to initialize TAP with the TRST# pin. TRST# is provided with an internal pull-up resistor; so, even if an open circuit fault occurs, the TAP logic can still be used.

7.0 MECHANICAL DATA

Figures 7.1 and 7.2 show the locations of pins; Tables 7.1 and 7.2 help to locate pin identifiers.

Table 6.3. Instruction Functions

Instruction:	**EXTEST**		**SAMPLE/PRELOAD**	
Control Cell:	**LOW**	**HIGH**	**LOW**	**HIGH**
Input BSR cells . . .	. . . sample values driven to processor by system		. . . sample values driven to processor by system	
Values of input cells used by processor?	NO		NO	
Output BSR cells . . .	. . . drive output pins with cell values		. . . sample values driven by processor	
Input/output BSR cells:	Treat as output	Treat as input	Treat as output	Treat as input

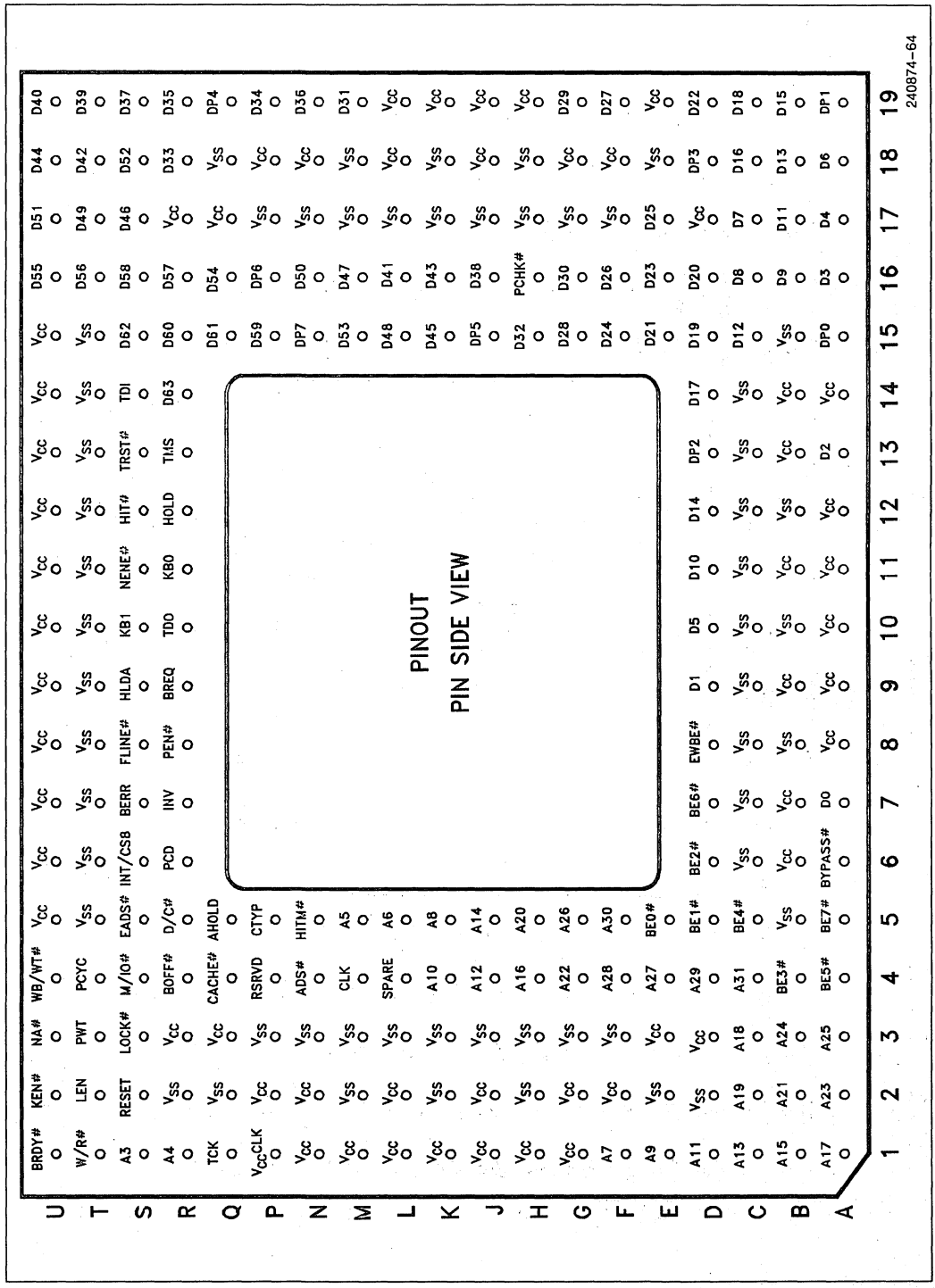


Figure 7.1. i860™ XP Microprocessor Pin Configuration—View from Pin Side

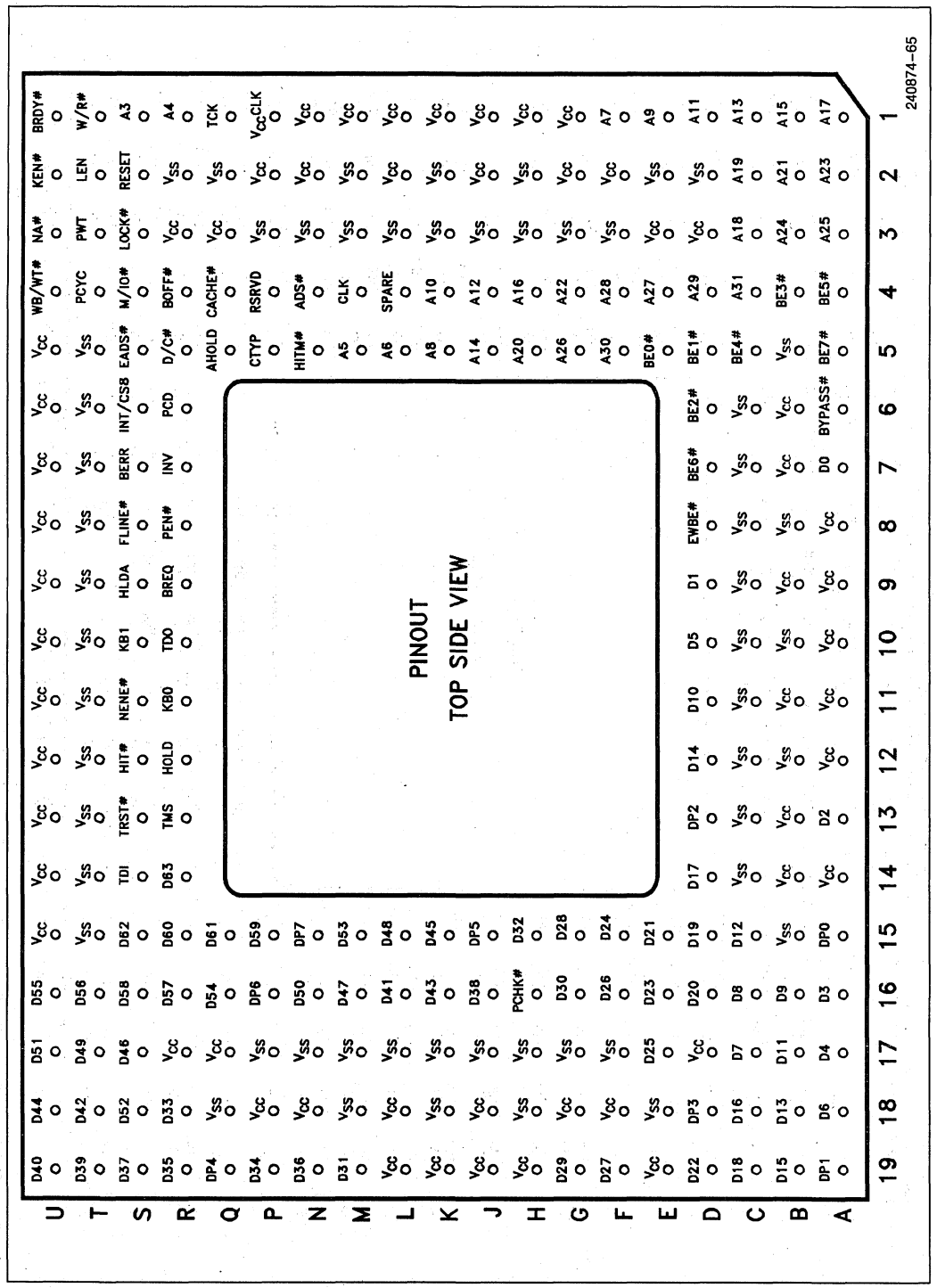


Figure 7.2. i860™ XP Microprocessor Pin Configuration—View from Top Side

Table 7.1. Pin Cross Reference by Location

Location	Signal	Location	Signal	Location	Signal	Location	Signal
A01	A17	C15	D12	G18	V_{CC}	N01	V_{CC}
A02	A23	C16	D8	G19	D29	N02	V_{CC}
A03	A25	C17	D7	H01	V_{CC}	N03	V_{SS}
A04	BE5#	C18	D16	H02	V_{SS}	N04	ADS#
A05	BE7#	C19	D18	H03	V_{SS}	N05	HITM#
A06	BYPASS#	D01	A11	H04	A16	N15	DP7
A07	D0	D02	V_{SS}	H05	A20	N16	D50
A08	V_{CC}	D03	V_{CC}	H15	D32	N17	V_{SS}
A09	V_{CC}	D04	A29	H16	PCHK#	N18	V_{CC}
A10	V_{CC}	D05	BE1#	H17	V_{SS}	N19	D36
A11	V_{CC}	D06	BE2#	H18	V_{SS}	P01	V_{CC} CLK
A12	V_{CC}	D07	BE6#	H19	V_{CC}	P02	V_{CC}
A13	D2	D08	EWBE#	J01	V_{CC}	P03	V_{SS}
A14	V_{CC}	D09	D1	J02	V_{CC}	P04	RSRVD
A15	DP0	D10	D5	J03	V_{SS}	P05	CTYP
A16	D3	D11	D10	J04	A12	P15	D59
A17	D4	D12	D14	J05	A14	P16	DP6
A18	D6	D13	DP2	J15	DP5	P17	V_{SS}
A19	DP1	D14	D17	J16	D38	P18	V_{CC}
B01	A15	D15	D19	J17	V_{SS}	P19	D34
B02	A21	D16	D20	J18	V_{CC}	Q01	TCK
B03	A24	D17	V_{CC}	J19	V_{CC}	Q02	V_{SS}
B04	BE3#	D18	DP3	K01	V_{CC}	Q03	V_{CC}
B05	V_{SS}	D19	D22	K02	V_{SS}	Q04	CACHE#
B06	V_{CC}	E01	A9	K03	V_{SS}	Q05	AHOLD
B07	V_{CC}	E02	V_{SS}	K04	A10	Q15	D61
B08	V_{SS}	E03	V_{CC}	K05	A8	Q16	D54
B09	V_{CC}	E04	A27	K15	D45	Q17	V_{CC}
B10	V_{SS}	E05	BE0#	K16	D43	Q18	V_{SS}
B11	V_{CC}	E15	D21	K17	V_{SS}	Q19	DP4
B12	V_{SS}	E16	D23	K18	V_{SS}	R01	A4
B13	V_{CC}	E17	D25	K19	V_{CC}	R02	V_{SS}
B14	V_{CC}	E18	V_{SS}	L01	V_{CC}	R03	V_{CC}
B15	V_{SS}	E19	V_{CC}	L02	V_{CC}	R04	BOFF#
B16	D9	F01	A7	L03	V_{SS}	R05	D/C#
B17	D11	F02	V_{CC}	L04	SPARE	R06	PCD
B18	D13	F03	V_{SS}	L05	A6	R07	INV
B19	D15	F04	A28	L15	D48	R08	PEN#
C01	A13	F05	A30	L16	D41	R09	BREQ
C02	A19	F15	D24	L17	V_{SS}	R10	TDO
C03	A18	F16	D26	L18	V_{CC}	R11	KB0
C04	A31	F17	V_{SS}	L19	V_{CC}	R12	HOLD
C05	BE4#	F18	V_{CC}	M01	V_{CC}	R13	TMS
C06	V_{SS}	F19	D27	M02	V_{SS}	R14	D63
C07	V_{SS}	G01	V_{CC}	M03	V_{SS}	R15	D60
C08	V_{SS}	G02	V_{CC}	M04	CLK	R16	D57
C09	V_{SS}	G03	V_{SS}	M05	A5	R17	V_{CC}
C10	V_{SS}	G04	A22	M15	D53	R18	D33
C11	V_{SS}	G05	A26	M16	D47	R19	D35
C12	V_{SS}	G15	D28	M17	V_{SS}	S01	A3
C13	V_{SS}	G16	D30	M18	V_{SS}	S02	RESET
C14	V_{SS}	G17	V_{SS}	M19	D31	S03	LOCK#

Table 7.1. Pin Cross Reference by Location (Continued)

Location	Signal	Location	Signal	Location	Signal	Location	Signal
S04	M/IO#	S18	D52	T13	V_{SS}	U08	V_{CC}
S05	EADS#	S19	D37	T14	V_{SS}	U09	V_{CC}
S06	INT/CS8	T01	W/R#	T15	V_{SS}	U10	V_{CC}
S07	BERR	T02	LEN	T16	D56	U11	V_{CC}
S08	FLINE#	T03	PWT	T17	D49	U12	V_{CC}
S09	HLDA	T04	PCYC	T18	D42	U13	V_{CC}
S10	KB1	T05	V_{SS}	T19	D39	U14	V_{CC}
S11	NENE#	T06	V_{SS}	U01	BRDY#	U15	V_{CC}
S12	HIT#	T07	V_{SS}	U02	KEN#	U16	D55
S13	TRST#	T08	V_{SS}	U03	NA#	U17	D51
S14	TDI	T09	V_{SS}	U04	WB/WT#	U18	D44
S15	D62	T10	V_{SS}	U05	V_{CC}	U19	D40
S16	D58	T11	V_{SS}	U06	V_{CC}		
S17	D46	T12	V_{SS}	U07	V_{CC}		

Table 7.2. Pin Cross Reference by Pin Name

Signal	Location	Signal	Location	Signal	Location	Signal	Location
A3	S01	AHOLD	Q05	D13	B18	D43	K16
A4	R01	BE0#	E05	D14	D12	D44	U18
A5	M05	BE1#	D05	D15	B19	D45	K15
A6	L05	BE2#	D06	D16	C18	D46	S17
A7	F01	BE3#	B04	D17	D14	D47	M16
A8	K05	BE4#	C05	D18	C19	D48	L15
A9	E01	BE5#	A04	D19	D15	D49	T17
A10	K04	BE6#	D07	D20	D16	D50	N16
A11	D01	BE7#	A05	D21	E15	D51	U17
A12	J04	BERR	S07	D22	D19	D52	S18
A13	C01	BOFF#	R04	D23	E16	D53	M15
A14	J05	RSRVD	P04	D24	F15	D54	Q16
A15	B01	BRDY#	U01	D25	E17	D55	U16
A16	H04	BREQ	R09	D26	F16	D56	T16
A17	A01	CACHE#	Q04	D27	F19	D57	R16
A18	C03	CLK	M04	D28	G15	D58	S16
A19	C02	CTYP	P05	D29	G19	D59	P15
A20	H05	D0	A07	D30	G16	D60	R15
A21	B02	D1	D09	D31	M19	D61	Q15
A22	G04	D2	A13	D32	H15	D62	S15
A23	A02	D3	A16	D33	R18	D63	R14
A24	B03	D4	A17	D34	P19	D/C#	R05
A25	A03	D5	D10	D35	R19	DP0	A15
A26	G05	D6	A18	D36	N19	DP1	A19
A27	E04	D7	C17	D37	S19	DP2	D13
A28	F04	D8	C16	D38	J16	DP3	D18
A29	D04	D9	B16	D39	T19	DP4	Q19
A30	F05	D10	D11	D40	U19	DP5	J15
A31	C04	D11	B17	D41	L16	DP6	P16
ADS#	N04	D12	C15	D42	T18	DP7	N15

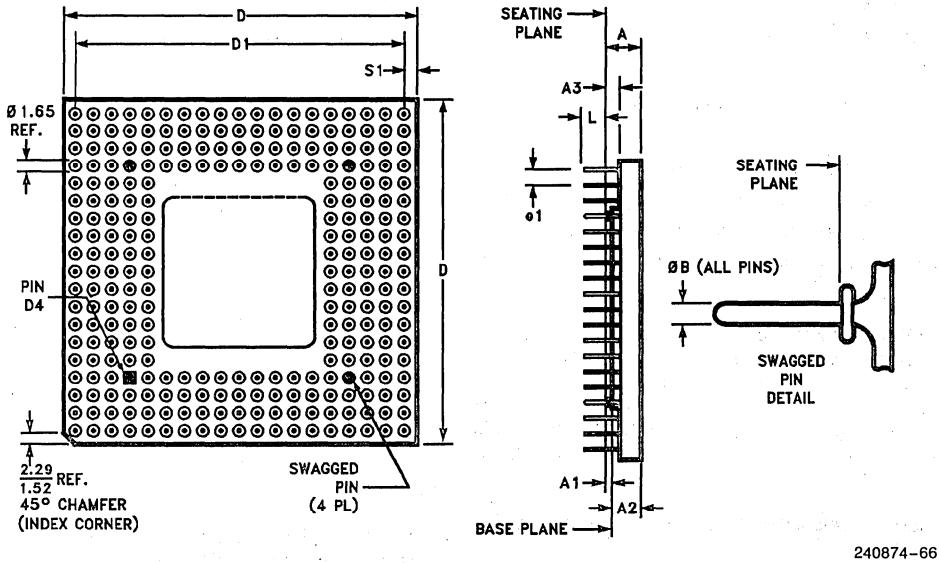
Table 7.2. Pin Cross Reference by Pin Name (Continued)

Signal	Location	Signal	Location	Signal	Location	Signal	Location
EADS#	S05	V_{CC}	B06	V_{CC}	R17	V_{SS}	H17
FLINE#	S08	V_{CC}	B07	V_{CC}	U05	V_{SS}	H18
HIT#	S12	V_{CC}	B09	V_{CC}	U06	V_{SS}	J03
HITM#	N05	V_{CC}	B11	V_{CC}	U07	V_{SS}	J17
HLDA	S09	V_{CC}	B13	V_{CC}	U08	V_{SS}	K02
HOLD	R12	V_{CC}	B14	V_{CC}	U09	V_{SS}	K03
INT/CS8	S06	V_{CC}	D03	V_{CC}	U10	V_{SS}	K17
INV	R07	V_{CC}	D17	V_{CC}	U11	V_{SS}	K18
KB0	R11	V_{CC}	E03	V_{CC}	U12	V_{SS}	L03
KB1	S10	V_{CC}	E19	V_{CC}	U13	V_{SS}	L17
KEN#	U02	V_{CC}	F02	V_{CC}	U14	V_{SS}	M02
LEN	T02	V_{CC}	F18	V_{CC}	U15	V_{SS}	M03
LOCK#	S03	V_{CC}	G01	V_{CC}CLK	P01	V_{SS}	M17
M/IO#	S04	V_{CC}	G02	V_{SS}	B05	V_{SS}	M18
NA#	U03	V_{CC}	G18	V_{SS}	B08	V_{SS}	N03
NENE#	S11	V_{CC}	H01	V_{SS}	B10	V_{SS}	N17
PCD	R06	V_{CC}	H19	V_{SS}	B12	V_{SS}	P03
PCHK#	H16	V_{CC}	J01	V_{SS}	B15	V_{SS}	P17
PCYC	T04	V_{CC}	J02	V_{SS}	C06	V_{SS}	Q02
PEN#	R08	V_{CC}	J18	V_{SS}	C07	V_{SS}	Q18
PWT	T03	V_{CC}	J19	V_{SS}	C08	V_{SS}	R02
RESET	S02	V_{CC}	K01	V_{SS}	C09	V_{SS}	T05
SPARE	L04	V_{CC}	K19	V_{SS}	C10	V_{SS}	T06
EWBE#	D08	V_{CC}	L01	V_{SS}	C11	V_{SS}	T07
BYPASS#	A06	V_{CC}	L02	V_{SS}	C12	V_{SS}	T08
TCK	Q01	V_{CC}	L18	V_{SS}	C13	V_{SS}	T09
TDI	S14	V_{CC}	L19	V_{SS}	C14	V_{SS}	T10
TDO	R10	V_{CC}	M01	V_{SS}	D02	V_{SS}	T11
TMS	R13	V_{CC}	N01	V_{SS}	E02	V_{SS}	T12
TRST#	S13	V_{CC}	N02	V_{SS}	E18	V_{SS}	T13
V_{CC}	A08	V_{CC}	N18	V_{SS}	F03	V_{SS}	T14
V_{CC}	A09	V_{CC}	P02	V_{SS}	F17	V_{SS}	T15
V_{CC}	A10	V_{CC}	P18	V_{SS}	G03	W/R#	T01
V_{CC}	A11	V_{CC}	Q03	V_{SS}	G17	WB/WT#	U04
V_{CC}	A12	V_{CC}	Q17	V_{SS}	H02		
V_{CC}	A14	V_{CC}	R03	V_{SS}	H03		

Table 7.3. Ceramic PGA Package Dimension Symbols

Letter or Symbol	Description of Dimensions
A	Distance from seating plane to highest point of body
A_1	Distance between seating plane and base plane (lid)
A_2	Distance from base plane to highest point of body
A_3	Distance from seating plane to bottom of body
B	Diameter of terminal lead pin
D	Largest overall package dimension of length
D_1	A body length dimension, outer lead center to outer lead center
e_1	Linear spacing between true lead position centerlines
L	Distance from seating plane to end of lead
S_1	Other body dimension, outer lead center to edge of body

NOTES:
1. Controlling dimension: millimeter.
2. Dimension "e_1" ("e") is noncumulative.
3. Seating plane (standoff) is defined by P.C. board hole size: 0.0415–0.0430 inch.
4. Dimensions "B", "B_1", and "C" are nominal.
5. Details of Pin 1 identifier are optional.



240874-66

Family: Ceramic Pin Grid Array Package						
Symbol	Millimeters			Inches		
	Min	Max	Notes	Min	Max	Notes
A	3.56	4.57		.140	.180	
A1	0.64	1.14	Solid Lid	.025	.045	Solid Lid
A2	2.79	3.56	Solid Lid	.110	.140	Solid Lid
A3	1.14	1.40		.045	.055	
B	0.43	0.51		.017	.020	
D	49.28	49.96		1.940	1.967	
D1	45.59	45.85		1.795	1.805	
e1	2.29	2.79		.090	.110	
L	2.54	3.30		.100	.130	
N	240	280		240	280	
S1	1.52	2.54		.060	.100	
ISSUE	9/90					

Figure 7.3. 262-Lead Ceramic PGA Package Dimensions

8.0 PACKAGE THERMAL SPECIFICATIONS

For this section, let:

- P = maximum power consumption
- T_C = case temperature
- T_A = ambient air temperature
- θ_{CA} = thermal resistance from case to ambient air
- θ_{JC} = thermal resistance from junction to case
- θ_{JA} = thermal resistance from junction to ambient air

The i860 XP microprocessor is specified for operation when T_C is within the range of 0°C-85°C. T_C may be measured in any environment to determine whether the i860 XP microprocessor is within specified operating range. The case temperature should be measured at the center of the top surface opposite the pins.

T_A can be calculated from θ_{CA} with the following equation:

$$T_A = T_C - P * \theta_{CA}$$

Typical values for θ_{CA} at various airflows and for θ_{JC} are given in Table 8.1 for the 1.95 sq. in., 262 pin, ceramic PGA. θ_{JC} is shown so that θ_{JA} can be calculated by:

$$\theta_{JA} = \theta_{JC} - \theta_{CA}$$

Note that θ_{JC} with a heatsink differs from θ_{JC} without a heatsink because case temperature is measured differently. Case temperature for θ_{JC} with heatsink is measured at the center of the heat fin base. Case temperature for θ_{JC} without heatsink is measured at the center of the package top surface.

Table 8.2 shows the maximum T_A allowable (without exceeding T_C) at various airflows.

Note that T_A is greatly improved by attaching "fins" or a "heat sink" to the package. P (the maximum power consumption) is calculated by using the maximum I_{CC} at 5V as tabulated in the *D.C. Characteristics* of section 9.

Figure 8.1 gives typical I_{CC} derating with case temperature. For more information on heat sinks, measurement techniques, or package characteristics, refer to *Intel Packaging Handbook*, order number 240800.

Table 8.1. Thermal Resistance—In °C/Watt

	θ_{JC}	θ_{CA} as a Function of Airflow — ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
With Heat Sink*	1.6	10.1	6.3	4.3	3.2	2.5	2.2
Without Heat Sink	1.0	13.5	11.0	8.0	6.5	5.5	5.0

NOTE:
* Nine-fin, unidirectional heat sink (fin dimensions: 0.250″ height, 0.040″ fin width, 0.100″ center-to-center spacing, 1.730″ length)

Table 8.2. Maximum T_A at Various Airflows—In °C

		Airflow — ft/min (m/sec)					
	f_{CLK} (MHz)	0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
T_A with Heat Sink*	50	24	47	59	66	70	72
T_A without Heat Sink	50	4	19	37	46	52	55
T_A with Heat Sink*	40	34.5	53.5	63.5	69	72.5	74
T_A without Heat Sink	40	17.5	30	45	52.5	57.5	60

NOTE:
* Nine-fin, unidirectional heat sink (fin dimensions: 0.250″ height, 0.040″ fin width, 0.100″ center-to-center spacing, 1.730″ length)

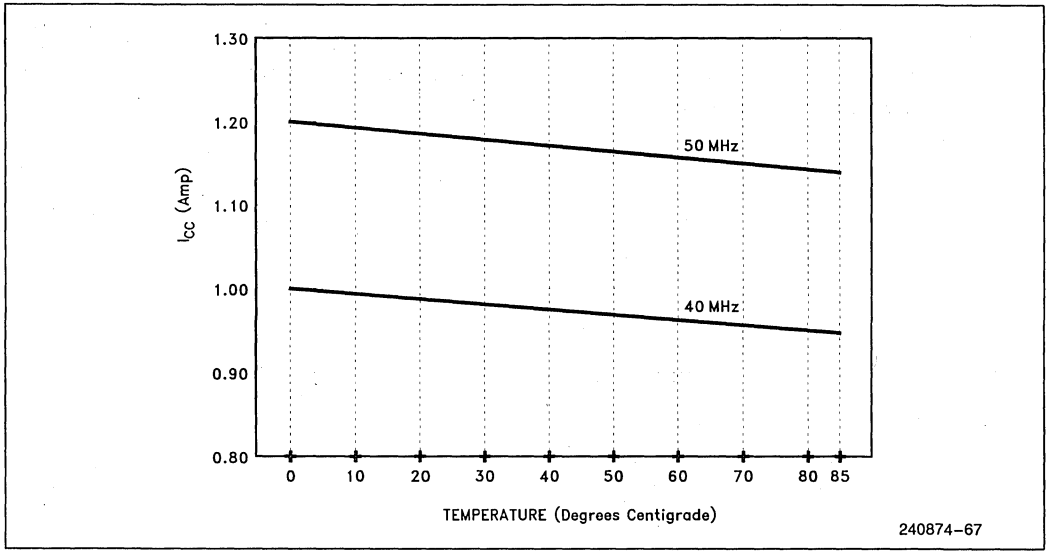


Figure 8.1. I_{CC} Derating with Case Temperature

9.0 ELECTRICAL DATA

All input and output timings are specified relative to the 1.5V level of the rising edge of CLK and refer to the point that the signal reaches 1.5V.

9.1 Absolute Maximum Ratings

Case Temperature T_C under Bias .....0°C to 85°C
Storage Temperature .....−65°C to +150°C
Voltage on Any Pin with
Respect to Ground.....−0.5 to V_{CC}+0.5V

NOTICE: This data sheet contains preliminary information on new products in production. The specifications are subject to change without notice. Verify with your local Intel Sales office that you have the latest data sheet before finalizing a design.

**WARNING: Stressing the device beyond the "Absolute Maximum Ratings" may cause permanent damage. These are stress ratings only. Operation beyond the "Operating Conditions" is not recommended and extended exposure beyond the "Operating Conditions" may affect device reliability.*

9.2 D.C. Characteristics

Table 9.1. D.C. Characteristics Operating Conditions: V_{CC} = 5V ±5%; T_C = 0°C to 85°C

Symbol	Parameter	Min	Max	Units	Notes
V_{IL}	Input LOW voltage (TTL)	−0.3	+0.8	V	
V_{IH}	Input HIGH voltage (TTL)	2.0	V_{CC}+0.3	V	
V_{IHC}	CLK Input HIGH (TTL)	2.5	V_{CC} + 0.3	V	
V_{OL}	Output LOW voltage (TTL)		0.45	V	1
V_{OH}	Output HIGH voltage (TTL)	2.4		V	2
I_{CC}	Power supply current (@ 50 MHz)		1.2	Amp	3
I_{CC}	Power supply current (@40 MHz)		1.0	Amp	3
I_{LI}	Input leakage current		±15	μA	4
I_{LIP}	Input leakage current (pull-up)		−400	μA	5
I_{LO}	Output leakage current		±15	μA	6
C_{IN}	Input capacitance		11.5	pF	7
C_O	I/O or output capacitance		14	pF	7

NOTES:
1. This parameter is measured with current load of 5 mA.
2. This parameter is measured with current load of 1 mA. Typical value is V_{CC} − 0.45V.
3. Measured at 50 MHz and V_{CC} = 5V.
4. This parameter is for inputs without pullups. V_{CC} is on, and 0V ≤ V_{IN} ≤ V_{CC}.
5. This parameter is for inputs with pullups and V_{IL} = 0.45V. Note that if the pull-ups are put in high-impedance state via the DCTL boundary scan cell that also tri-states the data outputs, then the leakage is ±15 μA.
6. 0.45V ≤ V_{IN} ≤ V_{CC} − 0.45V.
7. These parameters are not tested; they are guaranteed by design characterization.

9.3 A.C. Characteristics

Table 9.2. A.C. Characteristics

C_L = 0 pF Unless Otherwise Specified; V_{CC} = 5V ±5%; T_C = 0°C to 85°C

Symbol	Parameter	Fig	40 MHz		50 MHz		Notes
			Min (ns)	Max (ns)	Min (ns)	Max (ns)	
tc	CLK Period	9.1	25	40	20	40	
ttc	TCK Period	9.2	40	1000	40	1000	
	CLK Stability	9.1		0.1%		0.1%	
tch	CLK High Time	9.1	7		7		
tcl	CLK Low Time	9.1	7		7		
tr	CLK Rise Time	9.1		3		3	h
tf	CLK Fall Time	9.1		3		3	h
ts	TCK to CLK Skew	9.3		±1		±1	i
ttch	TCK High Time	9.2	10		10		
ttcl	TCK Low Time	9.2	10		10		
ttcr	TCK Rise Time	9.2		4		4	
ttcf	TCK Fall Time	9.2		4		4	
tsu.1	RESET, HOLD, BERR, FLINE#, PEN#, INT/CS8 Setup Time	9.1	8		7		
tsu.2	BOFF#, AHOLD, KEN#, NA#, INV, WB/WT# Setup Time	9.1	8		7		
tsu.3	EADS# Setup Time	9.1	9		8		
tsu.4	EWBE# Setup Time	9.1	8.5		7.5		
tsu.5	BRDY# Setup Time	9.1	8.5		7.5		
tsu.6	D63–D0, DP7–DP0 Setup Time	9.1	8.5		7.5		
tsu.7	D63–D0, DP7–DP0 Setup Time (Late Backoff Mode)	9.1	5.5		4.5		
tsu.8	A31–A5 Setup Time	9.1	11		10		
ttsu	TDI, TMS, TRST# Setup Time	9.2	8		8		
tth	TDI, TMS, TRST# Hold Time	9.2	2		1		b
th.1	Hold Time, All Inputs except D63–D0, DP7–D0	9.1	2		1		c
th.2	D63–D0, DP7–DP0 Hold Time (Normal and Late Back-Off Mode)	9.1	3		2		c
ttco	TDO Valid Delay and All Outputs Valid Delay in EXTEST Mode	9.2	1.5	17.5	1.5	16.5	a, f
tco.1	A31–A22 Valid Delay	9.1	1.5	12	1.5	11	a
tco.2a	A21–A3 Valid Delay (High Current Mode)	9.1	1.5	11.5	1.5	10.5	a, g
tco.2b	A21–A3 Valid Delay (Normal Current Mode)	9.1	1.5	12	1.5	11	a

Table 9.2. A.C. Characteristics (Continued)
C_L = 0 pF Unless Otherwise Specified; V_{CC} = 5V ±5%; T_C = 0°C to 85°C

Symbol	Parameter	Fig	40 MHz		50 MHz		Notes
			Min (ns)	Max (ns)	Min (ns)	Max (ns)	
tco.3	D63–D0, DP7–DP0 Valid Delay	9.1	2.5	14	2.5	13	a, d
tco.4	BREQ, HLDA, PCHK#, NENE#, KB0, KB1 Valid Delay	9.1	1.5	13	1.5	12	a
tco.5a	ADS# Valid Delay (High Current Mode)	9.1	1.5	10	1.5	9	a, g
tco.5b	ADS# Valid Delay (Normal Current Mode)	9.1	1.5	11	1.5	10	a
tco.6a	W/R# Valid Delay (High Current Mode)	9.1	1.5	11	1.5	10	a, g
tco.6b	W/R# Valid Delay (Normal Current Mode)	9.1	1.5	12	1.5	11	a
tco.7a	HITM# Valid Delay (High Current Mode)	9.1	1.5	12	1.5	11	a, g
tco.7b	HITM# Valid Delay (Normal Current Mode)	9.1	1.5	13	1.5	12	a
tco.8	PWT, PCD, HIT#, CTYP, D/C# M/IO#, PCYC, LOCK#, CACHE#, LEN Valid Delay	9.1	1.5	12	1.5	11	a
tco.9a	BE0#–BE7# Valid Delay (High Current Mode)	9.1	1.5	12	1.5	11	a, g
tco.9b	BE0#–BE7# Valid Delay (Normal Current Mode)	9.1	1.5	13	1.5	12	a
tz.1	Float Time All Outputs except D63–D0, DP7–DP0	9.1	2	19	2	18	e
tz.2	Float Time D63–D0, DP7–DP0	9.1	3	19	3	18	e
tzt	Float Time during Boundary Scan EXTEST	9.1		20		20	f

NOTES:
a. Minimum and maximum delays are for 0pF load.
b. These hold times are referenced to the falling edge of TCK.
c. These hold times are referenced to the rising edge of CLK.
d. Output delay for D63–D0, DP7–DP0 is from the CLK after ADS# activation.
e. Float time = delay until maximum output current is less than $\pm I_{LO}$. Float time is not tested.
f. Delay from falling edge of TCK.
g. These pins can be configured as normal or high-current buffers. When they are configured as high-current buffers for interface with cache memory or other large loads, use the derating curves in Figure 9.3. Otherwise, all normal buffers use the derating curves in Figure 9.4.
h. tr and tf should be measured between 0.8V and 2.5V.
i. Assumes TCK and CLK both at 25 MHz.

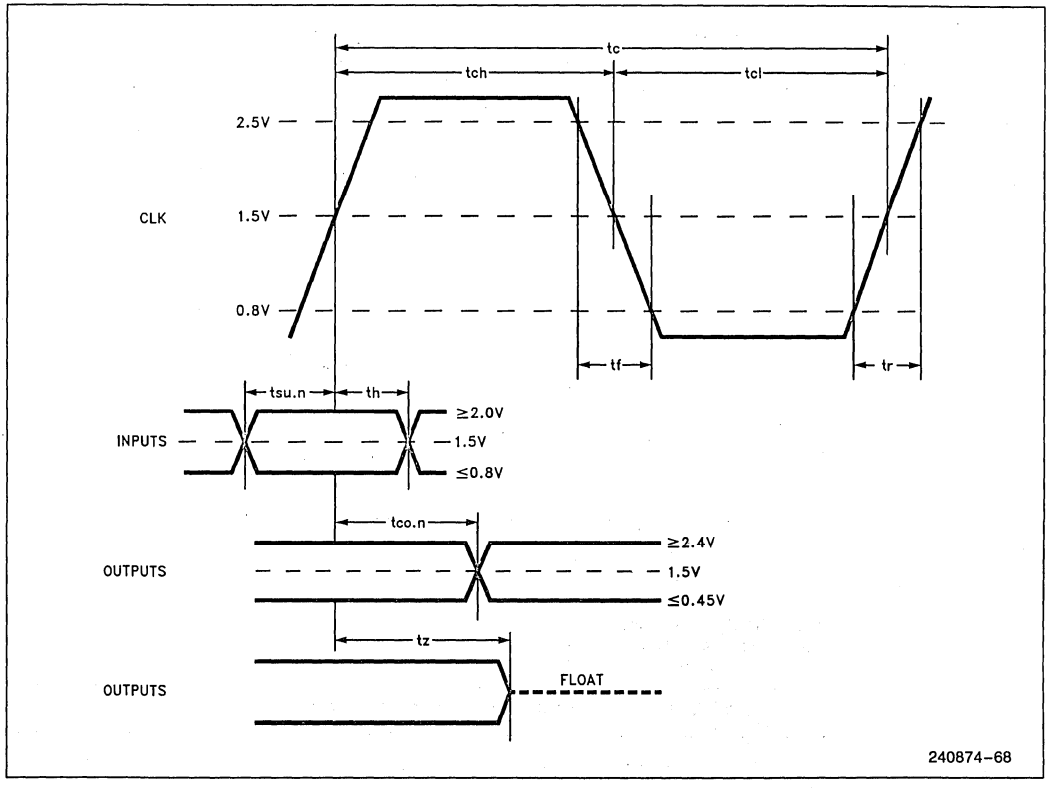


Figure 9.1. CLK, Input, and Output Timings

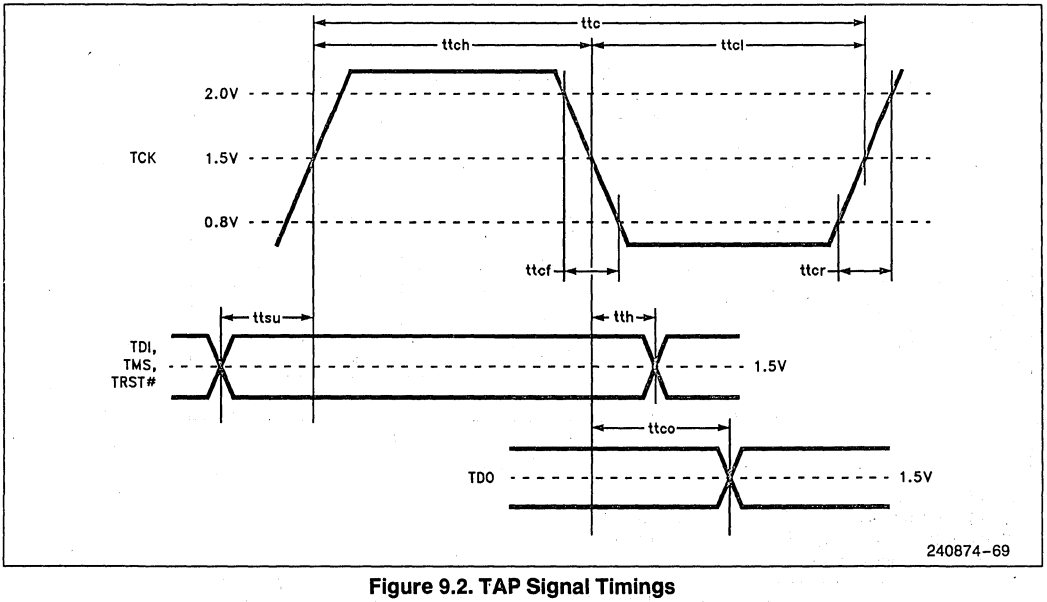
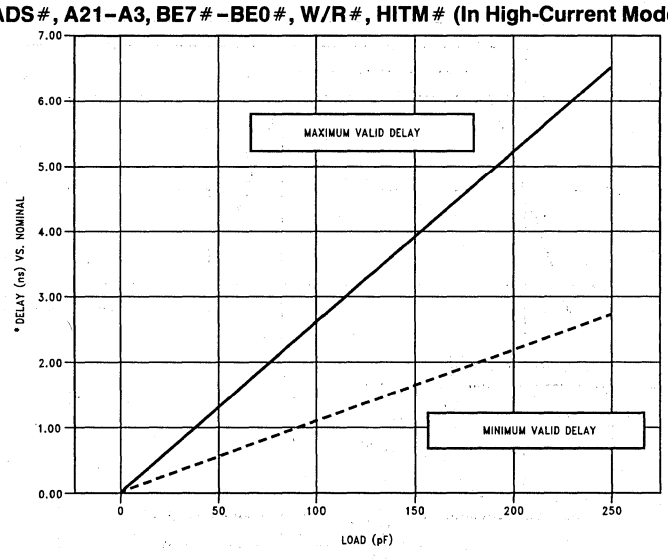
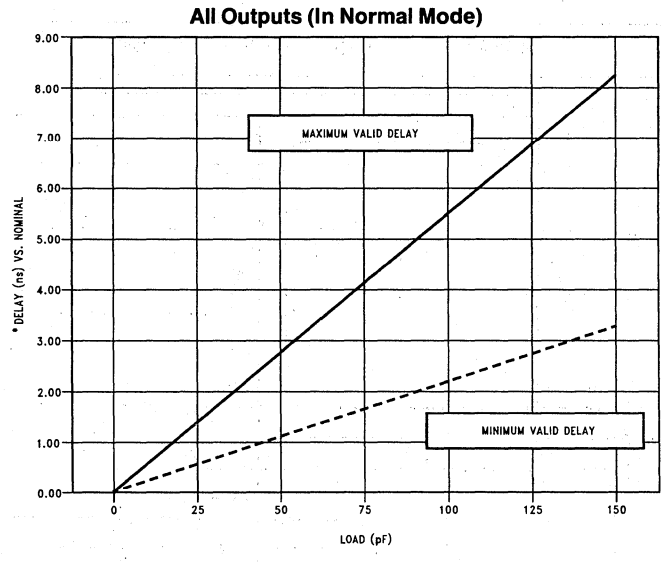


Figure 9.2. TAP Signal Timings



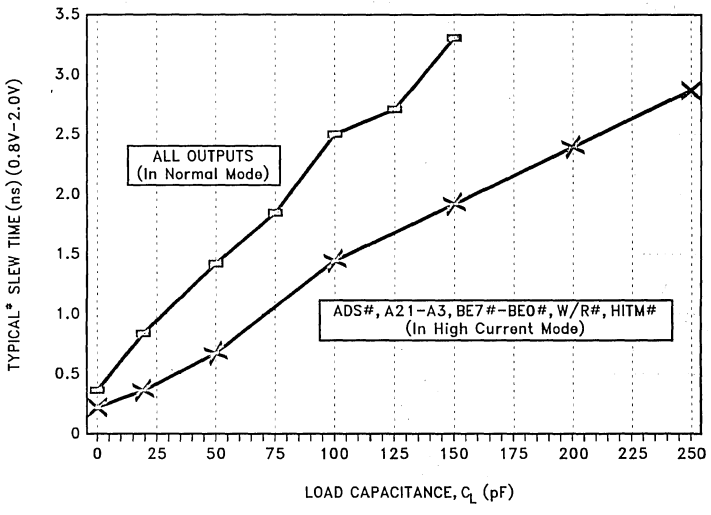
NOTES:
Graphs are not linear outside the C_L range shown.
NOMINAL = 0pF value given in the A.C. Timings table.
*Typical part under worst-case conditions.

Figure 9.3. Typical Output Delay vs Load Capacitance



NOTES:
Graphs are not linear outside the C_L range shown.
NOMINAL = 0 pF value given in the A.C. Timings table.
*Typical part under worst-case conditions.

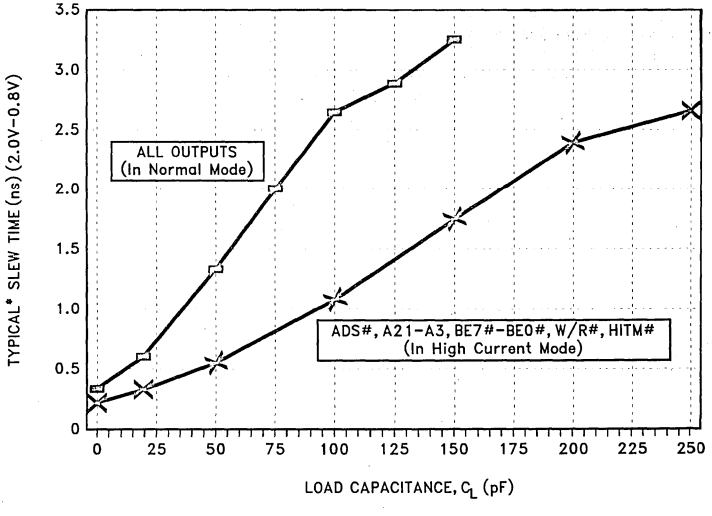
Figure 9.4. Typical Output Delay vs Load Capacitance



240874-81

NOTES:
Graphs are not linear outside the C_L range shown.
*Typical part under worst-case conditions.

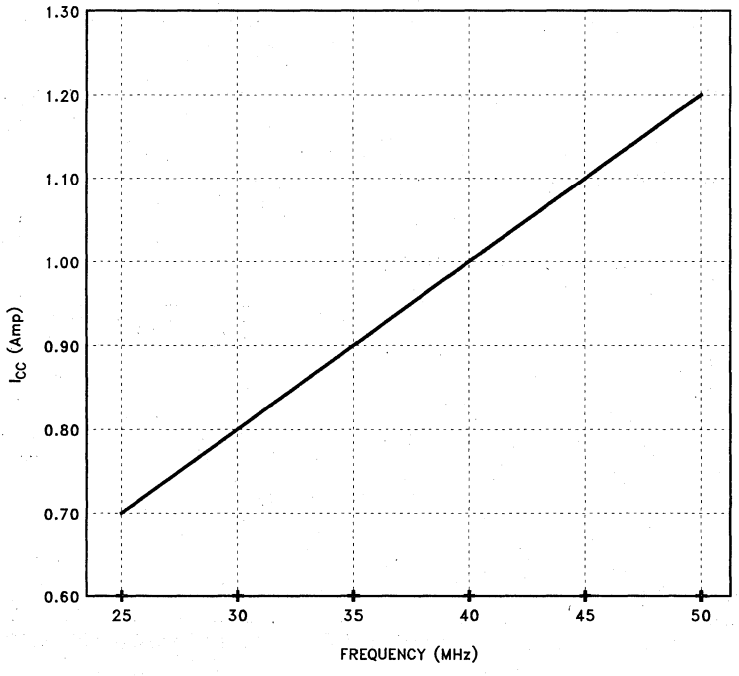
Figure 9.5a. Typical Slew Time vs Load Capacitance under Worst-Case Conditions (Rising Voltage)



240874-82

NOTES:
Graphs are not linear outside the C_L range shown.
*Typical part under worst-case conditions.

Figure 9.5b. Typical Slew Time vs Load Capacitance under Worst-Case Conditions (Falling Voltage)



NOTES:
Graph is not linear outside the frequency range shown.
*Worst-case supply current at 5V.

Figure 9.6. Typical I_{CC} vs. Frequency

9.4 Component Buffer Model

9.4.1 FIRST ORDER ELECTRICAL BUFFER MODEL

The first order electrical buffer model provides an accurate and simple representation of the buffers used in the inputs and outputs of the CHMOS i860 XP CPU. The model output consists of four components:

1. Linear voltage waveform (dV/dt)
2. Intrinsic buffer delay due to C_L (t_o)
3. Buffer output impedance (R_O)
4. Buffer output capacitance (C_O)

as shown in Figure 9.7a

A fitting algorithm has been used to arrive at values for dV/dt, t_o, C_O, and R_O such that R_O matches the actual buffer impedance and C_O, the intrinsic buffer output capacitance whether the output is on or off, remains constant across the operating range while minimizing the difference between the full buffer circuit and its simplified electrical model for a set of different loads (lumped capacitance, and short and long transmission lines). dV/dT is the slope of the voltage ramp, while t_o is the intrinsic buffer delay associated with a given C_L. t_o accounts for the intrinsic delay by offsetting the excitation of the model by the amount of the delay.

NOTE:
t_o is zero for C_L = 0 and when the load is represented by a transmission line.

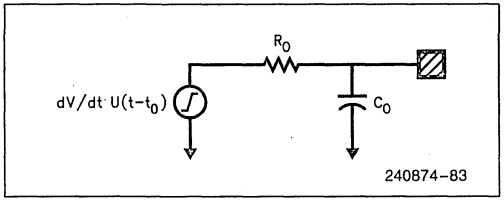


Figure 9.7a. Output Model

The input model consists of one component, buffer capacitance (C_{IN}), as shown in Figure 9.7b.

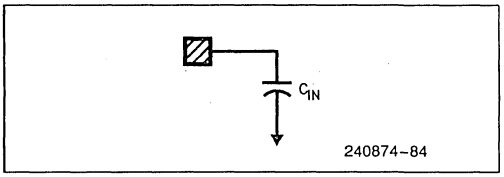


Figure 9.7b. Input Model

9.4.2 FIRST ORDER ELECTRICAL MODEL PARAMETER VALUES

The parameters that make up the first order electrical model vary with the buffer design. In addition, these parameters also vary with the operating condition (i.e., temperature and V_{CC}) of the buffer process. The typical process corner is being modeled. Two sizes of buffer are used on these components, labelled here as small and large. The parameter values found in Table 9.3 and 9.4 list dV/dt, t_o, R_O, and C_O. These parameters are provided for both low-to-high and high-to-low transitions at the typical process corner for three operating conditions (V_{CC} = 5.5V and $T_J = -10°C$, V_{CC} = 5.0V and T_J = 80°C, and V_{CC} = 4.5V and T_J = 125°C.

9.4.3 PACKAGE PARAMETERS

In addition to the buffer characteristics, package characteristics are also included to complete the model. Package inductance, capacitance and resistance values vary with design geometry and material properties of the package. Figure 9.8 shows a model of the package including these parameters and should be placed between the first order electrical buffer model as shown in Figure 9.9 and the board interconnects. Notice the package model only includes the package inductance (L_P) and capacitance (C_P). This is sufficient since the package resistance is so small it is negligible.

Table 9.5 lists the buffer model parameters for each pin of the i860 XP microprocessor. The table gives the package model parameters for each pin, followed by the input capacitance (input and I/O pins) and/or output buffer size (outputs and I/O). In those cases where the buffer used by a pin is an option selected at reset by the PEN# input, the output buffer column lists the sizes available. Large buffers correspond to high-current mode, while small buffers correspond to normal current mode.

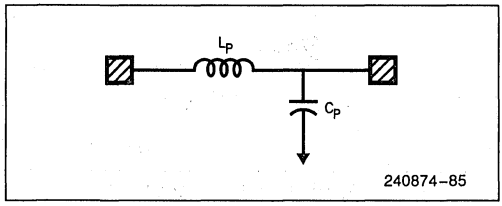


Figure 9.8. Package Model

9.4.4 BOARD INTERCONNECTS

The board interconnect can be considered as a lumped parameter (capacitive load) or as a transmission line. As a rule of thumb, an unterminated board interconnect may be considered as a capacitive load if the round trip time (time for signal to travel from one end of the interconnect to the other and back) is short compared to the transition time of the signal. At frequencies of 50 MHz and above most interconnects behave as transmission lines (Figure 9.10). For accurate results at high frequencies, these transmission line effects must be taken into account and modeled.

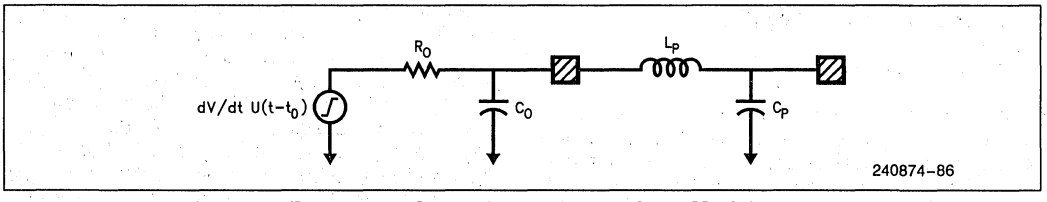


Figure 9.9a. Output Buffer and Package Model

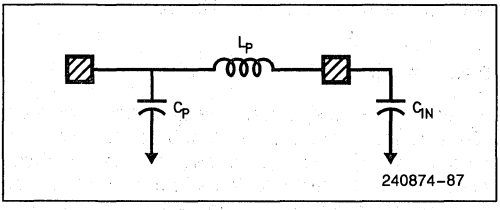


Figure 9.9b. Input Buffer and Package Model

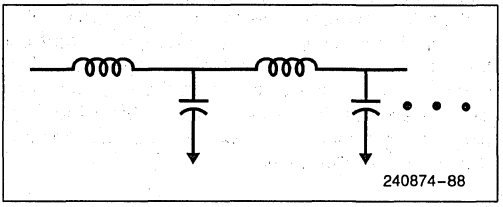


Figure 9.10. Transmission Line Model

Table 9.3. Small Output Buffer First Order Electrical Model Parameter Values

Transition	V_{CC}	T_J (C)	R_O (ohms)	C_O (pF)	dV/dT	t_o (ns) at various C_L					
						0 (pF)	5 (pF)	25 (pF)	50 (pF)	100 (pF)	150 (pF)
Low-to-High	5.5	−10	28.0	4.3	5.5/1.2	0	0.0	0.1	0.3	0.7	1.1
Low-to-High	5.5	80	36.4	4.3	5.5/1.4	0	0.0	0.1	0.8	0.8	1.2
Low-to-High	5.5	125	40.4	4.3	5.5/1.5	0	0.0	0.1	0.4	0.8	1.2
Low-to-High	5.0	−10	30.2	4.3	5.0/1.2	0	0.0	0.1	0.4	0.8	1.2
Low-to-High	5.0	80	39.2	4.3	5.0/1.4	0	0.0	0.2	0.4	0.9	1.3
Low-to-High	5.0	125	43.5	4.3	5.0/1.6	0	0.0	0.2	0.4	0.9	1.3
Low-to-High	4.5	−10	33.0	4.3	4.5/1.2	0	0.0	0.2	0.5	1.0	1.4
Low-to-High	4.5	80	42.8	4.3	4.5/1.6	0	0.0	0.2	0.5	1.0	1.5
Low-to-High	4.5	125	47.4	4.3	4.5/1.6	0	0.0	0.3	0.6	1.1	1.6
High-to-Low	5.5	−10	23.2	4.3	5.5/1.0	0	0.0	0.4	0.7	1.2	1.6
High-to-Low	5.5	80	31.4	4.3	5.5/1.4	0	0.0	0.4	0.9	1.3	1.8
High-to-Low	5.5	125	36.1	4.3	5.5/1.6	0	0.0	0.5	0.8	1.3	1.8
High-to-Low	5.0	−10	24.0	4.3	5.0/1.1	0	0.0	0.5	0.9	1.2	1.7
High-to-Low	5.0	80	32.8	4.3	5.0/1.4	0	0.0	0.5	0.9	1.5	1.9
High-to-Low	5.0	125	37.8	4.3	5.0/1.7	0	0.0	0.5	0.9	1.4	1.8
High-to-Low	4.5	−10	25.1	4.3	4.5/1.2	0	0.0	0.4	0.7	1.2	1.7
High-to-Low	4.5	80	34.5	4.3	4.5/1.6	0	0.0	0.4	0.8	1.3	1.8
High-to-Low	4.5	125	39.9	4.3	4.5/1.8	0	0.0	0.5	0.9	1.4	1.9

Table 9.4. Large Output Buffer First Order Electrical Model Parameter Values

Transition	V_{CC}	T_J (C)	R_O (ohms)	C_O (pF)	dV/dT	t_o (ns) at various C_L								
						0 (pF)	5 (pF)	25 (pF)	50 (pF)	100 (pF)	150 (pF)	200 (pF)	250 (pF)	300 (pF)
Low-to-High	5.5	−10	12.1	4.3	5.5/0.7	0	0.0	0.1	0.3	0.6	0.8	1.0	1.3	1.5
Low-to-High	5.5	80	15.5	4.3	5.5/0.9	0	0.0	0.2	0.3	0.6	0.9	1.1	1.4	1.7
Low-to-High	5.5	125	17.2	4.3	5.5/1.1	0	0.0	0.2	0.4	0.7	1.0	1.2	1.4	1.7
Low-to-High	5.0	−10	13.0	4.3	5.0/0.9	0	0.0	0.1	0.3	0.6	0.9	1.1	1.4	1.7
Low-to-High	5.0	80	16.7	4.3	5.0/1.0	0	0.0	0.2	0.4	0.8	1.1	1.4	1.7	2.0
Low-to-High	5.0	125	18.5	4.3	5.0/1.2	0	0.0	0.2	0.4	0.8	1.1	1.4	1.7	2.0
Low-to-High	4.5	−10	14.1	4.3	4.5/0.9	0	0.0	0.2	0.4	0.7	1.1	1.4	1.7	2.0
Low-to-High	4.5	80	18.0	4.3	4.5/1.2	0	0.0	0.2	0.4	0.9	1.2	1.5	1.9	2.2
Low-to-High	4.5	125	19.9	4.3	4.5/1.3	0	0.0	0.2	0.5	0.8	1.2	1.5	1.9	2.2
High-to-Low	5.5	−10	10.6	4.3	5.5/0.7	0	0.0	0.3	0.6	0.9	1.2	1.5	1.8	2.0
High-to-Low	5.5	80	13.9	4.3	5.5/1.0	0	0.0	0.4	0.7	1.2	1.5	1.9	2.2	2.5
High-to-Low	5.5	125	15.8	4.3	5.5/1.1	0	0.0	0.4	0.8	1.3	1.7	2.0	2.4	2.8
High-to-Low	5.0	−10	11.0	4.3	5.0/0.8	0	0.0	0.4	0.7	1.0	1.3	1.6	1.9	2.1
High-to-Low	5.0	80	14.5	4.3	5.0/1.0	0	0.0	0.4	0.8	1.2	1.6	2.0	2.3	2.6
High-to-Low	5.0	125	16.5	4.3	5.0/1.2	0	0.0	0.4	0.8	1.3	1.7	2.1	2.5	2.8
High-to-Low	4.5	−10	11.3	4.3	4.5/0.9	0	0.0	0.4	0.7	1.1	1.4	1.7	2.0	2.4
High-to-Low	4.5	80	15.2	4.3	4.5/1.2	0	0.0	0.4	0.8	1.3	1.6	2.0	2.3	2.7
High-to-Low	4.5	125	17.4	4.3	4.5/1.3	0	0.0	0.4	0.8	1.3	1.7	2.1	2.5	2.8

Table 9.5 Buffer Models

Pin Name	Location	Cp (pF) Typical	Lp (nH) Typical	Input Buffer C_{IN} (pF) Typical	Output Buffer Size (Large or Small)
A_3	S01	7.6	13.8	6.7	L/S
A_4	R01	6.2	14.5	6.7	L/S
A_5	M05	6.5	7.8	6.7	L/S
A_6	L05	5.3	8.0	6.7	L/S
A_7	F01	7.7	16.2	6.7	L/S
A_8	K05	5.1	7.7	6.7	L/S
A_9	E01	8.0	16.4	6.7	L/S
A_{10}	K04	5.1	8.8	6.7	L/S
A_{11}	D01	8.3	16.8	6.7	L/S
A_{12}	J04	5.2	9.0	6.7	L/S
A_{13}	C01	8.7	17.2	6.7	L/S
A_{14}	J05	5.2	7.8	6.7	L/S
A_{15}	B01	9.0	17.8	6.7	L/S
A_{16}	H04	5.2	9.0	6.7	L/S
A_{17}	A01	9.4	18.2	6.7	L/S
A_{18}	C03	7.8	14.5	6.7	L/S
A_{19}	C02	9.0	15.3	6.7	L/S
A_{20}	H05	7.5	7.7	6.7	L/S
A_{21}	B02	8.5	15.7	6.7	L/S
A_{22}	G04	7.5	9.1	4.4	S
A_{23}	A02	8.1	15.7	4.4	S
A_{24}	B03	7.0	14.5	4.4	S
A_{25}	A03	7.7	14.6	4.4	S
A_{26}	G05	6.7	7.9	4.4	S
A_{27}	E04	7.6	9.6	4.4	S
A_{28}	F04	6.5	9.2	4.4	S
A_{29}	D04	7.4	10.0	4.4	S
A_{30}	F05	5.9	8.2	4.4	S
A_{31}	C04	6.6	10.4	4.4	S
ADS#	N04	6.2	9.1		L/S
AHOLD	Q05	6.0	8.8	2.0	
BE0#	E05	5.7	8.8		L/S
BE1#	D05	6.7	8.8		L/S
BE2#	D06	5.7	9.0		L/S

Table 9.5. Buffer Models (Continued)

Pin Name	Location	Cp (pF) Typical	Lp (nH) Typical	Input Buffer C_{IN} (pF) Typical	Output Buffer Size (Large or Small)
BE3#	B04	6.5	11.2		L/S
BE4#	C05	5.9	10.6		L/S
BE5#	A04	6.5	12.0		L/S
BE6#	D07	4.9	8.6		L/S
BE7#	A05	6.1	11.5		L/S
BERR	S07	5.8	8.7	2.0	
BOFF#	R04	6.3	10.4	2.0	
RSRVD	P04	6.4	9.4	2.0	
BRDY#	U01	8.0	14.7	2.0	
BREQ	R09	4.4	7.5		S
BYPASS#	A06	Strapping Option			
CACHE#	Q04	6.6	9.8		S
CLK	M04	6.2	8.9	2.0	
CTYP	P05	6.5	8.6		S
D_0	A07	5.5	10.6	4.4	S
D_1	D09	7.6	7.6	4.4	S
D_2	A13	7.4	15.0	4.4	S
D_3	A16	7.7	17.7	4.4	S
D_4	A17	9.2	17.9	4.4	S
D_5	D10	7.5	7.6	4.4	S
D_6	A18	9.4	18.3	4.4	S
D_7	C17	8.6	15.9	4.4	S
D_8	C16	8.6	14.5	4.4	S
D_9	B16	9.3	14.7	4.4	S
D_{10}	D11	8.3	7.5	4.4	S
D_{11}	B17	8.9	14.7	4.4	S
D_{12}	C15	8.1	7.8	4.4	S
D_{13}	B18	8.6	15.4	4.4	S
D_{14}	D12	7.2	7.8	4.4	S
D_{15}	B19	8.2	15.6	4.4	S
D_{16}	C18	7.9	10.7	4.4	S
D_{17}	D14	6.7	9.2	4.4	S
D_{18}	C19	7.6	14.2	4.4	S
D_{19}	D15	6.4	10.0	4.4	S

Table 9.5 Buffer Models (Continued)

Pin Name	Location	Cp (pF) Typical	Lp (nH) Typical	Input Buffer C_{IN} (pF) Typical	Output Buffer Size (Large or Small)
D_{20}	D16	7.4	10.7	4.4	S
D_{21}	E15	5.6	8.8	4.4	S
D_{22}	D19	6.7	12.7	4.4	S
D_{23}	E16	5.5	9.7	4.4	S
D_{24}	F15	5.3	8.3	4.4	S
D_{25}	E17	6.6	9.9	4.4	S
D_{26}	F16	5.3	9.7	4.4	S
D_{27}	F19	6.2	11.7	4.4	S
D_{28}	G15	5.1	7.9	4.4	S
D_{29}	G19	6.2	11.8	4.4	S
D_{30}	G16	5.1	8.9	4.4	S
D_{31}	M19	8.6	16.2	4.4	S
D_{32}	H15	5.2	7.7	4.4	S
D_{33}	R18	11.0	19.6	4.4	S
D_{34}	P19	8.0	18.4	4.4	S
D_{35}	R19	9.1	18.8	4.4	S
D_{36}	N19	8.1	16.9	4.4	S
D_{37}	S19	9.2	20.7	4.4	S
D_{38}	J16	8.4	8.9	4.4	S
D_{39}	T19	10.5	19.6	4.4	S
D_{40}	U19	10.8	19.1	4.4	S
D_{41}	L16	8.3	10.9	4.4	S
D_{42}	T18	10.5	17.8	4.4	S
D_{43}	K16	8.4	8.8	4.4	S
D_{44}	U18	10.1	17.7	4.4	S
D_{45}	K15	9.3	7.5	4.4	S
D_{46}	S17	9.5	14.5	4.4	S
D_{47}	M16	8.0	9.8	4.4	S
D_{48}	L15	8.0	7.7	4.4	S
D_{49}	T17	8.7	14.6	4.4	S
D_{50}	N16	7.8	9.9	4.4	S
D_{51}	U17	8.6	15.2	4.4	S
D_{52}	S18	7.6	14.3	4.4	S

Table 9.5 Buffer Models (Continued)

Pin Name	Location	Cp (pF) Typical	Lp (nH) Typical	Input Buffer C_{IN} (pF) Typical	Output Buffer Size (Large or Small)
D_{53}	M15	7.7	7.1	4.4	S
D_{54}	Q16	7.0	11.1	4.4	S
D_{55}	U16	8.0	14.3	4.4	S
D_{56}	T16	7.8	12.8	4.4	S
D_{57}	R16	6.5	11.8	4.4	S
D_{58}	S16	7.5	11.3	4.4	S
D_{59}	P15	6.2	8.7	4.4	S
D_{60}	R15	7.1	9.6	4.4	S
D_{61}	Q15	5.9	9.3	4.4	S
D_{62}	S15	6.9	10.7	4.4	S
D_{63}	R14	5.6	9.7	4.4	S
D/C#	R05	5.8	9.7		S
DP0	A15	7.7	18.3	4.4	S
DP1	A19	9.7	18.9	4.4	S
DP2	D13	7.1	8.5	4.4	S
DP3	D18	6.7	11.3	4.4	S
DP4	Q19	10.4	19.0	4.4	S
DP5	J15	9.9	7.7	4.4	S
DP6	P16	9.3	10.7	4.4	S
DP7	N15	6.8	8.9	4.4	S
EADS#	S05	5.5	10.5	2.0	
EWBE#	D08	7.5	7.6	2.0	
FLINE#	S08	5.4	8.1	2.0	
HIT#	S12	5.9	11.1		S
HITM#	N05	6.2	8.2		L
HLDA	S09	5.3	7.9		S
HOLD	R12	6.1	11.1	2.0	
INT/CS8	S06	5.2	10.0	2.0	
INV	R07	5.3	8.2	2.0	
KB0	R11	6.1	9.2		S
KB1	S10	6.4	7.9		S
KEN#	U02	7.4	13.4	2.0	
LEN	T02	7.9	12.8		S

Table 9.5 Buffer Models (Continued)

Pin Name	Location	Cp (pF) Typical	Lp (nH) Typical	Input Buffer C_{IN} (pF) Typical	Output Buffer Size (Large or Small)
LOCK#	S03	7.7	11.2		S
M/IO#	S04	7.3	10.3		S
NA#	U03	7.1	13.0	2.0	
NENE#	S11	6.3	9.6		S
PCD	R06	5.6	8.9		S
PCHK#	H16	5.1	8.8		S
PCYC	T04	7.2	11.4		S
PEN#	R08	4.8	7.8	2.0	
PWT	T03	7.4	12.1		S
RESET	S02	7.9	12.5	2.0	
SPARE	L04				NC
TCK	Q01	5.8	14.1	2.0	
TDI	S14	6.5	9.8	2.0	
TDO	R10	6.3	7.6		S
TMS	R13	5.6	9.6	2.0	
TRST#	S13	6.3	9.6	2.0	
W/R#	T01	7.8	14.3		L/S
WB/WT#	U04	6.7	12.3	2.0	

10.0 INSTRUCTION SET

Key to abbreviations:

For register operands, the abbreviations that describe the operands are composed of two parts. The first part describes the type of register:

c One of the control registers **fir**, **psr**, **epsr**, **dirbase**, **db**, **fsr**, **bear**, **ccr**, **p0**, **p1**, **p2**, or **p3**

f One of the floating-point registers: **f0** through **f31**

i One of the integer registers: **r0** through **r31**

The second part identifies the field of the machine instruction into which the operand is to be placed:

src1 The first of the two source-register designators, which may be either a register or a 16-bit immediate constant or address offset. The immediate value is zero-extended for logical operations and is sign-extended for add and subtract operations (including **addu** and **subu**) and for all addressing calculations.

src1ni Same as *src1* except that no immediate constant or address offset value is permitted.

src1s Same as *src1* except that the immediate constant is a 5-bit value that is zero-extended to 32 bits.

src2 The second of the two source-register designators.

dest The destination register designator.

Thus, the operand specifier *isrc2*, for example, means that an integer register is used and that the encoding of that register must be placed in the *src2* field of the machine instruction.

Other (nonregister) operands are specified by a one-part abbreviation that represents both the type of operand required and the instruction field into which the value of the operand is placed:

#const A 16-bit immediate constant or address offset that the i860 XP microprocessor sign-extends to 32 bits when computing the effective address.

lbroff A signed, 26-bit, immediate, relative branch offset.

sbroff A signed, 16-bit, immediate, relative branch offset.

brx A function that computes the target address by shifting the offset (either *lbroff* or *sbroff*) left by two bits, sign-extending it to 32 bits, and adding the result to the current instruction pointer plus four. The resulting target address may lie anywhere within the address space.

Table 10.1. Precision Specification

Suffix	Source Precision	Result Precision
.ss	single	single
.sd	single	double
.dd	double	double
.ds	double	single

Unless otherwise specificed, floating-point operations accept single- or double-precision source operands and produce a result of equal or greater precision. Both input operands must have the same precision. The source and result precision are specified by a two-letter suffix to the mnemonic of the operation.

Other abbreviations include:

.p Precision specification **.ss**, **.sd**, or **.dd** (**.ds** not permitted). Refer to Table 10.1.

.r Precision specification **.ss**, **.sd**, **.ds**, or **.dd**. Refer to Table 10.1.

.v **.sd** or **.dd** Refer to Table 10.1.

.w **.ss** or **.dd**. Refer to Table 10.1.

.x **.b** (8 bits), **.s** (16 bits), or **.l** (32 bits)

.y **.l** (32 bits), **.d** (64 bits), or **.q** (128 bits)

mem.x(address) The memory location indicated by *address* with a size of *x*.

port.x(address) The I/O port indicated by *address* with a size of *x*.

int_vector.x(address) The interrupt vector with a size of *x* returned from I/O port *address*.

PM The pixel mask, which is considered as an array of eight bits PM(7)..PM(0), where PM(0) is the least-significant bit.

10.1 Instruction Definitions in Alphabetical Order

adds *isrc1, isrc2, idest* ..... **Add Signed**
idest ← *isrc1* + *isrc2*
OF ← (bit 31 carry ≠ bit 30 carry)
CC set if *isrc2* + *isrc1* < 0 (signed)
CC clear if *isrc2* + *isrc1* ≥ 0 (signed)

addu *isrc1, isrc2, idest* ..... **Add Unsigned**
idest ← *isrc1* + *isrc2*
OF ← bit 31 carry
CC ← bit 31 carry

and *isrc1, isrc2, idest* ..... **Logical AND**
idest ← *isrc1* and *isrc2*
CC set if result is zero, cleared otherwise

andh *#const, isrc2, idest* ..... **Logical AND High**
idest ← (*#const* shifted left 16 bits) and *isrc2*
CC set if result is zero, cleared otherwise

andnot *isrc1, isrc2, idest* ..... **Logical AND NOT**
idest ← (not *isrc1)* and *isrc2*
CC set if result is zero, cleared otherwise

andnoth *#const, isrc2, idest* ..... **Logical AND NOT High**
idest ← (not (*#const* shifted left 16 bits)) and *isrc2*
CC set if result is zero, cleared otherwise

bc *lbroff* ..... **Branch on CC**
IF CC = 1
THEN continue execution at *brx(lbroff)*
FI

bc.t *lbroff* ..... **Branch on CC, Taken**
IF CC = 1
THEN execute one more sequential instruction
continue execution at *brx(lbroff)*
ELSE skip next sequential instruction
FI

bla *isrc1ni, isrc2, sbroff* ..... **Branch on LCC and Add**
LCC-temp clear if *isrc2* + *isrc1ni* < 0 (signed)
LCC-temp set if *isrc2* + *isrc1ni* ≥ 0 (signed)
isrc2 ← *isrc1ni* + *isrc2*
Execute one more sequential instruction
IF LCC
THEN LCC ← LCC-temp
continue execution at *brx(sbroff)*
ELSE LCC ← LCC-temp
FI

bnc *lbroff* ..... **Branch on Not CC**
IF CC = 0
THEN continue execution at *brx(lbroff)*
FI

bnc.t *lbroff* ..... **Branch on Not CC, Taken**

```
IF      CC = 0
THEN    execute one more sequential instruction
        continue execution at brx(lbroff)
ELSE    skip next sequential instruction
FI
```

br *lbroff* ..... **Branch Direct Unconditionally**

Execute one more sequential instruction.
Continue execution at *brx(lbroff)*.

bri [*isrc1ni*] ..... **Branch Indirect Unconditionally**

```
Execute one more sequential instruction
IF      any trap bit in psr is set
THEN    copy PU to U, PIM to IM in psr
        clear trap bits
        IF      DS is set and DIM is reset
        THEN    enter dual-instruction mode after executing one
                    instruction in single-instruction mode
        ELSE    IF      DS is set and DIM is set
                THEN    enter single-instruction mode after executing one
                            instruction in dual-instruction mode
                ELSE    IF      DIM is set
                        THEN    enter dual-instruction mode
                                  for next instruction pair
                        ELSE    enter single-instruction mode
                                  for next instruction pair
                        FI
                FI
        FI
FI
```

Continue execution at address in *isrc1ni*
(The original contents of *isrc1ni* is used even if the next instruction modifies *isrc1ni*. Does not trap if *isrc1ni* is misaligned.)

bte *isrc1s, isrc2, sbroff* ..... **Branch If Equal**

```
IF      isrc1s = isrc2
THEN    continue execution at brx(sbroff)
FI
```

btne *isrc1s, isrc2, sbroff* ..... **Branch If Not Equal**

```
IF      isrc1s ≠ isrc2
THEN    continue execution at brx(sbroff)
FI
```

call *lbroff* ..... **Subroutine Call**

r1 ← address of next sequential instruction + 4 (or + 8 in dual mode)
Execute one more sequential instruction
Continue execution at *brx(lbroff)*

calli [*isrc1ni*] ..... **Indirect Subroutine Call**

r1 ← address of next sequential instruction + 4 (or + 8 in dual mode)
Execute one more sequential instruction
Continue execution at address in *isrc1ni*
(The original contents of *isrc1ni* is used even if the next instruction modifies *isrc1ni*. Does not trap if *isrc1ni* is misaligned. The register *isrc1ni* must not be **r1**.)

fadd.p *fsrc1, fsrc2, fdest* ..... **Floating-Point Add**

fdest ← *fsrc1* + *fsrc2*

faddp *fsrc1, fsrc2, fdest* ..... **Add with Pixel Merge**
fdest ← *fsrc1* + *fsrc2* (using integer arithmetic; 8-byte operands and destination)
Shift and load MERGE register from *fsrc1* + *fsrc2* as defined in Table 10.2

faddz *fsrc1, fsrc2, fdest* ..... **Add with Z Merge**
fdest ← *fsrc1* + *fsrc2* (using integer arithmetic; 8-byte operands and destination)
Shift MERGE right 16 and load fields 31..16 and 63..48 from *fsrc1* + *fsrc2*

famov.r *fsrc1, fdest* ..... **Floating-Point Adder Move**
fdest ← *fsrc1*

fiadd.w *fsrc1, fsrc2, fdest* ..... **Long-Integer Add**
fdest ← *fsrc1* + *fsrc2* (2's complement integer arithmetic)

fisub.w *fsrc1, fsrc2, fdest* ..... **Long-Integer Subtract**
frdest ← *fsrc1* − *fsrc2* (2's complement integer arithmetic)

fix.v *fsrc1, fdest* ..... **Floating-Point to Integer Conversion**
fdest ← 64-bit value with low-order 32 bits equal to integer part of *fsrc1* rounded

Floating-Point Load
fld.y *isrc1(isrc2), fdest* ..... **(Normal)**
fld.y *isrc1(isrc2)*++, *fdest* ..... **(Autoincrement)**
fdest ← mem.y (*isrc1* + *isrc2*)
IF autoincrement
THEN *isrc2* ← *isrc1* + *isrc2*
FI

Cache Flush
flush *#const(isrc2)* ..... **(Normal)**
flush *#const(isrc2)*++ ..... **(Autoincrement)**
Write back (if modified) the line in data cache that has address (*#const* + *isrc2*)
80860XR: and set tag value to (*#const* + *isrc2*).
80860XP: and invalidate its virtual and physical tags.
Contents of line undefined.
IF autoincrement
THEN *isrc2* ← *#const* + *isrc2*
FI

fmlow.dd *fsrc1, fsrc2, fdest* ..... **Floating-Point Multiply Low**
fdest ← low-order 53 bits of (*fsrc1* mantissa × *fsrc2* mantissa)
fdest bit 53 ← most significant bit of (*fsrc1*mantissa × *fsrc2* mantissa)

fmov.r *fsrc1, fdest* ..... **Floating-Point Reg-Reg Move**
Assembler pseudo-operation

fmov.ss *fsrc1, fdest*	= **fiadd.ss** *fsrc1*, **f0**, *fdest*
fmov.dd *fsrc1, fdest*	= **fiadd.dd** *fsrc1*, **f0**, *fdest*
fmov.sd *fsrc1, fdest*	= **famov.sd** *fsrc1, fdest*
fmov.ds *fsrc1, fdest*	= **famov.ds** *fsrc1, fdest*

fmul.p *fsrc1, fsrc2, fdest* ..... **Floating-Point Multiply**
fdest ← *fsrc1* × *fsrc2*

fnop ..... **Floating-Point No Operation**
Assembler pseudo-operation
fnop = **shrd r0, r0, r0**

form *fsrc1, fdest* ..... **OR with MERGE Register**
fdest ← *fsrc1* OR MERGE
MERGE ← 0

frcp.p *fsrc2, fdest* ..... **Floating-Point Reciprocal**
fdest ← 1 / *fsrc2* with maximum mantissa error < 2^{-7}

frsqr.p *fsrc2, fdest* ..... **Floating-Point Reciprocal Square Root**
fdest ← 1 / $\sqrt{fsrc2}$ with maximum mantissa error < 2^{-7}

Floating-Point Store
fst.y *fdest, isrc1(isrc2)* ..... **(Normal)**
fst.y *fdest, isrc1(isrc2)++* ..... **(Autoincrement)**
mem.y (*isrc2* + *isrc1*) ← *fdest*
IF autoincrement
THEN *isrc2* ← *isrc1* + *isrc2*
FI

fsub.p *fsrc1, fsrc2, fdest* ..... **Floating-Point Subtract**
fdest ← *fsrc1* – *fsrc2*

ftrunc.v *fsrc1, fdest* ..... **Floating-Point to Integer Conversion**
fdest ← 64-bit value with low-order 32 bits equal to integer part of *fsrc1*

fxfr *fsrc1, idest* ..... **Transfer F-P to Integer Register**

idest ← *fsrc1*

fzchkl *fsrc1, fsrc2, fdest* ..... **32-Bit Z-Buffer Check**
Consider the 64-bit operands as arrays of two 32-bit
fields *fsrc1*(1)..*fsrc1*(0), *fsrc2*(1)..*fsrc2*(0), and *fdest*(1)..*fdest*(0)
where zero denotes the least-significant field.
PM ← PM shifted right by 2 bits
FOR i = 0 to 1
DO
PM [i + 6] ← *fsrc2*(i) ≤ *fsrc1*(i) (unsigned)
fdest(i) ← smaller of *fsrc2*(i) and *fsrc1*(i)
OD
MERGE ← 0

fzchks *fsrc1, fsrc2, fdest* ..... **16-Bit Z-Buffer Check**
Consider the 64-bit operands as arrays of four 16-bit
fields *fsrc1*(3)..*fsrc1*(0), *fsrc2*(3)..*fsrc2*(0), and *fdest*(3)..*fdest*(0)
where zero denotes the least-significant field.
PM ← PM shifted right by 4 bits
FOR i = 0 to 3
DO
PM [i + 4] ← *fsrc2*(i) ≤ *fsrc1*(i) (unsigned)
fdest(i) ← smaller of *fsrc2*(i) and *fsrc1*(i)
OD
MERGE ← 0

intovr ..... **Software Trap on Integer Overflow**
IF OF = 1
THEN generate trap with IT set in **psr**
FI

ixfr *isrc1ni, fdest* ..... **Transfer Integer to F-P Register**
fdest ← *isrc1ni*

ld.c *csrc2, idest* ..... **Load from Control Register**
idest ← *csrc2*

ld.x *isrc1*(*isrc2*), *idest* ..... **Load Integer**
idest ← *mem.x* (*isrc1* + *isrc2*)

ldint.x *isrc2, idest* ..... **Load Interrupt Vector**
idest ← *int_vector.x (isrc2)*
NOTE: Not available with the i860 XR CPU

ldio.x *isrc2, idest* ..... **Load I/O**
idest ← *port.x (isrc2)*
NOTE: Not available with the i860 XR CPU

lock ..... **Begin Interlocked Sequence**
Set BL in **dirbase.**
The next load or store that appears on the bus locks that location.
Disable interrupts until the bus is unlocked.

mov *isrc2, idest* ..... **Register-Register Move**
Assembler pseudo-operation
mov *isrc2, idest* = **shl r0**, *isrc2, idest*

mov *const32, idest* ..... **Constant-to-Register Move**
Assembler pseudo-operation
when 0xFFFF8000 ≤ *const32* < 0x8000 . . .
adds *l%const32*, **r0**, *idest*
otherwise . . .
orh *h%const32*, **r0**, *idest*
or *l%const32, idest, idest*

nop ..... **Core-Unit No Operation**
Assembler pseudo-operation
nop = **shl r0, r0, r0**

or *isrc1, isrc2, idest* ..... **Logical OR**
idest ← *isrc1* OR *isrc2*
CC set if result is zero, cleared otherwise

orh *#const, isrc2, idest* ..... **Logical OR high**
idest ← (*#const* shifted left 16 bits) OR *isrc2*
CC set if result is zero, cleared otherwise

pfadd.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Add**
fdest ← last stage adder result
Advance A pipeline one stage
A pipeline first stage ← *fsrc1* + *fsrc2*

pfaddp *fsrc1, fsrc2, fdest* ..... **Pipelined Add with Pixel Merge**
fdest ← last-stage graphics-unit result
last-stage graphics-unit result ← *fsrc1* + *fsrc2*
(using integer arithmetic; 8-byte operands and destination)
Shift, then load MERGE register from *fsrc1* + *fsrc2* as defined in Table 10.2

pfaddz *fsrc1, fsrc2, fdest* ..... **Pipelined Add with Z Merge**
frdest ← last-stage graphics-unit result
last-stage graphics-unit result ← *fsrc1* + *fsrc2*
(using integer arithmetic; 8-byte operands and destination)
Shift MERGE right 16, then load fields 31..16 and 63..48 from*fsrc1* + *fsrc2*

pfam.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Add and Multiply**
fdest ← last stage adder result
Advance A and M pipeline one stage (operands accessed before advancing pipeline)
A pipeline first stage ← A-op1 + A-op2
M pipeline first stage ← M-op1 × M-op2

pfamov.r *fsrc1, fdest* ..... **Pipelined Floating-Point Adder Move**
fdest ← last stage adder result
Advance A pipeline one stage
A pipeline first stage ← *fsrc1*

pfeq.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Equal Compare**
fdest ← last stage adder result
CC set if *fsrc1* = *fsrc2*, else cleared
Advance A pipeline one stage
A pipeline first stage is undefined, but no result exception occurs

pfgt.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Greater-Than Compare**
(Assembler clears R-bit of instruction)
fdest ← last stage adder result
CC set if *fsrc1* > *fsrc2*, else cleared
Advance A pipeline one stage
A pipeline first stage is undefined, but no result exception occurs

pfiadd.w *fsrc1, fsrc2, fdest* ..... **Pipelined Long-Integer Add**
fdest ← last-stage graphics-unit result
last-stage graphics-unit result ← *fsrc1* + *fsrc2* (2's complement integer arithmetic)

pfisub.w *fsrc1, fsrc2, fdest* ..... **Pipelined Long-Integer Subtract**
fdest ← last-stage graphics-unit result
last-stage graphics-unit result ← *fsrc1* − *fsrc2* (2's complement integer arithmetic)

pfix.v *fsrc1, fdest* ..... **Pipelined Floating-Point to Integer Conversion**
fdest ← last stage adder result
Advance A pipeline one stage
A pipeline first stage ← 64-bit value with low-order 32 bits
equal to integer part of *fsrc1* rounded

Pipelined Floating-Point Load

pfld.y *isrc1*(*isrc2*), *fdest* ..... **(Normal)**
pfld.y *isrc1*(*isrc2*)++, *fdest* ..... **(Autoincrement)**
fdest ← mem.y (third previous **pfld**'s (*isrc1* + *isrc2*))
(where .y is precision of third previous **pfld.y**)
IF autoincrement
THEN *isrc2* ← *isrc1* + *isrc2*
FI
NOTE: pfld.q is not available with the i860 XR CPU

pfle.p *fsrc1, fsrc2, fdest* ..... **Pipelined F-P Less-Than or Equal Compare**
Assembler sets R-bit of instruction
fdest ← last stage adder result
CC clear if *fsrc1* ≤ *fsrc2*, else set
Advance A pipeline one stage
A pipeline first stage is undefined, but no result exception occurs

pfmam.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Add and Multiply**
fdest ← last stage multiplier result
Advance A and M pipeline one stage (operands accessed before advancing pipeline)
A pipeline first stage ← A-op1 + A-op2
M pipeline first stage ← M-op1 × M-op2

pfmov.r *fsrc1, fdest* ..... **Pipelined Floating-Point Reg-Reg Move**
Assembler pseudo-operation
pfmov.ss *fsrc1, fdest* = **pfiadd.ss** *fsrc1*, **f0**, *fdest*
pfmov.dd *fsrc1, fdest* = **pfiadd.dd** *fsrc1*, **f0**, *fdest*
pfmov.sd *fsrc1, fdest* = **pfamov.sd** *fsrc1*, *fdest*
pfmov.ds *fsrc1, fdest* = **pfamov.ds** *fsrc1*, *fdest*

pfmsm.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Subtract and Multiply**
fdest ← last stage multiplier result
Advance A and M pipeline one stage (operands accessed before advancing pipeline)
A pipeline first stage ← A-op1 − A-op2
M pipeline first stage ← M-op1 × M-op2

pfmul.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Multiply**
fdest ← last stage multiplier result
Advance M pipeline one stage
M pipeline first stage ← *fsrc1* × *fsrc2*

pfmul3.dd *fsrc1, fsrc2, fdest* ..... **Three-Stage Pipelined Multiply**
fdest ← last stage multiplier result
Advance 3-Stage M pipeline one stage
M pipeline first stage ← *fsrc1* × *fsrc2*

pform *fsrc1, fdest* ..... **Pipelined OR to MERGE Register**
fdest ← last-stage graphics-unit result
last-stage graphics-unit result ← *fsrc1* OR MERGE
MERGE ← 0

pfsm.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Subtract and Multiply**
fdest ← last stage adder result
Advance A and M pipeline one stage (operands accessed before advancing pipeline)
A pipeline first stage ← A-op1 − A-op2
M pipeline first stage ← M-op1 × M-op2

pfsub.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Subtract**
fdest ← last stage adder result
Advance A pipeline one stage
A pipeline first stage ← *fsrc1* − *fsrc2*

pftrunc.v *fsrc1, fdest* ..... **Pipelined Floating-Point to Integer Conversion**
fdest ← last stage adder result
Advance A pipeline one stage
A pipeline first stage ← 64-bit value with low-order 32 bits
equal to integer part of *fsrc1*

pfzchkl *fsrc1, fsrc2, fdest* ..... **Pipelined 32-Bit Z-Buffer Check**

Consider the 64-bit operands as arrays of two 32-bit
fields *fsrc1*(1)..*fsrc1*(0), *fsrc2*(1)..*fsrc2*(0), and *fdest*(1)..*fdest*(0)
where zero denotes the least-significant field.
PM ← PM shifted right by 2 bits
FOR i = 0 to 1
DO
PM [i + 6] ← *fsrc2*(i) ≤ *fsrc1*(i) (unsigned)
fdest(i) ← last-stage graphics-unit result
last-stage graphics-unit result ← smaller of *fsrc2*(i) and *fsrc1*
OD
MERGE ← 0

pfzchks *fsrc1, fsrc2, fdest* ..... **Pipelined 16-Bit Z-Buffer Check**

Consider the 64-bit operands as arrays of four 16-bit
fields *fsrc1*(3)..*fsrc1*(0), *fsrc2*(3)..*fsrc2*(0), and *fdest*(3)..*fdest*(0)
where zero denotes the least-significant field.
PM ← PM shifted right by 4 bits
FOR i = 0 to 3
DO
PM [i + 4] ← *fsrc2*(i) ≤ *fsrc1*(i) (unsigned)
fdest ← last-stage graphics-unit result
last-stage graphics-unit result(i) ← smaller of *fsrc2*(i) and *fsrc1*(i)
OD
MERGE ← 0

pst.d *fdest, #const(isrc2)* ..... **Pixel Store**
pst.d *fdest, #const(isrc2)++* ..... **Pixel Store Autoincrement**

Pixels enabled by PM in *mem.d* (*isrc2* + *#const*) ← *fdest*
Shift PM right by 8/pixel size (in bytes) bits
IF autoincrement
THEN *isrc2* ← *#const* + *isrc2*
FI

scyc.x *isrc2* ..... **Special Cycles**

Generate a special bus cycle (D/C# = 0, W/R# = 1, M/IO# = 0) and
set BE7# – BE0# according to the value contained in the register *isrc2*
NOTE: Not available with the i860 XR CPU

shl *isrc1, isrc2, idest* ..... **Shift Left**

idest ← *isrc2* shifted left by *isrc1* bits

shr *isrc1, isrc2, idest* ..... **Shift Right**

SC (in **psr**) ← *isrc1*
idest ← *isrc2* shifted right by *isrc1* bits

shra *isrc1, isrc2, idest* ..... **Shift Right Arithmetic**

idest ← *isrc2* arithmetically shifted right by *isrc1* bits

shrd *isrc1ni, isrc2, idest* ..... **Shift Right Double**

idest ← low-order 32 bits of *isrc1ni:isrc2* shifted right by SC bits

st.c *isrc1ni, csrc2* ..... **Store to Control Register**

csrc2 ← *src1ni*

st.x *isrc1ni, #const(isrc2)* ..... **Store Integer**

mem.x (*isrc2* + *#const*) ← *isrc1ni*

stio.x *isrc1ni, isrc2* ..... **Store I/O**
port.x (isrc2) ← *isrc1ni*
NOTE: Not available with the i860 XR CPU

subs *isrc1, isrc2, idest* ..... **Subtract Signed**
idest ← *isrc1* − *isrc2*
OF ← (bit 31 carry ≠ bit 30 carry)
CC set if *isrc2* > *isrc1* (signed)
CC clear if *isrc2* ≤ *isrc1* (signed)

subu *isrc1, isrc2, idest* ..... **Subtract Unsigned**
idest ← *isrc1* − *isrc2*
OF ← NOT (bit 31 carry)
CC ← bit 31 carry
(i.e. CC set if *isrc2* ≤ *isrc1* (unsigned)
CC clear if *isrc2* > *isrc1* (unsigned))

trap *isrc1ni, isrc2, idest* ..... **Software Trap**
Generate trap with IT set in **psr**

unlock ..... **End Interlocked Sequence**
Clear BL in **dirbase**. The next load or store unlocks the bus. Interrupts are enabled.

xor *isrc1, isrc2, idest* ..... **Logical Exclusive OR**
idest ← *isrc1* XOR *isrc2*
CC set if result is zero, cleared otherwise

xorh *#const, isrc2, idest* ..... **Logical Exclusive OR High**
idest ← (*#const* shifted left 16 bits) XOR *isrc2*
CC set if result is zero, cleared otherwise

Table 10.2. FADDP MERGE Update

Pixel Size (from PS)	Fields Loaded from Result into MERGE				Right Shift Amount (Field Size)
8	63..56,	47..40,	31..24,	15..8	8
16	63..58,	47..42,	31..26,	15..10	6
32	63..56,		31..24		8

10.2 Instruction Format and Encoding

All instructions are 32 bits long and begin on a four-byte boundary. When operands are registers, the encodings shown in Table 10.3 are used.

There are two general core-instruction formats (REG-format and CTRL-format) and a separate format for floating-point instructions.

Table 10.3. Register Encoding

Register	Encoding
r0	0
.	.
.	.
.	.
r31	31
f0	0
.	.
.	.
.	.
f31	31
Fault Instruction	0
Processor Status	1
Directory Base	2
Data Breakpoint	3
Floating-Point Status	4
Extended Processor Status	5
Bus Error Address*	6
Concurrency Control*	7
p0*	8
p1*	9
p2*	10
p3*	11

NOTE:
*Available only with i860 XP CPU. Using these encodings with the i860 XR CPU produces undefined results.

10.2.1 REG-FORMAT INSTRUCTIONS

Within the REG-format are several variations as shown in Figure 10.1. Table 10.4 gives the encodings for these instructions. One encoding is an escape code that defines yet another variation: the core escape instructions. Figure 10.2 shows the format of this group, and Table 10.5 shows the encodings.

In these instructions, the *src2* field selects one of the 32 integer registers (most instructions) or one of the control registers (**st.c** and **ld.c**). *Dest* selects one of the 32 integer registers (most instructions) or floating-point registers (**fld**, **fst**, **pfld**, **pst**, **ixfr**). For instructions where *src1* is optionally an immediate value, bit 26 of the opcode (I-bit) indicates whether *src1* is an immediate. If bit 26 is clear, an integer register is used; if bit 26 is set, *src1* is contained in the low-order 16 bits, except for **bte** and **btne** instructions. For **bte** and **btne**, the five-bit immediate value is contained in the *src1* field. For **st**, **bte**, **btne**, and **bla**, the upper five bits of the *offset* or *broffset* are contained in the *dest* field instead of *src1*, and the lower 11 bits of *offset* are the lower 11 bits of the instruction.

For **ld** and **st**, bits 28 and zero determine operand size as follows:

Bit 28	Bit 0	Operand Size
0	0	8-bits
0	1	8-bits
1	0	16-bits
1	1	32-bits

When *src1* is immediate and bit 28 is set, bit zero of the immediate value is forced to zero.

For **fld, fst, pfld, pst,** and **flush,** bit 0 selects autoincrement addressing if set. For **fld, fst, pfld,** and **pst,** bits one and two select the operand size as follows:

Bit 1	Bit 2	Operand Size
0	0	64-bits
0	1	128-bits
1	0	32-bits
1	1	32-bits

For **flush,** bits one and two must be zero.

When *src1* is immediate, bits zero and one of the immediate value are forced to zero to maintain alignment. When bit one of the immediate value is clear, bit two is also forced to zero.

For the instructions **ldio, stio, ldint,** and **scyc**, the operand size is encoded by bits 9 and 10 as follows. For other instructions, these bits are *reserved* and should be set to zero.

Operand Size	Bit 10	Bit 9
8 Bits (.b)	0	0
16 Bits (.s)	0	1
32 Bits (.l)	1	0
reserved	1	1

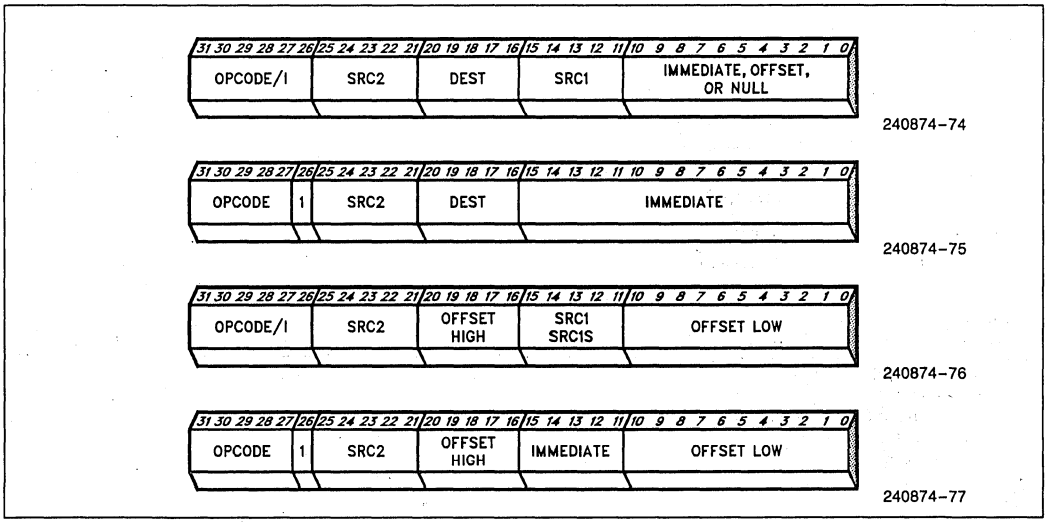


Figure 10.1. REG-Format Variations

Table 10.4. REG-Format Opcodes

		31	30	29	28	27	26
ld.x	Load Integer	0	0	0	L	0	I
st.x	Store Integer	0	0	0	L	1	1
ixfr	Integer to F-P Reg Transfer	0	0	0	0	1	0
—	*(reserved)*	0	0	0	1	1	0
fld.x, fst.x	Load/Store F-P	0	0	1	0	LS	I
flush	Flush	0	0	1	1	0	1
pst.d	Pixel Store	0	0	1	1	1	1
ld.c, st.c	Load/Store Control Register	0	0	1	1	LS	0
bri	Branch Indirect	0	1	0	0	0	0
trap	Trap	0	1	0	0	0	1
—	(Escape for F-P Unit)	0	1	0	0	1	0
—	(Escape for Core Unit)	0	1	0	0	1	1
bte, btne	Branch Equal or Not Equal	0	1	0	1	E	I
pfld.y	Pipelined F-P Load	0	1	1	0	0	I
—	(CTRL-Format Instructions)	0	1	1	x	x	x
addu, -s, subu, -s	Add/Subtract	1	0	0	SO	AS	I
shl, shr	Logical Shift	1	0	1	0	LR	I
shrd	Double Shift	1	0	1	1	0	0
bla	Branch LCC Set and Add	1	0	1	1	0	1
shra	Arithmetic Shift	1	0	1	1	1	I
and(h)	AND	1	1	0	0	H	I
andnot(h)	ANDNOT	1	1	0	1	H	I
or(h)	OR	1	1	1	0	H	I
xor(h)	XOR	1	1	1	1	H	I
—	*(reserved)*	1	1	x	x	1	0

L Integer Length
- 0 —8 bits
- 1 —16 or 32 bits (selected by bit 0)

LS Load/Store
- 0 —Load
- 1 —Store

SO Signed/Ordinal
- 0 —Ordinal
- 1 —Signed

H High
- 0 **—and, or, andnot, xor**
- 1 **—andh, orh, andnoth, xorh**

AS Add/Subtract
- 0 —Add
- 1 —Subtract

LR Left/Right
- 0 —Left Shift
- 1 —Right Shift

E Equal
- 0 —Branch on Unequal
- 1 —Branch on Equal

I Immediate
- 0 —*src1* is register
- 1 —*src1* is immediate

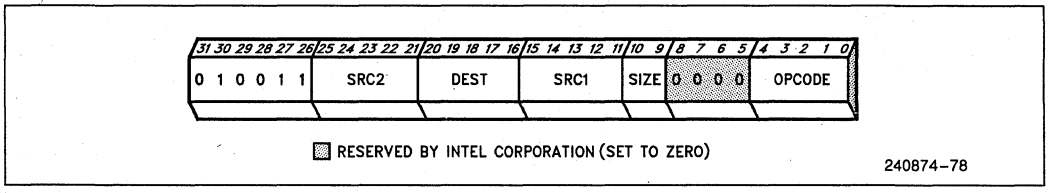


Figure 10.2. Core Escape Instructions

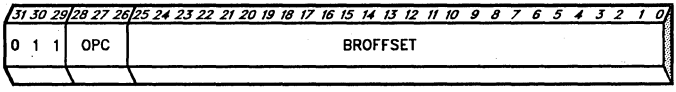
Table 10.5. Core Escape Opcodes

		4	3	2	1	0
—	*(reserved)*	0	0	0	0	0
lock	Begin Interloacked Sequence	0	0	0	0	1
calli	Indirect Subroutine Call	0	0	0	1	0
—	*(reserved)*	0	0	0	1	1
introvr	Trap on Integer Overflow	0	0	1	0	0
—	*(reserved)*	0	0	1	0	1
—	*(reserved)*	0	0	1	1	0
unlock	End Interlocked Sequence	0	0	1	1	1
ldio*	Load I/O	0	1	0	0	0
stio*	Store I/O	0	1	0	0	1
ldint*	Load Interrupt Vector	0	1	0	1	0
scyc*	Special Cycles	0	1	0	1	1
—	*(reserved)*	0	1	1	x	x
—	*(reserved)*	1	0	x	x	x
—	*(reserved)*	1	1	x	x	x

NOTE:
*Available only with i860 XP CPU, not with i860 XR CPU

10.2.2 CTRL-FORMAT INSTRUCTIONS

The CTRL-Format instructions do not refer to registers; so, instead of the register fields, they have a 26-bit relative branch offset. Figure 10.3 shows the format of these instructions and Table 10.6 defines the encodings.



240874-79

NOTE:
BROFFSET is a signed 26-bit relative branch offset

Figure 10.3. CTRL-Format Instructions

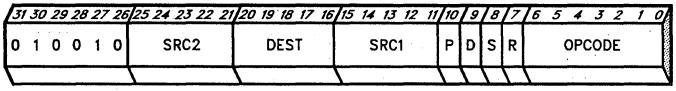
Table 10.6. CTRL-Format Opcodes

		28	27	26
—	*(reserved)*	0	0	0
—	*(reserved)*	0	0	1
br	Branch Direct	0	1	0
call	Call	0	1	1
bc(.t)	Branch on CC Set	1	0	T
bnc(.t)	Branch on CC Clear	1	1	T

T Taken
0 —**bc** or **bnc**
1 —**bc.t** or **bnc.t**

10.2.3 FLOATING-POINT INSTRUCTION ENCODING

The floating-point instructions also constitute an escape series. All these instructions begin with the bit sequence 010010. Figure 10.4 shows the format of the floating-point instructions, and Table 10.7 gives the encodings. Within the dual-operation instructions is a subcode DPC whose values are given in Table 10.9 along with the mnemonic that corresponds to each.



240874-80

SRC1, SRC2 Source; one of 32 floating-point registers
DEST Destination; one of 32 floating-point registers (except **fxfr**; one of 32 integer registers)

P Pipelining
- 1 Pipelined instruction mode
- 0 Scalar instruction mode

D Dual-Instruction Mode
- 1 Dual-instruction mode
- 0 Single-instruction mode

S Source Precision
- 1 Double-precision source operands
- 0 Single-precision source operands

R Result Precision
- 1 Double-precision result
- 0 Single-precision result

Figure 10.4. Floating-Point Instruction Encoding

Table 10.7. Floating-Point Opcodes

		6	5	4	3	2	1	0
pfam	Add and Multiply*	0	0	0	DPC			
pfmam	Multiply with Add*							
pfsm	Subtract and Multiply*	0	0	1	DPC			
pfmsm	Multiply with Subtract*							
(p)fmul	Multiply	0	1	0	0	0	0	0
fmlow	Multiply Low	0	1	0	0	0	0	1
frcp	Reciprocal	0	1	0	0	0	1	0
frsqr	Reciprocal Square Root	0	1	0	0	0	1	1
pfmul3.dd	3-Stage Pipelined Multiply	0	1	0	0	1	0	0
(p)fadd	Add	0	1	1	0	0	0	0
(p)fsub	Subtract	0	1	1	0	0	0	1
(p)fix	Fix	0	1	1	0	0	1	0
(p)famov	Adder Move	0	1	1	0	0	1	1
pfgt/pfle**	Greater Than	0	1	1	0	1	0	0
pfeq	Equal	0	1	1	0	1	0	1
(p)ftrunc	Truncate	0	1	1	1	0	1	0
fxfr	Transfer to Integer Register	1	0	0	0	0	0	0
(p)fiadd	Long-Integer Add	1	0	0	1	0	0	1
(p)fisub	Long-Integer Subtract	1	0	0	1	1	0	1
(p)fzchkl	Z-Check Long	1	0	1	0	1	1	1
(p)fzchks	Z-Check Short	1	0	1	1	1	1	1
(p)faddp	Add with Pixel Merge	1	0	1	0	0	0	0
(p)faddz	Add with Z Merge	1	0	1	0	0	0	1
(p)form	OR with MERGE Register	1	0	1	1	0	1	0

NOTE:
All opcodes not shown are *reserved*.
* **pfam** and **pfsm** have P-bit set; **pfmam** and **pfmsm** have P-bit clear.
** **pfgt** has R bit cleared; **pfle** has R bit set.

Table 10.8. DPC Encoding

DPC	PFAM Mnemonic	PFSM Mnemonic	M-Unit o1	M-Unit op2	A-Unit op1	A-Unit op2	T Load	K Load*
0000	**r2p1**	r2s1	KR	src2	src1	M result	No	No
0001	**r2pt**	r2st	KR	src2	T	M result	No	Yes
0010	**r2ap1**	r2as1	KR	src2	src1	A result	Yes	No
0011	**r2apt**	r2ast	KR	src2	T	A result	Yes	Yes
0100	**i2p1**	i2s1	KI	src2	src1	M result	No	No
0101	**i2pt**	i2st	KI	src2	T	M result	No	Yes
0110	**i2ap1**	i2as1	KI	src2	src1	A result	Yes	No
0111	**i2apt**	i2ast	KI	src2	T	A result	Yes	Yes
1000	**rat1p2**	rat1s2	KR	A result	src1	src2	Yes	No
1001	**m12apm**	m12asm	src1	src2	A result	M result	No	No
1010	**ra1p2**	ra2s2	KR	A result	src1	src2	No	No
1011	**m12ttpa**	m12ttsa	src1	src2	T	A result	Yes	No
1100	**iat1p2**	iat1s2	KI	A result	src1	src2	Yes	No
1101	**m12tpm**	m12tsm	src1	src2	T	M result	No	No
1110	**ia1p2**	ia1s2	KI	A result	src1	src2	No	No
1111	**m12tpa**	m12tsa	src1	src2	T	A result	No	No
DPC	**PFMAM Mnemonic**	**PFMSM Mnemonic**	**M-Unit op1**	**M-Unit op2**	**A-Unit op1**	**A-Unit op2**	**T Load**	**K Load***
0000	**mr2p1**	mr2s1	KR	src2	src1	M result	No	No
0001	**mr2pt**	mr2st	KR	src2	T	M result	No	Yes
0010	**mr2mp1**	mr2ms1	KR	src2	src1	M result	Yes	No
0011	**mr2mpt**	mr2mst	KR	src2	T	M result	Yes	Yes
0100	**mi2p1**	mi2s1	KI	src2	src1	M result	No	No
0101	**mi2pt**	mi2st	KI	src2	T	M result	No	Yes
0110	**mi2mp1**	mi2ms1	KI	src2	src1	M result	Yes	No
0111	**mi2mpt**	mi2mst	KI	src2	T	M result	Yes	Yes
1000	**mrmt1p2**	mrmt1s2	KR	M result	src1	src2	Yes	No
1001	**mm12mpm**	mm12msm	src1	src2	M result	M result	No	No
1010	**mrm1p2**	mrm1s2	KR	M result	src1	src2	No	No
1011	**mm12ttpm**	mm12ttsm	src1	src2	T	M result	Yes	No
1100	**mimt1p2**	mimt1s2	KI	M result	src1	src2	Yes	No
1101	**mm12tpm**	mm12tsm	src1	src2	T	M result	No	No
1110	**mim1p2**	mim1s2	KI	M result	src1	src2	No	No
1111	Intel Reserved							

NOTE:
* If K-load is set, KR is loaded when operand-1 of the multiplier is KR; KI is loaded when operand-1 of the multiplier is KI.

10.3 Instruction Timings

Generally, i860 XP microprocessor instructions take one clock to execute unless a freeze condition is invoked. Detailed times, along with freeze conditions and their associated delays, are shown in the table on the following pages. The following symbols are used for brevity in the timing table:

+n *n* clocks must be added to the execution time if the stated conditions apply.

⟷ n The processor requires at least *n* clocks between the indicated instructions. The actual delay will be *n* minus the number of clocks for executing intervening instructions (or dual-mode pairs). If the time for intervening instructions is ≥ *n*, there is no delay.

n..m Indicates a range of clocks. These cases are accompanied by a reference to a note where further explanation is available.

XR: Applies to i860 XR microprocessors only.

XP: Applies to i860 XP microprocessors only.

OA The number of clocks to finish all outstanding accesses.

R1 The number of clocks from ADS# through the first READY# (80860XR) or BRDY# (80860XP) of the indicated bus activity.

R2 The number of clocks from ADS# through the second READY# or BRDY#.

RL The number of clocks from ADS# through the last READY# or BRDY#.

RL1 XP: The number of clocks through last BRDY# of first access.

RN XR: The number of clocks until next nonrepeated address can be issued (i.e., an address that is not the 2nd–4th cycle of a cache fill, the 2nd–8th cycle of a CS8 mode instruction fetch, nor the 2nd cycle of a 128-bit write).

RX The number of clocks through READY# or BRDY# for the next 64-bit-or-less write cycle or second READY# or BRDY# for the next 128-bit write cycle.

NOTES:

a. "Address path full" means one address internally waiting for bus while external bus pipeline full.

b. "Store path full" means two stores or one 256-bit write-back internally waiting for bus plus external bus pipeline full.

c. If a floating-point instruction, graphics-unit instruction, **fst**, or **pst** is executed when a scalar floating-point operation (other than **frcp** or **frsqr**) is in progress, the scalar operation must complete first: two additional clocks for **fadd**, **fix**, **fmlow**, **fmul.ss**, **fmul.sd**, **ftrunc**, and **fsub**; three additional clocks for **fmul.dd**. Add one if either or both of these situations occur:

1. There is an overlap between the result register of the previous scalar operation and the source of the floating-point operation, and the destination precision of the scalar operation differs from the source precision of the floating-point operation.
2. The floating-point operation is pipelined and its destination is not **f0**.

TLB TLB miss. Five clocks plus the number of clocks to finish two reads plus the number of clocks to set A-bits (if necessary).

In addition, any instruction may be delayed due to an instruction cache miss or TLB miss during the instruction fetch. The time for a TLB miss is shown above in note **TLB**. An instruction cache miss adds the following delays:

- The number of clocks to get the next instruction from the bus (ADS# clock to first READY# or BRDY# clock, inclusive).
- XR: When any of the instructions in the new instruction-cache line is a branch or call or causes a freeze, the time through the last READY# for the new line.
- If the data cache is being accessed when the instruction-cache miss occurs, two clocks for data cache miss; one clock for hit.

Not included in the table is the delay caused by a trap. This depends on the trap handler.

In dual instruction mode, each pair of instructions requires the maximum of the times required by each individual instruction.

Instruction	Execution Clocks	Condition
adds	1	
addu	1	
and	1	
andh	1	
andnot	1	
andnoth	1	
bc	1	If branch not taken.
	2	If branch taken.
	+	If the prior instruction is **addu, adds, subu, subs, pfeq,** or **pfgt**.
bc.t	1	If branch taken.
	2	If branch not taken.
	+1	If the prior instruction is **addu, adds, subu, subs, pfeq,** or **pfgt**.
bla	1	If branch taken.
	2	If branch not taken.
bnc	(same as **bc**)	
bnc.t	(same as **bc.t**)	
br	1	
brl	2	
bte	1	If branch not taken.
	3	If branch taken.
btne	(same as **bte**)	
call	1	
	+1	If **r1** referenced in next instruction.
	+1+R1	If data cache load miss in progress for a read of less than 128 bits.
	+1+R2	If data cache load miss in progress for 128-bit read.
calli	2	
	+1	If **r1** referenced in next instruction.
	+1+R1	If data cache load miss in progress for a read of less than 128 bits.
	+1+R2	If data cache load miss in progress for 128-bit read.
fadd.p	1	(. . . and all other A-unit instructions except dual operations)
	⟷ 2..4	If executed when a scalar floating-point operation (other than **frcp** or **frsqr**) is in progress.(c)

Instruction	Execution Clocks	Condition
faddp	1	(. . . and all other G-unit instructions except **fiadd.w, fxfr**)
	+1	If *fdest* is used by next instruction and next instruction is G-, M- or A-unit instruction
	⟷ 2..4	If executed when a scalar floating-point operation (other than **frcp** or **frsqr**) is in progress.(c)
faddz	(same as **faddp**)	
famov.r	(same as **fadd.p**)	
fiadd.w	1	
	+1	If *fdest* is used by next instruction and next instruction is M- or A-unit instruction (except when **fiadd** is used for **fmov.dd** or **fmov.ss**).
	+1	If *fdest* is used by next instruction and next instruction is G-unit instruction.
	⟷ 2..4	If executed when a scalar floating-point operation (other than **frcp** or **frsqr**) is in progress.(c)
fisub.w	(same as **faddp**)	
fix.v	(same as **fadd.p**)	
fld.y	1	
	+1	If this is the instruction after a **st, fst** or **pst** that hits the data cache.
	⟷ 2	If *fdest* is referenced in the next two instructions.
	+1+R1	If 32-bit **fld.l** or 64-bit **fld.d** misses the data cache.
	+1+R2	If 128-bit **fld.q** misses the data cache.
	+1+RL	If data cache load miss in progress (except in the following case).
	⟷ 2	XP: If this instruction follows a data cache access that misses in the virtual tags but hits in the physical tags.
	+2	XP: If the prior instruction is a **pfld.y** that hits a modified line in the data cache.
	+R2	XP: If data-cache line write-back due to snoop is in progress.
	+RN	XR: If address path full.(a)
	+RL1	XP: If address path full.(a)
	+TLB	If TLB miss.
flush	1	
	⟷ 3	XR: If preceded by another **flush**.
	⟷ 2	XP: If preceded by another **flush**.
	+R2	XP: If data-cache line write-back due to snoop is in progress.
	+1+RX	If flush to modified line when store path full.(b)
	+TLB	If TLB miss.
fmlow.dd	1	(. . . and all other M-unit instruction except dual operations)
	+1	If *fsrc1* refers to result of the prior operation (either scalar or pipelined).
	+1	If the prior operation is a double-precision multiply.
	⟷ 2..4	If executed when a scalar floating-point operation (other than **frcp** or **frsqr**) is in progress.(c)
fmov.r		**fmov.ss** and **fmov.dd** same as **fiadd.w** **fmov.sd** and **fmov.ds** same as **fadd.p**
fmul.p	(same as **fmlow.dd**)	

Instruction	Execution Clocks	Condition
fnop	1	
form	(same as **faddp**)	
frcp.p	(same as **fmlow.dd**)	
frsqr.p	(same as **fmlow.dd**)	
fst.y	1	
	+1	If followed by pipelined floating-point operation that overwrites the register being stored.
	+1+RL	If data cache load miss in progress.
	+2	XP: If the prior instruction is a **pfld.y** that hits a modified line in the data cache.
	←→ 2	XP: If this instruction follows a data cache access that misses in the virtual tags but hits in the physical tags.
	+R2	XP: If data-cache line write-back due to snoop is in progress.
	←→ 2..4	If executed when a scalar floating-point operation (other than **frcp** or **frsqr**) is in progress.(c)
	+RN	XR: If address path full.(a)
	+RL1	XP: If address path full.(a)
	+1+RX	If cache miss when store path full.(b)
	+TLB	If TLB miss.
fsub.p	(same as **fadd.p**)	
ftrunc.v	(same as **fadd.p**)	
fxfr	1	
	+1	If *idest* referenced in next instruction.
	+1+R1	If data cache load miss in progress for 64-bit read.
	+1+R2	If data cache load miss in progress for 128-bit read.
	←→ 2..4	If executed when a scalar floating-point operation (other than **frcp** or **frsqr**) is in progress.(c)
fzchkl	(same as **faddp**)	
fzchks	(same as **faddp**)	
intovr	1	
ixfr	1	
	+1+R1	If data cache load miss in progress for 64-bit read.
	+1+R2	If data cache load miss in progress for 128-bit read.
	←→ 2	If *fdest* is referenced in the next two instructions.
ld.c	1	
	+1	If *idest* referenced in next instruction.
	+1+R1	If data cache load miss in progress for 64-bit read.
	+1+R2	If data cache load miss in progress for 128-bit read.

Instruction	Execution Clocks	Condition
ld.x	1	
	+1	If *idest* referenced in next instruction.
	+1	If this is the instruction after a **st, fst** or **pst** that hits the data cache.
	+1+RL	If data cache load miss in progress.
	⟷ 1+R1	If **ld.x** misses the data cache and a subsequent instruction references the *idest* of the **ld.x** (except for following case).
	⟷ 2	XP: If this instruction follows a data cache access that misses in the virtual tags but hits in the physical tags.
	+2	XP: If the prior instruction is a **pfld.y** that hits a modified line in the data cache.
	+R2	XP: If data-cache line write-back due to snoop is in progress.
	+RN	XR: If address path full.(a)
	+RL1	XP: If address path full.(a)
	+1+RX	If cache miss when store path full.(b)
	+TLB	If TLB miss.
ldint.x	1	+ OA
ldio.x	1	+ OA
lock	1	
mov	1	
nop	1	
or	1	
orh	1	
pfadd.p	(same as **fadd.p**)	
pfaddp	(same as **faddp**)	
pfaddz	(same as **faddp**)	
pfam.p	1	(. . . and all other dual operations)
	+1	If *fsrc1* refers to result of the prior operation (either scalar or pipelined).
	+1	If the prior operation is a double-precision multiply.
	⟷ 2..4	If executed when a scalar floating-point operation (other than **frcp** or **frsqr**) is in progress.(c)
pfamov.r	(same as **fadd.p**)	
pfeq.p	(same as **fadd.p**)	
pfgt.p	(same as **fadd.p**)	
pfiadd.w	(same as **faddp**)	
pfisub.w	(same as **faddp**)	
pfix.v	(same as **fadd.p**)	

Instruction	Execution Clocks	Condition
pfld.y	1	
	+1+RL	If data cache load miss in progress.
	⟷ 2	If *fdest* is referenced in the next two instructions.
	+1+RL1	If three **pfld's** are outstanding.
	+2+OA	XR: If **pfld** hits data cache.
	+2	XP: If the prior instruction is a **pfld.y** that hits a modified line in the data cache.
	⟷ 2	XP: If this instruction follows a data cache access that misses in the virtual tags but hits in the physical tags.
	+R2	XP: If data-cache line write-back due to snoop is in progress.
	+RN	XR: If address path full.(a)
	+RL1	XP: If address path full.(a)
	+TLB	If TLB miss.
pfle.p	1	
pfmam.p	(same as **pfam.p**)	
pfmov.r		**pfmov.ss** and **pfmov.dd** same as **faddp** **pfmov.sd** and **pfmov.ds** same as **fadd.p**
pfmsm.p	(same as **pfam.dd**)	
pfmul.p	(same as **fmlow.dd**)	
pfmul3.dd	(same as **fmlow.dd**)	
pform	(same as **faddp**)	
pfsm.p	(same as **pfam.dd**)	
pfsub.p	(same as **fadd.p**)	
pftrunc.v	(same as **fadd.p**)	
pfzchkl	(same as **faddp**)	
pfzchks	(same as **faddp**)	
pst.d	(same as **fst.d**)	
scyc.x	1 + OA	
shl	1	
shr	1	
shra	1	
shrd	1	
st.c	3	
	+1+R1	If data cache load miss in progress for a read of less than 128 bits.
	+1+R2	If data cache load miss in progress for 128-bit read.

Instruction	Execution Clocks	Condition
st.x	1	
	+1+RL	If data cache load miss in progress.
	+2	XP: If the prior instruction is a **pfld.y** that hits a modified line in the data cache.
	⟷ 2	XP: If this instruction follows a data cache access that misses in the virtual tags but hits in the physical tags.
	+R2	XP: If data-cache line write-back due to snoop is in progress.
	+RN	XR: If address path full.(a)
	+RL1	XP: If address path full.(a)
	+1+RX	If cache miss when store path full.(b)
	+TLB	If TLB miss.
stio.x	1 + OA	
subs	1	
subu	1	
trap	1	
unlock	1	
xor	1	
xorh	1	

10.4 Instruction Characteristics

The following table lists some of the characterisics of each instruction. The characteristics are:

- What processing unit executes the instruction. The codes for processing units are:
  - **A** Floating-point adder unit
  - **E** Core execution unit
  - **G** Graphics unit
  - **M** Floating-point multiplier unit
- Whether the instruction is pipelined or not. A *P* indicates that the instruction is pipelined.
- Whether the instruction is a delayed branch instruction. A *D* marks the delayed branches.
- Whether execution is suppressed in user mode. An *SU* marks supervisor-only instructions.
- Whether the instruction is available on both the i860 XR and i860 XP microprocessors. An *XL* marks instructions that are available only on the i860 XP microprocessor.
- Whether the instruction changes the condition code CC. A *CC* marks those instructions that change CC.
- Which faults can be caused by the instruction. The codes used for exceptions are:
  - **IT** Instruction Fault
  - **SE** Floating-Point Source Exception
  - **RE** Floating-Point Result Exception, including overflow, underflow, inexact result
  - **DAT** Data Access Fault

Note that this is not the same as specifying at which instructions faults may be reported. A result exception is reported on the subsequent floating-point instruction, **pst**, **fst**, or sometimes **fld**, **pfld**, and **ixfr**.

The instruction access fault IAT and the interrupt trap IN are not shown in the table because they can occur for any instruction.

- Performance notes. These comments regarding optimum performance are recommendations only. If these recommendations are not followed, the i860 XP microprocessor automatically waits the necessary number of clocks to satisfy internal hardware requirements. The following notes define the numeric codes that appear in the instruction table:
  1. The following instruction should not be a conditional branch (**bc**, **bnc**, **bc.t**, or **bnc.t**).
  2. The destination should not be a source operand of the next two instructions.
  3. A load should not directly follow a store that is expected to hit in the data cache.
  4. When the prior instruction is scalar, *fsrc1* should not be the same as the *fdest* of the prior operation.

5. The *fdest* should not reference the destination of the next instruction if that instruction is a pipelined floating-point operation.
6. The destination should not be a source operand of the next instruction. (For **call** and **calli**, the destination is **r1**.)
7. When the prior operation is scalar and multiplier *op1* is *fsrc1, fsrc2* should not be the same as the *fdest* of the prior operation.
8. When the prior operation is scalar, *src1* and *src2* of the current operation should not be the same as *dest* of the prior operation.
9. A **pfld** should not immediately follow a **pfld**.

- Programming restrictions. These indicate combinations of conditions that must be avoided by programmers, assemblers, and compilers. The following notes define the alphabetic codes that appear in the instruction table:
  - **a.** The sequential instruction following a delayed control-transfer instruction may not be another control-transfer instruction, nor a **trap** instruction, nor the target of a control-transfer instruction.
  - **b.** When using a **bri** to return from a trap handler, programmers should take care to prevent traps from occurring on that or on the next sequential instruction. IM should be zero (interrupts disabled) when the **bri** is executed.
  - **c.** If *fdest* is not zero, *fsrc1* must not be the same as *fdest*.
  - **d.** When *fsrc1* goes to multiplier *op1* or to KR or KI, *fsrc1* must not be the same as *fdest*.
  - **e.** If *dest* is not zero, *src1* and *src2* must not be the same as *dest*.
  - **f.** *Isrc1* must not be the same register as *isrc2* for the autoincrementing form of this instruction.
  - **g.** *Isrc1* must not be the same register as *isrc2*.
  - **h.** **flush** must not be used in a locked sequence or in dual instruction mode.

2

Instruction	Execution Unit	Pipelined? Delayed? Supervisor? i860™ XP Only?	Sets CC?	Faults	Performance Notes	Programming Restrictions
adds	E		CC		1	
addu	E		CC		1	
and	E		CC			
andh	E		CC			
andnot	E		CC			
andnoth	E		CC			
bc	E					
bc.t	E	D				a
bla	E	D				a,g
bnc	E					
bnc.t	E	D				a
br	E	D				a
bri	E	D				a,b
bte	E					
btne	E					
call	E	D			6	a
calli	E	D			6	a
fadd.p	A			SE,RE		
faddp	G				8	
faddz	G				8	
famov.r	A			SE,RE		
fiadd.w	G				8	
fisub.w	G				8	
fix.p	A			SE,RE		
fld.y	E			DAT	2,3	f

NOTES:
* On the i860 XP microprocessor, the pipelined instructions can generate ITR with PI.
** On the i860 XR micropocessor, the 128-bit **pfld.q** is not available. If used it causes an instruction trap.

Instruction	Execution Unit	Pipelined? Delayed? Supervisor? i860™ XP Only?	Sets CC?	Faults	Performance Notes	Programming Restrictions
flush	E					h
fmlow.dd	M				4	
fmul.p	M			SE,RE	4	
form	G				8	
frcp.p	M			SE,RE		
frsqr.p	M			SE,RE		
fst.y	E			DAT	5	f
fsub.p	A			SE,RE		
ftrunc.p	A			SE,RE		
fxfr	G				6,8	
fzchkl	G				8	
fzchks	G				8	
Intovr	E			IT		
ixfr	E				2	
ld.c	E					
ld.x	E			DAT	6	
ldint.x	E	SU,XP		DAT		
ldio.x	E	SU,XP		DAT		
lock	E					
or	E		CC			
orh	E		CC			
pfadd.p	A	P		SE,RE*		
pfaddp	G	P		*	8	e
pfaddz	G	P		*	8	e
pfam.p	A&M	P		SE,RE*	7	d
pfamov.r	A	P		SE,RE*		
pfeq.p	A	P	CC	SE*	1	
pfgt.p	A	P	CC	SE*	1	
pfiadd.w	G	P		*	8	e
pfisub.w	G	P		*	8	e
pfix.p	A	P		SE,RE*		
pfld.y	E	P,(XP)**		DAT*	2,9	f
pfmam.p	A&M	P		SE,RE*	7	d
pfmsm.p	A&M	P		SE,RE*	7	d
pfmul.p	M	P		SE,RE*	4	c
pfmul3.dd	M	P		SE,RE*	4	c
pform	G	P		*	8	e
pfsm.p	A&M	P		SE,RE*	7	d
pfsub.p	A	P		SE,RE*		
pftrunc.p	A	P		SE,RE*		
pfzchkl	G	P		*	8	

NOTES:
* On the i860 XP microprocessor, the pipelined instructions can generate ITR with PI.
** On the i860 XR micropocessor, the 128-bit **pfld.q** is not available. If used it causes an instruction trap.

Instruction	Execution Unit	Pipelined? Delayed? Supervisor? i860™ XP Only?	Sets CC?	Faults	Performance Notes	Programming Restrictions
pfzchks	G	P		*	8	
pst.d	E			DAT	5	f
scyc.x	E	SU,XP		DAT		
shl	E					
shr	E					
shra	E					
shrd	E					
st.c	E					
st.x	E			DAT		
stio.x	E	SU,XP		DAT		
subs	E		CC		1	
subu	E		CC		1	
trap	E			IT		
unlock	E					
xor	E		CC			
xorh	E		CC			

NOTES:
*On the i860 XP microprocessor, the pipelined instructions can generate ITR with PI.
On the i860 XR micropocessor, the 128-bit **pfld.q is not available. If used it causes an instruction trap.

10.5 Software Compatibility

10.5.1 REQUIRED CHANGES

To port existing systems software from the i860 XR microprocessor to the i860 XP microprocessor, the following changes may be required. Applications software does not require changes.

1. Data cache flush. All four ways of the data cache must be flushed on the i860 XP microprocessor. The cache flush routine can be modified to check processor type in **epsr** or the DCS field of **dirbase** and flush the appropriate number of ways.
2. Parity and bus error traps. If the i860 XP system signals these errors, the trap handler must be extended to handle them. Software must avoid testing the BEF and PEF bits unless executing on the i860 XP microprocessor.
3. LOCK# deactivation. On the i860 XP microprocessor, traps do not automatically deactivate the LOCK# signal, so the trap handler must do a data access to deactivate LOCK#. Trap handlers that already access data soon after invocation do not require this modification.
4. Load pipe precision. The precision of the last stage of the load pipeline is specified by the LRP bit on the i860 XR microprocessor but by the LRP0 and LRP1 bits on the i860 XP microprocessor. The procedure that restores the load pipe must check the processor type, use the appropriate bits, and restore the correct precision. Pipe restoration code for the i860 XR microprocessor will work correctly on the i860 XP microprocessor if **pfld.q** is not used.
5. Pre-accessed trap handler pages. Page-directory and page-table entries for the instruction pages of the trap handler and for the first data page accessed by the trap handler must always have A = 1. Software modified to allocate page tables this way works on both i860 XR and i860 XP microprocessors.
6. Page directory entry bit 7 must be zero. This is the bit that selects four Mbyte or four Kbyte page size. On the i860 XR microprocessor, it is *reserved* and should be set to zero. It must be set to zero for four Kbyte pages to work on the i860 XP microprocessor.

10.5.2 PERFORMANCE OPTIMIZATIONS

Software developers may wish to make the following performance enhancements in systems software for the i860 XP microprocessor. Systems software that must execute on both i860 XP and i860 XR systems can contain code both with and without the optimizations. By testing the processor type, the appropriate instruction path can be determined.

1. Data cache flush. On the i860 XP microprocessor, a complete flushing of the data cache is not needed when changing context or marking a page not present.
2. The **epsr** bits AI, DI, PI, and PT can be used on the i860 XP microprocessor to make trap handlers more efficient.
3. Four-Mbyte pages can be allocated to frame buffers and the operating-system kernel, thereby reducing the cost of TLB misses.

10.5.3 NEW FEATURES

Software that uses the new features available only on the i860 XP microprocessor will not be compatible with the i860 XR microprocessor unless alternate instruction paths are provided.

Systems software features:

1. New instructions **ldio, stio, ldint,** and **scyc.**
2. Four-Mbyte pages.
3. Privileged Registers **p0, p1, p2,** and **p3.**
4. Concurrency control unit.
5. 128-bit load instruction **pfld.q.**
6. Support for virtual address aliases.

Applications software features:

1. Concurrency control unit.
2. 128-bit load instruction **pfld.q**. The i860 XR microprocessor traps on **pfld.q**; therefore, software has the opportunity to emulate a **pfld.q** with two **pfld.d** instructions. However, this strategy does not yield optimal performance on the i860 XR microprocessor.

10.5.4 NOTES

On the i860 XP microprocessor, pages with WT = 1 are cached with the write-through policy; whereas, on the i860 XR microprocessor, they are not cached at all. Because this change in the function of WT was anticipated in the i860 XR microprocessor documentation, no incompatibility should arise.

11.0 REVISION HISTORY

DATA SHEET REVISION REVIEW

The following list represents the major differences between version 002 and version 001 of the i860 XP Microprocessor Data Sheet.

Section 2.2.4	**AI** bit has been changed to **TAI** in Figure 2.5. The explanation for **PI** bit has been expanded.
Section 4.2.33	**PCHK#** signal description has been expanded.
Section 4.2.35	Output buffer configuration has been added in **PEN#** signal description.
Section 4.2.37	RESET description has been expanded.
Section 5.1.3	Table 5.2 has been corrected. The explanation of write/read and read/write pipelining has been revised.
Section 5.2.2.4-5	The explanation of late back-off mode has been expanded.
Section 5.2.4	Figure 5.27 has been corrected.
Section 5.3.4	The explanation of EWBE# timing has been corrected.
Section 5.5	RESET initialization description has been expanded.
Section 9.2	D.C. Characteristics are corrected.
Section 9.3	A.C. Characteristics are replaced with nominal timings based on C_L = 0 pF. Figure 9.3 and Figure 9.4 have been replaced with nominal A.C. timings based on C_L = 0 pF. Figure 9.5 has been corrected for normal and high-current output buffers.
Section 9.4	Component buffer model has been added.
Section 10.4	Programming restriction on **flush** instruction has been added.

S

i860™ XR 64-BIT MICROPROCESSOR

- **Parallel Architecture that Supports Up to Three Operations per Clock**
  - **One Integer or Control Instruction per Clock**
  - **Up to Two Floating-Point Results per Clock**
- **High Performance Design**
  - **25/33.3/40 MHz Clock Rates**
  - **80 Peak Single Precision MFLOPs**
  - **60 Peak Double Precision MFLOPs**
  - **64-Bit External Data Bus**
  - **64-Bit Internal Instruction Cache Bus**
  - **128-Bit Internal Data Cache Bus**
- **High Level of Integration on One Chip**
  - **32-Bit Integer and Control Unit**
  - **32/64-Bit Pipelined Floating-Point Adder and Multiplier Units**
  - **64-Bit 3-D Graphics Unit**
  - **Paging Unit with Translation Lookaside Buffer**
  - **4 Kbyte Instruction Cache**
  - **8 Kbyte Data Cache**
- **Compatible with Industry Standards**
  - **ANSI/IEEE Standard 754-1985 for Binary Floating-Point Arithmetic**
  - **Intel386™/486™ Microprocessor Data Formats and Page Table Entries**
  - **JEDEC 168-pin Ceramic Pin Grid Array Package (see *Packaging Outlines and Dimensions*, order #231369)**
- **Easy to Use**
  - **On-Chip Debug Register**
  - **Assembler, Linker, Simulator, Debugger, C and FORTRAN Compilers, FORTRAN Vectorizer, Scalar and Vector Math Libraries for both OS/2* and UNIX* Environments**

The Intel i860™ XR Microprocessor (order codes A80860XR-25, A80860XR-33 and A80860XR-40) delivers supercomputing performance in a single VLSI component. The 64-bit design of the i860 XR microprocessor balances integer, floating point, and graphics performance for applications such as engineering workstations, scientific computing, 3-D graphics workstations, and multiuser systems. Its parallel architecture achieves high throughput with RISC design techniques, pipelined processing units, wide data paths, large on-chip caches, million-transistor design, and fast one-micron CHMOS IV silicon technology.

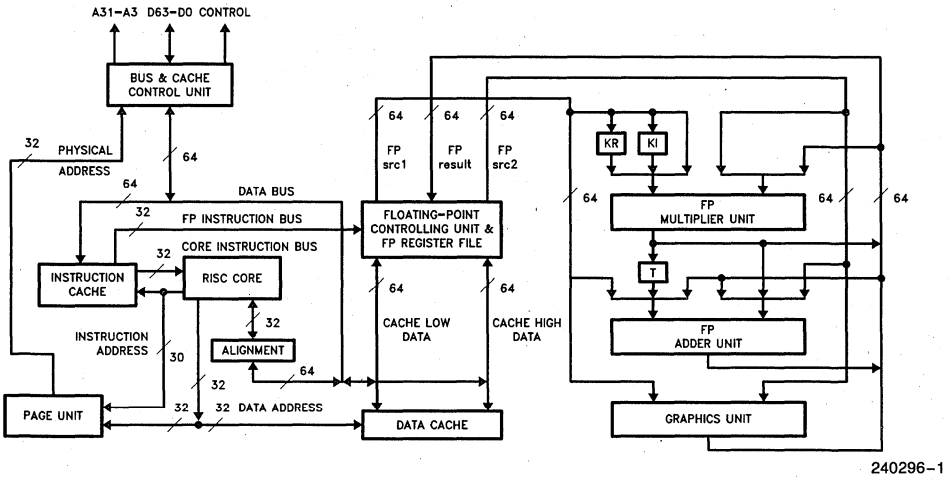


Figure 0.1. Block Diagram

Intel, intel, Intel386™, Intel486™, i860 XR, Multibus II and Parallel System Bus are trademarks of Intel Corporation.
*UNIX is a registered trademark of UNIX System Laboratories, Inc. OS/2 is a trademark of International Business Machines Corporation.

August 1991
Order Number: 240296-005

i860™ XR 64-Bit Microprocessor

CONTENTS

CONTENTS

CONTENTS

CONTENTS

FIGURES

CONTENTS

PAGE

FIGURES (Continued)

CONTENTS

PAGE

TABLES

1.0 FUNCTIONAL DESCRIPTION

As shown by the block diagram on the front page, the i860 XR microprocessor consists of 9 units:

1. Core Execution Unit
2. Floating-Point Control Unit
3. Floating-Point Adder Unit
4. Floating-Point Multiplier Unit
5. Graphics Unit
6. Paging Unit
7. Instruction Cache
8. Data Cache
9. Bus and Cache Control Unit

The core execution unit controls overall operation of the i860 XR microprocessor. The core unit executes load, store, integer, bit, and control-transfer operations, and fetches instructions for the floating-point unit as well. A set of 32 x 32-bit general-purpose registers are provided for the manipulation of integer data. Load and store instructions move 8-, 16-, and 32-bit data to and from these registers. Its full set of integer, logical, and control-transfer instructions give the core unit the ability to execute complete systems software and applications programs. A trap mechanism provides rapid response to exceptions and external interrupts. Debugging is supported by the ability to trap on data or instruction reference.

The floating-point hardware is connected to a separate set of floating-point registers, which can be accessed as 16 x 64-bit registers, or 32 x 32-bit registers. Special load and store instructions can also access these same registers as 8 x 128-bit registers. All floating-point instructions use these registers as their source and destination operands.

The floating-point control unit controls both the floating-point adder and the floating-point multiplier, issuing instructions, handling all source and result exceptions, and updating status bits in the floating-point status register. The adder and multiplier can operate in parallel, producing up to two results per clock. The floating-point data types, floating-point instructions, and exception handling all support the IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985).

The floating-point adder performs addition, subtraction, comparison, and conversions on 64- and 32-bit floating-point values. An adder instruction executes in three clocks; however, in pipelined mode, a new result is generated every clock.

The floating-point multiplier performs floating-point and integer multiply and floating-point reciprocal operations on 64- and 32-bit floating-point values. A multiplier instruction executes in three to four clocks; however, in pipelined mode, a new result can be generated every clock for single-precision and every other clock for double precision.

The graphics unit has special integer logic that supports three-dimensional drawing in a graphics frame buffer, with color intensity shading and hidden surface elimination via the Z-buffer algorithm. The graphics unit recognizes the pixel as an 8-, 16-, or 32-bit data type. It can compute individual red, blue, and green color intensity values within a pixel; but it does so with parallel operations that take advantage of the 64-bit internal word size and 64-bit external bus. The graphics features of the i860 XR microprocessor assume that the surface of a solid object is drawn with polygon patches whose shapes approximate the original object. The color intensities of the vertices of the polygon and their distances from the viewer are known, but the distances and intensities of the other points must be calculated by interpolation. The graphics instructions of the i860 XR microprocessor directly aid such interpolation.

The paging unit implements protected, paged, virtual memory via a 64-entry, four-way set-associative memory called the TLB (Translation Lookaside Buffer). The paging unit uses the TLB to perform the translation of logical address to physical address, and to check for access violations. The access protection scheme employs two levels of privilege: user and supervisor.

The instruction cache is a two-way set-associative memory of four Kbytes, with 32-byte blocks. It transfers up to 64 bits per clock (320 Mbyte/sec at 40 MHz).

The data cache is a two-way set-associative memory of eight Kbytes, with 32-byte blocks. It transfers up to 128 bits per clock (640 Mbyte/sec at 40 MHz). The i860 XR microprocessor normally uses write-back caching, i.e. memory writes update the cache (if applicable) without necessarily updating memory immediately; however, caching can be inhibited by software where necessary.

The bus and cache control unit performs data and instruction accesses for the core unit. It receives cycle requests and specifications from the core unit, performs the data-cache or instuction-cache miss processing, controls TLB translation, and provides the interface to the external bus. Its pipelined structure supports up to three outstanding bus cycles.

2.0 PROGRAMMING INTERFACE

The programmer-visible aspects of the architecture of the i860 XR microprocessor include data types, registers, instructions, and traps.

2.1 Data Types

The i860 XR microprocessor provides operations for integer and floating-point data. Integer operations are performed on 32-bit operands with some support also for 64-bit operands. Load and store instructions can reference 8-bit, 16-bit, 32-bit, 64-bit, and 128-bit operands. Floating-point operations are performed on IEEE-standard 32- and 64-bit formats. Graphics oriented instructions operate on arrays of 8-, 16-, or 32-bit pixels.

2.1.1 INTEGER

An integer is a 32-bit signed value in standard two's complement form. A 32-bit integer can represent a value in the range $-2{,}147{,}483{,}648$ (-2^{31}) to $2{,}147{,}483{,}647$ $(+2^{31}-1)$. Arithmetic operations on 8- and 16-bit integers can be performed by sign-extending the 8- or 16-bit values to 32 bits, then using the 32-bit operations.

There are also add and subtract instructions that operate on 64-bit long integers.

Load and store instructions may also reference (in addition to the 32- and 64-bit formats previously mentioned) 8- and 16-bit items in memory. When an 8- or 16-bit item is loaded into a register, it is converted to an integer by sign-extending the value to 32 bits. When an 8- or 16-bit item is stored from a register, the corresponding number of low-order bits of the register are used.

2.1.2 ORDINAL

Arithmetic operations are available for 32-bit ordinals. An ordinal is an unsigned integer. An ordinal can represent values in the range 0 to $4{,}294{,}967{,}295$ $(+2^{32}-1)$.

Also, there are add and subtract instructions that operate on 64-bit ordinals.

2.1.3 SINGLE- AND DOUBLE-PRECISION REAL

Figure 2.1 shows the real number formats. A single-precision real (also called "single real") data type is a 32-bit binary floating-point number. Bit 31 is the sign bit; bits 30..23 are the exponent; and bits 22..0 are the fraction. In accordance with ANSI/IEEE standard 754, the value of a single-precision real is defined as follows:

1. If $e = 0$ and $f \neq 0$ or $e = 255$ then generate a floating-point source-exception trap when encountered in a floating-point operation.
2. If $0 < e < 255$, then the value is $(-1)^s \times 1.f \times 2^{e-127}$.
3. If $e = 0$ and $f = 0$, then the value is signed zero.

A double-precision real (also called "double real") data type is a 64-bit binary floating-point number. Bit 63 is the sign bit; bits 62..52 are the exponent; and bits 51..0 are the fraction. In accordance with ANSI/IEEE standard 754, the value of a double-precision real is defined as follows:

1. If $e = 0$ and $f \neq 0$ or $e = 2047$, then generate a floating-point source-exception trap when encountered in a floating-point operation.
2. If $0 < e < 2047$, then the value is $(-1)^s \times 1.f \times 2^{e-1023}$.

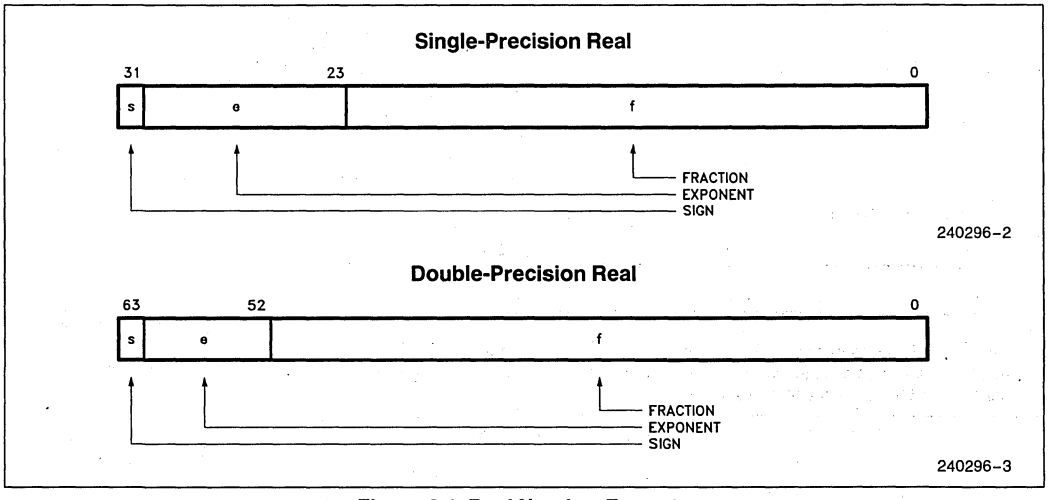


Figure 2.1. Real Number Formats

3. If e = 0 and f = 0, then the value is signed zero.

The special values infinity, NaN ("Not a Number"), indefinite, and denormal generate a trap when encountered. The trap handler implements IEEE-standard results.

A double real value occupies an even/odd pair of floating-point registers. Bits 31..0 are stored in the even-numbered floating-point register; bits 63..32 are stored in the next higher odd-numbered floating-point register.

2.1.4 PIXEL

A pixel may be 8, 16, or 32 bits long depending on color and intensity resolution requirements. Regardless of the pixel size, the i860 XR microprocessor always operates on 64 bits worth of pixels at a time. The pixel data type is used by two kinds of instructions:

- The selective pixel-store instruction that helps implement hidden surface elimination.
- The pixel add instruction that helps implement 3-D color intensity shading.

To perform color intensity shading efficiently in a variety of applications, the i860 XR microprocessor defines three pixel formats according to Table 2.1.

Figure 2.2 illustrates one way of assigning meaning to the fields of pixels. These assignments are for illustration purposes only. The i860 XR microprocessor defines only the field sizes, not the specific use of each field. Other ways of using the fields of pixels are possible.

Table 2.1. Pixel Formats

Pixel Size (in bits)	Bits of Color 1 Intensity	Bits of Color 2 Intensity	Bits of Color 3 Intensity	Bits of Other Attribute (Texture)
8	N (≤ 8) bits of intensity*			8 − N
16	6	6	4	
32	8	8	8	8

The intensity attribute fields may be assigned to colors in any order convenient to the application.

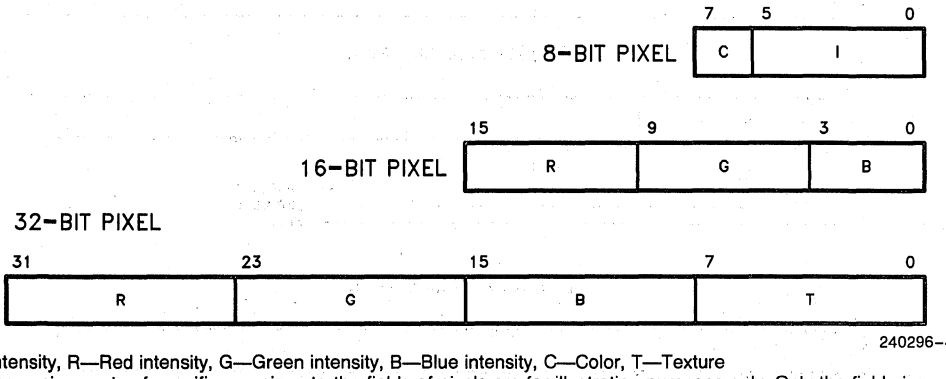
*With 8-bit pixels, up to 8 bits can be used for intensity; the remaining bits can be used for any other attribute, such as color. The intensity bits must be the low-order bits of the pixel.

2.2 Register Set

As Figure 2.3 shows, the i860 XR microprocessor has the following registers:

- An integer register file
- A floating-point register file
- Six control registers (**psr, epsr, db, dirbase, fir,** and **fsr**)
- Four special-purpose registers (KR, KI, T, and MERGE)

The control registers are accessible only by load and store control-register instructions; the integer and floating-point registers are accessed by arithmetic operations and load and store instructions. The special-purpose registers KR, KI, T, and MERGE are used by a few specific instructions.



I—Intensity, R—Red intensity, G—Green intensity, B—Blue intensity, C—Color, T—Texture
These assignments of specific meanings to the fields of pixels are for illustration purposes only. Only the field sizes are defined, not the specific use of each field.

Figure 2.2. Pixel Format Example

2.2.1 INTEGER REGISTER FILE

There are 32 integer registers, each 32 bits wide, referred to as **r0** through **r31**, which are used for address computation and scalar integer computations. Register **r0** always returns zero when read, independently of what is stored in it.

2.2.2 FLOATING-POINT REGISTER FILE

There are 32 floating-point registers, each 32-bits wide, referred to as **f0** through **f31**, which are used for floating-point computations. Registers **f0** and **f1** always return zero when read, independently of what is stored in them. The floating-point registers are also used by a set of graphics operations, primarily for 3D graphics computations.

When accessing 64-bit floating-point or integer values, the i860 XR microprocessor uses an even/odd pair of registers. When accessing 128-bit values, it uses an aligned set of four registers (**f0**, **f4**, **f8**, . . . , **f28**). The instruction must designate the lowest register number of the set of registers containing 64- or 128-bit values. Misaligned register numbers produce undefined results. The register with the lowest number contains the least significant part of the value. For 128-bit values, the register pair with the lower numbers contain the least significant 64 bits while the register pair with the higher numbers contain the most significant 64 bits.

The 128-bit load and store instructions, along with the 128-bit data path between the floating-point registers and the data cache help to sustain the extraordinarily high rate of computation.

2.2.3 PROCESSOR STATUS REGISTER

The processor status register (**psr**) contains miscellaneous state information for the current process. Figure 2.4 shows the format of the **psr**.

- BR (Break Read) and BW (Break Write) enable a data access trap when the operand address matches the address in the **db** register and a read or write (respectively) occurs.
- Various instructions set CC (Condition Code) according to tests they perform. The branch-on-condition-code instructions test its value. The **bla** instruction sets and tests LCC (Loop Condition Code).
- IM (Interrupt Mode) enables external interrupts if set; disables interrupts if clear.
- U (User Mode) is set when the i860 XR microprocessor is executing in user mode; it is clear when the i860 XR microprocessor is executing in supervisor mode. In user mode, writes to some control registers are inhibited. This bit also controls the memory protection mechanism. See section 2.4.4.3 for a description of memory protection in user and supervisor modes.

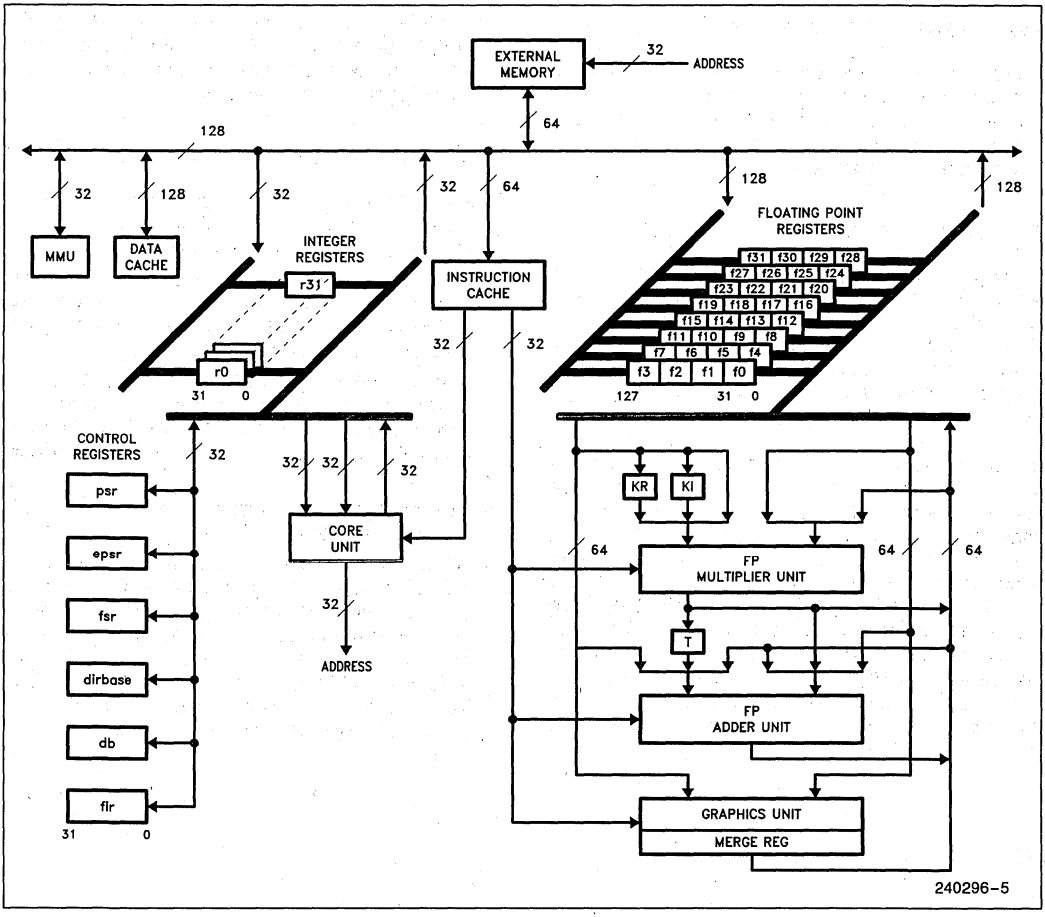
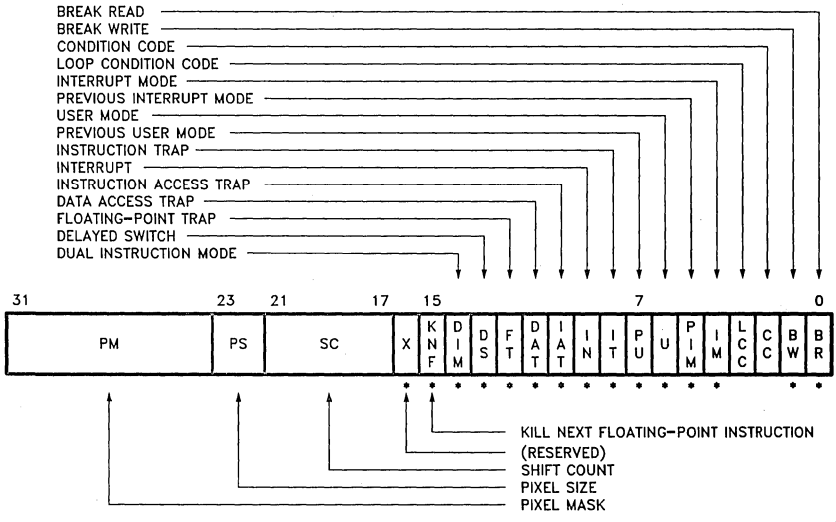
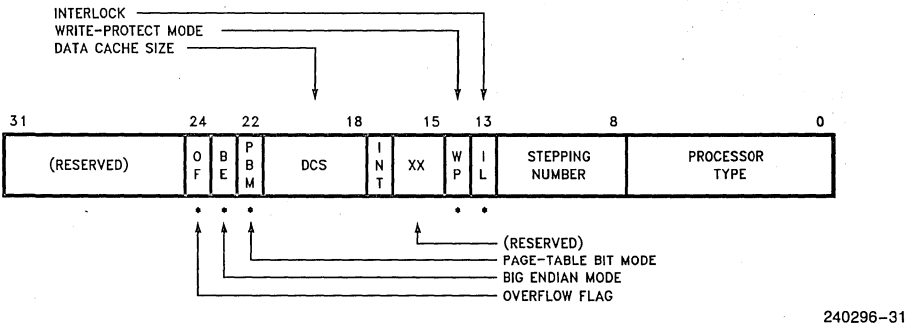


Figure 2.3. Registers and Data Paths



*Can be changed only from supervisor level.

Figure 2.4 Processor Status Register



*Can be changed only from supervisor level.

Figure 2.5 Extended Processor Status Register

- PIM (Previous Interrupt Mode) and PU (Previous User Mode) save the corresponding status bits (IM and U) on a trap, because those status bits are changed when a trap occurs. They are restored into their corresponding status bits when returning from a trap handler with a branch indirect instruction when a trap flag is set in the **psr**.
- FT (Floating-Point Trap), DAT (Data Access Trap), IAT (Instruction Access Trap), IN (Interrupt), and IT (Instruction Trap) are trap flags. They are set when the corresponding trap condition occurs. The trap handler examines these bits to determine which condition or conditions have caused the trap.
- DS (Delayed Switch) is set if a trap occurs during the instruction before dual-instruction mode is entered or exited. If DS is set and DIM (Dual Instruction Mode) is clear, the i860 XR microprocessor switches to dual-instruction mode one instruction after returning from the trap handler. If DS and DIM are both set, the i860 XR microprocessor switches to single-instruction mode one instruction after returning from the trap handler.
- When a trap occurs, the i860 XR microprocessor sets DIM if it is executing in dual-instruction mode; it clears DIM if it is executing in single-instruction mode. If DIM is set after returning from a trap handler, the i860 XR microprocessor resumes execution in dual-instruction mode.

- When KNF (Kill Next Floating-Point Instruction) is set, the next floating-point instruction is suppressed (except that its dual-instruction mode bit is interpreted). A trap handler sets KNF if the trapped floating-point instruction should not be reexecuted.
- SC (Shift Count) stores the shift count used by the last right-shift instruction. It controls the number of shifts executed by the double-shift instruction.
- PS (Pixel Size) and PM (Pixel Mask) are used by the pixel-store instruction and by the graphics instructions. The values of PS control pixel size as defined by Table 2.2. The bits in PM correspond to pixels to be updated by the pixel-store instruction **pst.d**. The low-order bit of PM corresponds to the low-order pixel of the 64-bit source operand of **pst.d**. The number of low-order bits of PM that are actually used is the number of pixels that fit into 64-bits, which depends upon PS. If a bit of PM is set, then **pst.d** stores the corresponding pixel. Refer also to the **pst.d** instruction in section 8.

Table 2.2. Values of PS

Value	Pixel Size in bits	Pixel Size in bytes
00	8	1
01	16	2
10	32	4
11	(undefined)	(undefined)

2.2.4 EXTENDED PROCESSOR STATUS REGISTER

The extended processor status register (**epsr**) contains additional state information for the current process beyond that stored in the **psr**. Figure 2.5 shows the format of the **epsr**.

- The processor type is one for the i860 XR microprocessor.
- The stepping number has a unique value that distinguishes among different revisions of the processor.
- IL (Interlock) is set if a trap occurs after a **lock** instruction but before the load or store following the subsequent **unlock** instruction. IL indicates to the trap handler that a locked sequence has been interrupted. When the trap handler finds IL set, it should scan backwards for the **lock** instruction and restart at that point. The absence of a **lock** instruction within 30–33 instructions of the trap indicates a programming error.
- WP (write protect) controls the semantics of the W bit of page table entries. A clear W bit in either the directory or the page table entry causes writes to be trapped. When WP is clear, writes are trapped in user mode, but not in supervisor mode. When WP is set, writes are trapped in both user and supervisor modes. After the value of the WP bit is changed, the TLB must be invalidated by setting the ITI bit of the **dirbase** register, before any stores are performed.
- INT (Interrupt) is the value of the INT input pin.
- DCS (Data Cache Size) is a read-only field that tells the size of the on-chip data cache. The number of bytes actually available is 2^{12+DCS}; therefore, a value of zero indicates 4 Kbytes, one indicates 8 Kbytes, etc.

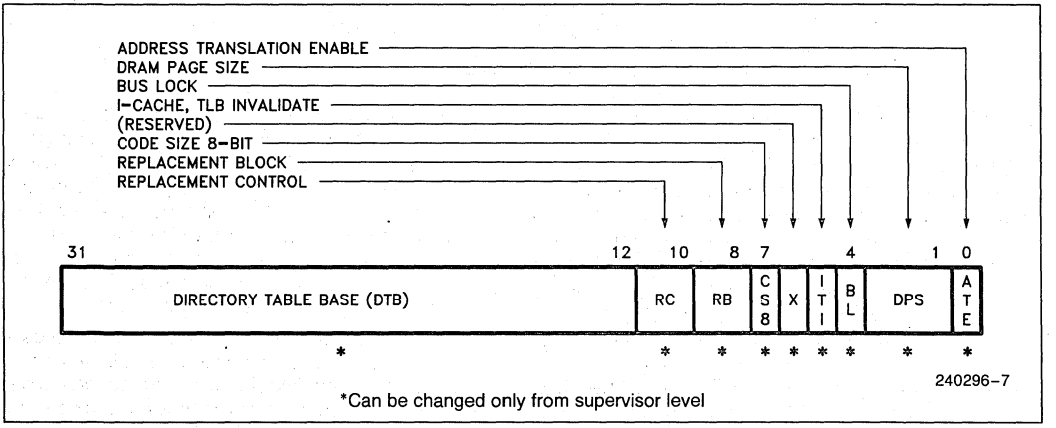


Figure 2.6. Directory Base Register

- PBM (Page-Table Bit Mode) determines which bit of page-table entries is output on the PTB pin. When PBM is clear, the PTB signal reflects bit CD of the page-table entry used for the current cycle. When PBM is set, the PTB signal reflects bit WT of the page-table entry used for the current cycle.
- BE (Big Endian) controls the ordering of bytes within a data item in memory. Normally (i.e. when BE is clear) the i860 XR microprocessor operates in little endian mode, in which the addressed byte is the low-order byte. When BE is set (big endian mode), the low-order three bits of all load and store addresses are complemented, then masked to the appropriate boundary for alignment. This causes the addressed byte to be the most significant byte. Section 2.3 discusses little and big endian addressing.
- OF (Overflow Flag) is set by **adds**, **addu**, **subs**, and **subu** when integer overflow occurs. For **adds** and **subs**, OF is set if the carry from bit 31 is different than the carry from bit 30. For **addu**, OF is set if there is a carry from bit 31. For **subu**, OF is set if there is no carry from bit 31. Under all other conditions, it is cleared by these instructions. OF controls the function of the **intovr** instruction. OF cannot be written in user mode using ST.C.

2.2.5 DATA BREAKPOINT REGISTER

The data breakpoint register (**db**) is used to generate a trap when the i860 XR microprocessor makes a data-operand access to the address stored in this register. The trap is enabled by BR and BW in **psr**. The **db** register can only be changed from supervisor level. When comparing, a number of low order bits of the address are ignored, depending on the size of the operand. For example, a 16-bit access ignores the low-order bit of the address when comparing to **db**; a 32-bit access ignores the low-order two bits. This ensures that any access that overlaps the address contained in the register will generate a trap. The DAT occurs before the data is accessed and prevents the load or store from completing.

2.2.6 DIRECTORY BASE REGISTER

The directory base register **dirbase** (shown in Figure 2.6) controls address translation, caching, and bus options. The **dirbase** register can only be changed from supervisor level. The BL bit is changed from user level with the **lock** and **unlock** instructions.

- ATE (Address Translation Enable), when set, enables the virtual-address translation algorithm. The data cache must be flushed before changing the ATE bit.
- DPS (DRAM Page Size) controls how many bits to ignore when comparing the current bus-cycle address with the previous bus-cycle address to generate the NENE# signal. This feature allows for higher speeds when using static column or page-mode DRAMs and consecutive reads and writes access the row. The comparison ignores the low-order 12 + DPS bits. A value of zero is appropriate for one bank of 256K × *n* RAMs, 1 for 1M × *n* RAMS, etc. For interleaved memory, increase DPS by one for each power of interleaving—add one for 2-way, and two for 4-way, etc.
- When BL (Bus Lock) is set, external bus accesses are locked. The LOCK# signal is asserted the next bus cycle whose internal bus request is generated after BL is set. It remains set on every subsequent bus cycle as long as BL remains set. The LOCK# signal is deasserted on the next load or store instruction after BL is cleared. Traps immediately clear BL. The **lock** and **unlock** instructions control the BL bit. The result of modifying BL with the **st.c** instruction is not defined.
- ITI (I-Cache, TLB Invalidate), when set in the value that is loaded into **dirbase**, causes all entries in the instruction cache and address-translation cache (TLB) to be invalidated. The ITI bit does not remain set in **dirbase**. ITI always appears as zero when reading **dirbase**. Section 2.5 discusses flushing the data cache before invalidating the TLB.
- When CS8 (Code Size 8-Bit) is set, instruction cache misses are processed as 8-bit bus cycles. When this bit is clear, instruction cache misses are processed as 64-bit bus cycles. This bit can not be set by software; hardware sets this bit at initialization time. It can be cleared by software (one time only) to allow the system to execute out of 64-bit memory after bootstrapping from 8-bit EPROM. A nondelayed branch to code in 64-bit memory should directly follow the **st.c** (store control register) instruction that clears CS8, in order to make the transition from 8-bit to 64-bit memory occur at the correct time. The branch instruction must be aligned on a 64-bit boundary.
- RB (Replacement Block) identifies the cache block to be replaced by cache replacement algorithms. The high-order bit of RB is ignored by the instruction and data caches. RB conditions the cache flush instruction **flush**, which is discussed in Section 8. Table 2.3 explains the values of RB.
- RC (Replacement Control) controls cache replacement algorithms. Table 2.4 explains the significance of the values of RC.
- DTB (Directory Table Base) contains the high-order 20 bits of the physical address of the page directory when address translation is enabled (i.e. ATE = 1). The low-order 12 bits of the address are zeros.

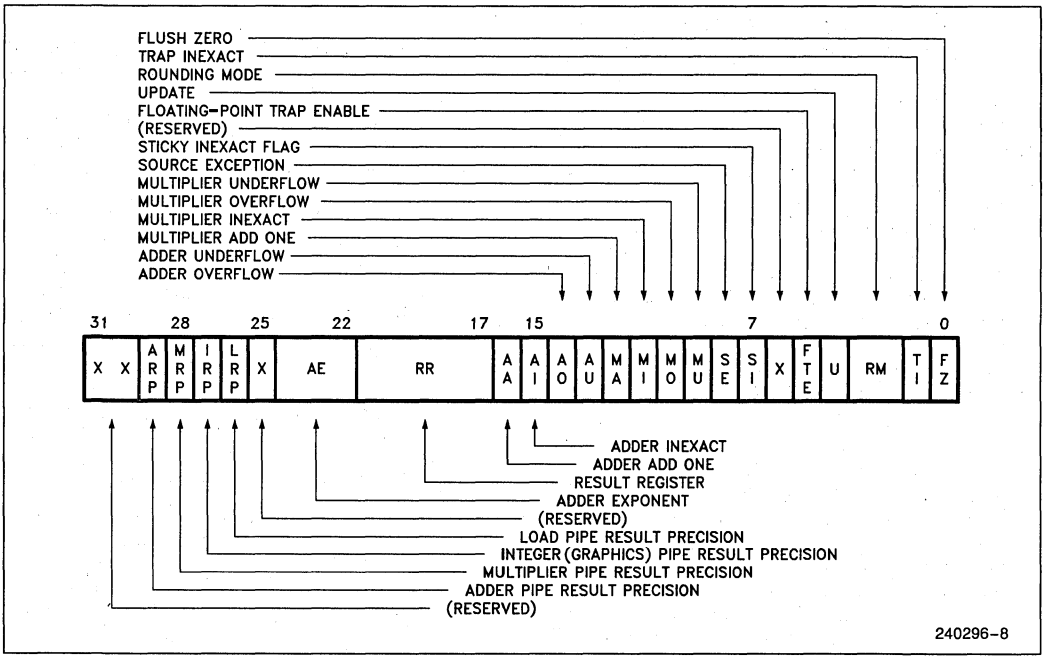


Figure 2.7. Floating-Point Status Register

Table 2.3. Values of RB

Value	Replace TLB Block	Replace Instruction and Data Cache Block
0 0	0	0
0 1	1	1
1 0	2	0
1 1	3	1

Table 2.4. Values of RC

Value	Meaning
00	Selects the normal replacement algorithm where any block in the set may be replaced on cache misses in all caches.
01	Instruction, data, and TLB cache misses replace the block selected by RB. The instruction and data caches ignore the high-order bit of RB. This mode is used for instruction cache and TLB testing.
10	Data cache misses replace the block selected by the low-order bit of RB. Instruction and TLB caches use random replacement.
11	Disables data cache replacement. Instruction and TLB caches use random replacement.

2.2.7 FAULT INSTRUCTION REGISTER

When a trap occurs, this register contains the address of the trapping instruction (not necessarily the instruction that created the conditions that required the trap). The **fir** is a read-only register. In single-instruction mode, using a **ld.c** instruction to read the **fir** anytime except the first time after a trap saves in *idest* the address of the **ld.c** instruction; in dual-instruction mode, the address of its floating-point companion (address of the **ld.c** − 4) is saved.

2.2.8 FLOATING-POINT STATUS REGISTER

The floating-point status register (**fsr**) contains the floating-point trap and rounding-mode status for the current process. Figure 2.7 shows its format. The **fsr** is writable in user level.

- If FZ (Flush Zero) is clear and underflow occurs, a result-exception trap is generated. When FZ is set and underflow occurs, the result is set to zero, and no trap due to underflow occurs.
- If TI (Trap Inexact) is clear, inexact results do not cause a trap. If TI is set, inexact results cause a trap. The sticky inexact flag (SI) is set whenever an inexact result is produced, regardless of the setting of TI.
- RM (Rounding Mode) specifies one of the four rounding modes defined by the IEEE standard. Given a true result *b* that cannot be represented

Table 2.5. Values of RM

Value	Rounding Mode	Rounding Action
00	Round to nearest or even	Closer to *b* of *a* or *c*; if equally close, select even number (the one whose least significant bit is zero).
01	Round down (toward $-\infty$)	*a*
10	Round up (toward $+\infty$	*c*
11	Chop (toward zero)	Smaller in magnitude of *a* or *c*.

by the target data type, the i860 XR microprocessor determines the two representable numbers *a* and *c* that most closely bracket *b* in value ($a < b < c$). The i860 XR microprocessor then rounds (changes) *b* to *a* or *c* according to the mode selected by RM as defined in Table 2.5. Rounding introduces an error in the result that is less than one least-significant bit.

- The U-bit (Update Bit), if set in the value that is loaded into **fsr** by a **st.c** instruction, enables updating of the result-status bits (AE, AA, AI, AO, AU, MA, MI, MO, and MU) in the first-stage of the floating-point adder and multiplier pipelines. If this bit is clear, the result-status bits are unaffected by a **st.c** instruction; **st.c** ignores the corresponding bits in the value that is being loaded. A **st.c** always updates **fsr** bits 21..17 and 8..0 directly. The U-bit does not remain set; it always appears as zero when read.
- The FTE (Floating-Point Trap Enable) bit, if clear, disables all floating-point traps (invalid input operand, overflow, underflow, and inexact result).
- SI (Sticky Inexact) is set when the last stage result of either the multiplier or adder is inexact (i.e. when either AI or MI is set). SI is "sticky" in the sense that it remains set until reset by software. AI and MI, on the other hand, can by changed by the subsequent floating-point instruction.
- SE (Source Exception) is set when one of the source operands of a floating-point operation is invalid; it is cleared when all the input operands are valid. Invalid input operands include denormals, infinities, and all NaNs (both quiet and signaling).
- When read from the **fsr,** the result-status bits MA, MI, MO, and MU (Multiplier Add-One, Inexact, Overflow, and Underflow, respectively) describe the last stage result of the multiplier.

  When read from the **fsr**, the result-status bits AA, AI, AO, AU, and AE (Adder Add-One, Inexact, Overflow, Underflow, and Exponent, respectively) describe the last stage result of the adder. The high-order three bits of the 11-bit exponent of the adder result are stored in the AE field.

  The Adder Add One and Multiplier Add One bits indicate that the absolute value of the result fraction grew by one least-significant bit due to rounding. AA and MA are not influenced by the sign of the result.

  After a floating-point operation in a given unit (adder or multiplier), the result-status bits of that unit are undefined until the point at which result exceptions are reported.

  When written to the **fsr** with the U-bit set, the result-status bits are placed into the first stage of the adder and multiplier pipelines. When the processor executes pipelined operations, it propagates the result-status bits of a particular unit (multiplier or adder) one stage for each pipelined floating-point operation for that unit. When they reach the last stage, they replace the normal result-status bits in the **fsr.** When the U-bit is not set, result-status bits in the word being written to the **fsr** are ignored.

  In a floating-point dual-operation instruction (e.g. add-and-multiply or subtract-and-multiply), both the multiplier and the adder may set exception bits. The result-status bits for a particular unit remain set until the next operation that uses that unit.
- RR (Result Register) specifies which floating-point register (**f0–f31**) was the destination register when a result-exception trap occurs due to a scalar operation.
- LRP (Load Pipe Result Precision), IRP (Integer (Graphics) Pipe Result Precision), MRP (Multiplier Pipe Result Precision), and ARP (Adder Pipe Result Precision) aid in restoring pipeline state after a trap or process switch. Each defines the precision of the last stage result in the corresponding pipeline. One of these bits is set when the result in the last stage of the corresponding pipeline is double precision; it is cleared if the result is single precision. These bits cannot be changed by software.

2.2.9 KR, KI, T, AND MERGE REGISTERS

The KR, KI, and T registers are special-purpose registers used by the dual-operation floating-point instructions **pfam, pfmam, pfsm,** and **pfmsm,**

which initiate both an adder (A-unit) operation and a multiplier (M-unit) operation. The KR, KI, and T registers can store values from one dual-operation instruction and supply them as inputs to subsequent dual-operation instructions. (Refer to Figure 2.14.)

The MERGE register is used only by the graphics instructions. The purpose of the MERGE register is to accumulate (or merge) the results of multiple-addition operations that use as operands the color-intensity values from pixels or distance values from a Z-buffer. The accumulated results can then be stored in one 64-bit operation.

Two multiple-addition instructions and an OR instruction use the MERGE register. The addition instructions are designed to add interpolation values to each color-intensity field in an array of pixels or to each distance value in a Z-buffer.

Refer to the instruction descriptions in section 8 for more information about these registers.

2.3 Addressing

Memory is addressed in byte units with a paged virtual-address space of 2^{32} bytes. Data and instructions can be located anywhere in this address space. Address arithmetic is performed using 32-bit input values and produces 32-bit results. The low-order 32 bits of the result are used in case of overflow.

Normally, multibyte data values are stored in memory in little endian format, i.e., with the least significant byte at the lowest memory address. As an option, the ordering can be dynamically selected by software in supervisor mode. The i860 XR microprocessor also offers big endian mode, in which the most significant byte of a data item is at the lowest address. Figure 2.8 shows the difference between the two storage modes. Big endian and little endian data areas should not be mixed within a 64-bit data word. Illustrations of data structures in this data sheet show data stored in little endian mode, i.e., the low-order byte is at the lowest memory address.

Code accesses are always done with little endian addressing. This implies that code will appear differently than documented here when accessed as big endian data. Intel recommends that disassemblers running in a big endian system, convert instructions which have been read as data back to little endian form and present them in the format documented here.

Page directories and page tables are also accessed in little endian mode, regardless of the value of the BE bit.

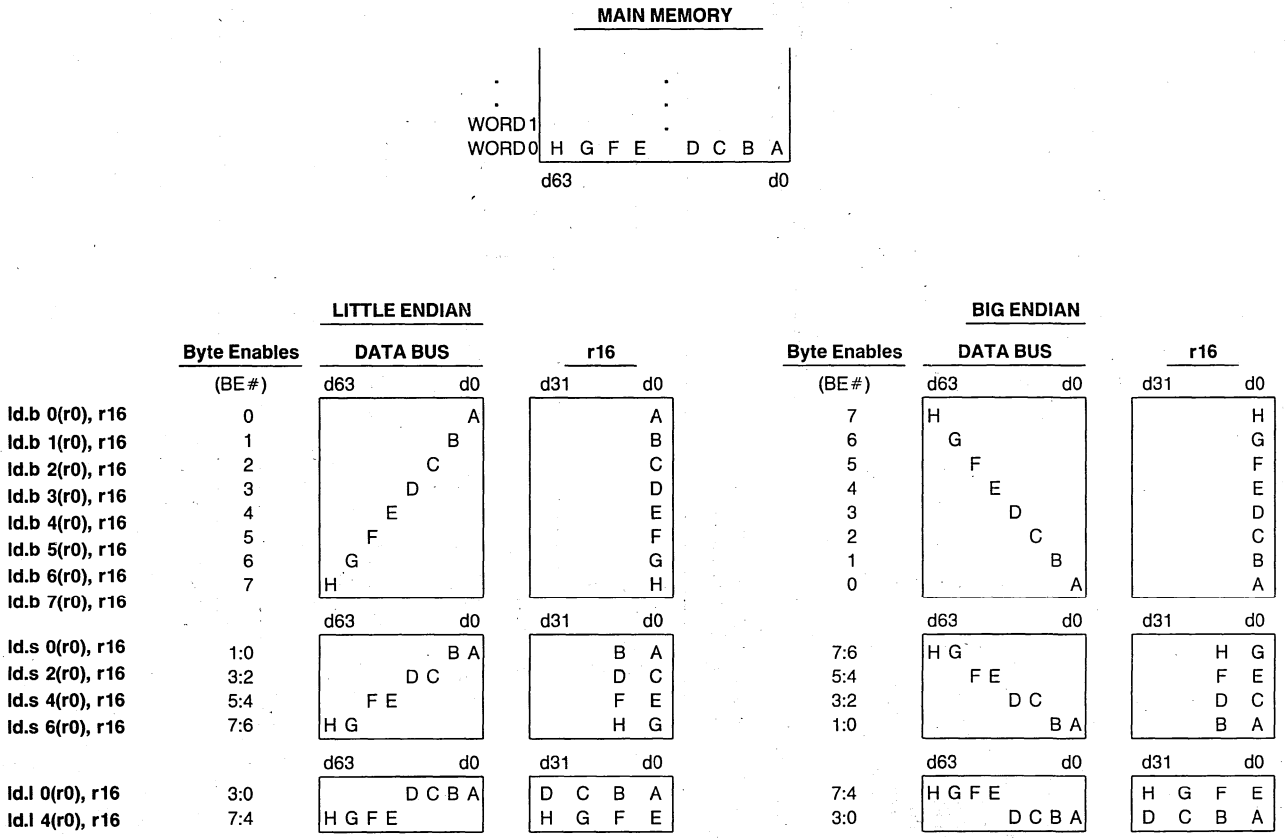
Alignment requirements are as follows (any violation results in a data-access trap):

- 128-bit values are aligned on 16-byte boundaries when referenced in memory (i.e. the four least significant address bits must be zero).
- 64-bit values are aligned on 8-byte boundaries when referenced in memory (i.e. the three least significant address bits must be zero).
- 32-bit values are aligned on 4-byte boundaries when referenced in memory (i.e. the two least significant address bits must be zero).
- 16-bit values are aligned on 2-byte boundaries when referenced in memory (i.e. the least significant address bit must be zero).

2.4 Virtual Addressing

When address translation is enabled, the i860 XR microprocessor maps instruction and data virtual addresses into physical addresses before referencing memory. This address transformation is compatible with that of the Intel386™ microprocessor and implements the basic features needed for page-oriented virtual-memory systems and page-level protection.

The address translation is optional. Address translation is in effect only when the ATE bit of **dirbase** is set. This bit is typically set by the operating system during software initialization. The ATE bit must be set if the operating system is to implement page-oriented protection or page-oriented virtual memory.



NOTE:
64- and 128-bit big endian accesses are treated the same as little endian accesses.

Figure 2.8 Little and Big Endian Accesses

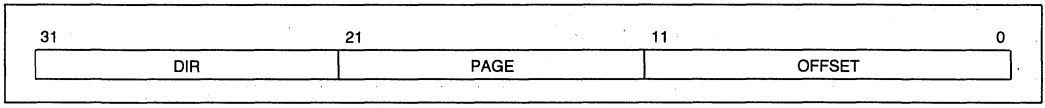


Figure 2.9. Format of a Virtual Address

Address translation is disabled when the processor is reset. It is enabled when a store to **dirbase** sets the ATE bit. It is disabled again when a store clears the ATE bit.

2.4.1 PAGE FRAME

A **page frame** is a 4-Kbyte unit of contiguous addresses of physical main memory. Page frames begin on 4-Kbyte boundaries and are fixed in size. A **page** is the collection of data that occupies a page frame when that data is present in main memory. The data may also occupy some location in secondary storage when there is not sufficient space in main memory.

2.4.2 VIRTUAL ADDRESS

A virtual address refers indirectly to a physical address by specifying a page table, a page within that table, and an offset within that page. Figure 2.9 shows the format of a virtual address.

Figure 2.10 shows how the i860 XR microprocessor converts the DIR, PAGE, and OFFSET fields of a virtual address into the physical address by consulting two levels of page tables. The addressing mechanism uses the DIR field as an index into a page directory, uses the PAGE field as an index into the page table determined by the page directory, and uses the OFFSET field to address a byte within the page determined by the page table.

2.4.3 PAGE TABLES

A page table is simply an array of 32-bit page specifiers. A page table is itself a page, and therefore contains 4 Kbytes of memory or at most 1K 32-bit entries.

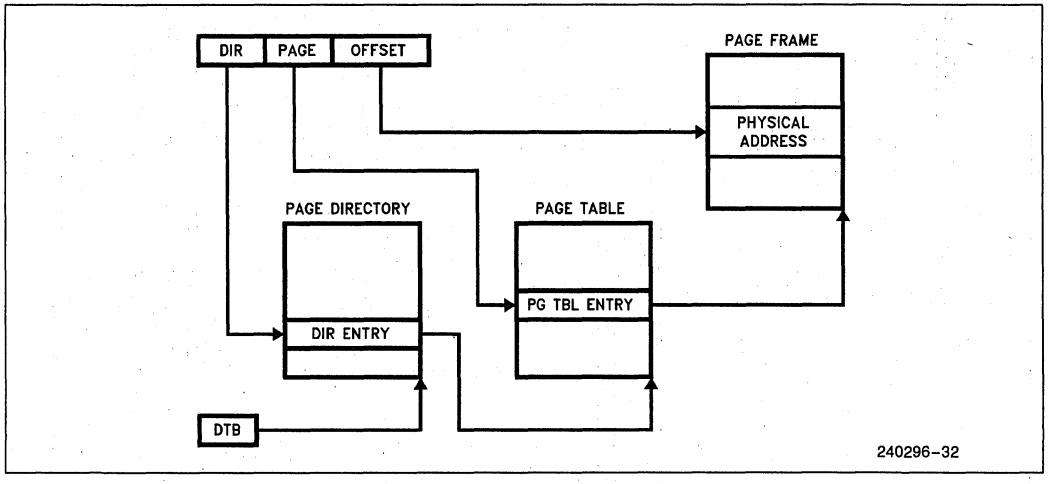


Figure 2.10. Address Translation

Two levels of tables are used to address a page of memory. At the higher level is a page directory. The page directory addresses up to 1K page tables of the second level. A page table of the second level addresses up to 1K pages. All the tables addressed by one page directory, therefore, can address 1M pages (2^{20}). Because each page contains 4 Kbytes (2^{12} bytes), the tables of one page directory can span the entire physical address space of the i860 XR microprocessor ($2^{20} \times 2^{12} = 2^{32}$).

The physical address of the current page directory is stored in DTB field of the **dirbase** register. Memory management software has the option of using one page directory for all processes, one page directory for each process, or some combination of the two.

2.4.4 PAGE-TABLE ENTRIES

Page-table entries (PTEs) in either level of page tables have the same format. Figure 2.11 illustrates this format.

2.4.4.1 Page Frame Address

The page frame address specifies the physical starting address of a page. Because pages are located on 4K boundaries, the low-order 12 bits are always zero. In a page directory, the page frame address is the address of a page table. In a second-level page table, the page frame address is the address of the page frame that contains the desired memory operand.

2.4.4.2 Present Bit

The P (present) bit indicates whether a page table entry can be used in address translation. P = 1 indicates that the entry can be used. When P = 0 in either level of page tables, the entry is not valid for address translation, and the rest of the entry is available for software use; none of the other bits in the entry is tested by the hardware. If P = 0 in either level of page tables when an attempt is made to use a page-table entry for address translation, the processor signals either a data-access fault or an instruction-access fault. In software systems that support paged virtual memory, the trap handler can bring the required page into physical memory.

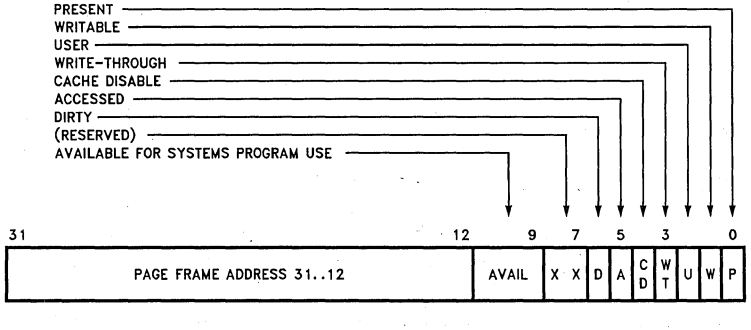
Note that there is no P bit for the page directory itself. The page directory may be not-present while the associated process is suspended, but the operating system must ensure that the page directory indicated by the **dirbase** image associated with the process is present in physical memory before the process is dispatched.

2.4.4.3 Writable and User Bits

The W (writable) and U (user) bits are used for page-level protection, which the i860 XR microprocessor performs at the same time as address translation. The concept of privilege for pages is implemented by assigning each page to one of two levels:

1. Supervisor level (U = 0)—for the operating system and other systems software and related data.
2. User level (U = 1)—for applications procedures and data.

The U bit of the **psr** indicates whether the i860 XR microprocessor is executing at user or supervisor level. The i860 XR microprocessor maintains the U bit of **psr** as follows:



NOTE:
X indicates Intel reserved. Do not use.

Figure 2.11. Format of a Page Table Entry

- The i860 XR microprocessor clears the **psr** U bit to indicate supervisor level when a trap occurs (including when the **trap** instruction causes the trap). The prior value of U is copied into PU.
- The i860 XR microprocessor copies the **psr** PU bit into the U bit when an indirect branch is executed and one of the trap bits is set. If PU was one, the i860 XR microprocessor enters user level.

With the U bit of **psr** and the W and U bits of the page table entries, the i860 XR microprocessor implements the following protection rules:

- When at user level, a read or write of a supervisor-level page causes a trap.
- When at user level, a write to a page whose W bit is clear causes a trap.
- When at user level, **st.c** to certain control registers is ignored.

When the i860 XR microprocessor is executing at supervisor level, all pages are addressable, but, when it is executing at user level, only pages that belong to the user-level are addressable.

When the i860 XR microprocessor is executing at supervisor level, all pages are readable. Whether a page is writable depends upon the write-protection mode controlled by WP of **epsr**:

WP = 0	All pages are writable.
WP = 1	A write to a page whose W bit is clear causes a trap.

When the i860 XR microprocessor is executing at user level, only pages that belong to user level and are marked writable are actually writable; pages that belong to supervisor level are neither readable nor writable from user level.

2.4.4.4 Write-Through Bit

The i860 XR microprocessor does not implement a write-through caching policy for the on-chip data cache; however, the WT (write-through) bit in the second-level page-table entry does determine internal caching policy. If WT is set in a PTE, on-chip caching of data from the corresponding page is inhibited. The i860 XR CPU may place pages having WT = 1 into the instruction cache. Future implementations of the i860 XR architecture may adhere to a write-through data caching policy. Therefore, they may cache pages having the WT bit of the PTE set. If WT is clear, the normal write-back policy is applied to data from the page in the on-chip caches. The WT bit of page directory entries is not referenced by the processor, but is **reserved**.

The WT bit is independent of the CD bit; therefore, data may be placed in a second-level coherent cache, but kept out of the on-chip caches.

2.4.4.5 Cache Disable Bit

If the CD (cache disable) bit in the second-level page-table entry is set, data from the associated page is not placed in instruction or data caches. Clearing CD permits the cache hardware to place data from the associated page into caches. The CD bit of page directory entries is not referenced by the processor, but is **reserved.**

To control external caches, the i860 XR microprocessor outputs on its PTB pin either the CD or WT bit. The PBM bit of **epsr** determines which bit is output.

2.4.4.6 Accessed and Dirty Bits

The A (accessed) and D (dirty) bits provide data about page usage in both levels of the page tables.

The i860 XR microprocessor sets the corresponding accessed bits in both levels of page tables before a read or write operation to a page. The processor tests the dirty bit in the second-level page table before a write to an address covered by that page table entry, and, under certain conditions, causes traps. The trap handler then has the opportunity to maintain appropriate values in the dirty bits. The dirty bit in directory entries is not tested by the i860 XR microprocessor. The precise algorithm for using these bits is specified in Section 2.4.5.

An operating system that supports paged virtual memory can use these bits to determine what pages to eliminate from physical memory when the demand for memory exceeds the physical memory available. The D and A bits in the PTE (page-table entry) are normally initialized to zero by the operating system. The processor sets the A bit when a page is accessed either by a read or write operation. When a data- or instruction-access fault occurs, the trap handler sets the D bit if an allowable write is being performed, then re-executes the instruction.

The operating system is responsible for coordinating its updates to the accessed and dirty bits with updates by the CPU and by other processors that may share the page tables. The i860 XR microprocessor automatically asserts the LOCK# signal while setting the A bit. If an A-bit of a PTE is found not set during a locked sequence (created by the **lock** instruction), a trap will occur and the processor will not update the A-bit.

2.4.4.7 Combining Protection of Both Levels of Page Tables

For any one page, the protection attributes of its page directory entry may differ from those of its page table entry. The i860 XR microprocessor computes the effective protection attributes for a page

by examining the protection attributes in both the directory and the page table. Table 2.6 shows the effective protection provided by the possible combinations of protection attributes.

2.4.5 ADDRESS TRANSLATION ALGORITHM

The algorithm below defines the translation of each virtual address to a physical address. Let DIR, PAGE, and OFFSET be the fields of the virtual address; let PFA1 and PFA2 be the page frame address fields of the first and second level page tables respectively; DTB is the page directory table base address stored in the **dirbase** register.

1. Read the PTE (page table entry) at the physical address formed by DTB:DIR:00.
2. If P in the PTE is zero, generate a data- or instruction-access fault.
3. If W in the PTE is zero, the operation is a write, and either the U-bit of the PSR is set or WP = 1, generate a data or instruction access fault.
4. If the U-bit in the PTE is zero and the U-bit in the **psr** is set, generate a data or instruction access fault.
5. If A in the PTE is zero, and if the TLB miss occurred while the bus was locked, generate a data or instruction access fault. (The trap allows software to set A to one and restart the sequence. This avoids ambiguity in determining what address corresponds to a locked semaphore for external bus hardware use.)
6. If A in the PTE is zero, and if the TLB miss occurred while the bus was not locked, assert LOCK#. Re-fetch and check the PTE, set A, and store the PTE. Deassert LOCK# during the store.
7. Locate the PTE at the physical address formed by PFA1:PAGE:00.
8. Perform the P, W, U, and A checks as in steps 2 through 6 with the second-level PTE.
9. If D in the PTE is clear and the operation is a write, generate a data or instruction access fault.
10. Form the physical address as PFA2:OFFSET.

The i860 XR microprocessor looks only in external memory for Page Directories and Page Tables, in the translation process. The data cache is not searched. Therefore, any code which modifies Page Directories or Page Tables must keep them out of the cache. The tables should be kept in non-cacheable memory, or flushed from the cache.

Table 2.6. Combining Directory and Page Protections

Page Directory Entry		Page Table Entry		Combined Protection		
				User Access	Supervisor Access	
U-bit	W-bit	U-bit	W-bit	WP = X	WP = 0	WP = 1
0	0	0	0	N	R/W	R
0	0	0	1	N	R/W	R
0	0	1	0	N	R/W	R
0	0	1	1	N	R/W	R
0	1	0	0	N	R/W	R
0	1	0	1	N	R/W	R/W
0	1	1	0	N	R/W	R
0	1	1	1	N	R/W	R/W
1	0	0	0	N	R/W	R
1	0	0	1	N	R/W	R
1	0	1	0	R	R/W	R
1	0	1	1	R	R/W	R
1	1	0	0	N	R/W	R
1	1	0	1	N	R/W	R/W
1	1	1	0	R	R/W	R
1	1	1	1	R/W	R/W	R/W

NOTES:
N = No access allowed R/W = Both reads and writes allowed
R = Read access only X = Don't care

The i860 XR microprocessor expects Page Directories and Page Tables to be in little endian format. The operating system must maintain these tables in little endian format by either setting BE = 0 when manipulating the tables or by complementing bit 2 of the address when loading or storing entries.

2.4.6 ADDRESS TRANSLATION FAULTS

The address translation fault is one instance of the data-access fault. The instruction causing the fault can be re-executed upon returning from the trap handler.

2.4.7 PAGE TRANSLATION CACHE

For greatest efficiency in address translation, the i860 XR microprocessor stores the most recently used page-table data in an on-chip cache called the TLB (translation lookaside buffer). Only if the necessary paging information is not in the cache must both levels of page tables be referenced.

2.5 Caching and Cache Flushing

The i860 XR microprocessor has the ability to cache instruction, data, and address-translation information in on-chip caches. Caching uses virtual-address tags. The effects of mapping two different virtual addresses in the same address space to the same physical address are undefined.

Instruction, data, and address-translation caching on the i860 XR microprocessor are not transparent. Because the data cache uses a write-back protocol, writes do not immediately update memory, and writes to memory by other bus devices do not update the cache. Changes to page tables do not automatically update the TLB, and changes to instructions do not automatically update the instruction cache. Under certain circumstances, such as I/O references, self-modifying code, page-table updates, or shared data in a multiprocessing system, it is necessary to bypass or to flush the caches. The i860 XR microprocessor provides the following methods for doing this:

- **Bypassing Instruction and Data Caches.** If deasserted during cache-miss processing, the KEN# pin disables instruction and data caching of the referenced data. If the CD bit of the associated second-level PTE is set, caching of data and instructions is disabled. The i860 XR CPU may place pages having WT = 1 into the instruction cache. Future implementations of the i860 XR architecture may adhere to a write-through data cache policy. Thus, they may cache pages having the WT bit of the PTE set. The value of the CD bit or the WT bit is output on the PTB pin for use by external caches.
- **Invalidating Instruction and Address-Translation Caches.** Storing to the **dirbase** register with the ITI bit set invalidates the contents of the instruction and address-translation caches. This bit should be set when modifying a page table, when modifying a page containing instructions, or when changing the DTB field of **dirbase** or the WP bit of the **epsr**. Note that in order to make the instruction or address-translation caches consistent with the data cache, the data cache must be flushed *before* invalidating the other caches.

  NOTE:
  The mapping of the page containing the currently executing instruction and the next six instructions should not be different in the new page tables when **st.c dirbase** changes DTB or activates ITI. The six instructions following the **st.c** should be **nop**s and should lie in the same page as the **st.c**.
- **Flushing the Data Cache.** The data cache is flushed by a software routine using the **flush** instruction. The data cache must be flushed prior to invalidating the instruction or address-translation caches (as controlled by the ITI bit of **dirbase**) or enabling or disabling address translation (via the ATE bit). The data cache does not need flushing if the program is modifying only the P, U, W, A, or D bits of a PTE (as long as the Page Frame Address is not changed and the PTE itself was not in the data cache.) The i860 XR CPU does not check these protection bits on cache line write-back. Thus, a trap handler can service a DAT for D-bit-zero by setting D = 1 and then ITI = 1. In the case of setting the P or A bits active, there is no need to invalidate or flush any caches because the processor does not load entries into the TLB that have P = 0 or A = 0. The i860 XR microprocessor searches only external memory for Page Directories and Page Tables in the translation process. The data cache is not searched. Therefore, Page Tables and Directories should be kept in non-cacheable memory, or flushed from the cache by any code which accesses them.

2.6 Instruction Set

Table 2.7 shows the complete set of instructions grouped by function within processing unit. Refer to Section 8 for an algorithmic definition of each instruction.

The architecture of the i860 XR microprocessor uses parallelism to increase the rate at which operations may be introduced into the unit. Parallelism in the i860 XR microprocessor is **not** transparent; rather, programmers have complete control over parallelism and therefore can achieve maximum performance for a variety of computational problems.

2.6.1 PIPELINED AND SCALAR OPERATIONS

One type of parallelism used within the floating-point unit is "pipelining". The pipelined architecture treats each operation as a series of more primitive operations (called "stages") that can be executed in parallel. Consider just the floating-point adder unit as an example. Let **A** represent the operation of the adder. Let the stages be represented by $\mathbf{A_1}$, $\mathbf{A_2}$, and $\mathbf{A_3}$. The stages are designed such that $\mathbf{A_{i+1}}$ for one adder instruction can execute in parallel with $\mathbf{A_i}$ for the next adder instruction. Furthermore, each $\mathbf{A_i}$ can be executed in just one clock. The pipelining within the multiplier and graphics units can be described similarly, except that the number of stages may be different.

Figure 2.12 illustrates three-stage pipelining as found in the floating-point adder (also in the floating-point multiplier when single-precision input operands are employed). The columns of the figure represent the three stages of the pipeline. Each stage holds intermediate results and also (when introduced into first stage by software) holds status information pertaining to those results. The figure assumes that the instruction stream consists of a series of consecutive floating-point instructions, all of one type (i.e. all adder instructions or all single-precision multiplier instructions). The instructions are represented as **i**, **i + 1**, etc. The rows of the figure represent the states of the unit at successive clock cycles. Each time a pipelined operation is performed, the result of the last stage of the pipeline is stored in the destination register *fdest*, the pipeline is advanced one stage, and the input operands *fsrc1* and *fsrc2* are transferred to the first stage of the pipeline.

In the i860 XR microprocessor, the number of pipeline stages ranges from one to three. A pipelined operation with a three-stage pipeline stores the result of the third prior operation. A pipelined operation with a two-stage pipeline stores the result of the second prior operation. A pipelined operation with a one-stage pipeline stores the result of the prior operation.

There are four floating-point pipelines: one for the multiplier, one for the adder, one for the graphics unit, and one for floating-point loads. The adder pipeline has three stages. The number of stages in the multiplier pipeline depends on the precision of the source operands in the pipeline. Single precision has three stages and double precision has two stages. The graphics unit has one stage for all precisions. The load pipeline has three stages for all precisions.

Changing the FZ (flush zero), RM (rounding mode), or RR (result register) bits of **fsr** while there are results in either the multiplier or adder pipeline produces effects that are not defined.

2.6.1.1 Scalar Mode

In addition to the pipelined execution mode, the i860 XR microprocessor also can execute floating-point instructions in "scalar" mode. Most floating-point instructions have both pipelined and scalar variants, distinguished by a bit in the instruction encoding. In scalar mode, the floating-point unit does not start a new operation until the previous floating-point operation is completed. The scalar operation passes through all stages of its pipeline before a new operation is introduced, and the result is stored automatically. Scalar mode is used when the next operation depends on results from the previous few floating-point operations (or when the compiler or programmer does not want to deal with pipelining).

2.6.1.2 Pipelining Status Information

Result status information in the **fsr** consists of the AA, AI, AO, AU, and AE bits, in the case of the adder, and the MA, MI, MO, and MU bits, in the case of the multiplier. This information arrives at the **fsr** via the pipeline in one of two ways:

Table 2.7. Instruction Set

Core Unit	
Mnemonic	**Description**
Load and Store Instructions	
ld.x	Load integer
st.x	Store integer
fld.y	F-P load
pfld.z	Pipelined F-P load
fst.y	F-P store
pst.d	Pixel store
Register to Register Moves	
ixfr	Transfer integer to F-P register
Integer Arithmetic Instructions	
addu	Add unsigned
adds	Add signed
subu	Subtract unsigned
subs	Subtract signed
Shift Instructions	
shl	Shift left
shr	Shift right
shra	Shift right arithmetic
shrd	Shift right double
Logical Instructions	
and	Logical AND
andh	Logical AND high
andnot	Logical AND NOT
andnoth	Logical AND NOT high
or	Logical OR
orh	Logical OR high
xor	Logical exclusive OR
xorh	Logical exclusive OR high
Control-Transfer Instructions	
trap	Software trap
intovr	Software trap on integer overflow
br	Branch direct
bri	Branch indirect
bc	Branch on CC
bc.t	Branch on CC taken
bnc	Branch on not CC
bnc.t	Branch on not CC taken
bte	Branch if equal
btne	Branch if not equal
bla	Branch on LCC and add
call	Subroutine call
calli	Indirect subroutine call
System Control Instructions	
flush	Cache flush
ld.c	Load from control register
st.c	Store to control register
lock	Begin interlocked sequence
unlock	End interlocked sequence

Floating-Point Unit	
Mnemonic	**Description**
Register to Register Moves	
fxfr	Transfer F-P to integer register
F-P Multiplier Instruction	
fmul.p	F-P multiply
pfmul.p	Pipelined F-P multiply
pfmul3.dd	3-Stage pipelined F-P multiply
fmlow.p	F-P multiply low
frcp.p	F-P reciprocal
frsqr.p	F-P reciprocal square root
F-P Adder Instructions	
fadd.p	F-P add
pfadd.p	Pipelined F-P add
famov.r	F-P adder move
pfamov.r	Pipelined F-P adder move
fsub.p	F-P subtract
pfsub.p	Pipelined F-P subtract
pfgt.p	Pipelined F-P greater-than compare
pfeq.p	Pipelined F-P equal compare
fix.p	F-P to integer conversion
pfix.p	Pipelined F-P to integer conversion
ftrunc.p	F-P to integer truncation
pftrunc.p	Pipelined F-P to integer truncation
Dual-Operation Instructions	
pfam.p	Pipelined F-P add and multiply
pfsm.p	Pipelined F-P subtract and multiply
pfmam.p	Pipelined F-P multiply with add
pfmsm.p	Pipelined F-P multiply with subtract
Long Integer Instructions	
fisub.z	Long-integer subtract
pfisub.z	Pipelined long-integer subtract
fiadd.z	Long-integer add
pfiadd.z	Pipelined long-integer add
Graphics Instructions	
fzchks	16-bit Z-buffer check
pfzchks	Pipelined 16-bit Z-buffer check
fzchkl	32-bit Z-buffer check
pfzchkl	Pipelined 32-bit Z-buffer check
faddp	Add with pixel merge
pfaddp	Pipelined add with pixel merge
faddz	Add with Z merge
pfaddz	Pipelined add with Z merge
form	OR with MERGE register
pform	Pipelined OR with MERGE register

Assembler Pseudo-Operations	
Mnemonic	**Description**
mov	Integer register-register move
fmov.r	F-P reg-reg move
pfmov.r	Pipelined F-P reg-reg move
nop	Core no-operation
fnop	F-P no-operation
pfle.p	Pipelined F-P less-than or equal

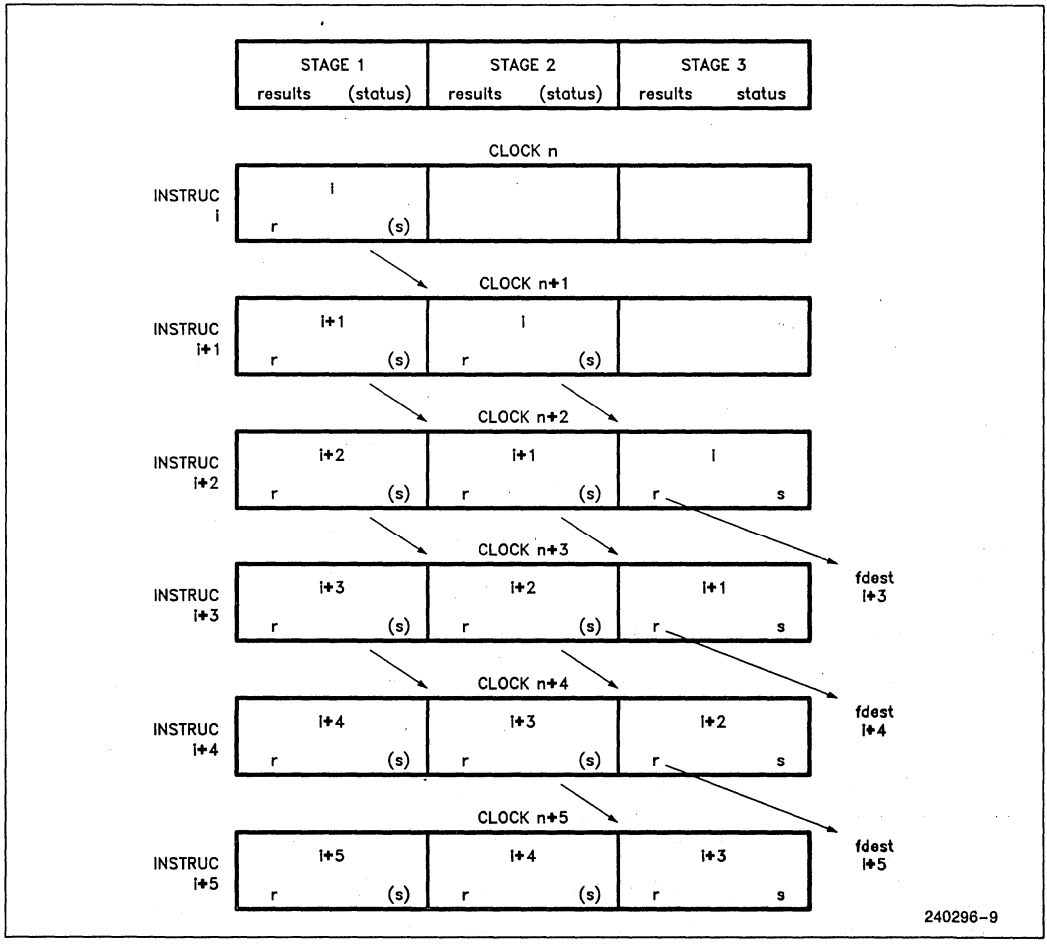


Figure 2.12. Pipelined Instruction Execution

1. It is calculated by the last stage of the pipeline. This is the normal case.
2. It is propagated from the first stage of the pipeline. This method is used when restoring the state of the pipeline after a preemption. When a store instruction updates the **fsr** and the value of the U bit in the word being written into the **fsr** is set, the store updates the result status bits in the first stage of both the adder and multiplier pipelines. When software changes the result-status bits of the first stage of a particular unit (multiplier or adder), the updated result-status bits are propagated one stage for each pipelined floating-point operation for that unit. In this case, each stage of the adder and multiplier pipelines holds its own copy of the relevant bits of the **fsr**. When they reach the last stage, they override the normal result-status bits computed from the last stage result.

At the next floating-point instruction (or, at certain core instructions), after the result reaches the last stage, the i860 XR microprocessor traps if any of the status bits of the **fsr** indicate exceptions. Note that the instruction that creates the exceptional condition is not the instruction at which the trap occurs.

2.6.1.3 Precision in the Pipelines

In pipelined mode, when a floating-point operation is initiated, the result of an earlier pipelined floating-point operation is returned. The result precision of the current instruction applies to the operation being initiated. The precision of the value stored in *fdest* is that which was specified by the instruction that initiated that operation.

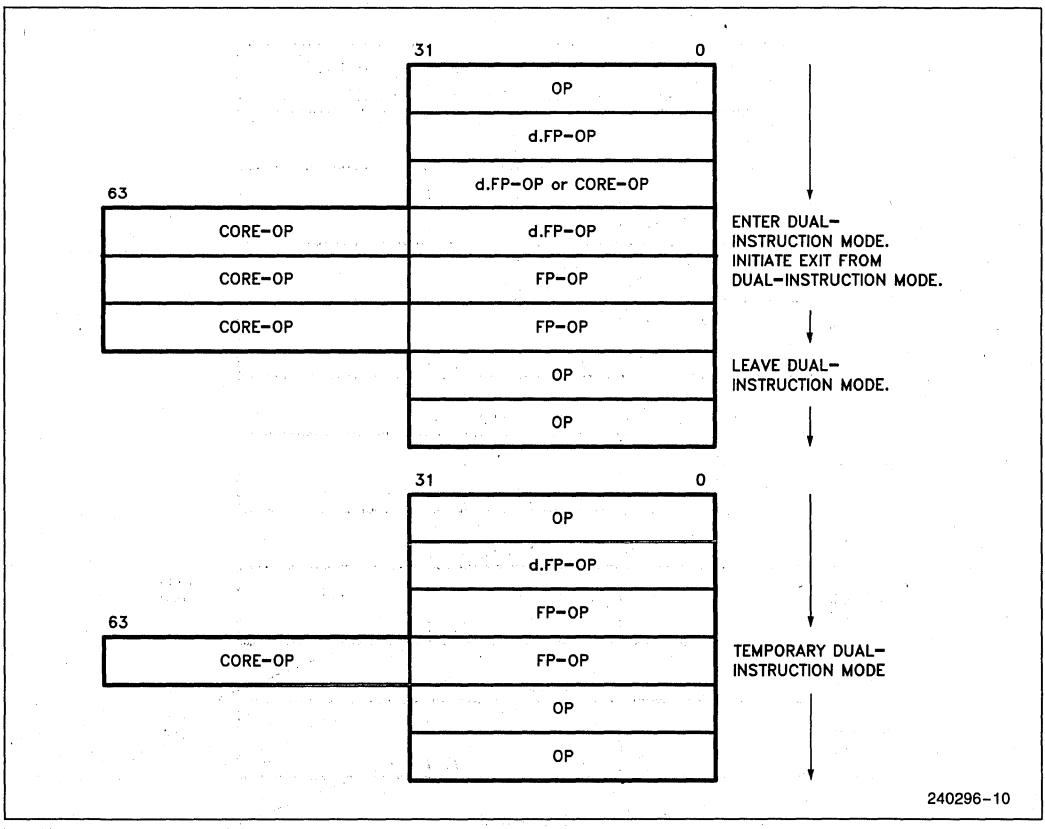


Figure 2.13. Dual-Instruction Mode Transitions

If *fdest* is the same as *fsrc1* or *fsrc2*, the value being stored in *fdest* is used as the input operand. In this case, the precision of *fdest* must be the same as the source precision.

The multiplier pipeline has two stages when the source operand is double-precision and three stages when the precision of the source operand is single. This means that a pipelined multiplier operation stores the result of the second previous multiplier operation for double-precision inputs and third previous for single-precision inputs (except when changing precisions).

2.6.1.4 Transition between Scalar and Pipelined Operations

When a scalar operation is executed, it passes through all stages of the pipeline; therefore, any unstored results in the affected pipeline are lost. To avoid losing information, the last pipelined operations before a scalar operation should be dummy pipelined operations that unload unstored results from the affected pipeline.

After a scalar operation, the values of all pipeline stages of the affected unit (except the last) are undefined. No spurious result-exception traps result when the undefined values are subsequently stored by pipelined operations; however, the values should not be referenced as source operands.

For best performance a scalar operation should not immediately precede a pipelined operation whose *fdest* is nonzero.

2.6.2 DUAL-INSTRUCTION MODE

Another form of parallelism results from the fact that the i860 XR microprocessor can execute both a floating-point and a core instruction simultaneously. Such parallel execution is called dual-instruction mode. When executing in dual-instruction mode, the instruction sequence consists of 64-bit aligned instructions with a floating-point instruction in the lower 32 bits and a core instruction in the upper 32 bits. Table 2.7 identifies which instructions are executed by the core unit and which by the floating-point unit.

Programmers specify dual-instruction mode either by including in the mnemonic of a floating-point instruction a **d.** prefix or by using the Assembler directives **.dual** . . . **.enddual**. Both of the specifications cause the D-bit of floating-point instructions to be set. If the i860 XR microprocessor is executing in single-instruction mode and encounters a floating-point instruction with the D-bit set, one more 32-bit instruction is executed before dual-mode execution begins. If the i860 XR microprocessor is executing in dual-instruction mode and a floating-point instruction is encountered with a clear D-bit, then one more pair of instructions is executed before resuming single-instruction mode. Figure 2.13 illustrates two variations of this sequence of events: one for extended sequences of dual-instructions and one for a single instruction pair.

When a 64-bit dual-instruction pair sequentially follows a delayed branch instruction in dual-instruction mode, both 32-bit instructions are executed.

2.6.3 DUAL-OPERATION INSTRUCTIONS

Special dual-operation floating-point instructions (add-and-multiply, subtract-and-multiply) use both the multiplier and adder units within the floating-point unit in parallel to efficiently execute such common tasks as evaluating systems of linear equations, performing the Fast Fourier Transform (FFT), and performing graphics transformations.

The instructions **pfam** *fsrc1*, *fsrc2*, *fdest* (add and multiply), **pfsm** *fsrc1*, *fsrc2*, *fdest* (subtract and multiply), **pfmam** *fscr1*, *fsrc2*, *fdest* (multiply and add), and **pfmsm** *fsrc1*, *fsrc2*, *fdest* (multiply and subtract) initiate both an adder operation and a multiplier operation. Six operands are required, but the instruction format specifies only three operands; therefore, there are special provisions for specifying the operands. These special provisions consist of:

- Three special registers (KR, KI, and T), that can store values from one dual-operation instruction and supply them as inputs to subsequent dual-operation instructions.
  1. The constant registers KR and KI can store the value of *fsrc1* and subsequently supply that value to the multiplier pipeline in place of *fsrc1*.
  2. The transfer register T can store the last stage result of the multiplier pipeline and subsequently supply that value to the adder pipeline in place of *fsrc1*.
- A four-bit data-path control field in the opcode (DPC) that specifies the operands and loading of the special registers.
  1. Operand-1 of the multiplier can be KR, KI, or *fsrc1*.
  2. Operand-2 of the multiplier can be *fsrc2* or the last stage result of the adder pipeline.
  3. Operand-1 of the adder can be *fsrc1*, the T-register, or the last stage result of the adder pipeline.
  4. Operand-2 of the adder can be *fsrc2*, the last stage result of the multiplier pipeline, or the last stage result of the adder pipeline.

Figure 2.14 shows all the possible data paths surrounding the adder and multiplier. A DPC field in these instructions select different data paths. Table 8.8 shows the various encodings of the DPC field. Refer to Dual Operation Instructions section in the i860 Microprocessor Programmer's Reference Manual for pictorial description.

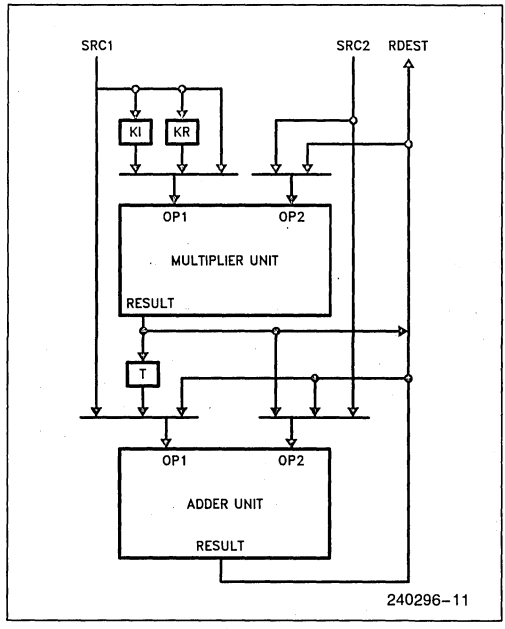


Figure 2.14. Dual-Operation Data Paths

Note that the mnemonics **pfam.p**, **pfsm.p**, **pfmam.p**, and **pfmsm.p** are never used as such in the assembly language; these mnemonics are used here to designate classes of related instructions. Each value of DPC has a unique mnemonic associated with it.

2.7 Addressing Modes

Data access is limited to load and store instructions. Memory addresses are computed from two fields of load and store instructions: *isrc1* and *isrc2*.

1. *isrc1* either contains the identifier of a 32-bit integer register or contains an immediate 16-bit address offset.
2. *isrc2* always specifies a register.

Table 2.8. Types of Traps

Type	Indication		Caused by	
	PSR,EPSR	FSR	Condition	Instruction
Instruction Fault	IT OF IL		Software traps Missing **unlock**	**trap, intovr** Any
Floating Point Fault	FT	SE AO, MO AU, MU AI, MI	Floating-point source exception Floating-point result exception overflow underflow inexact result	Any M- or A-unit except **fmlow** Any M- or A-unit except **fmlow, pfgt,** and **pfeq**. Reported on any F-P instruction plus **pst, fst,** and sometimes **fld, pfld, ixfr**
Instruction Access Fault	IAT		Address translation exception during instruction fetch	Any
Data Access Fault	DAT*		Load/store address translation exception Misaligned operand address Operand address matches **db** register	Any load/store Any load/store Any load/store
Interrupt	IN		External interrupt	
Reset	No trap bits set		Hardware RESET signal	

NOTES:
*These cases can be distinguished by examining the operand addresses.
The IL bit of the **epsr** must be checked by the trap handler to tell if the bus is currently in a locked sequence.

Because either *isrc1* or *isrc2* may be null (zero), a variety of useful addressing modes result:

offset + *register* — Useful for accessing fields within a record, where *register* points to the beginning of the record. Useful for accessing items in a stack frame, where *register* is **r3**, the register used for pointing to the beginning of the stack frame.

register + *register* — Useful for two-dimensional arrays or for array access within the stack frame.

register — Useful as the end result of any arbitrary address calculation.

offset — Absolute address into the first or last 32K of the logical address space.

In addition, the floating-point load and store instructions may select autoincrement addressing. In this mode *isrc2* is replaced by the sum of *isrc1* and *isrc2* after performing the load or store. This mode makes stepping through arrays more efficient, because it eliminates one address-calculation instruction.

2.8 Traps and Interrupts

Traps are caused by exceptional conditions detected in programs or by external interrupts. Traps cause interruption of normal program flow to execute a special program known as a trap handler. Traps are divided into the types shown in Table 2.8. Interrupts and traps start execution in single instruction mode at virtual address 0xFFFFFF00 in supervisor level (U = 0).

2.8.1 TRAP HANDLER INVOCATION

This section applies to traps other than reset. When a trap occurs, execution of the current instruction is aborted. The instruction is restartable. The processor takes the following steps while transferring control to the trap handler:

1. Copies U (user mode) of the **psr** into PU (previous U).
2. Copies IM (interrupt mode) into PIM (previous IM).
3. Sets U to zero (supervisor mode).
4. Sets IM to zero (interrupts disabled).
5. If the processor is in dual instruction mode, it sets DIM; otherwise it clears DIM.
6. If the processor is in single-instruction mode and the next instruction will be executed in dual-instruction mode or if the processor is in dual-instruction mode and the next instruction will be executed in single-instruction mode, DS is set; otherwise, it is cleared.
7. The appropriate trap type bits in **psr** are set (IT, IN, IAT, DAT, FT). Several bits may be set if the corresponding trap conditions occur simultaneously.

8. An address is placed in the fault instruction register (**fir**) to help locate the trapped instruction. In single-instruction mode, the address in **fir** is the address of the trapped instruction itself. In dual-instruction mode, the address in **fir** is that of the floating-point half of the dual instruction. If an instruction or data access fault occurred, the associated core instruction is the high-order half of the dual instruction (**fir** + 4). In dual-instruction mode, when a data access fault occurs in the absence of other trap conditions, the floating-point half of the dual instruction will already have been executed.

The processor begins executing the trap handler by transferring execution to virtual address 0xFFFFFF00. The trap handler begins execution in single-instruction mode. The trap handler must examine the trap-type bits in **psr** (IT, IN, IAT, DAT, FT) to determine the cause or causes of the trap.

2.8.2 INSTRUCTION FAULT

This fault is caused by any of the following conditions. In all cases the processor sets the IT bit before entering the trap handler.

1. By the **trap** instruction. When **trap** is executed in dual-instruction mode, the floating-point companion of the **trap** instruction is not executed before the trap is taken.
2. By the **intovr** instruction. The trap occurs only if OF in **epsr** is set when **intovr** is executed. The trap handler should clear OF before returning. When **intovr** causes a trap in dual-instruction mode, the floating-point companion of the **intovr** instruction is completely executed before the trap is taken.
3. By violation of lock/unlock protocol, explained below. (Note that **trap** and **intovr** should not be used within a locked sequence; otherwise, it would be difficult to distinguish between this and the prior cases.)

   The lock protocol requires the following sequence of activities:

   1. **lock**
   2. Any load or store instruction that misses the cache
   3. **unlock**
   4. Any load or store instruction (regardless of whether it misses the cache)

There may be other instructions between any of these steps. The bus is locked after step 2, and remains locked until step 4. Step 4 must follow step 1 by 30 instructions or less, otherwise the instruction trap occurs. In case of a trap, IL is also set. If the load or store instruction in step 2 hits the cache, the sequence is legal, but the bus is not locked.

2.8.3 FLOATING-POINT FAULT

The floating-point fault is reported on floating-point instructions, **pst**, **fst**, and sometimes **fld**, **pfld**, **ixfr**. The floating-point faults of the i860 XR microprocessor support the floating-point exceptions defined by the IEEE standard as well as some other useful classes of exceptions. The i860 XR microprocessor divides these into two classes: source exceptions and result exceptions. The numerics library supplied by Intel provides the IEEE standard default handling for all these exceptions.

2.8.3.1 Source Exception Faults

When used as inputs to the multiplier or adder, all exceptional operands, including infinities, denormalized numbers and NaNs, cause a floating-point fault and set SE in the **fsr**. Source exceptions are reported on the instruction that initiates the operation. For pipelined operations, the pipeline is not advanced.

The SE value is undefined for faults on **fld**, **pfld**, **fst**, **pst**, and **ixfr** instructions when in single-instruction mode or when in dual-instruction mode and the companion instruction is not a multiplier or adder operation.

2.8.3.2 Result Exception Faults

The class of result exceptions includes any of the following conditions:

- **Overflow.** The absolute value of the rounded true result would exceed the largest positive finite number in the destination format.
- **Underflow** (when FZ is clear). The absolute value of the rounded true result would be smaller than the smallest positive finite number in the destination format.
- **Inexact result** (when TI is set). The result is not exactly representable in the destination format. For example, the fraction 1/3 cannot be precisely represented in binary form. This exception occurs frequently and indicates that some (generally acceptable) accuracy has been lost.

The point at which a result exception is reported depends upon whether pipelined operations are being used:

- **Scalar (nonpipelined) operations.** Result exceptions are reported on the next floating-point, **fst.x**, or **pst.x** (and sometimes **fld**, **pfld**, **ixfr**) instruction after the scalar operation. When a trap occurs, the last stage of the affected unit contains the result of the scalar operation.
- **Pipelined operations.** Result exceptions are reported when the result is in the last stage and the next floating-point, **fst.x** or **pst.x** (and sometimes **fld**, **pfld**, **ixfr**) instruction is executed. When a trap occurs, the pipeline is not advanced, and the last stage results (that caused the trap) remain unchanged.

When no trap occurs (either because FTE is clear or because no exception occurred), the pipeline is advanced normally by the new floating-point operation.

The result-status bits of the affected unit are undefined until the point that result exceptions are reported. At this point, the last stage result-status bits (bits 29..22 and 16..9 of the **fsr**) reflect the values in the last stages of both the adder and multiplier. For example, if the last stage result in the multiplier has overflowed and a pipelined floating-point **pfadd** is started, a trap occurs and MO is set.

For scalar operations, the RR bits of **fsr** specify the register in which the result was stored. RR is updated when the scalar instruction is initiated. The trap, however, occurs on a subsequent instruction. Programmers must prevent intervening stores to **fsr** from modifying the RR bits. Prevention may take one of the following forms:

- Before any store to **fsr** when a result exception may be pending, execute a dummy floating-point operation to trigger the result-exception trap.
- Always read from **fsr** before storing to it, and mask updates so that the RR bits are not changed.

For pipelined operations, RR is cleared and the result is in the last stage of the pipeline of the appropriate unit. The trap handler must flush the pipeline, saving the results and the status bits.

In either pipelined or scalar mode, the trap handler must then compute the trapping result. In either case, the result has the same fraction as the true result and has an exponent which is the low-order bits of the true result. The trap handler can inspect the result, compute the result appropriate for that instruction (a NaN or an infinity, for example), and store the correct result. The result is either stored in the register specified by RR (if nonzero) or (if RR = 0) the trap handler must reload the pipeline with the saved results and status bits.

Result exceptions may be reported for both the adder and multiplier units at the same time. In this case, the trap handler should fix up the last stage of both pipelines.

2.8.4 INSTRUCTION ACCESS FAULT

This trap occurs during address translation for instruction fetches in any of these cases:

- The address fetched is in a page whose P (present) bit in the page table is clear (not present).
- The address fetched is in a supervisor mode page, but the processor is in user mode.
- The address fetched is in a page whose PTE has A = 0, and the access occurs during a locked sequence (i.e., between **lock** and **unlock**).

Note that several instructions are fetched at one time, either due to instruction prefetching or to instruction caching. Therefore, a trap handler can change from supervisor to user mode and continue to execute instructions fetched from a supervisor page. An instruction access trap occurs only when the next group of instructions is fetched from a supervisor page (up to eight instructions later). If, in the meantime, the handler branches to a user page, no instruction access trap occurs. No protection violation results, because the processor does not permit data accesses to supervisor pages while running in user mode.

2.8.5 DATA ACCESS FAULT

This trap results from an abnormal condition detected during data operand fetch or store. Such an exception can be due only to one of the following causes:

- An attempt is being made to write to a page whose D (Dirty) bit is clear.
- A memory operand is misaligned (is not located at an address that is a multiple of the length of the data).
- The address stored in the **db** register is equal to one of the addresses spanned by the operand.
- The operand is in a not-present page.
- An attempt is being made from user level to write to a read-only page or to access a supervisor-level page.
- The operand was in a page whose PTE had A = 0, and the access occurred during a locked sequence. (i.e., between **lock** and **unlock**.)
- Write protection (determined by **epsr** bit WP = 1) is violated in supervisor mode.

2.8.6 INTERRUPT TRAP

An interrupt is an event that is signaled from an external source. If the processor is executing with interrupts enabled (IM set in the **psr**), the processor sets the interrupt bit IN in the **psr**, and generates an interrupt trap. Vectored interrupts are implemented by interrupt controllers and software.

2.8.7 RESET TRAP

When the i860 XR microprocessor is reset, execution begins in single-instruction mode at physical address 0xFFFFFF00. This is the same address as for other traps. The reset trap can be distinguished from other traps by the fact that no trap bits are set. The instruction cache is flushed. The bits DPS, BL, and ATE in **dirbase** are cleared. CS8 is initialized by the value at the INT pin at the end of reset. The read-only fields of the **espr** are set to identify the processor, while the IL, WP, and PBM bits are cleared. The

bits U, IM, BR, and BW in **psr** are cleared, as are the trap bits FT, DAT, IAT, IN, and IT. All other bits of **psr** and all other register contents are **undefined.**

Refer to Table 2.9 for a summary of these initial settings.

Table 2.9. Register and Cache Values after Reset

Registers	Initial Value
Integer Registers	*Undefined*
Floating-Point Registers	*Undefined*
psr	U, IM, BR, BW, FT, DAT, IAT, IN, IT = 0; others are *undefined*
epsr	IL, WP, PBM, BE = 0; Processor Type, Stepping Number, DCS are read only; others are *undefined*
db	*Undefined*
dirbase	DPS, BL, ATE = 0; others are *undefined*
fir	*Undefined*
fsr	*Undefined*
KR, KI, T, MERGE	*Undefined*
Caches	**Initial Value**
Instruction Cache	Flushed
Data Cache	*Undefined*
TLB	Flushed

The software must ensure that the data cache is flushed and control registers are properly initialized before performing operations that depend on the values of the cache or registers. The data cache has no "validity" bits, so memory accesses before the flush may result in false data cache hits.

Reset code must initialize the floating-point pipeline state to zero with floating-point traps disabled to ensure that no spurious floating-point traps are generated.

After a RESET the i860 XR microprocessor starts execution at supervisor level (U = 0). Before branching to the first user-level instruction, the RESET trap handler or subsequent initialization code has to set PU and a trap bit so that an indirect branch instruction will copy PU to U, thereby changing to user level.

2.9 Debugging

The i860 XR microprocessor supports debugging with both data and instruction breakpoints. The features of the i860 XR architecture that support debugging include:

- **db** (data breakpoint register) which permits specification of a data addresses that the i860 XR microprocessor will monitor.
- BR (break read) and BW (break write) bits of the **psr**, which enable trapping of either reads or writes (respectively) to the address in **db**.
- DAT (data access trap) bit of the **psr**, which allows the trap handler to determine when a data breakpoint was the cause of the trap.
- **trap** instruction that can be used to set breakpoints in code. Any number of code breakpoints can be set. The values of the *isrc1* and *isrc2* fields help identify which breakpoint has occurred.
- IT (instruction trap) bit of the **psr**, which allows the trap handler to determine when a **trap** instruction was the cause of the trap.

3.0 HARDWARE INTERFACE

2

In the following description of hardware interface, the # symbol at the end of a signal name indicates that the active or asserted state occurs when the signal is at a low voltage. When no # is present after the signal name, the signal is asserted when at the high voltage level.

3.1 Signal Description

Table 3.1 identifies functional groupings of the pins, lists every pin by its identifier, gives a brief description of its function, and lists some of its characteristics. All output pins are tristate, except HLDA and BREQ. All inputs are synchronous, except HOLD and INT.

3.1.1 CLOCK (CLK)

The CLK input determines execution rate and timing of the i860 XR microprocessor. Timing of other signals is specified relative to the rising edge of this signal. The i860 XR microprocessor can utilize a clock rate of 25 MHz, 33.3 MHz or 40 MHz. The internal operating frequency is the same as the external clock.

3.1.2 SYSTEM RESET (RESET)

Asserting RESET for at least 16 CLK periods causes initialization of the i860 XR microprocessor. Refer to section 3.2 "Initialization" for more details related to RESET.

3.1.3 BUS HOLD (HOLD) AND BUS HOLD ACKNOWLEDGE (HLDA)

These pins are used for i860 XR microprocessor bus arbitration. At some clock after the HOLD signal is asserted, the i860 XR microprocessor releases con-

Table 3.1. Pin Summary

Pin Name	Function	Active State	Input/ Output
	Execution Control Pins		
CLK	CLocK		I
RESET	System reset	High	I
HOLD	Bus hold	High	I
HLDA	Bus hold acknowledge	High	O
BREQ	Bus request	High	O
INT/CS8	Interrupt, code-size	High	I
	Bus Interface Pins		
A31–A3	Address bus	High	O
BE7#–BE0#	Byte Enables	Low	O
D63–D0	Data bus	High	I/O
LOCK#	Bus lock	Low	O
W/R#	Write/Read bus cycle	High/Low	O
NENE#	NExt NEar	Low	O
NA#	Next Address request	Low	I
READY#	Transfer Acknowledge	Low	I
ADS#	ADdress Status	Low	O
	Cache Interface Pins		
KEN#	Cache ENable	Low	I
PTB	Page Table Bit	High	O
	Testability Pins		
SHI	Boundary Scan Shift Input	High	I
BSCN	Boundary Scan Enable	High	I
SCAN	Shift Scan Path	High	I
	Intel-Reserved Configuration Pins		
CC1–CC0	Configuration	High	I
	Power and Ground Pins		
V_{CC}	System power		
V_{SS}	System ground		

A # after a pin name indicates that the signal is active when at the low voltage level.

trol of the local bus and puts all bus interface outputs (except BREQ and HLDA) into a floating state, then asserts HLDA—all during the same clock period. It maintains this state until HOLD is deasserted. Instruction execution stops only if required instructions or data cannot be read from the on-chip instruction and data caches.

The time required to acknowledge a hold request is one clock plus the number of clocks needed to finish any outstanding bus cycles. HOLD is recognized even while RESET or LOCK# is asserted.

When leaving a bus hold, the i860 XR microprocessor deactivates HLDA and, in the same clock period, initiates a pending bus cycle, if any.

Hold is an asynchronous input.

3.1.4 BUS REQUEST (BREQ)

This signal is asserted when the i860 XR microprocessor has a pending memory request, even when HLDA is asserted. This allows an external bus arbiter to implement an "on demand only" policy for granting the bus to the i860 XR microprocessor. BREQ is asserted the clock after the i860 XR microprocessor realizes an internal request for the bus. In normal operation, BREQ goes low the clock after ADS# goes low for the final pending bus cycle. (Refer to Figure 4.10 for timing information.) During data or instuction cache fills, however, BREQ may be deasserted for one or more clocks, due to cache and TLB logic.

3.1.5 INTERRUPT/CODE-SIZE (INT/CS8)

This input allows interruption of the current instruction stream. If interrupts are enabled (IM set in **psr**) when INT is asserted, the i860 XR microprocessor fetches the next instruction from address

0xFFFFFF00. To assure that an interrupt is recognized, INT should remain asserted until the software acknowledges the interrupt (by writing, for example, to a memory-mapped port of an interrupt controller). When the bus is not locked, the maximum time between the assertion of INT and the execution of the first instruction of the trap handler is ten clocks, plus the time for four sets of four pipelined read cycles and two sets of four pipelined writes (instruction- and data-cache misses and write-back cycles to update memory), plus the time for twenty nonpipelined read cycles (six TLB misses, with eight refetches when the A-bit is zero), plus the time for eight nonpipelined writes (updates to the A-bit).

If the bus is locked from a **lock** instruction, the INT pin is ignored and the INT bit of **epsr** is always zero. The **lock** instruction can only assert LOCK# for 30–33 instructions before trapping.

If INT is asserted during the clock before the falling edge of RESET, the eight-bit code-size mode is selected. For more about this mode, refer to section 3.2 "Initialization".

INT is an asynchronous input.

3.1.6 ADDRESS PINS (A31–A3) AND BYTE ENABLES (BE7#–BE0#)

The 29-bit address bus (A31–A3) identifies addresses to a 64-bit location. Separate byte-enable signals (BE7#–BE0#) identify which bytes should be accessed within the 64-bit location. In all noncacheable read cycles (KEN# deasserted), the byte enables match the length and address of the requested data. Cacheable read cycles (KEN# asserted), however, result in four 64-bit memory cycles to fill an entire 32-byte cache line. The BE*n*# pins activated are those that represent the operand of the load instruction that caused the line fill, and these same BE*n*# pins remain activated for all four cycles of the line fill. All 64 bits must be returned for each cycle without regard for the BE*n*# signals. In all write cycles (noncacheable writes as well as cache line write-backs) the BE*n*# signals indicate the bytes that must be written.

Instruction fetches (W/R# is low) are distinguished from data accesses by the unique combinations of BE7#–BE0# defined in Table 3.2. For an eight-bit code fetch in eight-bit code-size (CS8) mode, BE2#–BE0# are redefined to be A2–A0 of the address. In this case BE7#–BE3# form the code shown in Table 3.2 that identifies an instruction fetch. The A2 in the table does not represent a physical pin, just a conceptual internal address line value. The "x" under A2 for CS8 mode means "not applicable", or "don't care". All other combinations of byte enables indicate data accesses.

The address and byte-enable pins are driven until either NA# or READY# is asserted.

3.1.7 DATA PINS (D63–D0)

The bus interface has 64 bidirectional data pins (D63–D0) to transfer data in eight- to 64-bit quantities. Pins D7–D0 transfer the least significant byte; pins D63–D56 transfer the most significant byte.

In read bus cycles, all 64 bits of the data bus are latched, even in CS8-mode instruction fetches when only the low-order eight bits are used.

In write bus cycles, the point at which data is driven onto the bus depends on the type of the preceding cycle. If there was no preceding cycle (i.e. the bus was idle), data is driven with the address. If the preceding cycle was a write, data is driven as soon as READY# is returned from the previous cycle. If the preceding cycle was a read, data is driven one clock after READY# is returned from the previous cycle, thereby allowing time for the bus to be turned around. Data continues to be driven until READY# for the current cycle is returned.

3.1.8 BUS LOCK (LOCK#)

This signal is used to provide atomic (indivisible) read-modify-write sequences in multiprocessor systems. A multiprocessor bus arbiter must permit only one processor a locked access to the address which is on the bus when LOCK# first activates. The system must maintain the lock of that location until LOCK# deactivates.

The i860 XR microprocessor coordinates the external LOCK# signal with the software-controlled BL bit of the **dirbase** register. Programmers do not have to be concerned about the fact that bus activity is not always synchronous with instruction execution. LOCK# is asserted with ADS# for the address operand of the first load or store instruction executed after the BL bit is set by the **lock** instruction. Pending bus cycles are locked according to the value of the BL bit when the instruction was executed. Even if the BL bit is changed between the time that an instruction generates an internal bus request and the time that the cycle appears on the bus, the i860 XR microprocessor still asserts LOCK# for that bus cycle.

If ADS# is active when LOCK# deactivates, then that request should complete before the hardware relinquishes the lock. If ADS# is not active, the locking of the location can immediately end when LOCK# deactivates. Of course the simplest arbitration hardware can just lock the entire bus against all other accesses during LOCK# assertion through RDY# of the cycle in which LOCK# goes inactive.

Table 3.2. Identifying Instruction Fetches

Code Fetch	A2	BE7#	BE6#	BE5#	BE4#	BE3#	BE2#	BE1#	BE0#
Normal (Non-CS8)	0	1	1	1	1	1	0	1	0
Normal (Non-CS8)	1	1	0	1	0	1	1	1	1
CS8 Mode	x	1	0	1	0	1	Low-order address bits		

When the BL bit is deasserted with the **unlock** instruction, LOCK# is deasserted with the next load or store but after any pending bus cycles. Between locked sequences, at least one cycle of no LOCK# is guaranteed by the behavior of the **unlock** instruction. LOCK# deassertion may occur independently of ADS# for the case of a trap or a cache hit after **unlock**.

The i860 XR microprocessor also asserts LOCK# during TLB miss processing for updates of the accessed bit in page-table entries. The maximum time that LOCK# can be asserted in this case is five clocks plus the time required to perform a read-modify-write sequence. Instruction fetches do not alter the LOCK# pin.

Between **lock** and **unlock** instructions, the INT pin is ignored and the INT bit of **epsr** is zero when read by **ld.c epsr**. The time that interrupts are disabled is limited by the lock protocol outlined in Section 2.8.2.

3.1.9 WRITE/READ BUS CYCLE (W/R#)

This pin specifies whether a bus cycle is a read (LOW) or write (HIGH) cycle. It is driven until either NA# or READY# is asserted.

3.1.10 NEXT NEAR (NENE#)

This signal allows higher-speed reads and writes in the case of consecutive reads and writes that access static column or page-mode DRAMs. The i860 XR microprocessor asserts NENE# when the current address is in the same DRAM page as the previous bus cycle. The i860 XR microprocessor determines the DRAM page size by inspecting the DPS field in the **dirbase** register. The page size can range from 2^9 to 2^{16} 64-bit words, supporting DRAM sizes from 256K × 1, 256K × 4, and up. NENE# is never asserted on the next bus cycle after HLDA is deasserted.

3.1.11 NEXT ADDRESS REQUEST (NA#)

NA# makes address pipelining possible. The system asserts NA# for at least one clock to indicate that it is ready to accept the next address from the i860 XR microprocessor. NA# may be asserted before the current cycle ends. (If the system does not implement pipelining, NA# does not have to be activated.) The i860 XR microprocessor samples NA# every clock, starting one clock after the prior activation of ADS#. When NA# is active, the i860 XR microprocessor is free to drive address and bus-cycle definition for the next pending bus cycle. The i860 XR microprocessor remembers that NA# was asserted when no internal request is pending; therefore, NA# can be deactivated after the next rising edge of the CLK signal. Up to three bus cycles can be outstanding simultaneously.

3.1.12 TRANSFER ACKNOWLEDGE (READY#)

The system must assert the READY# signal during read cycles when valid data is on the data pins and during write cycles when the system has accepted data from the data pins. READY# must be asserted for at least one clock. Sampling of READY# begins in the clock after an ADS# or in the second clock after a prior READY#.

3.1.13 ADDRESS STATUS (ADS#)

The i860 XR microprocessor asserts ADS# during the first clock of each bus cycle to identify the clock period during which it begins to assert outputs on the address bus. This signal is held active for one clock.

3.1.14 CACHE ENABLE (KEN#)

The i860 XR microprocessor samples KEN# to determine whether the data being read for the current cache-miss cycle is to be cached. This pin is internally NORed with the CD and WT bits to control cacheability on a page by page basis (refer to Table 3.3).

If the address is one that is permitted to be in the cache, KEN# must be continuously asserted during the sampling period starting from the second rising clock edge after ADS# is asserted, through the clock NA# or READY# is asserted. The entire 64 bits of the data bus will be used for the read, regardless of the state of the byte-enable pins. Three additional 64-bit bus cycles will be generated to fill the rest of the 32-byte cache block.

If KEN# is found deasserted at any clock from the clock after ADS# through the clock of the **first** NA# or READY#, the data being read will not be cached and two scenarios can occur: 1) if the cycle is due to data-cache miss, no subsequent cache-fill cycles will be generated; 2) if the cycle is due to an instruction-cache miss, additional cycle(s) will be generated until the address reaches a 32-byte boundary. To avoid caching a line, external hardware must deassert KEN# during or before the first NA# or READY#.

3.1.15 PAGE TABLE BIT (PTB)

Depending on the setting of the PBM (page-table bit mode) bit of the **epsr**, the PTB reflects the value of either the CD (cache disable) bit or the WT (write through) bit of the page-table entry used for the current cycle. When paging is disabled, PTB remains inactive.

Table 3.3. Cacheability based on KEN# and CD OR WT

CD OR WT	KEN#	Meaning
0	0	Cacheable access
0	1	Noncacheable access
1	0	Noncacheable page
1	1	Noncacheable page

3.1.16 BOUNDARY SCAN SHIFT INPUT (SHI)

This pin is used with the testability features. Refer to section 3.3.

3.1.17 BOUNDARY SCAN ENABLE (BSCN)

This pin is used with the testability features. Refer to section 3.3.

3.1.18 SHIFT SCAN PATH (SCAN)

This pin is used with the testability features. Refer to section 3.3.

3.1.19 CONFIGURATION (CC1-CC0)

These two pins are reserved by Intel. Strap both pins LOW.

3.1.20 SYSTEM POWER (V_{CC}) AND GROUND (V_{SS})

The i860 XR microprocessor has 48 pins for power and ground. All pins must be connected to the appropriate low-inductance power and ground signals in the system.

3.2 Initialization

Initialization of the i860 XR microprocessor is caused by assertion of the RESET signal for at least 16 clocks. Table 3.4 shows the status of output pins during the time that RESET is asserted. Note that HOLD requests are honored during RESET and that the status of output pins depends on whether a HOLD request is being acknowledged.

Table 3.4. Output Pin Status during Reset

Pin Name	Pin Value	
	HOLD Not Acknowledged	HOLD Acknowledged
ADS#, LOCK#	HIGH	Tri-State OFF
W/R#, PTB	LOW	Tri-State OFF
BREQ	LOW	LOW
HLDA	LOW	HIGH
D63–D0	Tri-State OFF	Tri-State OFF
A31–A3, BE7#–BE0#, NENE#	Undefined	Tri-State OFF

After a reset, the i860 XR microprocessor begins executing at physical address 0xFFFFFF00. The program-visible state of the i860 XR microprocessor after reset is detailed in section 2.8.7.

Eight-bit code-size mode is selected when INT/CS8 is asserted during the clock before the falling edge of RESET. While in eight-bit code-size mode, instruction cache misses are byte reads (transferred on D7-D0 of the data bus) instead of eight-byte reads. This allows the i860 XR microprocessor to be bootstrapped from an eight-bit EPROM. For these code reads, byte enables BE2#–BE0# are redefined to be the low order three bits of the address, so that a complete byte address is available. These reads update the instruction cache if KEN# is asserted (refer to section 3.1.14) and are not pipelined even if NA# is asserted. While in this mode, instructions must reside in an eight-bit wide memory, while data must reside in a separate 64-bit wide memory. After the code has been loaded into 64-bit memory, initialization code can initiate 64-bit code fetches by clearing the CS8 bit of the **dirbase** register (refer to section 2). Once eight-bit code-size mode is disabled by software, it cannot be reenabled except by resetting the i860 XR microprocessor.

3.3 Testability

The i860 XR microprocessor has a *boundary scan mode* that may be used in component- or board-level testing to test the signal traces leading to and from the i860 XR microprocessor. Boundary scan mode provides a simple serial interface that makes it possible to test all signal traces with only a few probes. Probes need be connected only to CLK, BSCN, SCAN, SHI, BREQ, RESET, and HOLD.

The pins BSCN and SCAN control the boundary scan mode (refer to Table 3.5). When BSCN is asserted, the i860 XR microprocessor enters boundary scan mode on the next rising clock edge. Boundary scan mode can be activated even while RESET is active. When BSCN is deasserted while in boundary scan mode, the i860 XR microprocessor leaves boundary scan mode on the next rising clock edge. After leaving boundary scan mode, the internal state is undefined; therefore, RESET should be asserted.

Table 3.5. Test Mode Selection

BSCN	SCAN	Testability Mode
LO	LO	No testability mode selected
LO	HI	(Reserved for Intel)
HI	LO	Boundary scan mode, normal
HI	HI	Boundary scan mode, shift SHI as input; BREQ as output

For testing purposes, each signal pin has associated with it an internal latch. Table 3.6 indentifies these latches by name and classifies them as input, output, or control. The input and output latches carry the name of the corresponding pins.

Table 3.6. Test Mode Latches

Input Latch	Output Latch	Associated Control Latch
SHI		
BSCN		
SCAN		
RESET		
D0–D63	D0–D63	DATAt
CC1-CC0		
	A31–A3	ADDRt
	NENE#	NENEt
	PTB#	PTBt
	W/R#	W/Rt
	ADS#	ADSt
	HLDA	
	LOCK#	LOCKt
READY#		
KEN#		
NA#		
INT/CS8		
HOLD		
	BE7#–BE0#	BEt
	BREQ	

Within boundary scan mode the i860 XR microprocessor operates in one of two submodes: normal mode or shift mode, depending on the value of the SCAN input. A typical test sequence is . . .

1. Enter shift mode to assign values to the latches that correspond with the pins.
2. Enter normal mode. In normal mode the i860 XR microprocessor transfers the latched values to the output pins and latches the values that are being driven onto the input pins.
3. Reenter shift mode to read the new values of the input pins.

3.3.1 NORMAL MODE

When SCAN is deasserted, the normal mode is selected. For each input pin (RESET, HOLD, INT/CS8, NA#, READY#, KEN#, SHI, BSCN, SCAN, CC1, and CC0), the corresponding latch is loaded with the value that is being driven onto the pin.

The tristate output pins (A31–A3, BE7#–BE0#, W/R#, NENE#, ADS#, LOCK#, and PTB) are enabled by the control latches ADDRt (for A31–A3), BEt, W/Rt, NENEt, ADSt, LOCKt, and PTBt. If a control latch is set, the corresponding output latches drive their output pins; otherwise the pins are not driven.

The I/O pins (D63–D0) are enabled by the control latch DATAt, which is similar to the other control latches. In addition, when DATAt is not set, the data pins are treated as input pins and their values are latched.

3.3.2 SHIFT MODE

When SCAN is asserted, the shift mode is selected. In shift mode, the pins are organized into a *boundary scan chain*. The scan chain is configured as a shift register that is shifted on the rising edge of CLK. The SHI pin is connected to the input of one end of the boundary scan chain. The value of the most significant bit of the scan chain is output on the BREQ pin. To avoid glitches while the values are being shifted along the chain, the tester should assert both the RESET and HOLD pins. Then all tristate outputs are disabled. The order of the pins within the chain is shown in Figure 3.1.

A tester causes entry into this mode for one of two purposes:

1. To assign values to output latches to be driven onto output pins upon subsequent entry into normal mode.
2. To read the values of input pins previously latched in normal mode.

4.0 BUS OPERATION

A bus cycle begins when ADS# is activated and ends when READY# is sampled active. READY# is sampled one clock after assertion of ADS# and thereafter until it becomes active. New cycles can start as often as every other clock until three cycles are outstanding. A bus cycle is considered outstanding as long as READY# has not been asserted to terminate that cycle. After READY# becomes active, it is not sampled again for the following (outstanding) cycle until the second clock after the one during which it became active. READY# is assumed to be inactive when it is not sampled.

With regard to how a bus cycle is generated by the i860 XR microprocessor, there are two types of cycles: pipelined and nonpipelined. Both types of cycles can be either read or write cycles. A pipelined cycle is one that starts while one or two other bus cycles are outstanding. A nonpipelined cycle is one that starts when no other bus cycles are outstanding.

4.1 Pipelining

A **m-n** read or write cycle is a cycle with a total cycle time of **m** clocks and a cycle-to-cycle time of **n** clocks (**m** ≥ **n**). Total cycle time extends from the clock in which ADS# is activated to the clock in which READY# becomes active, whereas cycle-to-cycle time extends from the time that READY# is sampled active for the previous cycle to the time that it is sampled active again for the current cycle. When **m** = **n**, a nonpipelined cycle is implied; **m** > **n** implies a pipelined cycle.

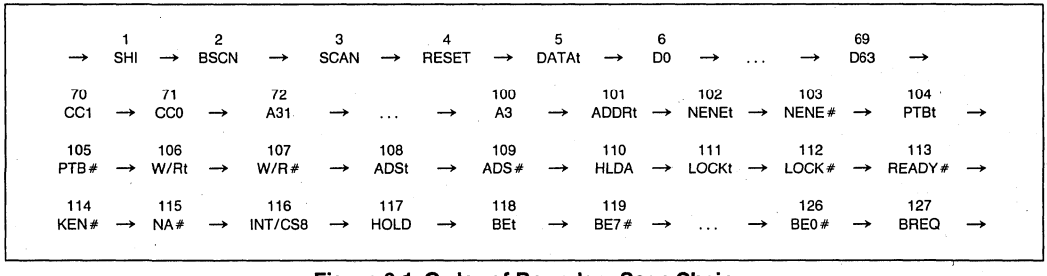


Figure 3.1. Order of Boundary Scan Chain

Pipelining may occur for the next bus cycle any time the current bus cycle requires more than two clock periods to finish (**m** > **2**). If a bus request is pending, the next cycle will be initiated when NA# is sampled active, even if the current cycle has not terminated. In this case, pipelining occurs. NA# is not recognized unitl after ADS# has become inactive.

To allow high transfer rates in large memory systems, two-level pipelining is supported (i.e., there may be up to three cycles in progress at one time). Pipelining enables a new word of data to be transferred every two clocks, even though the total cycle time may be up to six clocks.

4.2 Bus State Machine

The operation of the bus is described in terms of a bus state machine using a state transition diagram. Figure 4.1 illustrates the i860 XR microprocessor bus state machine. A bus cycle is composed of two or more states. Each bus state lasts for one CLK period.

The i860 XR microprocessor supports up to two levels of address pipelining. Once it has started the first bus cycle, it can generate up to two more cycles as long as READY# remains inactive. To start a new bus cycle while other cycles are still outstanding, NA# must be active for at least one clock cycle starting with the clock after the previous ADS#. NA# is latched internally.

States T_j and T_{jk}, for **j** = {1,2,3} and **k** = {1,2}, are used to describe the state of the i860 XR microprocessor Bus State Machine. Index **j** indicates the number of outstanding bus cycles while index **k** distinguishes the intermediate states for the **j**-th outstanding cycle. Therefore there can be up to three outstanding cycles, and there are two possible intermediate states for each level of pipelining. T_{j1} is the next state after T_j, as long as **j** cycles are outstanding. T_{j2} is entered when NA# is active but the i860 XR microprocessor is not ready to start a new cycle.

Five conditions have to be met to start a new cycle while one or more cycles are already pending:

1. READY# inactive
2. NA# having been active
3. An internal request pending (BREQ active)
4. HOLD not active
5. Fewer than three cycles outstanding

Note that BREQ is asserted on the clock after the i860 XR microprocessor realizes an internal request for the bus.

Upon hardware RESET, the bus control logic enters the idle state T_I and awaits an internal request for a bus cycle. If a bus cycle is requested while there is no hold request from the system, a bus cycle begins, advancing to state T_1. On the next cycle, the state machine automatically advances to state T_{11}. If READY# is active in state T_{11}, the bus control logic returns either to T_I, if no new cycle is started, or to T_1, if a new cycle request is pending internally. In fact, if an internal bus request is pending each time READY# is active, the state machine continues to cycle between T_{11} and T_1.

However, if READY# is not active but the next address request is pending (as indicated by an active NA#), the state machine advances either to state T_2 (if an internal bus request is pending, signifying that two bus cycles are now outstanding), or to state T_{12} (if no bus internal request is pending, signifying NA# has been found active). Transitions from state T_{12} are similar to those from T_{11}.

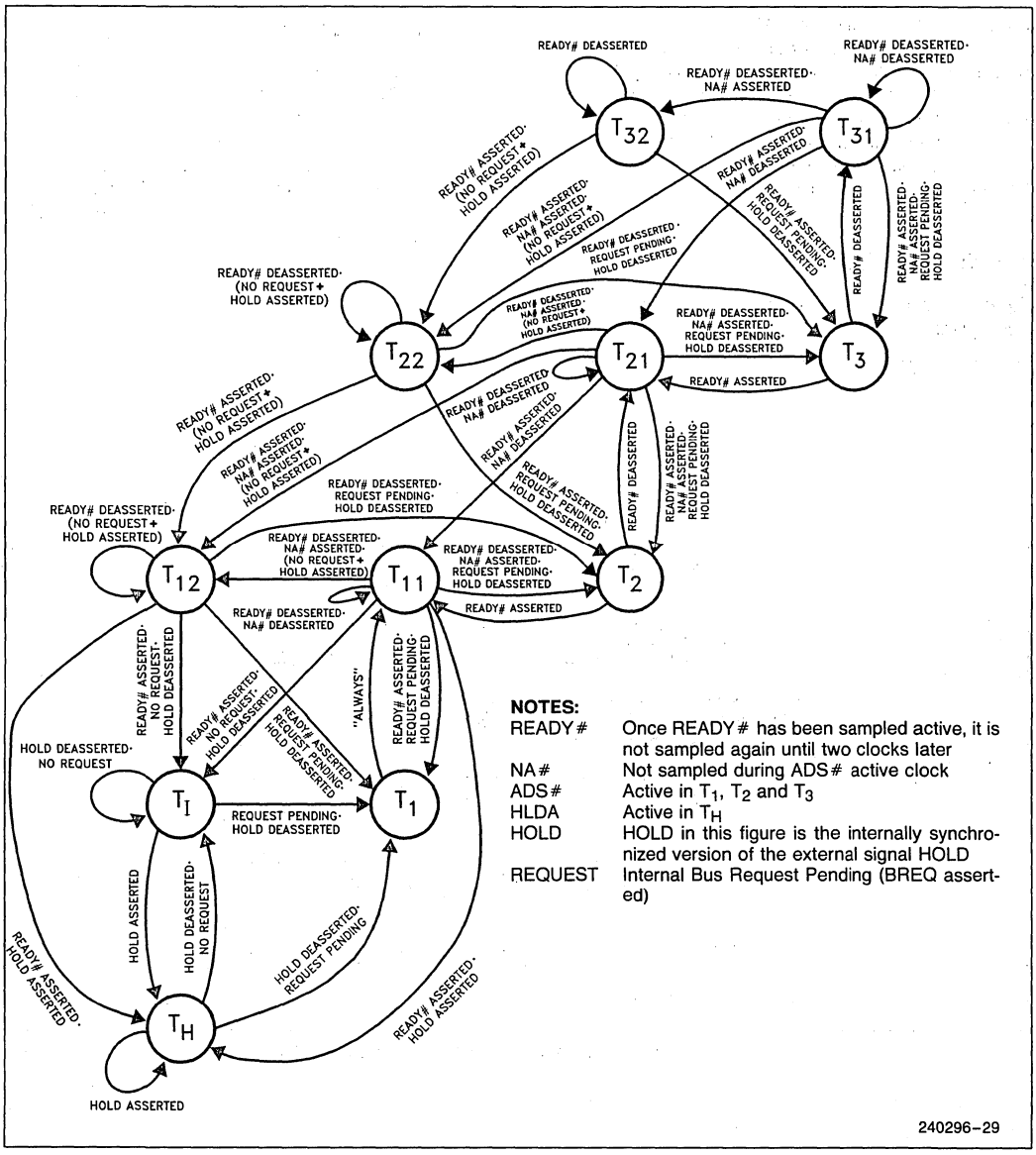


Figure 4.1. Bus State Machine

If two bus cycles are already outstanding (as indicated by T_{2k} for **k** = {1,2}) and NA# is latched active but READY# is not active, one more bus request causes entry into state T_3. Transitions from this state are similar to those from T_2.

In general, if there is an internal bus request each time both READY# and NA# are active, the state machine continues to oscillate between T_{j1} and T_j, for **j** = {2,3}.

When NA# is sampled active while there is a pending bus request, ADS# is activated in the next clock period (provided no more than two cycles are already outstanding).

Internal pending bus requests start new bus cycles only if no HOLD request has been recognized. T_H is entered from the idle state T_I, T_{11}, and T_{12}. HLDA is active in this state. There is a one clock delay to synchronize the HOLD input when the signal meets the respective minimum setup and hold time requirements. The state machine uses the synchronized HOLD to move from state to state.

4.3 Bus Cycles

Figures 4.2 through 4.10 illustrate combinations of bus cycles.

4.3.1 NONPIPELINED READ CYCLES

A read cycle begins with the clock in which ADS# is asserted. The i860 XR microprocessor begins driving the address during this clock. It samples READY# for active state every clock after the first clock. A minimum of two clocks is required per cycle. Data is latched when READY# is found active when sampled at the end of a clock period. Figure 4.2 illustrates nonpipelined read cycles with zero wait states.

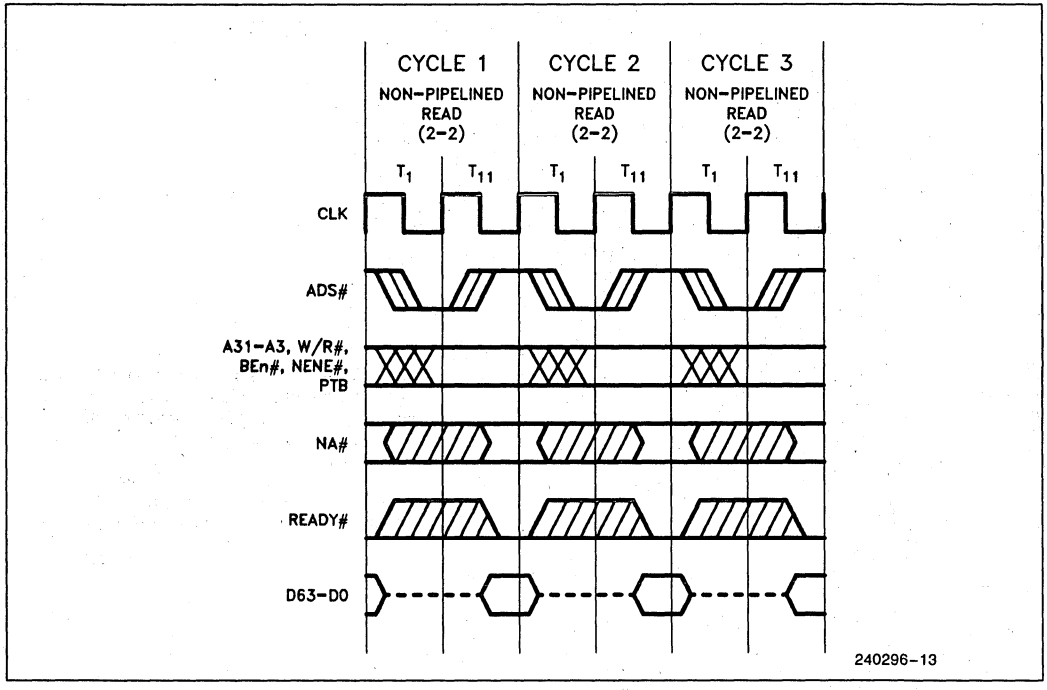


Figure 4.2. Fastest Read Cycles

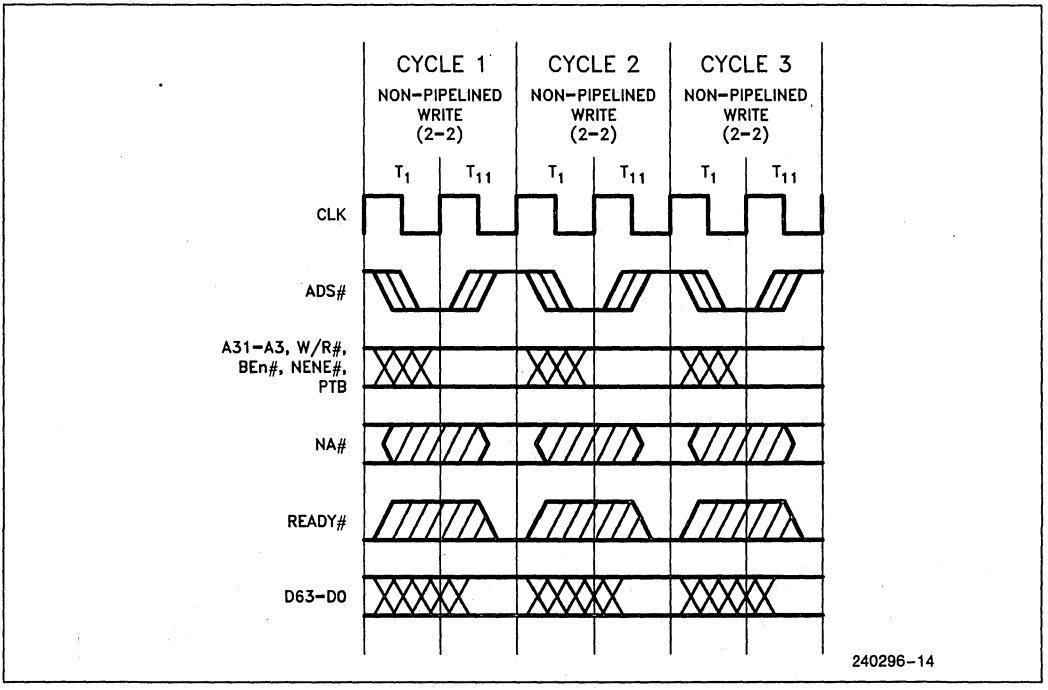


Figure 4.3. Fastest Write Cycles

4.3.2 NONPIPELINED WRITE CYCLES

The ADS# and READY# activity for write cycles follows the same logic as that for read cycles, as Figure 4.3 illustrates for back-to-back, nonpipelined write cycles with zero wait-states.

The fastest write cycle takes only two clocks to complete. However, when a read cycle immediately precedes a write cycle, the write cycle must contain a wait state, as illustrated in Figure 4.4. Because the device being read might still be driving the data bus during the first clock of the write cycle, there is a potential for bus contention. To help avoid such contention, the i860 XR microprocessor does not drive the data bus until the second clock of the write cycle. The wait state is required to provide the additional time necessary to terminate the write cycle. In other read-write combinations, the i860 XR microprocessor does not require a wait state.

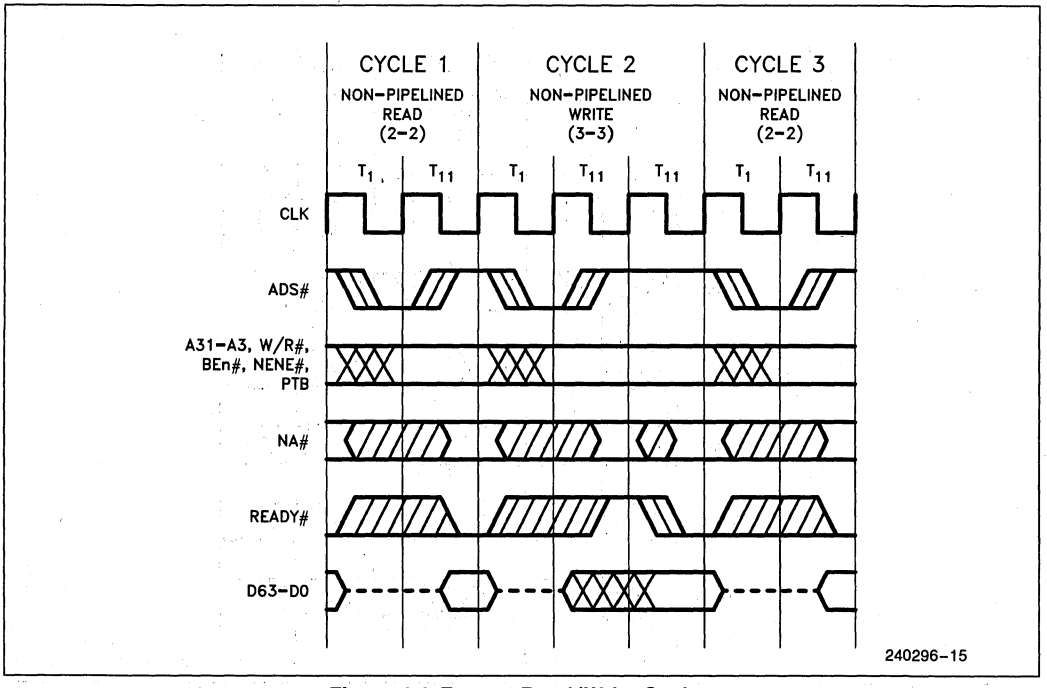


Figure 4.4. Fastest Read/Write Cycles

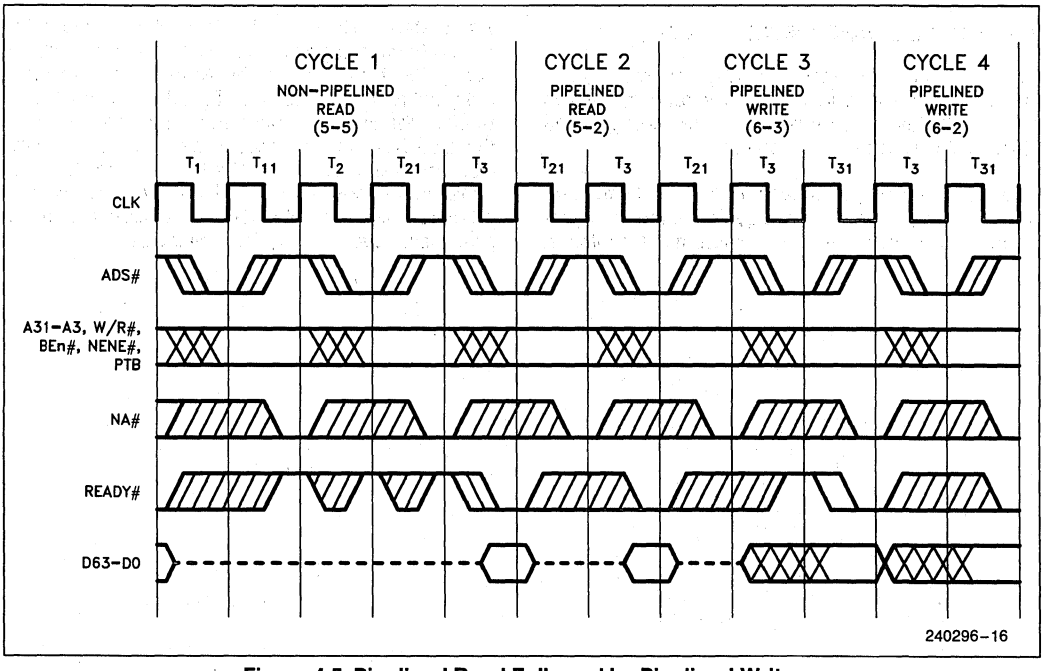


Figure 4.5. Pipelined Read Followed by Pipelined Write

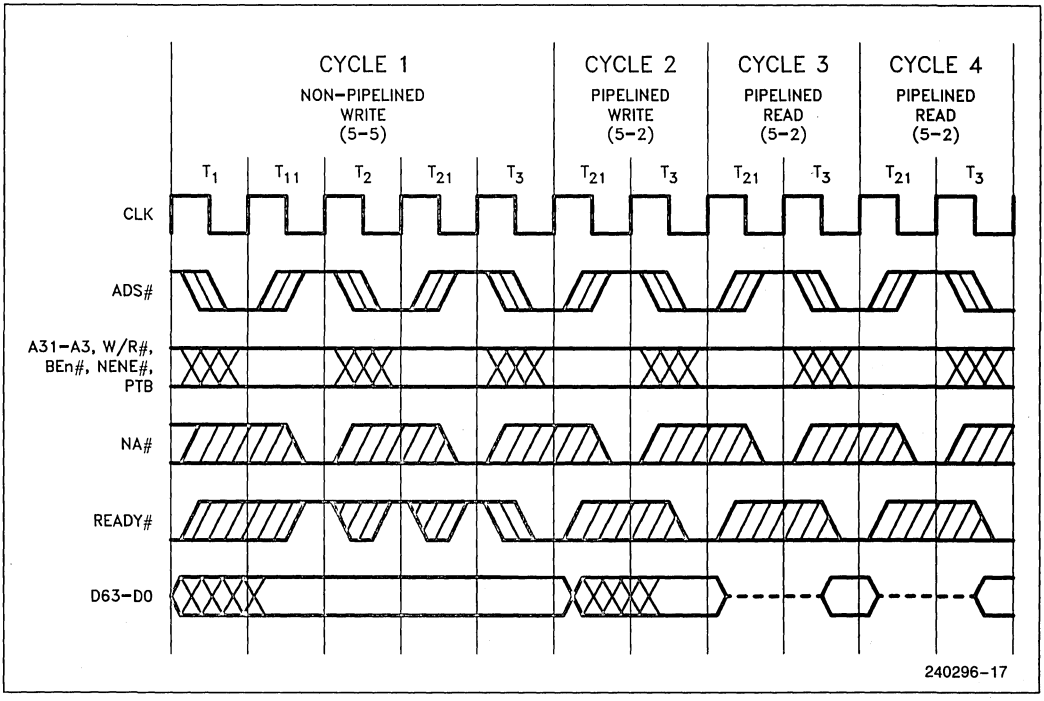


Figure 4.6. Pipelined Write Followed by Pipelined Read

4.3.3 PIPELINED READ AND WRITE CYCLES

Figures 4.5 and 4.6 illustrate combinations of non-pipelined and pipelined read and write cycles. The following description applies to both diagrams. While Cycle 1 is still in progress, two new cycles are initiated. By the time READY# first becomes active, the state machine has moved through states T_1, T_{11}, T_2, T_{21}, and T_3. Cycles 3 and 4 show how activating READY# terminates the corresponding outstanding cycle, and yet activating NA# while there is an internal request pending adds a new outstanding cycle.

In Figure 4.5, Cycle 3 is a write cycle following a read cycle; therefore, one wait state must be inserted. The i860 XR microprocessor does not drive the data bus until one clock after the read data is returned from the preceding read cycle. During Cycles 3 and 4, the state machine oscillates between states T_3 and T_{31} maintaining full bus capacity (two levels of pipelining; three outstanding cycles). Cycles 2, 3, and 4 in Figure 4.6 are 5-2 cycles; i.e. each requires a total cycle time of five clocks while the throughput rate is one cycle every two clocks.

Figure 4.7 illustrates in a more general manner how the NA# signal controls pipelining. Cycle 1 is a 2-2 cycle, the fastest possible. The next cycle cannot be started any earlier; therefore, there is no need to activate NA# to start the next cycle early. Cycle 2, a 3-3 read, is different. Cycle 3 can be started during the third state (a wait state) of Cycle 2, and NA# is asserted to accomplish this.

NA# is not activated following the ADS# clock of Cycle 3, thereby allowing Cycle 3 to terminate before the start of Cycle 4. As a result, Cycle 4 is a nonpipelined cycle.

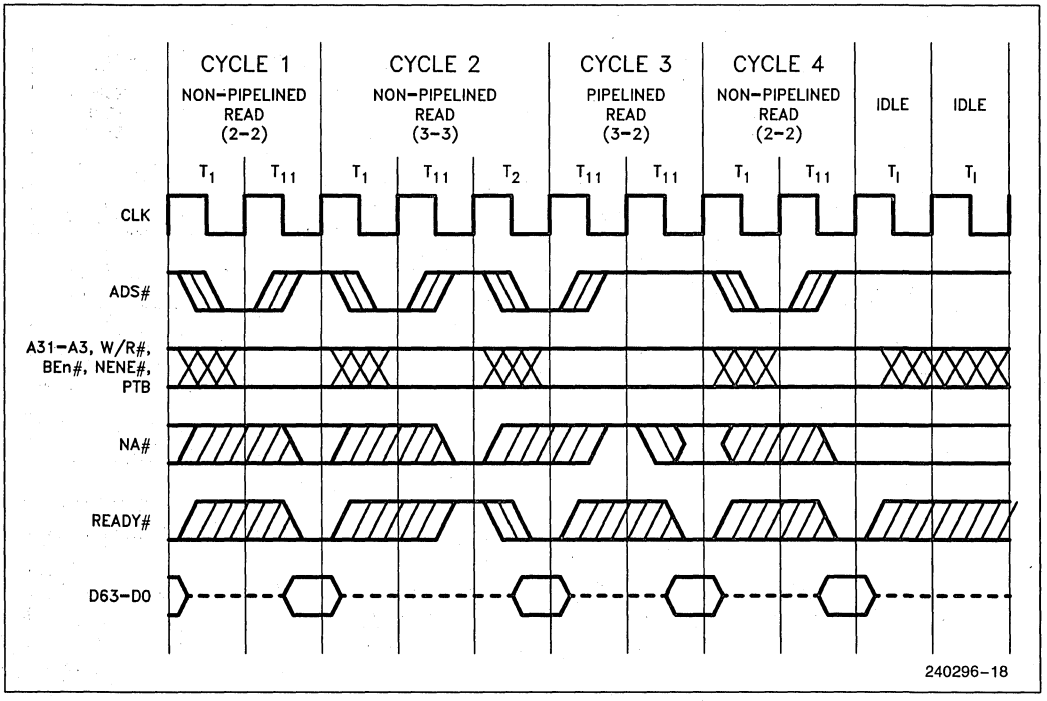


Figure 4.7. Pipelining Driven by NA#

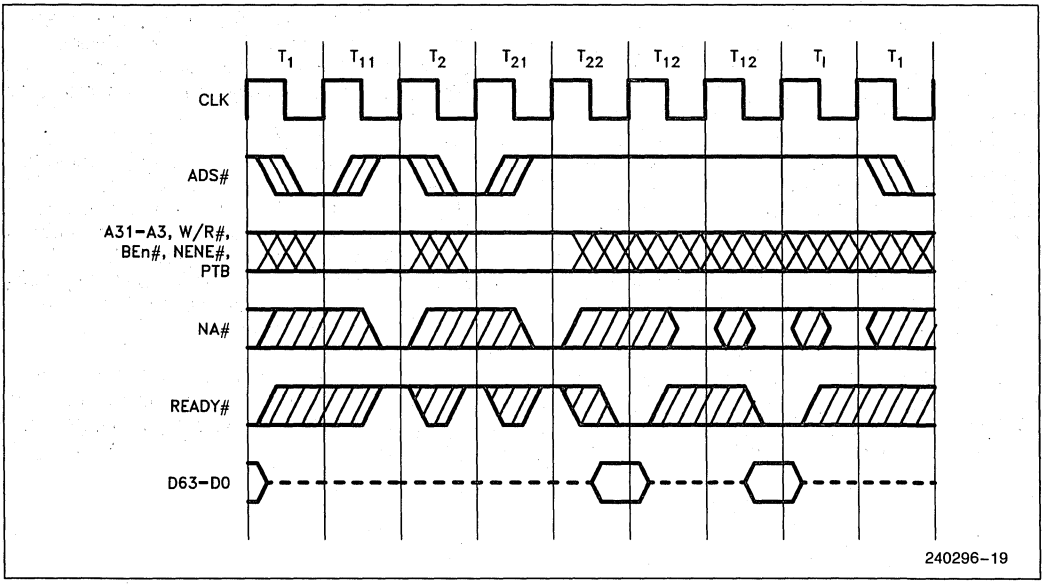


Figure 4.8. NA# Active with No Internal Bus Request

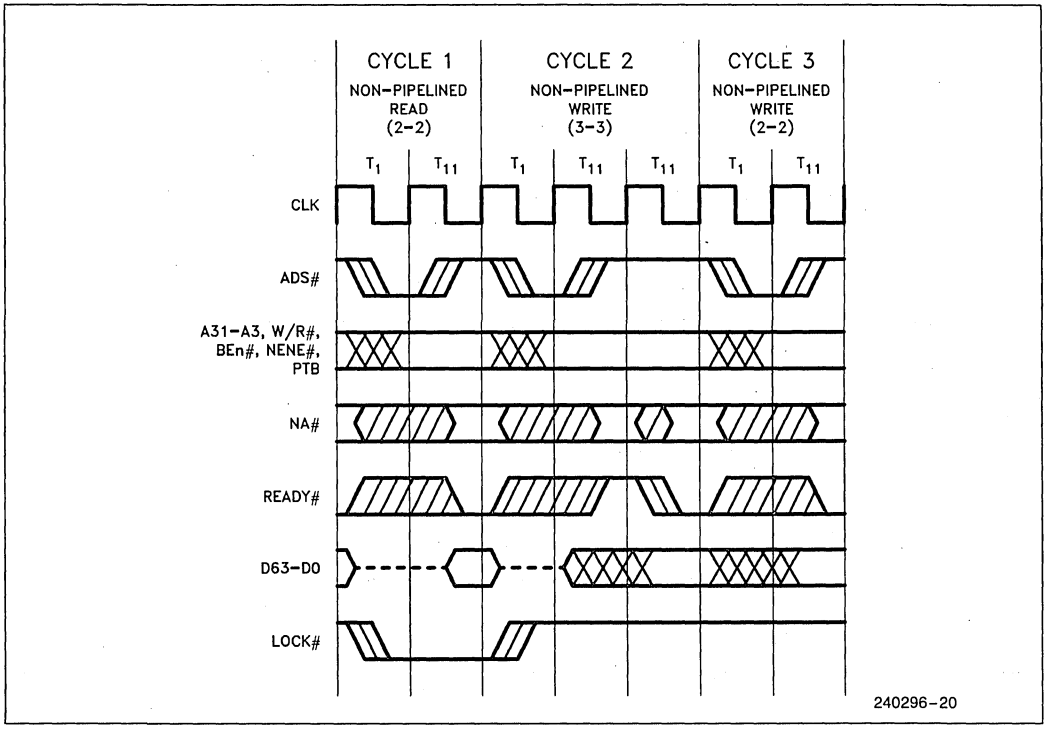


Figure 4.9. Locked Cycles

When there is no internal bus request, activating NA# does not start a new cycle; the i860 XR microprocessor, however, remembers that NA# has been activated. Figure 4.8 illustrates the situation where NA# is active but no internal bus request is pending. NA# is activated when two cycles are outstanding. Because there is no internal request pending until after one idle state, no new bus cycle is started during that period.

4.3.4 LOCKED CYCLES

The LOCK# signal is asserted when the current bus cycle is to be locked with the next bus cycle. Assertion of LOCK# may be initiated by a program's setting the BL bit of the **dirbase** register using the **lock** instruction (refer to section 2) or by the i860 XR microprocessor itself during page table updates.

In Figure 4.9, the first read cycle is to be locked with the following write cycle. If there were idle states between the cycles, the LOCK# signal would remain asserted. This is the case for a read/modify/write operation. Cycle 3 is not locked because LOCK# is no longer asserted when Cycle 2 starts.

4.3.5 HOLD AND BREQ ARBITRATION CYCLES

The HOLD, HLDA, and BREQ signals permit bus arbitration between the i860 XR microprocessor and another bus master.

See Figure 4.10. When HOLD is asserted, the i860 XR microprocessor does not relinquish control of the bus until all outstanding cycles are completed. If HOLD were asserted one clock earlier, the last i860 XR microprocessor bus cycle before HLDA would not be started.

HOLD is sampled at the end of the clock in which it is activated. Recommended setup and hold times must be met to guarantee sampling one clock after external HOLD activation. When HOLD is sampled active, a one clock delay for internal synchronization follows. Likewise when HOLD is deasserted, there is a one-clock delay for internal synchronization before HLDA is deasserted. The outputs (except HLDA and BREQ) float when HLDA is asserted.

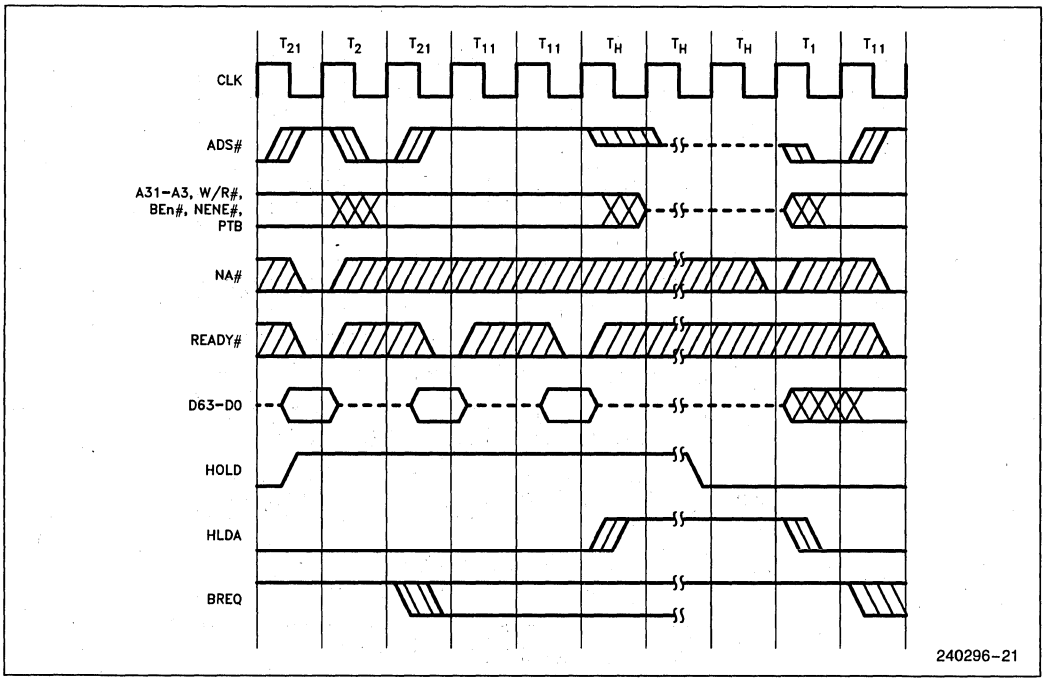


Figure 4.10. HOLD, HLDA, and BREQ

If, during a HOLD cycle, an internal bus request is generated, BREQ is activated even though HLDA is asserted. It remains active at least until the clock after ADS# is activated for the requested cycle.

4.4 Bus States During RESET

Figure 4.11 shows how INT/CS8 is sampled during the clock period just before the falling edge of RESET. If INT/CS8 is sampled active, the i860 XR microprocessor enters CS8 mode. No inputs (except for HOLD and INT/CS8) are sampled during RESET.

Note that, because HOLD is recognized even while RESET is active, the HLDA output signal may also become active during RESET. Refer to Table 3.4 "Output Pin Status during Reset".

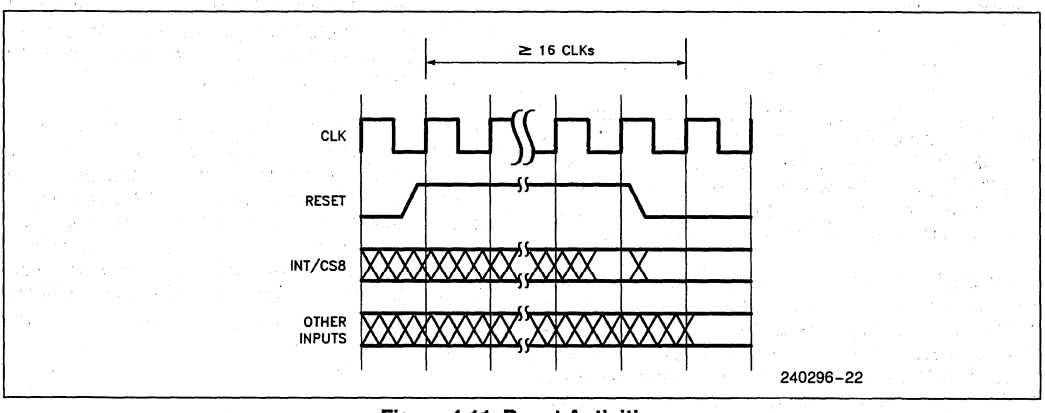


Figure 4.11. Reset Activities

5.0 MECHANICAL DATA

Figures 5.1 and 5.2 show the locations of pins; Tables 5.1 and 5.2 help to locate pin identifiers.

	S	R	Q	P	N	M	L	K	J	H	G	F	E	D	C	B	A	
1	( ) V_{CC}	( ) V_{SS}	( ) V_{CC}	( ) V_{SS}	( ) A12	( ) A17	( ) A19	( ) A21	( ) A23	( ) A25	( ) A29	( ) A31	( ) V_{CC}	( ) V_{SS}	( ) V_{CC}	( ) V_{SS}	( ) V_{CC}	1
2	( ) V_{SS}	( ) V_{CC}	( ) V_{SS}	( ) A8	( ) A10	( ) A13	( ) A15	( ) A18	( ) A20	( ) A24	( ) A27	( ) A28	( ) CC0	( ) V_{CC}	( ) V_{SS}	( ) V_{CC}	( ) V_{SS}	2
3	( ) V_{CC}	( ) V_{SS}	( ) A6	( ) A7	( ) A9	( ) A11	( ) A14	( ) A16	( ) CLK	( ) A22	( ) A26	( ) A30	( ) CC1	( ) D62	( ) D60	( ) V_{SS}	( ) V_{CC}	3
4	( ) V_{SS}	( ) V_{CC}	( ) A5												( ) D63	( ) D59	( ) V_{SS}	4
5	( ) V_{CC}	( ) A4	( ) A3												( ) D61	( ) D58	( ) D56	5
6	( ) W/R#	( ) NENE#	( ) PTB												( ) D57	( ) D54	( ) D52	6
7	( ) ADS#	( ) HLDA	( ) BREQ												( ) D55	( ) D53	( ) D50	7
8	( ) LOCK#	( ) KEN#	( ) READY#												( ) D51	( ) D49	( ) D48	8
9	( ) INT/CS8	( ) NA#	( ) HOLD												( ) D47	( ) D45	( ) D46	9
10	( ) BE5#	( ) BE7#	( ) BE6#												( ) D43	( ) D42	( ) D44	10
11	( ) BE3#	( ) BE2#	( ) BE4#												( ) D39	( ) D41	( ) D40	11
12	( ) SHI	( ) BE1#	( ) BE0#												( ) D37	( ) D36	( ) D38	12
13	( ) RESET	( ) SCAN	( ) BSCN												( ) D35	( ) D34	( ) V_{CC}	13
14	( ) V_{SS}	( ) D0	( ) D1												( ) D33	( ) V_{CC}	( ) V_{SS}	14
15	( ) V_{CC}	( ) V_{SS}	( ) D2	( ) D3	( ) D5	( ) D7	( ) D11	( ) D13	( ) D17	( ) D21	( ) D23	( ) D27	( ) D29	( ) D31	( ) D32	( ) V_{SS}	( ) V_{CC}	15
16	( ) V_{SS}	( ) V_{CC}	( ) V_{SS}	( ) V_{CC}	( ) D4	( ) D9	( ) D8	( ) D15	( ) D14	( ) D19	( ) D22	( ) D25	( ) D28	( ) D30	( ) V_{SS}	( ) V_{CC}	( ) V_{SS}	16
17	( ) V_{CC}	( ) V_{SS}	( ) V_{CC}	( ) V_{SS}	( ) V_{CC}	( ) D6	( ) D10	( ) D12	( ) D16	( ) D18	( ) D20	( ) D24	( ) D26	( ) V_{SS}	( ) V_{CC}	( ) V_{SS}	( ) V_{CC}	17
	S	R	Q	P	N	M	L	K	J	H	G	F	E	D	C	B	A	

240296-23

Figure 5.1. Pin Configuration—View from Top Side

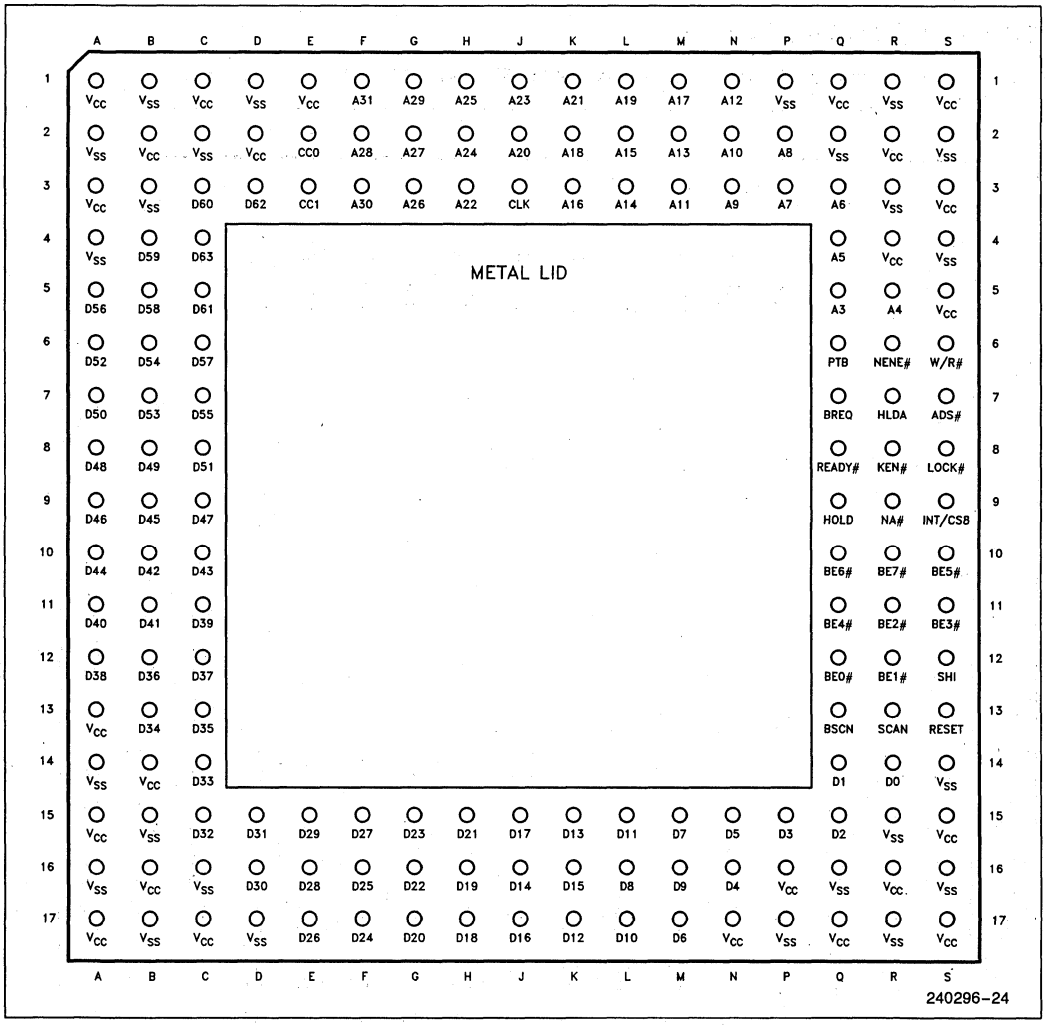


Figure 5.2. Pin Configuration—View from Pin Side

Table 5.1. Pin Cross Reference by Location

Location	Signal	Location	Signal	Location	Signal	Location	Signal
A1	V_{CC}	C9	D47	J15	D17	Q10	BE6#
A2	V_{SS}	C10	D43	J16	D14	Q11	BE4#
A3	V_{CC}	C11	D39	J17	D16	Q12	BE0#
A4	V_{SS}	C12	D37	K1	A21	Q13	BSCN
A5	D56	C13	D35	K2	A18	Q14	D1
A6	D52	C14	D33	K3	A16	Q15	D2
A7	D50	C15	D32	K15	D13	Q16	V_{SS}
A8	D48	C16	V_{SS}	K16	D15	Q17	V_{CC}
A9	D46	C17	V_{CC}	K17	D12	R1	V_{SS}
A10	D44	D1	V_{SS}	L1	A19	R2	V_{CC}
A11	D40	D2	V_{CC}	L2	A15	R3	V_{SS}
A12	D38	D3	D62	L3	A14	R4	V_{CC}
A13	V_{CC}	D15	D31	L15	D11	R5	A4
A14	V_{SS}	D16	D30	L16	D8	R6	NENE#
A15	V_{CC}	D17	V_{SS}	L17	D10	R7	HLDA
A16	V_{SS}	E1	V_{CC}	M1	A17	R8	KEN#
A17	V_{CC}	E2	CC0	M2	A13	R9	NA#
B1	V_{SS}	E3	CC1	M3	A11	R10	BE7#
B2	V_{CC}	E15	D29	M15	D7	R11	BE2#
B3	V_{SS}	E16	D28	M16	D9	R12	BE1#
B4	D59	E17	D26	M17	D6	R13	SCAN
B5	D58	F1	A31	N1	A12	R14	D0
B6	D54	F2	A28	N2	A10	R15	V_{SS}
B7	D53	F3	A30	N3	A9	R16	V_{CC}
B8	D49	F15	D27	N15	D5	R17	V_{SS}
B9	D45	F16	D25	N16	D4	S1	V_{CC}
B10	D42	F17	D24	N17	V_{CC}	S2	V_{SS}
B11	D41	G1	A29	P1	V_{SS}	S3	V_{CC}
B12	D36	G2	A27	P2	A8	S4	V_{SS}
B13	D34	G3	A26	P3	A7	S5	V_{CC}
B14	V_{CC}	G15	D23	P15	D3	S6	W/R#
B15	V_{SS}	G16	D22	P16	V_{CC}	S7	ADS#
B16	V_{CC}	G17	D20	P17	V_{SS}	S8	LOCK#
B17	V_{SS}	H1	A25	Q1	V_{CC}	S9	INT/CS8
C1	V_{CC}	H2	A24	Q2	V_{SS}	S10	BE5#
C2	V_{SS}	H3	A22	Q3	A6	S11	BE3#
C3	D60	H15	D21	Q4	A5	S12	SHI
C4	D63	H16	D19	Q5	A3	S13	RESET
C5	D61	H17	D18	Q6	PTB	S14	V_{SS}
C6	D57	J1	A23	Q7	BREQ	S15	V_{CC}
C7	D55	J2	A20	Q8	READY#	S16	V_{SS}
C8	D51	J3	CLK	Q9	HOLD	S17	V_{CC}

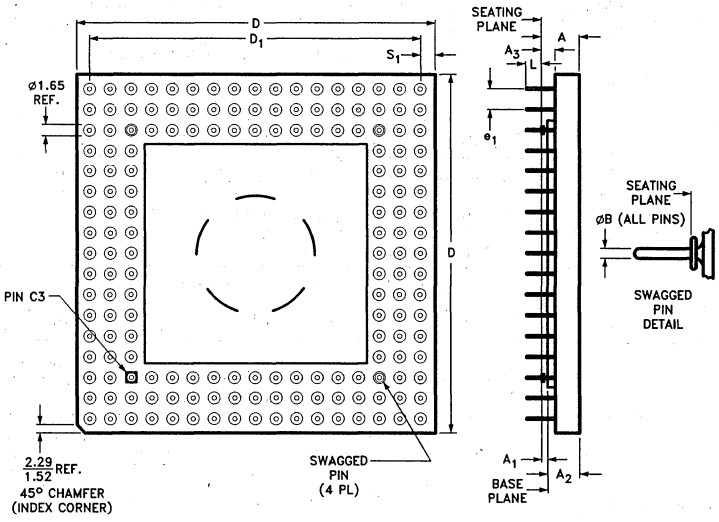
Table 5.2. Pin Cross Reference by Pin Name

Signal	Location	Signal	Location	Signal	Location	Signal	Location
A3	Q5	CLK	J3	D41	B11	V_{CC}	B16
A4	R5	D0	R14	D42	B10	V_{CC}	C1
A5	Q4	D1	Q14	D43	C10	V_{CC}	C17
A6	Q3	D2	Q15	D44	A10	V_{CC}	D2
A7	P3	D3	P15	D45	B9	V_{CC}	E1
A8	P2	D4	N16	D46	A9	V_{CC}	N17
A9	N3	D5	N15	D47	C9	V_{CC}	P16
A10	N2	D6	M17	D48	A8	V_{CC}	Q1
A11	M3	D7	M15	D49	B8	V_{CC}	Q17
A12	N1	D8	L16	D50	A7	V_{CC}	R2
A13	M2	D9	M16	D51	C8	V_{CC}	R4
A14	L3	D10	L17	D52	A6	V_{CC}	R16
A15	L2	D11	L15	D53	B7	V_{CC}	S1
A16	K3	D12	K17	D54	B6	V_{CC}	S3
A17	M1	D13	K15	D55	C7	V_{CC}	S5
A18	K2	D14	J16	D56	A5	V_{CC}	S15
A19	L1	D15	K16	D57	C6	V_{CC}	S17
A20	J2	D16	J17	D58	B5	V_{SS}	A2
A21	K1	D17	J15	D59	B4	V_{SS}	A4
A22	H3	D18	H17	D60	C3	V_{SS}	A14
A23	J1	D19	H16	D61	C5	V_{SS}	A16
A24	H2	D20	G17	D62	D3	V_{SS}	B1
A25	H1	D21	H15	D63	C4	V_{SS}	B3
A26	G3	D22	G16	HLDA	R7	V_{SS}	B15
A27	G2	D23	G15	HOLD	Q9	V_{SS}	B17
A28	F2	D24	F17	INT/CS8	S9	V_{SS}	C2
A29	G1	D25	F16	KEN#	R8	V_{SS}	C16
A30	F3	D26	E17	LOCK#	S8	V_{SS}	D1
A31	F1	D27	F15	NA#	R9	V_{SS}	D17
ADS#	S7	D28	E16	NENE#	R6	V_{SS}	P1
BE0#	Q12	D29	E15	PTB	Q6	V_{SS}	P17
BE1#	R12	D30	D16	READY#	Q8	V_{SS}	Q2
BE2#	R11	D31	D15	RESET	S13	V_{SS}	Q16
BE3#	S11	D32	C15	SCAN	R13	V_{SS}	R1
BE4#	Q11	D33	C14	SHI	S12	V_{SS}	R3
BE5#	S10	D34	B13	V_{CC}	A1	V_{SS}	R15
BE6#	Q10	D35	C13	V_{CC}	A3	V_{SS}	R17
BE7#	R10	D36	B12	V_{CC}	A13	V_{SS}	S2
BREQ	Q7	D37	C12	V_{CC}	A15	V_{SS}	S4
BSCN	Q13	D38	A12	V_{CC}	A17	V_{SS}	S14
CC0	E2	D39	C11	V_{CC}	B2	V_{SS}	S16
CC1	E3	D40	A11	V_{CC}	B14	W/R#	S6

Table 5.3. Ceramic PGA Package Dimension Symbols

Letter or Symbol	Description of Dimensions
A	Distance from seating plane to highest point of body
A_1	Distance between seating plane and base plane (lid)
A_2	Distance from base plane to highest point of body
A_3	Distance from seating plane to bottom of body
B	Diameter of terminal lead pin
D	Largest overall package dimension of length
D_1	A body length dimension, outer lead center to outer lead center
e_1	Linear spacing between true lead position centerlines
L	Distance from seating plane to end of lead
S_1	Other body dimension, outer lead center to edge of body

NOTES:
1. Controlling dimension: millimeter.
2. Dimension "e_1" ("e") is non-cumulative.
3. Seating plane (standoff) is defined by P.C. board hole size: 0.0415–0.0430 inch.
4. Dimensions "B", "B_1" and "C" are nominal.
5. Details of Pin 1 identifier are optional.



Family: Ceramic Pin Grid Array Package						
Symbol	Millimeters			Inches		
	Min	Max	Notes	Min	Max	Notes
A	3.56	4.57		0.140	0.180	
A_1	0.64	1.14	SOLID LID	0.025	0.045	SOLID LID
A_2	2.79	3.56	SOLID LID	0.110	0.140	SOLID LID
A_3	1.14	1.40		0.045	0.055	
B	0.43	0.51		0.017	0.020	
D	44.07	44.83		1.735	1.765	
D_1	40.51	40.77		1.595	1.605	
e_1	2.29	2.79		0.090	0.110	
L	2.54	3.30		0.100	0.130	
N	168		# of Pins	168		# of Pins
S_1	1.52	2.54		0.060	0.100	
ISSUE	IWS REV X 7/15/88					

Figure 5.3. 168 Lead Ceramic PGA Package Dimensions

6.0 PACKAGE THERMAL SPECIFICATIONS

For this section, let:

P = maximum power consumption

T_C = case temperature

T_A = ambient air temperature

θ_{CA} = thermal resistance from case to ambient air

θ_{JC} = thermal resistance from junction to case

θ_{JA} = thermal resistance from junction to ambient air

The i860 XR microprocessor is specified for operation when T_C is within the range of 0°C–85°C. T_C may be measured in any environment to determine whether the i860 XR microprocessor is within specified operating range. The case temperature should be measured at the center of the top surface opposite the pins.

T_A can be calculated from θ_{CA} (thermal resistance from case to ambient) with the following equation:

$$T_A = T_C - P^*\theta_{CA}$$

Typical values for θ_{CA} and θ_{JC} at various airflows are given in Table 6.1 for the 1.75 sq. in., 168 pin, ceramic PGA. θ_{JC} is also shown so that θ_{JA} can be calculated by:

$$\theta_{CA} = \theta_{JA} - \theta_{JC}$$

Note that θ_{JC} with a heatsink differs from θ_{JC} without a heatsink because case temperature is measured differently. Case temperature for θ_{JC} with heatsink is measured at the center of the heat fin base. Case temperature for θ_{JC} without heatsink is measured at the center of package top surface.

Table 6.2 shows the maximum T_A allowable (without exceeding T_C) at various airflows and operating frequencies (f_{CLK}).

Note that T_A is greatly improved by attaching "fins" or a "heat sink" to the package. P (the maximum power consumption) is calculated by using the maximum I_{CC} at 5V as tabulated in the *DC Characteristics* of section 7.

Figure 6.1 gives typical I_{CC} derating with case temperature. For more information on heat sinks, measurement techniques, or package characteristics, refer to *Intel Packaging Handbook*, order number 240800.

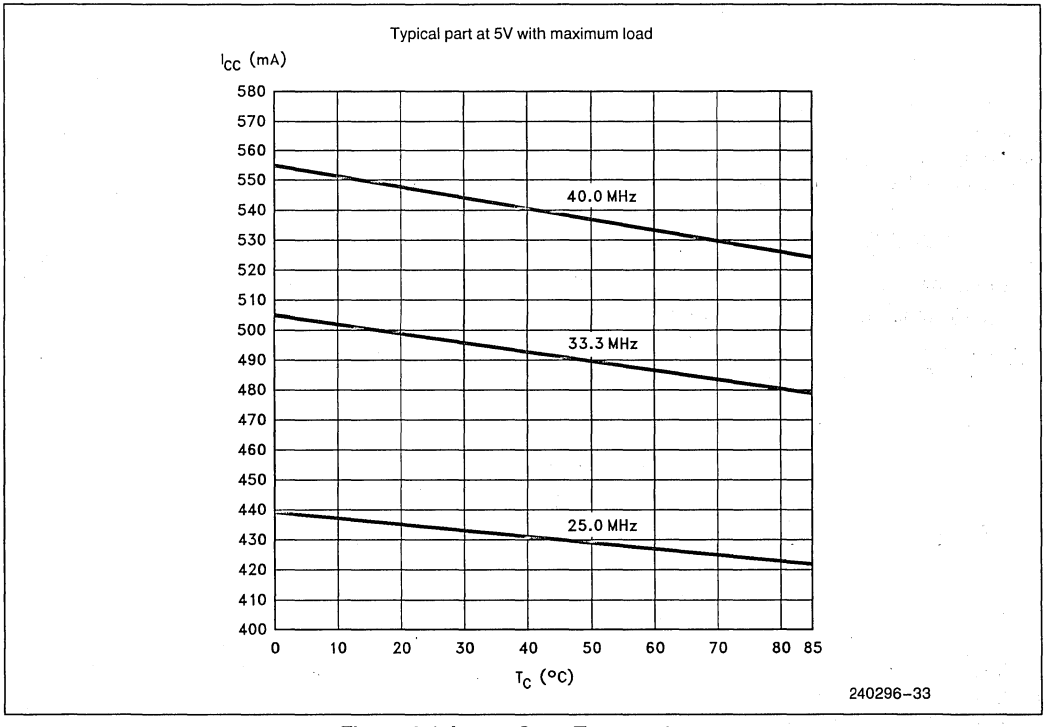


Figure 6.1. I_{CC} vs Case Temperature

Table 6.1. Thermal Resistance (°C/W) θ_{JC} and θ_{CA}

	θ_{JC}	θ_{CA} at Airflow-ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
With Heat Sink*	2	11	6	4	3.2	2.5	2.2
Without Heat Sink	1.5	17.5	13	11	9.5	8.5	8

*Nine-fin, unidirectional heat sink (fin dimensions: 0.350" height, 0.040 width, 0.115" center-to-center spacing, 1.530" length).

Table 6.2. Maximum Allowable T_A at Various Airflows

In °C

	f_{CLK} (MHz)	Airflow-ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
T_A with Heat Sink*	25.0	57.5	70	75	77	78.8	79.5
	33.3	52	67	73	75.5	77.4	78.5
	40.0	49.3	65.5	72	74.6	76.9	77.9
T_A without Heat Sink	25.0	41.3	52.5	57.5	61.3	63.8	65
	33.3	32.5	46	52	56.5	59.5	61
	40.0	28.1	42.8	49.3	54.1	57.4	59

*Nine-fin unidirectional heat sink (fin dimensions: 0.350″ height, 0.040 width, 0.115″ center-to-center spacing, 1.530″ length).

7.0 ELECTRICAL DATA

Inputs and outputs are TTL compatible, except for CLK. All input and output timings are specified relative to the 1.5 volt level of the rising edge of CLK and refer to the point that the signals reach 1.5V.

NOTICE: This data sheet contains preliminary information on new products in production. The specifications are subject to change without notice. Verify with your local Intel Sales office that you have the latest data sheet before finalizing a design.

**WARNING: Stressing the device beyond the "Absolute Maximum Ratings" may cause permanent damage. These are stress ratings only. Operation beyond the "Operating Conditions" is not recommended and extended exposure beyond the "Operating Conditions" may affect device reliability.*

7.1 Absolute Maximum Ratings

Case Temperature T_C under Bias .....0°C to 85°C

Storage Temperature .....−65°C to +150°C

Voltage on Any Pin
with Respect to Ground.....−0.5 to 6.5V

7.2 D.C. Characteristics

Table 7.1. DC Characteristics

T_C = 0°C to 85°C, V_{CC} = 5V ±5%

Symbol	Parameter	Min	Max	Units	Notes
V_{IL}	Input LOW Voltage	−0.3	+0.8	V	
V_{IH}	Input HIGH Voltage	2.0	V_{CC}+0.3	V	
V_{ILC}	CLK Input LOW Voltage	−0.3	+0.8	V	
V_{IHC}	CLK Input HIGH Voltage	3.0	V_{CC} + 0.3	V	
V_{OL}	Output LOW Voltage		0.45	V	(Note 1)
V_{OH}	Output HIGH Voltage	2.4		V	(Note 2)
I_{CC}	Power Supply Current				
	CLK = 25.0 MHz		500	mA	V_{CC} @5V
	CLK = 33.3 MHz		600	mA	V_{CC} @5V
	CLK = 40.0 MHz		650	mA	V_{CC} @5V
I_{LI}	Input Leakage Current		±15	μA	No pullup or pulldown
I_{LO}	Output Leakage Current		±15	μA	
C_{IN}	Input Capacitance		15	pF	(Note 3)
C_O	I/O or Output Capacitance		15	pF	(Note 3)
C_{CLK}	Clock Capacitance		20	pF	(Note 3)

NOTES:
1. This parameter is measured at 4.0 mA for A31–A3, D63–D0, BE7#–BE0#; at 5.0 mA for all other outputs.
2. This parameter is measured at 1.0 mA for A31–A3, D63–D0, BE7#–BE0#; at 0.9 mA all other outputs.
3. These are not tested. They are guaranteed by design characterization.

7.3 A.C. Characteristics

Table 7.2. A.C. Characteristics

T_C = 0°C to 85°C, V_{CC} = 5V ±5%

All timings measured at CLK = 1.5V unless otherwise specified.

Symbol	Parameter	25 MHz		33 MHz		40 MHz		Notes
		Min (ns)	Max (ns)	Min (ns)	Max (ns)	Min (ns)	Max (ns)	
t1	CLK Period	40	125	30	125	25	125	
t2	CLK High Time	6		5		3		,at 3V
t3	CLK Low Time	8		7		5		at 0.8V
t4	CLK Fall Time		7		7		7	3V–0.8V
t5	CLK Rise Time		7		7		7	0.8V–3V
t6a	A31–A3, PTB, W/R#, NENE# Valid Delay	3.5	25	3.5	23	3.5	19	50 pF Load
t6b	BEn#* Valid Delay	3.5	27	3.5	25	3.5	21	50 pF Load
t7	Float Time, All	3.5	40	3.5	30	3.5	25	(Note 1)
t8	ADS#, BREQ, LOCK#, HLDA Valid Delay	3.5	22	3.5	20	3.5	15	50 pF Load
t9	D63–D0 Valid Delay	3.5	38	3.5	35	3.5	31	50 pF Load
t10	Setup Time, All Inputs	13		11		8		(Note 2)
t11a	Hold Time, All Inputs except DATA	4		4		3		(Note 2)
t11b	DATA Hold Time	5		4		3		

NOTES:

1. Float condition occurs when maximum output current becomes less than I_{LO} in magnitude. Float delay is not tested.
2. INT and HOLD are asynchronous inputs. The setup and hold specifications are given for test purposes or to assure recognition on a specific rising edge of CLK.

* n = 0, 1, ..., 7

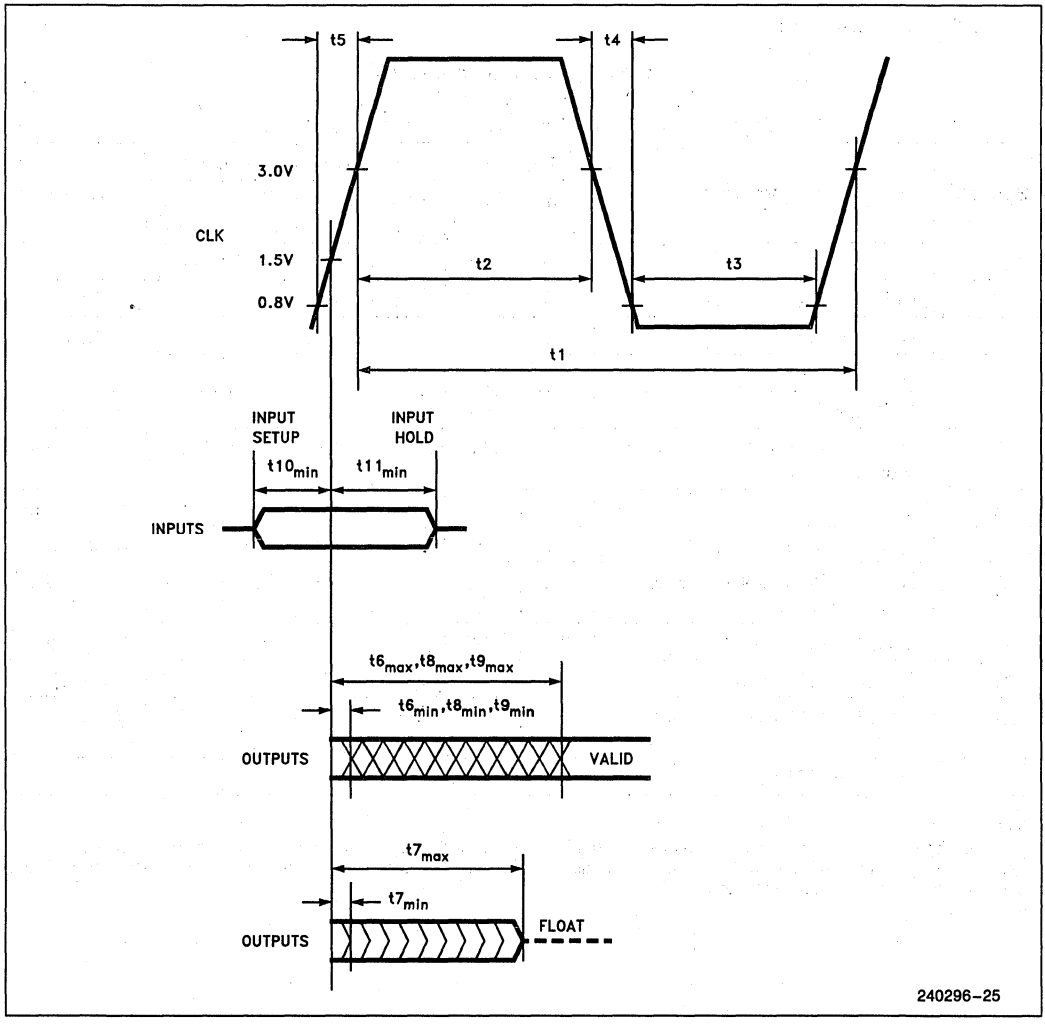
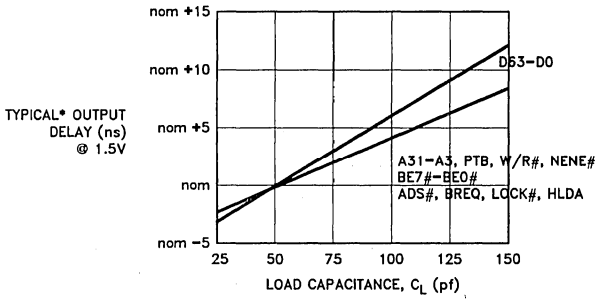


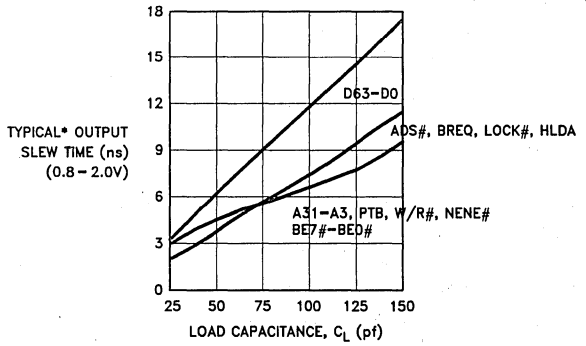
Figure 7.1. CLK, Input, and Output Timings



240296-26

NOTES:
Graphs are not linear outside the C_L range shown.
nom = nominal value given in the AC timing table.
*Typical part under worst-case conditions.

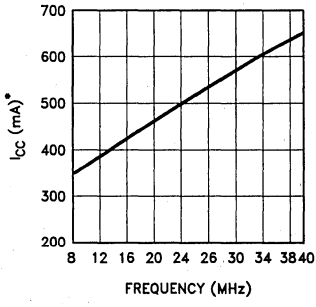
Figure 7.2. Typical Output Delay vs Load Capacitance under Worst-Case Conditions



240296-27

NOTES:
Graphs are not linear outside the C_L range shown.
*Typical part under worst-case conditions.

Figure 7.3. Typical Slew Time vs Load Capacitance under Worst-Case Conditions



240296-28

NOTES:
Graphs are not linear outside the frequency range shown.
*Worst-case supply current at 5V.

Figure 7.4. Typical I_{CC} vs Frequency

8.0 INSTRUCTION SET

Key to abbreviations:

For register operands, the abbreviations that describe the operands are composed of two parts. The first part describes the type of register:

c — One of the control registers **fir, psr, epsr, dirbase, db,** or **fsr**

f — One of the floating-point registers: **f0** through **f31**

i — One of the integer registers: **r0** through **r31**

The second part identifies the field of the machine instruction into which the operand is to be placed:

src1 — The first of the two source-register designators, which may be either a register or a 16-bit immediate constant or address offset. The immediate value is zero-extended for logical operations and is sign-extended for add and subtract operations (including **addu** and **subu**) and for all addressing calculations.

src1ni — Same as *src1* except that no immediate constant or address offset value is permitted.

src1s — Same as *src1* except that the immediate constant is a 5-bit value that is zero-extended to 32 bits.

src2 — The second of the two source-register designators.

dest — The destination register designator.

Thus, the operand specifier *isrc2*, for example, means that an integer register is used and that the encoding of that register must be placed in the *src2* field of the machine instruction.

Other (nonregister) operands are specified by a one-part abbreviation that represents both the type of operand required and the instruction field into which the value of the operand is placed:

#const — A 16-bit immediate constant or address offset that the i860 XR microprocessor sign-extends to 32 bits when computing the effective address.

lbroff — A signed, 26-bit, immediate, relative branch offset.

sbroff — A signed, 16-bit, immediate, relative branch offset.

brx — A function that computes the target address by shifting the offset (either *lbroff* or *sbroff*) left by two bits, sign-extending it to 32 bits, and adding the result to the current instruction pointer plus four. The resulting target address may lie anywhere within the address space.

Unless otherwise specified, floating-point operations accept single- or double-precision source operands and produce a result of equal or greater precision. Both input operands must have the same precision. The source and result precision are specified by a two-letter suffix to the mnemonic of the operation.

Other abbreviations include:

.p — Precision specification **.ss, .sd,** or **.dd** (**.ds** not permitted). Refer to Table 8.1.

.r — Precision specification **.ss, .sd, .ds,** or **.dd**. Refer to Table 8.1.

.v — **.sd** or **.dd**. Refer to Table 8.1.

.w — **.ss** or **.dd.** Refer to Table 8.1.

.x — **.b** (8 bits), **.s** (16 bits), or **.l** (32 bits)

.y — **.l** (32 bits), **.d** (64 bits), or **.q** (128 bits)

.z — **.l** (32 bits), or **.d** (64 bits)

Table 8.1. Precision Specification

Suffix	Source Precision	Result Precision
.ss	single	single
.sd	single	double
.dd	double	double
.ds	double	single

mem.x(address) The contents of the memory location indicated by *address* with a size of *x*.

PM The pixel mask, which is considered as an array of eight bits PM[7]..PM[0], where PM[0] is the least significant bit.

8.1 Instruction Definitions in Alphabetical Order

adds *isrc1, isrc2, idest* ..... **Add Signed**
idest ← *isrc1* + *isrc2*
OF ← (bit 31 carry ≠ bit 30 carry)
CC set if *isrc2* < −*isrc1* (signed)
CC clear if *isrc2* ≥ −*isrc1* (signed)

addu *isrc1, isrc2, idest* ..... **Add Unsigned**
idest ← *isrc1* + *isrc2*
OF ← bit 31 carry
CC ← bit 31 carry

and *isrc1, isrc2, idest* ..... **Logical AND**
idest ← *isrc1* and *isrc2*
CC set if result is zero, cleared otherwise

andh *#const, isrc2, idest* ..... **Logical AND High**
idest ← (*#const* shifted left 16 bits) and *isrc2*
CC set if result is zero, cleared otherwise

andnot *isrc1, isrc2, idest* ..... **Logical AND NOT**
idest ← not *isrc1* and *isrc2*
CC set if result is zero, cleared otherwise

andnoth *#const, isrc2, idest* ..... **Logical AND NOT High**
idest ← not (*#const* shifted left 16 bits) and *isrc2*
CC set if result is zero, cleared otherwise

bc *lbroff* ..... **Branch on CC**
IF CC = 1
THEN continue execution at *brx(lbroff)*
FI

bc.t *lbroff* ..... **Branch on CC, Taken**
IF CC = 1
THEN execute one more sequential instruction
continue execution at *brx(lbroff)*
ELSE skip next sequential instruction
FI

bla *isrc1ni, isrc2, sbroff* ..... **Branch on LCC and Add**
LCC-temp clear if *isrc2* < −*isrc1ni* (signed)
LCC-temp set if *isrc2* ≥ −*isrc1ni* (signed)
isrc2 ← *isrc1ni* + *isrc2*
Execute one more sequential instruction
IF LCC
THEN LCC ← LCC-temp
continue execution at *brx(sbroff)*
ELSE LCC ← LCC-temp
FI

bnc *lbroff* ..... **Branch on Not CC**
IF CC = 0
THEN continue execution at *brx(lbroff)*
FI

bnc.t *lbroff* ..... **Branch on Not CC, Taken**
IF CC = 0
THEN execute one more sequential instruction
continue execution at *brx(lbroff)*
ELSE skip next sequential instruction
FI

br *lbroff* ..... **Branch Direct Unconditionally**

Execute one more sequential instruction.
Continue execution at *brx(lbroff)*.

bri [*isrc1ni*] ..... **Branch Indirect Unconditionally**

```
Execute one more sequential instruction
IF      any trap bit in psr is set
THEN    copy PU to U, PIM to IM in psr
        clear trap bits
        IF      DS is set and DIM is reset
        THEN    enter dual-instruction mode after executing one
                  instruction in single-instruction mode
        ELSE    IF      DS is set and DIM is set
                THEN    enter single-instruction mode after executing one
                          instruction in dual-instruction mode
                ELSE    IF      DIM is set
                        THEN    enter dual-instruction mode
                                  for next two instructions
                        ELSE    enter single-instruction mode
                                  for next two instructions
                        FI
                FI
        FI
FI
```

Continue execution at address in *isrc1ni*
(The original contents of *isrc1ni* is used even if the next instruction modifies *isrc1ni*. Does not trap if *isrc1ni* is misaligned.)

bte *isrc1s, isrc2, sbroff* ..... **Branch If Equal**

```
IF      isrc1s = isrc2
THEN    continue execution at brx(sbroff)
FI
```

btne *isrc1s, isrc2, sbroff* ..... **Branch If Not Equal**

```
IF      isrc1s ≠ isrc2
THEN    continue execution at brx(sbroff)
FI
```

call *lbroff* ..... **Subroutine Call**

r1 ← address of next sequential instruction + 4 (+8 in dual mode)
Execute one more sequential instruction
Continue execution at *brx(lbroff)*

calli [*isrc1ni*] ..... **Indirect Subroutine Call**

r1 ← address of next sequential instruction + 4 (+8 in dual mode)
Execute one more sequential instruction
Continue execution at address in *isrc1ni*
(The original contents of *isrc1ni* is used even if the next instruction modifies *isrc1ni*. Does not trap if *isrc1ni* is misaligned.
The register *isrc1ni* must not be **r1**.)

fadd.p *fsrc1, fsrc2, fdest* ..... **Floating-Point Add**

fdest ← *fsrc1* + *fsrc2*

faddp *fsrc1, fsrc2, fdest* ..... **Add with Pixel Merge**

fdest ← *fsrc1* + *fsrc2*
Shift and load MERGE register as defined in Table 8.2

faddz *fsrc1, fsrc2, fdest* ..... **Add with Z Merge**

fdest ← *fsrc1* + *fsrc2*
Shift MERGE right 16 and load fields 31..16 and 63..48

famov.r *fsrc1, fdest* ..... **Floating-Point Adder Move**

fdest ← *fsrc1*
Send *fsrc1* through the floating-point adder. (Preserves −0 (minus zero) when *fsrc1 is −0. fsrc2* must be coded as **f0** by the assembler.)

fiadd.w *fsrc1, fsrc2, fdest* ..... **Long-Integer Add**
fdest ← *fsrc1* + *fsrc2*

fisub.w *fsrc1, fsrc2, fdest* ..... **Long-Integer Subtract**
fdest ← *fsrc1* − *fsrc2*

fix.v *fsrc1, fdest* ..... **Floating-Point to Integer Conversion**
fdest ← 64- bit value with low-order 32 bits equal to integer part of *fsrc1* rounded

Floating-Point Load

fld.y *isrc1(isrc2), fdest* ..... **(Normal)**
fld.y *isrc1(isrc2)*++, *fdest* ..... **(Autoincrement)**
fdest ← mem.y (*isrc1* + *isrc2*)
IF autoincrement
THEN *isrc2* ← *isrc1* + *isrc2*
FI

Cache Flush

flush *#const(isrc2)* ..... **(Normal)**
flush *#const(isrc2)*++ ..... **(Autoincrement)**
Replace block in data cache with address (*#const* + *isrc2*).
Contents of block undefined.
IF autoincrement
THEN *isrc2* ← *#const* + *isrc2*
FI

fmlow.dd *fsrc1, fsrc2, fdest* ..... **Floating-Point Multiply Low**
fdest ← low-order 53 bits of *fsrc1* mantissa × *fsrc2* mantissa
fdest bit 53 ← most significant bit of mantissa

fmov.r *fsrc1, fdest* ..... **Floating-Point Reg-Reg Move**
Assembler pseudo-operation
fmov.ss *fsrc1, fdest* = **fiadd.ss** *fsrc1*, **f0,** *fdest*
fmov.dd *fsrc1, fdest* = **fiadd.dd** *fsrc1*, **f0,** *fdest*
fmov.sd *fsrc1, fdest* = **famov.sd** *fsrc1*, *fdest*
fmov.ds *fsrc1, fdest* = **famov.ds** *fsrc1*, *fdest*

fmul.p *fsrc1, fsrc2, fdest* ..... **Floating-Point Multiply**
fdest ← *fsrc1* × *fsrc2*

fnop ..... **Floating-Point No Operation**
Assembler pseudo-operation
fnop = **shrd r0, r0, r0**

form *fsrc1, fdest* ..... **OR with MERGE Register**
fdest ← *fsrc1* OR MERGE
MERGE ← 0

frcp.p *fsrc2, fdest* ..... **Floating-Point Reciprocal**
fdest ← 1/*fsrc2* with maximum mantissa error < 2^{-7}

frsqr.p *fsrc2, fdest* ..... **Floating-Point Reciprocal Square Root**
fdest ← 1/SQRT (*fsrc2*) with maximum mantissa error < 2^{-7}

Floating-Point Store

fst.y *fdest, isrc1(isrc2)* ..... **(Normal)**
fst.y *fdest, isrc1(isrc2)*++ ..... **(Autoincrement)**
mem.y (*isrc2* + *isrc1*) ← *fdest*
IF autoincrement
THEN *isrc2* ← *isrc1* + *isrc2*
FI

fsub.p *fsrc1, fsrc2, fdest* ..... **Floating-Point Subtract**
fdest ← *fsrc1* − *fsrc2*

ftrunc.v *fsrc1, fdest* ..... **Floating-Point to Integer Conversion**
fdest ← 64-bit value with low-order 32 bits equal to integer part of *fsrc1*

fxfr *fsrc1, idest* ..... **Transfer F-P to Integer Register**
idest ← *fsrc1*

fzchkl *fsrc1, fsrc2, fdest* ..... **32-Bit Z-Buffer Check**

Consider *fsrc1, fsrc2,* and *fdest* as arrays of two 32-bit
fields *fsrc1*(0)..*fsrc1*(1), *fsrc2*(0)..*fsrc2*(1), and *fdest*(0)..*fdest*(1)
where zero denotes the least-significant field.
PM ← PM shifted right by 2 bits
FOR i = 0 to 1
DO
PM [i + 6] ← *fsrc2*(i) ≤ *fsrc1*(i) (unsigned)
fdest(i) ← smaller of *fsrc2*(i) and *fsrc1*(i)
OD
MERGE ← 0

fzchks *fsrc1, fsrc2, fdest* ..... **16-Bit Z-Buffer Check**

Consider *fsrc1, fsrc2,* and *fdest* as arrays of four 16-bit
fields *fsrc1*(0)..*fsrc1*(3), *fsrc2*(0)..*fsrc2*(3), and *fdest*(0)..*fdest*(3)
where zero denotes the least-significant field.
PM ← PM shifted right by 4 bits
FOR i = 0 to 3
DO
PM [i + 4] ← *fsrc2*(i) ≤ *fsrc1*(i) (unsigned)
fdest(i) ← smaller of *fsrc2*(i) and *fsrc1*(i)
OD
MERGE ← 0

intovr ..... **Software Trap on Integer Overflow**

If OF in **epsr** = 1, generate trap with IT set in **psr**.

ixfr *isrc1ni, fdest* ..... **Transfer Integer to F-P Register**

fdest ← *isrc1ni*

ld.c *csrc2, idest* ..... **Load from Control Register**

idest ← *csrc2*

ld.x *isrc1*(*isrc2*), *idest* ..... **Load Integer**

idest ← *mem.x* (*isrc1* + *isrc2*)

lock ..... **Begin Interlocked Sequence**

Set BL in **dirbase**. The next load or store that misses the cache locks that location.
Disable interrupts until the bus is unlocked.

mov *isrc2, idest* ..... **Register-Register Move**

Assembler pseudo-operation
mov *isrc2, idest* = **shl r0,** *isrc2, idest*

mov *const32, idest* ..... **Constant-to-Register Move**

Assembler pseudo-operation
adds *l%const32*, **r0**, *idest*
... when *const32* < 0x8000

orh *h%const32*, **r0**, *idest*
or *l%const32, idest, idest*
... when *const32* ≥ 0x8000

nop ..... **Core-Unit No Operation**

Assembler pseudo-operation
nop = **shl r0, r0, r0**

or *isrc1, isrc2, idest* ..... **Logical OR**

idest ← *isrc1* OR *isrc2*
CC set if result is zero, cleared otherwise

orh *#const, isrc2, idest* ..... **Logical OR High**

idest ← (*#const* shifted left 16 bits) OR *isrc2*
CC set if result is zero, cleared otherwise

pfadd.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Add**
fdest ← last stage Adder result
Advance A pipeline one stage
A pipeline first stage ← *fsrc1* + *fsrc2*

pfaddp *fsrc1, fsrc2, fdest* ..... **Pipelined Add with Pixel Merge**
fdest ← last stage Graphics result
last stage Graphics result ← *fsrc1* + *fsrc2*
Shift and load MERGE register from last stage Graphics result as defined in Table 8.2

pfaddz *fsrc1, fsrc2, fdest* ..... **Pipelined Add with Z Merge**
fdest ← last stage Graphics result
last stage Graphics result ← *fsrc1* + *fsrc2*
Shift MERGE right 16 and load fields 31..16 and 63..48 from last stage Graphics result

pfam.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Add and Multiply**
fdest ← last stage Adder result
Advance A and M pipeline one stage (operands accessed before advancing pipeline)
A pipeline first stage ← A-op1 + A-op2
M pipeline first stage ← M-op1 × M-op2

pfamov.r *fsrc1, fdest* ..... **Pipelined Floating-Point Adder Move**
fdest ← last stage Adder result
Advance A pipeline one stage
A pipeline first stage ← *fsrc1*

pfeq.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Equal Compare**
fdest ← last stage Adder result
CC set if *fsrc1* = *fsrc2*, else cleared
Advance A pipeline one stage
A pipeline first stage is undefined, but no result exception occurs

pfgt.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Greather-Than Compare**
(Assembler clears R-bit of instruction)
fdest ← last stage Adder result
CC set if *fsrc1* > *fsrc2*, else cleared
Advance A pipeline one stage
A pipeline first stage is undefined, but no result exception occurs

pfiadd.w *fsrc1, fsrc2, fdest* ..... **Pipelined Long-Integer Add**
fdest ← last stage Graphics result
last stage Graphics result ← *fsrc1* + *fsrc2*

pfisub.w *fsrc1, fsrc2, fdest* ..... **Pipelined Long-Integer Subtract**
fdest ← last stage Graphics result
last stage Graphics result ← *fsrc1* − *fsrc2*

pfix.v *fsrc1, fdest* ..... **Pipelined Floating-Point to Integer Conversion**
fdest ← last stage Adder result
Advance A pipeline one stage
A pipeline first stage ← 64-bit value with low-order 32 bits
equal to integer part of *fsrc1* rounded

Pipelined Floating-Point Load

pfld.z *isrc1*(*isrc2*), *fdest* ..... **(Normal)**
pfld.z *isrc1*(*isrc2*)++, *fdest* ..... **(Autoincrement)**
fdest ← mem.z (third previous **pfld's** (*isrc1* + *isrc2*))
(where .z is precision of third previous **pfld.z)**
If autoincrement
THEN *isrc2* ← *isrc1* + *isrc2*
FI

pfle.p *fsrc1, fsrc2, fdest* ..... **Pipelined F-P Less-Than or Equal Compare**
Assembler pseudo-operation, identical to **pfgt.p** except that
assembler sets R-bit of instruction.
fdest ← last stage Adder result
CC clear if *fsrc1* ≤ *fsrc2*, else set
Advance A pipeline one stage
A pipeline first stage is undefined, but no result exception occurs

pfmam.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Add and Multiply**
fdest ← last stage Multiplier result
Advance A and M pipeline one stage (operands accessed before advancing pipeline)
A pipeline first stage ← A-op1 − A-op2
M pipeline first stage ← M-op1 × M-op2

pfmov.r *fsrc1, fdest* ..... **Pipelined Floating-Point Reg-Reg Move**
Assembler pseudo-operation
pfmov.ss *fsrc1, fdest* = **pfiadd.ss** *fsrc1*, **f0,** *fdest*
pfmov.dd *fsrc1, fdest* = **pfiadd.dd** *fsrc1*, **f0,** *fdest*
pfmov.sd *fsrc1, fdest* = **pfamov.sd** *fsrc1*, *fdest*
pfmov.ds *fsrc1, fdest* = **pfamov.ds** *fsrc1*, *fdest*

pfmsm.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Subtract and Multiply**
fdest ← last stage Multiplier result
Advance A and M pipeline one stage (operands accessed before advancing pipeline)
A pipeline first stage ← A-op1 − A-op2
M pipeline first stage ← M-op1 × M-op2

pfmul.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Multiply**
fdest ← last stage Multiplier result
Advance M pipeline one stage
M pipeline first stage ← *fsrc1* × *fsrc2*

pfmul3.dd *fsrc1, fsrc2, fdest* ..... **Three-Stage Pipelined Multiply**
fdest ← last stage Multiplier result
Advance 3-Stage M pipeline one stage
M pipeline first stage ← *fsrc1* × *fsrc2*

pform *fsrc1, fdest* ..... **Pipelined OR to MERGE Register**
fdest ← last stage Graphics result
last stage Graphics result ← *fsrc1* OR MERGE
MERGE ← 0

pfsm.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Subtract and Multiply**
fdest ← last stage Adder result
Advance A and M pipeline one stage (operands accessed before advancing pipeline)
A pipeline first stage ← A-op1 − A-op2
M pipeline first stage ← M-op1 × M-op2

pfsub.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Subtract**
fdest ← last stage Adder result
Advance A pipeline one stage
A pipeline first stage ← *fsrc1* + *fsrc2*

pftrunc.v *fsrc1, fdest* ..... **Pipelined Floating-Point to Integer Conversion**
fdest ← last stage Adder result
Advance A pipeline one stage
A pipeline first stage ← 64-bit value with low-order 32 bits
equal to integer part of *fsrc1*

pfzchkl *fsrc1, fsrc2, fdest* ..... **Pipelined 32-Bit Z-Buffer Check**
Consider *fsrc1, fsrc2,* and *fdest*, as arrays of two 32-bit
fields *fsrc1*(0)..*fsrc1*(1), *fsrc2*(0)..*fsrc2*(1), and *fdest*(0)..*fdest*(1)
where zero denotes the least significant field.
PM ← PM shifted right by 2 bits
FOR i = 0 to 1
DO
PM [i + 6] ← *fsrc2*(i) ≤ *fsrc1*(i) (unsigned)
fdest(i) ← last stage Graphics result
last stage Graphics result ← smaller of *fsrc2*(i) and *fsrc1*(i)
OD
MERGE ← 0

pfzchks *fsrc1, fsrc2, fdest* ..... **Pipelined 16-Bit Z-Buffer Check**
Consider *fsrc1, fsrc2,* and *fdest*, as arrays of four 16-bit
fields *fsrc1*(0)..*fsrc1*(3), *fsrc2*(0)..*fsrc2*(3), and *fdest*(0)..*fdest*(3)
where zero denotes the least significant field.
PM ← PM shifted right by 4 bits
FOR i = 0 to 3
DO
PM [i + 4] ← *fsrc2*(i) ≤ *fsrc1*(i) (unsigned)
fdest(i) ← last stage Graphics result
last stage Graphics result ← smaller of *fsrc2*(i) and *fsrc1*(i)
OD
MERGE ← 0

pst.d *fdest, #const(isrc2)* ..... **Pixel Store**
pst.d *fdest, #const(isrc2)++* ..... **Pixel Store Autoincrement**
Pixels enabled by PM in mem.d (*isrc2* + *#const*) ← *fdest*
Shift PM right by 8/pixel size (in bytes) bits
IF autoincrement
THEN *isrc2* ← *#const* + *isrc2*
FI

shl *isrc1, isrc2, idest* ..... **Shift Left**
idest ← *isrc2* shifted left by *isrc1* bits

shr *isrc1, isrc2, idest* ..... **Shift Right**
SC (in **psr**) ← *isrc1*
idest ← *isrc2* shifted right by *isrc1* bits

shra *isrc1, isrc2, idest* ..... **Shift Right Arithmetic**
idest ← *isrc2* arithmetically shifted right by *isrc1* bits

shrd *isrc1, isrc2, idest* ..... **Shift Right Double**
idest ← low-order 32 bits of *isrc1:isrc2* shifted right by SC bits

st.c *isrc1ni, csrc2* ..... **Store to Control Register**
csrc2 ← *isrc1ni*

st.x *isrc1ni, #const(isrc2)* ..... **Store Integer**
mem.x (isrc2 + *#const)* ← *isrc1ni*

subs *isrc1, isrc2, idest* ..... **Subtract Signed**
idest ← *isrc1* − *isrc2*
OF ← (bit 31 carry ≠ bit 30 carry)
CC set if *isrc2* > *isrc1* (signed)
CC clear if *isrc2* ≤ *isrc1* (signed)

subu *isrc1, isrc2, idest* ..... **Subtract Unsigned**
idest ← *isrc1* − *isrc2*
OF ← NOT (bit 31 carry)
CC ← bit 31 carry
(i.e. CC set if *isrc2* ≤ *isrc1* (unsigned)
CC clear if *isrc2* > *isrc1* (unsigned)

trap *isrc1ni, isrc2, idest* ..... **Software Trap**
Generate trap with IT set in **psr**

unlock ..... **End Interlocked Sequence**
Clear BL in **dirbase**. The next load or store unlocks the bus.
Enable interrupts after bus is unlocked.

xor *isrc1, isrc2, idest* ..... **Logical Exclusive OR**
idest ← *isrc1* XOR *isrc2*
CC set if result is zero, cleared otherwise

xorh *#const, isrc2, idest* ..... **Logical Exclusive OR High**
idest ← (*#const* shifted left 16 bit) XOR *isrc2*
CC set if result is zero, cleared otherwise

Table 8.2. FADDP MERGE Update

Pixel Size (from PS)	Fields Loaded From Result into MERGE	Right Shift Amount (Field Size)
8	63..56, 47..40, 31..24, 15..8	8
16	63..58, 47..42, 31..26, 15..10	6
32	63..56, 31..24	8

8.2 Instruction Format and Encoding

All instructions are 32 bits long and begin on a four-byte boundary. When operands are registers, the register encodings shown in Table 8.3 are used. There are two general core-instruction formats, REG-format and CTRL-format, as well as a separate format for floating-point instructions.

8.2.1 REG-FORMAT INSTRUCTIONS

Within the REG-format are several variations as shown in Figure 8.1. Table 8.4 gives the encodings for these instructions. One encoding is an escape code that defines yet another variation: the core escape instructions. Figure 8.2 shows the format of this group, and Table 8.5 shows the encodings.

In these instructions, the *src2* field selects one of the 32 integer registers (most instructions) or five control registers (**st.c** and **ld.c**). *Dest* selects one of the 32 integer registers (most instructions) or floating-point registers (**fld**, **fst**, **pfld**, **pst**, **ixfr**). For instructions where *src1* is optionally an immediate value, bit 26 of the opcode (I-bit) indicates whether *src1* is an immediate. If bit 26 is clear, an integer register is used; if bit 26 is set, *src1* is contained in the low-order 16 bits, except for **bte** and **btne** instructions. For **bte** and **btne**, the five-bit immediate value is contained in the *src1* field. For **st**, **bte**, **btne**, and **bla**, the upper five bits of the *offset* or *broffset* are contained in the *dest* field instead of *src1*, and the lower 11 bits of *offset* are the lower 11 bits of the instruction.

Table 8.3. Register Encoding

Register	Encoding
r0	0
.	.
.	.
.	.
r31	31
f0	0
.	.
.	.
.	.
f31	31
Fault Instruction	0
Processor Status	1
Directory Base	2
Data Breakpoint	3
Floating-Point Status	4
Extended Process Status	5

For **ld** and **st**, bits 28 and zero determine operand size as follows:

Bit 28	Bit 0	Operand Size
0	0	8-bits
0	1	8-bits
1	0	16-bits
1	1	32-bits

When *src1* is an immediate and bit 28 is set, bit zero of the immediate value is forced to zero.

For **fld**, **fst**, **pfld**, **pst**, and **flush**, bit 0 selects autoincrement addressing if set. For **fld**, **fst**, **pfld**, and **pst**, bits one and two select the operand size as follows:

Bit 1	Bit 2	Operand Size
0	0	64-bits
0	1	128-bits
1	0	32-bits
1	1	32-bits

When *src1* is an immediate value, bits zero and one of the immediate value are forced to zero to maintain alignment. When bit one of the immediate value is clear, bit two is also forced to zero.

For **flush**, bits one and two must be zero.

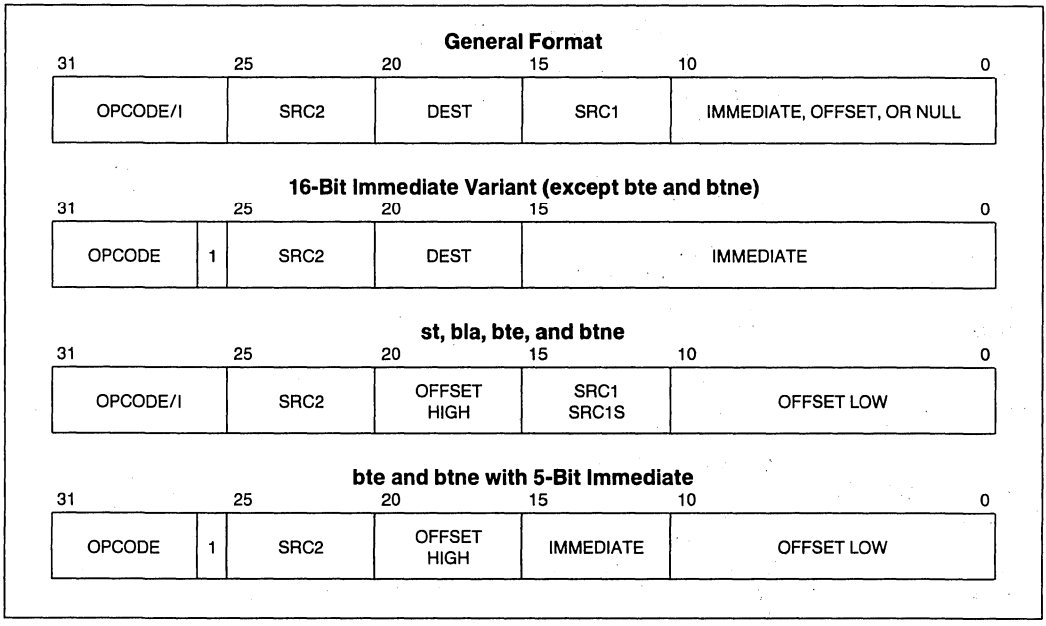


Figure 8.1. REG-Format Variations

Table 8.4. REG-Format Opcodes

		31					26
ld.x	Load Integer	0	0	0	L	0	I
st.x	Store Integer	0	0	0	L	1	1
ixfr	Integer to F-P Reg Transfer	0	0	0	0	1	0
	(reserved)	0	0	0	1	1	0
fld.x, fst.x	Load/Store F-P	0	0	1	0	LS	I
flush	Flush	0	0	1	1	0	1
pst.d	Pixel Store	0	0	1	1	1	1
ld.c, st.c	Load/Store Control Register	0	0	1	1	LS	0
bri	Branch Indirect	0	1	0	0	0	0
trap	Trap	0	1	0	0	0	1
	(Escape for F-P Unit)	0	1	0	0	1	0
	(Escape for Core Unit)	0	1	0	0	1	1
bte, btne	Branch Equal or Not Equal	0	1	0	1	E	I
pfld.y	Pipelined F-P Load	0	1	1	0	0	I
	(CTRL-Format Instructions)	0	1	1	x	x	x
addu, -s, subu, -s,	Add/Subtract	1	0	0	SO	AS	I
shl, shr	Logical Shift	1	0	1	0	LR	I
shrd	Double Shift	1	0	1	1	0	0
bla	Branch LCC Set and Add	1	0	1	1	0	1
shra	Arithmetic Shift	1	0	1	1	1	I
and(h)	AND	1	1	0	0	H	I
andnot(h)	ANDNOT	1	1	0	1	H	I
or(h)	OR	1	1	1	0	H	I
xor(h)	XOR	1	1	1	1	H	I
	(reserved)	1	1	x	x	1	0

L Integer Length
- 0 —8 bits
- 1 —16 or 32 bits (selected by bit 0)

LS Load/Store
- 0 —Load
- 1 —Store

SO Signed/Ordinal
- 0 —Ordinal
- 1 —Signed

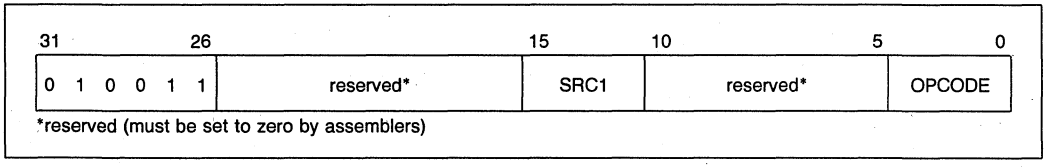
H High
- 0 —**and, or, andnot, xor**
- 1 —**andh, orh, andnoth, xorh**

AS Add/Subtract
- 0 —Add
- 1 —Subtract

LR Left/Right
- 0 —Left Shift
- 1 —Right Shift

E Equal
- 0 —Branch on Not Equal
- 1 —Branch on Equal

I Immediate
- 0 —*src1* is register
- 1 —*src1* is immediate



*reserved (must be set to zero by assemblers)

Figure 8.2. Core Escape Instruction Format

Table 8.5. Core Escape Opcodes

		4				0
	(reserved)	0	0	0	0	0
lock	Begin Interlocked Sequence	0	0	0	0	1
calli	Indirect Subroutine Call	0	0	0	1	0
	(reserved)	0	0	0	1	1
intovr	Trap on Integer Overflow	0	0	1	0	0
	(reserved)	0	0	1	0	1
	(reserved)	0	0	1	1	0
unlock	End Interlocked Sequence	0	0	1	1	1
	(reserved)	0	1	x	x	x
	(reserved)	1	0	x	x	x
	(reserved)	1	1	x	x	x

8.2.2 CTRL-FORMAT INSTRUCTIONS

The CTRL instructions do not refer to registers, so instead of the register fields, they have a 26-bit relative branch offset. Figure 8.3 shows the format of these instructions and Table 8.6 defines the encodings.

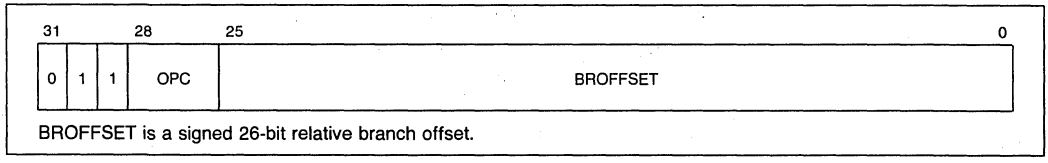


Figure 8.3. CTRL Instruction Format

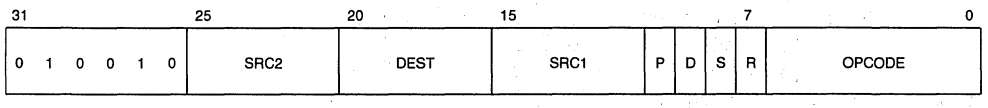
Table 8.6. CTRL-Format Opcodes

		28		26
	(reserved)	0	0	0
	(reserved)	0	0	1
br	Branch Direct	0	1	0
call	Call	0	1	1
bc(.t)	Branch on CC Set	1	0	T
bnc(.t)	Branch on CC Clear	1	1	T

T Taken
0 —**bc** or **bnc**
1 —**bc.t** or **bnc.t**

8.2.3 FLOATING-POINT INSTRUCTIONS

The floating-point instructions also constitute an escape series. All these instructions begin with the bit sequence 010010. Figure 8.4 shows the format of the floating point instructions, and Table 8.7 gives the encodings. Within the dual-operation instructions is a subcode DPC whose values are given in Table 8.8 along with the mnemonic that corresponds to each.



SRC1, SRC2 —Source; one of 32 floating-point registers
DEST —Destination register
(instructions other than **fxfr**) one of 32 floating-point registers
(**fxfr**) one of 32 integer registers

P Pipelining
1 —Pipelined instruction mode
0 —Scalar instruction mode

D Dual-Instruction Mode
1 —Dual-instruction mode
0 —Single-instruction mode

S Source Precision
1 —Double-precision source operands
0 —Single-precision source operands

R Result Precision
1 —Double-precision result
0 —Single-precision result

Figure 8.4. Floating-Point Instruction Encoding

Table 8.7. Floating-Point Opcodes

		6						0
pfam **pfmam**	Add and Multiply* Multiply with Add*	0	0	0	DPC			
pfsm **pfmsm**	Subtract and Multiply* Multiply with Subtract*	0	0	1	DPC			
(p)fmul	Multiply	0	1	0	0	0	0	0
fmlow	Multiply Low	0	1	0	0	0	0	1
frcp	Reciprocal	0	1	0	0	0	1	0
frsqr	Reciprocal Square Root	0	1	0	0	0	1	1
pfmul3.dd	3-Stage Pipelined Multiply	0	1	0	0	1	0	0
(p)fadd	Add	0	1	1	0	0	0	0
(p)fsub	Subtract	0	1	1	0	0	0	1
(p)fix	Fix	0	1	1	0	0	1	0
(p)famov	Adder Move	0	1	1	0	0	1	1
pfgt/pfle**	Greater Than	0	1	1	0	1	0	0
pfeq	Equal	0	1	1	0	1	0	1
(p)ftrunc	Truncate	0	1	1	1	0	1	0
fxfr	Transfer to Integer Register	1	0	0	0	0	0	0
(p)fiadd	Long-Integer Add	1	0	0	1	0	0	1
(p)fisub	Long-Integer Subtract	1	0	0	1	1	0	1
(p)fzchkl	Z-Check Long	1	0	1	0	1	1	1
(p)fzchks	Z-Check Short	1	0	1	1	1	1	1
(p)faddp	Add with Pixel Merge	1	0	1	0	0	0	0
(p)faddz	Add with Z Merge	1	0	1	0	0	0	1
(p)form	OR with MERGE Register	1	0	1	1	0	1	0

*pfam and pfsm have P-bit set; pfmam and pfmsm have P-bit clear.
****pfgt** has R bit cleared; **pfle** has R bit set.

NOTE:
All opcodes not shown are **reserved**.

The following table shows the opcode mnemonics that generate the various encodings of DPC and explains each encoding.

Table 8.8. DPC Encoding

DPC	PFAM Mnemonic	PFSM Mnemonic	M-Unit op1	M-Unit op2	A-Unit op1	A-Unit op2	T Load	K Load*
0000	**r2p1**	**r2s1**	KR	src2	src1	M result	No	No
0001	**r2pt**	**r2st**	KR	src2	T	M result	No	Yes
0010	**r2ap1**	**r2as1**	KR	src2	src1	A result	Yes	No
0011	**r2apt**	**r2ast**	KR	src2	T	A result	Yes	Yes
0100	**i2p1**	**i2s1**	KI	src2	src1	M result	No	No
0101	**i2pt**	**i2st**	KI	src2	T	M result	No	Yes
0110	**i2ap1**	**i2as1**	KI	src2	src1	A result	Yes	No
0111	**i2apt**	**i2ast**	KI	src2	T	A result	Yes	Yes
1000	**rat1p2**	**rat1s2**	KR	A result	src1	src2	Yes	No
1001	**m12apm**	**m12asm**	src1	src2	A result	M result	No	No
1010	**ra1p2**	**ra1s2**	KR	A result	src1	src2	No	No
1011	**m12ttpa**	**m12ttsa**	src1	src2	T	A result	Yes	No
1100	**iat1p2**	**iat1s2**	KI	A result	src1	src2	Yes	No
1101	**m12tpm**	**m12tsm**	src1	src2	T	M result	No	No
1110	**ia1p2**	**ia1s2**	KI	A result	src1	src2	No	No
1111	**m12tpa**	**m12tsa**	src1	src2	T	A result	No	No

DPC	PFMAM Mnemonic	PFMSM Mnemonic	M-Unit op1	M-Unit op2	A-Unit op1	A-Unit op2	T Load	K Load*
0000	**mr2p1**	**mr2s1**	KR	src2	src1	M result	No	No
0001	**mr2pt**	**mr2st**	KR	src2	T	M result	No	Yes
0010	**mr2mp1**	**mr2ms1**	KR	src2	src1	M result	Yes	No
0011	**mr2mpt**	**mr2mst**	KR	src2	T	M result	Yes	Yes
0100	**mi2p1**	**mi2s1**	KI	src2	src1	M result	No	No
0101	**mi2pt**	**mi2st**	KI	src2	T	M result	No	Yes
0110	**mi2mp1**	**mi2ms1**	KI	src2	src1	M result	Yes	No
0111	**mi2mpt**	**mi2mst**	KI	src2	T	M result	Yes	Yes
1000	**mrmt1p2**	**mrmt1s2**	KR	M result	src1	src2	Yes	No
1001	**mm12mpm**	**mm12msm**	src1	src2	M result	M result	No	No
1010	**mrm1p2**	**mrm1s2**	KR	M result	src1	src2	No	No
1011	**mm12ttpm**	**mm12ttsm**	src1	src2	T	A result	Yes	No
1100	**mimt1p2**	**mimt1s2**	KI	M result	src1	src2	Yes	No
1101	**mm12tpm**	**mm12tsm**	src1	src2	T	M result	No	No
1110	**mim1p2**	**mim1s2**	KI	M result	src1	src2	No	No
1111	Intel-Reserved							

*If K-load is set, KR is loaded when operand-1 of the multiplier is KR; KI is loaded when operand-1 of the multiplier is KI.

8.3 Instruction Timings

i860 XR microprocessor instructions take one clock to execute unless a freeze condition is invoked. Freeze conditions and their associated delays are shown in the table below. Freezes due to multiple simultaneous cache misses result in a delay that is the sum of the delays for processing each miss by itself. Other multiple freeze conditions usually add only the delay of the longest individual freeze.

Freeze Condition	Delay
Instruction-cache miss	Number of clocks to read instruction (from ADS clock to first READY# clock) plus time to last READY# of block when jump or freeze occurs during miss processing plus two clocks if data-cache being accessed when instruction-cache miss occurs.
Reference to destination of **ld** instruction that misses	One plus number of clocks to read data (from ADS# clock to first READY# clock) minus number of instructions executed since load (not counting instruction that references load destination)
fld miss	One plus number of clocks until first READY# returned (for 32- or 64-bit read cycles) or until second READY# returned (for 128-bit **fld.q** read cycles)
call, calli, ixfr, fxfr, ld.c, or **st.c** and data cache load miss processing in progress	One plus number of clocks until first READY# returned (for 64-bit read cycles) or until second READY# returned (for 128-bit **fld.q** read cycles)
ld/st/pfld/fld/fst and data cache load miss processing in progress	One plus number of clocks until last READY# returned
Reference to *dest* of **ld, call, calli, fxfr,** or **ld.c** in the next instruction. (*Dest* of **call** and **calli** is **r1**.)	One clock

Freeze Condition	Delay
Reference to *dest* of **fld/pfld/ixfr** in the next two instructions	Two clocks in the first instruction; one in the second instruction
bc/bnc/bc.t/bnc.t following **addu/adds/subu/subs/pfeq/pfle/pfgt**	One clock
Fsrc1 of multiplier operation refers to result of previous operation	One clock
Floating-point operation or graphics-unit instruction or **fst,** and scalar operation in progress other than **frcp** or **frsqr**	If the scalar operation is **fadd, fix, fmlow, fmul.ss, fmul.sd, ftrunc,** or **fsub,** two minus the number of instructions (or dual-mode pairs) already executed after the scalar operation. If the scalar operation is **fmul.dd,** three minus the number of instructions (or dual-mode pairs) executed after it. Add one if either or both of these two situations occur: 1. There is an overlap between the result register of the previous scalar operation and the source of the floating-point operation, and the destination precision of the scalar operation is different than the source precision of the floating-point operation. 2. The floating-point operation is pipelined and its destination is not **f0.** There is no delay if the result is negative.
Multiplier operation preceded by a double precision multiply	One clock
TLB miss	Five plus the number of clocks to finish two reads plus the number of clocks to set A-bits (if necessary)
pfld when three **pfld**'s are outstanding	One plus the number of clocks to return data from first **pfld**
pfld hits in the data cache	Two plus the number of clocks to finish all outstanding accesses
st, pst or **fst** miss, **ld** miss, or **flush** with modified block when store path full (two stores or one 256-bit write-back internally waiting for bus plus external bus pipeline full)	One plus the number of clocks until READY# active on next 64-bit write cycle or second READY# of next 128-bit write cycle.
ld, fld, pfld, st, pst, or **fst** when address path full (one address internally waiting for bus plus external bus pipeline full)	Number of clocks until next nonrepeated address can be issued (i.e., an address that is not the 2nd–4th cycle of a cache fill, the 2nd–8th cycle of a CS8 mode instruction fetch, nor the 2nd cycle of a 128-bit write)
ld/fld following **st/fst** hit	One clock

Freeze Condition	Delay
Delayed branch not taken	One clock
Nondelayed branch taken:	
bc, bnc	One clock
bte, btne	Two clocks
Indirect branch **bri** or call **calli**	One clock
st.c	Two clocks
Result of graphics-unit instruction (other than **fmov.dd**) used in next instruction when the next instruction is an adder- or multiplier-unit instruction	One clock
Result of graphics-unit instruction used in next instruction when the next instruction is a graphics-unit instruction	One clock
flush followed by **flush**	Three clocks minus the number of instructions between the two **flush** instructions. There is no delay if the result is negative.
fst or **pst** followed by pipelined floating-point operation that overwrites the register being stored	One clock

8.4 Instruction Characteristics

The following table lists some of the characteristics of each instruction. The characteristics are:

- What processing unit executes the instruction. The codes for processing units are:
  - A Floating-point adder unit
  - E Core execution unit
  - G Graphics unit
  - M Floating-point multiplier unit
- Whether the instruction is pipelined or not. A *P* indicates that the instruction is pipelined.
- Whether the instruction is a delayed branch instruction. A *D* marks the delayed branches.
- Whether the instruction changes the condition code CC. A *CC* marks those instructions that change CC.
- Which faults can be caused by the instruction. The codes used for exceptions are:
  - IT Instruction Fault
  - SE Floating-Point Source Exception
  - RE Floating-Point Result Exception, including overflow, underflow, inexact result
  - DAT Data Access Fault

  Note that this is not the same as specifying at which instructions faults may be reported. A result exception is reported on the subsequent floating-point instruction, **pst, fst,** or sometimes **fld, pfld**, and **ixfr**.

  The instruction access fault IAT and the interrupt trap IN are not shown in the table because they can occur for any instruction.
- Performance notes. These comments regarding optimum performance are recommendations only. If these recommendations are not followed, the i860 XR microprocessor automatically waits the necessary number of clocks to satisfy internal hardware requirements. The following notes define the numeric codes that appear in the instruction table:
  1. The following instruction should not be a conditional branch (**bc**, **bnc**, **bc.t**, or **bnc.t**).
  2. The destination should not be a source operand of the next two instructions.
  3. A load should not directly follow a store that is expected to hit in the data cache.
  4. When the prior instruction is scalar, *fsrc1* should not be the same as the *fdest* of the prior operation.
  5. The *fdest* should not reference the destination of the next instruction if that instruction is a pipelined floating-point operation.
  6. The destination should not be a source operand of the next instruction. (For **call** and **calli**, the destination is **r1**.)

7. When the prior operation is scalar and multiplier *op1* is *fsrc1*, *fsrc2* should not be the same as the *fdest* of the prior operation.
8. When the prior operation is scalar, *fsrc1* and *fsrc2* of the current operation should not be the same as *fdest* of the prior operation.
9. A **pfld** should not immediately follow a **pfld**.

- Programming restrictions. These indicate combinations of conditions that must be avoided by programmers, assemblers, and compilers. The following notes define the alphabetic codes that appear in the instruction table:
  a. The sequential instruction following a delayed control-transfer instruction may not be another control-transfer instruction (except in the case of external interrupts), nor a trap instruction, nor the target of a control-transfer instruction.
  b. When using a **bri** to return from a trap handler, programmers should take care to prevent traps from occurring on that or on the next sequential instruction. IM should be zero (interrupts disabled) when the **bri** is executed.
  c. If *fdest* is not zero, *fsrc1* must not be the same as *fdest*.
  d. When *fsrc1* goes to the multiplier *op1*, KR, or KI, *fsrc1* must not be the same as *fdest*.
  e. If *fdest* is not zero, *fsrc1* and *fsrc2* must not be the same as *fdest*.
  f. *isrc1* must not be the same as *isrc2* for the autoincrementing form of this instruction.
  g. *isrc1* must not be the same as *isrc2*.
- Core and Floating-Point Instruction Interaction in Dual-Instruction Mode
  1. If one of the branch-on-condition instructions **bc** or **bnc** is paired with a floating-point compare, the branch tests the value of the condition code prior to the compare.
  2. If an **ixfr, fld,** or **pfld** loads the same register as a source operand in the floating point instruction, the floating-point instruction references the register value before the load updates it.
  3. An **fst** or **pst** that stores a register that is the destination register of the companion pipelined floating-point operation will store the result of the companion operation.
  4. When the core instruction sets CC and the floating-point instruction is **pfgt, pfle,** or **pfeq,** CC is set according to the result of **pfgt, pfle,** or **pfeq.**
  5. When a **trap** instruction causes a trap in dual-instruction mode, the floating-point instruction has neither completed execution nor has updated the FT bit or any result status bits. This is not a problem when the **trap** is inserted by a debugger, because the **trap** is replaced by the original instruction, and the dual-mode pair is reexecuted. However, when the **trap** is programmed, the trap handler must avoid reexecuting the **trap** by returning to user code at the address in **fir** + 8. In this case, the trap handler must emulate the floating-point instruction before returning to the user code. Emulation of the instruction must include all side-effects (for example, the effect of its D-bit, effect on the pipelines, and effect on FT and result-status bits), just as if the instruction had been executed by the processor in the original context.
  6. In dual-instruction mode, when the **intovr** instruction causes a trap, the floating-point companion instruction has completely finished execution before the trap is taken.

- Programming Restrictions for Dual-Instruction Mode

1. The result of placing a core instruction in the low-order 32 bits or a floating-point instruction in the high-order 32 bits is not defined (except for **shrd r0, r0, r0** which is interpreted as **fnop**).
2. A floating-point instruction that has the D-bit set must be aligned on a 64-bit boundary (i.e., the three least-significant bits of its address must be zero). This applies as well to the initial 32-bit floating-point instruction that triggers the transition into dual-instruction mode, but does not apply to the following instruction.
3. When the floating-point operation is scalar and the core operation is **fst** or **pst,** the store should not reference the result register of the floating-point operation. When the core operation is **pst,** the floating-point instruction cannot be **(p)fzchks** or **(p)fczhkl.**
4. When the core instruction of a dual-mode pair is a control-transfer operation and the previous instruction had the D-bit set, the floating-point instruction must also have the D-bit set. In other words, an exit from dual-instruction mode cannot be initiated (first instruction pair without D-bit set) when the core instruction is a control-transfer instruction.
5. When the core operation is a **ld.c** or **st.c,** the floating-point operation must be **d.fnop.**
6. When the floating-point operation is **fxfr,** the core instruction cannot be **ld, ld.c, st, st.c, call ixfr,** or any instruction that updates an integer register (including autoincrement indexing). Furthermore, the core instruction cannot be a **fld, fst, pst,** or **pfld** that uses as *isrc1* or *isrc2* the same register as the *idest* of the **fxfr.** Additionally, in dual instruction mode, **fxfr** may not be used in a branch delay slot if its destination register is referenced by the preceding branch instruction.
7. A **bri** must not be executed in dual-instruction mode if any trap bits are set.
8. When the core operation is **bc.t** or **bnc.t,** the floating point operation cannot be **pfeq** or **pfgt.** The floating-point operation in the sequentially following instruction pair cannot be **pfeq** or **pfgt,** either.
9. A transition to or from dual-instruction mode cannot be initiated on the instruction following a **bri.**
10. An **ixfr, fld, or pfld** cannot update the destination of the companion floating-point instruction (unless the destination is **f0** or **f1**) or of the following pipelined floating-point instruction (regardless of its destination register). No overlap of register destinations is permitted; for example, the following instructions must not be paired:

```
// Illegal case 1
    d.fmul.ss  f9, f10, f5
      fld.d    address, f4
        ; Overlaps f5

// Illegal case 2
    d.fmul.ss  f0, f0, f3
      fld.q    address, f0
        ; Overlaps f3

// Illegal case 3
    d.fmul.ss  f9, f10, f11
      fld.l    address, f5
    d.pfadd.ss fx, fx, f4
        ; Overlaps f5, if last
          stage result is double-
          precision
```

11. During a locked sequence, a transition to or from dual-instruction mode is not permitted.

Table 8.9 Instruction Characteristics

Instruction	Execution Unit	Pipelined? Delayed?	Sets CC?	Faults	Performance Notes	Programming Restrictions
adds	E		CC		1	
addu	E		CC		1	
and	E		CC			
andh	E		CC			
andnot	E		CC			
andnoth	E		CC			
bc	E					
bc.t	E	D				a
bla	E	D				a, g
bnc	E					
bnc.t	E	D				a
br	E	D				a
bri	E	D				a, b
bte	E					
btne	E					
call	E	D			6	a
calli	E	D			6	a
fadd.p	A			SE, RE		
faddp	G				8	
faddz	G				8	
famov.r	A			SE, RE		
fiadd.z	G				8	
fisub.z	G				8	
fix.p	A			SE, RE		
fld.y	E			DAT	2, 3	f
flush	E					
fmlow.p	M				4	
fmul.p	M			SE, RE	4	
form	G				8	
frep.p	M			SE, RE		
frsqr.p	M			SE, RE		
fst.y	E			DAT	5	f
fsub.p	A			SE, RE		
ftrunc.p	A			SE, RE		
fxfr	G				6, 8	
fzchkl	G				8	
fzchks	G				8	
intovr	E			IT		
ixfr	E				2	
ld.c	E					
ld.x	E			DAT	6	
or	E		CC			
orh	E		CC			
pfadd.p	A	P		SE, RE		
pfaddp	G	P			8	e

Table 8.9 Instruction Characteristics (Continued)

Instruction	Execution Unit	Pipelined? Delayed?	Sets CC?	Faults	Performance Notes	Programming Restrictions
pfaddz	G	P			8	e
pfam.p	A&M	P		SE, RE	7	d
pfamov.r	A	P		SE, RE		
pfeq.p	A	P	CC	SE	1	
pfgt.p	A	P	CC	SE	1	
pfiadd.z	G	P			8	e
pfisub.z	G	P			8	e
pfix.p	A	P		SE, RE		
pfld.z	E	P		DAT	2, 9	f
pfmam.p	A & M	P		SE, RE	7	d
pfmsm.p	A & M	P		SE, RE	7	d
pfmul.p	M	P		SE, RE	4	c
pfmul3.dd	M	P		SE, RE	4	c
pform	G	P			8	e
pfsm.p	A&M	P		SE, RE	7	d
pfsub.p	A	P		SE, RE		
pftrunc.p	A	P		SE, RE		
pfzchkl	G	P			8	
pfzchks	G	P			8	
pst.d	E			DAT		f
shl	E					
shr	E					
shra	E					
shrd	E					
st.c	E					
st.x	E			DAT		
subs	E		CC		1	
subu	E		CC		1	
trap	E			IT		
xor	E		CC			
xorh	E		CC			

DATA SHEET REVISION REVIEW

The following list represents the key differences between version 002 and version 001 of the i860 XR Microprocessor Data Sheet.

1. Big-endian description in section 2.3 has been expanded.
2. Bit 17 of the Extended Processor Status Register (EPSR) is the INT bit which reflects the value on the interrupt pin (INT), as described in section 2.2.4 entitled "EXTENDED PROCESSOR STATUS REGISTER". This is a documentation update only.
3. The cacheability of a page is controlled by NOR'ing the value of the CD, WT bits and the KEN# input pin, as described in section 2.5 entitled "Caching and Cache Flushing" and section 3.1.14 entitled "Cache Enable (KEN#)". This is a documentation update only.
4. The NOTE section in section 2.5 entitled "Caching and Cache Flushing" has been updated to clarify the paging requirement on changing the DTB field in the **dirbase** register.
5. Information on register encoding is added in section 8.2 entitled "Instruction Format and Encoding". This is a documentation update only.

The following list represents the key differences between version 003 and version 002 of the i860 XR Microprocessor Data Sheet.

Specification Changes:

1. Specification changes for improved AC performance are in section 7.3.
2. HOLD is acknowledged during locked bus cycles. See section 3.1.8.
3. Additional paths have been added to the bus state diagram to allow direct transitions from states T12 and T11 to state TH. See Figures 4.1 and 4.10.
4. Two new instructions, **(p)famov.r,** have been added. These replace **(p)fadd.ds** and **(p)fadd.sd** in the assembler pseudo-ops **(p)fmov.r**. These changes are in section 8.1 and tables 2.7, 8.7, and 8.9.

Documentation Changes:

1. Big and little endian description has been expanded in sections 2.2.2, 2.3, and Figure 2.8.
2. The actions and explanations of the **lock, unlock,** and **st.c dirbase** changing the BL bit have been updated in sections 2.2.4, 3.1.5, 3.1.8, 4.3.4, 4.3.5, and 8.1.
3. The explanation of the AA and MA bits of the **fpsr** have been expanded in section 2.2.8.
4. The explanation of the WT bit of the Page Table Entries has been expanded in sections 2.4.4.4 and 2.5.
5. A change concerning the locking of the bus during address translation is explained in sections 2.4.5 and 2.8.5.
6. A further explanation on when to flush the data cache is given in section 2.5.
7. The explanation of the floating point multiplier pipeline has been expanded in section 2.6.1.
8. The explanation of BREQ has been expanded in section 3.1.4 and Figure 4.1.
9. The explanation of result exceptions has been expanded in sections 2.8 and 3.2.
10. Instruction fetch identification has been clarified in section 3.1.6 and table 3.2.
11. Bus cycle diagrams in Figures 4.7, 4.8, and 4.10 have been clarified/corrected.
12. Precision specification **.r** has been added to section 8.0 and table 8.1.
13. In section 8.4, performance note 9 has been added, programming restriction d has been changed, and programming restriction f has been added. Table 8.9 has been updated to reflect these changes.
14. The description of testability has changed in sections 3.3. and 3.3.2. RESET and HOLD must be asserted by the tester to force the chip outputs to float (tri-state).

The following list represents the major differences between version 004 and version 003 of the i860 XR Microprocessor Data Sheet:

Section 2.2.4 The explanation of the WP bit of the **espr** has been expanded.

Section 2.8.2 More information on the instruction trap has been added.

Section 2.8.4 The instruction access trap has been clarified.

Section 2.8.7 The values of registers after a reset trap have been specified.

Section 3.1.4 BREQ timing has been clarified.

Section 3.1.5 The calculation of interrupt latency has bee corrected.

Section 3.1.6 The description of the byte-enable signals has been expanded.

Section 3.1.8 The relation between the **lock** instruction and the LOCK# signal has been clarified. The BL bit should no longer be changed by writing to the **dirbase** register.

Section 6.0 The thermal specifications have been updated.

Section 7.3 The A.C. Characteristics for CLK have changed.

Section 7.3 Advance timing information for the 50 MHz clock rate has been added. These timings are subject to change without notice.

Section 8.0 The operand naming conventions have improved.

Section 8.2.1 The encoding of the **flush** instruction has been corrected.

Section 8.3 The data-dependent multiplier freeze has been eliminated. Other freeze conditions have been corrected or clarified.

The following list represents the major differences between version 005 and version 004 of the i860 XR Microprocessor Data Sheet.

Section 2.2.4 OF bit is writable only in supervisor mode using ST.C.

Section 3.1.1 CLK rate has been updated.

Section 5.0 Figure 5.3 has been corrected.

Section 6.0 More information on measuring case temperature has been added.

Section 6.0 Figure 6.1 has been updated to include 25 MHz.

Section 6.0 Table 6.1 has been corrected.

Section 6.0 Table 6.2 has been updated to include 25 MHz.

Section 7.2 The D.C. Characteristics have been updated to include 25 MHz power supply current.

Section 7.3 The A.C. Characteristics for CLK have been changed.

Section 7.3 50 MHz clock rate has been deleted.

Section 7.3 25 MHz A.C. Specifications have been added.

Section 7.3 Figure 7.1 has been corrected.

Section 8.3 The data-dependent multiplier rounding freeze has been eliminated.

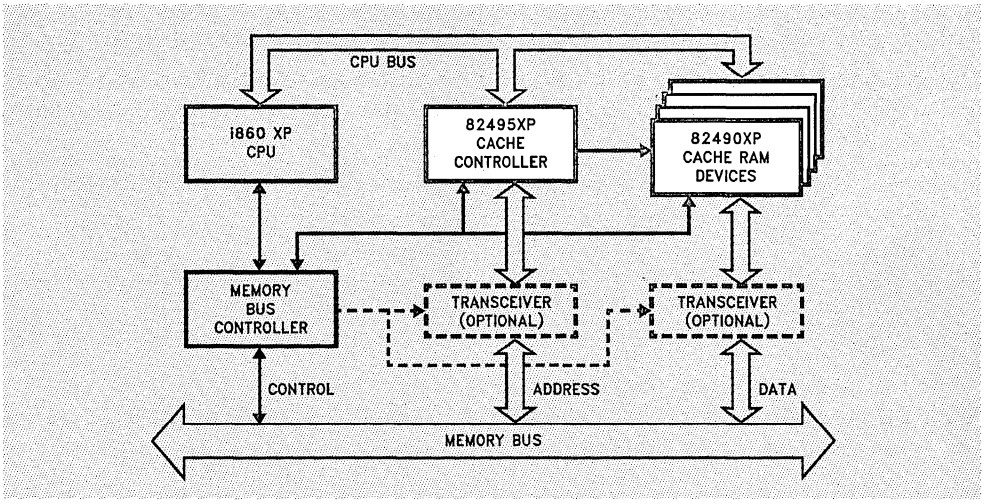
Section 8.4 Programming restrictions for dual-instruction mode are added.

82495XP CACHE CONTROLLER/ 82490XP CACHE RAM

- Two-Way, Set Associative, Secondary Cache for i860™ XP Microprocessor
- 50 MHz "No Glue" Interface with CPU
- Configurable
  - — Cache Size 256 or 512 Kbytes
  - — Line Width 32, 64 or 128 Bytes
  - — Memory Bus Width 64 or 128 Bits
- Dual-Ported Structure Permits Simultaneous Operations on CPU and Memory Buses
- Efficient MRU Way Prediction
  - — Zero Wait States on MRU Hit
  - — One Wait State on MRU Miss
- Dynamically Selectable Update Policies
  - — Write-Through
  - — Write-Once
  - — Write-Back
- MESI Cache Consistency Protocol
- Hardware Cache Snooping
- Maintains Consistency with Primary Cache via Inclusion Principle
- Flexible User-Implemented Memory Interface Enables Wide Range of Product Differentiation
  - — Clocked or Strobed
  - — Synchronous or Asynchronous
  - — Pipelining
  - — Memory Bus Protocol
- 82495XP Cache Controller Available in 208-Lead Ceramic Pin Grid Array Package
- 82490XP Cache RAM Available in 84-Lead Plastic Quad Flatpack Package
  (See *Packaging Handbook*, Order #240800)

2

The Intel 82495XP cache controller and 82490XP cache RAM, when coupled with a user-implemented memory bus controller, provide a second-level cache subsystem that eliminates the memory latency and bandwidth bottleneck for a wide range of multiprocessor systems based on the i860 XP microprocessor. The CPU interface is optimized to serve the i860 XP microprocessor with zero wait states at up to 50 MHz. A secondary cache built from the 82495XP and 82490XP isolates the CPU from the memory subsystem; the memory can run slower and follow a different protocol than the i860 XP microprocessor.



240956-60

Figure 0-1. Secondary Cache Configuration

Intel, int_el, and i860 are trademarks of Intel Corporation.

Order Number: 240956-001

1.0 82495XP/82490XP PINOUTS

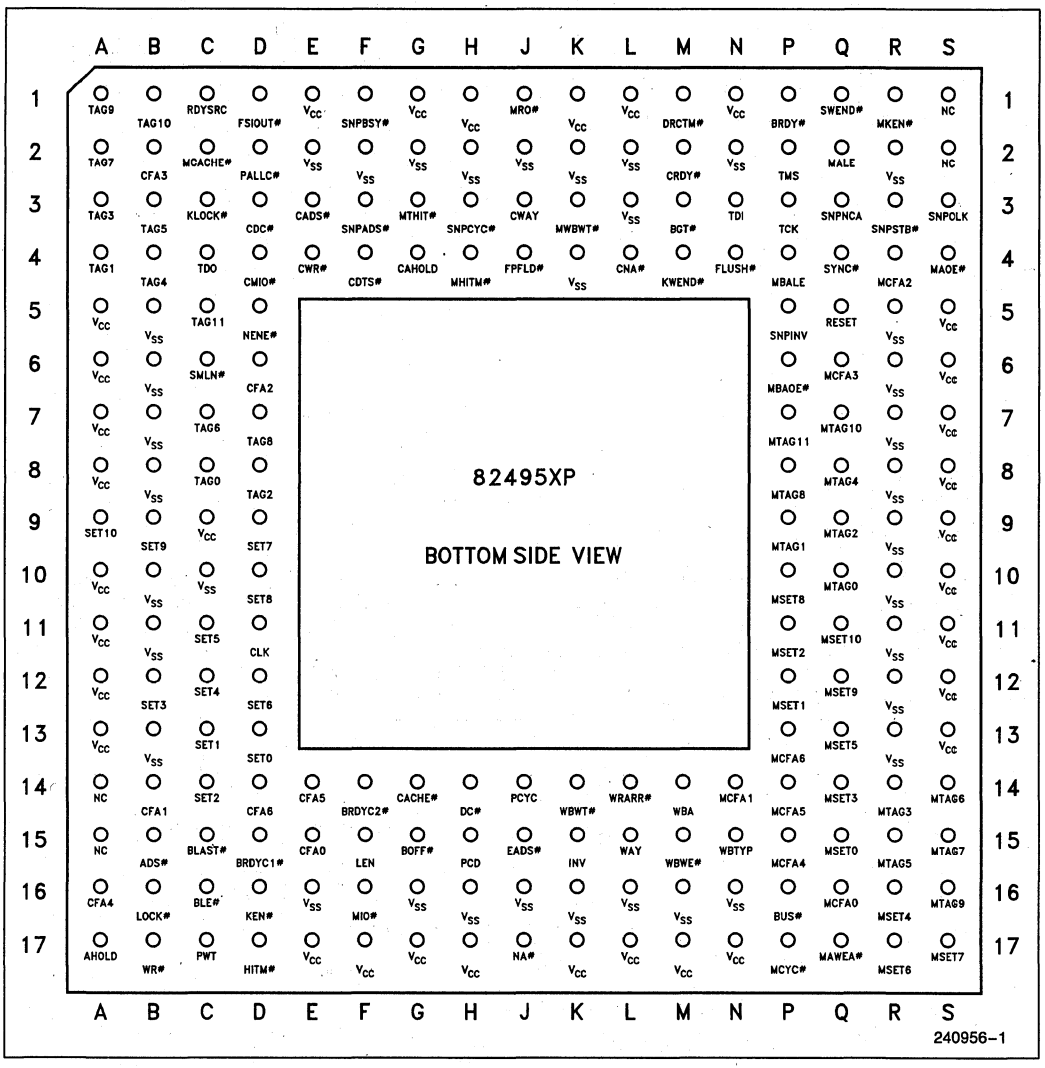


Figure 1-1. 82495XP Pinout (Bottom View)

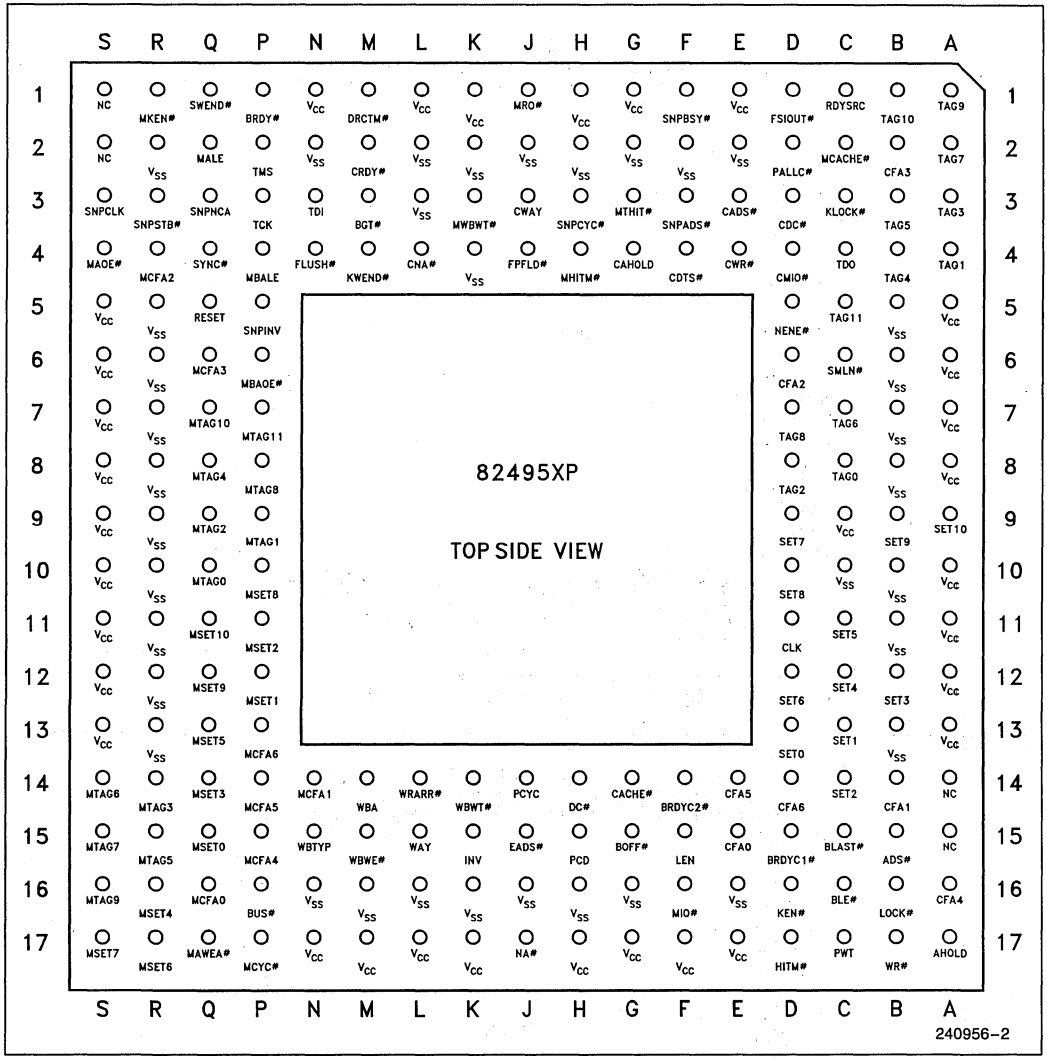


Figure 1-2. 82495XP Pinout (Top View)

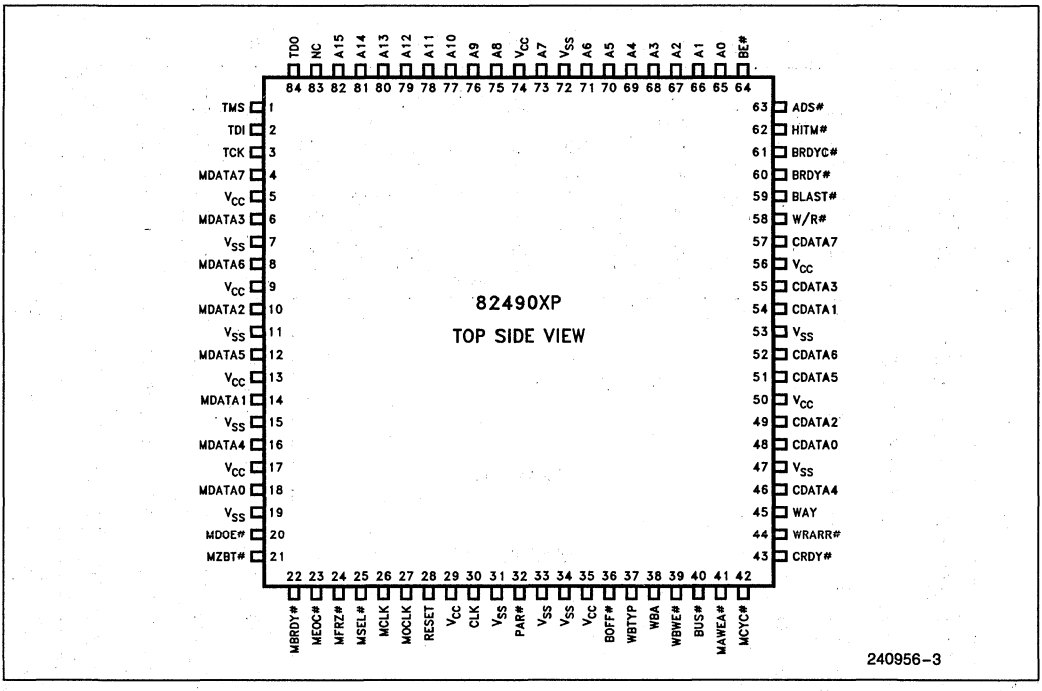


Figure 1-3. 82490XP Pinout (Top View)

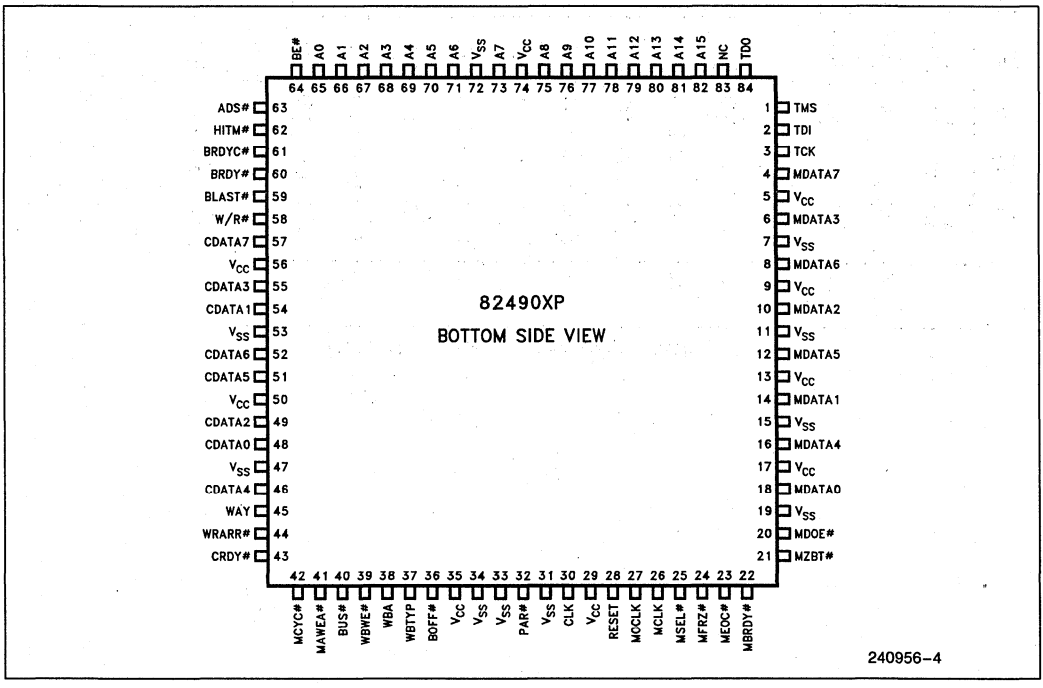


Figure 1-4. 82490XP Pinout (Bottom View)

1.1 Pin Cross Reference Tables

Table 1-1. 82495XP Pin Cross Reference by Name

Signal	Location	Signal	Location	Signal	Location
ADS#	B15	AHOLD	A17	BGT#	M03
BLAST#	C15	BLE#	C16	BOFF# [CLEN0]	G15
BRDY#	P01	BRDYC1#	D15	BRDYC2#	F14
BUS#	P16	CACHE#	G14	CADS#	E03
CAHOLD	G04	CDC#	D03	CDTS#	F04
CFA0	E15	CFA1	B14	CFA2#	D06
CFA3	B02	CFA4	A16	CFA5	E14
CFA6	D14	CLK	D11	CMIO#	D04
CNA# [CFG0]	L04	CRDY# [SLFTST#]	M02	CWAY	J03
CWR#	E04	DC#	H14	DRCTM#	M01
EADS#	J15	FLUSH# [NCPFLD#]	N04	FPFLD# [FPFLDEN]	J04
FSIOUT#	D01	HITM# [CPUTYP]	D17	INV[CLEN1]	K15
KEN#	D16	KLOCK#	C03	KWEND# [CFG2]	M04
LEN	F15	LOCK#	B16	MALE[WWOR#]	Q02
MAOE#	S04	MAWEA#	Q17	MBALE[HIGHZ#]	P04
MBAOE#	P06	MCACHE#	C02	MCFA0	Q16
MCFA1	N14	MCFA2	R04	MCFA3	Q06
MCFA4	P15	MCFA5	P14	MCFA6	P13
MCYC#	P17	MHITM#	H04	MIO#	F16
MKEN#	R01	MRO#	J01	MSET0	Q15
MSET1	P12	MSET10	Q11	MSET2	P11
MSET3	Q14	MSET4	R16	MSET5	Q13
MSET6	R17	MSET7	S17	MSET8	P10
MSET9	Q12	MTAG0	Q10	MTAG1	P09
MTAG10	Q07	MTAG11	P07	MTAG2	Q09
MTAG3	R14	MTAG4	Q08	MTAG5	R15
MTAG6	S14	MTAG7	S15	MTAG7	S17
MTAG8	P08	MTAG9	S16	MTHIT#	G03
MWBWT#	K03	NA#	J17	NENE#	D05
PALLC#	D02	PCD	H15	PCYC	J14
PWT	C17	RDYSRC	C01	RESET	Q05
SET0	D13	SET1	C13	SET10	A09
SET2	C14	SET3	B12	SET4	C12
SET5	C11	SET6	D12	SET7	D09
SET8	D10	SET9	B09	SMLN#	C06
SNPADS#	F03	SNPBSY#	F01	SNPCLK[SNPMD]	S03
SNPCYC#	H03	SNPINV	P05	SNPNCA	Q03

Table 1-1. 82495XP Pin Cross Reference by Name (Continued)

Signal	Location	Signal	Location	Signal	Location
SNPSTB#	R03	SWEND# [CFG1]	Q01	SYNC# [MEMLDRV]	Q04
TAG0	C08	TAG1	A04	TAG10	B01
TAG11	C05	TAG2	D08	TAG3	A03
TAG4	B04	TAG5	B03	TAG6	C07
TAG7	A02	TAG8	D07	TAG9	A01
TCK	P03	TDI	N03	TDO	C04
TMS	P02	WAY	L15	WBA	M14
WBTYP	N15	WBWE#	M15	WBWT# [WRMRST]	K14
WR#	B17	WRARR#	L14		
NC A14, A15, S01, S02		Vcc A05–A08, A10–A13, E01, E17, H01, H17, K01, K17, L01, L17, C09, N17, F17, G01, G17, M17, N01, S05–S13		Vss B05–B08, B10–B11, B13, E02, E16, F02, H02, H16, J02, J16, K02, K04, K16, L02–L03, L16, C10, N16, G02, G16, R02, R05–R10, M16, N02, R11–R13	

Table 1-2. 82490XP Pin Cross Reference by Name

Signal	Location	Signal	Location	Signal	Location
A0	65	A1	66	A10	77
A11	78	A12	79	A13	80
A14	81	A15	82	A2	67
A3	68	A4	69	A5	70
A6	71	A7	73	A8	75
A9	76	ADS#	63	BE#	64
BLAST#	59	BOFF#	36	BRDY#	60
BRDYC#	61	BUS#	40	CDATA0	48
CDATA1	54	CDATA2	49	CDATA3	55
CDATA5	51	CDATA6	52	CDATA7	57
CDATA4	46	CLK	30	CRDY#	43
HITM#	62	MAWEA#	41	MBRDY# [MISTB]	22
MCLK[MSTB# #]	26	MCYC#	42	MDATA0	18
MDATA1	14	MDATA2	10	MDATA3	6
MDATA4	16	MDATA5	12	MDATA6	8
MDATA7	4	MDOE#	20	MEOC#	23
MFRZ# [MEMLDRV]	24	MOCLK[MOSTB]	27	MSEL# [MTR4# / . . .]	25
MZBT# [MX4# / . . .]	21	PAR#	32	RESET	28
TCK	3	TDI	2	TDO	84
TMS	1	WAY	45	WBA	38
WBTYP	37	WBWE#	39	WR#	58
WRARR#	44				
NC	83	Vcc 5, 9, 13, 17, 29, 35, 50, 56, 74		Vss 7, 11, 15, 19, 31, 33, 34, 47, 53, 72	

1.2 Quick Pin Reference

BGT# [C490LDRV]	I	Bus Guaranteed Transfer, [82490XP Low Drive] This signal is generated by the MBC to the 82495XP. It indicates to the 82495XP a commitment by the MBC to complete the cycle on the memory bus. Until BGT# activation the 82495XP owns the cycle and will abort it if intervening snoops happen. After BGT# the cycle is owned by the MBC until its completion. From BGT# until SWEND# snoops will be accepted, but none will be processed until SWEND# activation. During RESET's falling edge, this signal controls the driver's strength of the 82495XP to 82490XP interface signals. This strength is a function of the cache size, and therefore the number of 82490XP's. Refer to the layout specifications section for more details.
BLE#	O	BE Latch Enable The BLE# signal is used to control the enable line of an external '377-type latch. The latch captures the i860 XP CPU's BE (Byte Enable) signals and other CPU provided cycle attributes which do not go through the 82495XP.
BRDY#	I	82495XP Burst Ready This is the burst ready indication from the memory bus controller. The MBC should connect its burst ready indication to the CPU BRDY#, the 82495XP BRDY# and the 82490XP BRDY#. In the CPU, it provides the same function as that described in the CPU data sheet. The 82495XP will only use this indication for burst tracking purposes. In the 82490XP, it increments the CPU latch burst counter.
CADS#	O	Cache Address Strobe This signal is generated by the 82495XP and used by the memory bus controller. Its assertion requests execution of a memory bus cycle by the memory bus controller. This signal when active indicates that the cache cycle control and attribute signals are valid.
CAHOLD	O	82495XP AHOLD This signal is generated by the 82495XP to track the CPU AHOLD signal when used for warm-reset and LOCKed sequences. It also provides information about CPU and cache BIST.
CD/C#	O	Cache Data/Control This is a cycle definition signal driven by the 82495XP. It indicates the type of memory bus cycle requested. This signal is valid with CADS# and can be pipelined by the memory bus controller.
CDTS#	O	Cache Data Strobe This signal is driven by the 82495XP to the memory bus controller. CDTS# for read cycles indicates that in the next CLK the memory bus controller can generate the first BRDY# for the read cycle. For write cycles it indicates when data is available on the memory bus. Usage of this signal allows complete independency between address strobes (CADS#, SNPADS#) and data strobe.
CFG0-2	I	Cache Configuration bits 0-2 These signals are inputs to the 82495XP. CFG0-2 allow the 82495XP to be configured to 5 different modes. Different modes indicate 82495XP/CPU line ratio, tag size (4K/8K), lines per sector.

1.2 Quick Pin Reference (Continued)

CLK	I	Clock This signal provides the fundamental timing for the 82495XP, 82490XP and CPU. It must be provided to the 82495XP, 82490XPs, CPU and memory bus controller components with minimal skew.
CM/IO#	O	Cache Memory/IO This signal is driven by the 82495XP and is a cycle definition signal. It indicates the type of memory bus cycle requested. This signal is valid with CADS# and can be pipelined by the memory bus controller.
CNA# [CFG0]	I	82495XP Next Address Enable, [Configuration Pin 0] This signal is driven by the memory bus controller and supplied to the 82495XP. It is used by the memory bus controller to dynamically pipeline CADS# cycles. During RESET falling edge it functions as the 82495XP CFG0 input.
CRDY# [SLFTST#]	I	Cache Memory Bus Ready, [82495XP Self Test] This signal is generated by the memory bus controller and informs the 82495XP and 82490XP that a memory bus cycle has been completed. CRDY# activation ends the memory bus cycle. During RESET's falling edge, if this signal is sampled low(active) and MBALE is sampled high(active), 82495XP self test will be invoked.
CWAY	O	Cache Way CWAY is driven by the 82495XP and is a cycle definition signal that indicates to the memory bus controller the WAY to be used by the requested cycle. On line-fills it indicates the way the line will be loaded. For write-backs it indicates the WAY that was written-back. This signal is valid with CADS#.
CW/R#	O	Cache Write/Read This signal is driven by the 82495XP and is a 82495XP cycle definition signal. It indicates the type of memory bus cycle requested. This signal is valid with CADS# and can be pipelined by the memory bus controller.
DRCTM#	I	Memory Bus Direct to [M] State This signal is an input to the 82495XP. It is the mechanism by which the memory bus can dynamically inform the 82495XP of a request to skip the [E] state and move the line directly to the [M] state. This signal is sampled by the 82495XP when SWEND# is asserted.
FLUSH# [NCPFLD#]	I	Flush the 82495XP cache, [Enable Non-Cacheable PFLD] This signal is an input to the 82495XP. Flush when active will cause the 82495XP to write-back all of its modified lines into main memory then invalidate all tag locations. At the end of a flush operation the 82495XP tag array will be completely invalidated. During RESET activation, this pin functions as the NCPFLD# configuration signal which, with FPFLDEN, selects one of three modes for handling i860 XP CPU floating point load cycles.

1.2 Quick Pin Reference (Continued)

FPFLD# [FPFLDEN]	I/O	FIFO PFLD Enable [PFLD Mode Select] During RESET, FPFLDEN and NCPFLDEN# inputs select one of three modes to handle i860 XP CPU pipelined floating point load cycles. In the mode which supports an external FIFO, the FPFLD# output indicates a PFLD cycle to be loaded into the FIFO.
FSIOUT#	O	Flush/Sync/Initialization Output This signal is an output of the 82495XP and indicates the start and end of three operations: Flush, Sync, and Initialization. The output is activated when the operation internally begins and is de-activated when the operation ends.
KLOCK#	O	82495XP LOCK# This signal is driven by the 82495XP and indicates to the memory bus controller a request to execute atomic read-modify-write sequences. KLOCK# is active with the CADS# of the first LOCKed operation and remains active until at least the clock following CADS# of the last cycle of LOCKed operation.
KWEND# [CFG2]	I	Cacheability Window End, [Configuration Pin 2] This signal is generated by the MBC and indicates to the 82495XP that the Cacheability Window has expired. At this point the 82495XP will latch the memory cacheability signal (MKEN#) and make decisions based on the cacheability attribute. MRO# which indicates the Read-Only cycle attribute is also sampled at this point. During RESET's falling edge this line functions as the CFG2 configuration signal which is used to configure the 82495XP/82490XP with cache parameters.
MALE[WWOR#]	I	Memory Bus, Address Latch Enable[Weak Write Ordering] This signal is generated by the memory bus controller, and controls a 82495XP internal transparent address latch (373 like). CADS# will generate a new address at the input of the internal address latch. MALE activation(high) will allow the flowing of this address to the memory bus provided MAOE# is active. When MALE inactive(low), the address at the latch input is latched. WWOR# configures the 82495XP into strong or weak write-ordering mode.
MAOE#	I	Memory Bus Address Output Enable This signal is generated by the memory bus controller and controls the 82495XP's output buffer of the memory bus address latches. The 82495XP drives the memory bus address lines if MAOE# is active (low). Otherwise, it is tristated. MAOE# also serves as a qualifier for snooping cycles: when inactive snoops will be enabled.
MBALE[HIGHZ#]	I	Memory Bus, 82495XP sub-line-address Latch Enable[High Impedance Output] This signal has an exact function as MALE but controls only the 82495XP sub-line addresses. This signal is generated by the memory bus controller, and controls a 82495XP internal transparent address latch (373 like). CADS# will generate a new address at the input of the internal address latch. MBALE activation(high) will allow the flowing of the sub-line address to the memory bus provided MBAOE# is active. When MALE inactive(low), the sub-line address at the latch input is latched. HIGHZ#, if active along with SLFTST#, causes the 82495XP to float all of its outputs.

1.2 Quick Pin Reference (Continued)

MBAOE#	I	Memory Bus, 82495XP sub-line Address Output Enable This signal has a similar function than MAOE#, but controls only the 82495XP sub-line addresses. If MBAOE# is active(low), the 82495XP will drive the sub-line portion of the address onto the memory bus. Otherwise, it is tristated. MBAOE# is also sampled during snoop cycles. If MBAOE# is sampled inactive with SNPSTB#, the snoop write back cycle(if any) will begin at the sub-line address provided. If MBAOE# is active with SNPSTB#, the snoop write back will begin at sub-line address 0.
MBRDY#(MISTB)	I	Memory Bus Ready, (Memory Input Strobe) This pin is an input to the 82490XP. It is used in clocked bus mode to indicate the end of a transfer. When active(low) it indicates that the 82490XP should increment the burst counter and either output the next data or get ready to accept the next data. In strobed memory bus mode this pin is the input data strobe to the 82490XP. On each MISTB edge, the 82490XP latches the data and increments the burst counter.
MCACHE#	O	82495XP Internal Cacheability This signal is driven by the 82495XP. On read cycles, this signal indicates the cycle's internal cacheability attribute. In write cycles MCACHE# is only active for write-back cycles. MCACHE# is not activated for I/O, special cycles and Locked Cycles.
MCFA6-MCFA0 **MSET10-MSET0** **MTAG11-MTAG0**	I/O I/O I/O	Memory Bus Configurable address lines Memory bus SET number Memory bus TAG bits These are the memory bus address lines of the 82495XP and should be connected to the A31–A2 (A31–A3 for 64 bit bus) signals of the Memory Bus. These signals, along with the byte enables, define the physical area of memory or I/O accessed. The 82495XP drive these signals in normal memory bus cycles and have them as inputs during snooping.
MCLK[MSTBM#]	I	Memory Bus Clock, [Memory Input Strobe] In clocked memory bus mode this pin provides the memory bus clock to the 82490XP. In clocked mode, memory bus signals and memory bus data are sampled on the rising edge of the **MCLK**. In a clocked memory bus write, data is driven off of **MCLK** or **MOCLK** depending upon the configuration. This pin is an input to the 82490XP. It is sampled during reset and determines the memory bus type. If active(low), the memory bus will be strobed. If inactive (high), the memory bus will be clocked. If a clock is detected at this input, this pin becomes the memory bus clock, and clocked memory bus mode is selected.
MDATA0–MDATA7	I/O	Memory Bus Data These pins are the 8 memory data pins of the 82490XP. All or part of these pins will be used depending on the cache configuration. In clocked memory bus mode, these pins are sampled with the rising edge of **MCLK**. New data is driven out on these pins with **MEOC#** or the rising edge of **MCLK** or **MOCLK** together with **MBRDY#** active. In strobed memory bus mode, these pins are sampled on each **MISTB** edge. New data is driven out on these pins with each **MOSTB** edge.

1.2 Quick Pin Reference (Continued)

MDOE#	I	Memory Data Output Enable This signal is an input to the 82490XP. The memory bus output enable is used to control the 82490XP's driving of data onto the memory bus. When this pin is inactive(high), the **MDATA[0:7]** pins are tristated. When this pin is active(low), the **MDATA[0:7]** pins are actively driving data. The function of this pin is the same for strobed or clocked memory bus operation as MDOE# has no relation to CLK or MCLK.
MEOC#	I	Memory End of Cycle This signal is an input to the 82490XP. Since it is synchronous to the memory bus, it may be used to end a cycle on the memory bus and begin a pending cycle without waiting for synchronization to the CPU CLK. MEOC# also causes the latching or driving of data and resetting of the memory burst counter.
MFRZ# [MEMLDRV]	I	Memory Freeze, [Memory Bus Low Drive] This signal is an input to the 82490XP. It is used for write cycles that could cause allocation cycles. When this pin is active(low), write data is latched in the 82490XP. The subsequent allocation will not overwrite data latched by the write. This prevents the actual write to memory from having to be performed on the memory bus. The allocated line will be placed in the [M] state in the cache since memory has not been updated. During RESET's falling edge, this signal is sampled to indicate the 82490XP's memory bus driving strength. The 82490XP provides normal and high drive capability buffers.
MHITM#	O	Memory Bus Hit to Modified Line This signal is driven by the 82495XP during snoop cycles and indicates whether the snooping address hit a Modified line in the 82495XP cache. The 82495XP automatically schedules the writing-back of modified lines when snoop hits occur. MHITM# is activated the CLK after SNPCYC# and will remain active until the next SNPSTB#.
MKEN#	I	Memory Bus Cacheability This signal is an input to the 82495XP. It is the memory bus cache enable pin. It is used to indicate to the 82495XP if the current memory bus cycle is cacheable or not. This pin is sampled by the 82495XP with KWEND# assertion.
MOCLK(MOSTB)	I	Memory Output Clock, (Memory Output Strobe) MOCLK controls a transparent latch at the 82490XP data outputs. By providing a clock input, skewed from MCLK, MDATA hold time may be increased. In strobed bus mode this pin is the data output strobe. On each MOSTB edge, new data will be output onto the memory bus.
MRO#	I	Memory Bus Read-Only This pin is an input to the 82495XP. It is the READ-ONLY attribute pin. It is used to indicate to the 82495XP that the accessed line should get a READ-ONLY attribute. READ-ONLY lines will be non-cacheable in the first level cache. READ-ONLY lines will be cached in the 82495XP if MKEN# is sampled active during KWEND# and will be cached in the [S] state. This pin is sampled by the 82495XP with KWEND# assertion.

1.2 Quick Pin Reference (Continued)

MSEL# [MTR4/TR8#]	I	Memory Select, [Memory Transfer] This signal is a chip select input to the 82490XP. MSEL# activation qualifies the MBRDY# input of the 82490XP. MSEL# going active causes the sampling of MZBT# for the next cycle. MSEL# going inactive resets the 82490XP's internal memory burst counter. This pin is used to determine the number of transfers necessary on the memory bus for each cache line. If high, there are 4 transfers on the memory bus for each cache line. If low, there are 8 transfers on the memory bus for each cache line.
MTHIT#	O	Memory Bus Tag Hit This signal is driven by the 82495XP during snoop cycles. It indicates whether the snooping address hit any line (exclusive, shared, or modified) in the 82495XP cache. MTHIT# is activated the CLK after SNPCYC# and will remain active until the next SNPSTB#.
MWB/WT#	I	Memory Bus Write Policy This signal is an input to the 82495XP. It is the mechanism by which the memory bus can dynamically inform the 82495XP of the cycle write policy (Write-Through/Write-Back). This signal is sampled by the 82495XP with SWEND# activation.
MZBT# [MX4/MX8#]	I	Memory Zero Based Transfer, [Memory I/O Bits] This signal is an input to the 82490XP. When this pin is sampled active (with MSEL# or MEOC#) it indicates that the memory bus cycle should start with burst location zero independent of the sub-line address requested by the CPU. This pin is used to determine the number of IO pins used for the memory bus. When HIGH it indicates that 4 IO pins are used per 82490XP. When LOW it indicates that 8 IO pins are used.
NENE#	O	Next Near This signal is generated by the 82495XP and indicates to the memory bus controller if the address of the requested memory cycle is "near" the address of the previously generated one (in the same 2K DRAM page). This information can be used by the memory bus controller to optimize access to paged or static column DRAMs. This signal is valid together with CADS#.
PALLC#	O	Potential Allocate This signal is generated by the 82495XP and indicates to the memory bus controller that the current write cycle can potentially allocate a cache line. Potential allocate cycles are cycles which are 82495XP misses with PCD, PWT inactive.
RDYSRC	O	Ready Source This signal is an output of the 82495XP. It indicates the source of the BRDY generation for the CPU. When high it indicates that the memory bus controller should generate BRDYs to the CPU, when low it indicates that the 82495XP will be the one providing BRDYs.

1.2 Quick Pin Reference (Continued)

RESET	I	Reset This signal forces the 82495XP and 82490XP to begin execution at a known state. It's falling edge will sample the state of the configuration pins. RESET is an asynchronous input to the 82495XP and 82490XP. **The following 82495XP pins are sampled during reset falling edge:** CNA# [CFG0]: CFG0 line of 82495XP configuration inputs. SWEND# [CFG1]: CFG1 line of 82495XP configuration inputs. KWEND# [CFG2]: CFG2 line of 82495XP configuration inputs. FLUSH# [NCPFLD#]: Enables decoding of the non-cacheable PFLD mode. Active if low. FPFLD# [FPFLDEN]: Enables the external FIFO pfld mode. Active high. BGT# [C490LDRV]: Indicates the driving strength of the 82495XP/82490XP interface. If high, the 82495XP can drive up to 10 82490XP's without derating. If low, the 82495XP can drive up to 18 82490XP's without derating. SYNC# [MEMLDRV]: Indicates the 82495XP's memory bus driving strength. SNPCLK[SNPMD]: Indicates the snoop mode, synchronous or asynchronous. CFG0–CFG2 signals are used to configure the 82495XP/82490XP with cache parameters. They define the lines/sector, line ratio, and number of tags. MALE[WWOR#]: Enforces strong or weak write-ordering consistency. MBALE[HIGHZ#]: If active along with SLFTST# will tristate all 82495XP outputs. **The following 82490XP pins are sampled during reset falling edge:** PAR#: If active(low), this pin configures the 82490XP as a parity storage device. The parity configuration stores the paritybits belonging to data stored in other 82490XP's. MZBT# [MX4/MX8#]: Determines the number of IO pins used for the memory bus interface. If high, four IO pins are chosen. If low, eight IO pins are chosen. MSEL# [MT4/MT8#]: Determines the number of transfers necessary on the memory bus for each cache line. If high, four memory bus transfers are needed to fill a cache line. If low, eight memory bus tranfers are needed to fill a cache line. MCLK[MSTBM#]: If active(low), this pin indicates a strobed memory bus configuration. If inactive(high), a clocked memory bus is chosen. MFRZ# [MEMLDRV]: Indicates the 82490XP's memory bus driving strength.
SMLN#	O	Same Cache Line This signal is an output of the 82495XP. It is used to indicate to the memory bus controller that the current cycle is to the same 82495XP line as the previous one. This indication can be used by the memory bus controller to selectively activate its SNPSTB# signal to other caches. For example, back to back snoop hits to the same line may be snooped only once. This signal is valid together with CADS#.

1.2 Quick Pin Reference (Continued)

SNPADS#	O	Cache Snoop Address Strobe This signal is an output of the 82495XP. It has an identical functionality as CADS#, but is generated only on snooping-write-back cycles. Considering that snoop write-back cycles are the only ones which are generated independent of CPU bus activity, this separate address strobe should ease implementation of the memory bus controller. Whenever active, the memory bus controller should abort all pending cycles (cycles for which BGT# was not issued yet. After BGT# the memory bus controller is responsible for the cycle completion). The 82495XP assumes that non-committed cycles are aborted upon SNPADS# and may re-issue them again after the completion of the snoop.
SNPBSY#	O	Snoop Busy This signal is driven by the 82495XP. When inactive(high), it indicates that the 82495XP is ready to accept another snoop cycle. SNPBSY# will be activated for one of two reasons: A snoop hit to a modified line, a back-invalidation is needed when there is one already in progress. In either of these cases, the 82495XP will not perform the look-up for a pending snoop until SNPBSY# is de-activated.
SNPCLK[SNPMD]	I	Snoop Clock [Snoop Mode] This pin provides the 82495XP with the snoop clock to be used in clocked memory interfaces. During clocked mode SNPSTB#, SNPINV, SNPNCA, MBAOE#, MAOE#, and the Address lines will be sampled by SNPCLK. During RESET activation, this pin functions as the SNPMD (snoop mode) signal. If high it indicates strobed snooping mode. If low it indicates synchronous snooping mode. For clocked snooping mode, SNPCLK is connected to the snoop clock source.
SNPCYC#	O	Snoop Cycle This signal is an output of the 82495XP. It indicates when the snooping look-up is actually taking place in the 82495XP tag RAM.
SNPINV	I	Snoop Invalidation This signal is an input to the 82495XP and indicates the resulting line state in case of a snoop hit cycle. If active, it forces the line to go to an invalid state. This signal is sampled with SNPSTB#.
SNPNCA	I	Snoop Non Caching Device Access This signal is an input to the 82495XP and provides the 82495XP information on whether the current memory bus master is a non caching device (DMA, etc). This indication allows the 82495XP to avoid changing line states from exclusive to shared unnecessarily.
SNPSTB#	I	Snoop Strobe This signal is an input to the 82495XP which is used to initiate a snoop. SNPSTB# causes the latching of the snoop address and parameters. The 82495XP supports three latching modes: Clocked, Strobed, Synchronous. In the clocked mode, address and attribute signals will be latched with the activation of SNPSTB#.SNPCLK. In the strobed mode, address and attributes will be latched by the SNPSTB# falling edge. In synchronous mode, address and attribute signals will be latched with the activation of SNPSTB#.CLK.

1.2 Quick Pin Reference (Continued)

SWEND# [CFG1]	I	Snoop Window End, [Configuration Pin 1] This signal is generated by the MBC and indicates to the 82495XP that the Snoop Window has expired. At this point the 82495XP will latch the memory bus attributes: write policy (MWB/WT#), and direct to [M] transfer (DRCTM#). At the end of the snooping window, all other devices have snooped the bus master's address and have generated address caching attributes on the bus. Once a cycle begins, the 82495XP prevents snooping until it has received SWEND#. The 82495XP will act based on those attributes and will update its tag RAM. During RESET's falling edge this line functions as the CFG1 configuration signal which is used to configure the 82495XP/82490XP with cache parameters.
SYNC# [MEMLDRV]	I	Synchronize 82495XP cache, [Memory Bus Low Drive] This signal is an input to the 82495XP. Activation of this line will cause the synchronization of the 82495XP tag array with main memory. All 82495XP modified lines will be written back to main memory. The difference between FLUSH and SYNC is that on SYNC the 82495XP and CPU tag array will NOT be invalidated. All the valid entries will be kept, with all modified lines (M state) becoming non-modified (E state). During RESET's falling edge, this signal is sampled to indicate the memory bus driving strength. If it is sampled low, the maximum capacitive load without derating is 100pf. If it is sampled high, the maximum capacitive load without derating is 50pf.
TCK	I	Testability Clock This signal is an input to both the 82495XP and 82490XP. This is the boundary scan clock. This signal has to be connected to a clock synchronous to CLK to insure initialization of the test logic.
TDI	I	Testability serial input This signal is an input to both the 82495XP and 82490XP.
TDO	O	Testability serial output This signal is an output of both the 82495XP and 82490XP.
TMS	I	Testability Control This signal is an input to both the 82495XP and 82490XP.

The following pins have internal pull-ups:

ADS#, NA#, FPFLD#, TDI, TMS, BGT#, KWEND#, SWEND#, CNA#, BRDY#, SYNC#, FLUSH#, SNPSTB#, MRO#, DRCTM#, TCK, SNPCLK, MFRZ#, MZBT#, MCLK, MOCLK.

During tri-state output testing sequence, all pull-ups will be disabled.

The following signals are glitch free. These signals are always at a valid logic level following RESET:

CADS#, CDTS#, SNPADS#, SNPCYC#.

1.3 Output Pins

Table 1-3 lists all output pins, from which part(s) they are driven, and their active levels.

Table 1-3. Output Pins

Name	Part	Active Level	Name	Part	Active Level
BLE#	82495XP	LOW	MTHIT#	82495XP	LOW
CADS#	82495XP	LOW	NENE#	82495XP	LOW
CAHOLD	82495XP	HIGH	PALLC#	82495XP	LOW
CDTS#	82495XP	LOW	RDYSRC	82495XP	HIGH
CWAY	82495XP	-	SMLN#	82495XP	LOW
CW/R#,CD/C#,CM/IO#	82495XP	-	SNPADS#	82495XP	LOW
FSIOUT#	82495XP	LOW	SNPBSY#	82495XP	LOW
KLOCK#	82495XP	LOW	SNPCYC#	82495XP	LOW
MCACHE#	82495XP	LOW	TDO	82495XP/82490XP	-
MHITM#	82495XP	LOW			

1.4 Input Pins

Table 1-4 lists all input pins, which part(s) they are input to, their active level, and whether they are synchronous or asynchronous inputs.

Table 1-4. Input Pins

Name	Part	Active Level	Synchronous/Asynchronous
BGT# [C490LDRV]	82495XP	LOW	Synchronous to CLK
BRDY#	82495XP/82490XP	LOW	Synchronous to CLK
CLK	82495XP/82490XP	-	-
CFG3	82495XP	-	Synchronous to CLK
CNA# (CFG0)	82495XP	LOW	Synchronous to CLK
CRDY# [SLFTST#]	82495XP/82490XP	LOW	Synchronous to CLK
DRCTM#	82495XP	LOW	Note 2
FLUSH# [NCPFLD#]	82495XP	LOW	Asynchronous
CPUTYP	82495XP	LOW	Synchronous to CLK
KWEND# (CFG2)	82495XP	LOW	Synchronous to CLK
MALE, MBALE	82495XP	HIGH	Asynchronous
MAOE#, MBAOE#	82495XP	LOW	Asynchronous
MCLK[MSTBM#]	82490XP	LOW	Synchronouos to MCLK
MBRDY# (MISTB)	82490XP		-
MDOE#	82490XP	LOW	Asynchronous
MEOC#	82490XP	LOW	Synchronous/Asynchronous, Note 1

Table 1-4. Input Pins (Continued)

Name	Part	Active Level	Synchronous/Asynchronous
MFRZ#	82490XP	Low	Synchronous/Asynchronous, Note 1
MOCLK(MOSTB)	82490XP		
MSEL[MTR4/TR8#]	82490XP	Low	Synchronous/Asynchronous, Note 1
MZBT#[MX4/MX8#]	82490XP	Low	Synchronous/Asynchronous, Note 1
MKEN#	82495XP	LOW	Note 2
MRO#	82495XP	LOW	Note 2
MWB/WT#	82495XP	-	Note 2
PAR#	82490XP	Low	Synchronous to CLK
RESET	82495XP/82490XP	HIGH	Asynchronous
SNPCLK[SNPMD]	82495XP	-	-
SNPINV	82495XP	HIGH	Note 3
SNPNCA	82495XP	HIGH	Note 3
SNPSTB#	82495XP	LOW	Note 3
SWEND# (CFG1)	82495XP	LOW	Synchronous to CLK
SYNC#[MEMLDRV]	82495XP	LOW	Asynchronous
TCK	82495XP/82490XP	-	
TDI	82495XP/82490XP	-	Synchronous to TCK
TMS	82495XP/82490XP	-	Synchronous to TCK

NOTES:
(1) In Clocked memory bus mode these pins are synchronous with **MCLK**. In Strobed memory bus mode these pins are asynchronous.
(2) MWB/WT#, DRCTM# must be synchronous to CLK during SWEND#. MKEN#, MRO# must be synchronous to CLK during KWEND#.
(3) In clocked memory bus mode these pins are synchronous with SNPCLK. In strobed memory mode these pins are asynchronous.

1.5 Input/Output Pins

Table 1-5 lists all input/output pins, which part they interface with, and when they are floated.

Table 1-5. Input/Output Pins

Name	Part	Synch/Asynch	When Floated
FPFLD#[FPFLDEN]	82495XP	Synchronous to CLK	-
MCFA0–MCFA6	82495XP	Note 1	MAOE# = High
MDATA0–MDATA7	82490XP	Note 2	MDOE# = Hight and during Reset
MSET0–MSET10	82495XP	Note 1	MAOE# = High
MTAG0–MTAG11	82495XP	Note 1	MAOE# = High

NOTES:
(1) With MALE high and MAOE# low, these pins are synchronous to CLK.
(2) In Clocked memory bus mode these pins are synchronous with **MCLK**. In Strobed memory bus mode these pins are asynchronous.

1.6 Pin State During Reset

Table 1-6. Pin State During Reset

Pin Name	Pin State during Reset
CADS#, CDTS#, SNPADS#	High
CW/R#, CD/C#, CM/IO#, MCACHE#	Undefined
RDYSRC, PALLC#, CWAY	Undefined
NENE#, SMLN#	Undefined
KLOCK#	High
FPFLD#	High
MSET0–MSET10, MTAG0–MTAG11, MCFA0–MCFA6	Note 1
CAHOLD	Note 2
MHITM#, MTHIT#	High
SNPCYC#, SNPBSY#	High
TDO	Note 3

NOTES:
(1)MSET, MTAG, and MCFA signals are high impedance during reset if MAOE# and MBAOE# are deasserted.
(2) The state of CAHOLD depends on whether self-test is selected (see testability chapter for details).
(3) The State of TDO is controlled by the boundary scan which is independent of other signals including RESET (see testability chapter for details).

2.0 CHIPSET INTRODUCTION

The 82495XP/82490XP is a second-level cache controller chipset for the i860 XP CPU. The chipset provides a unified code and data cache which is software transparent. The 82495XP/82490XP has been designed to support a high-speed CPU/cache core interface, and a same or lower speed memory bus interface.

The 82495XP is the cache controller. It contains 8K tags and control logic to control up to a 512K size cache. The 82490XP is a custom cache data RAM designed to be used with the 82495XP. Between 8 and 18 82490XPs are required to create a 256K to 512K cache, respectively. The memory bus controller (MBC) is the set of logic required to interface the 82495XP and 82490XP to the memory bus. The MBC provides product differentiation, and its implementation ultimately determines system performance.

2.1 Main Features

The 82495XP/82490XP have the following main features:

— Tracks the speed of the i860 XP CPU
— Large Cache Size support:
  4K or 8K Tags
  1 or 2 lines per sector
  4 or 8 transactions per line
  64 or 128-bit wide memory bus
  256K or 512K cache
— Write-Back cache with full multiprocessing consistency support:
  supports the MESI protocol
  watches memory bus to guarantee 1st level, 2nd level cache consistency
  maintains inclusion
— Two-way set-associative with MRU hit prediction algorithm

— Zero wait state hit cycles on MRU hit. One wait state on MRU misses
— Concurrent CPU and Memory Bus transactions
— Supports synchronous, asynchronous, and strobed memory bus architectures

2.2 CPU/Cache Core Description

Figure 2-1 depicts a block diagram of the basic cache subsystem. The cache subsystem provides a gateway between the CPU and the memory bus. All CPU accesses which can be serviced locally by the cache subsystem will be filtered out from the memory bus traffic. Therefore local cycles (CPU cycles which hit the cache and do not require a memory bus cycle) will be completely invisible to the memory bus providing the reduction in memory bus bandwidth necessary for multiprocessing systems. Another very important function of the 82495XP cache subsystem is to provide speed decoupling between the CPU and memory busses. Processors are quickly achieving operating frequencies which can be very difficult for the memory subsystem to meet. The 82495XP cache subsystem is optimized to serve the CPU with zero wait-states up to very high frequencies (50 Mhz), at the same time providing the decoupling necessary to run slower memory bus cycles.

The Basic Functions of the cache subsystem elements are:

82495XP: Main control element, includes the tags and line states and provides hit or miss decisions. It handles the CPU bus requests completely and coordinates with the memory bus controller when an access needs the memory bus. It controls the 82490XP data paths for both hits/misses to provide the CPU with the correct data. It dynamically adds wait states based on the MRU prediction mechanism. The 82495XP is also responsible for performing memory bus snoop operations while other devices are using the memory bus. The 82495XP drives the cycle address and other attributes during a memory bus access. A block diagram of the 82495XP is shown in Figure 2-2.

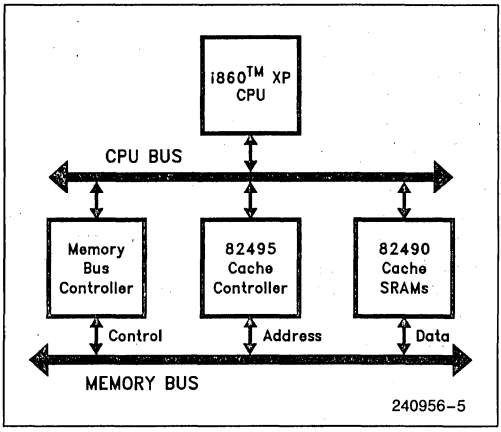


Figure 2-1. 82495XP Cache Subsystem

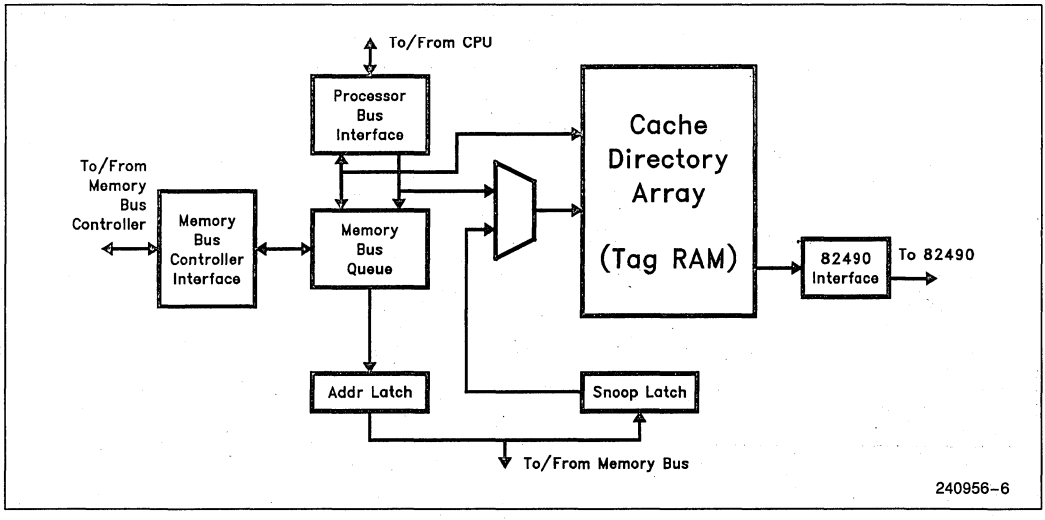


Figure 2-2. 82495XP Block Diagram

82490XP: Implements the cache SRAM storage and data path. It includes latches, muxes, logic which allow it to work in lock-step with the 82495XP to efficiently serve both hit and miss accesses. It takes full advantage of internal silicon flexibility to provide a degree of performance otherwise unachievable with discrete implementations. It supports zero wait state hit accesses, concurrent CPU and memory bus accesses, and includes a replication of the MRU bits for autonomous way prediction. During memory bus cycles it acts as a gateway between CPU and memory buses. A block diagram of the 82490XP is shown in Figure 2-3.

Memory Bus Controller: Server for memory bus cycles. It adapts the CPU/Cache core to a specific memory bus protocol. It coordinates with the 82495XP line fills, flushes, write-backs, etc. The memory bus controller's flexibility allows customers to easily adapt the 82495XP cache subsystem to their specific architectures, and to provide their own differentiation. Figure 2-4 shows an example memory bus controller. The MBC handles all cycle control, data transferring, snooping, and any synchronization.

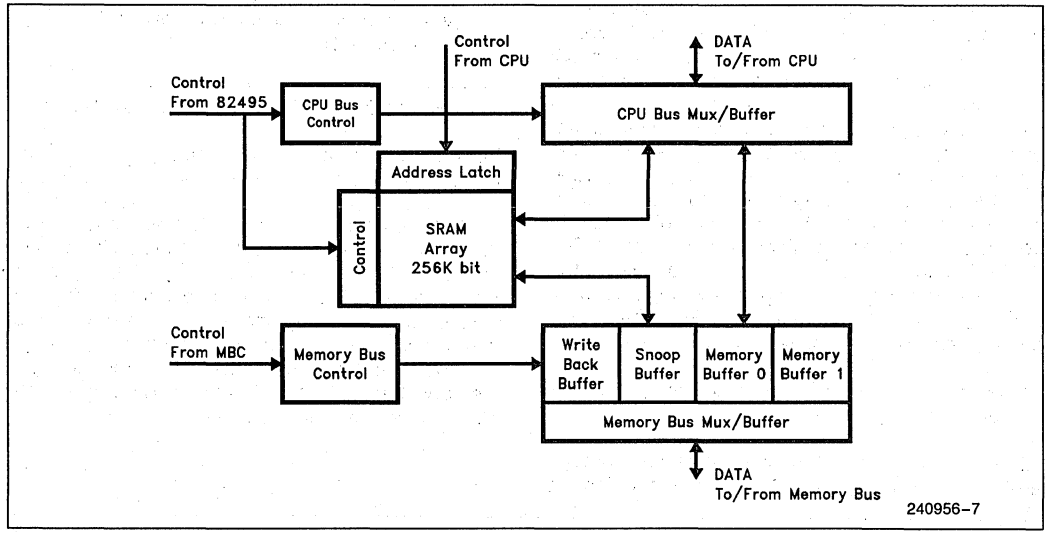


Figure 2-3. 82490XP Block Diagram

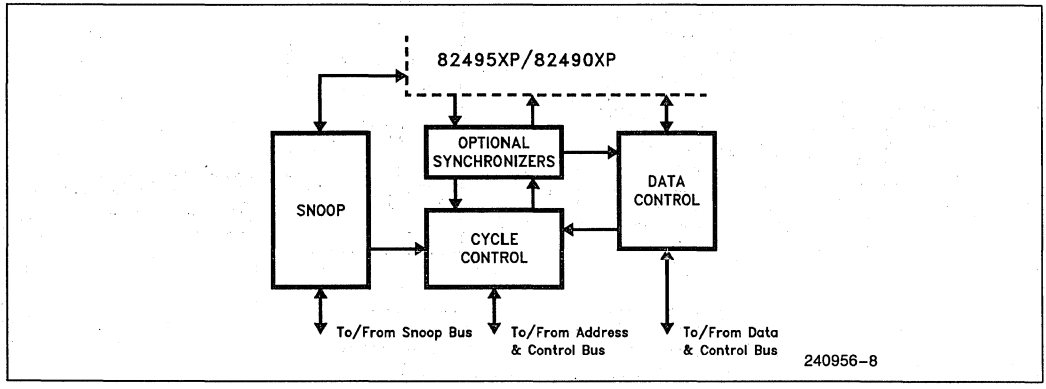


Figure 2-4. MBC Example Block Diagram

3.0 CACHE OVERVIEW

This chapter gives a brief description of 82495XP/82490XP configurations, interface, snooping mechanism, cycle control mechanism, and memory bus control mechanism. Each section of this overview is described in more detail in later chapters.

3.1 Configuration

The 82495XP/82490XP cache chipset offers a number of configuration options. The system designer can choose from a number of different operating characteristics, including memory bus modes, snooping modes, and internal physical attributes (line size, lines per sector, etc.). The flexibility of these configuration options allow the 82495XP/82490XP cache to be used in a wide range of applications.

Configurations are selected by altering the 82495XP/82490XP inputs during RESET. They are not dynamically changeable, and to conserve pins some configuration inputs become 82495XP or 82490XP inputs/outputs after RESET.

3.1.1 PHYSICAL CACHE

Physically, the 82495XP/82490XP can be configured to support many different cache configurations. By selecting one cache configuration, other configurations may be excluded. The 82495XP/82490XP can be configured to support:

— 256K or 512K cache
— 64 or 128 bit wide memory bus
— One or two lines per sector
— 1:1, 1:2, or 1:4 CPU to 82495XP line size ratio
— 4 or 8 memory bus transactions per line
— 4K or 8K tag size
— Strong or weak write ordering

Figure 3-1 summarizes the basic configurations available when using the 82495XP/82490XP.

3.1.2 SNOOP MODES

When another master snoops the 82495XP, the MBC must initiate the snoop request and pass on the response. The 82495XP allows the MBC to initiate this snoop request in one of three modes: synchronous, clocked, and strobed. The snoop response of the 82495XP is always synchronous.

When initiating the snoop in synchronous snoop mode, all snoop information is latched by the 82495XP synchronous to the CPU CLK. The snoop is then performed on the next CLK edge and the response given on the CLK edge after that. This is the fastest possible method of snooping.

In clocked snooping mode, information is latched by the 82495XP with respect to an external snoop clock (slower than CLK) source. The 82495XP must internally synchronize this information to CLK and provide a response.

In strobed snooping mode, information is latched into the 82495XP with respect to the falling edge of another signal. Thus, the snoop initiation is clock independent. The 82495XP again synchronizes this information with CLK.

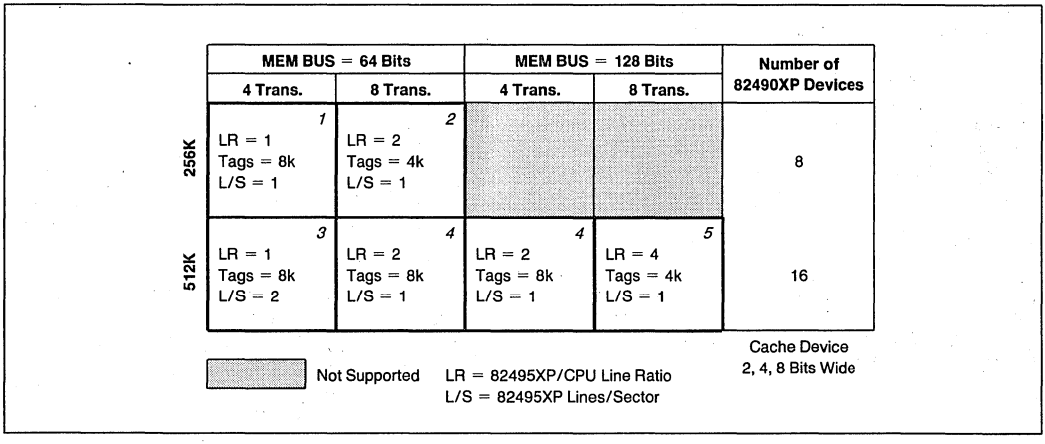


Figure 3-1. 82495XP/82490XP Configurations

3.1.3 MEMORY BUS MODES

The 82490XP may be configured to be in one of two memory bus modes. This mode determines how data will be passed on to and off of the data bus. The two modes are clocked mode and strobed mode. These modes need not have any relation to the snoop mode chosen.

In clocked mode, data is driven from an external memory clock source called MCLK, or read with respect to MCLK. MCLK is completely independent of the CPU CLK source. There are inherent performance advantages, however, in making this clock source synchronous or half-clock (divided) synchronous to the CPU CLK.

In strobed mode, data is driven from the rising edge of one signal, and read with the rising edge of another. Like the strobed snooping mode, this carries no clock skew problems, or memory bus speed limitations.

3.2 CPU Bus Interface

The CPU bus interface is the connection of the 82495XP and 82490XP to the i860 XP CPU. Because this interface is optimized to achieve the high speed performance, it is not a flexible interface. The majority of the signals in the CPU bus interface must be connected strictly between the 82495XP/82490XP cache and the i860 XP CPU. Chapter 10 addresses the use of such signals.

Some CPU signals are, however, accessible by the MBC. These are the following pins: RESET, CLK, BRDY2#, INT, BERR, PCHK#, PEN#, TCK, TDI, TMS, TRST#, and TDO. CPU pins KB0, KB1, HIT#, and BREQ are also available to the MBC, but are of limited use in an 82495XP/82490XP system.

Other CPU pins flow through a '377 type latch to the MBC. The latch enable is controlled by the 82495XP through the BLE# pin. The following CPU signals flow through this latch: PCD, PWT, BE0#–BE7#, CACHE#, LEN, PCYC, and CTYP.

3.3 82495XP/82490XP Interface

The 82495XP/82490XP interface is the connection between 82495XP and 82490XP. Like the CPU bus interface, this isolated interface is not flexible and may not be altered beyond what Intel has provided.

3.4 Memory Bus and Memory Bus Controller Interface

The memory bus controller (MBC) is the interface logic required to control the 82495XP/82490XP and connect it to the memory bus and rest of the system. The MBC may be simple enough to support a single-CPU write-through cache, or complex enough to support a multiprocessing cache with external tags. The 82495XP/82490XP is a very flexible chipset, and the MBC determines exactly how the 82495XP/82490XP will work in a system.

An MBC consists of a few basic blocks: a snoop logic block, a cycle control block (with synchronizers if necessary), and data path control block. The snoop block must be able to communicate with the other caches when snooping is necessary. At the same time, the cycle control block must interface to some arbitration logic for bus arbitration.

3.4.1 SNOOPING LOGIC

The MBC snooping logic is responsible for initiating a snoop in the 82495XP and providing the response to the rest of the system. Snoop logic must recognize what other caches are doing, and snoop if necessary. Snoop logic must also recognize when its 82495XP is not capable of snooping and delay its snoop initiation.

When a cycle begins on the bus, all other caches snoop. Once all the snoop results are returned to the master 82495XP, its snoop logic must recognize the result and alter the cycle appropriately. This could mean aborting the current cycle in memory, delaying the cycle until a write-back is performed, or changing the master's tag state according to the snoop information.

3.4.2 CYCLE CONTROL LOGIC

Cycle control logic is responsible for initiating a memory bus cycle, providing proper 82495XP cycle attributes during the cycle, and terminating the cycle. Cycle control logic determines the cacheability of the cycle, whether cycles are allocatable, pipelining, and all aspects of the progress of the current cycle.

Since cycle control logic interfaces memory bus signals to the 82495XP, and since the memory bus is not necessarily synchronous to the 82495XP CLK, it may also provide proper synchronization. Careful design of this synchronization logic can minimize or eliminate synchronization penalties.

3.4.3 DATA PATH CONTROL

Data path control logic controls how data is written from the 82490XP or read into the 82490XP and CPU. It handles the actual transferring of data to/from the memory data bus. Data path control logic also handles the CPU burst order, and the holding of data during allocation cycles. In systems with memory busses that are wider than the CPU bus, the data path control logic appropriately steers data to the correct 82490XP's.

3.5 Test

The 82495XP/82490XP provide two means of cache testing. These are a built-in self-test, and boundary scan test. The built-in self-test (BIST) is initiated during RESET. The boundary scan test uses separate and dedicated pins on the 82495XP. These are described in a later chapter.

4.0 CACHE CONSISTENCY PROTOCOL

One of the 82495XP objectives is to implement a high performance second level cache for multiprocessor systems. To fulfill this objective the 82495XP implements a "write-back" cache with full support for multiprocessing data consistency. Being a write-back cache means that the 82495XP may contain data which is not updated in the main memory. Therefore a mechanism is implemented to insure that data read by any system bus master, at any time, is correct.

A key feature for multiprocessing systems is reduction of the memory bus utilization. The memory bus quickly becomes a resource bottleneck with the addition of multiple processors. The 82495XP cache consistency mechanism insures minimal usage of memory bus bandwidth.

The 82495XP allows portions of memory to be defined as non-cacheable. For the cacheable areas, the 82495XP allows selected portions to be defined as write-through locations.

The 82495XP protocol is implemented by assigning state bits for each cached line. Those states are dependent on both 82495XP data transfer activities performed as the bus master, and snooping activities performed in response to snoop requests generated by other memory bus masters.

4.1 Cache Consistency Protocol Model

The 82495XP consistency protocol is the set of rules which allows the 82495XP to contain data that is not updated in main memory while ensuring that memory accesses by other devices do not receive stale data. This consistency is accomplished by assigning a special consistency state to every cached entry (line) in the 82495XP.

NOTE:
The following rules apply to memory read and write cycles. All I/O and special cycles bypass the cache.

The 82495XP protocol consists of 4 states. They define whether a line is valid (hit or miss), if it is available in other caches (shared or exclusive), and if it is modified (has been modified).

The 4 States are:

[I] - INVALID — Indicates that the line is not available in the cache. A read to this line will be a miss and cause the 82495XP to execute a line fill (fetch the whole line and deposit it into the cache SRAM). A write to this line will cause the 82495XP to execute a write-through cycle to the memory bus and in some circumstances initiate an ALLOCATION.

[S] - SHARED — This state indicates that this line is potentially shared with other caches (The same line may exist in more than one cache). A **S**hared line can be read out of the cache SRAM without a main memory access. Writing to a **S**hared line updates the 82495XP/82490XP cache, but also requires the 82495XP to generate a write-through cycle to the memory bus. In addition to updating main memory, the write-through cycle will invalidate this line in other caches. Since writing to a **S**hared line causes a write-through cycle, the system can enforce a "write-through policy" to selected addresses by forcing those addresses into the [S] state. This can be done by setting the PWT attribute in the CPU page table or asserting the MWB/WT# pin each time the address is referenced.

[E] - EXCLUSIVE This state indicates a line which is exclusively available in ONLY this cache, and that this line is NOT MODIFIED (main memory also has a valid copy). Writing to an **E**xlusive line causes it to change to the **M**odified state and can be done without informing other caches, so no memory bus activity is generated.

[M] - MODIFIED This state indicates a line which is exclusively available in ONLY this cache, and is MODIFIED (main memory's copy is stale). A **M**odified line can be updated locally in the cache without acquiring the memory bus. Because a **M**odified line is the only up-to-date copy of data, it is the 82495XP's responsibility to flush this data to memory on accesses to it. Flushing of this data to memory will be executed immediately after completion of the current CPU bus cycle.

4.2 Basic State Transitions

This section covers the most common, basic memory accesses. The special functions which force a cycle to be noncacheable, locked, read only, or direct-to-**M**odified are not in use. These might be used, for example, in read for ownership and cache to cache transfers, and are covered in section 4.3. This basic transitions section is divided into two parts: the first covers MESI state changes which occur in a CPU/cache core due to its own actions; the second describes MESI state transitions in a CPU/cache core caused by the actions of other, external devices. Figure 4-1 shows a partial state diagram of the MESI coherency protocol which includes these basic transitions.

The 82495XP accepts line attributes from the CPU and memory buses. The 82495XP assumes that all caches on the memory bus have the SAME number of bytes per line.

4.2.1 TRANSITIONS IN CACHE STATES CAUSED BY OWN CPU TRANSACTIONS

The MESI state of each 82495XP/82490XP cache line changes as the 82495XP/82490XP services the read and write requests generated by its CPU.

4.2.1.1 Read Hit

A read hit occurs when the CPU generates a read cycle on its bus, and the data is present in and returned by the 82495XP/82490XP. The state of the cache line (M, E, or S) remains unchanged by a read operation which hits the cache.

4.2.1.2 Read Miss

A read miss arises when the CPU generates a read, and the data is not present in the 82495XP/82490XP cache—either the tag lookup does not produce a match or a match occurs but the data is **I**nvalid. The 82495XP generates a memory access to fetch the data (which is assumed cacheable for this discussion) and the surrounding data needed to fill the cache line. This data is placed in the 82495XP/82490XP cache in an invalid line or (if both valid) replaces the least recently used line, which is written back to memory if **M**odified.

The new line is placed in the **E**xclusive state, unless either the CPU or memory indicates that it should be a write-through on its next write access using PWT or MWB/WT#, respectively. If either of these is asserted, the new line is placed in **S**hared state. A new line could also be read in and placed directly into **M**odified state: see section 4.3.4 for details and use.

4.2.1.3 Write Hit

When the CPU generates a write cycle, if the data is present in the 82495XP/82490XP cache, it is updated and may undergo a MESI state change.

If the hit line is originally in the **E**xclusive state, it changes to **M**odified state upon a write. If the hit line is originally in the **M**odified state, it remains in that state. Neither of these cases generates any bus activity.

A write to a line which is in the **S**hared state causes the 82495XP to write the data out to memory as well as update the 82495XP/82490XP cache. The write to main memory also serves to invalidate any copy of the data which resides in another cache. The cache line state changes according to activity on the PWT and MWB/WT# pins. If neither of these pins is asserted, the write hit line becomes Exclusive. If either of these pins is asserted, the line is forced to remain write-through, so the state remains **S**hared.

An existing line can also be written and forced directly into Modified state: see section 4.3.4 for details and use.

4.2.1.4 Write Miss

The CPU generates a write cycle, and the data is not present in the 82495XP/82490XP cache. In a simple write miss, the 82495XP/82490XP assists CPU in delivering data to memory, but the data is not placed in the cache. No cache lines are affected, so no state changes take place.

4.2.1.5 Write Miss with Allocate

This is a special case of a write miss where the memory location written by the CPU is not currently in the 82495XP/82490XP cache, but is brought into the cache and updated. Like a regular write miss, the 82495XP/82490XP assists the CPU in writing the data out to main memory. After the data is written to memory, the 82495XP/82490XP reads back the same data following the rules of a read miss, above.

The ability to perform an allocation depends on all of the following conditions:

the write is cacheable

PWT is not asserted, forcing write-through

the write is not LOCKed

the write is to memory (not to I/O)

4.2.2 TRANSITIONS CAUSED BY OTHER DEVICES ON BUS

MESI state transitions in the 82495XP/82490XP cache of one core (CPU/82495XP/82490XP) can be induced by actions initiated by other cores or devices on the shared memory bus. In the following, the 82495XP which is responding to actions of other devices does not currently own the bus, and may be referred to as a "slave" or, in the case of snooping, a "snooper". The device which currently owns the bus is the "master".

4.2.2.1 Snooping

The master which is accessing data from memory on the bus sends a request to all caching devices on the bus (snoopers) that they check or snoop their caches for a more recently updated version of the data being accessed. If one of the snoopers has a copy of the requested data, it is termed a "snoop hit".

If a snooper has a modified version of the data ("snoop hit to a Modified line"), it proceeds to generate an "inquire cycle" to the i860 XP CPU, asking the i860 XP CPU if it also has a Modified copy of the line (which would be more recently modified than the 82495XP/82490XP's version). The most up-to-date line is written out by the snooping 82495XP/82490XP to the bus (to main memory or directly to the requesting master) so that the requesting master can utilize it.

The changes in MESI protocol state in a snooping cache which has a snoop hit depend on attribute inputs SNPINV and SNPNCA, which are driven by the master.

The SNPINV input tells a snooping 82495XP/82490XP to invalidate the line being snooped if hit: the master requesting the snoop is about to write to its copy of this line and will therefore have the most up-to-date copy. When SNPINV is asserted on the snoop request, any snoop hit is placed in Invalid state, and a "back invalidation" is generated which instructs the CPU to check its cache and likewise invalidate a copy of the line. When the snooping 82495XP has a snoop hit to a Modified line and SNPINV was asserted by the bus master, the back invalidate is combined with the inquire cycle.

The SNPNCA input tells a snooping 82495XP/82490XP whether the requesting master is performing a Non-Caching Access. If the requesting master is not caching the data, a snoop hit to a Modified or Exclusive line can be placed in the Exclusive state: since the requester isn't caching the

line, if the snooper has a future write hit to the line, an invalidation does not have to be broadcast. If the requesting master is caching the data, then a snoop hit to a **M**odified or **E**xclusive line must be placed in the **S**hared state, which insures that a future write hit causes an invalidation to other caches. Note that a snoop hit to a **S**hared line must remain in the **S**hared state regardless of SNPNCA. Also note that an asserted SNPINV always overrides SNPNCA.

4.2.2.2 Cache Synchronization

Cache synchronization is performed to bring the main memory up-to-date with respect to the 82495XP/82490XP. Two devices exist in the 82495XP/82490XP to accomplish this: FLUSH and SYNC.

A cache flush is initiated by asserting the 82495XP FLUSH# pin. Once initiated, the 82495XP writes all **M**odified lines out to main memory, performing back invalidations and inquire cycles on the CPU. When completed, all 82495XP/82490XP and CPU cache entries will be in the Invalid state.

Activation of the SYNC# pin also causes all of the 82495XP's **M**odified lines to be written to memory. Unlike the FLUSH# pin, the cache lines remain valid after the SYNCH# process has completed, with **M**odified lines changing to the **E**xclusive state.

4.3 The Effects of Special Cycles on MESI States

4.3.1 NON-CACHEABLE ACCESS

The 82495XP allows cacheability to be determined on both a per page and per line basis. The page cacheability function is determined by software, while cacheability on a line-by-line basis is driven by hardware.

The PCD (**P**age **C**aching **D**isabled) pin is a 82495XP input driven by the CPU's PCD output, which corresponds to a cacheability bit in the page table entry of a memory location's virtual address. If the PCD bit is asserted when the CPU presents a memory address, that location will not be cached in either the 82495XP or the CPU.

MKEN# is a 82495XP input which connects to the memory bus controller or the memory bus. MKEN# inactive prevents the caching of the memory location in both the 82495XP and the CPU, affecting only the current access.

If a read miss is indicated non-cacheable by either of these, the line is not placed in the 82495XP/82490XP or CPU cache, and no cache states are modified. On a write miss, a noncacheable indication from either input forces a write miss

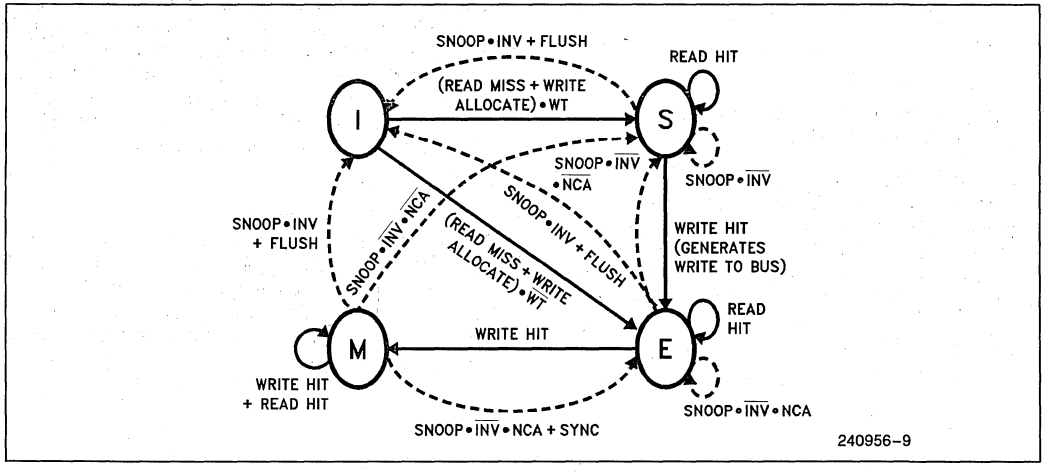


Figure 4-1. Major State Transitions

without allocation. Note that if the 82495XP/82490XP already has a valid copy of the line, the PCD attribute from the CPU is ignored.

4.3.2 READ ONLY ACCESSES: MRO#

The MRO# (Memory Read Only) input is driven by the memory bus to indicate that a memory location is read only.

When asserted during a read miss line fill, MRO# causes the line to be placed in the 82495XP/82490XP cache in the Shared state and also sets a read-only bit in the cache tag. MRO# accesses are not cached in the CPU. On subsequent write hits to a read-only line, the write is actually written through to memory without updating the 82495XP/82490XP line, which remains in the Shared state with the read-only bit set.

4.3.3 LOCKED ACCESSES: LOCK#

The LOCK# signal driven by the CPU indicates that the requested cycle should lock the memory location for an atomic memory access. Because locked cycles are used for interprocessor and intertask synchronization, all locked cycles will appear on the memory bus.

On a locked write, the 82495XP treats the access as a write-through cycle, sending the data to the memory bus—updating memory and invalidating other cached copies. If the data is also present in the 82495XP/82490XP cache, it is updated but its M, E, or S state remains unchanged.

For locked reads, the 82495XP assumes a cache miss and starts a memory read cycle. If the data resides in the 82495XP/82490XP, the M-E-S state of the data remains unchanged. If the requested data is in the 82495XP/82490XP and is in the Modified state when the memory bus returns data, the 82495XP will use the 82490XP data and ignore the memory bus data.

LOCKed read and write cycles which miss the 82495XP/82490XP cache are noncacheable in both the 82495XP/82490XP and CPU.

4.3.4 FORCING LINES DIRECT-TO-MODIFIED: DRCTM#

The DRCTM# (Direct To Modified) pin is an input which informs the 82495XP to skip the Exclusive state and place a line directly in the Modified state. The signal can be asserted during 82495XP/82490XP reads of the memory for special 82495XP/82490XP data accesses like read-for-ownership and cache-to-cache-transfer. The signal can also be asserted during writes, for purposes of cache tracking.

4.4 State Tables

Lines cached by the 82495XP can change states as a result of either the CPU bus activity (that sometimes require the 82495XP to become a memory bus master) or as a result of memory bus activity generated by other system masters (snooping).

State transitions are affected by the type of CPU/memory bus transactions (reads, writes) and by a set of external input signals and internally generated variables. In addition, the 82495XP will drive certain CPU/memory bus signals as a result of the consistency protocol.

4.4.1 CPU BUS

- *PWT* (Page Write Through, **PWT** Input pin) Indicates a CPU bus write-through request. Activated by the i860 XP CPU **PWT** pin. This signal affects line fills and will cause a line to be put in the [S] state if active. The 82495XP will NOT execute ALLOCATIONS (line fills triggered by a write) for write-through lines. If **PWT** is asserted, it overrides a write-back indication on the **MWB/WT#** pin.
- *PCD* (Page Cacheability Disable, **PCD** input pin): indicates that the accessed line is noncacheable. If **PCD** is asserted, it overrides a cacheable indication from an asserted **MKEN#**.
- *NWT* (i860 XP CPU Write-Through Indication, 82495XP's **WB/WT#** Output Pin): When low forces the i860 XP CPU to keep the accessed line into the SHARED state.

Write back mode (WB = 1) will be indicated by the */NWT* notation. In those cases the i860 XP CPU is allowed to go into exclusive states [E], [M]. *NWT* is normally active unless explicitly stated.

— *KEN* (CPU caching enable, **KEN#** output pin): When active indicates that the requested line can be cached by the CPU 1st level cache. *KEN* is normally active unless explicitly stated.

4.4.2 MEMORY BUS

— *MWT* (Memory Bus Write-Through Indication, **MWB/WT#** Input Pin): When active forces the 82495XP to keep the accessed line into the SHARED state. Write back mode (MWB = 1) will be indicated by the */MWT* notation. In those cases the 82495XP is allowed to go into exclusive states [E], [M].

— *DRCTM* (Memory Bus Direct To [M] indication, DRCTM# Input Pin): When active forces skipping of the [E] state and direct transfer to [M].

— *MKEN* (Memory Bus Cacheability Enable, **MKEN#** Input pin): When Active Indicates that the memory bus cycle is cacheable.

— *MRO* (Memory Bus Read-Only Indication, **MRO#** Input Pin): When Active forces line to be READ-ONLY.

— *MTHIT* (Tag Hit, MTHIT# Output pin): Activated by the 82495XP during snoop cycles and indicates that the current snooped address hits the 82495XP cache.

— *MHITM* (Hit to a line in the [M] State, **MHITM#** Output pin): Activated by the 82495XP during snoop cycles and indicates that the current snooped address hits a modified line in the 82495XP cache.

— *SNPNCA* (Non Caching device access): When active indicates to the 82495XP that the current bus master is a non-caching device.

— *SNPINV* (Invalidation): When active indicates to the 82495XP that the current snoop cycle will invalidate that address.

4.4.3 TAG STATE

— *TRO* (Tag Read Only, 82495XP Tag bit): This bit when set indicates that the 1 or 2 lines associated with this tag are Read-Only lines.

As a function of State Changes the 82495XP may execute the following cycles:

— *BINV:* Execution of a CPU Back Invalidation Cycle (Snoop with **INV** active)

— *INQR:* Execution of a i860 XP CPU Inquire Cycle(1).

— *WBCK:* 82495XP Write-Back Cycle. This is a Memory Bus write cycle generated by the 82495XP when MODIFIED data cached in the 82495XP needs to be copied back into main memory. A write-back cycle affects a complete 82495XP line.

— *WTHR:* 82495XP Write Through Cycle. This is a system write cycle in response to a processor write. It may or may not affect the cache SRAM (update). In a write-through cycle, the 82495XP drives the Memory Bus with the same Address, Data and Control signals as the CPU does on the CPU Bus. Main Memory is updated, and other Caches invalidate their copies.

— *RTHR:* 82495XP Read Through cycle. This is a special cycle to support locked reads to lines that hit the 82495XP cache. The 82495XP will request a Memory Bus cycle for lock synchronization reasons, data will be supplied from the BUS except for [M] state which will have data supplied from the CACHE.

— *LFIL:* 82495XP Cache line fill. 82495XP will generate Memory Bus cycles to fetch a new line and deposit into the cache.

— *RNRM:* 82495XP Read Normal Cycle: This is a normal read cycle which will be executed by the 82495XP for non-cacheable accesses.

— *SRUP:* 82495XP SRAM UPDATE. Occurs any time new information is placed in the 82495XP cache. An SRAM update is implied in the *LFIL* cycle.

— *ALLOC:* 82495XP ALLOCATION. Write Miss cycle that has determined to be cacheable so the 82495XP issues a line read.

NOTE:

1. An inquire cycle may be executed with INV active, performing a back-invalidation simultaneously.

STATE TABLES

Table 4-1. Master 82495XP Read Cycle

Pres. State	Condition: Next State	Mem Bus Activity	CPU Bus Activity	Comments
M	!LOCK: M	—	!NWT	Normal Read Hit [M]
	LOCK: M	RTHR	!KEN	Read Through Cycle, Data From Array
E	!LOCK: E	—	NWT	Normal Read Hit [E]
	LOCK: E	RTHR	!KEN	Read Through Cycle, Data From Memory
S	!LOCK.!TRO: S	—	NWT	Normal Read Hit [S]
	!LOCK.TRO: S	—	!KEN	Normal Read to Read-Only sector. Stays in [S] state and deactivate KEN to prevent CPU from caching line
	LOCK: S	RTHR	!KEN	Read Through Cycle, Data from Memory
I	PCD + !MKEN + LOCK: I	RNRM	!KEN	Non-Cacheable Read, Locked cycles
	!PCD.MKEN.!LOCK.MRO: S	LFIL	!KEN	Cacheable read, Read-Only. Fill line to 82495XP. Do not allow CPU to cache line by deactivating KEN#. Set the 82495XP's TRO bit to indicate the sector read only attribute
	!PCD.MKEN.!LOCK.!MRO.(PWT + MWT):S	LFIL	NWT	Cacheable Reads, forced Write-Through
	!PCD.MKEN.!LOCK.!MRO.!PWT.!MWT.!DRCTM: E	LFIL	NWT	Line not shared, thus enabling the 82495XP to move into tan exclusive state
	!PCD.MKEN.!LOCK.!MRO.!PWT.!MWT.DRCTM: M	LFIL	NWT	As before with direct [M] state transfer. Keep i860 XP CPU in Write Through mode

Table 4-2. Master 82495XP Write Cycle

Pres. State	Condition: Next State	Mem Bus Activity	CPU Bus Activity	Comments
M	!LOCK: M	-	SRUP, !NWT	Write hit. Write to cache. Allow i860 XP CPU to perform internal write cycles (Enter intor [E], [M] states).
	LOCK: M	WTHR	SRUP, !NWT	Locked Cycle. Write-Through updating cache SRAM. Most updated copy of the line is still owned by 82495XP. All Locked write cycles are posted.
E	!LOCK: M	-	SRUP, !NWT	Write hit. Update SRAM. Let i860 XP CPU execute internal write cycles.
	LOCK: E	WTHR	SRUP, NWT	Lock forces cycle to memory bus. Main memory remains updated.
S	TRO: S	WTHR	-	Read-Only. Write cycle with write through attribute from CPU or Memory Bus. Locked Cycles.
	!TRO.(PWT + MWT + LOCK): S	WTHR	SRUP, NWT	Not Read-Only. Write cycle with write through attribute from CPU or Memory Bus. Locked Cycles.
	!TRO.!PWT.!LOCK.!MWT.!DRCTM: E	WTHR	SRUP, NWT	Not Read-Only. No write-through cycle, no lock request allow going into exclusive state.
	!TRO.!PWT.!LOCK.!MWT.DRCTM: M	WTHR	SRUP, NWT	Not Read-Only. No write-through cycle, no lock request allow going into exclusive state. DRCTM forces final state to M.
I	PCD + !MKEN + PWT + LOCK + MRO: I	WTHR	-	Write Miss Non-Cacheable, Write-Through, locked cycle or Read-Only.
	!PCD.MKEN.!PWT.!LOCK.!MRO: I !PCD.MKEN.!PWT.!LOCK.MRO:S Allocation Final State MWT:S !MWT.!DRCTM:E !MWT.DRCTM:M	WTHR, LFIL ALLOC	-	Write Mis with allocation. After the write cycle, a line fill (allocation) is scheduled. If MKEN and MRO are asserted, an allocation to the [S] state will occur Allocation final state as a function of line fill attributes.

NOTE:
The **WB/WT#** pin will only be activated for 82495XP lines that are in the [M] state. In this state, the 82495XP always assumes that the line owner MAY be the i860 XP CPU. On all other states the i860 XP CPU will be forced to perform Write-Through cycles. This mechanism will make sure that any i860 XP CPU write cycle is seen at least once on the CPU Bus. Allocations, which are consequences of write-misses, will disregard the MKEN# and MRO# attributes during the line fill. In other words, once an allocation is scheduled, it cannot be cancelled.

Table 4-3. Snooping 82495XP without Invalidation Request

Pres. State	Condition: Next State	Mem Bus Activity	CPU Bus Activity	Comments
M	!SNPNCA: S SNPNCA: E	MTHIT MHITM WBCK	INQR	Snoop hit to modified line. 82495XP indicates tag hit and modified hit. 82495XP schedules flushing of the modified line to memory. If non-cacheable device, stay in [E] state.
E	!SNPNCA: S SNPNCA: E	MTHIT	-	If snooping by cacheable device, indicate MTHIT and go to shared state. If no caching device only indicate MTHIT, stay exclusive.
S	S	MTHIT	-	
I	I	-	-	

NOTE:
Usage of DRCTM# to avoid [E] states may be in conflict with the SNPCNA cycle attribute. Note in the table that snoops with SNPNCA may cause an [E] state transition.

Table 4-4. Snooping 82495XP with Invalidation Request

Pres. State	Next State	Mem Bus Activity	CPU Bus Activity	Comments
M	I	MTHIT MHITM WBCK	INQR, BINV	Snoop hit to modified line. 82495XP indicates tag hit and modified hit. 82495XP schedules flushing of the modified line to memory. Invalidate CPU.
E	I	MTHIT	BINV	Inidicate tag hit, infalidate 82495XP, CPU lines.
S	I	MTHIT	BINV	Same as before
I	I	-	-	

Table 4-5. SYNC Cycles

Pres. State	Next State	Mem Bus Activity	CPU Bus Activity	Comments
M	E E	WBCK WBCK	INQR -	Get modified data from i860 XP CPU, flush to memory
E	E	-	-	Memory already synchronized
S	S	-	-	Memory already synchronized
I	I	-	-	

Table 4-6. FLUSH Cycles

Pres. State	Next State	Mem Bus Activity	CPU Bus Activity	Comments
M	I	WBCK	INQR, BINV	Flush and invalidate i860™ XP CPU
E	I	—	BINV	Invalidate i860 XP CPU
S	I	—	BINV	Invalidate i860 XP CPU
I	I	—	—	

NOTE:
Usage of DRCTM# to avoid [E] states may be in conflict with the SYNC cycle. Note in the table that SYNC cycles move an [M] state line to [E].

5.0 CONFIGURATIONS

The 82495XP/82490XP cache system was designed to fit a variety of applications. For the greatest performance, each application requires the 82495XP/82490XP to be configured differently. The 82495XP/82490XP therefore has many possible configurations that are set on RESET and affect the 82495XP/82490XP architecture, operation, and electrical characteristics.

5.1 Physical Cache

The physical configurations of the 82495XP/82490XP consist of parameters that alter the 82495XP/82490XP basic architecture. These are line ratio, tag size, lines per sector, bus width, and cache size. These parameters are sampled at the falling edge of RESET and are not dynamically changeable.

Because of physical cache constraints, choosing one parameter limits the flexibility of other parameters. The following table summarizes the possible i860 XP CPU basic cache configurations. CFG0–CFG2 are multiplexed to select one of 5 possible line ratio/tag size/lines per sector configurations. This information is automatically passed from the 82495XP to 82490XP during RESET. CFG0–CFG3 must be valid at least 10 clocks before RESET's falling edge.

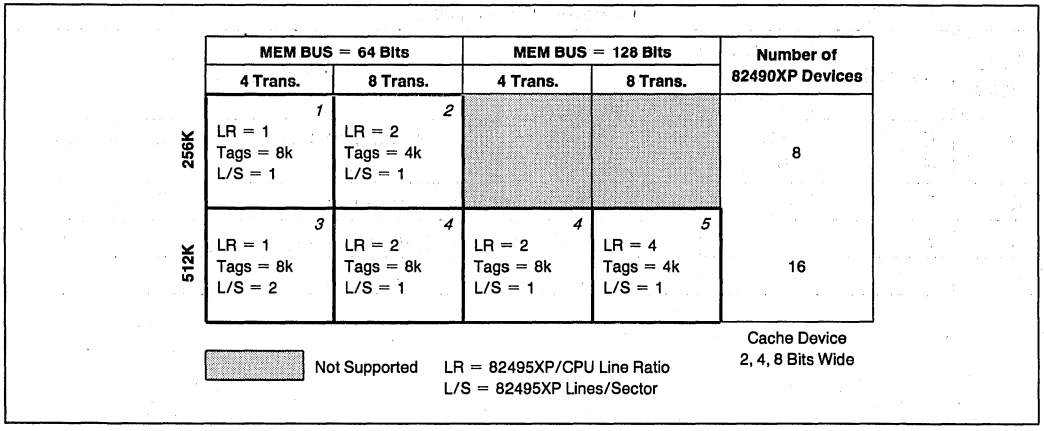


Figure 5-1. 82495XP/82490XP Configurations

Table 5-1. CFG Configuration Inputs

Cfig No.	Line Ratio	Lines/ sec	No. of Tags	CFG2	CFG1	CFG0
1	1	1	8K	0	0	1
2	2	1	4K	1	1	1
3	1	2	8K	0	0	0
4	2	1	8K	0	1	1
5	4	1	4K	1	1	0

5.1.1 LINE RATIO (LR)

Line Ratio (LR) is the ratio of the 82495XP/82490XP cache line size to the CPU cache line size. For example, if LR=2 then the 82495XP/82490XP line size is 64 bytes. This information is also used to determine the number of back invalidations or inquire cycles to the i860 XP CPU.

5.1.2 TAG SIZE (TAGS)

The 82495XP/82490XP cache tag size may be 4K or 8K tag entries. By reducing tag size, the line ratio (LR) can be doubled without a change in cache size.

5.1.3 LINES PER SECTOR (L/S)

The 82495XP/82490XP may be non-sectored (L/S = 1) or contain two lines per sector (L/S = 2). If L/S = 2, then the 82495XP contains one tag for two consecutive cache lines and each cache line has its own set of MESI state bits. This allows just one line to be filled on replacements or written back on snoop hits. Both lines are written back during replacements, if both are modified.

5.1.4 BUS SIZE

The 82495XP/82490XP supports 64 and 128 bit memory bus widths for the i860 XP CPU.

5.1.5 CACHE SIZE

The 82495XP/82490XP may be configured to be 256K or 512K. Cache size is a direct result of the number of 82490XP devices used. It takes 8 82490XP's to make a 256K byte cache and 16 82490XP's for a 512K cache.

5.1.6 FUNCTION AND ADDRESS CONNECTIONS (CFA0–CFA6)

The following table lists which address lines should be connected to each of the CFA0–CFA6 lines for each cache configuration. CFA0–CFA6 provide the 82495XP with proper multiplexed addresses for each of the possible cache configurations. Depending on the mode selected, either CFA5 or CFA4 will operate as the 82495XP's CTYP input. This input is connected to the i860 XP CPU's CTYP output.

Table 5-2. CFA Address Connections

Cfig No.	Line Ratio	Lines/ sec	No. of Tags	CFA6	CFA5	CFA4	CFA3	CFA2	CFA1	CFA0
1	1	1	8K	A5	CTYP	A31	A30	A29	A4	A3
2	2	1	4K	A5	CTYP	A31	A30	A29	A4	A3
3	1	2	8K	A6	A5	CTYP	A31	A30	A4	A3
4	2	1	8K	A6	A45	CTYP	A31	A30	A4	A3
5	4	1	4K	A6	A5	CTYP	A31	A30	A4	A3

5.2 Cache Modes

Cache modes are ways of configuring the 82495XP/82490XP to operate differently. These options are all sampled at RESET and are not dynamically changeable. If some of these configuration options share a pin, such as the 82495XP's SYNC# and MEMLDRV, the configuration option must meet a specific setup and hold time to RESET's falling edge. For the 82495XP, setup time is usually 4 clocks, and for the 82490XP, setup time is usually 1 clock. For both parts, the configuration option must be held until RESET is detected low.

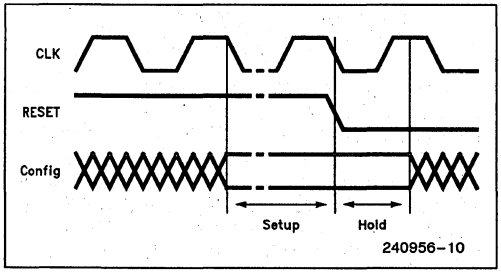


Figure 5-2. Configuration Input Sampling

5.2.1 MEMORY BUS MODES

The 82495XP/82490XP may be configured to have a clocked or strobed memory bus. Memory bus mode is selected by the 82490XP MSTBM pin (same as MCLK pin). If MSTBM is strapped high, the 82490XP's operate in strobed mode. If MSTBM is toggling, ie it is connected to the memory bus clock, the 82490XP operates in clocked mode. MCLK need not be synchronous to CLK.

5.2.2 SNOOPING MODES

The 82495XP/82490XP supports three snooping modes: synchronous, clocked, and strobed. Snooping mode is selected by the SNPMD (same as SNPCLK) pin. If SNPMD is low the 82495XP snoops synchronously. If SNPMD is high the 82495XP snoops in strobed mode. If SNPMD is toggling, clocked mode is selected and SNPMD becomes a snoop clock source, SNPCLK, which clocks in the snoop requests.

These three snooping modes only alter the way the memory bus controller may initiate a snoop request to the 82495XP. The 82495XP response is always synchronous to the CPU CLK.

5.2.3 BUS DRIVERS

The 82495XP/82490XP provide 2 types of memory bus drivers: High capacitance drivers and low capacitance drivers. The high capacitance drivers are selected by driving both the 82495XP and 82490XP MEMLDRV pins low at RESET. Similarly, the low capacitance drivers are selected with MEMLDRV high.

With C490LDRV the 82495XP also provides two types of drivers when driving the 82490XP's. Refer to the interface document to determine C490LDRV.

5.2.4 STRONG/WEAK WRITE ORDERING

If the 82495XP pin WWOR# is sampled low at RESET, the 82495XP enforces weak write-ordering. If sampled high, the 82495XP enforces strong write-ordering. Strong write-ordering prevents the 82495XP from completing a write cycle that would go to 'M' state if a posted write is pending (has not been granted the bus with BGT#). By doing this, strong ordering ensures that write cycles from the CPU are written to memory in the same order that they appear in the i860 XP CPU program.

5.2.5 i860™ XP CPU PFLD SUPPORT

The i860 XP microprocessor executes PFLD (Pipelined Floating-Point Load) instructions to implement special data handling, typically for vector operations. This instruction allows loading of data through a FIFO pipeline, to hide memory latency. The i860 XP CPU does not cache data returned by a PFLD cycle.

The 82495XP can be configured to decode the i860 XP microprocessor's PFLD cycles. The 82495XP supports 3 operational modes for PFLD cycle decoding:

Mode #1. PFLD cycles are cached in the 82495XP.

This mode is used in applications that can fit entirely in the 82495XP/82490XP cache. The 82495XP treats PFLD cycles as normal read cycles.

Mode #2. PFLD cycles are not cached in the 82495XP, without an external PFLD extension FIFO.

This mode is used when applications are too large to fit in the 82495XP/82490XP cache. The 82495XP treats PFLD cycles as noncacheable, using the same protocol as cycles with PCD=1 (if data is already cached, it will be supplied from the cache).

Mode #3. PFLD cycles not cached in the 82495XP, with an external PFLD extension FIFO.

This mode allows the PFLD FIFO to be extended beyond the three stages built into the i860 XP CPU by adding external FIFO hardware. The 82495XP, treats PFLD cycles in the same manner as its treatment of LOCKed cycles (all cycles go to the bus, even if data already present in cache). To support the external FIFO, the 82495XP identifies PFLD cycles by asserting its FPFLD output. For proper operation, data which can be accessed by PFLD must never be in the cache in the Modified state, and software must be aware of the length of the combined PFLD pipeline. Because this mode is not software transparent, it must be used with extreme care.

The choice of PFLD mode is largely application dependent. The PFLD mode of the 82495XP is selected by configuration pins FPFLDEN and NCPFLD#, which are sampled at RESET. FPFLDEN shares a pin with FPFLD, and NCPFLD# shares a pin with FLUSH#. Depending on the PFLD mode, data for reads will either be supplied to the CPU from the 82495XP, or from the memory bus. Table 5-3 summarizes, the 82495XP's support for i860 XP CPU PFLD cycles.

Table 5-3. 82495XP PFLD Modes

Mode#	FPFLDEN	NCPFLD#	Data Supplied From				Line Fill on [I]
			[I]	[S]	[E]	[M]	
1	0	1	MEMBUS	CACHE	CACHE	CACHE	Yes
2	0	0	MEMBUS	CACHE	CACHE	CACHE	No
3	1	1	MEMBUS	MEMBUS	MEMBUS	MEMBUS	No
X	1	0	Illegal Mode				

5.3 82490XP Bus Configuration

The 82490XP needs to be configured so it knows to drive 4 or 8 MDATA lines and whether it should do 4 or 8 memory transfers per line fill. This is done through the MX4/MX8# and the MTR4/MTR8# configuration inputs. For a given line ratio (memory bus line size / CPU line size), they should be sampled as follows:

Table 5-4. MX/MTR Configurations

Line Ratio	MX4/ MX8#	MTR4/ MTR8#	Membus I/O	CPUbus I/O
1	1	1	4	4
2	1	0	4	4
2	0	1	8	4
4	0	0	8	4
1	0	1	8	8
2	0	0	8	8

5.3.1 82490XP PARITY CONFIGURATION

A 82490XP may be designated as a parity device. This is done by strapping the PAR# pin low. In this configuration CDATA[0:3] are used to store 4 parity bits, and CDATA[4:7] are used as 4 bit enables. The four bit enables allow the writing of individual parity bits.

Every mode and configuration of a non-parity 82490XP may be used and selected on the parity 82490XP device. The 82490XP parity configurations are as follows:

Table 5-5. Parity Configurations

Cache Size	Memory Bus Width	Number of Parity Devices	82490XP I/O bits (CPU/Mem)
256K	64	2	4/4
512K	128	2	4/8

5.3.2 CPU 82490XP ADDRESS CONFIGURATIONS

The 82490XP Address inputs (A) are multiplexed to the CPU address lines (CA) according to the cache size:

Table 5-6. 82490XP Address Connections

Size	82490XP Address Pins															
	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
256K	CA 16	CA 15	CA 14	CA 13	CA 12	CA 11	CA 10	CA 9	CA 8	CA 7	CA 6	CA 5	CA 4	CA 3	CA	Vss
512K	CA 17	CA 16	CA 15	CA 14	CA 13	CA 12	CA 11	CA 10	CA 9	CA 8	CA 7	CA 6	CA 5	CA 4	CA 3	CA 2

NC = No Connect.

6.0 CACHE OPERATION

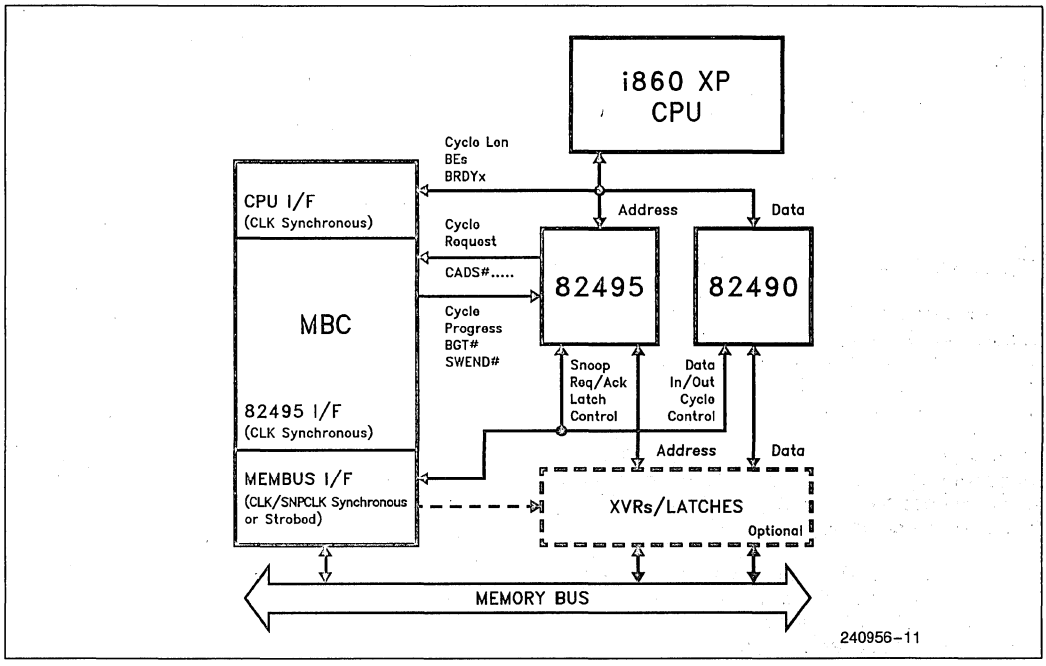


Figure 6-1. Memory Bus Controller Interface Model

Figure 6-1 shows the memory bus controller (MBC) interface model. The memory bus controller interfaces to the i860 XP CPU, 82495XP, 82490XP, and memory bus. The MBC interface was defined with a minimal set of assumptions as to the memory bus implementation. The chipset was designed to enable flexibility in the design of a memory bus and controller.

The 82495XP requests control of the memory bus by signalling the memory bus controller. The memory bus controller is responsible for arbitrating and granting the bus to the 82495XP. Once granted, the memory bus controller is responsible for executing the requested cycle, snooping the other caches, and ending the cycle. The 82495XP supports different modes of snooping, different modes of memory bus operation, and various special cycles. Memory Bus Controller design dictates which of these features are used, and exactly how they are used.

6.1 Cycle Attribute and Progress

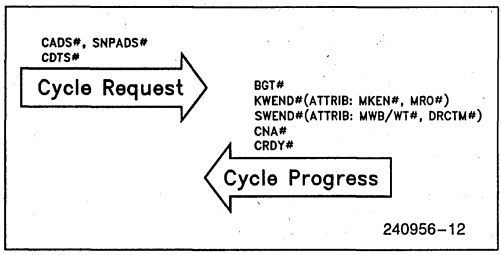


Figure 6-2. Cycle Attribute and Progress Signals

CADS# indicates the start of the cycle address phase. CDTS# tracks CADS# and indicates the start of the cycle data phase. For READ cycles it indicates that starting in the next CLK the CPU data bus is in read mode under the control of the MBC until the last BRDY#. In Read cycles, if the MBC already owns the CPU data bus, CDTS# will be activated with CADS#. For ALLOCATE cycles the MBC does not need the CPU data bus, therefore CDTS# is activated together with CADS#.

For Write cycles CDTS# indicates that the 1st piece of data is available on the memory bus. For write-back cycles CDTS# indicates that all data is available (write-back buffer or snoop buffer loaded with correct write-back data).

As a response to the cycle request, the memory bus controller responds with cycle progress signals. All cycle progress signals are sampled ONCE in specific windows and then ignored until CRDY# of the corresponding cycle. BGT# indicates a commitment by the memory bus controller to complete the cycle execution on the memory bus. Up until this point the 82495XP owns the cycle. This means that intervening snoop-write-backs will abort it and the 82495XP re-issues the cycle to the MBC. There is only one case where the 82495XP will issue a new, not a re-issued, cycle; if the original CADS# operation is a write-back cycle, and the interrupting snoop cycle hits that write-back buffer, then the subsequent CADS# will be for a completely new cycle (not a re-issuing of the interrupted CADS# operation).

After BGT# the memory bus controller owns the cycle. The 82495XP assumes the cycle will terminate and will not re-issue it on snoop-write-backs. Following BGT# comes KWEND# which indicates that the cacheability window is closed and that the 82495XP can sample MKEN#, MRO# attributes. Those indicate to the 82495XP cacheability and read-only respectively. These attributes can be determined by decoding the 82495XP address. Based on those attributes the 82495XP executes ALLOCATIONS, Line-fills, Replacements, etc.

Following KWEND#, SWEND# is activated. It indicates that the Snoop Window is closed. The 82495XP samples MWB/WT# and DRCTM# attributes. These attributes are determined by snooping the other caches in the system. At this point the 82495XP updates its TAGRAM state related to the line access in progress.

Lastly the MBC issues CRDY#, which indicates to the 82495XP the end of the transaction data phase.

The 82495XP allows memory bus pipelining by providing CNA# which allows the MBC to request a new address phase before the conclusion of the current data phase. The 82495XP supports a 1 level deep address pipeline on the Memory Bus.

6.2 Snoop Operations

The 82495XP provides the capability of snooping operations on the memory bus to ensure cache consistency. A snoop operation consists of two phases: 1) initiation phase and 2) response phase.

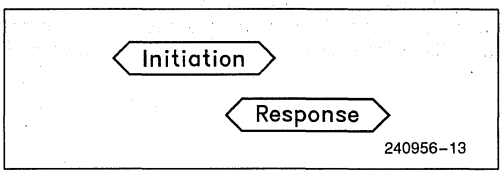


Figure 6-3. 82495XP Snooping Operations

During the initiation phase the MBC provides the 82495XP with the snoop address information. During the response phase the 82495XP provides the snoop status information.

6.2.1 SNOOP INITIATION PHASE

The 82495XP provides three modes for initiating snoops:

1. Strobed: the falling edge of SNPSTB# is used.
2. Clocked: SNPSTB# is sampled with SNPCLK.
3. Synchronous: SNPSTB# is sampled with CLK.

These three snooping modes are configured as follows:

1. Strobed: The SNPCLK[SNPMD] signal must be strapped high.
2. Clocked: The SNPCLK[SNPMD] signal must be connected to the snoop clock source.
3. Synchronous: The SNPCLK [SNPMD] signal must be strapped low.

NOTE:
The 82495XP samples the SNPCLK[SNPMD] signal at the falling edge of RESET to determine the snoop mode. If a rising edge occurs on the SNPCLK[SNPMD] after RESET has gone inactive, clocked mode will be selected. Systems using stobed or synchronous mode must ensure that no rising edge occur on SNPCLK[SNPMD] after RESET has gone inactive.

Figure 6-4 shows the strobed method of snoop initiation. The memory address, SNPNCA, SNPINV, and MBAOE# are latched with the falling edge of the SNPSTB#. If MAOE# is sampled active (low), the SNPSTB# will not cause a snoop. The snoop initiation is recognized by the 82495XP, is synchronized in the next clock, and causes a snoop in the following clock.

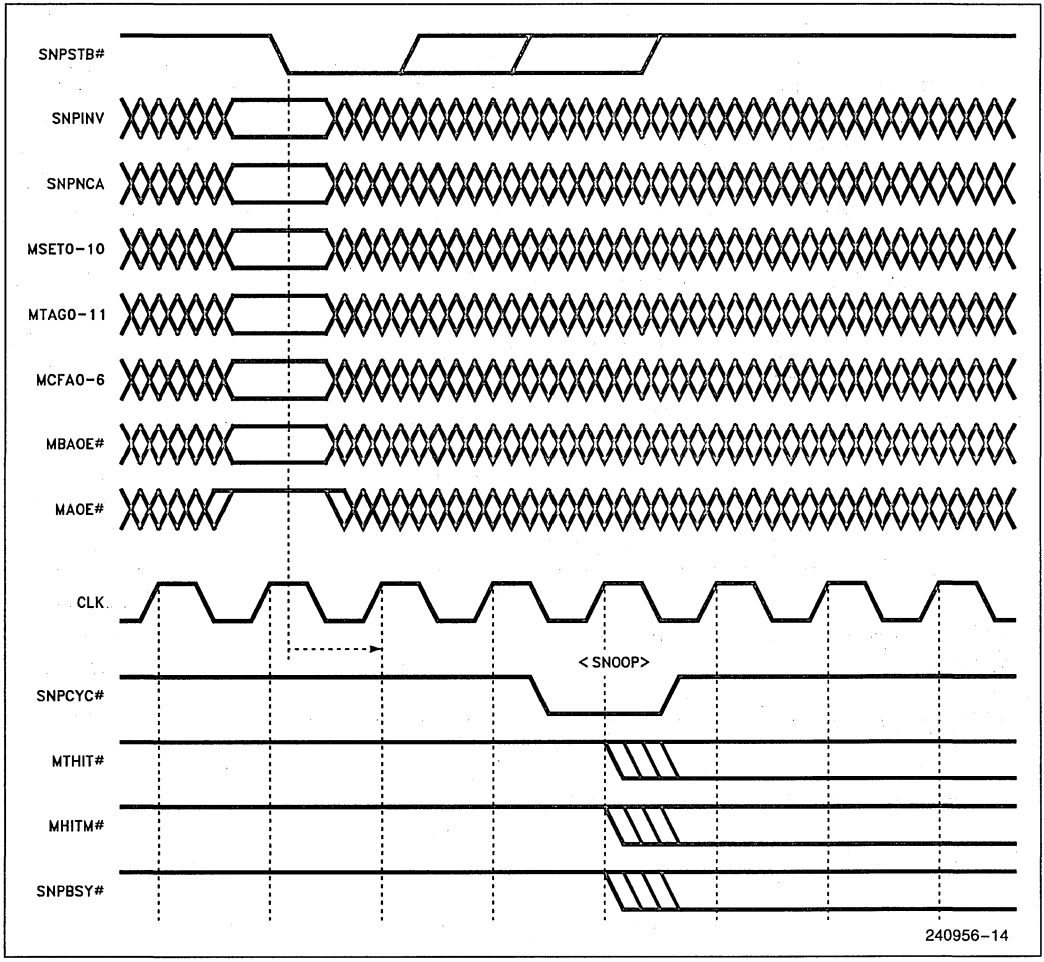


Figure 6-4. Strobed Snoop Mode

Figure 6-5 shows the clocked method of snoop initiation. The memory address, SNPNCA, SNPINV, and MBAOE# are latched with the rising edge of SNPCLK when SNPSTB# is first sampled low. SNPSTB# must be sampled high for at least one SNPCLK in order to rearm for another snoop. If MAOE# is sampled active (low), the SNPSTB# will not cause a snoop. The snoop initiation is recognized by the 82495XP, is synchronized in the next clock, and causes a snoop in the following clock.

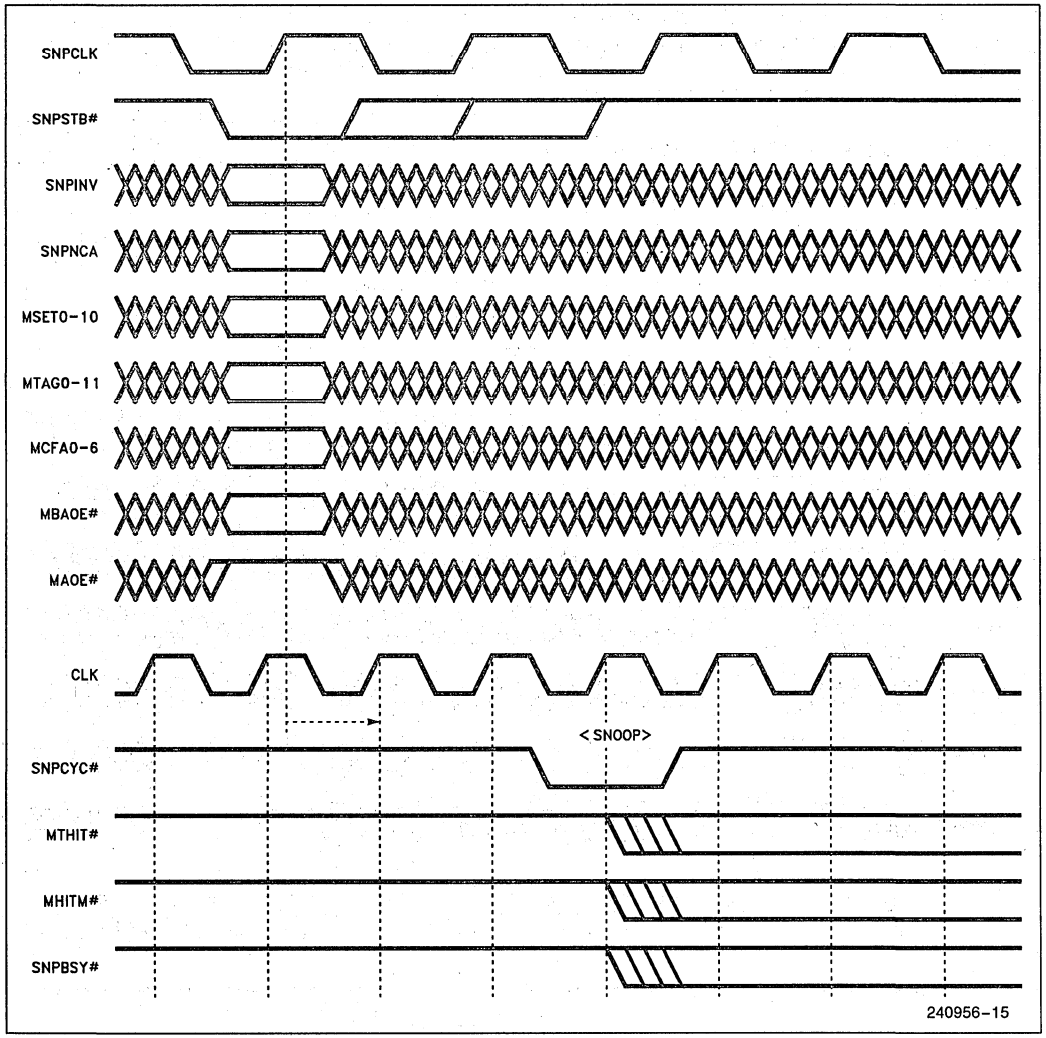


Figure 6-5. Clocked Snoop Mode

Figure 6-6 shows the synchronous snoop mode. The memory address, SNPNCA, SNPINV, and MBAOE# are latched with the rising edge of CLK when SNPSTB# is first sampled low. SNPSTB# must be sampled high for at least one CLK in order to rearm for another snoop. If MAOE# is sampled active (low), the SNPSTB# will not cause a snoop. The snoop initiation is recognized by the 82495XP, and causes a snoop in the next clock.

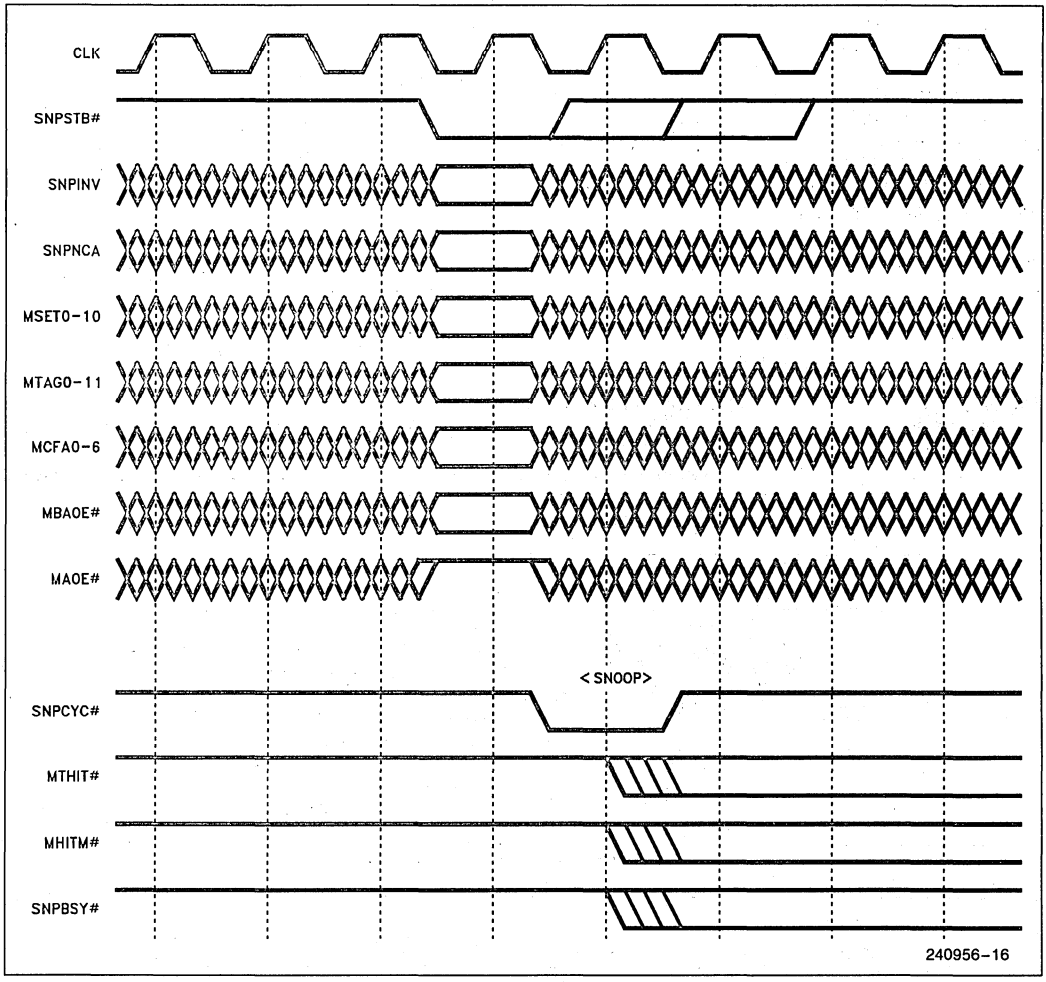


Figure 6-6. Synchronous Snoop Mode

6.2.2 RESPONSE PHASE

The snoop response phase consists of two parts: 1) 82495XP state indication 2) 82495XP snoop processing completion. The response phase is ALWAYS synchronous with the CPU CLK. The 82495XP state indication is presented on MHITM# and MTHIT# and remains stable until the next snoop. These signals indicate the state of the 82495XP line just prior to the snoop operation. The memory bus controller can predict the final state of the 82495XP line knowing the initial state and the SNPINV and SNPNCA inputs. The snoop completion information is determined by the SNPBSY# output. The SNPBSY# output inactive indicates that the 82495XP is ready to accept another snoop cycle.

Figure 6-7 shows the 82495XP response to snoops without invalidation. The first snoop is to a line which is not currently stored in the cache.

Figure 6-8 shows the 82495XP response to snoops with invalidation.

The SNPBSY# signal will be activated for one of two reasons: 1) a snoop hit to a modified line, SNPBSY# will remain active until the modified line has been written back. 2) a Back invalidation is needed and there is a back invalidation in process. The SNPBSY# minimum active time is two CLK periods. This allows an external logic to trap-hold active SNPBSY# using CLK. The external logic must first look for active SNPCYC# and then trap-hold SNPBSY#.

6.2.3 PIPELINED SNOOPS

The 82495XP allows the memory bus controller to pipeline snoop operations. The 82495XP allows the next snoop address to be supplied and the next snoop requested before the last snoop has completed.

There are a set of rules which govern the operation of pipelined snoops. These rules are as follows:

(1) For strobed mode snoops, the memory bus controller cannot cause a second falling edge of SNPSTB# until after the falling edge of SNPCYC#.

(2) For clocked mode snoops, the memory bus controller cannot cause a second falling edge of SNPSTB# to be sampled by SNPCLK, until after the falling edge of SNPCYC#.

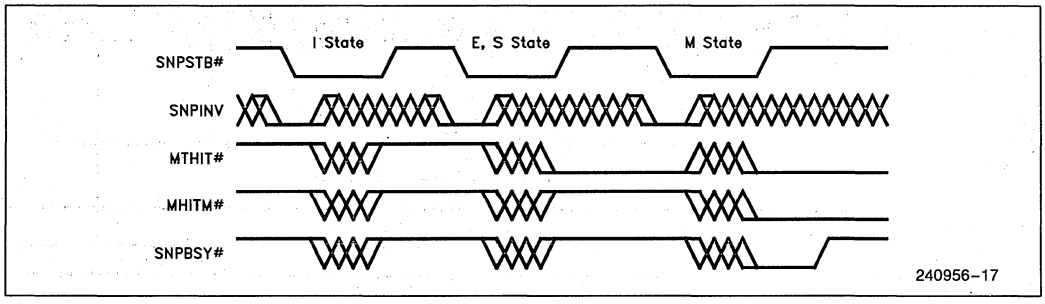


Figure 6-7. Snoops without Invalidation

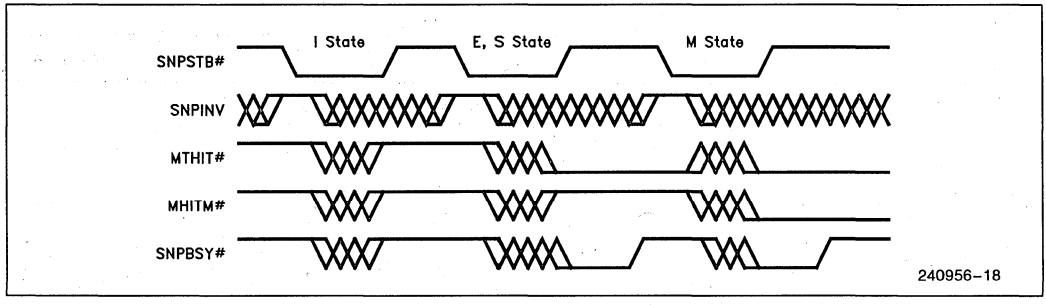


Figure 6-8. Snoops with Invalidation

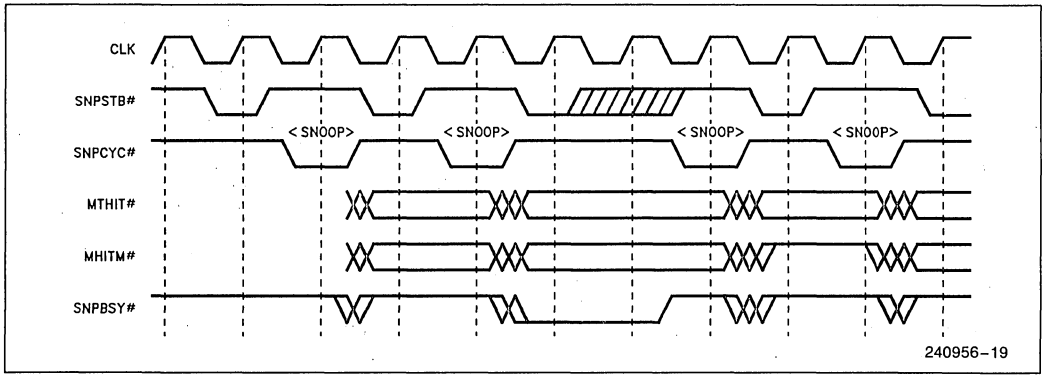


Figure 6-9. Fastest Possible Synchronous Snooping

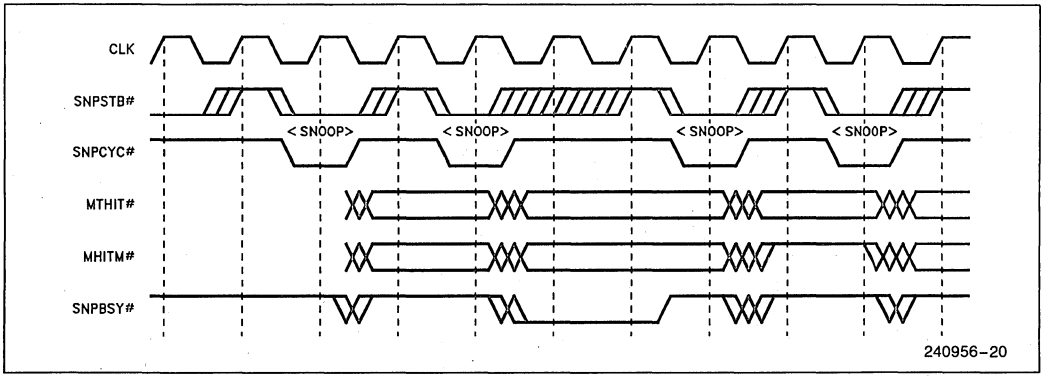


Figure 6-10. Fastest Possible Asynchronous Snooping

(3) For synchronous mode snoops, the memory bus controller cannot cause a second falling edge of SNPSTB# to be sampled by CLK, until the CLK after SNPCYC# is active.

6.2.4 OVERLAPPING SNOOPS WITH MEMORY BUS CYCLES

The 82495XP allows snoops to be overlapped with data transfers. The 82495XP divides the memory bus cycle into 4 main regions as shown below:

CRDY#	CADS#	BGT#	SWEND#	CRDY#	CADS#
1	2	3	4	1	

Region 1 is after a previous memory bus cycle (i.e. after CRDY#) and before the new memory bus cycle starts (before CADS#). A snoop in this region is looked up immediately and serviced immediately.

Region 2 is after a memory bus cycle has started (CADS#) but before the 82495XP has been granted the bus (BGT#). A snoop in this region is looked up immediately and serviced immediately. CADS# is re-issued for the aborted cycle once the snoop completes.

Region 3 is after the 82495XP has been granted the bus and before the SWEND# is completed. A snoop in this region has its lookup blocked until after the SWEND#. After SWEND#, the snoop response is given, but no write-back will be initiated until after CRDY#.

Region 4 is after SWEND# and before CRDY#. A snoop in this region is looked up immediately but serviced after CRDY#. This snoop is logically treated as if it occurred after CRDY# (snoop hits to modified data will schedule a write-back which will be executed after the conclusion of the current memory bus cycle). Note that the result of the snoop MHITM#, MTHIT# will be available immediately with the look-up.

6.2.5 SNOOP INTERLOCK

The 82495XP uses two interlock mechanisms to ensure that Snoops are identified within the proper region. The first interlock ensures that once a BGT# has been given snoops are blocked until after SWEND#. The second interlock ensures that once a snoop has been started BGT# cannot be given until after the snoop has been serviced.

Figure 6-11 shows how once the 82495XP sees a BGT# it blocks all snoops until after SWEND#. If a snoop has been initiated, and no SNPCYC# has been issued before BGT# assertion, the snoop has been blocked.

Figure 6-12 shows a snoop occurring before BGT#. Once the 82495XP has honored a snoop, the 82495XP, depending on the result of the snoop, may ignore BGT# until the snoop is serviced. The 82495XP will always ignore BGT# when SNPCYC# is active. If the snoop result is a hit to a modified line (MHITM# active), the 82495XP will ingore BGT# as long as both SNPBSY# and MHITM# remain active. In this case, it is the memory bus controller's responsibility to hold BGT# until SNPBSY# goes inactive or reassert it after SNPBSY# becomes inactive. If the snoop result is not a hit a modified line (MHITM# inactive), the 82495XP is capable of accepting BGT# even when SNPBSY# is active. This allows the memory bus controller to preceed with a memory bus cycle by asserting BGT# while the 82495XP is performing back-invalidations.

These two interlock mechanisms provide a flexible method of ensuring predictable handling of overlapped snoops.

NOTE:
Even when snoops are delayed, address latching is performed with SNPSTB# activation.

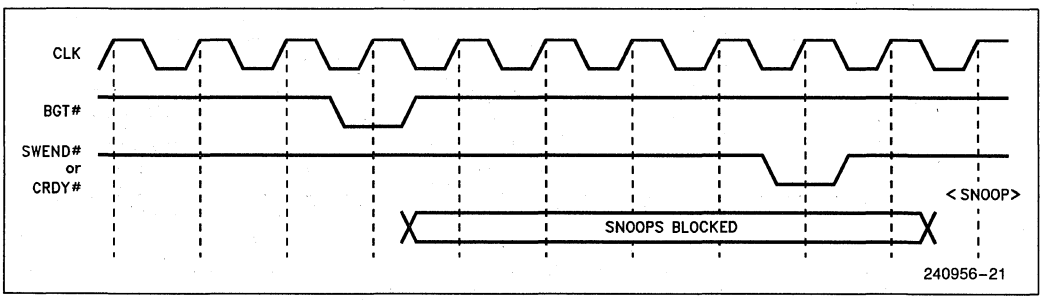


Figure 6-11. BGT# Blocking a Snoop

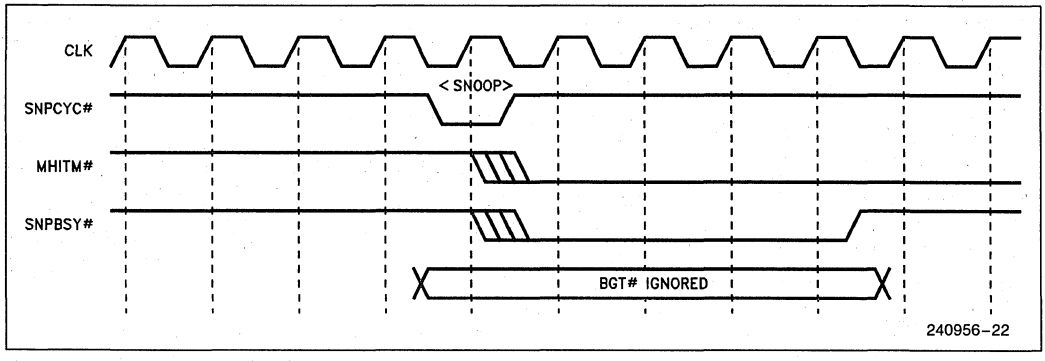


Figure 6-12. Snoop Occurring before BGT#

6.2.6 SNOOPS CONCURRENT WITH LINE FILL CYCLES

During snoops concurrent with line-fills/allocates, the following responsibility boundaries must be fullfilled in order to insure data consistency:

- If a snoop happens before BGT#, more precisely if SNPCYC# is active before BGT#, it is the system's responsibility not to return stale data within the line-fill/allocation.
- If a snoop happens after BGT#, more precisely if SNPCYC# is active after BGT#, then the 82495XP insures data consistency by providing interlocks with the CPU which avoid caching of stale data.

6.3 Memory Bus Controller Interface Rules

To begin a cache cycle, the 82495XP outputs the CADS# signal. The cache address and other cycle parameters are guaranteed to be stable with CADS# assertion. These parameters are guaranteed to be stable until CNA# or CRDY# of that cycle. After CNA# or CRDY# these parameters are undefined.

Either during, or after CADS# the CDTS# signal is asserted. Data is guaranteed to stable with CDTS# assertion, or the data path is available.

BGT# and CRDY# are required for all (non-snoop) cycles. KWEND# and SWEND# are only required for those cycles which sample them.

Once a signal has been sampled, it is a "don't care" until CRDY# of that cycle. Additionally, these signals plus the attributes MRO#, MKEN#, MWB/WT#, and DRCTM# need only follow setup and hold times when they are being sampled.

For pipelined cycles, the cycle attributes (BGT#, KWEND#, . . .) will only be sampled after CRDY# of the previous cycle.

Note that there are many other rules that govern when signals may be asserted in relation to one another. These may be found in the specific pin descriptions of each signal in chapter 7.

Snoop-Write-Back cycles are a subset of the normal cycles. Snoop-Write-Back cycles are requested as a consequence of snoop hits to Modified lines. Those are intervening cycles and are requested by activating SNPADS# instead of CADS#. For those cycles, the 82495XP only samples the CRDY# response. The 82495XP assumes that the memory bus controller owns the bus to perform the intervening write-back (restricted back-off protocol) and that no other agents will snoop this cycle. Also the 82495XP will ignore CNA# during Snoop-Write-Backs.

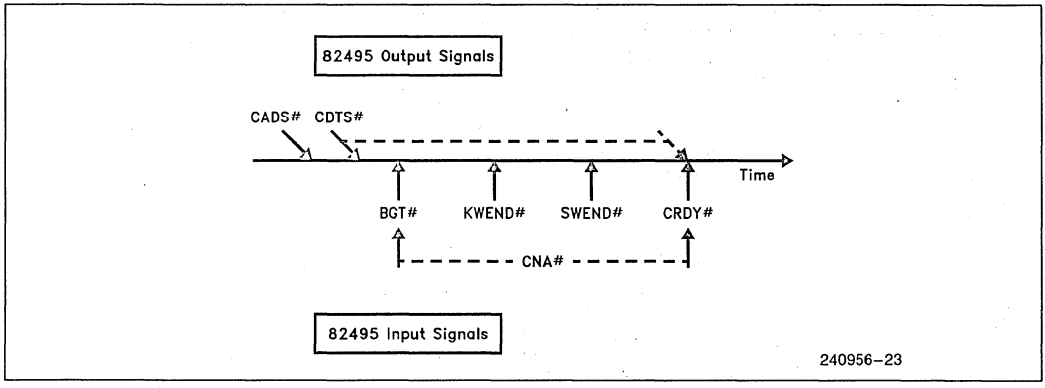


Figure 6-13. Cycle Progress

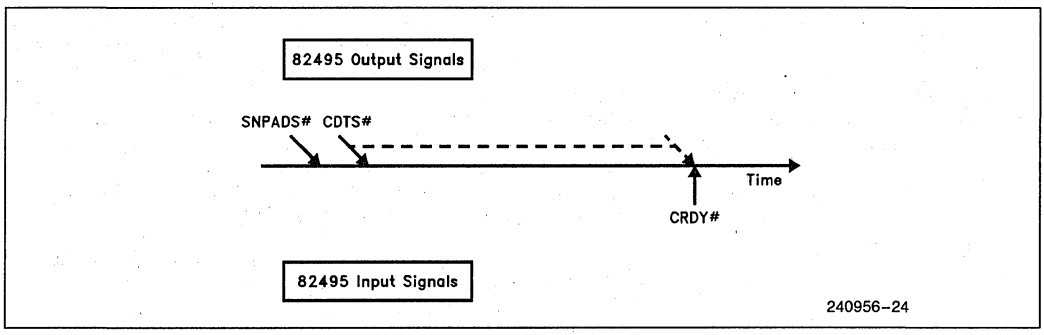


Figure 6-14. Cycle Progress for Snoop Cycles

6.4 LOCK# Protocol

The 82495XP provides a LOCK signal for the memory bus called KLOCK#. KLOCK# is generated by the 82495XP whenever the CPU generates the LOCK# signal. KLOCK#, like the other cycle attributes, is valid with CADS# assertion.

When the CPU generates a LOCK cycle, the 82495XP always generates a bus cycle. LOCK cycles are non-cacheable to both the 82495XP and CPU, so the information is passed through the 82490XPs to the CPU with BRDYs generated by the MBC. If the LOCKed read cycle is a hit in the 82495XP, the 82495XP ignores the data that it is receiving and supplies data from the 82490XP array (in accordance with the BRDYs supplied by the MBC). Locked writes are posted like any other write. LOCKed cycles, both reads and writes, never change the 82495XP tag state.

During a LOCKed cycle, the MBC must prevent other masters from snooping the 82495XP. Specifically, the MBC must prevent SNPSTB# between BGT# of the first LOCKed transfer, and SWEND# of the last LOCKed transfer.

6.5 Cycle Length

When CADS# is generated, the 82495XP outputs CW/R# and MCACHE#. These signals provide the MBC with enough information to determine the type of 82495XP cycle. Table 6-1 summarizes the cycle types for the 82495XP/82490XP. All line-fills and write-backs to the 82495XP/82490XP cache operate on the entire length of a cache line.

In addition to the length of the cycle from the 82495XP/82490XP, the memory bus controller may need to determine the length of the cycle to the CPU. Specifically, for those 82495XP cycles where RDYSRC=1, the MBC must decode the i860 XP CPU's W/R#, LEN, and CACHE# outputs to determine the number of BRDY#s which the MBC will provide to the CPU. These signals are captured for the current cycle by a user-provided BE latch (see Section 7.2 for details). Table 6-2 presents the CPU cycle length definitions; see the i860 XP microprocessor Data Sheet (Order #240874) for further details.

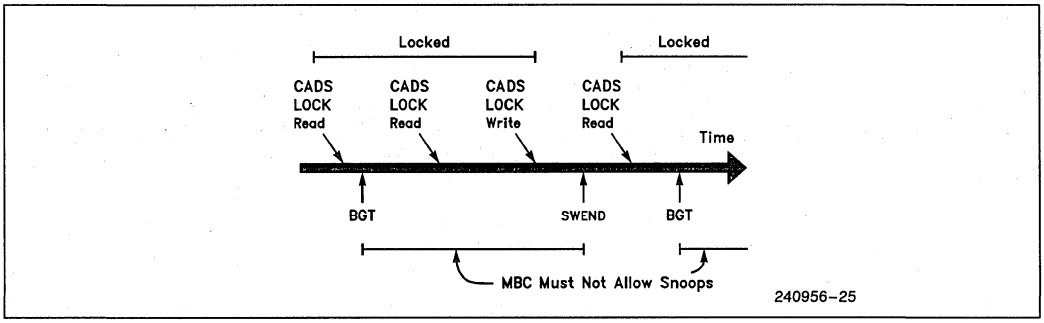


Figure 6-15. Snooping During LOCKed Cycles

Table 6-1. 82495XP/82490XP Cycle Determination

Cycle Type	CW/R#	RDYSRC	MCACHE#	MKEN#
Posted Write	1	0	1	X
Write Backs	1	0	0	X
Non-Cacheable Read	0	1	1	X
Non-Cacheable Read	0	1	0	1
Cacheable Read	0	1	0	0
Allocation	0	0	0	X

Table 6-2. i860 XP CPU Cycle Determination

W/R#	LEN	CACHE#	MKEN#	Cycle Description	Burst Length
0	0	1	—	Non-Cacheable 64-Bit Read	1
0	0	—	1	Non-Cacheable 64-Bit Read	1
1	0	1	—	64-Bit Write	1
—	0	1	—	I/O and Special Cycles	1
0	1	1	—	Non-Cacheable 128-Bit Read	2
0	1	—	1	Non-Cacheable 128-Bit Read	2
1	1	1	—	128-Bit Write	2
0	—	0	0	Cache Line Fill	4
1	—	0	—	Cache Write-Back	4

NOTE:
If MRO# is asserted to the 82495XP, the effect on i860 XP CPU cycle determination is the same as when MKEN# = 1.

6.6 Consecutive Cycles

Because a 82495XP line can be longer than a CPU line, there are circumstances where a read miss will be to a line that is currently being filled. If this is the case, the 82495XP treats this like a read hit, but supplies data after CRDY# for the line fill. Data is supplied from the 82490XP array.

6.7 CPU/Memory Bus Concurrency

The 82495XP allows concurrency between the CPU and memory buses. CPU bus cycles will either be serviced locally by the 82495XP (hits) or require memory bus service. Whenever a CPU cycle requires memory bus service, it will be scheduled to run on the memory bus, and CPU bus activity will be allowed to continue.

Examples of concurrency are:

— Snoops and CPU bus operations

— Posted writes with CPU and memory bus operations

— CPU bus operation on the back of long line fills (82495XP line longer than the CPU line)

— Allocations and replacements with CPU and memory bus operations.

In certain cases, consistency of data and prevention of deadlocks preclude concurrency. Problems may occur when the current memory bus cycle changes the tag state and therefore affects the operation of the next CPU cycle request. In those cases the 82495XP will hold concurrency to ensure data consistency. Handling of those cases is completely transparent to the MBC.

The 82490XP supports two modes of memory bus operation: clocked mode and strobed mode. In clocked mode, memory bus signals are sampled by the 82490XP on rising edges of MCLK. Similarly, memory bus data and signals are output by the 82490XP with respect to MCLK (or MOCLK) rising edge transitions.

In strobed mode, memory bus signals are sampled or output with respect to rising and falling edges of other signals. Strobed mode has the advantage of not requiring setup and hold times to a CLK or MCLK edge.

6.8 Memory Bus Modes

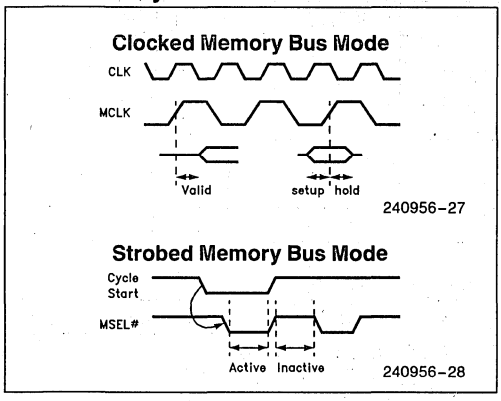


Figure 6-16. Clocked and Strobed Mode Sampling

6.8.1 CLOCKED MODE

In clocked mode operation MCLK is used to reference the signals MDATA0–MDATA7, MSEL#, MFRZ#, MZBT#, MBRDY#, and MEOC#. Clocked mode will be selected if the 82490XP detects a clock at its MCLK input after RESET. MCLK need not have any relation to CLK. If this is the case, the memory bus is said to be operating in "clocked asynchronous" mode. If MCLK = CLK, the memory bus is operating in "clocked synchronous" mode. If MCLK × N = CLK (where N = 2, 3, 4 ...), the memory bus is operating in "clocked divided synchronous" mode. These three clocked modes, asynchronous, synchronous, and divided synchronous, are not differentiated by the 82490XP.

MOCLK controls a transparent latch at the 82490XP data output pins. If a clock is provided at this input, data is latched with MOCLK going low. This clock is available in clocked mode only. MOCLK allows the system to provide a greater MDATA hold time by skewing MOCLK from MCLK. If MOCLK is tied high, MDATA is driven from MCLK.

6.8.1.1 Synchronous Clocked Mode

In synchronous clocked mode MCLK = CLK. This means the CPU clock is used for 82495XP, 82490XP, and the memory bus. A synchronous memory bus allows memory to communicate with the 82495XP without synchronizers since the 82495XP runs with CLK. With a synchronous design, however, high clock frequencies must be routed to all parts of a system with minimal skew. This may not be possible with future projected frequencies. A synchronous memory system and memory bus controller must be redesigned when future speed upgrades are required.

6.8.1.2 Asynchronous Clocked Mode

In asynchronous clocked mode, MCLK is not the same frequency as CLK. Some memory signals, since they reference MCLK, must be synchronized to CLK to communicate with the 82495XP. For example, when a cycle completes, the memory system asserts a signal, driven from MCLK, to the memory bus controller which will be synchronized to CLK to become CRDY#. This is because CRDY# is synchronous to CLK and not MCLK.

Asynchronous mode allows the rest of the system to run at a lower frequency than the CPU CLK. Not only does this simplify system design, but allows the designer to place hooks to allow the same design to scale easily to a higher frequency. If all the features of the 82495XP are used properly, an asynchronous memory design does not have to incur much synchronization penalty. For example, MEOC# is synchronous to the memory environment (MCLK). This allows the memory system to end the current cycle and start the next before CRDY# is synchronized in the CPU environment.

6.8.1.3 Divided Synchronous Clocked Mode

Divided synchronous clocked mode is a subset of synchronous clocked mode. It allows two things to happen: One, the memory system is capable of communicating with the 82495XP without synchronization. Two, a slower frequency clock may be routed around the system.

Divided synchronous mode still requires clock skew restrictions. It also carries the same scalability drawbacks that full synchronous mode does.

6.8.2 STROBED MODE

Strobed mode is configured on the 82490XP by strapping MCLK high. In strobed mode:

— MDATA0–MDATA7 are sampled with respect to edges of MEOC#, MISTB, and MOSTB.

— For write cycles, MFRZ# is sampled when MEOC# goes active.

— MZBT# is sampled when MSEL# is inactive, and is latched when MSEL# goes active. MZBT# is also sampled for the next operation when MSEL# is active and MEOC# goes active.

By not using MCLK, strobed mode has no setup and hold time restrictions, and is scalable to higher frequencies. Strobed mode does, however, require synchronization to 82495XP CLK synchronous signals.

6.9 Memory Bus Operation

All data is handled by the 82490XP cache RAMs. The 82495XP instructs the 82490XP whether to use the data array or buffers, and specifically which buffer to use. The MBC is responsible for bursting data in and out of the 82490XP's, in and out of the CPU during miss cycles, and indicating when the operation is finished. Communication between the 82490XPs and memory bus may be done in a clocked mode or strobed mode. See the Memory Bus Modes section for more details.

A 82490XP has 4 memory buffers. It has 2 memory cycle buffers, one write-back buffer, and one snoop buffer. Each buffer is capable of holding an entire 82495XP line of the longest configurable length.

The memory cycle buffers of the 82490XP are used for posting writes and holding data during 82495XP/82490XP line-fills. The write-back buffer is used for holding data from a cache replacement. This data is ready to be written out, and the write-back buffer is snoopable. The snoop buffer is used to hold modified data that has been hit by a snoop. Since snoop hits are the highest priority cycle, this buffer will be emptied before any other cycle or snoop request begins.

6.9.1 82490XP BUFFERS AND MUXES

The 4 82490XP memory buffers are all multiplexed (muxed) to the memory bus. The mux is used to select which buffer is on the bus, and specifically which slice of that buffer is on the bus. MBRDY# assertion increments a counter for this mux which selects the next slice of that buffer.

The counter used to increment through the buffer slices is called the memory burst counter. The memory burst counter follows the CPU burst order depending on the subline address of the initial slice. Once the MBC is finished with a buffer, MEOC# is asserted to switch the mux to the next buffer to be used. MEOC# will also reset the counter and latch the last slice of data.

On the CPU side, the 82490XP contains a CPU buffer and mux. The CPU buffer captures data from the appropriate memory buffer or 82490XP array to feed it to the CPU. The mux selects which slice is muxed to the CPU bus. The counter for this mux is incremented with BRDY#.

The 82490XP array contains a mux that selects which way, based on the MRU algorithm, will be read during hit cycles. This mux is used during write cycles to write to the correct way.

6.9.2 MEMORY CYCLE BUFFERS

There are 2 memory cycle buffers in the 82490XP. They are used for line-fills, allocates, and memory writes. The buffers are 64-bits wide (per 82490XP) to support 8 transfers with 8 memory bus I/O pins (maximum configuration). The 82490XP alternates use of these buffers. When one buffer has a posted write or is being used for a memory read, the other one is available for the next cycle.

During allocation cycles, read for ownership may be implemented by using the MFRZ# signal. If MFRZ# is sampled active during the write cycle, the memory cycle buffer will freeze the write data in the buffer so the subsequent line-fill fills around it. This way the write cycle need not be written to memory. The line must be tagged as modified.

6.9.3 WRITE BACK BUFFER AND SNOOP BUFFER

The write back buffer and snoop buffer are both 64-bits to handle the maximum 82495XP line length. The write back buffer is used when replaced data must be written back to main memory (including FLUSH and SYNC cycles) and the snoop buffer is used when data must be written out from a snoop hit.

Before a line fill begins, the 82495XP checks to see if it must remove a modified line to make room for the line-filled line. If so, the modified line is placed in the write back buffer and the line-fill is filled through a memory cycle buffer. Should the line-fill be selected as non-cacheable, both buffer contents are discarded and the 82490XP array value remains as it was before the line-fill.

There is no need to run the line-fill, replacement (write back), FLUSH, or SYNC cycles contiguously. If a snoop is requested between the two cycles, the write back buffer is snoopable, and data can be written directly out of it if need be.

6.9.4 MEMORY BUS CONTROL SIGNALS

The main memory bus control signals are MSEL#, MEOC#, MBRDY#, and CRDY#. These signals control the 82490XP data path, buffers, and muxes.

MSEL# selects which 82490XPs are being used in the current cycle by qualifying the MBRDY# signal. If MSEL# is inactive, MBRDY# is not recognized for that 82490XP. MSEL# is also used to reset the memory burst counter. If MSEL# goes inactive, the counter is initialized to its starting value. This is use-

ful for aborted/restarted cycles. MSEL# may remain active for many or all cycles. MSEL# must, however, be inactive sometime after RESET to initialize the memory burst counter for the first time.

MEOC# is asserted by the MBC to end finish with the current buffer, and switch the memory bus to the next buffer to be used. MEOC# latches in the last piece of data and resets the memory burst counter before switching to the new buffer.

MBRDY# is used to increment the memory burst counter to select the next slice of data. This will strobe data out of the 82490XP (write cycles) or load data into the 82490XP (read cycles). MBRDY# is ignored by the 82490XP if MSEL# is inactive.

CRDY# finishes the current cycle. Once CRDY# is asserted, the 82490XP disposes of the information in the buffers used in that cycle, and loads information into the 82490XP array. CRDY# must be asserted on the clock or after MEOC# is asserted for a particular cycle.

6.9.5 82490XP DATA PATH

An example 82490XP read data path is shown in Figure 6-6. The path between the CPU and memory bus is a flow-thru' path, not a clocked path. Each entire 82495XP cache line of data in the CPU buffer is available at the memory buffer with some propagation delay. Likewise, each entire 82495XP cache line of data in the memory buffer is available in the CPU buffer with some propagation delay. Data is burst into and out of the memory buffer using MBRDY# or MISTB/MOSTB. Data is burst into and out of the CPU buffer using BRDY#. This means there is no synchronization required between memory and CPU data paths.

To give an example how the path works, during a CPU line fill, data may be returned to the CPU in two different fashions. One, each time the memory buffer fills a dword, BRDY# may be asserted a clock later to burst it back to the CPU. Two, the memory buffer can be filled and then BRDY# asserted on four consecutive clocks to burst data back to the CPU.

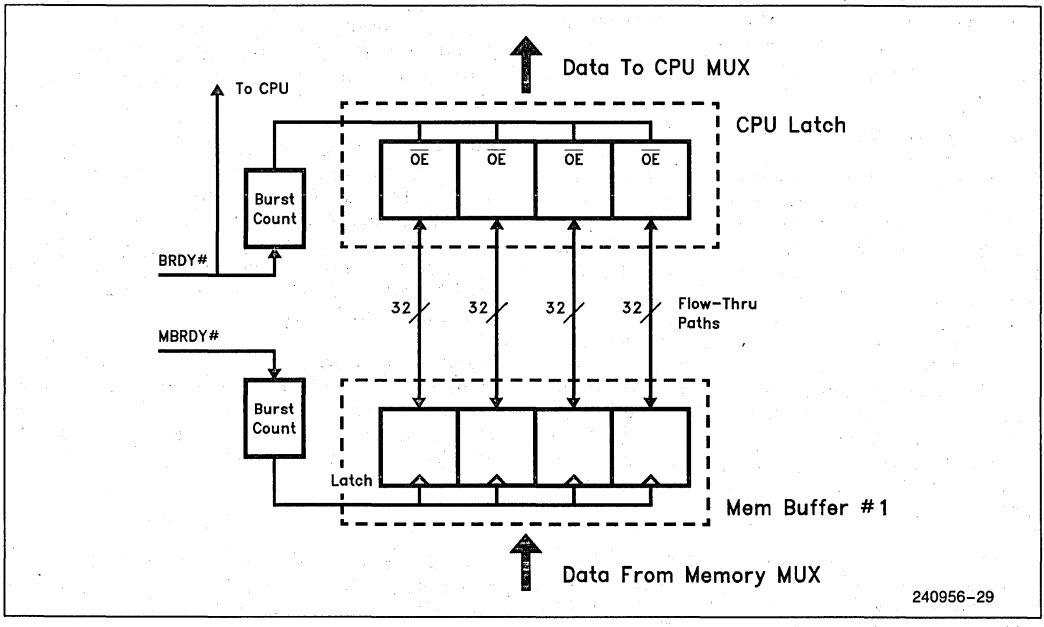


Figure 6-17. 82490XP Read Data Path

6.9.6 WRITE CYCLES

There are 3 basic types of write cycles: CPU generated write cycles, write back cycles caused by a cache replacement, and snoop write back cycles caused by a snoop hit. All write cycles, except the snoop write back, begin with CADS# assertion. The snoop write back cycle begins with SNPADS#.

6.9.6.1 CPU Generated Write Cycles

When the CPU begins a write cycle, four things can happen to it. One, the CPU write is a hit to a modified or exclusive line. In this case the write is terminated by the cache immediately and invisibly to the MBC.

Two, the write is to a shared location. This type of write is posted to the 82490XP memory cycle buffers and the cycle is terminated by the cache. If a memory cycle buffer is occupied with a write cycle, the CPU waits until the previous write completes. The write cycle must be written to the memory bus so that other copies of the write in other caches be invalidated.

Three, the write is a cache miss. This type of write is posted to a memory cycle buffer if the 82490XP is not waiting for another posted write to complete. If PALLC# is asserted, the write may be turned into an allocation.

Four, the write is a LOCKed write. LOCKed writes are posted regardless of the tag state. The write is then treated as if it were a miss except that there is no change in the tag state and no allocation allowed.

6.9.6.2 Cache Generated Write Cycles

The 82495XP/82490XP will generate a write cycle in three situations: a line fill or allocation causing a cache replacement, a snoop hit to a modified location, and write backs caused by FLUSH or SYNC. Write back caused by FLUSH or SYNC are indestinguishable from write-back cycles caused by replacement. Cache generated write cycles are the length of a cache line.

Cache replacements and FLUSH/SYNC cycles cause a line (or two lines if sectored) of cache data to be placed in the write-back buffer of the 82490XP. If no cycle is pending, CADS# is asserted and the data is written out. If a snoop hits the write-back buffer, the data is written out via SNPADS# like a normal snoop hit. The write back is then cancelled since the data was written through the snoop hit.

A snoop hit to a modified location causes a line of cache data to be written out to memory. Snoop hits are the highest priority cycle and must be serviced immediately. A snoop hit to a modified location causes the snooped line to be written to the snoop buffer of the 82490XP. SNPADS# is then asserted and the snoop is written out.

6.9.6.3 Memory Bus Controller Responsibility

The MBC recognizes a write cycle with CADS# and CW/R# (or SNPADS# for snoop cycles). If MCACHE# is active, the MBC knows the cycle is a write back cycle, otherwise it is a CPU-generated cycle.

CPU-generated write cycles are written to the main memory bus so that other caches can invalidate their copies of this information. The other caches do this by snooping with SNPINV active during snoop initiation if they detect a write cycle on the bus.

Once the MBC detects CDTS# active, the data will be available for writing in the next clock in the appropriate 82490XP buffer. The MBC should assert MSEL# so bursting is enabled, and burst through the write using MBRDY# (or MOSTB). MSEL# activation also caused MZBT# to be sampled. MZBT# must be inactive at this time if the data will be written according to CPU burst order.

Once the write cycle is complete, MEOC# must be asserted to end the write cycle and switch to the next pending cycle. If this write cycle is turned into an allocation, MFRZ# is sampled with MEOC# to freeze the write data in the 82490XP.

MEOC# simply switches buffers from the current one in use to the buffer of the next pending cycle. CRDY# needs to be asserted to actually end the cycle and allow the 82495XP and 82490XP to dispose of the information.

6.9.6.4 Write Allocation and Read for Ownership

The 82495XP/82490XP supports write allocation. An allocation cycle is a read of a cache line caused by a write miss to the same location. In its simplest form, a write miss is written to memory, then the 82495XP requests a line from that same location. Meanwhile, the CPU only sees the write cycle.

Write allocation may only be done if PALLC# is active during CADS# of the write cycle. For the allocation to occur, MKEN# must be returned active during KWEND# of the write cycle. The write cycle may

be an actual write or a "dummy" write. Dummy writes are write cycles that are terminated in the 82495XP and 82490XP as if they were normal writes, but the data is not actually written to memory. This saves a data write to memory.

During write allocation, the write cycle will progress like a normal write cycle except MKEN# must be active during KWEND# activation. If the write cycle is a dummy write, MFRZ# must be used with MEOC# so that the line filled data is read around the write data into the 82490XP buffer. The line fill cycle is like any other line fill cycle except the CPU doesn't get any data. If a dummy write was performed, DRCTM# must be asserted during SWEND# activation to fill the line to the M state, and any cache supplying the data must invalidate its copy.

Using dummy write cycles and filling data to the M state from another cache or memory is called Read For Ownership. This is because ownership is being transferred. In read for ownership cycles, memory is avoided as much as possible. First, the dummy write cycle avoids memory. Second, a line fill is performed as a cache to cache transfer with DRCTM# asserted. All caches were snooped with invalidate to eliminate their copies.

For allocation cycles, SWEND# is not sampled for the write portion of the allocation.

6.9.7 READ CYCLES

The CPU initiates all read cycles. These are usually line fills to the CPU and line fills to the 82495XP/82490XP. The signal MCACHE# is output with CADS# to indicate whether this cycle may or may not be cacheable. If cacheable, MKEN# is returned by the MBC to ultimately determine cacheability.

Read hit cycles are serviced by the cache without MBC intervention. The only read cycles seen by the MBC (except I/O or special) are read misses and locked read cycles.

Read misses cause CADS# to be asserted at most two clocks after ADS# of the CPU read cycle. If cacheable, as determined from MCACHE#, the MBC will return 4 BRDYs back to the CPU and 4 or 8 MBRDYs to the 82495XP/82490XP. If the transfer is non-cacheable, the i860 XP CPU LEN and CACHE# outputs indicate the number of transfers to be given to the CPU. MBRDY# need not be used in the transfer if only a single piece of data is required by the CPU.

If the read cycle is cacheable, it may cause another cached line to be bumped out of the cache. This is called a replacement and, if modified, causes a write back cycle. While one of the 82490XP memory buffers is being filled for the line fill, the write back buffer is loaded. If the line fill turns out to be non-cacheable at the end of the transfer, the write-back buffer is discarded, and the line in the cache remains valid. Otherwise, CADS# will be generated after the read cycle so the write back can be performed. The write back need not happen immediately after the line fill since the write-back buffer is snoopable.

All locked reads go to the memory bus. If the read is a cache hit to M', the 82495XP/82490XP will ignore the data that the MBC returns, and provide it from its array. Locked reads are not cacheable by the CPU or the 82495XP/82490XP. Snoop write-backs that are a result of a LOCKed read/write request must update memory.

6.9.7.1 Memory Bus Controller Responsibility

Once the MBC sees a read cycle on the memory bus, it must determine whether the read is cacheable or non-cacheable using MCACHE# and its own address decoding. If non-cacheable, the CPU expects a number of transfers as determined by its LEN and CACHE# outputs. If cacheable, the CPU expects 4 transfers, and the cache expects 4 or 8 (configuration dependent).

MKEN# is sampled during KWEND# to determine cacheability. Before MKEN# is sampled, KEN# is active assuming cacheability for the CPU. MKEN# must be sampled 1 clock before the first BRDY# to make the cycle non-cacheable.

Once the read cycle is given to the memory system, all 82495XP/82490XP caches snoop to see if they contain the data in modified form. If so, the MBC must abort the cycle in memory and receive the data directly from the 82495XP/82490XP that has it, or wait until that cache writes it to memory. If the data transfer avoids memory, ie goes cache to cache, DRCTM# must be asserted with SWEND# to place the line in the M' state and the cache giving the data must invalidate its copy.

MSEL# is activated and MBRDY# (or MISTB) used to sample input data from the read cycle. Once CDTS# has been seen active, the CPU read data path is clear. BRDY# may be returned to the CPU sometime after each MBRDY# for each piece of input data (see MDATA setup to CLK). Once the transfer completes, MEOC# and CRDY# are asserted to complete the cycle in the 82495XP/82490XP.

6.9.8 I/O AND SPECIAL CYCLES

I/O and special cycles (flush, etc) are decoded by the 82495XP and not posted. These cycles wait until all buffers have been written, and all cycles have been completed, before they cause CADS# assertion. The CPU waits until the special cycle ends with the MBC's BRDY# assertion before it continues.

When the 82495XP/82490XP is performing a FLUSH or SYNC, many write back cycles are required. These cycles look like ordinary write back cycles, and should be handled as such. FSIOUT# is active during these write back cycles, so when FSIOUT# goes inactive the cycle is complete and the memory bus controller can supply BRDY# to the CPU.

6.10 Different Bus Widths

The 82490XP is capable of supporting either 64- or 128-bit memory bus widths. Depending on the configuration, the 82490XP's CPU and I/O busses may be multiplexed. The following diagram shows how an i860 XP CPU may be connected to a 128-bit memory bus:

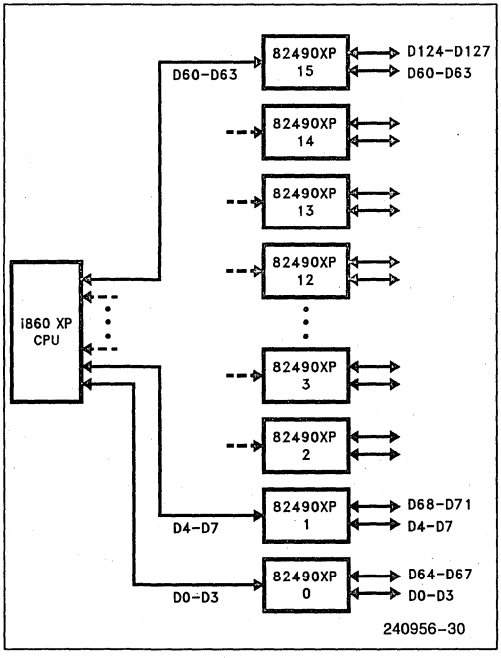


Figure 6-18. 82490XP On Wide Bus

In this example, the CPU port of the 82490XPs is in x4 mode and the memory bus port is in x8 mode. This allows all 128 bits of the memory bus to be multiplexed to the 64-bit CPU bus.

For read cycles, each MBRDY# loads 8 bits into each 82490XP. This is 128-bits of data. It will take 2 BRDY# assertions to load this into the CPU. The first BRDY# assertion loads the first 4 bits onto the CPU bus, and the next BRDY# assertion loads the remaining 4 bits.

For a 64-bit write cycle, the data is available at the on the appropriate data bits. On the i860 XP CPU with a 128-bit bus, this is determined by CPU address bit A3. The other data bits are undefined. For write-back cycles, all 128 bits are available at once. MBRDY# assertion will strobe the next 128 bits on the memory bus.

2

7.0 DETAILED PIN DESCRIPTIONS

The following chapter provides a detailed description of each pin of the 82495XP and 82490XP. The pins have been categorized by function. Each pin description has a heading which summarizes the most important aspects of the pin. The heading is organized as:

Pin Name

Name Meaning

Pin Function

I/O, 82495XP/82490XP/i860 XP CPU, (location) Signal Type

Synchronous/Asynchronous

for example,

CADS#

Cache Address Strobe

Indicates beginning of cache cycle

Output from 82495XP (pin E3) Cycle Control Signal

Synchronous to CLK

Following the heading are three sections. The first section, Signal Description, provides information of what the signal does, how to use it, and in what modes it operates. The second section, When Sampled or When Driven, indicates all the exact places where the part samples the signal, generates the signal, or neither. The third section, Relation to Other Signals, mentions the other signals that are affected by this signal, synchronization requirements, and shared pins.

All specific information about each pin is provided in this chapter.

7.0.1 CONFIGURATION SIGNALS

These signals are inputs to the 82495XP and 82490XP that are sampled at RESET and alter the configuration and operation of the cache.

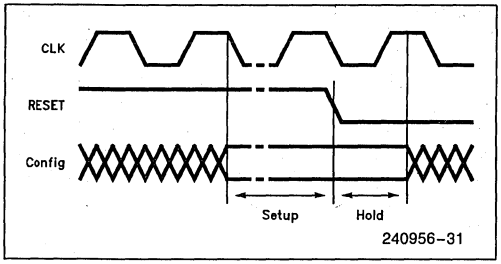


Figure 7-1. Configuration input Setup and Hold

Each set of configuration inputs may have different setup times, but all signals have the same hold time: The signals may be released on the CPU clock edge that RESET is detected inactive. There are some configuration signals that are strapping options and cannot change their value during 82495XP operation.

7.0.2 CPU BUS INTERFACE SIGNALS

These pins comprise the interface between CPU and 82495XP/82490XP. The signals in this interface are not flexible; Chapter 10 addresses the use of these signals. The following are the CPU bus interface signals:

SET0–SET10	TAG0–TAG11	CFA0–CFA6
ADS#	W/R#	D/C#
M/IO#	HITM#	LOCK#
PWT	PCD	LEN
BRDYC1#	KEN#	AHOLD
EADS#	BE0–BE7#	INV
BOFF#		

The majority of these signals must be connected strictly between the i860 XP CPU and the 82495XP. However, a subset of these signals is needed by the MBC to decode the i860 XP CPU cycle in cases where the MBC provides BRDYs to the CPU. For these purposes the following signals must also be inputs to a latch controlled by the 82495XP's BLE# output:

BE0#-BE7#	CACHE#	CTYP
LEN	PCD	PCYC
PWT		

7.0.3 82495XP/82490XP INTERFACE SIGNALS

These pins comprise the interface between the 82495XP and 82490XP. The 82495XP uses these pins to control the 82490XP and its buffers. The signals in this interface are not flexible; Chapter 10 addresses the use of these signals. The following are the 82495XP/82490XP interface signals:

WRARR#	WAY	MAWEA#
BUS#	MCYC#	WBWE#[LR1]
WBA[SEC2]	WBTYP[LR0]	BRDYC2#
BLAST#	BOFF#	

SIGNAL DESCRIPTIONS

7.1 BGT#

Bus Guaranteed Transfer

Signals 82495XP of memory bus controller's commitment to complete the bus cycle.

Input to 82495XP (pin M4) Cycle Progress Signal

Synchronous

7.1.1 SIGNAL DESCRIPTION

The 82495XP owns all bus cycles (initiated by CADS#) until the memory bus controller accepts ownership. During this time cycles may be aborted due to a snoop. The memory bus controller signals its acceptance of ownership by driving BGT# active into the 82495XP. Once BGT# is driven active, the memory bus controller is responsible for completing the cycle on the memory bus. CRDY# signals completion of the cycle.

Once BGT# is asserted, other devices may not perform snoops into the 82495XP until the end of the snooping window, SWEND# activation. The snoop address is latched if SWEND is asserted between BGT# and SWEND#, but the snoop does not begin until after SWEND# is asserted. SNPCYC# will not be asserted until the snoop window ends with SWEND# asserted. The advantage of asserting BGT# early is that it allows the 82495XP to start inquiries to the CPU, load the write-back buffer, and progress forward in the CPU bus pipeline. The disadvantage is that snooping of this 82495XP is now blocked until SWEND# is asserted.

7.1.2 WHEN SAMPLED

After the 82495XP asserts CADS#, it begins sampling BGT# until it is sampled active.

BGT# is a "Don't Care" after it has been recognized for cycle N and prior to the assertion of

CADS# for cycle N+1. In addition, BGT# is a "Don't Care" once a cycle started by CADS# is aborted by a snoop, until the cycle is restored by the re-issueing of CADS#.

7.1.3 RELATION TO OTHER SIGNALS

When implementing BGT# in the MBC the following rules should be used:

1. BGT# must follow every assertion of CADS#, unless the cycle is aborted due to a snoop.
2. It must preceed CRDY# (for line fills and allocations BGT# must preceed CRDY# by at least 3 CLKS).
3. In addition BGT# must be asserted with or before the assertion of KWEND# and SWEND#.
4. BGT# must be asserted with or before the assertion of BRDY# by the MBC.
5. BGT# is not required following the assertion of SNPADS#.
6. BGT# must be asserted with or before MEOC# is asserted.

7.2 BLE#

BE Latch Enable

Controls latching of i860 XP CPU's byte enable and cycle attribute signals

Output of 82495XP (pin C16) Cycle Control Signal

Synchronous to CLK

7.2.1 SIGNAL DESCRIPTION

BLE# is used to control the enable line of an external latch (clock edge triggered '377 type). This latch is used to capture the i860 XP CPU's byte enables (BE0#-BE7#) and CPU cycle attribute signals which do not go through the 82495XP. The 82495XP manages the opening and closing of this latch: when BLE# is active, new values from the CPU enter the latch at each rising edge of CLK.

The 82495XP latches the byte enables after ADS# of a memory bus bound cycle. It relatches this information with CRDY# or CNA# of that cycle if another cycle is pending.

7.2.2 WHEN DRIVEN

The 82495XP latches the BE latch signals 1 clock after ADS# of a memory-bound cycle. Thus latched BE0#-BE7# are valid with CADS#. The 82495XP opens, then closes this latch if a cycle is pending and CNA# or CRDY# is asserted. Thus latched BE0#-BE7# are valid two clocks after CNA# or CRDY#, which is one clock AFTER CADS# for back-to-back cycles. The signals latched in the BE latch are only valid for CPU generated memory bus cycles (ie, not a 82495XP generated writeback or allocation).

7.2.3 RELATION TO OTHER SIGNALS

The following CPU signals must be latched in the BE latch:

BE0#-BE7#	CACHE#	CTYP
LEN	PCD	PCYC
PWT		

All other signals in the 82495XP to CPU interface (listed in sec. 7.0.2) must be connected only between the i860 XP CPU and the 82495XP.

7.3 BRDY#

Burst Ready

Memory Bus Controller Burst Ready input to 82495XP, 82490XP, and i860 XP CPU

Input to 82495XP and 82490XP (82495XP pin P1, 82490XP pin 60) Cycle Progress Signal

Input to i860 XP CPU (BRDY2#, pin U1)

Synchronous to CLK

7.3.1 SIGNAL DESCRIPTION

The BRDY# input to both the 82495XP and 82490XP must be connected to the BRDY# signal which the MBC is providing to the i860 XP CPU's BRDY2# pin. The signal is used by the 82495XP for burst tracking purposes. In the 82490XP, it increments the CPU latch burst counter.

During CPU read cycles, BRDY# allows the next 32 or 64-bit slice of read data to be available at the 82490XP's CDATA outputs (CPU bus) by advancing the CPU latch burst counter. At the same time, BRDY# is latching the previous slice of data into the i860 XP CPU. Refer to chapter 6 for more details.

During CPU write cycles, BRDY# is used to latch each slice of write data into the CPU latches and advance the latch counter.

During CPU special and I/O cycles (which are not posted) BRDY# is used to end the cycle.

BRDY# must not be asserted until the bus is granted (BGT# asserted) and until the data path is ready for transferring (CDTS# is asserted).

7.3.2 WHEN SAMPLED

BRDY# is sampled by the CPU, the 82495XP, and the 82490XP at every CLK edge. It must always meet proper setup and hold times to CLK. Even though the CPU latch may not be in use, BRDY# assertion will still advance the latch counter.

7.3.3 RELATION TO OTHER SIGNALS

BRDY# controls the CPU and 82490XP CPU latches. BRDY# has the following implication rules:

1. The last BRDY# for cycle N must be asserted 2 clocks before MEOC# for cycle N+1.
2. BRDY# ≥ BGT#
3. BRDY# > CDTS#

7.4 C490LDRV

82490XP Low Drive Buffer

Selects the 82495XP low capacitance driving buffers

Input to 82495XP (pin M3) Configuration Signal

Synchronous to CLK

7.4.1 SIGNAL DESCRIPTION

C490LDRV selects the driving strength of the 82495XP buffers that interface to the 82490XP. Refer to the layout specifications for information how C490LDRV should be connected.

7.4.2 WHEN SAMPLED

C490LDRV is a configuration input sampled like Figure 7-1. C490LDRV requires a setup time of 4 CPU clocks. After sampling, C490LDRV is a "don't care" until it is sampled as the BGT# pin after the first CADS# assertion.

7.4.3 RELATION OT OTHER SIGNALS

C490LDRV shares a pin with BGT#.

7.5 CADS#

Cache Address Strobe

Indicates beginning of cache cycle

Output from 82495XP (pin E3) Cycle Control Signal

Synchronous to CLK

7.5.1 SIGNAL DESCRIPTION

CADS# requests the execution of a memory bus cycle to the MBC, and indicates that the cycle attributes (ie. CD/C#, CM/IO#, CW/R#, PALLC#, etc.) are valid.

If the 82495XP receives a snoop hit to an [M] state line before BGT# is asserted by the MBC, the current CADS# is aborted and reissued after the snoop has completed. If the current line (issued by the stalled CADS#) is invalidated by the snoop, then that CADS# is cancelled (ie. will not be reissued after the snoop is completed).

CADS# is a glitch-free signal.

7.5.2 WHEN DRIVEN

CADS# is asserted by the 82495XP for exactly one CLK, and is always a valid logic level.

7.5.3 RELATION TO OTHER SIGNALS

CADS#, when asserted, indicates that the cache cycle control and attribute signals (ex. CD/C#, NENE#, CW/R#, etc.) are valid.

Since allocations do not require BRDY#s to the CPU, the CDTS# of an allocation cycle will always

occur with CADS# of the allocation. In normal cycles the 82495XP will generate CADS# followed by CDTS#.

CADS# == CDTS# for all write-through cycles.

Once CADS# is active, PALLC#, CWAY, CDTS#, and BUS# are valid. Address and cycle specification signals (MSET0–MSET10, MTAG0–MTAG11, MCFA0–MCFA6, CW/R#, CM/IO#, CD/C#, RDYSRC, MCACHE#, NENE#, SMLN#, KLOCK#, and CPLOCK#) will be valid with CADS# active as well.

Every CADS# initiated cycle requires a BGT# and CRDY# input from the MBC.

CADS# and SNPADS# will never be asserted on the same CLK.

7.6 CAHOLD

82495XP AHOLD Output

Self-test result and AHOLD output status

Output of 82495XP (pin G4) Test Signal

Synchronous to CLK

7.6.1 SIGNAL DESCRIPTION

CAHOLD has two functions. One, it indicates the result of the built-in self-tests of the 82495XP. Two, it represents the 82495XP AHOLD into the i860 XP CPU.

The 82495XP drives CAHOLD after the 82495XP self-tests have completed. CAHOLD should be latched when FSIOUT# goes inactive after reset. If CAHOLD is high, the self-tests have passed, otherwise they have failed.

When the 82495XP drives AHOLD to the i860 XP CPU, it also drives CAHOLD, thus providing a means of tracking inquire cycles and back invalidations for performance monitoring.

7.6.2 WHEN DRIVEN

CAHOLD is always at a valid logic level. During self-test, CAHOLD is held until the clock edge that FSIOUT# is sampled inactive. After self-test, or reset, CAHOLD is asserted whenever the 82495XP asserts AHOLD.

7.6.3 RELATION TO OTHER SIGNALS

CAHOLD reflects the value of AHOLD except during self-test. During self-test, the value of CAHOLD should be latched with the falling edge of FSIOUT# to determine pass/fail.

7.7 CD/C#

Cache Data/Code

Indicates whether current cycle is Code or Data

Output from 82495XP (pin D3) Cycle Control Signal

Synchronous to CLK

7.7.1 SIGNAL DESCRIPTION

CD/C#, along with CW/R# and CM/IO#, is a 82495XP cycle definition signal. It indicates the type of bus cycle being requested of the MBC. CD/C# can be pipelined by the memory bus controller (by using the CNA# input to the 82495XP).

7.7.2 WHEN DRIVEN

CD/C# is valid in the same CLK as CADS# and remains valid until CRDY# or CNA#. C/DC# is always a valid logic level.

7.7.3 RELATION TO OTHER SIGNALS

Address and cycle specification signals (MSET0–MSET10, MTAG0–MTAG11, MCFA0–MCFA6, CW/R#, CM/IO#, CD/C#, RDYSRC, MCACHE#, NENE#, SMLN#, KLOCK#, and CPLOCK#) will be valid with CADS#.

7.8 CDATA0–CDATA7

CPU Data Bus Connection

Data Bus Connection from 82490XP to CPU

Input/Output to 82490XP (pins 48, 54, 49, 55, 46, 51, 52, 57)

Isolated Interface

7.8.1 SIGNAL DESCRIPTION

CDATA0–7 is the 82490XP data bus connection to the CPU. All or part of these 8 pins will be used in connecting the 82490XP to the CPU depending on the cache configuration. See layout information for details.

7.9 CDTS#

Cache Data Strobe

Indicates availability of CPU data/data bus

Output from 82495XP (pin F4) Cycle Control Signal

Synchronous to CLK

7.9.1 SIGNAL DESCRIPTION

For read cycles, CDTS#, when asserted, indicates that in the next CPU clock the data bus path is available. This is the earliest time in which BRDY# may be supplied to the CPU. For CPU initiated write cycles, it indicates that the data is available on the memory bus. For i860 XP CPU inquire cycles, CDTS# informs the MBC that the last piece of inquire data is valid on the CPU bus.

Usage of this signal allows complete independence between address strobes (CADS# and SNPADS#) and data strobe. CDTS# allows the 82495XP to signal the MBC that a new cycle has begun as soon as addresses are available. This allows memory bus cycles to start before data is ready to be given/taken.

CDTS# is a glitch-free signal.

7.9.2 WHEN DRIVEN

CDTS# is asserted for one CLK, at the same time or later than CADS# for any given cycle.

7.9.3 RELATION TO OTHER SIGNALS

When the MBC samples CDTS# asserted, it can begin providing BRDY#s for the read cycle to the CPU in the next CLK. CDTS# must always be asserted before CRDY# and must be asserted prior to the first BRDY#.

The CDTS# of an allocation will always occur with CADS# of the allocation. In normal cycles the 82495XP will generate CDTS# following CADS#.

CDTS# will be asserted at least one CLK after SNPADS#.

7.10 CFG0–CFG2

Configuration Pins

Determine Cache Characteristics

Input to 82495XP (pins L4, Q1, M4,) Configuration Signals

Synchronous to CLK

7.10.1 SIGNAL DESCRIPTION

CFG0–CFG2 are the 3 cache configuration inputs that determine cache characteristics such as line ratio, tag size, and lines per sector. During RESET, this information is passed on to the 82490XPs. The following table maps CFG0–CFG2 to their respective configurations for the i860 XP CPU:

Config No.	Line Ratio	Lines/ Sector	No. of Tags	CFG2	CFG1	CFG0
1	1	1	8K	0	0	1
2	2	1	4K	1	1	1
3	1	2	8K	0	0	0
4	2	1	8K	0	1	1
5	4	1	4K	1	1	0

7.10.2 WHEN SAMPLED

CFG0–CFG2 are sampled like Figure 7-1 with a set-up time of at least 10 CPU clocks. After sampling, CFG0, CFG1, and CFG2 become cycle progress input signals to the 82495XP and are sampled after CADS# of the first cycle.

7.10.3 RELATION TO OTHER SIGNALS

CFG0 shares a pin with CNA#, CFG1 shares a pin with SWEND#, and CFG2 shares a pin with KWEND#.

7.11 CLK

i860 XP CPU, 82495XP, 82490XP Clock

Input to the 82495XP (D11)

7.11.1 SIGNAL DESCRIPTION

The CLK input determines the execution rate and timing of the 82495XP, 82490XP, and CPU. Pin timings are specified relative to the rising edge of this signal. The i860 XP CPU, 82495XP, and 82490XP requires TTL levels on CLK for proper operation.

7.12 CM/IO#

Cache Memory/IO

Indicates whether current cycle is Memory or IO

Output from 82495XP (D4) Cycle Control Signal

Synchronous to CLK

7.12.1 SIGNAL DESCRIPTION

CM/IO#, along with CW/R# and CD/C#, is a 82495XP cycle definition signal. It indicates the type of bus cycle being requested of the MBC. CM/IO# can be pipelined by the memory bus controller (CNA# input to the 82495XP).

7.12.2 WHEN DRIVEN

CM/IO# is valid in the same CLK as CADS#, and remains active until CRDY# or CNA#.

7.12.3 RELATION TO OTHER SIGNALS

Address and cycle specification signals (MSET0–MSET10, MTAG0–MTAG11, MCFA0–MCFA6, CW/R#, CM/IO#, CD/C#, RDYSRC, MCACHE#, NENE#, SMLN#, KLOCK#, and CPLOCK#) will be valid with CADS# assertion.

7.13 CNA# [CFG0]

82495XP Next Address Enable

Dynamically pipelines CADS# cycles

Input to 82495XP (pin L4) Cycle Progress Signal

Synchronous to CLK

7.13.1 SIGNAL DESCRIPTION

CNA# is used by the MBC to dynamically pipeline CADS# cycles. When active it indicates to the 82495XP that the next MBC request can be started. Only one level of pipelining is allowed in the 82495XP.

CNA# is an optional input for all cycles initiated with CADS#.

7.13.2 WHEN SAMPLED

CNA# is sampled starting in the first CLK in which BGT# is sampled active until CRDY# is sampled active. CNA# is then ignored until the BGT# of the next cycle.

CNA# is ignored during snoop write-back cycles.

7.13.3 RELATION TO OTHER SIGNALS

Once the 82495XP samples this signal active, it issues the CADS# for the next memory bus cycle as soon as one begins.

CNA# is recognized between BGT# and CRDY# or CDTS# and CRDY# of a given cycle.

7.14 CRDY#

Cache Ready

Ends a cycle in the 82495XP/82490XP

Input to 82495XP and 82490XP (pins M2, 43) Cycle Progress Signal

Synchronous to CLK

7.14.1 SIGNAL DESCRIPTION

CRDY# is used by the 82495XP and 82490XP to end a memory bus cycle. CRDY# indicates full completion of the cycle and allows the 82495XP/82490XP to free internal resources for the next cycle. In the 82490XP, this means that the current memory buffer in use is emptied (put in array, discarded, etc). In the 82495XP, CRDY# assertion allows 82495XP cycle progress signals (BGT#, KWEND#, SWEND#) to be sampled for the next cycle if pipelining is used.

CRDY# is required for all 82495XP/82490XP memory bus cycles, including snoop cycles. CRDY# must be asserted to the 82495XP and 82490XP at the same time.

7.14.2 WHEN SAMPLED

CRDY# for a given cycle is ignored until KWEND# is returned for that cycle. If KWEND# is not required for the cycle, CRDY# is ignored until BGT#. When CRDY# is ignored, it may violate setup and hold times.

7.14.3 RELATION TO OTHER SIGNALS

CRDY# must be sampled by the 82495XP and 82490XP at the same time. For the 82495XP, CRDY# has many cycle implication rules:

1. CRDY# > CDTS#
2. CRDY# > BGT#
3. CRDY# > BGT# + 2 clocks if cycle is a line-fill or allocation
4. CRDY# > KWEND# if cycle is a line-fill or write-through with potential allocation (PALLC# = 0)

For the 82490XP, CRDY# has three basic rules:

1. MEOC# for cycle N must be sampled with or before CRDY# for cycle N.
2. MEOC# for cycle N+1 must be sampled at least 2 CPU clocks after CRDY# for cycle N.
3. CRDY# for cycle N+1 must be after the last BRDY# for cycle N.

MBRDY# fills the current 82490XP memory buffer. CRDY# emties this buffer and makes it available for new cycles. CRDY# may be asserted on the same clock as MEOC# which may be asserted on the same clock as MBRDY#.

CRDY# shares a pin with SLFTST#.

7.15 CWAY

Cache Way

Indicates WAY used by the current cycle

Output from 82495XP (pin J3) Cycle Control Signal

Synchronous to CLK

7.15.1 SIGNAL DESCRIPTION

CWAY is a cycle definition signal which indicates to the MBC the WAY used by the requested cycle. On line-fills it indicates the way the line will be loaded. For write-hits (to [S] state or LOCKed) it indicates the way which was a hit. For write-backs it indicates the way that was written-back.

CWAY is utilized by external tracking machines in order for the 82495XP tags to be accurately duplicated.

7.15.2 WHEN DRIVEN

CWAY is valid together with CADS# and remains valid until CRDY# or CNA#.

7.15.3 RELATION TO OTHER SIGNALS

CWAY is valid with CADS#.

7.16 CW/R#

Cache Write/Read

Indicates whether current cycle is write or read

Output from 82495XP (pin E4) Cycle Control Signal

Synchronous to CLK

7.16.1 SIGNAL DESCRIPTION

CW/R#, along with CD/C# and CM/IO#, is a 82495XP cycle definition signal. It indicates the type of bus cycle being requested of the MBC. CW/R# can be pipelined by the memory bus controller (CNA# input to the 82495XP).

7.16.2 WHEN DRIVEN

CW/R# is valid in the same CLK as CADS# and is valid until CRDY# or CNA#.

7.16.3 RELATION TO OTHER SIGNALS

Address and cycle specification signals (MSET0–MSET10, MTAG0–MTAG11, MCFA0–MCFA6, CW/R#, CM/IO#, CD/C#, RDYSRC, MCACHE#, NENE#, SMLN#, KLOCK#, and CPLOCK#) will be valid with CADS#.

7.17 DRCTM#

Memory Bus Direct to [M] State

Signals 82495XP to tag data direct to the [M] state, skipping the [E] and [S] states.

Input to the 82495XP (pin M1) Cycle Attribute Signal

Synchronous to CLK

7.17.1 SIGNAL DESCRIPTION

DRCTM# is an input to the 82495XP from the memory bus. When sampled active at the end of the snooping window (SWEND# activation), the 82495XP moves the line fill in progress directly to the [M] state.

There are three cases in which this is useful.

1. Simplifies External State Tracker

   External trackers can only track the [M], [S], and [I] states. The [E] state can not be tracked externally since cache write hits internally change [E] state lines to [M] state. DRCTM# can be used to eliminate the [E] state from the MESI protocol.

2. Read For Ownership

   During a write miss with allocation the write may go to the memory buffer and not be written to memory. A read from memory, in conjunction with the MFRZ# signal asserted, reads the data to fill around the bytes written by the CPU. The contents of the memory buffer are then entered into the cache. The cache would normally tag this data in the [E] state (The cache assumes the write went to main memory). The system has the option of never completing the write to memroy (increases performance by completing the allocation quicker). If the write is not performed to memory, the cache is the only owner of the new data and therefore the cache entry must be tagged to the [M] state.

3. Cache to Cache Transfer

   A cache to cache transfer may occur as a result of a snoop. For example, if CPU/Cache 1 performs a read from main memory and CPU/Cache 2 flags it as a snoop hit to an [M] state line. To expedite the transfer, the system may perform the writeback from CPU/Cache 2 directly to CPU/Cache 1, bypassing memory. CPU/Cache 1 assumes the write-back went to memory and would normally tag the line to the [S] state. Since the system did not perform the write to memory, the system should drive DRCTM# to force the line to the [M] state. In addition, the line should be invalidated in CPU/Cache 2 by driving SNPINV.

7.17.2 WHEN SAMPLED

DRCTM# is synchronous to CLK. It is only sampled when SWEND# is active (the end of the snooping window). When SWEND# is inactive DRCTM# is ignored and does not have to meet setup and hold times.

7.17.3 RELATION TO OTHER SIGNALS

DRCTM# (direct to [M]) and MWB/WT# (write policy) combine to define the memory bus attributes and are sampled on CLK at the end of the snooping window (SWEND# activation).

If MRO# is sampled active during KWEND#, DRCTM# is ignored.

7.18 FLUSH#

Flush

Causes a 82495XP Cache Flush

Input to 82495XP (N4) Cache Synchronization Signal

Asynchronous input

7.18.1 SIGNAL DESCRIPTION

This signal causes the 82495XP to flush all its modified lines to main memory. The flushing of modified lines require the 82495XP to perform back-invalidation and inquire cycles to the CPU. At the end of flush, the 82495XP tag array will be completely invalidated.

FLUSH# will invalidate the entire 82495XP tag array. It takes two clocks to look-up and invalidate a tag entry. The 82495XP will also invalidate tags in the CPU cache by running back-invalidation cycles. If the 82495XP tag state is modified, the 82495XP will run inquire cycles to the i860 XP CPU to see is the line is modified in its cache. If so, the i860 XP CPU will write back the line into the 82495XP write buffer. All modified 82495XP cache lines must be written to memory.

7.18.2 WHEN SAMPLED

FLUSH# can be asserted at any time. The 82495XP will complete all outstanding transactions on the CPU and memory bus before beginning the FLUSH# process. The memory bus controller does not have to prevent FLUSH# during locked cycles because the 82495XP will complete its locked transaction before the FLUSH# process will begin.

Once a FLUSH# operation has begun, the FLUSH# signal is ignored until the operation completes. If RESET is activated while the FLUSH# operation is in progress, the FLUSH# operation will be aborted and the RESET immediately executed.

FLUSH# is an asynchronous input. FLUSH# must have a pulse width of 2 CLK's in order to guarantee 82495XP recognition.

7.18.3 RELATION TO OTHER SIGNALS

To initiate a FLUSH#, the 82495XP will complete all pending cycles and prohibit the processor from issuing any further ADS#'s while the FLUSH# is in progress. The FSIOUT# output signal is used to indicate the start and end of the FLUSH# operation. It will become active when the FLUSH# signal is internally recognized (all outstanding cycles have completed) and will de-activate with the CRDY# of the last FLUSH# write-back.

The memory bus controller supplies BRDY# to the CPU once FSIOUT# has gone inactive and the FLUSH is complete. Once FLUSH# has begun, and FSIOUT# active, all CADS#'s and CRDY#'s correspond to write-backs caused by the FLUSH# operation.

The 82495XP can be snooped during FLUSH# cycles and the snooping protocols will be the same as that for any memory bus cycle.

7.19 FPFLD# [FPFLDEN]

External FIFO PFLD

Indicates PFLD cycle during external PFLD FIFO mode

Output of the 82495XP (J4) Cycle Control Signal

Sync to CLK

7.19.1 SIGNAL DESCRIPTION

During RESET, this pin functions as the FPFLDEN configuration signal. The 82495XP can be configured to decode the i860 XP microprocessor's PFLD cycles. The 82495XP supports 3 operational modes for PFLD cycle decoding, as defined by FPFLDEN and NCPFLD#:

Mode #1. PFLD cycles are cached in the 82495XP.

Mode #2. PFLD cycles are not cached in the 82495XP, without an external PFLD extension FIFO.

Mode #3. PFLD cycles not cached in the 82495XP, with an external PFLD extension FIFO.

Mode	FPFLDEN	NCPFLD#
1	0	1
2	0	0
3	1	1
Illegal Mode	1	0

If mode 3 has been selected, the 82495XP allows the PFLD pipeline to be extended with an external FIFO. After RESET, when this mode has been selected, the FPFLD output will indicate that the requested cycle is a PFLD cycle. See Section 5.2.5 for more details.

7.19.2 WHEN DRIVEN

FPFLDEN is sampled on RESET as in figure 7-1, with a setup time of 4 CPU clocks. In PFLD mode #3, the FPFLD# output is valid in the same CLK as CADS# and remains valid until CRDY# or CNA#.

7.19.3 RELATION TO OTHER SIGNALS

Address and cycle specification signals (MSET0–MSET10, MTAG0–MTAG11, MCFA0–MCFA6, CW/R#, CM/IO#, CD/C#, RDYSRC, MCACHE#, NENE#, SMLN#, KLOCK#, and CPLOCK#) will be valid with CADS#.

7.20 FSIOUT#

Flush, Sync, Initialization Output

Indicates the start and end of the Flush,

Sync, and Initialization operations.

Output of the 82495XP (D1) Cache Synchronization Signal

Sync to CLK

7.20.1 SIGNAL DESCRIPTION

This signal indicates the start and the end of either a Flush, Sync, or Initialization (including self-test if requested) operation. These operations are mutually exclusive. This signal is activated when the 82495XP begins the operation and goes inactive upon completion of the operation.

7.20.2 WHEN DRIVEN

This signal will be asserted whenever a Flush, Sync, or Initialization operation is internally recognized by the 82495XP and is in progress.

7.20.3 RELATION TO OTHER SIGNALS

FSIOUT# active indicates that either Flush, Sync, or Initialization operation is in progress. Only one of these operations can be run within the 82495XP at a time.

The table below shows the priorities of these three operations:

Operation	Trigger	Priority
Initialization	RESET	Highest
Flush	FLUSH#	
Sync	SYNC#	Lowest

If a trigger of higher priority occurs while a lower priority operation is running, the lower priority operation is aborted and the higher priority one executed. If a trigger of lower priority occurs when a higher priority one is running, the lower priority trigger is ignored. Once a FLUSH# or SYNC# operation has begun, its trigger is ignored until the operation completes.

When a higher priority operation aborts a lower priority one, FSIOUT# remains active.

Since RESET, FLUSH# and SYNC# are all asynchronous, FSIOUT# will be activated when the 82495XP is actually internally executing the operation.

7.21 HIGHZ#

High Impedance Outputs

Causes 82495XP outputs to be tristated

Input to 82495XP (pin P4) Test Signal

Synchronous to CLK

7.21.1 SIGNAL DESCRIPTION

The 82495XP will enter self-test if both SLFTST# is active and HIGHZ# is inactive during reset. If SLFTST# is sampled active and HIGHZ# is sampled active during reset, the 82495XP floats all its outputs until the 82495XP is reset again. Activation of HIGHZ# without SLFTST# does nothing.

7.21.2 WHEN SAMPLED

HIGHZ# is sampled like figure 7-1 with a setup time of 10 CPU clocks. HIGHZ# is then a don't care until the 82495XP reset sequence is complete (with FSIOUT# going inactive) where it becomes the MBALE pin.

7.21.3 RELATION TO OTHER SIGNALS

HIGHZ# shares a pin with MBALE. 82495XP outputs are tristated if both HIGHZ# and SLFTST# are sampled active during reset.

7.22 KLOCK#

82495XP LOCK#

Request to MBC of LOCKed cycle

Output from 82495XP (pin C3) Cycle Control Signal

Synchronous to CLK

7.22.1 SIGNAL DESCRIPTION

KLOCK# indicates to the MBC that there is a request to execute a locked cycle. This signal follows the CPU lock request.

KLOCK# is simply a one-clock flow-through version of the CPU LOCK# signal. The 82495XP will activate KLOCK# with CADS# of the first cycle of a LOCKed operation and it will remain active until the CADS# of the last cycle of the LOCKed operation.

Note that if the memory bus is pipelined, there may be a situation in which KLOCK# deactivation is in the same CLK as its new activation (together with CADS#). In this case KLOCK# won't go inactive between back-to-back locked sequences. KLOCK# will never go inactive if the CPU LOCK# does not go inactive. The 82495XP will not open arbitration windows between back-to-back locked sequences; it is the memory bus controller's responsibility to implement this functionality by detecting a LOCKed write followed by a LOCKed read.

KLOCK# activation is not qualified by the tag array look-up (hit/miss indications); therefore, KLOCK# can be active before CADS# is asserted.

7.22.2 WHEN DRIVEN

KLOCK# assertion is a flow-through of 1 CLK from the CPU LOCK# after the 82495XP completes all pending cycles. KLOCK# deassertion is a flow-through of 1 CLK from the CPU LOCK# signal, and must be at least 1 CLK after the last CADS# of a LOCKed sequence. KLOCK# is always driven to a valid logic level.

7.22.3 RELATION TO OTHER SIGNALS

Address and cycle specification signals (MSET0–MSET10, MTAG0–MTAG11, MCFA0–MCFA6, CW/R#, CM/IO#, CD/C#, RDYSRC, MCACHE#, NENE#, SMLN#, KLOCK#, and CPLOCK#) will be valid with CADS#.

7.23 KWEND#

Cacheability Window End

Closes 82495XP Cacheability Window

Input to 82495XP (pin M4) Cycle Progress Signal

Synchronous to CLK

7.23.1 SIGNAL DESCRIPTION

KWEND# is a cycle progress input to the 82495XP that, when active, closes the cacheability window and causes the cacheability attributes MKEN# and MRO# to be sampled.

KWEND# is sampled by the 82495XP after BGT# has been sampled active. KWEND# should be asserted by the MBC once the memory address has been decoded and cacheability (MKEN#) and read-only (MRO#) attributes have been determined.

The sampling of KWEND# active allows SWEND# to be sampled. Resolving KWEND# quickly allows the non-cacheable window between BGT# and SWEND# to be closed more quickly. KWEND# activation also allows the 82495XP to start allocations and begin replacements.

7.23.2 WHEN SAMPLED

KWEND# is sampled by the 82495XP on the clock, or after, BGT# has been sampled active. Once KWEND# is sampled active it is not sampled again until BGT# of the next cycle. KWEND# need not follow setup and hold times if it is not being sampled.

BGT#, KWEND# and SWEND# may be asserted on the same clock edge.

KWEND# need only be activated for those cycles which require the sampling of MKEN# and MRO#. These are line-fills and write cycles with potential allocation.

7.23.3 RELATION TO OTHER SIGNALS

KWEND# is sampled on or after BGT# and allows the sampling of SWEND#. KWEND# activation causes the sampling of MKEN# and MRO#.

According to cycle progress implication rules, CRDY# must be at least one clock after KWEND# for line fills and write-through cycles with potential allocate.

KWEND# shares a pin with CFG2.

7.24 MALE

Memory Address Latch Enable

Tristates/Enables Memory Address Outputs

Input to 82495XP (pin O2) Cycle Control Signal

Asynchronous

7.24.1 SIGNAL DESCRIPTION

The 82495XP contains an address latch which controls the last stage of the 82495XP address output. It is controlled by four signals: MAOE#, MBAOE#, MALE, and MBALE. The signals MALE and MBALE control the latching of the entire 82495XP address where MBALE controls the subline portion and MALE controls the rest.

MALE is provided so that the memory bus controller can control when the next pipelined address is driven. With MALE high, the 82495XP address latch is in 'flow-through' mode and the 82495XP address is available at the memory bus. Changes in the 82495XP address are seen immediately at the memory bus. When MALE is driven low the address at the latch input is latched. Any subsequent address driven by the 82495XP will not be seen at the memory bus outputs until MALE is driven high again.

MALE will latch 82495XP addresses regardless of the state of MAOE#. If MAOE# is inactive, MALE will still operate the latch properly, but the memory bus will be tristated.

7.24.2 WHEN SAMPLED

MALE is asynchronous and can be asserted and deasserted at any time. MALE should always be driven to a valid state since it directly controls the operation of the address latch.

7.24.3 RELATION TO OTHER SIGNALS

MALE together with MBALE control the latching of the entire 82495XP output address. The other latch control signals, MAOE# and MBAOE#, provide the memory bus controller complete command over the address outputs. MAOE# and MBAOE# do not affect the operation of MALE or MBALE.

MALE shares a pin with the WWOR# configuration pin.

7.25 MAOE#

Memory Address Output Enable
Tristates/Enables Memory Address Outputs
Input to 82495XP (pin S4) Cycle Control Signal
Asynchronous except during snoop cycles

7.25.1 SIGNAL DESCRIPTION

The 82495XP has an address latch which is controlled by a latch input, MALE, and an output enable input, MAOE#. MAOE# has two main functions. One, driving MAOE# active will enable the 82495XP to drive it's address lines MTAG0–11, MSET0–10, and MCFA0–6. Two, MAOE# is a qualifier for snoop cycles and must be inactive for the 82495XP to snoop.

In general, MAOE# should be active if its 82495XP is the current bus master. When that 82495XP gives up the bus, MAOE# should be inactive to float the address lines and allow another master to snoop.

MAOE# controls the output of the 82495XP address except the subline (burst) portion. This portion has a separate output control: MBAOE#.

7.25.2 WHEN SAMPLED

MAOE# is an asynchronous input (except during snoop cycles) and always has full control over the address output. For this reason, MAOE# must always be driven to a valid state.

The 82495XP does, however, sample MAOE# during snoop cycles. When sampled, MAOE# must meet proper setup and hold times. In synchronous snoop mode MAOE# is sampled on a CLK edge. In clocked mode MAOE# is sampled on a SNPCLK edge. In strobed mode MAOE# is sampled with the falling edge of SNPSTB#. If MAOE# is sampled active, the snoop will be ignored. This allows SNPSTB# to share a common line for multiple 82495XPs.

MAOE# need not meet any setup or hold time if it is not being sampled during a snoop cycle.

7.25.3 RELATION TO OTHER SIGNALS

MAOE# together with MBAOE# control the entire 82495XP address. Both signals are asynchronous and thus need never be synchronized to any clock. Both signals are, however, sampled during snoop cycles and require proper setup and hold times in these situations.

MALE and MAOE# together provide full control over the 82495XP address output latch.

7.26 MBALE

Memory Burst Address Latch Enable
Tristates/Enables Memory Burst Address Outputs
Input to 82495XP (pin P4) Cycle Control Signal
Asynchronous

7.26.1 SIGNAL DESCRIPTION

The 82495XP address latch is controlled by four signals: MAOE#, MBAOE#, MALE, and MBALE. The signals MALE and MBALE control the latching of the entire 82495XP address where MBALE controls the subline portion and MALE controls the rest.

2

MALE and MBALE are provided so that the memory bus controller has complete flexibility when the next address is driven. With MBALE high, the subline portion of the 82495XP address latch is in "flow-through" mode and the 82495XP subline address is available at the memory bus. Changes in the 82495XP subline address are seen immediately at the memory bus. When MBALE is driven low the subline address at the latch input is latched. Any subsequent subline address driven by the 82495XP will not be seen at the memory bus outputs until MBALE is driven high again.

MBALE will latch 82495XP addresses regardless of the state of MAOE# or MBAOE#. If MBAOE# is inactive, MBALE will still operate the latch properly, but the subline portion of the memory bus will be tristated.

Separate line and subline address latch controls are provided so that the latch outputs may be driven at different times. The table below indicates the subline address bits for each line size.

Line Size (Bytes)	Subline Address
32	A3, A4
64	A4, A5
128	A5, A6

7.26.2 WHEN SAMPLED

MBALE is asynchronous and can be asserted and deasserted at any time. MBALE should always be driven to a valid state since it directly controls the operation of the address latch.

7.26.3 RELATION TO OTHER SIGNALS

MALE together with MBALE control the latching of the entire 82495XP output address. The other latch control signals, MAOE# and MBAOE#, provide the memory bus controller complete command over the address outputs. MAOE# and MBAOE# do not affect the operation of MALE or MBALE.

MBALE shares a pin with the HIGHZ# configuration pin.

7.27 MBAOE#

Memory Burst Address Output Enable

Tristates/Enables Memory Subline Address Outputs

Input to 82495XP (pin P6) Cycle Control Signal

Asynchronous except during snoop cycles

7.27.1 SIGNAL DESCRIPTION

The 82495XP address latch is controlled by four signals: MAOE#, MBAOE#, MALE, and MBALE. MAOE# and MBAOE# are the output enables of this latch for the entire 82495XP address. Specifically, MBAOE# controls the subline address portion and MAOE# controls the rest.

MBAOE# has two functions. One, it can tristate the subline portion of the address separately from the rest of the address. Since the 82495XP does not sequence through burst addresses, the memory system may wish to provide the burst count. This requires that the 82495XP address burst portion be tristated after the first transfer. The Subline Address table appears in Section 7.26, MBALE.

Two, MBAOE# is sampled during snoop cycles. If MBAOE# is sampled inactive, the snoop write back cycle, if any, will begin at the subline address provided. If MBAOE# is sampled active, the snoop write back will begin at subline address 0. This allows snoop write backs to begin at the snooped subline address and progress through the normal burst order.

7.27.2 WHEN SAMPLED

Like MAOE#, MBAOE# is asynchronous except during snoop cycles and can be asserted or deasserted at any time. Since MBAOE# has direct control over the address latch, it must always be driven to a valid state.

MBAOE# is ,however, sampled during snoop cycles. In synchronous snooping mode, MBAOE# must meet proper setup and hold times to CLK's rising edge. In clocked mode, MBAOE# must meet setup and hold times to SNPCLK's rising edge. In strobed mode, MBAOE# must meet setup and hold times to SNPSTB#'s falling edge.

If MBAOE# is not being sampled for a snoop, ie. SNPSTB# is not asserted, MBAOE# need not meet any setup or hold time.

7.27.3 RELATION TO OTHER SIGNALS

MAOE# and MBAOE# control the entire 82495XP address output asynchronously. This address latch is completely controlled by MALE, MBALE, MAOE#, and MBAOE#.

MBAOE# is only sampled by the 82495XP during snoop cycles with SNPSTB#.

7.28 MBRDY#

Memory Burst Ready

Burst Ready input to 82490XP memory buffers

Input to 82490XP (pin 22) Cycle Progress Signal

Synchronous to MCLK

7.28.1 SIGNAL DESCRIPTION

When in clocked memory bus mode, MBRDY# (with MSEL# active) is used to advance the memory burst counter for the 82490XP buffer in use. This causes either new data to be latched from the memory bus (read cycle), or new data to be driven from the 82490XP buffer (write cycle). MBRDY# is sampled on all MCLK edges in which MSEL# is sampled active and has no relation to CLK. In strobed mode, MBRDY# must be tied high as MISTB/MOSTB strobes data in/out of the 82490XP.

For write cycles, the first piece of write data is available at the MDATA pins. MBRDY# assertion with MSEL# active causes the next 32, 64, or 128-bit slice of write data to be available. If only one slice is required, MSEL# and MBRDY# need never go active.

For read cycles, the first piece of read data flows through to the CPU. MBRDY# assertion with MSEL# active causes the next slice of memory data to be latched in the 82490XP buffer. BRDY# assertion will allow this data to be available on the CPU bus and latch it into the CPU. For cacheable cycles, MBRDY# needs to be asserted 4 or 8 times depending on the cache configuration.

7.28.2 WHEN SAMPLED

MBRDY# is sampled on all MCLK edges where MSEL# is sampled active. In this way MSEL# qualifies the MBRDY# input. If MSEL# is sampled inactive, MBRDY# need not follow setup and hold times to MCLK.

7.28.3 RELATION TO OTHER SIGNALS

MBRDY# is qualified by the MSEL# input. MBRDY# advances the memory burst counter for the 82490XP in use which either inputs or outputs data through MDATA.

MEOC# switches the 82490XP buffers to the next pending cycle, so the last MBRDY# must come before or on the clock of MEOC# assertion.

7.29 MCACHE#

82495XP Internal Cacheability

Indicates cycle cacheability attribute

Output from 82495XP (pin C2) Cycle Control Signal

Synchronous to CLK

7.29.1 SIGNAL DESCRIPTION

MCACHE# is driven by the 82495XP and indicates that the current cycle may be cached. Data cacheability is determined later in the cycle by MKEN# assertion. MCACHE# is asserted for allocation, replacement write-back cycles, and during cacheable read-miss cycles. (ie. read-miss cycles in which PCD is not asserted). It is not asserted for IO, special, or locked cycles.

Cycle Type	MCACHE#
Posted Writes	1
Write Backs	0
Read, PCD = 0	0
Read, PCD = 1	1
Allocation	0
I/O Cycles	1
Locked Cycles	1

7.29.2 WHEN DRIVEN

MCACHE# is valid in the same CLK as CADS# and remains valid until CRDY# or CNA#.

7.29.3 RELATION TO OTHER SIGNALS

Address and cycle specification signals (MSET0–MSET10, MTAG0–MTAG11, MCFA0–MCFA6, CW/R#, CM/IO#, CD/C#, RDYSRC, MCACHE#, NENE#, SMLN#, KLOCK#, and CPLOCK#) will be valid with CADS#.

7.30 MCFA0–MCFA6 MSET0–MSET10 MTAG0–MTAG11

MCFA0–MCFA6 Memory Configuration Address I/O

MSET0–MSET10 Memory Set Address I/O

MTAG0–MTAG11 Memory Tag Address I/O

82495XP Memory Address Inputs/Outputs

Input/Output of 82495XP (pins N14, P7–P15, O6–O16, R4, R14–R17, S14–S17) Cycle Control Signals

Input Synchronous to CLK, SNPCLK, or SNPSTB#.

Output from CLK, MAOE# active or MALE high.

7.30.1 SIGNAL DESCRIPTION

MSET0–10, MTAG0–11, and MCFA0–6 provide the complete 30 bit address input/output interface of the 82495XP to the memory bus. Together they span the entire CPU address range A2–A31. Depending on the cache configuration, each pin represents a different CPU address line (see configuration section for details).

MSET0–10, MTAG0–11, and MCFA0–6 pass through a 82495XP output latch. The latching of this latch is controlled by MALE/MBALE, and the output of this latch is controlled by MAOE#/MBAOE#.

With MAOE#/MBAOE# active, MSET/MTAG/MCFA are 82495XP outputs. They are valid at the start of a memory bus cycle at the input of the 82495XP address latch. If MALE/MBALE is high (flow-through) and MAOE#/MBAOE# is active (outputs enabled), they are driven to the memory bus with CADS#.

If a new cycle starts and MALE/MBALE is low, the previous address remains valid at the 82495XP MSET/MTAG/MCFA outputs. Once MALE/MBALE goes high, the new address flows through with the appropriate propagation delay (MSET/MTAG/MCFA address valid delay from MALE/MBALE going high). The new address will be driven to the 82495XP MSET/MTAG/MCFA outputs if MAOE#/MBAOE# is active.

If a new cycle starts, MALE/MBALE is high, and MAOE#/MBAOE# is inactive, the 82495XP MSET/MTAG/MCFA outputs will remain tristated. Once MAOE#/MBAOE# is asserted, the new address flows through with the appropriate propagation delay (MSET/MTAG/MCFA address valid from MAOE#/MBAOE# going active).

MSET0–10, MTAG0–11, and MCFA0–6 are used as inputs to the 82495XP during snoop cycles. Here, MAOE#/MBAOE# is inactive. MSET/MTAG/MCFA are sampled by the 82495XP during snoop initiation just like the other snoop attributes.

7.30.2 WHEN SAMPLED

If MALE/MBALE is high and MAOE#/MBAOE# is low, MSET0–10, MTAG0–11, and MCFA0–6 are valid with CADS# with a timing reference to CLK. Otherwise, they are asserted with a delay from MALE/MBALE high or MAOE#/MBAOE# active.

MSET0–10, MTAG0–11, and MCFA0–6 change once CNA# or CRDY# is sampled active. MSET0–10, MTAG0–11, and MCFA0–6 have a float delay from MAOE#/MBAOE# going inactive. These outputs are undefined after CRDY#/CNA# assertion and before the next CADS# assertion.

As inputs during snoop cycles (SNPSTB# asserted), they must be sampled like other snoop attributes with proper setup and hold times. In synchronous snoop mode this is with respect to CLK; in clocked mode, this is with respect to SNPCLK; and in strobed mode this is with respect to SNPSTB# falling edge.

If MAOE# is inactive and SNPSTB# is not asserted (no snoop), MSET0–10, MTAG0–11, and MCFA0–6 need not meet any setup or hold time.

7.30.3 RELATION TO OTHER SIGNALS

MSET0–10, MTAG0–11, and MCFA0–6 are asserted with CADS# so they are valid when CADS# is sampled active. This is true as long as MALE/MBALE is high and MAOE#/MBAOE# is active. If MSET0–10, MTAG0–11, and MCFA0–6 have been asserted but are blocked by MALE/MBALE or MAOE#/MBAOE#, they are asserted from MALE/MBALE going high or MAOE#/MBAOE# going active.

MSET0–10, MTAG0–11, and MCFA0–6 are deasserted or changed with CADS# or CNA# active. They may also be floated with MAOE# going inactive.

MSET0–10, MTAG0–11, and MCFA0–6 are used as inputs during snoop cycles. They are sampled with SNPSTB# like any other snoop attribute signal.

7.31 MCLK

Memory Bus Clock

Input to the 82490XP (Pin 26)

7.31.1 SIGNAL DESCRIPTION

In a clocked memory bus mode, this pin provides the memory bus clock. Memory bus signals and memory bus data are sampled on the rising edge of MCLK. Memory bus write data is driven off MCLK or MOCLK depending upon the configuration. MCLK has no relation to CLK.

7.31.3 RELATION TO OTHER SIGNALS

MCLK shares a pin with MISTB.

In clocked memory bus mode, the MDATA7–MDATA0, MSEL#, MFRZ#, MBRDY#, MZBT#, and MEOC# pins are sampled synchronously with the rising edge of MCLK. In a clocked memory bus write, MDATA7–MDATA0 are driven synchronous with MCLK or MOCLK.

MOCLK is a delayed version of MCLK. If a clocked memory bus configuration is chosen, and the MOCLK rising edge is detected by the 82490XP after RESET, data will be driven off of MOCLK rather then MCLK. Only data is effected by MOCLK. MOCLK is used to allow the system designer to increase the minimum output time of MDATA relative to MCLK.

7.32 MDATA0–MDATA7

Memory Bus Data Pins

82490XP Connection to the Memory Bus

Input/Output of 82490XP (pins 18, 14, 10, 6, 16, 12, 8, 4)

Synchronous to CLK or MCLK or MOCLK or MISTB or MOSTB.

7.32.1 SIGNAL DESCRIPTION

MDATA0–7 is the 82490XP data bus connection to the memory bus. All or part of these pins will be used depending on the cache configuration. These pins

are directly controlled by the MDOE# input. With MDOE# inactive, these pins are tristated and may be used as inputs.

For write cycles, the 82495XP asserts CDTS# to indicate that data will be available at the MDATA pins or in its buffer. Data is output with respect to CLK, MCLK, MOCLK, or MEOC# and is strobed with MBRDY#. In strobed memory bus mode, data is output using MOSTB.

For read cycles, CDTS# indicates that the CPU data path will be available for read data in the next clock. BRDY# reads data into the CPU from the 82490XP. Data is read into the 82490XPs through MDATA using MBRDY# or MISTB.

7.32.2 WHEN DRIVEN

When the CPU or 82495XP initiates a write cycle, the write data is written to the appropriate 82490XP buffer and CDTS# is asserted. If MDOE# is active, that first piece of write data will be available at the MDATA pins with some delay from the CPU CLK edge that CDTS# is asserted. Subsequent pieces of write data are output with some delay from MCLK or MOCLK (mode dependent) from the edge that MBRDY# is sampled active. In strobed mode, subsequent data is output with MOSTB assertion.

MDATA has no value before CDTS# assertion, after MEOC# with no pending cycle, or with MDOE# inactive.

For read cycles, the 82495XP asserts CDTS# the clock before the MDATA path is available for read data. MDOE# must be inactive for the 82490XP to read data. Read data is strobed into the 82490XP by asserting MBRDY# on MCLK edges. MEOC# will latch the last piece data as it switches buffers. In strobed mode, data is read by MISTB. Data that is read into MDATA must meet proper setup and hold times.

Data at the MDATA inputs need not follow setup and hold times to MCLK edges that sample MBRDY# inactive.

7.32.3 RELATION TO OTHER SIGNALS

CDTS# indicates that write data is in the 82490XP buffers. If MDOE# is active, write data is available at MDATA some time after CDTS# or MEOC# is sampled active. Subsequent write data is available at MDATA after MBRDY# assertion or MOSTB changing.

MDOE# must be inactive for MDATA to read data. CDTS# assertion by the 82495XP indicates that the read path is available in the next clock. Data must be read into MDATA with respect to MCLK or MISTB and must follow proper setup and hold times if MBRDY# is active or MISTB is changing.

The memory bus controller must account for the large setup time required to read data into the CPU. If properly done, data can be read into MDATA by asserting MBRDY# and in the next full CPU clock read into the CPU using BRDY#.

7.33 MDOE#

Memory Data Output Enable

Tristates/Enables Memory Data Outputs

Input to 82490XP (pin 20) Cycle Control Signal

Asynchronous

7.33.1 SIGNAL DESCRIPTION

MDOE# is an input to the 82490XP that, when asserted, causes the 82490XP to drive its MDATA0-MDATA7 outputs. When MDOE# is inactive, these lines are floated and may be used as inputs to the 82490XP. MDOE# is not sampled by any clock and is a direct connection to the 82490XP memory ouput driver.

7.33.2 WHEN SAMPLED

Since MDOE# is a direct connection to the 82490XP memory output drivers, MDOE# must always be driven to a valid level. With MDOE# inactive, data in the 82490XP's may be driven to MDATA outputs with some propagation delay from MDOE# going active. Similarly, there is some float delay from MDOE# going inactive.

MDOE# must be inactive for the 82490XP to read memory data.

7.33.3 RELATION TO OTHER SIGNALS

MDOE# has no relation to MCLK, MOCLK, or MOSTB. Since MDOE# controls the final stage of the MDATA output buffers, it has no effect on any other signal of the 82490XP.

7.34 MEMLDRV

Memory Low Capacitance Drivers

Selects the Low Capacitance Drivers for the 82495XP and the 82490XP

Inputs to 82495XP and 82490XP (pins Q4, 24) Configuration Signal

Synchronous to CLK

7.34.1 SIGNAL DESCRIPTION

MEMLDRV is a pin on both the 82495XP and 82490XP that, when high during reset, select normal driving memory output buffers. If this pin is driven low at reset, the high capacitance drivers are selected. Specifically, these are the 82495XP address outputs to the memory bus, and the 82490XP MDATA outputs. The normal output drivers are designed to drive up to 50 pF loads. The high capacitance drivers can drive up to 100 pF without derating.

7.34.2 WHEN SAMPLED

MEMLDRV is sampled like figure 7-1 with a setup time of 4 CPU clocks for the 82495XP and 1 CPU clock for the 82490XP. On the 82495XP, MEMLDRV becomes the SYNC# input once FSIOUT# goes inactive. On the 82490XP, MEMLDRV becomes the MFRZ# signal which is sampled after the first memory cycle begins.

7.34.3 RELATION TO OTHER SIGNALS

MEMLDRV shares a pin with SYNC# on the 82495XP and MFRZ# on the 82490XP.

7.35 MEOC#

Memory End of Cycle

Ends a cycle in 82490XP by switching buffers

Input to 82490XP (pin 23) Cycle Control Signal

Synchronous to MCLK or Asynchronous (strobed mode)

7.35.1 SIGNAL DESCRIPTIONS

MEOC# is an input to the 82490XP that ends the current cycle and switches memory buffers for new cycle. Switching to the next cycle does not cause information to be lost in the memory or CPU buffers in the 82490XP, but rather switches new buffers to the memory I/O bus of the 82490XP.

MEOC# is provided so that the memory system, which is synchronous to MCLK, can switch to a new cycle without synchronization. In clocked memory bus mode MEOC# is sampled with the rising edge of MCLK. In strobed memory bus mode the MEOC# function is performed with rising or falling edges of MEOC#.

For read or write cycles, MEOC# may be activated on or after the clock edge of the last MBRDY# of the current cycle. If a cycle is pending (pipelining is used), the next cycle will flow-through with a propagation delay from MEOC# assertion. MEOC# is required for all memory bus cycles.

In addition to switching memory buffers, MEOC# does three other things. One, MEOC# activation causes the memory burst counter to be reset to its start value and if MSEL# is active, MZBT# is sampled. This allows MSEL# to stay active between cycles. Two, MEOC# activation during a write cycle causes MFRZ# to be sampled for the a subsequent allocation (line-fill). Three, MEOC# latches in the last slice of data (like MBRDY#) before switching buffers.

7.35.2 WHEN SAMPLED

In clocked memory bus mode, MEOC# is sampled on every MCLK edge. It must always observe setup and hold times to MCLK. In strobed memory bus mode, MEOC# is always sampled and must meet proper active/inactive times.

7.35.3 RELATION TO OTHER SIGNALS

MEOC# is provided so that a cycle may end on the memory bus before CRDY# can be asserted. The implication rules surrounding MEOC# are:

1. MEOC# $\leq$ CRDY#
2. MEOC# for cycle N+1 $\geq$ 2 clocks after CRDY# of cycle N
3. MEOC# for cycle N+1 $\geq$ 2 clocks after last BRDY# of cycle N
4. MEOC# $\geq$ BGT#

MEOC# active with MSEL# active causes the sampling of MZBT# and MFRZ#.

7.36 MFRZ#

Memory Data Freeze

Freezes Memory Write Data in 82490XP Buffer

Input to 82490XP (pin 24) Cycle Control Signal

Synchronous to MCLK or Strobed

7.36.1 SIGNAL DESCRIPTION

MFRZ# is an input to the 82490XP that when active causes the 82490XP to "freeze" write data in the 82490XP memory buffer and allow a subsequent allocation to fill a cache line around it. MFRZ# is pro-

vided so that an actual write to memory need not be done to perform an allocation. Using MFRZ# to perform this dummy write cycle requires that the memory bus controller put the allocated line into the "M" state.

PALLC# must be active and MKEN# must be returned active for the write cycle to be turned into an allocation. MFRZ# is sampled when MEOC# goes active at the end of the write cycle. The subsequent line fill is then filled around the write data to complete the allocation.

7.36.2 WHEN SAMPLED

In clocked memory bus mode, MFRZ# is sampled with the MCLK rising edge that MEOC# is sampled active for all CPU write cycles. MFRZ# need only follow a proper setup and hold time in this situation.

In strobed mode, MFRZ# is sampled with the falling edge of MEOC# for write cycles. MFRZ# need only follow a proper setup and hold time in this situation.

7.36.3 RELATION TO OTHER SIGNALS

MFRZ# is sampled with the MEOC# going active or being active for write cycles. MFRZ# is used so that a dummy write cycle can be performed. If an allocation is done, DRCTM# must be asserted during the SWEND# window of the line fill to put the allocated line in the "M" state.

MFRZ# shares a pin with the MEMLDRV configuration input.

7.37 MHITM#

Memory Bus Hit [M]

Indicates snoop hit to modified line

Output from 82495XP (pin H4) Snooping Signal

Sync to CLK

7.37.1 SIGNAL DESCRIPTION

The MHITM# output is driven by the 82495XP during a snoop cycle to indicate that the snooping address has hit a Modified line. If the signal is logic high, the snoop has not hit a modified line; if the signal is logic low, the snoop has hit a modified line. When a snoop hits a modified line, the 82495XP automatically schedules a write-back of the hit modified line to the memory bus.

When the device which controls the memory bus (the master) performs a memory access, a snoop is requested of all other caching devices on the bus (snoopers). An asserted MHITM# pin from any of the snooper 82495XPs alerts the master that main memory's data is stale, and that the bus must be temporarily given to the snooper which has its MHITM# asserted so that the modified line can be written out to the memory bus.

7.37.2 WHEN DRIVEN

The snoop lookup is performed in the clock in which SNPCYC# is asserted. The MHITM# result for the snoop is driven on the CLK following SNPCYC#, and remains valid until the next assertion of SNPSTB#. The MHITM# signal is not valid from SNPSTB# until the CLK after SNPCYC#.

7.37.3 RELATION TO OTHER SIGNALS

MHITM# and MTHIT# outputs together indicate the results of a snoop lookup in the 82495XP.

A 82495XP can accept a snoop request while performing memory bus transfers of its own. If a snoop is requested of a 82495XP while it is performing a data transfer of its own, the results of the snoop may be delayed. If SNPSTB# is sampled at a 82495XP after it has received BGT# for its own cycle, the snoop lookup is performed (SNPCYC# active) after the SWEND# of its own cycle, and MHITM# is driven with valid results one CLK after SNPCYC# (see Sections 6.2.4 and 6.2.5).

7.38 MISTB

Memory Bus Input Strobe

Strobes data into the 82490XP

Input to 82490XP (pin 22) Cycle Control Signal

Asynchronous

7.38.1 SIGNAL DESCRIPTION

MISTB is an input to the 82490XP that, on rising or falling edges, causes the 82490XP to latch its MDATA inputs. MISTB is used in strobed memory bus mode. In clocked memory bus mode, MISTB is the MBRDY# input.

7.38.2 WHEN SAMPLED

MISTB is always sampled by the 82490XP. MISTB must meet proper strobed mode active and inactive times.

7.38.3 RELATION TO OTHER SIGNALS

MISTB causes the latching of the 82490XP MDATA inputs in strobed mode. MISTB shares a pin with MBRDY#.

7.39 MKEN#

Memory Cache Enable

Determines 82495XP and CPU cacheability

Input to 82495XP (pin R1) Cycle Attribute Signal

Synchronous to CLK

7.39.1 SIGNAL DESCRIPTION

MKEN# is an input to the 82495XP that is sampled at the closing of the cacheability window (KWEND# is sampled active). The 82495XP drives KEN# back to the CPU one clock after sampling the value of MKEN#. MKEN# thus determines whether the current cycle is cacheable in the 82495XP and in the CPU.

For read cycles, if MCACHE# is active (cacheable), KEN# is driven out of the 82495XP to the CPU to indicate cacheability. If MKEN# is sampled inactive during KWEND# activation, KEN# is brought inactive by the 82495XP, and the line will not be cacheable by the CPU or 82495XP. If MCACHE# is inactive, the line will be non-cacheable regardless of MKEN#. PCD active will cause MCACHE# to be inactive.

MKEN# is sampled during write-through cycles that are potentially allocatable (PALLC# is active during the write cycle). If MKEN# is sampled active during KWEND# activation of the write cycle, an allocation will occur, and a line-fill will follow the write cycle. MKEN# during the line-fill is ignored. The MBC indicates to the 82495XP that it intends to perform an allocation by asserting MKEN#.

MKEN# must be sampled 1 clock before the first BRDY# assertion to make a line-fill non-cacheable to the CPU.

7.39.2 WHEN SAMPLED

MKEN# is sampled on the clock edge that KWEND# is first sampled active. In all other places MKEN# may violate setup and hold times.

7.39.3 RELATION TO OTHER SIGNALS

MKEN# and MRO# are sampled with KWEND# active. MKEN# must be sampled at least 2 clocks before BRDY# assertion to make a line-fill non-cacheable.

7.40 MOCLK

Memory Data Output Clock

Separate Clock Reference for Memory Data Output

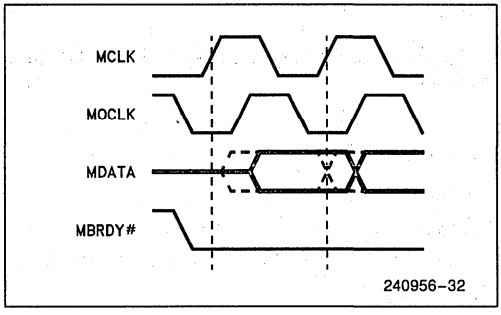
Input to 82490XP (pin 27)

Asynchronous

7.40.1 SIGNAL DESCRIPTION

MOCLK is the latch enable for the 82490XP memory data outputs (MDATA). MOCLK controls the latching of a transparent latch which, when high, causes MDATA to be driven from MCLK. When low, MDATA is latched. MOCLK may only be used in clocked memory bus mode and only affects output data. It is provided so that a greater MDATA output hold time can be generated.

To be used effectively, MOCLK must be a clock input that is skewed from MCLK. The following picture shows how MOCLK has increased the hold time of the output burst data:



7.40.2 WHEN SAMPLED

MOCLK is sampled during and after RESET to determine whether output data should be driven from MCLK or MOCLK. If toggling, MOCLK controls the MDATA outputs with MCLK. If high, data is driven from MCLK alone. Regardless, input data is never referenced to MOCLK.

In strobed memory bus mode the MOCLK signal becomes MOSTB. MOCLK is only used in clocked memory bus mode.

7.40.3 RELATION TO OTHER SIGNALS

To be used effectively, MOCLK must be the same frequency as MCLK but be skewed. This effectively increases MDATA hold time to main memory. Main memory must sample the data on MCLK edges.

MOCLK shares a pin with the MOSTB signal.

7.41 MOSTB

Memory Bus Output Strobe

Strobes data out of 82490XP

Input to 82490XP (pin 27) Cycle Control Signal

Asynchronous

7.41.1 SIGNAL DESCRIPTION

MOSTB is an input to the 82490XP that, on rising and falling edges, causes the 82490XP to output data through its MDATA outputs. MOSTB is only used in strobed memory bus mode. In clocked memory bus mode, MOSTB is the MOCLK input.

7.41.2 WHEN SAMPLED

MOSTB is always sampled by the 82490XP. MOSTB must meet strobed mode active and inactive times.

7.41.3 REALTION TO OTHER SIGNALS

MOSTB strobes data out of the 82490XP through MDATA. MOSTB shares a pin with MOCLK.

7.42 MRO#

Memory Read-Only

Designates current line as read-only

Input to 82495XP (pin J1) Cycle Attribute Signal

Synchronous to CLK

7.42.1 SIGNAL DESCRIPTION

MRO# is an input to the 82495XP that is sampled at the closing of the cacheability window (KWEND# activation). If sampled active, it causes the current line fill to the 82495XP to be put in the read-only state, and causes the line to be non-cacheable to the CPU. Writes to read-only lines in the 82495XP are treated as write-misses that are non-allocatable (PALLC# is inactive). MRO# is a bit in each 82495XP tag entry.

Once MRO# is sampled active during KWEND# activation, KEN# to the CPU is driven inactive regardless of the state of MKEN#. MKEN# does, however, determine whether the 82495XP will cache the read-only line. Once MRO# is returned active, the CPU will only require the number of transfers as indicated by LEN and CACHE#. If MKEN# is returned active, the 82495XP will require an entire cache line. 82495XP read-only cache lines are filled to the [S] state.

The line-fill portion of an allocation may be filled to the read-only state by returning MRO# active during KWEND# of the line-fill. MRO# is ignored during the write portion.

If MRO# is returned active during KWEND#, DRCTM# and MWB/WT# are ignored during SWEND#.

MRO# must be returned to the 82495XP at least 2 clocks before BRDY# is returned to the CPU so KEN# can be sampled properly.

There is one Read-Only bit per tag in the 82495XP.

7.42.2 WHEN SAMPLED

MRO# is sampled on the first clock that KWEND# is sampled active. In all other clocks, MRO# need not follow setup and hold times.

7.42.3 RELATION TO OTHER SIGNALS

MRO# and MKEN# are sampled with KWEND# activation. MRO# must be returned at least 2 clocks prior to the first BRDY#.

7.43 MSEL#

Memory Buffer Chip Select

Selects 82490XP, Causes Sampling of MZBT#

Input to 82490XP (pin 25) Cycle Control Signal

Synchronous to MCLK or Strobed

7.43.1 SIGNAL DESCRIPTION

MSEL# is an input to the 82490XP that has 3 main functions. One, MSEL# active qualifies the MBRDY# input to the 82490XP. If MSEL# is inactive for a particular 82490XP, MBRDY# will not be recognized by that 82490XP.

Two, MSEL# going active causes the sampling of MZBT# for the next transfer.

Three, MSEL# going inactive resets the 82490XP internal memory burst counter. The 82490XP contains a memory burst counter that counts through the CPU burst order with each MBRDY# assertion and increments a pointer to the 82490XP memory buffer being accessed.

MSEL# going inactive will reset this burst counter to its original burst value. By resetting this counter before MEOC# assertion, all information currently being read into the 82490XP is lost, but information that is being written out is maintained and may be rewritten.

In general, MSEL# may stay inactive for single transfer cycles such as posted 64-bit write cycles. Once active, MSEL# need not go inactive as the burst counter is reset with MEOC# activation. Since MZBT# may also be sampled with MEOC#, it is possible to leave MSEL# asserted throughout most basic transfers.

MSEL# or MEOC# must be used to reset the burst counter before any transfer begins. If transfers are interrupted (by a snoop hit before BGT# assertion for example), MSEL# must be brought inactive so the burst counter may be reset for the snoop write back.

MSEL# must be sampled inactive for at least 1 MCLK after reset. This resets the memory burst counter for the first transfer.

7.43.2 WHEN SAMPLED

In clocked memory bus mode, MSEL# is sampled with all rising edges of MCLK. In this mode, if MSEL# is sampled inactive, the memory burst counter is reset and MZBT# is sampled. If MSEL# is sampled active and MBRDY# is sampled active, the memory burst counter is incremented. Since it is constantly sampled with MCLK, MSEL# must always be driven to a known state and must always meet setup and hold times to every MCLK edge.

In strobed mode, MSEL# falling edge causes the sampling of MZBT#. While MSEL# is active, MISTB and MOSTB cause the memory burst counter to be incremented. The rising edge of MSEL# causes the memory burst counter to be reset.

MSEL# must be inactive sometime after RESET before the first transfer to initialize the burst counter.

7.43.3 RELATION TO OTHER SIGNALS

MSEL# causes the sampling of MZBT#, and qualifies the use of MBRDY#, MOSTB, and MISTB. Since MSEL# acts as a qualifier for these signals, MSEL# may be asserted at the same time as MBRDY#, MOSTB, or MISTB.

7.44 MTHIT#

Memory Bus Tag Hit

Indicates snoop hit

Output from 82495XP (pin G3) Snooping Signal

Sync to CLK

7.44.1 SIGNAL DESCRIPTION

The MTHIT# output is asserted by the 82495XP during snoop cycles to indicate that the snoop address has hit a line in the 82495XP cache. An asserted MTHIT# signal from any of the snooping 82495XP's alerts a bus master that the data being accessed resides in another cache. If SNPINV was not asserted on the snoop request, the copy of the data in a 82495XP asserting MTHIT# will remain valid and in the Shared state—so a caching master must also place his copy of the data in the Shared state.

7.44.2 WHEN DRIVEN

The snoop lookup is performed in the CLK in which SNPCYC# is asserted. The MTHIT# result for the snoop is driven on the next CLK and remains valid until the next assertion of SNPSTB#. The MTHIT# signal is not valid from SNPSTB# until the CLK after SNPCYC#.

7.44.3 RELATION TO OTHER SIGNALS

MTHIT# and MHITM# together indicate the results of a snoop lookup in the 82495XP.

An 82495XP can accept a snoop request while performing memory bus transfers of its own. If a snoop is requested while it is performing a transfer of its own, the results of the snoop may be delayed. If SNPSTB# is sampled at a 82495XP after it has received BGT# for its own cycle, the snoop lookup is performed (SNPCYC# active) after the SWEND# of its own cycle, and MTHIT# is driven with the valid result one CLK after SNPCYC# (see Sections 6.2.4 and 6.2.5).

Because an asserted MTHIT# from any snooping 82495XP requires the master to place the fetched line in the Shared state (unless it is an invalidating snoop), the memory bus controller should include the MTHIT# signals of other processors when generating the MWB/WT# signal to its own 82495XP.

7.45 MWB/WT#

Memory Write-back/Write-through

Forces lines to be filled to the [S] state

Input to 82495XP (pin K3) Cycle Attribute Signal

Synchronous to CLK

7.45.1 SIGNAL DESCRIPTION

MWB/WT# is an input to the 82495XP that is sampled at the closing of the snoop window (SWEND# activation). If sampled active, the current line-fill is filled to the [S] state in the 82495XP. The [S] state is a write-through state in the 82495XP.

MWB/WT# is used in many cases. If a cache to cache transfer updates memory and leaves the data valid in the other cache, the line must be filled to the [S] state instead of the [E] state default. A portion of memory may be designated as write-through by asserting MWB/WT# for appropriate addresses.

MWB/WT# has no effect on the 82495XP if DRCTM# is sampled active or MRO# has been sampled active during KWEND#. If PWT is active, MWB/WT# has no effect and the line is filled to the [S] state.

7.45.2 WHEN SAMPLED

MWB/WT# is sampled on the first clock edge that SWEND# is sampled active. If MWB/WT# is not being sampled, it need not follow setup and hold times.

7.45.3 RELATION TO OTHER SIGNALS

Both MWB/WT# and DRCTM# are sampled with SWEND#.

7.46 MX4/MX8# MTR4/MTR8#

Memory 4/8 I/O bits

Memory 4/8 Transfers

Selects MDATA Input/Output width and number of memory bus transfers

Inputs to 82490XP (pins 21, 25) Configuration Signals

Synchronous to CLK

7.46.1 SIGNAL DESCRIPTION

MX4/MX8# configures the 82490XP to use MDATA[0:3] or MDATA[0:7] memory bus I/O pins. MTR4/MTR8# selects whether the a cache line will take 4 or 8 transfers. These selections depend on the line ratio (82495XP line size / CPU line size) and must be configured according to the following table:

Line Ratio	MX4/ MX8#	MTR4/ MTR8 #	Membus I/O Pins	CPUbus I/O Pins
1	1	1	4	4
2	1	0	4	4
2	0	1	8	4
4	0	0	8	4
1	0	1	8	8
2	0	0	8	8

7.46.2 WHEN SAMPLED

These signals are sampled like Figure 7-1 with a setup time of 1 clock. Once the first CADS# is issued by the 82495XP these signals are sampled for the MZBT# and MSEL# functions.

7.46.3 RELATION TO OTHER SIGNALS

MX4/MX8# shares a pin with MZBT# and MTR4/MTR8# shares a pin with MSEL#.

7.47 MZBT#

Memory Zero Base Transfer

Forces cycles to begin at subline address 0

Input to 82490XP (pin 21) Cycle Control Signal

Synchronous to MCLK or Strobed

7.47.1 SIGNAL DESCRIPTION

MZBT# is an input to the 82490XP that forces a read or write cycle to begin with burst address 0 regardless of the CPU generated address.

MZBT# is sampled before the transfer begins. MZBT# is sampled with MSEL# and MEOC#. MZBT# is sampled with MSEL# going active for the current cycle. If MSEL# stays active between cycles, MZBT# is sampled with MEOC# going active for the previous cycle.

Once sampled, data input to the 82490XP's will start at burst address 0 and continue through 4, 8, C, etc. If the CPU is requesting a burst location other than 0, the memory bus controller must hold off any BRDY# until that bursted item is read from the memory bus.

7.47.2 WHEN SAMPLED

In clocked mode, MZBT# is sampled in two locations. First, MZBT# is sampled on all MCLK rising edges where MSEL# is sampled inactive. Once MSEL# is sampled active, the value of MZBT# that was sampled one MCLK before is used for the next transfer.

Second, MZBT# is sampled on MCLK rising edges where MEOC# is sampled active with MSEL# active. The MZBT# value sampled will be used for the next transfer. This allows MSEL# to stay asserted between transfers if so desired.

In strobed mode, MZBT# is sampled with the same two signals. First, it is sampled with the falling edge of MSEL#. Second, it is sampled with the falling edge of MEOC# if MSEL# is active.

In clocked memory bus mode MZBT# must follow setup and hold times to all MCLK edges where MSEL# is sampled inactive or MEOC# is sampled active with MSEL# active.

In strobed memory bus mode MZBT# must meet setup and hold times to MSEL# falling edge and MEOC# falling edge if MSEL# is active.

7.47.3 RELATION TO OTHER SIGNALS

MZBT# is sampled with MSEL# and MEOC# and has no effect otherwise. In systems that will never force a zero-based transfer, MZBT# may be driven high after RESET.

MZBT# shares a pin with the MX4/MX8# configuration input.

7.48 NCPFLD#

Non-Cacheable PFLD

Enables Non-Cacheable Floating Point Loads

Input to 82495XP (N4) Configuration Signal

Asychronous

7.48.1 SIGNAL DESCRIPTION

During RESET, this pin functions as the NCPLFD# configuration signal. The 82495XP can be configured to decode i860 XP CPU PFLD (Pipelined Floating Point Load) cycles. The 82495XP supports 3 operational modes for PFLD cycle decoding as defined by FPFLDEN and NCPFLD#:

Mode #1. PFLD cycles that are cached in the 82495XP.

Mode #2. PFLD cycles not cached in the 82495XP, without an external PFLD extension FIFO.

Mode #3. PFLD cycles not cached in the 82495XP, with an external PFLD extension FIFO.

Mode #	FPFLDEN	NCPFLD#
1	0	1
2	0	0
3	1	1
Illegal Mode	1	0

See Section 5.2.5 for details.

7.48.2 CASES IT IS ASSERTED AND DEASSERTED

NCPFLD# is sampled on the falling edge of RESET and is a don't care at any other time. NCPFLD# must be valid for at least 10 CLK's before RESET's falling edge.

7.48.3 RELATION TO OTHER SIGNALS

NCPFLD# shares a pin with FLUSH#. Both NCPFLD# and FPFLDEN describe the PFLD mode used.

7.49 NENE#

Next Near

Indicates current cycle address is near previous one.

Output from 82495XP (pin D5) Cycle Control Signal

Synchronous to CLK

7.49.1 SIGNAL DESCRIPTION

NENE# indicates to the MBC that the address of the requested memory cycle is "near" the address of the previously generated one (in the same 2K DRAM page). This information may be used by the MBC to optimize access to paged or static column DRAMs.

7.49.2 WHEN DRIVEN

NENE# is valid together with CADS# and will stay valid until CNA# or CRDY#.

7.49.3 RELATION TO OTHER SIGNALS

Address and cycle specification signals (MSET0–MSET10, MTAG0–MTAG11, MCFA0–MCFA6, CW/R#, CM/IO#, CD/C#, RDYSRC, MCACHE#, NENE#, SMLN#, KLOCK#, and CPLOCK#) will be valid with CADS#.

NENE# may change state after CNA# or CRDY# are asserted to the 82495XP.

7.50 PALLC#

Potential Allocate

Indicates 82495XP intent to allocate current cycle

Output from 82495XP (pin D2) Cycle Control Signal

Synchronous to CLK

7.50.1 SIGNAL DESCRIPTION

PALLC# indicates to the MBC that the current write cycle may allocate (perform a line-fill on) a cache line. The MBC chooses to perform an allocation by asserting MKEN# during KWEND# of the write cycle. Potential allocate cycles are cycles which are 82495XP misses with PCD and PWT inactive.

The exact condition for assertion of PALLC# is:

Miss * !PCD * !PWT * LOCK# * W/R# * D/C# * M/IO#

PALLC# is inactive (HIGH) for any write-hit to a Read-Only line.

7.50.2 WHEN DRIVEN

PALLC# is valid in the same CLK as CADS# and is valid until CRDY# or CNA#.

7.50.3 RELATION TO OTHER SIGNALS

PALLC# is valid with CADS#.

7.51 PAR#

Parity Selection

Selects 82490XP as a Parity Device

Input to 82490XP (pin 32) Configuration Signal

Synchronous to CLK

7.51.1 SIGNAL DESCRIPTION

PAR# is a strapping option on the 82490XP that, when strapped low, configures that 82490XP device to be a dedicated parity device. A 82490XP parity device must be configured the same as all the other devices, however, the data lines are defined differently. CDATA[0:3] are 4 parity bit I/O lines and CDATA[4:7] are 4 bit select lines so each parity line may be written individually. Parity devices must be used as follows:

Cache Size	Memory Bus Width	Number of Parity Devices	82490XP I/O Bits (CPU:Mem)
256K	64	2	4:4
512K	128	2	4:8

7.51.2 WHEN SAMPLED

PAR# is a strapping option and must be tied either high or low.

7.51.3 RELATION TO OTHER SIGNALS

PAR# affects the definition of the CDATA and MDATA lines of the 82490XP.

7.52 RDYSRC

Ready Source

Cycle control signal to the MBC

Output from 82495XP (pin C1) Cycle Control Signal

Synchronous to CLK

7.52.1 SIGNAL DESCRIPTION

RDYSRC serves as a cycle control signal to the MBC. It indicates the source of the BRDY# generation (either 82495XP or MBC) for the CPU. When high it indicates that the MBC should generate the BRDY#s to the CPU, when low it indicates that the 82495XP will provide the BRDY#s.

RDYSRC is asserted for line-fill and not asserted for the write portion of allocation cycles.

7.52.2 WHEN DRIVEN

RDYSRC is valid in the same CLK as CADS# and is valid until CRDY# or CNA#.

7.52.3 RELATION TO OTHER SIGNALS

Address and cycle specification signals (MSET0–MSET10, MTAG0–MTAG11, MCFA0–MCFA6, CW/R#, CM/IO#, CD/C#, RDYSRC, MCACHE#, NENE#, SMLN#, KLOCK#, and CPLOCK#) will be valid with CADS#.

7.53 RESET

Reset

Forces the 82495XP to begin execution in a known state

Input to 82495XP (Q5)

Asynchronous

7.53.1 SIGNAL DESCRIPTION

The falling edge of this signal tells the 82495XP to sample all configuration inputs and initializes the 82495XP to a known state. See the specific configuration signals for setup and hold times relative to RESET's falling edge. RESET can be asserted at any time.

During initlialization, the 82495XP LRU bits are set to 1 indicating that the 82495XP LRU way is way 1. The 82490XP MRU bits are initlialized to 0 as are all tag array bits.

RESET takes about 4100 clocks in the 82495XP. RESET with self-test takes about 80,000 clocks.

7.53.2 WHEN SAMPLED

RESET is an asynchronous input. RESET must have a pulse width of at least 8 CLK's in order to guarantee 82495XP recognition.

7.53.3 RELATION TO OTHER SIGNALS

The following signals are sampled at RESET:

CNA# [CFG0]:	CFG0 line of 82495XP configuration inputs
SWEND# [CFG1]:	CFG1 line of 82495XP configuration inputs
KWEND# [CFG2]:	CFG2 line of 82495XP configuration inputs
FLUSH# [NCPFLD#]:	If low, enables decoding of i860XL non- cacheable PFLD mode.
FPFLD# [FPFLDEN]:	If high, enables the external FIFO for i860XL PFLD mode.
BGT# [C490LDRV]:	Indicates the driving strength of the 82495XP/82490XP interface.
SYNC# [MEMLDRV]:	Indicates the memory bus driving strength.
SNPCLK# [SNPMD]:	Indicates the snooping mode; synchronous or strobed.
CFG2-CFG0	Configure cache parameters such as lines/sector, line ratio, and number of tags.

7.54 SLFTST#

Self Test

Executes 82495XP self-test

Input to 82495XP (pin M2) Test Signal

Synchronous to CLK

7.54.1 SIGNAL DESCRIPTION

If SLFTST# is sampled low and HIGHZ# is sampled high, the 82495XP will perform a self-test after reset. The results of the self-tests are given by CAHOLD when FSIOUT# goes inactive.

7.54.2 WHEN SAMPLED

SLFTST# is sampled with reset like figure 7-1 with a setup time of 10 CPU clocks. SLFTST# is then a "don't care" until after the first CADS# activation when it becomes the CRDY# pin.

7.54.3 RELATION TO OTHER SIGNALS

SLFTST# shares a pin with CRDY#. The 82495XP enters self-test if both SLFTST# is sampled active and HIGHZ# is sampled inactive.

7.55 SMLN#

Same Line

Current cycle is same 82495XP line as previous one.

Output from 82495XP (pin C6) Cycle Control Signal

Synchronous to CLK

7.55.1 SIGNAL DESCRIPTION

SMLN# is used to indicate to the MBC that the current cycle is accessing the same 82495XP cache line as the previous cycle. This indication can be used by the MBC to selectively activate its SNPSTB# signal to other caches in the system. For example, back-to-back snoop hits to the same line may be snooped only once.

7.55.2 WHEN DRIVEN

SMLN# is asserted with CADS# and will stay valid until CNA# or CRDY#.

7.55.3 RELATION TO OTHER SIGNALS

Address and cycle specification signals (MSET0–MSET10, MTAG0–MTAG11, MCFA0–MCFA6, CW/R#, CM/IO#, CD/C#, RDYSRC, MCACHE#, NENE#, **SMLN**#, KLOCK#, and CPLOCK#) will be valid with CADS#.

7.56 SNPADS#

Cache Snoop Address Strobe

Initiates a snoop write back cycle

Output from 82495XP (pin F3) Snooping Signal

Sync to CLK

7.56.1 SIGNAL DESCRIPTION

The SNPADS# signal indicates valid cache control and attribute signals, functioning identically to CADS#, but is generated only on snoop write-backs. The separation of address status signals for normal and snoop write-back cycles eases memory bus controller implementation. When SNPADS# is activated, the memory bus controller should abort all pending cycles for which BGT# has not been issued. The 82495XP reissues these non-committed cycles after the snoop write-back has completed.

7.56.2 WHEN DRIVEN

SNPADS# is produced when a snoop hits a modified line. A modified line condition exists when a line in the cache has been updated, and copies of that memory location in other devices are no longer valid. A snoop is initiated by the master of a shared bus when accessing a memory location on the shared bus.

The response of the 82495XP to a snoop appears on the MTHIT# and MHITM# pins in the clock after SNPCYC# is active. If these pins are both driven low, the snoop resulted in a hit to a modified line, and a snoop write-back is initiated with the assertion of SNPADS#. SNPADS# is driven, at earliest, two clocks after SNPCYC#. Like CADS#, SNPADS# is active for one CLK, and is always valid.

7.56.3 RELATION TO OTHER SIGNALS

Cycles initiated by SNPADS# require only CRDY#; they do not require the other cycle progress signals (BGT#, KWEND#, SWEND#).

The SNPADS# signal is driven by the 82495XP to indicate the start of the write-back cycle; the 82495XP drives the following address and cycle specification signals valid with SNPADS#: CW/R#, CD/C#, CM/IO#, MCACHE#, RDYSRC, NENE#, SMLN#, and the address on MSET[0:10], MTAG[0:11], and MCFA[0:6]. Upon assertion of SNPADS#, the memory bus controller should cancel all pending cycles for which BGT# has not yet been asserted, because they will be reissued after the snoop write-back. The 82495XP will ignore BGT# while SNPBSY# and MHITM# are active (ie, during the write-back).

The 82495XP can accept a snoop request while performing memory bus transfers of its own. If a snoop is requested while it is performing a transfer of its own, the results of the snoop and any necessary snoop write-backs may be delayed. If SNPSTB# is sampled at a 82495XP after it has received BGT# for its own cycle, and the snoop hits a modified line, the snoop write-back will occur after CRDY# for the 82495XP's own cycle. See Sections 6.2.4 and 6.2.5 for details.

7.57 SNPBSY#

Snoop Busy

Indicates additional snoop processing in progress

Output from 82495XP (pin F1) Snooping Signal

Sync to CLK

7.57.1 SIGNAL DESCRIPTION

SNPBSY# and SNPCYC# indicate a snoop in progress. The SNPCYC# signal is asserted on the actual snoop look-up to the 82495XP tags. If the snoop look-up indicates a valid line is hit and the snoop is invalidating, the 82495XP must perform a back invalidation on the CPU. If a snoop hit occurs to a modified line, a snoop write-back must occur. SNPBSY# is asserted and remains active while either a back invalidation or a snoop write-back is in progress.

7.57.2 WHEN DRIVEN

SNPBSY# is activated for two conditions. First, SNPBSY# is activated whenever a back invalidation is necessary: the snoop returns MTHIT# active and SNPINV was asserted on the snoop initiation. Second, SNPBSY# is activated when a modified cache line is hit on a snoop, as indicated by MHITM#, until the modified line has been written back (CRDY# returned for the write-back).

SNPBSY# is valid in the CLK following SNPCYC#, and if active, remains active for a minimum of two CLKS.

7.57.3 RELATION TO OTHER SIGNALS

After SNPCYC# occurs for a snoop, a new snoop may be initiated. If SNPBSY# is asserted for the initial snoop, the SNPCYC# of the second snoop is delayed until the SNPBSY# signal is deasserted for the initial snoop, indicating that its snoop processing has completed.

7.58 SNPCLK [SNPMD]

Snoop Clock [Snooping Mode]

Selects 82495XP snooping mode

Input to 82495XP (pin S3) Snooping Signal

Synchronous to CLK

7.58.1 SIGNAL DESCRIPTION

SNPMD selects whether the 82495XP snoop initiation be in synchronous, clocked, or strobed mode. 82495XP snoop response is always synchronous to CLK.

Synchronous mode (to CLK) is selected by SNPMD sampled low during reset. Strobed mode is selcted by SNPMD sampled high during reset. Clocked mode is selected by connecting the snoop clock source to SNPMD, and thus SNPMD becomes the actual snoop clock (SNPCLK).

7.58.2 WHEN SAMPLED

SNPMD is sampled like figure 7-1 with a setup time of 4 CPU clocks. SNPMD is then not used unless clocked mode is being selected. If clocked mode is selected, SNPMD becomes SNPCLK to clock in snoop requests.

7.58.3 RELATION TO OTHER SIGNALS

SNPMD becomes SNPCLK if a clock signal is detected at reset. In this clocked mode, SNPCLK is then used to clock-in SNPSTB#, the snoop address, and all snoop attributes.

7.59 SNPCYC#

Snoop Cycle

Indicates snoop look-up occurring in 82495XP tags

Output from 82495XP (pin H3) Snooping Signal

Sync to CLK

7.59.1 SIGNAL DESCRIPTION

SNPCYC# is asserted by the 82495XP during the clock when the actual tag look-up for the snoop is performed. SNPCYC# may appear as early as the CLK following SNPSTB# assertion, or may be delayed several clocks while a snoop write-back or 82495XP memory bus cycle take place.

7.59.2 WHEN DRIVEN

SNPCYC# is always a valid 82495XP output. It is asserted once, for a single clock, for every snoop which is initiated in the 82495XP.

7.59.3 RELATION TO OTHER SIGNALS

A snoop is initiated by assertion of the SNPSTB# input if MAOE# is not asserted. The actual snoop, signalled by the assertion of SNPCYC#, can be delayed by a prior snoop's write-back in progress (SNPBSY# asserted) or by a 82495XP memory cycle in progress (SNPSTB# occurs after BGT#)—see SNPSTB# for details. If neither of these is occurring, strobed and clocked snooping modes can also delay snoop look-up for a clock while the snoop address and attributes are synchronized.

In the clock following SNPCYC#, MHITM# and MTHIT# report valid snoop results.

7.60 SNPINV

Snoop Invalidation

Forces invalidation of snoop hits

Input to 82495XP (pin P5) Snooping Signal

Sampled with SNPSTB# (see SNPSTB#)

7.60.1 SIGNAL DESCRIPTION

Assertion of the SNPINV signal during the initiation of a snoop request forces a snoop hit for that request into the Invalid state.

The SNPINV pin is sampled upon initiation of a snoop request with SNPSTB# activation, depending on snooping mode: rising edge of first CLK when SNPSTB is asserted (synchronous snooping mode), or rising edge of first SNPCLK when SNPSTB# is asserted (clocked mode), or falling edge of strobed SNPSTB# (strobed mode).

7.60.2 WHEN SAMPLED

When a bus master performs a bus access, the SNPSTB# of all other 82495XPs is asserted to initiate a snoop for that address. If the master's access is one which is modifying the data (a write to memory, etc.), the SNPINV pin of all snooping 82495XPs must be asserted during SNPSTB# so that the line is properly marked Invalid.

SNPINV is not asserted during SNPSTB# assertion if snoop hits are to remain valid: the master issuing the snoop does not require their invalidation (a read).

SNPINV assertion forces all snoop hits to be invalidated, overriding other inputs or attributes (ie SNPNCA). When SNPINV is not asserted, cache states change according to normal protocol.

SNPINV is only sampled with SNPSTB#, which may be qualified by CLK or SNPCLK depending on the snooping mode, and must meet setup and hold times for the edge of its sampling. When SNPSTB# is not being asserted, SNPINV is a don't care and need not follow setup and hold times.

7.60.3 RELATION TO OTHER SIGNALS

SNPINV is sampled according to SNPSTB#, which may be qualified by SNPCLK or CLK, depending on the snooping mode. SNPINV overrides the SNPNCA input, which may also be asserted with SNPSTB#. If MAOE# is active with SNPSTB# sampling, the snoop request is ignored.

7.61 SNPNCA

Snoop Non Caching device Access

Indicates to snooping 82495XP that the initiating master is a non- caching device

Input to 82495XP (pin Q3) Snooping Signal

Sampled with SNPSTB# (see SNPSTB#)

7.61.1 SIGNAL DESCRIPTION

SNPNCA indicates that the master which is initiating the snoop request will not cache the data. If the SNPNCA pin is not asserted and the snoop is noninvalidating (where noninvalidating = SNPINV not asserted), a snoop hit line must be placed in the Shared state, since the data will exist in another

cache. If SNPNCA is asserted and the snoop is noninvalidating, a snoop hit line will not be entered into a new cache, so a hit Exclusive or Modified line will be placed in the Exclusive state by the 82495XP. A noninvalidating snoop hit to a Shared line must keep the hit line in the Shared state, regardless of SNPNCA.

SNPNCA is sampled upon initiation of a snoop request with SNPSTB# activation, depending on the snooping mode: rising edge of first CLK when SNPSTB# asserted (synchronous snooping mode), or the rising edge of SNPCLK when SNPSTB# is asserted (clocked snooping mode), or the falling edge of SNPSTB# (strobed snooping mode).

7.61.2 WHEN SAMPLED

To achieve maximum processor performance and minimum bus traffic, SNPNCA should be asserted when the noninvalidating snoop is caused by an access from a non-caching device like a DMA.

If the snoop is being caused by a device which will also be caching the data, SNPNCA must not be asserted, so that the 82495XP does not leave the hit line in an Exclusive state—subsequent writes to lines in this state do not appear on the bus, and stale data would result in the cache which incorrectly asserted SNPNCA.

If SNPNCA is asserted on a noninvalidating snoop request, the following outlines the behavior of the cache for a snoop hit in each of the MESI states:

Modified	The data is written to the bus, and the line is placed in the Exclusive state
Exclusive	The line remains in the Exclusive state
Shared	The line remains in the Shared state
Invalid	This is a cache miss. The line remains Invalid.

If SNPNCA is NOT asserted on a noninvalidating snoop request, an M, E, or S state hit line will be placed in the Shared state. Again, M state causes a write to the bus, Invalid lines remain Invalid.

SNPNCA is only sampled with SNPSTB#, which may be qualified by CLK or SNPCLK depending on the snooping mode, and must meet setup and hold times for the edge of this sampling. When SNPSTB# is not being sampled, SNPNCA is a don't care and need not follow set-up and hold times.

7.61.3 RELATION TO OTHER SIGNALS

SNPNCA is sampled with SNPSTB#, which may be qualified by SNPCLK or CLK, depending on snooping mode. The assertion of SNPINV overrides SNPNCA, and places all snoop hit lines into the Invalid state. If MAOE# is active on SNPSTB# sampling, the snoop request is ignored.

7.62 SNPSTB#

Snoop Strobe

Initiates 82495XP snoop and latches snoop address & attributes

Input to 82495XP (pin R3) Snooping Signal

Sync to CLK or SNPCLK, or strobed

7.62.1 SIGNAL DESCRIPTION

Snoop strobe initiates a 82495XP snoop request. It controls the latching of the snoop address and snoop attribute signals, in the manner specified by one of three snooping modes:

Snooping Modes

Mode	Snoop Address/ Attributes Sampled on:
Strobed	falling edge of SNPSTB#
Clocked	rising edge of SNPCLK when SNPSTB# sampled active
Synchronous	rising edge of CLK when SNPSTB# sampled

SNPSTB# must be asserted to initiate a snoop request. Snoops are initiated by a bus master for all memory accesses, to ensure that data residing in other caches is flushed if modified and invalidated if necessary.

SNPSTB# must be deasserted for at least one SNPCLK or CLK when clocked or synchronous snooping mode (respectively) is used, in order to rearm for the next snoop.

SNPSTB# can be asserted while a snoop is in progress, allowing one level of pipelining. However, the reassertion of SNPSTB# while snooping is in progress must not occur until after SNPCYC#—precisely, after the falling edge of SNPCYC# for strobed and clocked modes, or in the clock after SNPCYC# is active for synchronous mode. SNPSTB# must not be asserted between the first and last BGT# of a locked sequence. Similarly, SNPSTB# must not occur after the BGT# of the write through and before the BGT# of the allocation when a Read-for-Ownership transaction is occurring.

SNPSTB# itself does not affect the cache contents or states, but the snoop signals SNPINV and SNPNCA, latched upon SNPSTB#, force various changes in the cache on a snoop hit.

7.62.2 WHEN SAMPLED

SNPSTB# is sampled on every SNPCLK or CLK in clocked or synchronous modes, and is sampled constantly in strobed mode. While a snoop is in progress, a new SNPSTB# is recognized as a new, possibly pipelined, snoop request. After the assertion of a pipelined SNPSTB#, the SNPSTB# signal must not be reasserted until after the next SNPCYC#.

SNPSTB# should always meet proper set-up and hold times when operating in clocked or synchronous modes. When operating in strobed mode, it must meet minimum active/inactive times to be properly recognized in the next clock.

7.62.3 RELATION TO OTHER SIGNALS

SNPSTB# latches the following signals: SNPINV, SNPNCA, MBAOE#, and MAOE#, and the address on the MSET, MTAG, and MCFA pins. The address which appears on the MSET, MTAG, and MCFA address pins is to be snooped in the 82495XP. MAOE# acts as a qualifier for a snoop; if MAOE# is active when sampled on a SNPSTB# assertion, the snoop request is ignored. SNPINV and SNPNCA provide the 82495XP with snoop attributes which affect the state of a snoop hit cache entry.

If MBAOE# is active during SNPSTB# assertion, the 82495XP forces all bits in the subline address (those address bits which MBAOE# controls) to 0 on a snoop write back for that snoop.

Snoops and memory accesses are interlocked, such that after BGT# for a memory access has been issued, a SNPSTB# which is asserted will be latched, with its address and attributes, but will not cause a snoop until after SWEND# for that memory cycle. After BGT# has been issued for a cycle, snoop write-backs are delayed until after the CRDY# for that cycle. Likewise, once a snoop is underway (SNPCYC# active) BGT# is ignored until snoop completion.

SNPSTB# must not be deasserted and reasserted (specifically, cause a second falling edge) between its initial recognition and SNPCYC#—ie, SNPSTB# must not be asserted before the SNPCYC# of the previous SNPSTB#. In strobed and clocked modes, SNPSTB# can be reasserted after the falling edge of SNPCYC#; in synchronous mode, SNPSTB# can be reasserted in the CLK after SNPCYC# is active. This second assertion of SNPSTB#, after SNPCYC#, can occur while the first snoop is still progressing (SNPBSY# is active), allowing one level of snoop pipelining. In this case, a third assertion of SNPSTB# must not occur until after the SNPCYC# for the second, piped snoop request.

SNPSTB# must not be asserted while the 82495XP is executing a locked sequence (LOCK# active). Specifically, SNPSTB# must not be asserted after the BGT# for the first locked access and before the BGT# of the last locked access.

Systems which support Read-for-Ownership must not assert SNPSTB# between the BGT# of the write through and the BGT# of the allocation during a Read-for-Ownership operation.

7.63 SWEND#

Snoop Window End

Closes Snooping Window

Input to 82495XP (pin Q1) Cycle Progress Signal

Synchronous to CLK

2

7.63.1 SIGNAL DESCRIPTION

SWEND# is an input to the 82495XP that, when asserted, closes the snooping window and causes sampling of MWB/WT# and DRCTM#. Once snooping of all other 82495XP's is complete, DRCTM# and MWB/WT# can be determined.

Snoop response is blocked by the 82495XP between BGT# and SWEND# activation. Therefore, the faster SWEND# is closed, faster snoops can be determined.

All CPU-generated write cycles and cache read miss cycles must cause a snoop on the memory bus. SWEND# may be activated once snooping has completed for these cycles. SWEND# activation causes the 82495XP's internal tags to change state for the current cycle (if necessary). DRCTM# and MWB/WT# influence the state change decision.

SWEND# need only be activated for those cycles which require the sampling of DRCTM# and MWB/WT#.

If a cycle does not specifically require SWEND#, and SWEND# is not returned, snooping is blocked from BGT# to CRDY#. For this reason, it may be more efficient to always return SWEND#.

7.63.2 WHEN SAMPLED

SWEND# is sampled by the 82495XP on the clock or after KWEND# is sampled active for those cycles that sample KWEND#. For cycles that do not sam-

ple KWEND#, SWEND# is sampled with or after BGT#. Once SWEND# is sampled active, it is ignored until KWEND# of the next cycle. If SWEND# is not being sampled, it may violate setup and hold times.

Snoop response is blocked between BGT# and SWEND#. If a snoop is initiated between BGT# and SWEND#, the MTHIT# and MHITM# response is given after SWEND# activation. Any subsequent snoop write back would begin after CRDY#.

7.63.3 RELATION TO OTHER SIGNALS

SWEND# causes the sampling of MWB/WT# and DRCTM#. SWEND# is sampled once KWEND# is sampled active. BGT#, KWEND#, and SWEND# may be asserted in the same clock.

SWEND# shares a pin with CFG1.

7.64 SYNC#

Sync

Synchronizes 82495XP TAG array with Main Memory

Input to 82495XP (Q4) Cache Synchronization Signal

Asynchronous

7.64.1 SIGNAL DESCRIPTION

SYNC# activation will cause the synchronization of the 82495XP and i860 XP CPU tag arrays with main memory. The 82495XP will flush all modified entries to memory. All valid tag entries will be kept, with modified [M] state lines becoming non-modified [E] state lines.

7.64.2 WHEN SAMPLED

SYNC# can be asserted at any time. The 82495XP will complete all outstanding cycles on the CPU and memory bus before beginning the SYNC process. The memory bus controller does not have to prevent SYNC# during locked cycles because the 82495XP will complete its locked cycle before the SYNC process will begin.

Once a SYNC operation has begun, the SYNC# signal is ignored until the operation completes. If RESET or FLUSH# is asserted while the SYNC operation is in progress, the SYNC operation will be aborted and the RESET or FLUSH immediately executed.

SYNC# is an asynchronous input. SYNC# must have a pulse width of 2 CLK's in order to guarantee 82495XP recognition.

7.64.3 RELATION TO OTHER SIGNALS

To initiate a SYNC, the 82495XP will complete all pending cycles and prohibit further ADS#'s to occur while a SYNC is in progress. The FSIOUT# output signal is used to indicate the start and end of the SYNC operation. It will become active when the SYNC# signal is internally recognized (all outstanding cycles have completed) and will de-activate when the SYNC operation has completed.

The memory bus controller supplies BRDY# to the CPU once the SYNC has completed. Once SYNC has begun, and FSIOUT# active, all CADS#'s and CRDY#'s correspond to the write-backs caused by the SYNC operation.

The 82495XP can be snooped during SYNC cycles and the snooping protocols will be the same as that for any memory bus cycle.

7.65 TCK

Test Clock

Clock for the JTAG boundary scan tests

Input to the i860 XP CPU (pin Q1) Test Signal

Input to the 82495XP (pin P3)

Input to the 82490XP (pin 3)

Synchronous

7.65.1 SIGNAL DESCRIPTION

TCK is an input to the i860 XP CPU, 82495XP and 82490XP and provides the clocking function required by the JTAG boundary scan feature. TCK is used to clock state information and data into and out of the component. State select information and data are clocked into the component on the rising edge of TCK on TMS and TDI, respectively. Data is clocked out of the part on the falling edge of TCK on TDO.

In addition to using TCK as a free running clock, it may be stopped in a low, logic 0, state, indefinitely as described in IEEE 1149.1. While TCK is stopped in the low state, the boundary scan latches retain their state.

When boundary scan is not used, TCK should be tied low.

7.65.2 WHEN SAMPLED

TCK is a clock signal and is used as a reference for sampling other JTAG signals.

7.65.3 RELATION TO OTHER SIGNALS

On the rising edge of TCK, TMS and TDI are sampled. On the falling edge of TCK, RDO is driven.

7.66 TDI

Test Data Input

Receives serial test instructions and data

Input to the i860 XP CPU (pin S14) Test Signal

Input to the 82495XP (pin N3)

Input to the 82490XP (pin 2)

Synchronous to TCK

7.66.1 SIGNAL DESCRIPTION

TDI is the serial input used to shift JTAG instructions and data into the component. The shifting of instructions and data occurs during the SHIFT-IR and SHIFT- DR TAP controller states, respectively. These states are selected using the TMS signal as described in chapter 9.

An internal pull up resistor is provided on TDI to ensure a known logic state if an open circuit occurs on the TDI path. Note than when "1" is continuously shifted into the instruction register, the BYPASS instruction is selected.

7.66.2 WHEN SAMPLED

TDI is sampled on the rising edge of TCK, during the SHIFT-IR and the SHIFT-DR states. During all other TAP controller states, TDI is a "don't care".

7.66.3 RELATION TO OTHER SIGNALS

TDI is only sampled when TMS and TCK have been used to select the SHIFT-IR or SHIFT-DR states in the TAP controller.

For proper initialization of JTAG logic, TDI should be driven high, "1", for at least four TCK cycles following the rising edge of RESET.

7.67 TDO

Test Data Output

Outputs serial test instructions and data

Output from the i860 XP CPU (pin R10) Test Signal

Output from the 82495XP (pin C4)

Output from the 82490XP (pin 84)

Synchronous to TCK

7.67.1 SIGNAL DESCRIPTION

TDO is the serial output used to shift JTAG instructions and data out of the component. The shifting of instructions and data occurs during the SHIFT-IR and SHIFT- DR TAP controller states, respectively. These states are selected using the TMS signal as described in chapter 9.

When not in SHIFT-IR or SHIFT-DR state, TDO is driven to a high impedance state to allow connecting TDO of different devices in parallel.

7.67.2

TDO is driven on the falling edge of TCK during the SHIFT-IR and SHIFT- DR TAP controller states. At all other times TDO is driven to the high impedance state.

7.67.3

TDO is only driven when TMS and TCK have been used to select the SHIFT- IR or SHIFT-DR states in the TAP controller.

7.68 TMS

Test Mode Select

Controls testing by selecting mode of operation

Input to the i860 XP CPU Test Signal

Input to the 82495XP (pin P2)

Input to the 82490XP (pin 1)

Synchronous to TCK

7.68.1 SIGNAL DESCRIPTION

TMS is decoded by the JTAG TAP (Tap Access Port) to select the operation of the test logic, as described in chapter 9.

To guarantee deterministic behavior of the TAP controller TMS is provided with an internal pull-up resistor. If boundary scan is not used, TMS may be tied high or left unconnected.

7.68.2 WHEN SAMPLED

TMS is sampled on every rising edge of TCK.

7.68.3 RELATION TO OTHER SIGNALS

TMS is used to select the internal TAP states required to load boundary scan instructions to data on TDI.

For proper initialization of the JTAG logic, TMS should be driven high, "1", for at least four TCK cycles following the rising edge of RESET.

7.69 Vcc and Vss

Power and Ground Pins

See Tables 1.1 and 1.2 for locations.

7.70 WWOR#

Weak Write Ordering Mode

Enforces strong/weak write-ordering policy

Input to 82495XP (pin Q2) Configuration Signal

Synchronous to CLK

7.70.1 SIGNAL DESCRIPTION

When asserted during reset, the 82495XP enforces a weak write ordering policy. If WWOR# is deasserted during reset, the 82495XP enforces a strong write-ordering policy.

In a strong write-ordering mode, writes to the memory bus are forced to occur in the order in which they were posted by the CPU. In a weak write-ordering mode it is possible for:

1. A CPU posted write (A) to be waiting in a 82495XP/82490XP memory buffer.
2. A subsequent CPU write (B) to complete in the 82495XP/82490XP because it was a hit to M or E state.
3. A snoop hit to B to cause a write back of B before A is written.

In this scenario, B is written to memory before A is, and thus CPU writes have been reordered.

7.70.2 WHEN SAMPLED

WWOR# is sampled during reset like figure 7-1 with a setup time of 4 CPU clocks. WWOR# becomes MALE once FSIOUT# indicates that the 82495XP reset sequence has completed.

7.70.3 RELATION TO OTHER SIGNALS

WWOR# shares a pin with MALE.

8.0 BUS FUNCTIONAL DESCRIPTION AND TIMING

The 82495XP/82490XP cache core supports a wide variety of bus transfers to meet the needs of high performance systems. Bus transfers can be single cycle or multiple cycle, cacheable or non-cacheable, 64- or 128-bit (memory bus), and locked. To support multiprocessing systems there are cache back-invalidation, inquire, snooping, read for ownership, cache to cache transfers, and locked cycles.

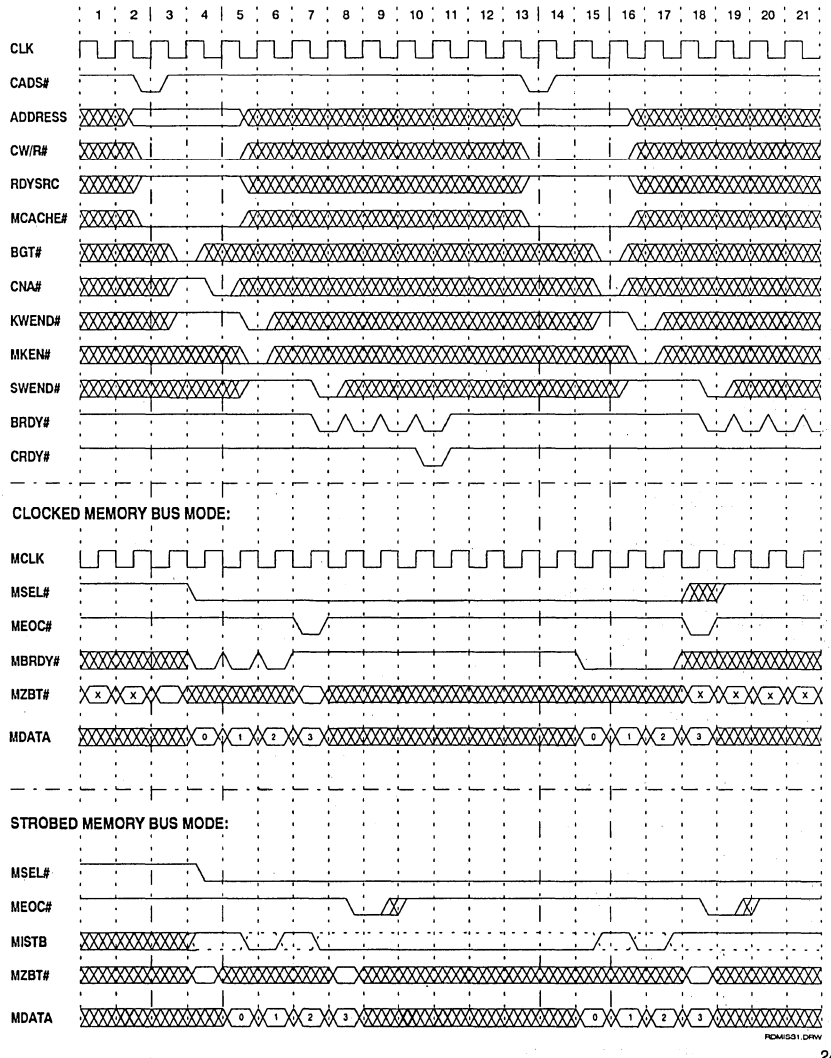
This section begins with read cycles, both cacheable and non-cacheable. It moves on to write cycles, cacheable and non-cacheable. Snooping cycles are discussed next with an example of each snooping mode. The remaining sections describe special cycles: read for ownership, I/O, and locked cycles.

The cycles shown in this chapter are examples of various types of 82495XP/82490XP cycles. The purpose of these examples is to show signal relationships, and are not necessarily best case scenarios.

8.1 Read Cycles

8.1.1 READ HITS

Read Hit cycles are executed completely within the CPU/Cache core, and will not be seen by the MBC.



240956-33

Figure 8-1. Cacheable Read Miss with Clean Replacement

8.1.2 CACHEABLE READ MISSES

8.1.2.1 Read Miss with Clean Replacement

Figure 8.1 illustrates CPU initiated Read cycles that miss the 82495XP/82490XP cache and replace a non-dirty (eg. clean or empty) line in the cache. In such cycles, the 82495XP will instruct the MBC to perform a cache line-fill cycle on the memory bus. A cache line-fill is a read of a complete 82495XP/82490XP line from main memory. The line is then written into the 82490XP's array, and data transferred to the CPU as requested. If the line fetched from main memory replaces a 82495XP/82490XP cache line which is in valid unmodified state ([E] or [S]), then a back-invalidation cycle is performed on the CPU bus to guarantee that the replaced data is also removed from the CPU's first level cache, thus maintaining the inclusion property.

CACHE CONTROL SIGNALS:

The CPU initiates the read cycle to the 82495XP/82490XP cache where the cache tag state is looked up. Once the 82495XP determines the cycle to be a cache miss, it issues CADS# (clock 2) and the associated cycle control signals to the MBC (eg. CW/R#, CM/IO#, CD/C#, RDYSRC, MCACHE#) in order to schedule the cache line-fill operation. MCACHE# is active, indicating that the read miss is potentially cacheable by the 82495XP; RDYSRC is active, indicating that the MBC must supply BRDY#s to the CPU cache core.

The memory bus address (MSET[10:0], MTAG[11:0], MCFA[6:0]) is valid with CADS# (clocks 2 and 13 for the two cycles in this example) and remain valid until after CNA# is sampled active by the 82495XP (clocks 5 and 16). MALE and MBALE may be used to hold the address as necessary.

The MBC arbitrates for the memory bus and returns BGT# asserted (clock 3), indicating that the cycle is guaranteed to complete on the memory bus. Once the 82495XP samples BGT# asserted, it must finish that cycle on the memory bus. Prior to this point, the cycle can be aborted by a snoop hit in the cache.

CNA# is asserted by the MBC (clock 4) to indicate that it is ready to schedule a new memory bus cycle. Note that after CNA# activation, cycle control signals are not guaranteed to be valid.

When the MBC has determined the cacheability attribute of the cycle, it drives the MKEN# signal accordingly. The MBC also drives the KWEND# signal at this time, indicating the end of the cacheability window. The 82495XP samples MKEN# during KWEND# (clock 5) to determine that the cycle is indeed cacheable.

The MBC asserts SWEND# when the snoop window ends on the memory bus. The 82495XP samples MWB/WT# and DRCTM# during SWEND# (clock 7) and updates the cache tag state according to the consistency protocol. The closure of the snoop window also enables the MBC to start providing the CPU with data that has been stored in the 82490XP's memory cycle buffer. The MBC supplies BRDY#s to the CPU (clocks 7-10).

The first cycle ends when CRDY# is driven active by the MBC (clock 10). It is at this time that the data in the 82490XP's memory cycle buffers is loaded into the cache SRAM.

The 82495XP issues a new CADS# in clock 13, which also misses the 82495XP/82490XP cache. Note that once the cycle progress signals (BGT#, CNA#, KWEND#, SWEND#) of a cycle are sampled asserted, the 82495XP ignores them until the CRDY# of that cycle. The 82495XP does not pipeline the cycle progress signals (ie. it will not sample them again until after CRDY# of the current memory bus cycle).

MEMORY BUS SIGNALS:

The memory address latch enables (MALE and MBALE) may remain asserted by the MBC to place the address latches in flow through mode. If the 82495XP is the current bus master, the memory address output enables (MAOE# and MBAOE#) should be asserted by the MBC. MDOE# must be inactive to allow the data pins to be used as inputs.

Some time after the address has been driven onto the memory bus, data will be supplied from the DRAM (main memory) to the 82490XP cache SRAM.

For Clocked Memory Bus Mode, MSEL# is driven active by the MBC (clock 4) to allow sampling of MBRDY# and to latch MZBT# for the transfer. MZBT# is sampled on all MCLK edges where MSEL# is inactive. Once MSEL# is sampled active by the 82495XP, the value of MZBT# sampled on the prior MCLK is used for the next transfer. MBRDY# is driven active by the MBC in clocks 4 to 6 to cause the memory burst counter to be incremented and data to be placed into the 82490XP

cache memory cycle buffers. The MBC drives MEOC# asserted (clock 7) to end the current cycle on the memory bus and switch memory cycle buffers for the new cycle. MZBT# is latched at this time (when MEOC# is sampled asserted and MSEL# remains low) for the next transfer.

MBRDY# is driven active by the MBC in clocks 15 to 17 to read data into the 82490XP cache memory cycle buffers. The MBC asserts MEOC# (clock 18) to end the second read miss cycle on the memory bus and switch the memory cycle buffers for a new cycle.

For Strobed Memory Bus Mode, MSEL# is driven active by the MBC (clock 4) to allow MISTB operation and to latch MZBT# (on the falling edge of MSEL#) for the transfer. MISTB is toggled in clocks 5 to 7 to cause the memory burst counter to be incremented, and data to be placed into the 82490XP cache memory cycle buffers. Note: MISTB latches the memory bus data on both the rising and falling edges. The MBC drives MEOC# asserted (clock 8) to end the current cycle on the memory bus and switch memory cycle buffers for the new cycle. MZBT# for the next cycle, is sampled at this time on the falling edge of MEOC#.

MISTB is toggled by the MBC (clocks 15 to 17) to read data into the 82490XP memory cycle buffers. The MBC asserts MEOC# (clock 18) to end the second read miss cycle on the memory bus and switch the memory cycle buffers for a new cycle.

8.1.2.2 Read Miss with Replacement of Dirty Line

Figure 8.2 illustrates a CPU read cycle which misses the 82495XP cache, and requires the replacement of a modified line (eg. tag replacement, lines/sector=1 line ratio=1). In such cycles, the 82495XP will instruct the MBC to perform a cache line-fill on the memory bus, instruct the 82490XP to fill its write-back buffer with the contents of the array location corresponding to the line which must be replaced, and perform a back invalidation to the CPU to maintain the first and second level cache consistency. Once the cache line-fill has completed, the 82495XP/82490XP will write back the contents of the replaced line to main memory from the 82490XP write-back buffer.

CACHE CONTROL SIGNALS:

The CPU initiates the read cycle to the 82495XP/82490XP cache where the cache tag state is looked up. Once the 82495XP determines the cycle to be a cache miss, it issues CADS# (clock 1) and the associated cycle control signals to the MBC (eg. CW/R#, CM/IO#, CD/C#, RDYSRC, MCACHE#) in order to schedule the cache line-fill operation. MCACHE# is active, indicating that the read miss is potentially cacheable by the 82495XP; RDYSRC is active, indicating that the MBC must supply BRDY#s to the CPU cache core.

The memory bus address (MSET[10:0], MTAG[11:0], MCFA[6:0]) is valid with CADS# (clocks 1 and 5 for the two cycles in this example) and remain valid until after CNA# is sampled active by the 82495XP (clocks 4 and 10). MALE and MBALE may be used to hold the address as necessary.

The MBC arbitrates for the memory bus and returns BGT# asserted (clock 2), indicating that the cycle is guaranteed to complete on the memory bus. At this point, the 82490XP's write-back buffer is prefilled with the line to be replaced. Once the 82495XP samples BGT# asserted, it must finish that cycle on the memory bus. Prior to this point, the cycle can be aborted by a snoop hit from another cache.

CNA# is asserted by the MBC (clock 3) to indicate that it is ready to schedule a new memory bus cycle. Note that after CNA# activation, cycle control signals are not guaranteed to be valid.

When the MBC has determined the cacheability attribute of the cycle, it drives the MKEN# signal accordingly. The MBC also drives the KWEND# signal at this time, indicating the end of the cacheability window. The 82495XP samples MKEN# during KWEND# (clock 4) to determine that the cycle is indeed cacheable.

The MBC asserts SWEND# (clock 6) when the snoop window ends on the memory bus. The closure of the snoop window enables the MBC to start providing the CPU with data that has been stored in the 82490XP's memory cycle buffer. The MBC supplies BRDY#s to the CPU (clocks 6-9) to serve the read cycle. Note that data may be supplied to the 82490XP's immediately after MSEL# activation, and need not wait for SWEND#.

On the memory bus, the 82495XP issues a write-back (WB) cycle. CNA# is sampled active in clock 3 causing the 82495XP to issue the CADS# (also CDTS#) of the write-back (clock 5). The MBC knows this is a write back cycle and not a CPU initiated write cycle by sampling MCACHE# asserted. This tells the MBC how many data transfers are necessary.

BGT#, CNA#, and KWEND# of the write-back are sampled asserted by the MBC (clock 9) after the CRDY# of the read miss cycle (clock 8). At this

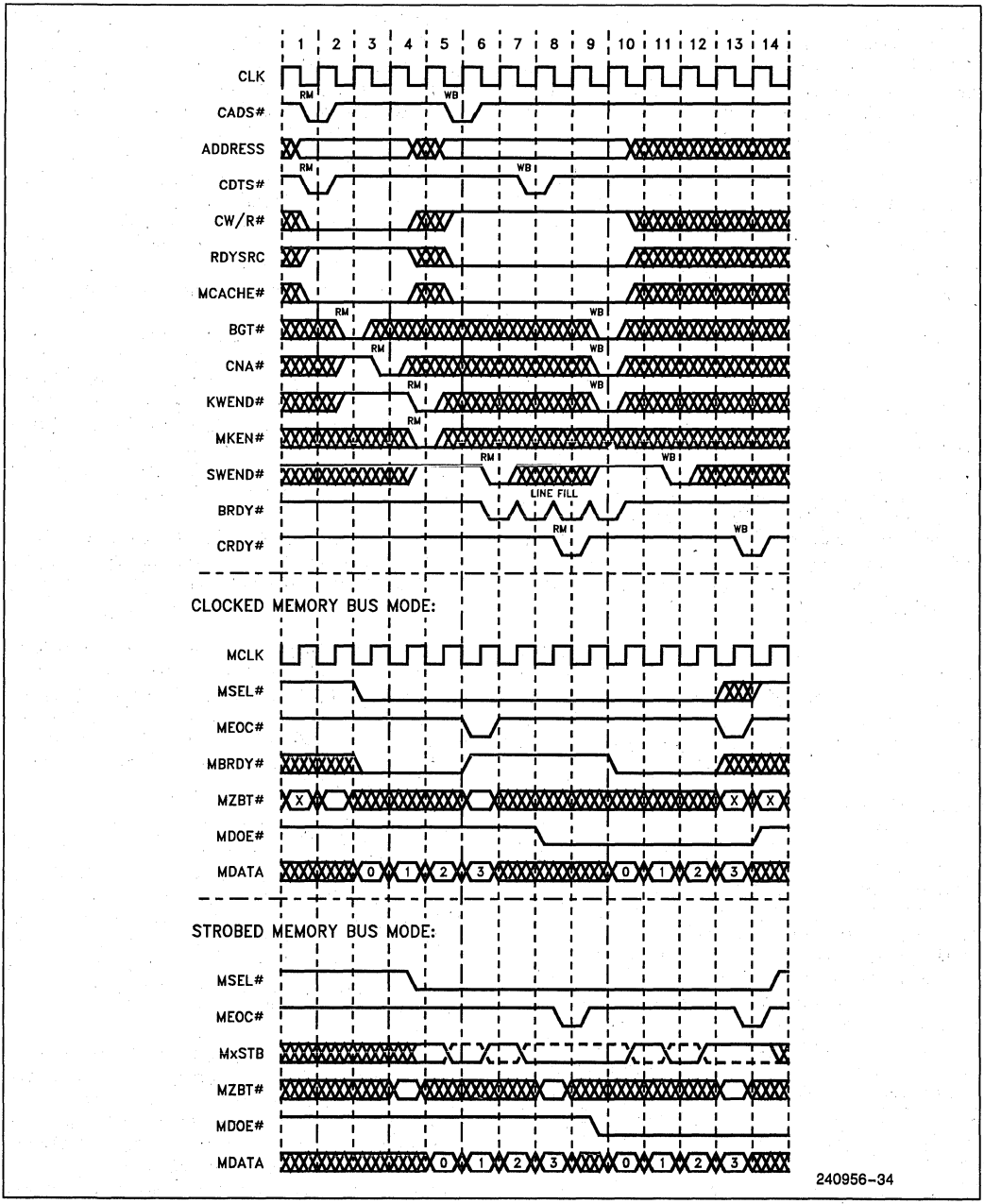


Figure 8-2. Cacheable Read Miss with Replacement of Dirty Line

point, the 82495XP may issue another CADS# for a new (unrelated) memory bus cycle. It is at this time that the data in the 82490XP's memory cycle buffers is loaded into the cache SRAM. The data to be written back to main memory is in the 82490XP's write back buffers.

The snoop window for the write back cycle is closed by the MBC in clock 11, and the cycle is ended by CRDY# sampled asserted in clock 13.

MEMORY BUS SIGNALS:

The memory address latch enables (MALE and MBALE) may remain asserted by the MBC to place the address latches in flow through mode. If the 82495XP is the current bus master, the memory address output enables (MAOE# and MBAOE#) should be asserted by the MBC.

Some time after the address has been driven onto the memory bus, data will be supplied from the DRAM (main memory) to the 82490XP cache SRAM.

For Clocked Memory Bus Mode, MSEL# is driven active by the MBC (clock 3) to allow sampling of MBRDY# and to latch MZBT# for the transfer. MZBT# is sampled on all MCLK edges where MSEL# is inactive. Once MSEL# is sampled active by the 82495XP, the value of MZBT# sampled on the prior MCLK is used for the next transfer. MBRDY# is driven active by the MBC in clocks 3 to 5 to cause the memory burst counter to be incremented and data to be placed into the 82490XP cache memory cycle buffers. The MBC drives MEOC# asserted (clock 6) to end the current cycle on the memory bus and switch memory cycle buffers for the new cycle. MZBT# is latched at this time (when MEOC# is sampled asserted) for the next transfer.

The MBC asserts the memory data output enable signal (MDOE#, clock 8) to drive the memory data outputs.

MBRDY# is driven active by the MBC in clocks 10 to 12 to write data from the 82490XP cache memory cycle buffers onto the memory bus. The MBC asserts MEOC# (clock13) to end the write back cycle on the memory bus and switch the memory cycle buffers for a new cycle.

For Strobed Memory Bus Mode, MSEL# is driven active by the MBC (clock 4) to allow MISTB operation and to latch MZBT# for the transfer (on MSEL# falling edge). MISTB is toggled in clocks 5 to 7 to cause the memory burst counter to be incremented, and data to be placed into the 82490XP cachememory cycle buffers. Note: MISTB latches the memory bus data on both the rising and falling edges. The MBC drives MEOC# asserted (clock 8) to end the current cycle on the memory bus and switch memory cycle buffers for the new cycle. MZBT# for the next cycle, is latched at this time on the falling edge of MEOC#.

The MBC asserts MDOE# (clock 9) to drive the memory data outputs.

MOSTB is toggled by the MBC (clocks 10 to 12) to write data from the 82490XP memory cycle buffers onto the memory bus. The MBC asserts MEOC# (clock 13) to end the write back cycle on the memory bus and switch the memory cycle buffers for a new cycle.

2

8.1.3 NON-CACHEABLE READ MISSES

8.1.3.1 Read Misses not Cacheable by CPU/ Cache Core and Cacheable by Core, but not by Memory Bus

Figure 8.3 illustrates two CPU read cycles which miss the 82495XP cache, and are non-cacheable. In the first cycle, the CPU/Cache core forces the read to be non-cacheable (as indicated by the MCACHE# output from the 82495XP). In the second cycle, non-cacheability of the data is forced by the memory bus (as indicated by the MKEN# input from the MBC). Since both cycles are not cacheable, there is no line-fill operation performed, the cycles are merely echoed to the memory bus.

CACHE CONTROL SIGNALS:

The CPU initiates the first read cycle to the 82495XP/82490XP cache where the cache tag state is looked up. Once the 82495XP determines the cycle to be a cache miss, it issues a cycle request (CADS# in clock 1) and the associated cycle control signals to the MBC (eg. CW/R#, CM/IO#, CD/C#, RDYSRC, MCACHE#) in order to schedule the read operation. RDYSRC is active, indicating that the MBC must provide BRDY# to the CPU; MCACHE# is not active, indicating that the read miss in not cacheable by the CPU/Cache core.

The memory bus address (MSET[10:0], MTAG[11:0], MCFA[6:0]) is valid with CADS# (clocks 1 and 5 for the two cycles in this example) and remain valid until after CNA# is sampled active by the 82495XP (clocks 4 and 10). MALE and MBALE may be used to hold the address as necessary.

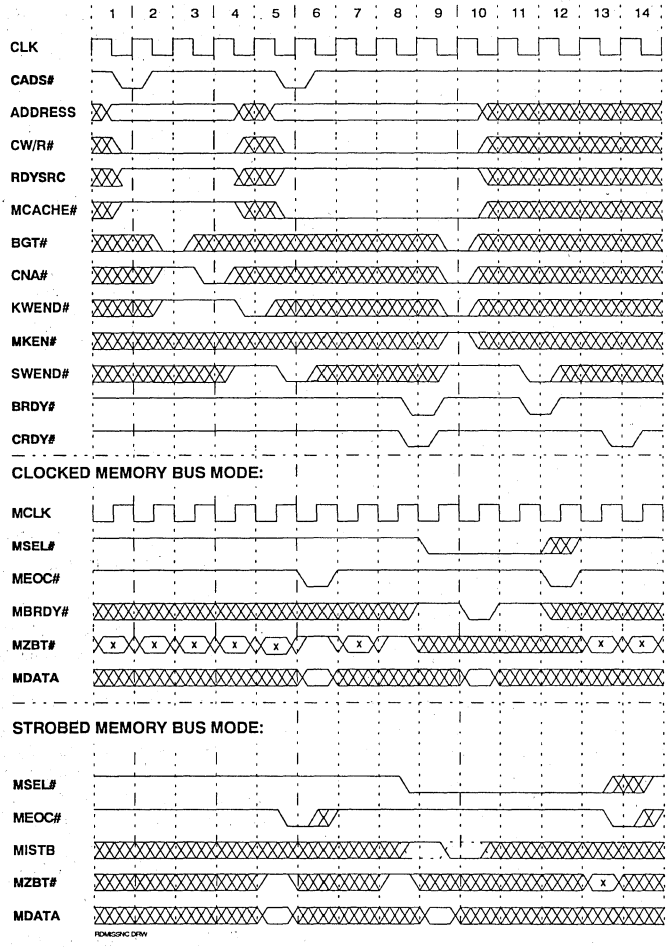


Figure 8-3. Non-Cacheable Read Misses

The MBC arbitrates for the memory bus and returns BGT# asserted (clock 2), indicating that the cycle is guaranteed to complete on the memory bus. Once the 82495XP samples BGT# asserted, it must finish that cycle on the memory bus. Prior to this point, the cycle can be aborted by a snoop hit from another cache.

CNA# is asserted by the MBC (clock3) to indicate that it is ready to schedule a new memory bus cycle. Note that after CNA# activation, cycle control signals are not guaranteed to be valid.

This cycle has already been determined to be non-cacheable; therefore, The MBC does not need to assert SWEND#, KWEND#, or MKEN# to the 82495XP/82490XP cache. The MBC supplies BRDY# to the CPU to complete the cycle to the CPU. The MBC asserts CRDY (clock 8) to the 82495XP/82490XP to complete the read miss cycle on the memory bus.

The 82495XP issues a new (unrelated) cycle request (CADS# in clock 5) which also misses the 82495XP/82490XP cache. Since the 82495XP has already sampled CNA# asserted, it issues a new CADS# prior to receiving CRDY# of the current cycle (ie. this cycle is pipelined within the MBC). Note that once the cycle progress signals of a cycle are sampled asserted, the 82495XP ignores them until the CRDY# of that cycle. The 82495XP will not sample the cycle progress signals again until after the CRDY# of the current memory bus cycle. The current read cycle is completed on the bus in clock 8 with CRDY# assertion.

The cycle progress signals for the second read miss are also valid at this time (clock 5). RDYSRC is active, indicating that the MBC must provide BRDY#s to the CPU/Cache core; and MCACHE# is active, indicating that the read miss is potentially cacheable by the 82495XP/82490XP.

The MBC issues BGT# and CNA# to the 82495XP in clock 9 to indicate that the cycle is guaranteed to complete on the memory bus, and that it is ready to schedule a new memory bus cycle. KWEND# is asserted at this time to close the cacheability window. MKEN# is not active, indicating to the 82495XP that the read miss cycle is not cacheable by the memory bus. KWEND# and MKEN# must be returned to the 82495XP at least two clocks prior to BRDY# to inform the CPU that a line fill will not follow.

The MBC asserts SWEND# (clock 11) to close the snoop window, and CRDY# (clock 13) to complete the cycle to the 82495XP/82490XP. Note: SWEND# is not needed since the cycle was not cacheable.

NOTE:
Both examples show single cycle read requests.

MEMORY BUS SIGNALS:

The memory address latch enables (MALE and MBALE) may remain asserted by the MBC to place the address latches in flow through mode. If the 82495XP is the current bus master, the memory address output enables (MAOE# and MBAOE#) should be asserted by the MBC. The memory data output enable (MDOE#) must be inactive to allow the data pins to be used as inputs.

Some time after the address has been driven onto the memory bus, data will be supplied from the DRAM (main memory) to the 82490XP memory cycle buffers.

For Clocked Memory Bus Mode, MEOC# is asserted by the MBC (clock 6) to latch MZBT# for the next transfer, and end the current cycle on the memory bus (MBRDY# and MSEL# are not necessary since this example shows a single transfer cycle). MZBT# is driven high by the MBC in order to force the read cycle to begin with a non-zero burst address.

For the second non-cacheable read cycle, MSEL# is driven active by the MBC (clock 8) to allow sampling of MBRDY# and to latch MZBT# for the transfer. MZBT# is sampled on all MCLK edges where MSEL# is inactive. Once MSEL# is sampled active by the 82495XP, the value of MZBT# sampled on the prior MCLK is used for the next transfer. Again, MZBT# is driven high by the MBC to force the transfer to begin with the correct burst address. MBRDY# is driven active by the MBC in clock 10 to cause the memory burst counter to be incremented and data to be placed into the 82490XP cache memory cycle buffers. The MBC drives MEOC# asserted (clock 12) to end the current cycle on the memory bus and switch memory cycle buffers for the new cycle.

For Strobed Memory Bus Mode, MEOC# is driven active by the MBC (clock 5) to latch MZBT# for the transfer (on MEOC# falling edge), and end the current cycle on the memory bus (MISTB is not necessary since this example shows a single transfer cycle). MZBT# is driven high by the MBC in order to force the read cycle to begin with the correct burst address.

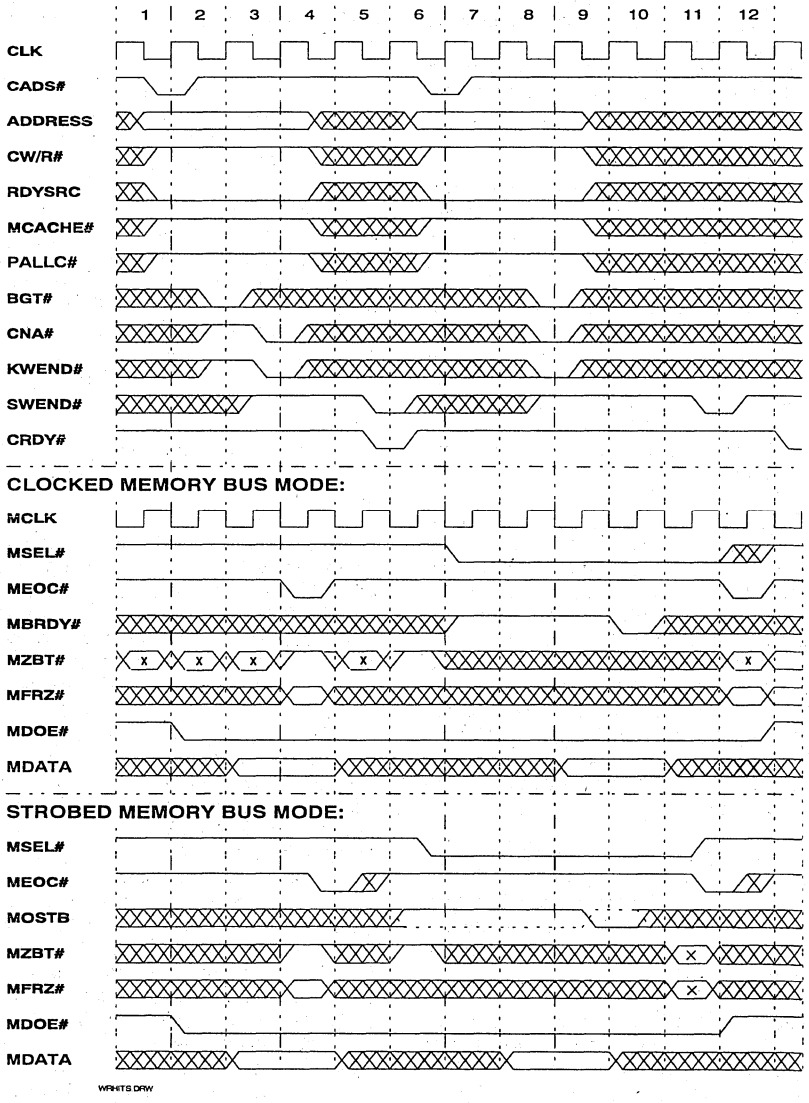


Figure 8-4. Write Hit to [S] State Line (Write-Through)

For the second non-cacheable read cycle, MSEL# is driven active by the MBC (clock 8) to allow MISTB operation and to latch MZBT# for the transfer (on MSEL# falling edge). Again, MZBT# is driven high by the MBC to force the transfer to begin with the correct burst address. MISTB is toggled in clock 9 to cause the memory burst counter to be incremented, and data to be placed into the 82490XP cache memory cycle buffers. Note: MISTB latches the memory bus data on both the rising and falling edges. The MBC drives MEOC# asserted (clock 13) to end the current cycle on the memory bus and switch memory cycle buffers for the new cycle. MZBT# for the next cycle (not shown), is sampled at this time on the falling edge of MEOC#.

8.2 Write Cycles

8.2.1 WRITE HITS

8.2.1.1 Write Hit to [E] or [M] States

CPU initiated write cycles which hit 82495XP entries tagged in the [E] or [M] states are executed completely within the CPU/Cache core, and will not be seen by the MBC.

8.2.1.2 Write Hit to [S] State

Figure 8.4 illustrates CPU initiated write cycles which hit lines in the 82495XP/82490XP cache array that are in the shared state. If the 82495XP/82490XP is used as a write through cache (not write back), the [S] state is the only state a cached line could be in. These cycles are posted as are all normal write cycles (as long as no other write miss is pending).

CACHE CONTROL SIGNALS:

The CPU initiates the write cycle to the 82495XP/82490XP cache where the cache tag state is looked up. Once the 82495XP determines the cycle to be a hit to shared state, it posts the write and returns BRDY# to the CPU.

The 82495XP next issues a cycle request (CADS# in clock 1), and the associated cycle control signals to the MBC (eg. CW/R#, CM/IO#, CD/C#, RDYSRC, MCACHE#, PALLC#) in order to schedule the write through operation. MCACHE# is not active since the write will be posted; RDYSRC is not active, indicating that the 82495XP will supply BRDY# to the CPU; PALLC# is not active, indicating that an allocation cycle will not be performed (regardless of MKEN# state) since the line is already available in the cache. The MBC must also latch PWT and PCD on BLE# falling edge in order to track hits and misses to the [S] state. This is how an external state tracker can track the [S] state.

The memory bus address (MSET[10:0], MTAG[11:0], MCFA[6:0]) is valid with CADS# (clocks 1 and 6 for the two cycles in this example) and remains valid until after CNA# is sampled active by the 82495XP (clocks 4 and 9). MALE and MBALE may be used to hold the address as necessary.

The MBC arbitrates for the memory bus and returns BGT# asserted (clock 2), indicating that the cycle is guaranteed to complete on the memory bus. Once the 82495XP samples BGT# asserted, it must finish that cycle on the memory bus. Prior to this point, the cycle can be aborted by a snoop hit from another cache.

CNA# is asserted by the MBC (clock 3) to indicate that it is ready to schedule a new memory bus cycle. Note that after CNA# activation, cycle control signals are not guaranteed to be valid. KWEND# is also driven at this time since the cacheability of this cycle is already known and MKEN# is a don't care. It is not necessary that KWEND# be asserted at this time.

The 82495XP provides BRDY# to the CPU since the cycles are posted writes. The MBC completes the first write hit to [S] state in clock 5 when it asserts CRDY# to the 82495XP/82490XP cache. The data is latched in to the 82490XP array from the memory cycle buffer at this time.

In this example, the 82495XP issues a second write to [S] state in clock 6. For this cycle, the 82495XP issues the memory bus request (CADS#) as soon as it can after sampling CNA# asserted. The 82495XP will not wait for KWEND# (if it does not get asserted immediately as in this example) to issue CADS# since this is not a potential allocate cycle (ie. PALLC# active).

The MBC asserts BGT#, CNA#, and KWEND# together in clock 8 to indicate that the current cycle is guaranteed to complete and the 82495XP is free to schedule a new memory bus cycle.

Again, the 82495XP provides BRDY# to the CPU since the cycles are posted writes. The MBC completes the second write hit to [S] state in clock 12 when it asserts CRDY# to the 82495XP/82490XP cache. The data is latched in to the 82490XP array from the memory cycle buffer at this time.

MEMORY BUS SIGNALS:

The memory address latch enables (MALE and MBALE) may remain asserted by the MBC to place the address latches in flow through mode. If the 82495XP is the current bus master, the memory address output enables (MAOE# and MBAOE#) should be asserted by the MBC.

For Clocked Memory Bus Mode, the memory data output enable signal (MDOE#) is asserted by the MBC in clock 2 to drive the memory data outputs.

MEOC# is asserted by the MBC (clock 4) to latch MZBT# for the transfer, and end the current cycle on the memory bus (MBRDY# is not necessary since this example shows a single transfer cycle). MZBT# is driven high by the MBC in order to force the write cycle to begin with the correct burst address . MFRZ# is sampled here (it need not be active since the cycle is not potentially allocatable).

For the second write through cycle, MSEL# is driven active by the MBC (clock 7) to allow sampling of MBRDY# and to latch MZBT# for the transfer. MZBT# is sampled on all MCLK edges where MSEL# is inactive. Once MSEL# is sampled active by the 82495XP, the value of MZBT# sampled on the prior MCLK is used for the next transfer. Again, MZBT# is driven high by the MBC to force the transfer to begin with the correct burst address. MBRDY# is driven active by the MBC in clock 10 to cause the memory burst counter to be incremented and data to be placed into the 82490XP cache memory cycle buffers. The MBC drives MEOC# asserted (clock 12) to end the current cycle on the memory bus and switch memory cycle buffers for the new cycle.

For Strobed Memory Bus Mode, the memory data output enable (MDOE#) is asserted by the MBC in clock 2 to drive the memory data outputs.

MEOC# is driven active by the MBC (clock 4) to latch MZBT# for the transfer (on MEOC# falling edge), and end the current cycle on the memory bus (MOSTB is not necessary since this example shows a single transfer cycle). MZBT# is driven high by the MBC in order to force the read cycle to begin with the correct burst address.

For the second write through cycle, MSEL# is driven active by the MBC (clock 6) to allow MOSTB operation and to latch MZBT# for the transfer (on MSEL# falling edge). Again, MZBT# is driven high by the MBC to force the transfer to begin with the correct burst address. MOSTB is toggled in clock 9 to cause the memory burst counter to be incremented, and data to be placed into the 82490XP cache memory cycle buffers. Note: MOSTB latches the memory bus data on both the rising and falling edges. The MBC drives MEOC# asserted (clock 11) to end the current cycle on the memory bus and switch memory cycle buffers for the new cycle. MZBT# for the next cycle (not shown), is sampled at this time on the falling edge of MEOC#.

8.2.2 WRITE MISSES

8.2.2.1 Write Miss with no Allocation

Figure 8.5 illustrates two CPU initiated write cycles which miss the 82495XP/82490XP cache and are not allocatable. The first write cycle begins as a potentially allocatable cycle, but MKEN# sampled inactive indicates that the cycle is not cacheable by the memory bus. The second write miss cycle is not cacheable by the CPU/82495XP/82490XP as indicated by the PALLC# output from the 82495XP.

CACHE CONTROL SIGNALS:

The CPU initiates the first write cycle to the 82495XP/82490XP cache where the cache tag state is looked up. Once the 82495XP determines the cycle to be a cache miss. It issues a cycle request (CADS# in clock 1) and the associated cycle control signals to the MBC (eg. CW/R#, CM/IO#, CD/C#, RDYSRC, MCACHE#, PALLC#) in order to schedule the write miss operation. RDYSRC is not active, indicating that the 82495XP will supply BRDY# to the CPU; MCACHE# is not active; PALLC# is active, indicating that the cycle is potentially allocatable.

The write miss data is posted in the 82490XP's memory cycle buffer, and the cycle completes with no wait states to the CPU. The CPU is then free to issue another (non-related) cycle while the 82495XP completes the current write miss cycle and possible allocation. If this new cycle is a cache hit, it will be serviced by the 82495XP immediately; but if it is a cache miss, its service will wait until the CRDY# of the write cycle (and allocation cycle, if executed).

The memory bus address (MSET[10:0], MTAG[11:0], MCFA[6:0]) is valid with CADS# (clocks 1 and 7 for the two cycles in this example) and remain valid until after CNA# is sampled active by the 82495XP (clocks 4 and 10). MALE and MBALE may be used to hold the address as necessary.

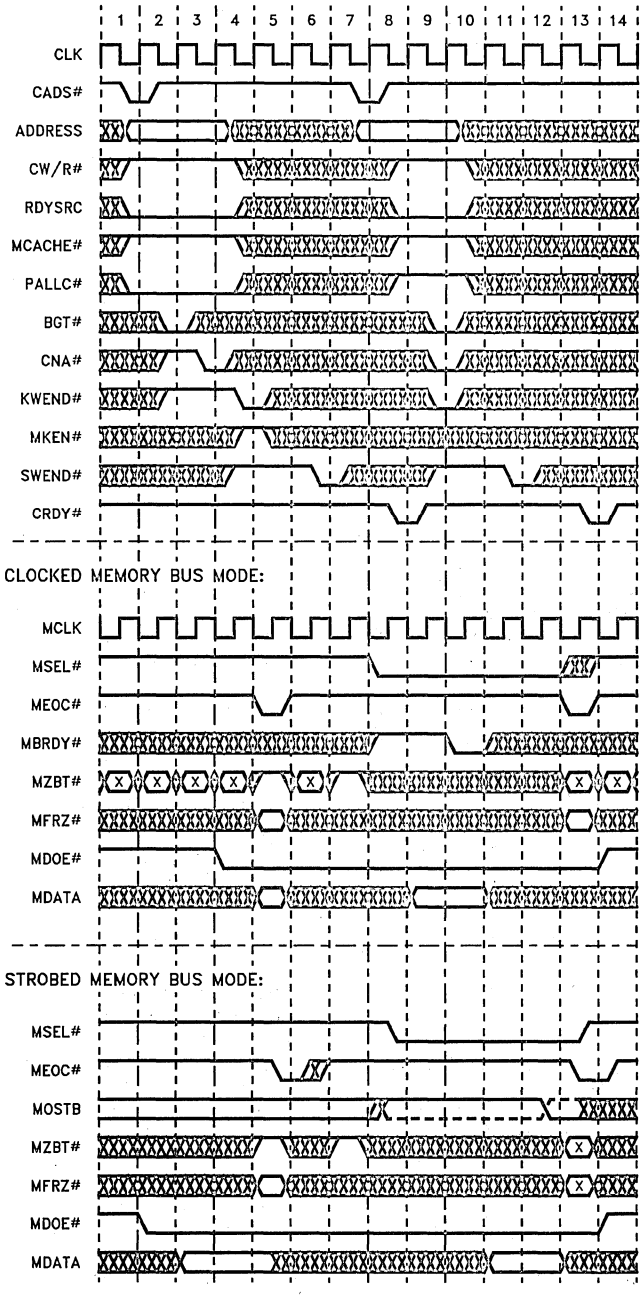


Figure 8-5. Write Miss with No Allocation

The MBC arbitrates for the memory bus and returns BGT# asserted (clock 2), indicating that the write through cycle is guaranteed to complete on the memory bus. Once the 82495XP samples BGT# asserted, it must finish that cycle on the memory bus. Prior to this point, the cycle can be aborted by a snoop hit from another cache.

CNA# is asserted by the MBC (clock 3) to indicate that it is ready to schedule a new memory bus cycle. Notice that the cycle control signals are not guaranteed to be valid after CNA# activation. NOTE that CNA# has no effect before KWEND#.

When the MBC has determined the cacheability attribute of the write through cycle, it drives the MKEN# signal accordingly. The MBC also drives the KWEND# signal at this time (clock 4), indicating the end of the cacheability window. The 82495XP samples MKEN# inactive during KWEND#, indicating that the missed cycle is not cacheable and should not be allocated.

The MBC asserts SWEND# (clock 6) when the snoop window of the write through cycle ends on the memory bus. The MBC may return CRDY# to the 82495XP/82490XP cache any time after the closure of the snoop window. In this example, CRDY# is issued by the MBC in clock 8.

The 82495XP issues a cycle request for the second write miss cycle in clock 7. The cycle control signals are valid at this time. Note that PALLC# is inactive, indicating that the 82495XP/82490XP has determined the cycle to not be allocatable.

The MBC# asserts BGT#, CNA#, and KWEND# in clock 9. MKEN# is a don't care during the cacheability window since the cycle is not allocatable. The snoop window is closed in clock 11, and the cycle is completed on the memory bus in clock 13 with the assertion of CRDY# by the MBC.

MEMORY BUS SIGNALS:

The memory address latch enables (MALE and MBALE) may remain asserted by the MBC to place the address latches in flow through mode. If the 82495XP is the current bus master, the memory address output enables (MAOE# and MBAOE#) should be asserted by the MBC.

For Clocked Memory Bus Mode, the memory data output enable (MDOE#) is asserted by the MBC in clock 4 to drive the memory data outputs.

MEOC# is asserted by the MBC (clock 5) to latch MZBT# for the transfer, and end the current cycle on the memory bus (MBRDY# is not necessary since this example shows a single transfer cycle). MZBT# is driven high by the MBC in order to force the read cycle to begin with the correct burst address. MFRZ# is sampled here (it need not be active since the cycle is not potentially allocatable).

For the second non allocatable write cycle, MSEL# is driven active by the MBC (clock 8) to allow sampling of MBRDY# and to latch MZBT# for the transfer. MZBT# is sampled on all MCLK edges where MSEL# is inactive. Once MSEL# is sampled active by the 82495XP, the value of MZBT# sampled on the prior MCLK is used for the next transfer. Again, MZBT# is driven high by the MBC to force the transfer to begin with the correct burst address. MBRDY# is driven active by the MBC in clock 10 to cause the memory burst counter to be incremented and data to be placed into the 82490XP cache memory cycle buffers.

The MBC drives MEOC# asserted (clock 13) to end the current cycle on the memory bus and switch memory cycle buffers for the new cycle. MFRZ# is sampled here (it need not be active since the cycle is not potentially allocatable). MZBT# is also sampled at this time.

For Strobed Memory Bus Mode, the memory data output enable (MDOE#) is asserted by the MBC in clock 2 to drive the memory data outputs.

MEOC# is driven active by the MBC (clock 5) to latch MZBT# for the transfer, and end the current cycle on the memory bus (MOSTB is not necessary since this example shows a single transfer cycle). MZBT# is driven high by the MBC in order to force the read cycle to begin with the correct burst address.

For the second write through cycle, MSEL# is driven active by the MBC (clock 8) to allow MOSTB operation and to latch MZBT# for the transfer. Again, MZBT# is driven high by the MBC to force the transfer to begin with the correct burst address. MOSTB is toggled in clock 12 to cause the memory burst counter to be incremented, and data to be read from the 82490XP cache memory cycle buffers. Note: MOSTB latches the memory bus data on both the rising and falling edges.

The MBC drives MEOC# asserted (clock 13) to end the current cycle on the memory bus and switch memory cycle buffers for the new cycle. MZBT# and MFRZ# for the next cycle (not shown), is sampled at this time on the falling edge of MEOC#.

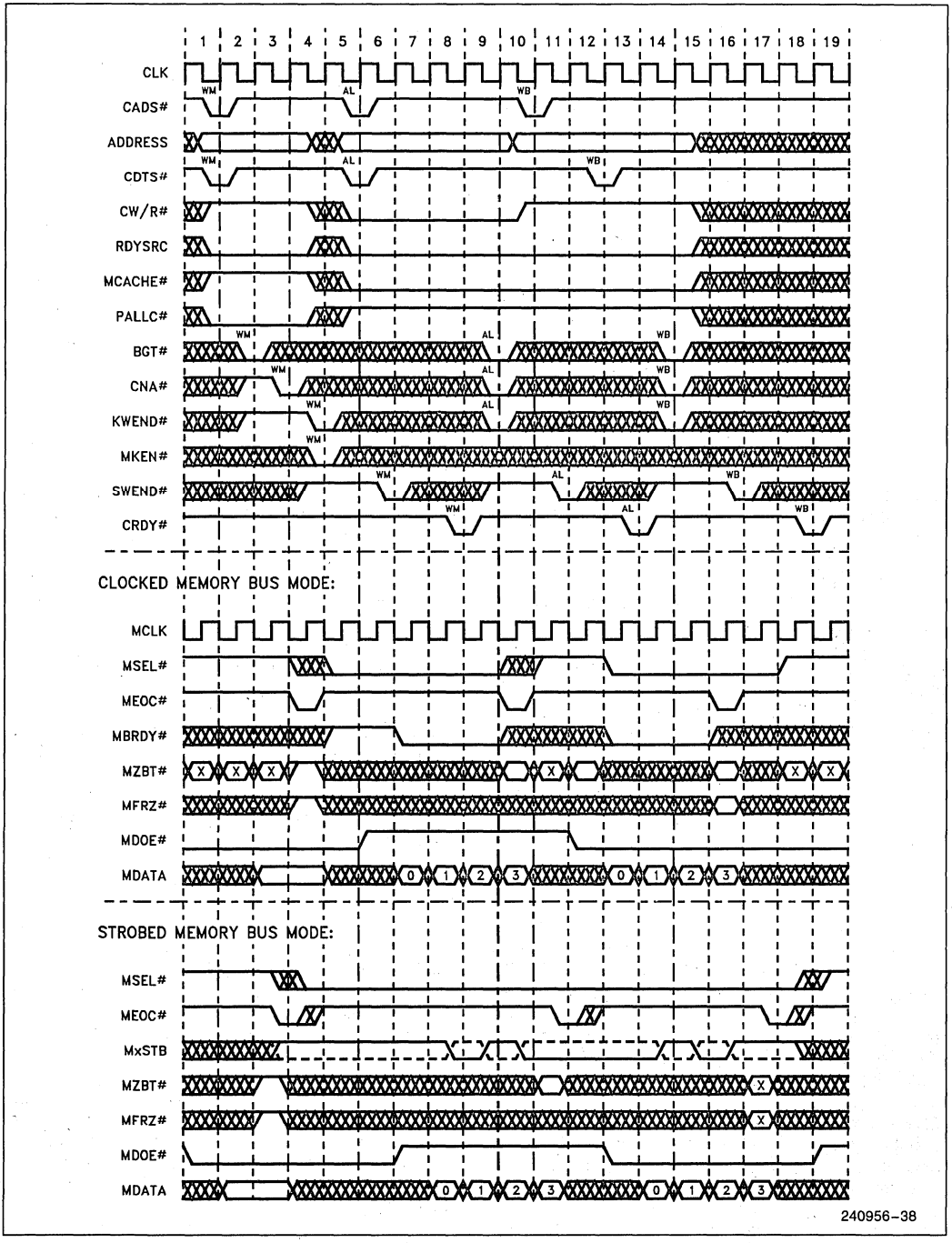


Figure 8-6. Write Miss with Allocation to [M] Line

8.2.2.2 Write Miss with Allocation

Figure 8.6 illustrates a CPU initiated write cycle which misses the 82495XP/82490XP cache and follows the write to main memory with an allocation cycle. An allocation is when the cache follows a write miss cycle with a line fill. This example assumes that allocating the new line requires the replacement of a modified line (ie. a write-back to main memory).

CACHE CONTROL SIGNALS:

The CPU initiates the write cycle to the 82495XP/82490XP cache where the cache tag state is looked up. Once the 82495XP determines the cycle to be a cache miss, it issues CADS# (clock 1) and the associated cycle control signals to the MBC (eg. CW/R#, CM/IO#, CD/C#, RDYSRC, MCACHE#, PALLC#) in order to schedule the write operation. MCACHE# is not active; RDYSRC is not active, indicating that the 82495XP will supply BRDY#s to the CPU; PALLC# is asserted, indicating a potential allocate cycle after the write-through cycle.

The write miss data is posted in the 82490XP's memory cycle buffer, and the cycle completes with no wait states to the CPU. The CPU is free to issue another (non-related) cycle while waiting for the 82495XP to complete the allocation. If this new cycle is a cache hit, it will be serviced by the 82495XP immediately; but if it is a cache miss, its service will wait until the CRDY# of the allocation.

The memory bus address (MSET[10:0], MTAG[11:0], MCFA[6:0]) is valid with CADS# (clocks 1, 5 and 10 for the three cycles in this example) and remain valid until after CNA# is sampled active by the 82495XP (clocks 4, 10 and 15). MALE and MBALE may be used to hold the address as necessary.

The MBC arbitrates for the memory bus and returns BGT# asserted (clock 2), indicating that the write through cycle is guaranteed to complete on the memory bus. Once the 82495XP samples BGT# asserted, it must finish that cycle on the memory bus. Prior to this point, the cycle can be aborted by a snoop hit from another cache.

CNA# is asserted by the MBC (clock 3) to indicate that it is ready to schedule a new memory bus cycle. Note that after CNA# activation, cycle control signals are not guaranteed to be valid.

When the MBC has determined the cacheability attribute of the write through cycle, it drives the MKEN# signal accordingly. The MBC also drives the KWEND# signal at this time, indicating the end of the cacheability window. The 82495XP samples MKEN# active during KWEND# (clock 4), indicating that the missed line should be allocated in the cache.

At the first available time (clock 5), the 82495XP asserts CADS# to request an allocation cycle. The cycle control signals are valid at this point: MCACHE# is active, indicating the cacheability of the line-fill cycle; RDYSRC is not active, indicating that the MBC need not supply BRDY#s to the CPU (no BRDY#s are necessary for an allocation cycle).

The MBC asserts SWEND# (clock 6) when the snoop window of the write through cycle ends on the memory bus.

The MBC may return CRDY# to the 82495XP/82490XP cache any time after the closure of the snoop window. In this example, CRDY# is issued by the MBC in clock 8. At this time, the cycle progress signals for the allocation cycle may be issued by the MBC to complete the line fill.

Once again, the MBC arbitrates for the memory bus and returns BGT# asserted (clock 9) for the allocation cycle. The MBC also asserts CNA# and KWEND# at this time. The 82495XP back-invalidates the CPU to maintain first and second level cache consistency.

In clock 10, the 82495XP asserts CADS# for the write back cycle (since the miss was to a dirty line). CDTS# is asserted by the 82495XP two clocks later (clock 12). Note that CDTS# of the write back cycle is not asserted with CADS# since the data is not yet available in the 82490XP's write-back buffer.

The MBC asserts SWEND# (clock 11) when the snoop window of the allocation cycle end on the memory bus.

At this time, the MBC may assert CRDY# to the 82495XP/82490XP cache for the allocation cycle. CRDY# assertion will cause the data stored in the 82490XP's memory cycle buffers to be latched into the cache array.

On the memory bus, BGT#, CNA#, and KWEND# are sampled active in clock 14 for the write back cycle. The snoop window is closed two clocks later (clock 16) by the MBC with SWEND#, and the write back cycle is completed with CRDY# asserted in clock 18.

MEMORY BUS SIGNALS:

The memory address latch enables (MALE and MBALE) may remain asserted by the MBC to place the address latches in flow through mode. If the 82495XP is the current bus master, the memory address output enables (MAOE# and MBAOE#) should be asserted by the MBC.

For Clocked Memory Bus Mode, the memory data output enable (MDOE#) has been asserted by the MBC to drive the memory data outputs.

MEOC# is asserted by the MBC (clock 4) to latch MZBT# for the transfer, and end the current cycle on the memory bus (MBRDY# is not necessary since this example shows a single transfer write miss cycle). MZBT# is driven high by the MBC in order to force the read cycle to begin with the correct burst address. MFRZ# is driven inactive by the MBC here, allowing the line to be placed into the exclusive ([E]) state and requiring the data to be written to main memory.

For the allocation (line fill) cycle, MSEL# is driven active by the MBC (clock 6) to allow sampling of MBRDY# and to latch MZBT# for the transfer. MZBT# is sampled on all MCLK edges where MSEL# is inactive. Once MSEL# is sampled active by the 82495XP, the value of MZBT# sampled on the prior MCLK is used for the next transfer. MDOE# is also deasserted in clock 6 to allow the data pins to be used as inputs for the allocation cycle.

MBRDY# is driven active by the MBC in clocks 7 to 9 to cause the memory burst counter to be incremented and data to be placed into the 82490XP cache memory cycle buffers. The MBC drives MEOC# asserted (clock 10) to end the allocation cycle on the memory bus and switch memory cycle buffers for the new cycle. MZBT# is sampled and latched at this time for the next data transfer.

MDOE# is asserted by the MBC (clock 12) to drive the memory data outputs for the write back cycle.

The MBC again asserts MBRDY# (clocks 13 to 15) for the write back cycle to increment the memory burst counter and cause data to be read from the 82490XP memory cycle buffers. The write back cycle ends on the memory bus and switches memory cycle buffers with MEOC# assertion (clock 16). MZBT# and MFRZ# for the next transfer are sampled at this time. MFRZ# need not be active since the cycle is not potentially allocatable.

For Strobed Memory Bus Mode, the memory data output enable (MDOE#) has been asserted by the MBC to drive the memory data outputs for the write miss cycle.

MEOC# is driven active by the MBC (clock 4) to latch MZBT# for the transfer, and end the current cycle on the memory bus (MOSTB is not necessary since this example shows a single transfer cycle). MZBT# is driven high by the MBC in order to force the read cycle to begin with the correct burst address. MFRZ# is driven deasserted by the MBC here, allowing the line to be placed into the exclusive ([E]) state.

For the allocation (line fill) cycle, MSEL# is driven active by the MBC (clock 6) to allow MISTB operation and to latch MZBT# for the transfer. MISTB is toggled in clocks 8 to 10 to cause the memory burst counter to be incremented, and data to be placed into the 82490XP cache memory cycle buffers. Note: MISTB latches the memory bus data on both the rising and falling edges. MDOE# is also deasserted in clock 6 to allow the data pins to be used as inputs for the allocation cycle.

The MBC drives MEOC# asserted (clock 11) to end the allocation cycle on the memory bus and switch memory cycle buffers for the new cycle. MZBT# for the next cycle, is latched at this time on the falling edge of MEOC#.

MDOE# is asserted by the MBC (clock 14) to drive the memory data outputs for the write back cycle.

The MBC toggles MOSTB (clocks 15 to 17) for the write back cycle to increment the memory burst counter and cause data to be read from the 82490XP memory cycle buffers.

The write back cycle ends on the memory bus and switches memory cycle buffers with MEOC# assertion (clock 18). MZBT# and MFRZ# for the next transfer are sampled at this time. MFRZ# need not be active since the cycle is not potentially allocatable.

8.3 Snooping Cycles

8.3.1 SYNCHRONOUS SNOOPING MODE (HIT TO [M] LINE)

Figure 8.7 illustrates a snoop hit to a dirty line sequence occurring simultaneously with a CPU initiated read miss cycle. This example assumes synchronous snooping mode (ie. requests for snoops are done via SNPSTB# from the MBC, sampled on the 82495XP's CLK).

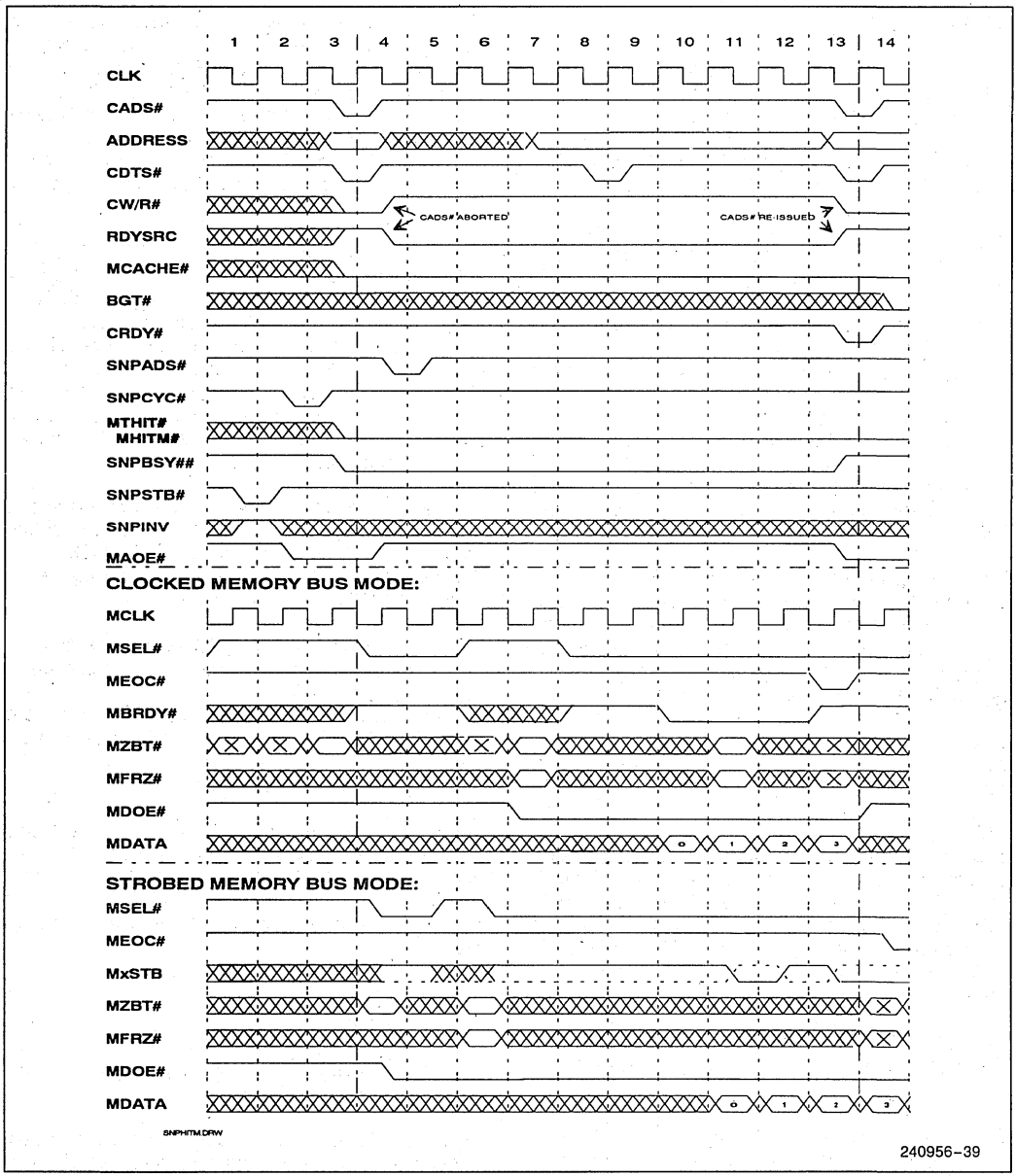


Figure 8-7. Synchronous Snooping Mode

CACHE CONTROL SIGNALS:

In clock 1 SNPSTB# is asserted by the MBC, indicating to the 82495XP a request for snooping. The 82495XP samples MAOE# (it must be inactive) in order to recognize the snoop request. It is latched together with the snoop address (MSET[0:10], MTAG[0:11], MCFA[0:6]), SNPINV, MBAOE#, and SNPNCA on the 82495XP's CLK during SNPSTB# assertion. The tag look-up is done immediately after SNPSTB# is sampled active since snoop operations have the highest priority in the cache tag state arbiter. The 82495XP issues SNPCYC# (clock 2), indicating that the snoop look-up is in progress. The results of the look-up are driven to the memory bus via MTHIT# and MHITM# in the next clock after SNPCYC#. Since the snoop hit a modified line, both signals are asserted (clock 3). SNPBSY# is also issued to indicate that the 82495XP is busy with CPU back-invalidations, the 82490XP's snoop buffer is full, or a write back is to follow. The 82495XP will accept snoops only when SNPBSY# is inactive.

Simultaneously with the memory bus activity due to the snoop request, the CPU initiates a read miss cycle. The 82495XP issues a memory bus request (CADS#), CDTS#, and cycle control signals to the MBC in clock 3. The MBC must wait for the pending snoop cycle to complete on the memory bus prior to servicing this read miss cycle.

The memory bus address (MSET[10:0], MTAG[11:0], MCFA[6:0]) is not valid until MAOE# goes active after CRDY# of the snoop write back cycle is sampled active by the 82495XP and the CADS# is reissued (clock 13).

In clock 4 the 82495XP issues SNPADS# and cycle control signals to the MBC, indicating a request to flush a modified line out of the cache. SNPADS# activation causes the MBC to abort the pending read miss cycle. It is the 82495XP responsibility to re-issue the aborted cycle after the completion of the write back, since BGT# was not asserted by the MBC.

Data is loaded into the 82490XP's snoop buffer. Since SNPINV was sampled asserted by the 82495XP (clock 1) during SNPSTB# assertion, it back-invalidated the CPUs first level cache.

The 82495XP asserts CDTS# (clock 8) indicating to the MBC that data is available in the snoop buffer. When the MBC complete the write back cycle on the memory bus, it activates CRDY# to the 82495XP/82490XP cache. At this time, the 82495XP deasserts SNPBSY# (clock 13) and re-issues the aborted read miss cycle (clock 13) by asserting CADS# and CDTS#.

MEMORY BUS SIGNALS:

For Clocked Memory Bus Mode, the memory data output enable (MDOE#) is not activated by the MBC to allow the memory data pins to be used as inputs.

MSEL# is driven active by the MBC (clock 4) to allow sampling of MBRDY# and to latch MZBT# for the read miss transfer. MZBT# is sampled on all MCLK rising edges where MSEL# is inactive. Once MSEL# is sampled active by the 82495XP, the value of MZBT# sampled on the prior MCLK is used for the next transfer.

Since the read miss cycle is aborted due to the snoop hit to a modified line (requires a write back cycle), no MEOC# is given. MSEL# is deasserted by the MBC (clock 6) and reasserted (clock 8) to allow latching of MZBT# for the snoop write back cycle and sampling of MBRDY# for that cycle. MFRZ# is also sampled at this time.

The memory data output enable (MDOE#) signal is driven active by the MBC (clock 7) to drive the memory data outputs.

MBRDY# is driven active by the MBC in clocks 10 to 12 to cause the memory burst counter to be incremented and data to be written from the 82490XP cache snoop buffers. The MBC drives MEOC# asserted (clock 13) to end the write back cycle on the memory bus and switch memory cycle buffers for the new cycle. MZBT# and MFRZ# are sampled and latched at this time for the next data transfer.

MDOE# is deasserted by the MBC (clock 14) to allow the memory data pins to be used as inputs for the reissued read cycle.

For Strobed Memory Bus Mode, the memory data output enable (MDOE#) has not been asserted by the MBC to allow the memory data pins to be used as inputs for the read miss cycle.

MSEL# is asserted by the MBC (clock 4) to allow sampling of MISTB and latch MZBT# (on the falling edge of MSEL#) for the read miss transfer.

Since the read miss cycle is aborted due to the snoop hit to a modified line (requires a write back cycle), no MEOC# is given. MSEL# is deasserted by the MBC (clock 5) and reasserted (clock 6) to allow latching of MZBT# for the snoop write back cycle and sampling of MOSTB for that cycle. MFRZ# is also sampled at this time.

MOSTB is toggled in clocks 11 to 13 to cause the memory burst counter to be incremented, and data

to be read from the 82490XP cache memory cycle buffers. Note: MOSTB latches the memory bus data on both the rising and falling edges. The MBC drives MEOC# asserted (clock 14) to end the snoop write back cycle on the memory bus and switch memory cycle buffers for the new cycle. MZBT# and MFRZ# for the next cycle, are latched at this time on the falling edge of MEOC#.

MDOE# is deasserted by the MBC (clock 14) to allow the memory data pins to be used as inputs for the reissued read miss cycle.

8.3.2 CLOCKED SNOOPING MODE

Figure 8.8 illustrates a CPU initiated Read cycle which misses the 82495XP/82490XP cache and the subsequent line fill replaces non dirty data (eg. clean or empty). Simultaneous with the read request to the MBC, that device initiates a snoop to the 82495XP which misses that line in the cache. The snoop is the result of a write cycle on the memory bus by some other cache core; therefore, asserting the snoop invalidation signal (SNPINV) to this 82495XP. This example assumes Clocked Snooping Mode (i.e. the requests for snoops are done via SNPSTB# from the MBC, sampled on the MBC's SNPCLK).

CACHE CONTROL SIGNALS:

The CPU initiates the read cycle to the 82495XP/82490XP cache where the cache tag state is looked up. Once the 82495XP determines the cycle to be a cache miss, it issues CADS# (clock 1) and the associated cycle control signals to the MBC (eg. CW/R#, CM/IO#, CD/C#, RDYSRC, MCACHE#) in order to schedule the cache line-fill operation. MCACHE# is active, indicating that the read miss in potentially cacheable by the 82495XP; RDYSRC is active, indicating that the MBC must supply BRDY#s to the CPU cache core.

In clock 3, SNPSTB# is asserted by the MBC at this time, indicating to the 82495XP a request for snooping. MAOE# is deasserted to allow the forthcoming snoop (the 82495XP will not recognize the snoop if MAOE# is active). It is latched together with the snoop address (MSET[0:10], MTAG[0:11], MCFA[0:6]), SNPINV, MBAOE#, and SNPNCA on the MBC's SNPCLK rising edge during SNPSTB# assertion. SNPINV is asserted from the MBC since the cache core which initiated the snoop issued a write cycle on the memory bus. If the response of the snoop to this 82495XP was a cache hit, the contents would no longer be valid due that write.

Following synchronization to the 82495XP CLK, it issues SNPCYC# (clock 5), indicating that the snoop look-up is in progress. The results of the look-up are driven to the memory bus via MTHIT# and MHITM# in the next clock after SNPCYC#. Since the snoop was a miss in the cache, both signals are inactive (clock 6). Note that SNPBSY# will not be asserted since the snoop was a miss to this cache. The snoop from another cache is complete at this point, and the read miss cycle will continue.

The MBC asserts MAOE# to allow this 82495XP to drive its address on the memory bus in order to complete the read miss cycle. The memory bus address (MSET[10:0], MTAG[11:0], MCFA[6:0]) is valid after MAOE# assertion # (clock 6 for the read cycle in this example) and remains valid until after CNA# is sampled active by the 82495XP (clock 8). MALE and MBALE may be used to hold the address as necessary.

The MBC arbitrates for the memory bus and returns BGT# asserted (clock 6), indicating that the cycle is guaranteed to complete on the memory bus. Once the 82495XP samples BGT# asserted, it must finish that cycle on the memory bus. Prior to this point, the cycle can be aborted by a snoop hit from another cache.

CNA# is asserted by the MBC (clock 7) to indicate that it is ready to schedule a new memory bus cycle. Note that after CNA# activation, cycle control signals are not guaranteed to be valid.

When the MBC has determined the cacheability attribute of the cycle, it drives the MKEN# signal accordingly. The MBC also drives the KWEND# signal at this time, indicating the end of the cacheability window. The 82495XP samples MKEN# during KWEND# (clock 7) to determine that the cycle is indeed cacheable.

The MBC asserts SWEND# when the snoop window ends on the memory bus. The 82495XP samples MWB/WT# during SWEND# (clock 9) and updates the cache tag state according to the consistency protocol. The closure of the snoop window also enables the MBC to start providing the CPU with data that has been stored in the 82490XP's memory cycle buffer. The MBC supplies BRDY#s to the CPU (clocks 9-12).

The read miss cycle ends when CRDY# is driven active by the MBC (clock 12). It is at this time that the data in the 82490XP's memory cycle buffers is loaded into the cache SRAM.

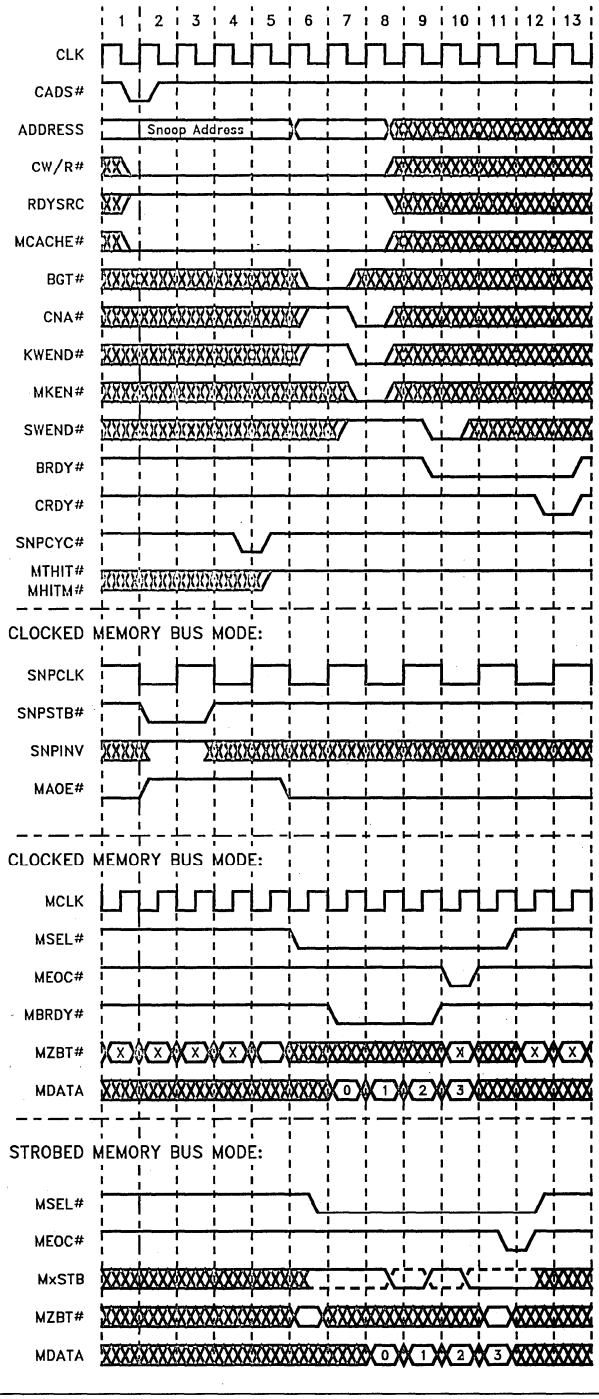


Figure 8-8. Clocked Snooping Mode

MEMORY BUS SIGNALS:

The memory address latch enables (MALE and MBALE) may remain asserted by the MBC to place the address latches in flow through mode. If the 82495XP is the current bus master, the memory address output enables (MAOE# and MBAOE#) should be asserted by the MBC. (Note the use of MAOE# for snooping at the beginning of the cache control signals section.) MDOE# must be inactive to allow the data pins to be used as inputs.

Some time after the address has been driven onto the memory bus, data will be supplied from the DRAM (main memory) to the 82490XP cache SRAM.

For Clocked Memory Bus Mode, MSEL# is driven active by the MBC (clock 6) to allow sampling of MBRDY# and to latch MZBT# for the transfer. MZBT# is sampled on all MCLK edges where MSEL# is inactive. Once MSEL# is sampled active by the 82495XP, the value of MZBT# sampled on the prior MCLK is used for the next transfer. MBRDY# is driven active by the MBC in clocks 7 to 9 to cause the memory burst counter to be incremented and data to be placed into the 82490XP cache memory cycle buffers. The MBC drives MEOC# asserted (clock 10) to end the current cycle on the memory bus and switch memory cycle buffers for the new cycle. MZBT# is sampled at this time (when MEOC# is sampled asserted and MSEL# remains low) for the next transfer.

For Strobed Memory Bus Mode, MSEL# is driven active by the MBC (clock 6) to allow MISTB operation and to latch MZBT# (on the falling edge of MSEL#) for the transfer. MISTB is toggled in clocks 8 to 10 to cause the memory burst counter to be incremented, and data to be placed into the 82490XP cache memory cycle buffers. Note: MISTB latches the memory bus data on both the rising and falling edges. The MBC drives MEOC# asserted (clock 11) to end the current cycle on the memory bus and switch memory cycle buffers for the new cycle. MZBT# for the next cycle, is sampled at this time on the falling edge of MEOC#.

8.3.3 STROBED SNOOPING MODE (HIT TO [M] LINE)

Figure 8.9 illustrates a snoop hit to a dirty line sequence occurring simultaneously with a CPU initiated read miss cycle. This example assumes strobed snooping mode (ie. requests for snoops are done from the falling edge of SNPSTB#).

CACHE CONTROL SIGNALS:

In clock 1 (totally asynchronous to any clock) SNPSTB# is asserted by the MBC, indicating to the 82495XP a request for snooping. The 82495XP samples MAOE# (it must be inactive) in order to recognize the snoop request. It is latched together with the snoop address (MSET[0:10], MTAG[0:11], MCFA[0:6]), SNPINV, MBAOE#, and SNPNCA on falling edge of SNPSTB#. The 82495XP issues SNPCYC# (clock 3), indicating that the snoop look-up is in progress. The results of the look-up are driven to the memory bus via MTHIT# and MHITM# in the next clock after SNPCYC#. Since the snoop hit a modified line, both signals are asserted (clock 4). SNPBSY# is also issued to indicate that the 82495XP is busy with CPU back-invalidations, the 82490XP's snoop buffer is full, or a write back is to follow. The 82495XP will accept snoops only when SNPBSY# is inactive.

Simultaneously with the memory bus activity due to the snoop request, the CPU initiates a read miss cycle. The 82495XP issues a memory bus request (CADS#), CDTS#, and cycle control signals to the MBC in clock 1. The MBC must wait for the pending snoop cycle to complete on the memory bus prior to servicing this read miss cycle.

The memory bus address (MSET[10:0], MTAG[11:0], MCFA[6:0]) is not valid until MAOE# goes active after CRDY# of the snoop write back cycle is sampled active by the 82495XP and the CADS# is reissued (clock 15).

In clock 5 the 82495XP issues SNPADS# and cycle control signals to the MBC, indicating a request to flush a modified line out of the cache. SNPADS# activation causes the MBC to abort the pending read miss cycle. It is the 82495XP responsibility to re-issue the aborted cycle after the completion of the write back, since BGT# was not asserted by the MBC.

Data is loaded into the 82490XP's snoop buffer. Since SNPINV was sampled asserted by the 82495XP (clock 1) during SNPSTB# assertion, it back-invalidated the CPUs first level cache.

The 82495XP asserts CDTS# (clock 9) indicating to the MBC that data is available in the snoop buffer. When the MBC complete the write back cycle on the memory bus, it activates CRDY# to the 82495XP/82490XP cache. At this time, the 82495XP deasserts SNPBSY# (clock 15) and re-issues the aborted read miss cycle by asserting CADS# and CDTS#.

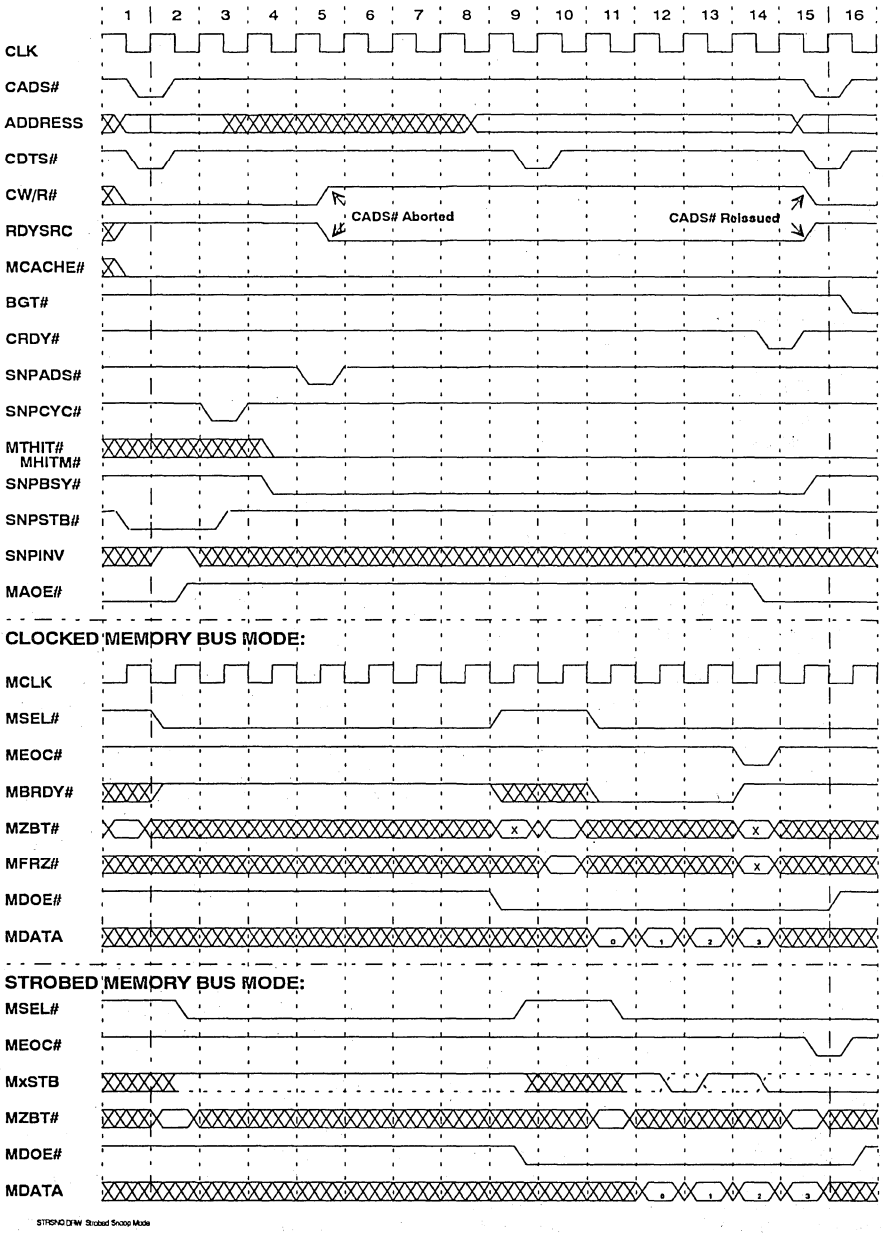


Figure 8-9. Strobed Snooping Mode

MEMORY BUS SIGNALS:

For Clocked Memory Bus Mode, the memory data output enable (MDOE#) is not activated by the MBC to allow the memory data pins to be used as inputs.

MSEL# is driven active by the MBC (clock 2) to allow sampling of MBRDY# and to latch MZBT# for the read miss transfer. MZBT# is sampled on all MCLK rising edges where MSEL# is inactive. Once MSEL# is sampled active by the 82495XP, the value of MZBT# sampled on the prior MCLK is used for the next transfer.

Since the read miss cycle is aborted due to the snoop hit to a modified line (requires a write back cycle), no MEOC# is given. MSEL# is deasserted by the MBC (clock 9) and reasserted (clock 11) to allow latching of MZBT# for the snoop write back cycle and sampling of MBRDY# for that cycle. MFRZ# is also sampled at this time.

The memory data output enable (MDOE#) signal is driven active by the MBC (clock 9) to drive the memory data outputs.

MBRDY# is driven active by the MBC in clocks 11 to 13 to cause the memory burst counter to be incremented and data to be written from the 82490XP cache memory cycle buffers. The MBC drives MEOC# asserted (clock 14) to end the write back cycle on the memory bus and switch memory cycle buffers for the new cycle. MZBT# and MFRZ# are sampled and sampled at this time for the next data transfer.

MDOE# is deasserted by the MBC (clock 16) to allow the memory data pins to be used as inputs for the reissued read cycle.

For Strobed Memory Bus Mode, the memory data output enable (MDOE#) has not been asserted by the MBC to allow the memory data pins to be used as inputs for the read miss cycle.

MSEL# is asserted by the MBC (clock 2) to allow sampling of MISTB and latch MZBT# (on the falling edge of MSEL#) for the read miss transfer.

Since the read miss cycle is aborted due to the snoop hit to a modified line (requires a write back cycle), no MEOC# is given. MSEL# is deasserted by the MBC (clock 9) and reasserted (clock 11) to allow latching of MZBT# for the snoop write back cycle and sampling of MOSTB for that cycle. MFRZ# is also sampled at this time.

MOSTB is toggled in clocks 12 to 14 to cause the memory burst counter to be incremented, and data to be read from the 82490XP cache memory cycle buffers. Note: MOSTB latches the memory bus data on both the rising and falling edges.

The MBC drives MEOC# asserted (clock 15) to end the snoop write back cycle on the memory bus and switch memory cycle buffers for the new cycle. MZBT# and MFRZ# for the next cycle, are sampled at this time on the falling edge of MEOC#.

MDOE# is deasserted by the MBC (clock 16) to allow the memory data pins to be used as inputs for the reissued read miss cycle.

8.3.4 CACHE TO CACHE TRANSFER

8.3.4.1 Read Cycles Causing a Snoop Hit to [M] Line

Figure 8.10 illustrates CPU initiated Read cycles that miss the 82495XP/82490XP cache and replace a non-dirty (eg. clean) line in the cache. During the snoop window, the memory bus attribute which indicates a direct to [M] state transfer is sampled active. In such cycles, the 82495XP will instruct the MBC to perform a cache line-fill cycle on the memory bus. The request for data will not go to main memory, but instead will go to the controller of the cache which contained the modified data. The line is then written into the 82490XP's array, and data transferred to the CPU as requested. If the line fetched from the second cache replaces a line which is in valid unmodified state ([E] or [S]), then a back-invalidation cycle is performed on the CPU bus to guarantee that the replaced data is also removed from the CPU's first level cache, thus maintaining the inclusion property.

CACHE CONTROL SIGNALS:

The CPU initiates the read cycle to the 82495XP/82490XP cache where the cache tag state is looked up. Once the 82495XP determines the cycle to be a cache miss, it issues CADS# (clock 2) and the associated cycle control signals to the MBC (eg. CW/R#, CM/IO#, CD/C#, RDYSRC, MCACHE#) in order to schedule the cache line-fill operation. MCACHE# is active, indicating that the read miss is potentially cacheable by the 82495XP; RDYSRC is active, indicating that the MBC must supply BRDY#s to the CPU cache core.

The memory bus address (MSET[10:0], MTAG[11:0], MCFA[6:0]) is valid with CADS# (clocks 2 and 13 for the two read miss cycles in this example) and remain valid until after CNA# is sampled active by the 82495XP (clocks 5 and 16). MALE and MBALE may be used to hold the address as necessary.

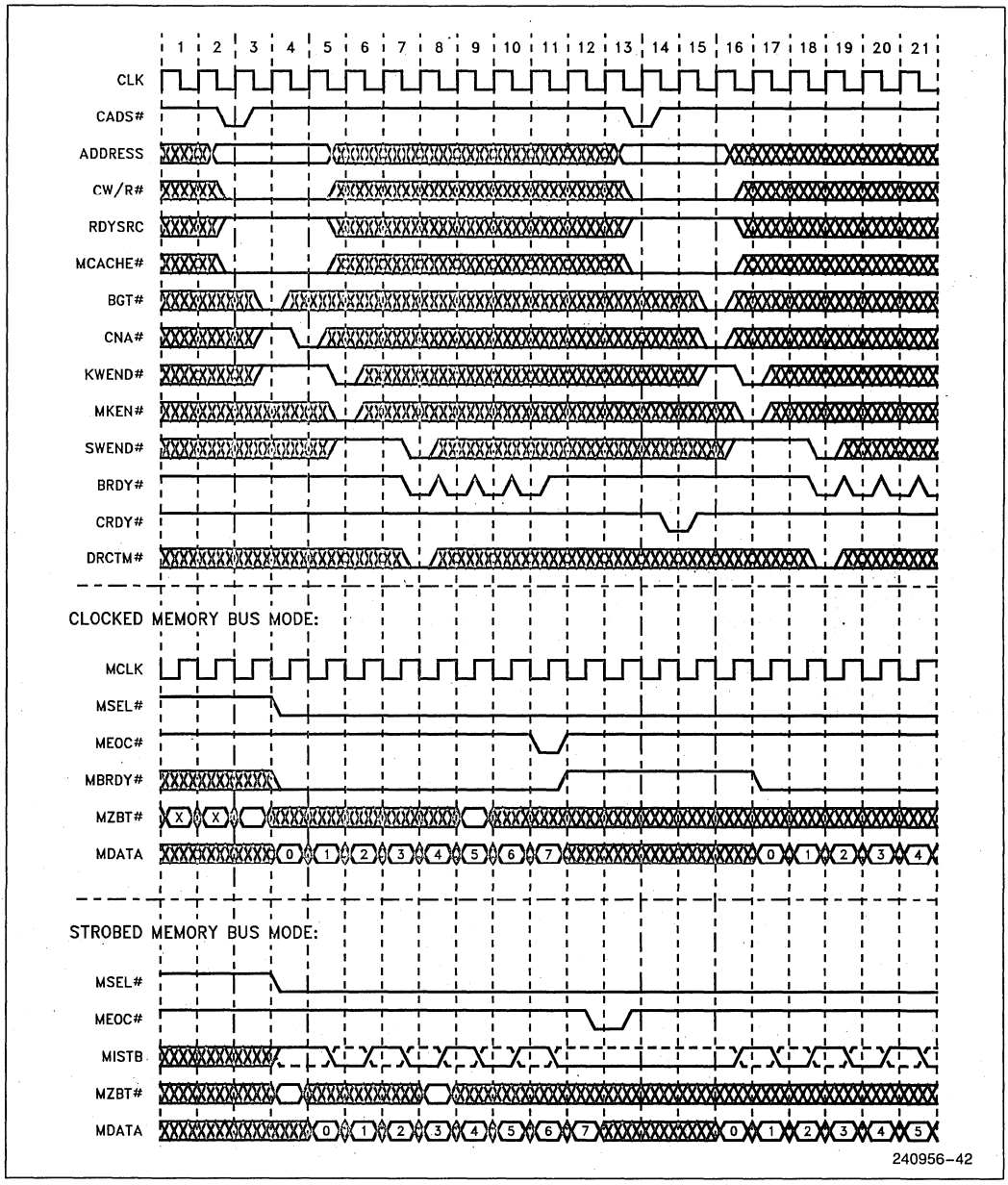


Figure 8-10. Cache to Cache Transfer: Cacheable Read Miss

The MBC arbitrates for the memory bus and returns BGT# asserted (clock 3), indicating that the cycle is guaranteed to complete on the memory bus. Once the 82495XP samples BGT# asserted, it must finish that cycle on the memory bus. Prior to this point, the cycle can be aborted by a snoop hit from another cache.

CNA# is asserted by the MBC (clock 4) to indicate that it is ready to schedule a new memory bus cycle. Note that after CNA# activation, cycle control signals are not guaranteed to be valid.

When the MBC has determined the cacheability attribute of the cycle, it drives the MKEN# signal accordingly. The MBC also drives the KWEND# signal at this time, indicating the end of the cacheability window. The 82495XP samples MKEN# and MRO# during KWEND# (clock 5) to determine that the cycle is indeed cacheable.

The MBC asserts SWEND# when the snoop window ends on the memory bus. The 82495XP samples MWB/WT# and DRCTM# during SWEND# (clock 7) and updates the cache tag state according to the consistency protocol. Since the result of the snoop was a hit to a modified line in another cache, the MBC asserts DRCTM# at this time (this is an option to save time by skipping the main memory access, not a requirement of the memory bus) so that the tag state will go immediately to the [M] state, skipping the [E] state. MWB/WT# must be in write back mode (high) to assure this transition. The closure of the snoop window also enables the MBC to start providing the CPU with data that has been stored in the 82490XP's memory cycle buffer. The MBC supplies BRDY#s to the CPU (clocks 7-10).

The 82495XP issues a new CADS# in clock 13, which also misses the 82495XP/82490XP cache. Since the 82495XP has already sampled CNA# asserted (clock 4), It issues a new CADS# prior to receiving CRDY# of the current cycle (ie. this cycle is pipelined within the MBC). Note that once the cycle progress signals (BGT#, CNA#, KWEND#, SWEND#) of a cycle are sampled asserted, the 82495XP ignores them until the CRDY# of that cycle. The 82495XP does not pipeline the cycle progress signals (ie. it will not sample them again until after CRDY# of the current memory bus cycle).

MEMORY BUS CYCLES:

At the start of this cycle, the master 82495XP does not know that the data will be coming from a slave 82495XP/82490XP and begins a read request to main memory to obtain the required data. Since the snoop resulted in a hit to a modified line in the second cache, the memory request must be backed off so that the snooped 82495XP may supply the data.

The memory address latch enables (MALE and MBALE) may remain asserted by the MBC to place the address latches in flow through mode. If the 82495XP is the current bus master, the memory address output enables (MAOE# and MBAOE#) should be asserted by the MBC. The memory data output enable signal (MDOE#) must remain inactive to allow the data pins to be used as inputs.

For Clocked Memory Bus Mode, MSEL# is driven active by the MBC (clock 4) to allow sampling of MBRDY# and to latch MZBT# for the transfer. MZBT# is sampled on all MCLK edges where MSEL# is inactive. Once MSEL# is sampled active by the 82495XP, the value of MZBT# sampled on the prior MCLK is used for the next transfer.

MBRDY# is driven active in clocks 4 to 10 to read data into the 82490XP cache memory cycle buffers. The MBC asserts MEOC# (clock 11) to end the read miss cycle on the memory bus and switch the memory cycle buffers for a new cycle. MZBT# is latched at this time for the next transfer. Note that there are 8 transfers needed to fill the 82495XP/82490XP cache line and only 4 needed for the CPU line fill.

MBRDY# is again driven active by the MBC in clocks 11 to 21 to cause the memory burst counter to be incremented and data to be placed into the 82490XP cache memory cycle buffers for the second read miss cycle.

For Strobed Memory Bus Mode, MSEL# is driven active by the MBC (clock 4) to allow MISTB operation and to latch MZBT# for the transfer (on the falling edge of MSEL#). MISTB is toggled in clocks 5 to 11 to cause the memory burst counter to be incremented, and data to be placed into the 82490XP cache memory cycle buffers. Note: MISTB latches the memory bus data on both the rising and falling edges. The MBC drives MEOC# asserted (clock 12) to end the current cycle on the memory bus and switch memory cycle buffers for the new cycle. MZBT# for the next cycle is latched at this time on the falling edge of MEOC#.

The MBC toggles MISTB (clocks 16 to 21) for the second read miss cycle to increment the memory burst counter and cause data to be written into the 82490XP memory cycle buffers.

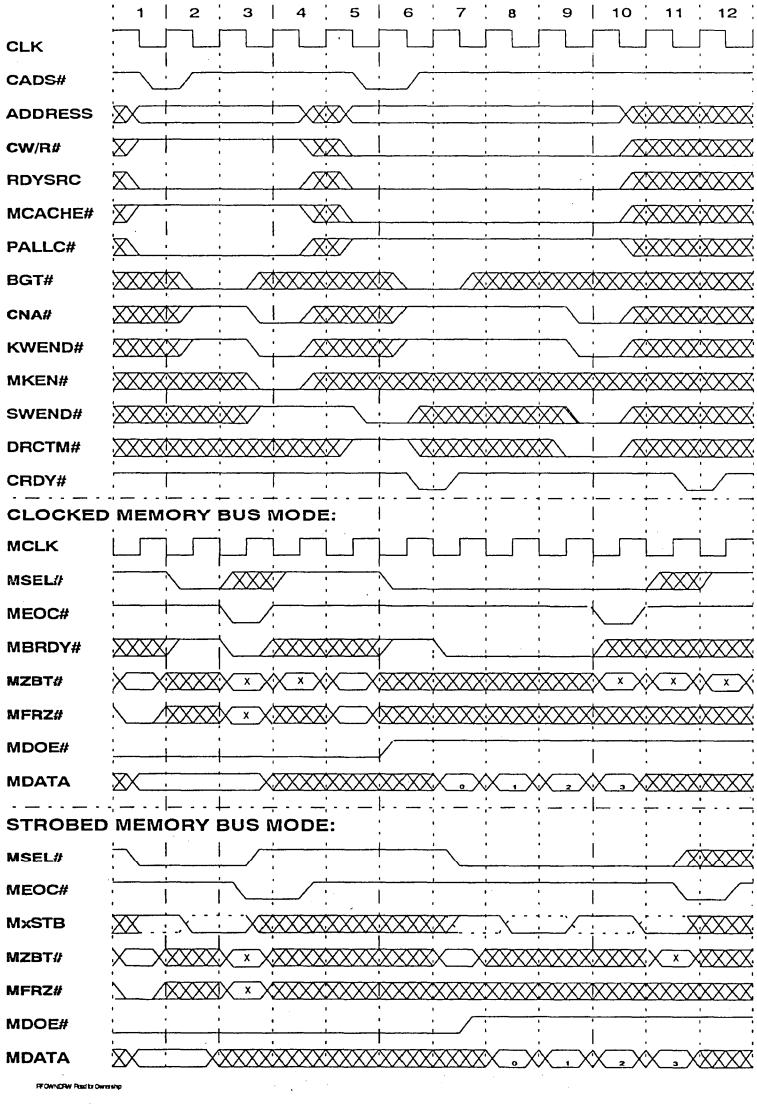


Figure 8-11. Read For Ownership

8.4 Read for Ownership

8.4.1 WRITE MISS WITH MFRZ# ASSERTED, FOLLOWED BY READ TO SAME LINE

Figure 8-11 illustrates a Read For Ownership cycle. First, a CPU initiates a write cycle which misses the 82495XP/82490XP cache. The MBC issues a "dummy" write to main memory (the write does not actually go out to main memory - to save valuable bus time). The 82490XP MFRZ# input is used by the MBC to indicate that the following line-fill (allocation) data (from either main memory or another cache) should be merged with the data of the write miss. The entire line is then placed into the internal tagram.

CACHE CONTROL SIGNALS:

The CPU initiates a write cycle to the 82495XP/82490XP cache where the cache tag state is looked up. Once the 82495XP determines the cycle to be a cache miss, it issues CADS# (clock 1) and the associated cycle control signals to the MBC (eg. CW/R#, CM/IO#, CD/C#, RDYSRC, MCACHE#, PALLC#) in order to schedule the write operation. MCACHE# is not active; RDYSRC is not active, indicating that the 82495XP will supply BRDY#s to the CPU; PALLC# is active, indicating a potential allocate cycle after the write through cycle.

The write miss data is posted in the 82490XP's memory cycle buffer, and the cycle completes with no wait states to the CPU. The CPU is free to issue another (non-related) cycle while the 82495XP is processing the allocation. If this new cycle is a cache hit, it will be serviced by the 82495XP immediately; but if it is a cache miss, its service will wait until the CRDY# of the allocation.

The memory bus address (MSET[10:0], MTAG[11:0], MCFA[6:0]) is valid with CADS# (clocks 1 and 5 for the write miss and allocation cycle in this example) and remain valid until after CNA# is sampled active by the 82495XP (clocks 4 and 10). MALE and MBALE may be used to hold the address as necessary.

The MBC arbitrates for the memory bus and returns BGT# asserted (clock 2), indicating that the write through cycle is guaranteed to complete on the memory bus. Once the 82495XP samples BGT# asserted, it must finish that cycle on the memory bus. Prior to this point, the cycle can be aborted by a snoop hit from another cache.

CNA# is asserted by the MBC (clock 3) to indicate that it is ready to schedule a new memory bus cycle. Note that after CNA# activation, cycle control signals are not guaranteed to be valid.

When the MBC has determined the cacheability attribute of the write through cycle, it drives the MKEN# signal accordingly. The MBC also drives the KWEND# signal at this time, indicating the end of the cacheability window. The 82495XP samples MKEN# active during KWEND# (clock 3), indicating that the missed line should be allocated in the cache.

The MBC asserts SWEND# (clock 5) when the snoop window of the write through cycle ends on the memory bus. Note that the direct to [M] state qualifier signal (DRCTM#) is sampled during SWEND# and is inactive for the write . The MBC also issued CRDY# to the 82495XP at this time so that the 82495XP thinks the write cycle completed on the memory bus when, in fact, it did not.

In this example, the 82495XP requests the allocation cycle by issuing CADS# in clock 5. The cycle control signals are valid at this point: MCACHE# is active, indicating the cacheability of the line-fill cycle; RDYSRC is not active, indicating that the MBC need not supply BRDY#s to the CPU (no BRDY#s are necessary for an allocation cycle).

Once again, the MBC arbitrates for the memory bus and returns BGT# asserted (clock 6) for the allocation cycle. The MBC asserts CNA#, KWEND#, and SWEND# (clock 9) to pipeline the memory bus and close the cacheability and snoop windows. Note that (for this example) DRCTM# is asserted during SWEND# to place the line in the modified state. Since this is done, all other caches must invalidate their copies.

CRDY# for the allocation (line-fill) cycle is issued by the MBC in clock 11 to complete the read cycle on the memory bus and place the data into the 82490XP cache array.

MEMORY BUS SIGNALS:

The memory address latch enables (MALE and MBALE) may remain asserted by the MBC to place the address latches in the flow through mode. If the 82495XP is the current bus master, the memory address output enables (MAOE# and MBAOE#) should be asserted by the MBC.

For Clocked Memory Bus Mode, the memory data output enable (MDOE#) has been asserted by the MBC to drive the memory data outputs.

The MBC asserts MSEL# (clock 2) to allow sampling of MBRDY# and to latch MZBT# and MFRZ# for the write. MBRDY# and MEOC# are asserted

by the MBC (clock 3) to place the write data into the memory cycle buffers, sample MZBT# and MFRZ# for the next transfer, and end the current cycle on the memory bus. MFRZ# is driven active by the MBC here, indicating to the 82495XP that the data of the write through will be merged with the following allocation data.

For the allocation (line fill) cycle, MSEL# is driven active again by the MBC (clock 6) to allow sampling of MBRDY# and to latch MZBT# for the transfer. MZBT# is sampled on all MCLK edges where MSEL# is inactive. Once MSEL# is sampled active by the 82495XP, the value of MZBT# sampled on the prior MCLK is used for the next transfer. MDOE# is also deasserted in clock 6 to allow the data pins to be used as inputs for the allocation cycle.

MBRDY# is driven active by the MBC in clocks 7 to 9 to cause the memory burst counter to be incremented and data to be placed into the 82490XP cache memory cycle buffers. During the line fill, the 82490XP will merge the data from the write through buffer with the incoming data from either main memory or another cache (if that line was a write hit to [M] in another cache).

The MBC drives MEOC# asserted (clock 10) to end the allocation cycle on the memory bus and switch memory cycle buffers for the new cycle. MZBT# is sampled at this time for the next data transfer.

For Strobed Memory Bus Mode, the memory data output enable (MDOE#) has been asserted by the MBC to drive the memory data outputs.

The MBC asserts MSEL# (clock 2) to allow toggling of MISTB and to latch MZBT# and MFRZ# for the write (on MSEL# falling edge). MISTB is toggled and MEOC# asserted by the MBC (clock 2) to place the write data into the memory cycle buffers, sample MZBT# and MFRZ# for the next transfer (on the falling edge of MEOC# while MSEL# is active), and end the current cycle on the memory bus. MFRZ# is driven active by the MBC here, indicating to the 82495XP that the data of the write through will be merged with the following allocation data.

For the allocation (line fill) cycle, MSEL# is driven active again by the MBC (clock7) to allow sampling of MOSTB and to latch MZBT# for the transfer. MDOE# is also deasserted in clock 7 to allow the data pins to be used as inputs for the allocation cycle.

MOSTB is toggled by the MBC in clocks 8 to 10 to cause the memory burst counter to be incremented and data to be placed into the 82490XP cache memory cycle buffers. During the line fill, the 82490XP will merge the data from the write through buffer with the incoming data from either main memory or another cache (if that line was a write hit to [M] in another cache).

The MBC drives MEOC# asserted (clock 11) to end the allocation cycle on the memory bus and switch memory cycle buffers for the new cycle. MZBT# is sampled at this time for the next data transfer.

8.5 I/O Cycles

Figure 8-12 illustrates CPU initiated I/O cycles, both read and write. I/O writes are the only write cycles not posted by the 82495XP/82490XP cache (ie. the cycle is not fully acknowledged to the CPU until it has completed on the memory bus).

CACHE CONTROL SIGNALS:

The CPU initiates an I/O write cycle to the 82495XP/82490XP. The 82495XP then issues CADS# and CDTS# (clock 1) and the associated cycle control signals to the MBC (eg. CW/R#, CM/IO#, CD/C#, RDYSRC, MCACHE#). MCACHE# in not active, indicating that the cycle is not cacheable; RDYSRC is active, indicating that the MBC must supply BRDY#s to the CPU/Cache core.

The memory bus address (MSET[10:0], MTAG[11:0], MCFA[6:0]) is valid with CADS# (clocks 1 and 10 for the two read s in this example) and remain valid until after CNA# is sampled active by the 82495XP (clocks 6 and 17). MALE and MBALE may be used to hold the address as necessary.

The MBC arbitrates for the memory bus and returns BGT# asserted (clock 2) for the I/O write cycle, indicating that the cycle is guaranteed to complete on the memory bus. Once the 82495XP samples BGT# asserted, it must finish that cycle on the memory bus. Prior to this point, the cycle can be aborted by a snoop hit from another cache.

CNA# for the write cycle is asserted by the MBC (clock 5) to indicate that it is ready to schedule a new memory bus cycle. Note that SWEND# and KWEND# are not needed for I/O cycles since they are not cacheable.

The MBC asserts BRDY# in clock 7 to complete the I/O write cycle from the CPU, and CRDY# in clock 8 to complete the cycle on the memory bus from the 82495XP/82490XP cache.

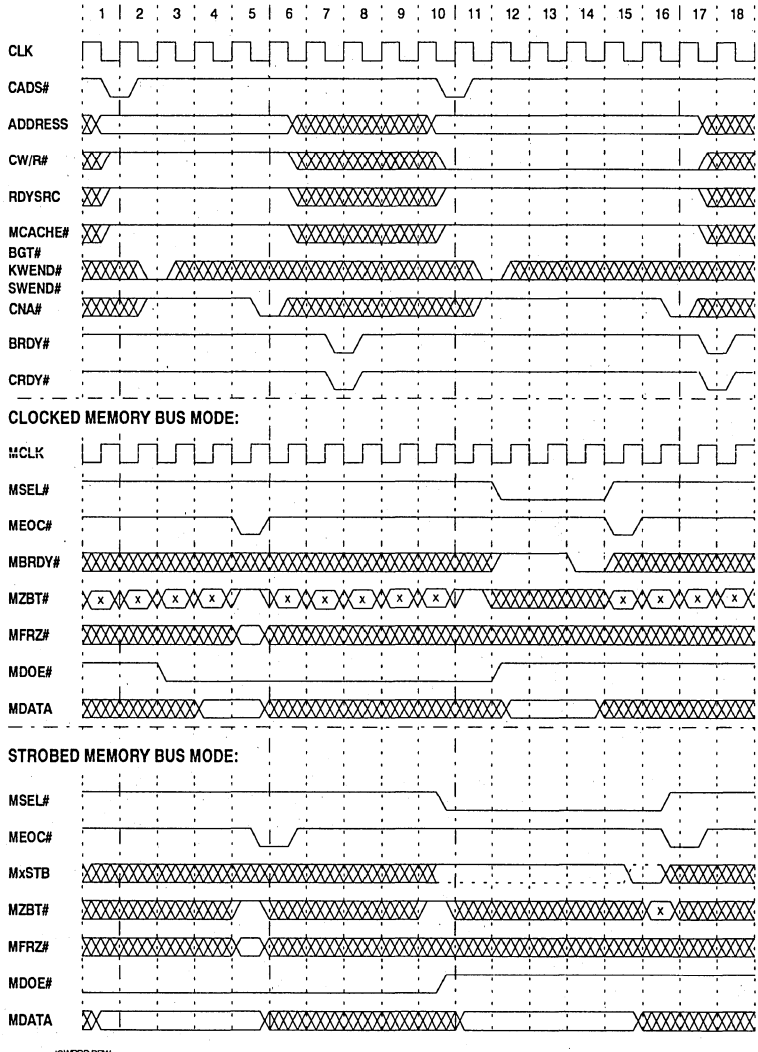


Figure 8-12. I/O Write and Read Cycles

A new CADS# is issued from the 82495XP in clock 10 for an I/O read cycle, along with the associated cycle control signals. MCACHE# is again not active, and RDYSRC is again active.

The MBC returns BGT# asserted right away (clock 11). The 82495XP can pipeline I/O cycles, but does not for the I/O read in this example.

Upon completing the access on the memory bus, the MBC activates BRDY# (clock 17) and CRDY# (clock 16). Note that BRDY# of a cycle may come before (as in the I/O write cycle of this example), with or after the CRDY# of the same cycle.

MEMORY BUS SIGNALS:

The memory address latch enables (MALE and MBALE) may remain asserted by the MBC to place the address latches in flow through mode. If the 82495XP is the current bus master, the memory address output enables (MAOE# and MBAOE#) should be asserted by the MBC.

For Clocked Memory Bus Mode, The memory data output enable signal (MDOE#) is asserted by the MBC in clock 3 to drive the memory data outputs.

MEOC# is asserted by the MBC (clock 5) to latch MZBT# for the I/O write transfer, and end that cycle on the memory bus (MBRDY# is not necessary since this example shows a single transfer cycle). MZBT# is driven high by the MBC in order to force the write cycle to begin with the correct burst address. MFRZ# is also sampled here (it need not be active since the cycle is not potentially allocatable).

For the I/O read cycle, MDOE# is deasserted (clock 12) by the MBC to allow the data pins to be used as inputs.

MSEL# is driven active by the MBC (clock 12) to allow sampling of MBRDY# and to latch MZBT# for the transfer. MZBT# is sampled on all MCLK edges where MSEL# is inactive. Once MSEL# is sampled active by the 82495XP, the value of MZBT# sampled on the prior MCLK is used for the next transfer. Again, MZBT# is driven high by the MBC to force the transfer to begin with the correct burst address.

The MBC asserts MBRDY# (clock 14) to cause the memory burst counter to be incremented and data to be placed into the 82490XP cache memory cycle buffers. The MBC drives MEOC# asserted (clock 15) to end the read cycle on the memory bus and switch memory cycle buffers for a new cycle. MZBT# for the next transfer is latched at this time.

For Strobed Memory Bus Mode, The memory data output enable signal (MDOE#) has been asserted by the MBC to drive the memory data outputs.

MEOC# is asserted by the MBC (clock 5) to latch MZBT# for the I/O write transfer (on MEOC# falling edge), and end that cycle on the memory bus (MOSTB is not necessary since this example shows a single transfer cycle). MZBT# is driven high by the MBC in order to force the write cycle to begin with the correct burst address. MFRZ# is also sampled here (it need not be active since the cycle is not potentially allocatable).

For the I/O read cycle, MDOE# is deasserted (clock 10) by the MBC to allow the data pins to be used as inputs.

MSEL# is driven active by the MBC (clock 10) to allow operation of MISTB and to latch MZBT# for the transfer (on MSEL# falling edge). Again, MZBT# is driven high by the MBC to force the transfer to begin with the correct burst address.

The MBC toggles MISTB (clock 15) to cause the memory burst counter to be incremented and data to be placed into the 82490XP cache memory cycle buffers for the I/O read cycle. Note: MISTB latches the memory bus data on both the rising and falling edges. The MBC drives MEOC# asserted (clock 16) to end the read cycle on the memory bus and switch memory cycle buffers for a new cycle. MZBT# for the next transfer is latched at this time (on the falling edge of MEOC#).

8.6 LOCKed Cycles

8.6.1 CPU READ MODIFY WRITE CYCLES

The 82495XP provides a facility to allow atomic accesses requested by the CPU (via CPU LOCK# activation) through the 82495XP KLOCK# signal. Figure 8-13 illustrates two back-to-back CPU initiated Locked read-modify-write cycles. KLOCK# activation indicates to the MBC that the memory bus should not be released between the KLOCKed cycles. KLOCK# will remain asserted from the beginning of the first cycle (with CADS#) until one clock after the CADS of the last cycle. The 82495XP does not distinguish between back-to-back locked operations and will not open an arbitration window (deassert KLOCK#) between them. It is the responsibility of the MBC to distinguish between the multiple RMW sequences, if it is so desired.

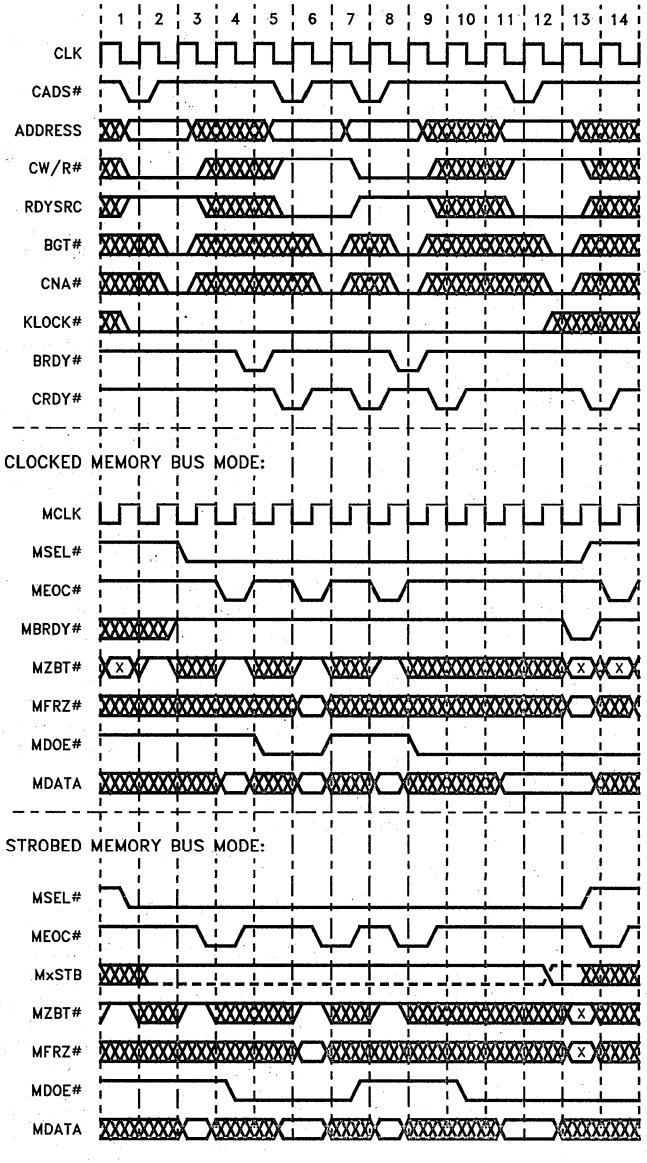


Figure 8-13. LOCKed Read-Modify-Write Cycles

The 82495XP issues a request for a memory bus access (CADS#) for every locked cycle (read or write) regardless if it hits the cache tag state or not. Locked read cycles are treated by the 82495XP as cache misses, and , if the line is in the [M] state, the 82495XP ignores the data on the memory bus and uses the data in the 82490XP array. Locked write cycles are treated as write through, and the tag state does not change even if the line is in the 82490XP array.

CACHE CONTROL SIGNALS:

The CPU initiates a Locked read cycle to the 82495XP/82490XP cache where, due to the assertion of CPU LOCK#, it assumes a cache miss and issues CADS# to the MBC (clock 1) along with the associated cycle control signals (eg. CW/R#, CM/IO#, CD/C#, RDYSRC, MCACHE#). MCACHE# is never asserted for LOCKed cycles; RDYSRC is active, indicating that the MBC must supply BRDY# to the CPU/Cache core.

The memory bus address (MSET[10:0], MTAG[11:0], MCFA[6:0]) is valid with CADS# (clocks 1 and 5, then 7 and 11 for the two locked RMW sequences in this example) and remain valid until after CNA# is sampled active by the 82495XP (clocks 3 and 7, then 9 and 13). MALE and MBALE may be used to hold the address as necessary.

The MBC arbitrates for the memory bus and returns BGT# asserted (clock 2), indicating that the cycle is guaranteed to complete on the memory bus. Once the 82495XP samples BGT# asserted, it must finish that cycle on the memory bus. Prior to this point, the cycle can be aborted by a snoop hit from another cache.

CNA# for the read cycle is also asserted by the MBC (clock 2) to indicate that it may schedule a new memory bus cycle. Note that the cycle control signals are not guaranteed to be valid after CNA# activation.

The MBC asserts BRDY# to the CPU/Cache core in clock 4. CRDY# for the locked read cycle is asserted to the 82495XP/82490XP from the MBC (clock 5) to load the data stored in the 82490XP's memory cycle buffers into the cache array. If the read was to a dirty line, the 82495XP is intelligent enough to ignore the data in the memory cycle buffers and use the data in the cache array.

Locked sequences always end in a write cycle, no new CPU initiated cycles may be inserted between the Locked read and Locked write cycles. Therefore, the 82495XP issues a new memory cycle request (CADS# in clock 5) for the Locked write as soon as it completes the Locked read cycle. The cycle control signals are also valid at this time. RDYSRC is not active, indicating that the 82495XP will supply BRDY# to the CPU.

The locked write cycle is posted like any other memory write cycle.

In this example, the CPU initiates a second read-modify-write cycle immediately. KLOCK# is not deasserted between the back-to-back locked sequences since the CPU LOCK# remains asserted. If snooping is required between these cycles, it is the MBC responsibility to predict this boundary and allow snooping. The 82495XP issues a memory bus request (CADS#) in clock 7 for the second locked read cycle, along with the new cycle control signals.

The second locked RMW sequence repeats the actions of the first. It's purpose in this example is to demonstrate that an arbitration window may not open between locked sequences if they follow one another with no idle or non-locked cycles between them.

MEMORY BUS SIGNALS:

The memory address latch enables (MALE and MBALE) may remain asserted by the MBC to place the address latches in flow through mode. If the 82495XP is the current bus master, the memory address output enables (MAOE# and MBAOE#) should be asserted by the MBC.

For Clocked Memory Bus Mode, MSEL# is driven active by the MBC (clock 3) to allow sampling of MBRDY# and to latch MZBT# for the transfer. MZBT# is sampled on all MCLK edges where MSEL# is inactive. Once MSEL# is sampled active by the 82495XP, the value of MZBT# sampled on the prior MCLK is used for the next transfer.

The memory data output enable signal (MDOE#) must be inactive to allow the data pins to be used as inputs for the first locked read cycle. The MBC asserts MEOC# (clock 4) to latch MZBT# for the next transfer, and end the current locked read cycle on the memory bus (MBRDY# is not necessary since this example shows a single transfer cycle). MZBT# is driven high by the MBC in order to force the read cycle to begin with the correct burst address.

For the locked write cycle, MDOE# is asserted by the MBC (clock 5) to drive the memory data outputs.

MEOC# is again asserted (clock 6) to latch MZBT# for the next transfer, and end the current locked write cycle on the memory bus (MBRDY# is not necessary since this is a single transfer cycle). MZBT# is again driven high. MFRZ# is also sampled during write cycles when MEOC# is sampled active by the 82495XP.

MDOE# is deasserted by the MBC (clock 7) to allow the data pins to be used as inputs for the second locked read cycle. MEOC# is again asserted (clock 8) to latch MZBT# for the next transfer, and end the locked read cycle on the memory bus. MZBT# is again driven high.

MDOE# is asserted by the MBC (clock 9) to drive the memory data outputs for the second locked write cycle. MBRDY# is asserted (clock 13) to cause the memory burst counter to be incremented and data to be placed into the 82490XP cache memory cycle buffers. The MBC drives MEOC# active and MSEL# inactive (clock 14) to end the locked write cycle on the memory bus and switch memory cycle buffers for a new cycle. MZBT# and MFRZ# for the next transfer are sampled at this time.

For Strobed Memory Bus Mode, MSEL# is driven active by the MBC (clock 1) to allow sampling of MxSTB and to latch MZBT# for the first locked read transfer (on the falling edge of MSEL#).

The memory data output enable signal (MDOE#) must be inactive to allow the data pins to be used as inputs for the first locked read cycle. The MBC asserts MEOC# (clock 3) to latch MZBT# for the next transfer (on MEOC# falling edge while MSEL# is active), and end the current locked read cycle on the memory bus (MISTB is not necessary since this example shows a single transfer cycle). MZBT# is driven high by the MBC in order to force the read cycle to begin with the correct burst address.

For the locked write cycle, MDOE# is asserted by the MBC (clock 4) to drive the memory data outputs. MEOC# is again asserted (clock 6) to latch MZBT# for the next transfer, and end the current locked write cycle on the memory bus (MOSTB is not necessary since this is a single transfer cycle). MZBT# is again driven high. MFRZ# is also sampled on the falling edge of MEOC#.

MDOE# is deasserted by the MBC (clock 7) to allow the data pins to be used as inputs for the second locked read cycle. MEOC# is again asserted (clock 8) to latch MZBT# for the next transfer, and end the locked read cycle on the memory bus. MZBT# is again driven high.

MDOE# is asserted by the MBC (clock 9) to drive the memory data outputs for the second locked write cycle. MOSTB is toggled (clock 12) to cause the memory burst counter to be incremented and data to be placed into the 82490XP cache memory cycle buffers. The MBC drives MEOC# active and MSEL# inactive (clock 13) to end the locked write cycle on the memory bus and switch memory cycle buffers for a new cycle. MZBT# and MFRZ# for the next transfer are sampled at this time.

9.0 TESTABILITY

Testing the 82495XP/82490XP chipset can be divided into three categories: Built-In Self Test (BIST), Boundary Scan, and external testing. BIST performs basic device testing on the 82495XP. Boundary Scan provides additional test hooks that conform to the IEEE Standard Test Access Port and Boundary Scan Architecture (IEEE Std.1149.1). Additional testing can be performed by using software written to test the 82490XP cache SRAM.

9.1 Built-In Self Test (BIST)

BIST tests the internal funcitonality of the 82495XP. The 82495XP's BIST tests approximately 90% of the cache controller. It tests the tag RAM and comparators.

The 82495XP BIST is initiated by driving SLFTST#(CRDY#) low and HIGHZ#(MBALE) high at least 10 clocks before RESET goes inactive. The 82495XP Cache Controller reports the result of BIST on the CAHOLD signal. When the self test completes, the 82495XP drives FSIOUT# inactive and the BIST result on CAHOLD. If CAHOLD is driven active the BIST successfully passed. If CAHOLD is driven inactive, BIST detected a flaw in the cache controller. CAHOLD is valid for one clock after FSIOUT# deactivation and should be sampled on the rising edge of FSIOUT#.

On the 82495XP, BIST only informs the system that a failure did or did not occur. BIST is not able to indicate where a failure occurred. After completing BIST the cache controller perform reset and begin normal operation.

9.2 Boundary Scan

The 82495XP/82490XP chipset provides additional test ability features compatible with the IEEE Standard Test Access Port and Boundary Scan Architecture (IEEE Std.1149.1). The test logic provided al-

lows for testing to insure that components function correctly, that interconnections between various components are correct, and that various components interact correctly on the printed circuit board.

The boundary scan test logic consists of a boundary scan register and support logic that are accessed through a test access port (TAP). The TAP provides a simple serial interface that makes it possible to test all signal traces with only a few probes.

The TAP can be controlled via a bus master. The bus master can be either automatic test equipment or a component (PLD) that interfaces to the four-pin test bus.

9.2.1 BOUNDARY SCAN ARCHITECTURE

The boundary scan test logic contains the following elements:

— Test access port (TAP), consisting of input pins TMS, TCK, and TDI; and ouput pin TDO.

— TAP controller, which interprets the inputs on the test mode select (TMS) line and performs the corresponding operation. The operations performed by the TAP include controlling the instruction and data registers within the component.

— Instruction register (IR), which accepts instruction codes shifted into the test logic on the test data input (TDI) pin. The instruction codes are used to select the specific test operation to be performed or the test data register to be accessed.

— Test data registers: The 82495XP/82490XP chipset components each contain three test data registers: Bypass register (BPR), Device Identification register (DID), and Boundary Scan register (BSR).

The instruction and test data registers are separate shift-register paths connected in parallel and have a common serial data input and a common serial data output connected to the TAP signals, TDI and TDO, respectively.

9.2.2 DATA REGISTERS

The 82495XP and 82490XP both contain the two required test data registers; bypass register and boundary scan register. In addition, they also have a device identification register.

Each test data register is serially connected to TDI and TDO, with TDI connected to the most significant bit and TDO connected to the least significant bit of the test data register. Data is shifted one stage (bit position within the register) on each rising edge of the test clock (TCK).

9.2.2.1 Bypass Register

The Bypass Register is a one-bit shift register that provides the minimal length path between TDI and TDO. This path can be selected when no test operation is being performed by the component to allow rapid movement of test data to and from other components on the board. While the bypass register is selected data is transferred from TDI to TDO without inversion.

9.2.2.2 Boundary Scan Register

The Boundary Scan Register is a single shift register path containing the boundary scan cells that are connected to all input and output pins of the 82495XP/82490XP chipset. Figure 9.1 shows the logical structure of the boundary scan register. While output cells determine the value of the signal driven on the corresponding pin, input cells only capture data; they do not affect the normal operation of the device. Data is transferred without inversion from TDI to TDO through the boundary scan register during scanning. The boundary scan register can be operated by the EXTEST and SAMPLE instructions. The boundary scan register order is described in section 9.2.5.

9.2.2.3 Device Identification Register

The Device Identification Register contains the manufacturer's identification code, part number code, and version code in the format shown in Figure 9.2. Table 9.1 lists the codes corresponding to the 82495XP and 82490XP.

Table 9-1. Device ID Register Values

Component Code	Version Code	Part Number Code	Manufacturer Identity
82495XP (A0 or A1) 0Ah	0495h	0495h	09h
82495XP (B0)	0Bh	0495h	09h
82490XP (A0 or A1)	00h	49A0h	09h

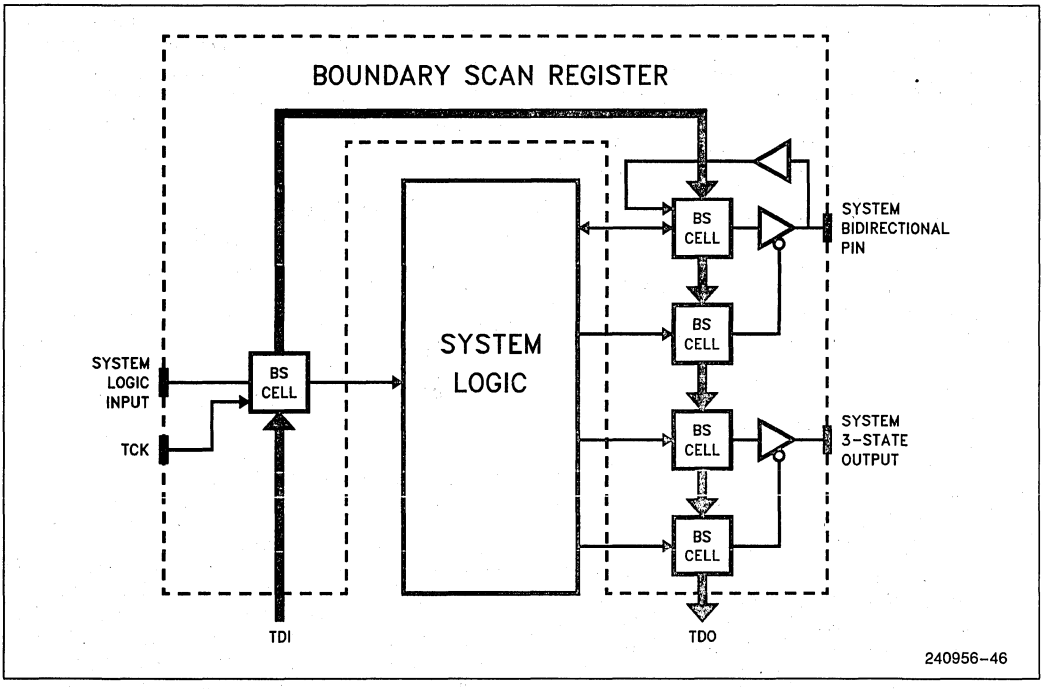


Figure 9-1. Boundary Scan Register Structure

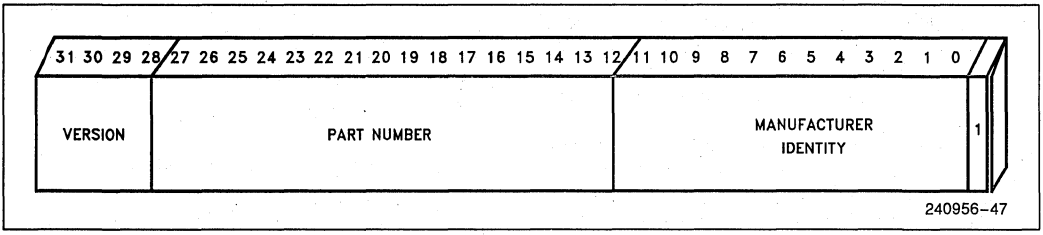


Figure 9-2. Device ID Register

9.2.2.4 Runbist Register

The Runbist Register is a one bit register used to report the results of the 82495XP BIST when it is initiated by the RUNBIST instruction. This register is loaded with a "1" prior to invoking the BIST and is loaded with "1" upon successfull completion. "0" indicates a failure occurred during BIST.

NOTE:
82495XP RUNBIST is not available in the A-stepping.

9.2.3 INSTRUCTION REGISTER

The Instruction Register (IR) allows instructions to be serially shifted into the device. The instruction selects the particular test to be performed, the test data register to be accessed, or both. The instruction register is four (4) bits wide. The most significant bit is connected to TDI and the least significant bit is connected to TDO. There are no parity bits associated with the Instruction register. Upon entering the Capture-IR TAP controller state, the Instruction register is loaded with the default instruction "0001", SAMPLE/PRELOAD. Instructions are shifted into the instruction register on the rising edge of TCK while the TAP controller is in the Shift-IR state.

9.2.3.1 82495XP Boundary Scan Instruction Set

The 82495XP cache controller supports all three mandatory boundary scan instructions (BYPASS, SAMPLE/PRELOAD, and EXTEST) along with one optional instruction (IDCODE). On the B-Stepping of the 82495XP two additional optional instructions will be implemented (RUNBIST and TRISTATE). Table 9.3 lists the 82495XP boundary scan instruction codes. The instructions listed as PRIVATE cause TDO to become enabled in the Shift-DR state and cause "0" to be shifted out of TDO on the rising edge of TCK. Execution of the PRIVATE instructions will not cause hazardous operation of the 82495XP. Note that system tests should not execute instruction codes labeled "RESERVED". These instructions can put the component in an undeterminant state which can only be cleared by power on reset.

Table 9-2. 82495XP Boundary Scan Instruction Codes

Instruction Code	Instruction Name
0000	EXTEST
0001	SAMPLE
0010	IDCODE
0011	*RESERVED*
0100	*RESERVED*
0101	*RESERVED*
0110	*RESERVED*
0111	*RUNBIST
1000	*TRISTATE
1001	*RESERVED*
1010	PRIVATE
1011	PRIVATE
1100	PRIVATE
1101	PRIVATE
1110	PRIVATE
1111	BYPASS

* RUNBIST and TRISTATE are boundary scan instructions that will be implemented in the B-stepping of the 82495XP. They are not available on the A-stepping.

EXTEST — The instruction code is "0000". The EXTEST instruction allows testing of circuitry external to the component package, typically board interconnects. It does so by driving the values loaded into the 82495XP boundary scan register out on the output pins corresponding to each boundary scan cell and capturing the values on 82495XP input pins to be loaded into their corresponding boundary scan register locations. I/O pins are selected as input or output, depending on the value loaded into their control setting locations in the boundary scan register. Values shifted into input latches in the boundary scan register are never used by the internal logic of the 82495XP. Note: after using the EXTEST instruction, the 82495XP must be reset before normal (non-boundary scan) use.

SAMPLE/PRELOAD — The instruction code is "0001". The SAMPLE/PRELOAD has two functions that it performs. When the TAP controller is in the Capture-DR state, the SAMPLE/PRELOAD instruction allows a "snap-shot" of the normal operation of the component without interfering with that normal operation. The instruction causes boundary scan register cells associated with outputs to sample the value being driven by the 82495XP. It causes the cells associated with inputs to sample the value being driven into the 82495XP. On both outputs and inputs the sampling occurs on the rising edge of TCK. When the TAP controller is in the Update-DR state, the SAMPLE/PRELOAD instruction preloads data to the device pins to be driven to the board by executing the EXTEST instruction. Data is preloaded to the pins from the boundary scan register on the falling edge of TCK.

IDCODE — The instruction code is "0010". The IDCODE instruction selects the device identification register to be connected to TDI and TDO, allowing the devices identification code to be shifted out of the device on TDO. Note that the device identification register is not altered by data being shifted in on TDI.

BYPASS — The instruction code is "1111". The BYPASS instruction selects the bypass register to be connected to TDI and TDO, effectively bypassing the test logic on the 82495XP by reducing the shift length of the device to one bit. Note that an open circuit fault in the board level test data path will cause the bypass register to be selected following an instruction scan cycle due to the pull-up resistor on the TDI input. This has been done to prevent any unwanted interference with the proper operation of the system logic.

RUNBIST The instruction code is "0111". The RUNBIST instruction selects the one (1) bit runbist register, loads a value of "0" into the runbist register, and connects it to TDO. It also initiates the built-in self test (BIST) feature of the 82495XP, which is able to detect approximately 90% of the stuck-at faults on the 82495XP. The 82495XP ac/dc specifications for VCC and CLK must be met and reset must have been asserted at least once prior to executing the RUNBIST boundary scan instruction. After loading the RUNBIST instruction code in the instruction register, the TAP controller must be placed in the Run-Test/Idle state. BIST begins on the first rising edge of TCK after entering the Run-Test/Idle state. The TAP controller must remain in the Run-Test/Idle state until BIST is completed. It requires 100K clock (CLK) cycles to complete BIST and report the result to the runbist register. After completing the 100K clock (CLK) cycles, the value in the runbist register should be shifted out on TDO during the Shift-DR state. A value of "1" being shifted out on TDO indicates BIST successfully completed. A value of "0" indicates a failure occurred. After executing the RUNBIST instruction, the 82495XP must be reset prior to normal operation. NOTE: This instruction is not available on the A-stepping of the 82495XP. It will be implemented in the B-stepping.

TRISTATE The instruction code is "1000". The TRISTATE instruction initiates the tri-state output test mode. After loading the TRISTATE boundary scan instruction into the instruction register, the TAP controller must be placed in the Run-Test/Idle state. To terminate the tristate output test mode, the 82495XP must be reset. NOTE: This instruction is not available on the A-stepping of the 82495XP. It will be implemented in the B-stepping.

9.2.3.2 82490XP Boundary Scan Instruction Set

The 82490XP cache controller supports all three mandatory boundary scan instructions (BYPASS, SAMPLE/PRELOAD, and EXTEST) along with one optional instruction (IDCODE). Table 9.4 lists the 82490XP boundary scan instruction codes. The instructions listed as PRIVATE cause TDO to become enabled in the Shift-DR state and cause "0" to be shifted out of TDO on the rising edge of TCK. Execution of the PRIVATE instructions will not cause hazardous operation of the 82490XP. Note that system tests should not execute instruction codes labeled "INTEL RESERVED". These instructions can put the component in an undeterminant state which can only be cleared by power on reset.

Table 9-3. 82490XP Boundary Scan Instruction Codes

Instruction Code	Instruction Name
0000	EXTEST
0001	SAMPLE
0010	IDCODE
0011	INTEL RESERVED
0100	INTEL RESERVED
0101	INTEL RESERVED
0110	INTEL RESERVED
0111	INTEL RESERVED
1000	INTEL RESERVED
1001	INTERL RESERVED
1010	PRIVATE
1011	PRIVATE
1100	PRIVATE
1101	PRIVATE
1110	PRIVATE
1111	BYPASS

EXTEST The instruction code is "0000". The EXTEST instruction allows testing of circuitry external to the component package, typically board interconnects. It does so by driving the values loaded into the 82490XP boundary scan register out on the output pins corresponding to each boundary scan cell and capturing the values on 82490XP input pins to be loaded into their corresponding boundary scan register locations. I/O pins are selected as input or output, depending on the value loaded into their control setting locations in the boundary scan register. Values shifted into input latches in the boundary scan register are never used by the internal logic of the 82490XP. Note: after using the EXTEST instruction, the 82490XP must be reset before normal (non-boundary scan) use.

SAMPLE/ PRELOAD — The instruction code is "0001". The SAMPLE/PRELOAD has two functions that it performs. When the TAP controller is in the Capture-DR state, the SAMPLE/PRELOAD instruction allows a "snap-shot" of the normal operation of the component without interfering with that normal operation. The instruction causes boundary scan register cells associated with outputs to sample the value being driven by the 82490XP. It causes the cells associated with inputs to sample the value being driven into the 82490XP. On both outputs and inputs the sampling occurs on the rising edge of TCK. When the TAP controller is in the Update-DR state, the SAMPLE/PRELOAD instruction preloads data to the device pins to be driven to the board by executing the EXTEST instruction. Data is preloaded to the pins from the boundary scan register on the falling edge of TCK.

IDCODE — The instruction code is "0010". The IDCODE instruction selects the device identification register to be connected to TDI and TDO, allowing the devices identification code to be shifted out of the device on TDO. Note that the device identification register is not altered by data being shifted in on TDI.

BYPASS — The instruction code is "1111". The BYPASS instruction selects the bypass register to be connected to TDI and TDO, effectively bypassing the test logic on the 82490XP by reducing the shift length of the device to one bit. Note that an open circuit fault in the board level test data path will cause the bypass register to be selected following an instruction scan cycle due to the pull-up resistor on the TDI input. This has been done to prevent any unwanted interference with the proper operation of the system logic.

9.2.4 TEST ACCESS PORT (TAP) CONTROLLER

The TAP controller is a synchronous, finite state machine. It controls the sequence of operations of the test logic. The TAP controller changes state only in response to the following events:

1. A rising edge of TCK
2. Power-up.

The value of the test mode state (TMS) input signal at a rising edge of TCK controls the sequence of the state changes. The state diagram for the TAP contoller is shown in figure 9.3. Test designers must consider the operation of the state machine in order to design the correct sequence of values to drive on TMS.

9.2.4.1 Test-Logic-Reset State

In this state, the test logic is disabled so that normal operation of the device can continue unhindered. This is achieved by initializing the instruction register such taht the IDCODE instruction is loaded. No matter what the original state of the controller, the controller enters Test-Logic-Reset state when the TMS input is held high (1) for at least five rising edges of TCK. The controller remains in this state while TMS is high. The TAP controller is also forced to enter this state at power-up.

9.2.4.2 Run-Test/Idle State

A controller state between scan operations. Once in this state, the controller remains in this state as long as TMS is held low. In devices supporting the RUNBIST instruction, the BIST is performed during this state and the result is reported in the runbist register. For instructions not causing functions to execute during this state, no activity occurs in the test logic. The instruction register and all test data registers retain their previous state. When TMS is high and a rising edge is applied to TCK, the controller moves to the Select-DR state.

9.2.4.3 Select-DR-Scan State

This is a temporary controller state. The test data register selected by the current instruction retains its previous state. If TMS is held low and a rising edge is applied to TCK when in this state, the controller moves into the Capture-DR state, and a scan sequence for the selected test data register is initiated. If TMS is held high and a rising edge is applied to TCK, the controller moves to the Select-IR-Scan state.

The instruction does not change in this state.

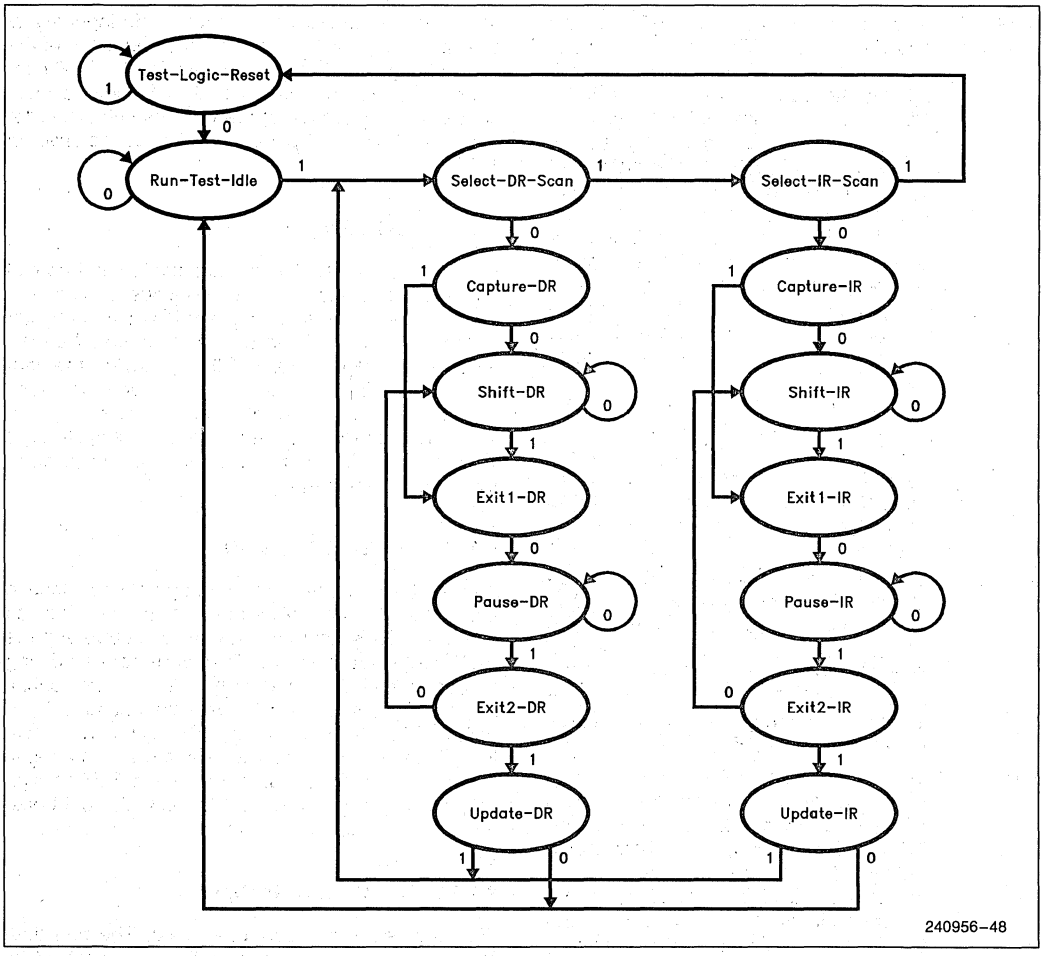


Figure 9-3. Tap Controller State Diagram

9.2.4.4 Capture-DR State

In this state, the boundary scan register captures input pin data if the current instruction is EXTEST or SAMPLE/PRELOAD. The other test data registers, which do not have parallel input, are not changed.

The instruction does not change in this state.

When the TAP controller is in this state and a rising edge is applied to TCK, the controller enters the Exit1-DR state if TMS is high or the Shift-DR state if TMS is low.

9.2.4.5 Shift-DR State

In this controller state, the test data register connected between TDI and TDO as a result of the current instruction, shifts data one stage toward its serial output on each rising edge of TCK.

The instruction does not change in this state.

When the TAP controller is in this state and a rising edge is applied to TCK, the controller enters the Exit1-DR state if TMS is high or remains in the Shift-DR state if TMS is low.

9.2.4.6 Exit1-DR State

This is a temporary state. While in this state, if TMS is held high, a rising edge applied to TCK causes the controller to enter the Update-DR state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the Pause-DR state.

The test data register selected by the current instruciton retains its previous value during this state. The instruction does not change in this state.

9.2.4.7 Pause-DR State

The pause state allows the test controller to temporarily halt the shifting of data through the test data register in the serial path between TDI and TDO. An example of using this state could be to allow a tester to reload its pin memory from disk during application of a long test sequence.

The test data register selected by the current instruciton retains its previous value during this state. The instruction does not change in this state.

The controller remains in this state as long as TMS is low. Whne TMS goes high and a rising edge is applied to TCK, the controller moves to the Exit2-DR state.

9.2.4.8 Exit2-DR State

This is a temporary state. While in this state, if TMS is held high, a rising edge applied to TCK causes the controller to enter the Update-DR state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the Shift-DR state.

The test data register selected by the current instruciton retains its previous value during this state. The instruction does not change in this state.

9.2.4.9 Update-DR State

The boundary scan register is provided with a latched parallel output to prevent changes at the parallel output while data is shifted in response to the EXTEST and SAMPLE/PRELOAD instructions. When the TAP controller is in this state and the boundary scan register is selected, data is latched onto the parallel output of this register from the shift-register path on the falling edge of TCK. The data held at the latched parallel output does not change other than in this state.

All shift-register stages in test data register selected by the current instruciton retains its previous value during this state. The instruction does not change in this state.

9.2.4.10 Select-IR-Scan State

This is a temporary controller state. The test data register selected by the current instruction retains its previous state. If TMS is held low and a rising edge is applied to TCK when in this state, the controller moves into the Capture-IR state, and a scan sequence for the instruction register is initiated. If TMS is held high and a rising edge is applied to TCK, the controller moves to the Test-Logic-Reset state.

The instruction does not change in this state.

9.2.4.11 Capture-IR State

In this controller state the shift register contained in the instruction register loads the fixed value "0001" on the rising edge of TCK.

The test data register selected by the current instruciton retains its previous value during this state. The instruction does not change in this state.

When the controller is in this state and a rising edge is applied to TCK, the controller enters the Exit1-IR state if TMS is held high, or the Shift-IR state if TMS is held low.

9.2.4.12 Shift-IR State

In this state the shift register contained in the instruction register is connected between TDI and TDO and shifts data one stage towards its serial output on each rising edge of TCK.

The test data register selected by the current instruciton retains its previous value during this state. The instruction does not change in this state.

When the controller is in this state and a rising edge is applied to TCK, the controller enters the Exit1-IR state if TMS is held high, or remains in the Shift-IR state if TMS is held low.

9.2.4.13 Exit1-IR State

This is a temporary state. While in this state, if TMS is held high, a rising edge applied to TCK causes the controller to enter the Update-IR state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the Pause-IR state.

The test data register selected by the current instruciton retains its previous value during this state. The instruction does not change in this state.

9.2.4.14 Pause-IR State

The pause state allows the test controller to temporarily halt the shifting of data through the instruction register.

The test data register selected by the current instruciton retains its previous value during this state. The instruction does not change in this state.

The controller remains in this state as long as TMS is low. When TMS goes high and a rising edge is applied to TCK, the controller moves to the Exit2-IR state.

9.2.4.15 Exit2-IR State

This is a temporary state. While in this state, if TMS is held high, a rising edge applied to TCK causes the controller to enter the Update-IR state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the Shift-IR state.

The test data register selected by the current instruciton retains its previous value during this state. The instruction does not change in this state.

9.2.4.16 Update-IR State

The instruction shifted into the instruction register is latched onto the parallel output from the shift-register path on the falling edge of TCK. Once the new instruction has been latched, it becomes the current instruction.

Test data registers selected by the current instruction retain the previous value.

9.2.5 BOUNDARY SCAN REGISTER CELL

The boundary scan register for each component contains a cell for each pin, as well as cells for control of I/O and tristate pins.

9.2.5.1 82495XP Boundary Scan Register Cell

The following is the bit order of the 82495XP boundary scan register: (from left to right and top to bottom)

TDI → MKEN# KWEND# SWEND# BGT# CNA# BRDY# RESERVED CRDY# MWBWT# DRCTM# MRO# CWAY# FPFLD# SNPCYC# SNPBSY# MHITM# MTHIT# CAHOLD FSIOUT# PALLC# SNPADS# CADS# CDTS# CWR# CDC# CMIO# RDYSRC MCACHE# KLOCK# SMLN# NENE# CFA3 CFA2 TAG11 TAG10 TAG9 TAG8 TAG7 TAG6 TAG5 TAG4 TAG3 TAG2 TAG1 TAG0 SET10 SET9 SET8 SET7 CLK SET6 SET5 SET4 SET3 SET2 SET1 SET0 CFA6 CFA5 CFA4 CFA1 CFA0 ADS# LEN BLAST# BRDYC1# BRDYC2# CACHE# LOCK# BLE# BOFF# KEN# AHOLD WR# MIO# DC# PWT PCD HITM# PCYC EADS# NA# INV WBWT# WAY WRARR# MCYC# BUS# MAWEA# WBWE# WBA WBTYP MCFA0 MCFA1 MCFA4 MCFA5 MCFA6 MSET0 MSET1 MSET2 MSET3 MSET4 MSET5 MSET6 MSET7 MSET8 MSET9 MSET10 MTAG0 MTAG1 MTAG2 MTAG3 MTAG4 MTAG5 MTAG6 MTAG7 MTAG8 MTAG9 MTAG10 MTAG11 MCFA2 MCFA3 RESET MAOE# MBAOE# SNPCLK SNPSTB# EWBE# MPIC# SNPINV FLUSH# SNYC# SNPNCA MBALE MALE MACTL OCTL CFA4CTL CFA5CTL CACTL FPFLDCTL WBWTCTL NACTL → TDO

"RESERVED" signals correspond to no connect "NC" signals on the 82495XP.

EWBE# and MPIC# will be implemented in the 82495XP B-stepping, omit from boundary scan register for A-stepping 82495XPs.

All the *CTL cells are control cells that are used to select the direction of bidirectional pins or tristate output pins. If "1" is loaded into the control cell(*CTL), the associated pin(s) are tristated or selected as input. The following lists the control cells and their corresponding pins.

1. MACTL controls the MSET0–10, MTAG0–11, and MCFA0–6 pins.
2. OCTL controls the WAY, WRARR#, MCYC#, MAWEA#, BUS#, WBWE#, WBA, WBTYP, INV, EADS#, AHOLD, KEN#, BOFF#, BLE#, BRDYC2#, BRDYC1#, BLAST#, NENE#, SMLN#, KLOCK#, MCACHE#, RDYSRC, CMIO#, CDC#, CWR#, CDTS#, CADS#, SNPADS#, PALLC#, FSIOUT#, CAHOLD, MTHIT#, MHITM#, SNPBSY#, SNPCYC#, CWAY, EWBE#, and MPIC# output pins.
3. CFA4CTL controls the CFA4 pin.
4. CFA5CTL controls the CFA5 pin.
5. CACTL controls the SET0–10, TAG0–11, CFA0–3, and CFA6 pins.
6. FPFLDCTL controls the FPFLD# pin.
7. WBWTCTL controls the WB/WT# pin.
8. NACTL controls the NA# pin.

9.2.5.2 82490XP Boundary Scan Register Cell

The following is the bit order of the 82490XP boundary scan register: (from left to right and top to bottom)

TDI → CDCTL WR# BLAST# BRDYC# BRDY# HITM# ADS# BE# A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15 MDATA7 MDATA3 MDATA6 MDATA2 MDATA5 MDATA1 MDATA4 MDATA0 MDCTL MDOE# MZBT# MBRDY# MOEC# MFRZ# MSEL# MCLK MOCLK RESET PAR# RESERVED BOFF# WBTYP WBA WBWE# BUS# MAWEA# MCYC# CRDY# WRARR# WAY CDATA4 CDATA0 CDATA2 CDATA5 CDATA6 CDATA1 CDATA3 CDATA7 → TDO

"RESERVED" signals correspond to no connect "NC" signals on the 82490XP.

All the *CTL cells are control cells that are used to select the direction of bidirectional pins or tristate output pins. If "1" is loaded into the control cell(*CTL), the associated pin(s) are tristated or selected as input. The following lists the control cells and their corresponding pins.

1. CDCTL controls the CDATA0–7 pins.
2. MDCTL controls the MDATA0–7 pins.

9.2.6 TAP CONTROLLER INITIALIZATION

The TAP controller is automatically intialized when a device is powered up. In addition, the TAP controller can be initialized by applying a high signal level on the TMS input for five TCK periods.

9.2.7 BOUNDARY SCAN SIGNAL DESCRIPTION AND TIMINGS

The functionality of TDI, TMS, TDO, and TCK are described in Chapter 7. The A.C. timing specifications for the boundary scan signals are located in Chapter 10.

9.3 Tri-State Output Test Mode

The 82495XP has the ability to tri-state all of its outputs and bidirectional pins and to disable all pull-ups and pull-downs. During tri-state output test mode all pins floated during bus hold as well as those which are never floated during normal operation are tri-stated. When the 82495XP is in tri-state output test mode, external testing can be used to test board interconnections.

On the 82495XP, tri-state output test mode is invoked by driving HIGHZ#(MBALE) and SLFTST#-(CRDY#) active to the 82495XP at least 10 clocks prior to the deassertion of RESET. Note that HIGHZ# has priority over SLFTST#. When both HIGHZ# and SLFTST# are driven active the 82495XP will invoke the tri-state output mode and not invoke BIST.

Once tri-state output test mode is invoked, the 82495XP remains in it until the next RESET.

9.4 82490XP Cache SRAM Testing

The 82490XP cache SRAM can be tested using standard cache memory testing techniques. Code must be written to:

1. Flush and reset the 82495XP/82490XP/CPU cache
2. Write 1's to every bit of a block of memory equal to the cache size
3. Read the block of memory to fill the cache, tagging the data as read-only using the MRO# signal
4. Write 0's to every bit in the block of memory
5. Read the block, the cache hits should be all 1's
6. Repeat the process, exchanging 0 for 1 and 1 for 0

In this example, the code to test the cache must be non-cacheable to the 82495XP. Also, the CPU cache must be on so that the 82495XP will perform line-fills.

10.0 AC/DC SPECIFICATIONS

10.1 Background

The 82495XP has four main interfaces: CPU Bus, memory bus controller, memory bus, and 82490XP. The memory bus controller is typically implemented with PLD devices. The MBC interface signal timings are, therefore, generated based on available, off-the-shelf PLD specs. The memory bus interface was specified to suit a generic memory interface which works up to CPU frequency.

10.2 D.C. Specifications

Table 10-1. D.C. Specifications

Vcc = 5V ±5%, Tcase = 0 to +85°C					
Symbol	**Parameter**	**Min**	**Max**	**Unit**	**Notes**
V_{IL}	Input Low Voltage	−0.3	+0.8	V	TTL Level
V_{IH}	Input High Voltage 2.0	2.0	Vcc+0.3	V	TTL Level
V_{OL}	Output Low Voltage		0.45	V	TTL Level (1)
V_{OH}	Output High Voltage	2.4		V	TTL Level (2)
I_{CC}	Power Supply Current		550 300	mA	82495XP @ 50 MHz, (3) 82490XP @ 50 MHz
Power	Power Dissipation		2.75 1.50	W	82495XP @ 50 MHz, (4) 82490XP @ 50 MHz
I_{LI}	Input Leakage Current		±15	uA	$0 < V_{IN} > V_{CC}$
I_{LO}	Output Leakage Current		±15	uA	$0 \leq V_{OUT} \leq V_{CC}$ Tristate
I_{IL}	Input Leakage Current		200	uA	V_{IN} = 0.45V, (5)
C_{IN}	Input Capacitance		14 5	pF	for 82495XP for 82490XP
C_O	Output Capacitance		18 15	pF	for 82495XP for 82490XP
$C_{I/O}$	I/O Capacitance		18 15	pF	for 82495XP for 82490XP
C_{CLK}	CLK Input Capacitance		14 5	pF	for 82495XP for 82490XP
C_{TIN}	Test Input Capacitance		15 10	pF	for 82495XP for 82490XP
C_{TOUT}	Test Output Capacitance		15 10	pF	for 82495XP for 82490XP
C_{TCK}	Test Clock Capacitance		15 10	pF	for 82495XP for 82490XP

NOTES:
(1) Parameter measured at 4mA Iload.
For MCFA6–FCFA0, MSET10–MSET0, and MTAG11–MTAG0, this parameter is measured at 16 mA Iload.
(2) Parameter measured at 1mA Iload.
For MCFA6–MCFA0, MSET10–MSET0, and MTAG11–MTAG0, this parameter is measured at 2 mA Iload.
(3) Typical Supply current 400mA.
(4) Typical Power dissipation is 2W.
(5) This parameter is for input with pullup.

10.3 A.C. Specifications

All TTL timing specs are measured at 1.5V for both "0" and "1" logic level.

Table 10-2. Clock, Reset, and Configuration

Vcc = 5V ± 5%, Tcase = 0 to +85 °C Maximum CL = 50 pF unless otherwise specified. Minimum CL = 20 pF unless otherwise specified. All Inputs and Outputs are TTL Level.						
Symbol	**Parameter**	**Min**	**Max**	**Unit**	**Figure**	**Notes**
t0	CLK, MCLK, MOCLK Frequency	16.6	50	MHz		1x clock
t1	CLK, MCLK, MOCLK Stability		0.1	%		
t2	CLK, MCLK, MOCLK Period	20	60	ns	10-1	
t3	CLK, MCLK, MOCLK High Time	7		ns	10-1	(1)
t4	CLK, MCLK, MOCLK Low Time	7		ns	10-1	(1)
t5	CLK, MCLK, MOCLK Rise Time		2	ns		(1)
t6	CLK, MCLK, MOCLK Fall Time		2	ns		(1)
t7	RESET Setup Time	7		ns	10-4	
t8	RESET Hold Time	2		ns	10-4	
t9	RESET Duration	8xt2 15xt2		ns	10-4	for 82495XP, (2) for 82490XP
t10	All Configurations CFG3–CFG0, CPUTYP, SNPMD, PLOCKEN, MEMLDRV, 82490XPLDRV, HIGHZ#, SLFTST# Setup Time	10x12		ns	10-4	(3), (4)
t11	All Configurations CFG3–CFG0, CPUTYP, SNPMD, PLOCKEN, MEMLDRV, 82490XPLDRV, HIGHZ#, SLFTST# Hold Time	0		ns	10-4	(3), (5)
t12	FLUSH#, SYNC# Setup Time	8		ns	10-3	for 82495XP, (6)
t13	FLUSH#, SYNC# Hold Time	1		ns	10-3	for 82495XP, (7)
t14	FLUSH#, SYNC# Duration	2xt2		ns		(8)
t15	MOCLK falling edge to MCLK rising edge	2		ns		
t16	FERR#, HLDA Valid Delay	2	15	ns	10-2	
t17	FERR#, HLDA Float Delay		18	ns		
t18	HOLD, BOFF# Setup Time	7		ns	10-3	
t19	HOLD, BOFF# Hold Time	2		ns	10-3	

NOTE:
(1) Rise/Fall, High/Low times measured between 0.8V and 2.0V.
(2) Power up reset duration should be 1 ms after Vcc and CLK are stable. If configuration inputs with pullups are left floated, 10 us RESET duration is required.
(3) Timing is referenced to reset falling edge.
(4) 8ns setup time is required to guarantee recognition on next clock.
(5) 1ns hold time is required to guarantee recognition on next clock.
(6) To guarantee recognition on next clock.
(7) Synchronous mode only.
(8) Asynchronous mode only. To guarantee recognition.

Table 10-3. Memory Bus Controller 82495XP/82490XP Interface

Vcc = 5V ± 5%, Tcase = 0 to +85 °C Maximum CL = 50 pF unless otherwise specified. Minimum CL = 20 pF unless otherwise specified. All Inputs and Outputs are TTL Level.						
Symbol	**Parameter**	**Min**	**Max**	**Unit**	**Figure**	**Notes**
t30	BRDY#, CRDY#, KWEND#, SWEND#, BGT#, CNA#, [WRMRST] Setup Time	8		ns	10-3	82495XP Only
t30a	BRDY#, CRDY# Setup Time	7		ns	10-3	82490XP Only
t31	BRDY#, CRDY#, KWEND#, SWEND#, BGT#, CNA#, [WRMRST] Hold Time	1		ns	10-3	82495XP Only
t32	CW/R#, CD/C#, CMI/O#, RDYSRC, MCACHE#, KLOCK#, BLE#, PALLC#, CAHOLD, CWAY, FSIOUT#, CADS#, CDTS#, SNPADS# Valid Delay	2	12	ns	10-2	
t33	NENE#, SMLN# Valid Delay	2	15	ns	10-2	
t34	MDATA Setup to CLK (clock before BRDY# active)	6		ns	10-3	
t35	MDATA Valid Delay from CLK (CLK from CDTS# valid, MDOE# active)	3	15	ns	10-2	
t36	MDATA Valid Delay from MDOE# active		10	ns	10-2	
t37	MDATA Fload Delay from MDOE# inactive	0	14	ns		

Table 10-4. 82495XP Memory Interface

Vcc = 5V ± 5%, Tcase = 0 to +85 °C Maximum CL = 50 pF unless otherwise specified. Minimum CL = 20 pF unless otherwise specified. All Inputs and Outputs are TTL Level.						
Symbol	**Parameter**	**Min**	**Max**	**Unit**	**Figure**	**Notes**
t50	SNPCLK Frequency		50	MHz		1x clock (10)
t51	SNPCLK Period	20		ns	10-1	(11)
t52	SNPCLK High Time	8		ns	10-1	
t53	SNPCLK Low Time	8		ns	10-1	
t54	SNPCLK Rise Time		2	ns		(1)
t55	SNPCLK Fall Time		2	ns		(1)
t56	MCFA6–MCFA0, MSET10–MSET0, MTAG11–MTAG0 Valid Delay	2	13	ns	10-5	(2), (3)
t56	MCFA6–MCFA0, MSET10–MSET0, MTAG11–MTAG0 Float Delay	2	15	ns	10-5	(4)
t58	MCFA6–MCFA0, MSET10–MSET0, MTAG11-MTAG0 Valid Delay	2	15	ns	10-5	(5)

Table 10-4. 82495XP Memory Interface (Continued)

Vcc = 5V ± 5%, Tcase = 0 to +85 °C Maximum CL = 50 pF unless otherwise specified. Minimum CL = 20 pF unless otherwise specified. All Inputs and Outputs are TTL Level.						
Symbol	**Parameter**	**Min**	**Max**	**Unit**	**Figure**	**Notes**
t60	MCFA6–MCFA0, MSET10–MSET0, MTAG11–MTAG0 Valid Delay	2	15	ns	10-2	(6), (12)
t62a	MCFA6–MCFA0, MSET10–MSET0, MTAG11–MTAG0, SNPINVV, SNPNCA, MAOE#, MBAOE#, SNPSTB# Setup Time	8		ns	10-3	(7a)
t62b	MCFA6–MCFA0, MSET10–MSET0, MTAG11-MTAG0, SNPINV, SNPNCA, MAOE#, MBAOE# Setup Time	1		ns	10-3	(7b)
t62c	MCFA6–MCFA0, MSET10–MSET0, MTAG11-MTAG0, SNPINV, SNPNCA, MAOE#, MBAOE#, SNPSTB# Setup Time	8		ns	10-3	(7c)
t63a	MCFA6–MCFA0, MSET10–MSET0, MTAG11–MTAG0, SNPINV, SNPNCA, MAOE#, MBAOE#, SNPSTB# Hold Time	1		ns	10-3	(7a)
t63b	MCFA6–MCFA0, MSET10–MSET0, MTAG11–MTAG0, SNPINV, SNPNCA, MAOE#, MBAOE# Hold Time	8		ns	10-3	(7b)
t63c	MCFA6–MCFA0, MSET10-MSET0, MTAG11–MTAG0, SNPINV, SNPNCA, MAOE#, MBAOE#, SNPSTB# Hold Time	1		ns	10-3	(7c)
t64	SNPSTB# Setup Time	8		ns	10-3	(8)
t65	SNPSTB# Hold Time	1		ns	10-3	(8)
t66	SNPSTB# Active/Inactive Time	8		ns	10-3	(9)
t67	MRO#, MKEN#, DRCTM#, MWB/WT# Setup Time	8		ns	10-3	
t68	MRO#, MKEN#, DRCTM#, MWB/WT# Hold Time	1		ns	10-3	
t69	MTHIT#, MHITM#, SNPBSY#, SNPCYC# Valid Delay	2	13	ns	10-2	
t69a	SNPCYC# Valid Delay	2	12	ns	10-2	

NOTES:
(1) Rise/fall times measured between 0.45V and 2.4V
(2) See capacitive derating curves for loads above the 50pF specification
(3) Valid delay from MAOE#, MBAOE# going active (low)
(4) Float delay from MAOE#, MBAOE# going inactive (high)
(5) Valid delay from MALE or MBALE if both MAOE#, MBAOE# are active
(6) Valid delay from CLK only if MALE or MBALE, MAOE# and MBAOE# are active
(7) a. In clocked mode referenced to SNPCLK rising edge
b. In strobed mode referenced to SNPSTB# falling edge
c. In synchronous mode, refer to CLK
(8) Asynchronous clocked mode only. Timings referenced to SNPCLK
(9) Asynchronous signal. Time to guarantee recognition on next clock
(10) SNPCLK is only used for the clocked memory bus mode
(11) t51 > t2
(12) This parameter is valid either from SNPCLK or CLK

Table 10-5. 82490XP Clocked Mode

Vcc = 5V ± 5%, Tcase = 0 to +85 °C Maximum CL = 50 pF unless otherwise specified. Minimum CL = 20 pF unless otherwise specified. All Inputs and Outputs are TTL Level.						
Symbol	**Parameter**	**Min**	**Max**	**Unit**	**Figure**	**Notes**
t38	MBRDY#, MSEL#, MEOC# Setup to MCLK	5		ns	10-3	
t39	MBRDY#, MSEL#, MEOC# Hold from MCLK	2		ns	10-3	
t40	MZBT#, MFRZ# Setup to MCLK	5		ns	10-3	
t41	MZBT#, MFRZ# Hold from MCLK	2		ns	10-3	
t42	MDATA Setup to MCLK	5		ns	10-3	
t43	MDATA Hold from MCLK	3		ns	10-3	
t44	MDATA Valid Delay from MCLK*MBRDY#	2	16	ns	10-2	
t45	MDATA Valid Delay from MCLK*MEOC#, MCLK*MSEL#	2	20	ns	10-2	
t46	MDATA Valid Delay from MOCLK	2	12	ns	10-2	

Table 10-6. 82490XP Strobed Mode

Vcc = 5V ± 5%, Tcase = 0 to +85 °C Maximum CL = 50 pF unless otherwise specified. Minimum CL = 20 pF unless otherwise specified. All Inputs and Outputs are TTL Level.						
Symbol	**Parameter**	**Min**	**Max**	**Unit**	**Figure**	**Notes**
t85	MISTB, MOSTB High Time	12		ns	10-6	
t86	MISTB, MOSTB Low time	12		ns	10-6	
t87	MEOC# High time	8		ns	10-6	
t88	MEOC# Low time	8		ns	10-6	
t89	MxSTB, MEOC# Rise time		2	ns		(1)
t90	MxSTB, MEOC# Fall time		2	ns		(1)
t91	MSEL# High time for restart	8		ns	10-6	
t92	MSEL# Setup before transition on MxSTB	5		ns	10-8	
t93	MSEL# Hold after transition on MxSTB	10		ns	10-8	
t92	MSEL# Hold after transition on MEOC#	2		ns	10-8	
t95	MxSTB transition to/from MEOC# falling transition	10		ns		
t96	MZBT# Setup to MSEL# or MEOC# falling edge	5		ns	10-7	
t97	MZBT# Hold from MSEL# or MEOC# falling edge	2		ns	10-7	
t98	MFRZ# Setup to MEOC# falling edge	5		ns	10-7	
t99	MFRZ# Hold from MEOC# falling edge	2		ns	10-7	
t100	MDATA Setup to MxSTB or MEOC# falling transition	5		ns	10-7	
t101	MDATA Hold from MxSTB or MEOC# falling transition	2		ns	10-7	
t102	MDATA Valid Delay from MxSTB transition	2	16	ns	10-9	
t103	MDATA Valid Delay from MEOC# falling transition or MSEL# deactivation	2	20	ns	10-9	

NOTE:
(1) Rise/Fall times are measured between 0.8V and 2.0V

Table 10-7. Test Mode

Vcc = 5V ± 5%, Tcase = 0 to +85 °C Maximum CL = 50 pF unless otherwise specified. Minimum CL = 20 pF unless otherwise specified. All Inputs and Outputs are TTL Level.						
Symbol	**Parameter**	**Min**	**Max**	**Unit**	**Figure**	**Notes**
t120	TCK Frequency		25	MHz		1x clock
t121	TCK Period	40		ns		(2)
t122	TCK High Time	10		ns		@ 2.0V
t123	TCK Low Time	10		ns		@ 0.8V
t124	TCK Rise Time		4	ns		(1)
t125	TCK Fall Time		4	ns		(1)
t126	TDI, TMS Setup Time	8		ns	10-10	
t127	TDI, TMS Hold Time	7		ns	10-10	
t128	TDO Valid Delay	3	25	ns	10-10	
t129	TDO Float Delay					
t130	All Outputs Valid Delay	3	25	ns	10-10	(3)
t131	All Outputs Float Delay		36	ns	10-10	(3)

NOTES:
(1) Rise/Fall times are measured between 0.8V and 2.0V Rise/Fall times can be relaxed by 1ns per 10ns increase in TCK period
(2) TCK period ≥ CLK period
(3) Parameter measured from TCK

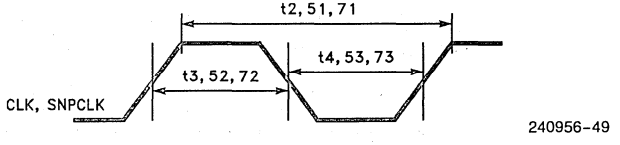
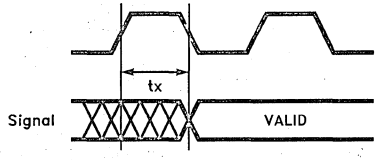


Figure 10-1. Clock Waveform



tx = t16, 32, 33, 35, 36, 44, 45, 60, 69

Figure 10-2. Valid Delay Timings

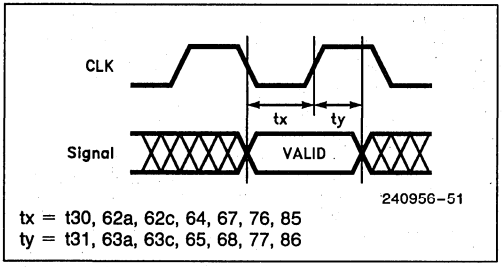


Figure 10-3. Setup and Hold Timings

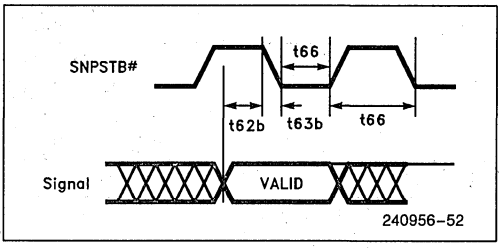


Figure 10-3a. Setup and Hold Timings in Strobed Snooping Mode

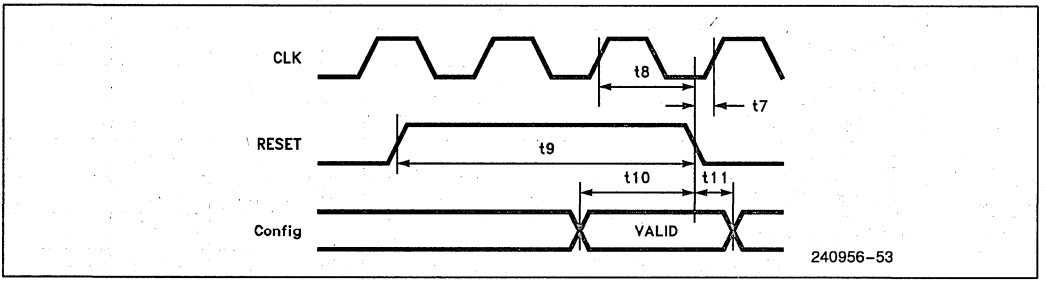


Figure 10-4. Reset and Configuration Timings

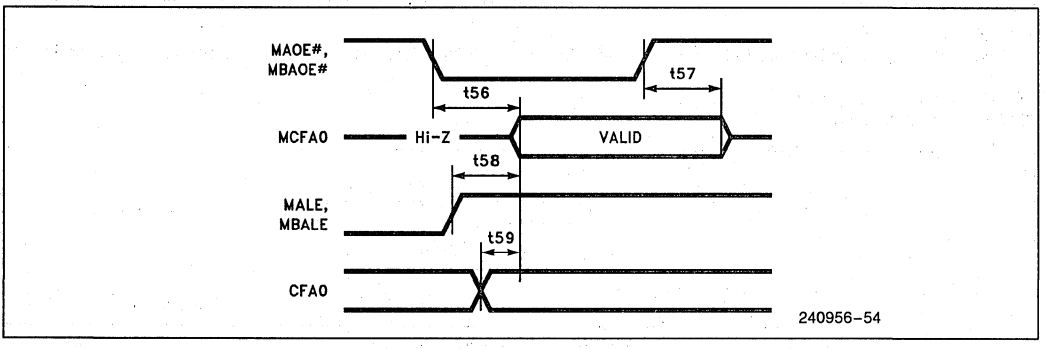


Figure 10-5. Memory Interface Signals

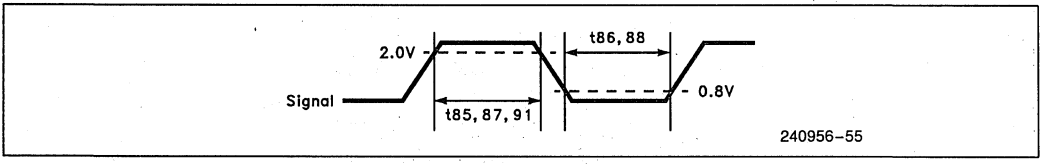


Figure 10-6. Active/Inactive Timing

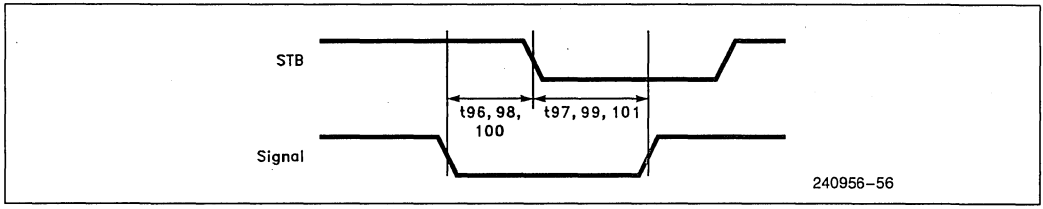


Figure 10-7. Setup and Hold Timing

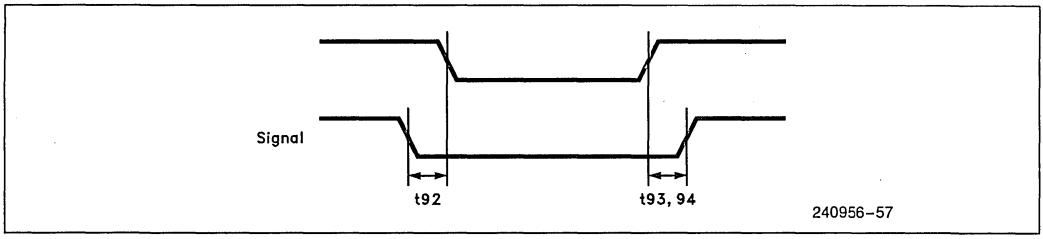


Figure 10-8. Setup and Hold Timing

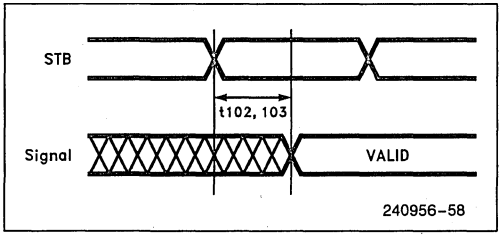


Figure 10-9. Valid Delay Timing

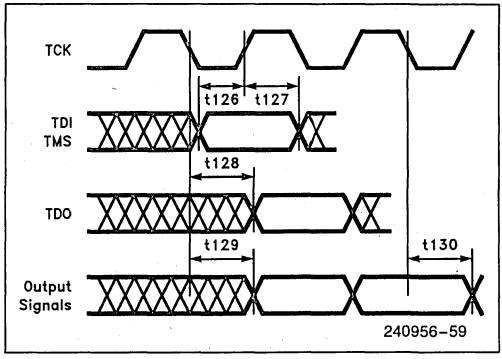


Figure 10-10. Test Timings

intel®

APPLICATION NOTE

AP-434

November 1989

Using i860™ Microprocessor Graphics Instructions for 3-D Rendering

Order Number: 240856-001

USING i860™ MICROPROCESSOR GRAPHICS INSTRUCTIONS FOR 3-D RENDERING

CONTENTS

2

FIGURES

TABLES

EXAMPLES

Introduction

The i860™ 64-bit microprocessor is a general-purpose CPU with on-chip integer unit, floating point, memory management, caches, and graphics. The i860 microprocessor supports 3-D graphics software with the following functions:

1. Hidden surface elimination
2. Distance interpolation
3. Intensity interpolation for 3-D shading

The **fzchks** (Z-buffer Check) and **pst** (Pixel Store) instructions expedite hidden surface elimination. Distance interpolation is accomplished with **faddz** (Add with Z merge), and intensity interpolation occurs with **faddp** (Add with Pixel Merge). The purpose of this application note is to illustrate the intended use of these instructions in a manner independent of any graphics environment in which the instructions might be used. It is not the purpose of this application note to present the most efficient instruction sequences. While the inner loop of Example 7 has as few instructions as logically possible, the other examples are intended to present general concepts, not optimum implementations. Tuning for maximum performance depends on the specific environment.

This application note assumes familiarity with the *i860™ 64-bit Microprocessor Programmer's Reference Manual* (Intel order number 240329); the i860 microprocessor instructions for graphics are detailed in section 6.6.

1.0 3-D RENDERING

This series of examples are routines that might be used at the lowest level of a graphics software system to convert a machine-independent description of a 3-D image into values for the frame buffer of a color video display. Typically, higher-level graphics routines represent an object as a set of polygons that together roughly describe the surfaces of the objects to be displayed. The graphics system maintains a database that describes these polygons in terms of their colors, properties of reflectance or translucence, and the locations in 3-D space of their vertices. Due to the roughness of the representation, the amount of information in the database is considerably less than that which must be delivered to the video display. A rendering procedure, such as Example 7, uses interpolation to derive the detailed information needed for each pixel in the graphics frame buffer. The rendering procedure also performs pixel-by-pixel hidden-surface elimination.

The focus of this series of examples is Example 7, which operates on a segment of a scan line. The segment is bounded by two points of given location and color: from point (*X1, Y0, Z1*) with color intensities *Red1, Grn1, Blu1* to point (*X2, Y0, Z2*) with color intensities *Red2, Grn2, Blu2*. The points and color intensities are determined by higher-level graphics software. The points represent the intersection of the scan line with two edges of the projected image of a polygon. For a given scan line, the rendering procedure is executed once for each polygon that projects onto that scan line. The higher-level graphics software is responsible for orienting the objects with respect to the viewer, for making perspective calculations, for scaling, and for determining the amount of light that falls on each polygon vertex.

The 16-bit pixel format is used, giving ample resolution for color shading: 2^6 intensity values for red, 2^6 intensity values for green, and 2^4 intensity values for blue. Example 1 shows how to set the pixel size. For hidden-surface elimination, the Z-buffer (or depth buffer) technique is employed, each Z value having a resolution of 16-bits.

Because the examples presented here use almost all of the registers of the i860 microprocessor, the registers are given symbolic names, as defined by Example 2. In a real application, it is likely that some of the inputs to the rendering procedure would be passed in floating-point registers instead of the integer registers employed here. The register allocation shown in Example 2 simplifies the examples by avoiding the need to use any register for multiple purposes.

```
// SET PIXEL SIZE TO 16
    ld.c        psr,    Ra     // Work on psr
    andnoth     0x00C0, Ra, Ra// Clear PS
    orh         0x0040, Ra, Ra// PS = 16-bit pixels
    st.c        Ra,     psr    //
```

Example 1. Setting Pixel Size

```
// REGISTER DEFINITIONS FOR RENDERING PROCEDURE
//      INTEGER LOCALS
    Ra    = r4    // Temporary
    Rb    = r5    // Temporary
    Rc    = r6    // Temporary
    Rd    = r7    // Temporary
//      INTEGER INPUTS
    Xl    = r16   // X coordinate of starting point of line segment in pixels
    dX    = r17   // Width of scan line segment in number of pixels
    ZBP   = r18   // Z-buffer pointer to the current line segment
    Zl    = r19   // Initial Z value, fixed-point 16.16 format
    mZ    = r20   // Z slope, fixed-point 16.16 format
    FBP   = r21   // Graphics frame buffer pointer to the current line segment
    Redl  = r22   // Initial red intensity, fixed-point 6.10 format, plus .5
    Grnl  = r23   // Initial green intensity, fixed-point 6.10 format, plus .5
    Blul  = r24   // Initial blue intensity, fixed-point 6.10 format, plus .5
    mR    = r25   // Red slope, fixed-point 6.10 format
    mG    = r26   // Green slope, fixed-point 6.10 format
    mB    = r27   // Blue slope, fixed-point 6.10 format
//      REAL LOCALS
    aZ    = f2    // Accumulated Z values
    aZh   = f3    //
    iZl   = f4    // Z interpolant, coefficient 1.0
    iZlh  = f5    //
    iZ3   = f6    // Z interpolant, coefficient 3.0
    iZ3h  = f7    //
    oldz  = f8    // Original values from the Z-buffer
    newz  = f10   // New Z-buffer values
    newzh = f11   //
    newi  = f12   // New pixel values
    iR    = f14   // Red interpolant, coefficient 4.0
    iRh   = f15   //
    aR    = f16   // Accumulated red intensities
    aRh   = f17   //
    iG    = f18   // Green interpolant, coefficient 4.0
    iGh   = f19   //
    aG    = f20   // Accumulated green intensities
    aGh   = f21   //
    iB    = f22   // Blue interpolant, coefficient 4.0
    iBh   = f23   //
    aB    = f24   // Accumulated blue intensities
    aBh   = f25   //
    lZmask   = f26   // left-end Z mask
    lZmaskh  = f27   //
    rZmask   = f28   // right-end Z mask
    rZmaskh  = f29   //
```

Example 2. Register Assignments

2.0 DISTANCE INTERPOLATION

To perform hidden surface elimination at each pixel, the rendering routine first interpolates the value of Z at each pixel. Distance interpolation consists of calculating the slope of Z over the given line segment, then increasing the Z value of each successive pixel by that amount, starting from *X1*. The width of the line segment in pixels is . . .

$$dX = X2 - X1$$

Calculate the reciprocal of *dX*:

$$RdX = 1/dX$$

The value of *dX* is used several times as a divisor. It is most efficient to calculate its reciprocal once, then, instead of dividing by *dX*, multiply by *RdX*. The slope of Z is . . .

$$mZ = (Z2 - Z1)^*RdX$$

Because each polygon is a plane, the value of *mZ* is constant for all scan lines that intersect the polygon; therefore *mZ* needs to be calculated only once for each polygon. Example 7 assumes that *dX* and *mZ* have already been calculated, and all that remains is to apply *mZ* to successive pixels. Let *Z(Xn)* be the Z value at pixel *Xn*. Then . . .

$$Z(X1) = Z1$$
$$Z(X1 + 1) = Z1 + mZ$$
$$Z(X1 + 2) = Z1 + 2^*mZ$$
$$\vdots$$
$$Z(X1 + N) = Z1 + N^*mZ$$
$$\vdots$$
$$Z(X1 + dX) = Z1 + dX^*mZ = Z(X2)$$

Figure 1 illustrates this Z-value interpolation.

The **faddz** instruction helps to perform the above calculations 64 bits at a time. Because a Z value is 16 bits wide, Example 7 operates on the Z buffer in groups of four. The **faddz** instruction, however, treats the interpolation values (*N*mZ*) as 32-bit fixed-point numbers; therefore, two **faddz** instructions are executed for each group of four pixels. Because of the way the **faddz** shifts

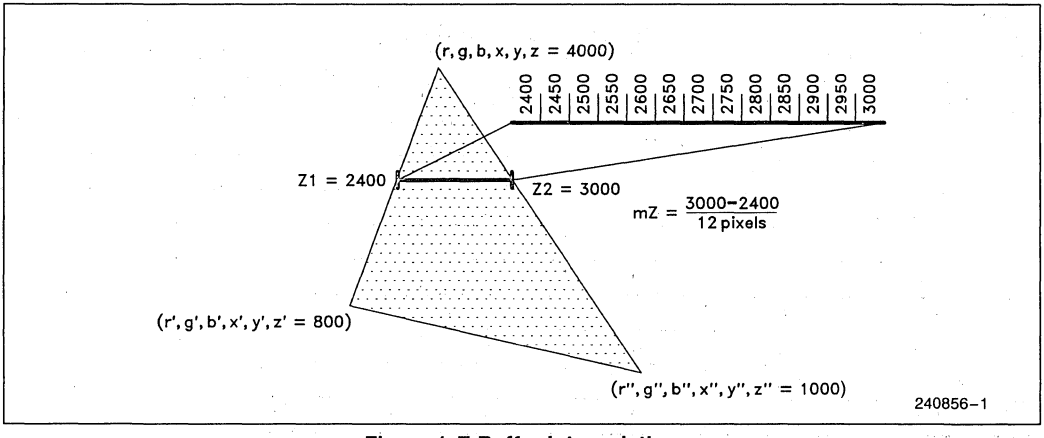


Figure 1. Z-Buffer Interpolation

the MERGE register, the first **faddz** corresponds to even-numbered pixels, while the second corresponds to odd-numbered pixels. Instead of starting with the value for the first pixel (*Z(X1)*) and adding *mZ* to each pixel to produce the value for the next pixel, the example procedure starts with the values for the first two even-numbered pixels and adds 1**mZ* to each of these values to produce the values for the adjacent odd-numbered pair. Adding 3**mZ* to each of the Z values of an odd-numbered pair produces the values for the next even-numbered pair. Figure 2 shows one way of constructing the operands before starting the distance interpolations. (The initial value given to *src1* depends on the alignment of the first pixel.) Table 1 helps to visualize the process.

After two **faddz** instructions, the MERGE register holds the Z values for four adjacent pixels (in the correct order). The **form** instruction copies MERGE into one of the 64-bit floating-point registers.

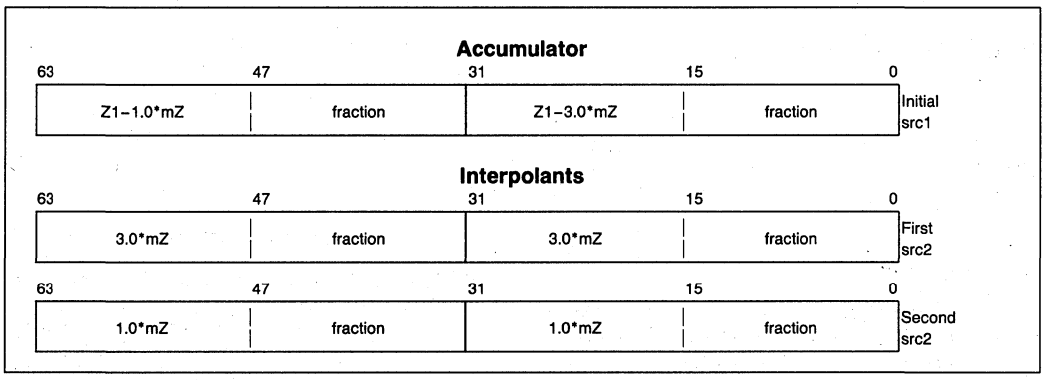


Figure 2. faddz Operands

Table 1. faddz Visualization

Operands	63-32	31-0	MERGE Register			
			63-48	47-32	31-16	15-0
src1	−1.0	−3.0				
src2	3.0	3.0				
rdest/src1	2.0	0.0	2		0	
src2	1.0	1.0				
rdest/src1	3.0	1.0	3	2	1	0
src2	3.0	3.0				
rdest/src1	6.0	4.0	6		4	
src2	1.0	1.0				
rdest/src1	7.0	5.0	7	6	5	4
src2	3.0	3.0				
rdest/src1	10.0	8.0	10		8	
src2	1.0	1.0				
rdest/src1	11.0	9.0	11	10	9	8
src2	3.0	3.0				
rdest/src1	14.0	12.0	14		12	
src2	1.0	1.0				
rdest	15.0	11.0	15	14	13	12

Because the values of *Z1* and *mZ* are constant for each loop through the rendering routine, the numbers shown here are the values of the coefficient *N*, where the actual operands have the values *Z1* + *N***mZ*. For each execution of **faddz**, *src1* is the same as *rdest* of the prior **faddz**. After every two **faddz** instructions, a **form** instruction empties the MERGE register.

```
// CONSTRUCT INTERPOLANTS iZ1 AND iZ3 GIVEN mZ
    ixfr        mZ,     iZ1         // Join each half in 64-bit register
    shl         1,      mZ,     Ra  // Ra = 2*mZ
    adds        Ra,     mZ,     Ra  // Ra = 3*mZ
    ixfr        Ra,     iZ3         // Join each half in 64-bit register
    fmov.ss     iZ1,    iZ1h        // Join each half in 64-bit register
    fmov.ss     iZ3,    iZ3h        // Join each half in 64-bit register
```

Example 3. Construction of Z Interpolants

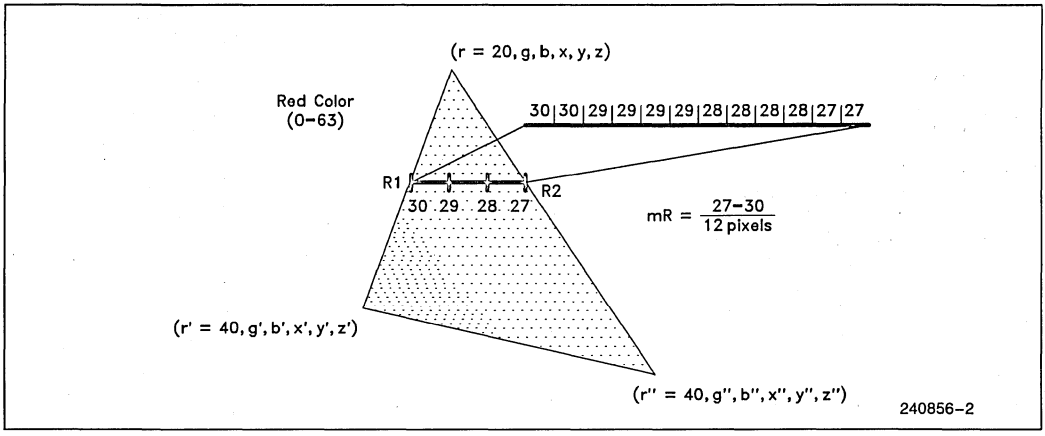


Figure 3. Pixel Interpolation for Gouraud Shading

The same register is used as both *src1* and *rdest* in all **faddz** instructions. This register serves to accumulate Z values for successive pixels; therefore, it is called an *accumulator.* The registers used as *src2* are called *interpolants.* The code in Example 3 constructs the interpolants; it needs to be executed only once for each polygon.

3.0 COLOR INTERPOLATION

To determine the RGB color intensities at each pixel, the rendering routine interpolates between the color intensities at the end points. (This rendering technique is called "Gouraud shading" after H. Gouraud, "Continuous Shading of Curved Sufaces," *IEEE Transactions on Computers,* C-20(6), June 1971, pp. 623-628.) Let the symbol C (color) represent either R (red), G (green), or B (blue). Color interpolation consists of calculating the slope of C over the given line segment, then increasing the C values of each successive pixel by that amount, starting from the values for *X1.* This must be done for C=R, C=G, and C=B. The slope of C is . . .

$$mC = (C2 - C1)*RdX$$

. . . where $RdX = 1/dX$

The value of *mC* is constant for all scan lines that intersect a given pair of polygon edges; therefore *mC* needs to be calculated only once for each such pair. Example 7 assumes that *mC* has already been calculated for all colors, and all that remains is to apply *mC* to successive pixels. Let *C(Xn)* be a C value at pixel *Xn.* Then . . .

$$C(X1) = C1$$
$$C(X1 + 1) = C1 + mC$$
$$C(X1 + 2) = C1 + 2*mC$$
$$\vdots$$
$$C(X1 + N) = C1 + N*mC$$
$$\vdots$$
$$C(X1 + dX) = C1 + dX*mC = C(X2)$$

Figure 3 illustrates Gouraud shading of a triangle.

The **faddp** instruction performs the above calculations 64 bits at a time. Because a pixel is 16 bits wide, Example 7 operates on pixels in groups of four. Instead of starting with the value for the first pixel (*C(X1)*) and adding *mC* to each pixel to produce the value for the next pixel, the example procedure starts with the values for the first four pixels and adds $4*mC$ to each group of

four to produce the values for the next four. Three **faddp** instructions are executed for each group of four pixels. The first increments the blue values; the second, green; the third, red. Figure 4 shows one way of constructing the operands for each color before starting the color interpolations. (The initial value given to *src1* depends on the alignment of the first pixel.)

Setup of the accumulator and interpolants is similar to that of the Z-buffer. The code in Example 4 constructs the interpolants; it needs to be executed only once for each pair of edges in each polygon.

4.0 BOUNDARY CONDITIONS

The i860 microprocessor operates on 64-bit quantities that are aligned on 8-byte boundaries. The code in this example takes full advantage of this design, handling four 16-bit pixels in each loop. However, if the first or last pixel of a line segment is not on an 8-byte boundary, two kinds of special considerations are required:

1. Masking of Z values near the end points.
2. Initialization of the accumulators.

4.1 Z-Buffer Masking

When either the first or last pixel of the line segment is not at an 8-byte boundary, the rendering procedure must mask the first or last set of new Z-buffer values (**newz**) so that the Z-buffer and the frame buffer are not erroneously updated. Sometimes both the first and last pixels are in the same 4-pixel set, in which case either one may not be on an 8-byte boundary. A function that looks up and calculates masks is shown in Example 5.

Because the value 0xFFFF is used for masking, the Z-buffer is initialized with 0xFFFE, so that the **fzchks** instruction always finds the mask to be greater than any Z-buffer contents.

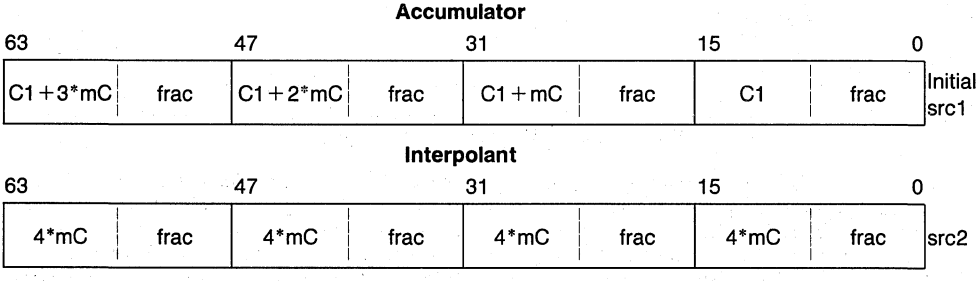


Figure 4. faddp Operands

```
// CONSTRUCT INTERPOLANTS iR, iG, iB GIVEN mR, mG, mB
    shl         18,     mR,     Ra  // Multiply each color slope by four, then
    shl         18,     mG,     Rb  //  shift by 16 to put the significant
    shl         18,     mB,     Rc  //  bits into the high-order half
    shr         16,     Ra,     mR  // Return significant 16 bits
    shr         16,     Rb,     mG  //  to low-order half. Any sign bits
    shr         16,     Rc,     mB  //  in high-order half are gone.
    or          mR,     Ra,     Ra  // Join 16-bit quarters
    or          rG,     Rb,     Rb  // in 32-bit register
    or          mB,     Rc,     Rc  //
    ixfr        Ra,     iR          // Join 32-bit halves
    ixfr        Rb,     iG          // in 64-bit register
    ixfr        Rc,     iB          //
    fmov.ss     iR,     iRh         //
    fmov.ss     iG,     iGh         //
    fmov.ss     iB,     iBh         //
```

Example 4. Construction of Color Interpolants

```
.macro zmask l_align, r_align, Rx, Ry
// l_align, r_align - left- and right-end alignment [0..3] in 2-byte units
// Rx, Ry           - scratch registers
    .data
    .align 8
left_mask:: //low        high
    .long 0x00000000, 0x00000000    // 0 mod 4
    .long 0x0000FFFF, 0x00000000    // 1 mod 4
    .long 0xFFFFFFFF, 0x00000000    // 2 mod 4
    .long 0xFFFFFFFF, 0x0000FFFF    // 3 mod 4
right_mask:://low        high
    .long 0xFFFF0000, 0xFFFFFFFF    // 0 mod 4
    .long 0x00000000, 0xFFFFFFFF    // 1 mod 4
    .long 0x00000000, 0xFFFF0000    // 2 mod 4
    .long 0x00000000, 0x00000000    // 3 mod 4

    .text
    shl     3,  l_align,    l_align // Multiply by 8
    mov     left_mask,      Rx      //
    fld.d   l_align (Rx),   lZmask  // Load 8-byte mask

    shl     3,  r_align,    r_align // Multiply by 8
    mov     right_mask,     Rx      //
    fld.d   r_align (Rx),   rZmask  // Load 8-byte mask
// If the first and last pixels are contained in the same 64-bit
// aligned set, then lZmask = lZmask OR rZmask.
    andh    0x8000, dX,     r0      // Is dX negative
    bc      L2                      // If not, right end is in other set
    fxfr    lZmask, Rx              //
    fxfr    rZmask, Ry              //
    or      Rx,     Ry,     Rx      // OR low-order half
    ixfr    Rx,     lZmask          //
    fxfr    lZmaskh,        Rx      //
    fxfr    rZmaskh,        Ry      //
    or      Rx,     Ry,     Rx      // OR high-order half
    ixfr    Rx,     lZmaskh         //
L2: nop                             //
    .endm
```

Example 5. Z Mask Procedure

Table 2. Accumulator Initial Values

Alignment	Initial Z Accumulator Values	
0	Z1 − 1*mZ	Z1 − 3*mZ
2	Z1 − 2*mZ	Z1 − 4*mZ
4	Z1 − 3*mZ	Z1 − 5*mZ
6	Z1 − 4*mZ	Z1 − 6*mZ

Alignment	Initial Color Accumulator Values C = R, G, B			
0	C1 − 1*mC	C1 − 2*mC	C1 − 3*mC	C1 − 4*mC
2	C1 − 2*mC	C1 − 3*mC	C1 − 4*mC	C1 − 5*mC
4	C1 − 3*mC	C1 − 4*mC	C1 − 5*mC	C1 − 6*mC
6	C1 − 4*mC	C1 − 5*mC	C1 − 6*mC	C1 − 7*mC

Table 3. Accumulator Initialization Table

Alignment	Table Values			
	*mZ	*mR	*mG	*mB
0	−1, −3	−1, −2, −3, −4	−1, −2, −3, −4	−1, −2, −3, −4
2	−2, −4	−2, −3, −4, −5	−2, −3, −4, −5	−2, −3, −4, −5
4	−3, −5	−3, −4, −5, −6	−3, −4, −5, −6	−3, −4, −5, −6
6	−4, −6	−4, −5, −6, −7	−4, −5, −6, −7	−4, −5, −6, −7

4.2 Accumulator Initialization

When the first pixel of the line segment is not at an 8-byte boundary, initial values placed in the accumulators (*aZ*, *aB*, *aG*, and *aR*) must be selected so that *Z1*, *Red1*, *Grn1*, and *Blu1* correspond to the correct pixel. The desired result is that shown by Table 2. However, each value is a composite of two terms: one that is constant for each edge pair (*n*mZ*, *n*mR*, *n*mG*, *n*mB*) and one that can vary with each scan line (*Z1*, *Red1*, *Grn1*, *Blu1*). The example assumes that the constant values have all been calculated and stored in a memory table of the format shown by Table 3. At the beginning of each line segment the values appropriate to the alignment of the line segment are retrieved from the table and added to the initial Z and color values, as shown in Example 6.

5.0 THE INNER LOOP

Once the proper preparations have been made, only a minimal amount of code is needed to render each scan-line segment of a polygon. The code shown in Example 7 operates on four pixels in each loop. The left and right ends of the line segment go through different logic paths so that the Z-buffer masks can be applied by the **form** instruction. All the interior points are handled by the tight inner loop.

The controlling variable **dX** is zero-relative and is expressed as a number of pixels. The value of **dX** also indicates alignment of the end-points with respect to the 4-pixel groups. Unaligned left-end pixels are subtracted from **dX** before entering the inner loop; therefore, subsequent values of **dX** indicate the alignment of the right end. A value that is 3 mod 4 indicates that the right end is aligned, which explains the test for a value of −5 near the end of the loop (−5 mod 4 = 3). The fact that the value −5 is loaded into register **Rb** on every execution of the loop does not represent a programming inefficiency, because there is nothing else for the core unit to do at that point anyway.

```
// ACCUMULATOR INITIALIZATION TABLE
    .data; .align .double
acc_init_tab::  .double [16] 0
    .dsect
aBi:    .double     // Four initial 16-bit blue values
aGi:    .double     // Four initial 16-bit green values
aRi:    .double     // Four initial 16-bit red values
aZi:    .double     // Two initial 32-bit Z values
    .end
    .text
// INITIALIZE ACCUMULATORS
.macro acc_init  Lalign, Rtab, Rx, Ry, Fx, Fxh
//  Lalign - left-end alignment (0..3) in two-byte units
//  Rtab   - register to use for addressing the table
//  Rx, Ry, Fx, Fxh - scratch registers
    mov         acc_init_tab,   Rtab    //
    shl,        5,      Lalign, Lalign  // Multiply by row width
    adds        Lalign, Rtab,   Rtab    // Index row corresponding to alignment
    fld.d       aZi(Rtab),      aZ      // Z
    ixfr        Zl,     Fx              // Z
    fld.d       aRi(Rtab),      aR      // R-Load constant values
    shl         16,     Redl,   Rx      // R-Shift starting value to hi-order
    fmov.ss     Fx,     Fxh             // Z
    shr         16,     Rx,     Ry      // R-Redl stripped of sign bits
    fiadd.dd    Fx,     aZ,     aZ      // Z
    or          Rx,     Ry,     Ry      // R-Form (Redl,Redl)
    ixfr        Ry,     Fx              // R-Put in 64-bit register
    fld.d       aGi(Rtab),      aG      // G
    shl         16,     Grnl,   Rx      // G
    fmov.ss     Fx,     Fxh             // R-Form (Redl,Redl,Redl,Redl)
    shr         16,     Rx,     Ry      // G
    fiadd.dd    Fx,     aR,     aR      // R-Add variables to constants
    or          Rx,     Ry,     Ry      // G
    ixfr        Ry,     Fx              // G
    fld.d       aBi(Rtab),      aB      // B
    shl         16,     Blul,   Rx      // B
    fmov.ss     Fx,     Fxh             // G
    shr         16,     Rx,     Ry      // B
    fiadd.dd    Fx,     aG,     aG      // G
    or          Rx,     Ry,     Ry      // B
    ixfr        Ry,     Fx              // B
    fmov.ss     Fx,     Fxh             // B
    fiadd.dd    Fx,     aB,     aB      // B
.endm
```

Example 6. Accumulator Initialization

```
//  RENDERING PROCEDURE
//      16-bit pixels, 16-bit Z-buffer
    and         3,      Xl,     Ra  // Determine alignment of starting-point
    acc_init  Ra, Rb, Rc, Rd, Fa, Fah   // Initialize accumulators
    subs        4,      Ra,     Rb  // 4 - alignment
    subs        dX,     Rb,     dX  // Adjust dX by Xl alignment
    // If dX <= 0, then right end is in same set as left end
    and         3,      dX,     Rb  // Determine alignment of right end
    zmask       Ra, Rb, Rc, Rd      // Prepare both left- and right-end masks
left_end::  // Handle boundary conditions
    d.faddz     aZ,     iZ3,    aZ  // Interpolate 2 even Z values
      adds      -8,     FBP,    FBP // Anticipate autoincrement
    d.faddz     aZ,     iZl,    aZ  // Interpolate 2 odd Z values
      adds      -8,     ZBP,    ZBP // Anticipate autoincrement
    d.form      lZmask, newz        // Mask 4 new Z values
      fld.d     8(ZBP), oldz        // Fetch 4 old Z values
    d.faddp     aB,     iB,     aB  // Interpolate 4 blue intensities
      mov       -4,     Ra          // Loop increment: 4 pixels
    d.faddp     aG,     iG,     aG  // Interpolate 4 green intensities
      adds      -4,     dX,     dX  // Prepare dX for bla at end of loop
    d.faddp     aR,     iR,     aR  // Interpolate 4 red intensities
      bla       Ra,     dX,     Ll  // Initialize LCC
    d.form      f0,     newi        // Move 4 new pixels to 64-bit reg
      adds      5,      dX,     r0  // Are there any whole sets (dX < -5)?
Ll: d.fzchks    oldz,   newz,   newz// Mark closer points in PM[7..4]
      bc        short_segment       // Get out now if no whole set
    d.fnop                          //
      fld.d     16(ZBP),    oldz    // Fetch 4 old Z values
inner_loop:: // Handle all interior points
    d.faddz     aZ,     iZ3,    aZ  // Interpolate 2 even Z values
      nop                           //
    d.faddz     aZ,     iZl,    aZ  // Interpolate 2 odd Z values
      fst.d     newz,   8(ZBP)++    // Update Z buf from prior loop
    d.form      f0,     newz        // Move 4 new Z values to 64-bit reg
      nop                           //
    d.fzchks    f0,     f0,     f0  // Shift PM[7..4] to PM[3..0]
      mov       -5,     Rb          // -5 mod 4 = 3, aligned right end
    d.faddp     aB,     iB,     aB  // Interpolate 4 blue intensities
      pst.d     newi,   8(FBP)++    // Store pixels indicated by PM[3..0]
    d.faddp     aG,     iG,     aG  // Interpolate 4 green intensities
      xor       Rb,     dX,     r0  // Are we at an aligned right end?
    d.faddp     aR,     iR,     aR  // Interpolate 4 red intensities
      bc        aligned_end         // Taken if at an aligned right end
    d.form      f0,     newi        // Move 4 new pixels to 64-bit reg
      bla       Ra, dX, inner_loop  // Loop if not at end of line segment
    d.fzchks    oldz,   newz,   newz// Mark closer points in PM[7..4]
      fld.d     16(ZBP),    oldz    // Fetch 4 old Z values for next loop
// End of inner_loop.  Right end not aligned
```

Example 7. 3-D Rendering (1 of 2)

```
right_end:: // Handle boundary conditions
    d.faddz     aZ,      iZ3,     aZ  // Interpolate 2 even Z values
      nop                             //
    d.faddz     aZ,      iZ1,     aZ  // Interpolate 2 odd Z values
      fst.d     newz,    8(ZBP)++     // Update Z buf from prior loop
    d.form      rZmask,  newz         // Mask 4 new Z values
      nop                             //
    d.fzchks    f0,      f0,      f0  // Shift PM[7..4] to PM[3..0]
      nop                             //
    d.faddp     aB,      iB,      aB  // Interpolate 4 blue intensities
      pst.d     newi,    8(FBP)++     // Store pixels indicated by PM[3..0]
    d.faddp     aG,      iG,      aG  // Interpolate 4 green intensities
      nop                             //
    d.faddp     aR,      iR,      aR  // Interpolate 4 red intensities
      nop                             //

aligned_end::  // No special boundary conditions
    d.form      f0,      newi         // Move 4 new pixels to 64-bit reg
      br        wrap_up               //
    d.fzchks    oldz,    newz,    newz// Mark closer points in PM[7..4]
      nop                             //

short_segment::
    d.fnop                            //
      adds      8,       dX,      r0  // Is right end in same set as left?
    d.fnop                            //
      bnc.t     right_end             // Branch taken if no.
    d.fnop                            //
      fld.d     16(ZBP),     oldz     // Fetch 4 old Z values

wrap_up:: // Store the unstored and leave dual mode.
    fzchks      f0,      f0,      f0  // Shift PM[7..4] to PM[3..0]
      fst.d     newz,    8(ZBP)++     // Update Z buf from prior loop
    fnop
      pst.d     newi,    8(FBP)++     // Store pixels indicated by PM[3..0]
```

Example 7. 3-D Rendering (2 of 2)

6.0 ALTERNATIVE IMPLEMENTATIONS

Example 8 contrasts the inner loop of the 16-bit pixel rendering procedure with that of an 8-bit procedure. For 8-bit pixels, two **faddp** instructions accomplish 64-bits of pixel intensity interpolation; there is no need to maintain three separate color accumulators. Four **faddz** instructions (rather than two) are required, because eight Z values are created for the eight pixels per loop.

```
      // 8-bit Pixels, 16-Bit Zbuffer = 8 Pixels in 15 Clocks
      //    G-Unit                    |          Core Unit
inner_loop::                          ;
      d.faddz  aZ,deltaZ1,aZ          ;     fld.q  16(ZBP),oldZ_A
      d.faddz  aZ,deltaZ2,aZ          ;     nop
      d.form   f0,newZ_A              ;     nop
      d.faddz  aZ,deltaZ1,aZ          ;     andh   0x8000,dX, r0
      d.faddzz aZ,deltaZ2,aZ          ;     bnc    rightend
      d.form   f0,newZ_B              ;     nop
      d.fzchks oldZ_A,newZ_A,newZ_A   ;     nop
      d.fzchks oldZ_B,newZ_B,newZ_B   ;     nop
      d.faddp  intens,dI,intens       ;     fst.q  newZ_A ,16(ZBP)++
      d.faddp  intens,dI2,intens      ;     bte    0,dX,end
      d.form   f0,newi                ;     bla    neg8,dX,inner_loop
      d.fnop                          ;     pst.d  newi,8(FBP)++
      //-------------------------------------------------------------

      // 16-Bit Pixels, 16-Bit Zbuffer = 4 Pixels in 10 Clocks
      //    G-Unit                    |          Core Unit
inner_loop::                          ;
      d.faddz  aZ,iz3,aZ              ;     nop
      d.faddz  aZ,iz1,aZ              ;     fst.d  newz,8(ZBP)++
      d.form   f0,newz                ;     nop
      d.fzchks f0,f0,f0               ;     mov    -5,Rb
      d.faddp  aB,iB,aB               ;     pst.d  newi,8(FBP)++
      d.faddp  aG,iG,aG               ;     xor    Rb,dX,r0
      d.faddp  aR,iR,aR               ;     bc     aligned_end
      d.form   f0,newi                ;     bla    neg4,dX,inner_loop
      d.fzchks oldz,newz,newz         ;     fld.d  16(ZBP),oldz
      //-------------------------------------------------------------
```

Example 8. Inner Loop of Renderers for Two Pixel Sizes

intel®

APPLICATION NOTE

PRELIMINARY
AP-435

March 1990

2

FAST Fourier Transforms on the i860™ Microprocessor

MARK ATKINS
APPLICATIONS ENGINEER

Order Number: 240658-001

FAST FOURIER TRANSFORMS ON THE i860™ MICROPROCESSOR

CONTENTS

CONTENTS

ABSTRACT

The i860 Processor computes floating-point results rapidly, lending itself to DSP (digital signal processing) as well as general-purpose computing. With this high performance, DSP functions can be added to any system containing an i860 CPU. A Fast Fourier Transform (FFT) illustrates this DSP power. Complete code for the FFT is presented in this application note, as well as performance measurements. Both complex and real input data FFTs are included, as well as both Decimation in Time and Decimation in Frequency.

1.0 INTRODUCTION TO FAST FOURIER TRANSFORMS

Discrete Fourier Transforms (DFTs) change time-domain data samples into a frequency-domain profile of the sampled signal. The frequency-domain representation consists of the magnitudes of sine waves at various frequencies, which would recreate the original data if superimposed. To accomplish the transform, a DFT adds combinations of the input data samples, after multiplying some of those inputs with weighting factors. The number of samples, "N", is usually a power of two.

Each result in the frequency domain comes from a weighted sum of all data samples. The weighting ("W") factors are called "twiddles", and are complex cosine/sine values for each particular frequency.

The FFT (Fast Fourier Transform) is an efficient implementation of the DFT, defined by:

x(n) = time domain samples of the signal, n = 0, 1, . . . N-1

X(k) = the Discrete Fourier Transform of x(n), k = 0,1, . . . N- 1

= a "frequency domain" equivalent of x(n)

$= \Sigma\ x(n) * W^{nk}$, n = 0 to N-1, and $W^{nk} = e^{-j2\pi nk/N}$, where $j = \sqrt{-1}$

$= \Sigma\ x(n) * (\cos(2\pi nk/N) - j * \sin(2\pi nk/N))$

The (N-1) complex adds and (N-1) complex multiplications required for each X(k) make the DFT an Order (N^2) computation. Fortunately, the FFT decomposes this to an Order ($N * \log_2 N$) algorithm by splitting the N-sum into units of 2-sums. These units are called "butterflies" because they produce 2 output values from 2 inputs, with the butterfly-shaped dataflow shown below. (Some FFT algorithms, called Radix-4, use 4-input, 4-output butterflies.) The butterfly calculations are executed in stages, with $\log_2 N$ stages and N/2 butterflies per stage.

The subdivision, or decimation, of the N-sum into butterflies can be done via two different methods: "Decimation in Time" (DIT) or "Decimation in Frequency" (DIF). The methods differ in the ordering of twiddles and the form of the butterfly arithmetic, but they yield the same answer. They are based on different mathematical derivations of the FFT: DIT results from recursively splitting the input time-domain samples into an even-indexed group and an odd-indexed, while DIF comes from splitting the DFT output frequency-domain points into odd/even groups.

2.0 BUTTERFLY DEFINED

Let A = the first input to the butterfly (complex number, composed of Real part AR and Imaginary part AI)

B = the second input to the butterfly (complex, BR and BI)

W = twiddle factor (also complex, WR and WI)

Anew = complex result #1, which overwrites A

Bnew = result #2, which overwrites B

For a "Decimation-in-Frequency" butterfly,

Anew = A+B

Bnew = (A − B) * W

The complex add, subtract, and multiply of a butterfly decompose into 4 real multiplies, 3 real adds, and 3 real subtracts:

AnewR = AR + BR tempR = AR-BR

AnewI = AI + BI tempI = AI-BI

BnewR = (tempR * WR) − (tempI * WI)

BnewI = (tempR * WI) + (tempI * WR)

For a "Decimation-in-Time" butterfly,

Anew = A + (B * W)

Bnew = A − (B * W)

The number of real operations remains 4 multiplies and 6 add/subtracts, but the equations differ and the multiplies must be done first:

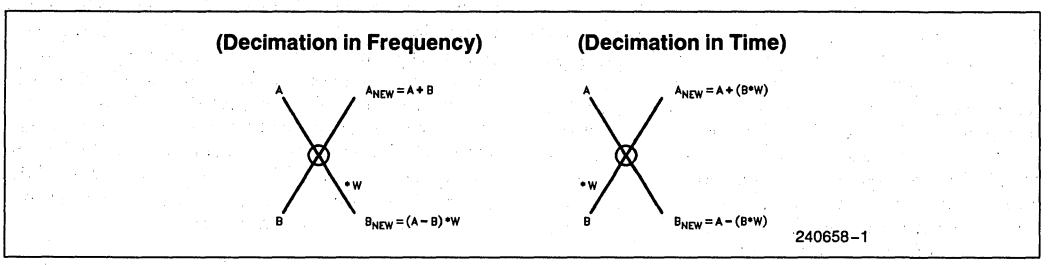
tempR = (WR * BR) − (WI * BI)

tempI = (WR * BI) + (WI * BR)

AnewR = AR + tempR BnewR = AR-tempR

AnewI = AI + tempI BnewI = AI-tempI

Butterfly Dataflow:



The stages, twiddles, and butterflies for 8-point FFTs are shown in Figures 1 and 2. For larger values of N, the dataflow patterns are very similar, with N/2 butterflies executed at each stage, and a greater number of stages. Refer to a text on Digital Signal Processing for a complete discussion of FFT design, such as chapter 6 of *Theory and Application of Digital Signal Processing* (see the Bibliography at the end of this note).

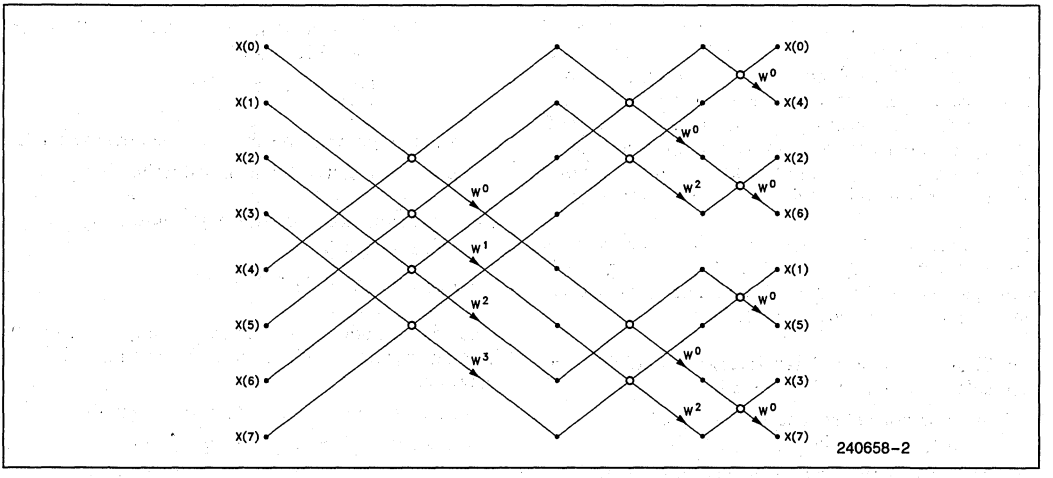


Figure 1. Decimation-In-Frequency FFT for 8 points

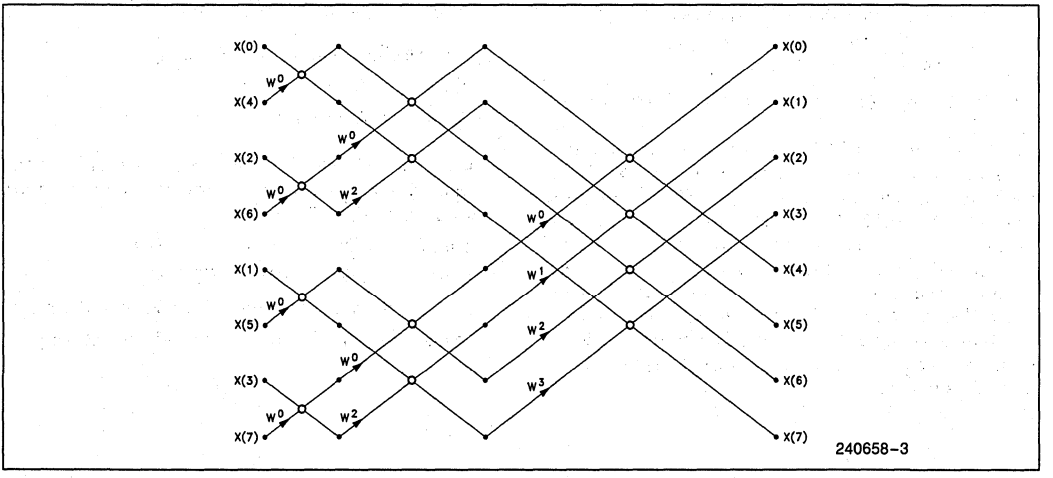


Figure 2. Decimation-In-Time FFT for 8 points

3.0 BIT REVERSAL

Due to their structure, FFT algorithms have the side-effect of scrambling the ordering of output data. For radix-2 FFTs, the output is in "bit-reversed" order—for example, the value for frequency one is NOT at location one in the output array, but at location N/2. Time to unscramble the output is often NOT included in FFT benchmarking, because scrambled output is fine for some signal-processing uses such as convolution. In any event, unscrambling consists of swapping the locations of pairs of output values. Alternatively, input values can be shuffled, as Decimation in Time usually does before the first stage (as shown in Figure 2). Otherwise, to avoid the shuffling of input in DIT, the twiddles must be accessed in bit-reversed order. As an example of bit-reversal, for 256 points the reordering involves:

SWAP X(i) and X(j), where i = 'klmnopqr'b and j = 'rqponmlk'b. The second index (j) contains the same bits as (i), but in opposite order.

4.0 FFT IMPLEMENTATION ON THE i860 CPU

Several features of the i860 CPU contribute to FFT performance. The floating-point multiplier and adder can simultaneously produce 1 product and 1 sum per cycle, using **Dual-Operation** FP instructions. To fetch the butterfly inputs and store outputs, **Dual-Instruction-Mode** allows a memory fetch or store simultaneous with the multiply and add. Four floating-point numbers can be stored by one instruction, using the 16-byte-operand "**fst.q**" instruction. Likewise, 16 bytes can be fetched from the data cache in one **fld.q** op.

The floating-point arithmetic of the i860 CPU conforms to IEEE 754 format, which some DSPs fail to do. Shown below is code for the crucial inner loop of the FFT:

2

```
//--------------------
//inner_loop: do 2 Decimation-In-Frequency FFT butterflies.
//  Twelve clocks for 2 butterflies - 12 FP add/sub, 8 multiplies,
//   6 8-byte loads, 4 8-byte stores.
//      FP-op                       ;     Core-op
inner_loop::
 d.r2pt.ss    WR,DI,BnewR          ; pfld.d  wind (wstart),WRo
 d.pfsub.ss   AR,BR,AnewRo         ; fld.d   8 (fetch)++,ARo
 d.ratls2.ss  AI,BI,AnewIo         ; fld.d   offset (fetch),BRo
 d.i2st.ss    WI,DR,BnewI          ; fst.q   AnewR,16(store)++
 d.ratlp2.ss  AR,BR,DR             ; adds    wincr,wind,wind
 d.ialp2.ss   AI,BI,DI             ; pfld.d  wind (wstart),WR
//--------------------
 d.r2pt.ss    WRo,DI,BnewRo        ; adds    wincr,wind,wind
 d.pfsub.ss   ARo,BRo,AnewR        ; fld.d   8 (fetch)++,AR
 d.ratls2.ss  AIo,BIo,AnewI        ; fld.d   offset (fetch),BR
 d.i2st.ss    WIo,DR,BnewIo        ; fst.q   BnewR, offset (store)
 d.ratlp2.ss  ARo,BRo,DR           ; bla     decrem,count,inner_loop
 d.ialp2.ss   AIo,BIo,DI           ; and     wlimit,wind,wind //modulo.
//--------------------
```

5.0 CODE DESIGN

Refer to the inner_loop above and code listings at the end of this application note for the discussions that follow. Refer to the *"i860™ 64-bit Microprocessor Programmer's Reference Manual"* (Intel order number 240329) for details on instructions and formats.

The programs include both assembly and Fortran components. Input data can number any power of 2 from 16 to 1024 points. The algorithms are radix-2, floating-point, in-place. Included in the listing are both Decimation-in-Time and Frequency, and both complex-input and real-input FFTs.

5.1 Cache Utilization

Because the instruction cache contains 4-Kbytes, all required code easily fits in cache. However, a 1024-point complex FFT fills the 8-Kbyte data cache with the input X() array. Thus the more rarely-used twiddle W() array is intentionally kept out of cache, as described in the "pfld" section.

A subroutine ("fetch.ss") is used to move the input data array efficiently into cache for the 1024-point FFT. "Fetch" allows all data to be brought into cache using the next-near (NENE#) accesses to DRAM. Without that routine, getting A and B from locations separated by 4 Kbytes (NOT the same DRAM page) makes fetches and writebacks from DRAM for the first stage slower, and adds 30% to overall execution time.

For larger FFTs (2048 points = 16 kB), straightforward expansion of the present algorithm would cause increased cache misses. Thus a larger FFT should be broken into multiple FFTs of 1024 points so that all 10 stages of each can achieve high cache hits. The algorithm becomes (assuming 2048 points, Decimation-In-Time):

1) Bit-reverse the entire input array
2) Do a 10-stage FFT on the second set of 1024 points. Cache hits should be high on those, since they were most recently accessed by the bit-reversal.
3) Do a 10-stage FFT on the first 1024 points. Prefetch before the first stage to ensure cache hits.
4) Combine the 2 separate 1024-point results with a final stage of butterflies, where A is offset from B by 8 Kbytes.

5.2 Pfld

Twiddle factors (W) are fetched with **pfld** (Pipelined Floating-Point Load), to avoid caching them. Only in the first stage are all the W() elements used; successive stages use fewer and fewer elements, which are separated by larger and larger strides. Thus placing W() in cache would be inefficient. The streaming of W() from main memory actually yields better performance than caching W(), for 512 and 1024 points. With the i860 CPU's 8-byte external data bus, a complex W() value can be transferred in a single bus cycle. Some FFT routines calculate W() on the fly, rather than fetching pre-calculated values; however, performance decreases due to the added run-time calculations.

5.3 Fst.q

Quad-word (16-byte) stores allow 4 floating-point register values to update the cache in one cycle. Likewise, **fld.q** (Quad Floating Point Load) transfers 4 values to the registers in a cycle. However, in some FFT stages, double-word fetches (**fld.d**) are used instead of **fld.q**; that allows the "background" fetch of a set of operands concurrent with arithmetic on the other set. For the same reason, the inner loop does 2 butterflies, rather than one.

5.4 Bit Reversal Code

The code for bit-reversal fetches the indices of 2 elements to be swapped from a pre-allocated array of indices, and swaps the data elements. Again, **pfld.d** keeps the indices out of cache, for the 1024 point case. That assembly version of bit-reversal is approximately 7 times faster than the standard Fortran routine. The array of indices was generated by printing out the values generated during operation of the standard Fortran version; similarly, the twiddle W() values can be pre-allocated and generated using a high-level- language program.

6.0 PIPELINE SCHEDULING

The adder pipeline is 3 stages, as is the multiplier; for the calculation of

$$BnewR = (AR - BR) * WR - (AI - BI) * WI$$

the adder result is fed back into the multiplier, and the product again feeds into the adder. The adder and multiplier pipes each advance one stage for each floating-point instruction issued.

The butterfly decomposes into 6 real add/subtracts and 4 real multiplies. Thus the best possible performance would be 6 clocks per butterfly, with the multiplies totally overlapping the adds. The overlap is accomplished with the Dual-Operation instructions:

r2pt	(KR*src2, Treg+Mout, load KR ← src1)
rat1s2	(KR*Aout, src1-src2, load T ← Mout)
i2st	(KI*src2, Treg-Mout, load KI ← src1)
rat1p2	(KR*Aout, src1+src2, load T ← Mout)
ia1p2	(KI*Aout, src1+src2, load KI ← src1)

KR, KI, and T are operand registers feeding the multiplier and adder, separate from the floating-point register file. They permit the 4 inputs for multiply and add, even thought the instruction format holds only 2 registers. "Aout" and "Mout" are adder and multiplier outputs.

The data path arrangements of some of these ops are illustrated in Figures 3 and 4. Fetching and storing of butterfly operands is overlapped with the calculations, using Dual Instruction Mode — the integer core op (such as a load or branch) and FP op are fetched simultaneously from the instruction cache and executed simultaneously.

Scheduling of instructions was done with a pipeline diagram, as illustrated in the comments of the code listing of difstep.ss in the Appendix. (The comments show the machine state after the instruction is processed.) Begin by placing the desired results in the rightmost column, then tracing progress backwards through the adder. When adder inputs are products (of the multiplier), one product is kept in the Treg for a cycle while the other propogates through the multiplier final stage. Those products can be traced back on the multiplier pipeline, to determine at what instruction the multiplier inputs must be provided.

For example, place the BnewR label in the "Write" stage of the pipe (the output of the Adder). Now

$$BnewR = WR * DR - WI * DI$$

Three instructions earlier, the adder inputs for BnewR must be fed to adder; those inputs are products, one of which comes directly from the multiplier output, and the other from the Treg. The multiplier output and Treg value must then be traced back through multiplier stages, requiring the following instructions:

i2st.ss	WIo,DR,BnewIo	as the 10th op of 12, to start (T − Mout)
rat1s2.ss	AIo,BIo,AnewI	as the 9th instruction, to update the Treg
ia1p2.ss	AI,BI,DI	as the 6th op, to multiply DI * WI
rat1p2.ss	AR,BR,DR	as the 5th op, to multiply DR * WR
rat1s2.ss	AI,BI,AnewIo	as the 3rd, to start DI into the adder
pfsub.ss	AR,BR,AnewRo	as the 2nd, to start DR into the adder

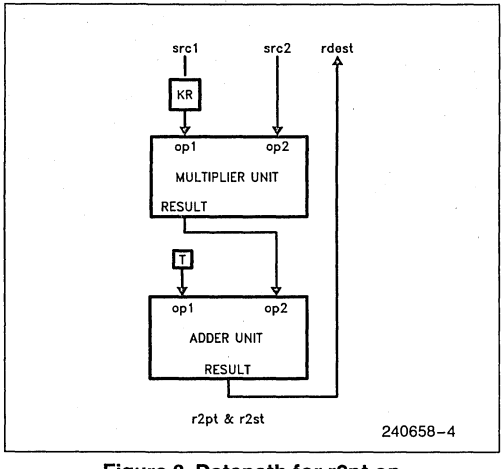


Figure 3. Datapath for r2pt op

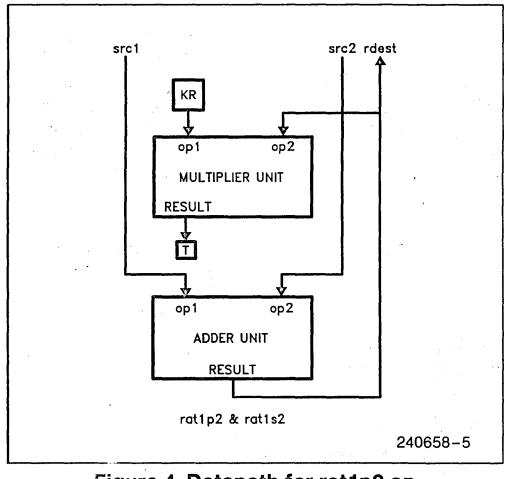


Figure 4. Datapath for rat1p2 op

Some trial-and-error ordering of the desired outputs is needed to devise a sequence which keeps the adder pipeline full. An op is chosen for each slot for its ability to load the KR or KI register, or to initiate an adder operation simultaneous with the multiplies required to calculate BnewR and BnewI.

Handy hints to assist dual-operation scheduling include:

1) *Feedback* the adder result to the multiplier, or visa versa, whenever possible. For example, the ratlp2 op feeds adder-out to multiplier. Thus both src1 and src2 fields of the instruction are available to feed the adder-in, and a simultaneous useful add and multiply are initiated.
2) *Freeze* one of the pipes, by using a pfadd or pfmul, when appropriate. In the butterfly, where 6 adds are done for every 4 multiplies, freezing of the multiplier does not degrade performance. The freeze allows multiplier results to be held until needed in the adder.
3) The *Treg* can hold a multiplier result for several cycles until needed in the adder.
4) *Unroll a* loop to do 2 iterations per loop. That provides time to fetch inputs for iteration 2 while calculating iteration 1, and store results of iteration 1 (and fetch more inputs) while calculting iteration 2.

7.0 PERFORMANCE MEASUREMENTS

The code was run on an evaluation card with DRAM memory only, no external cache, 33.33 MHz clock, and 5 wait-states or more for some accesses. Next-near accesses (address falls into the same DRAM page as the previous access) are zero wait-state, but far accesses take 5 or more wait-states. The code was run under a virtual-memory multitasking executive. Shown below are measured results:

System: 33.3 MHz 80860 with a single bank of static-column DRAM

Algorithm: Radix-2 FFT, in-place. Data is IEEE 754 single-precision floating point. Implemented in assembly-language and Fortran code.

Type of FFT	Time	Time (including bit-reversal)
1024-point-complex, DIF	1.17 ms	1.33 ms
1024-point-real		0.67 ms
512-point-complex, DIF	0.48 ms	0.56 ms
512-point-real		0.33 ms
256-point-complex, DIF	0.22 ms	0.26 ms
1024-point-complex, DIT		1.37 ms
512-point-complex, DIT		0.59 ms

7.1 Cache Fill and Writeback Time

Measured times do not include cache-fill and write-back. That is, the timings measured 200,000 executions of the FFT using the same input array. (Performance figures offered by other manufacturers for DSP chips likewise assume that the data is already in on-chip RAM. Of course, the i860 CPU will do that fetching automatically into its data cache.) The additional time for cache fill and writeback were measured as:

1024-point-complex 0.25 ms (8 Kbytes fetched, 8 Kbytes writeback)

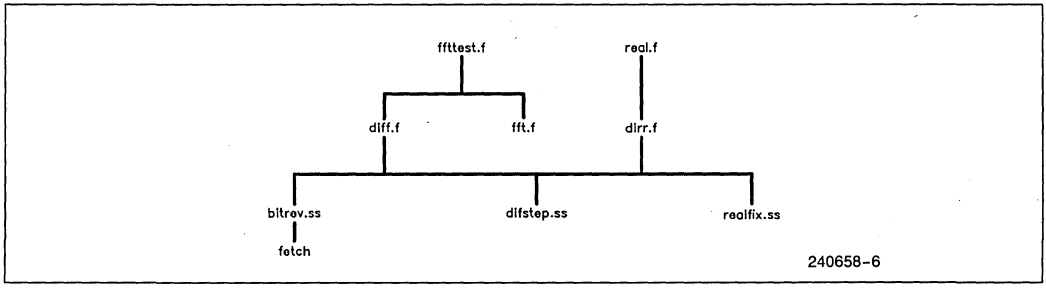
512-point-complex 0.12 ms (4 Kbytes)

To quantify the calculations in MFlops (Millions of FLoating-point OPerations per Second), consider that the 1024-point complex FFT is implemented with about 16,400 multiplies and 28,700 adds/subtracts. Thus the 1.17 ms translates to a sustained 38.5 MFlops rate. For 512 points, the required 20,000 Flops means 41.6 MFlops.

The overall FFT is about 10 times faster than the equivalent Fortran. Inner loop performance was measured at 13 cycles for the 24 instructions, which is 6.5 cycles per butterfly.

8.0 CODE HIERARCHY

Pictured below are the programs developed for the i860 CPU FFT:



The Fortran program **ffttest.f** is the highest-level program of those listed on the following pages. It calls two FFT subroutines, **diff.f** and **fft.f,** then compares their outputs. **Fft.f** is a Fortran decimation-in-time algorithm, while **diff.f** is the high-speed DIF routine. **Diff.f** is callable by C or Fortran applications. It in turn calls **difstep**, which is implemented in assembly code (**difstep.ss**). **Difstep** is called once per stage of the FFT. A Fortran version (**difstepf.f**) is shown, for comparison. Other assembly routines are the bit-reversal-data-movement (**bitrev.ss**) and prefetch ("fetch" inside **bitrev.ss**).

Difstep.ss contains approximately 225 assembly instructions, and **bitrev.ss** contains about 24. The Fortran **diff.f** compiles to about 80 instructions.

A Decimation-in-Time version of **diff.f** and **difstep.ss** can be found in **ditt.f** and **ditstep.ss**. The DIT version performs 5-10% slower than the Decimation-in-Frequency because the DIT loop takes 7 cycles per butterfly, while DIF takes 6.

A real-input algorithm is **dirr.f**, which can be called and tested using program **real.f**. **Dirr.f** calls **difstep** to do a complex DIF FFT on N real data points, but treats them as N/2 complex points. Then **realfix.ss** is called by dirr.f to fix the DIF output, compensating for the treatment of the N real points as N/2 complex. The derivation of the real-fix can be found in reference 3, *Numerical Recipes in C.*

The mixture of Fortran, C, and assembly code is accomplished by passing function inputs and outputs in registers. Only pointers and integer values were used in the above code, but floating point parameters can also be exchanged. A calling program feeds arguments to a function in r16, r17, and higher-numbered integer registers. The callee is permitted to destroy the contents of those registers, but r1:r15 must be preserved. For more details on parameter-passing conventions see the *i860 64-bit Microprocessor Programmer's Reference Manual,* Chapter 8.

9.0 CONCLUSION

The i860 CPU computes very Fast Fourier Transforms, quicker than most high-end dedicated DSP chips. Contributing to the FFT performance are the 8-kByte on-chip data cache and 4-kByte instruction cache. Also the 8-byte external data bus, pfld instruction, and 16-byte data cache width provide sufficient bandwidth to keep the arithmetic units busy. Dual-Operation instructions and Dual-Instruction-Mode allow parallel data movement and calculations. The 33.3 MHz clock rate allows both an add and a multiply every 30 ns, giving a time of 1.17 ms for a 1024-point complex FFT. A 40 MHz i860 Microprocessor will yield a time of less than 1 mSec.

ACKNOWLEDGEMENTS

The author wishes to thank Tricord Systems, Inc. for providing the key inner loop kernel design of the FFT.

BIBLIOGRAPHY

1. Gold, Bernard and Rabiner, Lawrence, *Theory and Application of Digital Signal Processing*, 1975, Prentice-Hall Inc., Englewood Cliffs, NJ. Pages 356-381,573ff

   [This text explains DFT and FFT basics well, with ample pictures]

2. Horden, Ira, "An FFT Algorithm For MCS(c)-96 Products Including Supporting Routines and Examples", Intel Application Note AP-275, order number 270189. (That Application Note can also be found in the Intel Embedded Controller Handbook, Volume II, order number 210918)

   [The note, dated 9/87, reviews FFT theory, real vs. complex, A/D issues, and waveforms]

3. Press, William, Flannery, Brian, et. al., *Numerical Recipes in C*, 1988, Cambridge University Press. Pages 398-424.

   [Numerical Recipes contains the C-code source for "realfix"]

APPENDIX A
PROGRAM LISTINGS

```
C-------------------
C File: diff.f
C  FFT - Decimation in Freq, radix-2, inplace, 1-dimen

C Intel assumes no responsibility for use or misuse of this code.

C  5/19/89: call fetch8() added for 1024-point caching.
C  6/01/89: fetch() CRUCIAL-30% performance loss if removed

C Inputs:
C   A= complex array of input, up to 1024 pts, single-prec float
C   M= log of number of pts
C    = (number of stages of FFT)
C   N = number of points. ie, N= 2**M = number of pts
C   W= complex array of twiddle factors, length N/2.
C   REV= 0 if bitreversed output ok. 1=must re-order output
C
C Outputs:
C  A= complex fft of input A
C
   subroutine diff(a,m,N,W,REV)
   integer m,N, i, j,k, REV,wlimit
   integer offset, stage, groups, wincr,powers2(0:10)
   complex a(n),w(N/2),temp

   data powers2 /1,2,4,8,16,32,64,128,256,512,1024/
C Powers2 to avoid calls to POW, DIV

C Twiddle factor array w(k) has (cos,-sin) of 2pi*k/N
CC Assume the caller provides w(k) constants ALREADY initialized
C------------
C Pre-touch data, lock into cache, for 8kByte fft:
   IF (N .gt. 513) THEN
            call fetch(a,%VAL(n))
   ENDIF
C------------
   wlimit = 8*((N/2) - 1)

C "DO 20" stage-loop
   DO 20 stage = 1,m
            groups = powers2(stage-1)
C groups=number of times the twiddle factors are used, ie, the number of
C  smaller DFTs the stage is split into.

C  offset gets N/2,N/4,N/8,N/16,...
            offset = powers2(m-stage)
            wincr = groups
            call difstep(a,w,groups,offset,wincr,wlimit)
20 CONTINUE

   IF (REV .ne. 0) THEN
cc REV .ne. 0 means must do bit-reversal reordering of output
   call bitrev(a,%VAL(M),n)
   ENDIF

   RETURN
   END
C------------
```

```
//--------------
// difstep.ss: do one stage of fft butterflies
// DIF = Decimation in Frequency, radix-2, inplace, 1-dimension
// (C) Copyright 1989 INTEL Corporation.
//   Inner loop developed with assistance from Tricord Systems, Inc.
//--------------
// 5/18/89: 1 pm - offset_2 added, as next-to-last stage was slow
// 5/19/89: 4 pm - fetch8() routine added, for cache miss avoidance.
// 5/31/89: am - use fst.q (13% perf improvement of inner_loop!)
//  last_bfly added, for performance.
// 6/02/89: am - bptr deleted. Modulo-address W (5% perf improved)
//------------
// Intel is not responsible for use nor for misuse of this program.
//------------
// Do one entire stage (n/2 butterflies). Sample invocation:
//  call difstep(a,w,groups,offset,wincr,wlimit)
//================================================
// Inputs:
//   A= complex array of input, single-prec float
//        (complex stored as 4byte real, 4byte imag contiguously)
//   W= pointer to array of twiddle factors. Assuming W(k) is
//      CMPLX(cos(2pi*k/N)),-sin(2pi*k/N))  for k=0 to (N/2)-1.
//   offset = distance (except for scale-by-8byte sizeof(complex)) between
//      the 2 input values for each butterfly.
//      Offset also is the number of butterflies done per "group".
//   groups = N/(2*offset). The number of sub-DFTs this stage is split into.
//   wincr = distance (except for scale-by-8byte sizeof(complex)) between
//      successive w values for successive butterflies
//   wlimit =max index, in bytes, of W table.
//
// Outputs:
//   A= complex radix-2 butterflied version of input.
//------------------
define(astart, r16)  //input data base address
define(wstart,r17) //twiddle array ptr. Because w-contents depend on N,
// we will assume the caller has initialized w() array.
define(groups,r18) //groups=number of sub-DFTs this stage is split into.
define(offset,r19)  //offset (initially elements, mult by 8 to get bytes)
// between node and its dual (the 2 numbers to butterfly, ie. A and B)
define(wincr,r20) //increment between successive W values. Remains constant
// within a given stage. For Decimation in Freq, wincr addressing is:
   //  +8 for offset=N/2 (W0,W1,W2,W3,...W(n-1))
   //  +16    offset=N/4 (W0,   W2,   W4, ...  ) etc...
define(wlimit,r21) //max index, in bytes, of W table.
define(wind,r22) //current index, in bytes, of W table.
define(offset2,r23)  //offset*2

define(decrem,r24) //bla decrement
define(somecount,r25) // bla counter

define(FEtch, r26)  //pointer to 1st component of butterfly (load)
define(STore,r27)  // " " 1st component of butterfly (store)
```

```
// f4:f7 spare
define(AR, f12)  //element A, real component
define(AI, f13) // " ", imag
define(ARo,f14) // extra A value, for prefetch (o="odd")
define(AIo,f15)
define(BR, f16) //element B, real component
define(BI, f17)
define(BRo,f18) // extra B value, for prefetch
define(BIo,f19)

define(ER, f20) //A+B, real (ER = AR + BR)
define(EI, f21) // "   imag "
define(ERo,f22) //A+B, real, previous loop's value
define(EIo,f23) // "   imag "

define(FR, f24) //W*(A-B), real
define(FI, f25) // "       imag "
define(FRo,f26)
define(FIo,f27)

define(DR, f28) //Difference of A-B, real part
define(DI, f29) // " ", imag "
define(WR, f30) //W (twiddle factor), real part
define(WI, f31) // " " , imag
define(WRo,f10) //W (twiddle factor), real part (EXTRA copy)
define(WIo,f11) // " " , imag

.text
.align .quad
_difstep_::
 ld.l      0(groups),groups  //fix Fortran call-by-ref
 ld.l      0(offset),offset  //
 shl       3,offset,offset  // change from elements to bytes
 shl       1,offset,offset2

 fst.q   f8 ,-16(sp)++ //save "local" regs
 fst.q   f12,-16(sp)++ // " "

 adds      -1,groups,groups // pre-decrement for bnc usage, or bla usage
 adds      -16,r0,decrem //bla decrement

//  We code the last 2 stages as special cases:
//--------
 xor     8,offset,r0 //offset=1, special case, no complex mult, funny addressing
 bcoffset_1// (ASSUMING offset=1 means wincr=0, and no twiddle used)
 xor     16,offset,r0 //offset=2, special case, no complex mult, funny addressing
 bcoffset_2//(ASSUMING offset=2 means wincr=N/4)
//--------
 ld.l      0(wincr),wincr
 ld.l      0(wlimit),wlimit
```

```
 pfadd.ss f0,f0,f0
 pfadd.ss f0,f0,f0
 pfadd.ss f0,f0,f0  // init A1,A2,A3=0
 pfmul.ss f0,f0,f0
 pfmul.ss f0,f0,f0
 pfmul.ss f0,f0,f0
//--------
 // init pointers:
 shl       3,wincr,wincr     //scale for bytes.
 shl       1,wincr,wind      //init wind =2*wincr

 pfld.d 0 ( wstart),f0
 pfld.d wincr ( wstart),f0
 adds      -8,astart,FEtch
 pfld.d wind (wstart),f0
 adds      wincr,wind,wind //wind now 3*wincr
// here fetch first set of A,B,W before bla-loop
 pfld.d wind (wstart),WR
 adds      wincr,wind,wind
 and       wlimit,wind,wind //modulo-wlimit the w index
// We do modulo-addressing on W(), to keep the pfld pipeline full. We
//  never do a W-fetch beyond the end of the table.
// And the modulo-check needs to be done only every 4th pfld, as always
//  we use a multiple of 4 W() factors.


 fld.d 8 (FEtch)++,AR
 fld.d offset (FEtch),BR
d.r2apl.ss f0,f0,f0 //clear Treg.
 adds -32,offset,somecount // bla counter (predecrement by 4 elements)
// -----------
// Definitions for pipe diagram:
//  (the complex multiply product, F, broken into 4 real mult and 2 adds):
//   WR = cos(), WI=-sin().
//   DR = AR - BR; (diffence of Real components of A,B)
//   DI = AI - BI; (diffence of Imag components)
//   ER = AR + BR; EI = AI + BI;
//   FR = K - L; where K= WR*DR, L=WI*DI
//   FI = N + M; where M= WI*DR, N=WR*DI

// For 1st time thru inner_loop, don't have correct values to store.
// Must do 1 loop before the loop, sans the stores.

first_bfly:: //fill pipe
                    // KR...KI...M1....M2....M3   T    A1....A2....A3....Write
d.r2pt.ss WR,f0,f0   // WR0  -
 pfld.d wind (wstart),WRo
d.pfsub.ss AR,BR,f0  //          -     -     -     -   DR0   -     -
 fld.d 8 (FEtch)++,ARo
d.ratls2.ss AI,BI,f0 //          -     -     -     -   DI0   DR0   -     -
 fld.d offset (FEtch),BRo
d.i2st.ss  WI,f0,f0  //      WI0  -     -     -     -     -    DI0   DR0   -
 adds       wincr,wind,wind
```

```
d.ratlp2.ss AR,BR,DR   //          K0   -    -    -    ER0  -    DI0  DR0
 nop
d.ialp2.ss AI,BI,DI    //          L0   K0   -         EI0  ER0  -    DI0
 pfld.d  wind (wstart),WR
d.r2pt.ss WRo,DI,f0    // WR1  -    N0   L0   K0   -    -    EI0  ER0  -
 fld.d  8 (FEtch)++,AR
d.pfsub.ss ARo,BRo,ER  //          N0   L0   K0   -    DR1  -    EI0  ER0
 fld.d  offset (FEtch),BR
d.ratls2.ss AIo,BIo,EI //          -    N0   L0   K0   DI1  DR1  -    EI0
 adds       wincr,wind,wind
d.i2st.ss  WIo,DR,f0   //      WI1  M0   -    N0   K0   K-L  DI1  DR1  -
 and        wlimit ,wind,wind

quickstart::
d.ratlp2.ss ARo,BRo,DR //          K1   M0   -    N0   ER1  FR0  DI1  DR1
 bla        decrem,somecount,inner_loop //init LCC
d.ialp2.ss AIo,BIo,DI  //          L1   K1   M0   N0   EI1  ER1  FR0  DI1
 adds       -16,astart,STore  // ptrs init 16 low, for fst.q instructions
//-------------------
// Each butterfly = 1 complx multiply, 1 complx add, 1 complx subtract
// =        4 multiply,
//          3 add
//          3 subtract
//          3 8-byte fetches (A, B, W)
//          2 8-byte stores (A, B)
//
// 6 cycles per butterfly
//
// inner_loop: iterates "offset/2" times (eg, N/4 for stage 1, N/8 for stage2),
//   for each group. It does 2 butterflies per iteration

inner_loop::
                       // KR...KI...M1...M2..M3    T    A1..A2...A3..Write
                       // |    |    |    |    |    |    |    |    |    |
d.r2pt.ss WR,DI,FR     // WR2  -    N1   L1   K1   N0   N+M  EI1  ER1  FR0
 pfld.d  wind (wstart),WRo
d.pfsub.ss AR,BR,ERo   //          N1   L1   K1   N0   DR2  FI0  EI1  ER1
 fld.d  8 (FEtch)++,ARo
d.ratls2.ss AI,BI,EIo  //          -    N1   L1   K1   DI2  DR2  FI0  EI1
 fld.d offset (FEtch),BRo
d.i2st.ss WI,DR,FI     //      WI2  M1   -    N1   K1   K-L  DI2  DR2  FI0
 fst.q ER,16(STore)++ //update ER/EI/ERo/EIo
d.ratlp2.ss AR,BR,DR   //          K2   M1   -    N1   ER2  FR1  DI2  DR2
 adds       wincr,wind,wind
d.ialp2.ss AI,BI,DI    //          L2   K2   M1   N1   EI2  ER2  FR1  DI2
//no need for modulo-check ("and") here, as odd num of W's have been fetched.
 pfld.d wind (wstart),WR
//.............................................................
```

```
                    // KR...KI...M1....M2....M3   T    A1....A2....A3....Write
d.r2pt.ss WRo,DI,FRo // WR3  -    N2    L2    K2   N1   N+M   EI2   ER2   FR1
 adds       wincr,wind,wind
d.pfsub.ss ARo,BRo,ER//          N2    L2    K2   N1   DR3   FI1   EI2   ER2
 fld.d 8 (FEtch)++,AR
d.ratls2.ss AIo,BIo,EI//         -     N2    L2   K2   DI3   DR3   FI1   EI2
 fld.d  offset (FEtch),BR
d.i2st.ss  WIo,DR,FIo//      WI3 M2    -     N2   K2   K-L   DI3   DR3   FI1
 fst.q  FR, offset (STore)
 //update FR/FI/FRo/FIo
d.ratlp2.ss ARo,BRo,DR//         K3    M2    -    N2   ER3   FR2   DI3   DR3
 bla decrem,somecount, inner_loop
d.ialp2.ss AIo,BIo,DI//          L3    K3    M2   N2   EI3   ER3   FR2   DI3
 and        wlimit,wind,wind //modulo.

end_inner_loop:: //KEEP Pipelines full
// RE-init pointers for fetches
d.fiadd.ss f0,f0,f0
 adds       offset2,astart,astart //bump to next group
      //redo A,B fetches, with proper ptr.
d.fiadd.ss f0,f0,f0
 fld.d 0(astart)  ,AR //get first AR/AI in next group
d.fiadd.ss f0,f0,f0
 fld.d  offset (astart),BR
d.fiadd.ss f0,f0,f0
 adds       0,astart,FEtch

last_bfly:: //do final 2 butterflies, start next group
                    // KR...KI...M1....M2....M3   T    A1....A2....A3....Write
d.r2pt.ss WR,DI,FR   // WR4  -    N3    L3    K3   N2   N+M   EI3   ER3   FR2
 pfld.d  wind (wstart),WRo
d.pfsub.ss AR,BR,ERo //          N3    L3    K3   N2   DR4   FI2   EI3   ER3
 fld.d  8 (FEtch)++,ARo
d.ratls2.ss AI,BI,EIo//          -     N3    L3   K3   DI4   DR4   FI2   EI3
 fld.d  offset (FEtch),BRo
d.i2st.ss  WI,DR,FI  //      WI4 M3    -     N3   K3   K-L   DI4   DR4   FI2
 fst.q ER,16(STore)++
d.ratlp2.ss AR,BR,DR //          K4    M3    -    N3   ER4   FR3   DI4   DR4
 adds       wincr,wind,wind
d.ialp2.ss AI,BI,DI  //          L4    K4    M3   N3   EI4   ER4   FR3   DI4
 pfld.d  wind (wstart),WR
//.............................................................
                    // KR...KI...M1....M2....M3   T    A1....A2....A3....Write
d.r2pt.ss WRo,DI,FRo // WR5  -    N4    L4    K4   N3   N+M   EI4   ER4   FR3
 fld.d  8 (FEtch)++,AR
d.pfsub.ss ARo,BRo,ER//          N4    L4    K4   N3   DR5   FI3   EI4   ER4
 adds -32,offset,somecount // reset bla counter
d.ratls2.ss AIo,BIo,EI//         -     N4    L4   K4   DI5   DR5   FI3   EI4
 adds       wincr,wind,wind
d.i2st.ss  WIo,DR,FIo//      WI5 M4    -     N4   K4   K-L   DI5   DR5   FI3
 adds -1,groups,groups
d.fnop
 fld.d  offset (FEtch),BR
d.fnop
 bnc.t quickstart //branch on value of groups
d.fnop
 fst.q  FR, offset (STore)
```

```
end_last_bfly::
d.fnop
 br endit
fiadd.ss f0,f0,f0
 fst.q  FR, offset (STore) //repeated for bnc.t untaken case
.align      .quad
//=============================================================
offset_1::
// want FEtch=0,2,4,6,8,... elements.  ASSUMING wincr=0,
//  and that w=(1,0), so that no complex mult needed, and NO W will be fetched.
//  E=A+B, F=A-B.  (Per double-butterfly loop:  8 pfadd,4 dword fld, 4 fst,
//  1 bla)(fld.q required, to reduce # flds to avoid pipe stalls)
//  Performance = 4 cyc/bfly best case.

//Redefine regs for fld.q,fst.q usage, when A and B adjacent:
define(AR3,f12)  //element A, real component
define(AI3,f13) // " ", imag
define(BR3,f14)  //element B, real component
define(BI3,f15)
define(AR4,f16) // extra A value, for prefetch
define(AI4,f17)
define(BR4,f18) // extra A value, for prefetch
define(BI4,f19)

define(ER3, f20) //A+B, real (ER = AR + BR)
define(EI3, f21) // "   imag "
define(FR3, f22) //(A-B), real
define(FI3, f23) // "    imag "

define(ER4,f24) //A+B, real, extra copy
define(EI4,f25) // "   imag

define(FR4,f26)
define(FI4,f27)
//=======================================
 adds       -16,astart,FEtch
 fld.q  16 (FEtch)++,AR4
 adds       -1,groups,somecount // bla counter (predecremented already by 1)
   //using groups=blacount on the offset_1 loop, intentionally.
 adds       -16,FEtch,STore
//startup the loop:
// --------------------// A1......A2......A3......Write:
d.pfadd.ss AR4,BR4,f0   // ARn+BRn -       -       -
 fld.q  16 (FEtch)++,AR3
d.pfadd.ss AI4,BI4,f0   // AIn+BIn ERn     -       -
 adds       -2,r0,decrem //2 bflies per loop
d.pfsub.ss AR4,BR4,f0   // ARn-BRn EIn     ERn     -
 bla decrem,somecount, offset1_loop  //init LCC
d.pfsub.ss AI4,BI4,ER4  // AIn-BIn FRn     EIn     ERnext
 nop
// --------------------// A1......A2......A3......Write:
offset1_loop::
```

```
d.pfadd.ss AR3,BR3,EI4  // AR+BR   FI-     FR-     EI-
 nop
d.pfadd.ss AI3,BI3,FR4  // AI+BI   ER      FI-     FR-
 fld.q 16 (FEtch)++,AR4
d.pfsub.ss AR3,BR3,FI4  // AR-BR   EI      ER      FI-
 fst.q      ER4,16(STore)++
d.pfsub.ss AI3,BI3,ER3  // AI-BI   FR      EI      ER
 nop
d.pfadd.ss AR4,BR4,EI3  // AR2+BR2 FI      FR      EI
 fld.q  16 (FEtch)++,AR3
d.pfadd.ss AI4,BI4,FR3  // AI2+BI2 ER2     FI      FR
 nop
d.pfsub.ss AR4,BR4,FI3  // AR2-BR2 EI2     ER2     FI
 bla decrem,somecount, offset1_loop
d.pfsub.ss AI4,BI4,ER4  // AI2-BI2 FR2     EI2     ERnext
 fst.q  ER3,16(STore)++
//-------------------------
end_offset1_loop::
d.fiadd.ss f0,f0,f0
 br endit
fiadd.ss f0,f0,f0
 nop
//-------------------------
.align .quad
offset_2::
// want FEtch=0,1;4,5;8,9;12,13;... elements.
// ASSUMING wincr=N/4  (W_addr=0,N/4,0,N/4,0,...). Trivial W() factors.
// USE bla loop, incrementing FEtch by 16 (2*offset).
//  Even-indexed elements identical to offset_1,W=W0, no complex mult.
//    So FReven=(AR-BR), FIeven=(AI-BI).
//  Odd components have W=(0,-1).  So FRodd=(AI-BI), FIodd=(BR-AR).
//  Each fld.q fetches AReven,AIeven,ARodd,AIodd.

//Assume ER,EI,ERo,EIo are 4 contiguous regs.
//Assume FR,FI,FRo,FIo are 4 contiguous regs.

 adds       -16,astart,FEtch
 fld.q  16 (FEtch)++,AR
 fld.q  16 (FEtch)++,BR
 adds       0,groups,somecount //bla counter
//startup the loop:
// ---------------------// A1......A2......A3......Write:
  pfadd.ss AR ,BR ,f0   // AR+BRe                   -
  pfadd.ss AI ,BI ,f0   // AI+BIe  ER               -
d.pfadd.ss ARo,BRo,f0   // ARo+BRo EI      ER       -
 nop
d.pfadd.ss AIo,BIo,ER   // AIo+BIo ERo     EI       ER
 nop
d.pfsub.ss AR ,BR ,EI   // AR-BRe  EIo     ERo      EI
 adds       -1,r0,decrem //2 bflies per loop,but groups is half desired value.
d.pfsub.ss AI ,BI ,ERo  // AI-BIe  FR      EIo      ERo
 adds       -16,astart,STore
d.pfsub.ss AIo,BIo,EIo  // AIo-BIo FI      FR       EIo
 bla decrem,somecount, offset2_loop //init LCC
d.pfsub.ss BRo,ARo,FR   // BRo-ARo FRo     FI       FR
 nop
```

```
offset2_loop::
d.fnop
 fld.q  16 (FEtch)++,AR //fetch AR,AI,ARo,AIo
d.fnop
 fld.q  16 (FEtch)++,BR //fetch BR,BI,BRo,BIo
// ---------------------// Al......A2......A3......Write:
d.pfadd.ss AR ,BR ,FI   // AR+BRe  FIo     FRo     FI
 nop
d.pfadd.ss AI ,BI ,FRo  // AI+BIe  ER      FIo     FRo
 nop
d.pfadd.ss ARo,BRo,FIo  // ARo+BRo EI      ER      FIo
 fst.q      ER ,16(STore)++
 //update ER ,EI ,ERo,EIo
d.pfadd.ss AIo,BIo,ER   // AIo+BIo ERo     EI      ER
 nop
d.pfsub.ss AR ,BR ,EI   // AR-BRe  EIo     ERo     EI
 nop
d.pfsub.ss AI ,BI ,ERo  // AI-BIe  FR      EIo     ERo
 fst.q  FR ,16(STore)++
d.pfsub.ss AIo,BIo,EIo  // AIo-BIo FI      FR      EIo
 bla decrem,somecount,offset2_loop
d.pfsub.ss BRo,ARo,FR   // BRo-ARo FRo     FI      FR
 nop

endit::
// restore regs
fiadd.ss f0,f0,f0 //exit DIM
 fld.q   0(sp),f12
fiadd.ss f0,f0,f0 //last DIM pair
 fld.q   16(sp),f8
adds    32,sp,sp
 bri rl
  nop
//---------------------------------
```

```
c-------------------------------------------------------------
c difstepf.f: do one stage of fft (DIF) butterflies
c (C) Copyright 1989 INTEL Corporation. ALL RIGHTS RESERVED.
c-------------------------------------------------------------
c Decimation in Freq, radix-2, inplace, 1-dimen
c 6/20/89

c Do one entire stage (n/2 butterflies). Sample invocation:
c  call difstep(a,w,groups,offset,wincr)

c Inputs:
c   A= complex array of input, single-prec float
c        (complex stored as 4byte real, 4byte imag contiguously)
c   W= pointer to array of twiddle factors. Assuming W(k) is
c        CMPLX(cos(2pi*k/N)),-sin(2pi*k/N)) for k=0 to (N/2)-1.
c   offset = distance (in "elements") between
c        the 2 input values for each butterfly
c   groups = number of sub-DFTs this stage is split into.
c        (groups*offset*2 = N)
c   wincr = distance between successive w values for successive butterflies
c
c Outputs:
c A= complex butterflied version of input.

   SUBROUTINE difstep(a,w,groups,offset,wincr)
   integer groups,offset,wincr
   integer i,j,indexl,iplus
   complex a(groups*offset*2),w(groups*offset),wtemp,temp
c-------------------------------------------------------------
c We implement a...
c Special case for offset=1(last stage): no complex multiplies, simple add
c  (Performance enhancement)
   IF (offset .eq. 1) THEN
CVD$ NODEPCHK
            DO 8 i = 1,(2*groups),2
              iplus = i + 1
              temp = a(iplus)
              a(iplus) = a(i) - temp
8             a(i) = a(i) + temp
   ELSE
C------------
C Special case for offset=2 (next-to-last stage): no complex multiplies,
cc    simple add.  (Performance enhancement)
cc For half the butterflies, W=(1,0). For the other half, W=(0,-1)
   IF (offset .eq. 2) THEN
CVD$       NODEPCHK
           DO 90 i = 1,(4*groups),4
             iplus = i + 2
             temp = a(iplus)
             a(iplus) = a(i) - temp
90           a(i) = a(i) + temp
C  2nd call to i-loop: w=cmplx(0,-1.)
CVD$       NODEPCHK
CVD$       NOVECTOR
   DO 92 i = 2,(4*groups),4
     iplus = i + 2
     temp = a(i) - a(iplus)
     a(i) = a(i) + a(iplus)
92   a(iplus) = CMPLX(AIMAG(temp),-REAL(temp))
```

```
      ELSE
C-------------
c "DO 20" indexl-loop is "outer loop"
CVD$         VECTOR
CVD$         NODEPCHK
      DO 20 indexl = 1,(2*offset*groups),(2*offset)
              j = 1
CVD$          NODEPCHK
CVD$          ALTCODE
              DO 10 i = indexl,(indexl+offset-1)
                iplus = i + offset
                temp = a(i) - a(iplus)
                a(i) = a(i) + a(iplus)
                a(iplus) = w(j) * temp
10              j = j + wincr
20    CONTINUE
      ENDIF
      ENDIF
      RETURN
      END
ccccccccccccccccccccccccccccccccc
      subroutine fetch(a,n)
      integer n
      complex a(n),temp
cc Kludge do-nothing prefetch.
      temp = a(1)
      RETURN
      END
ccccccccccccccccccccccccccccccccc
      subroutine bitrev(a,dummy,n)
C Bit-Reverse
C Inputs:
C     A= complex array of input, single-prec float
C     dummy = %val(m).  Probably unusable from Fortran.
C     N = number of input points (and output points)

C Ouput:
C     A = original A data, but in bit-reversed order from A

      integer n,i,j,k,ndiv2
      complex a(n),temp
C-------------
C "DO 7" loop to in-place-bit-reverse-shuffle output
      j=1
      ndiv2 = n / 2
      DO 7  i= 1, n-1
        IF (i .lt. j) THEN
              temp = a(j)
              a(j) = a(i)
              a(i) = temp
        ENDIF
        k = ndiv2
C     "While (j .gt. k)" /*decrease j by 2**something */
6       IF (j .gt. k) THEN
              j = j-k
              k = k / 2
              GOTO 6
        ENDIF
C     Add next lower power of 2 to j
7     j = j+k
      RETURN
      END
C-------------
```

```
//--------------
// bitrev.ss
// (C) Copyright 1989 INTEL Corporation. ALL RIGHTS RESERVED.
//
//   BIT-reversal of 8byte array elements.
//   IN PLACE.
// (Allows arrays of 8,16,32,64,128,256,512, or 1024 elements)
//--------------
//  INTEL is not responsible for use nor misuse of this code.
//--------------
// 8/13/89
//================================================
//  Invocation: (from Fortran)
// call bitrev(a,%VAL(m))

// Inputs:
//   a = r16 = pointer to array of 8byte elements
//   m = r17 (call by value)= base-2 log of total number of elements
//         (2**m = N)
// Outputs:
//         a= Bit-reversed ordered version of A
//
// Expected best-can-do performance, and measured performance=
//  approx 4*N clocks (0.06 mSec for 512 points)
//------------------
define(astart, r16)  //initial input data base address
define(m, r17)
define(logN,r17)
define(dest1,r19)
define(dest2,r20)
define(dest3,r21)
define(dest4,r22)
define(iptr, r23)  //index-array pointer

define(decrem,r24) //bla decrement
define(count,r25) // bla counter

.text
.align .quad
//=======================================
_bitrev_::
_bitr_::
//fetch base address for index table (rbasetab)
// base-addr-table elements = (baseaddr, number_of_swaps-2)
// base-addr-table indexed by logN.
 shl       3,logN,r30  //scale to 8-byte-entry length
 mov       rbasetab,r29
 ld.l      r29(r30), iptr
 addu      4,r29,r29
 ld.l      r29(r30), count //number of swaps required for this value N

 pfld.d 0(iptr),f0 //initiate fetch of first 2 bit-rev indices
 pfld.d 8(iptr)++,f0
 adds      -2,r0,decrem//2 swaps per loop
 pfld.d 8(iptr)++,f0

 bla       decrem,count, revloop  //init LCC
  pfld.d 8(iptr)++,f16       //get 2 indices, but don't cache the indices
```

```
revloop:: //2 swaps per loop
 //7.5 cycles consumed for each swap, best case.
 pfld.d 8(iptr)++,f18        //2 more indices
 fxfr f16,dest1  //transfer to integer index regs
 fxfr f17,dest2
 fld.d  dest1 (astart),f24 //fetch 2 elements to swap
 fld.d  dest2 (astart),f26
 fxfr f18,dest3
 fst.d     f24, dest2 (astart)
 fst.d     f26, dest1 (astart)
 fxfr f19,dest4
 fld.d  dest3 (astart),f28
 fld.d  dest4 (astart),f30
 pfld.d 8(iptr)++,f16        //2 more indices
 fst.d     f28, dest4 (astart)
 bla decrem,count, revloop  //
  fst.d    f30, dest3 (astart)

 bri       r1
  nop
//---------------
// _fetch8_: Touch all 32-byte lines in the 8k data bytes, to get them
//   into dcache. (ASSUMING .lte. 8Kbytes and .gte. 4Kbytes)
//
//  Invocation= fetch(astart,num8)
//  Inputs=
//    astart=r16=pointer to data which is to be touched.
//    num8=r17 (passed by VALUE, %VAL(), not by reference)
//------------
// Using RC and RB to improve dcache hit rates, for FFTs bigger than
//  1024 complex (8kB).
//  RC=10 causes replacement only of block denoted by RB lsbit. RC=11 disables
//  replacement.
//--------
define(num8,r17)
define(FEtch, r26)

_fetch8_::
_fetch_::
 ld.c dirbase,r30
 or 0x800,r30,r30 // Replace Dcache slot 0 only (RC=10,RB=00)
 st.c r30,dirbase
// Put 4Kbytes into Dcache slot 0. (The rest after 4kB goes to slot1).
 adds      -4,r0,decrem  //4 8-byte-groups per cache line
 adds      508,r0,count  //512, but pre-decremented for bla usage
 bla       decrem,count,floop
  adds     -32,astart,FEtch
floop::
 bla       decrem,count,floop
  fld.d    32(FEtch)++,f30 //dummy load.

 adds      -512,num8,count
 bc        fdone  //if data exhausted, quit
// ld.c dirbase,r30
 or 0x900,r30,r30 // Replace Dcache slot 1 only (RC=10,RB=01)
 st.c r30,dirbase
```

```
 adds       -8,count,count //predecr for bla
 bla        decrem,count,floop2 //set LCC
  fld.d     32(FEtch)++,f30
floop2::
 bla        decrem,count,floop2
  fld.d     32(FEtch)++,f30 //dummy load.
fdone::
// unlock dcache
 andnot 0xF00,r30,r30  //clear RC,RB (dirbase(11:8))
 st.c r30,dirbase
 bri        r1
  nop

.data
//---------------
// rbasetab:: (Table of bit-reversed indices for bitrev subroutine)
//  base-addr-table elements = (baseaddr, number_of_swaps-2)
//  base-addr-table indexed by logN.
.align .quad
rbasetab::
.long [6]0 //don't bother with log(n)=0,1,2
.long rev8,   0
.long rev16,   4
.long rev32,   10
.long rev64,   26
.long rev128,  54
.long rev256,  118
.long rev512,  238
.long rev1024, 494
//=====================

//number of swaps=240 for N=512 (ie, 32 symmetrical patterns
//  exist between 0 and 511.)
// rev512: array of bit-reversed indices, for N=512.
//  Each entry is ("i", and "bit-reversed-i"), shifted left by 3
//    to account for 8-byte-elements.
// NOTE: This listing DOES NOT SHOW all the table elements, to save paper.

.align .quad
rev512::
.long     8, 2048,   16, 1024
.long    24, 3072,   32,  512
.long    40, 2560,   48, 1536
// ETC..., ETC...., ETC...
//===============
.align .quad
rev1024::
.long     8, 4096,   16, 2048
.long    24, 6144,   32, 1024
.long    40, 5120,   48, 3072
.long    56, 7168,   64,  512
// ETC..., ETC...., ETC...
```

```
//Number of swaps = 496
//N (Number of elements) = 1024
//=================
.align .quad
rev16::
   .long 1*8,8*8,2*8,4*8
   .long 3*8,12*8,5*8,10*8
   .long 7*8,14*8,11*8,13*8
rev8::
   .long 1*8,4*8,3*8,6*8
//=================
.align .quad
rev32::
.long     8, 128,16, 64, 24, 192, 40, 160, 48, 96, 56, 224
.long    72, 144, 88, 208, 104, 176, 120, 240, 152, 200, 184, 232
//=================
.align .quad
rev64::
.long     8,  256,   16,  128
.long    24,  384,   32,   64
.long    40,  320,   48,  192
.long    56,  448,   72,  288
// ETC..., ETC...., ETC...
//=================
.align .quad
rev128::
.long     8,  512,   16,  256
.long    24,  768,   32,  128
.long    40,  640,   48,  384
.long    56,  896,   72,  576
// ETC..., ETC...., ETC...
//Number of swaps = 56 (Number of elements) =128
//=================
.align .quad
rev256::
.long     8, 1024,   16,  512
.long    24, 1536,   32,  256
.long    40, 1280,   48,  768
.long    56, 1792,   64,  128
// ETC..., ETC...., ETC...
//Number of swaps = 120, N (Number of elements) = 256
```

```
      PROGRAM FFTTEST
C
C   1-D FFT TEST PROGRAM
C
C Intel assumes no responsibility for use or misuse of this code.
C
C 7/20/89
C-----------------
C
      character*8 REALLY
      PARAMETER (IREV=0)
      PARAMETER (REALLY='complex')
      PARAMETER (TIMEIT=1, CACHETIME=0)
      DATA IT/200000/
c      PARAMETER (N=1024,M=10)
       PARAMETER (N=512,M= 9)
c      PARAMETER (N=256,M= 8)
c      PARAMETER (N=128,M= 7)
c      PARAMETER (N=64,M= 6)
c      PARAMETER (N=32,M= 5)
c     PARAMETER (N=16, M=4)
      PARAMETER (PI=3.1415926536)
      COMPLEX X(N),X1(N),X2(N),X3(N), W(N/2)
c Fortran complex values stored R,I, R,I for arrays.
    Real  ASQR(N),ASQR2(N),XR(N)
    complex wtemp
    real rtemp
C

    PRINT *,' FFT test program (ffttest.f) ....'
    print *, '================================'
    IF (IREV .eq. 0) THEN
     print *,'NOT counting time for bit-reversal.'
     print *,'DO NOT expect matching answers,without bit-rev'
    ELSE
     print *, 'Time for bit-reversal included.'
    ENDIF

     print *, 'Time for cache writeback and fills...'
    IF (CACHETIME .eq. 0) THEN
     print *,' NOT included, if iterating.'
    ELSE
     print *,' ... included.'
    ENDIF

     print *, '================================'
     print *, 'If iterating... Number of Iterations =',IT
     print *, '================================'
     print *, 'Number of Points     = ', N
     print *, '(',REALLY,' data)'
     print *, '================================'
```

```
C------------------
C Init twiddle factor array w(k) with (cos,-sin) of 2pi*k/N
C (Should just declare this as constant, if N is non-variable)
C (OR could have one constant 512-entry W (for N=1024), adjust winer accordingly
C   in diff.f for smaller N)
   rtemp = 2.0*pi/N
   wtemp= CMPLX(cos(rtemp), -sin(rtemp))
   w(1) = (1.0, 0.0)
   DO  200  k  =  2,N/2
200         w(k) = wtemp * w(k-1)
cc print *,' W (twiddle) initialization completed......'
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C   INITIALIZE input data
C
      PIN = (4*PI)/ N
      DO 100 I = 1, N
c  For testing with sinewave input data:
c       Treal = COS( I*PIN)
c       Timag = SIN( I*PIN)

c  For testing with squarewave input:
cc IF (I .lt. N/2) THEN

cc    Treal = 1.0
cc    Timag = 0.5
cc ELSE
cc    Treal = 0.0
cc    Timag = 0.0
cc ENDIF
C  For testing with ramp function input data:
     Treal = I - 1.0
     Timag = Treal + 0.5
     X(I) = CMPLX (Treal, Timag)
     X1(I) = CMPLX (Treal, Timag)
     X2(I) = CMPLX (Treal, Timag)
     X3(I) = CMPLX (Treal, Timag)
100   CONTINUE
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
   IF (TIMEIT .ne. 0) THEN

     CALL fft (X2, M, N)
cc Subroutine fft is Decimation-In-Time, Fortran version.

c         CALL ditt(X, M, N,W,IREV)
          CALL diff(X, M, N,W,IREV)
   ENDIF

ccccccccccccccccccccccccccccccccccccccc
   IF (IREV .ne. 0) THEN
   IF (TIMEIT .eq. 0) THEN
     call vcompare(X,X2,2*N)
     call cmags(X,N,ASQR)
c cmags to take squared magnitude of complex values
     call cmags(X2,N,ASQR2)
```

```
c----------------------c
C print non-zero results:
   J=0
   DO 700 I = 1,N
   IF ((ASQR(I) .GT. 1.0) .OR. (ASQR2(I) .GT. 1.0)) THEN
     WRITE (6,22) (I-1), ASQR(I), ASQR2(I)
22   FORMAT (' I-1=',I4,' ASQR(I)= ',F14.2, ' ASQR2(I)= ',F14.2//)
   J = J+1
   IF (J .GT. 32) GOTO 725
   ENDIF
700 CONTINUE

725     CALL TIME
   ENDIF
   ENDIF

   IF (TIMEIT .ne. 0) THEN
ccccccccccccccccccccccccccccccccccccc
cc- Timing loop follows:

     print *,' Start Ass.FFT'
   IF (CACHETIME .eq. 0) THEN
     DO 500 I = 1, IT,4
C Reuse same array, so cache fill and writeback time NOT included.
          CALL diff(X, M, N,W,IREV)
          CALL diff(X, M, N,W,IREV)
          CALL diff(X, M, N,W,IREV)
500       CALL diff(X, M, N,W,IREV)
   ELSE
     DO 504 I = 1, IT,4
C Alternating between X,X1,X2,X3 should provide cache misses.
          CALL diff(X, M, N,W,IREV)
          CALL diff(X1, M, N,W,IREV)
          CALL diff(X2, M, N,W,IREV)
504       CALL diff(X3, M, N,W,IREV)
   ENDIF
     print *,' END Ass. FFT'
ccccccccccccccccccccccccccccccccccccc
   ENDIF
     STOP
     END
```

```
c----------------------c
      subroutine vcompare(res,exp,n)
c VCOMPARE compares 2 REAL vectors, prints out 1st few miscompares
c
      integer n, errcnt
      real res(n), exp(n)

      write(6,12)
12    format('*** VCOMPARE: vector comparison beginning ***')

   data errcnt/0/
        do 30 i = 1,n
             if(AINT(res(i)) .ne. AINT(exp(i))) then
c  {print out error, exit if alot already}
120                  print *,'*** Error in compares ***'
                     write(6,121) i
121                  format('  Item number = ',I6)
                     write(6,124) res(i), exp(i)
124                  format('  Res_=',F14.2,' Expected_=',F14.2)
                     errcnt = errcnt + 1
                     if (errcnt .gt. 19) then
                             return
                     end if
               end if
30      continue

   if (errcnt .eq. 0) then
190  print *,' *** vector compares SUCCESSFUL ***'
   end if

99 return
   end
c----------------------c
```

2

```
C---------------
C File: ditt.f
C  6/15/89

C Intel assumes no responsibility for use or misuse of this code.

C  FFT - Decimation in TIME, radix-2, inplace, 1-dimen
C Inputs:
C   A= complex array of input, up to 1024 pts, single-prec float
C   M= log of number of pts
C    = (Number of stages of FFT)
C   N = number of points. ie, N= 2**M = number of pts
C   W= complex array of twiddle factors, length=N/2.
C   REV= ignored parameter.
C
C Outputs:
C  A= complex fft of input A. Correct order (bit-reversal done).
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

   subroutine ditt(a,m,N,W,REV)
   integer m,N, i, REV,wlimit
   integer offset, stage, groups, wincr,powers2(0:10)
   complex a(n),w(N/2),temp

   data powers2 /1,2,4,8,16,32,64,128,256,512,1024/
C Powers2 to avoid calls to POW, DIV

C Twiddle factor array w(i) has (cos,-sin) of 2pi*i/N
CC Assume the caller provides w(i) constants ALREADY initialized
C------------
C Pre-touch data, lock into cache, for 8kByte fft:
   IF (N .gt. 513) THEN
           call fetch(a,%VAL(n))
   ENDIF
C------------
   call bitrev(a,%VAL(M),n)
C Bitreversal of input needed for in-place decim in time FFT, to avoid
C  fetching twiddle-factors in bitrev order.
   wlimit = 8*((N/2) - 1)

DO 20 stage = 1,m
           groups = powers2(m-stage)
C groups=number of times the twiddle factors are used, ie, the number of
C  smaller DFTs the stage is split into.

C   offset gets 1,2,4,8,...N/2
           offset = powers2(stage-1)
           wincr = groups
           call ditstep(a,w,groups,offset,wincr,wlimit)
20 CONTINUE

   RETURN
   END
C------------
```

```
//--------------
// ditstep.ss: do one stage of fft butterflies
// DIT = Decimation in Time, radix-2, inplace, 1-dimension
// (C) Copyright 1989 INTEL Corporation. ALL RIGHTS RESERVED.
// 7/15/89
//------------
// Intel is not responsible for use nor for misuse of this program.
//------------
// Do one entire stage (n/2 butterflies). Sample invocation:
//  call ditstep(a,w,groups,offset,wincr,wlimit)
//================================================
// Inputs:
//   A= complex array of input, single-prec float
//        (complex stored as 4byte real, 4byte imag contiguously)
//   W= pointer to array of twiddle factors. Assuming W(k) is
//      CMPLX(cos(2pi*k/N)),-sin(2pi*k/N)) for k=0 to (N/2)-1.
//   offset = distance (except for scale-by-8byte sizeof(complex)) between
//      the 2 input values for each butterfly.
//      Offset also is the number of butterflies done per "group".
//   groups = N/(2*offset). The number of sub-DFTs this stage is split into.
//   wincr = distance (except for scale-by-8byte sizeof(complex)) between
//      successive w values for successive butterflies
//   wlimit =max index, in bytes, of W table.
//
// Outputs:
//   A= complex radix-2 butterflied version of input.
//
//------------------
define(astart, r16)  // input data base address
define(wstart,r17) //twiddle array ptr. Because w-contents depend on N,
// we will assume the caller has initialized w() array.
define(groups,r18) //groups=number of sub-DFTs this stage is split into.
define(offset,r19)  //offset (initially elements, mult by 8 to get bytes)
// between node and its dual (the 2 numbers to butterfly, ie. A and B)
define(wincr,r20) //increment between successive W values. Remains constant
// within a given stage.
define(wlimit,r21) //max index, in bytes, of W table.
define(wind,r22) //current index, in bytes, of W table.
define(offset2,r23)  //offset*2

define(decrem,r24) //bla decrement
define(somecount,r25) // bla counter

define(FEtch, r26)  //pointer to 1st component of butterfly (load)
define(STore,r27)  // " " 1st component of butterfly (store)

define(offsetp8,r28) //offset+8
```

```
// f4:f7 spare
define(ARe,f12)  //element A, real component
define(AIe,f13) // " ", imag
define(ARo,f14) // extra A value, for prefetch (o="odd")
define(AIo,f15)
define(BRe,f16)  //element B, real component
define(BIe,f17)
define(BRo,f18) // extra B value, for prefetch
define(BIo,f19)

define(ERe,f20) //A+(B*W), real (ER = AR + BR)
define(EIe,f21) // "   imag "
define(ERo,f22) // previous loop's value
define(EIo,f23) // "   imag "

define(FRe,f24) //A-(B*W), real
define(FIe,f25) // "       imag "
define(FRo,f26) // previous loop's value
define(FIo,f27) // "   imag "

define(PR, f28) //(B*W), real
define(PI, f29) //(B*W), imag

define(WRe,f30)  //W (twiddle factor), real part
define(WIe,f31)  // " " , imag

define(WRo,f10)  //W (twiddle factor), real part (EXTRA copy)
define(WIo,f11)  // " " , imag

.text
.align .quad
_ditstep_::
 ld.l      0(groups),groups  //fix Fortran call-by-ref
 ld.l      0(offset),offset  //
 shl       3,offset,offset  // change from elements to bytes
 shl       1,offset,offset2
 adds      8,offset,offsetp8

 fst.q   f8 ,-16(sp)++ //save "local" regs
 fst.q   f12,-16(sp)++ // " "

 adds      -1,groups,groups // pre-decrement for bnc usage, or bla usage
 adds      -16,r0,decrem //bla decrement

//  We code the last 2 stages as special cases:
//--------
 xor       8,offset,r0 //offset=1, special case, no complex mult, funny addressing
 bc        offset_1// (ASSUMING offset=1 means wincr=0, and no twiddle used)
 xor       16,offset,r0 //offset=2, special case, no complex mult
 bc        offset_2
//--------
 ld.l      0(wincr),wincr
 ld.l      0(wlimit),wlimit
```

```
 pfadd.ss f0,f0,f0
 pfadd.ss f0,f0,f0
 pfadd.ss f0,f0,f0  // init A1,A2,A3=0
 pfmul.ss f0,f0,f0
 pfmul.ss f0,f0,f0
 pfmul.ss f0,f0,f0
//--------
 // init pointers:
 shl       3,wincr,wincr   //scale for bytes.
 shl       1,wincr,wind //init wind =2*wincr

 pfld.d 0 ( wstart),f0
 pfld.d wincr ( wstart),f0
 adds      -8,astart,FEtch
 pfld.d wind (wstart),f0
 adds      wincr,wind,wind //wind now 3*wincr
// here fetch first set of B,W before bla-loop
 pfld.d  wind (wstart),WRe
 adds      wincr,wind,wind
//first Bfetch from offset, then 1st afetch from 0.
 fld.d  offsetp8 (FEtch),BRe //first B value

 and       wlimit,wind,wind //modulo-wlimit the w index
// We do modulo-addressing on W(), to keep the pfld pipeline full. We
//  never do a W-fetch beyond the end of the table.
// And the modulo-check needs to be done only every 4th pfld, as always
//  we use a multiple of 4 W() factors.

d.r2apl.ss f0,f0,f0  //clear Treg.
 adds -32,offset,somecount // bla counter (predecrement by 4 elements)
// -----------
// Definitions for pipe diagram:
//   Anew = E = A+(B*W)
//   Bnew = F = A-(B*W)
//     Let P=(B*W).
//--------------
//  (the complex multiply product, P, broken into 4 real mult and 2 adds):
//   WR = cos(), WI=-sin().
//   PR = K - L; where K= WR*BR, L=WI*BI
//   PI = N + M; where N= WI*BR, M=WR*BI
//   ER = AR + PR (Overwrites AR)
//   EI = AI + PI (   "     AI)
//   FR = AR - PR (   "     BR)
//   FI = AI - PI (   "     BI)

// For 1st time thru inner--loop, don't have correct values to store.
// Must do 1 loop before the loop, sans the stores.
//-----------------
first_bfly::  //fill pipe
```

```
                        // KR...KI...M1....M2....M3    T   A1....A2....A3....Write
d.r2pt.ss WRe,f0,f0     // WRe  -    -     -     -     -   -     -     -     -
 pfld.d  wind (wstart),WRo
d.i2st.ss  WIe,f0,f0  //       WIe
 adds       wincr,wind,wind
d.r2apl.ss f0 ,BRe,f0 //            K0    -     -     -   -     -     -     -
 fld.d  8 (FEtch)++,ARe //first A value
d.pfmul.ss WIe,BIe,f0 //            L0    K0    -     -   -     -     -     -
 pfld.d  wind (wstart),WRe
d.r2pt.ss WRo,BIe,f0  // WRo       M0    L0    K0    -   -     -     -     -
 fld.d  offsetp8 (FEtch),BRo
d.ratls2.ss f0 ,PR ,f0//            -     M0    L0    K0  -     -     -     -
 adds       wincr,wind,wind
d.i2st.ss  WIo,BRe,f0 //       WIo  N0    -     M0    K0  K-L0  -     -     -
 nop
//.....................................................................
d.r2apl.ss f0 ,BRo,f0 //            K1    N0    -     M0  -     PR0
 and        wlimit,wind,wind
d.pfsub.ss f0 ,PI ,f0 //            K1    N0    -     M0  -     -     PR0
 fld.d  8 (FEtch)++,ARo
d.pfadd.ss ARe,PR ,PR //            K1    N0    -     M0  ER0   -     -     PR0
 fld.d  offsetp8 (FEtch),BRe
d.pfmul.ss WIo,BIo,f0 //            L1    K1    N0    M0  ER0   -     -     -
 nop
d.r2pt.ss WRe,BIo,f0  // WRe       M1    L1    K1    M0  M+N0  ER0   -     -
 bla        decrem,somecount,restart //init LCC
d.ratls2.ss ARe,PR ,f0//            -     M1    L1    K1  FR0   PI0   ER0   -
 nop
restart::
d.i2st.ss WIe,BRo,ERe//       WIe  N1    -     M1    K1  K-L1  FR0   PI0   ER0
 adds       -16,astart,STore  // ptrs init 16 low, for fst.q instructions
//-------------------
// Each butterfly = 1 complx multiply, 1 complx add, 1 complx subtract
//  =         4 multiply, 3 add, 3 subtract
//            3 8-byte fetches (A, B, W)
//            2 8-byte stores (A, B)
//
// 7 cycles per butterfly
//
// inner_loop: iterates "offset/2" times
//   for each group. It does 2 butterflies per iteration

// AR/AI fetches need to be a cycle behind BR/BI fetches here. So we
//  must index with offset+8 into B.
// AR is used 1/2 loop before AI.
// Pattern= AI0,AR1,BR2,BI2;AI1,AR2,BR3,BI3.

inner_loop::          // KR...KI...M1....M2....M3    T   A1....A2....A 3....Write
d.r2apl.ss AIe,BRe,PI //            K2    N1    -     M1  EI0   PR1   FR0   PI0
 pfld.d  wind (wstart),WRo
d.pfsub.ss AIe,PI ,FRe//            K2    N1    -     M1  FI0   EI0   PR1   FR0
 fld.d  8(FEtch)++,ARe
d.pfadd.ss ARo,PR ,PR //            K2    N1    -     M1  ER1   FI0   EI0   PR1
 fld.d  offsetp8 (FEtch),BRo
d.pfmul.ss WIe,BIe,f0 //            L2    K2    N1    M1  ER1   FI0   EI0   -
 adds   wincr,wind,wind
```

```
d.r2pt.ss WRo,BIe,EIe // WRo        M2    L2    K2        M+N1  ER1   FI0   EI0
 pfld.d wind (wstart),WRe
d.ratls2.ss ARo,PR ,FIe//          -     M2    L2    K2  FR1   PI1   ER1   FI0
 adds   wincr,wind,wind
d.i2st.ss  WIo,BRe,ERo//      WIo  N2    -     M2    K2  K-L2  FR1   PI1   ER1
 and    wlimit,wind,wind //modulo.
                      // KR...KI...M1....M2....M3    T   A1....A2....A3....Write
d.r2apl.ss AIo,BRo,PI //           K3    N2    -     M2  EI1   PR2   FR1   PI1
 nop
d.pfsub.ss AIo,PI ,FRo//           K3    N2    -     M2  FI1   EI1   PR2   FR1
 fld.d  8 (FEtch)++,ARo
d.pfadd.ss ARe,PR ,PR //           K3    N2    -     M2  ER2   FI1   EI1   PR2

 fld.d offsetp8 (FEtch),BRe
d.pfmul.ss WIo,BIo,f0 //           L3    K3    N2    M2  ER2   FI1   EI1   -
 nop
d.r2pt.ss WRe,BIo,EIo // WRe       M3    L3    K3        M+N2  ER2   FI1   EI1
 fst.q ERe,16(STore)++ //update ERe/EIe/ERo/EIo
d.ratls2.ss ARe,PR ,FIo//          -     M3    L3    K3  FR2   PI2   ER2   FI1
 bla decrem,somecount, inner_loop
d.i2st.ss  WIe,BRo,ERe//      WIe  N3    -     M3    K3  K-L3  FR2   PI2   ER2
 fst.q  FRe, offset (STore)
 //update FRe/FIe/FRo/FIo

end_inner_loop:: //KEEP Pipelines full
// RE-init pointers for fetches
d.fiadd.ss f0,f0,f0
 adds   offset2,astart,astart //bump to next group
      //redo A,B fetches, with proper ptr.
d.fiadd.ss f0,f0,f0
 fld.d  offset (astart),BRe //get first BR/BI in next group
d.fiadd.ss f0,f0,f0
 adds   -8,astart,FEtch

last_bfly:: //do final 2 butterflies, start next group
                      // KR...KI...M1....M2....M3    T   A1....A2....A3....Write
d.r2apl.ss AIe,BRe,PI //           K0    N3    -     M3  EI2   PR3   FR2   PI2
 pfld.d wind (wstart),WRo
d.pfsub.ss AIe,PI ,FRe//           K0    N3    -     M3  FI2   EI2   PR3   FR2
 fld.d  8(FEtch)++,ARe
d.pfadd.ss ARo,PR ,PR //           K0    N3    -     M3  ER3   FI2   EI2   PR3
 fld.d  offsetp8 (FEtch),BRo
d.pfmul.ss WIe,BIe,f0 //           L0    K0    N3    M3  ER3   FI2   EI2   -
 adds       wincr,wind,wind
d.r2pt.ss WRo,BIe,EIe // WRo       M0    L0    K0        M+N3  ER3   FI2   EI2
 pfld.d wind (wstart),WRe
d.ratls2.ss ARo,PR ,FIe//          -     M0    L0    K0  FR3   PI3   ER3   FI2
 adds       wincr,wind,wind
d.i2st.ss  WIo,BRe,ERo//      WIo  N0    -     M0    K0  K-L0  FR3   PI3   ER3
 and        wlimit,wind,wind //modulo
//.....................................................
d.r2apl.ss AIo,BRo,PI //           K1    N0    -     M0  EI3   PR0   FR3   PI3
 adds -32,offset,somecount // reset bla counter
d.pfsub.ss AIo,PI ,FRo//           K1    N0    -     M0  FI3   EI3   PR0   FR3
 fld.d  8 (FEtch)++,ARo
```

```
d.pfadd.ss ARe,PR ,PR //          K1    NO    -     MO  ERO   FI3   EI3   PRO
 fld.d  offsetp8 (FEtch),BRe
d.pfmul.ss WIo,BIo,f0 //          L1    K1    NO    MO  ERO   FI3   EI3   -
 bla       decrem,somecount,nowhere //re-init LCC=1
d.r2pt.ss WRe,BIo,EIo // WRe      M1    L1    K1        M+NO  ERO   FI3   EI3
 adds -1,groups,groups
nowhere::
d.ratls2.ss ARe,PR ,FIo//         -     M1    L1    K1  FRO   PIO   ERO   FI3
 fst.q  ERe,16(STore)++
d.fnop
 bnc.t restart //branch on value of groups
d.fnop
 fst.q  FRe, offset (STore)

end_last_bfly::
d.fnop
 br endit
fiadd.ss f0,f0,f0
 fst.q  FRe, offset (STore) //repeated for bnc.t untaken case
.align      .quad
//============================================================
offset_1::
// want FEtch=0,2,4,6,8,... elements. ASSUMING wincr=0,
//  and that w=(1,0), so that no complex mult needed.
//  E=A+B, F=A-B. (Per double-butterfly loop:  8 pfadd,4 dword fld, 4 fst,
//  1 bla)(fld.q used to reduce # flds)
//  Performance = 4 cyc/bfly best case.

//Redefine regs for fld.q,fst.q usage, when A and B adjacent:
define(AR3,f12)  //element A, real component
define(AI3,f13) // " ", imag

define(BR3,f14) //element B, real component
define(BI3,f15)
define(AR4,f16) // extra A value, for prefetch
define(AI4,f17)
define(BR4,f18)
define(BI4,f19)

define(ER3, f20) //A+B, real (ER = AR + BR)
define(EI3, f21) // "   imag "
define(FR3, f22) //(A-B), real
define(FI3, f23) // "     imag

define(ER4,f24) //A+B, real
define(EI4,f25) // "   imag
define(FR4,f26) //(A-B), real
define(FI4,f27) // "   imag
//=======================================
 adds       -16,astart,FEtch
 fld.q  16 (FEtch)++,AR4
 adds       -1,groups,somecount // bla counter (predecremented already by 1)
   //using groups=blacount on the offset_1 loop, intentionally.
 adds       -16,FEtch,STore
//startup the loop:
```

```
// ---------------------// A1......A2......A3......Write:
d.pfadd.ss AR4,BR4,f0   // ARn+BRn -       -       -
 fld.q  16 (FEtch)++,AR3
d.pfadd.ss AI4,BI4,f0   // AIn+BIn ERn     -       -
 adds       -2,r0,decrem //2 bflies per loop
d.pfsub.ss AR4,BR4,f0   // ARn-BRn EIn     ERn     -
 bla decrem,somecount, offsetl_loop  //init LCC
d.pfsub.ss AI4,BI4,ER4  // AIn-BIn FRn     EIn     ERnext
 nop
// ---------------------// A1......A2......A3......Write:
offsetl_loop::
d.pfadd.ss AR3,BR3,EI4  // AR+BR   FI-     FR-     EI-
 nop
d.pfadd.ss AI3,BI3,FR4  // AI+BI   ER      FI-     FR-
 fld.q  16 (FEtch)++,AR4
d.pfsub.ss AR3,BR3,FI4  // AR-BR   EI      ER      FI-
 fst.q      ER4,16(STore)++
d.pfsub.ss AI3,BI3,ER3  // AI-BI   FR      EI      ER
 nop
d.pfadd.ss AR4,BR4,EI3  // AR2+BR2 FI      FR      EI
 fld.q  16 (FEtch)++,AR3
d.pfadd.ss AI4,BI4,FR3  // AI2+BI2 ER2     FI      FR
 nop
d.pfsub.ss AR4,BR4,FI3  // AR2-BR2 EI2     ER2     FI
 bla decrem,somecount, offsetl_loop
d.pfsub.ss AI4,BI4,ER4  // AI2-BI2 FR2     EI2     ERnext
 fst.q  ER3,16(STore)++
//---------
end_offsetl_loop::
d.fiadd.ss f0,f0,f0
 br endit
fiadd.ss f0,f0,f0
  nop
//---------
.align .quad
offset_2::
// want FEtch=0,1;4,5;8,9;12,13;... elements.
// ASSUMING wincr=N/4  (W_addr=0,N/4,0,N/4,0,...). Trivial W() factors.
//  Even-indexed elements identical to offset_1,W=W0, no complex mult.
//    So              EReven=(AR+BR), EIeven=(AI+BI).
//    So              FReven=(AR-BR), FIeven=(AI-BI).

//  Odd components have W=(0,-1).  So B*W = (BI,-BR).
//   So ERodd=Re(A+(B*W)) = (AR+BI)    EIodd=(AI-BR).
///  So FRodd=Re(A-(B*W)) = (AR-BI)    FIodd=(AI+BR).
//  Each fld.q fetches AReven,AIeven,ARodd,AIodd.

//Assume ERe,EIe,ERo,EIo are 4 contiguous regs.
//Assume FRe,FIe,FRo,FIo are 4 contiguous regs.
//Assume ARe,AIe,ARo,AIo are 4 contiguous regs.
```

```
 adds       -16,astart,FEtch
 fld.q  16 (FEtch)++,ARe
 fld.q  16 (FEtch)++,BRe
 adds       0,groups,somecount //bla counter
//startup the loop:
// --------------------// A1......A2......A3......Write:
  pfadd.ss ARe,BRe,f0   // AR+BRe                    -
  pfadd.ss AIe,BIe,f0   // AI+BIe  ER                -
d.pfadd.ss ARo,BIo,f0   // ARo+BIo EI      ER        -
 nop
d.pfsub.ss AIo,BRo,ERe  // AIo-BRo ERo     EI        ER
 nop
d.pfsub.ss ARe,BRe,EIe  // AR-BRe  EIo     ERo       EI
 ads       -1,r0,decrem //2 bflies per loop,but groups is half desired value.
d.pfsub.ss AIe,BIe,ERo  // AI-BIe  FR      EIo       ERo
 adds       -16,astart,STore
d.pfsub.ss ARo,BIo,EIo  // ARo-BIo FI      FR        EIo
 bla decrem,somecount, offset2_loop  //init LCC
d.pfadd.ss AIo,BRo,FRe  // AIo+BRo FRo     FI        FR
 nop
offset2_loop::
d.fnop
 fld.q  16 (FEtch)++,ARe//fetch AR,AI,ARo,AIo

d.fnop
 fld.q  16 (FEtch)++,BRe
// -------              // A1......A2......A3......Write:
d.pfadd.ss ARe,BRe,FIe  // AR+BRe  FIo     FRo       FI
 nop
d.pfadd.ss AIe,BIe,FRo  // AI+BIe  ER      FIo       FRo
 nop
d.pfadd.ss ARo,BIo,FIo  // ARo+BIo EI      ER        FIo
 fst.q      ERe,16(STore)++ //update ER ,EI ,ERo,EIo
d.pfsub.ss AIo,BRo,ERe  // AIo-BRo ERo     EI        ER
 nop
d.pfsub.ss ARe,BRe,EIe  // AR-BRe  EIo     ERo       EI
®nop
d.pfsub.ss AIe,BIe,ERo  // AI-BIe  FR      EIo       ERo
 fst.q  FRe,16(STore)++
d.pfsub.ss ARo,BIo,EIo  // ARo-BIo FI      FR        EIo
 bla decrem,somecount,offset2_loop
d.pfadd.ss AIo,BRo,FRe  // AIo+BRo FRo     FI        FR
 nop
endit::
// restore regs
fiadd.ss f0,f0,f0 //exit DIM
 fld.q   0(sp),f12
fiadd.ss f0,f0,f0 //last DIM pair
 fld.q   16(sp),f8
adds    32,sp,sp
 bri r1
  nop
//=============================================
```

```
C---------------
C File: dirr.f
C  FFT - Decimation in Freq, radix-2, inplace, 1-dimen,
C        REAL input
C   Intel is not responsible for use nor misuse of this code.

C  8/14/89

C Inputs:
C   A= REAL array of input, up to 1024 pts, single-prec float
C   M= log of number of pts
C    = (Number of stages of FFT)
C   N = number of points. ie, N= 2**M = number of pts
C   W= complex array of twiddle factors, length N/2.
C   REV= 0 if bitreversed output ok. 1=must re-order output
C      (REV will be ignored, and output will be properly ordered. Bit
C       reversal WILL be done.)
C
C Outputs:
C  A= complex fft of input A, but only the positive frequency half.
C     Length = N/2+1 complex numbers. A(0:n/2)
C
   subroutine dirr(a,m,N,W,REV)
   integer m,N, i, j,k, REV,wlimit
   integer offset, stage, groups, wincr,powers2(0:10)
   real a(N)
   complex w(N/2),temp

   data powers2 /1,2,4,8,16,32,64,128,256,512,1024/
C Powers2 to avoid calls to POW, DIV

C Twiddle factor array w(k) has (cos,-sin) of 2pi*k/N
CC Assume the caller provides w(k) constants ALREADY initialized
C------------
C Pre-touch data, for 8kByte fft: (2048 points real)
   IF (N .gt. 1025) THEN
           call fetch(a,%VAL(n/2))
   ENDIF
C------------
   wlimit = 8*((N/2) - 1)

C "DO 20" stage-loop: doing Complex FFT on length N/2 array. Twiddles are
C   for a length N array, so wincr gets scaled by 2.
   DO 20 stage = 1,m-1
           groups = powers2(stage-1)
C groups=number of times the twiddle factors are used, ie, the number of
C  smaller DFTs the stage is split into.

C  offset gets N/4,N/8,N/16,...
           offset = powers2(m-1-stage)
           wincr = groups * 2
           call difstep(a,w,groups,offset,wincr,wlimit)
20 CONTINUE

   call bitrev(a,%VAL(M-1),n/2)
   call realfix(a,w,%VAL(n))

   RETURN
   END
C------------
```

```
// realfix.ss:  This is i860(tm) CPU assembly code to revise data from an
//   N/2 length Complex FFT.
//   (assumes the input data fed to Complex FFT was N real values)
//
//  INTEL is not responsible for use nor misuse of this code.
//
// 8/14/89
//   This 18-cycle-butterfly loop may be sub-optimal.
//
//   output = overwrite the data array used for input. Results are
//    complex.     Re0,Im0,Rel,Iml,..., Re(N/2),Im(N/2).
//     NOTE that output array is 1 element longer than input.
//
//  Input is H(k), output is F(k)...
//   F(k)=.5*( H(k)+ Hconj(N/2-k) -j*(H(k) -Hconj(N/2-k))*Wconj(k))
//
//  Algorithm from "Numerical Recipes in C", by Flannery, Press, Teukolsky, and
//    Vetterling, Cambridge Univ. Press 1988, p.417.
//*************************/

//* The C-version of realfix: */ void realfix_(a,w,n)
///*Input =
// a(0:n+1): length n/2+1 complex array. Entries 0:n/2-1 are the complex FFT
// *   result, in correct (NON BIT REVERSED) order. Entry n/2 is undefined.
// * w: length n/2 complex array of twiddles. (cos,-sin(2pi*k/n))
// * n: call-by-value, number of REAL input samples

// *Output =
// *  a(0:n+1): length n/2+1 complex array.
// *    Format is Re0,Im0,Rel,Iml,..., Re(N/2),Im(N/2).
// *    NOTE: To generate entire N-length complex output spectrum, you can copy
// *     conjugate of element(i) to element(N-i).
// */
//float a[], w[]; int n;   { int aptr,bptr, wptr; float half=0.5,
//      AR,AI,BR,BI, /* input values for A,B*/
//      PR,PI,SR,SI,DR,DI, /*temporary differences,sums,products*/
//      K,L,M,N, /*temporary products */
//      ER,EI,ERD,EID,
//      FR,FI,FRD,FID,
//      WR,WI;

///*We do first and last elements as special case(Imag=0, W=(1,0))*/
// AR = a[0];       AI = a[1];
// a[0] = AR + AI; a[1] = 0;
// a[n] = AR - AI; a[n+1] = 0;
```

```
//for(aptr=2, bptr=(n-2), wptr=2; aptr < n/2; aptr +=2, bptr -=2, wptr +=2)
//{WR = w[wptr];            WI = w[wptr+1];
// AR = a[aptr];    AI = a[aptr+1];
// BR = a[bptr];    BI = a[bptr+1];
// /* aptr =2,4,6...,14;  bptr=30,28,26,...,18 (if n=32) */
// /* Note that there is no need to revise the value at the middle of the
//    list, as it is already correct. (.5*(H(n/4)+Hconj(n/4)) */
// SI = (AI + BI);
// DR = (BR - AR);
// K = WR*SI; L= WI*DR;     PR = K-L;
// M = WR*DR; N= WI*SI;     PI = M+N;
// SR = (AR + BR);
// DI = (AI - BI);

// ERD = SR+PR; ER = half*ERD;
//  a[aptr] = ER;
// EID = DI+PI; EI = half*EID;
//  a[aptr+1]= EI;
// FRD = SR-PR; FR = half*FRD;
//  a[bptr] = FR;
// FID = PI-DI; FI = half*FID;
//  a[bptr+1]= FI;  } /*end of for-loop */ }
//************* End of C-code for realfix.***********************
.text
.align .quad
//--------------
define(astart, r16)  //input data base address

define(wptr,r17) // pointer to W table. Because w-contents depend on N,
// we will assume the caller has initialized w() array.
define(N,r18) //
define(aptr, r20)  //pointer to 1st component of butterfly (load)
define(bptr, r21)  //pointer to 2nd component of bfly (load); DOWNCOUNTER

define(decrem,r24) //bla decrement
define(count,r25) // bla counter

define(WR, f18)  //W (twiddle factor), real part
define(WI, f19)  // " " , imag

define(AR, f12)  //element A, real component
define(AI, f13) // " ", imag
define(ARo,f14) // extra A value, for prefetch (o="odd")
define(AIo,f15)
define(BR, f16)  //element B, real component
define(BI, f17)

define(ER, f20) //Result of butterfly which overwrites AR
define(EI, f21) // "    " " " AI

define(half,f22)  //constant 0.5

define(FR, f24) //Result of butterfly which overwrites BR
define(FI, f25)
define(PR,f26)
define(PI,f27)

define(DR, f28)
define(DI, f29)
```

```
define(SR, f30) //Sum of A+B, real part
define(SI, f31) // " ", imag "

.data
.align .double
halfloc::  .float 0.5
//--------
.text
.align .quad
_realfix_::
 fst.q   f12,-16(sp)++ //save "local" regs
 adds      -4,r0,decrem //bla decrement
//--------
// We do not bother to initialize FP pipes to zero here, as we assume
//  this routine is called after another,"safe", pipelined FP routine.

pfld.l halfloc,f0
pfld.d 8( wptr)++,f0 //skip W(0) intentionally. Is a trivial (1,0) value
// init pointers:
 adds       0,astart,aptr
pfld.d 8( wptr)++,f0
 shl        2,N,bptr  //bptr=total # bytes of input data
pfld.d 8( wptr)++,half  //0.5 into an fpr
 adds       bptr,astart,bptr  // bptr points to a(N)

// here fetch first set of A,B,W before bla-loop
pfld.d 8( wptr)++,WR
 fld.d 0 (aptr),AR //for 1st and last elements
 adds -8,N,count // bla counter (predecrement by 2 butterflies worth)
// -----------
// Do n/4 butterflies: (computing only N/2 elements of complex output, because
//    the second N/2 are just complex conjugates of the 1st N/2)

// Definitions for pipe diagram:
//   WR = cos(), WI=-sin().
//   DR = BR - AR; (diffence of Real components of A,B)
//   DI = AI - BI; (diffence of Imag components)
//   SR,SI = sum of A,B
//   PR = K - L; where K= WR*SI, L=WI*DR
//   PI = M + N; where M= WR*DR, N=WI*SI
//  (ER,EI)=complex result to overwrite A.
//  (FR,FI)=" " " " B.

first_fly::  //fill pipe.
// For 0th butterfly:
// AR = a[0];       AI = a[1];
// a[0] = AR + AI;  a[1] = 0;
// a[n] = AR - AI;  a[n+1] = 0;

                     // KR..KI..M1....M2....M3   T   A1....A2....A3....Write
  r2pt.ss  f0,f0,f0  // 0       0
  mrmlp2.ss AR,AI,f0 //         0     0     -       ER0   -     -     -
  mrmls2.ss AR,AI,f0 //         0     0     0       FR    ER    -     -
 fld.d  8 (aptr)++,AR
 fld.d  -8(bptr)++,BR
d.pfadd.ss f0,f0,f0  //         0     0     0       0     FR    ER    -
d.pfadd.ss  f0,f0,ER //         0     0     0       0     0     FR    ER0
```

```
d.ralp2.ss  AI ,BI ,FR //          -     0     0     -     SI1   -     -     FR0
 nop
d.mrmls2.ss BR ,AR ,EI //          -     -     0     -     DR1   SI1   -     EI0
 fst.d ER,-8(aptr)
d.mr2pt.ss  WR ,f0, FI // WR       -     -     -     -     -     DR1   SI1   FI0
 fst.d FR, 8(bptr)
d.ralp2.ss  BR ,AR ,SI //          K1    -     -     -     SR1   -     DR1   SI1
 andh 0x8000,count,r0  //check for negative
d.ml2tpm.ss WI ,DR ,DR //          L1    K1    -     -     -     SR1   -     DR1
 bnc  endfix
d.r2pt.ss   half,DR, f0 //half     M1    L1    K1    -     -     -     SR1   -
 nop
d.ml2ttpa.ss WI ,SI ,SR//          N1    M1    L1    K1    -     -     -     SR1
 nop
d.i2st.ss    f0 ,f0 ,f0//     f0    -     N1    M1    K1    PR1   -     -     -
 nop
                        // KR..KI..M1....M2....M3   T     A1....A2....A3....Write
d.ratls2.ss AI ,BI ,f0 //          -     -     N1    M1    DI1   PR1   -     -
 nop
d.i2pt.ss    f0 ,f0,  f0//     f0    -     -     -     M1    PI1   DI1   PR1   -
 fld.d  8 (aptr)++,AR
d.r2apl.ss  SR ,f0,  PR//          -     -     -     -     ERD   PI1   DI1   PR1
 fld.d  -8(bptr)++,BR
d.rals2.ss  SR ,PR, DI //          -     -     -     -     FRD   ERD   PI1   DI1
 pfld.d  8( wptr)++,WR
d.r2apl.ss  DI ,f0,  PI//          -     -     -     -     EID   FRD   ERD   PI1
 nop
d.rals2.ss  PI ,DI ,f0 //          ER1   -     -     -     FID   EID   FRD   -
 nop
d.ralp2.ss  f0 ,f0 ,f0 //          FR1   ER1   -     -     -     FID   EID   -
 nop
d.rals2.ss  f0 ,f0 ,f0 //          EI1   FR1   ER1   -     -     -     FID   -
 bla decrem,count,fix_loop
d.pfadd.ss  f0 ,f0 ,FI //          EI1   FR1   ER1   -     -     -     -FID
 nop
//-------------------
// Each butterfly = 1 complx multiply, 3 complx add, 1 real multiply
// =        8 multiply, 10 add/subtract
//          3 8-byte fetches (A, B, W)
//          2 8-byte stores (E, F)
//
// approx. 18 cycles per butterfly
//
```

```
fix_loop::              // KR..KI..M1....M2....M3   T   A1....A2....A3....Write
d.mr2pt.ss  f0 ,FI ,ER // 0        FI1   EI1   FR1  -   -     -     -      ER1
 nop
d.mrm1p2.ss AI ,BI ,FR //          -     FI1   EI1  -   SI2   -     -      FR1
 nop
d.mrm1s2.ss BR ,AR ,EI //          -     -     FI1  -   DR2   SI2   -      EI1
 fst.d ER,-8(aptr)
d.mr2pt.ss  WR ,f0, FI // WR       -     -     -    -   -     DR2   SI2    FI1
 fst.d FR, 8(bptr)
d.ralp2.ss  BR ,AR ,SI //          K2    -     -    -   SR2   -     DR2    SI2
 andh 0x8000,count,r0  //check for negative
d.m12tpm.ss WI ,DR ,DR //          L2    K2    -    -   -     SR2   -      DR2
 bnc  endfix
d.r2pt.ss  half,DR, f0 //half      M2    L2    K2   -   -     -     SR2    -
 nop
d.m12ttpa.ss WI ,SI ,SR//          N2    M2    L2   K2  -     -     -      SR2
 nop
d.i2st.ss   f0  ,f0 ,f0//     f0   -     N2    M2   K2  PR2   -     -      -
 nop
                        // KR..KI..M1....M2....M3   T   A1....A2....A3....Write
d.ratls2.ss AI ,BI , f0//          -     -     N2   M2  DI2   PR2   -      -
 nop
d.i2pt.ss   f0 ,f0,  f0//     f0   -     -     -    M2  PI2   DI2   PR2    -
 fld.d  8 (aptr)++,AR
d.r2apl.ss  SR ,f0,  PR//          -     -     -    -   ERD   PI2   DI2    PR2
 fld.d -8(bptr)++,BR
d.rals2.ss  SR ,PR,  DI//          -     -     -    -   FRD   ERD   PI2    DI2
 pfld.d  8( wptr)++,WR
d.r2apl.ss  DI ,f0,  PI//          -     -     -    -   EID   FRD   ERD    PI2
 nop
d.rals2.ss  PI ,DI ,f0 //          ER2   -     -    -   FID   EID   FRD    -
 nop
d.ralp2.ss  f0 ,f0 ,f0 //          FR2   ER2   -    -   -     FID   EID    -
 nop
d.rals2.ss  f0 ,f0 ,f0 //          EI2   FR2   ER2  -   -     -     FID    -
 bla decrem,count,fix_loop
d.pfadd.ss  f0 ,f0 ,FI //          EI2   FR2   ER2  -   -     -     -      FID
 nop
//---------
endfix::
// restore regs
fiadd.ss f0,f0,f0 //exit DIM
 fld.q   0(sp),f12
fiadd.ss f0,f0,f0 //last DIM pair
adds    16,sp,sp
 bri r1
  nop
//------------
```

```
      PROGRAM FFTTEST
c  file = real.f
C
C   1-D FFT TEST PROGRAM
C
C 8/14/89

C Intel assumes no responsibility for use or misuse of this code.
C-----------------
      PARAMETER (IREV=1)
      character*8 really
      PARAMETER (REALLY='real')
c     PARAMETER (REALLY='complex')
      PARAMETER (TIMEIT=0, CACHETIME=0)
c REALLY='real' means real-only input, otherwise assume complex input
      DATA IT/200000/
c      PARAMETER (N=2048,M=11)
      PARAMETER (N=1024,M=10)
c      PARAMETER (N=512,M= 9)
c      PARAMETER (N=256,M= 8)
c      PARAMETER (N=128,M= 7)
c      PARAMETER (N=64,M= 6)
c      PARAMETER (N=32,M= 5)
c       PARAMETER (N=16, M=4)
     PARAMETER (PI=3.1415926536)
     COMPLEX X2(N),X(N),X3(N), W(N/2)

   Real  ASQR(N),ASQR2(N),XR(N+2),XR1(N+2),XR2(N+2),XR3(N +2)
   complex wtemp
   real rtemp
C
   PRINT *,' FFT test program ....'
   print *,'================================'
   IF (IREV .eq. 0) THEN
    print *,'NOT counting time for bit-reversal.'
    print *,'DO NOT expect matching answers,without bit-rev'
   ELSE
    print *, 'Time for bit-reversal included.'
   ENDIF

    print *, 'Time for cache writeback and fills...'
   IF (CACHETIME .eq. 0) THEN
    print *,' NOT included, if iterating.'
   ELSE
    print *,' ... included.'
   ENDIF

    print *, '================================'
    print *, 'If iterating... Number of Iterations =',IT
    print *, '================================'
    print *, 'Number of Points     = ', N
    print *, '(',REALLY,' data)'
    print *, '================================'
```

```
 C-----------------
C Init twiddle factor array w(k) with (cos,-sin) of 2pi*k/N
   rtemp = 2.0*pi/N
   wtemp= CMPLX(cos(rtemp), -sin(rtemp))
   w(1) = (1.0, 0.0)
   DO  200  k  =  2,N/2
200        w(k) = wtemp * w(k-1)
cc print *,' W (twiddle) initialization completed......'
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C   INITIALIZE input data
C
      DO 100 I = 1, N
c:constant:
c     Treal = 1.0
c     Timag = 0.0

c:squarewave:
cc IF (I .lt. N/2) THEN
cc   Treal = 1.0
cc   Timag = 0.5

cc ELSE
cc   Treal = 0.0
cc   Timag = 0.0
cc ENDIF
C: ramp function:
     Treal = I - 1.0
     Timag = Treal + 0.5
   IF (REALLY .ne. 'real') THEN
         X(I) = CMPLX (Treal, Timag)
         X2(I) = CMPLX (Treal, Timag)
         X3(I) = CMPLX (Treal, Timag)
   ELSE
     X(I)  = CMPLX (Treal,0.0)
     X2(I) = CMPLX (Treal,0.0)
     XR(I) = Treal
     XR1(I) = Treal
     XR2(I) = Treal
     XR3(I) = Treal
   ENDIF
100   CONTINUE
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      CALL fft (X2, M, N)
cc Subroutine fft is Decimation-In-Time, Fortran version.
      CALL dirr(XR,M,N,W,1)
c   (Assuming dirr produces inplace result, items 0:N/2 complex results)
```

```
ccccccccccccccccccccccccccccccccccccccc
   IF (IREV .ne. 0) THEN
   IF (TIMEIT .eq. 0) THEN
   call vcompare(XR,X2,N/2+2)
   call cmags(XR,N/2+1,ASQR)
c cmags to take squared magnitude of complex values in X
   call cmags(X2,N,ASQR2)
c----------------------c
C print non-zero results:
   J=0

   DO 700 I = 1,N/2+1
   IF ((ASQR(I) .GT. 1.0) .OR. (ASQR2(I) .GT. 1.0)) THEN
      WRITE (6,22)  (I-1), ASQR(I), ASQR2(I)
22    FORMAT (' I-1=',I4,'  ASQR(I)= ',F14.2, '  ASQR2(I)= ',F14.2//)
   J = J+1
   IF (J .GT. 32) GOTO 725
   ENDIF
700        CONTINUE

725     CALL TIME
   ENDIF
   ENDIF

   IF (TIMEIT .ne. 0) THEN
ccccccccccccccccccccccccccccccccccccccc
cc- Timing loop follows:

     print *,' Start Ass.FFT'
   IF (CACHETIME .eq. 0) THEN
     DO 500 I = 1, IT,4
C Reuse same array, so cache fill and writeback time NOT included.
          CALL dirr(XR, M, N,W,IREV)
          CALL dirr(XR, M, N,W,IREV)
          CALL dirr(XR, M, N,W,IREV)
500       CALL dirr(XR, M, N,W,IREV)
   ELSE
     DO 504 I = 1, IT,4
C Alternating between XR,XR1,XR2,XR3 should provide cache misses.
          CALL dirr(XR, M, N,W,IREV)
          CALL dirr(XR1, M, N,W,IREV)
          CALL dirr(XR2, M, N,W,IREV)
504       CALL dirr(XR3, M, N,W,IREV)
   ENDIF
     print *,' END Ass. FFT'
ccccccccccccccccccccccccccccccccccccccc
```

```
   ENDIF

      STOP
      END
c----------------------c
     subroutine vcompare(res,exp,n)
c VCOMPARE compares 2 vectors, prints out 1st few miscompares
c
     integer n, errcnt
     real res(n), exp(n)

     write(6,12)
12   format('*** VCOMPARE: vector comparison beginning ***')

   data errcnt/0/
        do 30 i = 1,n
             if(AINT(res(i)) .ne. AINT(exp(i))) then
c  {print out error, exit if alot already}
120                  print *,'*** Error in compares ***'
                     write(6,121) i
121                  format('  Item number = ',I6)
                     write(6,124) res(i), exp(i)
124                  format('  Res_=',F14.2,'  Expected_=',F14.2)
                     errcnt = errcnt + 1
                     if (errcnt .gt. 19) then
                            return
                     end if
          end if
30      continue

   if (errcnt .eq. 0) then
190  print *,' *** vector compares SUCCESSFUL ***'
   end if

99 return
   end
c----------------------c
```

```
C---------------
C file: fft.f

C FFT routine from Rabiner & Gold, 1975, who copied it
C  from Cooley, Lewis, Welch
C 6/02/89
C
C Decimation in Time, radix-2, inplace, 1-dimen
C Inputs:
C   A= complex array of input, up to 1024 pts, single-prec float
C        (maybe more than 1024, uncertain what limit is)
C  M= log of number of pts
C    = (Number of stages of FFT)
C  N = number of points.  ie, N= 2**M = number of pts
C
C Outputs:
C  A= complex fft of input A, in NON-bit-reversed order.
C
C w (twiddle factor) calculated by recursion. Supposedly takes 15% more
C  operations than keeping entire twiddle array as constants pre-allocated.
C
   subroutine fft(a,m,n)
   integer m,n, i, j,k, ndiv2,powers2(0:10)
   integer iplus,offset, stage, indexl, groups
   complex a(n),wtemp(2),w(11),temp

C Init twiddle factor array w() with (cos,-sin) of pi,pi/2,pi/4,...
       data w(1) /(-1.0,0.0) /
       data w(2) /(0.0,-1.0) /
       data w(3) /(0.7071068,-0.7071068)/
       data w(4) /(0.9238795,-0.3826834)/
       data w(5) /(0.9807853,-0.1950903)/
       data w(6) /(0.9951847,-0.0980171)/
       data w(7) /(0.9987955,-0.0490677)/
       data w(8) /(0.9996988,-0.0245412)/
       data w(9) /(0.9999247,-0.0122715)/
       data w(10) /(0.9999812,-0.0061359) /
       data w(11) /(0.9999953,-0.003068) /

   data powers2 /1,2,4,8,16,32,64,128,256,512,1024/
C Powers2 to avoid calls to POW, DIV

C Setup for bit-reversal loop:
   ndiv2 = n / 2
   j =1
C-----------
C "DO 7" loop to in-place-bit-reverse-shuffle input
   DO 7  i= 1, n-1
     IF (i .lt. j) THEN
           temp = a(j)
           a(j) = a(i)
           a(i) = temp
     ENDIF
     k = ndiv2
```

```
C  "While (j .gt. k)" /*decrease j by 2**something */
6     IF (j .gt. k) THEN
            j = j-k
            k = k / 2
            GOTO 6
      ENDIF
C  Add next lower power of 2 to j
7  j = j+k
C------------
C Special case for stage 1: no complex multiplies, simple add
C  (Performance enhancement)
   groups = 2
   offset = 1
   indexl = 1
C  i-loop iterates N/2 times for 1st stage (and would do twice N/4 x for 2nd)
CVD$         NODEPCHK
   DO 8 i = 1,n,2
     iplus = i + 1

     temp = a(iplus)
     a(iplus) = a(i) - temp
8    a(i) = a(i) + temp
C------------
C Special case for stage 2: no complex multiplies, simple add
C  (Performance enhancement)
   groups = 4
   offset = 2
   indexl = 1

C  i-loop iterates N/4 times for 2nd stage
C  1st call to i-loop,in stage2: indexl=1, wtemp(1)=(1,0)
CVD$         NODEPCHK
   DO 90 i = 1,n,4
     iplus = i + 2
     temp = a(iplus)
     a(iplus) = a(i) - temp
90   a(i) = a(i) + temp

   indexl = 2
CVD$        NODEPCHK
CVD$        NOVECTOR
   DO 92 i = 2,n,4
     iplus = i + 2
          temp = CMPLX(AIMAG(a(iplus)),-REAL(a(iplus)))
     a(iplus) = a(i) - temp
92   a(i) = a(i) + temp
CVD$        VECTOR
C------------
C "DO 20" stage-loop executed once for each of the (m) stages of FFT
C  (Except 1st and 2nd stage)
C  offset gets 4,8,16,32,64,128,256...
   DO 20 stage = 3,m
            groups = powers2(stage)
            offset = groups/2
            wtemp(1) =(1.0, 0.0)
C   One twiddle seed (W) calc per stage.
C   We pre-allocated w(12)-array with those values, avoid cos/sin calls
```

```
C-------------------
   DO 20 indexl = 1,offset

C "DO 10" i-loop does each butterfly of each stage, with varying twiddles
C    i-loop iterates N/2 times for 1st stage, N/4 x for 2nd, N/8 x for 3rd
C     stage, N/16 x for 4th stage,... 1 time for last stage.

CVD$       NODEPCHK
CVD$       ALTCODE
           DO 10 i = indexl,n,groups
             iplus = i + offset
             temp = a(iplus) * wtemp(1)
             a(iplus) = a(i) - temp
10           a(i) = a(i) + temp
20 wtemp(1) = wtemp(1) * w(stage)
   RETURN
   END
C------------
   subroutine cmags(a,n,asqr)
C Complex magnitude squared.
C Inputs:
C   A= complex array of input, single-prec float
C  N = number of input points (and output points)
C Ouput:
C   asqr = real squared magnitude (R*R + I*I), N elements, single-prec float

   integer n,i
   real asqr(n)
   complex a(n)

   DO 100 i = 1, n
     asqr(i) = (REAL(a(i))*REAL(a(i))) + (AIMAG(a(i))*AIMAG(a(i)))
100        CONTINUE
   RETURN
   END
```

```
## makefile for i860(tm) CPU FFTs (for Unix V/386 programming environment)
##   8/7/89
##
GH=/usr/i860/bin
GHL=/usr/i860/lib
CC=$(GH)/c860
FC=$(GH)/f860

CFLAGS= -OLM -X393 -X405 -X188 -X370

FFLAGS= -OLM -X370 -X393 -X71 -X422
## -X71 uses single-precision math routines

FLFLAGS= -Mx map -e start

LFLAGS= -Mx map -e _main
CLIB=$(GHL)/libc.a
MLIBPSR=$(GHL)/860mtlib.a

MLIB=$(GHL)/libm.a
FLIB=$(GHL)/libf.a

ASM=$(GH)/as860

FLINK=$(GH)/ld860 $(FLFLAGS)

RT=$(GHL)/s5lib.a

LIBS= $(FLIB) $(MLIBPSR) $(MLIB) $(CLIB) $(RT)

LIBCC= $(MLIB) $(CLIB) $(RT)
## NOTE: Order of linked files is CRUCIAL, other orders may give errors

.SUFFIXES:
.SUFFIXES: .f .c .s .ss .o .8

.IGNORE:
## .ignore causes make to ignore error codes from compilers

## To test Fortran plus assembler-fft-stage version:
FILE= ffttest.o fft.o diff.o bitrev.o difstep.o start.o time.o

## To test all-Fortran version of fft:
##FILE= ffttest.o fft.o diff.o difstepf.o start.o time.o

## To test REAL-input version of fft:
RFILE= real.o fft.o dirr.o realfix.o difstep.o bitrev.o start.o time.o

.f.o:
   $(FC) $(FFLAGS) $*.f
   $(ASM) -x -o $*.o $*.s

.c.o:
   $(CC) $(CFLAGS) $*.c
   $(ASM) -x -o $*.o $*.s
```

```
.s.o:
   m4 $*.s temp2.s
   $(ASM) -x -o $*.o temp2.s
ffttest.8: $(FILE)
   $(FLINK)  -o ffttest.8 $(FILE) $(LIBS)
real.8:    $(RFILE)
   $(FLINK)  -o real.8 $(RFILE) $(LIBS)

clean:
   rm -f *.o *.8

.ss.o:
   m4 $*.ss temp.s
   $(ASM) -x -o $*.o temp.s
```

```
//start.ss
// 8/18/89
// Fortran runtime startoff routine
//
   .text
   .globl  start
   .globl  finish
start::
   orh     h%_stack+262128+262144,r0,sp
   or      l%_stack+262128+262144,sp,sp
   adds    -16,sp,sp
   st.l    r1,12(sp)
   call    _main
   nop
finish::
   call    _exit
   nop
   .file   "start.c"

   .data
   .align  .quad
   .lcomm  _stack,262144+262144
   .end
//=============================================================

/* file: time.c.  Purpose: establish a label to use for breakpoints */
long       time_(x)
long       *x;
{  x = x+4;
   return((long) x);
}
long       timestop_(x)
long       *x;
{  x = x+4;
   return((long) x);
}
```

intel®

APPLICATION NOTE

PRELIMINARY
AP-452

December 1991

2

Designing a Memory Bus Controller for the 82495/82490 Cache

MARK ATKINS
ISIC SILAS
CHRIS KARLE

Order Number: 240957-001

Designing a Memory Bus Controller for the 82495/82490 Cache

CONTENTS

CONTENTS

Designing a Memory Bus Controller for the 82495/82490 Cache

CONTENTS

PAGE

FIGURES

CONTENTS

PAGE

FIGURES

TABLES

1.0 BACKGROUND

The Intel 82495 Cache Controller and 82490 Cache RAM form a high-speed cache subsystem for the Intel486 DX CPU (82495DX/490DX) or the i860 XP CPU (82495XP/490XP). The reader should be familiar with these chips, as described in:

1) i860 XP CPU Microprocessor Data Sheet (Intel order #240874)
2) Intel486 DX Microprocessor Data Sheet (Intel order #240440)
3) 82495XP Cache Controller/82490XP Cache RAM Data Sheet (Intel order #240956, June 1991)

   or Intel486 DX CPU Microprocessor Cache-Chip Set Data Sheet (Intel order # 241084, June 1991)

Diagrams of systems containing the 82495 and 82490 appear in Figure 1, and a more detailed diagram of the CPU/82495/82490 core appears in Figure 2. (Note: for simplicity, the 82495XP/82490XP and 82495DX/82490DX will be referred to generally as 82495/82490—the XP or DX should be inferred depending upon the CPU being utilized.) In such systems, the 82495 controls a cache external to the CPU, and includes the cache tags. It can interface gluelessly to an Intel486 DX CPU or i860 XP CPU microprocessor, allowing the processor bus to run at 50 MHz with zero wait-states, while the memory bus can remain at a lower frequency. Both writeback and writethrough protocols are supported. Concurrent operations can occur simultaneously on the local CPUbus and the shared memory bus. All requisites for multiprocessors are included in the 82495, Intel486 DX CPU, and i860 XP CPUs, but the 82495 also is useful for a uniprocessor system performance enhancement.

The 82490 cache RAM contains 32 kBytes per chip, and is used in groups of 4, 8, or 16 to implement caches from 128 to 512 kBytes. It supports two-way associativity, delayed writebacks, burst transfers, and boundary scan test. The 82490 contains much more than RAM cells—it includes various buffers, queues, and support for several bus protocols. It is two-ported, with simultaneous access on both the CPU side and Memory-Bus side. The cache optionally supports parity using additional 82490 chips.

Configuration options allow a variety of memory bus widths (32 to 144 bits), cache line widths (16 to 128 bytes), and asynchronous or synchronous transfers. The configuration is selected by the polarity of various pins at reset time.

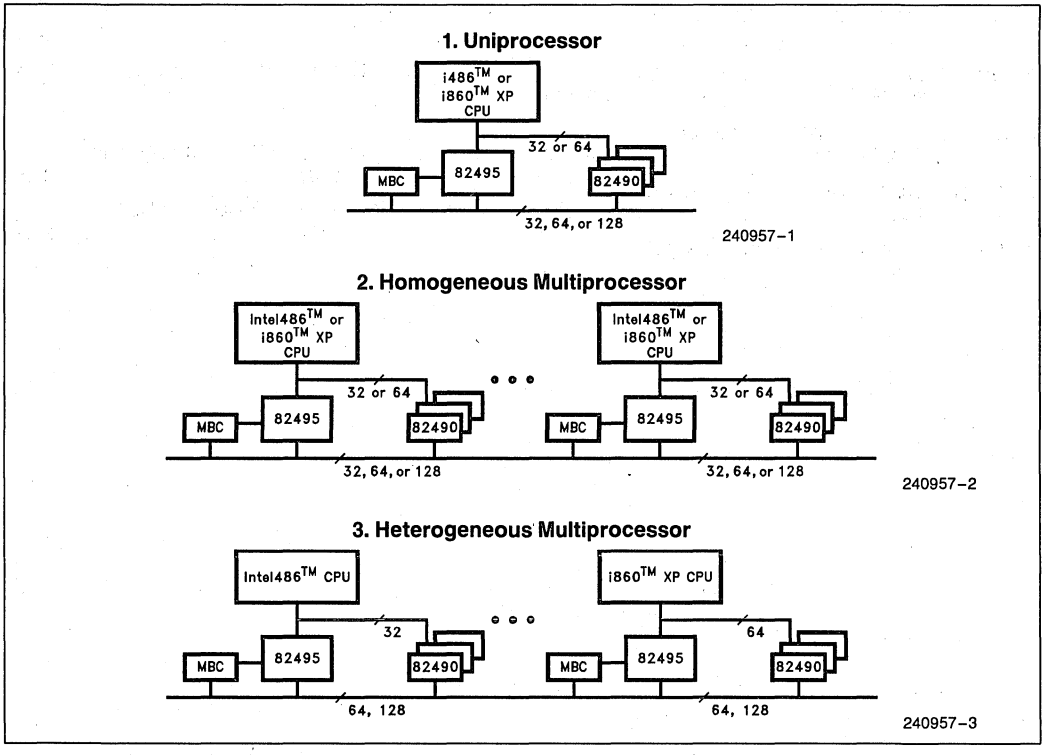


Figure 1. CPU + 82495 + 82490 Systems

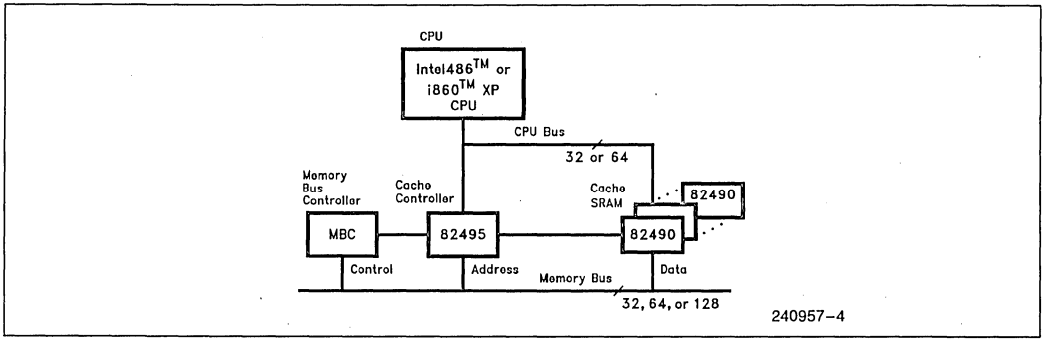


Figure 2. CPU + 82495 + 82490 Core

The Memory Bus Controller (MBC) portion of the system interfaces the 82495 and CPU to the system bus. The MBC converts bus status and command lines into requests to the 82495, for example, to monitor the progress of an ongoing bus transaction from another CPU subsystem to ensure consistency with 82495 + 82490 cache contents. Likewise the MBC adapts 82495 requests to the bus protocol and arbitrates for ownership of the bus. Most CPU requests will not require MBC action; only I/O cycles, cache bypass requests, and 82495 cache misses are forwarded by 82495 to the MBC, while external cache hits are handled totally by 82495 + 82490.

2.0 WHY A CUSTOM BUS INTERFACE?

Clearly the entire interface to a memory bus (abbreviated M-bus) could have been incorporated in the 82495 and 82490 chips. This approach has been followed by some other cache chipsets.

However, such integration suffers from inflexibility and bandwidth limitations. As shown in Figure 3, the performance and cost targets of the system determine the size and complexity of the bus, so if the bus is "hard-wired" into the cache controller chip, it will be too costly for small systems and too slow for larger systems. With the bus interface implemented separately, it can be a complex ASIC for a high-bandwidth complex system, or a few EPLDs for a PC. The same cache controller can improve performance of a variety of bus-based CPUs.

For a desktop PC, a 32-bit simple memory bus is adequate. For a workstation or small multiprocessor of two CPUs, a faster 64-bit bus may be required to give adequate bandwidth for graphics frame buffers and intensive numeric calculations. Bus bandwidth requirements grow as the MIPS rating of each CPU in a system grows; for example, a bus adequate for 12 386 CPUs may be too slow for 6 Intel486 DX CPUs, as they process far more data per second.

A large multiprocessor of 6 or more CPUs needs a wide and fast bus such as Futurebus+, with split-transaction capability to prevent bus bottlenecks from slowing the performance of every processor. Hierarchies of buses and caches can further allow more CPUs with reasonable performance increases as CPUs are added. A Futurebus+ hierarchy maintains concurrent transactions on each bus, and "bridge" caches at the junctions of buses echo them from bus to bus when the bridge detects that one transaction may affect cached copies on the other bus.

Compatibility with existing buses is often crucial in product design, so that new faster components can plug into existing machines and I/O devices. The flexible 82495/82490 bus interface allows compatibility as well as extension.

Thus the 82495 and 82490 will be used in a wide variety of systems, including standard buses like Futurebus+. For proprietary buses, the "proprietor" can design an ASIC or PAL MBC incorporating the required features.

3.0 GUIDELINES

This document exists to clarify the necessary components and tradeoffs of a Memory Bus Controller. **The example designs here have not been tested, and signal definitions of the i860 XP CPU, Intel486 DX CPU, 82495, and 82490 chips are subject to change.**

The memory bus controller is not allowed to use (and thus add capacitance to) any of the CPU pins used by the 82495/82490, except those listed in the 82495 Data Sheet [82495/490DS] description of the BLE# pin. Only the CPU pins BE7–0#, PWT, PCD, LEN, CACHE#, BRDY#, PCYC, and CTYP have sufficient timing margin to tolerate the MBC load.

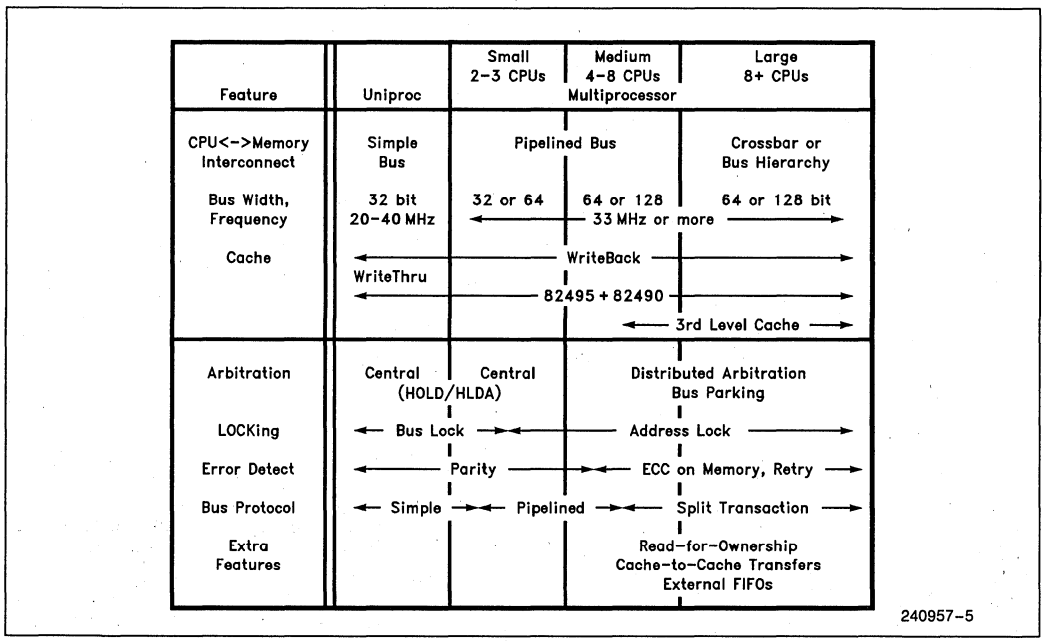


Figure 3. System Type and Bus Requirements

Shared Bus Interconnect

When used in a multiprocessor, the 82495 assumes a shared-memory, shared-bus environment so that it can observe and "snoop" accesses by others which might conflict with the memory locations it has cached. In a crossbar or other multipath interconnect, shared-bus coherency can be emulated for the 82495 or it can be used non-coherently. Either a centralized directory or a hierarchy of buses and caches can do the emulation. A directory would keep a record, for each line of main memory, of caches which have the line. When a cache first writes to a line of memory, the central directory broadcasts an invalidation message to all other caches containing that line. [Agarwal88]

4.0 MBC BLOCK DIAGRAM

Shown in Figure 4 is a high-level block diagram of the functions and interfaces involved in the Memory Bus Controller. Part of the MBC operates on the high-speed clock (CLK) which the CPU and 82495 use. While the M-bus could use the 50 MHz CPU CLK, such a fast M-bus is hard to design. The part of the MBC which interacts with the memory bus protocol runs on an M-bus clock (MCLK), if that protocol is clocked. Also possible is an unclocked M-bus protocol using the 82495/82490 in "strobed" mode. The MBC contains synchronizers and a few signals which cross between the two clock domains. Synchronizers, consisting of specially-designed flip-flops, allow a clocked state machine to use data which may be transitioning near the edge of the clock. Unsynchronized data can cause *metastability* in latches, where their output changes slowly and unpredictably.

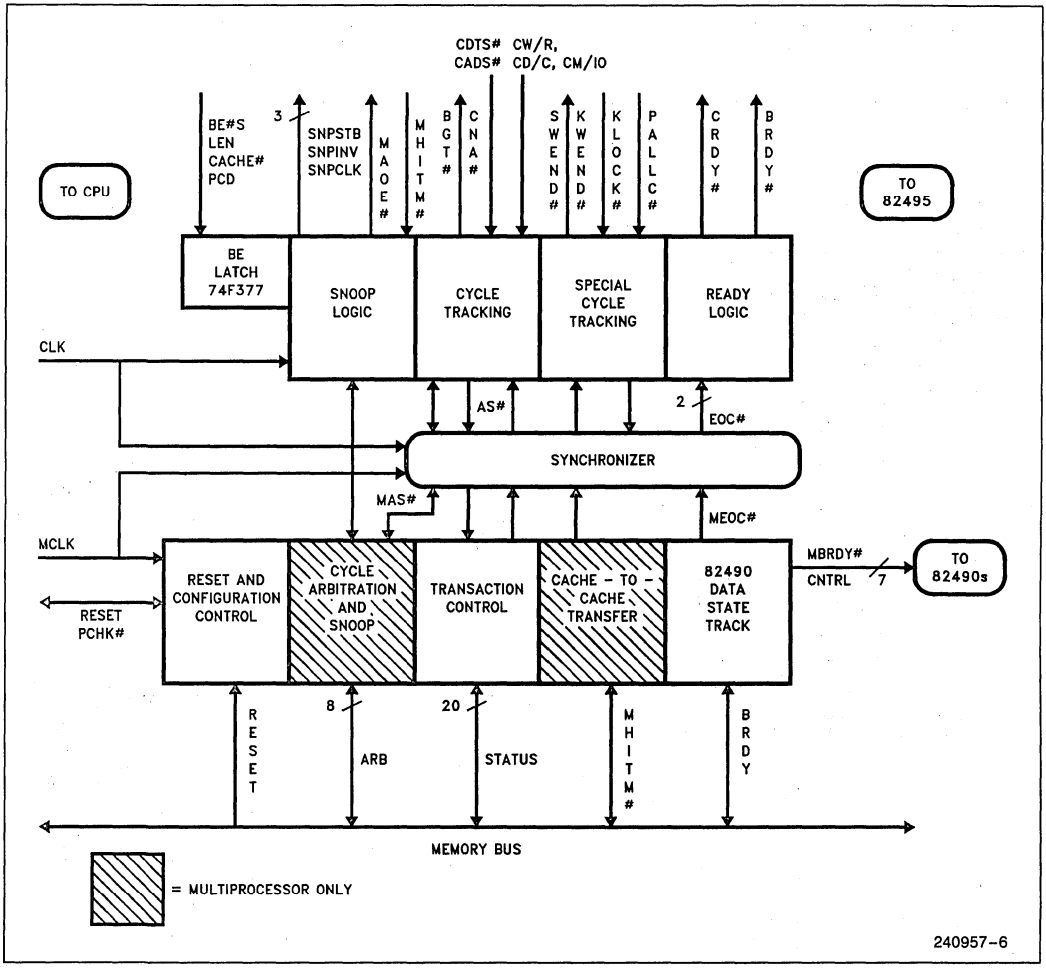


Figure 4. Generic Block Diagram of MBC

5.0 DESIGN EXAMPLE: A UNIPROCESSOR MBC

A simple MBC design example is an adapter to allow plugging a daughtercard module with an Intel486 DX CPU, 82495, and 4 82490s into an Intel486 DX CPU microprocessor PGA socket. The memory bus is an Intel486 DX CPU-bus, allowing the external cache to be a performance enhancing option. It assumes a "divided synchronous" M-bus clock, where the M-bus runs at ½ the CPU CLK speed. Thus no synchronizers are needed. The MBC uses both the CPU CLK and the M-bus MCLK.

This design requires

- 1 74F377 latch
- 6 PLDs containing 10 state machines
- 2 chips for clock generation, not part of the MBC

Approximately 70 signal pins connect the MBC block to the CPU, cache, and memory. Only a uniprocessor is supported, although the bus protocol and MBC could be enhanced for multiprocessing coherency. Figure 5 shows a block diagram. Details of the design can be found in Appendix B.

6.0 DESIGN EXAMPLE: A MULTIPROCESSOR MBC

An i860 XP CPU multiprocessor-capable MBC (Figure 6) using an M-bus similar to the i860 XP CPU bus is proposed. For clocking, it uses an MCLK of 33 MHz, totally asynchronous to the 50 MHz CPU CLK. It could therefore be upgraded to faster CPU CLK rates in the future without changing the design or M-bus.

The design requires:

- 2 74F377 octal latches (for BE7-0#, etc...)
- 2 74AS4374 dual-rank-synchronizer octal registers
- 16 PLDs
- 2 GA1110 clock drivers for clock distribution

These components could be integrated into a single ASIC chip, as about 120 signals connect to the MBC. The MBC can be used for a uniprocessor or multiprocessor i860 XP CPU design. Details can be found in Appendix C.

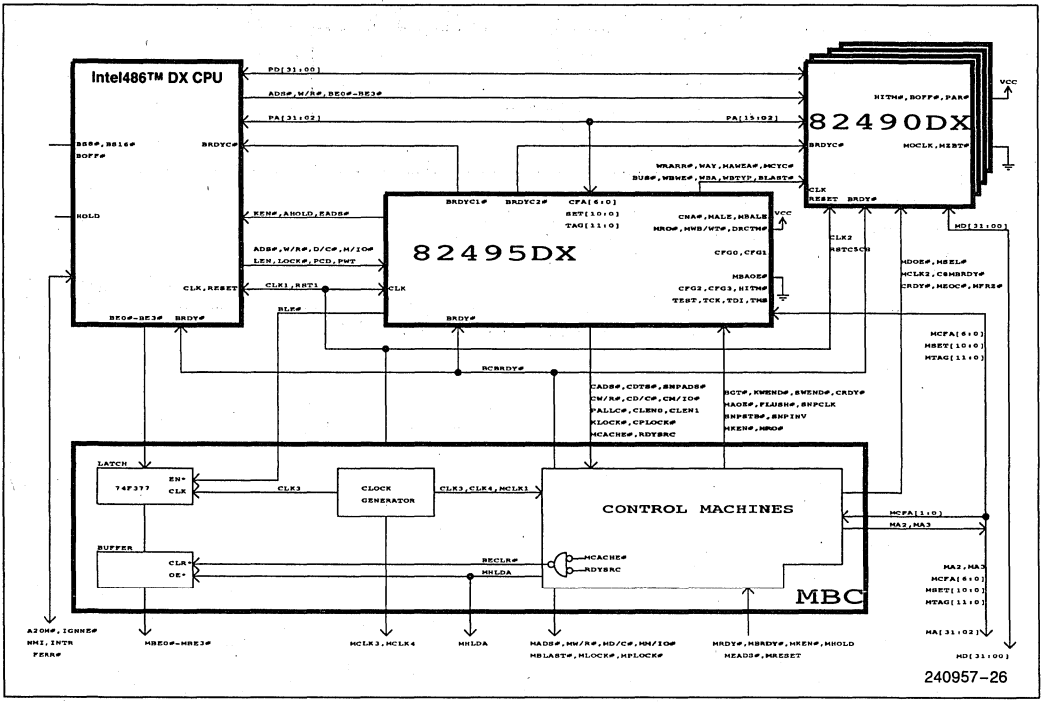


Figure 5. Block Diagram of Uniprocessor MBC

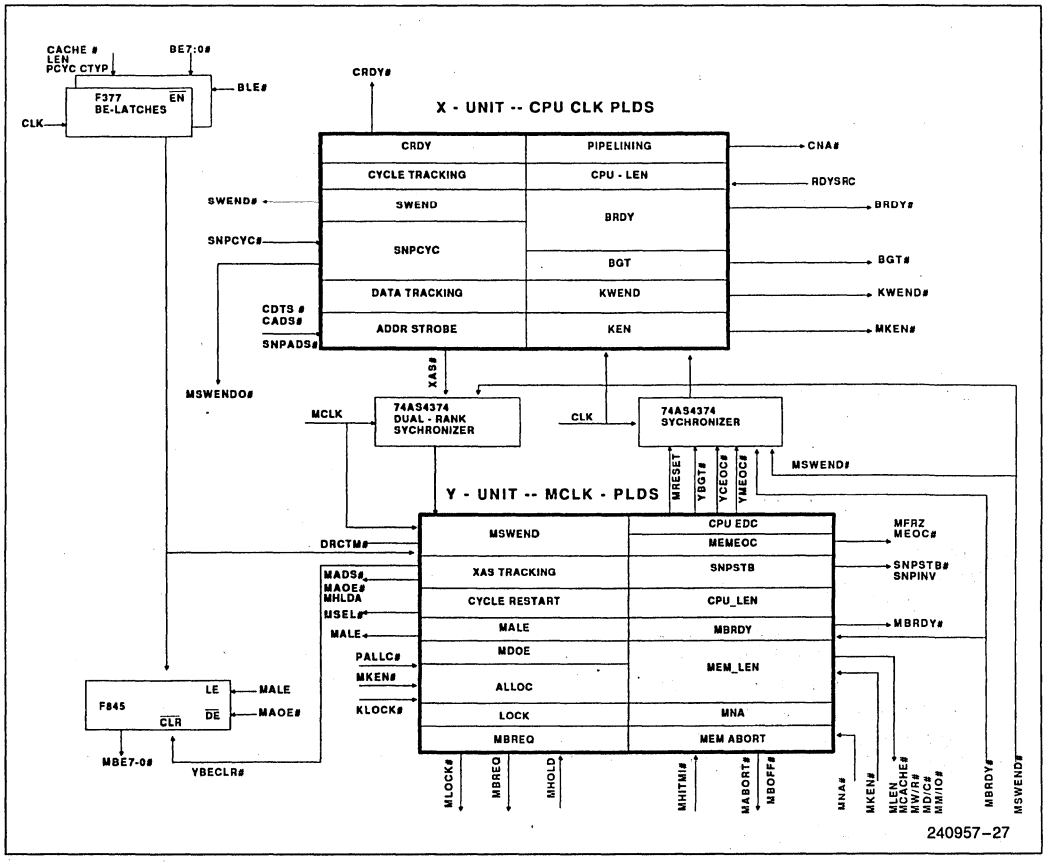


Figure 6. Block Diagram of Multiprocessor MBC

7.0 MBC FUNCTIONS

Table 1 shows the responsibilities of the Memory Bus Controller for uniprocessors and multiprocessors (MP). The multiprocessor features exist mainly to prevent bus over-utilization. However, some of the jobs common to both are more complex in MP for example, arbitration and snooping. The pin lists in the table are not exhaustive.

MBC Functions for Uni and Multiprocessors

Reset and configuration control includes strapping of the following pins to resistors at V_{CC} or Ground, or "temporary strapping" of multifunction pins whose state during the last 16 clocks before falling edge of RESET determines 82495, 82490, or CPU configura-

Table 1. Functions of the Memory Bus Controller

MBC Functions for Uni and Multiprocessors	Pins
1. RESET and Configuration	RESET,HOLD,CAHOLD
2. FLUSH# and SYNC#	CAHOLD,FSIOUT#,FLUSH#,SYNC#
** 3. Bus Error Detection, Retry	PCHK#,BERR
4. CPU transfer tracking (burst count)	CLEN1:0,RDYSRC,BRDY#
5. Mbus transfer tracking (burst count) (including writeback, allocation)	CRDY#,MBRDY#
6. Synchronization between clock domains	BGT#,CADS#,MBRDY#
** 7. Memory-bus pipelining	BGT#,CNA#,MEOC#,CRDY#
** 8. MBC-to-82495 pipelining	CNA#,MALE
9. Memory Bus Arbitration	BGT#
10. Cacheability decode	KWEND#,MKEN#
**11. Redrive bus signals for BTL or ECL levels or heavy capacitive loads	
**12. Packing (convert 32-bit M-bus for 64-bit 82490 size, or 8-bit ROM)	MBRDY#
**13. Bus messages (interrupts, flushes)	INT(R),FLUSH#
**14. Boundary scan and selftest	TCK,TMS,SLFTST#
**15. Performance monitoring (M-bus utilization, read vs. write)	CW/R#,CADS#
16. Snoop handshake (snooping DMA or other CPU)	SNPSTB#,SNPCLK,SNPCYC#
17. Snoop writebacks	MHITM#,SNPADS#

Additional MBC Functions for Multiprocessors	Pins
M1. Snoop window (as master)	SWEND#, MWB/WT#
M2. Backoff 82495 when request was to M-line in another 82495	MAOE#
**M3. Snoop filtering (via SMLN#)	SMLN#
**M4. Cache-to-cache transfers (CTCT)	DRCTM#,MBAOE#
**M5. Read-For-Ownership (RFO)	PALLC#,DRCTM#,MFRZ#
**M6. Split transactions (requires duplicate tag array)	CWAY
**M7. Memory cycle abort (after MHITM#)	MHITM#
M8. LOCK# protection	KLOCK#,CAHOLD,SNPCYC#
**M9. LOCK# de-assertion (for back-to-back Intel486 DX CPU locks)	KLOCK#
**M10. CPLOCK# (Intel486 DX CPU only)	CPLOCK#
**M11. Snoop during LOCK#	KLOCK#
**M12. Multiprocessor Interrupts (for Message-Based Interrupts or TLB shootdown)	INT,NMI(BERR)

** = optional and implementation dependent

tion. The circuit feeding RESET to these chips should keep it active at least 16 CLK periods. "Temporary strapping" means including RESET or ∧RESET in the logic equation for the pin. The multifunction pins are indicated with brackets [ ] below:

i860 XP CPU pins:

PEN#, FLINE#, HOLD

Intel486 DX CPU pins:

RDY#, BOFF#, BS8#, BS16#, HOLD, FLUSH

82495 pins:

CFG3, CFG2[KWEND#], CFG1 [SWEND#], CFG0 [CNA#], CPUTYP[HITM#], FPFLDEN[FPFLD#], NCPFLD#[FLUSH#], SNPMD[SNPCLK], C490LDRV [BGT#], MEMLDRV[SYNC#], SLFTST#[CRDY#], TEST, HIGHZ#[MBALE], CACHE# (NOTE: the FPFLDEN pin is defined for Intel486 DX CPU as PLOCKEN[CPLOCK#]. The 82495XP does not use CFG3 for configuration in i860 XP CPU systems.)

82490 pins:

MTR4/TR8#[MSEL#], MX4/MX8#[MZBT#], MSTBM[MCLK], MEMLDRV[MFRZ#] PAR#, MOCLK, (BOFF#, HITM#)

Intel486 DX CPU: The "unused" Intel486 DX CPU inputs (RDY#, BS8#, BS16#, BOFF#) with 82495 should be connected as described in the Intel486 DX CPU Chipset EDS.

The Intel486 DX CPU FLUSH# input should be tied up, unless the system requires FLUSH messages from the M-bus to be interpreted. Then the MBC must assert the FLUSH# inputs to both Intel486 DX CPU and 82495, because 82495 does not do back-invalidates to the Intel486 DX CPU for FLUSH#. During RESET, the Intel486 DX CPU FLUSH# input must be kept high to avoid putting the CPU in tristate-output-test-mode (Intel486 DX CPU Data Sheet Section 8.4).

i860 XP CPU: The i860 XP CPU input PEN# (Parity trap ENable) must be strapped high unless the memory data bus feeding the 82490s always contains good parity and the i860 XP CPU system uses 2 82490s in parity mode; in the latter case, strap PEN# low. HOLD should be strapped low and FLINE# strapped high, as those features cannot be used with 82495.

82495: The multiplexed 82495 pin FPFLDEN [FPFLD#] becomes an output after RESET, so the PAL or ASIC which creates FPFLDEN must float it as soon as RESET=0. The same multiplexing applies to Intel486 DX CPU mode, where the pin is named PLOCKEN[CPLOCK#]. Likewise, the multiplexed input FLUSH#[NCPFLD#] should be driven high the same clock that RESET falls, to prevent an unnecessary 82495 cache flush. In Intel486 DX CPU systems, the 82495 input CACHE# must be tied low and HITM#[CPUTYP] must be tied LOW, as it signals CPUTYPE to 82495.

82490: The 82490DX inputs HITM# and BOFF# must be tied high in an Intel486 DX CPU system, as they exist to support the i860 XP CPU writeback cache. With an i860 XP CPU, the 82490XP input BOFF# comes from 82495XP but HITM# from i860 XP CPU feeds 82495XP and 82490XP.

The 82490 input MOCLK must also be tied low or to a delayed version of MCLK, if clocked-M-bus mode is used. This is because the 82490 senses the state of MOCLK after RESET ends—if MOCLK stays low, the 82490 uses MCLK to drive MDATA. If MOCLK toggles after RESET, the 82490 will use MOCLK to switch output data. Using a delay-line externally to the 82490 to generate MOCLK from MCLK allows the design a longer hold-time at other receivers of MDATA in the system. For a clocked-M-bus (non- synchronous to CLK), the undelayed MCLK should be connected to the 82495's SNPCLK input and should be toggling during RESET to tell the 82495 to snoop in clocked mode.

During RESET, the 82495 and 82490 will float the bi-directional lines they share with the CPU, such as CDATA and A31:A3. Thus driver contention is avoided. The **RESET input should be synchronous to CLK and deasserted to the 82495, 82490s, and CPU at the same time,** to assure that the configuration controls get properly passed between them.

For Intel486 DX CPU resets, refer to [82495/490DS] for the sequencing of HOLD, HLDA, CAHOLD, and RESET required to reset only the processor without destroying 82495 cache contents. For that purpose, a separate RESET line is advised for the CPU and 82495/82490. The CPU RESET line must be wired to the WRMRST input of 82495, to force 82495 to assert the BRDY1# input to the CPU during a reset of CPU-only (the CPU uses the BRDY1# input during RESET to know of the 82495's existance). The HOLD input of the Intel486 DX CPU and i860 XP CPU processors should be kept low during normal operation with the 82495, because floating the processor outputs may yield undefined 82495 behavior.

FLUSH# (and SYNC#) of caches requested by software must be decoded from the 82495 outputs CM/IO#, CD/C#, and CW/R# (=001) and latched BE3–0# from the CPU. BE3–0# values of 0111 or 1101 should activate the 82495 FLUSH# input, as the Intel486 DX CPU outputs them in response to the INVD and WBINVD instructions, respectively. Synch and flush commands may also come from the bus as a

message in a multiprocessor system. The 82495 is smart enough to allow assertion of FLUSH# or SYNC# at any time, and will delay the beginning of the flushing action until all current CPU and M-bus cycles have completed. The inputs are edge-sensitive. If the bus defines cache flush messages, the MBC may activate the Intel486 DX CPU FLUSH# input as well as the 82495's in response to bus message decodes.

Bus Error or Timeout Detection logic in the MBC can use the CPU's PCHK# output or other M-bus-specific signals to detect errors. Note that the assertion of PCHK# will occur near the time of the error on the M-bus ONLY for non-cacheable reads or 82495-cache-miss reads. For 82495-hits and CPU-idle cycles, PCHK# may arise due to a floating or erroneous CPU data bus value transferred on the M-bus much earlier. PCHK# must be ignored by the MBC except during the CLK after data transfer to the CPU was signalled by the MBC's CPU BRDY#, because PCHK# indicates i860 XP CPU bus parity status at all times, not just during clocks of BRDY# activation. The processor inputs INT, BERR, or NMI can be asserted by the MBC to signal errors. To detect errors originating in the CPU or 82490 upon a write(back), the MBC can check parity on the 82490 MDATA pins or on the M-bus.

If the memory bus includes a retry protocol, the MBC bears the responsibility to implement it, because the 82495 will not retry accesses. For a pipelined MBC interface when the retry occurs after CNA# to the 82495, the MBC must latch the address and other controls (CW/R#, CM/IO#, etc...) from the 82495 to use in retries. Retry should be triggered by signals other than the CPU PCHK# output, because the CPU data transfer cannot be retried although the M-bus transfer can.

The 82490 can restart a burst data transfer (for the case of an error detected after the first MBRDY# but before MEOC# and before CRDY#). To restart the 82490, the MBC must deassert MSEL# for at least 1 MCLK.

While parity is supported by the 82495 and 82490, ECC (Error Correcting Codes) cannot conveniently be used within the cache. ECC can be implemented on the memory system, but no loads are permitted on the CPU-to-82495/82490 interface wires for error checking logic.

Scenarios requiring MBC action are

1) CPU based requests ("Master" mode):
- 82495 cache read miss (and line fill)
- 82495 cache write miss
- Non-cacheable CPU read (including i860 XP CPU pfld)
- Writethrough (to S-state line) or Non-cacheable CPU write
- I/O reads and writes
- LOCKed reads and writes (will be readthrough or writethrough)

2) 82495 based requests ("Master" mode):
- Allocation due to write-miss (line fill)
- Replacement writebacks
- SNPADS# writebacks

3) Requests from other masters ("Slave" mode):
- Snooping of DMA accesses
- Snooping of accesses of other CPUs (in a multiprocessor)
- Bus-specific requests, like interrupt messages, reset requests, cache flushes, configuration registers, ID registers, timeout detection, acknowledgements, TLB shootdown

Transfer Tracking

Tracking of transfers on the M-bus and CPUbus is required of the MBC during all of the above scenarios. This tracking (counting) of transfers involves activating BRDY# the correct number of times for the CPU and MBRDY# (a possibly different number) for the 82495 and 82490. Transactions on the CPUbus which must be MBC-controlled can be 1, 2, or 4 data transfers, decoded from the BLE#-latched CPU pins:

Intel486 DX CPU: BE3–0#, PWT, PCD
i860 XP CPU: BE7-0#, PWT, PCD, LEN, CACHE#

and from the 82495 pins CW/R#, MCACHE#, RDYSRC (and CLEN1:CLEN0 for Intel486 DX CPU mode).

See [82495/490DS] for a complete definition of the encodings. The BRDY# activations must be done only if RDYSRC = 1, and always correspond to the first 1, 2, or 4 MBRDY#s for the 82490-M-bus interface. The number of MBRDY#s always exceeds or is equal to the number of BRDY#s, even for a 128-bit M-bus.

Bursts for line fills and writebacks on the CPUbus always are 4 transfers, but with some 82495 configurations the M-bus is 8 transfers. The addresses are nonsequential when the first access is not at the zeroth word of the line. The addresses corresponding to each BRDY# and MBRDY# follow these rules:

1) CPU burst addresses wrap at CPU line length.
2) When the line address is odd (A2 = 1 for 4-byte bus; A3 = 1 for 8-byte bus; A4 = 1 for 16-byte M-bus), the next address transferred on CDATA and MDATA is the LOWER address (eg., 3 followed by 2). The odd-first-then-even pattern continues for all transfers of the burst. This order optimizes interleaved DRAM systems, and applies to both the M-bus and CPUbus.

3) 82490 bursts on CDATA wrap at CPU line length. 82490 MDATA burst addresses wrap at 82490 line length. For example, a linefill with LR = 4 and a first Intel486 DX CPU address (A5:A2) = E,

- 82490 CDATA ordering is E F C D
- 82490 MDATA ordering is CDEF 89AB 4567 0123 (128-bit M-bus) OR EF CD AB 89 67 45 23 01 (64-bit M-bus)

For LR = 2 (Line Ratio of 82495 to CPU) and CPUbus width = M-bus, below are the burst orders. Each address corresponds to one 4-byte transfer (for Intel486 DX CPUs) or 8-bytes (for i860 XP CPU). Time is increasing left-to-right:

First Address: 0
CPU transfers: 0 1 2 3
M-bus transfers: 0 1 2 3 4 5 6 7

First Address: 1
1 0 3 2
1 0 3 2 5 4 7 6

First Address: 2
CPU transfers: 2 3 0 1
M-bus transfers: 2 3 0 1 6 7 4 5

First Address: 3
3 2 1 0
3 2 1 0 7 6 5 4

First Address: 4
CPU transfers: 4 5 6 7
M-bus transfers: 4 5 6 7 0 1 2 3

First Address: 5
5 4 7 6
5 4 7 6 1 0 3 2

First Address: 6
CPU transfers: 6 7 4 5
M-bus transfers: 6 7 4 5 2 3 0 1

First Address: 7
7 6 5 4
7 6 5 4 3 2 1 0

For LR = 2 and M-bus = 2*CPUbus width (both buses using 4 transfers),

First Address: 0
CPU transfers: 0 1 2 3
M-bus transfers: 01 23 45 67

First Address: 1
1 0 3 2
01 23 45 67

First Address: 2
CPU transfers: 2 3 0 1
M-bus transfers: 23 01 67 45

First Address: 3
3 2 1 0
23 01 67 45

First Address: 4
CPU transfers: 4 5 6 7
M-bus transfers: 45 67 01 23

First Address: 5
5 4 7 6
45 67 01 23

First Address: 6
CPU transfers: 6 7 4 5
M-bus Transfers: 67 45 23 01

First Address: 7
7 6 5 4
67 45 23 01

The remaining transfer orderings for other LR values can be generated similarly, as an exercise for the reader.

For requests originated by the 82495, the MBC must ignore the CPU pins (CACHE#, LEN, PWT, PCD, PCYC, CTYP, and BE7# – BE0#). These requests are writebacks, allocations, or linefills. Also the MBC must prevent the transfer of those signals to the M-bus for 82495 requests—for example, it must force all BE7# – BE0# active during writebacks. The 82495 based requests can be recognized by:

RDYSRC = 0 .AND. MCACHE# = 0 (for writebacks, linefills, allocations)
RDYSRC = 0 .AND. MCACHE# = 0 .AND. MKEN# = 0 (for linefills, allocations)

For posted write requests (RDYSRC = 0 and MCACHE# = 1), the length is 1, 2, or 4 transfers and the MBC must heed the BLE#-latched BE7–0#, LEN, and CACHE#.

Clock Boundaries and Synchronization

To optimize performance, the 82495/82490 allow total/decoupling of the CPU clock at 50 MHz from the M-bus clock. While both the CPU and M-bus could run at 50 MHz, the physical size of the M-bus would be severely constrained. Future faster versions of CPU and 82495/82490 would make a synchronous M-bus even less feasible. However, with a 100% synchronous inter-

face, little time is lost in relaying requests from the 82495 CADS# to the M-bus, and in transfering data from the M-bus to the CPUbus.

Yet with careful design, a slower M-bus such as 33 MHz can handshake with a 50 MHz 82495 with **only a couple of clocks spent on synchronizing**. Furthermore, the transfers requiring synchronizing are fairly rare uncached cycles, cache misses, and snooping. CPU performance is improved further because 82495/82490 always post writes destined for the M-bus, allowing the CPU to continue processing upon write cache-misses and non-cacheable writes.

Most of the 82495 operates on the CPU CLK. Only the snooping control inputs operate on another clock, called SNPCLK (SNPSTB#, SNPINV, SNPNCA). SNPCLK can be the same as the MCLK controlling 82490 MDATA. A SNPCLK can be used with 82495, even if the 82490 is strobed without an MCLK. All 82495 outputs, including snooping results (MHITM#, MTHIT#, SNPCYC#, and SNPBSY#) remain on the CPU CLK.

The 82490 operates half in the CPU CLK domain and half in the M-bus domain. While no *control* signals flow through 82490 between memory and the CPU, 82490 implements a flow-through *data* connection of CDATA to MDATA. Synchronization of the 2 DATA paths is unneeded, as the control signal MBRDY# gets synched by the MBC to the CPU clocked BRDY#. The MBRDY# and BRDY# inputs control multiplexers inside 82490 to choose which part of a line-fill or write is transferred to/from the bus. The MDATA input latches are closed on MCLK (or MISTB for non-clocked operation), and CDATA input latches are closed with CLK.

If MCLK = CLK at 50 MHz, approximately 1.5 CLK periods are required to transfer data through the 82490, including 82490 propogation delay (15 ns) and setup time to both the 82490 (5 ns) and CPU (7 ns for i860 XP CPU "CMOS" levels). The MBC must assure data setup time at the CPU D0–D31 (D63) pins to the rising edge of CLK for the cycle of BRDY# assertion during reads, based on the propogation delay from MDATA to CDATA listed in the 82490 AC timing specs. Writes are not flow-through, as 82490 always buffers the write-data and later 82495 gives CDTS# for the write.

Most of the MBC-to-82490 signals are sampled by 82490 with MCLK, except for BRDY# and CRDY#:

MBC ⟷ 82490 Signals

MCLK	CLK
MBRDY#	BRDY#
MFRZ#	CRDY#
MZBT#	
MDATA	CDATA
MSEL#	
MEOC#	
MDOE# (asynchronous to both clocks)	

The MBC must be partitioned into an MCLK side and a CLK side. Fortunately, the CPU-side of MBC passes only a few signals to the MCLK side, and visa versa. The signals listed below from the dual-i860 XP CPU MBC design in Appendix C must go through a synchronizer. Refer to the Appendix for signal definitions. In the following diagram, a right-arrow (→) identifies synchronizing to CLK, while a left-arrow (←) means synchronizers on MCLK:

	Clock Domain of the Signal:	
MCLK or SNPCLK	**Neither**	**CLK**
MRESET	→	RESET
YBGT#	→	BGT#
YMEOC#	→	CRDY#
YCEOC#	→	BRDY#__maybe
MBRDY#	→	BRDY#__maybe
MSWEND#	← MSWENDA →	SWEND#
MADS#	←	CADS# .or. SNPADS# .or. CDTS#

The signals MKWEND# and MNA# might also need synchronizing to CLK, if they are derived from M-bus responses.

Two TI 74AS4374 "Dual-Rank Synchronizer" chips (Figure 7) are used to transfer critical signals between clock domains, while avoiding metastability. This 20-pin DIP has one clock input and 8 pairs of flip-flops. Thus each of the 8 "Q" outputs reflects the value of its "D" input after 2 clock periods. One chip is clocked by CLK and the other by MCLK. If fewer than 8 signals need synchronizing, chips such as the Signetics 74F50728 or Intel's 85C220 EPLD can combine synchronization with other functions [Ham90].

For an asynchronous or strobed memory bus, M-bus signals (such as MBRDY#) get **delayed by the synchronizer for 2 CLK periods** before the 82495 can see them. For a clocked (but not by CLK) M-bus, **82495 outputs (such as CADS#) get delayed by 2 MCLKs** by the other synchronizer before the M-bus sees them.

The following 82495 signals are defined as "asynchronous", meaning that no external synchronizer is required:

- FLUSH#, SYNC#
- MALE, MBALE
- MAOE#, MBAOE#

Many signals can cross clock boundaries without synchronizing, because they will be ignored until corresponding status signals such as SWEND# and CADS# have been synchronized by the MBC. Thus they will be stable when sampled:

- MWB/WT#, DRCTM#, MTHIT#, MHITM# (sampled when SWEND#)
- RDYSRC, KLOCK#, CPLOCK#, CW/R#, CD/C#, CM/IO#, MCACHE#, BE7:# (sampled when CADS#)

Other signals do not cross clock boundaries, but remain within the MBC CLK logic:

- CNA#, PALLC#, CACHE#, LEN, PCD, PWT, CTYP, PCYC, MFRZ# ...

Synchronizer Delays

To avoid lost time due to synchronizer delays, the following options exist:

1. **Pipeline** the 82495/MBC interface. This hides the delay in synchronizing CADS# to its MCLK counterpart MADS#.
2. Define the M-bus protocol so that **MBRDY# precedes MDATA** by 1 MCLK for reads. Thus the 2 CLK delay in creating BRDY# from MBRDY# is hidden. Likewise define MSWEND# to precede

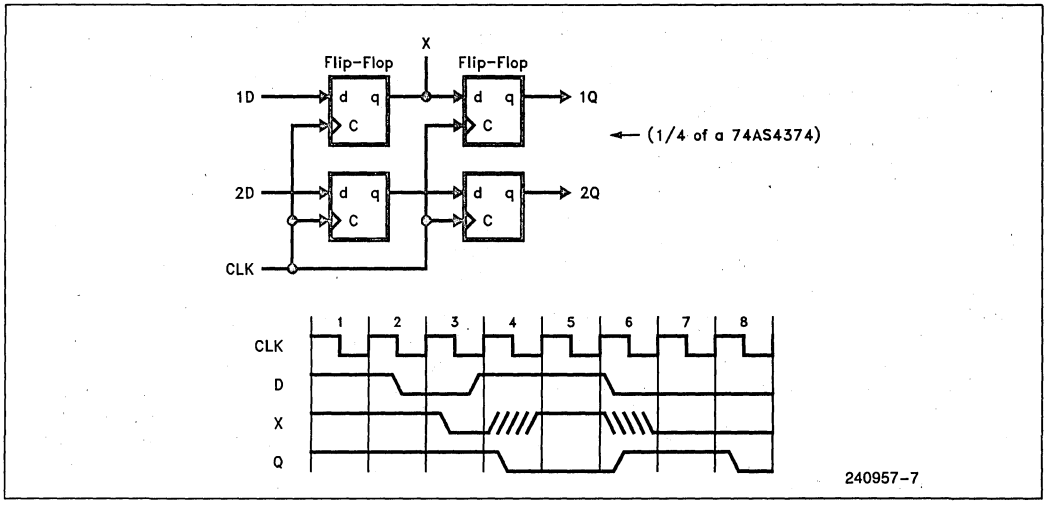


Figure 7. Synchronizer Hardware and Waveforms

MHITM# and MTHIT# by a CLK, by generating MSWEND# from SNPCYC#.

3. Keep the snooping signals (SWEND#, MHITM#, MTHIT#, SNPINV, SNPCYC#) which flow between 82495s on **the same CLK**, so that no synchronizers enter the snoop path. This is feasible only for a small number of physically proximate CPUs.
4. Synchronize the snooping feedback signals from the M-bus (MSWEND#, etc...) only at the destination. They will be asynchronous to MCLK, transitioning with the individual CLK of their source.
5. Avoid MCLK, using a **strobed-only M-bus**. Strobed buses appear in single-CPU systems with an unclocked DRAM interface.
6. **Activate MEOC# to 82490** as soon as possible after the last MBRDY#. MEOC# allows 82490 to begin the next data transfer without waiting for CRDY# synchronization.

BRDY# Generation

Below are recommended sequences of the 82490 and CPU burst-transfer "Readys" for CPU reads, assuming the bus widths are equal. Sequences with more clocks of delay are acceptable but suboptimal.

1) **Synchronous M-bus** (MCLK = CLK): MBRDY# precedes BRDY# by 1 or 2 CLKs, to allow propogation time for data through the 82490 and setup time at the CPU pins.
2) **"Divided Synchronous"** M-bus (e.g., CLK=50 MHz, MCLK=25 MHz, skew controlled): MBRDY# precedes BRDY# by 1 or 2 CLKs. The BRDY# state machine must ignore MBRDY# in the CLK period after it was sampled active.
3) **Other Clocked M-bus** (MCLK < CLK): MBRDY# must go through a dual-rank synchronizer latch (such as the TI 74AS4374) clocked by CLK to produce BRDY#. That means 2 CLK delays between MBRDY# and BRDY#. MBRDY# MUST remain active for at least 1 CLK period to assure that the synchronizer latched it active. To avoid one MBRDY# getting wrongly sampled active twice, the BRDY# state machine should ignore any second MBRDY# in the CLK period after it was sampled active.
4) **Strobed M-bus**: here MISTB# must go through the synchronizer with 2 CLK delays to create BRDY#. An edge-sensitive strobed M-bus avoids the problem of wrongly converting one M-bus transfer to 2 BRDY#s, as a level-change marks each M-bus transfer.

When M-bus width is greater than CPUbus width, the above rule holds only for the first BRDY#. Successive BRDY# activations follow the rules below:

- **M-bus = 2*CPUbus**: 2 BRDY#s occur for each of the first 2 MBRDY#s. The second BRDY# should occur 1 CLK after the first. The third BRDY# cannot begin until after the second MBRDY#.
- **M-bus = 4*CPUbus**: 4 BRDY#s occur for the MBRDY#. The last 3 BRDY#s can occur immediately in the 3 CLKs after the first BRDY#.

For asynchronous systems (MCLK < CLK), high performance design choices are:

M-bus width = 2 * CPUbus width OR
M-bus width = 4 * CPUbus width

The wider M-bus allows each M-bus transfer to satisfy 2 or 4 CPU transfers, so that the CPU is not starved for data during a line fill. The 82490 switches its CDATA outputs to the next value the CLK after BRDY# assertion by the MBC for the current value, so the MBC controls the provision of data to the CPU on linefills.

A low-cost MBC can use M-bus width = CPUbus with a slower MCLK, by converting the first MBRDY# to BRDY# through a synchronizer. The last 3 BRDY#s can be asserted by MBC after completion of all the M-bus transfers. That will allow the CPU to proceed executing after receiving the first datum, which is the one it was waiting for in most cases. Alternatively, the M-bus protocol can be defined so that no idle clocks occur on M-bus after the first MBRDY# and the MBC knows by counting CLKs when to assert successive BRDY#s.

Shown in the following timing diagrams are data transfers on both buses for CPU reads. Although they assume no dead clocks (wait states) during the M-bus burst, dead clocks are allowable.

Writes are not shown in the diagrams because the MBC never supplies the CPU BRDY#s for burst writes. RDYSRC=0 for most writes, and the 82495 controls the CPUbus transfers. The exception to this rule is I/O writes, which 82495 does not post; for I/O writes, the MBC supplies BRDY# to the CPU, but I/O accesses are always 1 non-bursting transfer.

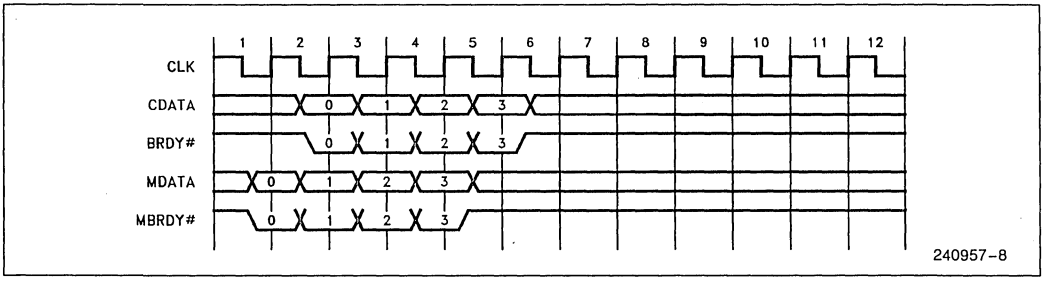


Figure 8. Data Transfers, M-bus Width = CPUbus Width. MCLK = CLK

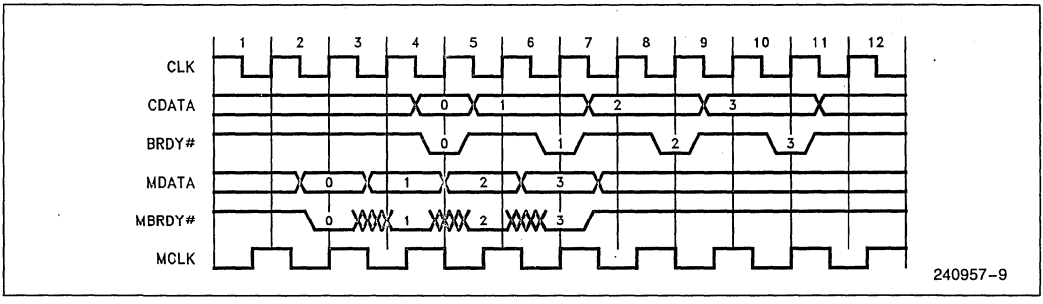


Figure 9. Data Transfers, M-bus Width = CPUbus Width. CLK/2 < MCLK < CLK. Note the starvation on the CPUbus (extra wait state)

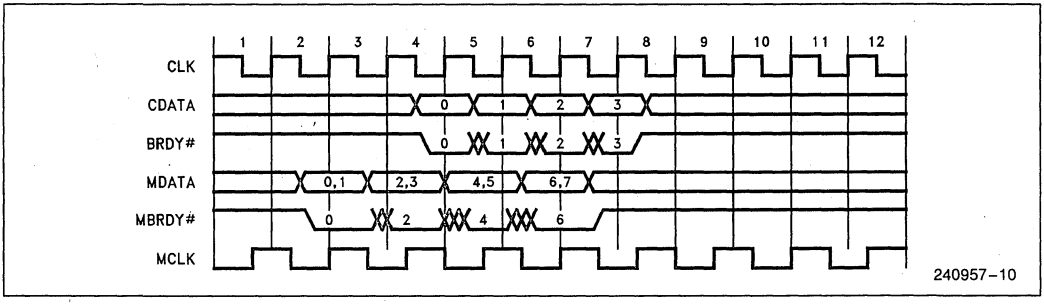


Figure 10. Data Transfers, M-bus Width = 2*CPUbus. CLK/2 < MCLK < CLK

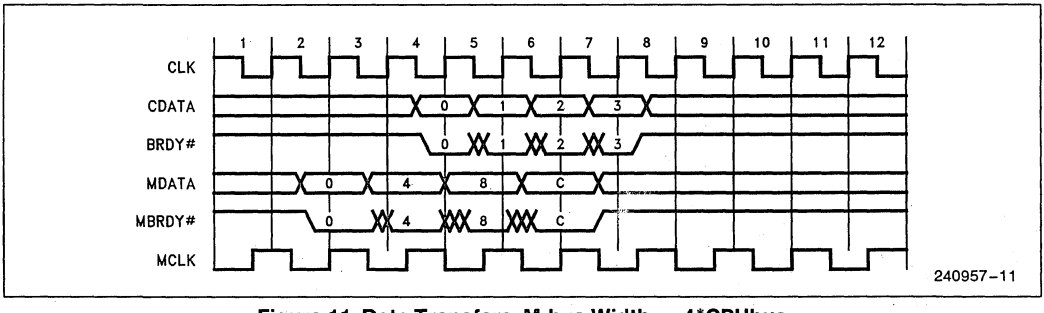


Figure 11. Data Transfers, M-bus Width = 4*CPUbus

Pipelining

Pipelining the MBC-to-82495 interface reduces latency by allowing the MBC to arbitrate for the next M-bus transaction while the first is proceeding. If the M-bus is also pipelined, it allows the snoop for the next to begin during the data transfer for the first.

Signals used in pipelining the 82495 are CNA#, BGT#, MALE, KWEND#, SWEND#, and CDTS#. The 82495 will not listen to CNA# until the clock of BGT# activation. Also, KWEND# activation sometimes allows the 82495 to create a next cycle, such as an allocation after a write miss. MALE deassertion allows the memory address to remain at the value for a previous request, even though the next request CADS# and other control signals have already occurred in response to CNA#. The MBC must latch the 82495 output signals which change in response to CNA#, until their status no longer matters to ongoing cycles.

Note that 82495 and 82490 automatically pipeline the CPUbus interface to i860 XP CPU by activating NA# and latching address and data.

Pipelining the M-bus itself involves sending a next address for snooping and DRAM access while data transfer from the current address still remains incomplete. This increases bandwidth by overlapping slow DRAM access with bus data and address transfers, as in the i860 XP CPU pipelined bus.

While each 82495 allows only a one-stage deep pipeline, the M-bus can have a deeper pipe as requests from several different 82495s can be in progress. The number of stages in the M-bus pipe should match memory access latency. For example, use 3 stages for a 240 ns memory with a 120 ns bus MADS#-to-MNA# (and SWEND#) time, so that a second and third request get issued during the memory latency of the first. Pipelining does not imply that multiple snoops are ongoing waiting for SWEND#; that is a *split-transaction* bus, defined in a later section. Thus a quick SWEND# turnaround time speeds a new request onto the M-bus.

The advantage of a pipelined bus using a 4-transfer burst is illustrated in Figures 12 and 13. Assumed is a fast memory access time of 4 MCLKs. With a slower access time, pipelining becomes more important for maintaining data bus bandwidth; even with the 4-MCLK access, the unpipelined data bus is idle 50% of the time.

M-bus Arbitration

If the M-bus possesses more than one master, each MBC must arbitrate to gain control of the M-bus when-

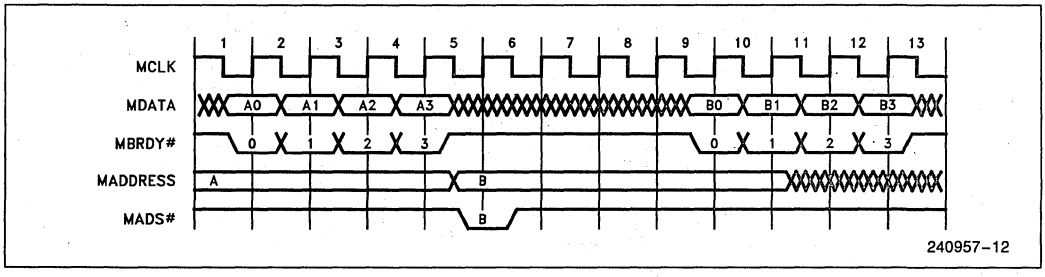


Figure 12. Data Transfers for Non-Pipelined M-bus. Note low MDATA Bandwidth.

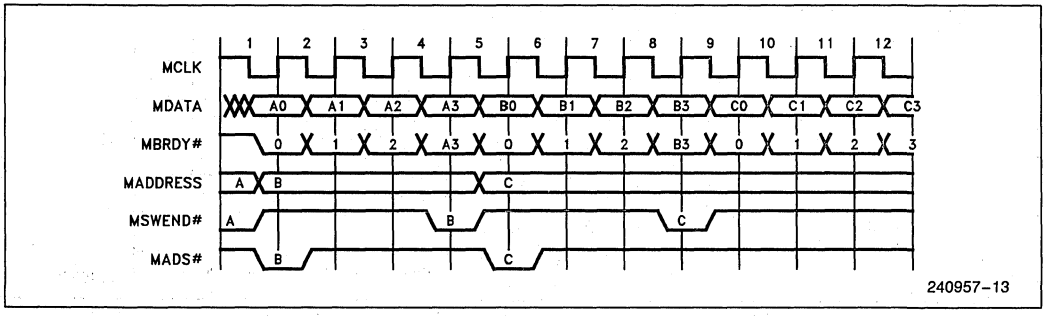


Figure 13. Data Transfers for Pipelined M-bus

ever its 82495 activates CADS#. No arbitration logic is included in 82495 nor 82490, except for the ability to float (Hi-Impedance) the 82495 and 82490 M-bus outputs via the MAOE# and MDOE# signals. The BGT# and MAOE# inputs to 82495 are from MBC arbitration logic. The simplest systems can use a HOLD/HLDA/BREQ protocol like the i860 XP CPU and Intel486 DX CPUs themselves, which is centralized arbitration.

Expandible buses like Futurebus+ and Multibus-II use distributed arbitration to allow a variable number of masters. Bus parking (retaining ownership of the M-bus until another master requests it) is advised to avoid unnecessary delay.

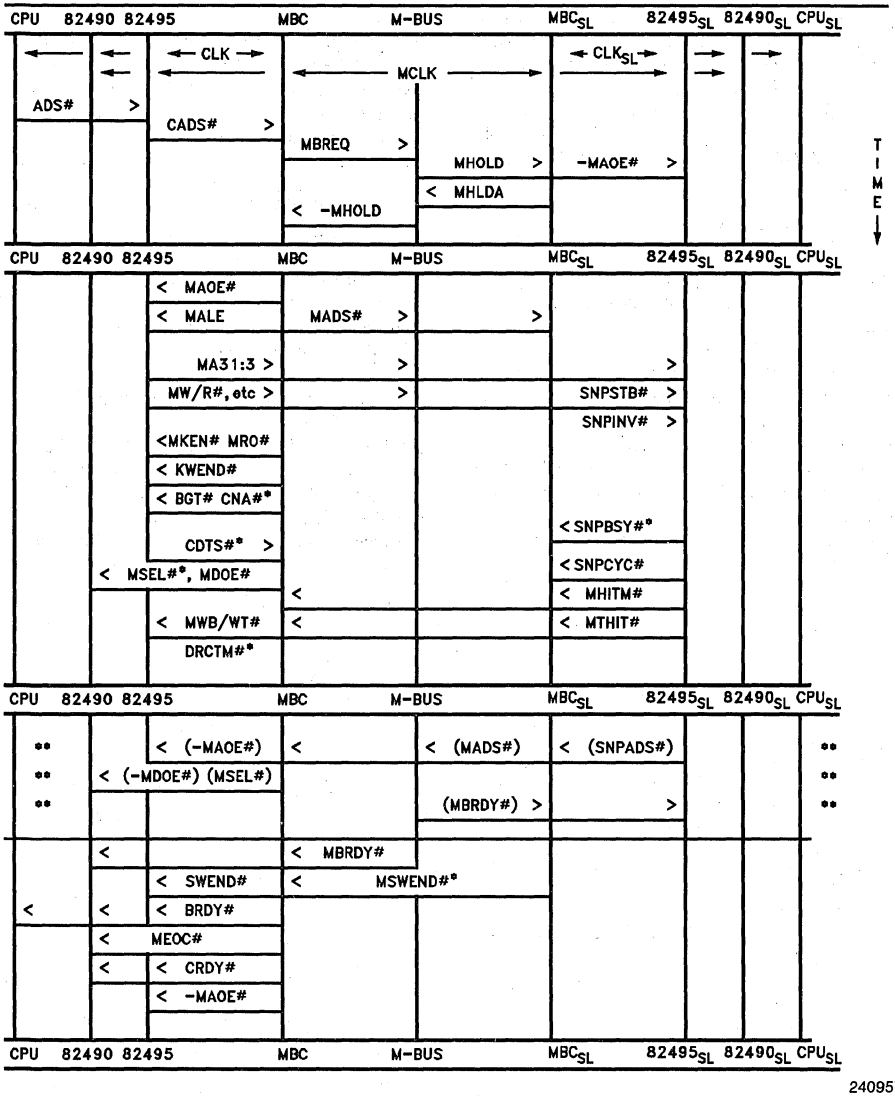
The "restricted backoff protocol" of 82495 requires that it be granted the bus for a modified-line writeback after it activates MHITM#, before it will snoop or initiate any other transactions. The snooping MBC must relinquish the M-bus immediately after the CRDY# of the M-line writeback so that the original owner can complete its work.

Sequencing

A typical sequence of request and response signals between the 82495 and MBC is shown in Figure 14. The "SL" entities (CPU_{SL}, 82495_{SL}, 82490_{SL}, MBC_{SL}) are for another CPU/Cache core, the SLave(s) who snoop when the master CPU owns the bus. No DMA (such as EISA or MCA) interaction is shown, but it will be similar to the CPU responses, except that no writeback will be done by DMA. Time increases downward. A minus-sign prefix means deassertion.

The arbitration for the M-bus shown in the diagram assumes a HOLD/HLDA protocol like the CPUs use. That is a primitive centralized scheme, suitable only for a small number of processors.

The sequencing may vary from that shown; for example, MSEL# may precede CDTS#. MADS#, MW/R#, MA31:3, MM/IO#, MD/C#, and MBE7-0# would all be valid simultaneously. The signals in parentheses would be asserted only in the case of a M-line hit in the snooper, and some signals for that writeback and possible cache-to-cache transfer are not shown.

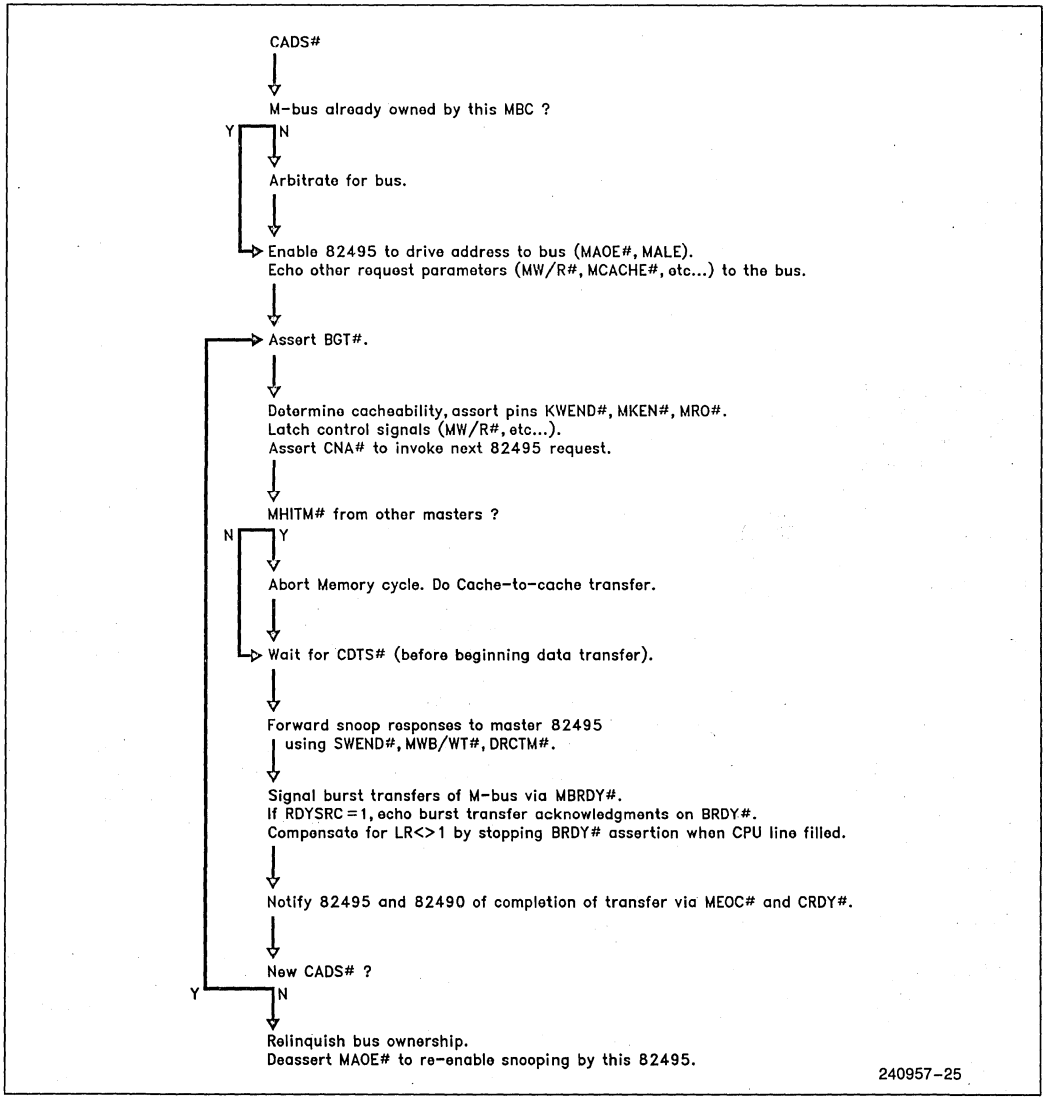


240957-14

* = Signal might occur sooner or not at all, depending on the type of request and bus protocol.
** = *These lines of the sequence occur only on a Hit-to-Modified (MHITM#)*

Figure 14. MBC Signals and Protocol Layers

Flowchart of MBC Algorithm (not applicable to all cases)



Cacheability of each request must be determined by the MBC to prevent the 82495 and CPU from caching things like memory-mapped I/O device registers. The i860 XP CPU CPU samples its KEN# (Cache ENable) pin at the time of the first BRDY# for a transfer or at NA#, whichever comes first. The 82495 offers more flexibility than the CPU cacheability indicators, by using the KWEND# (cacheability Windown END) input to indicate validity of the MKEN# and MRO# pins. The values of MKEN# and MRO# are based on address decode, either locally in the MBC or from a centralized decoder on the memory bus. For best performance, KWEND# should come as soon as possible, as it allows 82495 to decide what the next CADS# should be—for example, to begin an allocation for a write miss, or to start another writethrough.

A typical implementation would activate KWEND# 2 clocks after CADS#, using a PLD or fast SRAM to decode the upper bits of the address to generate MKEN# and MRO#.

Note that KWEND#, SWEND#, and BGT# need not be asserted by the MBC for SNPADS# cycles (snoop writebacks), but it may be simpler to assert them always.

Snooping

Snoop handshaking (bus watching) is useful in a multiprocessor system, and may be needed in a uniprocessor system where the 82495 and CPU caches must be kept consistent with DMA accesses. The 82495 must snoop all DMA accesses to memory. The MBC sees requests from DMA (or other processors) on M-bus and converts them to SNPSTB# activations to the 82495. The following scenarios are possible:

- DMA (or other processor) read causes 82495 MHITM#: 82495/82490 must **writeback** the modified line to memory before the first DMA data transfer occurs (unless the DMA controller is capable of re-trying the read. If the DMA can retry, then the 82495 writeback must cause the initial DMA access to be aborted.) The MBC can assert SNPNCA (SNooP Non-CAcheable Access) to the 82495 for a DMA read, so that the 82495 knows it can keep the block Exclusive upon a hit.
- DMA (or other) read causes 82495 MTHIT# but not MHITM#: MBC must assert the "shared" status line of the M-bus, if the bus includes such a line.
- DMA (or other) write causes 82495 MHITM#: 82495/82490 must **writeback** the modified line to memory before the first DMA data transfer occurs. SNPINV should be activated to 82495 to **invalidate** the line.
- DMA (or other) write causes 82495 MTHIT# but not MHITM#: SNPINV should be activated to 82495 to invalidate the line. Note that 82495/82490 cannot "write snarf"—they do not absorb write-data from the memory bus and merge it with current cached contents of the line. However, they can absorb a full-line writeback from the M-bus when doing a linefill of the same address (see the section on Cache-to-Cache-Transfers).

Bus size adaptation can be done by the MBC, although it is not necessary in most systems. In an Intel486 DX CPU or i860 XP CPU system without an 82495/82490, an 8-bit device like a ROM can be used to contain code, and the CPU will automatically fetch at byte-width when the BS8# (Intel486 DX CPU) or CS8 (i860 XP CPU) pin is asserted. However, if a byte-wide ROM is used with an 82495/82490, adaptation of this byte interface is required from the MBC.

If the ROM code is to be cacheable, the MBC must convert the 82495 line fetches at the ROM location to the appropriate number of byte-wide ROM reads. Latching transceivers must be employed at the 82490 MDATA inputs or at the ROM output, to assemble the single-byte ROM reads into 4 (or 8) bus-width-wide transfers to the 82490s.

If the particular M-bus protocol requires transfer widths shorter than the 82490 data width used, the address range requiring such transfers can be made non-cacheable to force 82495 and 82490 to use the width given in the request from the CPU.

Bus size adaptation would also be needed to support a 512kB cache on a 32-bit memory bus. In that case, the MBC must control transceivers and MBRDY#s to interface between the 64-bit 82490 MDATA path and the 32-bit M-bus.

Bus Signal Levels

Redriving 82495/82490 signals to the M-bus (such as MDATA, addresses, and 82495 control outputs) can optionally be done by the MBC. If the M-bus signal levels are not TTL, like ECL or Futurebus+ BTL (Backplane Transceiver Level), then appropriate transceivers must lie between the M-bus and 82495/82490. Also M-buses with heavy capacitive loads should be redriven by transceivers, although 82495 and 82490 can tolerate loads of up to 100 pF.

An additional advantage of buffering the 82495/82490 signals with transceivers in a multiprocessor is that a "local M-bus" will exist between the chips and the main system M-bus. That allows some local traffic from the CPU module to attached peripherals to avoid traversing the M-bus. Such peripherals might include an MPIC/CCU (MultiProcessor Interrupt Controller/ Concurrency Control Unit), a JTAG boundary-scan controller, or a time-of-day clock, as in the Sequent Symmetry multiprocessor.

8.0 MBC FUNCTIONS FOR MULTIPROCESSORS

Multiprocessor cache designs have additional motivations beyond the uniprocessor goal of reducing memory access latency. Reducing memory bus usage is especially important because the sharing of the bus creates a bottleneck. Thus multi-82495 systems need to minimize the number of transactions and make each one as short as possible. Large caches (256k or 512k) are recommended for multis, to keep the miss rate as low as possible.

In addition to the uniprocessor functions, an MBC in a multiprocessor must handle consistency with caching agents other than its own 82495. The multiprocessor MBC may also for performance reasons implement snoop filtering, cache-to-cache transfers, read-for-ownership, and split transactions.

Snooping results from listeners (slaves) on the bus must be fed back to the master 82495 by the time SWEND# is activated, if the system uses writeback policy (write-through requires no feedback). These results (DRCTM#, MWB/WT#) are translations of the slaves' MHITM# and MTHIT# outputs. As shown in Figure 15, typically all MHITM# outputs would be wired-or via open-collector transceivers. Because slaves on the bus may be busy with CPU operations and back-invalidations, the snoop delay can vary. Thus a latched derivative of the SNPCYC# output of all 82495s would be wired-or to derive SWEND#. Alternatively, the MBC can count CLKs to generate SWEND#, using the worst-case upper-bound of CLKs required for all 82495s to snoop, but that makes all snoop windows long.

Because 82495 will tolerate SWEND# arrival up until CRDY#, the M-bus data transfer for reads can overlap the snooping delay. The transfers (MBRDY#s) can occur during snoop latency, and an MHITM# activation would cause the MBC to restart the transfer using 82490's MSEL# pin.

If a 82495 linefill or writethrough hits a dirty line in another cache, the MBC cannot BACKOFF the 82495.

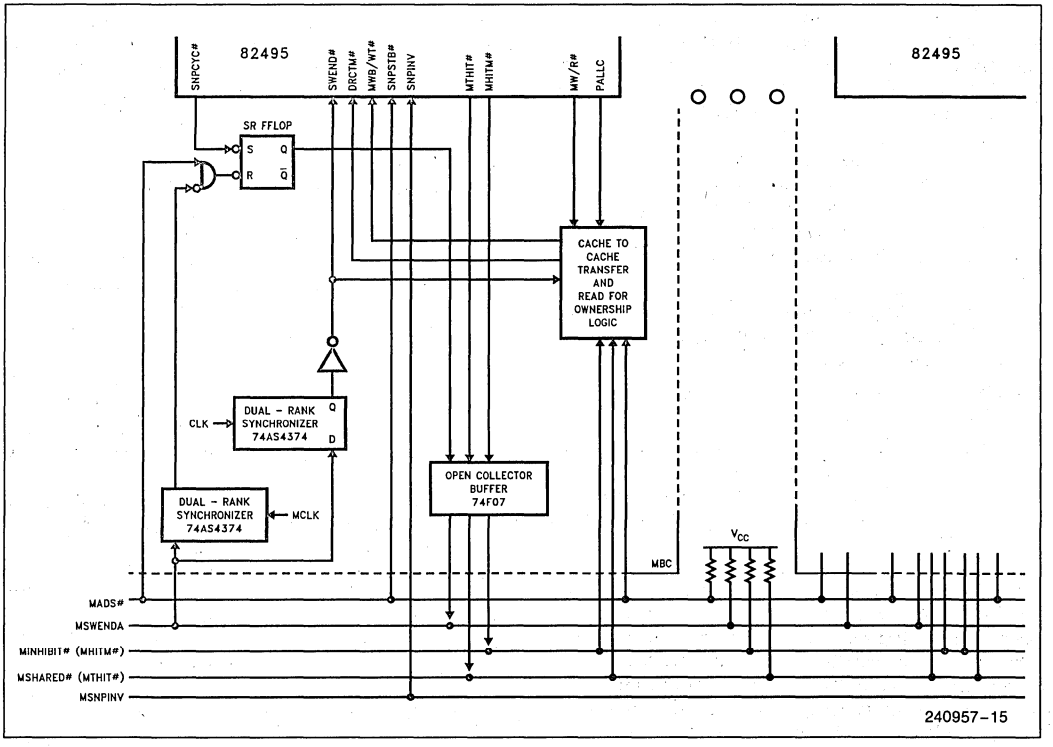


Figure 15. Creating Snoop Results from MHITM#, MTHIT#, and SNPCYC#

Labeling that other cache "the dirty 82495," and the initiating 82495 "the master 82495". The master MBC must force a retry of the access after the dirty 82495 dumps the line, but the master 82495 has no "Backoff and Retry" input pin. Rather, on a linefill the master 82495 must see the data transfer as if it had come from memory. On a write, the master 82495/82490 data write must wait until the modified line from the dirty 82495 has been dumped to memory. To do so, the master MBC can either:

1) Delay the corresponding MBRDY#s to the master 82490 until the modified line is completely written into memory and read out of memory. That implies the master MBC will remake the initial request to the memory controller after the writeback.

   OR

2) Create a cache-to-cache transfer, so that the write-back data movements go directly into the master 82490 over the M-bus. A later section describes cache-to-cache transfers. Such transfers are quicker than waiting for the entire modified line to be written back to memory.

Note that the 82490 can restart the data transfer for reads or writes, in the case of MHITM# activation after the first MBRDY# but before MEOC# and before CRDY#. To restart the 82490, the MBC must deassert MSEL# for at least 1 MCLK.

Snoop Window Time (the delay from MADS# to SWEND#) limits address-bus bandwidth. In the interval from the address on M-bus until the acknowledgement (SNPCYC#) by all listeners, no more requests (addresses) can be on the bus. This restriction is implied by:

1) A typical M-bus has only one MSWEND# wire, which cannot be identified with the proper request if several requsts are outstanding.
2) 82495 does not snoop between BGT# and SWEND#.
3) 82495's "restricted backoff protocol". That protocol requires the M-line writeback to be the first transaction by any 82495 which generates MHITM#, and 82495 cannot snoop anymore until it finishes the MHITM# writeback.

Data for read-misses cannot be transferred on the CPUbus until SWEND#, because the MBC cannot abort a CPU transfer after giving the first BRDY#. Thus the snoop window length influences CPU performance. Depending on the number of processors, bus speed, and memory speed, two scenarios arise from snoop window length versus memory access latency:

1) SWindow < Memory Latency: SWEND# precedes the MBRDY#s. If MHITM# occurs, the original memory access can be aborted and its MBRDY#s must be ignored.
2) SWindow > Memory Latency: data transfer on M-bus can proceed, with MBRDY#s causing 82490 linefill buffers to advance. After SWEND#, the MBC can begin BRDY#s to the CPU and 82490 if MHITM# is inactive. If MHITM# is active, the MBC must restart the M-bus data transfers after (or during) the writeback from the modified snooper, and can begin BRDY#s immediately after the first MBRDY#.

The typical snoop window in a multiprocessor using the hardware of Figure 15 is about 7 CLKs total snoop turnaround delay, shown in Figure 16:

1	CLK for propagation delay of master's MADS# (to slave 82495s' SNPSTB# inputs)
+ 0.5 to 1	CLK for 82495 to internally latch SNPSTB# and synchronize it to CLK.
+ 1	CLK for 82495 tag lookup and SNPCYC# (or more, if 82495 is busy with SNPBSY#)
+ 1	CLK to latch SNPCYC# into the MBC Set/Reset flip-flop generating MSWENDA.
+ 1	CLK for MSWENDA open-collector buffer and settling time from all slaves.
+ 2	CLKs for MSWENDA to get through synchronizer (on the master MBC's CLK) and inverter to generate SWEND# to the master 82495.

The window total assumes that the slave 82495s' one CLK delay from SNPCYC# until MHITM# is concurrent with the synchronizer delay for creating SWEND# from MSWENDA at the master. Those 2 CLKs can overlap with the next MADS# if it is asynchronously generated from MSWENDA. Shorter snoop window times can be obtained using duplicate external tags as explained later, but this is not trivial.

Read for Ownership (RFO) protocols decrease bus traffic by avoiding the M-bus write which would occur upon a write-miss. That is, a write-miss would go to the bus, followed by a 82495 line allocation request for the missed area. With RFO, the MBC does not echo the 82495 write request to the M-bus. Instead, it asserts MFRZ# to freeze the written data in the 82490 memory buffer, and allows the subsequent 82495 allocate line request to go to the bus. When the line data returns on the M-bus, MBC asserts DRCTM# to cause the 82495 to mark the line as Modified (the memory system and other caching agents do not know of the original write miss, so they have invalid copies of the line).

Signals which the MBC must use to do RFO are:

1) PALLC# (Potential ALLoCate): from the 82495 must be active on the write miss.If not, RFO cannot be performed.
2) MKEN# and CRDY#: must be activated by the MBC for the write, to trigger the 82495's subsequent allocation request

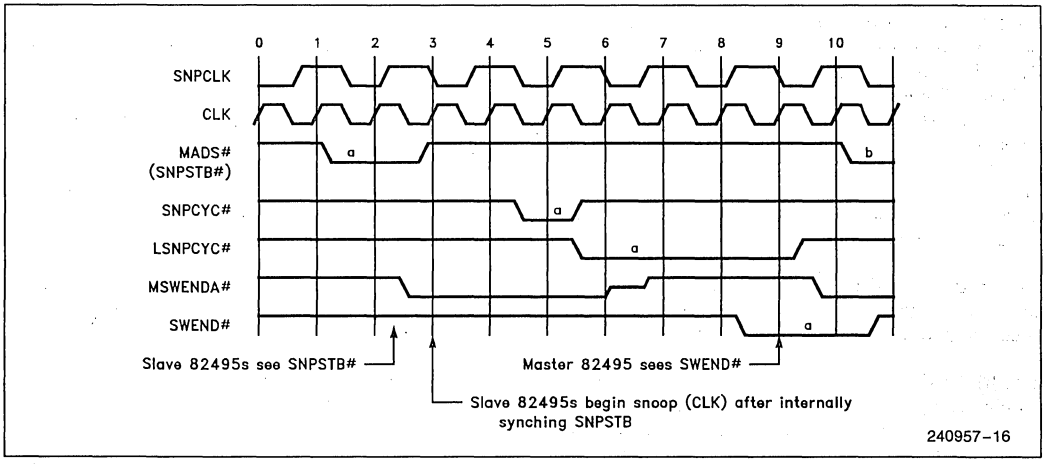


Figure 16. Snoop Waveforms

3) MFRZ#: must be activated by the MBC to the 82490 at the time of the MEOC# and CRDY# for the write.

4) INVAL (memory bus Invalidate indication): must be asserted by the MBC during the allocate-read to force all other 82495s to invalidate their now-obsolete copies of the line. Slave MBCs will assert SNPINV to 82495s.

5) DRCTM# (DiReCt To Modified): must be asserted by the MBC during the SWEND# of the allocate, to make the 82495 put the line in M-state.

6) MWB/WT#: must be asserted during the SWEND# of the allocate.

7) CPLOCK# (82495 Psuedo Lock in Intel486 DX CPU systems): if active, the MBC must NOT do RFO, because 82495 will activate PALLC# only on the second of the 2 writes. If the MBC tried to RFO, it would merge only half of the data into the modified line.

See [82495/490DS] for RFO information.

Cache-to-cache transfers (CTCT) optimize the speed of consistency actions in a multiprocessor. For a read linefill by a master causing an MHITM# from a slave, the writeback data movements go directly into the master 82490 over the M-bus from the dirty 82490. For a write, Read-for-Ownership (RFO) is required for the CTCT. If RFO is not implemented, then the cache-to-cache option can be used only on linefill (read) misses. In fact, RFO makes every write-miss into a linefill. The 82495/82490 do CTCT only on entire lines, not bytes or words.

For CTCT on a linefill causing MHITM#, the MBC doing the writeback must initiate the writeback at the subline address of the initial read. Starting the writeback from the first word of the line is NOT acceptable.

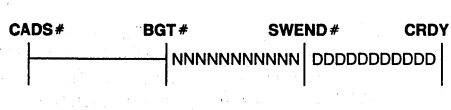
While CTCT is faster than re-reading the line after waiting for the dirty writeback, the latency will be longer in most systems than for fetching lines from main memory. CTCT would actually waste time for such items as shared instruction pages. For non-written data, transferring from memory to a CPU is probably faster than tranferring from another cache. So 82495 supports only M-line CTCT (no writeback occurs unless MHITM#).

Signals involved in CTCT are DRCTM#, MZBT#, MHITM#, MBAOE#, and MSEL#. See [82495/490DS] for CTCT information.

Snoop filtering can be implemented by the MBC using the 82495 SMLN# (SaMe LiNe) output to reduce the latency for snooping. That is, SWEND# can be asserted immediately to the requesting 82495, if the 82495 asserts SMLN# to indicate the current request is to the same line as the previous request. In that case, other caches already have checked this line. SMLN# must be ignored if the M-bus has been used by other agents between the 2 82495 requests. The M-bus protocol need not include a "non-snooped transfer type" for the use of this feature, as the MBC can simply ignore the snoop responses from other MBC/82495 modules.

Split transaction (ST) memory-buses such as Futurebus+ prove valuable in high performance systems. An ST (also called "connect/disconnect" or "packet switching") bus divides a single read request into a separate address-transfer phase and a data-transfer phase. Thus the bus is not monopolized during the long latency involved in accessing data across bus hierarchies. Writes typically are not split, as the data and address are available simultaneously from the writer. In a hierarchical bus, requests must be forwarded across bridges for the purposes of snooping and memory access at remote nodes, and the snoop latency may be long. Thus the bus should be freed between initial request and snoop-response for use in other transactions.

The 82495 does not support ST directly. That would require snooping current cache contents and queue-up possible writebacks, for the accesses from other bus agents between the time of the BGT# (the address phase) and SWEND# (end of the address phase or later). Also 82495 cannot writeback dirty data between SWEND# and CRDY# (end of the data phase) of an ongoing cycle; it cannot suspend a transfer for later resumption after a snoop writeback.



NN = No snooping by 82495 will occur in this area

DD = Delayed response by 82495 to snoop requests here. MTHIT# and MHITM# asserted immediately, but writeback of dirty data delayed until after CRDY# for ongoing cycle.

82495's inability to snoop during the NN period comes from the need to keep 2 addresses into the tags active—one for the outstanding 82495 request, whose tag must be updated at SWEND# based on MWB/WT# and DRCTM#, and one for the snoop inquiry. Furthermore, any MHITM# on the M-bus could not be easily linked to the request causing the snoop if 2 snoops are outstanding.

To support split transactions by snooping between BGT# and SWEND#, a set of **tags external to the 82495** can be implemented in the MBC. Those tags would replicate the contents of the 82495 internal tags, listening to all memory bus requests and responding with snoop results. Only when a 82495 state change (to I or S) is needed will the 82495 be informed of snooping action—only then will the external tags relay the snoop request to it.

Duplicate tags provide quicker snoop turnaround because no SNPCLK-to-CLK synchronization is required; the duplicate tags are in the SNPCLK/MCLK logic. While they are a high-performance option, they are costly and complex.

Memory cycle abort is required in multiprocessors when a snooping 82495 activates MHITM# to signal that the memory's copy of the data requested by another 82495 is obsolete. As explained above, the memory read or write must be INHIBITED until the writeback is done. Depending on implementation, the original access may need to be retried or abandoned. If CTCT and RFO are implemented, then abandonment is probably adequate. Although the complexity of aborting could be avoided by delaying all memory action until SWEND#, that would decrease performance. An M-bus signal such as "SIV" (System InterVene) or "MBOFF#" (M-bus Back OFF) allows the MBC of the snooper to tell memory to abort.

If the M-bus is pipelined, there may be constraints on when the MBC can assert the "abort" signal to avoid cancelling the access in progress for the transfer preceding the one causing MHITM#.

Locking

Locking of the M-bus using the 82495's KLOCK# output is required to ensure atomic accesses for CPU locks. For example, memory variables called semaphores in a multitasking airline-reservation system prevent two processes from trying to update the same list of flight reservations simultaneously. A task would read the value of the semaphore in an uninterrupted read-modify-write (RMW) sequence, asserting the CPU's LOCK# signal during the RMW to block interrupts[1] (and block locked accesses by other processors to the same semaphore in a multiprocessor). If interrupts or other accesses were allowed during the sequence, two processes (or processors) might both read the semaphore as "available" (zero) and both assume ownership, setting it to "unavailable" (nonzero). Then both might find the same empty seat and write their individual passenger's name in the same seat location. In the end, 101 passengers would have tickets for a 100-seat plane flight.

The 82495 and i860 XP CPU implement locks in a *sequentially consistent*, or serializing, manner. That is, all data loads and stores within the locked sequence occur on the external bus in the same order as they appear in the program. Also, all accesses in the program before the LOCK instruction are completed before the first locked read or write, and all the locked reads/writes complete before other accesses after the locked sequence. This sequentiality is required by the semaphore example above, to prevent the CPU from updating the reservation list before it has obtained ownership using the semaphore.

[1]The CPU automatically blocks interrupts during the LOCKed sequence. The bus arbiter is responsible for blocking other accesses.

The MBC must serialize by ensuring all back-invalidates from 82495 to the CPU have completed before activating BRDY# for any locked read or write. So the MBC must postpone locked BRDY#s until CAHOLD is inactive and SNPCYC# has been inactive at least 2 CLKs (refer to [82495/490DS] section 5.1.1).

Bus Lock vs. Address Lock

The 82495 echoes the CPU's LOCK# signal onto its KLOCK# output, and forces all CPU accesses to go to the M-bus, even if they are 82495 cache hits. That guarantees that other processors know of the LOCK and the accesses. The 82495 assumes a **BUS LOCK**, where all other processors are kept off the bus during KLOCK# activation. Most existing "standard" buses, such as Multibus-II, have lock protocols which do such an exclusive lock. 82495 snoop behavior during assertion of its own KLOCK# is undefined, since it expects no other requests will be permitted then. The 82495's KLOCK# can remain asserted for multiple cycles when used with the i860 XP CPU, because the processor allows up to 32 instructions inside a LOCKed sequence.

The 32-instruction i860 XP CPU LOCKed intervals may exceed 32 CLKs, as each instruction could take several clocks and cause a TLB miss (the intervals would be even longer if the i860 XP CPU did data cache line fills and line writebacks during LOCK#, but the 82495 prevents that by making KEN# = 1). Unfortunately, this limits bus concurrency. When several 82495s share a bus or interconnection network, performance would improve if a LOCK# from one processor did not block all others from accessing memory and I/O. Multiprocessors based on the Intel486 DX CPU are not affected as severely by LOCK#, because its lock endures only a few clocks—two memory accesses at most.

To improve performance of locks in a multiprocessor, a scheme of **ADDRESS LOCKING** may be implemented. This non-blocking protocol allows other accesses to the bus and memory in spite of LOCK# activation, and requires only that no other CPU tries to access the same LOCKed address. If another CPU does try to access the same location, that second CPU must be stalled until the first LOCK is de-asserted. To ensure that the second CPU continues to snoop accesses while stalled, BGT# to it for its request must be delayed until the lock is obtained, as signalled by the bus arbiter. Semaphore integrity is preserved if all CPUs follow the software convention of locking their RMW (Read-Modify-Write) semaphore accesses. Also by convention, the address corresponding to the first access with LOCK# asserted is the only locked location permitted to that processor, until LOCK# deasserts (refer to the i860 Microprocessor Family Programmer's Reference Manual Intel order #240875, Section 5-14).

Would software want to be able to cache lockable locations? Since they are used for interprocessor or interprocess communication, it might seem dangerous to keep them "hidden" in a cache. However, caching allows a CPU to read a semaphore repeatedly without generating bus traffic, waiting until the semaphore is free as indicated by a zero value. These reads can be done in non-locked fashion. If a copy of the semaphore is cached, no bus traffic is used for the reads, and the semaphore value still gets updated via the normal MESI consistency hardware when the semaphore's owner writes it with a new value.

KLOCK# de-assertion for back-to-back Intel486 DX CPU locked accesses is required of the MBC if it uses address-based locking, so that the lock-manager knows the correct address. The i860 XP CPU always deactivates LOCK# for at least one clock between separate locked regions, by virtue of its deactivation in the clock after the last locked ADS#. However, the Intel486 DX CPU deactivates LOCK# only in the clock after the last BRDY# of the last locked access. Thus LOCK# and KLOCK# may not deactivate when two XCHG instructions occur in succession. The MBC can insert a deactivation of the M-bus MLOCK# signal by knowing all Intel486 DX CPU locked accesses are Read-Modify-Write sequences. The MBC should deassert MLOCK# regardless of KLOCK#'s value, after the write.

Deassertion of KLOCK# by the MBC hardware may be required in any Intel486 DX CPU system, to avoid bus timeout and starvation of other bus masters when a continuous stream of locked accesses occurs in one processor's program. Without it, one processor could monopolize the bus and prevent re-arbitration.

CPLOCK#

CPLOCK# has a purpose similar to KLOCK# in Intel486 DX CPU systems, but is unused in i860 XP CPU systems. PLOCK# (Psuedo-LOCK) indicates an atomic 8-byte 2-transfer write for floating-point data which should not be interrupted. The 4-byte bus of the Intel486 DX CPU requires 2 transfers for an 8-byte datum, and if only half the transfer gets done before another bus master reads memory, half-wrong data could be read.

Thus the MBC should not relinquish the bus nor require snoops of its 82495 from the time of the BGT# for the first write (when CPLOCK# was asserted by 82495) through the BGT# of the second write. This increases the worst-case delay of writeback for a 82495-snoop-hit to a modified line; to avoid the delay, the MBC can tie the CPLOCK# [PLOCKEN] pin low to disable PLOCK functionality.

9.0 MORE ALTERNATIVES

In addition to the options discussed above, several other choices affect Memory Bus Controller design.

M-bus clocking should be chosen to allow future versions of 82495 and 82490 at higher clock speeds. Upgrading the CPU module performance by replacing the processor and 82495/82490 will be possible. While some redesign of the CPU-side MBC state machines may be needed for faster clocks, the memory bus can remain the same. Thus an asynchronous interface with either a strobed unclocked M-bus or a clocked M-bus at less than 50 MHz is advised. A fully synchronous M-bus/CPU MBC would be difficult to move to higher clock speed.

One convenient way to design the MBC is with the M-bus MCLK = 0.5*CLK. Probably it will be possible to keep the M-bus at half the CPU CLK rate, even with faster CPUs. The big advantage of this half-speed link is that no synchronizers are needed within the MBC if the MCLK and CLK edges are skew-controlled. The MBC can be totally on CLK, as in the design example of Appendix B.

The choice between a **Strobed or Clocked M-bus** is often determined by existing bus protocols in which 82495/82490 will be used. Most existing buses are clocked; however, Futurebus+ requires all bus entities to use strobed tranfers, but allows an optional clocked mode for high-speed packet transfers [Fbus90]. The tradeoffs are shown in Table 2.

Line size and M-bus width also determine upgradability to possible future versions of 82490 on the same M-bus, with more than 32kB per chip. If a higher-density 82490 becomes available, the fact that 82495 has 8k tags requires:

128 data bytes per tag (128 byte line, or sectored 64-byte lines)
AND
8-byte or 16-byte memory bus width

to allow a 1 MByte or 2 MByte 82490 configuration. If a smaller bus is used, a larger 82490 is possible, but the bus-size multiplexing described earlier would be needed.

Writeback (WB) cache policy is advised for high-performance (multi)processors to limit bus traffic. However, a **writethru** (WT) design is simpler for the MBC because there never is a need to backoff the 82495 due to MHITM#. In fact, the snoop window in a WT system becomes unnecessary and SWEND# can be activated simultaneous with KWEND#. In such a system, the only states of cache lines are S or I. Snooping has no effect during reads and only causes invalidations (in the slaves) for writes in a WT design. Cache-to-cache transfers and RFO are irrelevant.

Table 2. Clocked vs. Strobed MBUS Tradeoffs

CLOCKED MBUS Advantages	STROBED MBUS Disadvantages
Design techniques for clocked systems are well known.	MBC design may require delay lines and non-conventional design techniques.
Fast arbitration using MCLK state machines.	Arbitration slow because signal must be synchronized at arbiter and at modules.
Burst transfers proceed at one datum per MCLK	Burst throughput slowed if each transfer requires acknowledgement from receiver.
CLOCKED MBUS Disadvantages	**STROBED MBUS Advantages**
Must round-up delays to MCLK period quanta EG., 33 ns delay means two 30 ns MCLKs needed.	Delays determined by device speed and physics, not by MCLK quanta.
Some 82495-to-82495 signals must be twice synchronized: once at sender, once at receiver.	Each signal goes through sychronizer once, only at receiver, so less time is lost at synchronizers.
Backplane length limited.	Fewer limits on backplane length or capacitance or number of boards.
MCLK skew must be controlled.	No clock skew worries.
Requires assumptions on CLK vs.MCLK speed ratio: for example, CLK > MCLK > CLK/2.	Any CLK frequency will work.

10.0 MBC DIFFERENCES FOR i860 XP CPU VERSUS Intel486 DX CPU

The same MBC design can be used for either i860 XP CPU or Intel486 DX CPU if the MBC supersets the requirements of the two. A "CPU_TYPE" configuration pin can be included in the MBC to modify its behavior. First, make the features as common as possible:

- Choose a configuration acceptable for both CPUs:
  a) 256 kBytes, 4 transfers/line, 64-bit M-bus, 32-byte line.
  b) 512 kBytes, 4 transfers/line, 128-bit M-bus, 64-byte line.
  c) 256 kBytes, 8 transfers/line, 64-bit M-bus, 64-byte line.
  d) 512 kBytes, 8 transfers/line, 128-bit M-bus, 128-byte line.
- i860 XP CPU-pfld data is cached in 82490—no optimizations are included for pfld.
- Assume that LOCK# duration does not matter (IE, that back-to-back LOCK#ed requests from Intel486 DX CPUs and long LOCK# cycles in i860 XP CPU do not cause bus ownership timeout).

Features Strictly for the Intel486 DX CPU :

- BE7–4# for M-bus must be synthesized by the MBC from A2 and BE3–0#.
- CPLOCK# protection.
- WRMRST (warm reset) can be included for both CPUs, but is optional.

Features Strictly for the i860 XP CPU:

- Burst writes from the CPU (Length=2 and Length=4).
- A second 74F377 BE#-latch is needed, for i860 XP CPU pins BE7#–BE4#, LEN, and CACHE#. PCYC and CTYP can also be latched for debug purposes.
- PCHK# output from i860 XP CPU must be ignored except during the CLK after BRDY# comes from the MBC. PCHK# from Intel486 DX CPU is always valid.

Differences between the MBCs:

- Configuration pin strapping of 82495 inputs.
- Decoding CPU request burst length from CLEN1:0(82495 pins in Intel486 DX CPU systems) or LEN and CACHE#(i860 XP CPU).
- CPU Line length—16 bytes vs. 32 bytes (i860 XP CPU) means that the Intel486 DX CPU MBC will give 2 BRDY#s for every 1 BRDY# of the i860 XP CPU MBC.

Differences between Intel486 DX CPU and i860 XP CPUs which have no impact on MBC:

- Intel486 DX CPU FLUSH# input pin.
- i860 XP CPU writeback caching, HITM#, and BOFF#.
- i860 XP CPU CS8 vs. Intel486 DX CPU BS8#, BS16# (none are really useable).
- Intel486 DX CPU RDY# pin and interruptable bursts (not useable with 82495).
- i860 XP CPU acknowledges HOLD during LOCK#.
- EADS# duty cycle (50% maximum for i860 XP CPU and 100% for Intel486 DX CPU, but handled by 82495).
- KEN# pin sampling interval by the CPU.
- Behavior of CPU in response to BOFF# assertion.
- i860 XP CPU BERR (Bus ERRor) pin versus Intel486 DX CPU NMI (Non Maskable Interrupt).

11.0 SUMMARY

The interface between a CPU/82495/82490 chip set and a system memory bus allows much flexibility and a wide range of performance options. The simplest MBC can be a few PALs, while a top-performance multiprocessing version may take thousands of gates on an ASIC. Signal pin counts for the MBC can range from 70 to 120, varying with the memory bus definition implemented by the MBC.

While beyond the scope of this document, topics for consideration include detailed timing diagrams, critical path analysis, simulation of bus traffic, and hit rates. Useful also are simulations of performance impact of the number of CPUs, WB versus WT policy, memory latency, CTCT, RFO, and duplicate tags. Also at issue are interrupt controller hardware, PAX concurrency control, boundary scan and selftest, PC-compatibility-implications, i860 XP CPU pfld options, and high-speed design issues of impedance, termination, and noise.

12.0 BIBLIOGRAPHY

[Agarwal88] "An Evaluation of Directory Schemes for Cache Coherence", by A. Agarwal, R. Simoni, J. Hennessy, and M. Horowitz (CH2524-2/88/000/0280 IEEE 1988)

[Fbus90] "Futurebus+: Its features, and how to use them," John Black, in VMEbus Systems Magazine, Feb. 1990, p. 23:40.

[Ham90]Tucker Hammerstrom, "Metastability," Intel Techbit # PLD-0390, March 1990.

[82495/490DS] 82495XP Cache Controller/82490XP Cache RAM Data Sheet, Intel order #240956.

[i401] Intel486 DX CPU Microcomputer Model 401 Board Technical Reference Manual, order #504366.

[Intel486] Intel486 DX CPU Microprocessor Data Sheet, Intel order #240440.

[i860XPDS] i860 XP CPU Microprocessor Data Sheet, Intel order #240874.

[Thakkar90] "Performance of an OLTP Application on Symmetry Multiprocessor System;" by S. Thakkar and M. Sweiger (CH2887-8/90/0000/0228), 1990 IEEE Intl Conference on Computer Architecture.

APPENDIX A: Questions and Answers on MBC Design

Q: Why activate BGT# early, since 82495 won't snoop between BGT# and SWEND#?

ANS: CNA# for MBC pipelining ignored until BGT#. Also BGT# must precede CRDY# by at least 3 CLKs. And BGT# must precede BRDY#.

Q: How does PAX multiprocessing work with 82495 and an MBC?

ANS: A CCU chip must be included on the M-bus side of 82495 and 82490 for each i860 XP CPU in a PAX multiprocessor. Refer to [MPIC90].

Q: Can the i860 XR CPU use a 82495/82490 cache?

ANS: No, the bus protocol of 82495 and 82490 matches Intel486 DX CPU and i860 XP CPUs, but not i860 XR CPU.

Q: Can 2 CPUs plug into one 82495, getting efficiency from shared cache?

ANS: No, the protocol and physical capacitance of the interface do not allow it.

Q: Should the same MBC be used for Uni & Multi? (i.e., how much extra logic is added to make a multiprocessor MBC?)

ANS: It is possible, and the extra logic is reasonable for a Uni which could be upgraded to multi by adding another CPU + cache module.

Q: Are software models of 82495/82490 available for simulation of MBCs? What simulators are supported?

ANS: As of September 1990, beta versions of models will be available Q4 1990 from Silicon West, Inc. Phone = (213)597-5995, FAX = (213)494-4588. Contact Silicon West for information on simulators supported (currently Workview, Verilog, Zycad VHDL, Mentor Graphics).

Q: What is the fastest possible transfer of data from Mdata to Cdata? (i.e., how many CPU clks are spent?)

ANS: The initial timings are listed in [82495/490DS]. They are about 1.5 CLK periods including set-up-time at the CPU data pins. The connection from CDATA to MDATA is essentially a flow-through path.

Q: Can the CPU-bus and Memory-Bus be on the same 50 MHz clock?

ANS: Yes, but multiprocessor memory buses probably have too much capacitance and trace length to tolerate a 50 MHz clock.

Q: What are pin-counts for an MBC (i.e., will it fit in my ASIC)?

ANS: 70 to 120 signal pins, depending on the bus protocol and MBC features.

Q: How long is a reasonable cacheability window, in MCLKs?

ANS: KWEND# is activated when MKEN# and MRO# are stable. MKEN# and MRO# can come from address decoders in the MBC or on the MBUS. Thus KWEND# could be 2 CLKs after CADS# if the MBC itself determines cacheability, or as much as 5 MCLKs if the M-bus must see the request and determine MKEN#.

Q: How long is a reasonable snooping window, in CLKs?

ANS: MWB/WT# and DRCTM# are generated from the snoopers' MTHIT# and MHITM# signals. Thus SWEND# is activated when those signals (MWB/WT#, DRCTM#) are stable. That would be at least 7 CLKs, not counting the possible delay between CADS# and its M-bus counterpart MADS#. (see the discussion of snoop window above).

Q: Is the SWEND# window length deterministic, or must SNPBSY# determine it?

ANS: It is deterministic, but may be long when the 82495 is busy. Yes, the SNPCYC# signal is required to determine SWEND#. If SNPCYC# is not used, then the worst-case 82495 delay must be imbedded into the MBC logic, making the window longer than necessary most of the time.

Q: How long can 82495 be "busy", activating SNPBSY# and ignoring subsequent SNPSTB# activations?

ANS: 82495 busy-ness is not due to CPU requests, because 82495 gives higher priority to the snoops. But for snoops to M-state 82495 lines, 82495 must do inquiries to the i860 XP CPU and get the more-recently modified data from i860 XP CPU before 82495 can writeback. A 82495 connected to an Intel486 DX CPU does not need to get modified data, as the Intel486 DX CPU has only S-state lines in the CPU cache. However, if SNPINV was active, 82495 must back-invalidate either CPU for S, E, or M state lines. The 82495 must do multiple inquires or invalidates when the line ratio is 2 or 4.

Q: What is the synchronization penalty in snooping (ie, how long from M-bus request to MHITM# validity)?

ANS: About 3 CLKs. See the discussion of "snoop window" above.

Q: What is optimal 82495 cache-line length (32,64,128)?

ANS: This is TBD from simulations or measurements. It depends on the behavior of SW applications the HW is intended for.

Q: Can Futurebus+ be used as the M-bus for a 82495/82490 system?

ANS: Yes. The Futurebus+ spec is compatible with the 82495/82490. It supports MESI, strobed data transfer, address pipelining, cache to cache transfers, Read For Ownership, and many other features. 82490 would be used in strobed mode for Futurebus+.

Q: Can 82495 do a split-transaction bus (if not, why not?)?

ANS: Maybe. 82495 implements a restricted-backoff protocol to eliminate potential deadlock conditions in a shared bus multiprocessor environment. Because of that protocol, and the fact that 82495 will not snoop between BGT# and SWEND#, it is difficult to implement split transactions. It may be possible, using an additional set of tags which replicate 82495's and allow snoops to continue between BGT# and SWEND#.

Q: Can another 82495 be used for the "duplicate tags" for split transaction snooping?

ANS: No, the 82495 signal definitions and protocols make that very difficult.

Q: Why do the KWEND# and SWEND# signals exist?

ANS: SWEND#, by gating 82490-to-CPU-data-transfer, allows the M-bus data transfer simultaneous with snooping. In the usual case, no modified copy will be found by the snoopers, so that transfer was not wasted. The alternative (that data cannot be transfered from memory until snoops complete) costs performance or requires a central tag directory. SWEND# triggers the 82495 to update its tags.

KWEND# allows a variety of cacheability determination schemes—a long delay to determine MKEN# and MRO# might be needed if a programmable RAM or EEPROM decodes cacheability based on address. If not, KWEND# can be activated quickly if there is a local MBC decode of A31:A28 to determine MKEN#, for example.

Q: Why not just one WEND signal?

ANS: Performance. KWEND# can be determined quicker than line-status in most implementations. The early knowledge of cacheability to the 82495 allows it to begin line replacements and allocations, and activate the next CADS# to MBC.

Q: How to connect 8-bit (or 16-bit) devices such as ROM and serial ports to 82490?

ANS: If the devices are made non-cacheable, they can be tied to the MDATA pins of the least-significant 82490s. However, if fetches from them must be cacheable, then byte assembly logic (latching transceivers) must exist to allow 82490 to transfer from them 4 or 8 bytes at a time (1 M-bus width per transfer). 82495 and 82490 require all cacheable locations to do burst transfers an M-bus-width of data per transfer.

Q: Does the 82495 have a CS8 mode? Does 82495 support i860 XP CPU in CS8 mode?

ANS: To support i860 XP CPU CS8 mode with 82495, the 8-bit ROM must be marked non-cacheable. This means that code being fetched in CS8 mode won't be cacheable in the 82495 or the i860 XP CPU. For an 8-byte M-bus, the ROM data pins must be wired to the M-bus (MDATA of 82490) bits 7:0. For a 16-byte M-bus, the ROM must attach to M-bus bits 7:0 AND bits 71:64, which would require an 8- bit transceiver at the ROM.

Q: Should the DRAM controller be part of the MBC?

ANS: For a simple uniprocessor, perhaps. Multiprocessors would have a DRAM controller for (each bank of) main memory, separate from the MBCs.

Q: How can the system implement retry upon an M-bus parity error?

ANS: The MBC must re-issue the initial request, and reset the 82490 transfer logic using the MSEL# signal.

Q: Can 82490 use an ECC corrected-bus?

ANS: ECC (Error Correcting Code) can be used on the main memory bus, but the ECC check bits must be converted to parity or discarded before feeding the 82490. ECC would have to be generated at the 82490 MDATA pins for writes to memory.

Q: Can the MBC implement cache-to-cache transfer on a write?

ANS: No, the 82490 cannot "snarf" write data. That is, it does not merge a write (partial line) from the M-bus with existing cached lines. It can do Read-For-Ownership, merging write-miss data with an incoming line writeback from another cache.

Q: Can semaphores be cached in 82495/82490?

ANS: Yes, but all read/writes which are locked are forced onto M-bus. So the semaphore would be read repeatedly without locking, until it is "free". Then SW would re-read it in locked fashion to obtain ownership.

Q: Is there any advantage to making semaphores cacheable, if all locked accesses go to M-bus?

ANS: Yes, SW can repeatedly read the semaphore without LOCKing it, and no bus traffic thus is generated, waiting for the release of the semaphore by any other master.

Q: Can a single multiplexed address + data bus (like Multibus-II) be used for M-bus?

ANS: Yes, but transceivers external to the 82495 and 82490 are required.

Q: How does the MBC implement a "BACKOFF" when another 82495 activates MHITM#?

ANS: If the data requested from a master 82495 is Modified in a snooper 82495, the master BC must postpone CRDY# until the modified line is deposited in the master 82490, after the snooper flushes the modified line to M-bus.

Q: Can MBC duplicate the CPU cache tags, to avoid unnecessary inquire cycles?

ANS: Yes, but the performance benefit may not warrant the extra hardware.

Q: Can i860 XP CPU Late-Backoff mode be used with 82495?

ANS: No.

Q: What are the advantages and disadvantages of doing an asynchronous system (where MCLK is not the same as CLK)?

ANS: Designers can easily upgrade the CPU side to higher frequencies (above 50 MHz) by faster PLDs in the CPU side of the MBC. The M-bus interface and all modules on the M-bus will not need to be changed. It easier to design a board when most parts run at a lower frequency.

Q: If the 82490 is reading information from the memory bus and the MBC is generating BRDY#'s (RDYSRC = 1), can the MBC abort the cycle by giving a premature CRDY#, and restart it?

ANS: The MBC can abort a memory bus cycle but cannot abort a CPUbus cycle. Once the first BRDY# is generated the cycle must complete. On the memory bus, a cycle is not aborted by giving an early CRDY#. In fact the 82495 does not understand that a cycle has been aborted. Only the MBC and 82490 are involved. The 82490 allows its buffer to be reset using the MSEL# signal.

Q: What is the purpose of 82490 having a separate MOCLK for output data, in addition to the MCLK for input signals?

ANS: MOCLK allows greater hold time for writes from 82495, if it is skewed slightly from the MCLK which M-bus receivers use. MOCLK and MCLK must be exactly the same frequency. If the skew is not needed, MOCLK can be tied low.

Q: How many levels of pipelining can the 82495 use on the external memory bus?

ANS: Each 82495 can use one level of pipeline on the memory bus, so the bus pipe depth can be greater in a multiprocessor. A uniprocessor allows just one level of M-bus pipeline.

APPENDIX B:
Intel486 DX CPU Uniprocessor MBC Design

Please refer to Application Note AP-458, *Designing a Memory Bus Controller for a 50 MHz Intel486 DX Microprocessor Based System.* (Intel order #241166).

APPENDIX C:
i860™ XP CPU DUAL-PROCESSOR MBC

OVERVIEW

This section presents a design for a memory bus controller for a system containing two i860 XP processors, each with an 82495XP/82490XP secondary cache. This MBC, together with an i860 XP CPU, 82495XP, and 82490XP, comprises a core which interacts with a memory bus utilizing a bus protocol similar to that of the i860 XP CPU.

The design presented here features an i860 XP CPU and 256 KB of 82495/82490 cache running at 50 MHz in each core. The clocked 64 bit (+8 parity) memory bus is asynchronous to the CPU and cache clock, allowing memory to run at lower speeds for more economical and convenient memory design. The MBC features snooping and pipelining to the memory, as well as advanced 82495 processes like write allocation, read for ownership and cache-to-cache transfers.

ASSUMPTIONS

The implementation presented here is a two processor design which can be extended to more than two CPUs. The definitions and examples given in this appendix are specific to the two processor version. The section **Extension to 3 or More Processors** gives specifics for larger systems based on this design.

The memory bus is 64 bits data plus 8 bits parity.

The MBC design allows the processor to run at a higher clock frequency than the memory bus. The frequencies are constrained such that the ratio of the frequency of the processor CLK and the frequency of the memory bus MCLK is between 1 and 2:

$$\frac{CLK}{2} \leq MCLK < CLK$$

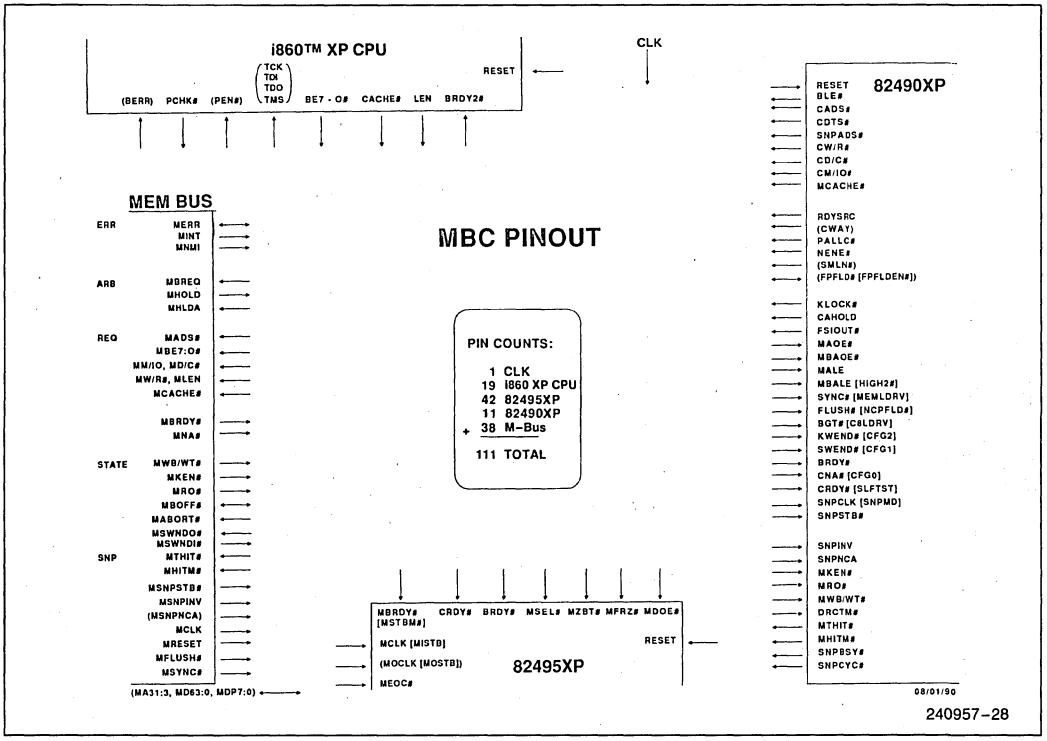


Figure C-1. Pinout Environment of MBC

This constraint ensures proper synchronization of signals which cross between the MCLK portion of the MBC and the CLK portion. The prototype was designed and simulated with a CPU speed of 50 MHz and a memory bus speed of 33 MHz.

Snooping mode can be independently set to strobed, or clocked in each core.

The main memory is responsible for returning the MKEN# attribute to the memory bus controller in the MCLK following MADS# assertion.

To save synchronization clocks, the MBRDY# signal of the protocol is defined to be asserted one MCLK before data is actually available.

The 82495 operates with 32 bytes/line, 1 line/sector, and requires 4 memory bus transfers per line fill.

OPTIONS

With modifications the 82495 can operate in a mode with 64 bytes/line, 1 line/sector, requiring 8 memory bus transfers per line fill.

The design here utilizes the 82490's clocked memory bus mode. The strobed mode can also be utilized by making modification to the design.

Support for various 82495 PFLD modes can be added to the design.

Operation with either write-through or write-once protocol can be performed.

MEMORY BUS PROTOCOL

M-bus Signals

The system M-bus resembles the i860 XP CPU bus. It allows CPU modules with or without external cache on the same M-bus, so that balance between high performance and low cost can be achieved. The signal specifications below indicate Input (I), Output (O), or bidirectional (I/O) from the MBC's point of view. Output signals to the memory bus such as MADS#, MLEN, and MA31:MA3 are floated by all MBCs except the one currently owning the M-bus.

Signals whose names begin with Y (as in YBGT#) are in the MCLK side of the MBC, while an X prefixed name is in the CPU CLK side of the MBC. The X and Y signals are internal to the MBC.

MRESET (I) - Memory bus RESET

This signal forces the CPU to begin execution in a known state. It resets all MBC machines which are driven by MCLK. It is also synchronized (via a 2-stage synchronizer) to CLK and fed to the RESET inputs of the CPU, 82495, 82490s and all MBC machines which are driven by CLK.

MADS# (I/O) - Memory bus ADdress Strobe

This signal indicates that a new valid bus cycle is currently being driven. The cycle address (A31:A3) and cycle specifications are valid in the MCLK that MADS# is asserted. A pipelined MADS# will be issued only after the MBC knows that the current cycle is guaranteed not to be aborted. For most memory accesses, the master will assert MSNPSTB# to snoop other caches on the bus. When MSNPSTB# has been asserted, MNA# will cause a new MADS# to be issued after MSWENDI# signifies snooping has completed. Furthermore, if MHITMI# was asserted with MSWENDI# in this case, the new MADS# cannot be issued until after the current cycle (now a snoop write-back) has been completed. When MHITMI# is not asserted with MSWENDI#, MADS# can be asserted immediately following MSWENDI#. If MSNPSTB# was not asserted for the current cycle, then MADS# could be issued immediately after MNA#, without waiting for MSWENDI#.

For read cycles MADS# is issued after CADS#, regardless of CDTS# state. Requesting the memory bus, via MBREQ, is also done immediately after CADS#. This is due to the fact that CDTS# in a read cycle does not affect the memory bus, but indicates when the first BRDY# can be issued to the CPU.

For memory writes MADS# is issued only after CDTS#. Requesting the memory bus, via MBREQ, is also done after CDTS#. This guarantees that for write cycles the memory bus data is valid 1 MCLK after MADS# (similar to the CPU).

MNA# (I) - Memory bus Next Address Acknowledgement

This is the memory bus next address signal, driven by the memory controller. It indicates to the MBC that the memory bus is ready to accept a new bus cycle, although the previous one has not been completed yet. If the MBC has a new cycle pending and the current cycle is guaranteed not to be aborted (see MADS# above), then a new MADS# will be issued. Note that the maximum level of pipelining on the memory bus is 1.

MBRDY# (I/O) - Memory bus Burst ReaDY#

This is the burst ready signal. For read cycles, MBRDY# indicates that in the following MCLK the memory bus will present valid data on the 82490 MDATA pins. For writes, MBRDY# indicates that in the following MCLK the memory bus will accept the data from the 82490 MDATA pins. Note that this signal is active 1 MCLK before the data is available on the memory data bus. This reduces the synchronization penalty between the M-bus and CPUbus by 1 MCLK period.

For a clocked-asynchronous MBC, MBRDY# is delayed by the MBC 1 MCLK and passed to the 82490 MBRDY# pin. For a strobed-asynchronous MBC, the 82490 MISTB and MOSTB will change value in response to MBRDY#.

For Cache to Cache Transfers, the MBC with the Modified line drives MBRDY# active once per MCLK without wait states for the duration of the line burst.

MSNPSTB# (I/O) - Memory bus SNPSTB#

This is the memory bus snoop strobe signal. It is asserted 1 MCLK after MADS# by the MBC which asserted MADS#, for all cycles that could be M-state in the other MBC. In writebacks and I/O cycles, MSNPSTB# is not asserted. The MSNPSTB# output of each MBC is connected to the 82495 SNPSTB# input of the other MBC, in this two processor design.

MSWENDO# (O) - Memory bus SWEND# Output

This is the memory bus snoop window end indication which is driven by the snooping MBC. It is connected to the master MBC's SWENDI# input, indicating that snooping is finished and the snoop attributes are valid.

MSWENDO# is an asynchronous signal which is triggered by the 82495 SNPCYC# falling edge, and is negated after sampling an active SNPSTB#. MSWENDO# of one MBC is connected directly to the MSWENDI# input of the other MBC.

MSWENDI# (I) - Memory bus SWEND# Input

MSWENDI# is connected directly to the other core's MSWENDO# output. It is internally sent to two synchronizers: synchronized to CLK to generate 82495 SWEND#, and synchronized to MCLK for MBC state machines which determine whether the current bus cycle should be aborted.

MSWENDI# indicates the end of the snoop window and that the snoop results MHITMO# and MTHIT# are valid. An active MHITMI# indicates a snoop hit to a modified line, and causes the master MBC to discard any data which has arrived from main memory, so that new data, which is being written out as the snooping core performs a snoop write back, can be accepted. MTHIT# of each core is connected to the MWB/WT# input of the other core, to generate the WB/WT# signal to the 82495.

MHITMO# (O) - Memory bus HITM# Output

This indicates a snoop hit to a modified line. In the two processor implementation of this MBC, it is connected directly to the other MBC's MHITMI# input.

MHITMI# (I) - Memory bus HITM# Input

MHITMI# is connected to the MHITMO# output of the other MBC, and determines if MBOFF# and MABORT# will be asserted. It is sampled on MSWENDI# activation.

MTHIT# (O) - Memory bus Snoop Hit Indication

This snoop hit indication is based on the 82495 MTHIT# output. The MTHIT# ouput of the snooping core is used by the master core to determine the WB/WT# state for the accessed line. The 82495 MTHIT# signal is passed directly onto the memory bus when the SNPINV signal is inactive for the snoop. On snoops with SNPINV active, the memory bus MTHIT# line is driven low, regardless of the value at the 82495 MTHIT# pin.

The MTHIT# signals from the memory bus controllers on the bus are wire-anded together. Because the 82495 MTHIT# output only changes state with each new snoop, the master memory bus controller must float its MTHIT#.

MBOFF# (O) - Memory bus BOFF#

This is the memory bus back-off signal which is driven by the master MBC. The master MBC floats its bus concurrent with MBOFF# activation. When the snooper MBC samples an active MBOFF# and it has a pending snoop write-back cycle, it issues the cycle to the memory bus. Note that the snooper issues the cycle even though it is still in a bus hold state (MHLDA asserted). If MHITMI# is sampled active during MSWENDI# and the previous cycle has completed, then MBOFF# will be asserted immediately after

MSWENDI#. If the previous cycle has not completed and the pipelined cycle hits a modified line, then MBOFF# will be asserted only after the previous cycle completes. The snooping MBC floats its bus only after the snoop write-back cycle has completed. Note that from the arbiter's viewpoint the bus is still granted to the master MBC.

MABORT# (O) - Memory bus Abort

This is the memory bus abortion signal which is driven by the master MBC. When the main memory samples an active MABORT# it aborts any cycle that is currently being serviced. The memory aborts the cycle regardless of the number of MBRDYs that have been issued. Thus MBRDY# of the aborted cycle will not be issued after MABORT#. A new cycle could be serviced immediately after MABORT#.

If MHITMI# is sampled active during MSWENDI# and the previous cycle has been completed, then MABORT# is asserted immediately after MSWENDI#. If the previous cycle has not been completed and the pipelined cycle hits a modified line, then MABORT# is asserted only after the current cycle has completed.

MABORT# can also be asserted during read for ownership with a hidden write (allocation after a non-completed write in the main memory). In this case if the master MBC samples an active MKEN# (1 MCLK after MADS#) during a potentially allocatable write cycle, it asserts MABORT# immediately, i.e. 2 MCLKs after MADS#.

Note that MABORT# is always guaranteed to be a 1 MCLK width pulse.

MLOCK# (I/O) - Memory bus LOCK

This signal does not exist in the current implementation. Instead, the MBC simply refuses to give up the M-bus to the arbiter when it is running locked accesses.

MHOLD (I) - Memory bus Hold Request

When this input to the MBC is asserted, the MBC asserts MHLDA and floats all inputs and outputs except MBREQ, MHLDA, MSWENDO#, and MBOFF#. If the MBC has outstanding bus cycles in progress (MADS# has been asserted), they are completed before the MBC relinquishes the bus. MHOLD is recognized during MRESET assertion.

MHLDA (O) - Memory bus Hold Acknowledge

The memory bus hold acknowledge signal goes active when an MBC relinquishes the bus in response to an MHOLD request. The memory bus controller floats its bus in the same MCLK that it issues the MHLDA. When the MBC leaves bus hold, MHLDA is negated and the core resumes driving the bus. If a cycle is pending when leaving bus hold, the MADS# will be issued in the same MCLK that MHLDA is negated.

MINT (I) - Memory bus Interrupt

This interrupt signal is connected directly to the i860 XP CPU in the core.

MKEN# (I) - Memory bus KEN#

This is the memory bus cache enable signal. It is used by the MBC to determine the length of the current bus cycle, and is also connected directly to the 82495 MKEN# input.

In potentially cacheable read cycles, it determines cycle length. In potentially allocatable write cycles, it determines whether read for ownership with hidden write will be performed.

In the current implementation, MKEN# must be driven by the memory controller in the MCLK after MADS# was issued.

MRO# (I) - Memory bus Read Only

Assertion of this signal causes an access to be treated as read only by the core. This signal is connected directly to the 82495 MRO# input, as well as to the MBC.

MWB/WT# (I) - Memory bus WB/WT#

This is the write-back/write-through input connected to the memory bus. It is connected through MBC logic to the 82495 MWB/WT# input.

MDRCTM (I) - Memory bus Direct-to-M

This is the memory bus DRCTM# signal which forces a line entering the cache to be placed directly in the [M] (modified) state. In addition to this signal which is connected from the memory bus to the 82495, the MBC can internally drive the 82495's DRCTM# pin during read-for-ownership cycles.

MFLUSH#, MSYNC# (I) - Memory bus FLUSH#, SYNC#

These signals cause the core to flush or sync its cache, by asserting FLUSH# or SYNC# to the 82495, respectively. The signals are driven by the main memory controller upon detecting a Core flush or sync command, which consists of a special cycle with either MBE1# or MBE3# active, respectively.

MBREQ (O) - Memory bus Request

The MBREQ# signal is asserted by an MBC to indicate to the memory bus arbiter that the MBC needs the memory bus. An MBC will generate this signal regardless of whether or not the MBC is currently driving the bus.

MBREQ# is not issued for snoop write-back cycles. If the snooping core already had its MBREQ# pin asserted, the pending cycle which caused the MBREQ# is aborted by the snoop write-back, according to 82495 protocol. The MBC state machines of the snooper, however, continue to assert MBREQ# until an internal time-out period has elapsed, allowing the snooping 82495 to reissue the aborted cycle after the snoop write-back has completed. Therefore a core which is waiting for the bus can service a snoop write-back without losing its request for the bus.

MLEN (O) - Memory bus LEN

This signal together with MCACHE#, MW/R# and MKEN# determine the memory bus cycle length according to the following table:

MW/R#	MLEN	MCACHE#	MKEN#	length	Notes
x	0	1	x	1	1
x	1	1	x	2	1
0	0	0	1	1	2
0	1	0	1	2	2
0	x	0	0	4	
1	x	0	x	4	

NOTES:
1. Locked i860 XP CPU write-back cycles (length=4), caused by the i860 XP CPU executing a FLUSH instruction during a LOCKed sequence, are treated as normal write cycles (length=1 or 2 according to LEN). This is allowed since i860 XP CPU write-back cycles always access a 82495 modified line (in [M] state) and are only written into the 82490, without updating memory.
2. MKEN# must be driven valid the clock following MADS# by the memory controller.

MMI/O#, MD/C# (O) - Memory bus I/O# and D/C#

These signals, together with MW/R#, define the memory bus cycle, according to the i860 XP CPU Data Sheet. They are driven in the same MCLK as MADS#.

MBE[7:0]# (O) - Memory bus BE[7:0]#

The byte enable signals to the memory bus identify which bytes are being accessed. They are identical to the CPU byte enables on CPU generated cycles. For 82495 generated cycles (write-backs and allocations) all MBE#s are asserted.

MCACHE# (I/O) - Memory bus CACHE#

In a master core MCACHE# is an output; in a snooping core it is an input. As an output, it indicates potentially cacheable reads or a 82495 write-back. MCACHE# is used by the system memory together with MLEN, MW/R# and MKEN# to determine cycle length. As an input, MCACHE# is connected to the 82495 SNPNCA pin.

MW/R# (I/O) - Memory bus W/R#

This signal is an output for a master core, an input for a snooping core. As an output, it indicates whether the memory access is a read ar a write, and is used by the system memory along with MMI/O# and MD/C# to determine the cycle type, according to the i860 XP CPU Data Sheet. As an input, the signal is connected directly to the 82495 SNPINV pin.

MA[31:3] (I/O) - Memory bus Address

These are the memory bus address lines of the MBC. Along with the byte enable signals, they define the physical area of memory or I/O accesses. In a master MBC they are driven by the 82495 onto the memory bus together with MADS# (same MCLK). In a snooping MBC, these lines are inputs to the 82495 which are latched by the MSNPSTB# signal.

MD[63:0], MDP[7:0] (I/O) - Memory bus Data and Data Parity

64 bits of data, 8 bits of parity, connected through transceivers to the i860 XP CPU and 82490s. When an MBC does not own the bus, these pins are tristated.

XAS#/XSAS# - X Unit Address Strobe

XAS# is generated in the X-unit (sync to CLK), and is synchronized and sent to the Y-unit as XSAS#.

XAS# indicates the start of a memory bus cycle from the X-unit (CLK side). XAS# is generated as a result of a CADS# from the 82495 on a read cycle or CDTS# from the 82495 on a write cycle. XAS# is held active until the X-unit receives YSBGT#.

YBGT#/YSBGT# - Memory bus Guarenteed Transfer

YBGT# is generated in the Y-unit, and is synchronized and sent as YSBGT# to the X-unit.

This signal is generated in the Y-unit after MADS# (the cycle has been issued on the memory bus). When YSBGT# arrives at the X-unit, the signal causes assertion of the 82495's BGT# input, and one clock later (non-pipelined cycle) the assertion of KWEND#. YSBGT# of a pipelined cycle (which is sampled during the initial cycle, i.e. before its CRDY#) causes the BGT# and KWEND# of the pipelined cycle to be issued immediately after CRDY# of the initial cycle.

YBGT# of a pipelined cycle cannot be issued before the MSWEND# of the previous cycle. This is guaranteed by the M-bus protocol, which ensures that a pipelined MADS# is not issued until after the MSWEND# of the previous cycle.

BGT#, KWEND# (O) - Bus Guaranteed Transfer, Cache Window End to 82495

BGT# and KWEND# are generated for every cycle (including snoop write-backs). In a non-pipelined cycle BGT# is issued immediately after sampling YSBGT# active, and KWEND# is issued 1 clock later. In pipelined cycles, these signals are asserted after the CRDY# of the initial cycle.

YMEOC#/YSMEOC# - (O) MBC Memory End Of Cycle

YMEOC# is generated in the Y unit, and is synchronous to MCLK, and sent to the X-unit as YSMEOC# . It indicates the M-bus transfer has finished, based on the MBC's tranfer length count. YMEOC# directly drives the 82490s' MEOC# inputs. YSMEOC# causes generation of the CRDY# signal to the 82495 and 82490s. For non-pipelined cycles CRDY# is issued immediately after an active YSMEOC# (if CDTS# was issued). For pipelined cycles CRDY# is issued after the CRDY# of the previous cycle (if YMEOC#, CDTS# of the pipelined cycle were issued).

YMEOC# is issued at least 2 MCLKs after YBGT# (for every cycle).

YCEOC#/YSCEOC# - MBC CPU End Of Cycle

This signal is internal to the MBC. YCEOC is generated synchronous to MCLK, and is synchronized to CLK to produce YSCEOC#. It indicates that the CPUbus transfer has finished, based on the MBC's tranfer length count. It generates the BRDY#s to the 82495, 82490, CPU, and to other MBC machines. For non-pipelined cycles all BRDY#s except the first are issued immediately after an active YCEOC# (if CDTS# was issued). For pipelined cycles all BRDY#s except the first are issued after the CRDY# or the last BRDY# (BRDY# * CLEN1) of the previous cycle.

YCEOC# can be issued before, with, or 1 clock after YMEOC#. When the line ratio is 2 or 4, YCEOC# precedes YMEOC# by a significant time, allowing CPU linefills to complete long before the M-bus tranfer completes.

YCEOC# is asserted only if RDYSRC is active (High).

BUS CYCLES

Non-aborted Read Cycles

Figure C-2 is a timing diagram for the memory bus controller executing a line fill after the i860 XP CPU issues a read which misses the 82495/82490. The diagram reveals a number of the signals which are internal to the MBC, to provide a better perspective on the timing of events. Note that signals which begin with an M are MBC signals to the memory bus. Signals that begin with Y originate in the Y side of the MBC which is synchronous to MCLK, and an X denotes origin in the X state machines, which are synchronous to CLK.

The i860 XP CPU microprocessor issues a read cycle in CLK 0, as indicated by the assertion of ADS#. The 82495 performs the tag lookup, and finds the request a cache miss. In CLK 2, the 82495 issues CADS# and the cycle control signals, alerting the memory bus controller that a 4 transfer 82495 read is requested.

The X side state machines, which run on the processor CLK, issue an XAS# on the CLK after CADS# for a 82495 read cycle (CW/R# = 0). The XAS# signal passes through the synchronizer running on MCLK to become synchronized in two MCLKs. The synchronized XAS# signal, called XSAS#, is sent to the Y side of the MBC in MCLK 4.

In MCLK 5, XSAS# has initiated the assertion of MBREQ to request the memory bus from the memory bus arbiter. If the bus is already owned (or once it is owned) by this MBC, XSAS# causes the assertion of MADS# to the memory bus, MAOE# to the 82495, and the internal YBGT# signal. The assertion of the 82495's MAOE signal allows the 82495 to drive its address lines to the memory bus. YBGT# indicates that the memory bus is owned by this MBC, and is sent to the synchronizer for the X side of the MBC as well as many Y side state machines.

On the Y side, YBGT# is used to deassert MBREQ#, to sample YALLOC# on writes, and to initiate MSNPSTB#. MSNPSTB# is asserted in MCLK 6 to request a snoop in the other MBC. YBGT# is also synchronized to CLK, appearing as YSBGT#, by CLK 9. YSBGT# causes the assertion of BGT# to the 82495 in CLK 10, and, 1 CLK later, KWNED#. The MKEN# input, which must be valid to the 82495 when KWEND# is asserted, must be driven by the main memory on the MCLK after MADS# for this implementation. These signal activities define the initiation of normal bus cycles (as opposed to snoop write-backs).

In this particular example, the memory bus responds quickly to the read request. Here, the memory subsystem drives MNA# to the MBC in MCLK 6, and presents data on the memory bus in MCLK 7. Since MBRDY# must be driven by the memory bus 1 MCLK before data is available, MBRDY# is asserted in MCLK 6, with successive MBRDY#s on the following MCLKs. The YMBRDY# output of the MBC is the MBRDY# signal delayed one clock, and drives the MBRDY# input on the 82490s to read in the incoming data.

While the data transfer is occurring, the second memory bus controller responds to the snoop request for this memory access in MCLK 8. Because the data is not present in the cache of the other core, that MBC will assert its MSWENDO# output with MHITMO# driven high. These outputs of the snooping core are tied directly to the MSWENDI# and MHITMI# inputs, respectively, of the master core in this two core implementation. Both of these signals are passed to the 82495 (MSWENDI# is synchronized first) as well as to the state machines of both sides of the MBC. The arrival of these signals allow the core to accept the data as valid, and conclude with the read operation when all of the data has been transferred.

The arrival of the fourth MBRDY# generates the YMEOC# and YCEOC# signals in MCLK 10. YMEOC# drives the MEOC# input on the 82490s. In addition, both signals are synchronized and sent to the X side of the MBC. Upon the arrival of YSCEOC#, the X state machines begin generating BRDY#s to the i860 XP CPU. Upon arrival of YSMEOC#, CRDY# is driven to the 82495, indicating the end of the cycle. YMEOC# and YCEOC# are used to reset many of the Y side state machines, including cycle type and length indicators, and the drivers of 82490 signals such as YMALE# and YMSEL#. On the X side, the reset functions are triggered by CRDY# and the last BRDY#.

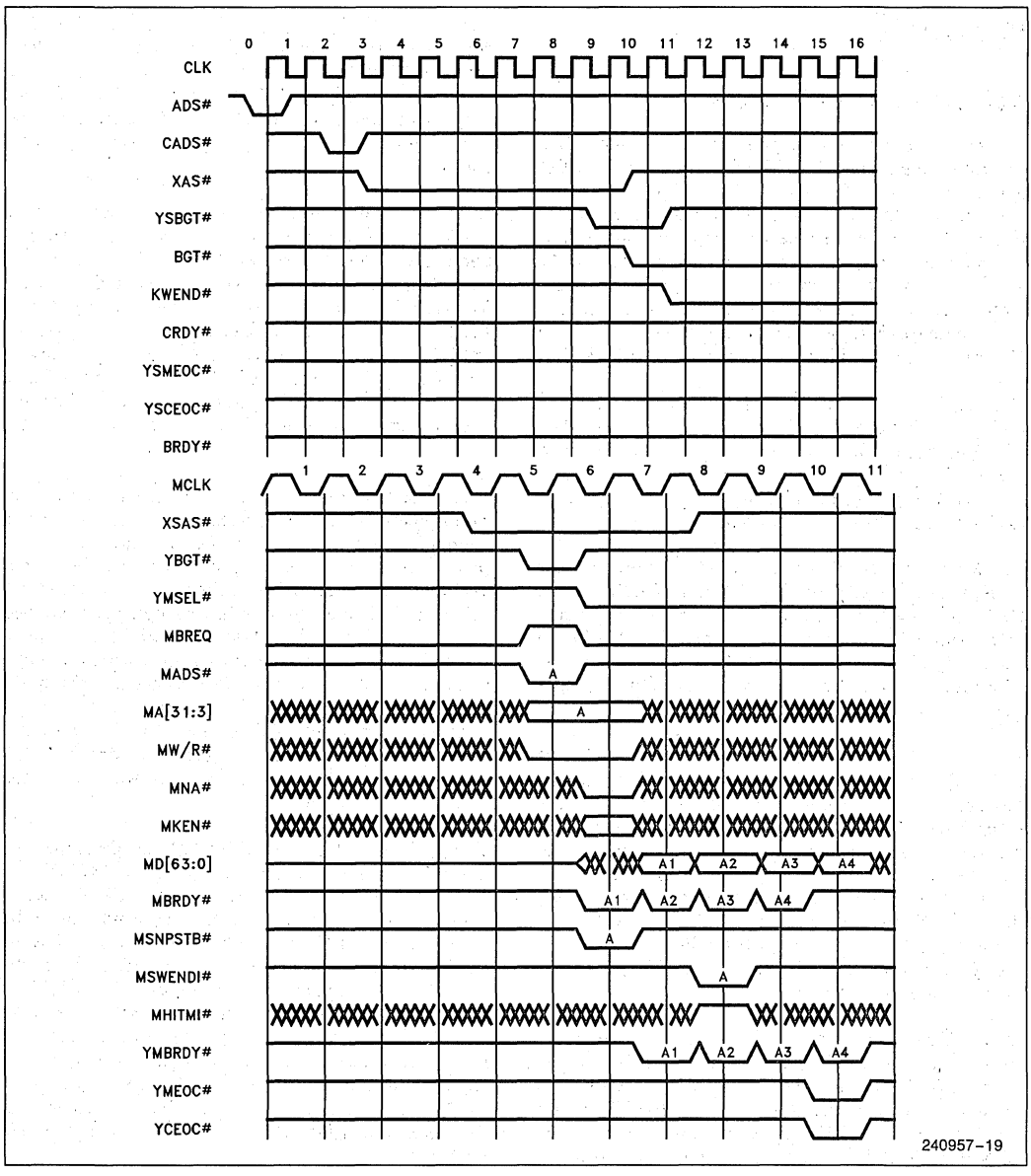


Figure C-2. Non-Aborted Read Cycles

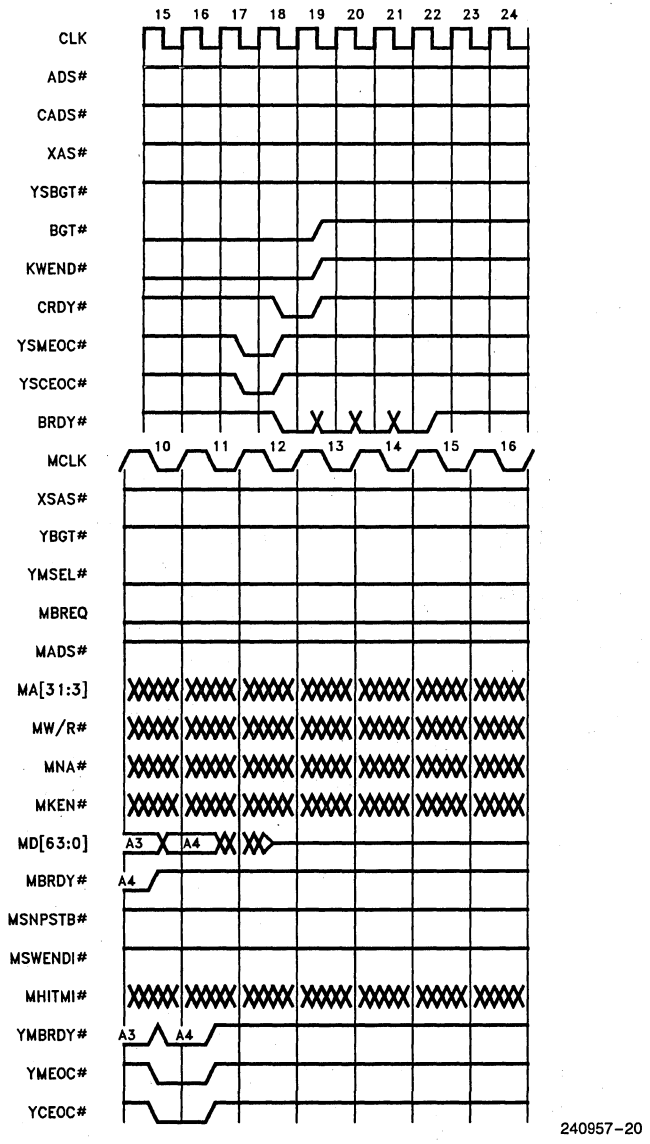


Figure C-2. Non-Aborted Read Cycles (Continued)

Aborted Non-Pipelined Cycles

Figure C-3 illustrates an aborted non-pipelined cycle. MHITMI# is sampled active during MSWENDI# (clock 4) indicating a snoop hit to a modified line. Since the cycle is non-pipelined, MABORT# is issued immediately and the core floats its bus (clock 5). Although the bus is floated by the master core, the master still owns the bus (MHLDA remains inactive).

MABORT# in clock 5 causes the main memory to abort its cycle regardless the number of MBRDYs that have been issued. MBOFF# is also asserted in clock 5 to indicate to the snooping core that the master is floating its signals and the write-back may begin. The main memory floats its data bus in clock 6 in response to MABORT#. In the following clocks a snoop write-back cycle is performed by the snooper. The snooper will release the bus at the end of the write-back.

Note that MSNPSTB# is not asserted during the write-back cycle since it obviously will not hit any cache.

Aborted Pipelined Cycles

Figure C-4 illustrates an aborted pipelined cycle. Although MHITMI# is sampled active during MSWENDI# (clock 7) MABORT# will not be issued immediately since the previous cycle has not been completed yet. MABORT# is issued in clock 9 after the last data slice was read into the core. The core floats its bus and asserts MBOFF# concurrently with MABORT#. Upon sampling MBOFF#, the snooping MBC begins the snoop write-back in clock 10.

Write Allocate

Figure C-5 illustrates a write cycle which is potentially allocatable. This write is performed on the bus only in order to sample the MKEN#, since the allocation cycle will only be guaranteed if MKEN# is active.

MKEN# is sampled active in clock 2 causing the MABORT# to be issued immediately. The reason to abort the write cycle, even before MSWEND#, is due to the fact that a read for ownership cycle is guaranteed to be performed after the aborted write.

In clock 4 the MADS# of the allocation cycle, which becomes the MADS# of the read for ownership cycle, is issued. This MADS# is issued only if MSWEND# has not been issued yet, or if MSWEND# was issued and MHITMI# was negated. If MHITMI# is asserted during the MSWEND# that was issued, MADS# will not be issued (since the snooper issues its MADS#).

A second MABORT# is issued in clock 8 indicating the memory to abort the allocation, and the snooper to start flushing the modified line. Note that a second MABORT# will be issued regardless if MADS# of

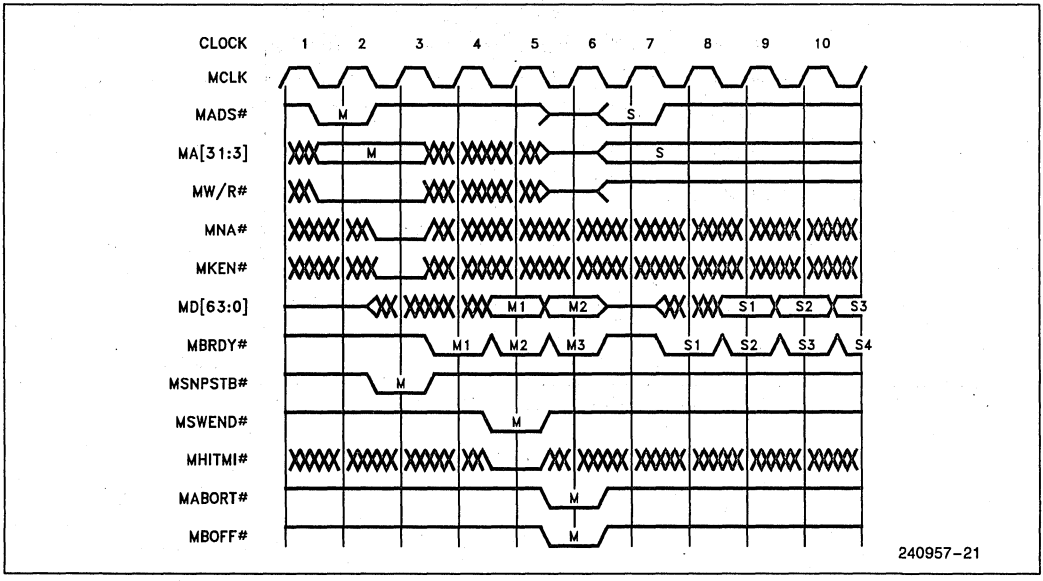


Figure C-3. Aborted Non-Pipelined Cycle

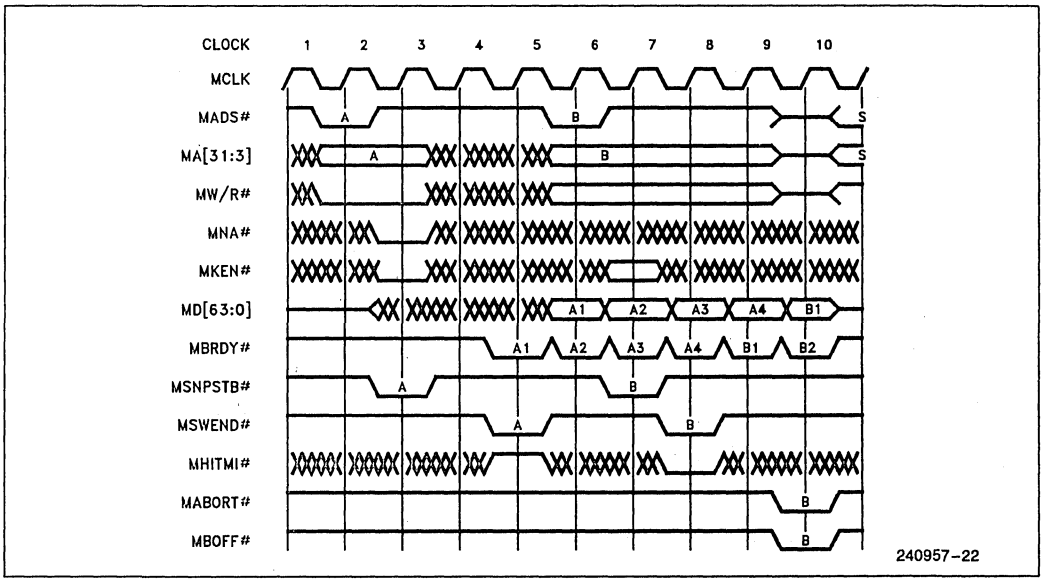


Figure C-4. Aborted Pipelined Cycles

the allocation was issued or not. The first MABORT# (clock 3) aborts the write cycle in the memory module and does not affect the snooper. The second MABORT# (clock 8) indicates to the snooper to start its write-back cycle (and if MADS# of an allocation was issued to also abort it in the memory module).

MSNPSTB# is not issued for the allocation cycle since write and allocation cycles access the same line.

If MKEN# had been negated in clock 2 then an allocation would not have been performed and the write cycle would have continued as a non-allocatable write cycle (see figure C-6).

Non-Allocatable Write

Figure C-6 illustrates a write cycle without an allocation. It can be either a non-potentially allocatable write cycle or a potentially allocatable write with inactive MKEN# (clock 1).

The write cycle is aborted (MABORT# in clock 3) after sampling active MHITM# during MSWEND# (clock 2). In clock 11 the master core re-issues the MADS# of the aborted write cycle (after the snoop write-back has been completed). MSNPSTB# will not be issued again since the updated data had been written into the main memory and the snooper has gone to the invalid state.

LIMITATIONS OF DESIGN

The primary limitation of the implementation as it has been presented so far is that it includes only two processors. The protocol set up in the design is not limited to two processors. The next section outlines the implementation details which must be modified to extend the design to more than two processors.

The design has no support for CS8 mode, so the processors cannot be booted from 8 bit EPROMS. Instead, both processors boot in 64 bit mode, which may complicate the use of the design in stand-alone systems.

The i860 XP CPU's BERR, or Bus ERRor, input is not utilized in this design. The pin could be used simply as a non-maskable interrupt pin, but the memory bus controller as designed makes no provision to use BERR to correct a faulty bus access. Likewise, the parity check results from the i860 XP CPU's PCHK# pin are of little value in this design outside of testing the i860 XP CPU's parity functions. The MBC itself does not check the PCHK# output, and has no means of reissuing an access in case of parity error.

The memory bus controller design here does not decode and utilize the i860 XP CPU INTA cycles. The INT pin itself is connected directly to the i860 XP CPU, without affecting MBC operation.

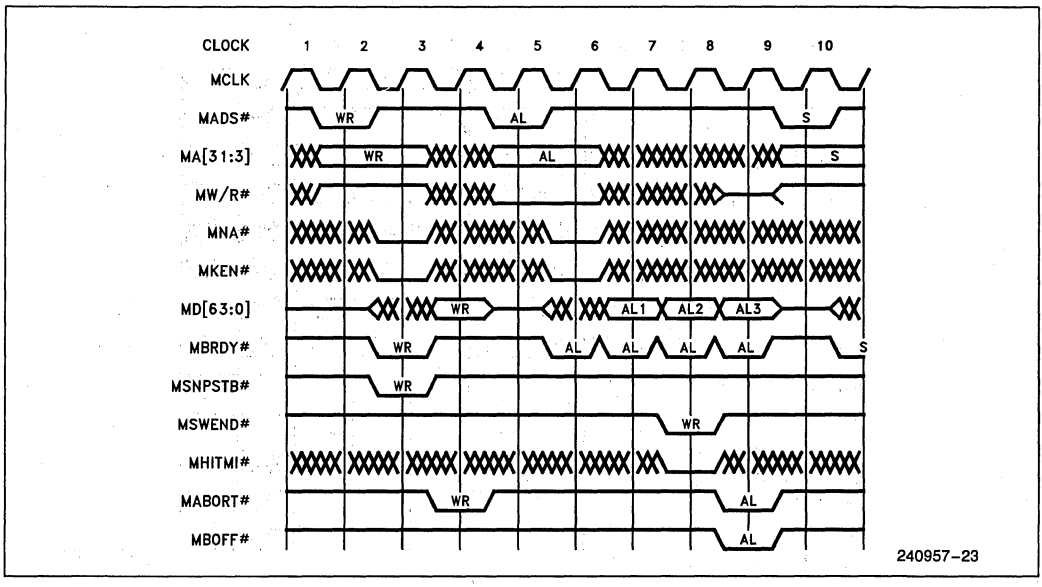


Figure C-5. Potentially Allocatable Write

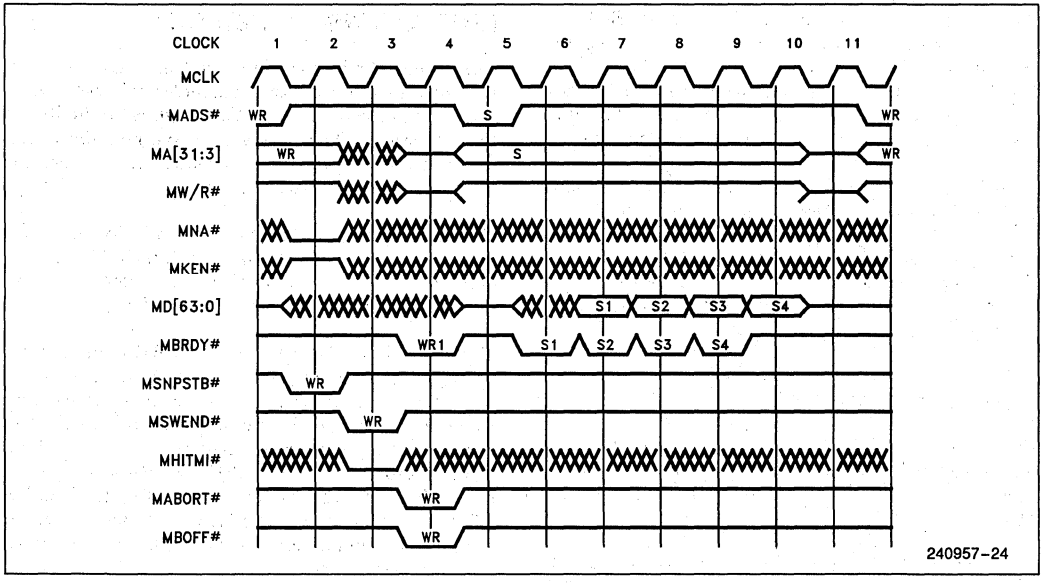


Figure C-6. Non-Allocatable Write

The MultiProcessor Interrupt Controller (MPIC) currently being designed by Intel is not utilized in or supported by this memory bus controller.

The memory bus controller's treatment of LOCKed cycles is simple but straightfoward: when the 82495 issues a memory access which is LOCKed (KLOCK# active), the MBC will not relinquish the bus until a cycle which is not LOCKed is issued. While this is adequate for simple systems, it will not suffice for dual ported memories, where a given block of memory can be accessed through more than one bus. In such systems, a LOCK signal must be introduced to alert all possible simultaneous users of memory that a LOCKed access is in progress.

EXTENSION OF DESIGN TO THREE OR MORE CPUs

Two Processor Implementation Overview

Figure C-7 presents a simplified view of the multiprocessing signals for the two processor implementation. The basic address, data, and memory cycle control lines are attached to a common bus. Only the core which controls the bus will drive these signals, with all other cores floating these lines and asserting MHLDA#.

When the bus master MBC issues a cycle, the MCACHE# and MW/R# cycle attributes also serve to drive the 82495s' SNPINV and SNPNCA inputs of both cores. SNPSTB# is issued by the master in the clock following MADS#. In reality, both cores have a SNPSTB# output at their Y-side state machines driving a common line which connects to the SNPSTB# input of both 82495s. The core which does not own the bus floats its state machine driver on MHLDA, so the signal acts only as an input in that core. The master drives the SNPSTB# line, but the action of SNPSTB# is blocked in its own 82495 because its MAOE# signal is asserted.

The results of the snoop are driven out on the snooping core's MTHIT# and MHITMO# outputs, and MSWENDO# is asserted. These signals are connected directly to the MHITMI#, MWB/WT#, and MSWENDI# inputs in the master core, respectively.

The MBOFF# signals of the two MBCs are also connected together. During MHLDA (in a snooping MBC) MBOFF# is an input, and in the master it is an output. If the master asserts MBOFF, control of the data and control busses is given to the snooping MBC so that a snoop write-back can be performed.

Three or More Processors

This section gives one method of extending the design given here to three or more processors. The solution presented here assumes that no changes are made to the state machines as they are written for the two processor system. Instead, some minor glue logic is added to three of the signals to make the core an element in a scalable multiprocessing system. However, modifying the state machines is also a plausible solution.

In an implementation with three or more processors, the primary address, data, and cycle control lines are still connected to a common bus, as in the two processor version. MCACHE# and MW/R# are also utilized in the same way as the two processor version: the outputs of the cores drive a common line which in turn also drives the 82495 SNPNCA and SNPINV inputs of all cores.

The SNPSTB# signal connects directly from core to core in a two processor version. In an implementation with three or more processors, the SNPSTB# line is simply extended to all the processors in the system. Only the bus master will actually drive the line, and snoopers will be floating the SNPSTB# output from their state machines. Again, the snoop request is ignored in the master because its MAOE# is asserted. Similarly, the MBOFF# signal becomes a common line which only the master will drive and which all other cores will sample.

The six signals in the upper portion of diagram C-7, which communicate MSWEND and the snoop results MHITMO# and MTHIT#, will require more glue logic to extend the design to three or more processors. The snoop results MHITMO# and MTHIT# must now be considered for multiple cores when a snoop has been issued, and the master MBC must not sample these results until all snooping cores have issued their MSWENDO#.

To resolve these issues, common bus lines carrying these signals are introduced, where all cores have outputs driving these lines, and inputs to sample them. The characteristics of such MTHIT# and MHITM# lines are straightforward: the line should default to 1, and if any core drives one of these outputs low, the line should be pulled low. The MTHIT# line has the simplest solution. As shown in figure C-8, by passing the signal which is produced by the core through an open collector buffer, the buffered MTHIT#s can be tied to a single line which is sampled directly by all cores' MWB/WT# pins. The open collector buffer sinks current like a normal gate output to drive a logic 0, but instead of driving current for a logic 1, the open collector device assumes a high impedance state for logic 1. Thus, if all of the cores outputs MTHIT# as 1, the MTHIT# line remains at a logic 1 level because of the pull-up resistor. If one or more cores outputs a logic 0, the MTHIT# line will be pulled to the logic 0 level. This precisely matches the desired behavior of MTHIT# for the system: if any 1 or more core(s) has the snooped data cached, the master MWB/WT# input must be asserted low. It is important to note that

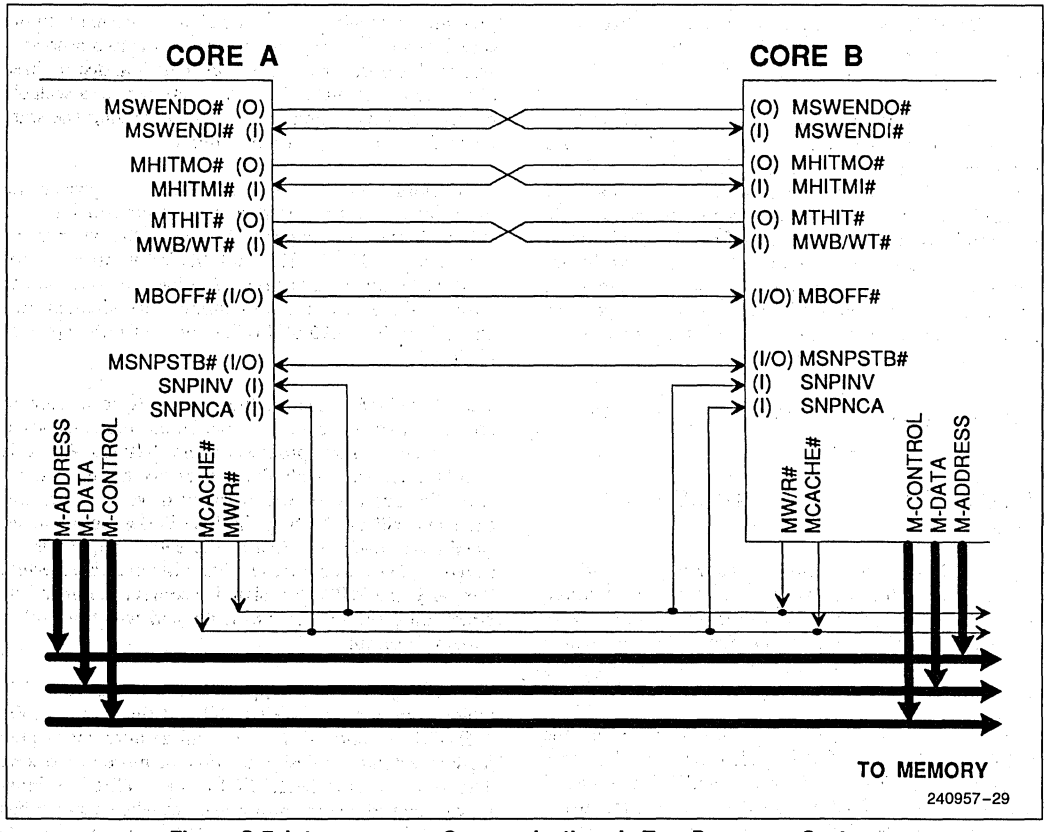


Figure C-7. Interprocessor Communications in Two Processor System

the MTHIT# output of the master is floated: because the 82495 MTHIT# output only changes on each new snoop, the value of the master MTHIT# output for the previous snoop would erroneously be included in deciding the level of the MTHIT# line.

The MHITM# line follows the same principle as the MTHIT# line. The MHITM# signal is not floated in the master core, and poses the problem which floating MTHIT# avoids: the value of the master's last MHITMO# output is still present when the new access is being made. To resolve this, the inverted value of MHLDA is ORed with MHITMO# before going to the open collector buffer. The master's MHLDA is always a 0, so the OR gate will always guarantee a 1 being passed from the master to the MHITM# line. Again, if one or more of the snooping MBCs outputs a logic 0, the MHITM# line will properly assume a 0 level.

The open collector buffer presents an easy way to add new MBCs to the shared lines. The desired behavior of a shared MSWENDA (MSWEND All) line is different from the attribute lines, MTHIT# and MHITM#. Where the master core should sample a 0 if any one or more snooping core(s) drives a 0 on these attribute lines, the master core must not receive its MSWENDI# indication until all cores in the system have asserted their MSWENDO# output. The answer is to invert the MSWENDO output of each snooper, so that a zero is driven onto the MSWENDA line when the snoop is being performed, and a one is output if the snoop has completed. From the MSWENDI# perspective, MSWENDI# should not be asserted at the master core if any snooping core is still driving a zero on the MSWENDA line (is not done snooping). Therefore, the MSWENDA line is the opposite logic polarity of the actual MSWENDO# signal. The master samples MSWENDA after the signal passes through an inverter, to recorrect the logic level. The output of each core is passed through inverter before going to the open collector buffer. The inverting device is a NAND gate because the SWENDO# signal shares the problem of MHITM#, and must be "faked" by the master. In this case, instead of the last snoop's results causing the problem, the master's SWENDO# signal is reset to 1 (still snooping) when the SNPSTB# line is asserted.

Again, these simple adaptations can be implemented in a similar manner in the logic of the state machines. The MHITMO# line can be forced to a logic one or floated when the core is a master (after YBGT, for example). The MSWEND signal might be implemented as an asserted-high system signal, if open collector buffers are used to attach new cores to the shared system bus.

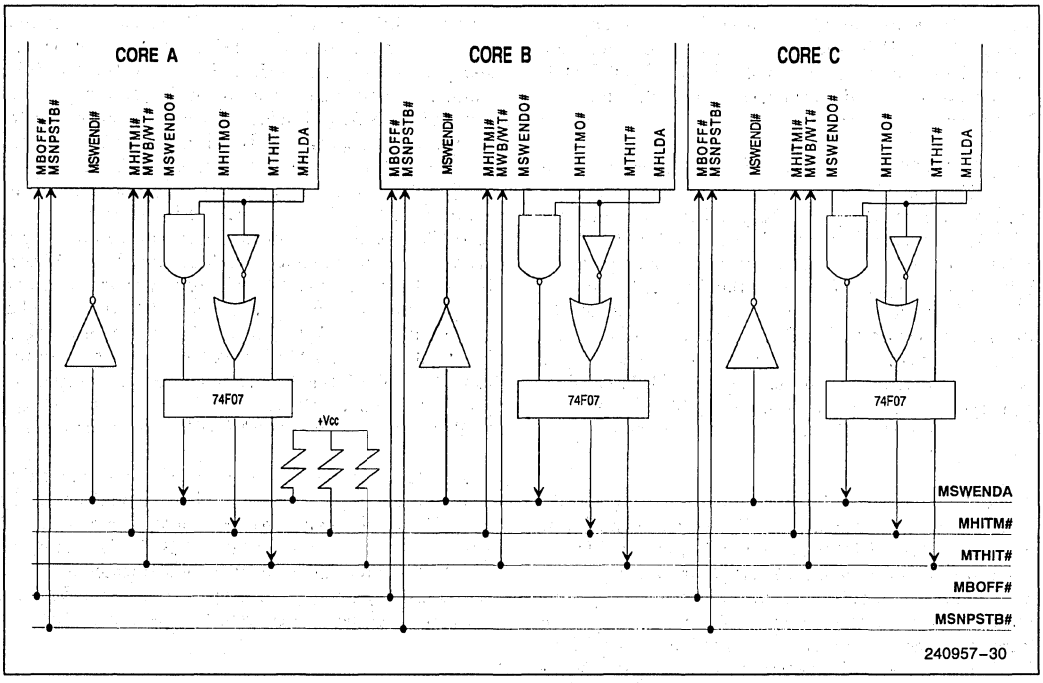
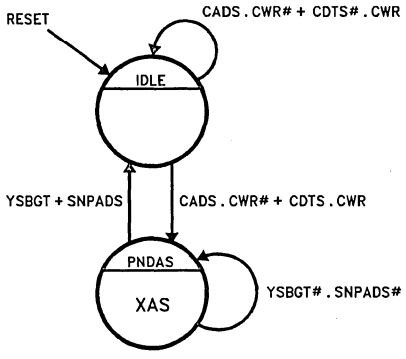


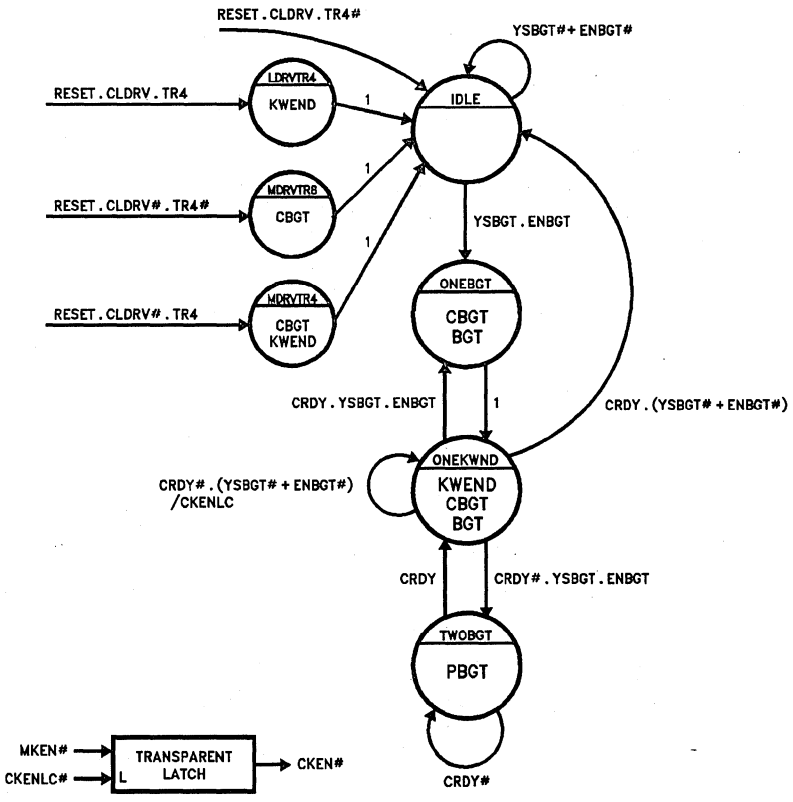
Figure C-8. Extension Glue

STATE MACHINES AND SCHEMATICS

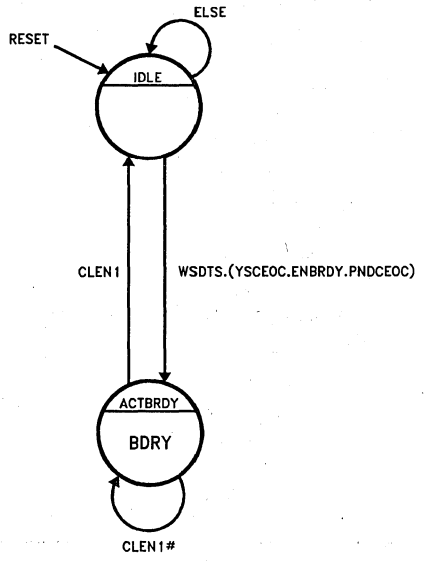
STATE DIAGRAMS



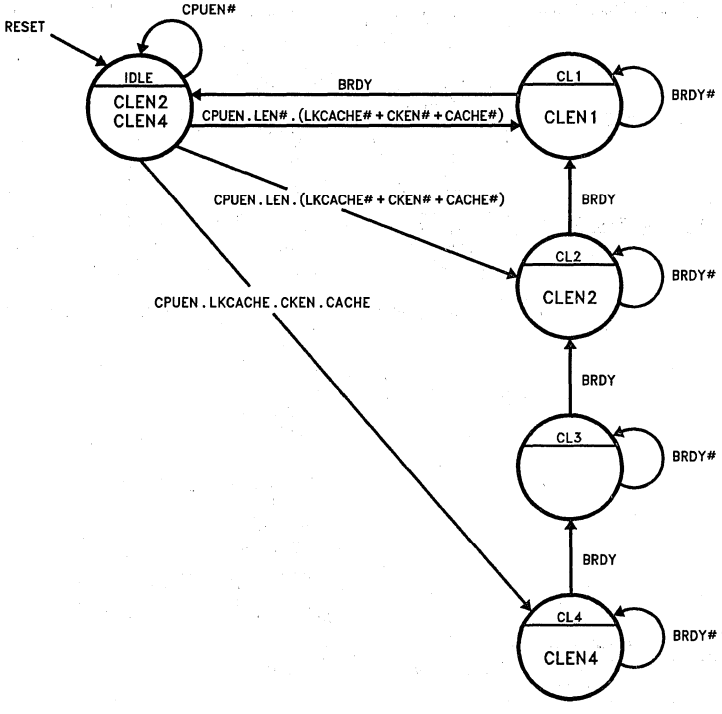
XASTB



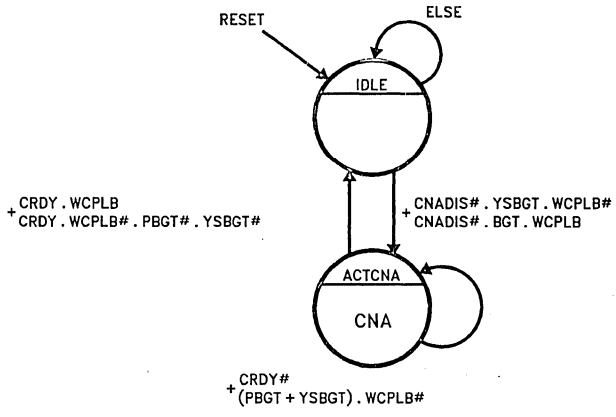
XBGTKWND


ELSE
RESET
IDLE
CLEN1
WSDTS.(YSCEOC.ENBRDY.PNDCEOC)
ACTBRDY
BDRY
CLEN1#
240957-33

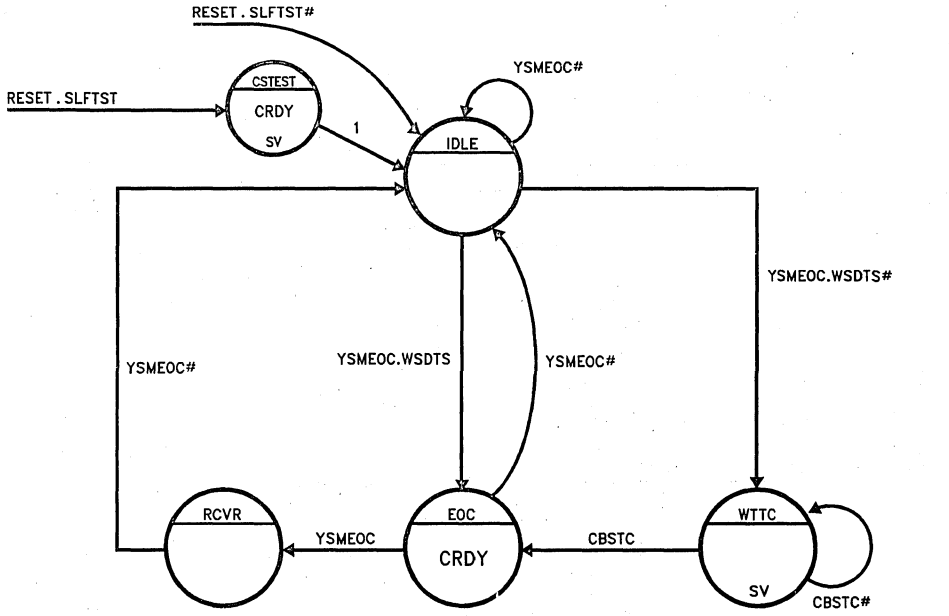
XBRDY


CPUEN#
RESET
IDLE
CLEN2
CLEN4
BRDY
CPUEN.LEN#.(LKCACHE# + CKEN# + CACHE#)
CL1
CLEN1
BRDY#
CPUEN.LEN.(LKCACHE# + CKEN# + CACHE#)
BRDY
CL2
CLEN2
BRDY#
CPUEN.LKCACHE.CKEN.CACHE
BRDY
CL3
BRDY#
BRDY
CL4
CLEN4
BRDY#
240957-34

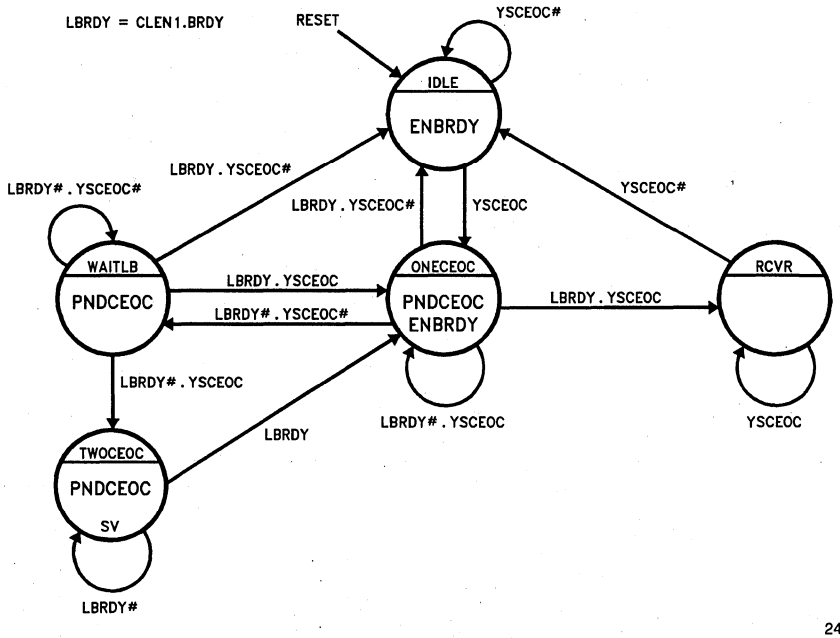
XCLEN


RESET
ELSE
IDLE
+ CRDY . WCPLB
+ CRDY . WCPLB# . PBGT# . YSBGT#
+ CNADIS# . YSBGT . WCPLB#
+ CNADIS# . BGT . WCPLB
ACTCNA
CNA
+ CRDY#
+ (PBGT + YSBGT) . WCPLB#
240957-35

XCNA

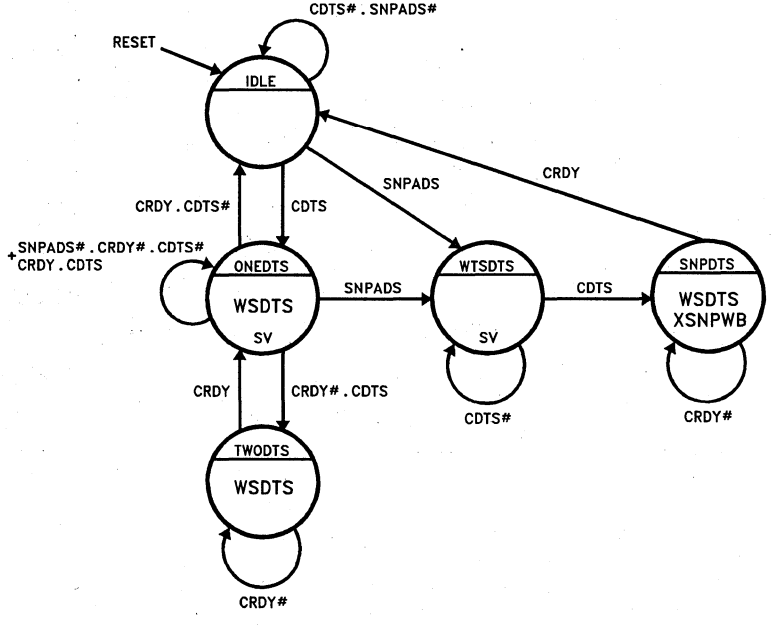

RESET . SLFTST#
YSMEOC#
RESET . SLFTST
CSTEST
CRDY
SV
1
IDLE
YSMEOC.WSDTS#
YSMEOC#
YSMEOC.WSDTS
YSMEOC#
RCVR
YSMEOC
EOC
CRDY
CBSTC
WTTC
SV
CBSTC#
240957-36

XCRDY



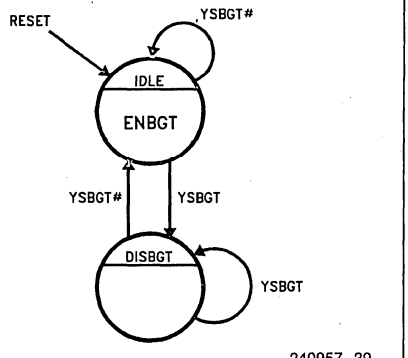
240957-37

XCTRCK



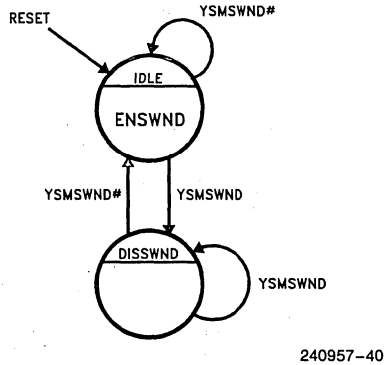
240957-38

XDTSTRCK



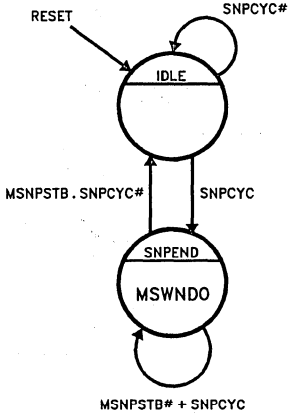
240957-39

XENBGT



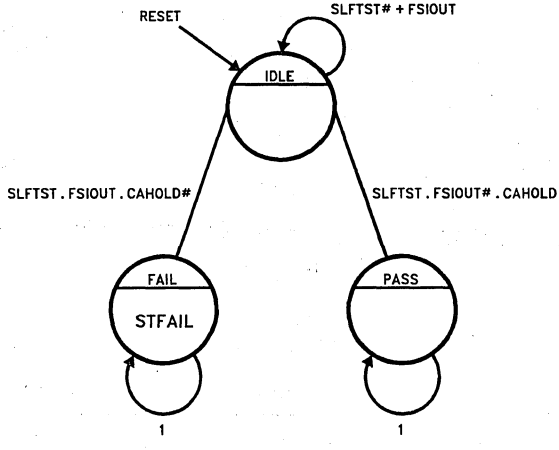
240957-40

XENSWND



240957-41

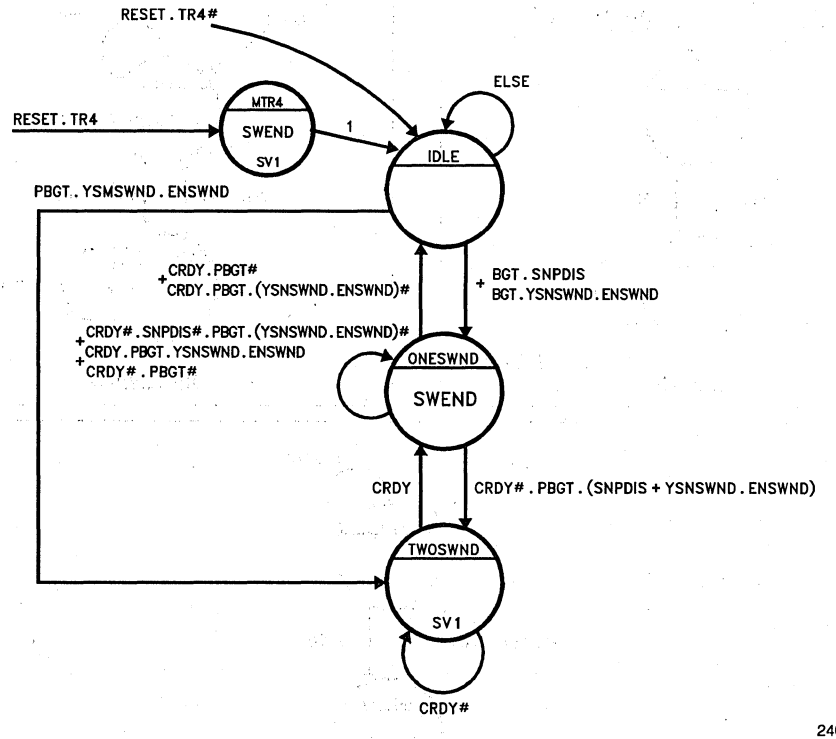
XMSWND0



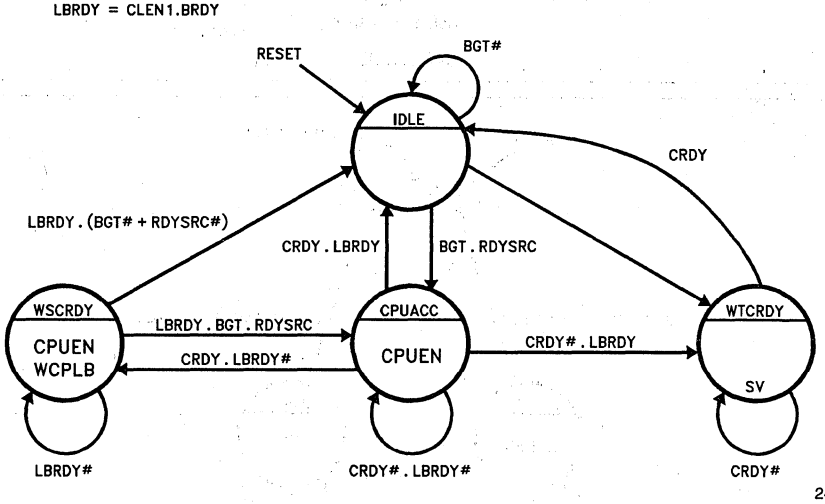
240957-42

XSTFAIL

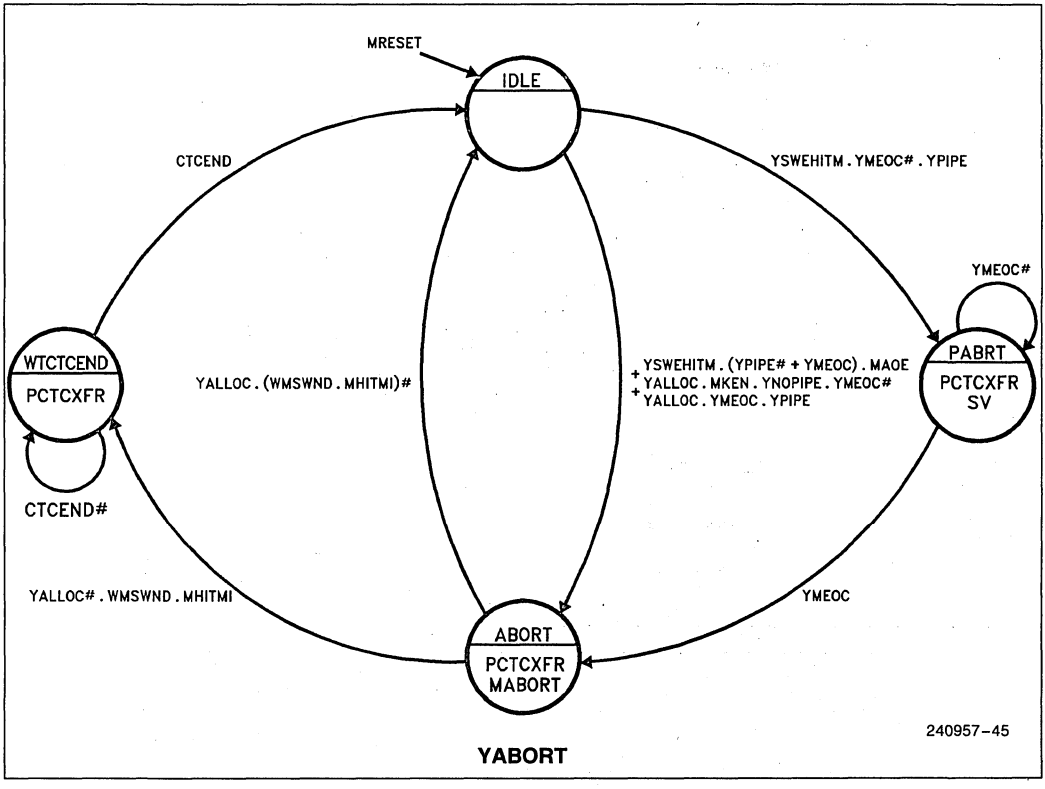
2



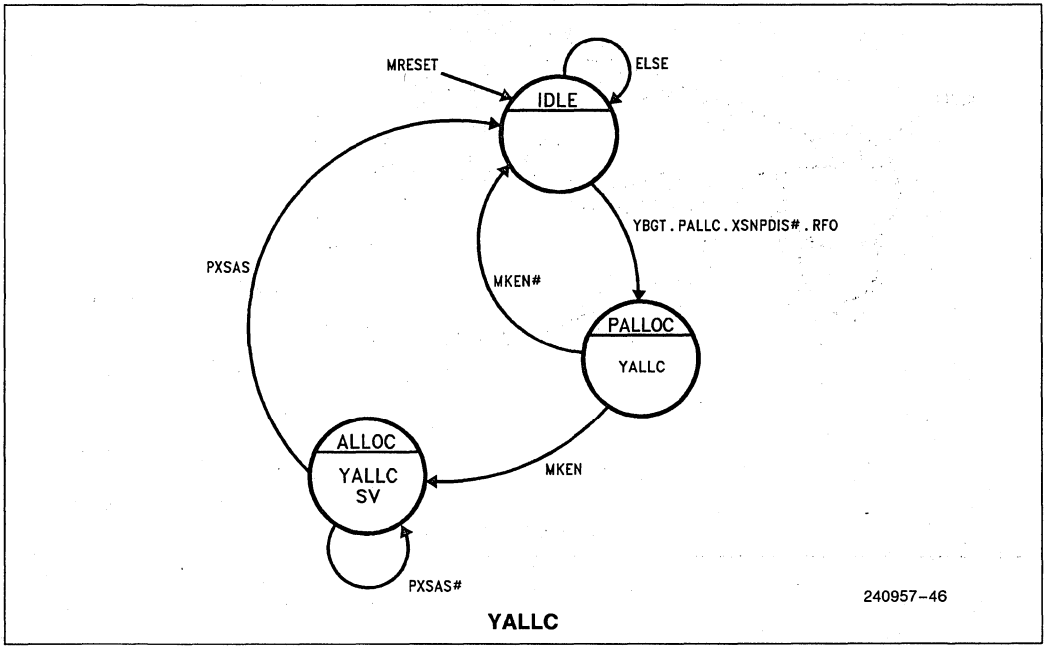
XSWIND



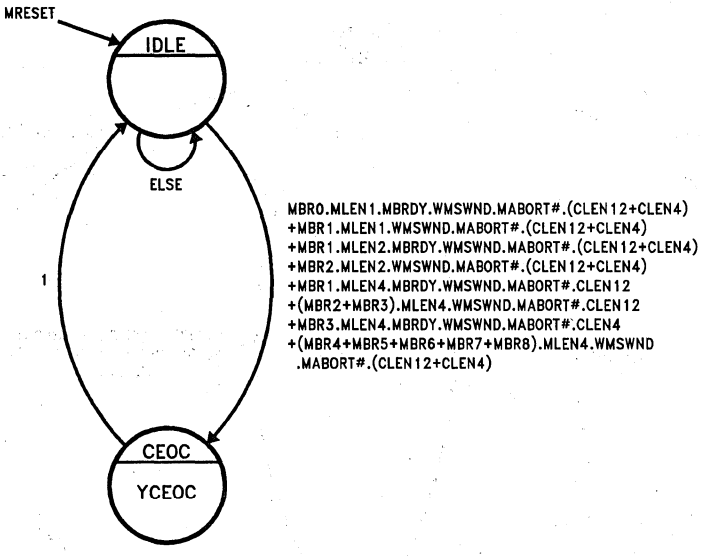
XWCPLB



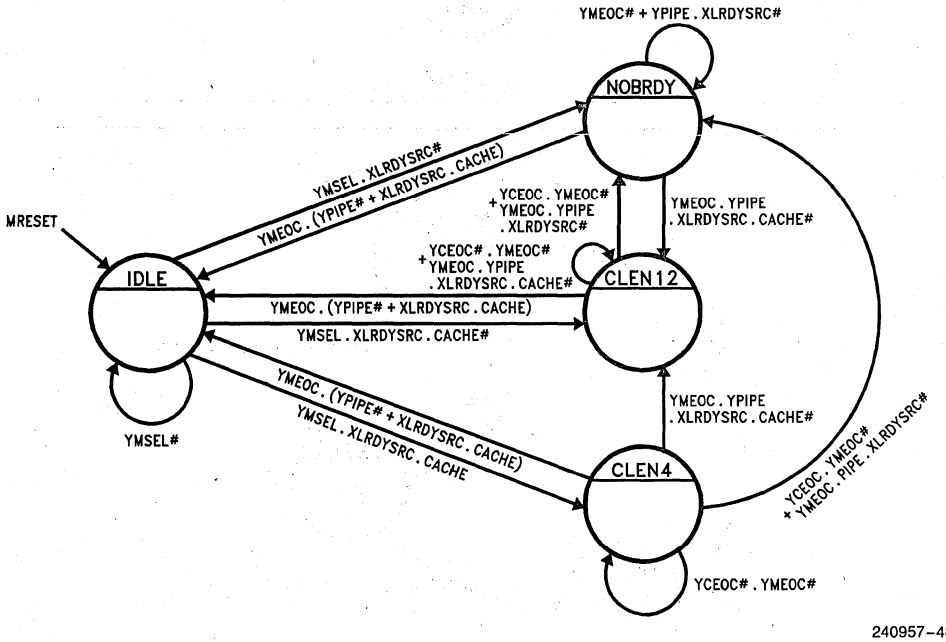
YABORT



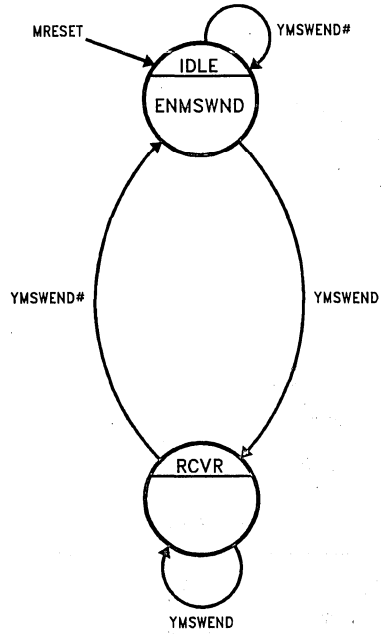
YALLC



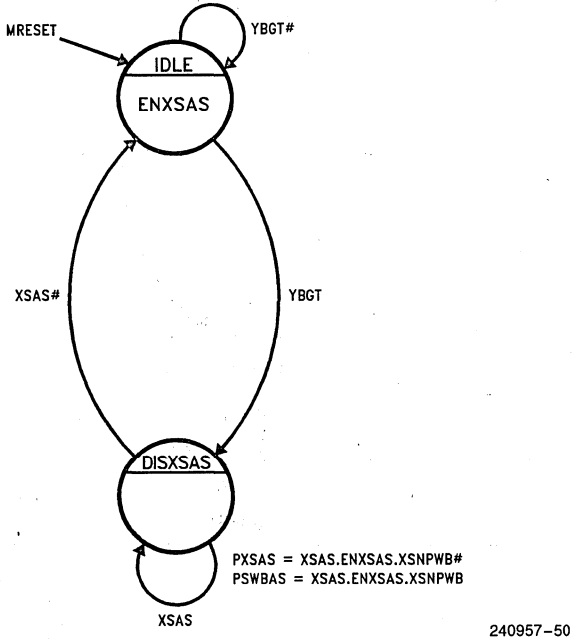
YCPUEOC



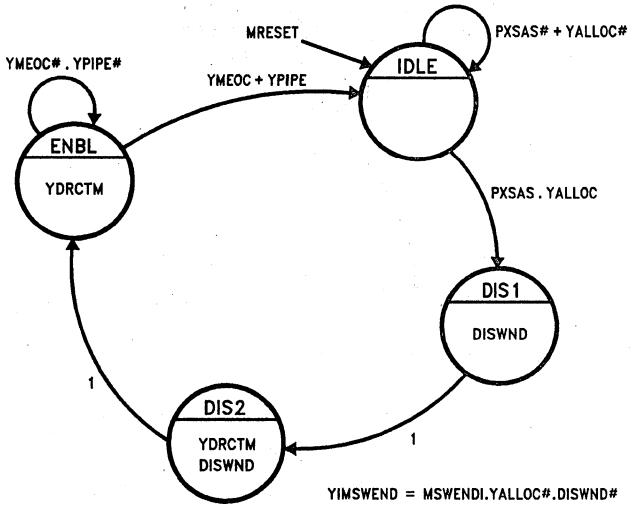
YCPULEN



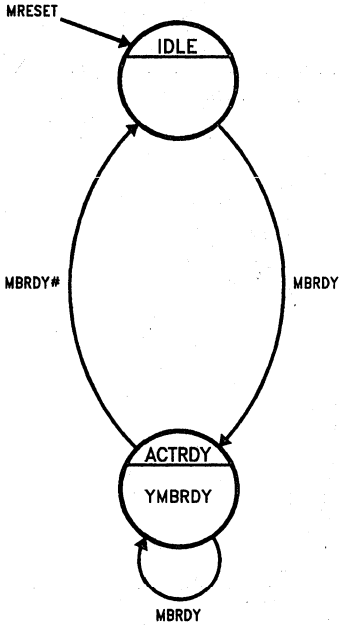
YENMSWND



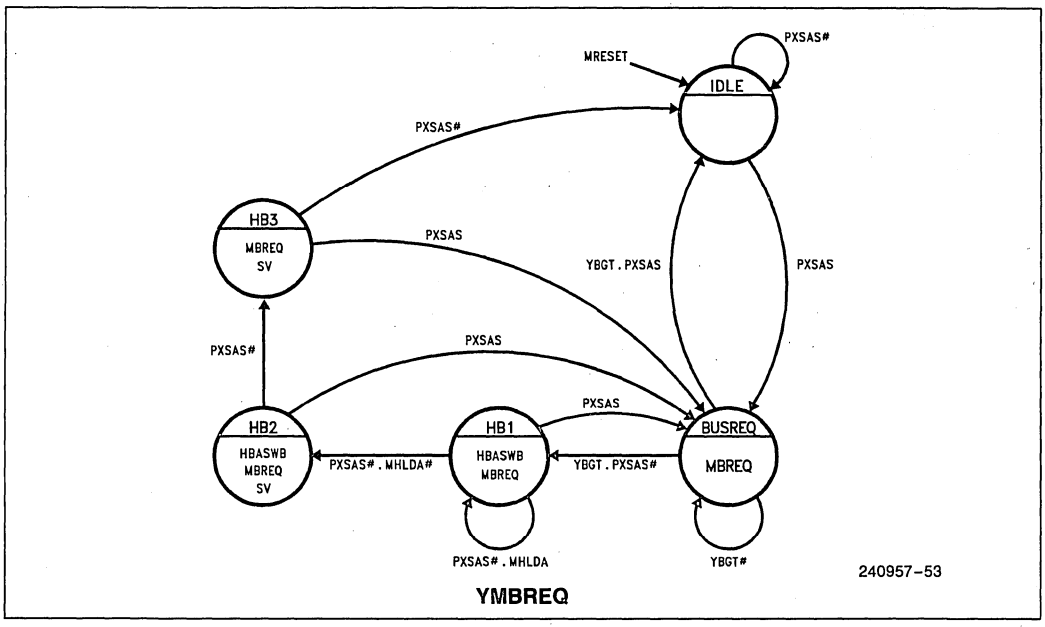
YENXSAS



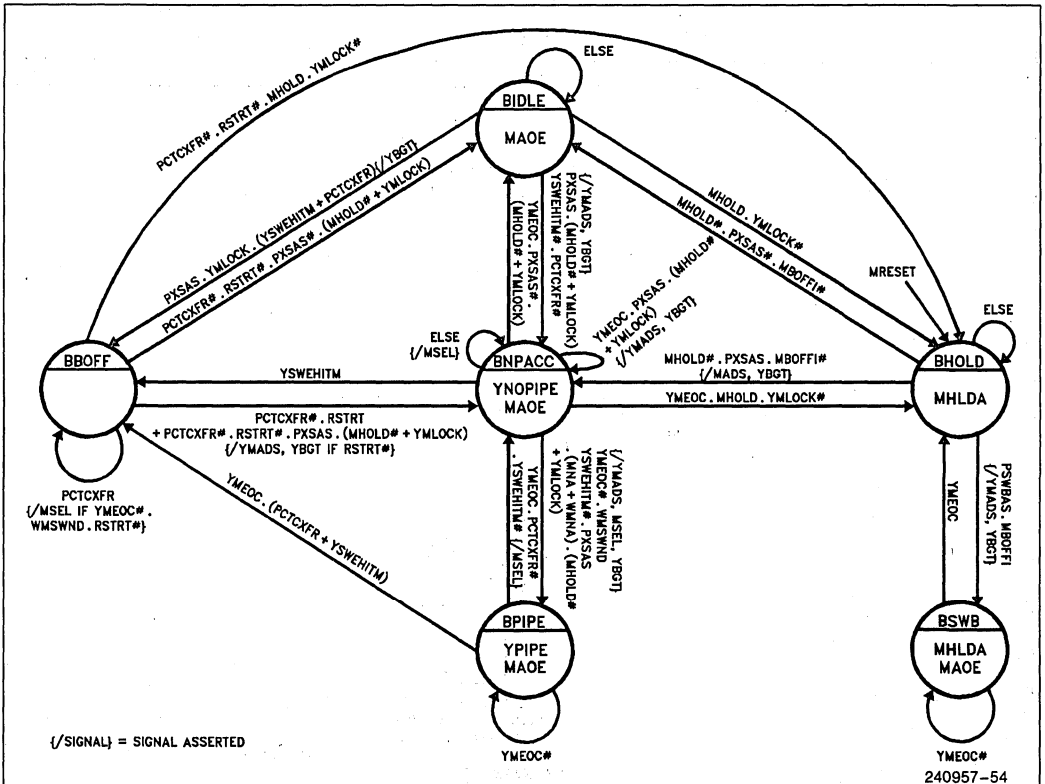
YIMSWND



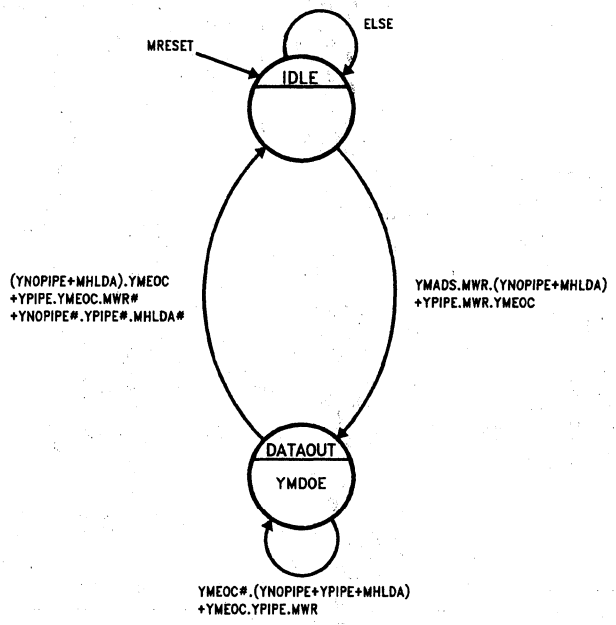
YMBRDY



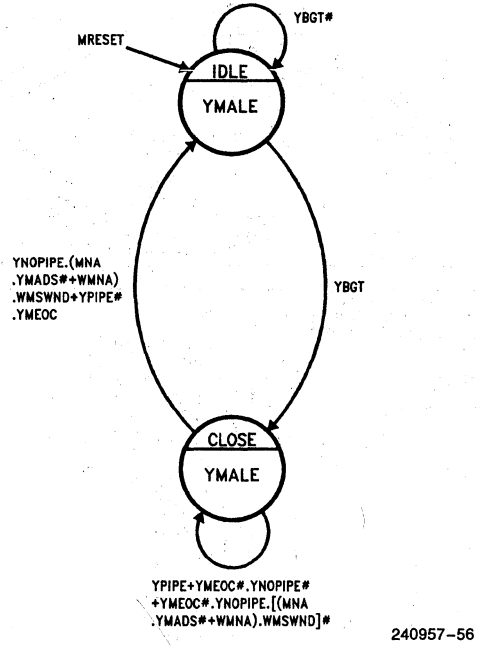
YMBREQ



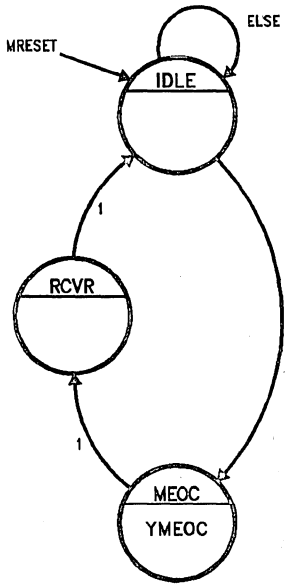
YMBTRCK



YMDOE



YMEMALE

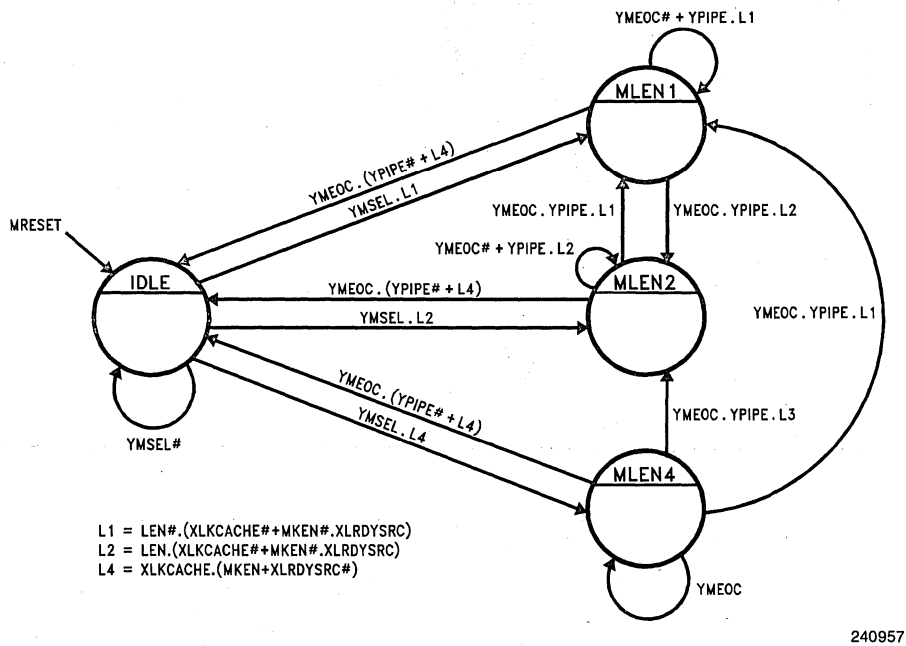


MBR0.MLEN1.MBRDY.WMSWND.MABORT#
+MBR1.MLEN1.WMSWND.MABORT#
+MBR1.MLEN2.MBRDY.WMSWND.MABORT#
+MBR2.MLEN2.WMSWND.MABORT#
+MBR3.MLEN4.MBRDY.WMSWND.MABORT#.TR4
+MBR4.MLEN4.WMSWND.MABORT#.TR4
+MBR7.MLEN4.MBRDY.WMSWND.MABORT#
+MBR8.MLEN4.WMSWND.MABORT#
+YALLOC.MABORT

{/YMFRZ IF YALLOC.MABORT}

240957-57

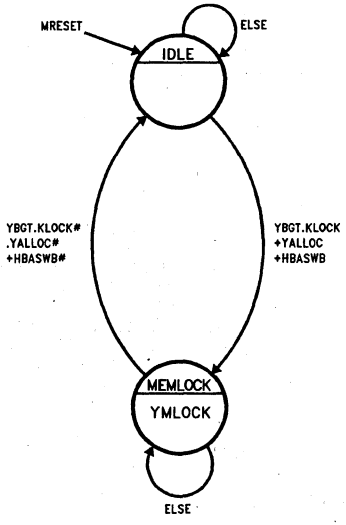
YMEMEOC



L1 = LEN#.(XLKCACHE#+MKEN#.XLRDYSRC)
L2 = LEN.(XLKCACHE#+MKEN#.XLRDYSRC)
L4 = XLKCACHE.(MKEN+XLRDYSRC#)

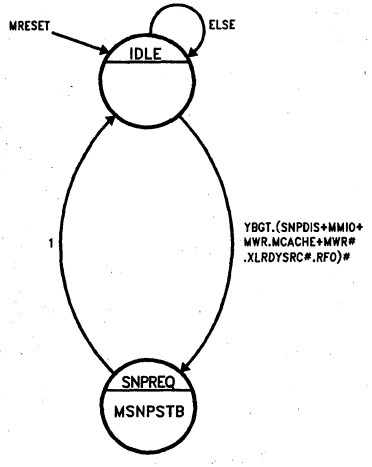
240957-58

YMEMLEN



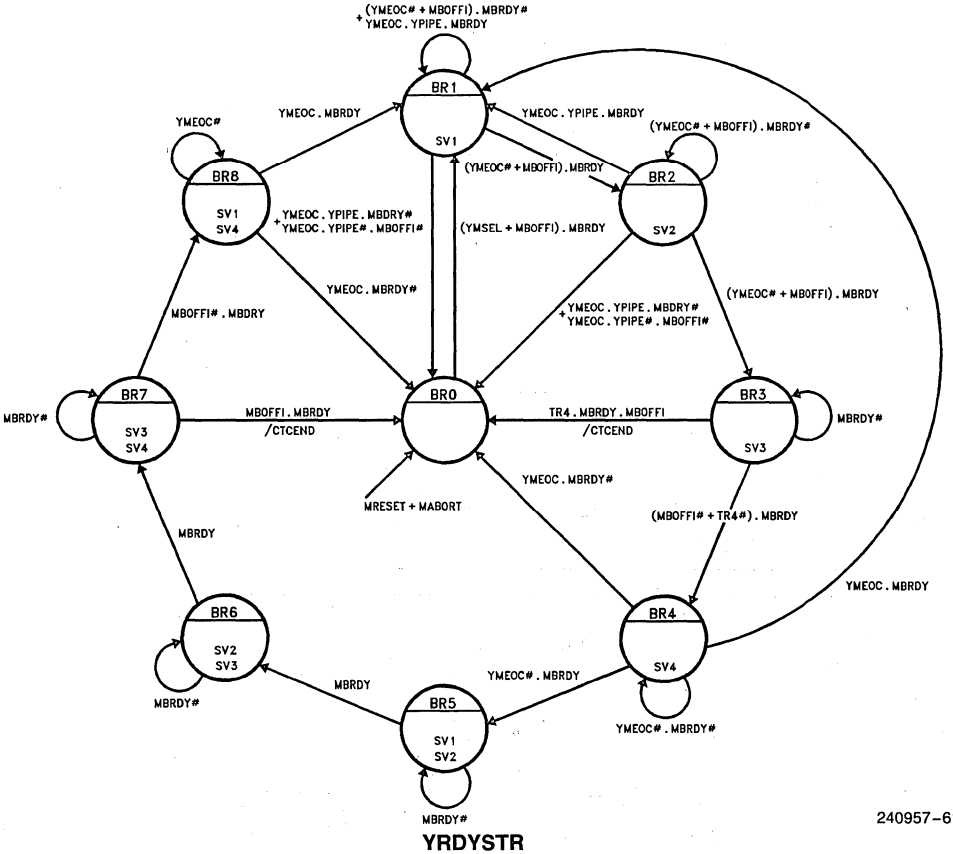
240957-59

YMEMLOCK



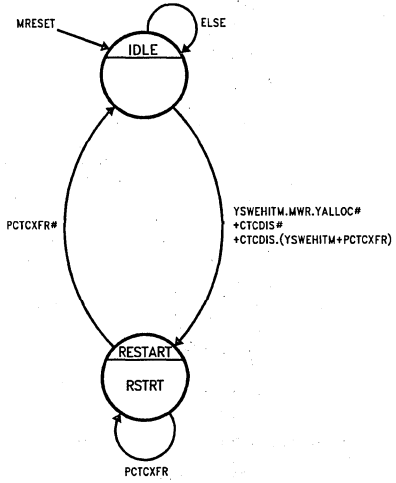
240957-60

YMSNPSTB



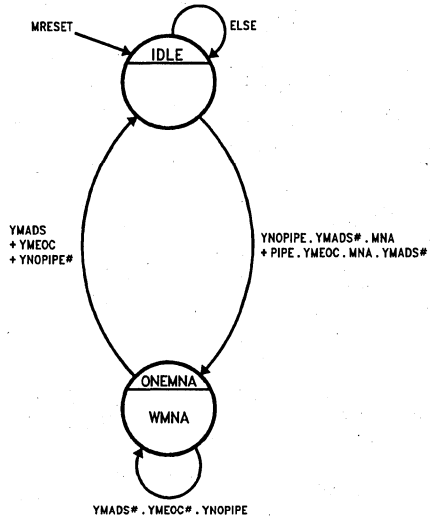
240957-61

YRDYSTR



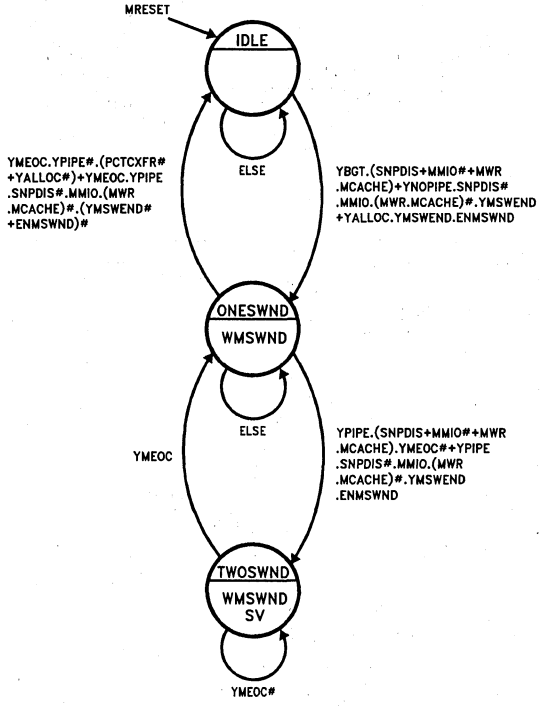
240957-62

YRSTRT



240957–63

YWMNA



YSWEHITMI = YMSWEND.MHITMI.ENMSWND.(YALLOC#+YNOPIPE#.YPIPE#)

240957–64

YWMSWND

PLD CODES

```
;--------------------------------- Declaration Segment ------------
TITLE AYMBTRCK
PATTERN  A
REVISION 2.0
AUTHOR   ISIC SILAS
COMPANY  INTEL
DATE     2/4/91
CHIP  x01  85C22v10
;
;       This PLD contains the YMBTRCK state machine.
;
;--------------------------------- PIN Declarations --------------
PIN  1          MCLK       COMBINATORIAL ;
PIN  2          MRESET      COMBINATORIAL ;
PIN  3          /WMSWND      COMBINATORIAL ;
PIN  4          /MBOFFI      COMBINATORIAL ;
PIN  5          /PXSAS      COMBINATORIAL ;
PIN  6          /PSWBAS      COMBINATORIAL ;
PIN  7          MHOLD      COMBINATORIAL ;
PIN  8          /MNA       COMBINATORIAL ;
PIN  9          /WMNA       COMBINATORIAL ;
PIN  10         /YMLOCK      COMBINATORIAL ;
PIN  11         /YSWEHITM     COMBINATORIAL ;
PIN  12         GND
PIN  13         /PCTCXFR      COMBINATORIAL ;
PIN  14         /RSTRT      COMBINATORIAL ;
PIN  15         /YMEOC      COMBINATORIAL ;
PIN  16         UNUSED     registered ;
PIN  17         /YBGT      registered ;
PIN  18         /YMADS      registered ;
PIN  19         /MAOE      registered ;
PIN  20         /YNOPIPE      registered ;
PIN  21         /YMSTR      registered ;
PIN  22         /YPIPE     registered ;
PIN  23         /YMSEL      registered ;
PIN  24         VCC
;--------------------------------- Boolean Equation Segment ------
EQUATIONS

YNOPIPE     :=  /MRESET * PXSAS * /MHOLD * YMEOC * YNOPIPE
            + /MRESET * PXSAS * YMLOCK * YMEOC * YNOPIPE
            + /MRESET * /PXSAS * /YMEOC * /YSWEHITM * YNOPIPE
            + /MRESET * /YMEOC * /WMSWND * /YSWEHITM * YNOPIPE
            + /MRESET * YMEOC * /YSWEHITM * /PCTCXFR * YPIPE
            + /MRESET * /PCTCXFR * RSTRT * YMSTR * /MAOE
            + /MRESET * /MNA * /WMNA * /YMEOC * /YSWEHITM * YNOPIPE
            + /MRESET * MHOLD * /YMLOCK * /YMEOC * /YSWEHITM * YNOPIPE
            + /MRESET * PXSAS * /MHOLD * /PCTCXFR * YMSTR * /MAOE
            + /MRESET * PXSAS * YMLOCK * /PCTCXFR * YMSTR * /MAOE
            + /MRESET * PXSAS * /MHOLD * /MBOFFI * /YMSTR * /MAOE
            + /MRESET * PXSAS * /MHOLD * /YSWEHITM * /PCTCXFR * /YNOPIPE * /YPIPE
           * YMSTR
            + /MRESET * PXSAS * YMLOCK * /YSWEHITM * /PCTCXFR * /YNOPIPE * /YPIPE
           * YMSTR

YPIPE       :=  /MRESET * /YMEOC * YPIPE
            + /MRESET * PXSAS * /MHOLD * MNA * /YMEOC * WMSWND * /YSWEHITM
           * YNOPIPE
            + /MRESET * PXSAS * /MHOLD * WMNA * /YMEOC * WMSWND * /YSWEHITM
           * YNOPIPE
            + /MRESET * PXSAS * MNA * YMLOCK * /YMEOC * WMSWND * /YSWEHITM
           * YNOPIPE
            + /MRESET * PXSAS * WMNA * YMLOCK * /YMEOC * WMSWND * /YSWEHITM
           * YNOPIPE

YMSTR       :=  /MRESET * YPIPE
```

240957-65

```
          + /MRESET * /YMEOC * YNOPIPE
          + /MRESET * YSWEHITM * YNOPIPE
          + /MRESET * /MHOLD * YMSTR
          + /MRESET * YMLOCK * YMSTR
          + /MRESET * /MHOLD * /MBOFFI * /MAOE
          + /MRESET * PCTCXFR * YMSTR * /MAOE
          + /MRESET * RSTRT * YMSTR * /MAOE

MAOE      :=  /MRESET * /PCTCXFR * RSTRT * YMSTR * /MAOE
          + /MRESET * /YMEOC * YPIPE
          + /MRESET * YMLOCK * YMEOC * YNOPIPE
          + /MRESET * /YMEOC * /YSWEHITM * YNOPIPE
          + /MRESET * /YSWEHITM * /PCTCXFR * YPIPE
          + /MRESET * /YMEOC * /YMSTR * MAOE
          + /MRESET * YMLOCK * /YSWEHITM * /PCTCXFR * YMSTR
          + /MRESET * /MHOLD * /PCTCXFR * YMSTR * /MAOE
          + /MRESET * YMLOCK * /PCTCXFR * YMSTR * /MAOE
          + /MRESET * /MHOLD * /MBOFFI * /YMSTR * /MAOE
          + /MRESET * PSWBAS * MBOFFI * /YMSTR * /MAOE
          + /MRESET * /MHOLD * /YMLOCK * /YMEOC * /YNOPIPE * MAOE
          + /MRESET * /MHOLD * /YMLOCK * YMEOC * /YPIPE * YMSTR * MAOE
          + /MRESET * /PXSAS * YMLOCK * /YNOPIPE * /YPIPE * YMSTR * MAOE

YMADS     :=  /MRESET * PXSAS * /MHOLD * YMEOC * YNOPIPE
          + /MRESET * PXSAS * YMLOCK * YMEOC * YNOPIPE
          + /MRESET * /PCTCXFR * RSTRT * YMSTR * /MAOE
          + /MRESET * PXSAS * /MHOLD * /PCTCXFR * YMSTR * /MAOE
          + /MRESET * PXSAS * YMLOCK * /PCTCXFR * YMSTR * /MAOE
          + /MRESET * PXSAS * /MHOLD * /MBOFFI * /YMSTR * /MAOE
          + /MRESET * PXSAS * /MHOLD * /YSWEHITM * /PCTCXFR * /YNOPIPE * /YPIPE
        * YMSTR
          + /MRESET * PXSAS * YMLOCK * /YSWEHITM * /PCTCXFR * /YNOPIPE * /YPIPE
        * YMSTR
          + /MRESET * PSWBAS * MBOFFI * /YMSTR * /MAOE
          + /MRESET * PXSAS * /MHOLD * MNA * WMSWND * /YSWEHITM * YNOPIPE
          + /MRESET * PXSAS * /MHOLD * WMNA * WMSWND * /YSWEHITM * YNOPIPE
          + /MRESET * PXSAS * MNA * YMLOCK * WMSWND * /YSWEHITM * YNOPIPE
          + /MRESET * PXSAS * WMNA * YMLOCK * WMSWND * /YSWEHITM * YNOPIPE

YBGT      :=  /MRESET * PXSAS * /MHOLD * YMEOC * YNOPIPE
          + /MRESET * PXSAS * YMLOCK * YMEOC * YNOPIPE
          + /MRESET * PXSAS * /MHOLD * /MBOFFI * /YMSTR * /MAOE
          + /MRESET * PSWBAS * MBOFFI * /YMSTR * /MAOE
          + /MRESET * PXSAS * /MHOLD * MNA * WMSWND * /YSWEHITM * YNOPIPE
          + /MRESET * PXSAS * /MHOLD * WMNA * WMSWND * /YSWEHITM * YNOPIPE
          + /MRESET * PXSAS * MNA * YMLOCK * WMSWND * /YSWEHITM * YNOPIPE
          + /MRESET * PXSAS * WMNA * YMLOCK * WMSWND * /YSWEHITM * YNOPIPE
          + /MRESET * PXSAS * YMLOCK * /YNOPIPE * /YPIPE * YMSTR * MAOE
          + /MRESET * PXSAS * /MHOLD * /PCTCXFR * /RSTRT * YMSTR * /MAOE
          + /MRESET * PXSAS * YMLOCK * /PCTCXFR * /RSTRT * YMSTR * /MAOE
          + /MRESET * PXSAS * /MHOLD * /YSWEHITM * /PCTCXFR * /YNOPIPE * /YPIPE
        * YMSTR * MAOE

YMSEL     :=  /MRESET * /YMEOC * YPIPE
          + /MRESET * /YMEOC * /YSWEHITM * YNOPIPE
          + /MRESET * /YSWEHITM * /PCTCXFR * YPIPE
          + /MRESET * /YMEOC * WMSWND * PCTCXFR * /RSTRT * YMSTR * /MAOE


UNUSED    := VCC
```

240957-66

```
;--------------------------------- Declaration Segment ------------
TITLE AYMEMLEN
PATTERN  A
REVISION 2.0
AUTHOR   ISIC SILAS
COMPANY  INTEL
DATE     2/4/91
CHIP  x01  85C22V10
;
;       This PLD contains the YMEMLEN and YCPUEOC state machines
;
;--------------------------------- PIN Declarations ---------------
PIN  1          MCLK       COMBINATORIAL ;
PIN  2          MRESET       COMBINATORIAL ;
PIN  3          /YMSEL       COMBINATORIAL ;
PIN  4          /YPIPE       COMBINATORIAL ;
PIN  5          /MABORT       COMBINATORIAL ;
PIN  6          /MBRDY       COMBINATORIAL ;
PIN  7          /WMSWND       COMBINATORIAL ;
PIN  8          XLRDYSRC       COMBINATORIAL ;
PIN  9          /XLKCACHE       COMBINATORIAL ;
PIN  10         /MKEN      COMBINATORIAL ;
PIN  11         LEN        COMBINATORIAL ;
PIN  12         GND                      ;
PIN  13         /CACHE       COMBINATORIAL ; INPUT
PIN  14         /YMEOC        COMBINATORIAL ; INPUT
PIN  15         /SVR0        COMBINATORIAL ; INPUT
PIN  16         /SVR1        COMBINATORIAL ; INPUT
PIN  17         /SVR2        COMBINATORIAL ; INPUT
PIN  18         /SVR3        COMBINATORIAL ; INPUT
PIN  19         /YCEOC       registered ;
PIN  20         /SVL0       registered ;
PIN  21         /SVL1      registered ;
PIN  22         /SVC0       registered ;
PIN  23         /SVC1       registered ;
PIN  24         VCC
;--------------------------------Boolean Equation Segment ------
EQUATIONS

SVL1      :=  /YMEOC * /MRESET * SVL1
            + YPIPE * LEN * /XLKCACHE * /MRESET * SVL1
            + YMSEL * LEN * /MRESET * /SVL1 * /SVL0
            + YPIPE * LEN * XLRDYSRC * /MKEN * /MRESET * SVL1
            + YPIPE * YMEOC * LEN * /XLKCACHE * /MRESET * SVL0
            + YMSEL * /XLRDYSRC * XLKCACHE * /MRESET * /SVL1 * /SVL0
            + YMSEL * XLKCACHE * MKEN * /MRESET * /SVL1 * /SVL0
            + YPIPE * YMEOC * LEN * XLRDYSRC * /MKEN * /MRESET * SVL0

SVL0      :=  YMSEL * /XLRDYSRC * XLKCACHE * /MRESET * /SVL1 * /SVL0
            + YMSEL * XLKCACHE * MKEN * /MRESET * /SVL1 * /SVL0
            + /YMEOC * /MRESET * SVL0
            + YPIPE * /LEN * /XLKCACHE * /MRESET * SVL0
            + YMSEL * /LEN * /MRESET * /SVL1 * /SVL0
            + YPIPE * YMEOC * /LEN * /XLKCACHE * /MRESET * SVL1
            + YPIPE * /LEN * XLRDYSRC * /MKEN * /MRESET * SVL0
            + YPIPE * YMEOC * /LEN * XLRDYSRC * /MKEN * /MRESET * SVL1

SVC1      :=  /YMEOC * /YCEOC * /MRESET * SVC1
            + YMSEL * XLRDYSRC * /MRESET * /SVC1 * /SVC0
            + YPIPE * YMEOC * /CACHE * XLRDYSRC * /MRESET * SVC1
            + YPIPE * YMEOC * /CACHE * XLRDYSRC * /MRESET * SVC0

SVC0      :=  /YMEOC * /MRESET * SVC0
            + /YMEOC * YCEOC * /MRESET * SVC1
            + YPIPE * /XLRDYSRC * /MRESET * SVC0
```

240957-67

```
          + YPIPE * YMEOC * /XLRDYSRC * /MRESET * SVC1
          + YMSEL * CACHE * /MRESET * /SVC1 * /SVC0
          + YMSEL * /XLRDYSRC * /MRESET * /SVC1 * /SVC0

YCEOC     :=  SVR3 * /SVR2 * /SVR1 * SVL1 * SVL0 * SVC1 * WMSWND
         * /MABORT * /MRESET * /YCEOC
          + SVR3 * /SVR1 * /SVR0 * SVL1 * SVL0 * SVC1 * WMSWND
         * /MABORT * /MRESET * /YCEOC
          + /SVR3 * /SVR2 * SVR1 * /SVR0 * SVL1 * /SVL0 * SVC1
         * WMSWND * /MABORT * /MRESET * /YCEOC
          + /SVR3 * /SVR2 * /SVR1 * SVR0 * /SVL1 * SVL0 * SVC1
         * WMSWND * /MABORT * /MRESET * /YCEOC
          + /SVR3 * SVR2 * SVR1 * /SVR0 * SVL1 * SVL0 * SVC1
         * WMSWND * /MABORT * /MRESET * /YCEOC
          + /SVR3 * /SVR2 * SVR1 * SVR0 * SVL1 * SVL0 * SVC1
         * WMSWND * /MABORT * /MRESET * /YCEOC
          + /SVR3 * /SVR2 * SVR1 * /SVR0 * SVL1 * SVC1 * /SVC0
         * WMSWND * /MABORT * /MRESET * /YCEOC
          + /SVR3 * SVR2 * /SVR0 * SVL1 * SVL0 * SVC1 * /SVC0
         * WMSWND * /MABORT * /MRESET * /YCEOC
          + /SVR3 * /SVR2 * /SVR1 * /SVL1 * SVL0 * SVC1 * MBRDY
         * WMSWND * /MABORT * /MRESET * /YCEOC
          + /SVR3 * SVR2 * /SVR0 * SVL1 * SVL0 * SVC1 * MBRDY
         * WMSWND * /MABORT * /MRESET * /YCEOC
          + /SVR3 * /SVR2 * /SVR1 * SVR0 * SVL1 * /SVL0 * SVC1
         * MBRDY * WMSWND * /MABORT * /MRESET * /YCEOC
          + /SVR3 * /SVR2 * /SVR1 * SVR0 * SVL1 * SVC1 * /SVC0
         * MBRDY * WMSWND * /MABORT * /MRESET * /YCEOC
```

240957-68

```
;--------------------------------- Declaration Segment ------------
TITLE BYRDYSTR
PATTERN  A
REVISION 2.0
AUTHOR   ISIC SILAS
COMPANY  INTEL
DATE     2/4/91
CHIP  x01 85C22V10
;
;       This PLD contains the YRDYSTR, YRDYSTR, and YMEMEOC state machines.
;
;---------------------------------- PIN Declarations --------------
PIN  1          MCLK       COMBINATORIAL ;
PIN  2          MRESET       COMBINATORIAL ;
PIN  3          TR4        COMBINATORIAL ;
PIN  4          /YALLOC       COMBINATORIAL ;
;PIN  5 ####
PIN  6          /MABORT       COMBINATORIAL ;
PIN  7          /MBRDY       COMBINATORIAL ;
PIN  8          /WMSWND       COMBINATORIAL ;
PIN  9          /MBOFFI       COMBINATORIAL ;
PIN  10         /YMSEL       COMBINATORIAL ;
PIN  11         /YPIPE      COMBINATORIAL ;
PIN  12         GND
PIN  13         /SVL1       COMBINATORIAL ; INPUT
PIN  14         /SVL0       COMBINATORIAL ; INPUT
PIN  15         /SVR3            registered ;
PIN  16         /SVR2            registered ;
PIN  17         /SVR1            registered ;
PIN  18         /SVR0            registered ;
PIN  19         /YMEOC1            registered ;
PIN  20         /YMEOC            registered ;
PIN  21         /YMFRZ           registered ;
PIN  22         /CTCEND            registered ;
PIN  23         /SV           registered ;
PIN  24         VCC                          ;
;---------------------------------- Boolean Equation Segment ------
EQUATIONS


SVR3     =  /MRESET * /YMEOC * /MBRDY * /MABORT * SVR3
          + /MRESET * MBRDY * /MABORT * /MBOFFI * SVR2
          + /MRESET * /MBRDY * /MABORT * SVR3 * SVR2
          + /MRESET * MBRDY * /MABORT * SVR2 * SVR1
          + /MRESET * /YMEOC * /MABORT * SVR3 * SVR0
          + /MRESET * MBRDY * /MABORT * /TR4 * /SVR3 * SVR2

SVR2     =  /MRESET * /MBRDY * /MABORT * SVR2
          + /MRESET * /MABORT * SVR2 * SVR1
          + /MRESET * /YMEOC * MBRDY * /MABORT * SVR1
          + /MRESET * MBRDY * /MABORT * MBOFFI * SVR1
          + /MRESET * MBRDY * /MABORT * SVR1 * SVR0

SVR1     =  /MRESET * /MABORT * SVR1 * SVR0
          + /MRESET * /YMEOC * /MBRDY * /MABORT * SVR1
          + /MRESET * /MBRDY * /MABORT * MBOFFI * SVR1
          + /MRESET * /MBRDY * /MABORT * SVR2 * SVR1
          + /MRESET * /YMEOC * MBRDY * /MABORT * /SVR3 * SVR0
          + /MRESET * MBRDY * /MABORT * MBOFFI * /SVR3 * SVR0
          + /MRESET * /YMEOC * MBRDY * /MABORT * SVR3 * /SVR2 * /SVR0

SVR0     =  /MRESET * MBRDY * /MABORT * /MBOFFI * SVR3
          + /MRESET * MBRDY * /MABORT * SVR3 * /SVR2
          + /MRESET * /YMEOC * /MBRDY * /MABORT * SVR0
```

240957-69

```
          + /MRESET * /MBRDY * /MABORT * SVR1 * SVR0
          + /MRESET * /MBRDY * /MABORT * MBOFFI * /SVR3 * SVR0
          + /MRESET * YPIPE * YMEOC * MBRDY * /MABORT * /SVR1 * SVR0
          + /MRESET * YMSEL * MBRDY * /MABORT * /SVR2 * /SVR1 * /SVR0
          + /MRESET * MBRDY * /MABORT * MBOFFI * /SVR2 * /SVR1 * /SVR0
          + /MRESET * YPIPE * YMEOC * MBRDY * /MABORT * /SVR2 * SVR1
         * /SVR0

CTCEND     =  /MRESET * MBRDY * /MABORT * MBOFFI * SVR3 * SVR2
          + /MRESET * MBRDY * /MABORT * MBOFFI * TR4 * SVR2 * /SVR1

YMEOC      =  /MRESET * MABORT * YALLOC * /YMEOC * /SV
         + /MRESET * /SVR3 * /SVR2 * SVR1 * /SVR0 * SVL1 * /SVL0
        * WMSWND * /MABORT * /YMEOC * /SV
         + /MRESET * /SVR3 * /SVR2 * /SVR1 * SVR0 * /SVL1 * SVL0
        * WMSWND * /MABORT * /YMEOC * /SV
         + /MRESET * SVR3 * /SVR2 * /SVR1 * SVR0 * SVL1 * SVL0
        * WMSWND * /MABORT * /YMEOC * /SV
         + /MRESET * SVR3 * /SVR2 * /SVR1 * SVL1 * SVL0 * TR4 * WMSWND
        * /MABORT * /YMEOC * /SV
         + /MRESET * /SVR3 * /SVR2 * /SVR1 * /SVL1 * SVL0 * MBRDY
        * WMSWND * /MABORT * /YMEOC * /SV
         + /MRESET * /SVR3 * /SVR2 * /SVR1 * SVR0 * SVL1 * /SVL0
        * MBRDY * WMSWND * /MABORT * /YMEOC * /SV
         + /MRESET * SVR3 * SVR2 * /SVR1 * /SVR0 * SVL1 * SVL0
        * MBRDY * WMSWND * /MABORT * /YMEOC * /SV
         + /MRESET * SVR2 * /SVR1 * /SVR0 * SVL1 * SVL0 * TR4 * MBRDY
        * WMSWND * /MABORT * /YMEOC * /SV

SV         =  /MRESET * YMEOC

/YMFRZ     =  MRESET
          + /MABORT
          + /YALLOC
          + YMEOC
          + SV

YMEOC1     =  /MRESET * MABORT * YALLOC * /YMEOC1 * /SV
         + /MRESET * /SVR3 * /SVR2 * SVR1 * /SVR0 * SVL1 * /SVL0
        * WMSWND * /MABORT * /YMEOC1 * /SV
         + /MRESET * /SVR3 * /SVR2 * /SVR1 * SVR0 * /SVL1 * SVL0
        * WMSWND * /MABORT * /YMEOC1 * /SV
         + /MRESET * SVR3 * /SVR2 * /SVR1 * SVR0 * SVL1 * SVL0
        * WMSWND * /MABORT * /YMEOC1 * /SV
         + /MRESET * SVR3 * /SVR2 * /SVR1 * SVL1 * SVL0 * TR4 * WMSWND
        * /MABORT * /YMEOC1 * /SV
         + /MRESET * /SVR3 * /SVR2 * /SVR1 * /SVL1 * SVL0 * MBRDY
        * WMSWND * /MABORT * /YMEOC1 * /SV
         + /MRESET * /SVR3 * /SVR2 * /SVR1 * SVR0 * SVL1 * /SVL0
        * MBRDY * WMSWND * /MABORT * /YMEOC1 * /SV
         + /MRESET * SVR3 * SVR2 * /SVR1 * /SVR0 * SVL1 * SVL0
        * MBRDY * WMSWND * /MABORT * /YMEOC1 * /SV
         + /MRESET * SVR2 * /SVR1 * /SVR0 * SVL1 * SVL0 * TR4 * MBRDY
        * WMSWND * /MABORT * /YMEOC1 * /SV
```

240957-70

```
;--------------------------------- Declaration Segment ------------
TITLE EABORT
PATTERN  A
REVISION 2.0
AUTHOR   ISIC SILAS
COMPANY  INTEL
DATE     2/5/91

CHIP  x01 85C224

;
;       This PLD contains the YABORT, YRSTRT, and YMEMDOE state machines.
;
;--------------------------------- Pin Definitions ----------------------
PIN     1     MCLK
PIN     2     MRESET
PIN     3     WMSWND
PIN     4     YSWEHITM
PIN     5     YALLOC
PIN     6     YPIPE
PIN     7     YNOPIPE
PIN     8     YMEOC
PIN     9     MHITMI
PIN     10     MAOE
PIN     11     CTCEND
PIN     13     MKEN
PIN     14     MHLDA
PIN     15     YWR
PIN     16     YMADS
PIN     23     CTCDIS

PIN     18     RSTRT
PIN     19     YMDOE
PIN     20     PCTCXFR
PIN     21     TRIABORT
PIN     22     MABORT
PIN     17     SV          ;Swapped pins 23 and 17 to fit 85C224

EQUATIONS

/RSTRT.D := /MRESET * /PCTCXFR * /CTCDIS
          + /MRESET * /PCTCXFR * /RSTRT
          + /MRESET * /YSWEHITM * /CTCDIS * RSTRT
          + /MRESET * /YSWEHITM * YWR * YALLOC * RSTRT
RSTRT.CLKF = MCLK
RSTRT.RSTF = GND
RSTRT.SETF = GND
RSTRT.TRST = VCC

/YMDOE.D := /MRESET * YWR * /YPIPE * /YMEOC
          + /MRESET * /YNOPIPE * YMEOC * /YMDOE
          + /MRESET * MHLDA * YMEOC * /YMDOE
          + /MRESET * /YPIPE * YMEOC * /YMDOE
          + /MRESET * /YMADS * YWR * /YNOPIPE * YMDOE
          + /MRESET * /YMADS * YWR * MHLDA * YMDOE
YMDOE.CLKF = MCLK
YMDOE.RSTF = GND
YMDOE.SETF = GND
YMDOE.TRST = VCC

/PCTCXFR.D := /MRESET * YALLOC * /MABORT
            + /MRESET * /YSWEHITM * /MAOE * PCTCXFR
            + /MRESET * /MHITMI * /WMSWND * /MABORT
            + /MRESET * CTCEND * /PCTCXFR * MABORT
            + /MRESET * /PCTCXFR * MABORT * /SV
```

240957-71

```
            + /MRESET * /YSWEHITM * /YPIPE * YMEOC * PCTCXFR
            + /MRESET * /YPIPE * /YMEOC * /YALLOC * PCTCXFR
            + /MRESET * /YNOPIPE * YMEOC * /YALLOC * /MKEN * PCTCXFR
PCTCXFR.CLKF = MCLK
PCTCXFR.RSTF = GND
PCTCXFR.SETF = GND
PCTCXFR.TRST = VCC

/TRIABORT.D := /MRESET * /YPIPE * /YMEOC * /YALLOC * PCTCXFR * /MHLDA
             + /MRESET * /YNOPIPE * YMEOC * /YALLOC * /MKEN * PCTCXFR
                    * /MHLDA
             + /MRESET * /YSWEHITM * /MAOE * YPIPE * PCTCXFR * /MHLDA
             + /MRESET * /YSWEHITM * /MAOE * /YMEOC * PCTCXFR * /MHLDA
             + /MRESET * /YMEOC * /PCTCXFR * MABORT * /SV * /MHLDA
TRIABORT.CLKF = MCLK
TRIABORT.RSTF = GND
TRIABORT.SETF = GND
TRIABORT.TRST = /MHLDA

/MABORT.D := /MRESET * /YPIPE * /YMEOC * /YALLOC * PCTCXFR
           + /MRESET * /YNOPIPE * YMEOC * /YALLOC * /MKEN * PCTCXFR
           + /MRESET * /YSWEHITM * /MAOE * YPIPE * PCTCXFR
           + /MRESET * /YSWEHITM * /MAOE * /YMEOC * PCTCXFR
           + /MRESET * /YMEOC * /PCTCXFR * MABORT * /SV
MABORT.CLKF = MCLK
MABORT.RSTF = GND
MABORT.SETF = GND
MABORT.TRST = VCC

/SV.D := MABORT * /SV
       + PCTCXFR
SV.CLKF = MCLK
SV.RSTF = GND
SV.SETF = GND
SV.TRST = VCC
```

240957-72

```
;--------------------------------- Declaration Segment ------------
TITLE EASTB
PATTERN  A
REVISION 2.0
AUTHOR   ISIC SILAS
COMPANY  INTEL
DATE     2/4/91


CHIP  x01 85C224
;
;       This PLD contains the XASTB, XSTFAIL, and XDTSTRCK state machines
;
;--------------------------------- Pin Declarations ----------------

PIN     1     CLK
PIN     2     RESET
PIN     3     CADS
PIN     4     CDTS
PIN     5     SNPADS
PIN     6     CWR
PIN     7     YSBGT
PIN     8     CRDY
PIN     9     CAHOLD
PIN     10     FSIOUT
PIN     11     SLFTST
PIN     13     OEx
PIN     14     RDYSRC

PIN     16     LRDYSRC
PIN     17     SV2
PIN     18     STFAIL
PIN     19     SV1
PIN     20     XSNPWB
PIN     21     WSDTS
PIN     22     XAS


;  OE control inverted during design conversion.
EQUATIONS

LRDYSRC.D := RDYSRC
LRDYSRC.CLKF = CLK
LRDYSRC.RSTF = GND
LRDYSRC.SETF = GND
/LRDYSRC.TRST = OEx

/SV2.D := /RESET * FSIOUT * /SLFTST * STFAIL * SV2
SV2.CLKF = CLK
SV2.RSTF = GND
SV2.SETF = GND
/SV2.TRST = OEx

/STFAIL.D := /RESET * FSIOUT * /CAHOLD * /SLFTST * STFAIL * SV2
STFAIL.CLKF = CLK
STFAIL.RSTF = GND
STFAIL.SETF = GND
/STFAIL.TRST = OEx

/SV1.D := /RESET * CDTS * CRDY * /SV1
        + /RESET * CDTS * WSDTS * /SV1
        + /RESET * /SNPADS * XSNPWB * SV1
        + /RESET * /CDTS * CRDY * /WSDTS * XSNPWB
SV1.CLKF = CLK
SV1.RSTF = GND
```

240957-73

```
SV1.SETF = GND
/SV1.TRST = OEx

/XSNPWB.D := /RESET * CRDY * /XSNPWB
           + /RESET * /CDTS * WSDTS * /SV1
XSNPWB.CLKF = CLK
XSNPWB.RSTF = GND
XSNPWB.SETF = GND
/XSNPWB.TRST = OEx

/WSDTS.D := /RESET * CRDY * /XSNPWB
          + /RESET * /CDTS * XSNPWB
          + /RESET * /WSDTS * /SV1
          + /RESET * SNPADS * CRDY * /WSDTS
WSDTS.CLKF = CLK
WSDTS.RSTF = GND
WSDTS.SETF = GND
/WSDTS.TRST = OEx

/XAS.D := /RESET * SNPADS * YSBGT * /XAS
        + /RESET * /CDTS * CWR * XAS
        + /RESET * /CADS * /CWR * XAS
XAS.CLKF = CLK
XAS.RSTF = GND
XAS.SETF = GND
/XAS.TRST = OEx
```

240957-74

```
;----------------------------------Declaration Segment ------------
TITLE           EBGTKWN
PATTERN
REVISION        1.0
AUTHOR
COMPANY         INTEL
DATE


CHIP  INTEL 85C224
;
;       This PLD contains the XBGTKWND, XCNA and XENBGT state machines
;
;---------------------------------Pin Declarations-------------------
PIN     1     CLK
PIN     2     RESET
PIN     3     YSBGT
PIN     4     CRDY
PIN     5     C8LDRV
PIN     6     TR4
PIN     7     NC5
PIN     8     NC6
PIN     9     WCPLB
PIN     10     CNADIS
PIN     11     NC1
PIN     13     OE
PIN     14     NC2
PIN     15     NC3
PIN     23     NC4

PIN     16     CKENLC
PIN     17     ENBGT
PIN     18     CNA
PIN     19     PBGT
PIN     20     KWEND
PIN     21     C5BGT
PIN     22     BGT


EQUATIONS

/CKENLC.D := /RESET * YSBGT * CRDY * /BGT * /KWEND
           + /RESET * CRDY * ENBGT * /BGT * /KWEND
CKENLC.CLKF = CLK
CKENLC.RSTF = GND
CKENLC.SETF = GND
/CKENLC.TRST = OE

ENBGT.D := /RESET * /YSBGT
ENBGT.CLKF = CLK
ENBGT.RSTF = GND
ENBGT.SETF = GND
/ENBGT.TRST = OE

/CNA.D := /RESET * CRDY * /CNA
        + /RESET * /YSBGT * WCPLB * CNADIS
        + /RESET * /YSBGT * WCPLB * /CNA
        + /RESET * /PBGT * WCPLB * /CNA
        + /RESET * /BGT * WCPLB * CNADIS * CNA
CNA.CLKF = CLK
CNA.RSTF = GND
CNA.SETF = GND
/CNA.TRST = OE

/PBGT.D := /RESET * CRDY * /PBGT
```

240957-75

```
        + /RESET * /YSBGT * CRDY * /ENBGT * /BGT * /KWEND
PBGT.CLKF = CLK
PBGT.RSTF = GND
PBGT.SETF = GND
/PBGT.TRST = OE

/KWEND.D := /RESET * /BGT * KWEND
          + /RESET * /CRDY * /PBGT
          + /RESET * YSBGT * CRDY * /BGT
          + /RESET * CRDY * ENBGT * /BGT
          + RESET * TR4
KWEND.CLKF = CLK
KWEND.RSTF = GND
KWEND.SETF = GND
/KWEND.TRST = OE

/C5BGT.D := /RESET * /BGT * KWEND
          + /RESET * /CRDY * /PBGT
          + /RESET * YSBGT * CRDY * /BGT
          + /RESET * CRDY * ENBGT * /BGT
          + /RESET * /YSBGT * /CRDY * /ENBGT * /BGT
          + /RESET * /YSBGT * /ENBGT * C5BGT * KWEND * PBGT
          + RESET * /C8LDRV
C5BGT.CLKF = CLK
C5BGT.RSTF = GND
C5BGT.SETF = GND
/C5BGT.TRST = OE

/BGT.D := /RESET * /BGT * KWEND
        + /RESET * /CRDY * /PBGT
        + /RESET * YSBGT * CRDY * /BGT
        + /RESET * CRDY * ENBGT * /BGT
        + /RESET * /YSBGT * /CRDY * /ENBGT * /BGT
        + /RESET * /YSBGT * /ENBGT * C5BGT * KWEND * PBGT
BGT.CLKF = CLK
BGT.RSTF = GND
BGT.SETF = GND
/BGT.TRST = OE
```

240957-76

```
;---------------------------------- Declaration Segment ------------
TITLE EBRDY
PATTERN  A
REVISION 2.0
AUTHOR   ISIC SILAS
COMPANY  INTEL
DATE     2/4/91


CHIP  x01 85C224

;
;       This PLD contains the XBRDY, XMSWNDO, and XCTRCK state machines.
;
;---------------------------------- Pin Declarations ---------------
PIN     1     CLK
PIN     2     RESET
PIN     3     CLEN1
PIN     4     WSDTS
PIN     5     YSCEOC
PIN     6     SNPCYC
PIN     7     MSNPSTB
PIN     8     CKENLC
PIN     9     MKEN
PIN     13     OEx

PIN     15     CKEN
PIN     17     SV
PIN     18     PNDCEOC
PIN     19     ENBRDY
PIN     20     BRDY
PIN     21     BRDY1
PIN     22     MSWNDO


EQUATIONS

/CKEN = CKENLC * /MKEN
      + /CKENLC * /CKEN
      + /MKEN * /CKEN
CKEN.TRST = VCC

/SV.D := /RESET * BRDY * /SV
       + /RESET * CLEN1 * /SV
       + /RESET * /YSCEOC * BRDY * ENBRDY * /PNDCEOC
       + /RESET * /YSCEOC * CLEN1 * ENBRDY * /PNDCEOC
SV.CLKF = CLK
SV.RSTF = GND
SV.SETF = GND
/SV.TRST = OEx

/PNDCEOC.D := /RESET * /SV
            + /RESET * BRDY * /PNDCEOC
            + /RESET * CLEN1 * /PNDCEOC
            + /RESET * /YSCEOC * ENBRDY * /PNDCEOC
            + /RESET * /YSCEOC * /ENBRDY * PNDCEOC
PNDCEOC.CLKF = CLK
PNDCEOC.RSTF = GND
PNDCEOC.SETF = GND
/PNDCEOC.TRST = OEx

/ENBRDY.D := YSCEOC * PNDCEOC
           + /ENBRDY * PNDCEOC
           + YSCEOC * /BRDY * /CLEN1
           + /YSCEOC * BRDY * /ENBRDY
```

240957-77

```
            + /YSCEOC * CLEN1 * /ENBRDY
            + /BRDY * /CLEN1 * ENBRDY * /PNDCEOC
            + RESET
ENBRDY.CLKF = CLK
ENBRDY.RSTF = GND
ENBRDY.SETF = GND
/ENBRDY.TRST = OEx

/BRDY.D := /RESET * CLEN1 * /BRDY
         + /RESET * /PNDCEOC * /WSDTS * BRDY
         + /RESET * /YSCEOC * /ENBRDY * /WSDTS * BRDY
BRDY.CLKF = CLK
BRDY.RSTF = GND
BRDY.SETF = GND
/BRDY.TRST = OEx

/BRDY1.D := /RESET * CLEN1 * /BRDY1
          + /RESET * /PNDCEOC * /WSDTS * BRDY1
          + /RESET * /YSCEOC * /ENBRDY * /WSDTS * BRDY1
BRDY1.CLKF = CLK
BRDY1.RSTF = GND
BRDY1.SETF = GND
/BRDY1.TRST = OEx

/MSWNDO = /RESET * /SNPCYC
        + /RESET * MSNPSTB * /MSWNDO
MSWNDO.TRST = VCC
```

240957-78

```
;--------------------------------- Declaration Segment ------------
TITLE ECYCDEF
PATTERN  A
REVISION 2.0
AUTHOR   ISIC SILAS
COMPANY  INTEL
DATE     2/5/91


CHIP  x01 85C224
;
;       This PLD contains the YMEMLEN state machine
;
;--------------------------------- Pin Declarations ---------------
PIN     1     YMALE
PIN     2     YMAOE
PIN     3     MHLDA
PIN     4     KCACHE
PIN     5     CWR
PIN     6     CMIO
PIN     7     CDC
PIN     8     LLEN
PIN     9     C5MTHIT
PIN     10     YSNPDIS
PIN     11     NC1
PIN     13     NC2
PIN     14     YNOSWNDI
PIN     23     YWRI

PIN     15     YWR
PIN     16     MTHIT
PIN     17     MLEN
PIN     18     MDC
PIN     19     MMIO
PIN     20     MWR
PIN     21     MCACHE
PIN     22     YNOSWND

EQUATIONS

/YWR = YMALE * /CWR
     + /YMALE * /YWRI
     + /CWR * /YWRI
YWR.TRST = VCC

/MTHIT = /C5MTHIT * /MWR * MHLDA
MTHIT.TRST = MHLDA

/MLEN = YMALE * /LLEN * /YMAOE
      + /YMALE * /MLEN * /YMAOE
      + /LLEN * /MLEN * /YMAOE
MLEN.TRST = /YMAOE

/MDC = YMALE * /CDC * /YMAOE
     + /YMALE * /MDC * /YMAOE
     + /CDC * /MDC * /YMAOE
MDC.TRST = /YMAOE

/MMIO = YMALE * /CMIO * /YMAOE
      + /YMALE * /MMIO * /YMAOE
      + /CMIO * /MMIO * /YMAOE
MMIO.TRST = /YMAOE

/MWR = YMALE * /CWR * /YMAOE
     + /YMALE * /MWR * /YMAOE
```

240957-79

```
     + /CWR * /MWR * /YMAOE
MWR.TRST = /YMAOE

/MCACHE = YMALE * /KCACHE * /YMAOE
        + /YMALE * /MCACHE * /YMAOE
        + /KCACHE * /MCACHE * /YMAOE
MCACHE.TRST = /YMAOE

/YNOSWND = YMALE * /YSNPDIS
         + YMALE * /CMIO
         + YMALE * CWR * /KCACHE
         + /YMALE * /YNOSWNDI
         + /YSNPDIS * /YNOSWNDI
         + /CMIO * /YNOSWNDI
         + CWR * /KCACHE * /YNOSWNDI
YNOSWND.TRST = VCC
```

240957-80

```
;--------------------------------- Declaration Segment ------------
TITLE EMBE
PATTERN  A
REVISION 2.0
AUTHOR   ISIC SILAS
COMPANY  INTEL
DATE     2/5/91


CHIP  x01 85C224
;
;       This PLD generates the memory bus byte enables (MBEs)
;
;------------------------------Pin Declarations--------------------

PIN     1     LBE0
PIN     2     LBE1
PIN     3     LBE2
PIN     4     LBE3
PIN     5     LBE4
PIN     6     LBE5
PIN     7     LBE6
PIN     8     LBE7
PIN     9     RDYSRC
PIN     10     KCACHE
PIN     11     YMALE
PIN     13     YMAOE
PIN     14     MBE6I
PIN     23     MBE7I

PIN     15     MBE7
PIN     16     MBE5
PIN     17     MBE4
PIN     18     MBE3
PIN     19     MBE2
PIN     20     MBE1
PIN     21     MBE0
PIN     22     MBE6

EQUATIONS

/MBE7 = /YMALE * /LBE7 * /YMAOE
      + /YMALE * /KCACHE * /RDYSRC * /YMAOE
      + YMALE * /MBE7I * /YMAOE
      + /LBE7 * /MBE7I * /YMAOE
      + /KCACHE * /RDYSRC * /MBE7I * /YMAOE
MBE7.TRST = /YMAOE

/MBE5 = /YMALE * /LBE5 * /YMAOE
      + /YMALE * /KCACHE * /RDYSRC * /YMAOE
      + YMALE * /MBE5 * /YMAOE
      + /LBE5 * /MBE5 * /YMAOE
      + /KCACHE * /RDYSRC * /MBE5 * /YMAOE
MBE5.TRST = /YMAOE

/MBE4 = /YMALE * /LBE4 * /YMAOE
      + /YMALE * /KCACHE * /RDYSRC * /YMAOE
      + YMALE * /MBE4 * /YMAOE
      + /LBE4 * /MBE4 * /YMAOE
      + /KCACHE * /RDYSRC * /MBE4 * /YMAOE
MBE4.TRST = /YMAOE

/MBE3 = /YMALE * /LBE3 * /YMAOE
      + /YMALE * /KCACHE * /RDYSRC * /YMAOE
      + YMALE * /MBE3 * /YMAOE
```

240957-81

2

```
      + /LBE3 * /MBE3 * /YMAOE
      + /KCACHE * /RDYSRC * /MBE3 * /YMAOE
MBE3.TRST = /YMAOE

/MBE2 = /YMALE * /LBE2 * /YMAOE
      + /YMALE * /KCACHE * /RDYSRC * /YMAOE
      + YMALE * /MBE2 * /YMAOE
      + /LBE2 * /MBE2 * /YMAOE
      + /KCACHE * /RDYSRC * /MBE2 * /YMAOE
MBE2.TRST = /YMAOE

/MBE1 = /YMALE * /LBE1 * /YMAOE
      + /YMALE * /KCACHE * /RDYSRC * /YMAOE
      + YMALE * /MBE1 * /YMAOE
      + /LBE1 * /MBE1 * /YMAOE
      + /KCACHE * /RDYSRC * /MBE1 * /YMAOE
MBE1.TRST = /YMAOE

/MBE0 = /YMALE * /LBE0 * /YMAOE
      + /YMALE * /KCACHE * /RDYSRC * /YMAOE
      + YMALE * /MBE0 * /YMAOE
      + /LBE0 * /MBE0 * /YMAOE
      + /KCACHE * /RDYSRC * /MBE0 * /YMAOE
MBE0.TRST = /YMAOE

/MBE6 = /YMALE * /LBE6 * /YMAOE
      + /YMALE * /KCACHE * /RDYSRC * /YMAOE
      + YMALE * /MBE6I * /YMAOE
      + /LBE6 * /MBE6I * /YMAOE
      + /KCACHE * /RDYSRC * /MBE6I * /YMAOE
MBE6.TRST = /YMAOE
```

240957-82

```
;--------------------------------- Declaration Segment ------------
TITLE EMEMALE
PATTERN  A
REVISION 2.0
AUTHOR   ISIC SILAS
COMPANY  INTEL
DATE     2/4/91


CHIP  x01 85C224
;
;       This PLD contains the YMALE, YMBRDY, YWMNA,
;       and YIMSWND state machines.
;
;--------------------------------- Pin Declarations ----------------

PIN    1    MCLK
PIN    2    MRESET
PIN    3    YBGT
PIN    4    YNOPIPE
PIN    5    YPIPE
PIN    6    MNA
PIN    7    WMSWND
PIN    8    YMEOC
PIN    9    PXSAS
PIN    10    YALLOC
PIN    11    MSWNDI
PIN    13    OE
PIN    14    MBRDY
PIN    23    YMADS

PIN    15    YIMSWND
PIN    16    YDRCTM
PIN    17    NC1
PIN    18    YMALE
PIN    19    DISWND
PIN    20    WMNA
PIN    21    YMBRDY
PIN    22    NC2


EQUATIONS

/YIMSWND = /MSWNDI * YALLOC * DISWND
YIMSWND.TRST = VCC

/YDRCTM.D := /MRESET * /DISWND
           + /MRESET * YMEOC * YPIPE * /YDRCTM
YDRCTM.CLKF = MCLK
YDRCTM.RSTF = GND
YDRCTM.SETF = GND
/YDRCTM.TRST = OE

NC1.D := VCC
NC1.CLKF = MCLK
NC1.RSTF = GND
NC1.SETF = GND
/NC1.TRST = OE

/YMALE.D := /MRESET * /YPIPE * /YMALE
          + /MRESET * /YBGT * YMALE
          + /MRESET * YNOPIPE * YMEOC * /YMALE
          + /MRESET * WMSWND * YMEOC * /YMALE
          + /MRESET * /YMADS * WMNA * YMEOC * /YMALE
          + /MRESET * MNA * WMNA * YMEOC * /YMALE
```

240957-83

```
YMALE.CLKF = MCLK
YMALE.RSTF = GND
YMALE.SETF = GND
/YMALE.TRST = OE

/DISWND.D := /MRESET * /DISWND * YDRCTM
           + /MRESET * /PXSAS * /YALLOC * YDRCTM
DISWND.CLKF = MCLK
DISWND.RSTF = GND
DISWND.SETF = GND
/DISWND.TRST = OE

/WMNA.D := /MRESET * /YNOPIPE * YMADS * YMEOC * /WMNA
         + /MRESET * /YNOPIPE * YMADS * /MNA * WMNA
         + /MRESET * /YPIPE * YMADS * /MNA * /YMEOC * WMNA
WMNA.CLKF = MCLK
WMNA.RSTF = GND
WMNA.SETF = GND
/WMNA.TRST = OE

/YMBRDY.D := /MRESET * /MBRDY
YMBRDY.CLKF = MCLK
YMBRDY.RSTF = GND
YMBRDY.SETF = GND
/YMBRDY.TRST = OE

NC2 = VCC
NC2.TRST = VCC
```

240957-84

```
;--------------------------------- Declaration Segment ------------
TITLE EMSNPST
PATTERN  A
REVISION 2.0
AUTHOR   ISIC SILAS
COMPANY  INTEL
DATE     2/4/91

CHIP  x01 85C224
;
;       This PLD contains the YALLC, YMEMLOCK, YSNPSTB,
;       and YMBREQ state machines.
;
;--------------------------------Pin Declarations------------------

PIN     1     MCLK
PIN     2     MRESET
PIN     3     MKEN
PIN     4     MHLDA
PIN     5     YWR
PIN     6     YNOSWND
PIN     7     YBGT
PIN     8     XLRDYSRC
PIN     9     RFO
PIN     10     SNPDIS
PIN     11     PALLC
PIN     13     KLOCK
PIN     23     PXSAS

PIN     16     YMLOCK
PIN     17     SV2
PIN     18     HBASWB
PIN     19     MBREQ
PIN     20     MSNPSTB
PIN     21     SV1
PIN     22     YALLOC

EQUATIONS

/YMLOCK.D := /MRESET * /YALLOC * YMLOCK
           + /MRESET * /HBASWB * YMLOCK
           + /MRESET * YBGT * /YALLOC * /HBASWB
           + /MRESET * /KLOCK * /YALLOC * /HBASWB
           + /MRESET * /YBGT * /KLOCK * YMLOCK
YMLOCK.CLKF = MCLK
YMLOCK.RSTF = GND
YMLOCK.SETF = GND
YMLOCK.TRST = VCC

/SV2.D := PXSAS * /SV2
        + PXSAS * /MHLDA * /HBASWB
SV2.CLKF = MCLK
SV2.RSTF = GND
SV2.SETF = GND
SV2.TRST = VCC

/HBASWB.D := /MRESET * PXSAS * /HBASWB * SV2
           + /MRESET * PXSAS * /YBGT * MBREQ * SV2
HBASWB.CLKF = MCLK
HBASWB.RSTF = GND
HBASWB.SETF = GND
HBASWB.TRST = VCC

/MBREQ.D := PXSAS * /MBREQ
          + PXSAS * HBASWB * /SV2
```

240957-85

```
          + /PXSAS * /YBGT * MBREQ * HBASWB * SV2
          + MRESET
MBREQ.CLKF = MCLK
MBREQ.RSTF = GND
MBREQ.SETF = GND
MBREQ.TRST = VCC

/MSNPSTB.D := /MRESET * /YBGT * YNOSWND * YWR * MSNPSTB
            + /MRESET * /YBGT * YNOSWND * XLRDYSRC * MSNPSTB
            + /MRESET * /YBGT * YNOSWND * RFO * MSNPSTB
MSNPSTB.CLKF = MCLK
MSNPSTB.RSTF = GND
MSNPSTB.SETF = GND
MSNPSTB.TRST = VCC

/SV1.D := /YALLOC
SV1.CLKF = MCLK
SV1.RSTF = GND
SV1.SETF = GND
SV1.TRST = VCC

/YALLOC.D := /MRESET * PXSAS * /YALLOC * /SV1
           + /MRESET * /MKEN * /YALLOC * SV1
           + /MRESET * /YBGT * /PALLC * SNPDIS * /RFO * YALLOC
YALLOC.CLKF = MCLK
YALLOC.RSTF = GND
YALLOC.SETF = GND
YALLOC.TRST = VCC
```

240957-86

```
;---------------------------------- Declaration Segment ------------
TITLE EMZBT
PATTERN  A
REVISION 3.1
AUTHOR   ISIC SILAS + Andy Bloom
COMPANY  INTEL
DATE     2/7/91

CHIP  x01 85C224
;
;       This PLD contains the YMBRDY state machine.
;
;------------------------------- Pin Declarations ---------------

PIN    1    MCLK
PIN    2    MRESET
PIN    3    MAOE
PIN    4    MHLDA
PIN    5    YNOPIPE
PIN    6    YPIPE
PIN    7    MCACHE
PIN    8    YMEOC
PIN    9    MEMZBTEN
PIN    10    SYNC
PIN    11    MALDRV
PIN    13    FLUSH
PIN    14    NCPFLD
PIN    15    FPFLDEN
PIN    23    NC4

PIN    16    NC1
PIN    17    NC2
PIN    18    NC3
PIN    19    YMZBT
PIN    20    FPFLD
PIN    21    YFLUSH
PIN    22    YSYNC

EQUATIONS

NC1 = VCC
NC1.TRST = VCC

NC2 = VCC
NC2.TRST = VCC

NC3 = VCC
NC3.TRST = VCC

/YMZBT.D := /MRESET * YPIPE * YMEOC * /YMZBT
          + /MRESET * /MCACHE * YMEOC * /YMZBT
          + /MRESET * YNOPIPE * /YPIPE * /MCACHE * /MEMZBTEN
          + /MRESET * YNOPIPE * /YPIPE * /MCACHE * /YMZBT
          + /MRESET * MHLDA * /MAOE * /MEMZBTEN * YMZBT
          + /MRESET * /YNOPIPE * /MCACHE * /MEMZBTEN * YMZBT
          + MRESET
YMZBT.CLKF = MCLK
YMZBT.RSTF = GND
YMZBT.SETF = GND
YMZBT.TRST = VCC
```

240957-87

```
/FPFLD = FPFLDEN * MRESET
FPFLD.TRST = MRESET

/YFLUSH = MRESET * /NCPFLD
        + /MRESET * /FLUSH
YFLUSH.TRST = VCC

/YSYNC = MRESET * /MALDRV
       + /MRESET * /SYNC
YSYNC.TRST = VCC
```

240957-88

```
;----------------------------Declaration Segment--------------------
TITLE           ESIGGEN
PATTERN
REVISION        1.0
AUTHOR
COMPANY         INTEL
DATE
CHIP  INTEL 85C224
;
;       This PLD drives memory bus and core signals based on the states
;       of other state machines
;
;----------------------------Pin Declarations-----------------------
PIN     1     YDRCTM
PIN     2     YMADS
PIN     3     YMAOE
PIN     4     MHLDA
PIN     5     NC1
PIN     6     MWBWT
PIN     7     MDRCTM
PIN     8     SNPDIS
PIN     9     UNI
PIN     10     YMSEL
PIN     11     TR4
PIN     13     YMFRZ
PIN     14     MDLDRV
PIN     23     LMRST

PIN     15     C8MSEL
PIN     16     NC2
PIN     17     CDRCTM
PIN     18     CWBWT
PIN     19     MBOFF
PIN     20     MADS
PIN     21     YSNPDIS
PIN     22     C8MFRZ

EQUATIONS

/C8MSEL = LMRST * /TR4
        + /LMRST * /YMSEL
C8MSEL.TRST = VCC

NC2 = VCC
NC2.TRST = VCC

CDRCTM = MDRCTM * YDRCTM
CDRCTM.TRST = VCC

/CWBWT = /MWBWT * YDRCTM
CWBWT.TRST = VCC

/MBOFF = YMAOE * /MHLDA
MBOFF.TRST = /MHLDA

/MADS = /YMADS * /YMAOE
MADS.TRST = /YMAOE

YSNPDIS = SNPDIS * UNI
YSNPDIS.TRST = VCC

/C8MFRZ = LMRST * /MDLDRV
        + /LMRST * /YMFRZ
C8MFRZ.TRST = VCC
```

240957-89

```
;-------------------------Declaration Segment-------------------------
TITLE ESWND
PATTERN  A
REVISION 2.0
AUTHOR   ISIC SILAS
COMPANY  INTEL
DATE     2/4/91


CHIP  x01 85C224

;
;       This PLD contains the XCRDY, XSWND, and XENSWND state machines.
;
;---------------------------Pin Declarations---------------------------
PIN     1     CLK
PIN     2     RESET
PIN     3     WSDTS
PIN     4     BGT
PIN     5     PBGT
PIN     6     TR4
PIN     7     YSMSWND
PIN     8     SNPDIS
PIN     9     YSMEOC
PIN     10     SLFTST
PIN     13     OEx

PIN     16     ENSWND
PIN     17     SV3
PIN     18     SWEND
PIN     19     SV2
PIN     20     SV1
PIN     21     CRDY
PIN     22     CRDY1


EQUATIONS

ENSWND.D := /RESET * /YSMSWND
ENSWND.CLKF = CLK
ENSWND.RSTF = GND
ENSWND.SETF = GND
/ENSWND.TRST = OEx

/SV3.D := RESET * TR4
        + /RESET * CRDY * SWEND * /SV3
        + /RESET * /PBGT * /ENSWND * /YSMSWND * CRDY * SV3
        + /RESET * /PBGT * CRDY * /SNPDIS * /SWEND * SV3
        + /RESET * /PBGT * /ENSWND * /YSMSWND * SWEND * SV3
SV3.CLKF = CLK
SV3.RSTF = GND
SV3.SETF = GND
/SV3.TRST = OEx

/SWEND.D := RESET * TR4
          + /RESET * /CRDY * SWEND * /SV3
          + /RESET * PBGT * CRDY * /SWEND * SV3
          + /RESET * /BGT * /SNPDIS * SWEND * SV3
          + /RESET * ENSWND * CRDY * SNPDIS * /SWEND * SV3
          + /RESET * YSMSWND * CRDY * SNPDIS * /SWEND * SV3
          + /RESET * /BGT * /ENSWND * /YSMSWND * SWEND * SV3
          + /RESET * /PBGT * /ENSWND * /YSMSWND * /CRDY * /SWEND * SV3
SWEND.CLKF = CLK
SWEND.RSTF = GND
SWEND.SETF = GND
```

240957-90

```
/SWEND.TRST = OEx

/SV2.D := RESET * /SLFTST
        + /RESET * /YSMEOC * CRDY * /SV2
        + /RESET * /YSMEOC * /CRDY * SV2
SV2.CLKF = CLK
SV2.RSTF = GND
SV2.SETF = GND
/SV2.TRST = OEx

/SV1.D := /RESET * /YSMEOC * CRDY
        + /RESET * CRDY * /SV1 * SV2
SV1.CLKF = CLK
SV1.RSTF = GND
SV1.SETF = GND
/SV1.TRST = OEx

/CRDY.D := RESET * /SLFTST
         + /RESET * /YSMEOC * /WSDTS * CRDY * SV2
         + /RESET * /WSDTS * CRDY * /SV1 * SV2
CRDY.CLKF = CLK
CRDY.RSTF = GND
CRDY.SETF = GND
/CRDY.TRST = OEx

/CRDY1.D := RESET * /SLFTST
          + /RESET * /YSMEOC * /WSDTS * CRDY1 * SV2
          + /RESET * /WSDTS * CRDY1 * /SV1 * SV2
CRDY1.CLKF = CLK
CRDY1.RSTF = GND
CRDY1.SETF = GND
/CRDY1.TRST = OEx
```

240957-91

2

```
;--------------------------------Declaration Segment---------------
TITLE EWCPLB
PATTERN  A
REVISION 2.0
AUTHOR   ISIC SILAS
COMPANY  INTEL
DATE     2/4/91


CHIP  x01 85C224
;
;       This PLD contains the XWCPLB and YCPULEN state machines.
;
;-----------------------------Pin Declaration----------------------
PIN     1     CLK
PIN     2     RESET
PIN     3     CRDY
PIN     4     RDYSRC
PIN     5     BGT
PIN     6     PBGT
PIN     7     KCACHE
PIN     8     LEN
PIN     9     CACHE
PIN     10     CKEN
PIN     11     BRDY
PIN     13     OEx

PIN     16     CLEN4
PIN     17     CLEN2
PIN     18     CLEN1
PIN     19     LKCACHE
PIN     20     SV
PIN     21     CPUEN
PIN     22     WCPLB


EQUATIONS

/CLEN4.D := CPUEN * /CLEN2 * /CLEN4
          + /BRDY * /CLEN1
          + BRDY * CLEN2 * /CLEN4
          + /CACHE * /CKEN * /LKCACHE * /CLEN2 * /CLEN4
          + RESET
CLEN4.CLKF = CLK
CLEN4.RSTF = GND
CLEN4.SETF = GND
/CLEN4.TRST = OEx

/CLEN2.D := CPUEN * /CLEN2 * /CLEN4
          + BRDY * /CLEN2 * CLEN4
          + /BRDY * CLEN2 * CLEN4
          + LEN * CACHE * /CLEN2 * /CLEN4
          + LEN * CKEN * /CLEN2 * /CLEN4
          + LEN * LKCACHE * /CLEN2 * /CLEN4
          + RESET
CLEN2.CLKF = CLK
CLEN2.RSTF = GND
CLEN2.SETF = GND
/CLEN2.TRST = OEx

/CLEN1.D := /RESET * BRDY * /CLEN1
          + /RESET * /BRDY * /CLEN2 * CLEN4
          + /RESET * /CPUEN * /LEN * CACHE * /CLEN2 * /CLEN4
          + /RESET * /CPUEN * /LEN * CKEN * /CLEN2 * /CLEN4
          + /RESET * /CPUEN * /LEN * LKCACHE * /CLEN2 * /CLEN4
CLEN1.CLKF = CLK
```

240957-92

```
CLEN1.RSTF = GND
CLEN1.SETF = GND
/CLEN1.TRST = OEx

/LKCACHE.D := /KCACHE
LKCACHE.CLKF = CLK
LKCACHE.RSTF = GND
LKCACHE.SETF = GND
/LKCACHE.TRST = OEx

/SV.D := /RESET * CRDY * /SV
       + /RESET * /RDYSRC * /BGT * CPUEN * SV
       + /RESET * CRDY * /BRDY * /CLEN1 * WCPLB * /CPUEN
SV.CLKF = CLK
SV.RSTF = GND
SV.SETF = GND
/SV.TRST = OEx

/CPUEN.D := /RESET * BRDY * /CPUEN
          + /RESET * CLEN1 * /CPUEN
          + /RESET * RDYSRC * /BGT * /WCPLB
          + /RESET * RDYSRC * /BGT * CPUEN * SV
CPUEN.CLKF = CLK
CPUEN.RSTF = GND
CPUEN.SETF = GND
/CPUEN.TRST = OEx

/WCPLB.D := /RESET * BRDY * /WCPLB
          + /RESET * CLEN1 * /WCPLB
          + /RESET * /CRDY * BRDY * /CPUEN
          + /RESET * /CRDY * CLEN1 * /CPUEN
WCPLB.CLKF = CLK
WCPLB.RSTF = GND
WCPLB.SETF = GND
/WCPLB.TRST = OEx
```

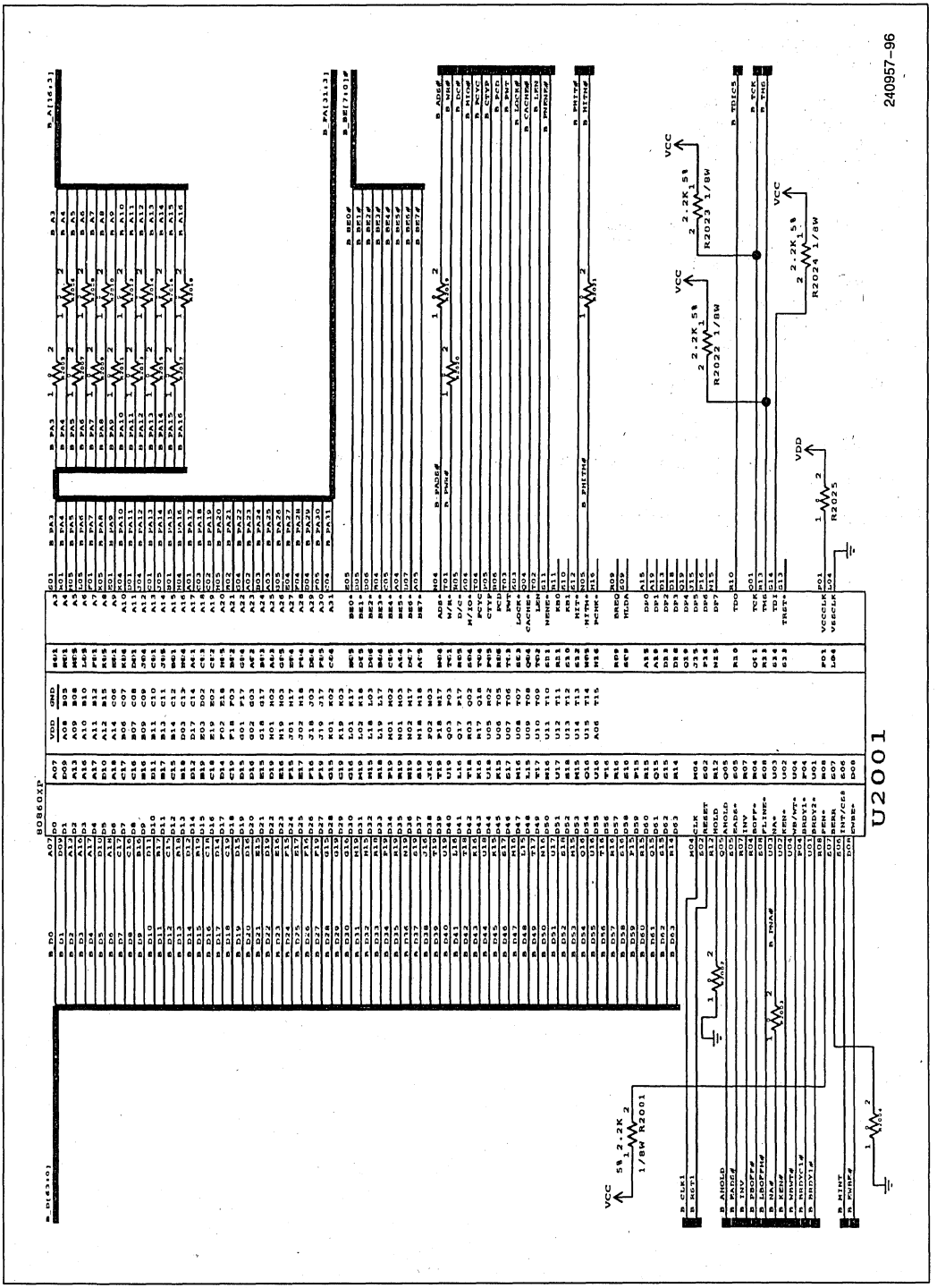
240957-93

```
;-----------------------------Declaration Segment-----------------------
TITLE EWMSWND
PATTERN  A
REVISION 2.0
AUTHOR   ISIC SILAS
COMPANY  INTEL
DATE     2/4/91


CHIP  x01 85C224
;
;       This PLD contains the YENMSWND, YWMSWND, and YENXSAS state machines.
;
;---------------------------Pin Declarations----------------------------
PIN     1     MCLK
PIN     2     MRESET
PIN     3     XSAS
PIN     4     XSNPWB
PIN     5     YPIPE
PIN     6     YNOPIPE
PIN     7     YMEOC
PIN     8     MHITMI
PIN     9     YMSWEND
PIN     10     YNOSWND
PIN     11     YBGT
PIN     13     OEx
PIN     14     YALLOC
PIN     23     PCTCXFR

PIN     15     UNUSED
PIN     16     PSWBAS
PIN     17     SV
PIN     18     WMSWND
PIN     19     ENMSWND
PIN     20     ENXSAS
PIN     21     PXSAS
PIN     22     YSWEHITM


EQUATIONS

UNUSED = VCC
UNUSED.TRST = VCC

/PSWBAS = /XSAS * /XSNPWB * /ENXSAS
PSWBAS.TRST = VCC

/SV.D := YMEOC * /WMSWND * /SV
       + /YNOSWND * /YPIPE * YMEOC * /WMSWND
       + /YPIPE * /YMSWEND * /ENMSWND * YMEOC * /WMSWND
SV.CLKF = MCLK
SV.RSTF = GND
SV.SETF = GND
/SV.TRST = OEx

/WMSWND.D := /MRESET * YMEOC * /WMSWND
           + /MRESET * /WMSWND * /SV
           + /MRESET * /YNOSWND * /YPIPE * /WMSWND
           + /MRESET * /YNOSWND * /YBGT * WMSWND
           + /MRESET * /YPIPE * /YMSWEND * /ENMSWND * /WMSWND
           + /MRESET * YPIPE * /PCTCXFR * /YALLOC * /WMSWND
           + /MRESET * /YMSWEND * /ENMSWND * /YALLOC * WMSWND
           + /MRESET * YNOSWND * /YNOPIPE * /YMSWEND * /ENMSWND * WMSWND
WMSWND.CLKF = MCLK
WMSWND.RSTF = GND
```

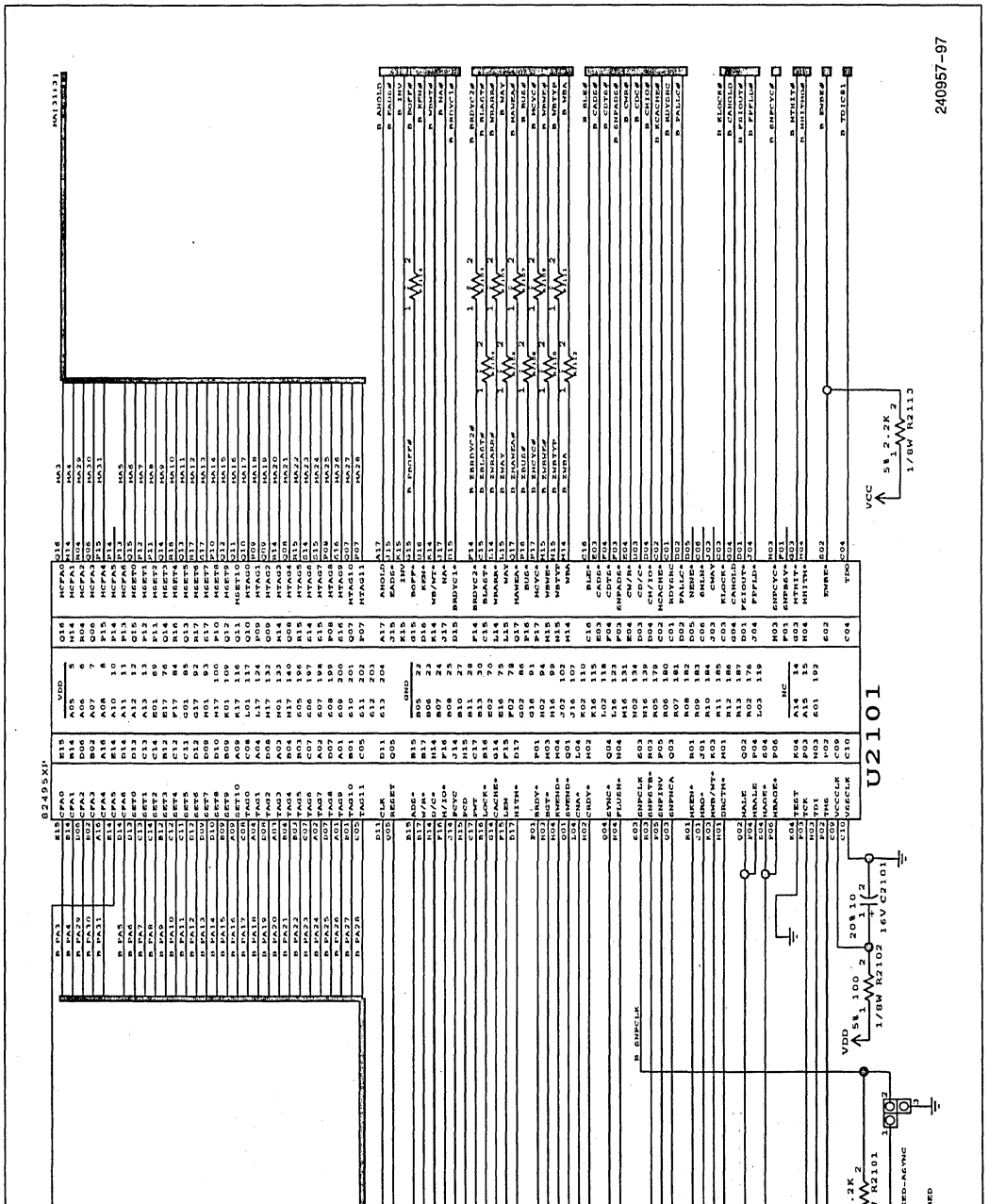
240957-94

```
WMSWND.SETF = GND
/WMSWND.TRST = OEx

/ENMSWND.D := YMSWEND
ENMSWND.CLKF = MCLK
ENMSWND.RSTF = GND
ENMSWND.SETF = GND
/ENMSWND.TRST = OEx

/ENXSAS.D := YBGT * /ENXSAS
           + XSAS * ENXSAS
           + MRESET
ENXSAS.CLKF = MCLK
ENXSAS.RSTF = GND
ENXSAS.SETF = GND
/ENXSAS.TRST = OEx

/PXSAS = /XSAS * XSNPWB * /ENXSAS
PXSAS.TRST = VCC

/YSWEHITM = /YMSWEND * /ENMSWND * /MHITMI * YALLOC
          + /YMSWEND * /ENMSWND * /MHITMI * YNOPIPE * YPIPE
YSWEHITM.TRST = VCC
```

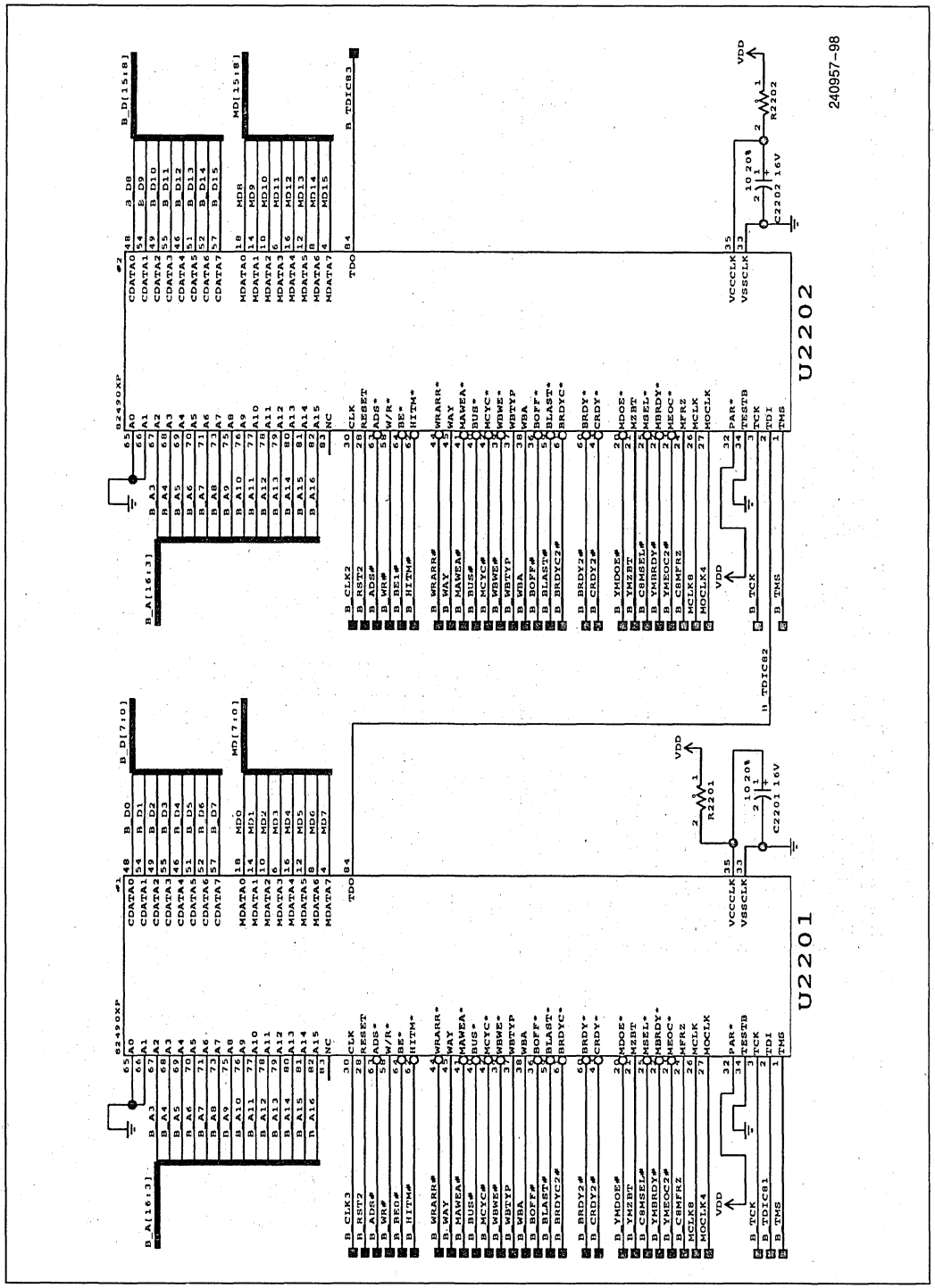
240957-95



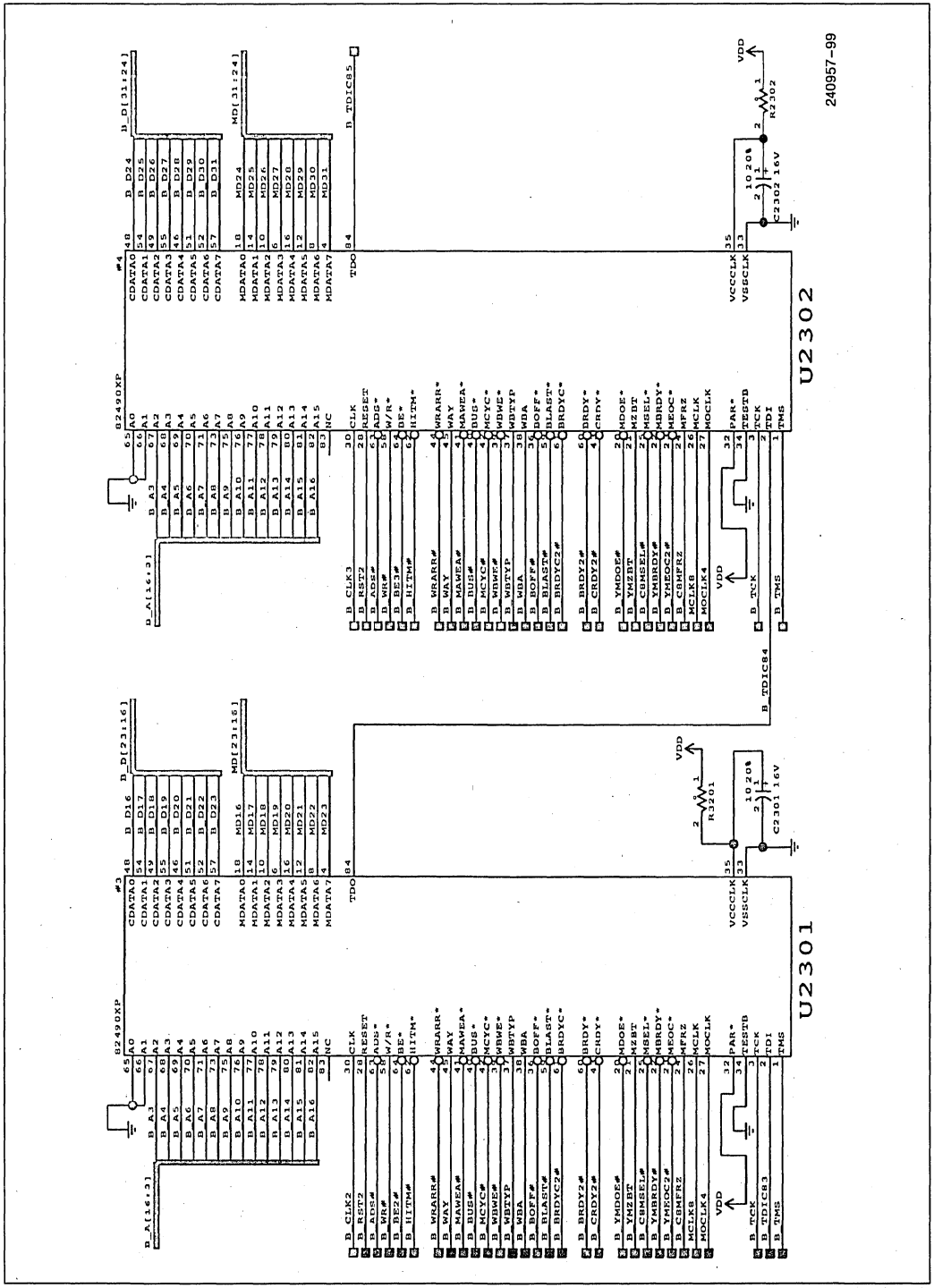
Appendix C Schematic: i860™ XP CPU



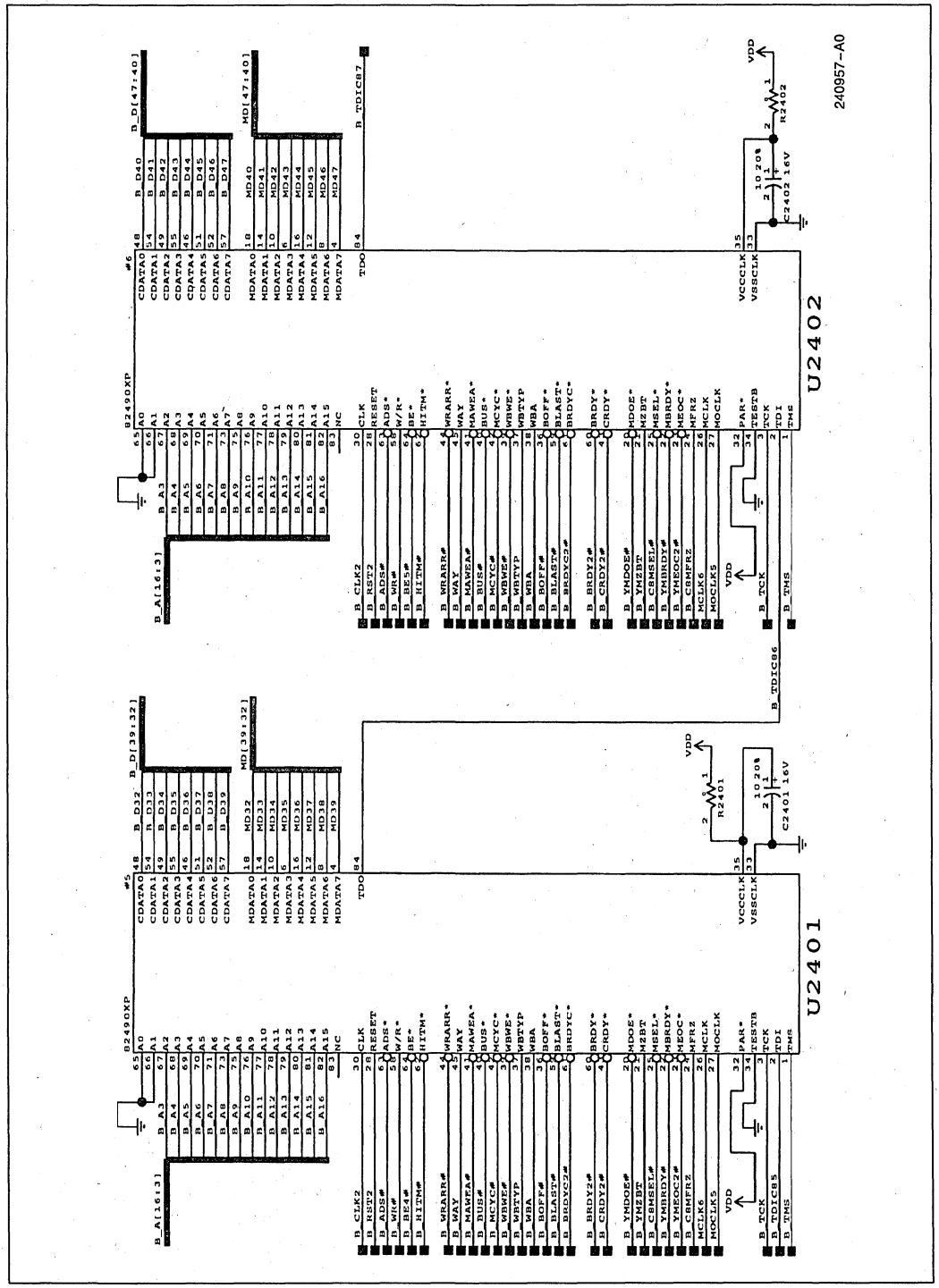
Appendix C Schematic: 82495XP Cache Controller



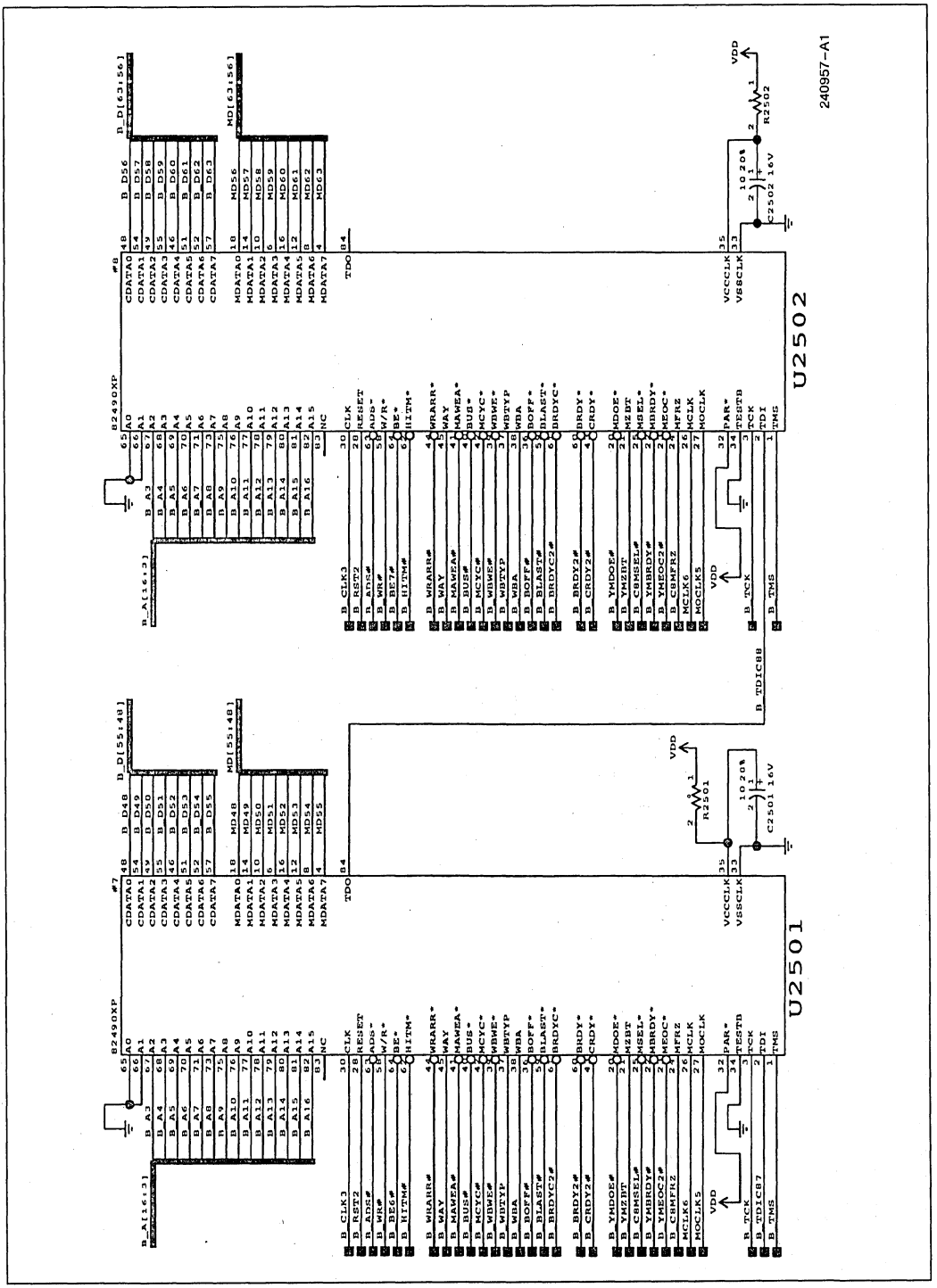
Appendix C Schematic: 82490XP Cache RAM



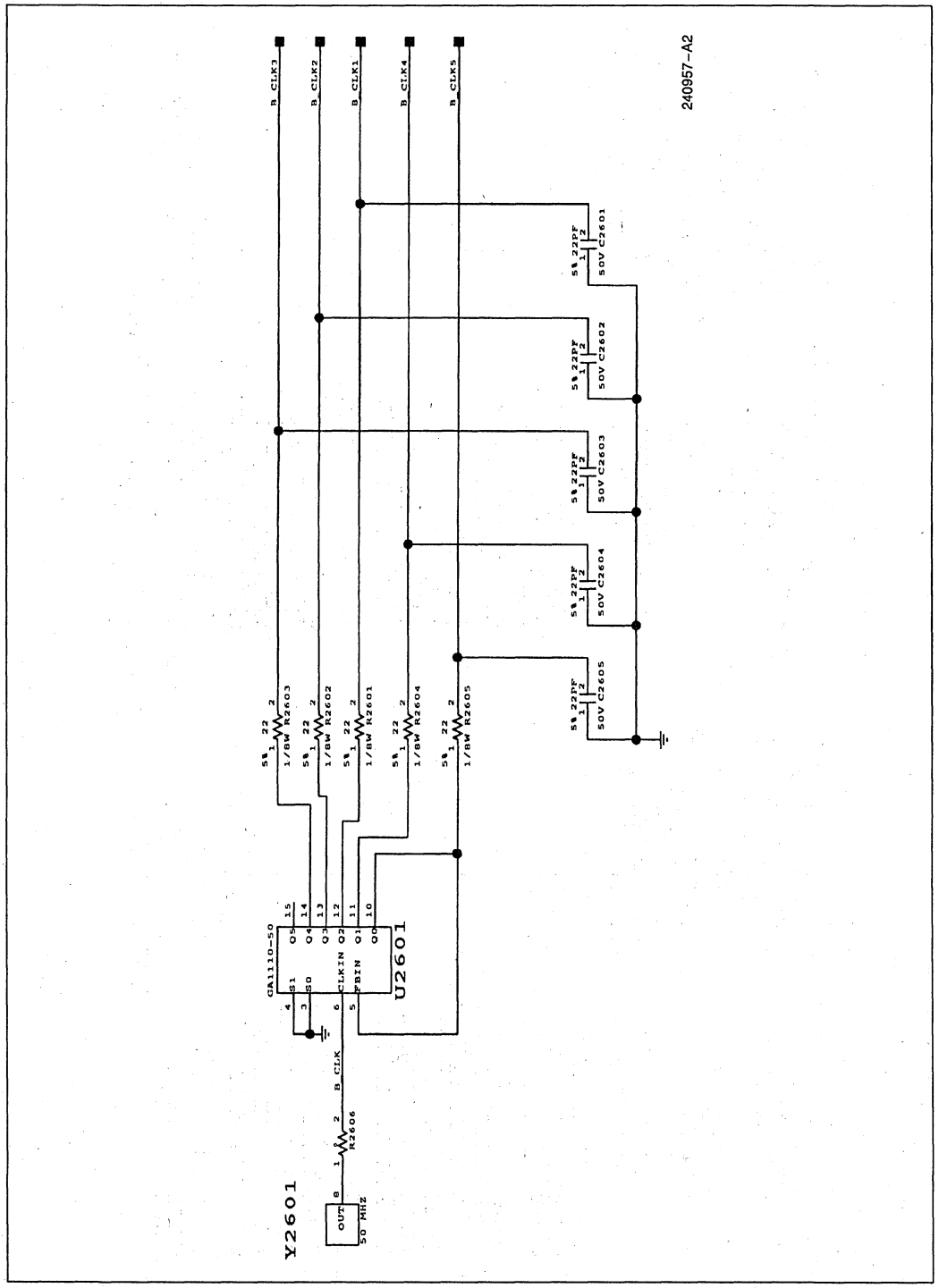
Appendix C Schematic: 82490XP Cache RAM



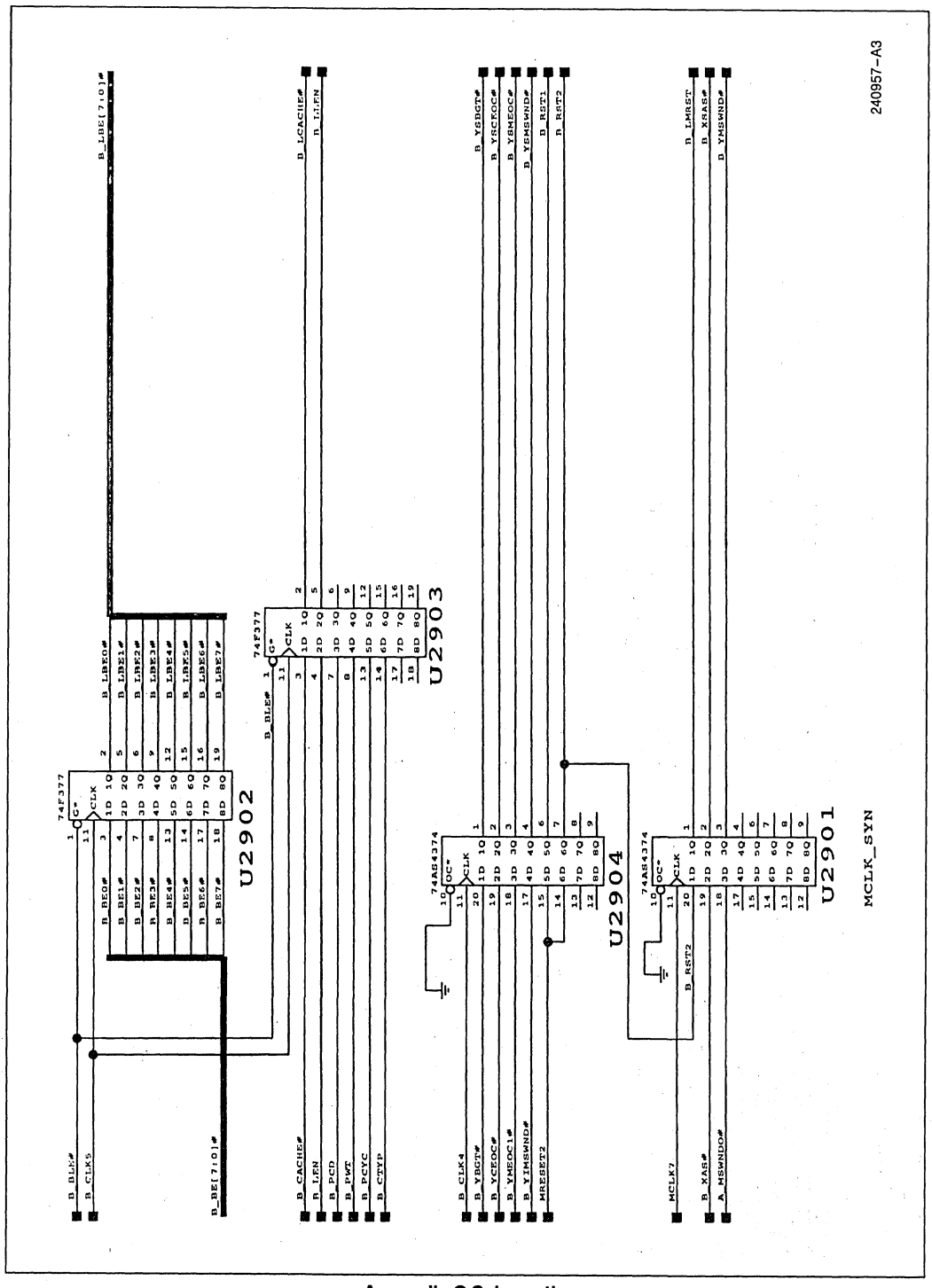
Appendix C Schematic: 82490XP Cache RAM



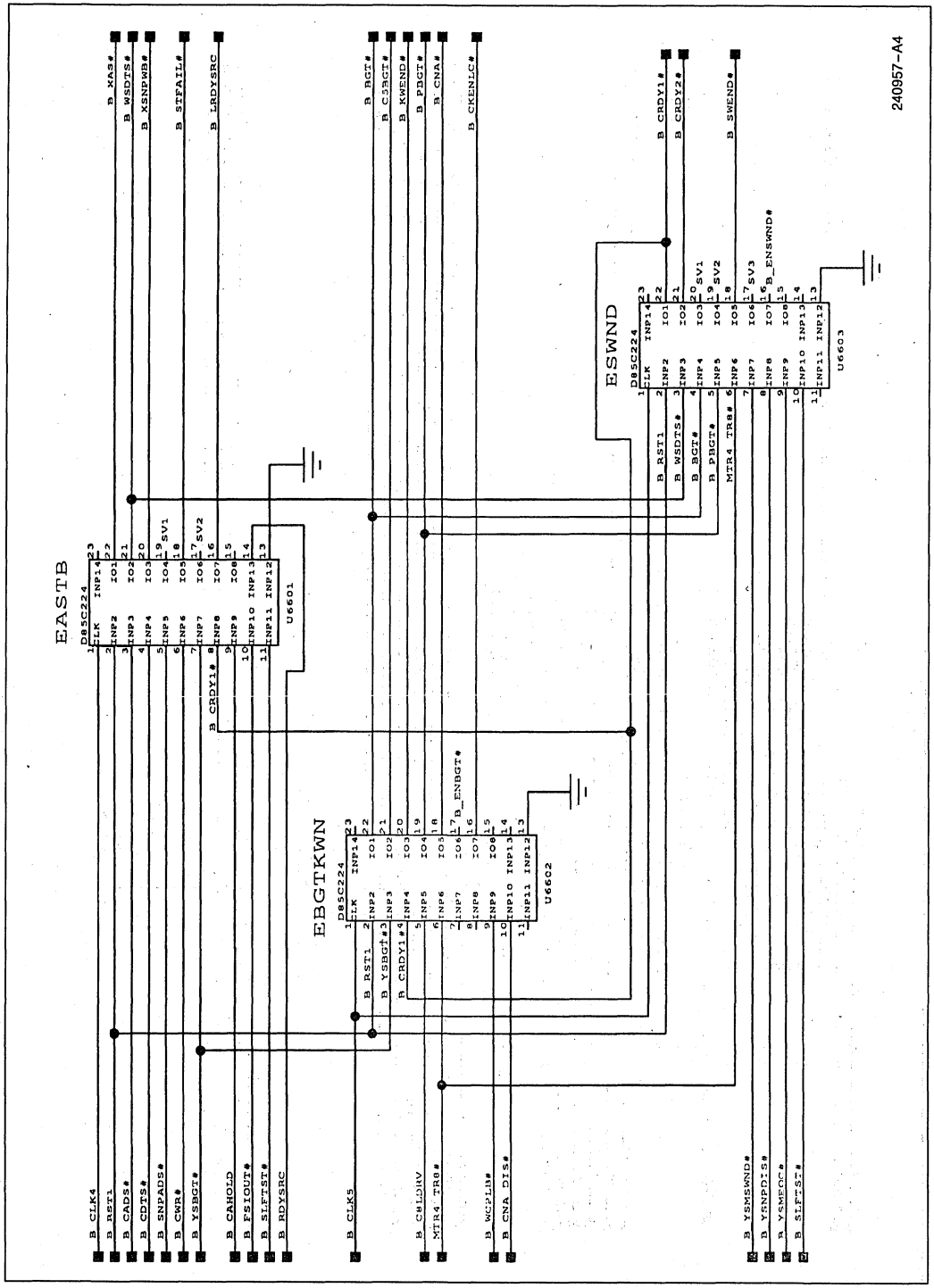
Appendix C Schematic: 82490XP Cache RAM



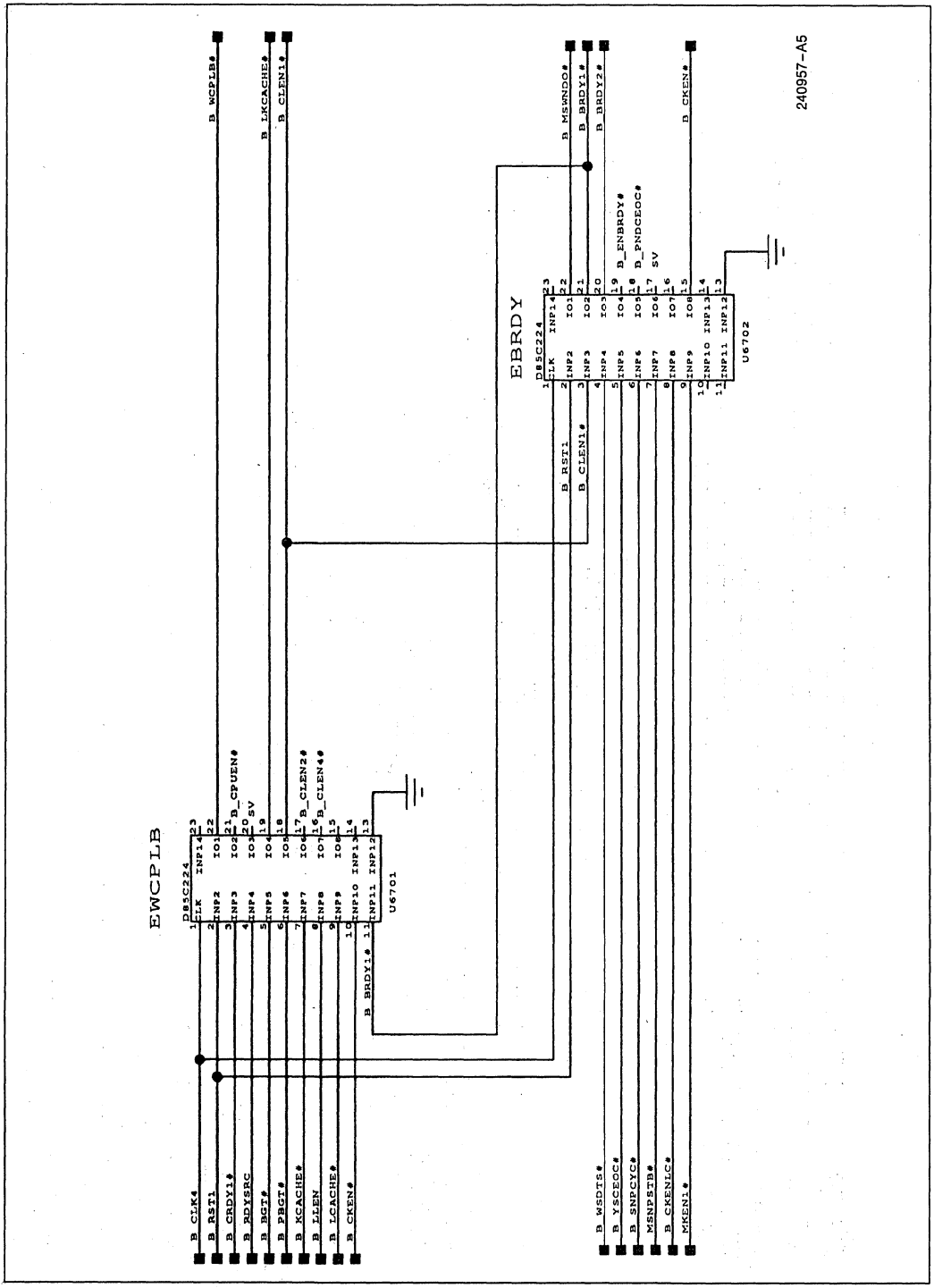
Appendix C Schematic: Clock Generator



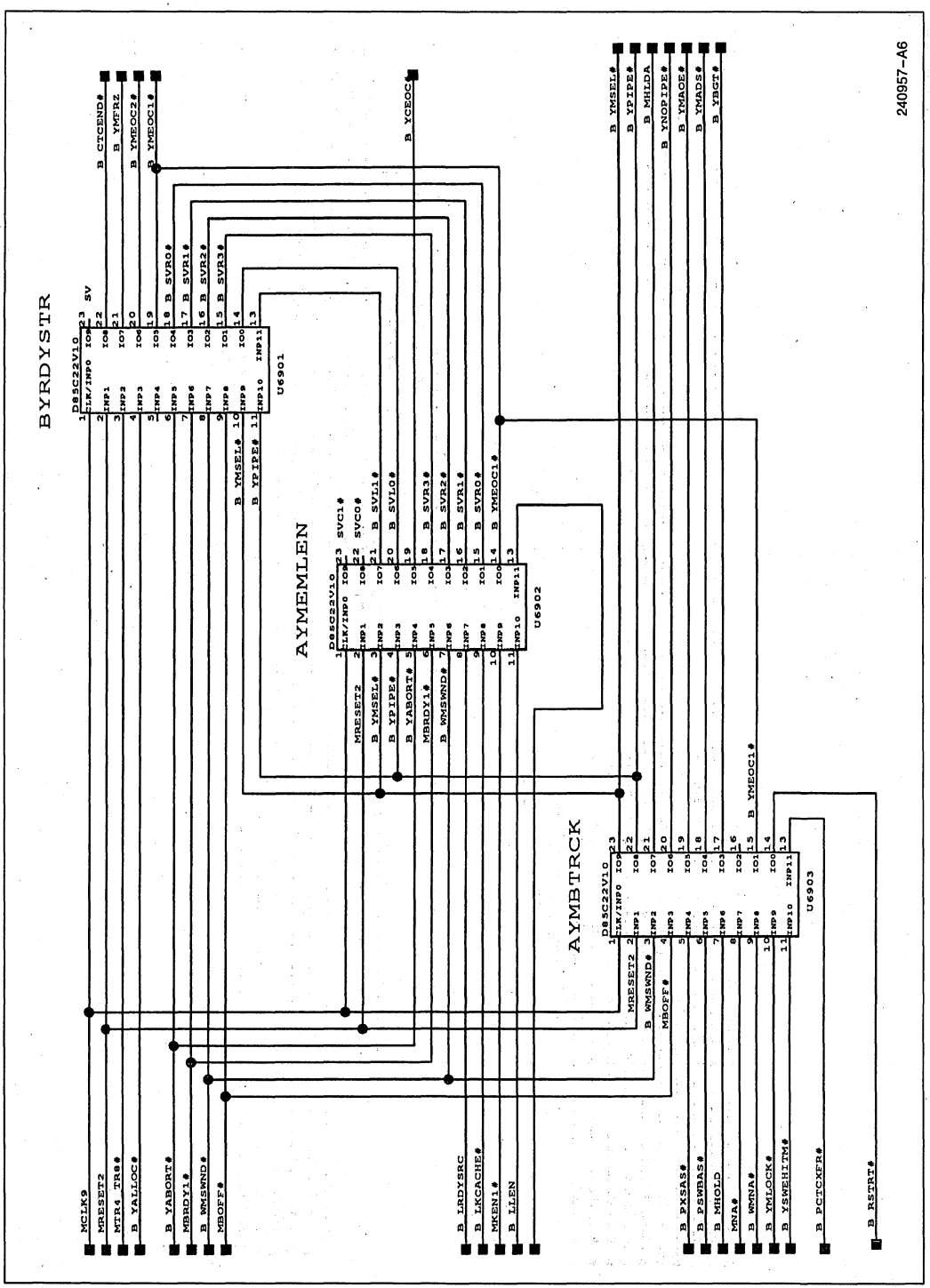
Appendix C Schematic



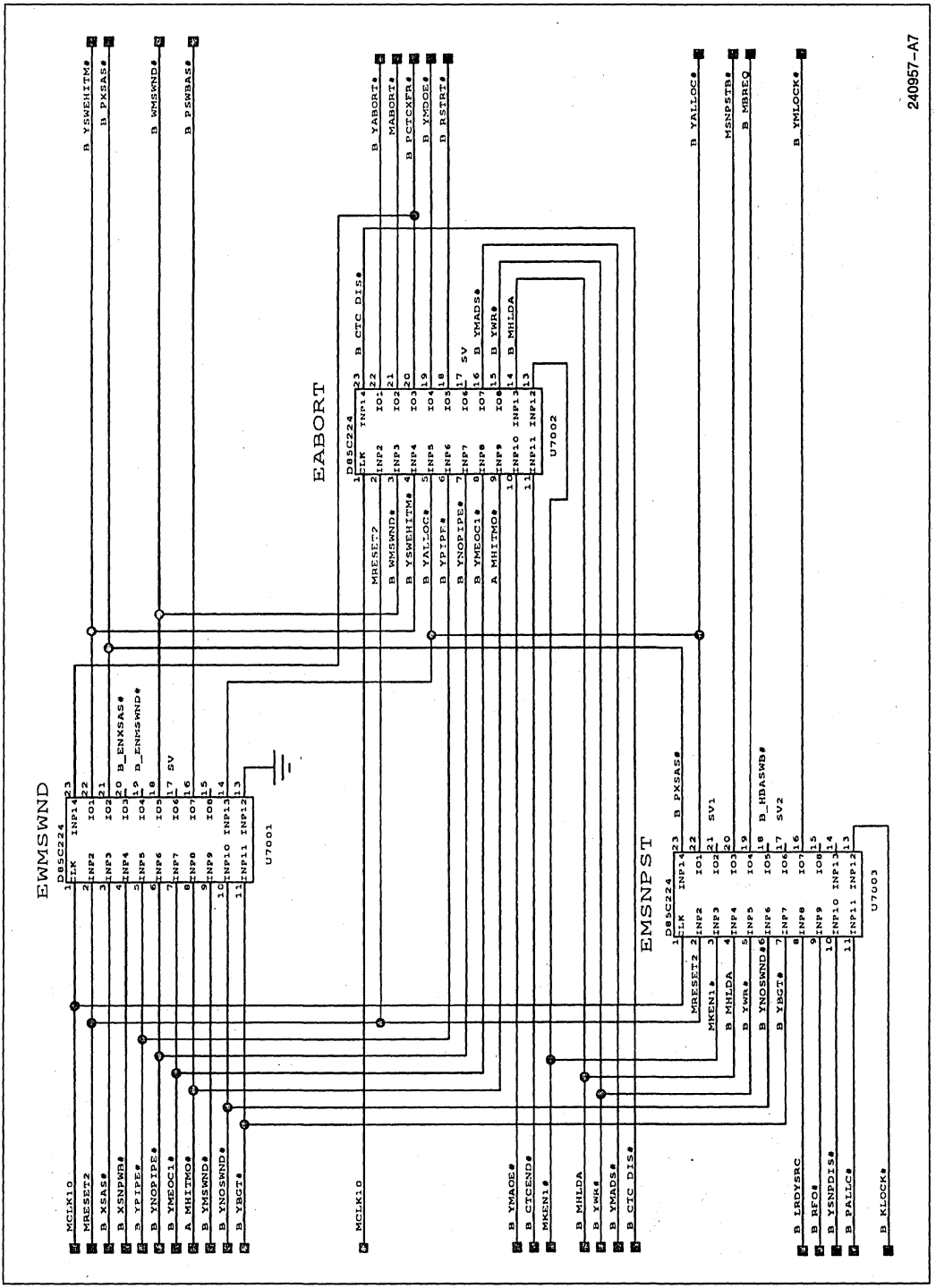
Appendix C Schematic



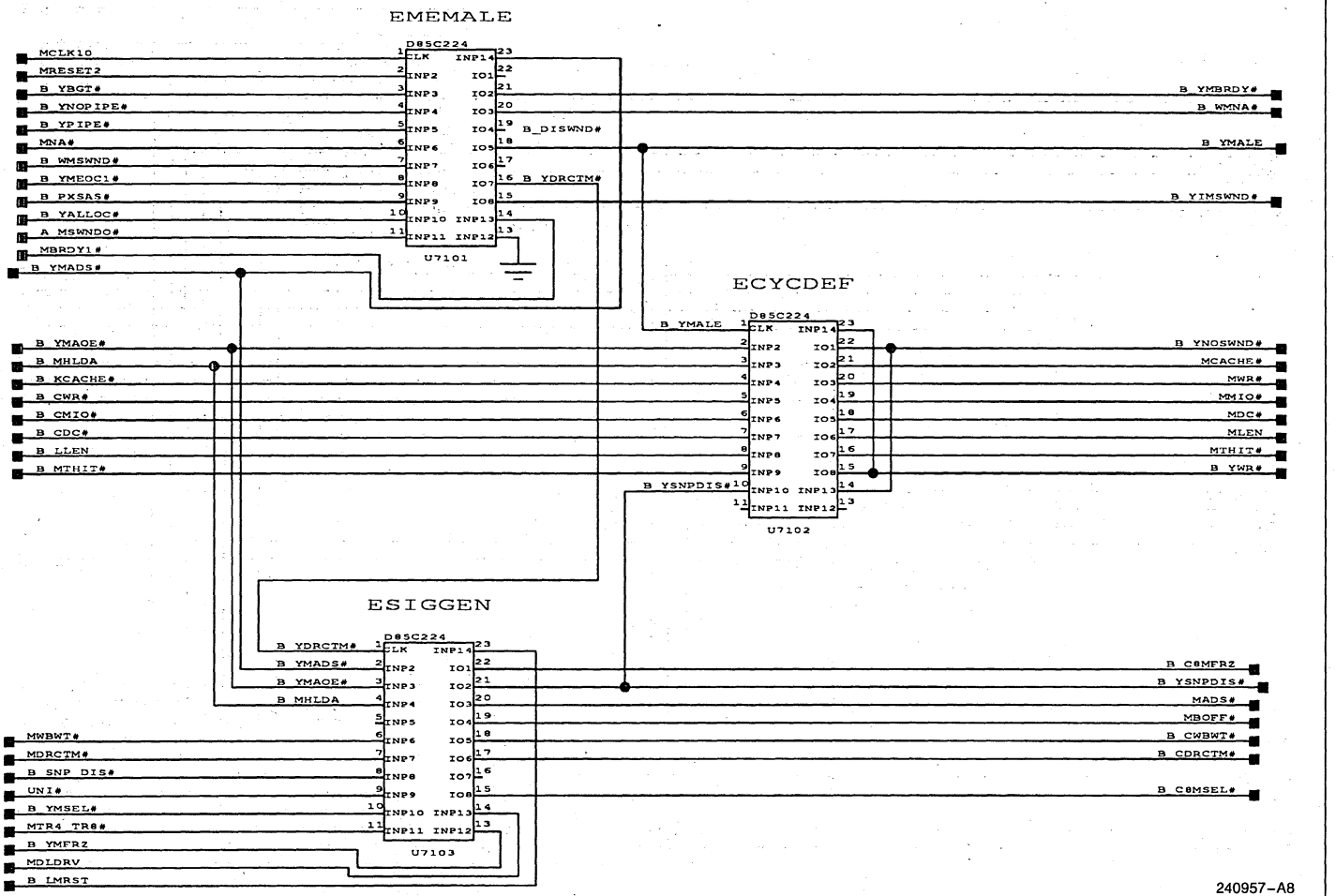
Appendix C Schematic

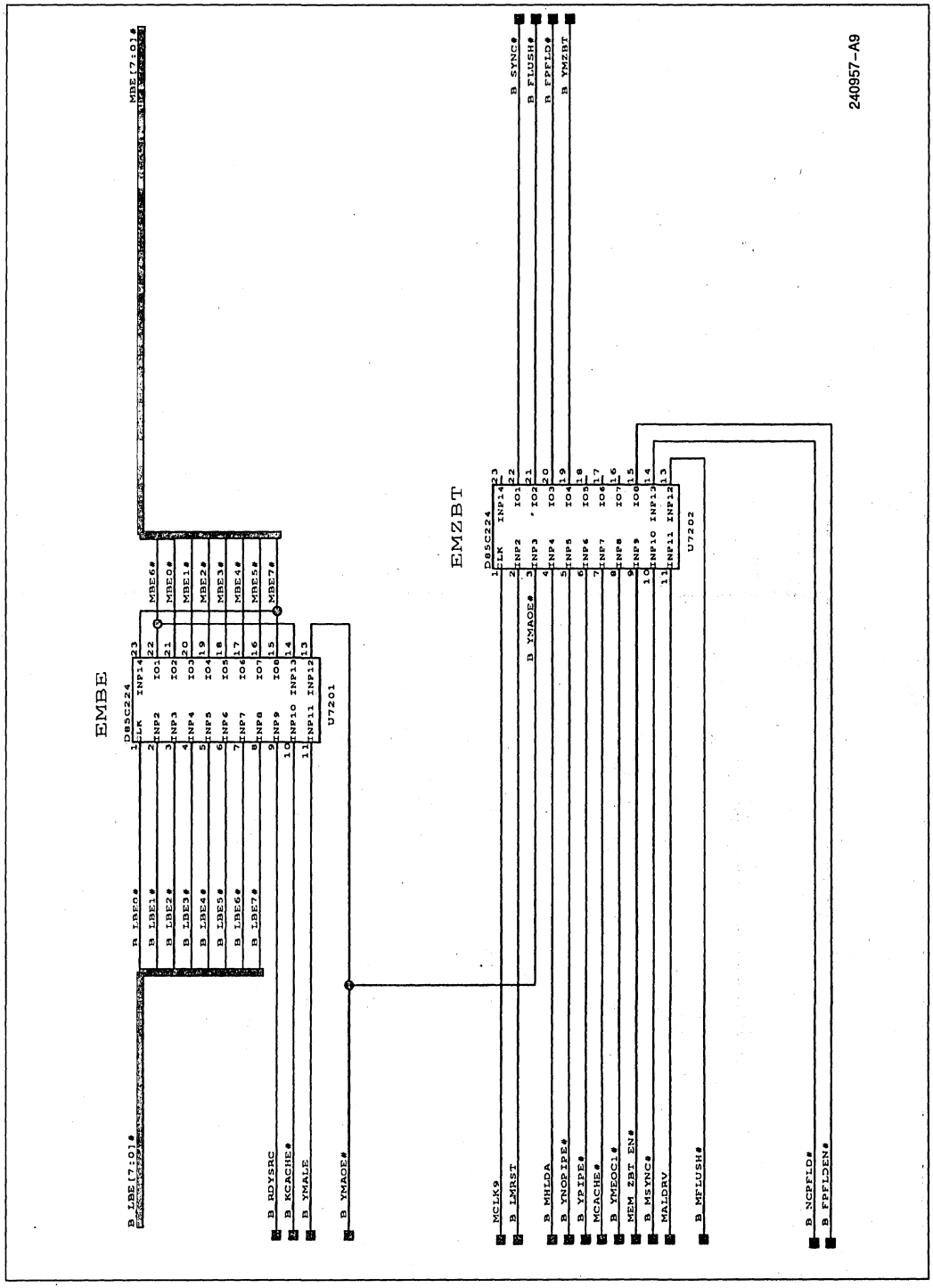


Appendix C Schematic

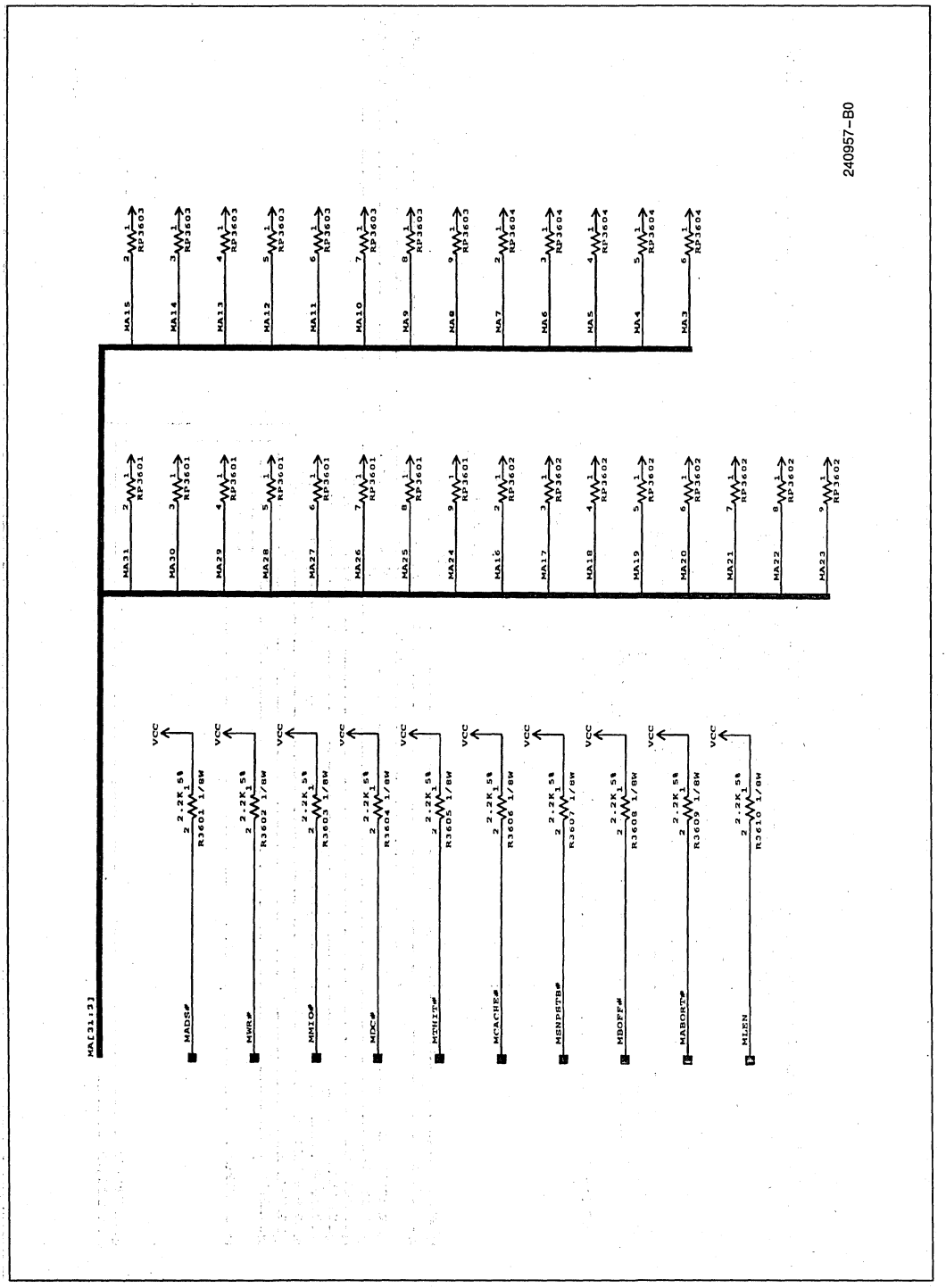


Appendix C Schematic

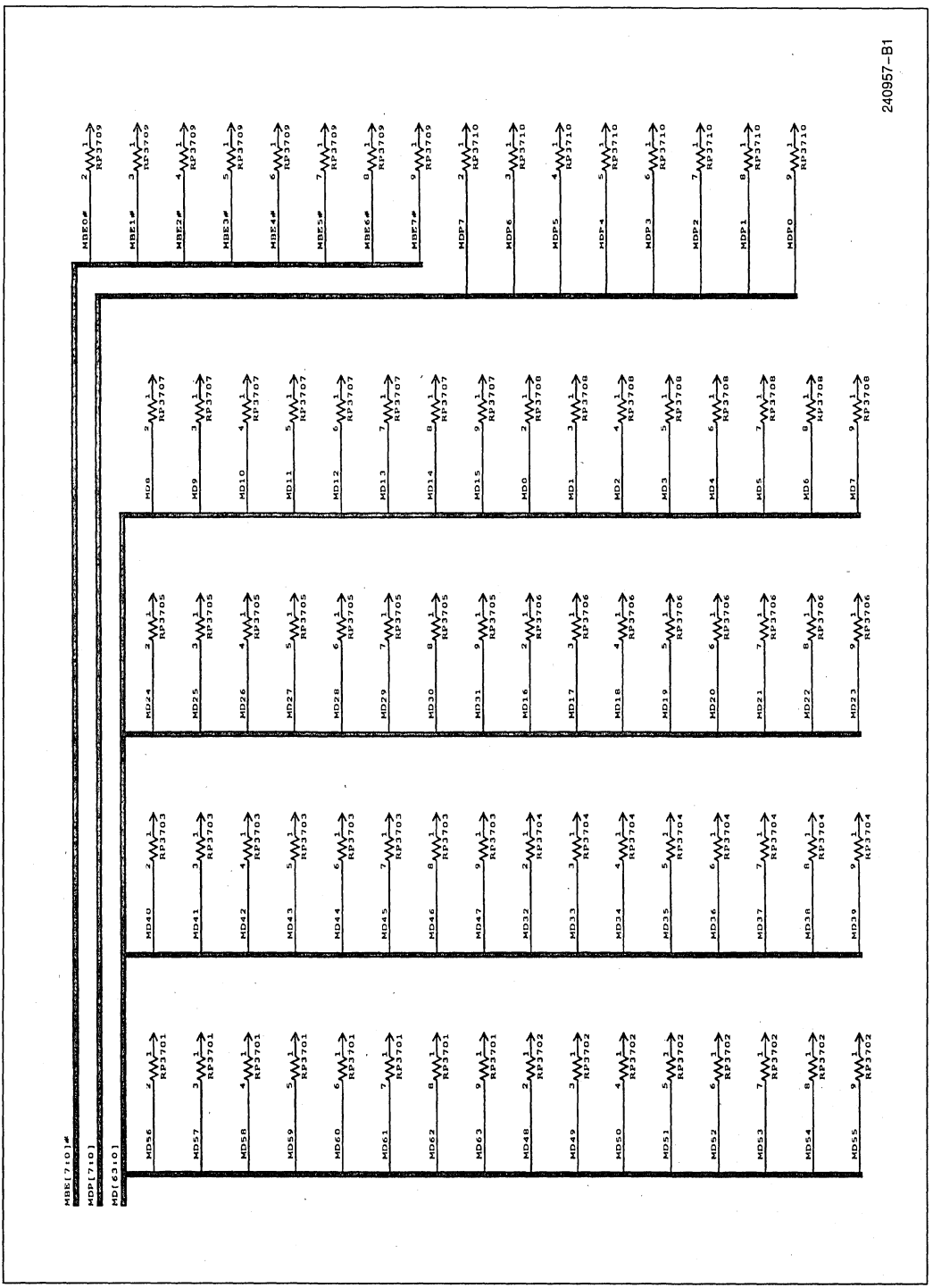

EMEMALE
D85C224
U7101
MCLK10
MRESET2
B_YBGT#
B_YNOPIPE#
B_YPIPE#
MNA#
B_WMSWND#
B_YMEOC1#
B_PXSAS#
B_YALLOC#
A_MSWNDO#
MBRDY1#
B_YMADS#
B_DISWND#
B_YDRCTM#
B_YMBRDY#
B_WMNA#
B_YMALE
B_YIMSWND#
ECYCDEF
D85C224
U7102
B_YMALE
B_YMAOE#
B_MHLDA
B_KCACHE#
B_CWR#
B_CMIO#
B_CDC#
B_LLEN
B_MTHIT#
B_YSNPDIS#
B_YNOSWND#
MCACHE#
MWR#
MMIO#
MDC#
MLEN
MTHIT#
B_YWR#
ESIGGEN
D85C224
U7103
B_YDRCTM#
B_YMADS#
B_YMAOE#
B_MHLDA
MWBWT#
MDRCTM#
B_SNP_DIS#
UNI#
B_YMSEL#
MTR4_TR8#
B_YMFRZ
MDLDRV
B_LMRST
B_C0MFRZ
B_YSNPDIS#
MADS#
MBOFF#
B_CWBWT#
B_CDRCTM#
B_C0MSEL#
240957-A8



Appendix C Schematic



Appendix C Schematic



Appendix C Schematic

i960™ Microprocessor Family 3

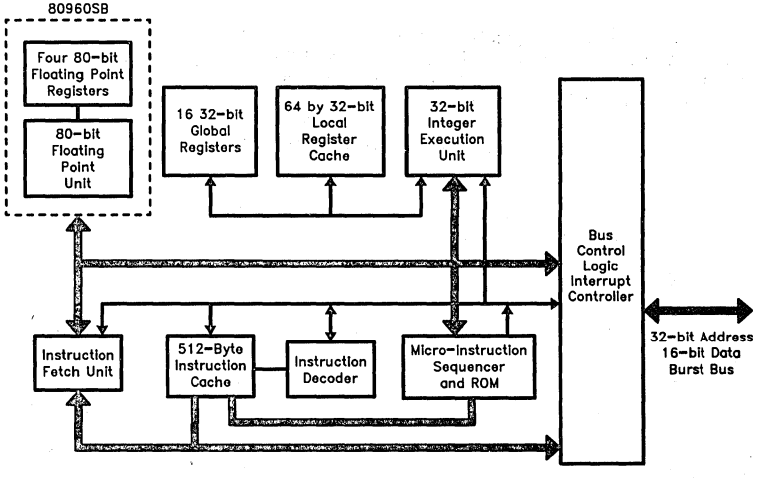
80960SA/80960SB EMBEDDED 32-BIT PROCESSORS WITH 16-BIT BURST DATA BUS

- High-Performance Embedded Architecture
  - 16 MIPS Burst Execution at 16 MHz
  - 5 MIPS* Sustained Execution at 16 MHz
- 512-Byte On-Chip Instruction Cache
  - Direct Mapped
  - Parallel Load/Decode for Uncached Instructions
- Multiple Register Sets
  - Sixteen Global 32-Bit Registers
  - Sixteen Local 32-Bit Registers
  - Four Local Register Sets Stored On-Chip
  - Register Scoreboarding
- Software Compatible with 80960KA/KB/CA Processors
- Built-In Interrupt Controller
  - 4 Direct Interrupt Pins
  - 32 Priority Levels 256 Vectors
- Built-In Floating Point Unit (80960SB only)
  - Fully IEEE 754 Compatible
- Easy to Use, High Bandwidth 16-Bit Bus
  - 25.6 Mbyte/sec Burst
  - Up to 16 Bytes Transferred per Burst
- 32-Bit Address Space, 4 Gigabytes
- 80-Lead Quad Flat Pack (EIAJ QFP)
- 84-Lead Plastic Leaded Chip Carrier (PLCC)

3

The 80960SA and 80960SB are members of Intel's i960 32-bit processor family, which are designed especially for low cost embedded applications. They are based on the family's high performance, common core architecture, and include a 512-byte instruction cache and a built-in interrupt controller. The 80960SA and 80960SB have a large register set, multiple parallel execution units and a high bandwidth, 16-bit, burst bus. Using advanced RISC technology, these high performance processors are capable of execution rates in excess of 5 million instructions per second.* The 80960SA and 80960SB are well-suited for a wide range of cost sensitive embedded applications such as laser printers, EISA and MCA adapters, disk controllers and X Terminals.

*Relative to Digital Equipment Corporation's VAX-11/780** at MIPS



270917-1

**VAX-11™ is a trademark of Digital Equipment Corporation.

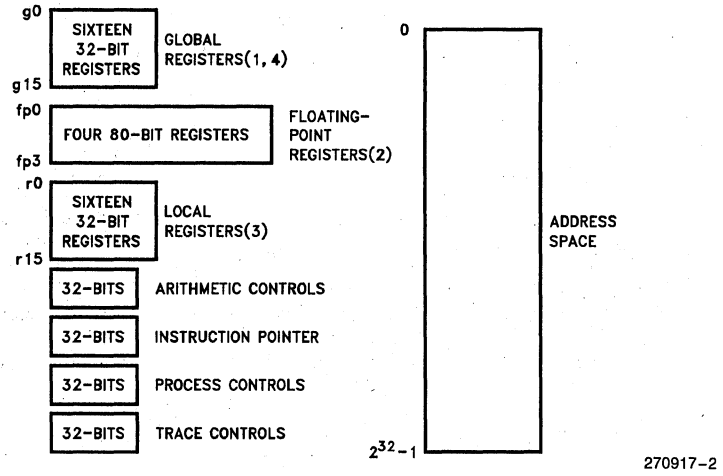
October 1991
Order Number: 270917-003

THE i960™ PROCESSOR SERIES

The 80960SA and 80960SB are members of a new family of 32-bit microprocessors from Intel known as the i960 Series. This series was especially designed to serve the needs of embedded applications. The embedded market includes applications as diverse as industrial automation, avionics, image processing, graphics, robotics, telecommunications and automobiles. These types of applications require high integration, low power consumption, quick interrupt response times and high performance. Since time to market is critical, embedded microprocessors need to be easy to use in both hardware and software designs.

All members of the 80960 series share a common core architecture which utilizes RISC technology so that, except for special functions, the family members are object code compatible. Each new processor in the series will add its own special set of functions to the core to satisfy the needs of a specific application or range of applications for the embedded market. For example, future processors may include a DMA controller, a timer or an A/D converter.

Software written for the 80960SA and 80960SB will run without modification on any other member of the 80960 family. The 80960SA is pin compatible with the 80960SB, which includes an integrated floating-point unit.



NOTES:
1. Register g15 is reserved for stack management functions.
2. Floating-Point registers and operations are available only in the 960SB and 960KB processors.
3. Registers r0, r1 and r2 are reserved for stack management functions.
4. Register g14 is used by BAL and BALX instructions.

Figure 2. 80960 Register Set

Key Performance Features

The 80960SA and 80960SB's architecture is based on the most recent advances in RISC technology and is grounded in Intel's long experience in designing embedded controllers. Many features contribute to the 80960SA and 80960SB exceptional performance.

1. Large Register Set. Modern compilers can take advantage of a large number of registers to optimize execution speed. For maximum flexibility, the 80960SA and 80960SB provide 32 32-bit registers and four 80-bit floating-point registers. (See Figure 2.)

2. Fast Instruction Execution. Simple functions make up the bulk of instructions in most programs, so that execution speed can be greatly improved by ensuring that these core instructions execute in as short a time as possible. The most-frequently executed instructions such as register-register moves, add/subtract, logical operations, and shifts execute in one to two cycles (Table 1 contains a list of instructions).

3. Load/Store Architecture. Like other processors based on RISC technology, the 80960SA and 80960SB has a Load/Store architecture. Only the LOAD and STORE instructions reference memory; all other instructions operate on registers. This type of architecture simplifies instruction decoding and is used in combination with other techniques to increase parallelism.

4. Simple Instruction Formats. All instructions in the 80960SA and 80960SB are 32 bits long and must be aligned on word boundaries. This alignment makes it possible to eliminate the instruction-alignment stage in the pipeline. To simplify the instruction decoder further, there are only five instruction formats and each instruction type uses only one format. (See Figure 3.)

5. Overlapped Instruction Execution. A load operation allows execution of subsequent instructions to continue before the data has been returned from memory, so that these instructions can overlap the load. The 80960SA and 80960SB manage this process transparently to software through the use of a register scoreboard. Conditional instructions also make use of a scoreboard so that subsequent unrelated instructions can be executed while the conditional instruction is pending.

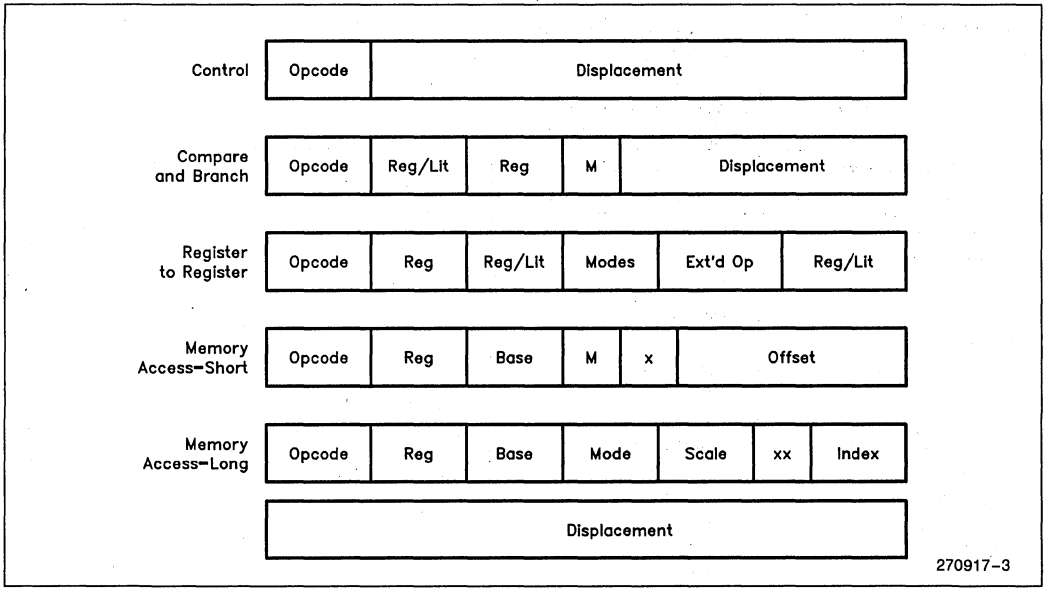


Figure 3. Instruction Formats

Table 1. 80960SA and 80960SB Instruction Set

Data Movement	Arithmetic	Logical	Bit and Bit Field
Load Store Move Load Address	Add Subtract Multiply Divide Remainder Modulo Shift Extended Multiply Extended Divide	And Not And And Not Or Exclusive Or Not Or Or Not Nor Exclusive Nor Not Nand Rotate	Set Bit Clear Bit Not Bit Check Bit Alter Bit Scan for Bit Scan over Bit Extract Modify
Comparison	**Branch**	**Call/Return**	**Fault**
Compare Conditional Compare Compare and Increment Compare and Decrement	Unconditional Branch Conditional Branch Compare and Branch	Call Call Extended Call System Return Branch and Link	Conditional Fault Synchronize Faults
Debug	**Miscellaneous**	**Decimal**	
Modify Trace Controls Mark Force Mark	Atomic Add Atomic Modify Flush Local Registers Modify Arithmetic Controls Scan Byte for Equal Test Condition Code	Move Add with Carry Subtract with Carry	
Conversion (80960SB only)	**Floating-Point (80960SB only)**	**Synchronous**	
Convert Real to Integer Convert Integer to Real	Move Real Add Subtract Multiply Divide Remainder Scale Round Square Root Sine Cosine Tangent Arctangent Log Log Binary Log Natural Exponent Classify Copy Real Extended Compare	Synchronous Load Synchronous Move	

6. Integer Execution Optimization. When the result of an operation is used as an operand in a subsequent calculation, the value is sent immediately to its destination register. Yet at the same time, the value is put back on a bypass path to the ALU, thereby saving the time that otherwise would be required to retrieve the value for the next operation.

7. Bandwidth Optimizations. The 80960SA and 80960SB get optimal use of their memory bus bandwidth because the bus is tuned for use with the cache; the line size of the instruction cache matches the maximum burst size for instruction fetches. The 80960SA and 80960SB automatically fetch four words in a burst and store them directly in the cache. Due to the size of the cache and the fact that it is continually filled in anticipation of needed instructions in the program flow, the 80960SA and 80960SB are exceptionally insensitive to memory wait states. In fact, each wait state causes only a 10% degradation in system performance. The benefit is that the 80960SA and 80960SB will deliver outstanding performance even with a low cost memory system.

8. Cache Bypass. If there is a cache miss, the processor fetches the needed instruction, then sends it on to the instruction decoder at the same time it updates the cache. Thus, no extra time is taken to load and read the cache.

Memory Space and Addressing Modes

The 80960SA and 80960SB offer a linear programming environment so that all programs running on the processors are contained in a single address space. The maximum size of the address space is 4 Gigabytes.

For ease of use, the 80960SA and 80960SB have a small number of addressing modes, but include all those necessary to ensure efficient compiler implementations of high-level languages such as C, Fortran and Ada. Table 2 lists the memory addressing modes.

Data Types

The 80960SA and 80960SB recognize the following data types:

Numeric:

- 8-, 16-, 32- and 64-bit ordinals
- 8-, 16-, 32- and 64-bit integers
- 8-, 16-, 32-, 64- and 80-bit reals

Non-Numeric:

- bit
- bit Field
- Triple-Word (96 bits)
- Quad-Word (128 bits)

Large Register Set

The following environment of the 80960SA and 80960SB include a large number of registers. In fact, 32 registers are available at any time. The availability of this many registers greatly reduces the number of memory accesses required to execute most programs, which leads to greater instruction processing speed.

There are two types of general-purpose registers: local and global. The global registers consist of sixteen 32-bit registers (G0 through G15). These registers perform the same function as the general-purpose registers provided in other popular microprocessors. The term global refers to the fact that these registers retain their contents across procedure calls.

The local registers, on the other hand, are procedure specific. For each procedure call, the 80960SA and 80960SB allocate 16 local registers (R0 through R15). Each local register is 32 bits wide.

Multiple Register Sets

To further increase the efficiency of the register set, multiple sets of local registers are stored on-chip. This cache holds up to four local register frames, which means that up to three procedure calls can be made without having to access the procedure stack resident in memory.

Table 2. Memory Addressing Modes

- 12-Bit Offset
- 32-Bit Offset
- Register-Indirect
- Register + 12-Bit Offset
- Register + 32-Bit Offset
- Register + (Index-Register x Scale-Factor)
- Register x Scale Factor + 32-Bit Displacement
- Register + (Index-Register x Scale-Factor) + 32-Bit Displacement

Scale-Factor is 1, 2, 4, 8 or 16

Although programs may have procedure calls nested many calls deep, a program typically oscillates back and forth between only two or three levels. As a result, with four stack frames in the cache, the probability of there being a free frame on the cache when a call is made is very high. In fact, runs of representative C-language programs show that 80% of the calls are handled without needing to access memory.

If there are four or more active procedures and a new procedure is called, the processor moves the oldest set of local registers in the register cache to a procedure stack in memory to make room for a new set of registers. Global register G15 is used by the processor as the frame pointer (FP) for the procedure stack.

Note that the global registers are not exchanged on a procedure call, but retain their contents, making them available to all procedures for fast parameter passing. An illustration of the register cache is shown in Figure 4.

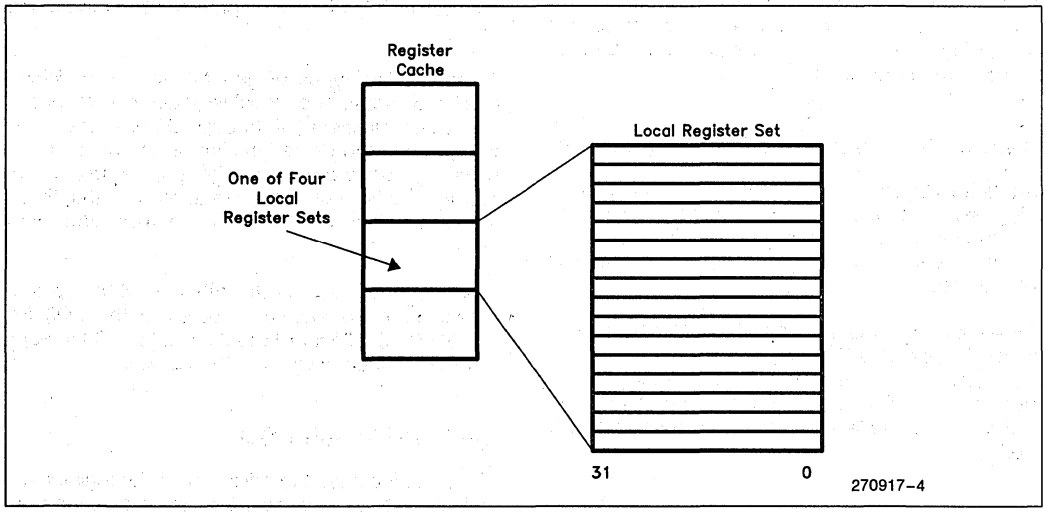


Figure 4. Multiple Register Sets are Stored On-Chip

Instruction Cache

To further reduce memory accesses, the 80960SA and 80960SB include a 512-byte on-chip instruction cache. The instruction cache is based on the concept of locality of reference; that is, most programs are not usually executed in a steady stream but consist of many branches and loops that lead to jumping back and forth within the same small section of code. Thus, by maintaining a block of instructions in a cache, the number of memory references required to read instructions into the processor can be greatly reduced.

To load the instruction cache, instructions are fetched in 16-byte blocks, so that up to four instructions can be fetched at one time.

Code for small loops will often fit entirely within the cache, leading to a great increase in processing speed since further memory references might not be necessary until the program exits the loop. Similarly, when calling short procedures, the code for the calling procedure is likely to remain in the cache, so it will be there on the procedure's return.

Register Scoreboarding

The instruction decoder has been optimized in several ways. One of these optimizations is the ability to do instruction overlapping by means of register scoreboarding.

Register scoreboarding occurs when a LOAD instruction is executed to move a variable from memory into a register. When the instruction is initiated, a scoreboard bit on the target register is set. When the register is actually loaded, the bit is reset. In between, any reference to the register contents is accompanied by a test of the scoreboard bit to insure that the load has completed before processing continues. Since the processor does not have to wait for the LOAD to be completed, it can go on to execute additional instructions placed in between the LOAD instruction and the instruction that uses the register contents, as shown in the following example:

```
LOAD address 1, R4
LOAD address 2, R5
Unrelated instruction
Unrelated instruction
ADD R4, R5, R6
```

In essence, the two unrelated instructions between the LOAD and ADD instructions are executed for free (i.e., take no apparent time to execute) because they are executed while the register is being loaded. Up to three instructions can be pending at one time with three corresponding scoreboard bits set. By exploiting this feature, system programmers and compilers have a useful tool for optimizing execution speed.

Floating-Point Arithmetic

In the 80960SB, floating-point arithmetic has been made an integral part of the architecture. Having the floating-point unit integrated on-chip provides two advantages. First, it improves the performance of the chip for floating-point applications, since no additional bus overhead is associated with floating-point calculations, thereby leaving more time for other bus operations such as I/O. Second, the cost of using floating-point operations is reduced because a separate coprocessor chip is not required.

The 80960SB floating-point (real number) data types include single-precision (32-bit), double-precision (64-bit) and extended precision (80-bit) floating-point numbers. Any register may be used to execute floating-point operations.

The processor provides hardware support for both mandatory and recommended portions of IEEE Standard 754 for floating-point arithmetic, exponential, logarithmic and other transcendental functions. Table 3 shows execution times for some representative instructions.

Table 3. Sample Floating-Point Execution Times (μs) at 16 MHz

	32-Bit	64-Bit
Add	0.6	0.8
Subtract	0.6	0.8
Multiply	1.1	2.0
Divide	2.0	4.5
Square Root	5.8	6.1
Arctangent	15.8	20.5
Exponent	17.7	19.5
Sine	23.8	25.9
Cosine	23.8	25.9

High Bandwidth Bus

The 80960SA and 80960SB CPUs reside on a high-bandwidth address/data bus. The bus provides a direct communication path between the processor and the memory and I/O subsystem interfaces. The processor uses the bus to fetch instructions, manipulate memory and respond to interrupts. Its features include:

- 16-bit data path multiplexed onto the lower bits of the 32-bit address path
- Eight 16-bit half-word burst capacity, which allows transfers from 1 to 16 bytes at a time
- High bandwidth reads and writes at 25.6 Mbytes per second

Figure 5 identifies the groups of signals which constitute the Bus. Table 4 lists the function of the Bus and other processor-support signals, such as the interrupt lines.

Interrupt Handling

The 80960SA and 80960SB can be interrupted in one of two ways: by the activation of one of four interrupt pins or by sending a message on the processor's data bus.

The 80960SA and 80960SB are unusual in that they automatically handle interrupts on a priority basis and track pending interrupts through their on-chip interrupt controller. Two of the interrupt pins can be configured to provide 8259A handshaking for expansion beyond four interrupt lines.

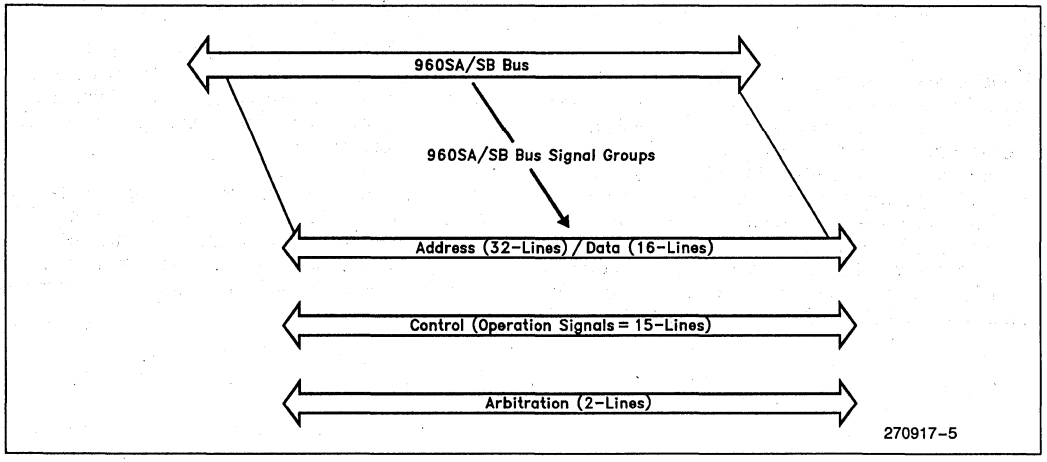


Figure 5. 80960SA and 80960SB Bus Signal Groups

Debug Features

The 80960SA and 80960SB have built-in debug capabilities. There are two types of breakpoints and six different trace modes. The debug features are controlled by two internal 32-bit registers, the Process-Controls Word and the Trace-Controls Word. By setting bits in these control words, a software debug monitor can closely control how the processor responds during program execution.

The 80960SA and 80960SB have both hardware and software breakpoints. They provide two hardware breakpoint registers on-chip which can be set by a special command to any value. When the instruction pointer matches the value in one of the breakpoint registers, the breakpoint will fire, and a breakpoint handling routine is called automatically.

Tracing is available for all instructions (single-step execution), calls and returns and branching. Each different type of trace may be enabled separately by a special debug instruction. In each case, the 80960SA and 80960SB execute the instruction first and then call a trace handling routine (usually part of a software debug monitor). Further program execution is halted until the trace routine is completed. When the trace event handling routine is completed, instruction execution resumes at the next instruction. The 80960SA and 80960SB's tracing mechanisms, which are implemented completely in hardware, greatly simplify the task of testing and debugging software.

FAULT DETECTION

The 80960SA and 80960SB have an automatic mechanism to handle faults. There are ten fault types including trace, arithmetic, and floating-point faults. When the processor detects a fault, it automatically calls the appropriate fault handling routine and saves the current instruction pointer and necessary state information to make efficient recovery possible. The processor posts diagnostic information on the type of fault to a Fault Record. Like interrupt handling routines, fault handing routines are usually written to meet the needs of a specific application and are often included as part of the operating system or kernel.

For each of the ten fault types, there are numerous subtypes that provide specific information about a fault. For example, a floating-point fault may have its subtype set to an Overflow or Zero-Divide fault. The fault handler can use this specific information to respond correctly to the fault.

BUILT-IN TESTABILITY

Upon reset, the 80960SA and 80960SB automatically conducts an extensive internal test (self-test) of its major blocks of logic. Then, before executing its first instruction, it does a zero check sum on the first eight words in memory to ensure that the system has been loaded correctly. If a problem is discovered at any point during the self-test, the 80960SA and 80960SB will indicate a failure and will not begin program execution. The self-test takes approximately 47,000 cycles to complete, and can be disabled.

System manufacturers can use the 80960SA and 80960SB's self-test feature during incoming parts inspection. No special diagnostic programs need to be written, and the test is both thorough and fast. The self-test capability helps ensure that defective parts will be discovered before systems are shipped, and once in the field, the self-test makes it easier to distinguish between problems caused by processor failure and problems resulting from other causes.

CHMOS

The 80960SA and 80960SB are fabricated using Intel's CHMOS IV (Complementary High Speed Metal Oxide Semiconductor) process. This advanced technology eliminates the frequency and reliability limitations of older CMOS processes and opens a new era in microprocessor performance. It combines the high performance capabilities of Intel's industry-leading HMOS technology with the high density and low power characteristics of CMOS. The 80960SA and 80960SB are available at 10 MHz in both PLCC and QFP packages, and at 16 MHz in the PLCC package.

Table 4. 80960SA and 80960SB Pin Description: Bus Signals

Symbol	Type	Name and Function
CLK2	I	**SYSTEM CLOCK** provides the fundamental timing for 80960SA and 80960SB systems. CLK2 is divided by two inside the 80960SA and 80960SB to generate the internal processor clock.
A31–A16	O T.S.	**ADDRESS BUS** carries the upper 16 bits of the 32-bit address to memory. It is valid throughout the burst cycle, no latch is required.
AD15–AD1, D0	I/O T.S.	**ADDRESS/DATA BUS** carries the low order 32-bit addresses and 16-bit data to and from memory. AD15–AD4 must be latched since the cycle following the address cycle carries data on the bus.
A3–A1	O T.S.	**ADDRESS BUS** carries the word addresses of the 32-bit address to memory. These three bits are incremented during a burst access indicating the next word address of the burst access. Note that A3–A1 are duplicated with AD3–AD1 during the address cycle.
ALE	O T.S.	**ADDRESS LATCH ENABLE** indicates the transfer of a physical address. ALE is asserted during a Ta cycle and deasserted before the beginning of the following Td state. It is active high and floats to a high impedance state during a hold cycle (Th or Thr).
$\overline{AS}$	O T.S.	**ADDRESS STATUS** indicates an address state. $\overline{AS}$ is asserted every Ta state and deasserted during the following Td state. $\overline{AS}$ is driven HIGH during reset.
W/$\overline{R}$	O T.S.	**WRITE/READ** specifies, during a Ta cycle, whether the operation is write or read. It is latched on-chip and remains valid during Td cycles.
$\overline{DEN}$	O T.S.	**DATA ENABLE** is asserted during Td cycles and indicates transfer of data on the AD lines. The AD lines should not be driven by an external source unless $\overline{DEN}$ is asserted. When $\overline{DEN}$ is asserted, the outputs from the previous cycle are guaranteed to be 3-stated. In addition, $\overline{DEN}$ deasserted indicates inputs have been captured and therefore input hold times can be disregarded. $\overline{DEN}$ is driven to a HIGH during reset.
READY	I	**READY** indicates that data on AD lines can be sampled or removed. If $\overline{READY}$ is not asserted during a Td cycle the Td cycle is extended to the next cycle by inserting a wait state (Tw).
DT/$\overline{R}$	O T.S.	**DATA TRANSMIT/RECEIVE** indicates the direction of the data transfer to and from the bus. It is low during Ta and Td cycles for a read or interrupt acknowledgement; it is high during Ta and Td cycles for a write. DT/$\overline{R}$ never changes state when $\overline{DEN}$ is asserted. DT/$\overline{R}$ is driven HIGH during reset.
$\overline{BLAST}$/$\overline{FAIL}$	O T.S.	**BURST LAST** indicates the last data cycle (Td) of a burst access. It is asserted low during the last Td and associated Tw cycles in a burst access. **INITIALIZATION FAILURE** indicates that the processor has failed to initialize correctly. The failure state is indicated by a combination of $\overline{BLAST}$ asserted and both $\overline{BE}$ signals not asserted. This condition occurs after $\overline{RESET}$ is deasserted and before the first bus transaction begins. $\overline{FAIL}$ is asserted while the processor performs a self-test. If the self-test completes successfully, then $\overline{FAIL}$ is deasserted. Next, the processor performs a zero checksum on the first eight words of memory. If it fails, $\overline{FAIL}$ is asserted for a second time and remains asserted; if it passes, system initialization continues and $\overline{FAIL}$ remains deasserted.

I/O = Input/Output, O = Output, I = Input, O.D. = Open-Drain, T.S. = 3-State.

Table 4. 80960SA and 80960SB Pin Description: Bus Signals (Continued)

Symbol	Type	Name and Function
$\overline{\text{RESET}}$	I	**RESET** clears the internal logic of the processor and causes it to reinitialize. During $\overline{\text{RESET}}$ assertion, the input pins are ignored (except for $\overline{\text{INT0}}$, $\overline{\text{INT1}}$, $\overline{\text{INT3}}$, $\overline{\text{LOCK}}$), the tri-state output pins are placed in a HIGH impedance state (except for DT/$\overline{\text{R}}$, $\overline{\text{DEN}}$, and $\overline{\text{AS}}$), and other output pins are placed in their non-asserted state. $\overline{\text{RESET}}$ must be asserted for at least 41 CLK2 cycles for a predictable reset. Optionally, for a synchronous reset, the LOW to HIGH transition of $\overline{\text{RESET}}$ should occur after the rising edge of both CLK2 and the external bus clock, and before the next rising edge of CLK2. The interrupt pins indicate the initializtion sequence executed. Typical initialization requires driving only $\overline{\text{INT0}}$ and $\overline{\text{INT3}}$ to a HIGH state. The reset conditions follow:

$\overline{\text{INT0}}$	INT1	$\overline{\text{INT3}}$	$\overline{\text{LOCK}}$	Action Taken
1	x	1	1	Run self-test (core initialization)
0	0	1	1	Disable self-test
0	1	x	x	Reserved
x	x	0	x	Reserved
x	x	x	0	ONCE mode (see LOCK pin)

$\overline{\text{BE1}}$–$\overline{\text{BE0}}$	O T.S.	**BYTE ENABLE LINES** specify which data bytes (up to two) on the bus take part in the current bus cycle. $\overline{\text{BE1}}$ corresponds to AD15–AD8 and $\overline{\text{BE0}}$ corresponds to AD7–AD1, D0. The byte enable lines are asserted appropriately during each data cycle. **INITIALIZATION FAILURE** indicates that the processor has failed to initialize correctly. The failure state is indicated by a combination of $\overline{\text{BLAST}}$ asserted and both $\overline{\text{BE}}$ signals not asserted. This condition occurs after $\overline{\text{RESET}}$ is deasserted and before the first bus transaction begins. $\overline{\text{FAIL}}$ is asserted while the processor performs a self-test. If the self-test completes successfully, then $\overline{\text{FAIL}}$ is deasserted. Next, the processor performs a zero checksum on the first eight words of memory. If it fails, $\overline{\text{FAIL}}$ is asserted for a second time and remains asserted; if it passes, system initialization continues and $\overline{\text{FAIL}}$ remains deasserted.
$\overline{\text{INT0}}$	I	**INTERRUPT 0** indicates a pending interrupt. The bus interrupt control register determines in which way the signal should be interpreted. To signal an interrupt request in a synchronous system, this pin (as well as the other interrupt pins) must be enabled by being deasserted for at least one bus cycle and then asserted for at least one additional bus cycle; in an asynchronous system, the pin must remain deasserted for at least two bus cycles and then be asserted for at least two more bus cycles. $\overline{\text{INT0}}$ is sampled during RESET to determine if the self-test sequence is to be executed.
INT1	I	**INTERRUPT 1** indicates a direct interrupt, like $\overline{\text{INT0}}$. INT1 is sampled during RESET to determine if the self-test sequence is to be executed.
INT2/INTR	I	**INTERRUPT 2/INTERRUPT REQUEST:** The interrupt control register determines how this pin is interpreted. If INT2, it has the same interpretation as the $\overline{\text{INT0}}$ and INT1 pins. If INTR, it is used to receive an interrupt request from an external 8259A compatible interrupt controller.
$\overline{\text{INT3}}$/$\overline{\text{INTA}}$	I/O T.S.	**INTERRUPT 3/INTERRUPT ACKNOWLEDGE:** The interrupt control register determines how this pin is interpreted. If $\overline{\text{INT3}}$, it has the same interpretation as the $\overline{\text{INT0}}$ and INT1 pins. If INTA, it is used as an output to control interrupt-acknowledge bus transactions. The INTA output is latched on-chip and remains valid during Td cycles. $\overline{\text{INT3}}$ must be pulled to a HIGH state during RESET.

I/O = Input/Output, O = Output, I = Input, O.D. = Open-Drain, T.S. = 3-State.

Table 4. 80960SA and 80960SB Pin Description: Bus Signals (Continued)

Symbol	Type	Name and Function
$\overline{LOCK}$	I/O O.D.	**BUS LOCK** prevents other bus masters from gaining control of the bus following the current cycle (if they would assert $\overline{LOCK}$ to do so). $\overline{LOCK}$ is used by the processor or any bus agent when it performs indivisible Read/Modify/Write (RMW) operations. Do not leave $\overline{LOCK}$ unconnected. It must be pulled HIGH for the processor to function properly. For a read that is designated as an RMW-read, $\overline{LOCK}$ is examined. If asserted, the processor waits until it is not asserted; if not asserted, the processor asserts $\overline{LOCK}$ during the Ta cycle and leaves it asserted. A write that is designated as an RMW-write deasserts $\overline{LOCK}$ in the Ta cycle. During the time $\overline{LOCK}$ is asserted, a bus agent can perform a normal read or write but no RMW operations. $\overline{LOCK}$ is also held asserted during an interrupt-acknowledge transaction. **ONCE MODE:** The $\overline{LOCK}$ pin is sampled during reset. If it is asserted LOW at the end of RESET, all outputs will be 3-stated until the part is reset. ONCE MODE is used in conjunction with an ICE.
HOLD	I	**HOLD:** HOLD indicates a request from a secondary bus master to acquire the bus. When the processor receives HOLD and grants another master control of the bus, it floats its tri-state bus lines and then asserts HLDA and enters the Th state. When HOLD is deasserted, the processor will deassert HLDA and go to either the Ti or Ta state.
HLDA	O T.S.	**HOLD ACKNOWLEDGE:** HLDA indicates that bus control has been relinquished to another bus master. This signal is always driven. At RESET it is driven LOW.
N.C.	N/A	**NOT CONNECTED** indicates pins should not be connected. Never connect any pin marked N.C.

I/O = Input/Output, O = Output, I = Input, O.D. = Open-Drain, T.S. = 3-State.

ELECTRICAL SPECIFICATIONS

Power and Grounding

The 80960SA and 80960SB are implemented in CHMOS IV technology and have modest power requirements. Their high clock frequency and numerous output buffers (address/data, control, error, and arbitration signals) can cause power surges as multiple output buffers drive new signal levels simultaneously. For clean on-chip power distribution at high frequency, 12 V_{CC} and 13 V_{SS} pins separately feed functional units of the 80960SA and 80960SB in the package.

Power and ground connections must be made to all power and ground pins of the 80960SA and 80960SB. On the circuit board, all V_{CC} pins must be strapped closely together, preferably on a power plane. Likewise, all V_{SS} pins should be strapped together, preferably on a ground plane. These pins may not be connected together within the chip.

Power Decoupling Recommendations

Liberal decoupling capacitance should be placed near the 80960SA and 80960SB. The processor can cause transient power surges when driving the bus, particularly when it is connected to a large capacitive load.

Low inductance capacitors and interconnects are recommended for best high frequency electrical performance. Inductance can be reduced by shortening the board traces between the processor and decoupling capacitors as much as possible.

Connection Recommendations

For reliable operation, always connect unused inputs to an appropriate signal level. In particular, if one or more interrupt lines are not used, they should be deasserted. No inputs should ever be left floating.

All open-drain outputs require a pullup device. While in some cases a simple pullup resistor will be adequate, we recommend a network of pullup and pulldown resistors biased to a valid V_{IH} ($\geq$2.0V) and terminated in the characteristic impedance of the circuit board. Figure 6 shows our recommendations for the resistor values for both a low and high current drive network, which assumes that the circuit board has a characteristic impedance of 100Ω. The advantage of terminating the output signals in this fashion is that it limits signal swing and reduces AC power consumption.

Characteristic Curves

The 80960SA and 80960SB characteristic curves shown in Figures 7 through 10 supply information regarding typical supply currents, typical current versus frequency, worst case voltage versus output current on open drain pins and capacitive derating curves.

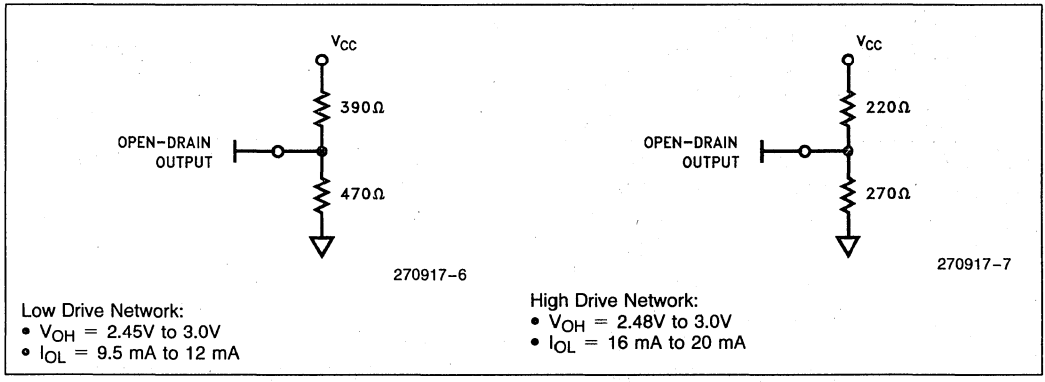
Figure 7 shows the typical supply current requirements over the operating temperture range of the processor at supply voltage (V_{CC}) of 5V. Figure 8 shows the typical power supply current (I_{CC}) required by the 80960SA and 80960SB at various operating frequencies when measured at three input voltage (V_{CC}) levels.

For a given output current (I_{OL}), the curve in Figure 9 shows the worst case output low voltage (V_{OL}). Figure 10 shows the typical capacitive derating curve for the 80960SA and 80960SB measured from 1.5V on the system clock (CLK) to 0.8V on the falling edge and 2.0V on the rising edge of the bus address/data (AD) signals.

Test Load Circuit

Figure 11 illustrates load circuit used to test the 80960SA and 80960SB's 3-state pins, and Figure 12 shows the load circuit used to test the open drain output. The open drain test uses an active load circuit in the form of a matched diode bridge. Since the open-drain output sinks current, only the I_{OL} legs of the bridge are necessary and the I_{OH} legs are not used. When the 80960SA and 80960SB driver under test is turned off, the output pin is pulled up to V_{REF} (i.e., V_{OH}). Diode D1 is turned off and the I_{OL} current source flows through diode D2.

When the 80960SA and 80960SB open-drain driver under test is on, diode D1 is also on, and the voltage on the pin being tested drops to V_{OL}. Diode D2 turns off and I_{OL} flows through diode D1.



Low Drive Network:
- V_{OH} = 2.45V to 3.0V
- I_{OL} = 9.5 mA to 12 mA

High Drive Network:
- V_{OH} = 2.48V to 3.0V
- I_{OL} = 16 mA to 20 mA

Figure 6. Open Drain Connection Recommendations for Low and High Current Drive Networks for the LOCK Pin

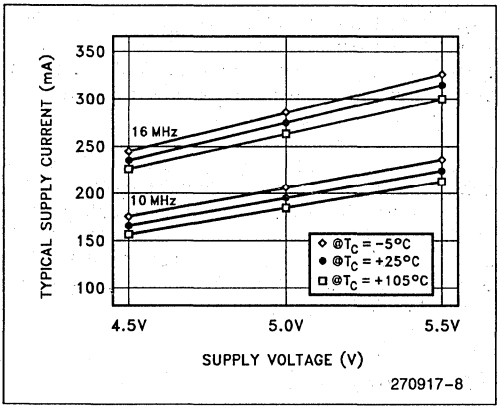


Figure 7. Typical Supply Current vs Supply Voltage

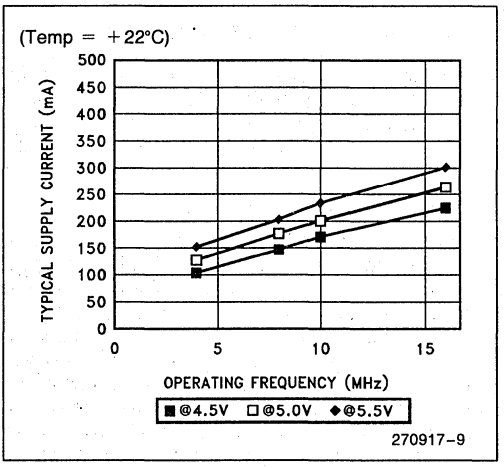


Figure 8. Typical Current vs Frequency

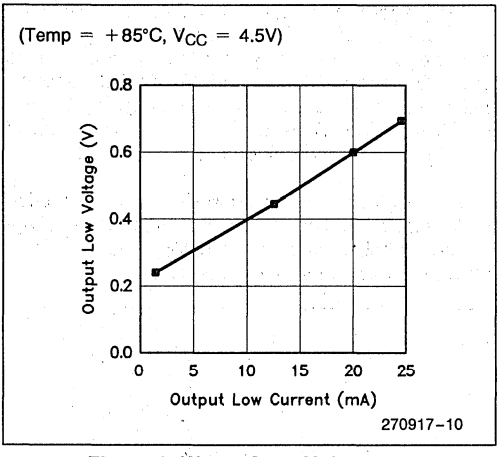


Figure 9. Worst Case Voltage vs Output Current on Open-Drain Pin

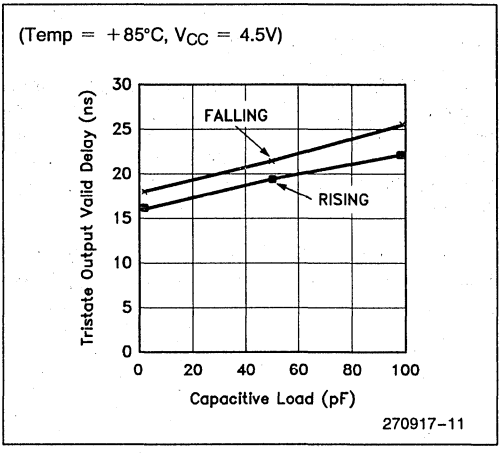


Figure 10. Capacitive Derating Curve

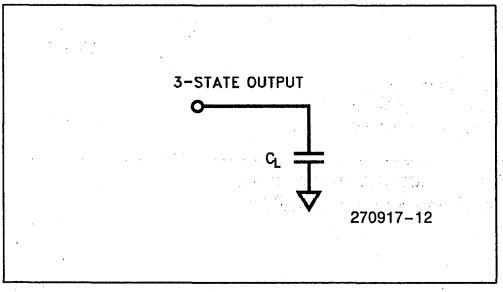


Figure 11. Test Load Circuit for 3-State Output Pins

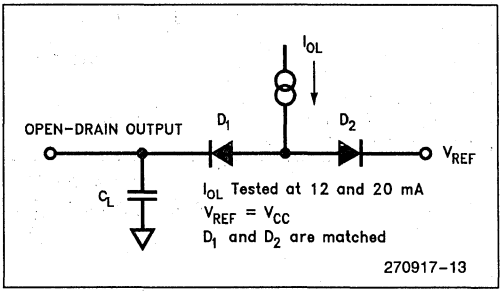


Figure 12. Test Load Circuit for Open-Drain Output Pins

ABSOLUTE MAXIMUM RATINGS

Operating Temperature (PLCC) . . . . . 0°C to +100°C Case
Operating Temperature (QFP) . . . . . 0°C to +100°C Case
Storage Temperature . . . . . −65°C to +150°C
Voltage on Any Pin (PLCC) . . . −0.5V to V_{CC} + 0.5V
Voltage on Any Pin (QFP) . . −0.25V to V_{CC} + 0.25V
Power Dissipation . . . . . 1.9W (16 MHz)

NOTICE: This data sheet contains preliminary information on new products in production. The specifications are subject to change without notice. Verify with your local Intel Sales office that you have the latest data sheet before finalizing a design.

**WARNING: Stressing the device beyond the "Absolute Maximum Ratings" may cause permanent damage. These are stress ratings only. Operation beyond the "Operating Conditions" is not recommended and extended exposure beyond the "Operating Conditions" may affect device reliability.*

DC CHARACTERISTICS

960SA/SB (10 MHz and 16 MHz): T_{CASE} = 0°C to +100°C, V_{CC} = 5V ±10% unless otherwise noted.

Symbol	Parameter	Min	Max	Units	Conditions
V_{IL}	Input Low Voltage	−0.3	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.3	V	
V_{CL}	CLK2 Input Low Voltage	−0.3	+0.8	V	
V_{CH}	CLK2 Input High Voltage	0.7 V_{CC}	V_{CC} + 0.3	V	
V_{OL}	Output Low Voltage		0.45 0.45 0.60	V V V	I_{OL} = 2.5 mA I_{OL} = 12 mA, $\overline{LOCK}$ Pin I_{OL} = 20 mA, $\overline{LOCK}$ Pin
V_{OH}	Output High Voltage	2.4		V	All TS, −2.5 mA(4)
I_{CC}	Power Supply Current: 10 MHz—QFP 10 MHz—PLCC 16 MHz—PLCC		 280 280 350	 mA mA mA	 T_{CASE} = 0°C(1) T_{CASE} = 0°C T_{CASE} = 0°C
I_{LO}	Output Leakage Current		±15	μA	(Note 5)
I_{LI}	Input Leakage Current		±15	μA	0 ≤ V_O ≤ V_{CC}(2)
C_{IN}	Input Capacitance		10	pF	fc = 1 MHz(3)
C_O	I/O or Output Capacitance		12	pF	fc = 1 MHz(3)
C_{CLK}	Clock Capacitance		10	pF	fc = 1 MHz(3)

NOTES:
1. T_{CASE} is specified at 0°C to +100°C for the QFP at 10 MHz and V_{CC} = 5V ± 5%.
2. INT0 has an internal pullup that sources 100 μA.
3. Input, output and clock capacitance are not tested.
4. Not measured on open-drain outputs.
5. Lock has an internal pullup that sources 100 μA.

AC SPECIFICATIONS

This section describes the AC specifications for the 80960SA and 80960SB pins. All input and output timings are specified relative to the 1.5V level of the rising edge of CLK2, and refer to the time at which the signal crosses (for output delay and input setup) 1.5V. All AC testing should be done with input voltages of 0.4V and 2.4V, except for the clock (CLK2), which should be tested with input voltages of 0.45V and 0.7 * V_{CC}. See Figure 13 for timing relationships for the 80960SA and 80960SB signals.

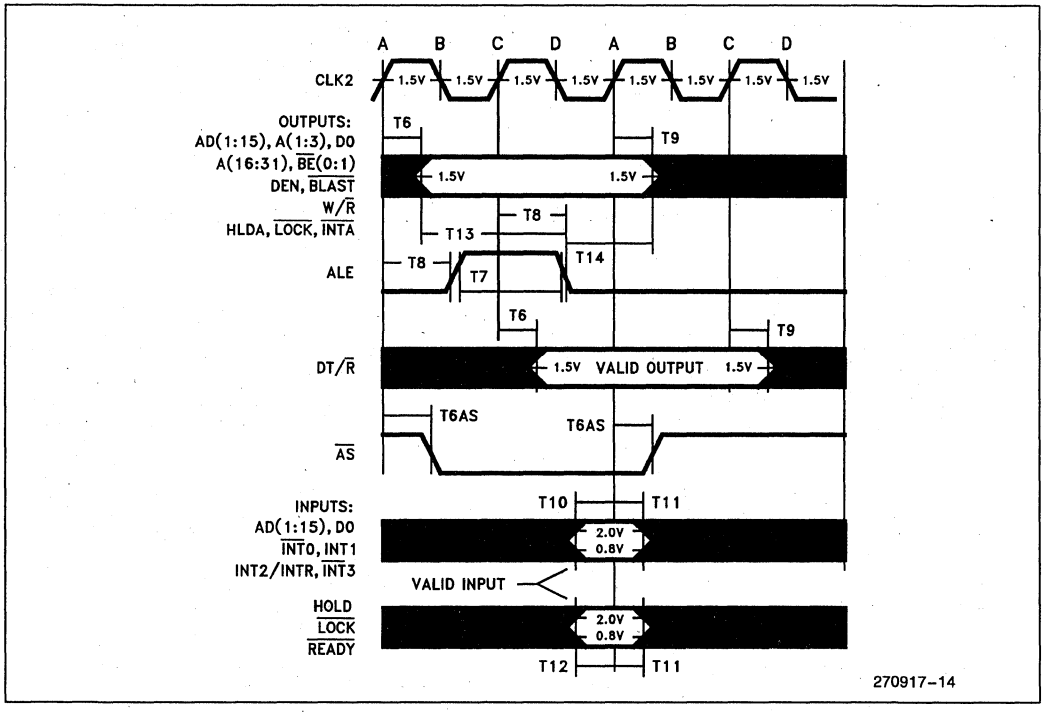


Figure 13. Drive Levels and Timing Relationships of 80960SA and 80960SB Signals

AC Specification Tables

80960SA and 80960SB AC Characteristics (10 MHz)

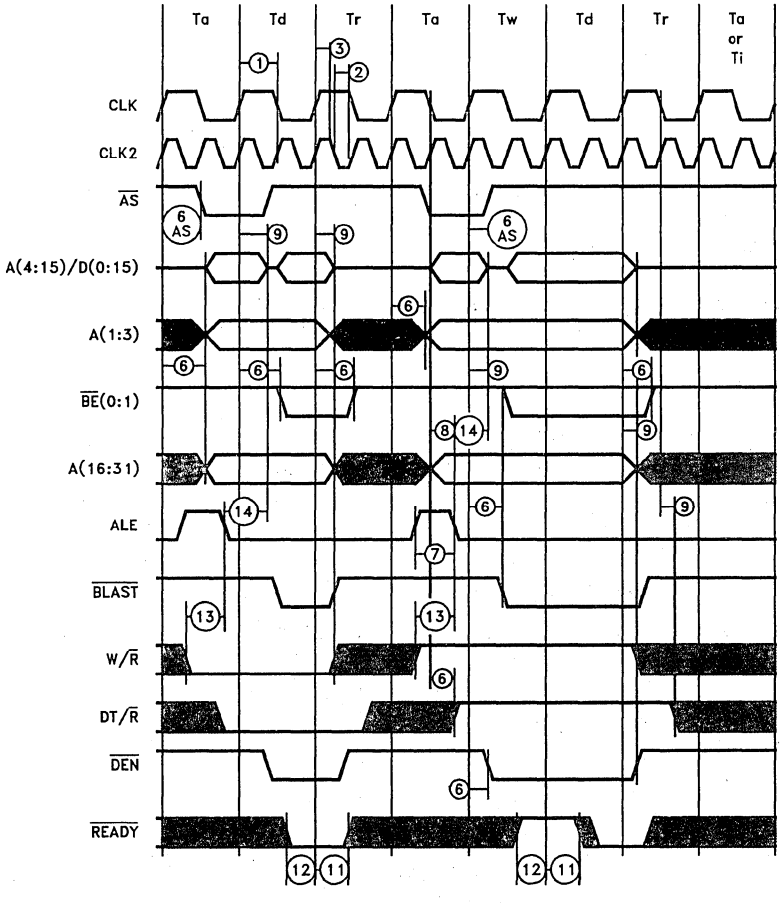
Symbol	Parameter	Min	Max	Units	Test Conditions
T1	Processor Clock Period (CLK2)	50	125	ns	V_{IN} = 1.5V
T2	Processor Clock Low Time (CLK2)	8		ns	V_T = 10% Point = $V_{CL} + (V_{CH} - V_{CL}) \times 0.1$
T3	Processor Clock High Time (CLK2)	8		ns	V_T = 90% Point = $V_{CL} + (V_{CH} - V_{CL}) \times 0.9$
T4	Processor Clock Fall Time (CLK2)		10	ns	V_T = 90% Point to 10% Point(3)
T5	Processor Clock Rise Time (CLK2)		10	ns	V_T = 10% Point to 90% Point(3)
T6	Output Valid Delay	2	31	ns	C_L = 100 pF (AD and Control)
T6AS	AS Output Valid Delay	2	25	ns	C_L = 50 pF
T7	ALE Width	24		ns	C_L = 100 pF
T8	ALE Output Valid Delay	4	33	ns	C_L = 100 pF(1)
T9	Output Float Delay	2	20	ns	C_L = 100 pF (AD) C_L = 100 pF (Controls)(1)
T10	Input Setup 1	10		ns	
T11	Input Hold	2		ns	(Note 4)
T12	Input Setup 2	13		ns	
T13	Setup to ALE Inactive	10		ns	C_L = 100 pF
T14	Hold after ALE Inactive	8		ns	C_L = 100 pF
T15	$\overline{\text{RESET}}$ Hold	3		ns	(Note 2)
T16	$\overline{\text{RESET}}$ Setup	5		ns	(Note 2)
T17	$\overline{\text{RESET}}$ Width	2050		ns	41 CLK2 Periods Minimum

NOTES:
1. A float condition occurs when the maximum output current becomes less than ILO. Float delay is not tested, but should be no longer than the valid delay.
2. Meeting $\overline{\text{RESET}}$ setup and hold times is an optional method of synchronizing your clocks. If you decide to use an asynchronous reset, then synchronizing the clock can be accomplished by using $\overline{\text{AS}}$.
3. Processor clock (CLK2) rise time and fall time are not tested.
4. ICE requires a minimum of 4 ns input hold time.

80960SA and 80960SB AC Characteristics (16 MHz PLCC)

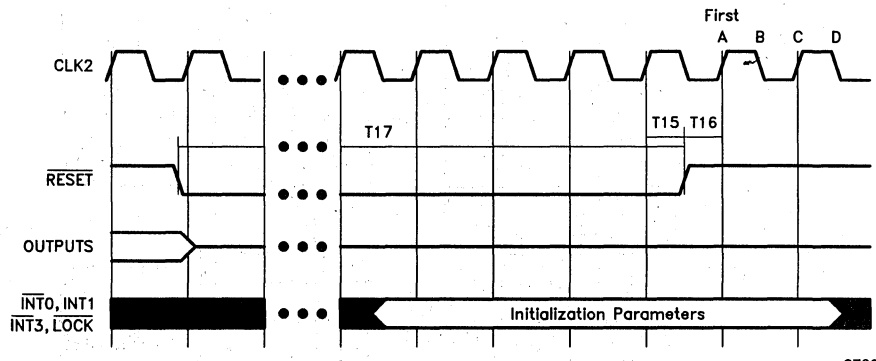
Symbol	Parameter	Min	Max	Units	Test Conditions
T1	Processor Clock Period (CLK2)	31.25	125	ns	V_{IN} = 1.5V
T2	Processor Clock Low Time (CLK2)	8		ns	V_T = 10% Point = $V_{CL} + (V_{CH} - V_{CL}) \times 0.1$
T3	Processor Clock High Time (CLK2)	8		ns	V_T = 90% Point = $V_{CL} + (V_{CH} - V_{CL}) \times 0.9$
T4	Processor Clock Fall Time (CLK2)		10	ns	V_T = 90% Point to 10% Point(3)
T5	Processor Clock Rise Time (CLK2)		10	ns	V_T = 10% Point to 90% Point(3)
T6	Output Valid Delay	2	25	ns	C_L = 100 pF (AD and Control)
T6AS	AS Output Valid Delay	2	21	ns	C_L = 50 pF
T7	ALE Width	15		ns	C_L = 100 pF
T8	ALE Output Valid Delay	2	22	ns	C_L = 100 pF(1)
T9	Output Float Delay	2	20	ns	C_L = 100 pF (AD) C_L = 100 pF (Controls)(1)
T10	Input Setup 1	10		ns	
T11	Input Hold	2		ns	(Note 4)
T12	Input Setup 2	13		ns	
T13	Setup to ALE Inactive	10		ns	C_L = 100 pF
T14	Hold after ALE Inactive	8		ns	C_L = 100 pF
T15	$\overline{\text{RESET}}$ Hold	3		ns	(Note 2)
T16	$\overline{\text{RESET}}$ Setup	5		ns	(Note 2)
T17	$\overline{\text{RESET}}$ Width	1281		ns	41 CLK2 Periods Minimum

NOTES:
1. A float condition occurs when the maximum output current becomes less than ILO. Float delay is not tested, but should be no longer than the valid delay.
2. Meeting $\overline{\text{RESET}}$ setup and hold times is an optional method of synchronizing your clocks. If you decide to use an asynchronous reset, then synchronizing the clock can be accomplished by using $\overline{\text{AS}}$.
3. Processor clock (CLK2) rise time and fall time are not tested.
4. ICE requires a minimum of 4 ns input hold time.



NOTES:
1. The AD and control signals are driven at all times except during a HOLD acknowledge (HLDA asserted) RESET, and ONCE mode.
2. The AD and control signals may toggle during idle (Ti) or recovery (Tr) cycles.

Figure 14. Timing Relationships of the 80960SA and 80960SB Bus



1. The A edge is defined as the first rising CLK2 edge after RESET is deasserted meeting the RESET hold and setup times.
2. Initialization Parameters must be setup at least four CLK2s prior to the first A edge.

Figure 15. RESET Signal Timing

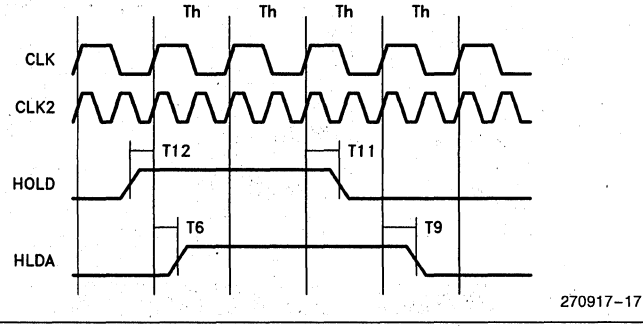


Figure 16. HOLD Timing Relationships

Design Considerations

Input hold times can be disregarded by the designer whenever the input is removed because a subsequent output from the processor is deasserted (e.g., DEN becomes deasserted).

Whenever the processor generates an output that indicates a transition into a subsequent state, any outputs that are specified to be 3-stated in this new state are guaranteed to be 3-stated. For example, in the Td cycle following a Ta cycle for a read, the minimum output delay of DEN is 2 ns, but the maximum float time of AD is 20 ns. When DEN is asserted, however, the AD outputs are guaranteed to have been 3-stated.

Designing for the ICE-960SB

The 80960SA and 80960SB In-Circuit Emulator assists in debugging 80960SA and 80960SB hardware and software designs. The product consists of a probe module, cable, control unit and power supply. Because of the high operating frequency of the 80960SA and 80960SB systems, the probe module

connects directly to the 80960SA and 80960SB component (EIAJ QFP or PLCC) or a socket for the PLCC.

When designing an 80960SA and 80960SB hardware system that uses the ICE-960SB to debug the system, several electrical and mechanical characteristics should be considered. These considerations include capacitive loading, drive requirement, power requirement, and physical layout.

The ICE-960SB probe module increases the load capacitance of each line by up to 25 pF. This load originates from the probe module and are driven by the 80960SA and 80960SB processor.

To achieve high noise immunity, the ICE-960SB probe is powered by the user's system. The high-speed probe circuitry draws up to 1.1A plus the maximum current (I_{CC}) of the 80960SA and 80960SB processor.

The AD bus should not be driven by an external source unless $\overline{DEN}$ is asserted. In addition, the ICE requires a minimum data hold time of 4 ns.

The ICE960SB probe will drive $\overline{LOCK}$ to a LOW state during RESET to force the target 80960SA and 80960SB to enter ONCE mode. To guarantee timings, the ICE requires ±5% supply voltage supplied to the 80960SA and 80960SB. The ICE probe requires a minimum of 0.25 inches clearance on all sides of both the EIAJ QFP and PLCC.

Lock Line Termination

You must terminate the LOCK line as described in Figure 6 in order for the ICE to properly function.

MECHANICAL DATA

Package Dimensions and Mounting

The 80960SA and 80960SB is available in two different packages: an 80-lead quad flat pack (EIAJ QFP), shown in Figure 17, and an 84-lead plastic leaded chip carrier (PLCC), shown in Figure 18.

Pin Assignment

The QFP and PLCC have different pin assignments. The QPF pins are numbered in order from 1 to 80 around the package's perimeter. The PLCC pins are numbered in order from 1 to 84 around the package's perimeter. Tables 9 and 10 list the function of each pin in the QFP. Tables 11 and 12 list the function of each pin in the PLCC.

V_{CC} and GND connection must be made to multiple V_{CC} and GND pins. Each V_{CC} and GND pin must be connected to the appropriate voltage or ground and externally strapped close to the package. We recommend that you include separate power and ground planes in your circuit board for power distribution.

NOTE:
Pins identified as N.C., "No Connect," should never be connected. The 80960SA and 80960SB QFP package contains two N.C. pins and PLCC package contains six N.C. pins.

Package Thermal Specification

The 80960SA and 80960SB is specified for operation when case temperature is within the range 0°C to + 85°C. The case temperature should be measured at the top center of the package.

The ambient temperture can be calculated from θ_{JC} and θ_{JA} by using the following equations:

$$T_J = T_C + P * \theta_{JC}$$
$$T_A = T_J - P * \theta_{JA}$$
$$T_C = T_A + P * [\theta_{JA} - \theta_{JC}]$$

Values for θ_{JA} and θ_{JC} are given in Table 7 for the QFP package and in Table 8 for the PLCC package for various airflows.

Example:

T_A = $T_C - P * (\theta_{JA} - \theta_{JC})$
T_C = Maximum Case Temperature
P = Maximum Supply Voltage times **I_{CC} at 100° and 10 MHz**
θ_{JA} and θ_{JC} = QFP Package Thermal Resistance at 0 ft/m airflow

$$T_A = 51 = 100 - (5.5 * 0.213) * (45.7 - 4)$$

WAVEFORMS

Figure 19 through 22 shows the waveforms for various signals on the 80960SA and 80960SB's bus.

Table 7. QFP Package, Thermal Resistance—°C/Watt

	Airflow—ft/min						
Parameter	**0**	**50**	**100**	**200**	**400**	**600**	**800**
θ_{JA} Junction to Ambient (Case measured in the middle of the top of the package) (No Heatsink)	45.7	na	na	40	31	na	na
θ_{JC} Junction to Case	4.0	na	na	4.5	5.5	na	na

NOTES:
1. This table applies to an 80960SA and 80960SB QFP soldered directly onto a board.
2. $\theta_{JA} = \theta_{JC} + \theta_{CA}$.
3. Thermal data are based on copper lead frames.

Table 8. PLCC Package, Thermal Resistance—°C/Watt

	Airflow—ft/min							
Parameter	**0**	**50**	**100**	**200**	**400**	**600**	**800**	**1000**
θ_{JA} Junction to Ambient (No Heatsink)	33	na	na	27	23.8	22	20	19.5
θ_{JC} Junction to Case	13	na	na	na	na	na	na	na

NOTES:
1. This table applies to an 80960SA and 80960SB PLCC soldered directly onto a board.
2. $\theta_{JA} = \theta_{JC} + \theta_{CA}$.

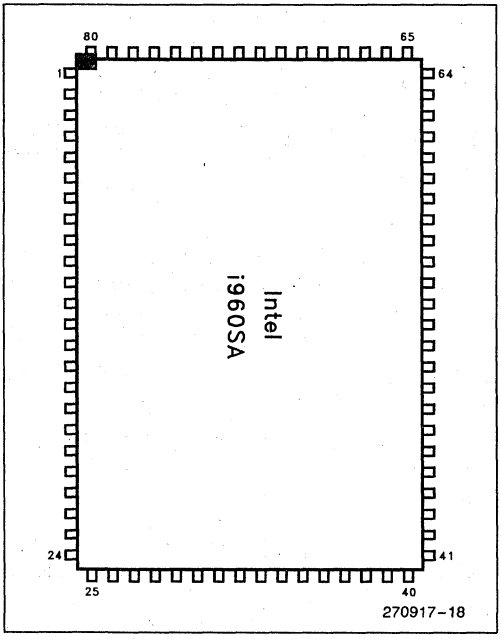


Figure 17. 80-Lead EIAJ Quad Flat Pack Package

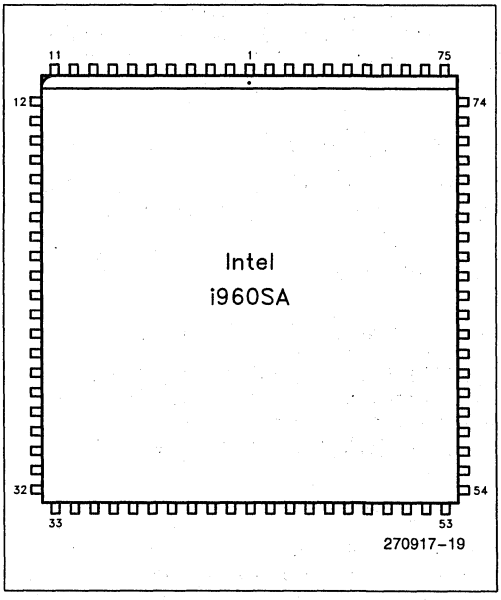


Figure 18. 84-Lead Plastic Leaded Chip Carrier

Table 9. 80960SA and 80960SB QFP Pinout—In Pin Order

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	A22	21	V_{CC}	41	$\overline{BE0}$	61	V_{CC}
2	A21	22	V_{SS}	42	V_{CC}	62	V_{SS}
3	A20	23	V_{CC}	43	V_{SS}	63	N.C.
4	A19	24	V_{SS}	44	CLK2	64	$\overline{AS}$
5	A18	25	AD6	45	$\overline{RESET}$	65	V_{SS}
6	A17	26	AD5	46	$\overline{INT0}$	66	ALE
7	A16	27	AD4	47	INT1	67	$\overline{READY}$
8	V_{CC}	28	AD3	48	INT2/INTR	68	A31
9	V_{SS}	29	AD2	49	$\overline{INT3}/\overline{INTA}$	69	A30
10	AD15	30	AD1	50	HLDA	70	A29
11	AD14	31	D0	51	V_{CC}	71	A28
12	V_{CC}	32	V_{SS}	52	V_{SS}	72	V_{SS}
13	V_{SS}	33	V_{CC}	53	HOLD	73	V_{CC}
14	AD13	34	A3	54	W/$\overline{R}$	74	A27
15	AD12	35	A2	55	$\overline{DEN}$	75	A26
16	AD11	36	V_{CC}	56	DT/$\overline{R}$	76	A25
17	AD10	37	V_{SS}	57	$\overline{BLAST}$	77	V_{CC}
18	AD9	28	A1	58	$\overline{LOCK}$	78	V_{SS}
19	AD8	39	N.C.	59	V_{CC}	79	A24
20	AD7	40	$\overline{BE1}$	60	V_{SS}	80	A23

Table 10. 80960SA and 80960SB QFP Pinout—In Signal Order

Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin
A1	38	A18	5	D0	31	V_{CC}	51
A2	35	A19	4	$\overline{DEN}$	55	V_{CC}	59
A3	34	A20	3	DT/$\overline{R}$	56	V_{CC}	61
AD1	30	A21	2	HLDA	50	V_{CC}	73
AD2	29	A22	1	HOLD	53	V_{CC}	77
AD3	28	A23	80	$\overline{INT0}$	46	V_{CC}	8
AD4	27	A24	79	INT1	47	V_{SS}	13
AD5	26	A25	76	INT2/INTR	48	V_{SS}	22
AD6	25	A26	75	$\overline{INT3}/\overline{INTA}$	49	V_{SS}	24
AD7	20	A27	74	$\overline{LOCK}$	58	V_{SS}	32
AD8	19	A28	71	N.C.	39	V_{SS}	37
AD9	18	A29	70	N.C.	63	V_{SS}	43
AD10	17	A30	69	$\overline{READY}$	67	V_{SS}	52
AD11	16	A31	68	$\overline{RESET}$	45	V_{SS}	60
AD12	15	ALE	66	V_{CC}	12	V_{SS}	62
AD13	14	$\overline{AS}$	64	V_{CC}	21	V_{SS}	72
AD14	11	$\overline{BE0}$	41	V_{CC}	23	V_{SS}	78
AD15	10	$\overline{BE1}$	40	V_{CC}	33	V_{SS}	9
A16	7	$\overline{BLAST}$	57	V_{CC}	36	V_{SS}	65
A17	6	CLK2	44	V_{CC}	42	W/$\overline{R}$	54

Table 11. 80960SA and 80960SB PLCC Pinout—In Pin Order

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	V_{CC}	22	V_{SS}	43	V_{SS}	64	HOLD
2	N.C.	23	N.C.	44	V_{CC}	65	N.C.
3	A27	24	AD13	45	A3	66	W/$\overline{R}$
4	A26	25	AD12	46	A2	67	$\overline{DEN}$
5	A25	26	AD11	47	V_{CC}	68	DT/$\overline{R}$
6	V_{CC}	27	AD10	48	V_{SS}	69	$\overline{BLAST}$
7	V_{SS}	28	AD9	49	A1	70	$\overline{LOCK}$
8	A24	29	AD8	50	N.C.	71	V_{CC}
9	A23	30	AD7	51	$\overline{BE1}$	72	V_{SS}
10	A22	31	V_{CC}	52	$\overline{BE0}$	73	V_{CC}
11	A21	32	V_{SS}	53	V_{CC}	74	V_{SS}
12	A20	33	V_{CC}	54	V_{SS}	75	N.C.
13	A19	34	V_{SS}	55	CLK2	76	$\overline{AS}$
14	A18	35	AD6	56	$\overline{RESET}$	77	V_{SS}
15	A17	36	AD5	57	$\overline{INT0}$	78	ALE
16	A16	37	AD4	58	$\overline{INT1}$	79	$\overline{READY}$
17	V_{CC}	38	AD3	59	INT2/INTR	80	A31
18	V_{SS}	39	AD2	60	$\overline{INT3}$/$\overline{INTA}$	81	A30
19	AD15	40	AD1	61	HLDA	82	A29
20	AD14	41	D0	62	V_{CC}	83	A28
21	V_{CC}	42	N.C.	63	V_{SS}	84	V_{SS}

Table 12. 80960SA and 80960SB PLCC Pinout—In Signal Order

Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin
A1	49	A18	14	DT/$\overline{R}$	68	V_{CC}	44
A2	46	A19	13	HLDA	61	V_{CC}	47
A3	45	A20	12	HOLD	64	V_{CC}	53
D0	41	A21	11	$\overline{INT0}$	57	V_{CC}	6
AD1	40	A22	10	INT1	58	V_{CC}	62
AD2	39	A23	9	INT2/INTR	59	V_{CC}	71
AD3	38	A24	8	$\overline{INT3}$/$\overline{INTA}$	60	V_{CC}	73
AD4	37	A25	5	LOCK	70	V_{SS}	18
AD5	36	A26	4	N.C.	2	V_{SS}	22
AD6	35	A27	3	N.C.	23	V_{SS}	32
AD7	30	A28	83	N.C.	42	V_{SS}	34
AD8	29	A29	82	N.C.	50	V_{SS}	43
AD9	28	A30	81	N.C.	65	V_{SS}	48
AD10	27	A31	80	N.C.	75	V_{SS}	54
AD11	26	ALE	78	$\overline{READY}$	79	V_{SS}	63
AD12	25	$\overline{AS}$	76	$\overline{RESET}$	56	V_{SS}	7
AD13	24	$\overline{BE0}$	52	V_{CC}	1	V_{SS}	72
AD14	20	$\overline{BE1}$	51	V_{CC}	17	V_{SS}	74
AD15	19	$\overline{BLAST}$	69	V_{CC}	21	V_{SS}	77
AD16	16	CLK2	55	V_{CC}	31	V_{SS}	84
A17	15	$\overline{DEN}$	67	V_{CC}	33	W/$\overline{R}$	66

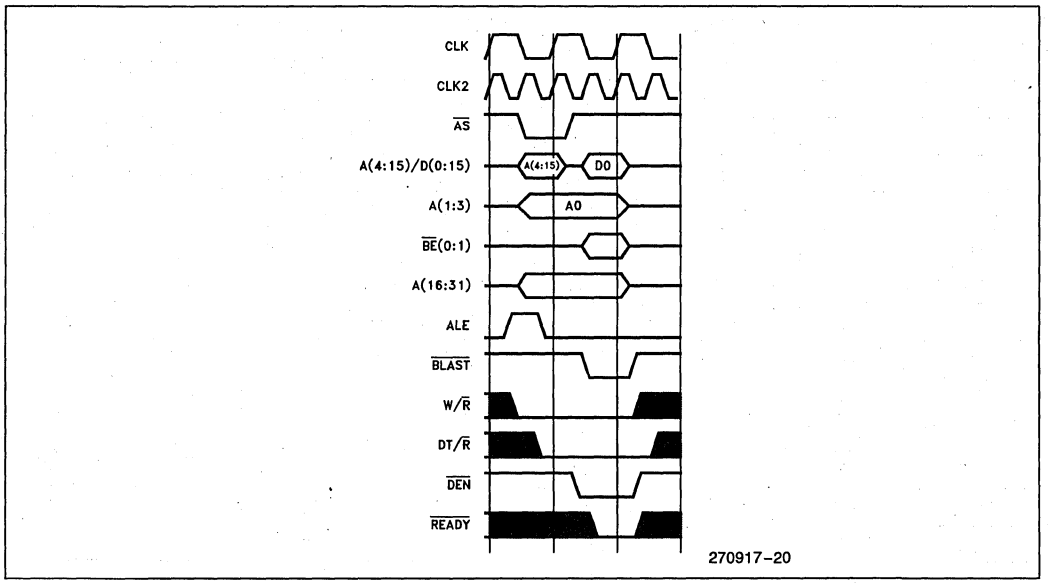


Figure 19. Basic 80960SA and 80960SB Timing

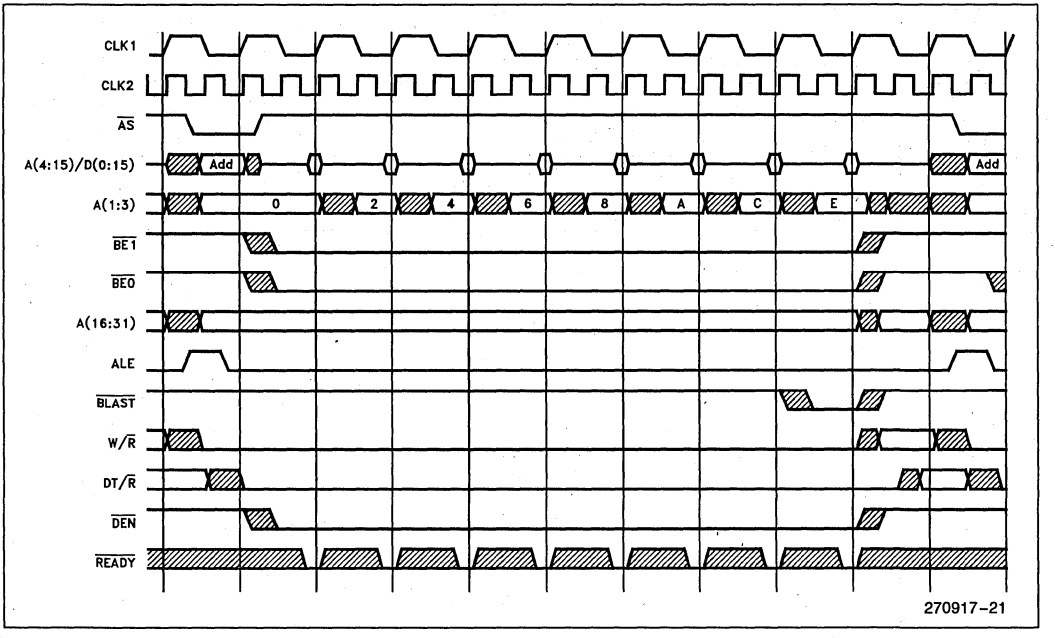


Figure 20. 80960SA and 80960SB Timing Showing a Four Word Aligned Read Burst

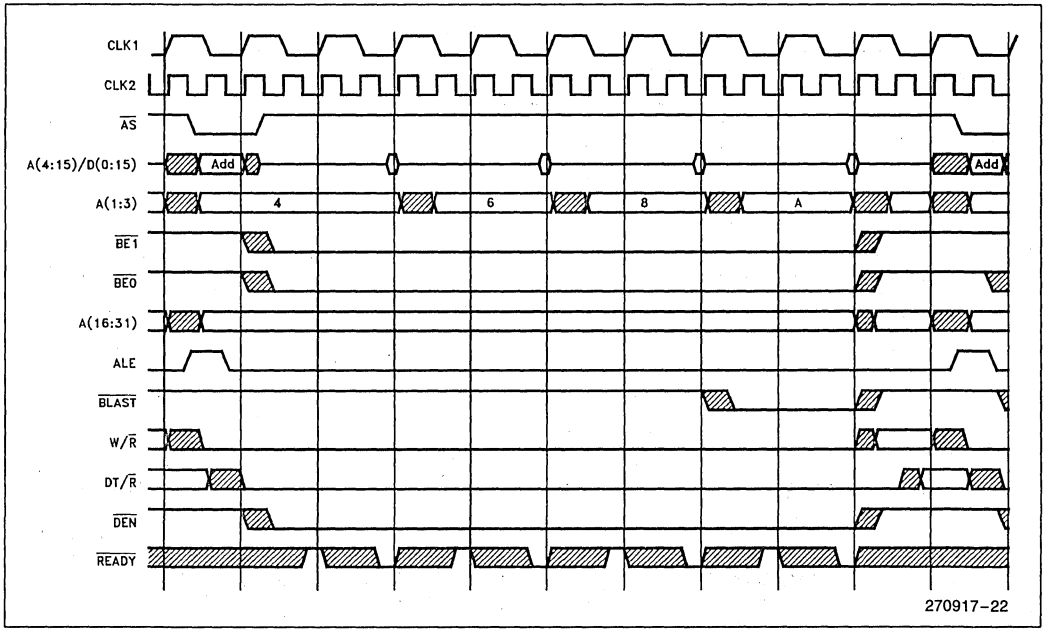


Figure 21. 80960SA and 80960SB Double Word Read Timing with Wait States

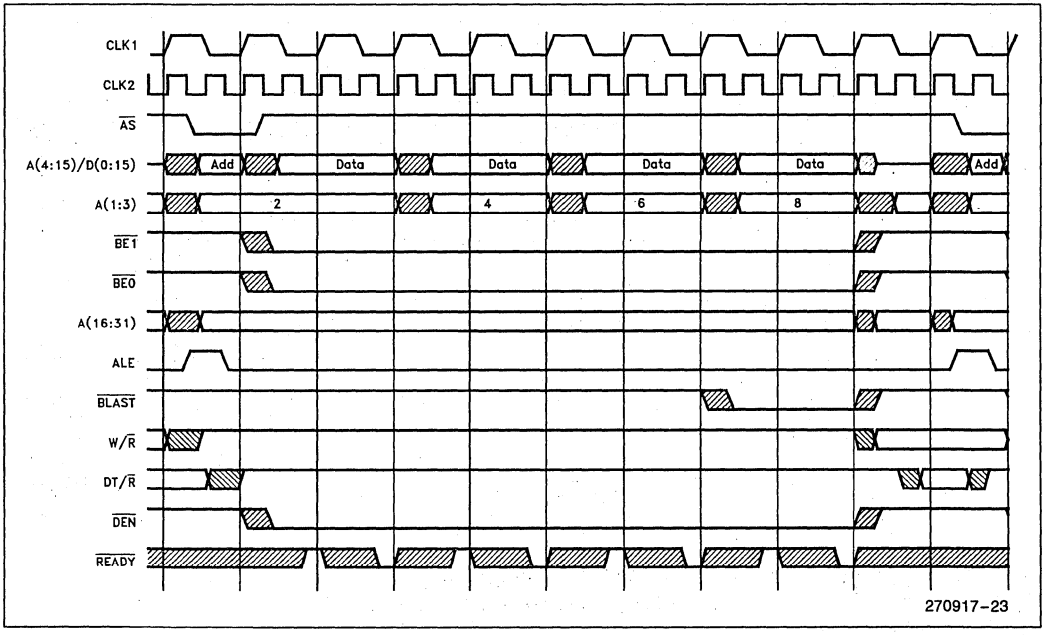


Figure 22. 80960SA and 80960SB Aligned Double Word Write Timing with Wait States

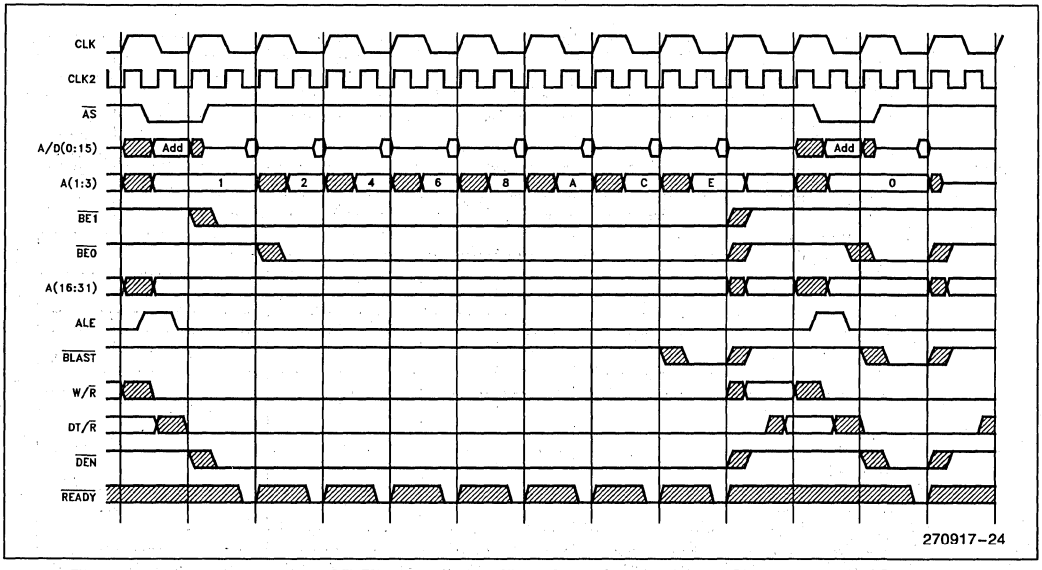


Figure 23. 80960SA 80960SB Timing with a Four Word Read Burst Misaligned by One Byte

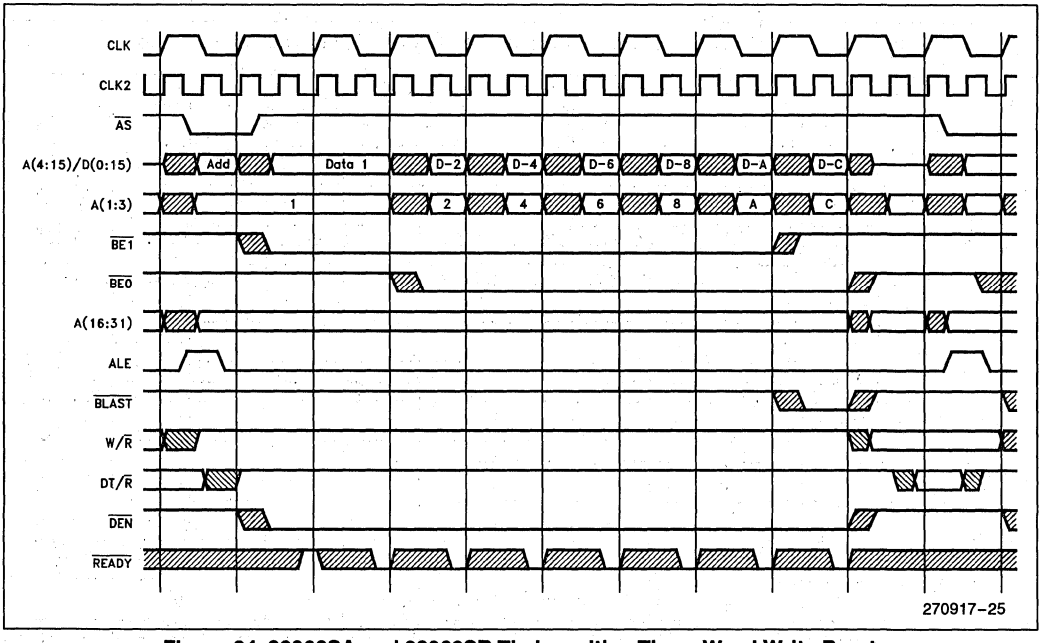


Figure 24. 80960SA and 80960SB Timing with a Three Word Write Burst Misaligned by One Byte and One Wait State

i960™ KA/KB PROCESSOR PRODUCT OVERVIEW

INTRODUCTION

This chapter provides an overview of the Intel i960 KB processor (which is part of the i960 K series of embedded-processor products).

All of the processors in the i960 K series of products are based on the Intel i960™ architecture. Most of the information in this overview also applies to the i960 KA processor. The only difference between the i960 KB and i960 KA processors is that the i960 KA processor does not provide on-chip support for floating-point operations or operations on decimal numbers.

OVERVIEW OF THE i960™ KB ARCHITECTURE

The i960 KB processor introduced the i960 architecture—a new 32-bit architecture from Intel. This architecture has been designed to meet the needs of embedded applications such as machine control, robotics, process control, avionics and instrumentation.

The i960 architecture can best be characterized as a high-performance computing engine. It features high-speed instruction execution and ease of programming. It is also easily extensible, allowing processors and controllers based on this architecture to be conveniently customized to meet the needs of specific processing and control applications.

The following are some of the important attributes of the i960 architecture:

- full 32-bit registers
- high-speed, pipelined instruction execution
- a convenient program execution environment with 32 general-purpose registers and a versatile set of special-function registers
- a highly optimized procedure call mechanism that features on-chip caching of local variables and parameters
- extensive facilities for handling interrupts and faults
- extensive tracing facilities to support efficient program debugging and monitoring
- register scoreboarding and write buffering to permit efficient operation when used with lower performance memory subsystems

OVERVIEW OF THE SINGLE PROCESSOR SYSTEM ARCHITECTURE

The central processing module, memory module and I/O module form the natural boundaries for the hardware system architecture. The modules are connected together by the high bandwidth 32-bit multiplexed L-bus, which can transfer data at a maximum sustained rate of 53 Mbytes per second for an i960 processor operating at 20 MHz.

Figure 1 shows a simplified block diagram of one possible system configuration. The heart of this system is the i960 KB processor, which fetches instructions, executes code, manipulates stored information and interacts with I/O devices. The high bandwidth L-bus connects the i960 KB processor to memory and I/O modules. The i960 KB processor stores system data, instructions and programs in the memory module. By accessing various peripheral devices in the I/O module, the i960 KB processor supports communication to terminals, modems, printers, disks and other I/O devices.

i960™ KB Processor and the L-Bus

The i960 KB processor performs bus operations using multiplexed address and data signals, and provides all the necessary control signals. For example standard control signals, such as Address Latch Enable ($\overline{ALE}$), Address/Data Status ($\overline{ADS}$), Write/Read Command (W/$\overline{R}$), Data Transmit/Receive (DT/$\overline{R}$) and Data Enable ($\overline{DEN}$), are provided by the i960 KB processor. The i960 processor also generates byte enable signals that specify which bytes on the 32-bit data lines are valid for the transfer.

The L-bus supports burst transactions, which access up to four data words at a maximum rate of one word per clock cycle. The i960 KB processor uses the two low-order address lines to indicate how many words are to be transferred. The i960 KB processor performs burst transactions to load the on-chip 512-byte instruction cache to minimize memory accesses for instruction fetches. Burst transactions can also be used for data access.

To transfer control of the bus to an external bus master, the i960 KB provides two arbitration signals: hold request (HOLD) and hold acknowledge (HLDA). After receiving HOLD, the processor grants control of the bus to an external master by asserting HLDA.

Order Number: 272030-001

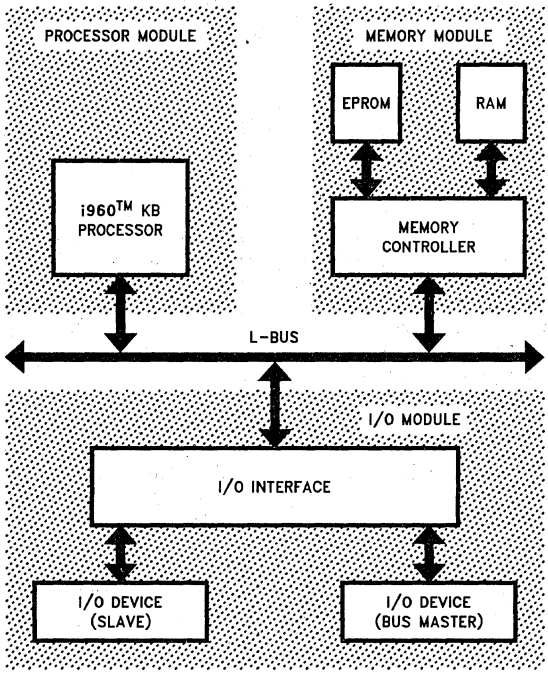


Figure 1. Basic i960™ KB System Configuration

The i960 KB processor provides a flexible interrupt structure by using an on-chip interrupt controller, an external interrupt controller or both. The type of interrupt structure is specified by an internal interrupt vector register. For a system with multiple processors, another method is available, called inter-agent communication (IAC) where a processor can interrupt another processor by sending an IAC message.

Memory Module

A memory module can consist of a memory controller, Erasable Programmable Read Only Memory (EPROM), and static or dynamic Random Access Memory (RMA). The memory controller first conditions the L-bus signals for memory operation. It demultiplexes the address and data lines, generates the chip select signals from the address, detects the start of the cycle for burst mode operation and latches the byte enable signals.

The memory controller generates the control signals for EPROM, SRAM and DRAM. Specifically, it provides the control signals, multiplexed row/column address and refresh control for dynamic RAMs. The controller can be designed to accommodate the burst transaction of the i960 KB processor by using the static column mode or nibble mode features of the dynamic RAM. In addition to supplying the operational signals, the controller generates the READY signal to indicate that data can be transferred to or from the i960 KB processor.

The i960 KB processor directly addresses up to 4 Gbytes of physical memory. The processor does not allow burst accesses to cross a 16-byte boundary, to ease the design of the controller. Each address specifies a four-byte data word within the block. Individual data bytes can be accessed by using the four byte-enable signals from the i960 KB processor. Chapter 5 provides design guidelines for the memory controller.

I/O Module

The I/O module consists of the I/O components and the interface circuit. I/O components can be used to allow the i960 KB processor to use most of its clock cycles for computational and system management activities. Time consuming tasks can be off-loaded to specialized slave-type components, such as the 8259A Pro-

grammable Interrupt Controller or the 82530 Serial Communication Controller. Some tasks may require a master-type component, such as the 82586 Local Area Network Control.

The interface circuit performs several functions. It demultiplexes the address and data lines, generates the chip select signals from the address, produces the I/O read or I/O write command from the processor's W/R signal, latches the byte enable signals and generates the READY signals. Since some of these functions are identical to those of the memory controller, the same logic can be used for both interfaces. For master-type peripherals that operate on a 16-bit data bus, the interface circuit translates the 32-bit data bus to a 16-bit data bus.

The i960 KB processor uses memory-mapped addresses to access I/O devices. This allows the CPU to use many of the same instuctions to exchange information for both memory and peripheral devices. Thus, the powerful memory-type instructions can be used to perform 8-, 16- and 32-bit data transfers.

HIGH PERFORMANCE PROGRAM EXECUTION

Much of the design of the i960 architecture has been aimed at maximizing the processor's computational and data processing speed through the use of increased parallelism. The following paragraphs describe several of the mechanisms and techniques used to accomplish this goal.

Load and Store Model

One of the more important features of the i960 architecture is its performance of most operations on operands in registers, rather than in memory. For example, all arithmetic, logic, comparison, branching and bit operations are performed with registers and literals.

This feature provides two benefits. First, it increases program execution speed by minimizing the number of memory accesses necessary to execute a program. Second, it reduces the memory latency encountered when using slower, lower-cost memory parts.

To support this concept, the architecture provides a generous supply of general-purpose registers. For each procedure, 32 registers are available, 28 of which are available for general use. These registers are divided into two types: global and local. Both types of registers can be used for general storage of operands. The only difference is that global registers retain their contents across procedure boundaries, whereas the processor allocates a new set of local registers each time a new procedure is called.

The architecture also provides a set of fast, versatile load and store instructions. These instructions allow burst transfers of 1, 2, 4, 8, 12 or 16 bytes of information between memory and the registers.

On-Chip Caching of Code and Data

To further reduce memory accesses, the architecture offers two mechanisms for caching code and data on chip: an instruction cache and multiple sets of local registers. The instruction cache allows prefetching of blocks of instruction from memory. This helps ensure that the instruction execution pipeline is supplied with a steady stream of instructions. It also reduces the number of memory accesses required when performing iterative operations such as loops. The architecture allows the size of the instruction cache to vary. For the i960 KB processor, it is 512 bytes.

To optimize the architecture's procedure call mehanism, the processor provides multiple sets of local registers. This allows the processor to perform procedure calls without having to write the local registers out to the stack in memory. The number of register sets depends on the processor implementation. The i960 KB processor provides four sets of local registers.

Overlapped Instruction Execution

The i960 architecture also enchances program execution speed by overlapping the execution of some instructions. In the i960 K series of processors, this is accomplished through register scoreboarding.

Register scoreboarding permits instruction execution to continue while data is being fetched from memory. When a load instruction is executed, the processor sets one or more scoreboard bits to indicate the target registers to be loaded. After the target registers are loaded, the scoreboard bits are cleared. While the target registers are being loaded, the processor is allowed to execute other instructions that do not use these registers.

The processor uses the scoreboard bits to ensure that the target registers are not used until the load is complete. (Scoreboard bits are checked transparently from software.) This technique allows code to be executed such that some instructions can be executed in zero clock cycles (that is, executed for free).

Single-Clock Instructions

The i960 architecture is designed to let a processor execute commonly used instructions, such as moves, adds, subtracts, logical operations and branches, in a minimum number of clock cycles (preferably one cycle). The architecture supports this concept in several

ways. For example, the load and store model described earlier eliminates the clock cycles required to perform memory-to-memory operations, by concentrating on register-to-register operations.

In addition, all of the instructions in the i960 architecture are 32 bits long and aligned on 32-bit boundaries. This lets instructions be decoded in one clock cycle, and eliminates the need for an instruction-alignment stage in the pipeline.

The i960 KB processor takes full advantage of these features of the architecture, resulting in more than 50 instructions that can be executed in a single clock cycle.

Efficient Interrupt Model

The i960 architecture provides an efficient mechanism for servicing interrupts from external sources. To handle interrupts, the processor maintains an interrupt table of 248 interrupt vectors, 240 of which are available for general use. When an interrupt is signaled, the processor uses a pointer to the interrupt table to perform an implicit call to an interrupt handler procedure. In performing this call, the processor automatically saves the state of the processor prior to receiving the interrupt, performs the interrupt routine, then restores the state of the processor. A separate interrupt stack is also provided to segregate interrupt handling from application programs.

The interrupt handling facilities also allow interrupts to be evaluated by priority. The processor is then able to store interrupt vectors that are lower in priority than the current processor task in a pending interrupt section of the interrupt table. The processor checks and services the pending interrupts at defined times.

SIMPLIFIED PROGRAMMING ENVIRONMENT

Because of its streamlined execution environment, processors based on the i960 architecture are particularly easy to program. The following paragraphs describe some of the architecture features that simplify programming.

Highly Efficient Procedure Call Mechanism

The procedure call mechanism makes procedure calls and parameter passing between procedures simple and compact. Each time a call instruction is issued, the processor automatically saves the current set of local registers and allocates a new set for the called procedure. Likewise, on a return from a procedure, the current set of local registers is deallocated and the local registers for the procedure being returned to are restored. This means a program never has to explicitly save and restore those local variables that are stored in local registers.

Versatile Instruction Set and Addressing

The selection of instructions and addressing modes also simplifies programming. A full set of load, store, move, arithmetic, comparison and branch instructions are provided, with operations on both integer and ordinal data types. Operations on bits and bit strings are simplified by a complete set of Boolean and bit-field instructions.

The addressing modes are efficient and straightforward, while at the same time providing the necessary indexing and scaling modes required to address complex arrays and record structures. The large 4-gigabyte address space provides ample room to store programs and data. The availability of 32 addressing lines allows some address lines to be memory-mapped to control hardware functions.

Extensive Fault Handling Capability

To aid in program development, the i960 architecture defines a wide range of faults that the processor detects, including, arithmetic, faults, invalid operations, invalid operands and machine faults. When a fault is detected, the processor makes an implicit call to a fault handler routine, in a way similar to the interrupt mechanism described previously. The information collected for each fault allows program developers to quickly correct faulting code, and allows automatic recovery from some faults.

Debugging and Monitoring

To support debugging systems, the i960 architecture provides a mechanism for monitoring processor activity by means of trace events. When the processor detects a trace event, it signals a trace fault and calls a fault handler. Intel provides several tools that use this feature, including an in-circuit emulator (ICE) device.

SUPPORT FOR ARCHITECTURAL EXENSIONS

The i960 architecture provides several features that enable processors based on this architecture to be easily customized to meet the needs of specific embedded applications, such as signal processing, array processing or graphics processing.

The most important of these features is the set of 32 special function registers. These regisers provide a convenient interface to circuitry in the processor or pins that can be connected to external hardware. They can be used to control timers, to perform operations on special data types or to perform I/O functions. The special function registers are similar to the global registers. They can be addressed by all of the register access instructions.

EXTENSIONS INCLUDED IN THE i960™ K SERIES PROCESSORS

The i960 K series of processors provides a complete implementation of the i960 architecture, plus several extensions to that architecture. These extensions fall into two categories: floating-point processing and interagent communication.

On-Chip Floating Point

The i960 KB processor provides a complete implementation of the IEEE standard for binary floating-point arithmetic (IEEE 754-185). This implementation includes a full set of floating-point operations, including add, subtract, multiply, divide, trigonometric functions and logarithmic functions. These operations are performed on single precision (32-bit), double precision (64-bit) and extended precision (80-bit) real numbers.

One of the benefits of this implementation is that the floating-point handling facilities are integrated into the normal instruction execution environment. Single and double precision floating-point values are stored in the same registers as non-floating point values. Four 80-bit floating-point registers are provided to hold extended-precision values.

Interagent Communication

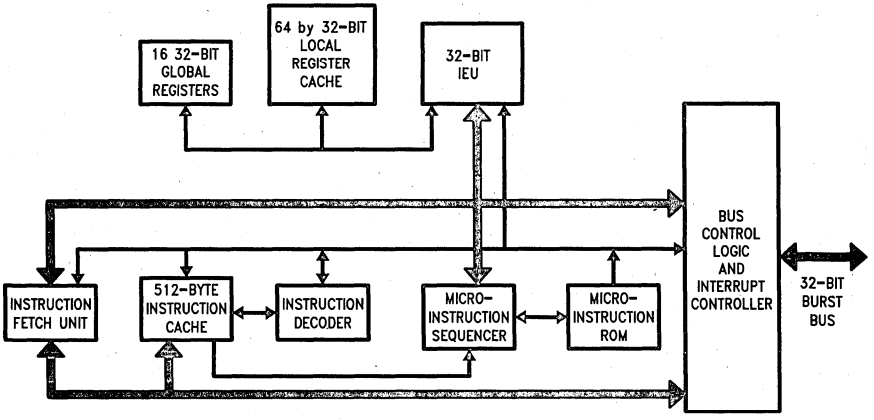
All of the processors in the i960 K series provide an inter-agent communication (IAC) mechanism, allowing agents connected to the processor's bus to communicate with one another. This mechanism operates similarly to the interrupt mechanism, except that IAC messages are passed through dedicated sections of memory. The sort of tasks handled with IAC messages are processor reinitialization, stopping the processor, purging the instruction cache and forcing the processor to check pending interrupts.

80960KA
EMBEDDED 32-BIT PROCESSOR

- **High-Performance Embedded Architecture**
  - **25 MIPS Burst Execution at 25 MHz**
  - **9.4 MIPS* Sustained Execution at 25 MHz**
- **512-Byte On-Chip Instruction Cache**
  - **Direct Mapped**
  - **Parallel Load/Decode for Uncached Instructions**
- **Pin Compatible with 80960KB**
- **Multiple Register Sets**
  - **Sixteen Global 32-Bit Registers**
  - **Sixteen Local 32-Bit Registers**
  - **Four Local Register Sets Stored On-Chip**
  - **Register Scoreboarding**
- **Built-In Interrupt Controller**
  - **32 Priority Levels 256 Vectors**
  - **3.4 μs Latency @ 25 MHz**
- **Easy to Use, High Bandwidth 32-Bit Bus**
  - **66.7 Mbytes/s Burst**
  - **Up to 16-Bytes Transferred per Burst**
- **4 Gigabyte, Linear Address Space**
- **132-Lead Pin Grid Array (PGA) Package**
- **132-Lead Plastic Quad Flat Pack (PQFP)**
- **Uses 85C960 Bus Controller**
- **Supported by 27960KX Burst EPROMs**

The 80960KA is a member of Intel's new 32-bit processor family, the i960 series, which is designed especially for embedded applications. It is based on the family's high performance, common core architecture, and includes a 512-byte instruction cache and a built-in interrupt controller. The 80960KA has a large register set, multiple parallel execution units and a high-bandwidth, burst bus. Using advanced RISC technology, this high performance processor is capable of execution rates in excess of 9.4 million instructions per second.* The 80960KA is well-suited for a wide range of embedded applications, including laser printers, image processing, industrial control, robotics and telecommunications.

*Relative to Digital Equipment Corporation's VAX-11/780** at 1 MIPS



270775-1

Figure 1. The 80960KA's Highly Parallel Microarchitecture

**VAX-11™ is a trademark of Digital Equipment Corporation.

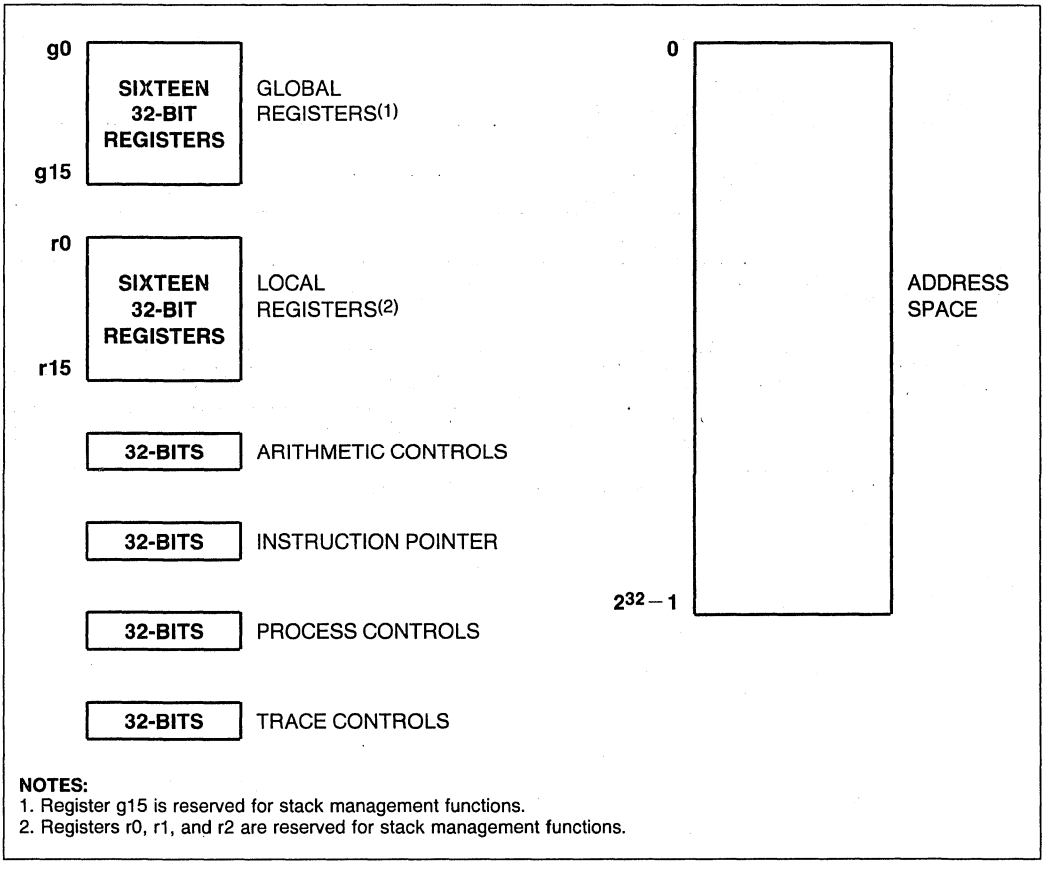
September 1991
Order Number: 270775-004

THE 960 SERIES

The 80960KA is a member of a new family of 32-bit microprocessors from Intel known as the i960 Series. This series was especially designed to serve the needs of embedded applications. The embedded market includes applications as diverse as industrial automation, avionics, image processing, graphics, robotics, telecommunications and automobiles. These types of applications require high integration, low power consumption, quick interrupt response times and high performance. Since time to market is critical, embedded microprocessors need to be easy to use in both hardware and software designs.

All members of the 80960 series share a common core architecture which utilizes RISC technology so that, except for special functions, the family members are object code compatible. Each new processor in the series will add its own special set of functions to the core to satisfy the needs of a specific application or range of applications in the embedded market. For example, future processors may include a DMA controller, a timer or an A/D converter.

Software written for the 80960KA will run without modification on any other member of the 80960 family. It is also pin-compatible with the 80960KB, which includes an integrated floating-point unit, and the 80960MC, a military-grade version with support for multitasking, memory management, multiprocessing and fault tolerance.



NOTES:
1. Register g15 is reserved for stack management functions.
2. Registers r0, r1, and r2 are reserved for stack management functions.

Figure 2. Register Set

KEY PERFORMANCE FEATURES

The 80960KA's architecture is based on the most recent advances in RISC technology and is grounded in Intel's long experience in designing embedded controllers. Many features contribute to the 80960KA's exceptional performance:

1. Large Register Set. Modern compilers can take advantage of a large number of registers to optimize execution speed. For maximum flexibility, the 80960KA provides 32 32-bit registers and four 80-bit floating-point registers. (See Figure 2.)

2. Fast Instruction Execution. Simple functions make up the bulk of instructions in most programs, so that execution speed can be greatly improved by ensuring that these core instructions execute in as short a time as possible. The most-frequently executed instructions such as register-register moves, add/subtract, logical operations, and shifts execute in one to two cycles (Table 1 contains a list of instructions.)

3. Load/Store Architecture. One way to improve execution speed is to reduce the number of times that the processor must access memory to perform an operation. Like other processors based on RISC technology, the 80960KA has a Load/Store architecture, only the LOAD and STORE instructions reference memory; all other instructions operate on registers.

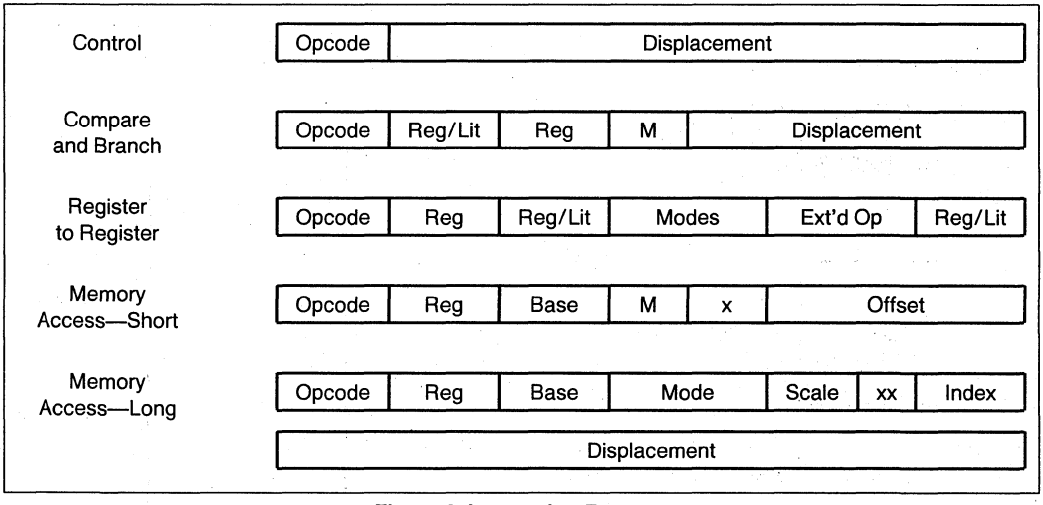


Figure 3. Instruction Formats

Table 1. 80960KA Instruction Set

Data Movement	Arithmetic	Logical	Bit and Bit Field
Load Store Move Load Address	Add Subtract Multiply Divide Remainder Modulo Shift	And Not And And Not Or Exclusive Or Not Or Or Not Nor Exclusive Nor Not Nand Rotate	Set Bit Clear Bit Not Bit Check Bit Alter Bit Scan for Bit Scan over Bit Extract Modify
Comparison	**Branch**	**Call/Return**	**Fault**
Compare Conditional Compare Compare and Increment Compare and Decrement	Unconditional Branch Conditional Branch Compare and Branch	Call Call Extended Call System Return Branch and Link	Conditional Fault Synchronize Faults
Debug	**Miscellaneous**	**Decimal**	
Modify Trace Controls Mark Force Mark	Atomic Add Atomic Modify Flush Local Registers Modify Arithmetic Controls Scan Byte for Equal Test Condition Code Modify Process Controls	Move Add with Carry Subtract with Carry	
		Synchronous	
		Synchronous Load Synchronous Move	

4. Simple Instruction Formats. All instructions in the 80960KA are 32-bits long and must be aligned on word boundaries. This alignment makes it possible to eliminate the instruction-alignment stage in the pipeline. To simplify the instruction decoder further, there are only five instruction formats and each instruction uses only one format. (See Figure 3.)

5. Overlapped Instruction Execution. A load operation allows execution of subsequent instructions to continue before the data has been returned from memory, so that these instructions can overlap the load. The 80960KA manages this process transparently to software through the use of a register scoreboard. Conditional instructions also make use of a scoreboard so that subsequent unrelated instructions can be executed while the conditional instruction is pending.

6. Integer Execution Optimization. When the result of an operation is used as an operand in a subsequent calculation, the value is sent immediately to its destination register. Yet at the same time, the value is put back on a bypass path to the ALU, thereby saving the time that otherwise would be required to retrieve the value for the next operation.

7. Bandwidth Optimizations. The 80960KA gets optimal use of its memory bus bandwidth because the bus is tuned for use with the cache: the line size of the instruction cache matches the maximum burst size for instruction fetches. The 80960KA automatically fetches four words in a burst and stores them directly in the cache. Due to the size of the cache and the fact that it is continually filled in anticipation of needed instructions in the program flow, the 80960KA is exceptionally insensitive to memory wait states. In fact, each wait state causes only a 7% degradation in system perfomance. The benefit is that the 80960KA will deliver outstanding performance even with a low cost memory system.

8. Cache Bypass. If there is a cache miss, the processor fetches the needed instruction, then sends it on to the instruction decoder at the same time it updates the cache. Thus, no extra time is taken to load and read the cache.

Memory Space and Addressing Modes

The 80960KA offers a linear programming environment so that all programs running on the processor are contained in a single address space. The maximum size of the address space is 4 Gigabytes (2^{32} bytes).

For ease of use, the 80960KA has a small number of addressing modes, but includes all those necessary to ensure efficient compiler implementations of high-level languages such as C, Fortran and Ada. Table 2 lists the memory addressing modes.

Data Types

The 80960KA recognizes the following data types:

Numeric:
- 8-, 16-, 32- and 64-bit ordinals
- 8-, 16, 32- and 64-bit integers

Non-Numeric:
- Bit
- Bit Field
- Triple-Word (96 bits)
- Quad-Word (128 bits)

Large Register Set

The programming environment of the 80960KA includes a large number of registers. In fact, 32 registers are available at any time. The availability of this many registers greatly reduces the number of memory accesses required to execute most programs, which leads to greater instruction processing speed.

There are two types of general-purpose registers: local and global. The global registers consist of sixteen 32-bit registers (G0 through G15) These registers perform the same function as the general-purpose registers provided in other popular microprocessors. The term global refers to the fact that these registers retain their contents across procedure calls.

The local registers, on the other hand, are procedure specific. For each procedure call, the 80960KA allocates 16 local registers (R0 through R15). Each local register is 32 bits wide.

Multiple Register Sets

To further increase the efficiency of the register set, multiple sets of local registers are stored on-chip. This cache holds up to four local register frames, which means that up to three procedure calls can be made without having to access the procedure stack resident in memory.

Although programs may have procedure calls nested many calls deep, a program typically oscillates back and forth between only two or three levels. As

Table 2. Memory Addressing Modes

- 12-Bit Offset
- 32-Bit Offset
- Register-Indirect
- Register + 12-Bit Offset
- Register + 32-Bit Offset
- Register + (Index-Register × Scale-Factor)
- Register × Scale Factor + 32-Bit Displacement
- Register + (Index-Register × Scale-Factor) + 32-Bit Displacement

Scale-Factor is 1, 2, 4, 8 or 16

a result, with four stack frames in the cache, the probability of there being a free frame on the cache when a call is made is very high. In fact, runs of representative C-language programs show that 80% of the calls are handled without needing to access memory.

If there are four or more active procedures and a new procedure is called, the processor moves the oldest set of local registers in the register cache to a procedure stack in memory to make room for a new set of registers. Global register G15 is used by the processor as the frame pointer (FP) for the procedure stack.

Note that the global registers are not exchanged on a procedure call, but retain their contents, making them available to all procedures for fast parameter passing. An illustration of the register cache is shown in Figure 4.

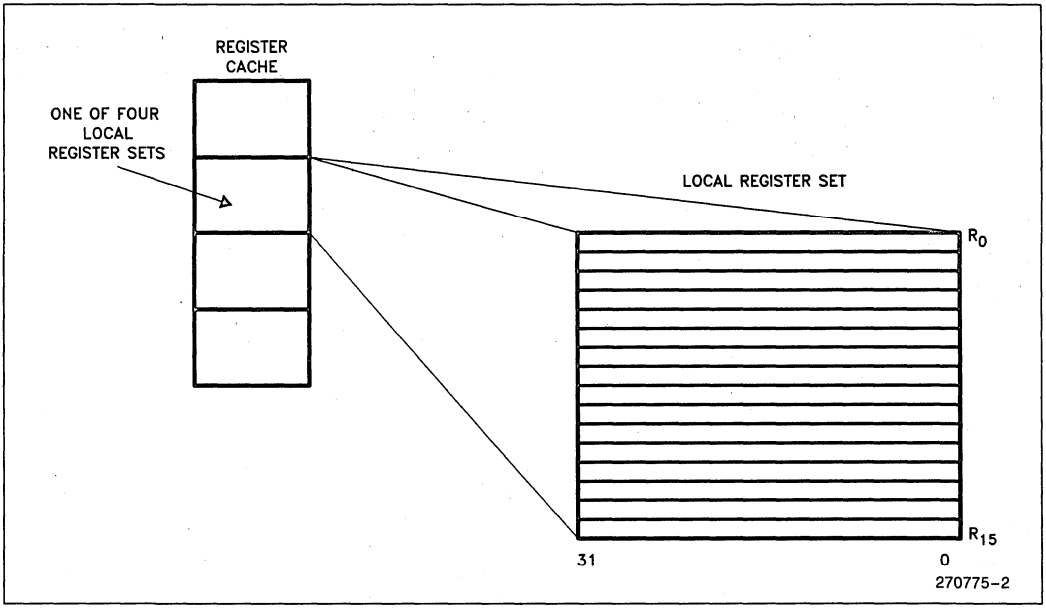


Figure 4. Multiple Register Sets Are Stored On-Chip

Instruction Cache

To further reduce memory accesses, the 80960KA includes a 512-byte on-chip instruction cache. The instruction cache is based on the concept of locality of reference; that is, most programs are not usually executed in a steady stream but consist of many branches and loops that lead to jumping back and forth within the same small section of code. Thus, by maintaining a block of instructions in a cache, the number of memory references required to read instructions into the processor can be greatly reduced.

To load the instruction cache, instructions are fetched in 16-byte blocks, so that up to four instructions can be fetched at one time. An efficient prefetch algorithm increases the probability that an instruction will already be in the cache when it is needed.

Code for small loops will often fit entirely within the cache, leading to a great increase in processing speed since further memory references might not be necessary until the program exits the loop. Similarly, when calling short procedures, the code for the calling procedure is likely to remain in the cache, so it will be there on the procedure's return.

Register Scoreboarding

The instruction decoder has been optimized in several ways. One of these optimizations is the ability to do instruction overlapping by means of register scoreboarding.

Register scoreboarding occurs when a LOAD instruction is executed to move a variable from memory into a register. When the instruction is initiated, a scoreboard bit on the target register is set. When the register is actually loaded, the bit is reset. In between, any reference to the register contents is accompanied by a test of the scoreboard bit to insure that the load has completed before processing continues. Since the processor does not have to wait for the LOAD to be completed, it can go on to execute additional instructions placed in between the LOAD instruction and the instruction that uses the register contents, as shown in the following example:

```
LOAD R4, address 1
LOAD R5, address 2
Unrelated instruction
Unrelated instruction
ADD R4, R5, R6
```

In essence, the two unrelated instructions between the LOAD and ADD instructions are executed for free (i.e., take no apparent time to execute) because they are executed while the register is being loaded. Up to three instructions can be pending at one time with three corresponding scoreboard bits set. By exploiting this feature, system programmers and compilers have a useful tool for optimizing execution speed.

High Bandwidth Local Bus

An 80960KA CPU resides on a high-bandwidth address/data bus known as the local bus (L-Bus). The L-Bus provides a direct communication path between the processor and the memory and I/O subsystem interfaces. The processor uses the local bus to fetch instructions, manipulate memory, and respond to interrupts. Its features include:

- 32-bit multiplexed address/data path
- Four-word burst capability, which allows transfers from 1 to 16 bytes at a time
- High bandwidth reads and writes at 66.7 Mbytes per second
- Special signal to indicate whether a memory transaction can be cached

Figure 5 identifies the groups of signals which constitute the L-Bus. Table 4 lists the function of the L-Bus and other processor-support signals, such as the interrupt lines.

Interrupt Handling

The 80960KA can be interrupted in one of two ways: by the activation of one of four interrupt pins or by sending a message on the processor's data bus.

The 80960KA is unusual in that it automatically handles interrupts on a priority basis and tracks pending interrupts through its on-chip interrupt controller. Two of the interrupt pins can be configured to provide 8259A handshaking for expansion beyond four interrupt lines.

Debug Features

The 80960KA has built-in debug capabilities. There are two types of breakpoints and six different trace modes. The debug features are controlled by two internal 32-bit registers, the Process-Controls Word and the Trace-Controls Word. By setting bits in these control words, a software debug monitor can closely control how the processor responds during program execution.

The 80960KA has both hardware and software breakpoints. It provides two hardware breakpoint registers on-chip which can be set by a special command to any value. When the instruction pointer matches the value in one of the breakpoint registers, the breakpoint will fire, and a breakpoint handling routine is called automatically.

The 80960KA also provides software breakpoints through the use of two instructions, MARK and FMARK. These instructions can be placed at any point in a program and will cause the processor to halt execution at that point and call the breakpoint handling routine. The breakpoint mechanism is easy to use and provides a powerful debugging tool.

Tracing is available for instructions (single-step execution), calls and returns, and branching. Each different type of trace may be enabled separately by a special debug instruction. In each case, the 80960KA executes the instruction first and then calls a trace handling routine (usually part of a software debug monitor). Further program execution is halted until the trace routine is completed. When the trace event handling routine is completed, instruction execution resumes at the next instruction. The

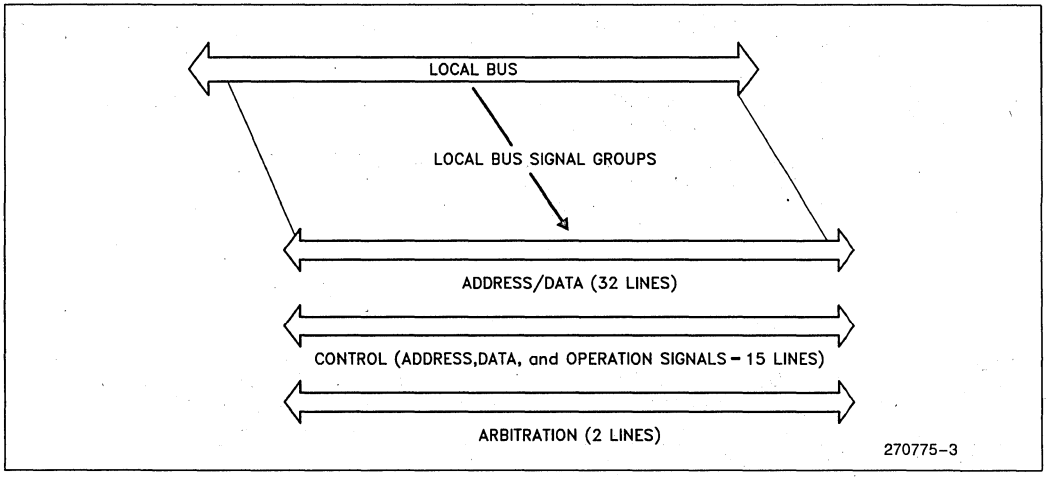


Figure 5. Local Bus Signal Groups

80960KA's tracing mechanisms, which are implemented completely in hardware, greatly simplify the task of testing and debugging software.

FAULT DETECTION

The 80960KA has an automatic mechanism to handle faults. There are ten fault types including trace, arithmetic, and floating-point faults. When the processor detects a fault, it automatically calls the appropriate fault handling routine and saves the current instruction pointer and necessary state information to make efficient recovery possible. The processor posts diagnostic information on the type of fault to a Fault Record. Like interrupt handling routines, fault handling routines are usually written to meet the needs of a specific application and are often included as part of the operating system or kernel.

For each of the ten fault types, there are numerous subtypes that provide specific information about a fault. For example, a floating-point fault may have its subtype set to an Overflow or Zero-Divide fault. The fault handler can use this specific information to respond correctly to the fault.

BUILT-IN TESTABILITY

Upon reset, the 80960KA automatically conducts an extensive internal test (self-test) of its major blocks of logic. Then, before executing its first instruction, it does a zero check sum on the first eight words in memory to ensure that the system has been loaded correctly. If a problem is discovered at any point during the self-test, the 80960KA will assert its FAILURE pin and will not begin program execution. The self-test takes approximately 47,000 cycles to complete.

System manufacturers can use the 80960KA's self-test feature during incoming parts inspection. No special diagnostic programs need to be written, and the test is both thorough and fast. The self-test capability helps ensure that defective parts will be discovered before systems are shipped, and once in the field, the self-test makes it easier to distinguish between problems caused by processor failure and problems resulting from other causes.

CHMOS

The 80960KA is fabricated using Intel's CHMOS IV (Complementary High Speed Metal Oxide Semiconductor) process. This advanced technology eliminates the frequency and reliability limitations of older CMOS processes and opens a new era in microprocessor performance. It combines the high performance capabilities of Intel's industry-leading HMOS technology with the high density and low power characteristics of CMOS. The 80960KA is available at 10, 16, 20 and 25 MHz.

Table 4a. 80960KA Pin Description: L-Bus Signals

Symbol	Type	Name and Function
CLK2	I	**SYSTEM CLOCK** provides the fundamental timing for 80960KA systems. It is divided by two inside the 80960KA to generate the internal processor clock.
LAD_{31} $-LAD_0$	I/O T.S.	**LOCAL ADDRESS/DATA BUS** carries 32-bit physical addresses and data to and from memory. During an address (T_a) cycle, bits 2–31 contain a physical word address (bits 0–1 indicate SIZE; see below). During a data (T_d) cycle, bits 0–31 contain read or write data. The LAD lines are active HIGH and float to a high impedance state when not active. **SIZE,** which is comprised of bits 0–1 of the LAD lines during a T_a cycle, specifies the size of a burst transfer in words. $\underline{LAD}_1$ $\underline{LAD}_0$ 0 0 1 Word 0 1 2 Words 1 0 3 Words 1 1 4 Words
$\overline{ALE}$	O T.S.	**ADDRESS-LATCH ENABLE** indicates the transfer of a physical address. ALE is asserted during a T_a cycle and deasserted before the beginning of the T_d state. It is active LOW and floats to a high impedance state during a hold cycle (T_h or T_{hr}).

I/O = Input/Output, O = Output, I = Input, O.D. = Open-Drain, T.S. = tri-state

Table 4a. 80960KA Pin Description: L-Bus Signals (Continued)

Symbol	Type	Name and Function
$\overline{ADS}$	O O.D.	**ADDRESS/DATA STATUS** indicates an address state. ADS is asserted every T_a state and deasserted during the following T_d state. For a burst transaction, $\overline{ADS}$ is asserted again every T_d state where $\overline{READY}$ was asserted in the previous cycle.
W/$\overline{R}$	O O.D.	**WRITE/READ** specifies, during a T_a cycle, whether the operation is a write or read. It is latched on-chip and remains valid during T_d cycles.
DT/$\overline{R}$	O O.D.	**DATA TRANSMIT/RECEIVE** indicates the direction of data transfer to and from the L-Bus. It is low during T_a and T_d cycles for a read or interrupt acknowledgement; it is high during T_a and T_d cycles for a write. $\overline{DT/R}$ never changes state when $\overline{DEN}$ is asserted (see Timing Diagrams).
$\overline{DEN}$	O O.D.	**DATA ENABLE** is asserted during T_d cycles and indicates transfer of data on the LAD bus lines.
$\overline{READY}$	I	**READY** indicates that data on LAD lines can be sampled or removed. If $\overline{READY}$ is not asserted during a T_d cycle, the T_d cycle is extended to the next cycle by inserting a wait state (T_W), and $\overline{ADS}$ is not asserted in the next cycle.
$\overline{LOCK}$	I/O O.D.	**BUS LOCK** prevents other bus masters from gaining control of the L-Bus following the current cycle (if they would assert $\overline{LOCK}$ to do so). $\overline{LOCK}$ is used by the processor or any bus agent when it performs indivisible Read/Modify/Write (RMW) operations. Do not leave $\overline{LOCK}$ unconnected. It must be pulled high for the processor to function properly. For a read that is designated as a RMW-read, $\overline{LOCK}$ is examined. if asserted, the processor waits until it is not asserted; if not asserted, the processor asserts $\overline{LOCK}$ during the T_a cycle and leaves it asserted. A write that is designated as an RMW-write deasserts $\overline{LOCK}$ in the T_a cycle. During the time $\overline{LOCK}$ is asserted, a bus agent can perform a normal read or write but no RMW operations. $\overline{LOCK}$ is also held asserted during an interrupt-acknowledge transaction.
$\overline{BE_3}$–$\overline{BE_0}$	O O.D.	**BYTE ENABLE LINES** specify which data bytes (up to four) on the bus take part in the current bus cycle. $\overline{BE_3}$ corresponds to LAD_{31}–LAD_{24} and $\overline{BE_0}$ corresponds to LAD_7–LAD_0. The byte enables are provided in advance of data. The byte enables asserted during T_a specify the bytes of the first data word. The byte enables asserted during T_d specify the bytes of the next data word (if any), that is, the word to be transmitted following the next assertion of $\overline{READY}$. The byte enables during the T_d cycles preceding the last assertion of $\overline{READY}$ are undefined. The byte enables are latched on-chip and remain constant from one T_d cycle to the next when $\overline{READY}$ is not asserted. For reads, the byte enables specify the byte(s) that the processor will actually use. L-Bus agents are required to assert only adjacent byte enables (e.g., asserting just $\overline{BE_0}$ and $\overline{BE_2}$ is not permitted), and are required to assert at least one byte enable. To produce address bits A_0 and A_1 externally, they can be decoded from the byte enables.

I/O = Input/Output, O = Output, I = Input, O.D. = Open-Drain, T.S. = tri-state

Table 4a. 80960KA Pin Description: L-Bus Signals (Continued)

Symbol	Type	Name and Function
HOLD/ HLDAR	I	**HOLD:** If the processor is the primary bus master (PBM), the input is interpreted as HOLD, a request from a secondary bus master to acquire the bus. When the processor receives HOLD and grants another master control of the bus, it floats its tri-state bus lines and then asserts HLDA and enters the T_h state. When HOLD is deasserted, the processor will deassert HLDA and go to either the T_i or T_a state. **HOLD ACKNOWLEDGE RECEIVED:** If the processor is a secondary bus master (SBM), the input is HLDAR, which indicates, when HOLDR output is high, that the processor has acquired the bus. Processors and other agents can be told at reset if they are the primary bus master (PBM).
HLDA/ HOLDR	O T.S.	**HOLD ACKNOWLEDGE:** If the processor is a primary bus master, the output is HLDA, which relinquishes control of the bus to another bus master. **HOLD REQUEST:** For secondary bus masters (SBM), the output is HOLDR, which is a request to acquire the bus. The bus is said to be acquired if the agent is a primary bus master and does not have its HLDA output asserted, or if the agent is a secondary bus master and has its HOLD input and HLDA output asserted.
CACHE	O T.S.	**CACHE** indicates if an access is cacheable during a T_a cycle. It is not asserted during any synchronous access, such as a synchronous load or move instruction used for sending an IAC message. The CACHE signal floats to a high impedance state when the processor is idle.

Table 4b. 80960KA Pin Description: Module Support Signals

Symbol	Type	Name and Function
$\overline{\text{BADAC}}$	I	**BAD ACCESS,** if asserted in the cycle following the one in which the last $\overline{\text{READY}}$ of a transaction is asserted, indicates that an unrecoverable error has occurred on the current bus transaction, or that a synchronous load/store instruction has not been acknowledged. **STARTUP:** During system reset, the $\overline{\text{BADAC}}$ signal is interpreted differently. If the signal is high, it indicates that this processor will perform system initialization. If it is low, another processor in the system will perform system initialization instead.
RESET	I	**RESET** clears the internal logic of the processor and causes it to re-initialize. During RESET assertion, the input pins are ignored (except for $\overline{\text{BADAC}}$ and $\overline{\text{IAC}}/\overline{\text{INT}_0}$), the tri-state output pins are placed in a high impedance state, and other output pins are placed in their non-asserted state. RESET must be asserted for at least 41 CLK2 cycles for a predictable RESET. The HIGH to LOW transition of RESET should occur after the rising edge of both CLK2 and the external bus CLK, and before the next rising edge of CLK2.
$\overline{\text{FAILURE}}$	O O.D.	**INITIALIZATION FAILURE** indicates that the processor has failed to initialize correctly. After RESET is deasserted and before the first bus transaction begins, $\overline{\text{FAILURE}}$ is asserted while the processor performs a self-test. If the self-test completes successfully, then $\overline{\text{FAILURE}}$ is deasserted. Next, the processor performs a zero checksum on the first eight words of memory. If it fails, $\overline{\text{FAILURE}}$ is asserted for a second time and remains asserted; if it passes, system initialization continues and $\overline{\text{FAILURE}}$ remains deasserted.
N.C.	N/A	**NOT CONNECTED** indicates pins should not be connected. Never connect any pin marked N.C.

I/O = Input/Output, O = Output, I = Input, O.D. = Open-Drain, T.S. = tri-state

Table 4b. 80960KA Pin Description: Module Support Signals (Continued)

Symbol	Type	Name and Function
$\overline{IAC}$ $\overline{INT0}$	I	**INTERAGENT COMMUNICATION REQUEST/INTERRUPT 0** indicates either that there is a pending IAC message for the processor or an interrupt. The bus interrupt control register determines in which way the signal should be interpreted. To signal an interrupt or IAC request in a synchronous system, this pin (as well as the other interrupt pins) must be enabled by being deasserted for at least one bus cycle and then asserted for at least one additional bus cycle; in an asynchronous system, the pin must remain deasserted for at least two bus cycles and then be asserted for at least two more bus cycles. **LOCAL PROCESSOR NUMBER:** This signal is interpreted differently during system reset. If the signal is at a high voltage level, it indicates that this processor is a primary bus master (Local Processor Number = 0); if it is at a low voltage level, it indicates that this processor is a secondary bus master (Local Processor Number = 1).
INT1	I	**INTERRUPT 1,** like $\overline{INT0}$, provides direct interrupt signaling.
INT2/ INTR	I	**INTERRUPT 2/INTERRUPT REQUEST:** The bus control registers determines how this pin is interpreted. If INT2, it has the same interpretation as the $\overline{INT0}$ and INT1 pins. If INTR, it is used to receive an interrupt request from an external interrupt controller.
$\overline{INT3}$/ $\overline{INTA}$	I/O O.D.	**INTERRUPT 3/INTERRUPT ACKNOWLEDGE:** The bus interrupt control register determines how this pin is interpreted. If $\overline{INT3}$, it has the same interpretation as the $\overline{INT0}$, INT1, and INT2 pins. If $\overline{INTA}$, it is used as an output to control interrupt-acknowledge bus transactions. The $\overline{INTA}$ output is latched on-chip and remains valid during T_d cycles; as an output, it is open-drain.

I/O = Input/Output, O = Output, I = Input, O.D. = Open-Drain, T.S. = tri-state

3

ELECTRICAL SPECIFICATIONS

Power and Grounding

The 80960KA is implemented in CHMOS IV technology and has modest power requirements. Its high clock frequency and numerous output buffers (address/data, control, error, and arbitration signals) can cause power surges as multiple output buffers drive new signal levels simultaneously. For clean on-chip power distribution at high frequency, 12 V_{CC} and 13 V_{SS} pins separately feed functional units of the 80960KA in the PGA.

Power and ground connections must be made to all power and ground pins of the 80960KA. On the circuit board, all V_{CC} pins must be strapped closely together, preferably on a power plane. Likewise, all V_{SS} pins should be strapped together, preferably on a ground plane. These pins may not be connected together within the chip.

Power Decoupling Recommendations

Liberal decoupling capacitance should be placed near the 80960KA. The processor can cause transient power surges when driving the L-Bus, particularly when it is connected to a large capacitive load.

Low inductance capacitors and interconnects are recommended for best high frequency electrical performance. Inductance can be reduced by shortening the board traces between the processor and decoupling capacitors as much as possible. Capacitors specifically designed for PGA packages are also commercially available and offer the lowest possible inductance.

Connection Recommendations

For reliable operation, always connect unused inputs to an appropriate signal level. In particular, if one or more interrupt lines are not used, they should be pulled up. No inputs should ever be left floating.

All open-drain outputs require a pullup device. While in some cases a simple pullup resistor will be adequate, we recommend a network of pullup and pulldown resistors biased to a valid V_{IH} ($\geq$3.4V) and terminated in the characteristic impedance of the circuit board. Figure 6 shows our recommendations for the resistor values for both a low and high current drive network, which assumes that the circuit board has a characteristic impedance of 100Ω. The advantage of terminating the output signals in this fashion is that it limits signal swing and reduces AC power consumption.

Characteristic Curves

Figure 7 shows the typical supply current requirements over the operating temperature range of the processor at supply voltage (V_{CC}) of 5V. Figure 8 shows the typical power supply current (I_{CC}) required by the 80960KA at various operating frequencies when measured at three input voltage (V_{CC}) levels.

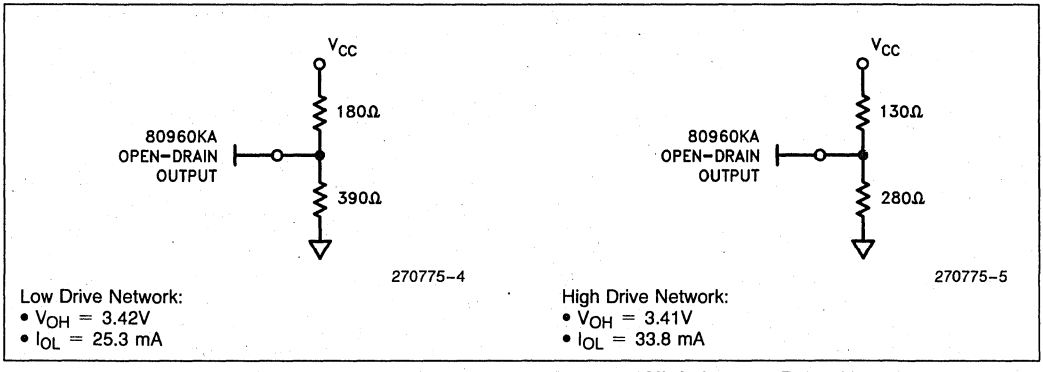
For a given output current (I_{OL}), the curve in Figure 9 shows the worst case output low voltage (V_{OL}).

Figure 10 shows the typical capacitive derating curve for the 80960KA measured from 1.5V on the system clock (CLK) to 1.5V on the falling edge and 1.5V on the rising edge of the L-Bus address/data (LAD) signals.

Test Load Circuit

Figure 13 illustrates the load circuit used to test the 80960KA's tristate pins, and Figure 14 shows the load circuit used to test the open drain outputs. The open drain test uses an active load circuit in the form of a matched diode bridge. Since the open-drain outputs sink current, only the I_{OL} legs of the bridge are necessary and the I_{OH} legs are not used. When the 80960KA driver under test is turned off, the output pin is pulled up to V_{REF} (i.e., V_{OH}). Diode D_1 is turned off and the I_{OL} current source flows through diode D_2.

When the 80960KA open-drain driver under test is on, diode D_1 is also on, and the voltage on the pin being tested drops to V_{OL}. Diode D_2 turns off and I_{OL} flows through diode D_1.



Low Drive Network:
- V_{OH} = 3.42V
- I_{OL} = 25.3 mA

High Drive Network:
- V_{OH} = 3.41V
- I_{OL} = 33.8 mA

Figure 6. Connection Recommendations for Low and High Current Drive Networks

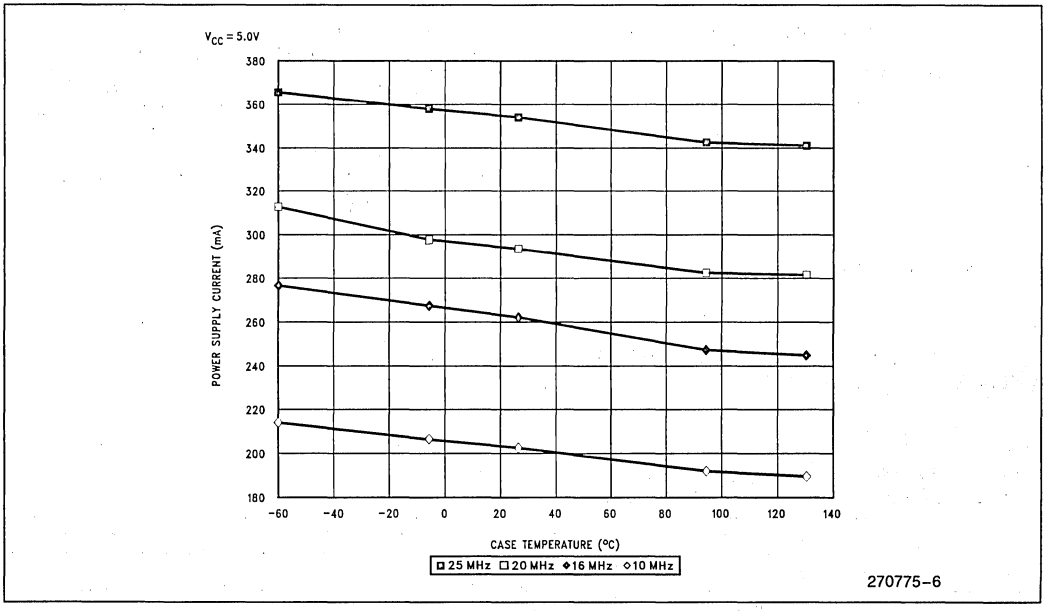


Figure 7. Typical Supply Current (I_{CC})

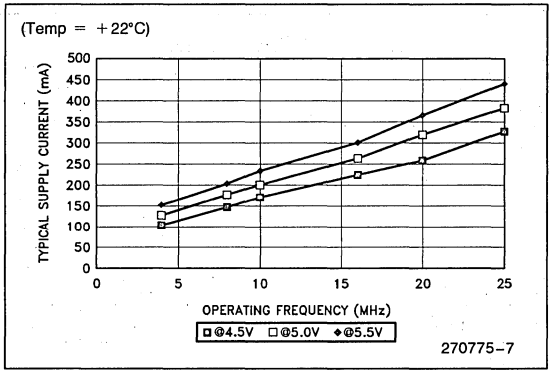


Figure 8. Typical Current vs Frequency

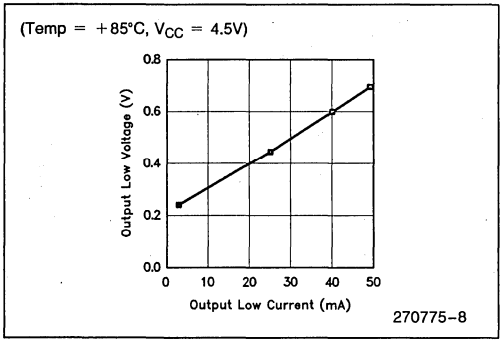


Figure 9. Worst Case Voltage vs Output Current on Open-Drain Pins

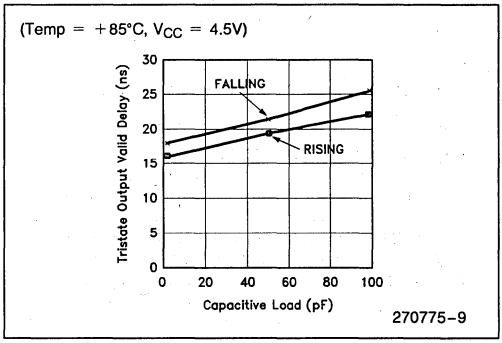


Figure 10. Capacitive Derating Curve

ABSOLUTE MAXIMUM RATINGS*

Operating Temperature .....0°C to +85°C Case
Storage Temperature .....−65°C to +150°C
Voltage on Any Pin.....−0.5V to V_{CC} + 0.5V
Power Dissipation .....2.5W (25 MHz)

NOTICE: This is a production data sheet. The specifications are subject to change without notice.

**WARNING: Stressing the device beyond the "Absolute Maximum Ratings" may cause permanent damage. These are stress ratings only. Operation beyond the "Operating Conditions" is not recommended and extended exposure beyond the "Operating Conditions" may affect device reliability.*

DC CHARACTERISTICS

PGA:
80960KA (16 MHz): T_{CASE} = 0°C to +85°C, V_{CC} = 5V ±10%
80960KA (20 and 25 MHz): T_{CASE} = 0°C to + 85°C, V_{CC} = 5V ±5%

PQFP:
80960KA (10 and 16 MHz): T_{CASE} = 0°C to +100°C, V_{CC} = 5V ±10%
80960KA (20 MHz): T_{CASE} = 0°C to +100°C, V_{CC} = 5V ±5%

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.3	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.3	V	
V_{CL}	CLK2 Input Low Voltage	−0.3	+0.8	V	
V_{CH}	CLK2 Input High Voltage	0.55 V_{CC}	V_{CC} + 0.3	V	
V_{OL}	Output Low Voltage		0.45	V	(1, 5)
V_{OH}	Output High Voltage	2.4		V	(2, 4)
I_{CC}	Power Supply Current: 10 MHz 16 MHz 20 MHz 25 MHz		 300 375 420 480	 mA mA mA mA	
I_{LI}	Input Leakage Current		±15	µA	$0 \le V_{IN} \le V_{CC}$
I_{LO}	Output Leakage Current		±15	µA	$0.45 \le V_O \le V_{CC}$
C_{IN}	Input Capacitance		10	pF	f_C = 1 MHz(3)
C_O	I/O or Output Capacitance		12	pF	f_C = 1 MHz(3)
C_{CLK}	Clock Capacitance		10	pF	f_C = 1 MHz(3)

NOTES:
1. For tri-state outputs, this parameter is measured at:
   Address/Data .....4.0 mA
   Controls .....5.0 mA
2. This parameter is measured at:
   Address/Data .....−1.0 mA
   Controls .....−0.9 mA
   ALE .....−5.0 mA
3. Input, output, and clock capacitance are not tested.
4. Not measured on open-drain outputs.
5. For open-drain outputs .....25 mA

AC SPECIFICATIONS

This section describes the AC specifications for the 80960KA pins. All input and output timings are specified relative to the 1.5V level of the rising edge. Four output timings, the specifications refer to the time it takes the signal to reach 1.5V. For input timings, the specifications refer to the time at which the signal reaches (for input setup) or leaves (for hold time) the TTL levels of LOW (0.8V) or HIGH (2.0V). All AC testing should be done with input clock voltages of 0.4V and 2.4V, except for the clock (CLK2), which should be tested with input voltages of 0.45 V_{CC} and 0.55 V_{CC}.

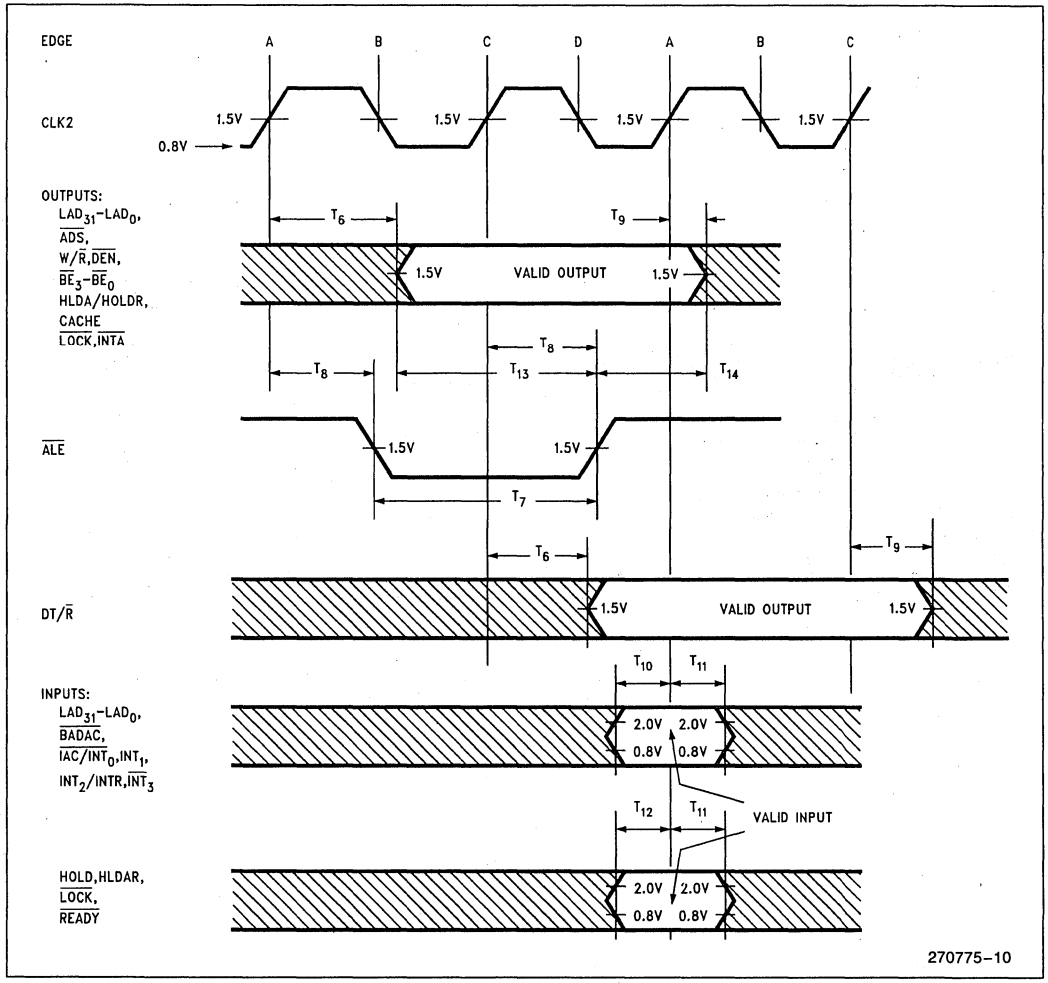


Figure 11. Drive Levels and Timing Relationships for 80960KA Signals

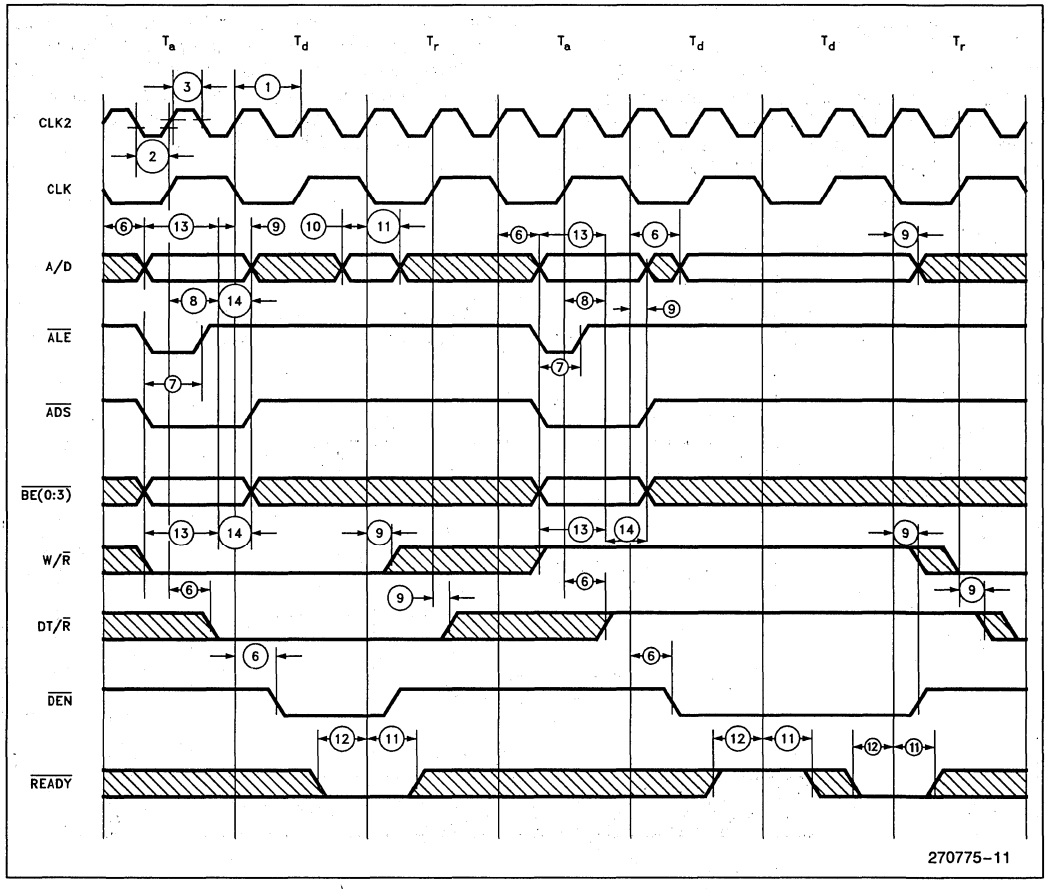


Figure 12. Timing Relationship of L-Bus Signals

AC Specification Tables

80960KA AC Characteristics (10 MHz, PQFP Only)

Symbol	Parameter	Min	Max	Units	Test Conditions
T_1	Processor Clock Period (CLK2)	50	125	ns	V_{IN} = 1.5V
T_2	Processor Clock Low Time (CLK2)	12		ns	V_{IL} = 10% Point = 1.2V
T_3	Processor Clock High Time (CLK2)	12		ns	V_{IH} = 90% Point = 0.1V + 0.5 V_{CC}
T_4	Processor Clock Fall Time (CLK2)		10	ns	V_{IN} = 90% Point to 10% Point
T_5	Processor Clock Rise Time (CLK2)		10	ns	V_{IN} = 10% Point to 90% Point
T_6	Output Valid Delay	2	25	ns	C_L = 100 pF (LAD) C_L = 75 pF (Controls)(2)
T_{6H}	HOLDA Output Valid Delay	4	31	ns	C_L = 75 pF
T_7	$\overline{ALE}$ Width	25		ns	C_L = 75 pF
T_8	$\overline{ALE}$ Output Valid Delay	0	20	ns	C_L = 75 pF(2)
T_9	Output Float Delay	2	20	ns	C_L = 100 pF (LAD) C_L = 75 pF (Controls)
T_{9H}	HOLDA Output Float Delay	4	20	ns	C_L = 75 pF
T_{10}	Input Setup 1	3		ns	
T_{11}	Input Hold	5		ns	
T_{11H}	HOLD Input Hold	4		ns	
T_{12}	Input Setup 2	8		ns	
T_{13}	Setup to $\overline{ALE}$ Inactive	10		ns	C_L = 100 pF (LAD) C_L = 75 pF (Controls)
T_{14}	Hold after $\overline{ALE}$ Inactive	8		ns	C_L = 100 pF (LAD) C_L = 75 pF (Controls)
T_{15}	Reset Hold	3		ns	
T_{16}	Reset Setup	5		ns	
T_{17}	Reset Width	1640		ns	41 CLK2 Periods Minimum

NOTES:
1. $\overline{IAC}/\overline{INT_0}$, INT_1, INT_2/INTR, $\overline{INT_3}$ can be asynchronous.
2. A float condition occurs when the maximum output current becomes less than I_{LO}. Float delay is not tested, but should be no longer than the valid delay.
3. Clock rise and fall time is not tested.

80960KA AC Characteristics (16 MHz)

Symbol	Parameter	Min	Max	Units	Test Conditions
T_1	Processor Clock Period (CLK2)	31.25	125	ns	V_{IN} = 1.5V
T_2	Processor Clock Low Time (CLK2)	8		ns	V_{IL} = 10% Point = 1.2V
T_3	Processor Clock High Time (CLK2)	8		ns	V_{IH} = 90% Point = 0.1V + 0.5 V_{CC}
T_4	Processor Clock Fall Time (CLK2)		10	ns	V_{IN} = 90% Point to 10% Point
T_5	Processor Clock Rise Time (CLK2)		10	ns	V_{IN} = 10% Point to 90% Point
T_6	Output Valid Delay	2	25	ns	C_L = 100 pF (LAD) C_L = 75 pF (Controls)
T_{6H}	HOLDA Output Valid Delay	4	31	ns	C_L = 75 pF
T_7	$\overline{ALE}$ Width	15		ns	C_L = 75 pF
T_8	$\overline{ALE}$ Output Valid Delay	0	20	ns	C_L = 75 pF(2)
T_9	Output Float Delay	2	20	ns	C_L = 100 pF (LAD) C_L = 75 pF (Controls)(2)
T_{9H}	HOLDA Output Float Delay	4	20	ns	C_L = 75 pF
T_{10}	Input Setup 1	3		ns	
T_{11}	Input Hold	5		ns	
T_{11H}	HOLD Input Hold	4		ns	
T_{12}	Input Setup 2	8		ns	
T_{13}	Setup to $\overline{ALE}$ Inactive	10		ns	C_L = 100 pF (LAD) C_L = 75 pF (Controls)
T_{14}	Hold after $\overline{ALE}$ Inactive	8		ns	C_L = 100 pF (LAD) C_L = 75 pF (Controls)
T_{15}	Reset Hold	3		ns	
T_{16}	Reset Setup	5		ns	
T_{17}	Reset Width	1281		ns	41 CLK2 Periods Minimum

NOTES:
1. $\overline{IAC}/\overline{INT_0}$, INT_1, INT_2/INTR, $\overline{INT_3}$ can be asynchronous.
2. A float condition occurs when the maximum output current becomes less than I_{LO}. Float delay is not tested, but should be no longer than the valid delay.
3. Clock rise and fall time is not tested.

80960KA AC Characteristics (20 MHz)

Symbol	Parameter	Min	Max	Units	Test Conditions
T_1	Processor Clock Period (CLK2)	25	125	ns	V_{IN} = 1.5V
T_2	Processor Clock Low Time (CLK2)	6		ns	V_{IL} = 10% Point = 1.2V
T_3	Processor Clock High Time (CLK2)	6		ns	V_{IH} = 90% Point = 0.1V + 0.5 V_{CC}
T_4	Processor Clock Fall Time (CLK2)		10	ns	V_{IN} = 90% Point to 10% Point
T_5	Processor Clock Rise Time (CLK2)		10	ns	V_{IN} = 10% Point to 90% Point
T_6	Output Valid Delay	2	20	ns	C_L = 60 pF (LAD) C_L = 50 pF (Controls)
T_{6H}	HOLDA Output Valid Delay	4	26	ns	C_L = 50 pF
T_7	$\overline{ALE}$ Width	12		ns	C_L = 50 pF
T_8	$\overline{ALE}$ Output Valid Delay	0	20	ns	C_L = 50 pF(2)
T_9	Output Float Delay	2	20	ns	C_L = 60 pF (LAD) C_L = 50 pF (Controls)(2)
T_{9H}	HOLDA Output Float Delay	4	20	ns	C_L = 50 pF
T_{10}	Input Setup 1	3		ns	
T_{11}	Input Hold	5		ns	
T_{11H}	HOLD Input Hold	4		ns	
T_{12}	Input Setup 2	7		ns	
T_{13}	Setup to $\overline{ALE}$ Inactive	10		ns	C_L = 60 pF (LAD) C_L = 50 pF (Controls)
T_{14}	Hold after $\overline{ALE}$ Inactive	8		ns	C_L = 60 pF (LAD) C_L = 50 pF (Controls)
T_{15}	Reset Hold	3		ns	
T_{16}	Reset Setup	5		ns	
T_{17}	Reset Width	1025		ns	41 CLK2 Periods Minimum

NOTES:
1. $\overline{IAC}/\overline{INT_0}$, INT_1, INT_2/INTR, $\overline{INT_3}$ can be asynchronous.
2. A float condition occurs when the maximum output current becomes less than I_{LO}. Float delay is not tested, but should be no longer than the valid delay.
3. Clock rise and fall time is not tested.

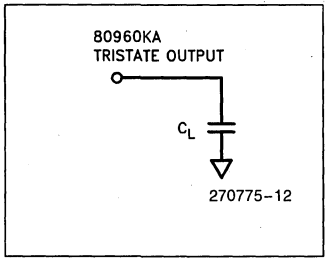


Figure 13. Test Load Circuit for Tri-State Output Pins

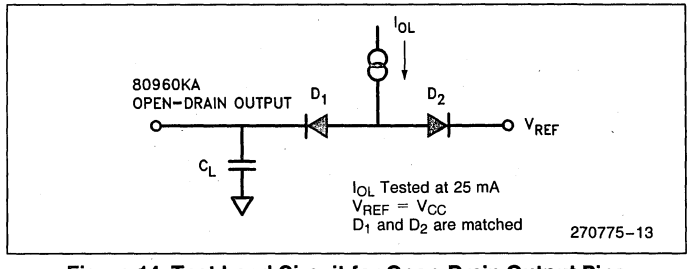


Figure 14. Test Load Circuit for Open-Drain Output Pins

80960KA AC Characteristics (25 MHz, PGA Only)

Symbol	Parameter	Min	Max	Units	Test Conditions
T_1	Processor Clock Period (CLK2)	20	125	ns	V_{IN} = 1.5V
T_2	Processor Clock Low Time (CLK2)	5		ns	V_{IL} = 10% Point = 1.2V
T_3	Processor Clock High Time	5		ns	V_{IH} = 90% Point = 0.1V + 0.5 V_{CC}
T_4	Processor Clock Fall Time (CLK2)		10	ns	V_{IN} = 90% Point to 10% Point
T_5	Processor Clock Rise Time (CLK2)		10	ns	V_{IN} = 10% Point to 90% Point
T_6	Output Valid Delay	2	18	ns	C_L = 60 pF (LAD) C_L = 50 pF (Controls)
T_{6H}	HOLDA Output Valid Delay	4	24	ns	C_L = 50 pF
T_7	$\overline{ALE}$ Width	12		ns	C_L = 50 pF
T_8	$\overline{ALE}$ Output Valid Delay	0	20	ns	C_L = 50 pF(2)
T_9	Output Float Delay	2	18	ns	C_L = 60 pF (LAD) C_L = 50 pF (Controls)
T_{9H}	HOLDA Output Float Delay	4	20	ns	C_L = 50 pF
T_{10}	Input Setup 1	3		ns	
T_{11}	Input Hold	5		ns	
T_{11H}	HOLD Input Hold	4		ns	
T_{12}	Input Setup 2	7		ns	
T_{13}	Setup to $\overline{ALE}$ Inactive	8		ns	C_L = 60 pF (LAD) C_L = 50 pF (Controls)
T_{14}	Hold after $\overline{ALE}$ Inactive	8		ns	C_L = 60 pF (LAD) C_L = 50 pF (Controls)
T_{15}	Reset Hold	3		ns	
T_{16}	Reset Setup	5		ns	
T_{17}	Reset Width	820		ns	41 CLK2 Periods Minimum

NOTES:
1. $\overline{IAC}/\overline{INT0}$, INT1, INT2/INTR, $\overline{INT3}$ can be asynchronous.
2. A float condition occurs when the maximum output current becomes less than I_{LO}. Float delay is not tested, but should be no longer than the valid delay.
3. Clock rise and fall time is not tested.

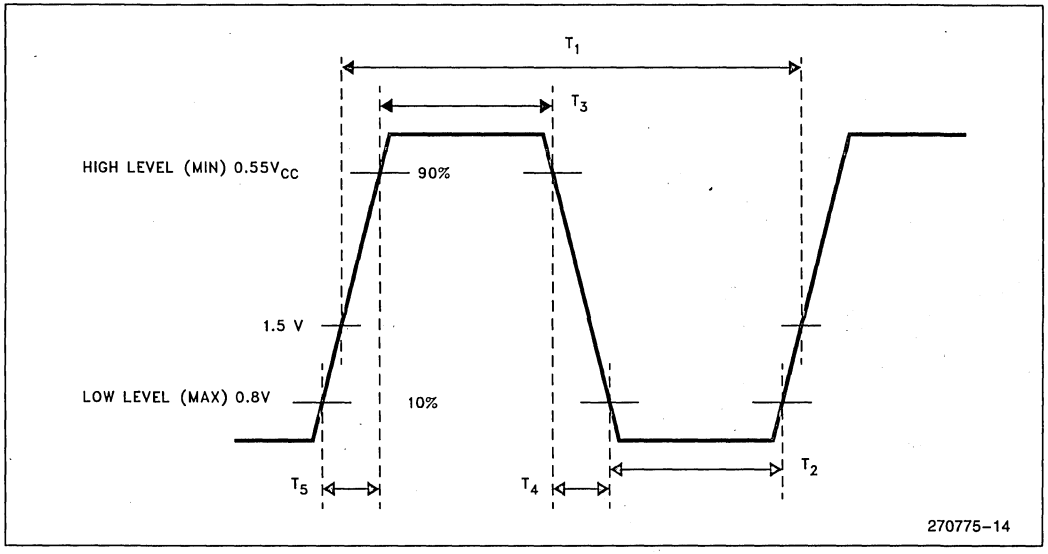


Figure 15. Processor Clock Pulse (CLK2)

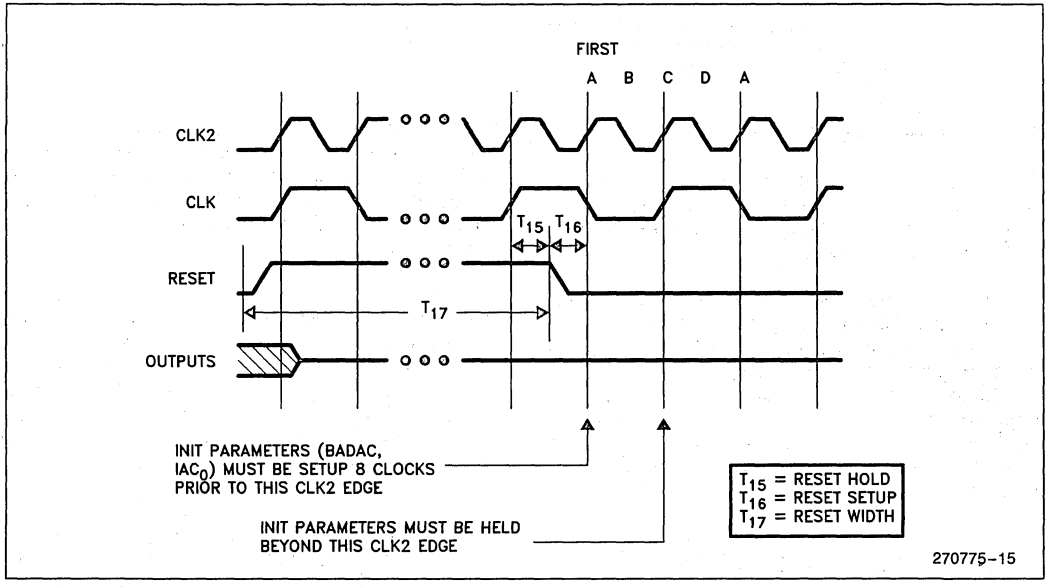


Figure 16. RESET Signal Timing

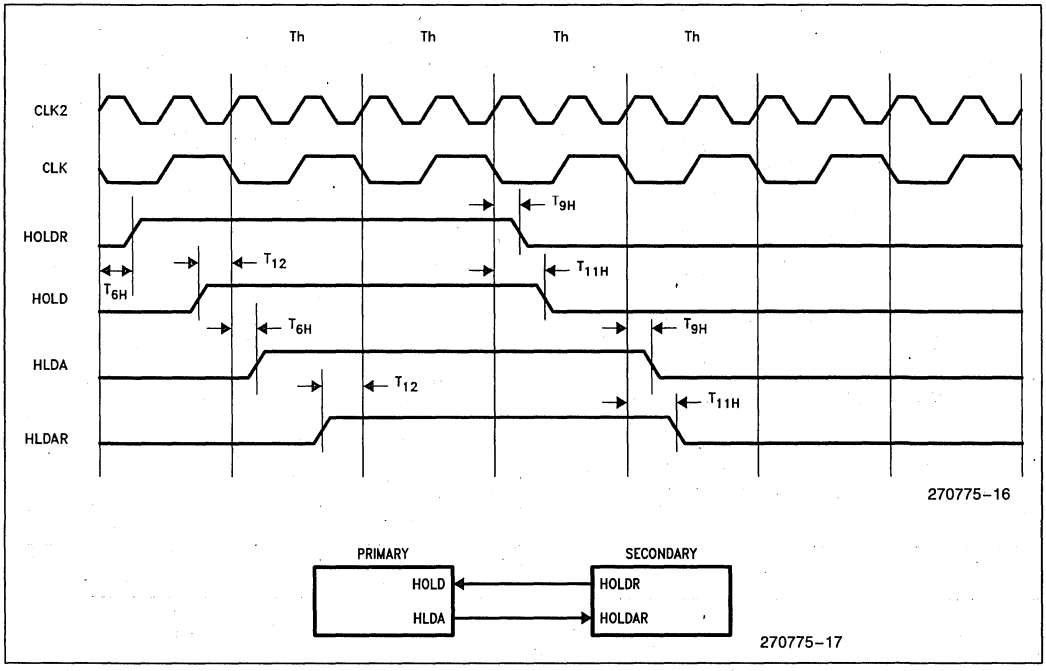


Figure 17. Hold Timing

Design Considerations

Input hold times can be disregarded by the designer whenever the input is removed because a subsequent output from the processor is deasserted (e.g., $\overline{DEN}$ becomes deasserted).

Whenever the processor generates an output that indicates a transition into a subsequent state, any outputs that are specified to be tri-stated in this new state are guaranteed to be tri-stated. For example, in the T_d cycle following a T_a cycle for a read, the minimum output delay of $\overline{DEN}$ is 2 ns, but the maximum float time of LAD is 20 ns. When $\overline{DEN}$ is asserted, however, the LAD outputs are guaranteed to have been tri-stated.

Designing for the ICE-960KB

The 80960KB In-Circuit Emulator assists in debugging both 80960KA and 80960KB hardware and software designs. The product consists of a probe module, cable, and control unit. Because of the high operating frequency of 80960KA systems, the probe module connects directly to the 80960KA socket.

When designing an 80960KA hardware system that uses the ICE-960KB to debug the system, several electrical and mechanical characteristics should be considered. These considerations include capacitive loading, drive requirement, power requirement and physical layout.

The ICE-960KB probe module increases the load capacitance of each line by up to 25 pF. It also adds one standard Schottky TTL load on the CLK2 line, up to one advanced low-power Schottky TTL load for each control signal line, and one advanced low-power Schottky TTL load for each address/data and byte enable line. These loads originate from the probe module and are driven by the 80960KA processor.

To achieve high noise immunity, the ICE-960KB probe is powered by the user's system. The high-speed probe circuitry draws up to 1.1A plus the maximum current (I_{CC}) of the 80960KA processor.

The mechanical considerations are shown in Figure 18, which illustrates the lateral clearance requirements for the ICE-960KB probe as viewed from above the socket of the 80960KA processor.

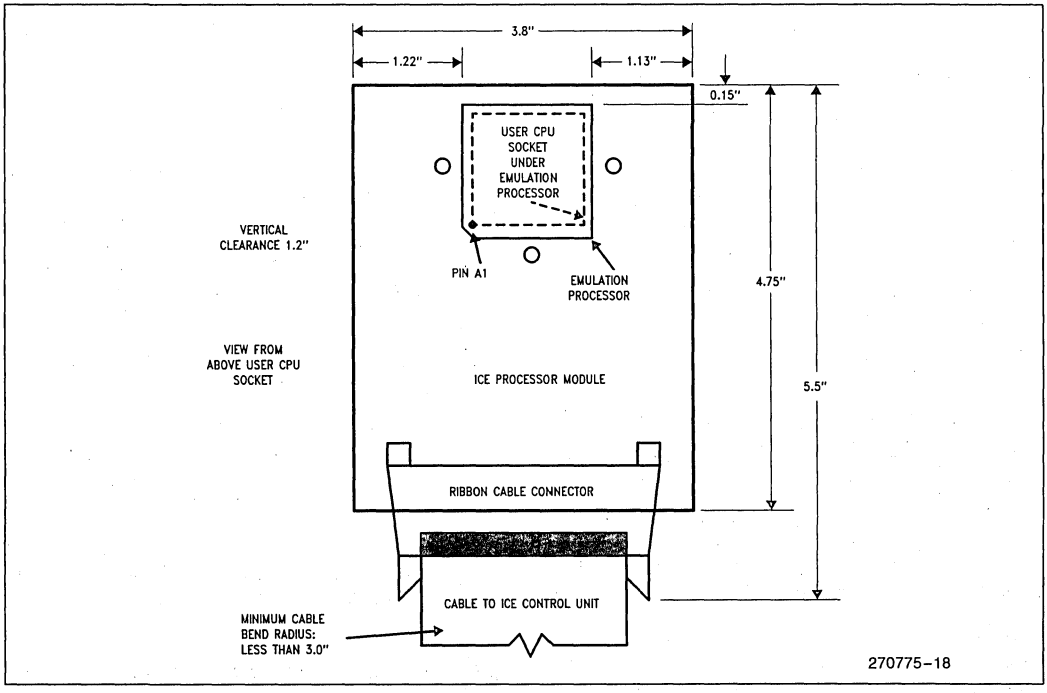


Figure 18. ICE-960KB Lateral Clearance Requirements

MECHANICAL DATA

Package Dimensions and Mounting

The 80960KA is available in two different packages: a 132-lead ceramic pin-grid array (PGA) and a 132-lead plastic quad flat pack (PQFP). Pins in the ceramic package are arranged 0.100 inch (2.54 mm) center-to-center, in a 14 by 14 matrix, three rows around. (See Figure 19.) The plastic package uses fine-pitch gull wing leads arranged in a single row along the perimeter of the package with 0.025 inch (0.64 mm) spacing. (See Figure 20.) Dimensions are given in Figure 21 and Table 7.

There are a wide variety of sockets available for the ceramic PGA package including low-insertion or zero-insertion force mountings, and a choice of terminals such as soldertail, surface mount, or wire wrap. Several applicable sockets are shown in Figure 22.

The PQFP is normally surface mounted to take best advantage of the plastic package's small footprint and low cost. In some applications, however, designers may prefer to use a socket, either to improve heat dissipation or reduce repair costs. Figures 23a and 23b show two of the many sockets available.

Pin Assignment

The PGA and PQFP have different pin assignments. Figure 24 shows the view from the bottom of the PGA (pins facing up) and Figure 25 shows a view from the top of the PGA (pins facing down). Figures 20 and 32 show the top view of the PQFP; notice that the pins are numbered in order from 1 to 132 around the package's perimeter. Tables 5 and 6 list the function of each pin in the PGA, and Tables 8 and 9 list the function of each pin in the PQFP.

V_{CC} and GND connections must be made to multiple V_{CC} and GND pins. Each V_{CC} and GND pin must be connected to the appropriate voltage or ground and externally strapped close to the package. We recommend that you include separate power and ground planes in your circuit board for power distribution.

NOTE:
Pins identified as N.C., "No Connect," should never be connected.

Package Thermal Specification

The 80960KA is specified for operation when case temperature is within the range 0°C to +85°C (PGA) or +100°C (PQFP). The case temperature should be measured at the top center of the package as shown in Figure 26.

The ambient temperature can be calculated from θ_{jc} and θ_{ja} by using the following equations:

$$T_J = T_C + P^*\theta_{jc}$$
$$T_A = T_J - P^*\theta_{ja}$$
$$T_C = T_A + P^*[\theta_{ja} - \theta_{jc}]$$

Values for θ_{ja} and θ_{jc} are given in Table 10 for the PGA package and in Table 11 for the PQFP for various airflows. Note that the θ_{ja} for the PGA package can be reduced by adding a heatsink, while a heatsink is not generally used with the plastic package since it is intended to be surface mounted. The maximum allowable ambient temperature (T_A) permitted without exceeding T_C is shown by the charts in Figures 27 through 30 for 10 MHz, 16 MHz, 20 MHz, and 25 MHz respectively.

The curves assume the maximum permitted supply current (I_{CC}) at each speed, V_{CC} of 5.0V, and a T_{CASE} of +85°C (PGA) or +100°C (PQFP).

If you will be using the 80960KA in a harsh environment where the ambient temperature may exceed the limits for the normal commercial part, you should consider using an extended temperature part. These parts are designed by the prefix "TA" and are available at 16 MHz, 20 MHz and 25 MHz in the ceramic PGA package. The extended operating temperature range is −40°C to +125°C case. Figure 30 shows the maximum allowable ambient temperature for the 20 MHz extended temperature TA80960KA at various airflows. The curve assumes an I_{CC} of 420 mA, V_{CC} of 5.0V, and a T_{CASE} of +125°C.

WAVEFORMS

Figures 33 through 38 show the waveforms for various transactions on the 80960KA's local bus.

SUPPORT COMPONENTS

85C960 Burst Bus Controller

The Intel 85C960 performs burst logic, ready generation, and address decode for the 80960KA and 80960KB. The burst logic supports both standard and burst mode memories and peripherals. The ready generation and timing control supports 0 to 15 wait states across eight address ranges for read/write and burst accesses. The address decoder decodes eight address inputs into four external and four internal chip selects. The wait state and chip select values may be programmed by the user; the timing control and burst logic are fixed.

The 85C960 operates with the 80960KA and 80960KB at all frequencies and consumes only 50 mA at 25 MHz. The 85C960 is housed in a 28-pin, 300-mil ceramic DIP and plastic DIP packages or 28-pin PLCC package for surface mount. In the ceramic DIP package the part is UV-erasable, which makes it easy to revise designs. Order the 85C960 data sheet (No. 290192) for full details.

27960KX Burst Mode EPROM

Intel 27960KX one-megabit EPROM is designed specifically to support the 80960KA and 80960KB. It uses a burst interface to offer near zero wait-state performance without the high cost of alternative memory technologies. The 27960KX removes the need for "dumping" code and data stored in slow EPROMs or ROMs into expensive high-speed "shadow" RAM.

Internally, the 27960KX is organized in blocks of four bytes that are accessed sequentially. The address of the four-byte block is latched and incremented internally. After a set number of wait-states (1 or 2), data is output one word at a time each subsequent clock cycle. High-performance outputs provide zero wait-state data-to-data burst accesses. Extra power and ground pins dedicated to the output reduce the effect of fast output switching on the device. The 27960KX offers 1-0-0-0 performance at 20 MHz and 2-0-0-0 performance at 25 MHz. Full details can be found in the 27960KX data sheet (No. 290237)

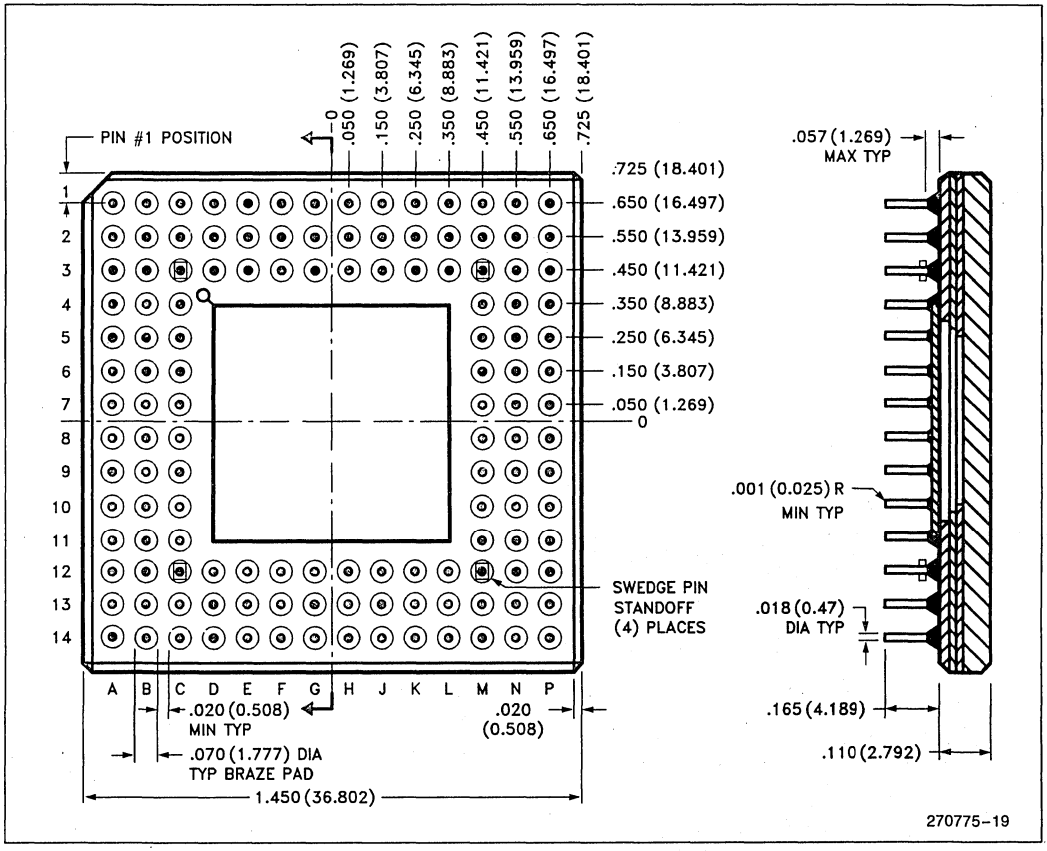


Figure 19. A 132-Lead Pin-Grid Array (PGA) Used to Package the 80960KA

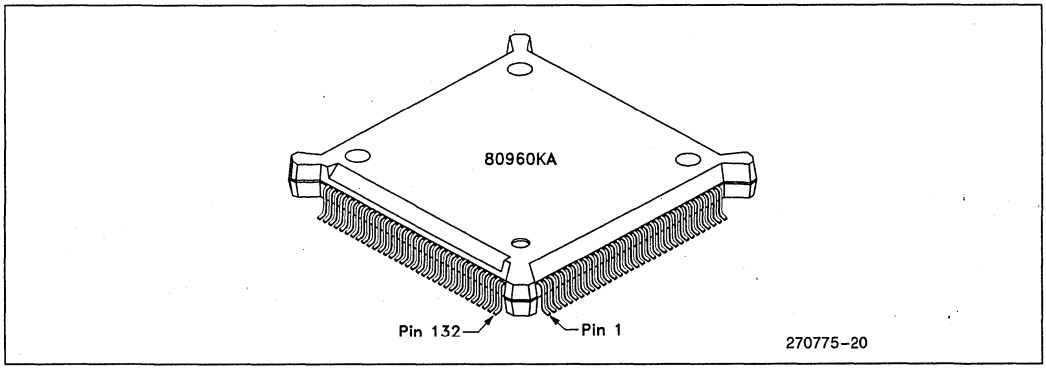


Figure 20. The 132-Lead Plastic Quad Flat Pack (PQFP) used to Package the 80960KA

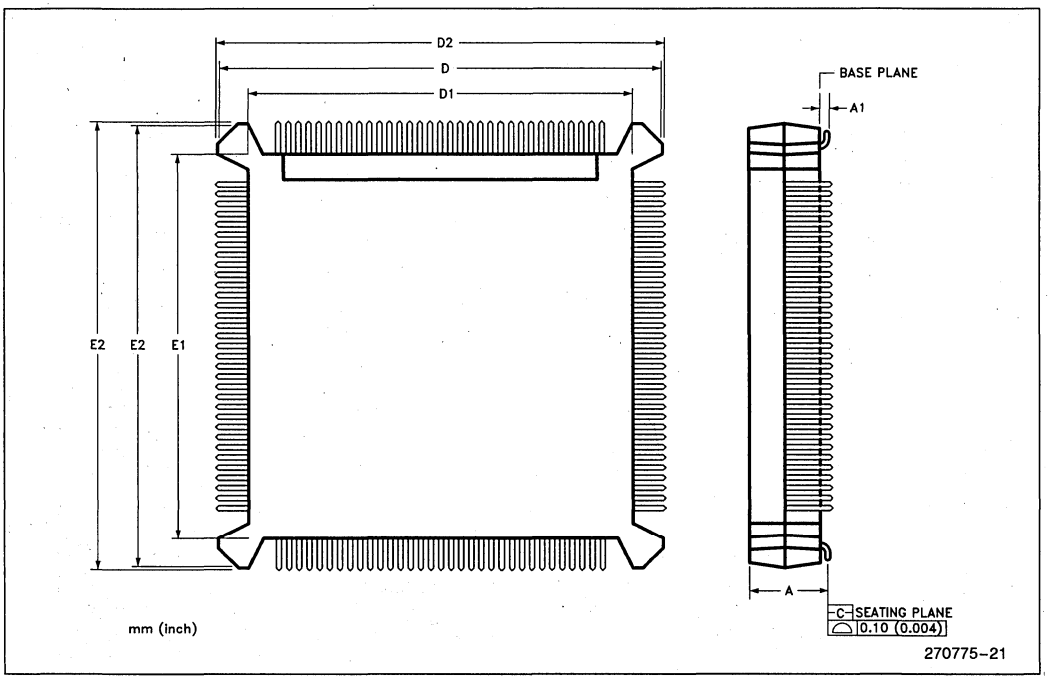


Figure 21a. Principal Dimensions of the 132-Lead PQFP

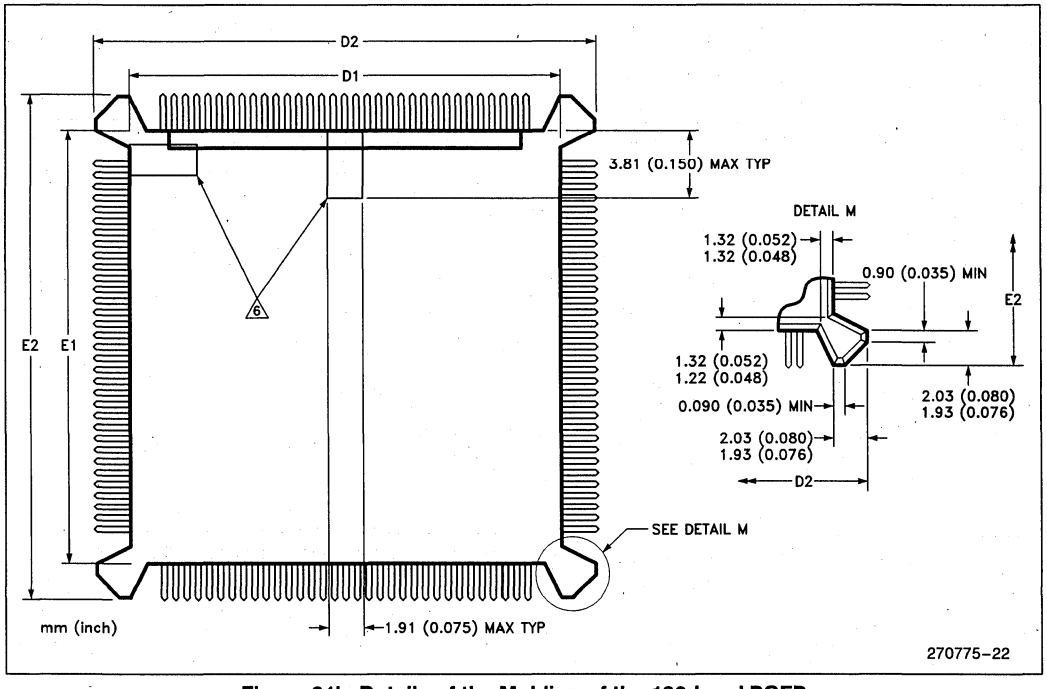


Figure 21b. Details of the Molding of the 132-Lead PQFP

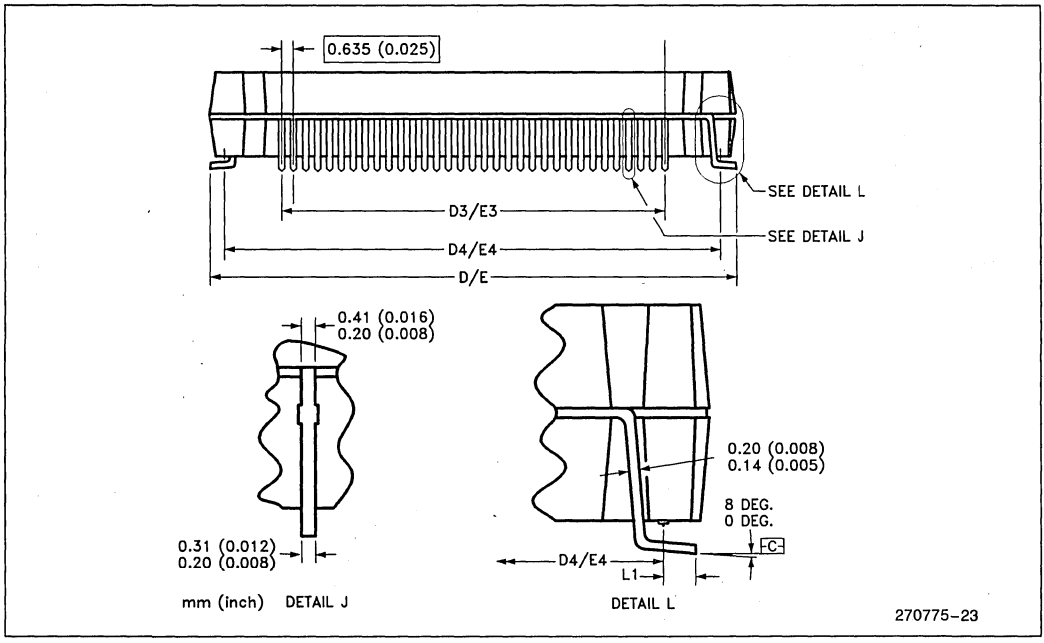


Figure 21c. Terminal Details for the 132-Lead PQFP

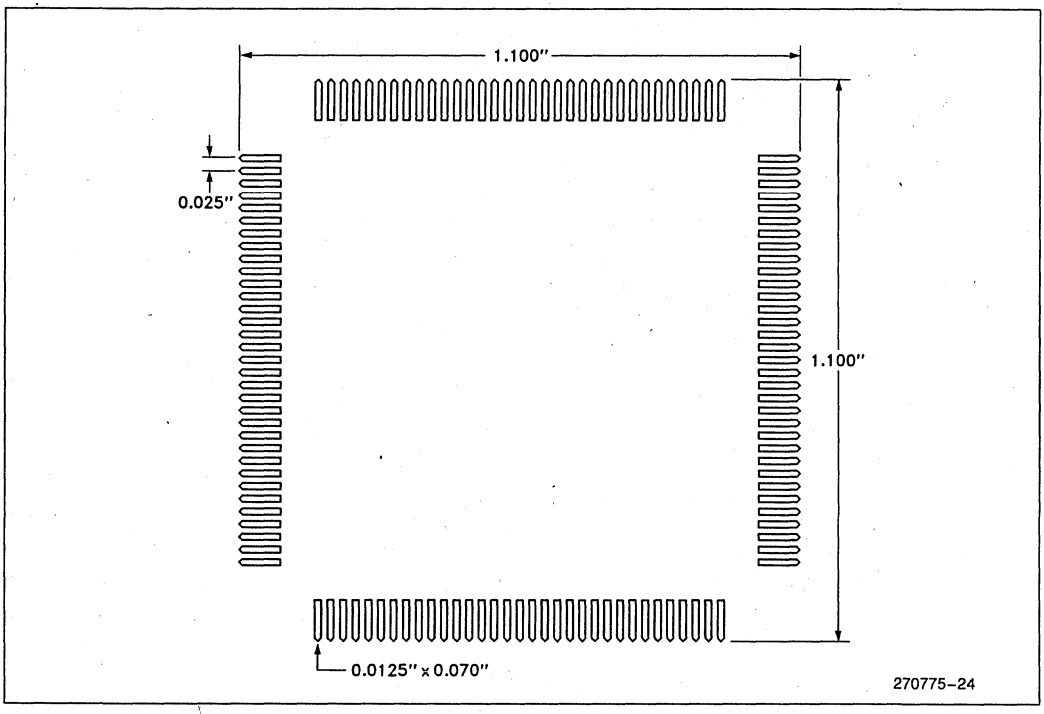


Figure 21d. Board Footprint Area for the 132-Lead PQFP

Table 7. Package Dimension: 80960KA PQFP

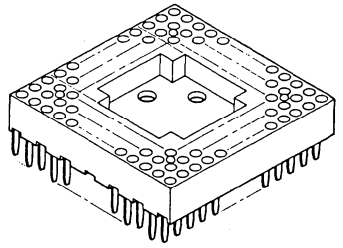
Symbol	Description	Inches		MM	
		Min	Max	Min	Max
N	Leadcount	132 Leads		132 Leads	
A	Package Height	0.160	0.170	4.060	4.320
A1	Standoff	0.020	0.030	0.510	0.760
D,E	Terminal Dimension	1.075	1.085	27.310	27.560
D1,E1	Package Body	0.947	0.953	24.050	24.210
D2,E2	Bumper Distance Without Flash With Flash	 1.097 1.097	 1.103 1.110	 27.860 27.860	 28.010 28.190
D3,E3	Lead Dimension	0.800 REF		20.32 REF	
D4,E4	Foot Radius Location	1.023	1.037	25.890	26.330
L1	Foot Length	0.020	0.030	0.510	0.760

- Low insertion force (LIF) soldertail 55274-1
- Amp tests indicate 50% reduction in insertion force compared to machined sockets

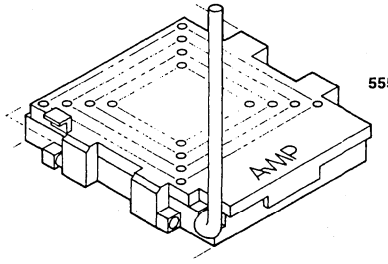
Other socket options

- Zero insertion force (ZIF) soldertail 55583-1
- Zero insertion force (ZIF) Burn-in version 55573-2

Amp Incorporated
(Harrisburg, PA 17105 U.S.A.
Phone 717-564-0100)



55274 = 1



55583 = 1

270775-25

Cam handle locks in low profile position when 80960KA is installed (handle UP for open and DOWN for closed positions).

Courtesy Amp Incorporated

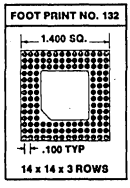
Peel-A-Way* Mylar and Kapton Socket Terminal Carriers

- Low insertion force surface mount CS132-37TG
- Low insertion force soldertail CS132-01TG
- Low insertion force wire-wrap CS132-02TG (two-level) CS132-03TG (thee-level)
- Low insertion force press-fit CS132-05TG

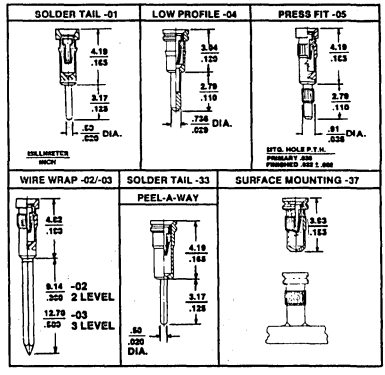
Advanced Interconnections
(5 Division Street)
Warwick, RI 02818 U.S.A.
Phone 401-885-0485)

Peel-A-Way Carrier No. 132:
Kapton Carrier is KS132
Mylar Carrier is MS132

Molded Plastic Body KS132 is shown below:



270775-26



270775-27

Courtesy Advanced Interconnections
(Peel-A-Way Terminal Carriers
U.S. Patent No. 4442938)

*Peel-A-Way is a trademark of Advanced Interconnections.

Figure 22. Several Socket Options for Mounting the 80960KA

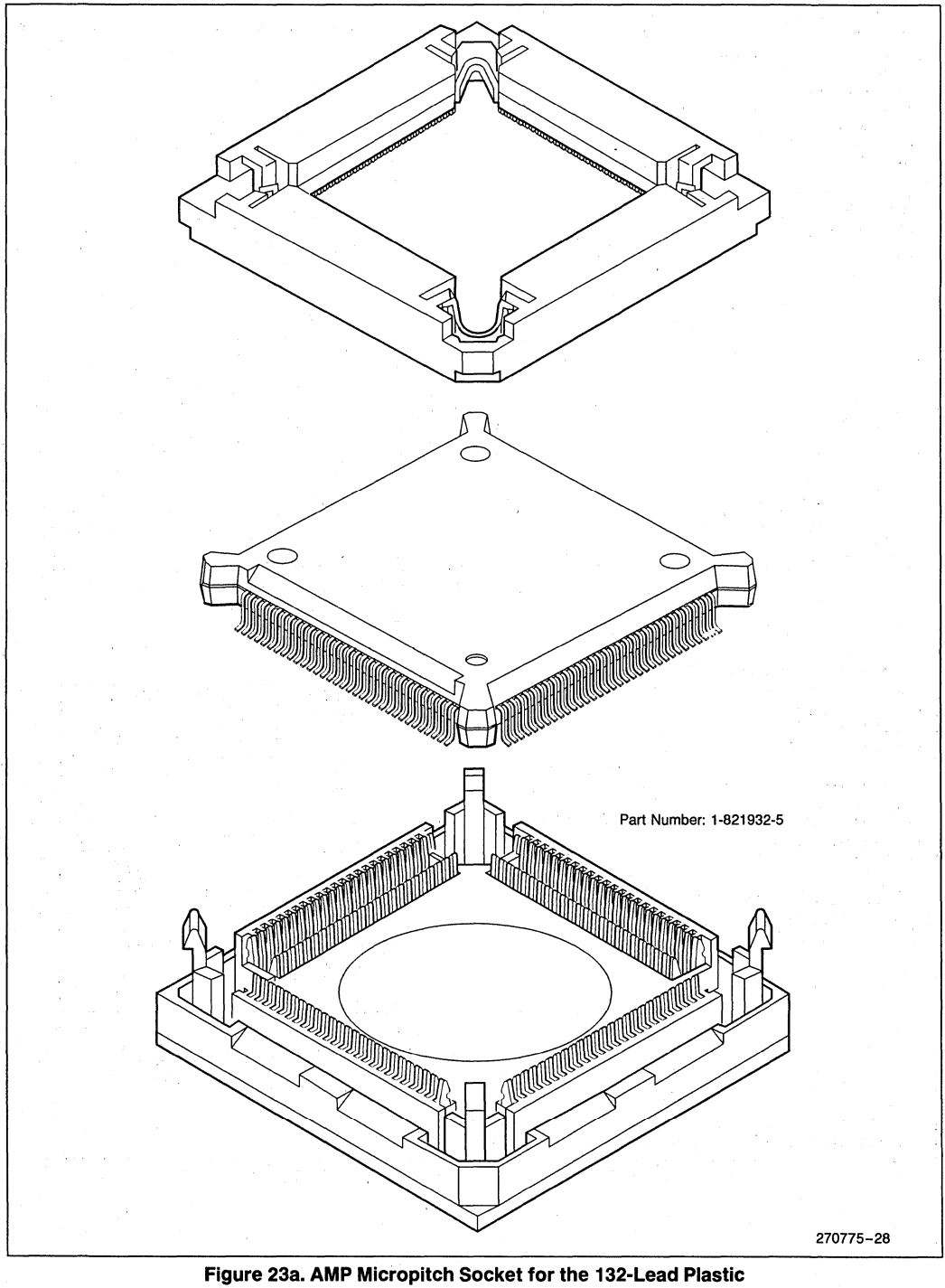


Figure 23a. AMP Micropitch Socket for the 132-Lead Plastic Quad Flat Pack, 0.025″ Lead Spacing, Gull Wing Leads

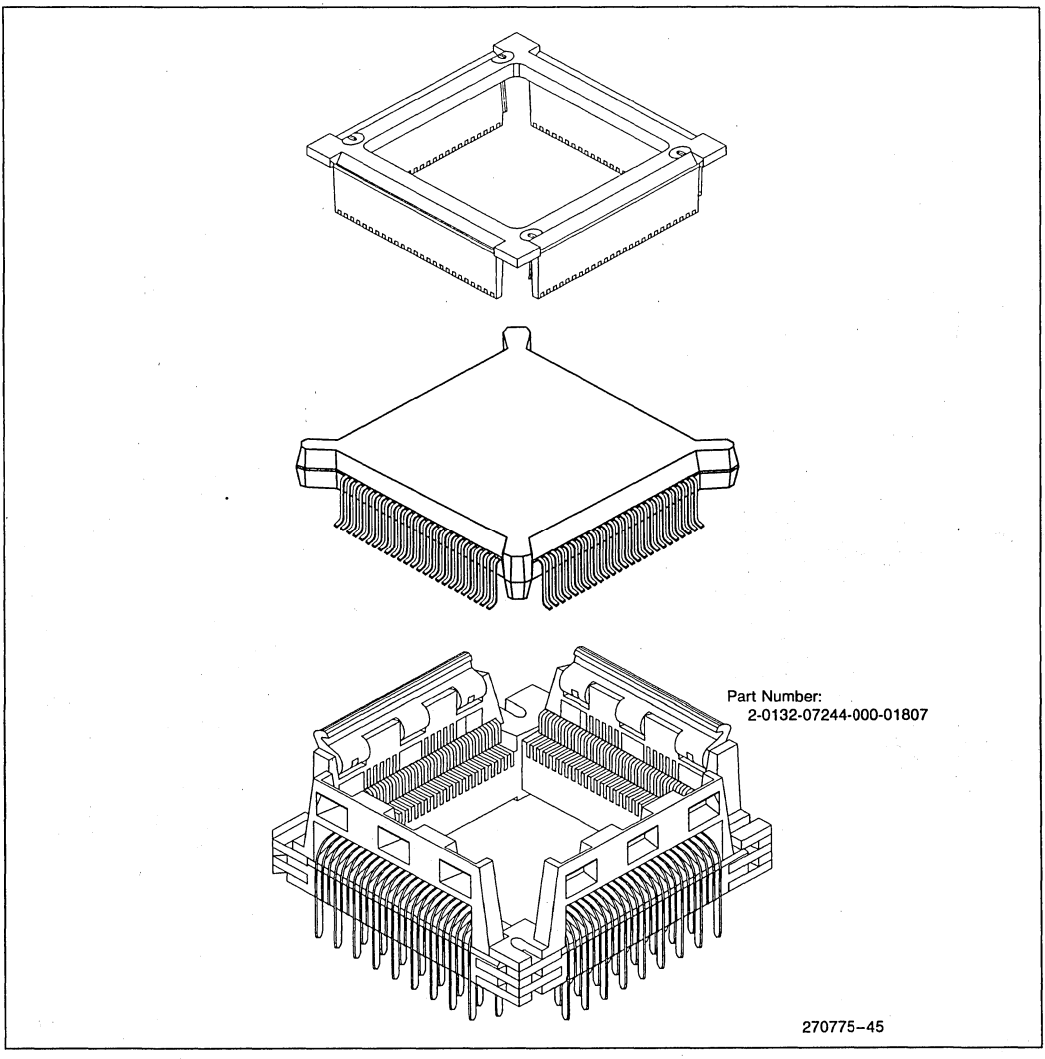


Figure 23b. 3M Company PQFP Socket and Lid

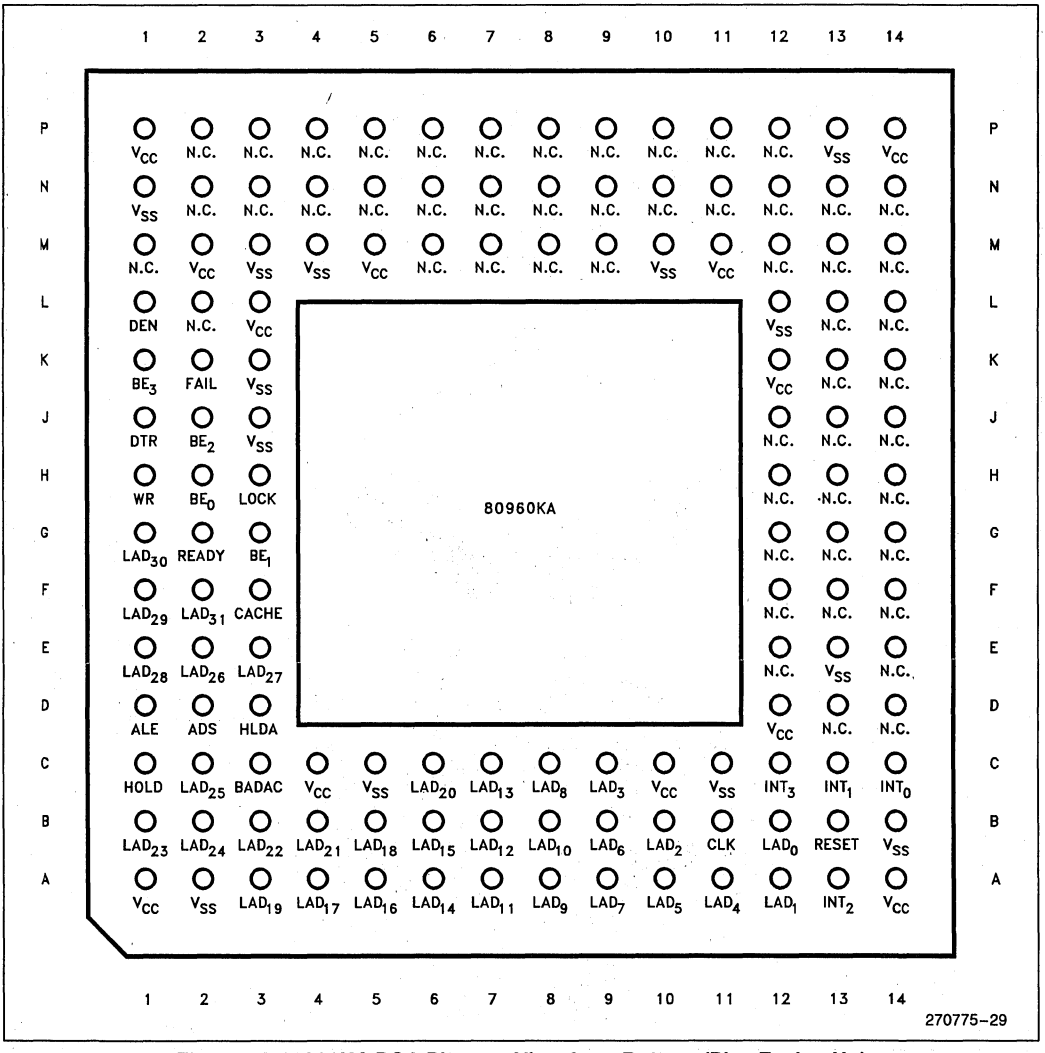


Figure 24. 80960KA PGA Pinout—View from Bottom (Pins Facing Up)

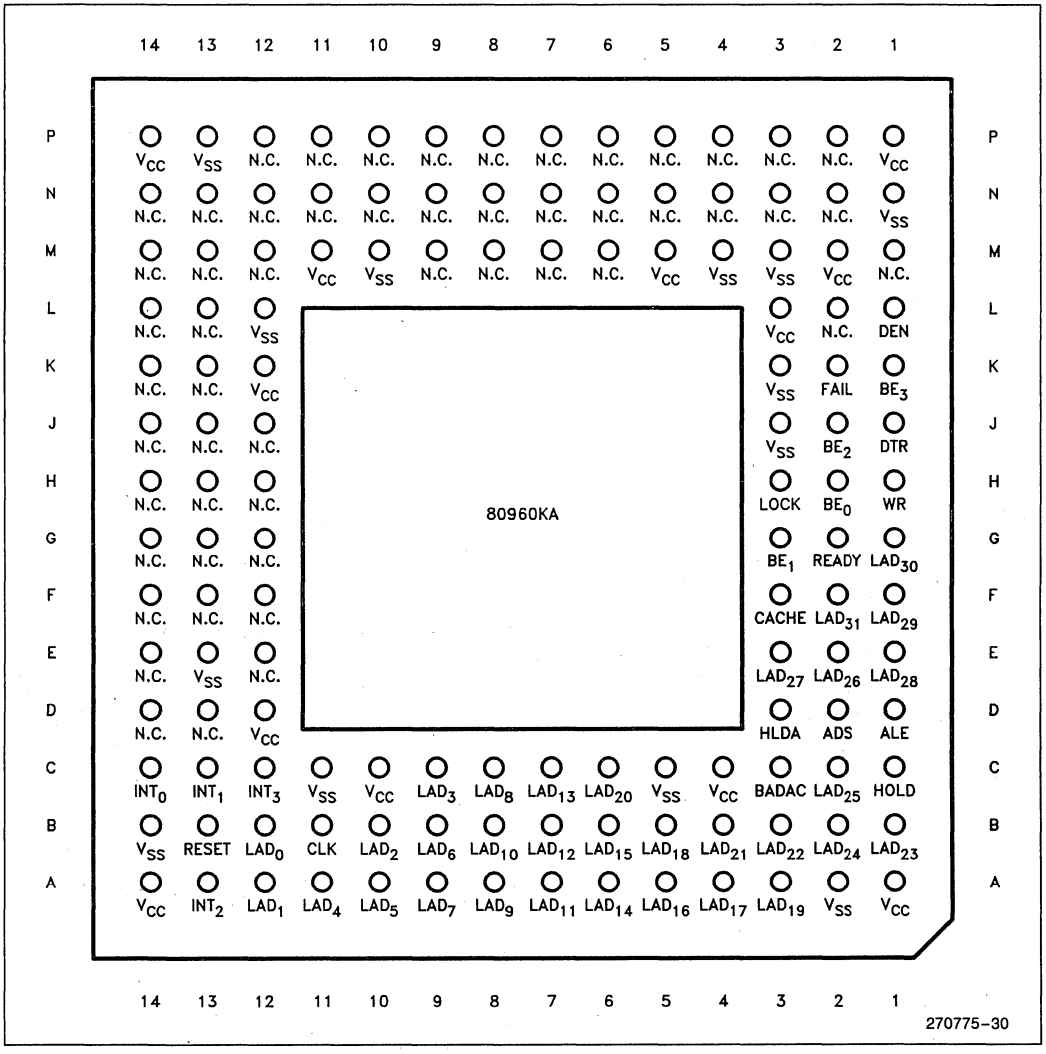


Figure 25. 80960KA PGA Pinout—View from Top (Pins Facing Down)

Table 5. 80960KA PGA Pinout—In Pin Order

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
A1	V_{CC}	C6	LAD_{20}	H1	W/$\overline{R}$	M10	V_{SS}
A2	V_{SS}	C7	LAD_{13}	H2	$\overline{BE_0}$	M11	V_{CC}
A3	LAD_{19}	C8	LAD_8	H3	$\overline{LOCK}$	M12	N.C.
A4	LAD_{17}	C9	LAD_3	H12	N.C.	M13	N.C.
A5	LAD_{16}	C10	V_{CC}	H13	N.C.	M14	N.C.
A6	LAD_{14}	C11	V_{SS}	H14	N.C.	N1	V_{SS}
A7	LAD_{11}	C12	$\overline{INT_3}$/$\overline{INTA}$	J1	DT/$\overline{R}$	N2	N.C.
A8	LAD_9	C13	INT_1	J2	$\overline{BE_2}$	N3	N.C.
A9	LAD_7	C14	$\overline{IAC}$/$\overline{INT_0}$	J3	V_{SS}	N4	N.C.
A10	LAD_5	D1	$\overline{ALE}$	J12	N.C.	N5	N.C.
A11	LAD_4	D2	$\overline{ADS}$	J13	N.C.	N6	N.C.
A12	LAD_1	D3	HLDA/HLDR	J14	N.C.	N7	N.C.
A13	INT_2/INTR	D12	V_{CC}	K1	$\overline{BE_3}$	N8	N.C.
A14	V_{CC}	D13	N.C.	K2	$\overline{FAILURE}$	N9	N.C.
B1	LAD_{23}	D14	N.C.	K3	V_{SS}	N10	N.C.
B2	LAD_{24}	E1	LAD_{28}	K12	V_{CC}	N11	N.C.
B3	LAD_{22}	E2	LAD_{26}	K13	N.C.	N12	N.C.
B4	LAD_{21}	E3	LAD_{27}	K14	N.C.	N13	N.C.
B5	LAD_{18}	E12	N.C.	L1	$\overline{DEN}$	N14	N.C.
B6	LAD_{15}	E13	V_{SS}	L2	N.C.	P1	V_{CC}
B7	LAD_{12}	E14	N.C.	L3	V_{CC}	P2	N.C.
B8	LAD_{10}	F1	LAD_{29}	L12	V_{SS}	P3	N.C.
B9	LAD_6	F2	LAD_{31}	L13	N.C.	P4	N.C.
B10	LAD_2	F3	CACHE	L14	N.C.	P5	N.C.
B11	CLK2	F12	N.C.	M1	N.C.	P6	N.C.
B12	LAD_0	F13	N.C.	M2	V_{CC}	P7	N.C.
B13	RESET	F14	N.C.	M3	V_{SS}	P8	N.C.
B14	V_{SS}	G1	LAD_{30}	M4	V_{SS}	P9	N.C.
C1	HOLD/HLDAR	G2	$\overline{READY}$	M5	V_{CC}	P10	N.C.
C2	LAD_{25}	G3	$\overline{BE_1}$	M6	N.C.	P11	N.C.
C3	$\overline{BADAC}$	G12	N.C.	M7	N.C.	P12	N.C.
C4	V_{CC}	G13	N.C.	M8	N.C.	P13	V_{SS}
C5	V_{SS}	G14	N.C.	M9	N.C.	P14	V_{CC}

Table 6. 80960KA PGA Pinout—In Signal Order

Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin
$\overline{ADS}$	D2	LAD_{15}	B6	N.C.	J14	N.C.	P9
$\overline{ALE}$	D1	LAD_{16}	A5	N.C.	K13	N.C.	P10
$\overline{BADAC}$	C3	LAD_{17}	A4	N.C.	K14	N.C.	P11
$\overline{BE_0}$	H2	LAD_{18}	B5	N.C.	L13	N.C.	P12
$\overline{BE_1}$	G3	LAD_{19}	A3	N.C.	L14	N.C.	L2
$\overline{BE_2}$	J2	LAD_{20}	C6	N.C.	M1	$\overline{READY}$	G2
$\overline{BE_3}$	K1	LAD_{21}	B4	N.C.	M6	RESET	B13
CACHE	F3	LAD_{22}	B3	N.C.	M7	V_{CC}	A1
CLK2	B11	LAD_{23}	B1	N.C.	M8	V_{CC}	A14
$\overline{DEN}$	L1	LAD_{24}	B2	N.C.	M9	V_{CC}	C4
DT/$\overline{R}$	J1	LAD_{25}	C2	N.C.	M12	V_{CC}	C10
$\overline{FAILURE}$	K2	LAD_{26}	E2	N.C.	M13	V_{CC}	D12
HLDA/HOLDR	D3	LAD_{27}	E3	N.C.	M14	V_{CC}	K12
HOLD/HLDAR	C1	LAD_{28}	E1	N.C.	N2	V_{CC}	L3
$\overline{IAC}/\overline{INT_0}$	C14	LAD_{29}	F1	N.C.	N3	V_{CC}	M2
INT_1	C13	LAD_{30}	G1	N.C.	N4	V_{CC}	M5
INT_2/INTR	A13	LAD_{31}	F2	N.C.	N5	V_{CC}	M11
$\overline{INT_3}/\overline{INTA}$	C12	$\overline{LOCK}$	H3	N.C.	N6	V_{CC}	P1
LAD_0	B12	N.C.	D13	N.C.	N7	V_{CC}	P14
LAD_1	A12	N.C.	D14	N.C.	N8	V_{SS}	A2
LAD_2	B10	N.C.	E12	N.C.	N9	V_{SS}	B14
LAD_3	C9	N.C.	E14	N.C.	N10	V_{SS}	C5
LAD_4	A11	N.C.	F12	N.C.	N11	V_{SS}	C11
LAD_5	A10	N.C.	F13	N.C.	N12	V_{SS}	E13
LAD_6	B9	N.C.	F14	N.C.	N13	V_{SS}	J3
LAD_7	A9	N.C.	G12	N.C.	N14	V_{SS}	K3
LAD_8	C8	N.C.	G13	N.C.	P2	V_{SS}	L12
LAD_9	A8	N.C.	G14	N.C.	P3	V_{SS}	M3
LAD_{10}	B8	N.C.	H12	N.C.	P4	V_{SS}	M4
LAD_{11}	A7	N.C.	H13	N.C.	P5	V_{SS}	M10
LAD_{12}	B7	N.C.	H14	N.C.	P6	V_{SS}	N1
LAD_{13}	C7	N.C.	J12	N.C.	P7	V_{SS}	P13
LAD_{14}	A6	N.C.	J13	N.C.	P8	W/$\overline{R}$	H1

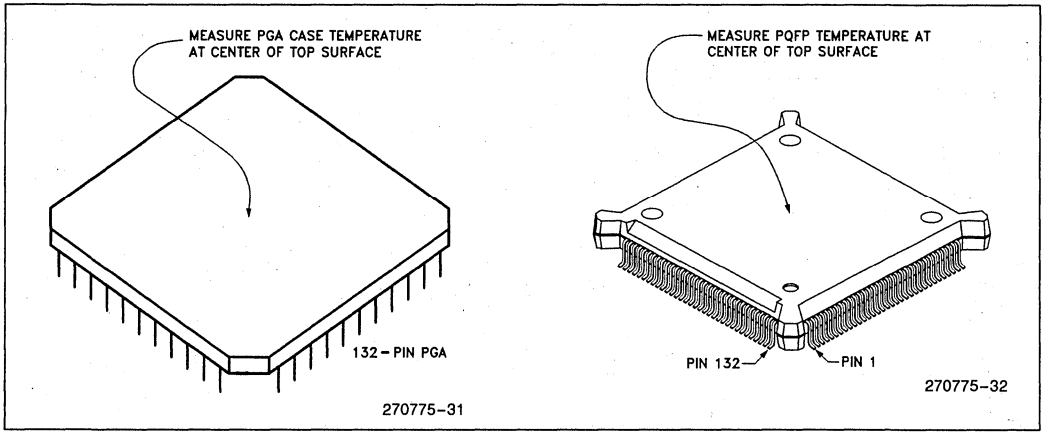


Figure 26. Measuring 80960KA PGA and PQFP Case Temperature

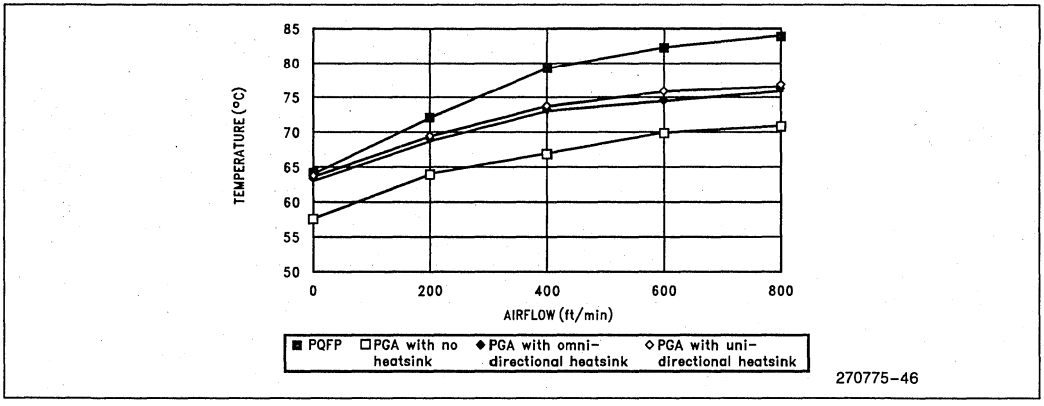


Figure 27. 10 MHz 80960 K-Series Maximum Allowable Ambient Temperature

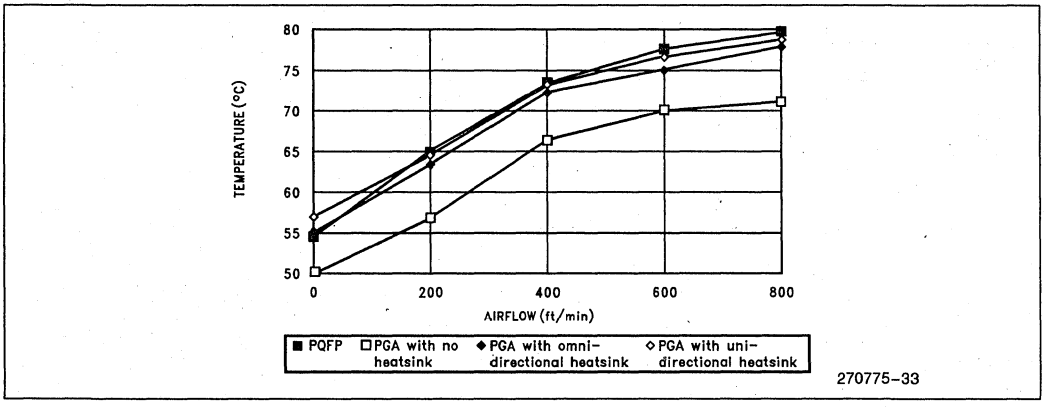


Figure 28. 16 MHz 80960 K-Series Maximum Allowable Ambient Temperature

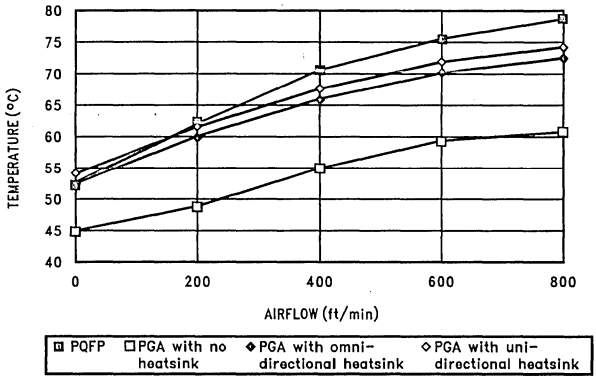


Figure 29. 20 MHz 80960 K-Series Maximum Allowable Ambient Temperature

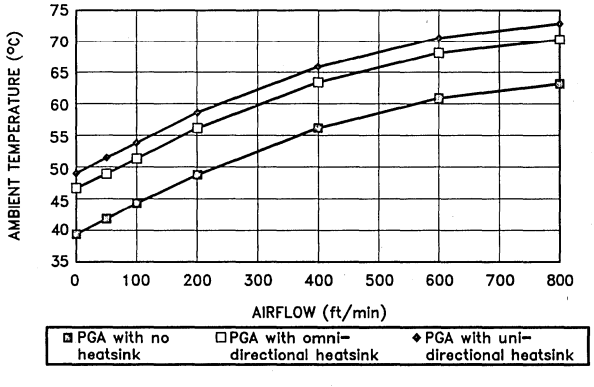


Figure 30. Maximum Allowable Ambient Temperature for the 80960KA at 25 MHz (available in PGA only)

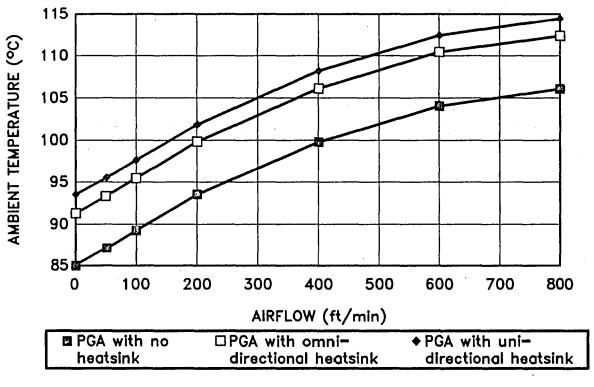


Figure 31. Maximum Allowable Ambient Temperature for the Extended Temperature TA-80960KA at 20 MHz (available in PGA only)

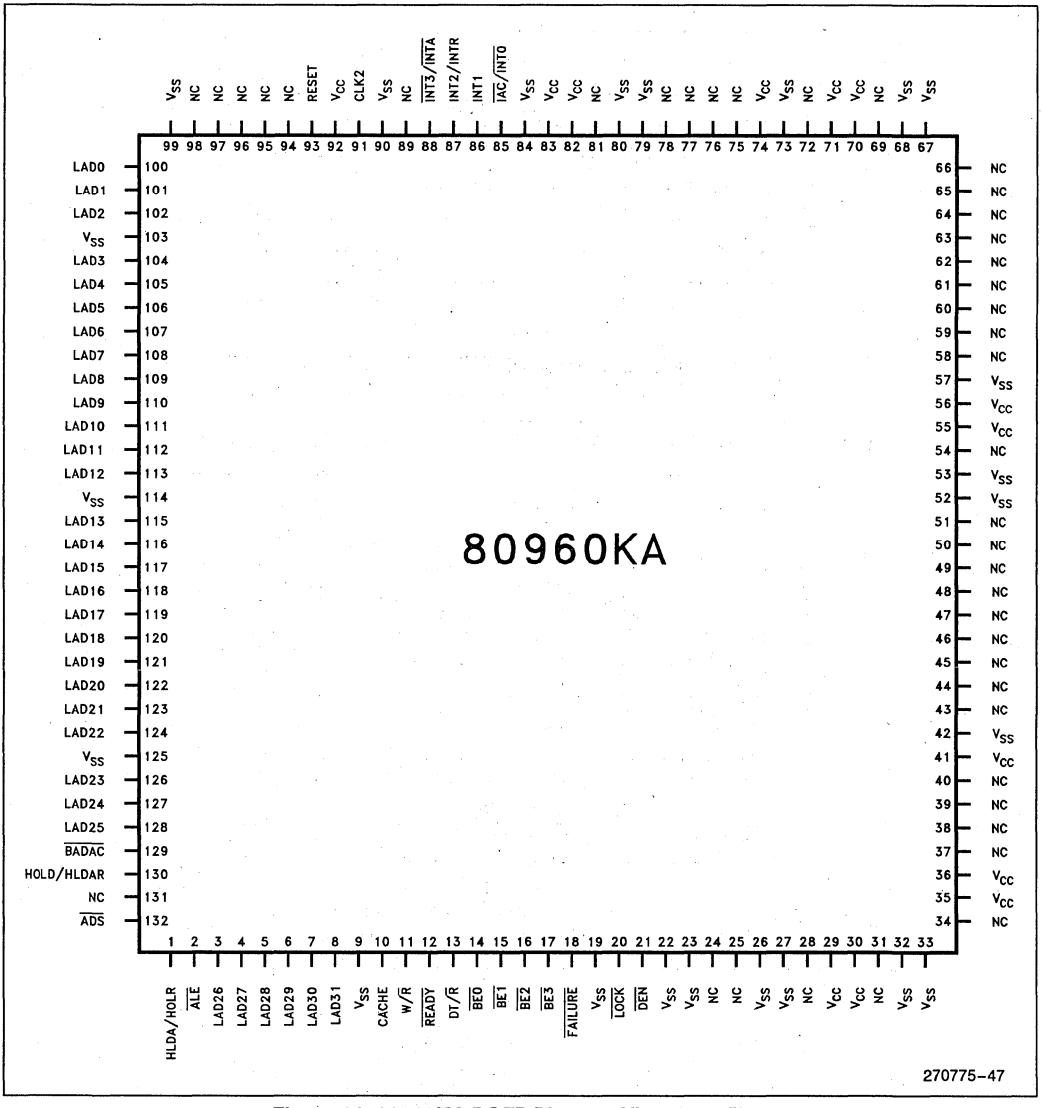


Figure 32. 80960KA PQFP Pinout—View from Top

Table 8. 80960KA Plastic Package Pinout—In Pin Order

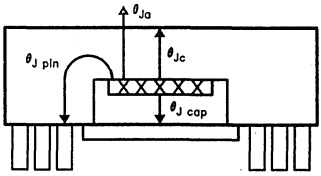
Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	HLDA/HOLDR	34	N.C.	67	V_{SS}	100	LAD0
2	$\overline{ALE}$	35	V_{CC}	68	V_{SS}	101	LAD1
3	LAD26	36	V_{CC}	69	N.C.	102	LAD2
4	LAD27	37	N.C.	70	V_{CC}	103	V_{SS}
5	LAD28	38	N.C.	71	V_{CC}	104	LAD3
6	LAD29	39	N.C.	72	N.C.	105	LAD4
7	LAD30	40	N.C.	73	V_{SS}	106	LAD5
8	LAD31	41	V_{CC}	74	V_{CC}	107	LAD6
9	V_{SS}	42	V_{SS}	75	N.C.	108	LAD7
10	CACHE	43	N.C.	76	N.C.	109	LAD8
11	W/$\overline{R}$	44	N.C.	77	N.C.	110	LAD9
12	$\overline{READY}$	45	N.C.	78	N.C.	111	LAD10
13	DT/$\overline{R}$	46	N.C.	79	V_{SS}	112	LAD11
14	$\overline{BE0}$	47	N.C.	80	V_{SS}	113	LAD12
15	$\overline{BE1}$	48	N.C.	81	N.C.	114	V_{SS}
16	$\overline{BE2}$	49	N.C.	82	V_{CC}	115	LAD13
17	$\overline{BE3}$	50	N.C.	83	V_{CC}	116	LAD14
18	$\overline{FAILURE}$	51	N.C.	84	V_{SS}	117	LAD15
19	V_{SS}	52	V_{SS}	85	$\overline{IAC}$/$\overline{INT0}$	118	LAD16
20	$\overline{LOCK}$	53	V_{SS}	86	INT1	119	LAD17
21	$\overline{DEN}$	54	N.C.	87	INT2/INTR	120	LAD18
22	V_{SS}	55	V_{CC}	88	$\overline{INT3}$/$\overline{INTA}$	121	LAD19
23	V_{SS}	56	V_{CC}	89	N.C.	122	LAD20
24	N.C.	57	V_{SS}	90	V_{SS}	123	LAD21
25	N.C.	58	N.C.	91	CLK2	124	LAD22
26	V_{SS}	59	N.C.	92	V_{CC}	125	V_{SS}
27	V_{SS}	60	N.C.	93	RESET	126	LAD23
28	N.C.	61	N.C.	94	N.C.	127	LAD24
29	V_{CC}	62	N.C.	95	N.C.	128	LAD25
30	V_{CC}	63	N.C.	96	N.C.	129	$\overline{BADAC}$
31	N.C.	64	N.C.	97	N.C.	130	HOLD/HLDAR
32	V_{SS}	65	N.C.	98	N.C.	131	N.C.
33	V_{SS}	66	N.C.	99	V_{SS}	132	$\overline{ADS}$

Table 9. 80960KA Plastic Package Pinout—In Signal Order

Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin
$\overline{ADS}$	132	LAD22	124	N.C.	49	V_{CC}	41
$\overline{ALE}$	2	LAD23	126	N.C.	50	V_{CC}	55
$\overline{BADAC}$	129	LAD24	127	N.C.	51	V_{CC}	56
$\overline{BE0}$	14	LAD25	128	N.C.	54	V_{CC}	70
$\overline{BE1}$	15	LAD26	3	N.C.	58	V_{CC}	71
$\overline{BE2}$	16	LAD27	4	N.C.	59	V_{CC}	74
$\overline{BE3}$	17	LAD28	5	N.C.	60	V_{CC}	82
CACHE	10	LAD29	6	N.C.	61	V_{CC}	83
CLK2	91	LAD3	104	N.C.	62	V_{CC}	92
$\overline{DEN}$	21	LAD30	7	N.C.	63	V_{SS}	9
DT/$\overline{R}$	13	LAD31	8	N.C.	64	V_{SS}	19
$\overline{FAILURE}$	18	LAD4	105	N.C.	65	V_{SS}	22
HLDA/HOLDR	1	LAD5	106	N.C.	66	V_{SS}	23
HOLD/HLDAR	130	LAD6	107	N.C.	69	V_{SS}	26
$\overline{IAC}$/$\overline{INT0}$	85	LAD7	108	N.C.	72	V_{SS}	27
INT1	86	LAD8	109	N.C.	75	V_{SS}	32
INT2/INTR	87	LAD9	110	N.C.	76	V_{SS}	33
$\overline{INT3}$/$\overline{INTA}$	88	$\overline{LOCK}$	20	N.C.	77	V_{SS}	42
LAD0	100	N.C.	24	N.C.	78	V_{SS}	52
LAD1	101	N.C.	25	N.C.	81	V_{SS}	53
LAD10	111	N.C.	28	N.C.	89	V_{SS}	57
LAD11	112	N.C.	31	N.C.	94	V_{SS}	67
LAD12	113	N.C.	34	N.C.	95	V_{SS}	68
LAD13	115	N.C.	37	N.C.	96	V_{SS}	73
LAD14	116	N.C.	38	N.C.	97	V_{SS}	79
LAD15	117	N.C.	39	N.C.	98	V_{SS}	80
LAD16	118	N.C.	40	N.C.	131	V_{SS}	84
LAD17	119	N.C.	43	$\overline{READY}$	12	V_{SS}	90
LAD18	120	N.C.	44	RESET	93	V_{SS}	99
LAD19	121	N.C.	45	V_{CC}	29	V_{SS}	103
LAD2	102	N.C.	46	V_{CC}	30	V_{SS}	114
LAD20	122	N.C.	47	V_{CC}	35	V_{SS}	125
LAD21	123	N.C.	48	V_{CC}	36	W/$\overline{R}$	11

Table 10. 80960KA PGA Package Thermal Characteristics

Thermal Resistance—°C/Watt							
Parameter	**Airflow—ft./min (m/sec)**						
	0 (0)	**50 (0.25)**	**100 (0.50)**	**200 (1.01)**	**400 (2.03)**	**600 (3.04)**	**800 (4.06)**
θ Junction-to-Case (Case Measured as shown in Figure 26)	2	2	2	2	2	2	2
θ Case-to-Ambient (No Heatsink)	19	18	17	15	12	10	9
θ Case-to-Ambient (with Omnidirectional Heatsink)	16	15	14	12	9	7	6
θ Case-to-Ambient (with Unidirectional) Heatsink)	15	14	13	11	8	6	5



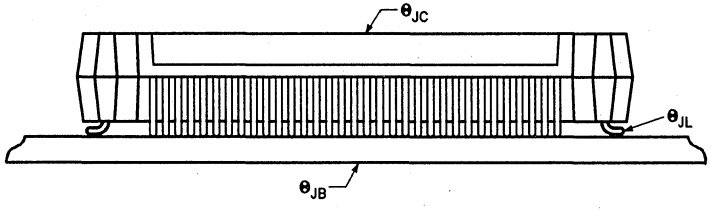
270775-38

NOTES:
1. This table applies to 80960KA PGA plugged into socket or soldered directly into board.
2. $\theta_{JA} = \theta_{JC} + \theta_{CA}$.
3. $\theta_{J\text{-}CAP}$ = 4°C/w (approx.)
   $\theta_{J\text{-}PIN}$ = 4°C/w (inner pins) (approx.)
   $\theta_{J\text{-}PIN}$ = 8°C/w (outer pins) (approx.)

Table 11. 80960KA PQFP Package Thermal Characteristics

PQFP Thermal Resistance—°C/Watt							
Parameter	**Airflow—ft./min (m/sec)**						
	0 (0)	**50 (0.25)**	**100 (0.50)**	**200 (1.01)**	**400 (2.03)**	**600 (3.04)**	**800 (4.06)**
θ Junction-to-Case (Case Measured as shown in Figure 26)	9	9	9	9	9	9	9
θ Case-to-Ambient (No Heatsink)	22	19	18	16	11	9	8

NOTES:
1. This table applies to 80960KA PQFP soldered directly into board.
2. $\theta_{JA} = \theta_{JC} + \theta_{CA}$.
3. θ_{JL} = 18°C/Watt
   θ_{JB} = 18°C/Watt



270775-39

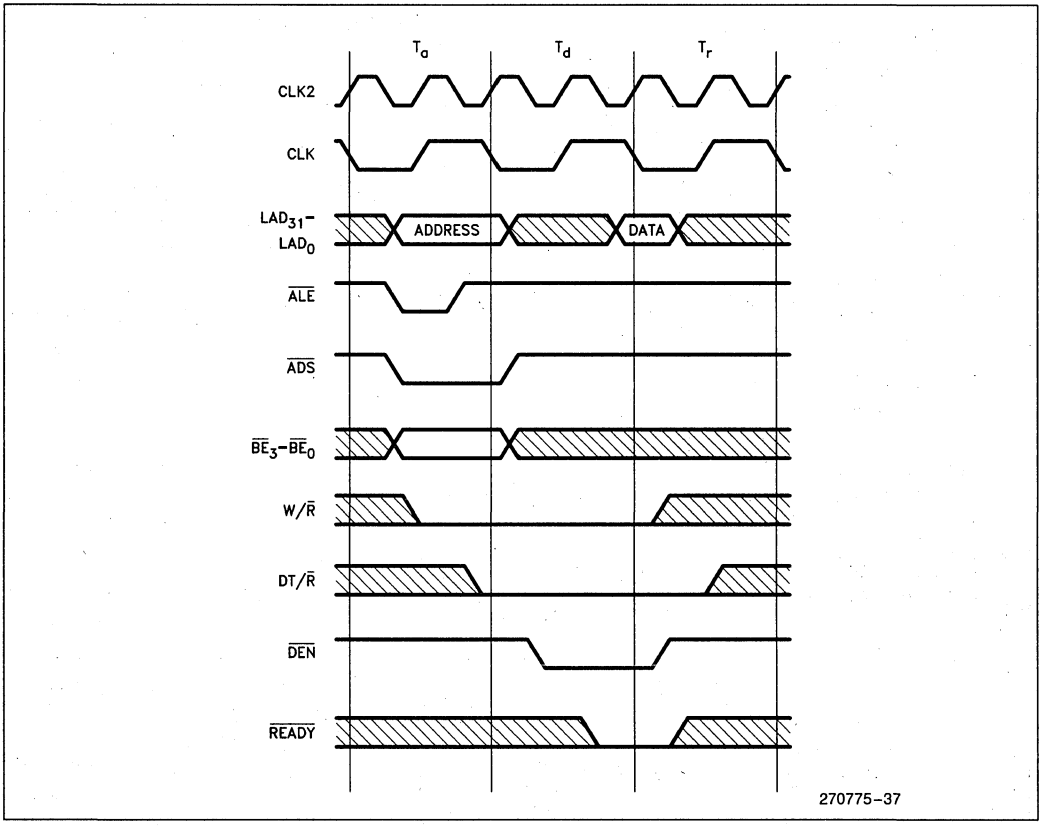


Figure 33. Read Transaction

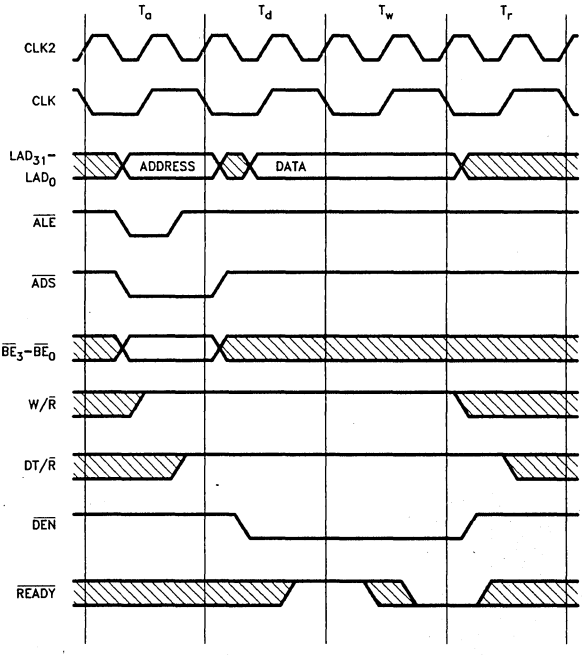


Figure 34. Write Transaction with One Wait State

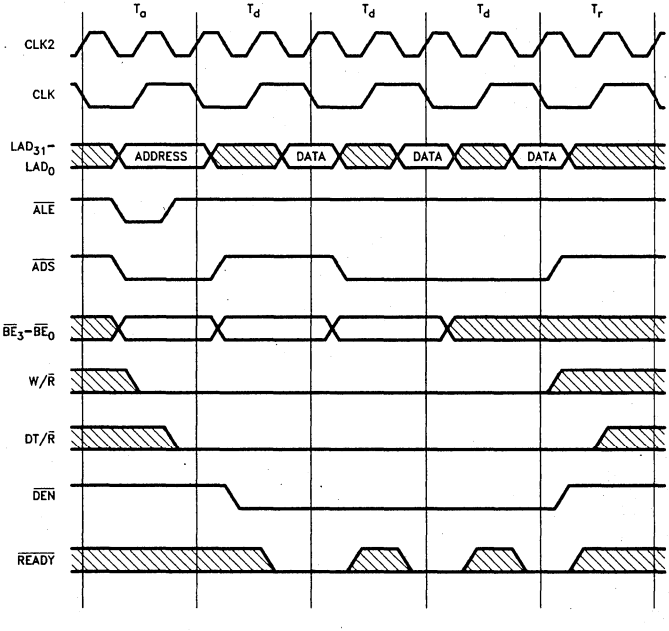


Figure 35. Burst Read Transaction

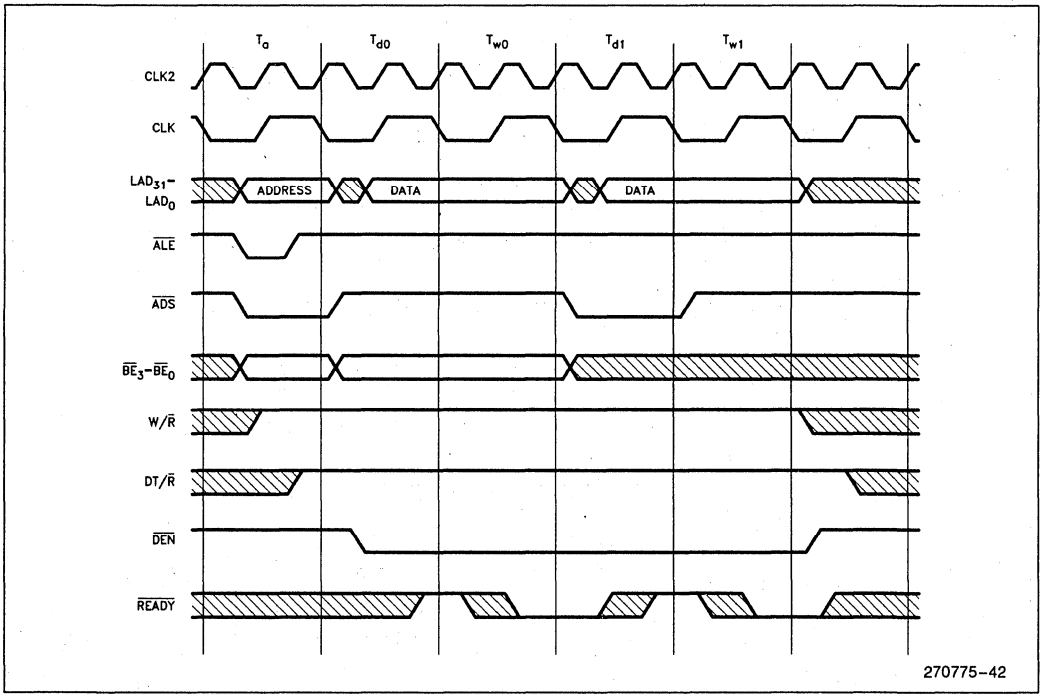
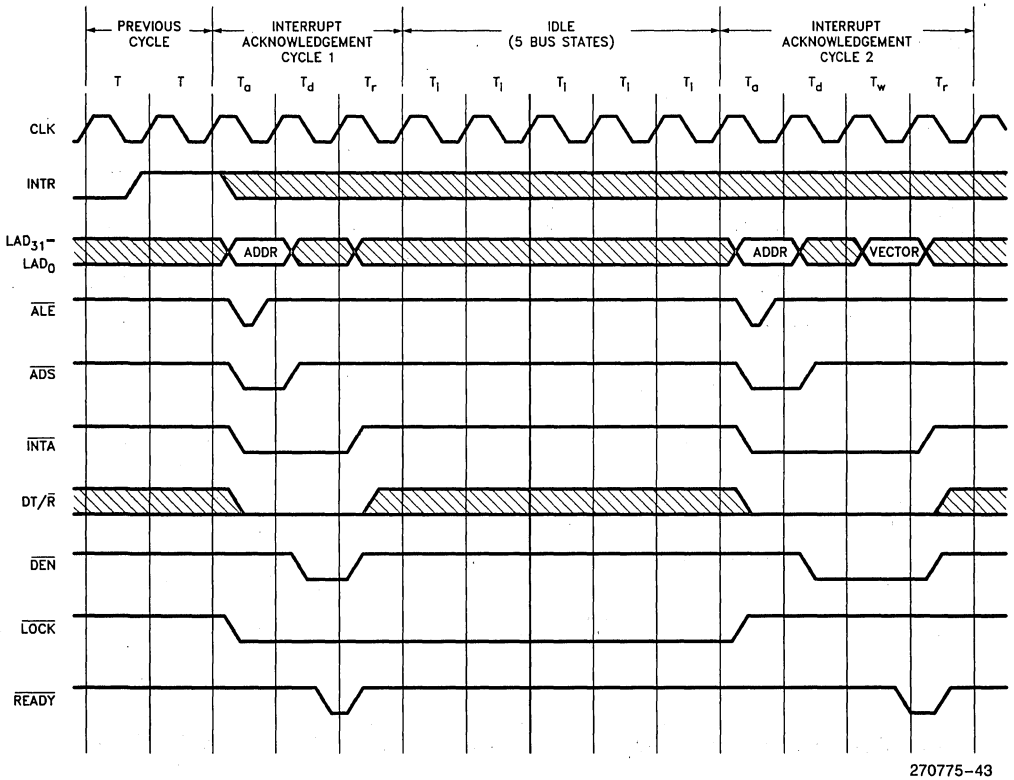


Figure 36. Burst Write Transaction with One Wait State



NOTE:
INTR can go low no sooner than 5 ns (input hold time) following the beginning of interrupt acknowledgement cycle 1. For a second interrupt to be acknowledged, INTR must be low for at least three cycles before it can be reasserted.

Figure 37. Interrupt Acknowledge Transaction

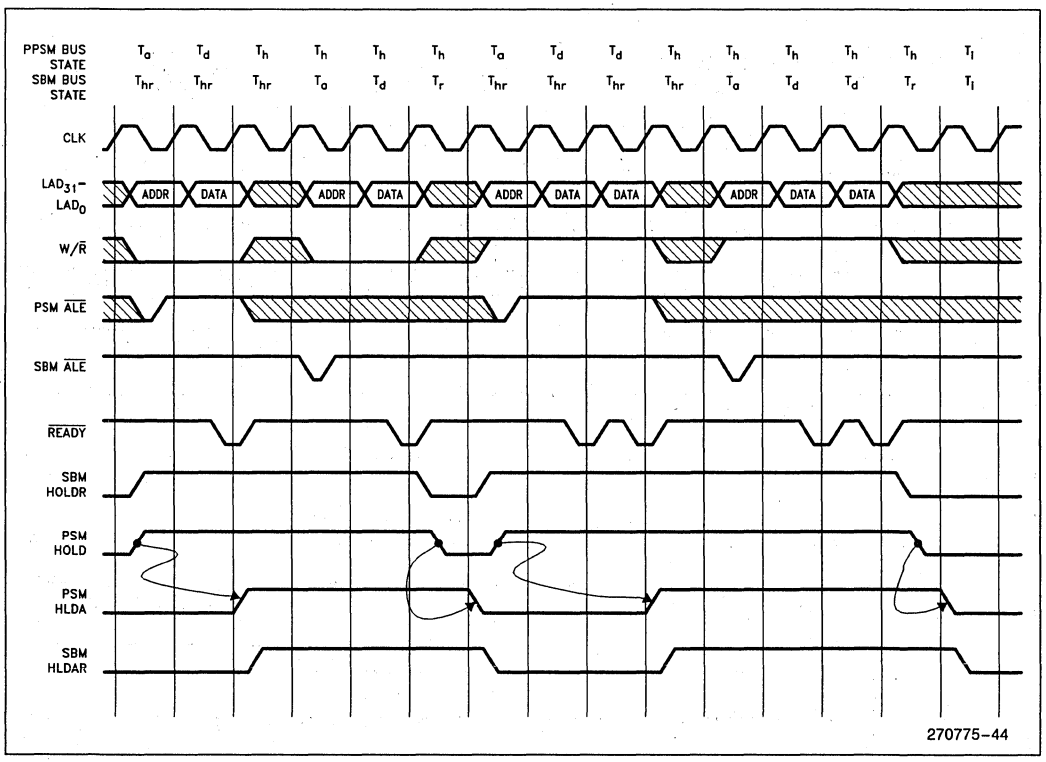


Figure 38. Bus Exchange Transaction (PBM = Primary Bus Master, SBM = Secondary Bus Master)

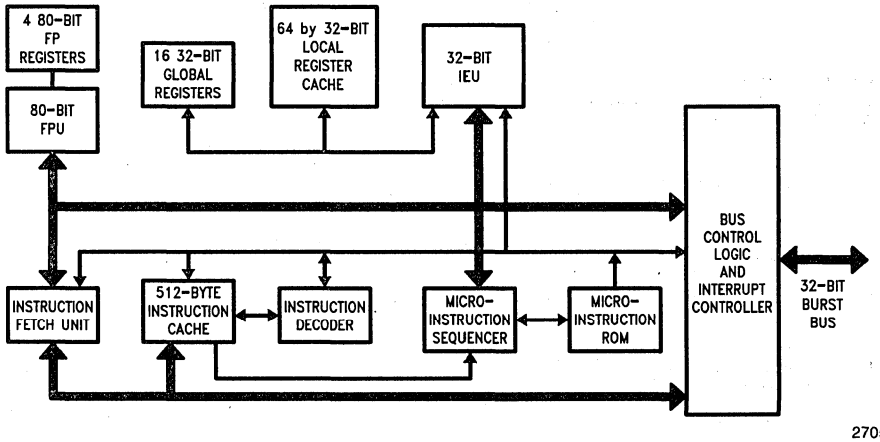
80960KB
EMBEDDED 32-BIT PROCESSOR WITH INTEGRATED FLOATING-POINT UNIT

- **High-Performance Embedded Architecture**
  - — 25 MIPS Burst Execution at 25 MHz
  - — 9.4 MIPS* Sustained Execution at 25 MHz
- **On-Chip Floating-Point Unit**
  - — Supports IEEE 754 Standard
  - — Four 80-Bit Registers
  - — 5.2 Million Whetstones/s at 25 MHz
- **512-Byte On-Chip Instruction Cache**
  - — Direct Mapped
  - — Parallel Load/Decode for Uncached Instructions
- **4 Gigabyte, Linear Address Space**
- **132-Lead PGA and PQFP Packages**
- **Multiple Register Sets**
  - — Sixteen Global 32-Bit Registers
  - — Sixteen Local 32-Bit Registers
  - — Four Local Register Sets Stored On-Chip
  - — Register Scoreboarding
- **Built-In Interrupt Controller**
  - — 32 Priority Levels 256 Vectors
  - — 3.4 μs Latency
- **Easy to Use, High Bandwidth 32-Bit Bus**
  - — 66.7 Mbytes/s Burst
  - — Up to 16-Bytes Transferred per Burst
- **Uses 85C960 Bus Controller**
- **Supported by 27960KX Burst EPROMs**

3

The 80960KB is the first member of Intel's new 32-bit processor family, the i960 series, which is designed especially for embedded applications. It is based on the family's high performance, common core architecture, and includes a 512-byte instruction cache, a built-in interrupt controller, and an integrated floating-point unit. The 80960KB has a large register set, multiple parallel execution units and a high-bandwidth, burst bus. Using advanced RISC technology, this high performance processor is capable of execution rates in excess of 9.4 million instructions per second.* The 80960KB is well-suited for a wide range of embedded applications, including laser printers, image processing, industrial control, robotics and telecommunications.

*Relative to Digital Equipment Corporation's VAX-11/780** at 1 MIPS



270565-1

Figure 1. The 80960KB's Highly Parallel Microarchitecture

**VAX-11™ is a trademark of Digital Equipment Corporation.

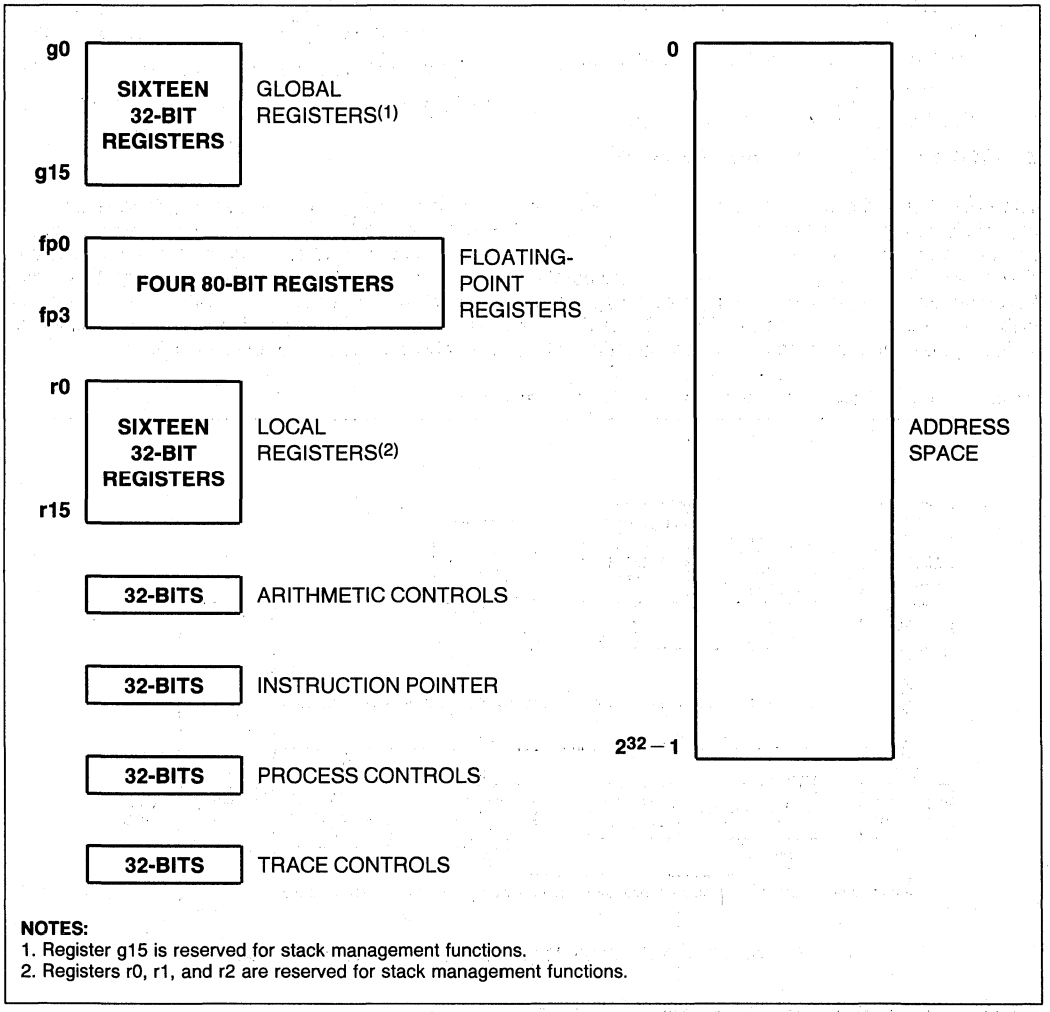
September 1991
Order Number: 270565-006

THE 960 SERIES

The 80960KB is the first member of a new family of 32-bit microprocessors from Intel known as the 960 Series. This series was especially designed to serve the needs of embedded applications. The embedded market includes applications as diverse as industrial automation, avionics, image processing, graphics, robotics, telecommunications and automobiles. These types of applications require high integration, low power consumption, quick interrupt response times and high performance. Since time to market is critical, embedded microprocessors need to be easy to use in both hardware and software designs.

All members of the 80960 series share a common core architecture which utilizes RISC technology so that, except for special functions, the family members are object code compatible. Each new processor in the series will add its own special set of functions to the core to satisfy the needs of a specific application or range of applications in the embedded market. For example, future processors may include a DMA controller, a timer or an A/D converter.

The 80960KB includes an integrated floating-point unit. Intel also offers a pin-compatible version, called the 80960KA, without an FPU, and a military-grade version, the 80960MC, with support for memory management, mutitasking, multiprocessing and fault tolerance.



NOTES:
1. Register g15 is reserved for stack management functions.
2. Registers r0, r1, and r2 are reserved for stack management functions.

Figure 2. Register Set

KEY PERFORMANCE FEATURES

The 80960KB's architecture is based on the most recent advances in RISC technology and is grounded in Intel's long experience in designing embedded controllers. Many features contribute to the 80960KB's exceptional performance:

1. Large Register Set. Having a large number of registers reduces the number of times that a processor needs to access memory. Modern compilers can take advantage of this feature to optimize execution speed. For maximum flexibility, the 80960KB provides 32 32-bit registers and four 80-bit floating-point registers. (See Figure 2.)

2. Fast Instruction Execution. Simple functions make up the bulk of instructions in most programs, so that execution speed can be greatly improved by ensuring that these core instructions execute in as short a time as possible. The most-frequently executed instructions such as register-register moves, add/subtract, logical operations, and shifts execute in one to two cycles (Table 1 contains a list of instructions.)

3. Load/Store Architecture. Like other processors based on RISC technology, the 80960KB has a Load/Store architecture, only the LOAD and STORE instructions reference memory; all other instructions operate on registers. This type of architecture simplifies instruction decoding and is used in combination with other techniques to increase parallelism.

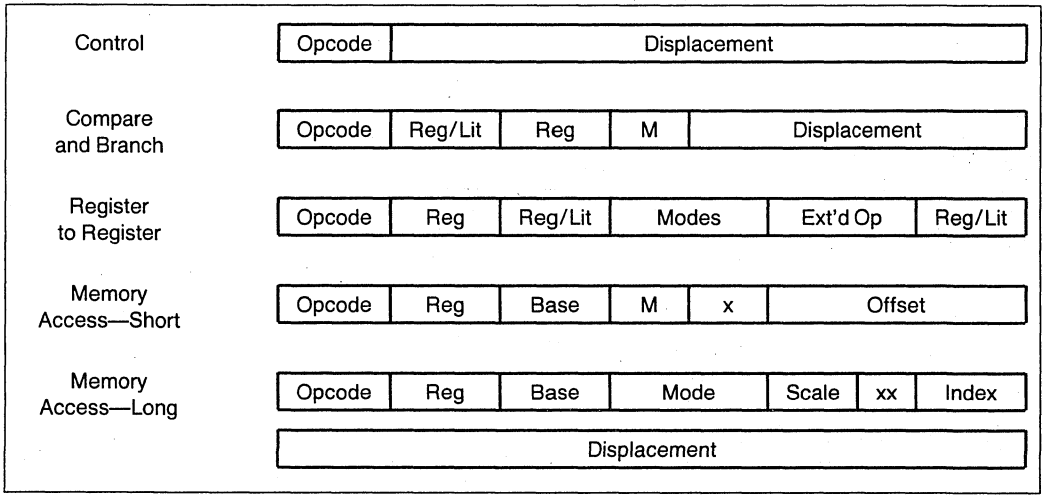


Figure 3. Instruction Formats

Table 1. 80960KB Instruction Set

Data Movement	Arithmetic	Logical	Bit and Bit Field
Load Store Move Load Address	Add Subtract Multiply Divide Remainder Modulo Shift Extended Multiply Extended Divide	And Not And And Not Or Exclusive Or Not Or Or Not Nor Exclusive Nor Not Nand Rotate	Set Bit Clear Bit Not Bit Check Bit Alter Bit Scan for Bit Scan over Bit Extract Modify
Comparison	**Branch**	**Call/Return**	**Fault**
Compare Conditional Compare Compare and Increment Compare and Decrement	Unconditional Branch Conditional Branch Compare and Branch	Call Call Extended Call System Return Branch and Link	Conditional Fault Synchronize Faults
Debug	**Miscellaneous**	**Decimal**	
Modify Trace Controls Mark Force Mark	Atomic Add Atomic Modify Flush Local Registers Modify Arithmetic Controls Modify Process Controls Scan Byte for Equal Test Condition Code	Move Add with Carry Subtract with Carry	
Conversion	**Floating-Point**	**Synchronous**	
Convert Real to Integer Convert Integer to Real	Move Real Add Subtract Multiply Divide Remainder Scale Round Square Root Sine Cosine Tangent Arctangent Log Log Binary Log Natural Exponent Classify Copy Real Extended Compare	Synchronous Load Synchronous Move	

4. Simple Instruction Formats. All instructions in the 80960KB are 32-bits long and must be aligned on word boundaries. This alignment makes it possible to eliminate the instruction-alignment stage in the pipeline. To simplify the instruction decoder further, there are only five instruction formats and each instruction uses only one format. (See Figure 3.)

5. Overlapped Instruction Execution. A load operation allows execution of subsequent instructions to continue before the data has been returned from memory, so that these instructions can overlap the load. The 80960KB manages this process transparently to software through the use of a register scoreboard. Conditional instructions also make use of a scoreboard so that subsequent unrelated instructions can be executed while the conditional instruction is pending.

6. Integer Execution Optimization. When the result of an operation is used as an operand in a subsequent calculation, the value is sent immediately to its destination register. Yet at the same time, the value is put back on a bypass path to the ALU, thereby saving the time that otherwise would be required to retrieve the value for the next operation.

7. Bandwidth Optimizations. The 80960KB gets optimal use of its memory bus bandwidth because the bus is tuned for use with the cache: the line size of the instruction cache matches the maximum burst size for instruction fetches. The 80960KB automatically fetches four words in a burst and stores them directly in the cache. Due to the size of the cache and the fact that it is continually filled in anticipation of needed instructions in the program flow, the 80960KB is exceptionally insensitive to memory wait states. In fact, each wait state causes only a 7% degradation in system perfomance. The benefit is that the 80960KB will deliver outstanding performance even with a low cost memory system.

8. Cache Bypass. If there is a cache miss, the processor fetches the needed instruction, then sends it on to the instruction decoder at the same time it updates the cache. Thus, no extra time is taken to load and read the cache.

Memory Space and Addressing Modes

The 80960KB offers a linear programming environment so that all programs running on the processor are contained in a single address space. The maximum size of the address space is 4 Gigabytes (2^{32} bytes).

For ease of use, the 80960KB has a small number of addressing modes, but includes all those necessary to ensure efficient compiler implementations of high-level languages such as C, Fortran and Ada. Table 2 lists the memory addressing modes.

Data Types

The 80960KB recognizes the following data types:

Numeric:
- 8-, 16-, 32- and 64-bit ordinals
- 8-, 16, 32- and 64-bit integers
- 32-, 64- and 80-bit real numbers

Non-Numeric:
- Bit
- Bit Field
- Triple-Word (96 bits)
- Quad-Word (128 bits)

Large Register Set

The programming environment of the 80960KB includes a large number of registers. In fact, 36 registers are available at any time. The availability of this many registers greatly reduces the number of memory accesses required to execute most programs, which leads to greater instruction processing speed.

There are two types of general-purpose registers: local and global. The 20 global registers consist of sixteen 32-bit registers (G0 through G15) and four 80-bit registers (FP0 through FP3). These registers perform the same function as the general-purpose registers provided in other popular microprocessors. The term global refers to the fact that these registers retain their contents across procedure calls.

The local registers, on the other hand, are procedure specific. For each procedure call, the 80960KB allocates 16 local registers (R0 through R15). Each local register is 32 bits wide. Any register can also be used for single or double-precision floating-point operations; the 80-bit floating-point registers are provided for extended precision.

Multiple Register Sets

To further increase the efficiency of the register set, multiple sets of local registers are stored on-chip. This cache holds up to four local register frames, which means that up to three procedure calls can be made without having to access the procedure stack resident in memory.

Although programs may have procedure calls nested many calls deep, a program typically oscillates back and forth between only two or three levels. As

Table 2. Memory Addressing Modes

- 12-Bit Offset
- 32-Bit Offset
- Register-Indirect
- Register + 12-Bit Offset
- Register + 32-Bit Offset
- Register + (Index-Register × Scale-Factor)
- Register × Scale Factor + 32-Bit Displacement
- Register + (Index-Register × Scale-Factor) + 32-Bit Displacement

Scale-Factor is 1, 2, 4, 8 or 16

a result, with four stack frames in the cache, the probability of there being a free frame on the cache when a call is made is very high. In fact, runs of representative C-language programs show that 80% of the calls are handled without needing to access memory.

If there are four or more active procedures and a new procedure is called, the processor moves the oldest set of local registers in the register cache to a procedure stack in memory to make room for a new set of registers. Global register G15 is used by the processor as the frame pointer (FP) for the procedure stack.

Note that the global and floating-point registers are not exchanged on a procedure call, but retain their contents, making them available to all procedures for fast parameter passing. An illustration of the register cache is shown in Figure 4.

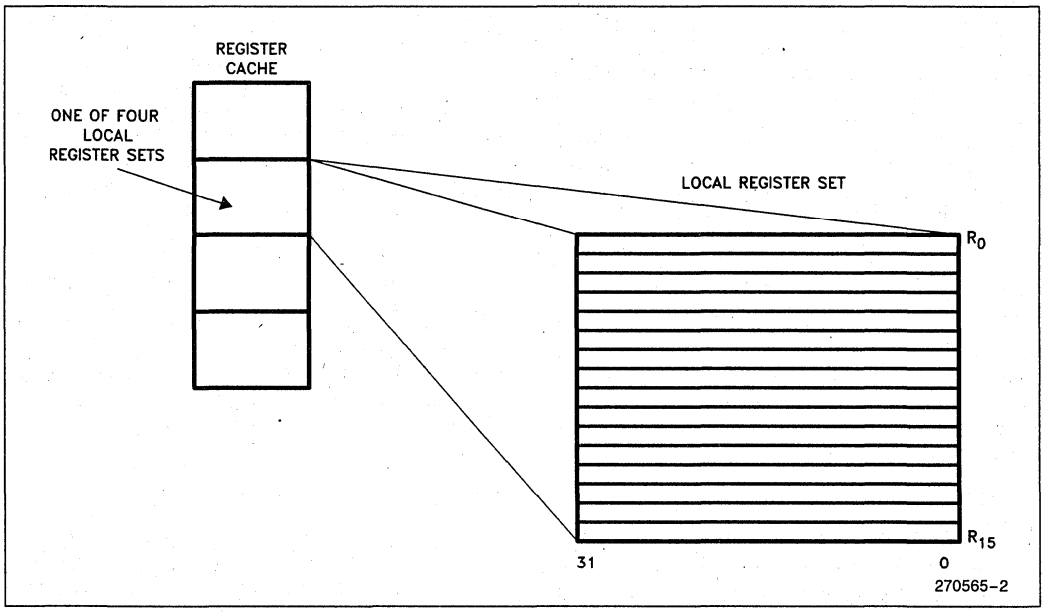


Figure 4. Multiple Register Sets Are Stored On-Chip

Instruction Cache

To further reduce memory accesses, the 80960KB includes a 512-byte on-chip instruction cache. The instruction cache is based on the concept of locality of reference; that is, most programs are not usually executed in a steady stream but consist of many branches and loops that lead to jumping back and forth within the same small section of code. Thus, by maintaining a block of instructions in a cache, the number of memory references required to read instructions into the processor can be greatly reduced.

To load the instruction cache, instructions are fetched in 16-byte blocks, so that up to four instructions can be fetched at one time. An efficient prefetch algorithm increases the probability that an instruction will already be in the cache when it is needed.

Code for small loops will often fit entirely within the cache, leading to a great increase in processing speed since further memory references might not be necessary until the program exits the loop. Similarly, when calling short procedures, the code for the calling procedure is likely to remain in the cache, so it will be there on the procedure's return.

Register Scoreboarding

The instruction decoder has been optimized in several ways. One of these optimizations is the ability to do instruction overlapping by means of register scoreboarding.

Register scoreboarding occurs when a LOAD instruction is executed to move a variable from memory into a register. When the instruction is initiated, a scoreboard bit on the target register is set. When the register is actually loaded, the bit is reset. In between, any reference to the register contents is accompanied by a test of the scoreboard bit to insure that the load has completed before processing continues. Since the processor does not have to wait for the LOAD to be completed, it can go on to execute additional instructions placed in between the LOAD instruction and the instruction that uses the register contents, as shown in the following example:

```
LOAD R4, address 1
LOAD R5, address 2
Unrelated instruction
Unrelated instruction
ADD R4, R5, R6
```

In essence, the two unrelated instructions between the LOAD and ADD instructions are executed for free (i.e., take no apparent time to execute) because they are executed while the register is being loaded. Up to three LOAD instructions can be pending at one time with three corresponding scoreboard bits set. By exploiting this feature, system programmers and compilers have a useful tool for optimizing execution speed.

Floating-Point Arithmetic

In the 80960KB, floating-point arithmetic has been made an integral part of the architecture. Having the floating-point unit integrated on-chip provides two advantages. First, it improves the performance of the chip for floating-point applications, since no additional bus overhead is associated with floating-point calculations, thereby leaving more time for other bus operations such as I/O. Second, the cost of using floating-point operations is reduced because a separate coprocessor chip is not required.

The 80960KB floating-point (real number) data types include single-precision (32-bit), double-precision (64-bit), and extended precision (80-bit) floating-point numbers. Any register may be used to execute floating-point operations.

The processor provides hardware support for both mandatory and recommended portions of IEEE Standard 754 for floating-point arithmetic, including all arithmetic, exponential, logarithmic, and other transcendental functions. Table 3 shows execution times for some representative instructions.

Table 3. Sample Floating-Point Execution Times (μs) at 25 MHz

	32-Bit	64-Bit
Add	0.4	0.5
Subtract	0.4	0.5
Multiply	0.7	1.3
Divide	1.3	2.9
Square Root	3.7	3.9
Arctangent	10.1	13.1
Exponent	11.3	12.5
Sine	15.2	16.6
Cosine	15.2	16.6

High Bandwidth Local Bus

An 80960KB CPU resides on a high-bandwidth address/data bus known as the local bus (L-Bus). The L-Bus provides a direct communication path between the processor and the memory and I/O subsystem interfaces. The processor uses the local bus to fetch instructions, manipulate memory, and respond to interrupts. Its features include:

- 32-bit multiplexed address/data path
- Four-word burst capability, which allows transfers from 1 to 16 bytes at a time
- High bandwidth reads and writes at 66.7 Mbytes per second
- Special signal to indicate whether a memory transaction can be cached

Figure 5 identifies the groups of signals which constitute the L-Bus. Table 4 lists the function of the L-Bus and other processor-support signals, such as the interrupt lines.

Interrupt Handling

The 80960KB can be interrupted in one of two ways: by the activation of one of four interrupt pins or by sending a message on the processor's data bus.

The 80960KB is unusual in that it automatically handles interrupts on a priority basis and tracks pending interrupts through its on-chip interrupt controller. Two of the interrupt pins can be configured to provide 8259A handshaking for expansion beyond four interrupt lines.

Debug Features

The 80960KB has built-in debug capabilities. There are two types of breakpoints and six different trace modes. The debug features are controlled by two internal 32-bit registers, the Process-Controls Word and the Trace-Controls Word. By setting bits in these control words, a software debug monitor can closely control how the processor responds during program execution.

The 80960KB has both hardware and software breakpoints. It provides two hardware breakpoint registers on-chip which can be set by a special command to any value. When the instruction pointer matches the value in one of the breakpoint registers, the breakpoint will fire, and a breakpoint handling routine is called automatically.

The 80960KB also provides software breakpoints through the use of two instructions, MARK and FMARK. These instructions can be placed at any point in a program and will cause the processor to halt execution at that point and call the breakpoint handling routine. The breakpoint mechanism is easy to use and provides a powerful debugging tool.

Tracing is available for instructions (single-step execution), calls and returns, and branching. Each different type of trace may be enabled separately by a special debug instruction. In each case, the 80960KB executes the instruction first and then calls a trace handling routine (usually part of a software debug monitor). Further program execution is halted until the trace routine is completed. When the trace event handling routine is completed, instruction execution resumes at the next instruction. The

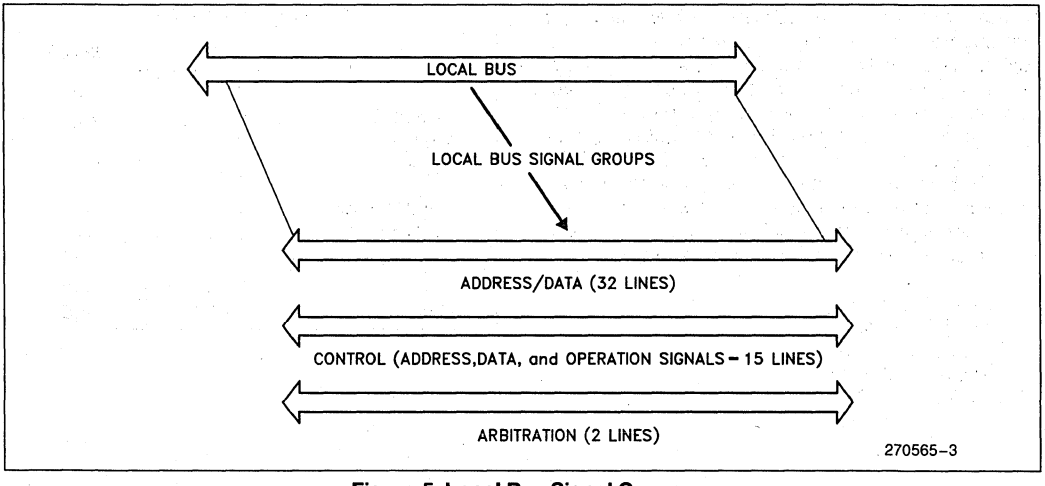


Figure 5. Local Bus Signal Groups

80960KB's tracing mechanisms, which are implemented completely in hardware, greatly simplify the task of testing and debugging software.

FAULT DETECTION

The 80960KB has an automatic mechanism to handle faults. There are ten fault types including trace, arithmetic, and floating-point faults. When the processor detects a fault, it automatically calls the appropriate fault handling routine and saves the current instruction pointer and necessary state information to make efficient recovery possible. The processor posts diagnostic information on the type of fault to a Fault Record. Like interrupt handling routines, fault handling routines are usually written to meet the needs of a specific application and are often included as part of the operating system or kernel.

For each of the ten fault types, there are numerous subtypes that provide specific information about a fault. For example, a floating-point fault may have its subtype set to an Overflow or Zero-Divide fault. The fault handler can use this specific information to respond correctly to the fault.

BUILT-IN TESTABILITY

Upon reset, the 80960KB automatically conducts an extensive internal test (self-test) of its major blocks of logic. Then, before executing its first instruction, it does a zero check sum on the first eight words in memory to ensure that the system has been loaded correctly. If a problem is discovered at any point during the self-test, the 80960KB will assert its FAILURE pin and will not begin program execution. The self-test takes approximately 47,000 cycles to complete.

System manufacturers can use the 80960KB's self-test feature during incoming parts inspection. No special diagnostic programs need to be written, and the test is both thorough and fast. The self-test capability helps ensure that defective parts will be discovered before systems are shipped, and once in the field, the self-test makes it easier to distinguish between problems caused by processor failure and problems resulting from other causes.

CHMOS

The 80960KB is fabricated using Intel's CHMOS IV (Complementary High Speed Metal Oxide Semiconductor) process. This advanced technology eliminates the frequency and reliability limitations of older CMOS processes and opens a new era in microprocessor performance. It combines the high performance capabilities of Intel's industry-leading HMOS technology with the high density and low power characteristics of CMOS. The 80960KB is available at 10, 16, 20 and 25 MHz.

Table 4a. 80960KB Pin Description: L-Bus Signals

Symbol	Type	Name and Function
CLK2	I	**SYSTEM CLOCK** provides the fundamental timing for 80960KB systems. It is divided by two inside the 80960KB to generate the internal processor clock.
LAD_{31} $-LAD_0$	I/O T.S.	**LOCAL ADDRESS/DATA BUS** carries 32-bit physical addresses and data to and from memory. During an address (T_a) cycle, bits 2–31 contain a physical word address (bits 0–1 indicate SIZE; see below). During a data (T_d) cycle, bits 0–31 contain read or write data. The LAD lines are active HIGH and float to a high impedance state when not active. **SIZE,** which is comprised of bits 0–1 of the LAD lines during a T_a cycle, specifies the size of a burst transfer in words. $\underline{LAD_1}$ $\underline{LAD_0}$ 0 0 1 Word 0 1 2 Words 1 0 3 Words 1 1 4 Words
$\overline{ALE}$	O T.S.	**ADDRESS-LATCH ENABLE** indicates the transfer of a physical address. ALE is asserted during a T_a cycle and deasserted before the beginning of the T_d state. It is active LOW and floats to a high impedance state during a hold cycle (T_h or T_{hr}).

I/O = Input/Output, O = Output, I = Input, O.D. = Open-Drain, T.S. = tri-state

Table 4a. 80960KB Pin Description: L-Bus Signals (Continued)

Symbol	Type	Name and Function
$\overline{ADS}$	O O.D.	**ADDRESS/DATA STATUS** indicates an address state. ADS is asserted every T_a state and deasserted during the the following T_d state. For a burst transaction, $\overline{ADS}$ is asserted again every T_d state where $\overline{READY}$ was asserted in the previous cycle.
W/$\overline{R}$	O O.D.	**WRITE/READ** specifies, during a T_a cycle, whether the operation is a write or read. It is latched on-chip and remains valid during T_d cycles.
DT/$\overline{R}$	O O.D.	**DATA TRANSMIT/RECEIVE** indicates the direction of data transfer to and from the L-Bus. It is low during T_a and T_d cycles for a read or interrupt acknowledgement; it is high during T_a and T_d cycles for a write. $\overline{DT/R}$ never changes state when $\overline{DEN}$ is asserted (see Timing Diagrams).
$\overline{DEN}$	O O.D.	**DATA ENABLE** is asserted during T_d cycles and indicates transfer of data on the LAD bus lines.
$\overline{READY}$	I	**READY** indicates that data on LAD lines can be sampled or removed. If $\overline{READY}$ is not asserted during a T_d cycle, the T_d cycle is extended to the next cycle by inserting a wait state (T_W), and $\overline{ADS}$ is not asserted in the next cycle.
$\overline{LOCK}$	I/O O.D.	**BUS LOCK** prevents other bus masters from gaining control of the L-Bus following the current cycle (if they would assert $\overline{LOCK}$ to do so). $\overline{LOCK}$ is used by the processor or any bus agent when it performs indivisible Read/Modify/Write (RMW) operations. Do not leave $\overline{LOCK}$ unconnected. It must be pulled high for the processor to function properly. For a read that is designated as a RMW-read, $\overline{LOCK}$ is examined. if asserted, the processor waits until it is not asserted; if not asserted, the processor asserts $\overline{LOCK}$ during the T_a cycle and leaves it asserted. A write that is designated as an RMW-write deasserts $\overline{LOCK}$ in the T_a cycle. During the time $\overline{LOCK}$ is asserted, a bus agent can perform a normal read or write but no RMW operations. $\overline{LOCK}$ is also held asserted during an interrupt-acknowledge transaction.
$\overline{BE_3}$–$\overline{BE_0}$	O O.D.	**BYTE ENABLE LINES** specify which data bytes (up to four) on the bus take part in the current bus cycle. $\overline{BE_3}$ corresponds to LAD_{31}–LAD_{24} and $\overline{BE_0}$ corresponds to LAD_7–LAD_0. The byte enables are provided in advance of data. The byte enables asserted during T_a specify the bytes of the first data word. The byte enables asserted during T_d specify the bytes of the next data word (if any), that is, the word to be transmitted following the next assertion of $\overline{READY}$. The byte enables during the T_d cycles preceding the last assertion of $\overline{READY}$ are undefined. The byte enables are latched on-chip and remain constant from one T_d cycle to the next when $\overline{READY}$ is not asserted. For reads, the byte enables specify the byte(s) that the processor will actually use. L-Bus agents are required to assert only adjacent byte enables (e.g., asserting just $\overline{BE_0}$ and $\overline{BE_2}$ is not permitted), and are required to assert at least one byte enable. To produce address bits A_0 and A_1 externally, they can be decoded from the byte enables.

I/O = Input/Output, O = Output, I = Input, O.D. = Open-Drain, T.S. = tri-state

Table 4a. 80960KB Pin Description: L-Bus Signals (Continued)

Symbol	Type	Name and Function
HOLD/ HLDAR	I	**HOLD:** If the processor is the primary bus master (PBM), the input is interpreted as HOLD, a request from a secondary bus master to acquire the bus. When the processor receives HOLD and grants another master control of the bus, it floats its tri-state bus lines and then asserts HLDA and enters the T_h state. When HOLD is deasserted, the processor will deassert HLDA and go to either the T_i or T_a state. **HOLD ACKNOWLEDGE RECEIVED:** If the processor is a secondary bus master (SBM), the input is HLDAR, which indicates, when HOLDR output is high, that the processor has acquired the bus. Processors and other agents can be told at reset if they are the primary bus master (PBM).
HLDA/ HOLDR	O T.S.	**HOLD ACKNOWLEDGE:** If the processor is a primary bus master, the output is HLDA, which relinquishes control of the bus to another bus master. **HOLD REQUEST:** For secondary bus masters (SBM), the output is HOLDR, which is a request to acquire the bus. The bus is said to be acquired if the agent is a primary bus master and does not have its HLDA output asserted, or if the agent is a secondary bus master and has its HOLD input and HLDA output asserted.
CACHE	O T.S.	**CACHE** indicates if an access is cacheable during a T_a cycle. It is not asserted during any synchronous access, such as a synchronous load or move instruction used for sending an IAC message. The CACHE signal floats to a high impedance state when the processor is idle.

Table 4b. 80960KB Pin Description: Module Support Signals

Symbol	Type	Name and Function
$\overline{\text{BADAC}}$	I	**BAD ACCESS,** if asserted in the cycle following the one in which the last $\overline{\text{READY}}$ of a transaction is asserted, indicates that an unrecoverable error has occurred on the current bus transaction, or that a synchronous load/store instruction has not been acknowledged. **STARTUP:** During system reset, the $\overline{\text{BADAC}}$ signal is interpreted differently. If the signal is high, it indicates that this processor will perform system initialization. If it is low, another processor in the system will perform system initialization instead.
RESET	I	**RESET** clears the internal logic of the processor and causes it to re-initialize. During RESET assertion, the input pins are ignored (except for $\overline{\text{BADAC}}$ and $\overline{\text{IAC}}/\overline{\text{INT}_0}$), the tri-state output pins are placed in a high impedance state, and other output pins are placed in their non-asserted state. RESET must be asserted for at least 41 CLK2 cycles for a predictable RESET. The HIGH to LOW transition of RESET should occur after the rising edge of both CLK2 and the external bus CLK, and before the next rising edge of CLK2.
$\overline{\text{FAILURE}}$	O O.D.	**INITIALIZATION FAILURE** indicates that the processor has failed to initialize correctly. After RESET is deasserted and before the first bus transaction begins, $\overline{\text{FAILURE}}$ is asserted while the processor performs a self-test. If the self-test completes successfully, then $\overline{\text{FAILURE}}$ is deasserted. Next, the processor performs a zero checksum on the first eight words of memory. If it fails, $\overline{\text{FAILURE}}$ is asserted for a second time and remains asserted; if it passes, system initialization continues and $\overline{\text{FAILURE}}$ remains deasserted.
N.C.	N/A	**NOT CONNECTED** indicates pins should not be connected. Never connect any pin marked N.C.

I/O = Input/Output, O = Output, I = Input, O.D. = Open-Drain, T.S. = tri-state

Table 4b. 80960KB Pin Description: Module Support Signals (Continued)

Symbol	Type	Name and Function
$\overline{IAC}$ $\overline{INT0}$	I	**INTERAGENT COMMUNICATION REQUEST/INTERRUPT 0** indicates either that there is a pending IAC message for the processor or an interrupt. The bus interrupt control register determines in which way the signal should be interpreted. To signal an interrupt or IAC request in a synchronous system, this pin (as well as the other interrupt pins) must be enabled by being deasserted for at least one bus cycle and then asserted for at least one additional bus cycle; in an asynchronous system, the pin must remain deasserted for at least two bus cycles and then be asserted for at least two more bus cycles. **LOCAL PROCESSOR NUMBER:** This signal is interpreted differently during system reset. If the signal is at a high voltage level, it indicates that this processor is a primary bus master (Local Processor Number = 0); if it is at a low voltage level, it indicates that this processor is a secondary bus master (Local Processor Number = 1).
INT1	I	**INTERRUPT 1,** like $\overline{INT0}$, provides direct interrupt signaling.
INT2/ INTR	I	**INTERRUPT 2/INTERRUPT REQUEST:** The bus control registers determines how this pin is interpreted. If INT2, it has the same interpretation as the $\overline{INT0}$ and INT1 pins. If INTR, it is used to receive an interrupt request from an external interrupt controller.
$\overline{INT3}$/ $\overline{INTA}$	I/O O.D.	**INTERRUPT 3/INTERRUPT ACKNOWLEDGE:** The bus interrupt control register determines how this pin is interpreted. If $\overline{INT3}$, it has the same interpretation as the $\overline{INT0}$, INT1, and INT2 pins. If $\overline{INTA}$, it is used as an output to control interrupt-acknowledge bus transactions. The $\overline{INTA}$ output is latched on-chip and remains valid during T_d cycles; as an output, it is open-drain.

I/O = Input/Output, O = Output, I = Input, O.D. = Open-Drain, T.S. = tri-state

ELECTRICAL SPECIFICATIONS

Power and Grounding

The 80960KB is implemented in CHMOS IV technology and has modest power requirements. Its high clock frequency and numerous output buffers (address/data, control, error, and arbitration signals) can cause power surges as multiple output buffers drive new signal levels simultaneously. For clean on-chip power distribution at high frequency, 12 V_{CC} and 13 V_{SS} pins separately feed functional units of the 80960KB in the PGA.

Power and ground connections must be made to all power and ground pins of the 80960KB. On the circuit board, all V_{CC} pins must be strapped closely together, preferably on a power plane. Likewise, all V_{SS} pins should be strapped together, preferably on a ground plane. These pins may not be connected together within the chip.

Power Decoupling Recommendations

Liberal decoupling capacitance should be placed near the 80960KB. The processor can cause transient power surges when driving the L-Bus, particularly when it is connected to a large capacitive load.

Low inductance capacitors and interconnects are recommended for best high frequency electrical performance. Inductance can be reduced by shortening the board traces between the processor and decoupling capacitors as much as possible. Capacitors specifically designed for PGA packages are also commercially available and offer the lowest possible inductance.

Connection Recommendations

For reliable operation, always connect unused inputs to an appropriate signal level. In particular, if one or more interrupt lines are not used, they should be pulled up. No inputs should ever be left floating.

All open-drain outputs require a pullup device. While in some cases a simple pullup resistor will be adequate, we recommend a network of pullup and pulldown resistors biased to a valid V_{IH} ($\geq$3.4V) and terminated in the characteristic impedance of the circuit board. Figure 6 shows our recommendations for the resistor values for both a low and high current drive network, which assumes that the circuit board has a characteristic impedance of 100Ω. The advantage of terminating the output signals in this fashion is that it limits signal swing and reduces AC power consumption.

Characteristic Curves

Figure 7 shows the typical supply current requirements over the operating temperature range of the processor at supply voltage (V_{CC}) of 5V. Figure 8 shows the typical power supply current (I_{CC}) required by the 80960KB at various operating frequencies when measured at three input voltage (V_{CC}) levels.

For a given output current (I_{OL}), the curve in Figure 9 shows the worst case output low voltage (V_{OL}).

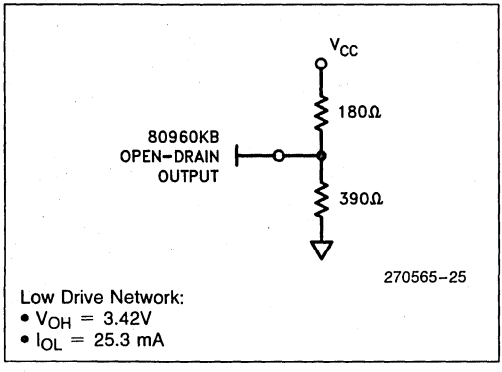
Figure 10 shows the typical capacitive derating curve for the 80960KB measured from 1.5V on the system clock (CLK) to 1.5V on the falling edge and 1.5V on the rising edge of the L-Bus address/data (LAD) signals.

Test Load Circuit

Figure 13 illustrates the load circuit used to test the 80960KB's tristate pins, and Figure 14 shows the load circuit used to test the open drain outputs. The open drain test uses an active load circuit in the form of a matched diode bridge. Since the open-drain outputs sink current, only the I_{OL} legs of the bridge are necessary and the I_{OH} legs are not used. When the 80960KB driver under test is turned off, the output pin is pulled up to V_{REF} (i.e., V_{OH}). Diode D_1 is turned off and the I_{OL} current source flows through diode D_2.

When the 80960KB open-drain driver under test is on, diode D_1 is also on, and the voltage on the pin being tested drops to V_{OL}. Diode D_2 turns off and I_{OL} flows through diode D_1.

3



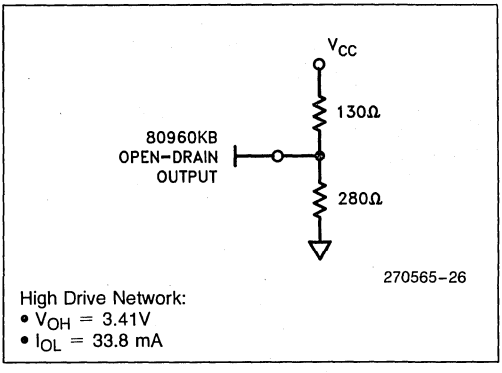


Figure 6. Connection Recommendations for Low and High Current Drive Networks

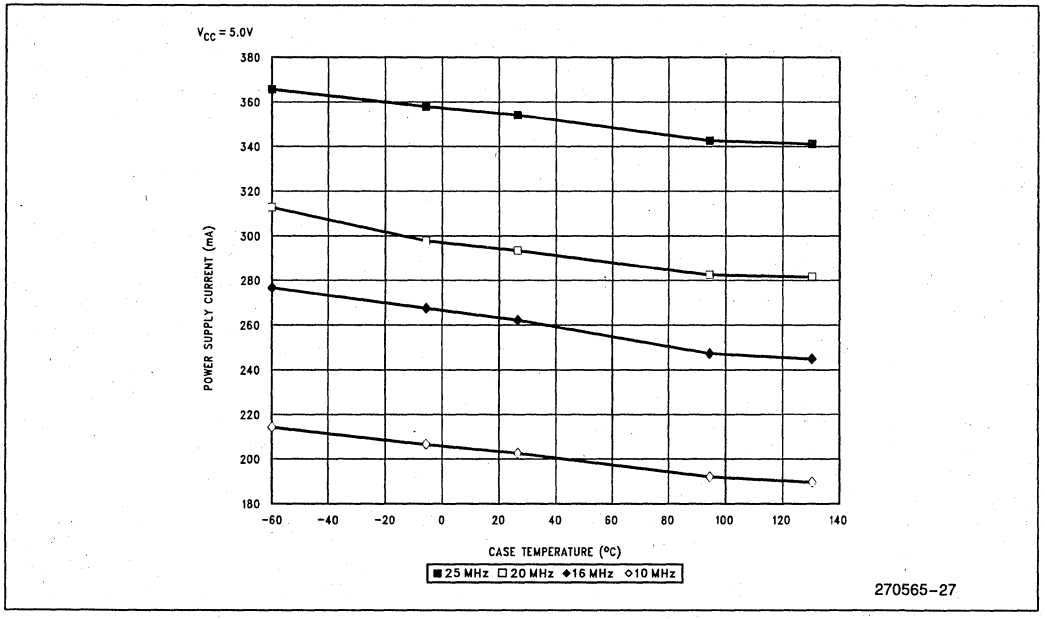


Figure 7. Typical Supply Current (I_{CC})

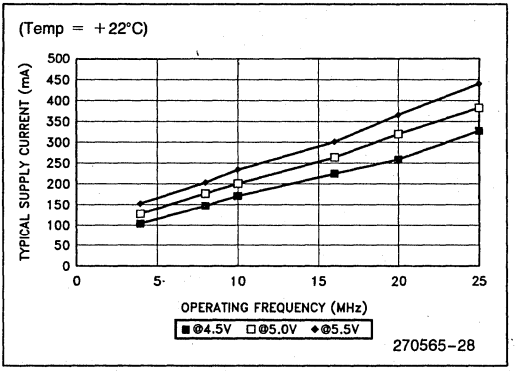


Figure 8. Typical Current vs Frequency

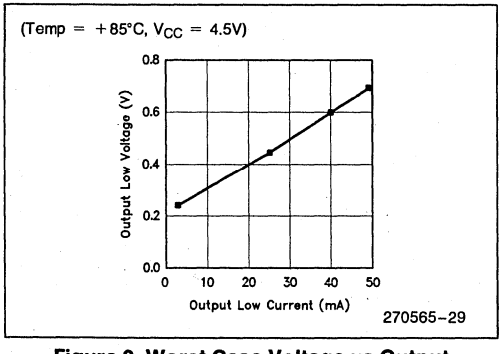


Figure 9. Worst Case Voltage vs Output Current on Open-Drain Pins

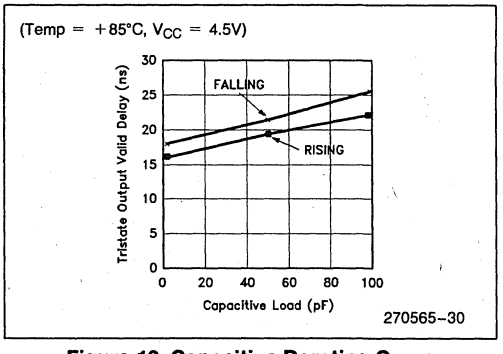


Figure 10. Capacitive Derating Curve

ABSOLUTE MAXIMUM RATINGS*

Operating Temperature . . . . . 0°C to +85°C Case
Storage Temperature . . . . . −65°C to +150°C
Voltage on Any Pin . . . . . −0.5V to V_{CC} + 0.5V
Power Dissipation . . . . . 2.5W (25 MHz)

NOTICE: This is a production data sheet. The specifications are subject to change without notice.

**WARNING: Stressing the device beyond the "Absolute Maximum Ratings" may cause permanent damage. These are stress ratings only. Operation beyond the "Operating Conditions" is not recommended and extended exposure beyond the "Operating Conditions" may affect device reliability.*

DC CHARACTERISTICS

PGA:
80960KB (16 MHz): T_{CASE} = 0°C to + 85°C, V_{CC} = 5V ± 10%
80960KB (20 and 25 MHz): T_{CASE} = 0°C to + 85°C, V_{CC} = 5V ± 5%

PQFP:
80960KA (10 and 16 MHz): T_{CASE} = 0°C to +100°C, V_{CC} = 5V ±10%
80960KA (20 MHz): T_{CASE} = 0°C to +100°C, V_{CC} = 5V ±5%

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.3	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.3	V	
V_{CL}	CLK2 Input Low Voltage	−0.3	+0.8	V	
V_{CH}	CLK2 Input High Voltage	0.55 V_{CC}	V_{CC} + 0.3	V	
V_{OL}	Output Low Voltage		0.45	V	(1, 5)
V_{OH}	Output High Voltage	2.4		V	(2, 4)
I_{CC}	Power Supply Current: 10 MHz 16 MHz 20 MHz 25 MHz		 300 375 420 480	 mA mA mA mA	
I_{LI}	Input Leakage Current		±15	μA	$0 \le V_{IN} \le V_{CC}$
I_{LO}	Output Leakage Current		±15	μA	$0.45 \le V_O \le V_{CC}$
C_{IN}	Input Capacitance		10	pF	f_C = 1 MHz(3)
C_O	I/O or Output Capacitance		12	pF	f_C = 1 MHz(3)
C_{CLK}	Clock Capacitance		10	pF	f_C = 1 MHz(3)

NOTES:
1. For tri-state outputs, this parameter is measured at:
   Address/Data . . . . . 4.0 mA
   Controls . . . . . 5.0 mA
2. This parameter is measured at:
   Address/Data . . . . . −1.0 mA
   Controls . . . . . −0.9 mA
   ALE . . . . . −5.0 mA
3. Input, output, and clock capacitance are not tested.
4. Not measured on open-drain outputs.
5. For open-drain outputs . . . . . 25 mA

AC SPECIFICATIONS

This section describes the AC specifications for the 80960KB pins. All input and output timings are specified relative to the 1.5V level of the rising edge. For output timings, the specifications refer to the time it takes the signal to reach 1.5V.

For input timings, the specifications refer to the time at which the signal reaches (for input setup) or leaves (for hold time) the TTL levels of LOW (0.8V) or HIGH (2.0V). All AC testing should be done with input clock voltages of 0.4V and 2.4V, except for the clock (CLK2), which should be tested with input voltages of 0.45 V_{CC} and 0.55 V_{CC}.

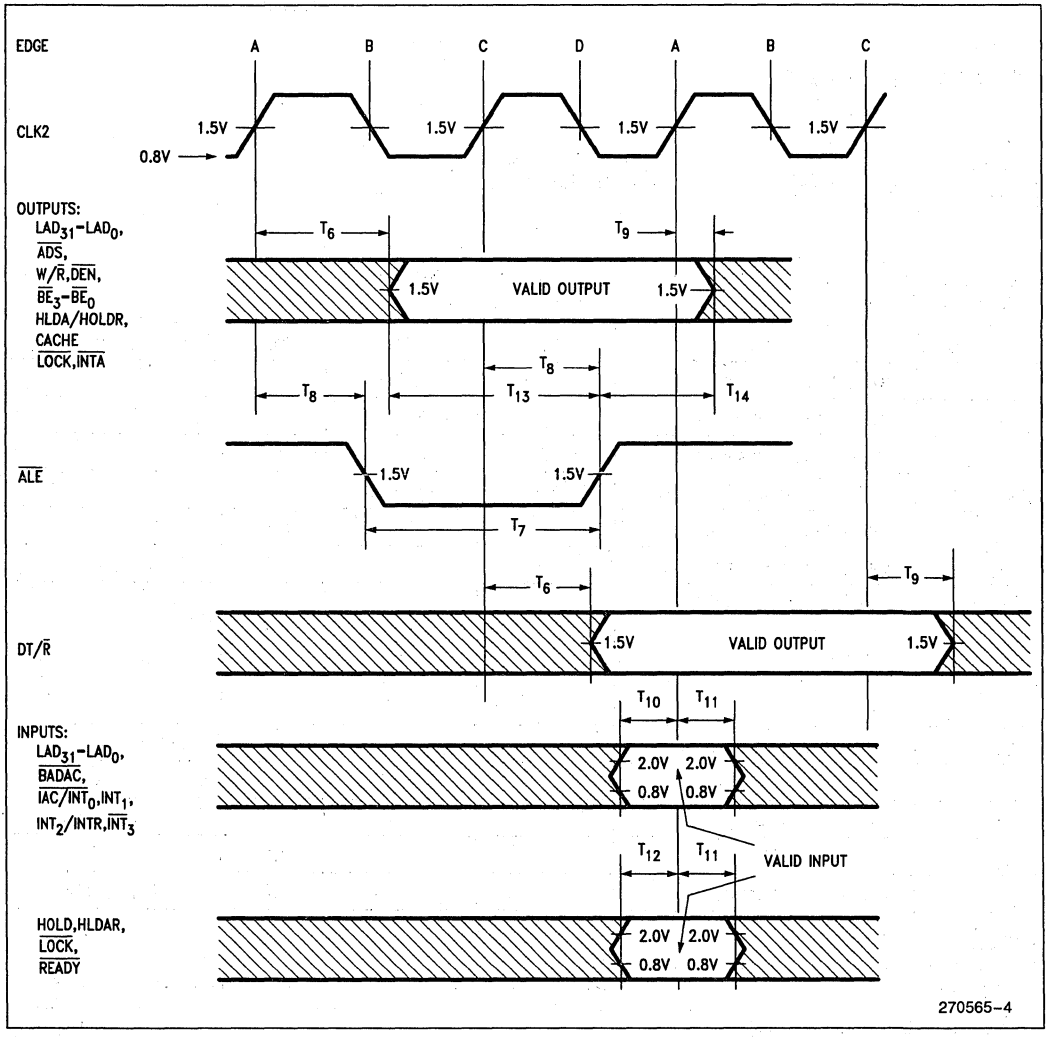


Figure 11. Drive Levels and Timing Relationships for 80960KB Signals

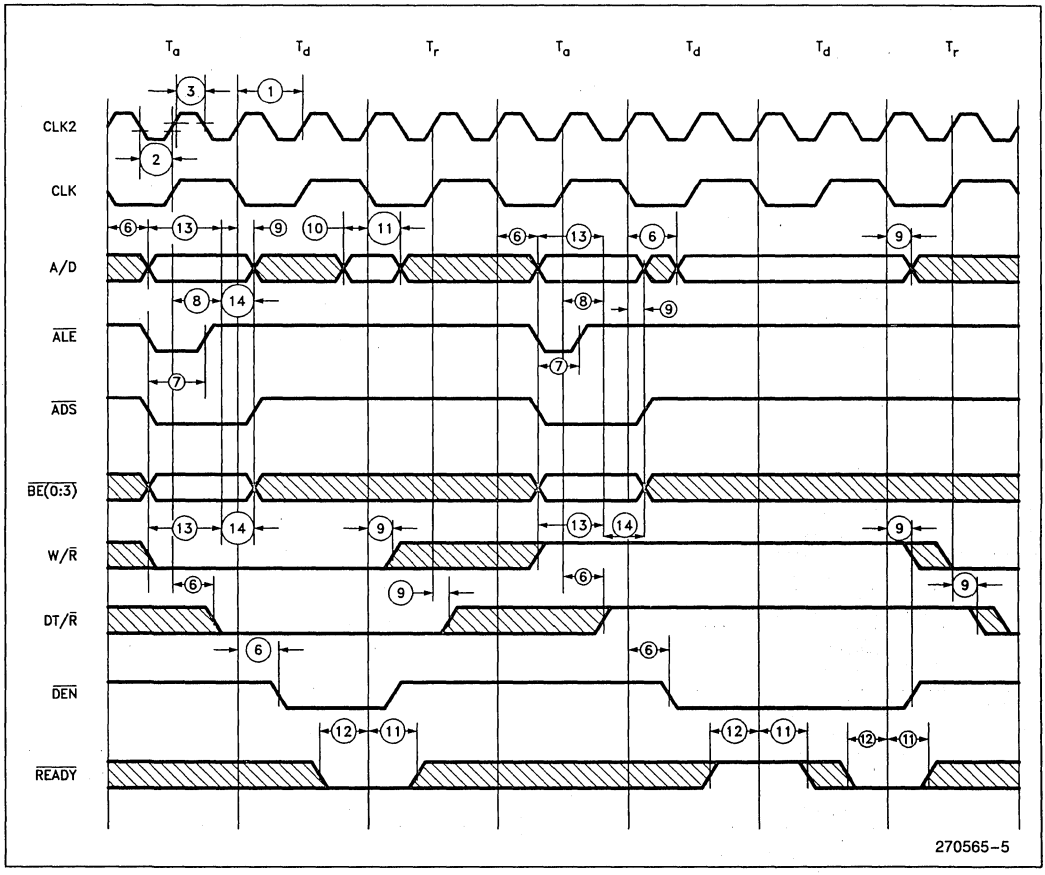


Figure 12. Timing Relationship of L-Bus Signals

AC Specification Tables

80960KB AC Characteristics (10 MHz, PQFP Only)

Symbol	Parameter	Min	Max	Units	Test Conditions
T_1	Processor Clock Period (CLK2)	50	125	ns	V_{IN} = 1.5V
T_2	Processor Clock Low Time (CLK2)	12		ns	V_{IL} = 10% Point = 1.2V
T_3	Processor Clock High Time (CLK2)	12		ns	V_{IH} = 90% Point = 0.1V + 0.5 V_{CC}
T_4	Processor Clock Fall Time (CLK2)		10	ns	V_{IN} = 90% Point to 10% Point
T_5	Processor Clock Rise Time (CLK2)		10	ns	V_{IN} = 10% Point to 90% Point
T_6	Output Valid Delay	2	25	ns	C_L = 100 pF (LAD) C_L = 75 pF (Controls)(2)
T_{6H}	HOLDA Output Valid Delay	4	31	ns	C_L = 75 pF
T_7	$\overline{ALE}$ Width	25		ns	C_L = 75 pF
T_8	$\overline{ALE}$ Output Valid Delay	0	20	ns	C_L = 75 pF(2)
T_9	Output Float Delay	2	20	ns	C_L = 100 pF (LAD) C_L = 75 pF (Controls)
T_{9H}	HOLDA Output Float Delay	4	20	ns	C_L = 75 pF
T_{10}	Input Setup 1	3		ns	
T_{11}	Input Hold	5		ns	
T_{11H}	HOLD Input Hold	4		ns	
T_{12}	Input Setup 2	8		ns	
T_{13}	Setup to $\overline{ALE}$ Inactive	10		ns	C_L = 100 pF (LAD) C_L = 75 pF (Controls)
T_{14}	Hold after $\overline{ALE}$ Inactive	8		ns	C_L = 100 pF (LAD) C_L = 75 pF (Controls)
T_{15}	Reset Hold	3		ns	
T_{16}	Reset Setup	5		ns	
T_{17}	Reset Width	1640		ns	41 CLK2 Periods Minimum

NOTES:
1. $\overline{IAC}/\overline{INT_0}$, INT_1, INT_2/INTR, $\overline{INT_3}$ can be asynchronous.
2. A float condition occurs when the maximum output current becomes less than I_{LO}. Float delay is not tested, but should be no longer than the valid delay.
3. Clock rise and fall times are not tested.

80960KB AC Characteristics (16 MHz)

Symbol	Parameter	Min	Max	Units	Test Conditions
T_1	Processor Clock Period (CLK2)	31.25	125	ns	V_{IN} = 1.5V
T_2	Processor Clock Low Time (CLK2)	8		ns	V_{IL} = 10% Point = 1.2V
T_3	Processor Clock High Time (CLK2)	8		ns	V_{IH} = 90% Point = 0.1V + 0.5 V_{CC}
T_4	Processor Clock Fall Time (CLK2)		10	ns	V_{IN} = 90% Point to 10% Point
T_5	Processor Clock Rise Time (CLK2)		10	ns	V_{IN} = 10% Point to 90% Point
T_6	Output Valid Delay	2	25	ns	C_L = 100 pF (LAD) C_L = 75 pF (Controls)
T_{6H}	HOLDA Output Valid Delay	4	31	ns	C_L = 75 pF
T_7	$\overline{ALE}$ Width	15		ns	C_L = 75 pF
T_8	$\overline{ALE}$ Output Valid Delay	0	20	ns	C_L = 75 pF(2)
T_9	Output Float Delay	2	20	ns	C_L = 100 pF (LAD) C_L = 75 pF (Controls)(2)
T_{9H}	HOLDA Output Float Delay	4	20	ns	C_L = 75 pF
T_{10}	Input Setup 1	3		ns	
T_{11}	Input Hold	5		ns	
T_{11H}	HOLD Input Hold	4		ns	
T_{12}	Input Setup 2	8		ns	
T_{13}	Setup to $\overline{ALE}$ Inactive	10		ns	C_L = 100 pF (LAD) C_L = 75 pF (Controls)
T_{14}	Hold after $\overline{ALE}$ Inactive	8		ns	C_L = 100 pF (LAD) C_L = 75 pF (Controls)
T_{15}	Reset Hold	3		ns	
T_{16}	Reset Setup	5		ns	
T_{17}	Reset Width	1281		ns	41 CLK2 Periods Minimum

NOTES:
1. $\overline{IAC}/\overline{INT_0}$, INT_1, INT_2/INTR, $\overline{INT_3}$ can be asynchronous.
2. A float condition occurs when the maximum output current becomes less than I_{LO}. Float delay is not tested, but should be no longer than the valid delay.
3. Clock rise and fall times are not tested.

80960KB AC Characteristics (20 MHz)

Symbol	Parameter	Min	Max	Units	Test Conditions
T_1	Processor Clock Period (CLK2)	25	125	ns	V_{IN} = 1.5V
T_2	Processor Clock Low Time (CLK2)	6		ns	V_{IL} = 10% Point = 1.2V
T_3	Processor Clock High Time (CLK2)	6		ns	V_{IH} = 90% Point = 0.1V + 0.5 V_{CC}
T_4	Processor Clock Fall Time (CLK2)		10	ns	V_{IN} = 90% Point to 10% Point
T_5	Processor Clock Rise Time (CLK2)		10	ns	V_{IN} = 10% Point to 90% Point
T_6	Output Valid Delay	2	20	ns	C_L = 60 pF (LAD) C_L = 50 pF (Controls)
T_{6H}	HOLDA Output Valid Delay	4	26	ns	C_L = 50 pF
T_7	$\overline{ALE}$ Width	12		ns	C_L = 50 pF
T_8	$\overline{ALE}$ Output Valid Delay	0	20	ns	C_L = 50 pF(2)
T_9	Output Float Delay	2	20	ns	C_L = 60 pF (LAD) C_L = 50 pF (Controls)(2)
T_{9H}	HOLDA Output Float Delay	4	20	ns	C_L = 50 pF
T_{10}	Input Setup 1	3		ns	
T_{11}	Input Hold	5		ns	
T_{11H}	HOLD Input Hold	4		ns	
T_{12}	Input Setup 2	7		ns	
T_{13}	Setup to $\overline{ALE}$ Inactive	10		ns	C_L = 60 pF (LAD) C_L = 50 pF (Controls)
T_{14}	Hold after $\overline{ALE}$ Inactive	8		ns	C_L = 60 pF (LAD) C_L = 50 pF (Controls)
T_{15}	Reset Hold	3		ns	
T_{16}	Reset Setup	5		ns	
T_{17}	Reset Width	1025		ns	41 CLK2 Periods Minimum

NOTES:
1. $\overline{IAC}/\overline{INT_0}$, INT_1, INT_2/INTR, $\overline{INT_3}$ can be asynchronous.
2. A float condition occurs when the maximum output current becomes less than I_{LO}. Float delay is not tested, but should be no longer than the valid delay.
3. Clock rise and fall times are not tested.

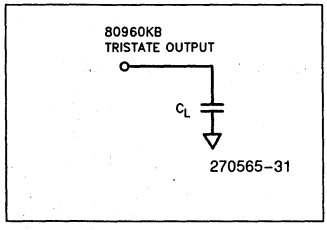


Figure 13. Test Load Circuit for Tri-State Output Pins

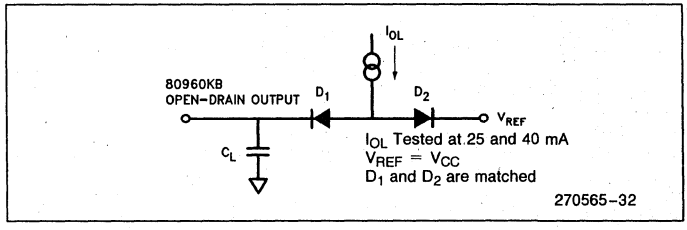


Figure 14. Test Load Circuit for Open-Drain Output Pins

80960KB AC Characteristics (25 MHz, PGA Only)

Symbol	Parameter	Min	Max	Units	Test Conditions
T_1	Processor Clock Period (CLK2)	20	125	ns	V_{IN} = 1.5V
T_2	Processor Clock Low Time (CLK2)	5		ns	V_{IL} = 10% Point = 1.2V
T_3	Processor Clock High Time	5		ns	V_{IH} = 90% Point = 0.1V + 0.5 V_{CC}
T_4	Processor Clock Fall Time (CLK2)		10	ns	V_{IN} = 90% Point to 10% Point
T_5	Processor Clock Rise Time (CLK2)		10	ns	V_{IN} = 10% Point to 90% Point
T_6	Output Valid Delay	2	18	ns	C_L = 60 pF (LAD) C_L = 50 pF (Controls)
T_{6H}	HOLDA Output Valid Delay	4	24	ns	C_L = 50 pF
T_7	$\overline{ALE}$ Width	12		ns	C_L = 50 pF
T_8	$\overline{ALE}$ Output Valid Delay	0	20	ns	C_L = 50 pF (2)
T_9	Output Float Delay	2	18	ns	C_L = 60 pF (LAD) C_L = 50 pF (Controls)
T_{9H}	HOLDA Output Float Delay	4	20	ns	C_L = 50 pF
T_{10}	Input Setup 1	3		ns	
T_{11}	Input Hold	5		ns	
T_{11H}	HOLD Input Hold	4		ns	
T_{12}	Input Setup 2	7		ns	
T_{13}	Setup to $\overline{ALE}$ Inactive	8		ns	C_L = 60 pF (LAD) C_L = 50 pF (Controls)
T_{14}	Hold after $\overline{ALE}$ Inactive	8		ns	C_L = 60 pF (LAD) C_L = 50 pF (Controls)
T_{15}	Reset Hold	3		ns	
T_{16}	Reset Setup	5		ns	
T_{17}	Reset Width	820		ns	41 CLK2 Periods Minimum

NOTES:
1. $\overline{IAC}/\overline{INT0}$, INT1, INT2/INTR, $\overline{INT3}$ can be asynchronous.
2. A float condition occurs when the maximum output current becomes less than I_{LO}. Float delay is not tested, but should be no longer than the valid delay.
3. Clock rise and fall times are not tested.

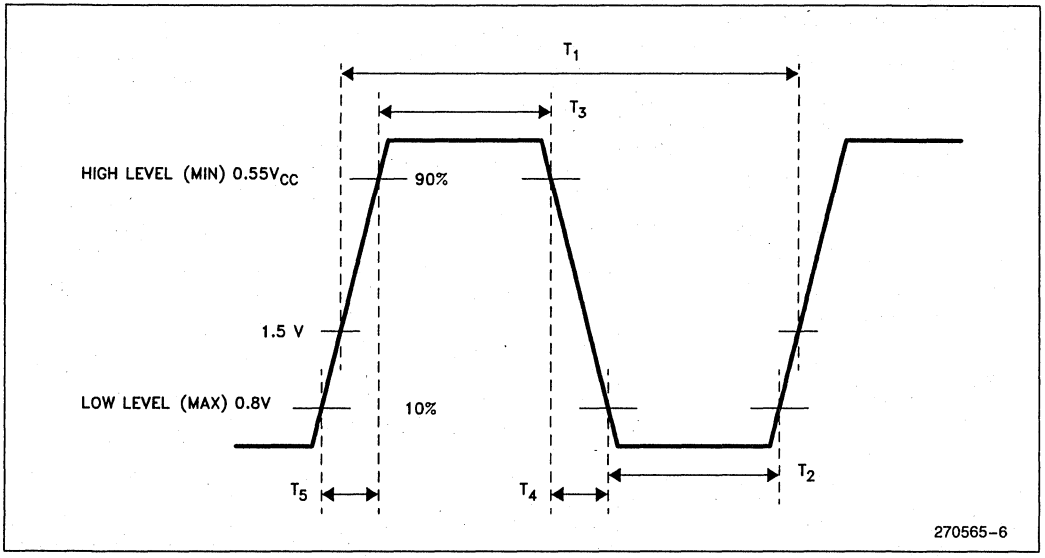


Figure 15. Processor Clock Pulse (CLK2)

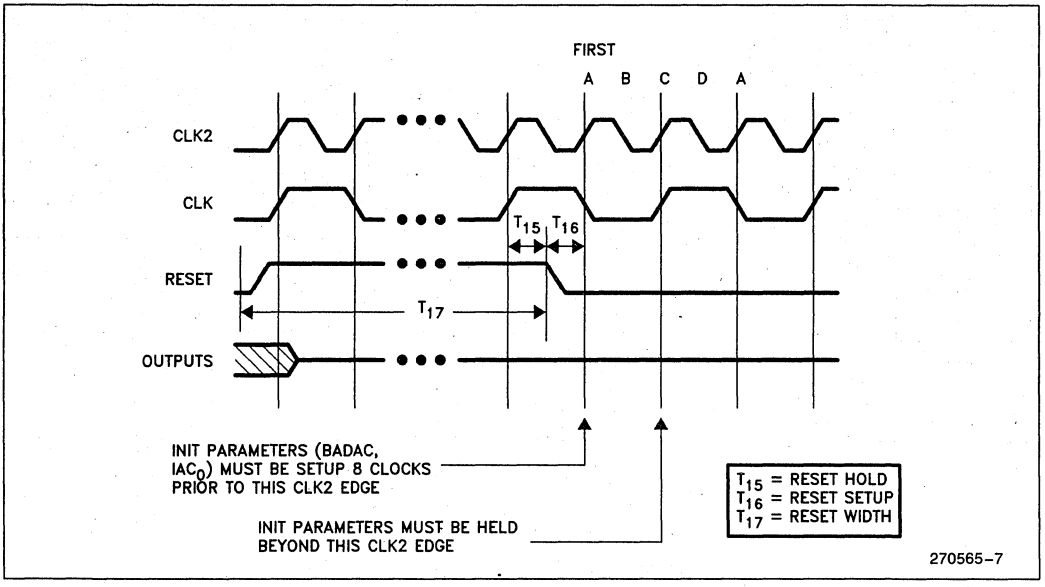


Figure 16. RESET Signal Timing

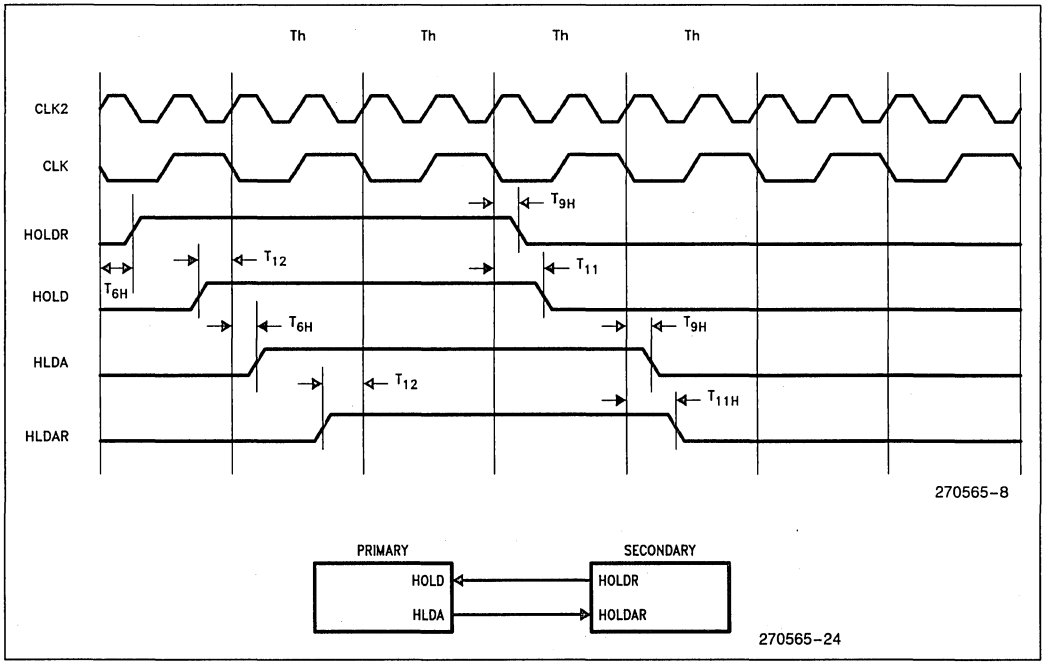


Figure 17. Hold Timing

Design Considerations

Input hold times can be disregarded by the designer whenever the input is removed because a subsequent output from the processor is deasserted (e.g., $\overline{DEN}$ becomes deasserted).

Whenever the processor generates an output that indicates a transition into a subsequent state, any outputs that are specified to be tri-stated in this new state are guaranteed to be tri-stated. For example, in the T_d cycle following a T_a cycle for a read, the minimum output delay of $\overline{DEN}$ is 2 ns, but the maximum float time of LAD is 20 ns. When $\overline{DEN}$ is asserted, however, the LAD outputs are guaranteed to have been tri-stated.

Designing for the ICE-960KB

The 80960KB In-Circuit Emulator assists in debugging both 80960KA and 80960KB hardware and software designs. The product consists of a probe module, cable, and control unit. Because of the high operating frequency of 80960KB systems, the probe module connects directly to the 80960KB socket.

When designing an 80960KB hardware system that uses the ICE-960KB to debug the system, several electrical and mechanical characteristics should be considered. These considerations include capacitive loading, drive requirement, power requirement and physical layout.

The ICE-960KB probe module increases the load capacitance of each line by up to 25 pF. It also adds one standard Schottky TTL load on the CLK2 line, up to one advanced low-power Schottky TTL load for each control signal line, and one advanced low-power Schottky TTL load for each address/data and byte enable line. These loads originate from the probe module and are driven by the 80960KB processor.

To achieve high noise immunity, the ICE-960KB probe is powered by the user's system. The high-speed probe circuitry draws up to 1.1A plus the maximum current (I_{CC}) of the 80960KB processor.

The mechanical considerations are shown in Figure 18, which illustrates the lateral clearance requirements for the ICE-960KB probe as viewed from above the socket of the 80960KB processor.

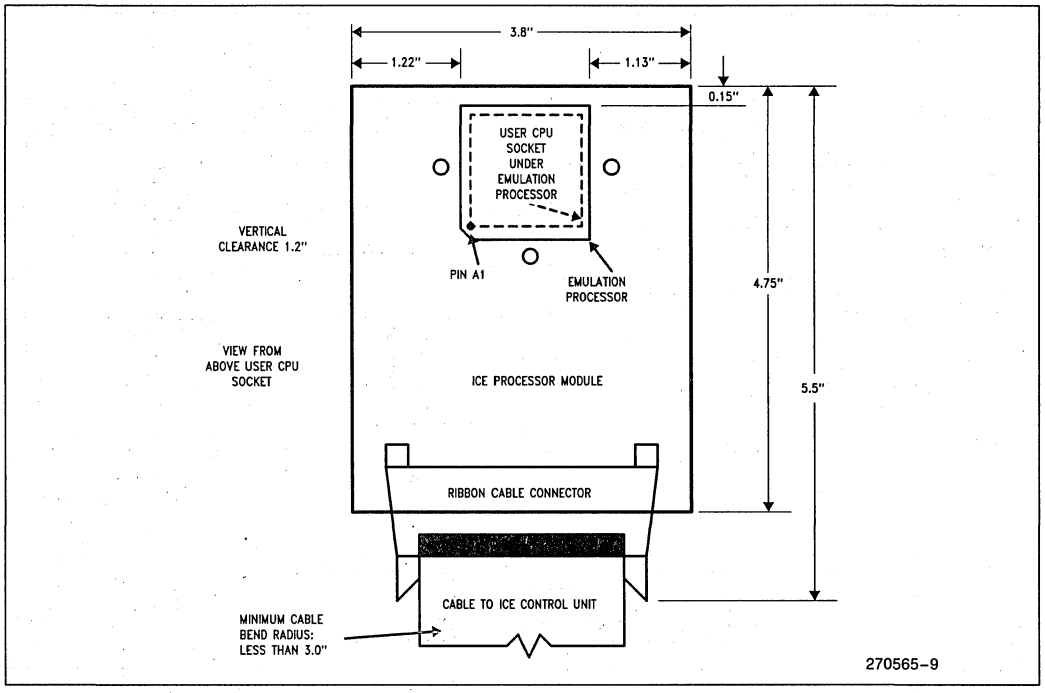


Figure 18. ICE-960KB Lateral Clearance Requirements

MECHANICAL DATA

Package Dimensions and Mounting

The 80960KB is available in two different packages: a 132-lead ceramic pin-grid array (PGA) and a 132-lead plastic quad flat pack (PQFP). Pins in the ceramic package are arranged 0.100 inch (2.54 mm) center-to-center, in a 14 by 14 matrix, three rows around. (See Figure 19.) The plastic package uses fine-pitch gull wing leads arranged in a single row along the perimeter of the package with 0.025 inch (0.64 mm) spacing. (See Figure 20.) Dimensions are given in Figure 21 and Table 7.

There are a wide variety of sockets available for the ceramic PGA package including low-insertion or zero-insertion force mountings, and a choice of terminals such as soldertail, surface mount, or wire wrap. Several applicable sockets are shown in Figure 22.

The PQFP is normally surface mounted to take best advantage of the plastic package's small footprint and low cost. In some applications, however, designers may prefer to use a socket, either to improve heat dissipation or reduce repair costs. Figures 23a and 23b show two of the many sockets available.

Pin Assignment

The PGA and PQFP have different pin assignments. Figure 24 shows the view from the bottom of the PGA (pins facing up) and Figure 25 shows a view from the top of the PGA (pins facing down). Figures 20 and 32 show the top view of the PQFP; notice that the pins are numbered in order from 1 to 132 around the package's perimeter. Tables 5 and 6 list the function of each pin in the PGA, and Tables 8 and 9 list the function of each pin in the PQFP.

V_{CC} and GND connections must be made to multiple V_{CC} and GND pins. Each V_{CC} and GND pin must be connected to the appropriate voltage or ground and externally strapped close to the package. We recommend that you include separate power and ground planes in your circuit board for power distribution.

NOTE:
Pins identified as N.C., "No Connect," should never be connected.

Package Thermal Specification

The 80960KB is specified for operation when case temperature is within the range 0°C to +85°C (PGA) or +100°C (PQFP). The case temperature should be measured at the top center of the package as shown in Figure 26.

The ambient temperature can be calculated from θ_{jc} and θ_{ja} by using the following equations:

$$T_J = T_C + P*\theta_{jc}$$
$$T_A = T_J - P*\theta_{ja}$$
$$T_C = T_A + P*[\theta_{ja} - \theta_{jc}]$$

Values for θ_{ja} and θ_{jc} are given in Table 10 for the PGA package and in Table 11 for the PQFP for various airflows. Note that the θ_{ja} for the PGA package can be reduced by adding a heatsink, while a heatsink is not generally used with the plastic package since it is intended to be surface mounted. The maximum allowable ambient temperature (T_A) permitted without exceeding T_C is shown by the charts in Figures 27 through 30 for 10 MHz, 16 MHz, 20 MHz, and 25 MHz respectively.

The curves assume the maximum permitted supply current (I_{CC}) at each speed, V_{CC} of 5.0V, and a T_{CASE} of +85°C (PGA) or +100°C (PQFP).

If you will be using the 80960KB in a harsh environment where the ambient temperature may exceed the limits for the normal commercial part, you should consider using an extended temperature part. These parts are designed by the prefix "TA" and are available at 16, 20 and 25 MHz in the ceramic PGA package. The extended operating temperature range is −40°C to +125°C case. Figure 30 shows the maximum allowable ambient temperature for the 20 MHz extended temperature TA80960KB at various airflows. The curve assumes an I_{CC} of 420 mA, V_{CC} of 5.0V, and a T_{CASE} of +125°C.

WAVEFORMS

Figures 33 through 38 show the waveforms for various transactions on the 80960KB's local bus.

SUPPORT COMPONENTS

85C960 Burst Bus Controller

The Intel 85C960 performs burst logic, ready generation, and address decode for the 80960KA and 80960KB. The burst logic supports both standard and burst mode memories and peripherals. The ready generation and timing control supports 0 to 15 wait states across eight address ranges for read/write and burst accesses. The address decoder decodes eight address inputs into four external and four internal chip selects. The wait state and chip select values may be programmed by the user; the timing control and burst logic are fixed.

The 85C960 operates with the 80960KA and 80960KB at all frequencies and consumes only 50 mA at 25 MHz. The 85C960 is housed in a 28-pin, 300-mil ceramic DIP and plastic DIP packages or 28-pin PLCC package for surface mount. In the ceramic DIP package the part is UV-erasable, which makes it easy to revise designs. Order the 85C960 data sheet (No. 290192) for full details.

27960KX Burst Mode EPROM

Intel 27960KX one-megabit EPROM is designed specifically to support the 80960KA and 80960KB. It uses a burst interface to offer near zero wait-state performance without the high cost of alternative memory technologies. The 27960KX removes the need for "dumping" code and data stored in slow EPROMs or ROMs into expensive high-speed "shadow" RAM.

Internally, the 27960KX is organized in blocks of four bytes that are accessed sequentially. The address of the four-byte block is latched and incremented internally. After a set number of wait-states (1 or 2), data is output one word at a time each subsequent clock cycle. High-performance outputs provide zero wait-state data-to-data burst accesses. Extra power and ground pins dedicated to the output reduce the effect of fast output switching on the device. The 27960KX offers 1-0-0-0 performance at 20 MHz and 2-0-0-0 performance at 25 MHz. Full details can be found in the 27960KX data sheet (No. 290337).

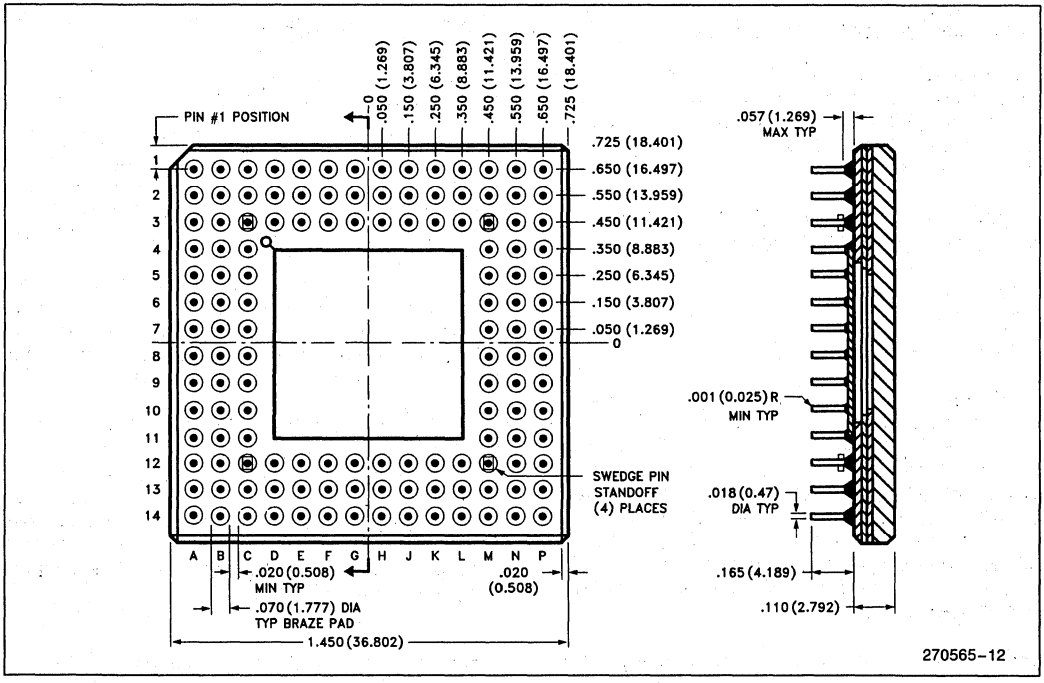


Figure 19. A 132-Lead Pin-Grid Array (PGA) Used to Package the 80960KB

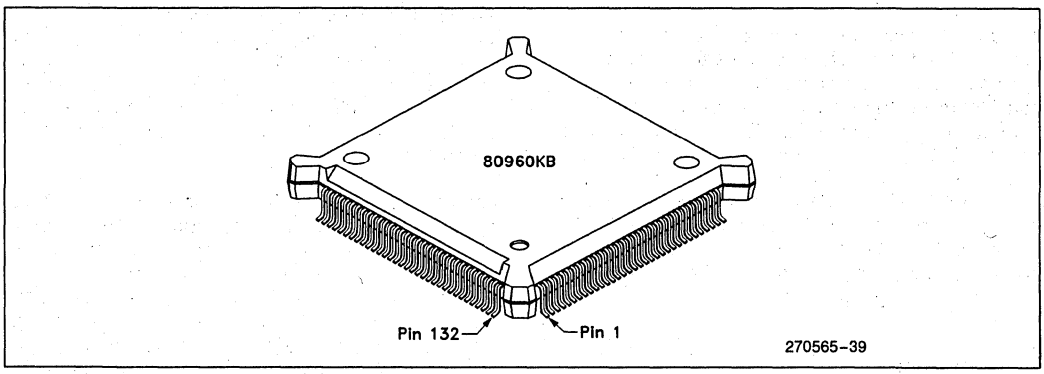


Figure 20. The 132-Lead Plastic Quad Flat Pack (PQFP) used to Package the 80960KB

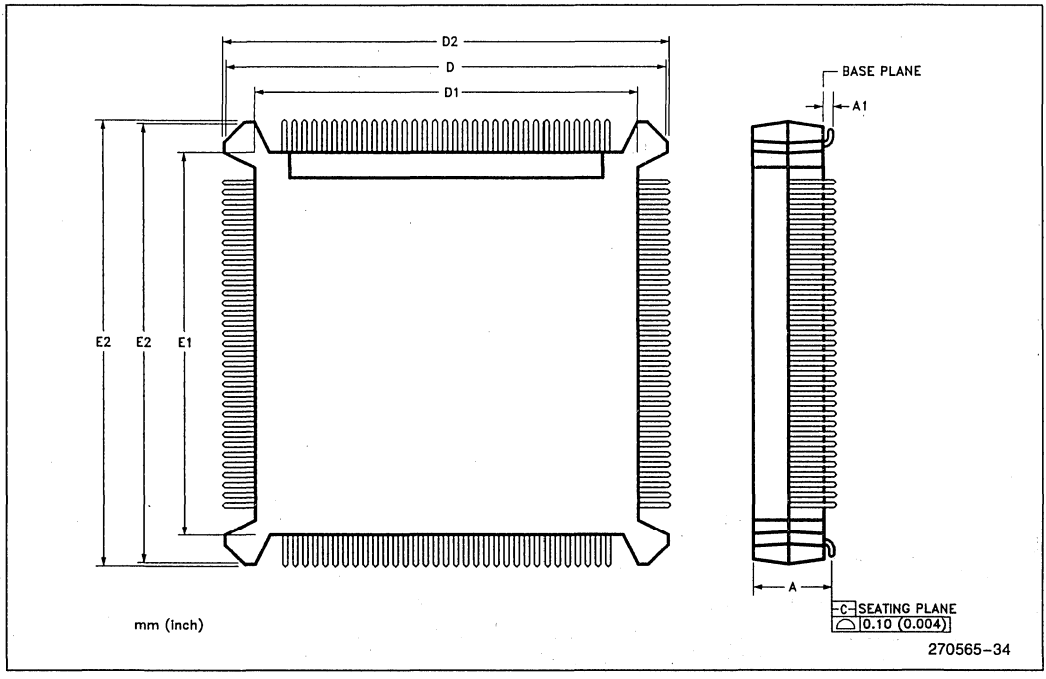


Figure 21a. Principal Dimensions of the 132-Lead PQFP

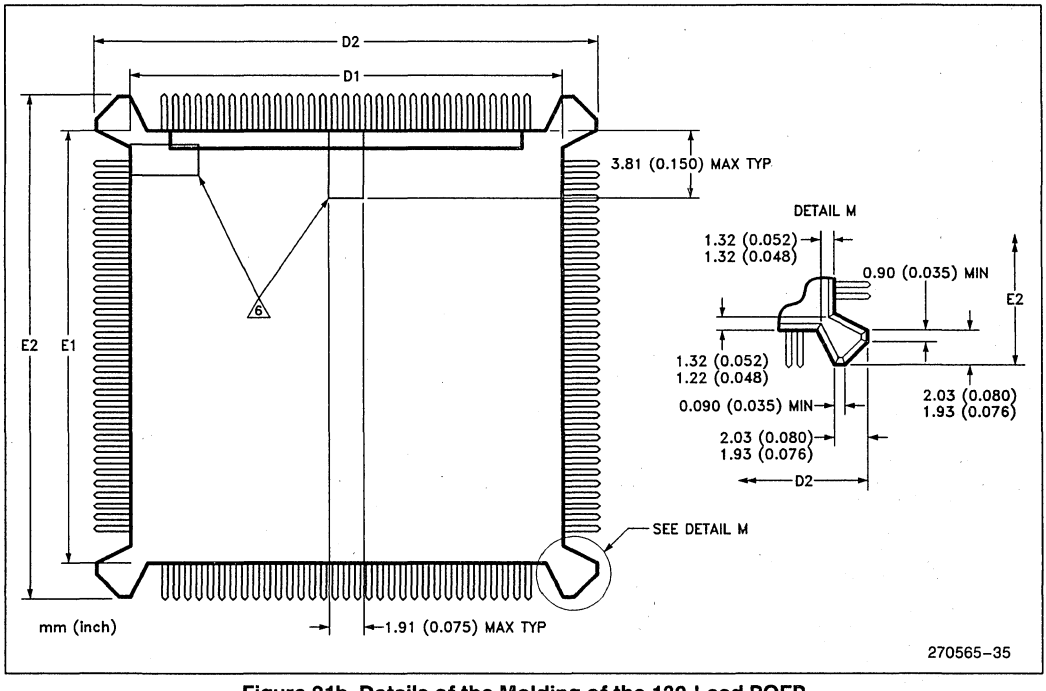


Figure 21b. Details of the Molding of the 132-Lead PQFP

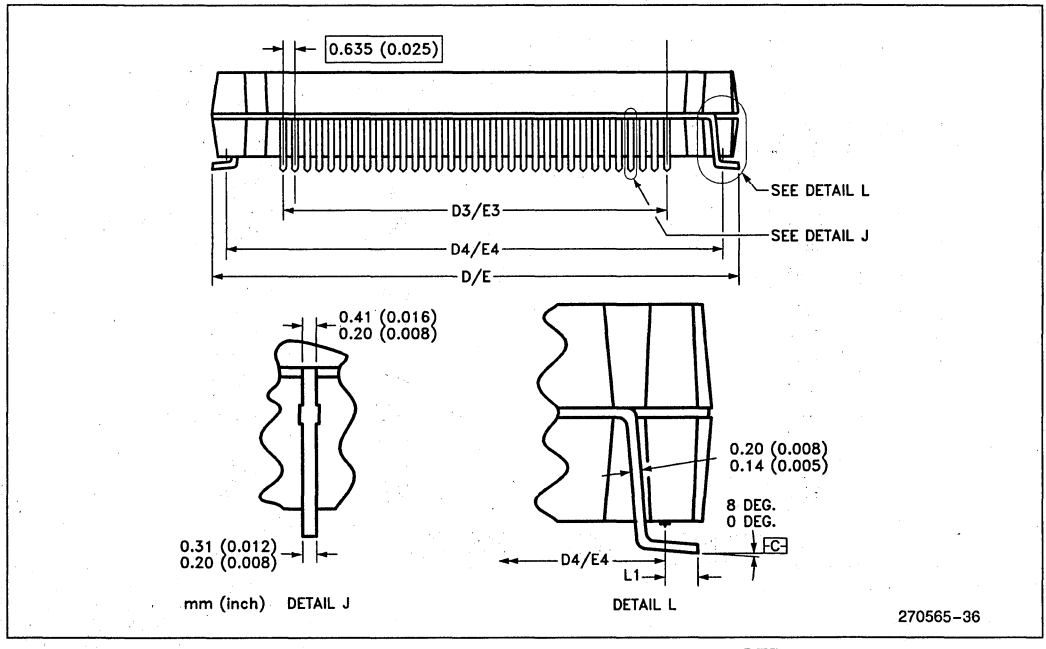


Figure 21c. Terminal Details for the 132-Lead PQFP

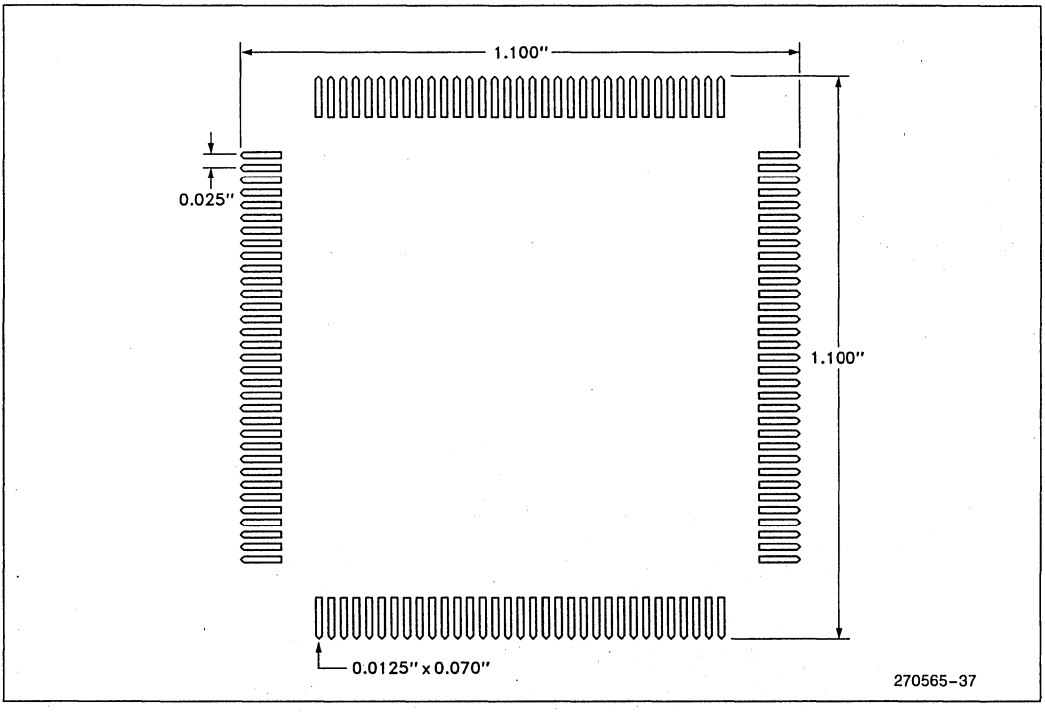


Figure 21d. Board Footprint Area for the 132-Lead PQFP

Table 7. Package Dimension: 80960KB PQFP

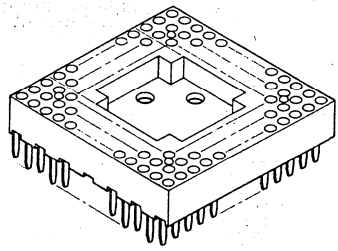
Symbol	Description	Inches		MM	
		Min	Max	Min	Max
N	Leadcount	132 Leads		132 Leads	
A	Package Height	0.160	0.170	4.060	4.320
A1	Standoff	0.020	0.030	0.510	0.760
D,E	Terminal Dimension	1.075	1.085	27.310	27.560
D1,E1	Package Body	0.947	0.953	24.050	24.210
D2,E2	Bumper Distance Without Flash With Flash	 1.097 1.097	 1.103 1.110	 27.860 27.860	 28.010 28.190
D3,E3	Lead Dimension	0.800 REF		20.32 REF	
D4,E4	Foot Radius Location	1.023	1.037	25.890	26.330
L1	Foot Length	0.020	0.030	0.510	0.760

- Low insertion force (LIF) soldertail 55274-1
- Amp tests indicate 50% reduction in insertion force compared to machined sockets

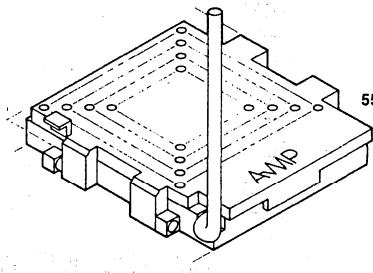
Other socket options

- Zero insertion force (ZIF) soldertail 55583-1
- Zero insertion force (ZIF) Burn-in version 55573-2

Amp Incorporated
(Harrisburg, PA 17105 U.S.A
Phone 717-564-0100)



55274-1



55583-1

270565-13

Cam handle locks in low profile position when 80960KB is installed (handle UP for open and DOWN for closed positions).

Courtesy Amp Incorporated

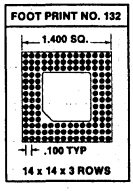
Peel-A-Way* Mylar and Kapton Socket Terminal Carriers

- Low insertion force surface mount CS132-37TG
- Low insertion force soldertail CS132-01TG
- Low insertion force wire-wrap CS132-02TG (two-level) CS132-03TG (thee-level)
- Low insertion force press-fit CS132-05TG

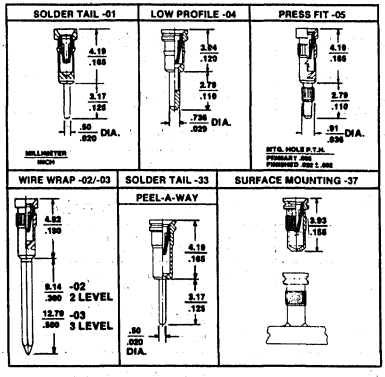
Advanced Interconnections
(5 Division Street)
Warwick, RI 02818 U.S.A.
Phone 401-885-0485)

Peel-A-Way Carrier No. 132:
Kapton Carrier is KS132
Mylar Carrier is MS132

Molded Plastic Body KS132 is shown below:



270565-14



270565-15

Courtesy Advanced Interconnections
(Peel-A-Way Terminal Carriers
U.S. Patent No. 4442938)

*Peel-A-Way is a trademark of Advanced Interconnections.

Figure 22. Several Socket Options for Mounting the 80960KB

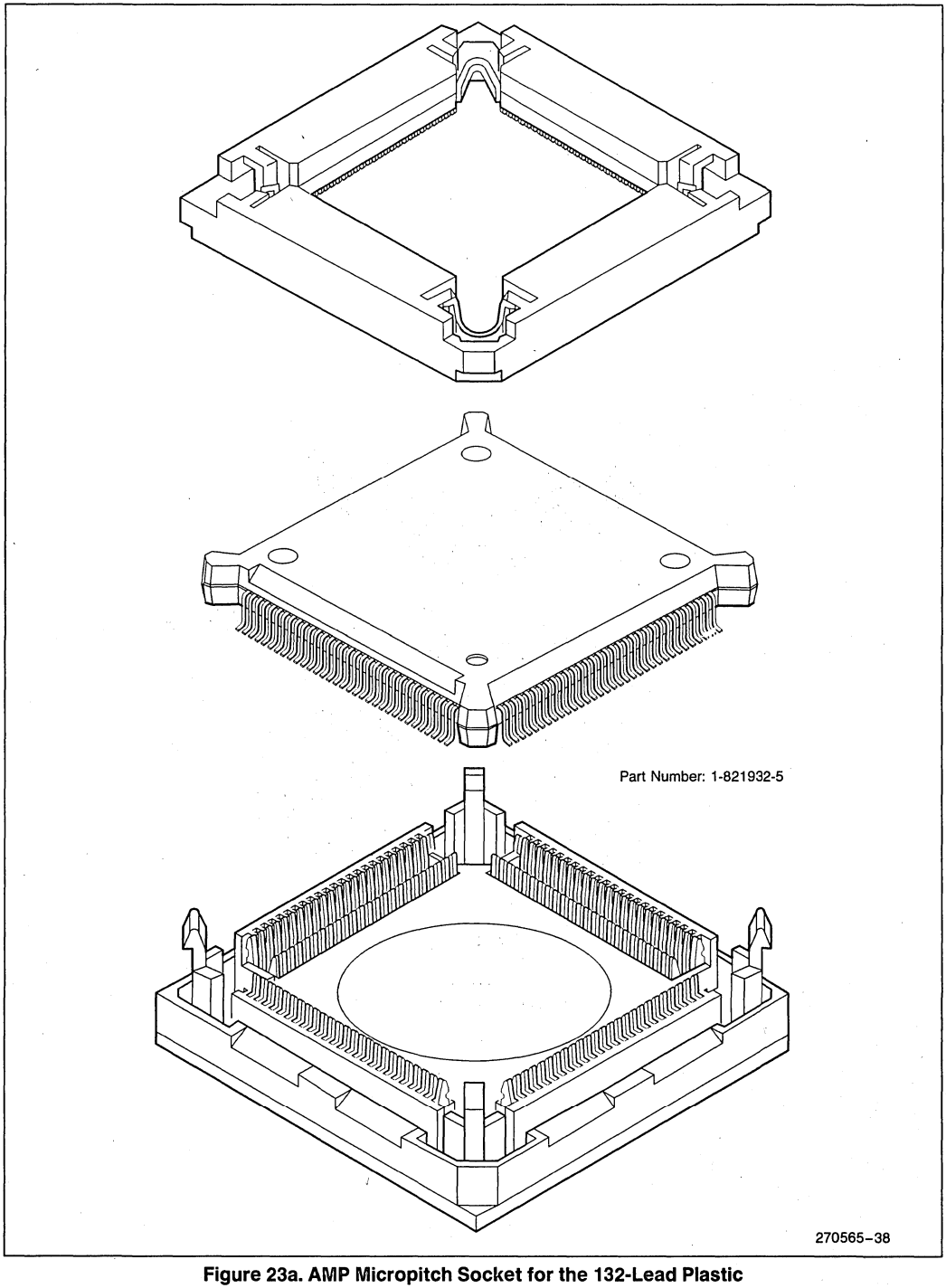


Figure 23a. AMP Micropitch Socket for the 132-Lead Plastic Quad Flat Pack, 0.025″ Lead Spacing, Gull Wing Leads

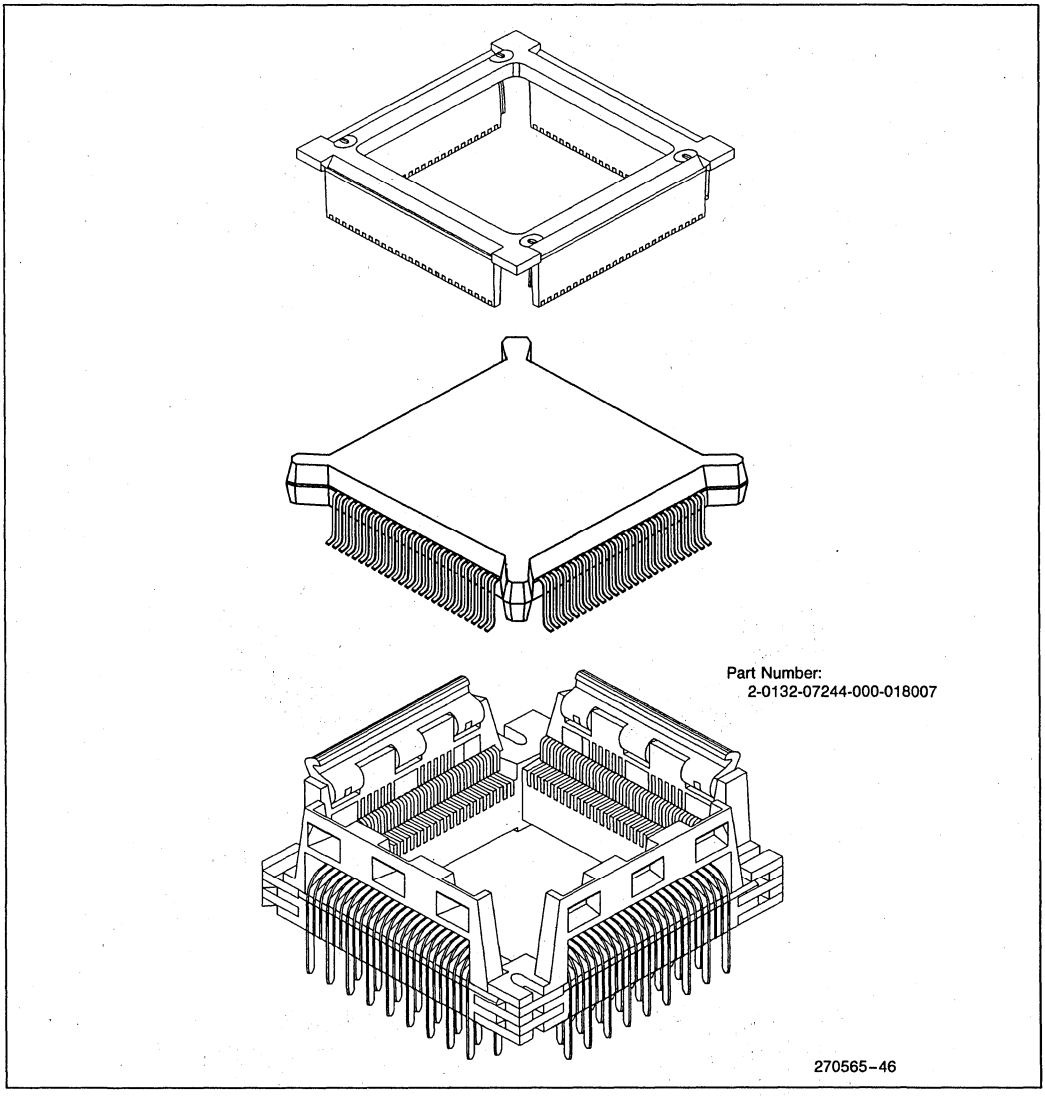


Figure 23b. 3M Company PQFP Socket and Lid

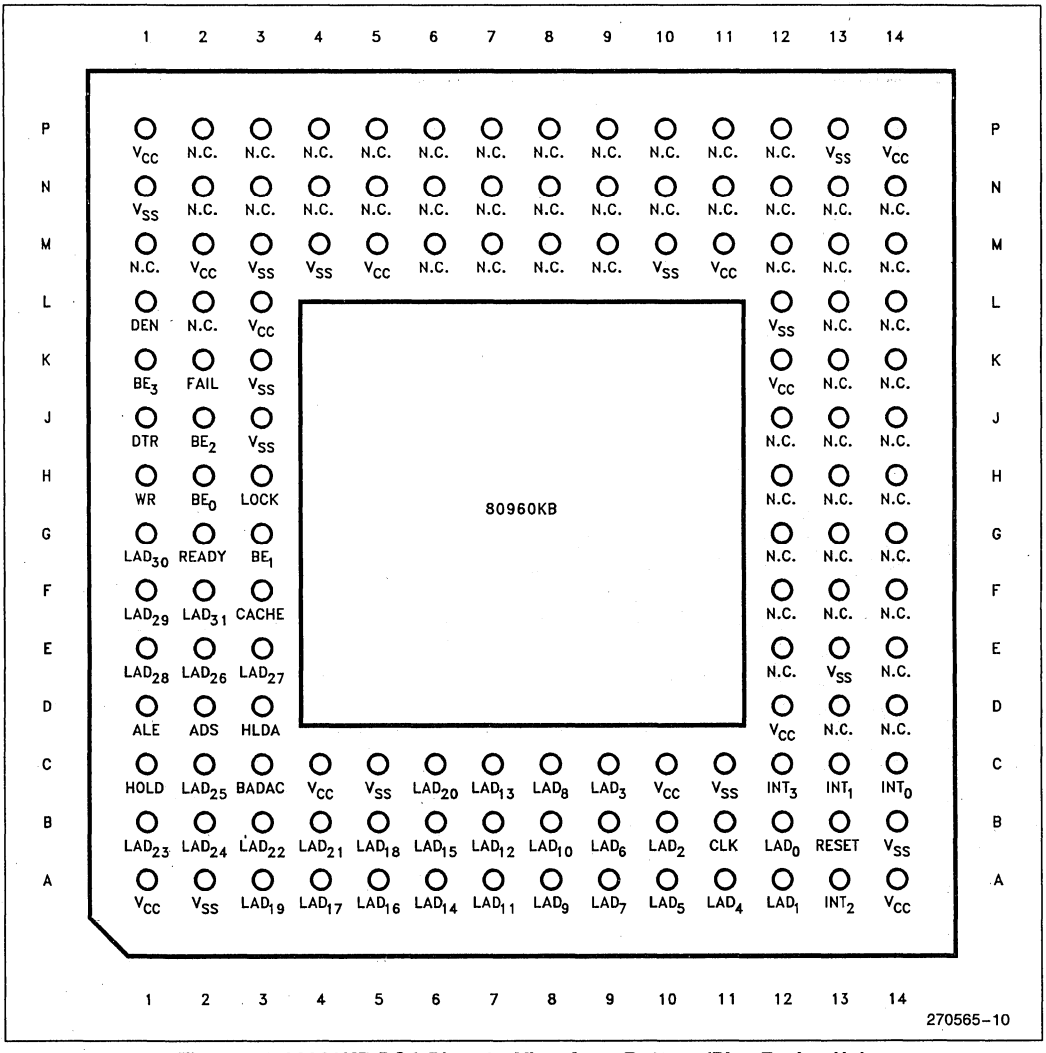


Figure 24. 80960KB PGA Pinout—View from Bottom (Pins Facing Up)

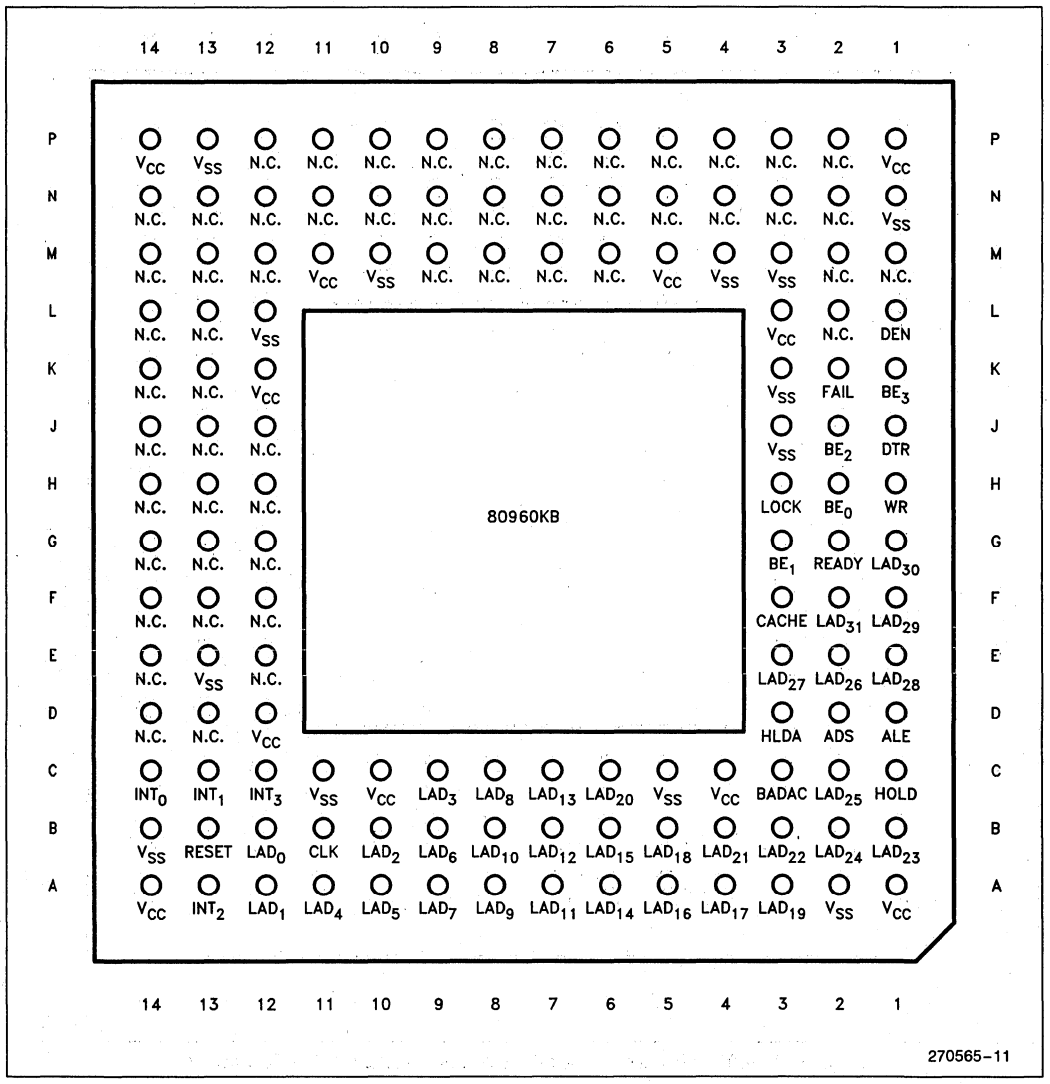


Figure 25. 80960KB PGA Pinout—View from Top (Pins Facing Down)

Table 5. 80960KB PGA Pinout—In Pin Order

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
A1	V_{CC}	C6	LAD_{20}	H1	$W/\overline{R}$	M10	V_{SS}
A2	V_{SS}	C7	LAD_{13}	H2	$\overline{BE_0}$	M11	V_{CC}
A3	LAD_{19}	C8	LAD_8	H3	$\overline{LOCK}$	M12	N.C.
A4	LAD_{17}	C9	LAD_3	H12	N.C	M13	N.C.
A5	LAD_{16}	C10	V_{CC}	H13	N.C.	M14	N.C.
A6	LAD_{14}	C11	V_{SS}	H14	N.C.	N1	V_{SS}
A7	LAD_{11}	C12	$\overline{INT_3}/\overline{INTA}$	J1	$DT/\overline{R}$	N2	N.C.
A8	LAD_9	C13	INT_1	J2	$\overline{BE_2}$	N3	N.C.
A9	LAD_7	C14	$\overline{IAC}/\overline{INT_0}$	J3	V_{SS}	N4	N.C.
A10	LAD_5	D1	$\overline{ALE}$	J12	N.C	N5	N.C.
A11	LAD_4	D2	$\overline{ADS}$	J13	N.C.	N6	N.C.
A12	LAD_1	D3	HLDA/HLDR	J14	N.C.	N7	N.C.
A13	INT_2/INTR	D12	V_{CC}	K1	$\overline{BE_3}$	N8	N.C.
A14	V_{CC}	D13	N.C.	K2	$\overline{FAILURE}$	N9	N.C.
B1	LAD_{23}	D14	N.C.	K3	V_{SS}	N10	N.C.
B2	LAD_{24}	E1	LAD_{28}	K12	V_{CC}	N11	N.C.
B3	LAD_{22}	E2	LAD_{26}	K13	N.C.	N12	N.C.
B4	LAD_{21}	E3	LAD_{27}	K14	N.C.	N13	N.C.
B5	LAD_{18}	E12	N.C.	L1	$\overline{DEN}$	N14	N.C.
B6	LAD_{15}	E13	V_{SS}	L2	N.C.	P1	V_{CC}
B7	LAD_{12}	E14	N.C.	L3	V_{CC}	P2	N.C.
B8	LAD_{10}	F1	LAD_{29}	L12	V_{SS}	P3	N.C.
B9	LAD_6	F2	LAD_{31}	L13	N.C.	P4	N.C.
B10	LAD_2	F3	CACHE	L14	N.C.	P5	N.C.
B11	CLK2	F12	N.C.	M1	N.C.	P6	N.C.
B12	LAD_0	F13	N.C.	M2	V_{CC}	P7	N.C.
B13	RESET	F14	N.C.	M3	V_{SS}	P8	N.C.
B14	V_{SS}	G1	LAD_{30}	M4	V_{SS}	P9	N.C.
C1	HOLD/HLDAR	G2	$\overline{READY}$	M5	V_{CC}	P10	N.C.
C2	LAD_{25}	G3	$\overline{BE_1}$	M6	N.C.	P11	N.C.
C3	$\overline{BADAC}$	G12	N.C.	M7	N.C.	P12	N.C.
C4	V_{CC}	G13	N.C.	M8	N.C.	P13	V_{SS}
C5	V_{SS}	G14	N.C.	M9	N.C.	P14	V_{CC}

Table 6. 80960KB PGA Pinout—In Signal Order

Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin
$\overline{ADS}$	D2	LAD_{15}	B6	N.C.	J14	N.C.	P9
$\overline{ALE}$	D1	LAD_{16}	A5	N.C.	K13	N.C.	P10
$\overline{BADAC}$	C3	LAD_{17}	A4	N.C.	K14	N.C.	P11
$\overline{BE_0}$	H2	LAD_{18}	B5	N.C.	L13	N.C.	P12
$\overline{BE_1}$	G3	LAD_{19}	A3	N.C.	L14	N.C.	L2
$\overline{BE_2}$	J2	LAD_{20}	C6	N.C.	M1	$\overline{READY}$	G2
$\overline{BE_3}$	K1	LAD_{21}	B4	N.C.	M6	RESET	B13
CACHE	F3	LAD_{22}	B3	N.C.	M7	V_{CC}	A1
CLK2	B11	LAD_{23}	B1	N.C.	M8	V_{CC}	A14
$\overline{DEN}$	L1	LAD_{24}	B2	N.C.	M9	V_{CC}	C4
DT/$\overline{R}$	J1	LAD_{25}	C2	N.C.	M12	V_{CC}	C10
$\overline{FAILURE}$	K2	LAD_{26}	E2	N.C.	M13	V_{CC}	D12
HLDA/HOLDR	D3	LAD_{27}	E3	N.C.	M14	V_{CC}	K12
HOLD/HLDAR	C1	LAD_{28}	E1	N.C.	N2	V_{CC}	L3
$\overline{IAC}/\overline{INT_0}$	C14	LAD_{29}	F1	N.C.	N3	V_{CC}	M2
INT_1	C13	LAD_{30}	G1	N.C.	N4	V_{CC}	M5
INT_2/INTR	A13	LAD_{31}	F2	N.C.	N5	V_{CC}	M11
$\overline{INT_3}/\overline{INTA}$	C12	$\overline{LOCK}$	H3	N.C.	N6	V_{CC}	P1
LAD_0	B12	N.C.	D13	N.C.	N7	V_{CC}	P14
LAD_1	A12	N.C.	D14	N.C.	N8	V_{SS}	A2
LAD_2	B10	N.C.	E12	N.C.	N9	V_{SS}	B14
LAD_3	C9	N.C.	E14	N.C.	N10	V_{SS}	C5
LAD_4	A11	N.C.	F12	N.C.	N11	V_{SS}	C11
LAD_5	A10	N.C.	F13	N.C.	N12	V_{SS}	E13
LAD_6	B9	N.C.	F14	N.C.	N13	V_{SS}	J3
LAD_7	A9	N.C.	G12	N.C.	N14	V_{SS}	K3
LAD_8	C8	N.C.	G13	N.C.	P2	V_{SS}	L12
LAD_9	A8	N.C.	G14	N.C.	P3	V_{SS}	M3
LAD_{10}	B8	N.C.	H12	N.C.	P4	V_{SS}	M4
LAD_{11}	A7	N.C.	H13	N.C.	P5	V_{SS}	M10
LAD_{12}	B7	N.C.	H14	N.C.	P6	V_{SS}	N1
LAD_{13}	C7	N.C.	J12	N.C.	P7	V_{SS}	P13
LAD_{14}	A6	N.C.	J13	N.C.	P8	W/$\overline{R}$	H1

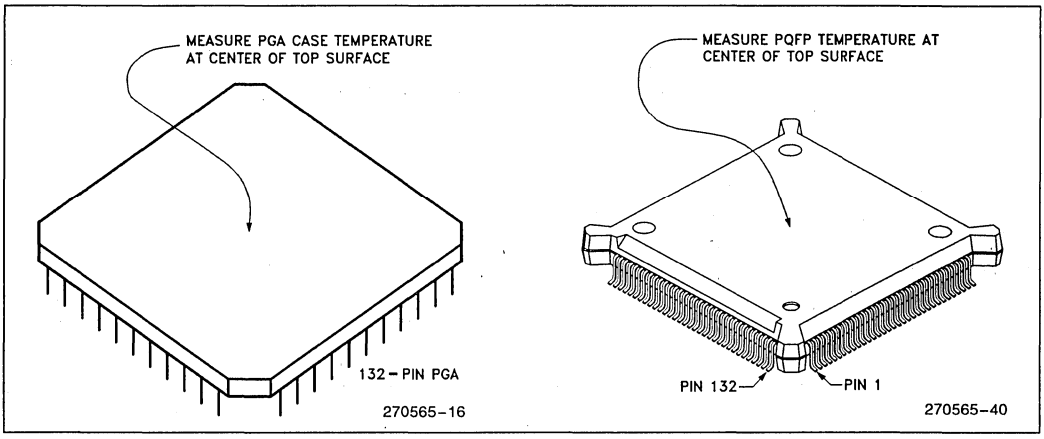


Figure 26. Measuring 80960KB PGA and PQFP Case Temperature

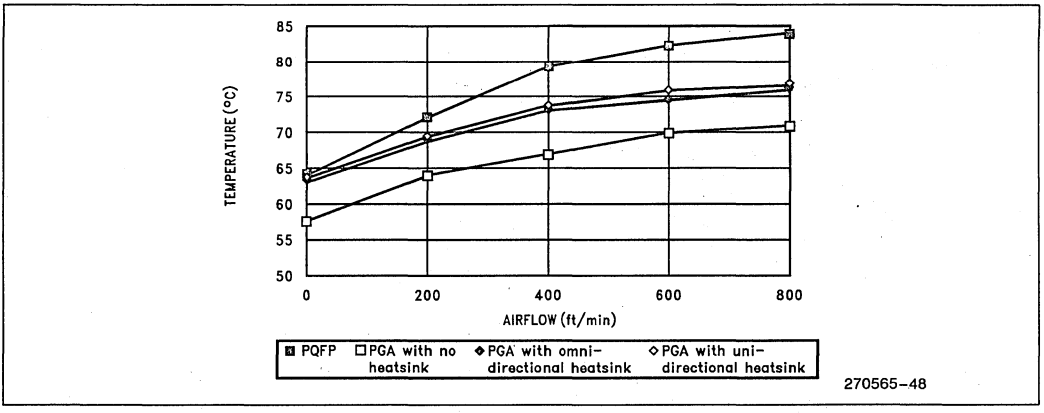


Figure 27. 10 MHz 80960 K-Series Maximum Allowable Ambient Temperature

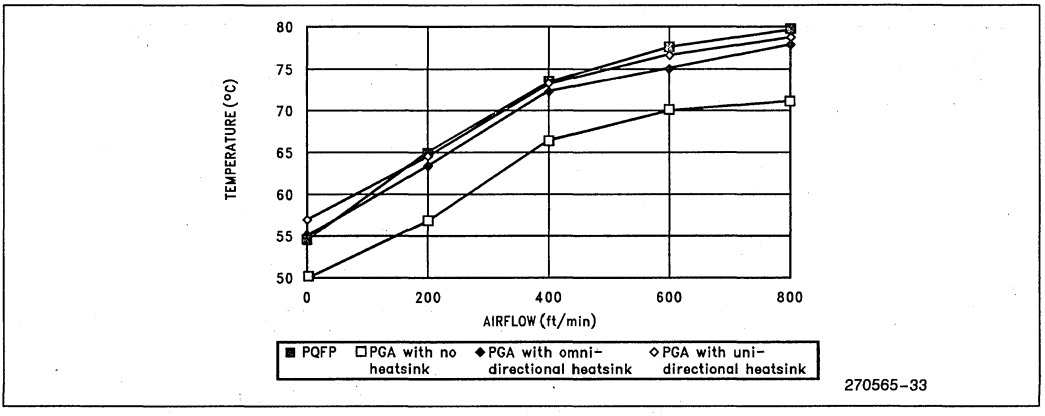


Figure 28. 16 MHz 80960 K-Series Maximum Allowable Ambient Temperature

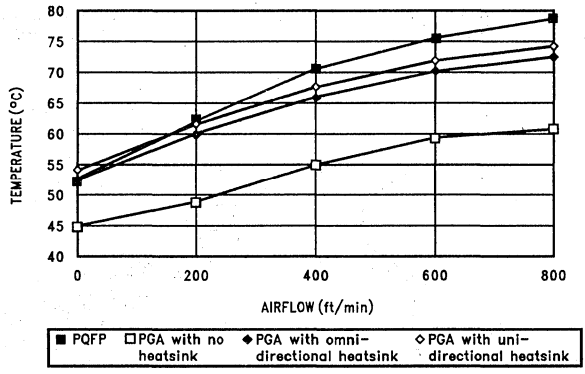


Figure 29. 20 MHz 80960 K-Series Maximum Allowable Ambient Temperature

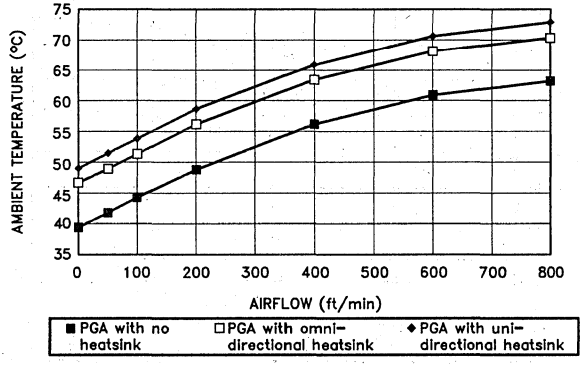


Figure 30. Maximum Allowable Ambient Temperature for the 80960KB at 25 MHz (available in PGA only)

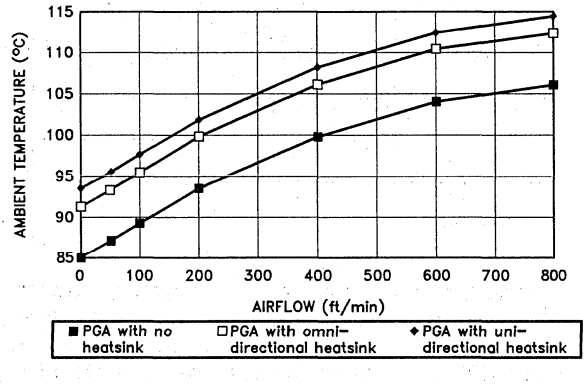


Figure 31. Maximum Allowable Ambient Temperature for the Extended Temperature TA-80960KB at 20 MHz (available in PGA only)

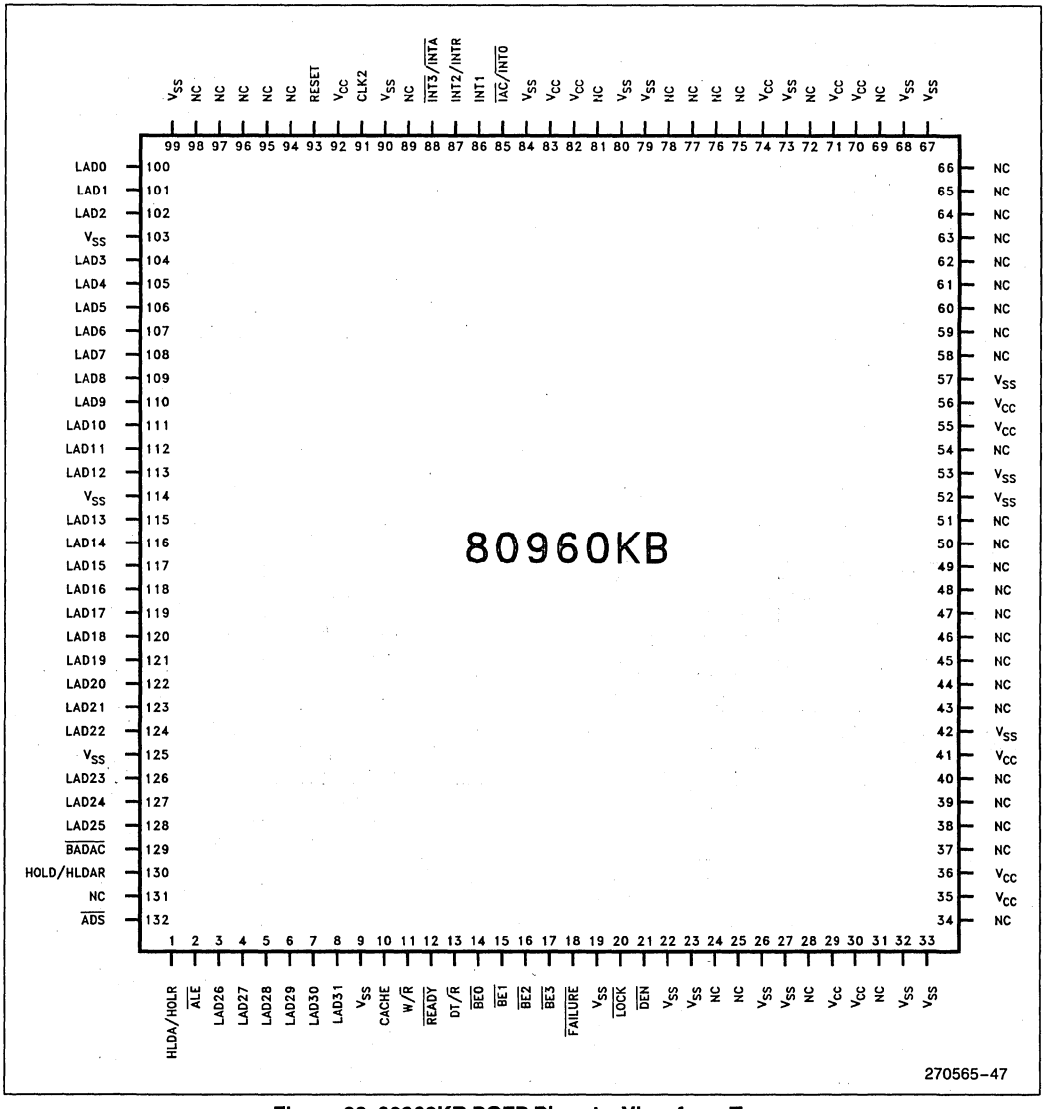


Figure 32. 80960KB PQFP Pinout—View from Top

Table 8. 80960KB Plastic Package Pinout—In Pin Order

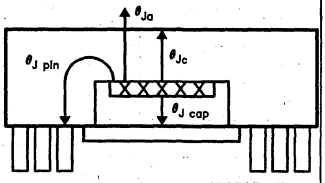
Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	HLDA/HOLR	34	N.C.	67	V_{SS}	100	LAD0
2	$\overline{ALE}$	35	V_{CC}	68	V_{SS}	101	LAD1
3	LAD26	36	V_{CC}	69	N.C.	102	LAD2
4	LAD27	37	N.C.	70	V_{CC}	103	V_{SS}
5	LAD28	38	N.C.	71	V_{CC}	104	LAD3
6	LAD29	39	N.C.	72	N.C.	105	LAD4
7	LAD30	40	N.C.	73	V_{SS}	106	LAD5
8	LAD31	41	V_{CC}	74	V_{CC}	107	LAD6
9	V_{SS}	42	V_{SS}	75	N.C.	108	LAD7
10	CACHE	43	N.C.	76	N.C.	109	LAD8
11	W/$\overline{R}$	44	N.C.	77	N.C.	110	LAD9
12	$\overline{READY}$	45	N.C.	78	N.C.	111	LAD10
13	DT/$\overline{R}$	46	N.C.	79	V_{SS}	112	LAD11
14	$\overline{BE0}$	47	N.C.	80	V_{SS}	113	LAD12
15	$\overline{BE1}$	48	N.C.	81	N.C.	114	V_{SS}
16	$\overline{BE2}$	49	N.C.	82	V_{CC}	115	LAD13
17	$\overline{BE3}$	50	N.C.	83	V_{CC}	116	LAD14
18	$\overline{FAILURE}$	51	N.C.	84	V_{SS}	117	LAD15
19	V_{SS}	52	V_{SS}	85	$\overline{IAC}$/$\overline{INT0}$	118	LAD16
20	$\overline{LOCK}$	53	V_{SS}	86	INT1	119	LAD17
21	$\overline{DEN}$	54	N.C.	87	INT2/INTR	120	LAD18
22	V_{SS}	55	V_{CC}	88	$\overline{INT3}$/$\overline{INTA}$	121	LAD19
23	V_{SS}	56	V_{CC}	89	N.C.	122	LAD20
24	N.C.	57	V_{SS}	90	V_{SS}	123	LAD21
25	N.C.	58	N.C.	91	CLK2	124	LAD22
26	V_{SS}	59	N.C.	92	V_{CC}	125	V_{SS}
27	V_{SS}	60	N.C.	93	RESET	126	LAD23
28	N.C.	61	N.C.	94	N.C.	127	LAD24
29	V_{CC}	62	N.C.	95	N.C.	128	LAD25
30	V_{CC}	63	N.C.	96	N.C.	129	$\overline{BADAC}$
31	N.C.	64	N.C.	97	N.C.	130	HOLD/HLDAR
32	V_{SS}	65	N.C.	98	N.C.	131	N.C.
33	V_{SS}	66	N.C.	99	V_{SS}	132	$\overline{ADS}$

Table 9. 80960KB Plastic Package Pinout—In Signal Order

Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin
$\overline{ADS}$	132	LAD22	124	N.C.	49	V_{CC}	41
$\overline{ALE}$	2	LAD23	126	N.C.	50	V_{CC}	55
$\overline{BADAC}$	129	LAD24	127	N.C.	51	V_{CC}	56
$\overline{BE0}$	14	LAD25	128	N.C.	54	V_{CC}	70
$\overline{BE1}$	15	LAD26	3	N.C.	58	V_{CC}	71
$\overline{BE2}$	16	LAD27	4	N.C.	59	V_{CC}	74
$\overline{BE3}$	17	LAD28	5	N.C.	60	V_{CC}	82
CACHE	10	LAD29	6	N.C.	61	V_{CC}	83
CLK2	91	LAD3	104	N.C.	62	V_{CC}	92
$\overline{DEN}$	21	LAD30	7	N.C.	63	V_{SS}	9
DT/$\overline{R}$	13	LAD31	8	N.C.	64	V_{SS}	19
$\overline{FAILURE}$	18	LAD4	105	N.C.	65	V_{SS}	22
HLDA/HOLR	1	LAD5	106	N.C.	66	V_{SS}	23
HOLD/HLDAR	130	LAD6	107	N.C.	69	V_{SS}	26
$\overline{IAC}$/$\overline{INT0}$	85	LAD7	108	N.C.	72	V_{SS}	27
INT1	86	LAD8	109	N.C.	75	V_{SS}	32
INT2/INTR	87	LAD9	110	N.C.	76	V_{SS}	33
$\overline{INT3}$/$\overline{INTA}$	88	$\overline{LOCK}$	20	N.C.	77	V_{SS}	42
LAD0	100	N.C.	24	N.C.	78	V_{SS}	52
LAD1	101	N.C.	25	N.C.	81	V_{SS}	53
LAD10	111	N.C.	28	N.C.	89	V_{SS}	57
LAD11	112	N.C.	31	N.C.	94	V_{SS}	67
LAD12	113	N.C.	34	N.C.	95	V_{SS}	68
LAD13	115	N.C.	37	N.C.	96	V_{SS}	73
LAD14	116	N.C.	38	N.C.	97	V_{SS}	79
LAD15	117	N.C.	39	N.C.	98	V_{SS}	80
LAD16	118	N.C.	40	N.C.	131	V_{SS}	84
LAD17	119	N.C.	43	$\overline{READY}$	12	V_{SS}	90
LAD18	120	N.C.	44	RESET	93	V_{SS}	99
LAD19	121	N.C.	45	V_{CC}	29	V_{SS}	103
LAD2	102	N.C.	46	V_{CC}	30	V_{SS}	114
LAD20	122	N.C.	47	V_{CC}	35	V_{SS}	125
LAD21	123	N.C.	48	V_{CC}	36	W/$\overline{R}$	11

Table 10. 80960KB PGA Package Thermal Characteristics

Thermal Resistance—°C/Watt							
Parameter	Airflow—ft./min (m/sec)						
	0 (0)	50 (0.25)	100 (0.50)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)
θ Junction-to-Case (Case Measured as shown in Figure 26)	2	2	2	2	2	2	2
θ Case-to-Ambient (No Heatsink)	19	18	17	15	12	10	9
θ Case-to-Ambient (with Omnidirectional Heatsink)	16	15	14	12	9	7	6
θ Case-to-Ambient (with Unidirectional) Heatsink)	15	14	13	11	8	6	5



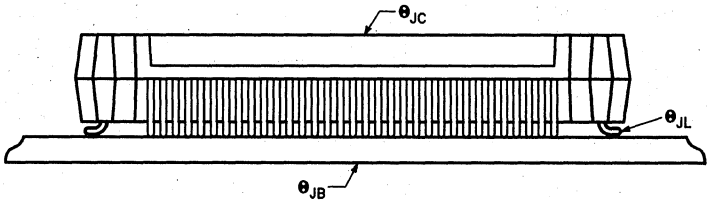
270565-45

NOTES:
1. This table applies to 80960KB PGA plugged into socket or soldered directly into board.
2. $\theta_{JA} = \theta_{JC} + \theta_{CA}$.
3. $\theta_{J\text{-}CAP} = 4°C/w$ (approx.)
   $\theta_{J\text{-}PIN} = 4°C/w$ (inner pins) (approx.)
   $\theta_{J\text{-}PIN} = 8°C/w$ (outer pins) (approx.)

Table 11. 80960KB PQFP Package Thermal Characteristics

PQFP Thermal Resistance—°C/Watt							
Parameter	Airflow—ft./min (m/sec)						
	0 (0)	50 (0.25)	100 (0.50)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)
θ Junction-to-Case (Case Measured as shown in Figure 26)	9	9	9	9	9	9	9
θ Case-to-Ambient (No Heatsink)	22	19	18	16	11	9	8

NOTES:
1. This table applies to 80960KB PQFP soldered directly into board.
2. $\theta_{JA} = \theta_{JC} + \theta_{CA}$.
3. $\theta_{JL} = 18°C/Watt$
   $\theta_{JB} = 18°C/Watt$



270565-44

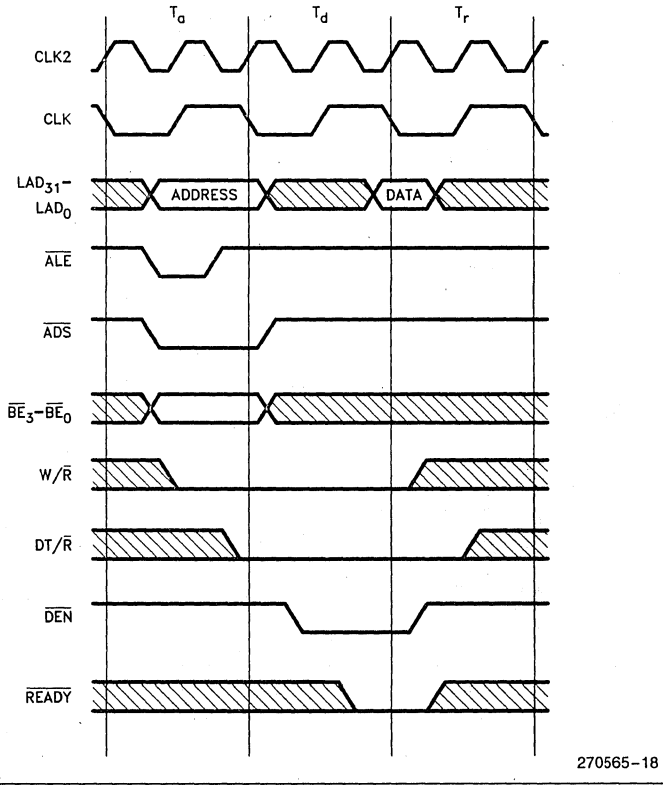


Figure 33. Read Transaction

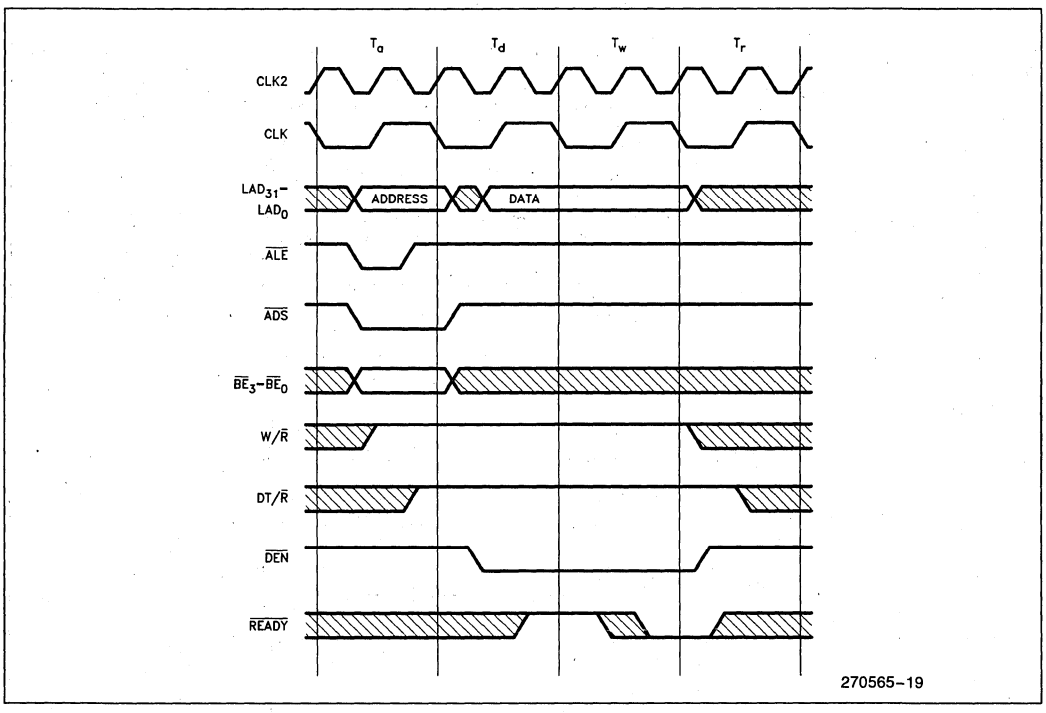


Figure 34. Write Transaction with One Wait State

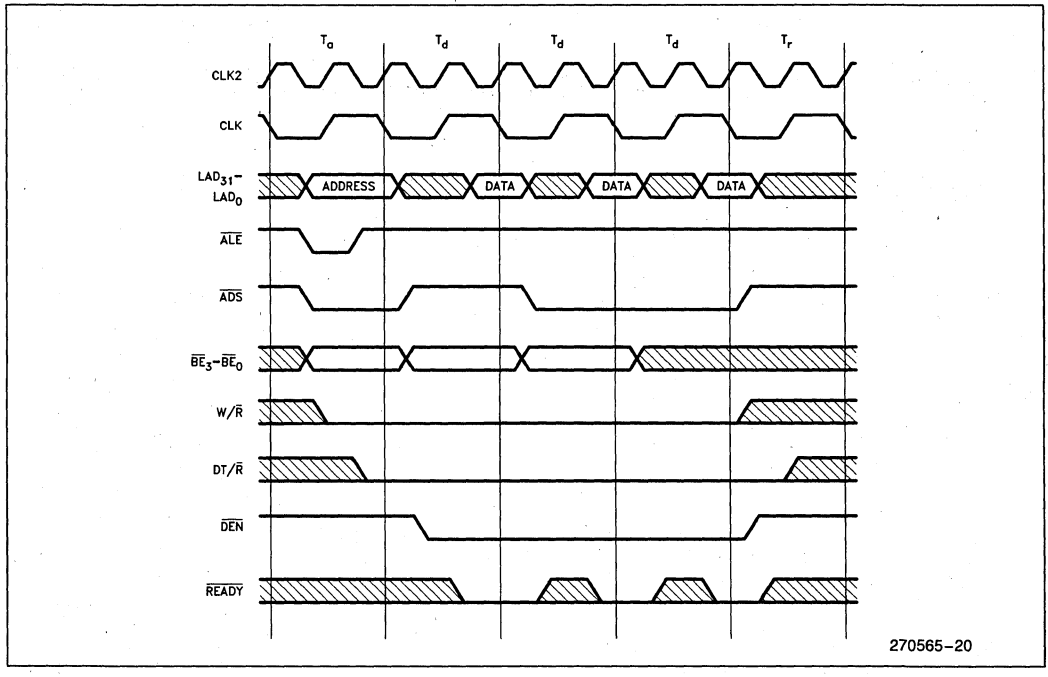


Figure 35. Burst Read Transaction

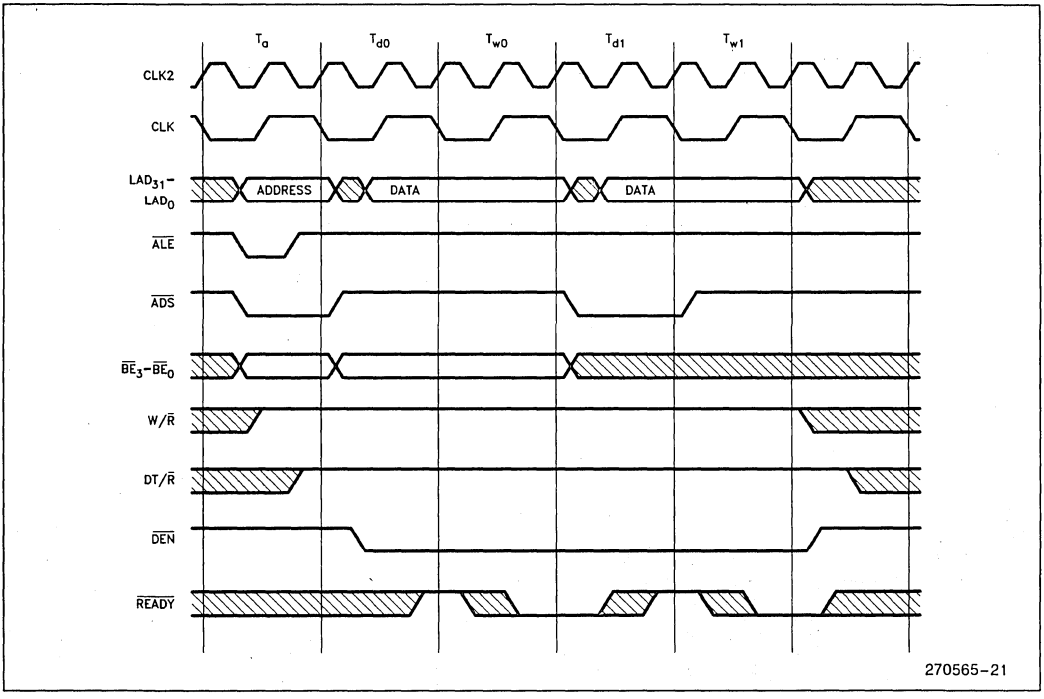
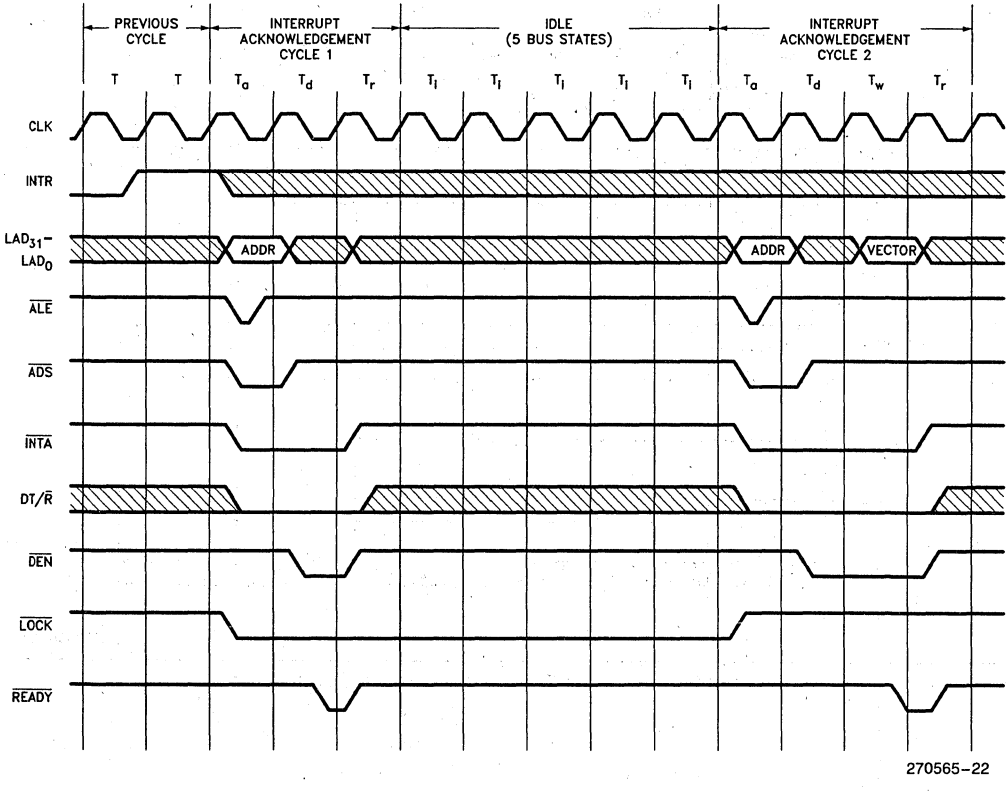


Figure 36. Burst Write Transaction with One Wait State



NOTE:
INTR can go low no sooner than 5 ns (input hold time) following the beginning of interrupt acknowledgement cycle 1. For a second interrupt to be acknowledged, INTR must be low for at least three cycles before it can be reasserted.

Figure 37. Interrupt Acknowledge Transaction

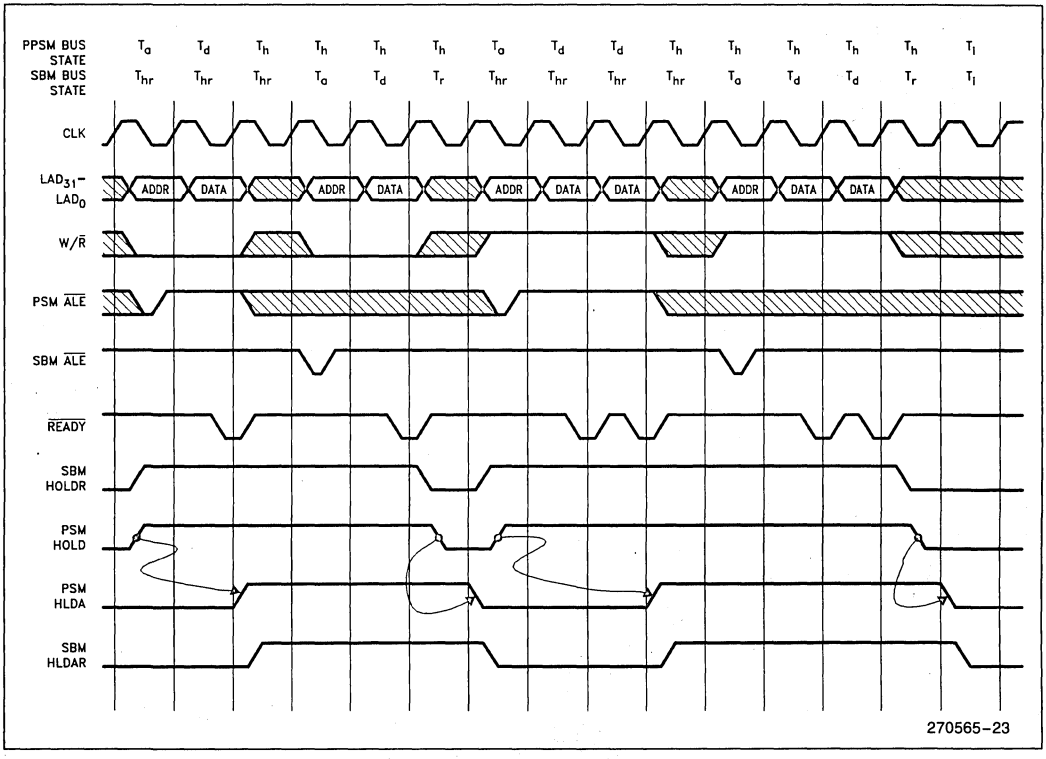


Figure 38. Bus Exchange Transaction (PBM = Primary Bus Master, SBM = Secondary Bus Master)

October 1989

80960CA
Product Overview

32-Bit High-Performance Embedded Processor with On-Chip DMA Controller, Interrupt Controller, High-Speed Bus Unit, Instruction and Register Caches

Order Number: 270669-001

80960CA PRODUCT OVERVIEW

CONTENTS

80960CA PRODUCT OVERVIEW

1.0 PURPOSE

The *80960CA Product Overview* is a summary of the features and operation of Intel's 80960CA Embedded Processor. The Product Overview is intended for those who are not familiar with the 80960 architecture or the 80960CA, a product built around this architecture. The 80960CA Product Overview provides a programmer or a system designer with a quick, global view of software and hardware design considerations for the 80960CA. For further information, refer to the following reference documents:

— The *80960CA User's Manual* contains detailed technical information and examples for designing embedded systems using the 80960CA.

— The *80960CA Data Sheet* provides electrical specifications for the device, such as the DC and AC parameters, operating conditions, and packaging specifications.

2.0 80960CA 32-BIT EMBEDDED PROCESSOR

The 80960CA (Figure 2-1) is optimized for embedded processing applications. This product features the high-performance C-Series core plus built-in system peripherals, effectively integrating a high-speed CPU and system components onto a single silicon die. The 80960CA is a member of Intel's 80960 embedded processor family. Each member of the 80960 family is based on a common architectural definition referred to as the *core architecture.*

An 80960 family member, such as the 80960CA, is made up of an implementation of the core architecture plus application-specific extensions. These extensions may consist of integrated peripherals, instruction-set extensions, or additional registers and caches beyond those defined by the architecture. The common core architecture provides a basis for code compatibility for all 80960 family products, while application-specific extensions optimize a particular product for a class of applications.

The 80960 architectural target is the execution of multiple instructions per clock (i.e., fractional clocks per instruction). By defining an architecture which supports parallel instruction execution and out-of-order instruction execution, performance advances are not constrained by the system clock.

The 80960CA is capable of launching and executing instructions in parallel. This is accomplished by the use of advanced silicon technology as well as innovative "microarchitectural" constructs. The term microarchitecture refers to the implementation of the instruction set and programming resources. For example, different microarchitectures may have different pipeline construction, internal bus widths, register set porting, degrees of parallelism, and cache parameterization (two-way, four-way, etc.).

A principal objective of the 80960 architecture is to provide the framework to allow microarchitectural advances to translate directly into increased performance without architectural limitations.

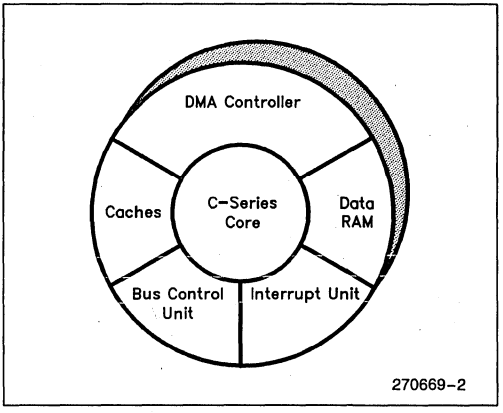


Figure 2-1. 80960CA

2.1 80960 Architecture

Embedded applications are cost sensitive, require a different mix of instructions than reprogrammable applications, have demanding interrupt response requirements, and often use real-time executives rather than full-blown operating systems. The 80960 architecture was developed with these factors in mind. Several key optimizations which are provided by the architecture are explained below.

Instruction Set: Powerful Boolean operations are provided. Frequently executed functions are available as single instructions for greater code density and performance. Call, Return, Compare-and-Branch, Conditional-Compare, Compare-and-Increment or Decrement, and Bit-Field-Extract are each single instructions.

Interrupts: A priority interrupt structure simplifies the management of real-time events. With 31 discrete levels of priority and 248 possible interrupt-handling procedures, this structure provides the low latency and high throughput interrupt handling required in embedded processor applications.

Faults: A generalized fault-handling mechanism simplifies the task of detecting errant arithmetic calculations or other conditions that typically require a significant amount of in-line user code.

Application-Specific Extensions: The core architecture is designed to accept application-specific extensions such as instruction set extensions (e.g., string functions, floating point), special purpose registers, larger caches, on-chip program and data memory, a memory management and protection unit, fault-tolerance support, multiprocessing support, and real-time peripherals (DMA, serial ports, etc.).

2.2 80960 C-Series Core

The C-series core is an implementation of the 80960 core architecture. The core can execute instructions at a sustained speed of 66 MIPS(1) with bursts of performance up to 99 MIPS. To achieve this level of performance, Intel has incorporated state-of-the-art silicon technology and innovative microarchitectural constructs into the C-Series core. Factors which contribute to the core's performance are listed below.

- Parallel instruction decoding allows the 80960CA to start two instructions in every clock, with bursts of three instructions per clock.
- Most instructions execute in a single clock cycle.
- Multiple independent execution units enable overlapping instruction execution.
- Advanced silicon technology allows operation with a 33 MHz internal clock.
- Efficient instruction pipeline is designed to minimize pipeline break losses.
- Register and resource scoreboarding transparently manage parallel execution.
- Branch look-ahead feature enables branches to execute in parallel with other instructions.
- Local register cache is integrated on-chip.
- 1 Kbyte two-way set associative instruction cache is integrated on-chip.
- 1 Kbyte Static Data RAM is integrated on-chip.

These factors combine to make the 80960CA an ultra-high performance computing engine.

NOTE:
1. Single clock instructions at 33 MHz.

2.3 80960CA System Peripherals

The 80960CA features several extensions to the core architecture in the form of integrated peripherals. These peripherals are intended to reduce the external system requirements needed for embedded applications. These peripherals are described below.

Bus Controller Unit: A 32-bit high-performance bus controller interfaces the 80960CA to external memory and peripherals. The bus controller transfers instructions or data at a maximum rate of 132 Mbytes per second.(2) Internally programmable wait states and 16 separately configurable memory regions allow the bus controller to interface with a variety of memory subsystems with minimum system complexity and maximum performance.

DMA Controller: A four channel DMA controller performs high speed data transfers between peripherals and memory. The DMA controller provides advanced features such as data chaining, byte assembly and disassembly, and a fly-by mode capable of transfer speeds of up to 66 Mbytes per second. The DMA controller features a performance and flexibility which is only possible by integrating the DMA controller and the 80960CA core.

Interrupt Controller: A priority interrupt controller manages 8 external interrupt inputs, 4 internal interrupt sources from the DMA controller, and a single non-maskable interrupt input (NMI). A total of 248 external interrupt sources are supported by the interrupt controller by configuring the 8 external interrupt pins as an 8-bit input port. The interrupt controller provides the mechanism for the low latency and high throughput interrupt service featured by the 80960CA. The interrupt latency for the 80960CA is typically less than 1 μs.

3

3.0 EXECUTION ENVIRONMENT

The *Execution Environment* (Figure 3-1) refers to the resources which are available for executing code on the 80960CA. The following sections describe the elements of the execution environment.

3.1 Registers and Literals

The 80960CA provides four types of working data registers: *Global Registers, Local Registers, Special Function Registers* (SFRs), and *Control Registers.*

Global and local registers are general purpose 32-bit data registers. The SFRs and the control registers provide a programmer's interface to the on-chip peripherals (i.e., the DMA controller, interrupt controller, and bus controller).

NOTE:
2. 33 MHz internal clock, load or instruction fetch on 0 wait state, pipelined burst bus.

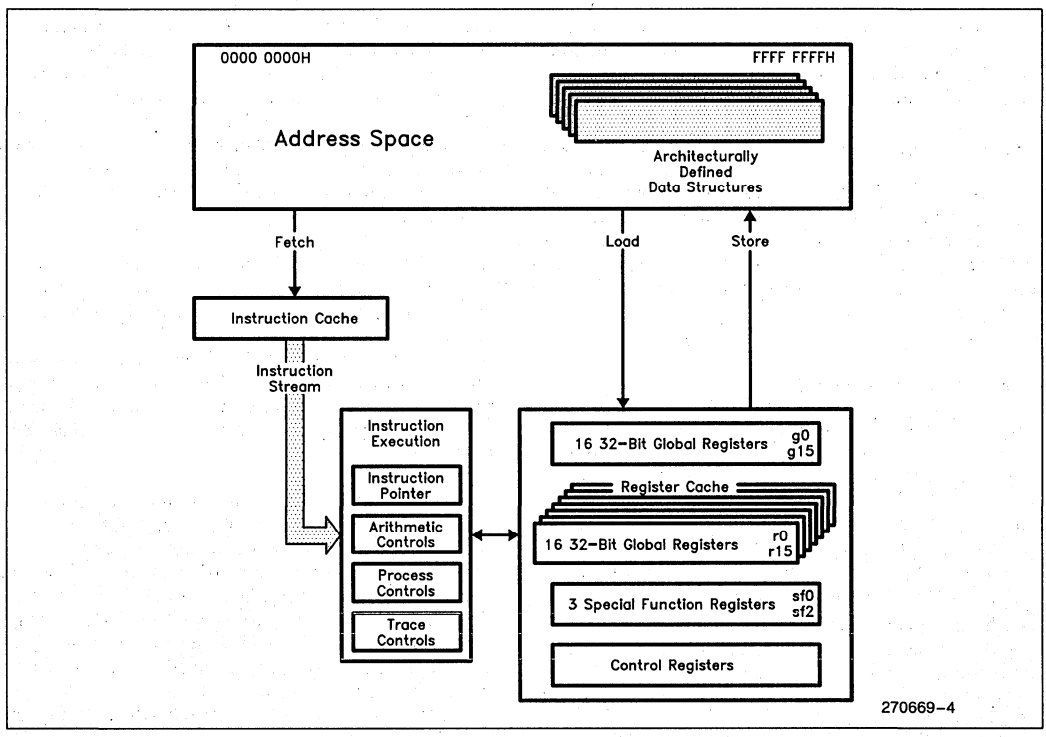


Figure 3-1. Execution Environment

The 80960 architecture is a register-oriented architecture. That is, operands and results of instructions are placed in working data registers rather than in memory. Since the architecture is register oriented, an ample supply of registers is provided. The architecture's working register set consists of 16, 32-bit global registers and 16, 32-bit local registers.

3.1.1 GLOBAL AND LOCAL REGISTERS

The procedure call and return mechanism, which is part of the 80960 architecture, inspires the names given to the local and global registers. When a procedure call or return is executed, the contents of global registers are preserved across procedure boundaries. In other words, the same set of global registers is used for each procedure. A new set of local registers, however, is allocated for each procedure. The 80960's call and return mechanism is explained in Section 3.8.

The 80960CA supplies 16, 32-bit global registers designated **g0** through **g15**. Registers **g0** through **g14** are general purpose global registers. Register **g15** is reserved for the current Frame Pointer. This register is available in assembly language as the **fp** register. The **fp** contains the address of the first byte in the current stack frame. The **fp** register and the stack frame are described in Section 3.8.

The 80960CA supplies 16, 32-bit **Local Registers** designated **r0** through **r15**. Registers **r3** through **r15** are general purpose local registers. Registers **r0**, **r1**, and **r2** are reserved for special functions as follows: **r0** contains the Previous Frame Pointer, **r1** contains the Stack Pointer, and **r2** is reserved for the Return Instruction Pointer. These registers are available in assembly language as, respectively, the **pfp**, **sp**, and **rip** registers. The **pfp**, **sp**, and **rip** registers manage stack frame linkage for the 80960's procedure call and return mechanism. The function of these registers is decribed in Section 3.8.

3.1.2 SPECIAL FUNCTION REGISTERS AND CONTROL REGISTERS

The 80960CA uses 3 Special Function Registers (SFRs) for communicating with on-chip peripherals. These SFR's are an architectural extension specific to the 80960CA. The SFRs on the 80960CA are designated as **sf0**, **sf1**, and **sf2**. SFRs are accessed as source operands by most of the 80960CA's instructions. The registers serve as part of the programmer's interface to the DMA and interrupt controller.

Control registers, like SFRs are used to communicate with the on-chip peripherals. Configuration information for the peripherals is generally stored in these registers. Control registers can only be accessed by using the system control (**sysctl**) instruction. The **sysctl** instruction is used to load the internal control register from a table in external memory called the control table. In order to simplify the process of peripheral configuration, the control registers are automatically loaded from this table at initialization.

3.1.3 LITERALS

The 80960CA provides *literals* which may be used in the place of source register operands in most instructions. The literals range from 0 to 31 (5 bits). When a literal is used as an operand, the processor expands it to 32 bits by adding leading zeros. If the instruction defines an operand larger than 32 bits, the processor zero extends the literal to the operand size.

3.2 Address Space and Memory

The address space of the 80960CA (Figure 3-2) is considered a subset of the execution environment since the code, data, data structures, and external peripherals for the processor reside here. The 80960 family has an address space which is 2^{32} bytes (4 Gbytes) in size. This address space is linear (unsegmented); therefore, code, data, and peripherals may be placed anywhere in the usable space. For the 80960CA, some memory locations are reserved or are assigned special functions as shown in Figure 3-2.

3.2.1 INTERNAL DATA RAM

The 80960CA provides 1 Kbyte of internal static RAM for fast access of frequently used data. The data RAM allows time critical data storage and retrieval, with no dependence on the performance of the external bus. Any load or store, including quad-word

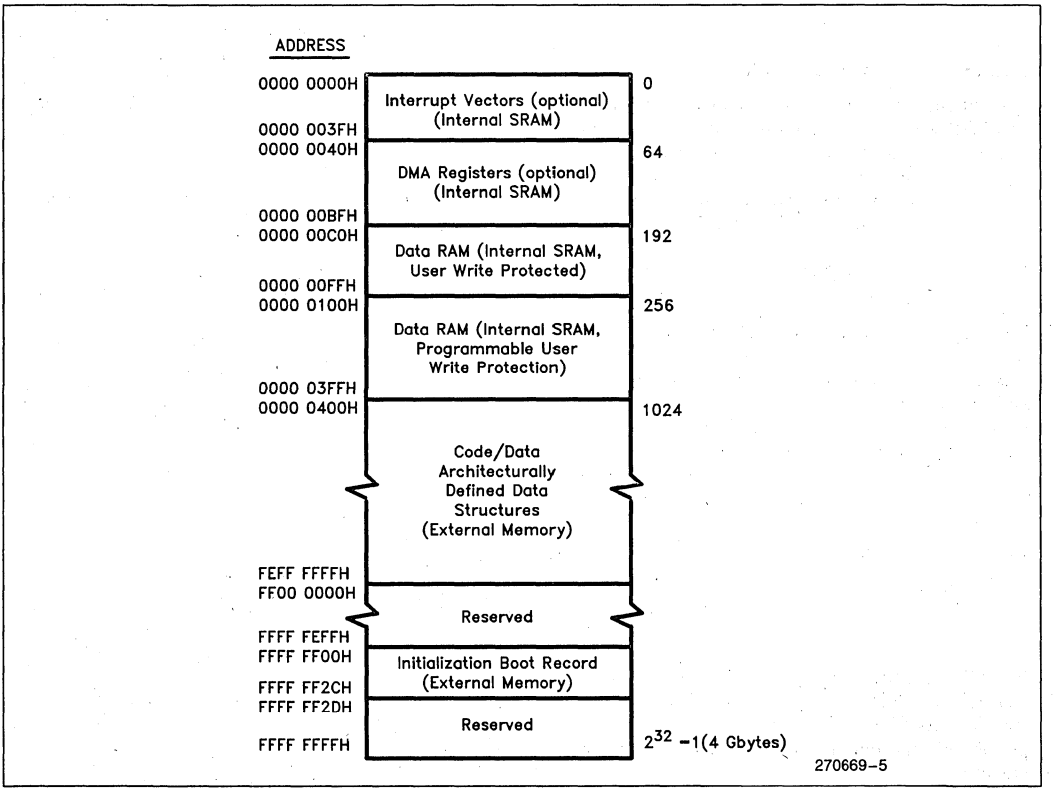


Figure 3-2. Address Space

operations, execute in a single clock cycle when directed to internal data RAM. The data RAM is located at address 00H in the processor's address space. When the DMA controller is in use, 32 bytes of data RAM are reserved for each active DMA channel. Additionally, 64 bytes of data RAM are reserved for 16 interrupt vectors which may be cached internally to reduce interrupt latency. The data RAM reserved for the DMA controller and the interrupt controller can be used for additional data storage when these peripherals are not used.

Two execution modes are possible on the 80960CA, *user mode* or *supervisor mode*. These modes are used to implement a protection model in which system data structures are isolated from user code. As shown in Figure 3-2, the first 256 bytes of data RAM are always write protected when a program is executing in user mode but may always be written when executing in supervisor mode. The remainder of the data RAM can be programmed for this protection feature. The user and supervisor modes are described further in Section 3.7.

3.2.2 RESERVED ADDRESS SPACE

The upper 16 Mbytes of memory (FF000000H–FFFFFFFFH) are reserved for specific functions and extensions to the 80960 architecture. The 12 words in reserved space (FFFFFF00H–FFFFFF2CH) are used to start up the processor when it comes out of reset. These 12 words are called the *initialization boot record*.

3.2.3 ARCHITECTURALLY DEFINED DATA STRUCTURES

To execute a program on the 80960CA, data structures specific to the 80960 architecture must reside in the processor's address space. Architecture-defined data structures include stacks, initialization structures, and various procedure entry tables. These data structures may generally be located anywhere in the address space. Pointers to each data structure are specified when the 80960CA is initialized. The architecture-defined data structures include:

- Interrupt Table
- System-Procedure Table
- Fault Table
- User Stack
- Interrupt Stack
- Supervisor Stack

In addition to the data structure defined by the architecture, the 80960CA requires several implementation-specific data structures which are used for configuring peripherals and initialization. These data structures include:

- Control Table
- Process Control Block
- Initialization Boot Record

Each data structure will be explained in more detail later in this product overview.

3.3 Memory Addressing Modes

The 80960CA offers a variety of modes for memory addressing. The addressing modes available are summarized in Table 3-1.

Absolute addressing is used to reference an address as an offset from address 0 of the processor's address space. At the machine level, absolute addressing may be implemented in one of two ways depending on the size of the absolute offset from address 0. Two instruction formats, MEMA and MEMB, are used to provide absolute addressing modes. For the MEMA format, the offset is an ordinal number ranging from 0 to 2048. For the MEMB format, the offset is an integer (called a displacement) ranging from $-2^{31}-1$ to 2^{31}. An assembler will choose the MEMA or MEMB format based on the size of the offset.

Register-indirect addressing modes use a 32-bit ordinal value in a register as the base for the address calculation. Offsets and indexes are added to this address base depending on the particular addressing mode. The *register-indirect-with-index* addressing mode adds a scaled index to the address base. The index is specified as a value in a register. The scale value may be selected as 1, 2, 4, 8, or 16.

The *index-with-displacement* addressing mode uses a scaled index plus an integer displacement. No address base is used in this address calculation.

The *IP-with-displacement* addressing mode is used with load and store instructions to make them IP relative. In this mode, an integer displacement plus a constant of 8 is added to the IP of the instruction to calculate the next address.

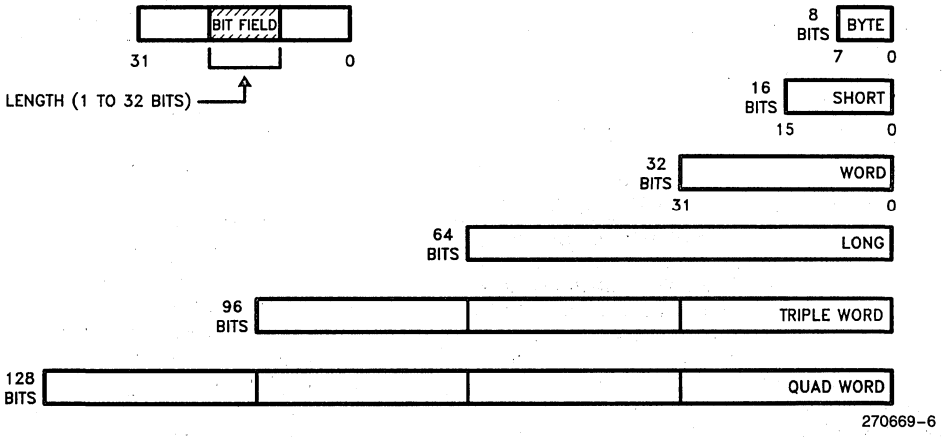
Table 3-1. Memory Addressing Modes

Mode	Description
Absolute Offset	Offset
Absolute Displacement	Displacement
Register Indirect	Abase
Register Indirect with Offset	Abase + Offset
Register Indirect with Index	Abase + (Index*Scale)
Register Indirect with Index and Displacement	Abase + (Index*Scale) + Displacement
Index with Displacement	(Index*Scale) + Displacement
Register Indirect with Displacement	Abase + Displacement
IP with Displacement	IP + Displacement + 8

3.4 Data Types

The 80960CA operates on the following data types (Figure 3-3):

— Integer (8, 16, 32, and 64 bits)
— Ordinal (8, 16, 32, and 64 bits)
— Bit
— Bit Field
— Triple Word (96 bits)
— Quad Word (128 bits)



Class	Data Type	Length	Range
Numeric (Integer)	Byte Integer	8 bits	-2^7 to $2^7 - 1$
	Short Integer	16 bits	-2^{15} to $2^{15} - 1$
	Integer	32 bits	-2^{31} to $2^{31} - 1$
	Long Integer	64 bits	-2^{63} to $2^{63} - 1$
Numeric (Ordinal)	Byte Ordinal	8 bits	0 to $2^8 - 1$
	Short Ordinal	16 bits	0 to $2^{16} - 1$
	Ordinal	32 bits	0 to $2^{32} - 1$
	Long Ordinal	64 bits	0 to $2^{64} - 1$
Non-Numeric	Bit	1-bit	N/A
	Bit Field	1–32 bits	
	Triple Word	96 bits	
	Quad Word	128 bits	

Figure 3-3. Data Types

The following sections describe the data types supported by the 80960CA.

3.4.1 NUMERIC DATA TYPES

Integers and ordinals are considered numeric data types since the processor performs arithmetic operations with this data. The integer data type is a signed binary value in standard 2's complement representation. The ordinal data type is an unsigned binary value.

3.4.2 NON-NUMERIC DATA TYPES

The remaining data types (bit field, triple word, and quad word) represent groupings of bits or bytes that the processor can operate on as a whole, regardless of the nature of the data contained in the group. These data types facilitate the moving of blocks of bits or bytes.

3.5 Instruction Set

The 80960CA features a comprehensive instruction set (Table 3-2). Much of the instruction set is that of a RISC architecture. Unlike pure RISC machines, however, the 80960CA provides an extension to the RISC instruction set with instructions that perform complex functions such as procedure calls and returns, high-speed multiplies, and other complex control, arithmetic, and logical operations. The instruction set allows functionally complex yet highly compact code to be written for embedded control applications where memory is a valuable commodity.

3.5.1 INSTRUCTION GROUPS

The 80960CA instruction set is most easily described if grouped by the functions listed below:

— Data Movement
— Address Computation
— Logical and Arithmetic
— Bit and Bit Field
— Comparison
— Branch
— Call and Return
— Fault
— Debug
— Processor Management

The instructions which make up each of these groups are described in the following sections.

3.5.1.1 Data Movement Instructions

The data movement instructions move data from memory to registers, from registers to memory, and between registers. The load instructions copy bytes, words, or multiple words from memory to a selected register or group of registers. Conversely, the store instructions copy bytes, words, or groups of words from a selected register or group of registers to memory. The move instructions copy data between registers.

Load Instructions

- **ld** load word
- **ldob** load ordinal byte
- **ldos** load ordinal short
- **ldib** load integer byte
- **ldis** load integer short
- **ldl** load long
- **ldt** load triple
- **ldq** load quad

Store Instructions

- **st** store word
- **stob** store ordinal byte
- **stos** store ordinal short
- **stib** store integer byte
- **stis** store integer short
- **stl** store long
- **stt** store triple
- **stq** store quad

Move Instructions

- **mov** move word
- **movl** move long
- **movt** move triple
- **movq** move quad

3.5.1.2 Address Computation Instructions

The load address (**lda**) instruction causes a 32-bit address to be computed and placed in a destination register. The address is computed based on the addressing mode selected. The load and store instructions perform a function identical to that of the **lda** instruction when calculating a source or destination address. The **lda** instruction is useful for loading a 32-bit constant into a register.

3.5.1.3 Logical and Arithmetic Instructions

Logical instructions perform bitwise Boolean operations on operands in registers. Since this group of instructions performs only bitwise manipulations of data, separate logical instructions for integer and ordinal data types do not exist. In the table below, src1 and src2 represent processor registers or literals which are the operands for these instructions.

Table 3-2. Instruction Set Summary

Data Movement	Arithmetic	Logical	Bit and Bit Field
Load Store Move	Add Subtract Multiply Divide Remainder Modulo Shift Extended Shift Extended Multiply Extended Divide Add with Carry Subtract with Carry	And Not And And Not Or Exclusive Or Not Or Or Not Nor Exclusive Nor Not Nand Rotate	Set Bit Clear Bit Not Bit Check Bit Alter Bit Scan for Bit Scan for Byte Span over Bit Extract Modify
Comparison	**Branch**	**Call and Return**	**Fault**
Compare Condition Compare Compare and Increment Compare and Decrement Condition Test	Unconditional Branch Conditional Branch Branch and Link Condition Compare and Conditional Branch	Call Call Extended Call System Return	Conditional Fault Synchronize Faults
Debug	**Processor Management**	**Address Computation**	**Atomic**
Modify Trace Controls Mark Force Mark	Modify Process Controls Modify Arithmetic Controls System Control Update DMA Setup DMA Flush Local Registers	Load Address	Atomic Add Atomic Modify

Logical Instructions

- **and**	src1 and src2
- **notand**	src1 and (not src2)
- **andnot**	(not src1) and src2
- **or**	src1 or src2
- **notor**	src1 or (not src2)
- **ornot**	(not src1) or src2
- **xor**	src1 xor src2
- **xnor**	src1 xnor src2
- **nor**	not (src1 or src2)
- **nand**	not (src1 and src2)
- **not**	not (src1)

Arithmetic instructions perform add, subtract, multiply, divide, and shift operations on integer or ordinal operands in registers.

Arithmetic Instructions

- **addi**	add integer
- **addo**	add ordinal
- **subi**	subtract integer
- **subo**	subtract ordinal
- **muli**	multiply integer
- **mulo**	multiply ordinal
- **divi**	divide integer
- **divo**	divide ordinal
- **remi**	remainder integer
- **remo**	remainder ordinal
- **modi**	modulo integer
- **rotate**	rotate bit left
- **shli**	shift left integer
- **shlo**	shift left ordinal
- **shri**	shift right integer
- **shro**	shift right ordinal
- **shrdi**	shift right dividing integer

Extended arithmetic instructions facilitate computation on ordinals and integers which are longer than 32 bits. In add with carry and subtract with carry instructions, the carry out from the previous arithmetic instruction is used in the computation. The extended multiply instruction multiplies two ordinal source operands producing a long ordinal result (64 bits). The extended divide instruction divides a long ordinal dividend by an ordinal divisor and produces a 64-bit result. The extended shift right instruction shifts a 64-bit source value and produces the lower order 32 bits of the shifted value.

Extended Arithmetic Instructions

- **addc**	add ordinal with carry
- **subc**	subtract ordinal with carry
- **emul**	extended multiply
- **ediv**	extended divide
- **eshro**	shift right extended ordinal

The atomic instructions perform read-modify-write operations on operands in memory. They allow a system to insure that when an atomic operation is performed on a specified memory location, the operation will be completed before another agent is allowed to perform an operation on the same memory. These instructions are required to enable synchronization between interrupt handlers and background tasks in any system. They are also particularly useful in systems where several agents (processors, coprocessors, or external logic) have access to the same system memory for communication.

Atomic Instructions

- **atadd**	atomic add
- **atmod**	atomic modify

3.5.1.4 Bit and Bit Field Instructions

The bit instructions operate on a specified bit in a register.

Bit Instructions

- **setbit**	set bit
- **clrbit**	clear bit
- **notbit**	not bit
- **alterbit**	alter bit
- **scanbit**	scan for bit
- **spanbit**	span over bit

Bit field instructions operate on a specified contiguous group of bits in a register. This group of bits can be from 0 to 32 bits in length.

Bit Field Instructions

- **extract**	extract field
- **modify**	modify field
- **scanbyte**	scan for byte

3.5.1.5 Branch Instructions

The branch instructions allow the direction of program flow to be changed by explicitly modifying the *Instruction Pointer (IP)*. The target IP in a branch instruction is generally specified as a displacement to be added to the current IP. The extended branch instructions allow IP calculation using any addressing mode.

The unconditional branch instructions always alter program flow when executed.

Unconditional Branch Instructions

- **b**	branch
- **bx**	branch extended

The RISC branch-and-link instructions automatically save a Return Instruction Pointer (RIP) before the

jump is taken. The RIP is the address of the instruction following the branch and link.

Branch and Link Instructions

- **bal** branch and link
- **balx** branch and link extended

Conditional branch instructions alter program flow only if the *condition code flags* in the arithmetic control register match a value specified in the instruction. The condition code flags indicate conditions of equality or inequality between two operands in a previously executed instruction. The arithmetic control register and condition code flags are described in Section 3.6.

Based on a *branch prediction flag* located in the machine level instruction, the 80960CA will assume that an instruction usually takes or does not take a conditional branch. By executing along the predicted path of program flow, delays due to breaks in the instruction stream are often avoided. This feature of the 80960CA is referred to as *branch prediction*. The 80960CA incorporates the branch prediction feature because code using a conditional branch instruction usually favors a single direction of program flow.

The branch prediction flag is specified at the assembly level by appending a *.t* or *.f* to a conditional branch instruction meaning, respectively, "assume branch taken" or "assume branch not taken". For example, the assembler mnemonic **be.t** means that the processor will assume that this branch-if-equal instruction usually branches when encountered. In the following table .p represents the branch prediction flag.

Conditional Branch Instructions

- **be.p** branch if equal
- **bne.p** branch if not equal
- **bl.p** branch if less
- **ble.p** branch if less or equal
- **bg.p** branch if greater
- **bge.p** branch if greater or equal
- **bo.p** branch if ordered
- **bno.p** branch if unordered

Compare and conditional branch instructions compare two operands, then branch according to the immediate results.

Conditional Compare and Conditions Branch Instructions

- **cmpibe.p** compare integer and branch if equal
- **cmpibne.p** compare integer and branch if not equal
- **cmpibl.p** compare integer and branch if less
- **cmpible.p** compare integer and branch if less or equal
- **cmpibg.p** compare integer and branch if greater
- **cmpibge.p** compare integer and branch if greater or equal
- **cmpibo.p** compare integer and branch if ordered
- **cmpibno.p** compare integer and branch if unordered
- **cmpobe.p** compare ordinal and branch if equal
- **cmpobne.p** compare ordinal and branch if not equal
- **cmpobl.p** compare ordinal and branch if less
- **cmpoble.p** compare ordinal and branch if less or equal
- **cmpobg.p** compare ordinal and branch if greater
- **cmpobge.p** compare ordinal and branch if greater or equal
- **bbs.p** check bit and branch if set
- **bbc.p** check bit and branch if clear

3.5.1.6 Compare and Condition Test Instructions

The 80960CA provides several types of instructions that are used to compare two operands. The condition code flags in the arithmetic control register are set to indicate whether one operand is less than, equal to, or greater than the other operand.

Compare Instructions

- **cmpi** compare integer
- **cmpo** compare ordinal
- **chkbit** check bit

Conditional compare instructions test the existing status of the condition code flags before a compare is

performed. These conditional compare instructions are provided to optimize two-sided range comparisons (i.e. to test if a value is less than one number but greater than another).

Conditional Compare Instructions

- **concmpi** conditional compare integer
- **concmpo** conditional compare ordinal

The compare and increment and compare and decrement instructions set the condition code flags based on a comparison of two register sources, decrements or increments one of the sources, and finally stores this result in a destination register.

- **cmpinci** compare and increment integer
- **cmpinco** compare and increment ordinal
- **cmpdeci** compare and decrement integer
- **cmpdeco** compare and decrement ordinal

The condition test instructions allow the state of the condition code flags to be tested. Based on the outcome of the comparison, a true or false code is stored in a destination register. The branch prediction flag is used in this instruction to reduce the execution time of the instruction when the test outcome is predicted correctly. For example **teste.t** (test if equal) will execute in a shorter time if the condition code flags test true for the equal condition. Analogous to the function of the branch prediction flag in the conditional compare and branch instructions, the prediction flag in this case eliminates breaks in the micro-instruction sequence which is used to implement the condition test instructions.

Condition Test Instructions

- **teste.p** test if equal
- **testne.p** test if not equal
- **testl.p** test if less
- **testle.p** test if less or equal
- **testg.p** test if greater
- **testge.p** test if greater or equal
- **testo.p** test if ordered
- **testno.p** test if not ordered

3.5.1.7 Call and Return Instructions

The 80960CA features an on-chip call and return mechanism for making procedure calls to local and system procedures. The call instructions and the call and return mechanism is described in Section 3.8.

Call and Return Instructions

- **call** call
- **callx** call extended
- **calls** call system
- **ret** return

3.5.1.8 Fault Instructions

The 80960CA will fault automatically as the result of certain errant operations which may occur when executing code. Fault procedures are then invoked automatically to handle the various types of faults. In addition, the fault instructions permit a fault to be generated explicitly based on the value of the condition code flags. The branch prediction flag in these instructions is used to reduce the execution time of these instructions when the state of the condition code flags are guessed correctly.

Conditional Fault Instructions

- **faulte.p** fault if equal
- **faultne.p** fault if not equal
- **faultl.p** fault if less
- **faultle.p** fault if less or equal
- **faultg.p** fault if greater
- **faultge.p** fault if greater or equal
- **faulto.p** fault if ordered
- **faultno.p** fault if unordered

The **syncf** instruction causes the processor to wait for all faults to be generated which are associated with any prior uncompleted instructions.

- **syncf** synchronize faults

3.5.1.9 Debug Instructions

The processor supports debugging and monitoring of program activity through the use of trace events. The debug instructions support debugging and monitoring software.

Debug Instructions

- **modtc** modify trace controls
- **mark** mark
- **fmark** force mark

3.5.1.10 Processor Management Instructions

The 80960CA provides several instructions for direct control of processor functions and for configuring the 80960CA's peripherals. A brief description of the processor management instructions is given below.

Processor Management Instructions

- **modpc** modify process controls
- **modac** modify arithmetic controls
- **sysctl** system control instruction
- **udma** update DMA SRAM
- **sdma** setup DMA
- **flushreg** flush local registers

3.6 Arithmetic Controls

The *Arithmetic Control (AC) Register* is a 32-bit on-chip register (Figure 3-4). The AC register is used primarily to monitor and control the execution of 80960CA arithmetic instructions. The processor reads and modifies bits in the AC register when performing many arithmetic operations. The AC register is also used to control the faulting conditions for some instructions. The **modac** instruction allows the user to directly read or modify the AC register.

The processor sets the condition code flags (bits 0–2) to indicate equality or inequality as the result of certain instructions (such as the compare instructions). Other instructions, such as the conditional branch instructions, take action based on the value of the condition code flags. Table 3-3 shows the functional assignment for each condition code flag.

Table 3-3. Arithmetic Condition Codes

Condition Code	Condition
001	Greater Than
010	Equal
100	Less Than

The integer overflow flag (bit 8) and the integer overflow mask (bit 12) are used in conjunction with the arithmetic integer overflow fault. The mask bit masks the integer overflow fault. When the fault is masked, and an integer overflow occurs, the integer overflow flag is set but no fault handling action is taken. If the fault is not masked, and an integer overflow occurs, the integer overflow fault is taken and the integer overflow flag is not set.

The no imprecise faults flag (bit 15) determines if imprecise faults are allowed to occur. Fault handling and precise and imprecise faults in the 80960CA are discussed in Section 3.10.

3.7 Process Management

Process management refers to the monitoring and control of certain properties of an executing process. The following sections describe the mechanisms available on the 80960CA to perform this function.

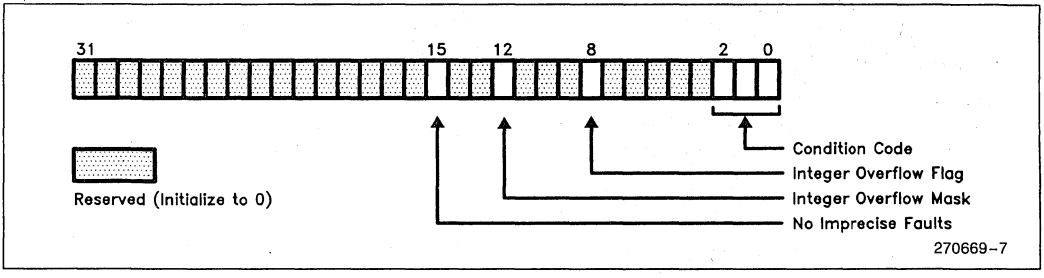


Figure 3-4. Arithmetic Control Register

3.7.1 PROCESS CONTROL REGISTER

The *Process Control (PC) Register* (Figure 3-5) provides access to process state information. The function for the PC register is described below.

Execution Mode Flag—This flag indicates that the processor is executing in user mode (0) or supervisor mode (1).

Priority Field—This 5-bit field indicates the current executing priority of the processor. Priority values range from 0 to 31, with 0 as the lowest and 31 as the highest priority.

State Flag—This flag determines the executing state of the processor. The processor state is either executing state (0) or interrupted state (1).

Trace Enable Bit and Trace Fault Pending Flags—These fields control and monitor trace activity in the processor. The Trace Enable Bit enables fault generation for trace events. The Trace Fault Pending Flag indicates that a trace event has been detected.

The process controls can be modified by software with the modify process controls (**modpc**) instruction. The **modpc** instruction may only write the PC register when the processor is in supervisor mode.

3.7.2 PRIORITIES

The 80960 architecture defines a means to assign priorities to executing programs and interrupts. The current priority of the processor is stored in the priority field of the PC register. This priority is used to determine if an interrupt will be serviced and in which order multiple pending interrupts will be serviced. Setting the priority of an executing program above that of interrupts allows critical code to be prioritized and executed without interruption.

The priority field of the PC register can be modified directly using the **modpc** instruction. The priority field is also modified to reflect the priority of serviced interrupts. On a return from an interrupt routine, the priority of the processor is restored to its priority before the interrupt occurred.

3.7.3 PROCESSOR STATES AND MODES

The 80960CA may execute programs in *user mode* or *supervisor mode*. The user-supervisor protection mechanism allows a system to be designed in which kernel code and data reside in the same address space as user code and data, but access to the kernel procedures and data is only allowed through a tightly controlled interface. This interface is the system call table and the interrupt mechanism. The 80960CA provides a supervisor pin ($\overline{SUP}$) to implement memory systems which protect code and data from possible corruption by programs executing in user mode. Some instructions and functions of the 80960CA are also insulated from code executing in user mode.

The processor has two operating states: executing and interrupted. In executing state, the processor can execute in user or supervisor mode. In the interrupted state, the processor always executes in supervisor mode.

3.8 Call and Return Mechanism

The 80960 architecture features a built-in call and return mechanism. This mechanism is designed to make procedure calls simple and fast, and to provide a flexible method for storing and handling variables that are local to a procedure. A call automatically allocates a new set of local registers and a new stack frame. All linkage information is maintained by the processor, making procedure calls and returns virtually transparent to the user. A system call instruction is provided as a method for calling privileged procedures such as a kernel service. The call and return model supports efficient translation of structured high level code (such as C, or ADA) to 80960 machine language.

The procedure call and return mechanism provides a number of significant benefits which contribute to the performance and ease of use of the 80960CA.

1) The call and return instructions are implemented entirely on-chip, resulting in an extremely high performance implementation of these commonly used functions.

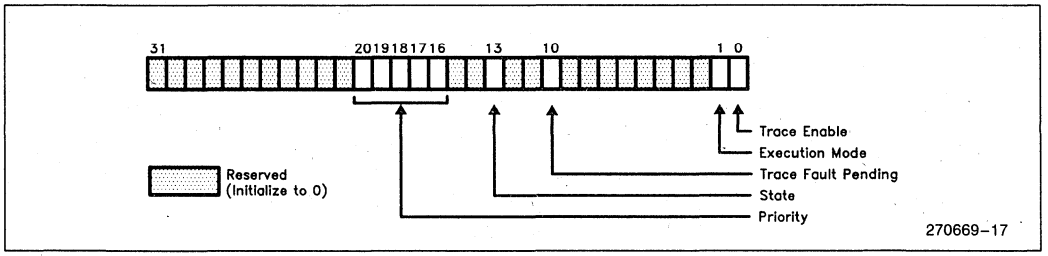


Figure 3-5. Process Control Register

2) A single instruction to implement each call or return operation results in code density improvements compared to processors which require multiple instructions to encode these functions.
3) By implementing the call and return functions as single instructions, the 80960 architecture is open for further optimization of these instructions, while maintaining assembly-level compatibility.
4) A program does not have to explicitly save or restore the variables stored in the local registers when a call or return is executed. The processor does this implicitly on procedure calls and on returns.
5) The call and return mechanism provides a structure for storing a virtually unlimited number of local variables for each procedure: the on-chip local registers provide quick access to often used variables and the stack provides space for additional variables.

3.8.1 LOCAL REGISTERS AND THE STACK FRAME

At any point in a program, the 80960 has access to a local register set and a section of the procedure stack referred to as a stack frame. When a call is executed, a new stack frame is allocated for the called procedure. Additionally, the current local register set is saved by the processor, freeing these registers for use by the newly called procedure. In this way, every procedure has a unique stack and unique set of local registers. When a return is executed, the current local register set and current stack frame are deallocated. The previous local register set and previous stack frame are restored. This call and return mechanism is illustrated in Figure 3-6 where n is procedure depth for the currently executing procedure.

The procedure stack structure is defined by the 80960 architecture. The procedure stack always grows upward (i.e. towards higher addresses) and the stack pointer (SP) always points to the next available byte of the stack frame. The 80960CA requires that each stack frame begins on a 16-byte boundary. Due to this alignment requirement, a padding space of 0 to 15 bytes may exist between adjacent stack frames in memory. When a stack frame is allocated, the first 16 words are always assigned as storage for the local registers; therefore, the SP initially points to the 17th word in the stack frame. It should be noted that although each stack frame is assigned storage space for the local registers, these locations in the stack are not guaranteed to contain the values of the saved local registers. This is because several sets of local registers are cached on-chip rather than written to the stack in external memory. This caching mechanism is described in detail later in this section.

3.8.2 PROCEDURE LINKING

The 80960 architecture automatically manages procedure linkage. One global register and three local registers are reserved for procedure linkage information.

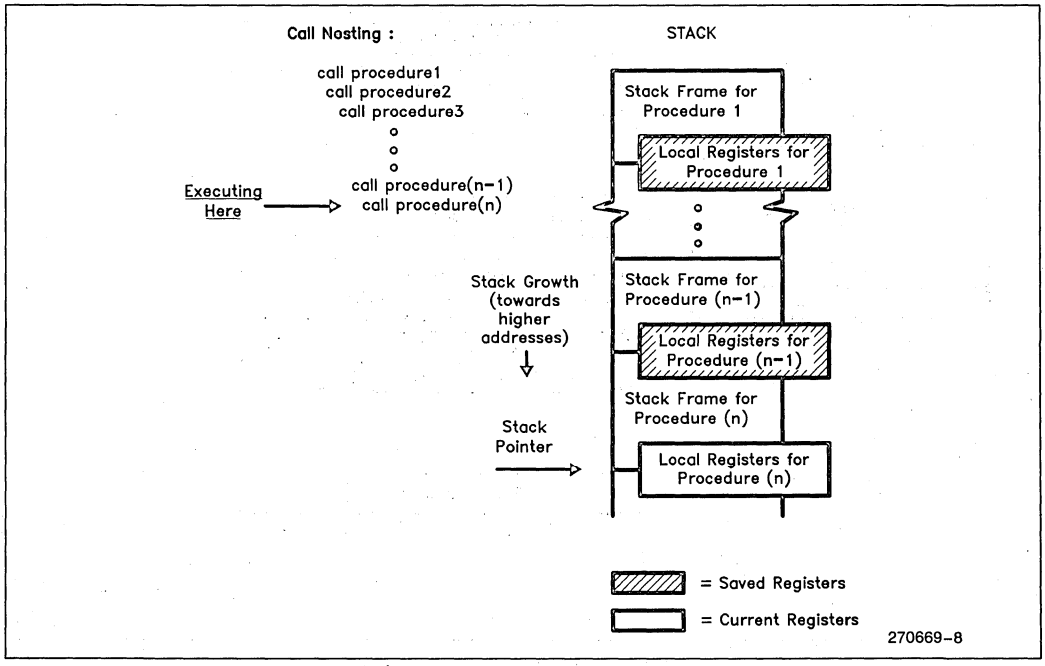


Figure 3-6. Call and Return Mechanism

Figure 3-7 describes the pointer structure used to link frames and to provide a unique SP for each frame. Register **g15** is the *Frame Pointer (FP)*. The FP is the address of the first byte of the current (topmost) stack frame. The FP is always updated to point to the current frame when calls and returns are executed. Register **r0** is the *Previous Frame Pointer (PFP)*. The PFP is the address of the first byte of the stack frame which was created prior to the frame containing this PFP. Register **r1** is the *Stack Pointer (SP)*. The SP points to the next available byte of the stack frame. Register **r2** is reserved for the *Return Instruction Pointer (RIP)*. The RIP is the address of the instruction which follows a call instruction, this is also the target address for the return from that procedure. The RIP is automatically stored in register **r2** of the calling procedure when a call is executed.

3.8.3 PARAMETER PASSING

Parameters may be passed by value or passed by reference between procedures. The global registers, the stack, or predefined data structures in memory may be used to pass these parameters.

The global registers provide the fastest method for passing parameters. The values to be passed into a procedure reside in the global registers of the calling procedure. When a procedure is called, the values in the global registers are preserved. If more parameters are to be passed than will fit in the global registers, additional parameters may be passed in the stack of the calling procedure, or in a data structure which is referenced by a pointer passed in the global registers.

3.8.4 LOCAL REGISTER CACHE

The 80960CA provides an on-chip cache for saving and restoring the local registers on calls and returns. This cache greatly enhances performance of the call and return mechanism on the 80960CA. Movement of data between the local registers and the register cache is typically accomplished in only 4 processor clocks with no external bus traffic. When this cache is filled, the registers associated with the oldest stack frame are moved to the area reserved for those registers on the physical stack (Figure 3-7).

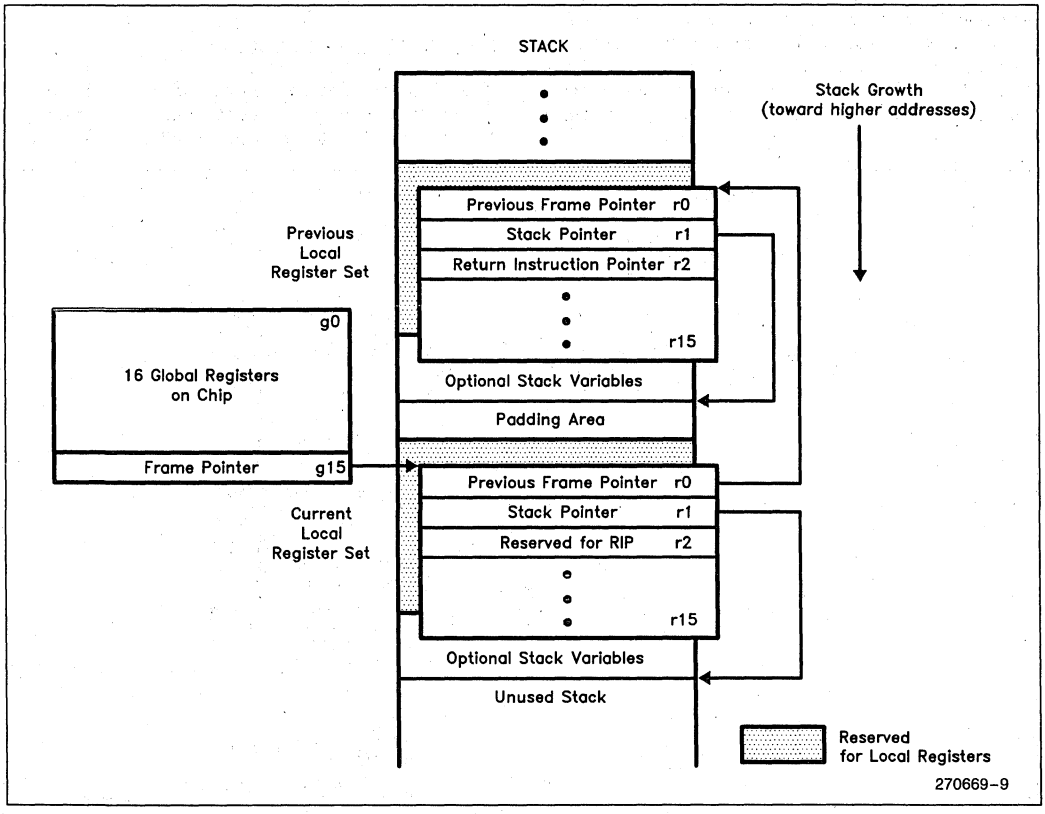


Figure 3-7. Stack Frame Linkage

The local register cache is a physical extension of the internal data RAM. The part of the data RAM used for this cache is not visible to the user and is large enough to hold up to 5 sets of local registers. The register cache may be extended to hold up to 15 sets of local registers. When extended, each new register set consumes 16 words of the user's data RAM, beginning at the highest address and growing downward. The size of the local register cache is selected when the processor is initialized.

In some cases, the contents of the cached local register sets may require examination or modification (e.g. for fault handling). Since the local registers are cached, the **flushreg** instruction is provided to flush the local register cache to the locations reserved for the registers on the stack. This insures that the values in external memory are consistent with the values held in the local register cache.

3.8.5 LOCAL AND SYSTEM CALLS

The 80960CA provides two methods for making procedure calls: local calls and system calls. Local and system calls differ in their operation and use in an application.

The local call instructions initiate a procedure call using the call and return mechanism described earlier. The stack frames for these procedure calls are allocated on the *local procedure stack*. A local call is made using either of two local call instructions: **call** or **callx**. The **call** instruction specifies the address of the called procedure using an IP plus displacement addressing mode with a range of -2^{23} to $2^{23}-4$ bytes from the current IP. The **callx** (call extended) instruction specifies the address of the calling procedure using any of the 80960's addressing modes.

A system call is made using the **calls** instruction. This call is similar to a local call except that the processor gets the IP for the called procedure from a data structure called the *system procedure table*. The **calls** instruction requires a procedure number operand. This procedure number serves as an index into the system procedure table, which contains IP's for specific procedures. The system procedure table is shown in Figure 3-8.

The system call mechanism supports two types of procedure calls: system-local calls and system-supervisor calls (also referred to as supervisor calls). The system-

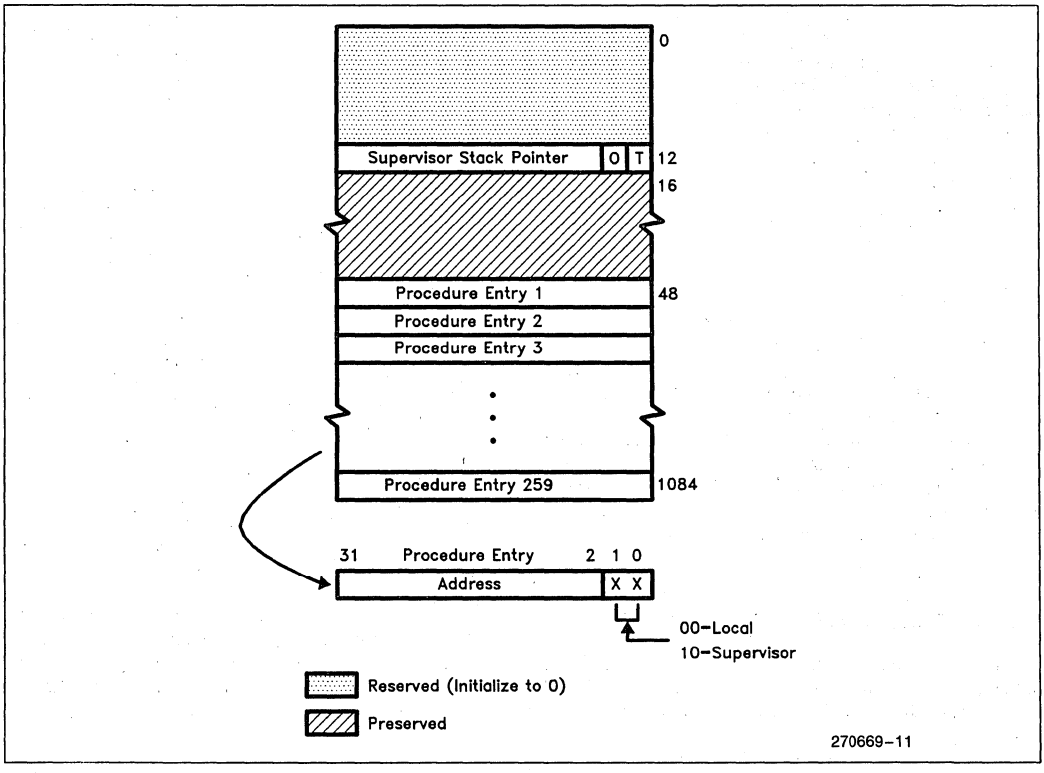


Figure 3-8. System Procedure Table

local call performs the same action as the local call instructions with one exception: the IP target for a system-local call is fetched from the system-procedure table. The supervisor call differs from the local call as follows:

1) A supervisor call causes the processor to switch to another stack (called the supervisor stack).
2) A supervisor call causes the processor to switch to the supervisor execution mode and asserts the 80960CA's supervisor ($\overline{SUP}$) pin for all bus accesses.

The system call mechanism offers several benefits. The system call promotes the portability of application software. System calls are commonly used for kernel services. By calling these services with a procedure number rather than a specific IP, application software does not have to be changed each time the implementation of the kernel service is modified. Additionally, the ability to switch to a different execution mode and stack allows kernel procedures and data to be insulated from application code.

3.8.6 IMPLICIT PROCEDURE CALLS

The call and return mechanism described for procedure calls applies to several classes of call instructions as well as to the context switching initiated by interrupts and faults. When an interrupt or fault condition occurs, an implicit call is performed that saves the current state of the processor before branching to the interrupt or fault handling procedure. When this context switch occurs, the local registers are saved and a new stack frame is allocated. Additionally, the values of the AC register and PC register are saved when the implicit call occurs. These values are restored on the return from the interrupt or fault handler.

3.9 Interrupts

An interrupt is a temporary break in the control stream of a program so that the processor can handle another task. Interrupts may be triggered by the instruction stream or by hardware sources internal and external to the 80960CA. An interrupt request is associated with a vector (i.e. an address) of an interrupt handling procedure. The processor will branch to the handling procedure when an interrupt is serviced. When the handling action is completed, the processor is restored to its state prior to the interrupt.

3.9.1 INTERRUPT VECTORS AND PRIORITY

Interrupt vectors are simply instruction pointers (addresses) to interrupt handling procedures. The 80960 architecture defines 248 interrupt vectors. This means that 248 unique interrupt handling procedures may be used. An 8-bit interrupt vector number is associated with each interrupt vector. This number ranges from 8 to 255. Each interrupt vector has a priority from 1 to 31, which is determined by the 5 most significant bits of the interrupt vector number. Priority 1 is the lowest priority and 31 is the highest. Priority 0 interrupts are not defined.

The 80960CA executes with a unique priority ranging from 0 to 31. When an interrupt is serviced, the processor's priority switches to the priority corresponding to that of the interrupt request. When a return from an interrupt procedure is executed, the process priority is restored to its value prior to servicing the interrupt. This priority switching is handled automatically by the 80960CA.

The 80960CA compares its current priority and the priority of an interrupt request to determine whether to service an interrupt immediately or to delay service. If a requested interrupt priority is greater than the processor's current priority or equal to 31, the processor services the interrupt immediately; otherwise, the processor saves (posts) the interrupt request as a pending interrupt so that it can be serviced later. When the processor's priority falls below the priority of a pending interrupt, the pending interrupt is serviced. With the mechanism described, interrupts with a priority of 0 will never be serviced. For this reason, vectors numbered 0 to 7 are not defined.

3.9.2 INTERRUPT TABLE

The interrupt table (Figure 3-9) is an architecturally defined data structure which holds the interrupt vectors and information on pending interrupts. The first 36 bytes of the table are used to post interrupts. The 31 most significant bits in the 32-bit pending priorities field represent a possible priority (1 to 31) of a pending interrupt. When the processor posts an interrupt in the interrupt table, the bit corresponding to the interrupt's priority is set. For example, if an interrupt with a priority of 10 is posted in the interrupt table, bit 10 is set in the pending priorities field.

The pending interrupts field contains a 256-bit string in which each bit represents an interrupt vector. When the processor posts an interrupt in the interrupt table, the bit corresponding to the vector number of that interrupt is set.

Portions of the interrupt table are cached on-chip in a non-transparent fashion. This caching is implemented to minimized interrupt latency by reducing the number of accesses to the table in external memory when an interrupt is serviced.

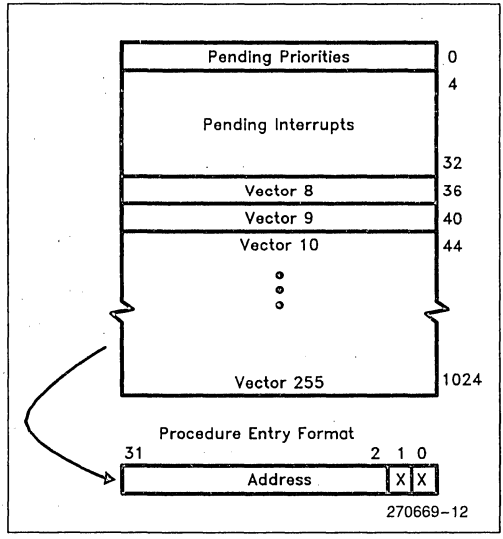


Figure 3-9. Interrupt Table

3.9.3 INTERRUPT STACK

Stack frames for interrupt handling procedures are allocated on a separate *interrupt stack*. The interrupt stack can be located anywhere in the processor's address space. The beginning address of the interrupt stack is specified when the processor is initialized.

3.9.4 INTERRUPT HANDLING ACTION

When an interrupt is serviced, the processor saves the processor state and calls the interrupt procedure. The processor state is restored upon return from the interrupt procedure.

This interrupt service mechanism is handled by an implicit call operation. When the interrupt is serviced, the current local registers are saved. A new local register set and stack frame are allocated on the interrupt stack for the interrupt handler procedure and the processor switches to supervisor execution mode. In addition to the local registers, the current value of the AC and PC registers are saved as an interrupt record on the interrupt stack.

3.9.5 PENDING INTERRUPTS

Any of the 248 interrupts can be requested by software. The system control instruction (**sysctl**) is provided to support this feature. When the system control instruction requests an interrupt, one of two actions may occur depending on the priority of the requested interrupt and the current process priority. 1) The interrupt is serviced immediately, or 2) the interrupt is posted (the pending priorities field and the pending interrupts field are modified to reflect a pending interrupt).

Interrupts may also be requested by hardware sources internal and external to the 80960CA. Managing the hardware sources and posting these interrupts is handled by the interrupt controller. Interrupts requested by hardware are posted in an internal register, not in the interrupt table. A mask register enables or disables interrupts from each hardware source. Requesting and posting hardware interrupts is described in Section 4.4 Interrupt Controller.

3.9.6 INTERRUPT LATENCY

The time required to perform an interrupt task switch is referred to as the *interrupt latency*. The latency is the time measured between the activation of an interrupt source and the execution of the first instruction for the interrupt-handling procedure for the source.

Interrupt latency for the 80960CA varies depending on conditions such as:

— Complex instructions are executing when the interrupt occurs (e.g. **sysctl, call, ret,** etc.).

— Outstanding loads to a local register are pending, delaying the interrupt context switch.

— Division, multiplication, or other multi-cycle instructions with a local register as destination are executing.

The 80960CA has been designed to optimize latency and throughput for interrupts. Two processor features are designed for this purpose:

First, in the interrupt table, all interrupt vectors with an index whose least significant four bits are 0010_2 can be cached in internal data RAM. The processor will automatically read these vectors from data RAM when the interrupt is serviced. This feature reduces the added latency due to an external access of the interrupt table for that vector. The NMI vector is always cached in data RAM.

Second, an instruction cache locking mechanism allows interrupt procedures or segments of interrupt procedures to be stored in the instruction cache. These routines are always executed from the internal cache, eliminating external code fetches and reducing latency and increasing throughput for the interrupt.

3.10 Fault Handling and Instruction Tracing

The 80960CA is able to detect various conditions in code or in its internal state that could cause the processor to deliver incorrect or inappropriate results or that could cause it to head down an undesirable control path. These conditions are referred to as faults. The 80960 architecture provides fault handling mechanisms to detect and, in most cases, fully recover from a fault.

The 80960CA provides on-chip debug support by triggering trace events and servicing the trace fault. A trace event is activated when a particular instruction or type of instruction is encountered in an instruction stream. The trace event optionally signals a fault. A fault handling procedure for the trace fault can act as a debug monitor and analyze the state of the processor when the trace event occurred.

3.10.1 FAULT TYPES AND SUBTYPES

All of the faults that the processor detects are predefined. These faults are divided into types and subtypes, each of which is given a number. Table 3-4 lists the faults that the processor detects arranged by type and subtype.

Table 3-4. Fault Types and Subtypes

Fault Type	Fault Subtype	Fault Record
Parallel		XX00 00XX
Trace	Instruction Type	XX01 0002
	Branch Trace	XX01 0004
	Call Trace	XX01 0008
	Return Trace	XX01 0010
	Prereturn Trace	XX01 0020
	Supervisor Trace	XX01 0040
	Breakpoint Trace	XX01 0080
Operation	Invalid Opcode	XX02 0001
	Unimplemented	X002 0002
	Invalid Operand	XX02 0004
Arithmetic	Integer Overflow	XX03 0001
	Arithmetic Zero-Divide	XX03 0002
Constraint	Range	XX05 0001
	Privileged	XX05 0002
Protection	Length	XX07 0001
Type	Mismatch	XX0A 0001

NOTE: X refers to preserved locations in the fault record.

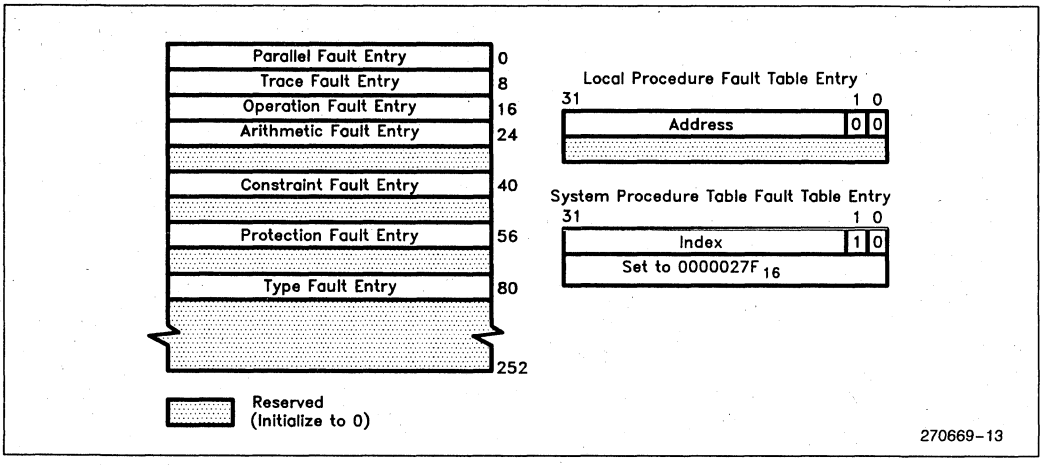


Figure 3-10. Fault Table

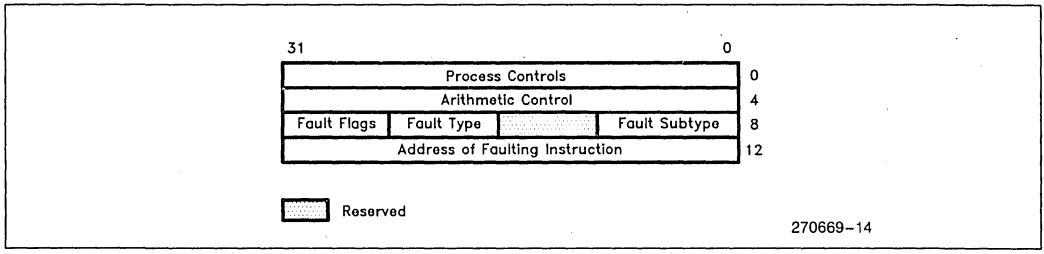


Figure 3-11. Fault Record

3.10.2 FAULT TABLE

The fault table (Figure 3-10) provides the processor with a pathway to fault handling procedures. The fault table is an architecture-defined data structure, which may be located anywhere in the processor's address space. The location of the fault table is specified at initialization. When a fault occurs, an entry in the table is selected based on the type of fault that occurs. The entry in the fault table contains a pointer to a specific fault handler.

The fault table can contain two types of entries (Figure 3-10). The first type of entry is simply a pointer to the address of the fault-handling procedure. The second type of entry is an index into the system-procedure table. Fault-handling procedures accessed through the system-procedure table may be executed in user or supervisor execution mode.

3.10.3 FAULT HANDLING ACTION

When a fault occurs, the processor performs an implicit call operation to the procedure specified in the fault table. In addition to performing the implicit call operation, the processor creates a fault record in its newly allocated stack frame. This fault record contains information on the state of the processor when the fault occurred and the fault type and subtype (Figure 3-11).

Some faults can be recovered from easily. When recovery from a fault is possible, the processor's fault handling mechanism allows the processor to automatically resume work where the fault was signalled. The resumption action is initiated with the **ret** instruction. If simple recovery from a fault is not possible, then the fault handling procedure may call a debug monitor, initiate a reset, or take other actions to recover from the fault.

3.10.4 TRACING AND DEBUG

The 80960CA provides a facility for monitoring the activity of the processor by tracing the instruction stream. A trace event occurs at points in a program where certain types of instructions are encountered or a certain IP or data address is encountered. When a trace event occurs, a trace fault can be generated and a trace-fault handler called which displays or analyzes the state of the processor.

3.10.4.1 Trace Events

The *Trace Control (TC) Register* (Figure 3-12) is used to specify the types of instructions which cause trace events. When a mode bit in the TC register is set, specific instructions will generate trace events. For example, if the branch trace mode bit is enabled and a branch instruction is executed, a branch trace event will be signalled. An event flag is used to record trace events. A single event flag is provided for each mode bit. Any trace event generates a trace fault when the trace enable bit in the process control register is set.

The 80960CA recognizes 7 trace events. These events are described below.

Instruction Trace Event—Signalled each time an instruction is executed. This trace event can be used with a debug monitor to single step the processor.

Branch Trace Event—Signalled each time a branch instruction is executed. For conditional branch instructions, this event is only signalled when the branch is taken. Branch-and-link, call, and return instructions do not signal this trace event.

Call Trace Event—Signalled each time a branch-and-link or call instruction is executed. Implicit calls, such as those used in interrupt or fault handling, signal this event. When a call trace event occurs, the prereturn trace flag (bit 3 in local register **r0**) is set by the processor to indicate a prereturn trace pending.

Pre-Return Trace Event—Signalled just prior to any **ret** instruction. This event is only signalled if the pre-return trace flag in register **r0** is set. Since the pre-return trace flag is set when a call trace event occurs, the call trace mode must be enabled before a pre-return trace event can be signalled.

Return Trace Event—Signalled each time a **ret** instruction is executed.

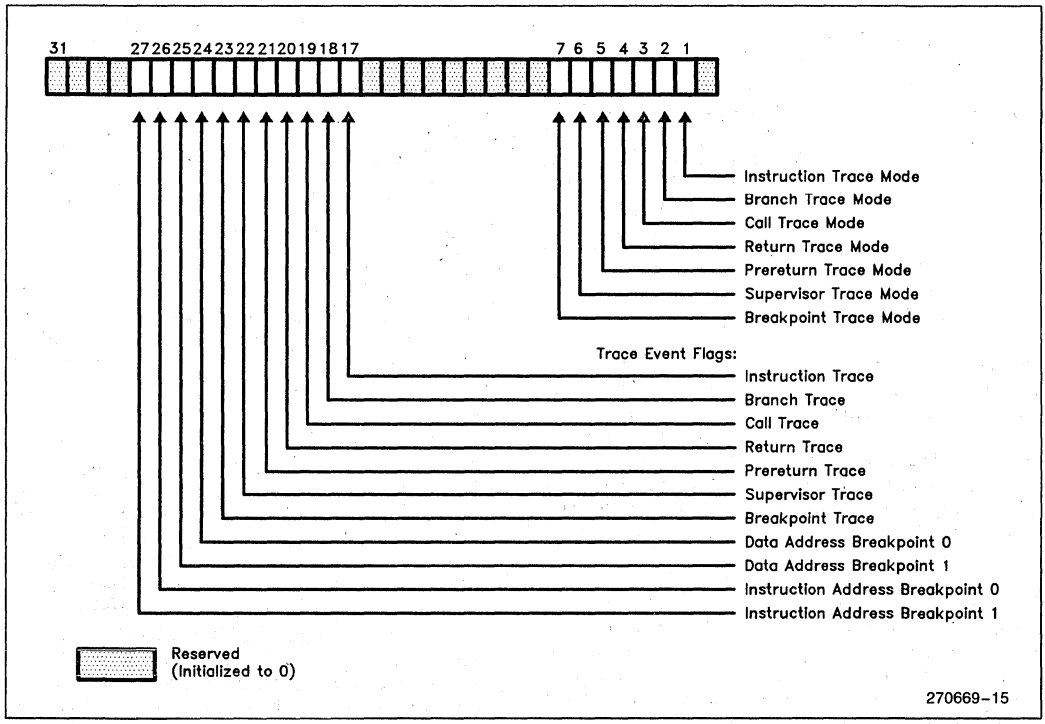


Figure 3-12. Trace Control Register

Supervisor Trace Event—Signalled each time a **calls** instruction is executed where the selected entry type is supervisor, or when a **ret** from supervisor mode is executed.

Breakpoint Trace Events—Signalled each time a **mark** instruction, **fmark** instruction, or specified address is encountered in the instruction stream. The **mark** instruction signals an event when the breakpoint trace mode is enabled, the **fmark** (force mark) instruction will generate a breakpoint trace event regardless of the value of the breakpoint trace mode bit.

Two IP breakpoint registers and two internal data address breakpoint registers are provided on the 80960CA. These breakpoints are loaded with an instruction or data address using the system control (**sysctl**) instruction. When the address is encountered and the breakpoint trace mode bit is set, a breakpoint trace event occurs. A corresponding instruction or data address event flag is set in the TC register when the address is encountered.

3.10.5 PROCESSOR INITIALIZATION

The Initial Memory Image (IMI) are the data structures needed to initialize the 80960CA (Figure 3-13). The initialization boot record, in reserved memory beginning at FFFFFF00H, contains a pointer to the Processor Control Block (PRCB). The PRCB in turn holds pointers to the data structures which are necessary to execute code on the 80960CA. The PRCB also holds several fields which contain information to initially configure the 80960CA.

Processor initialization begins by asserting the $\overline{\text{RESET}}$ pin. At initialization the processor optionally performs an internal self-test. A bus confidence test is also performed by calculating a checksum of 8 words read from external memory. If either of these self-tests fails, the $\overline{\text{FAIL}}$ pin indicates the failure and the processor aborts initialization. If the self-test passes, the 80960CA continues with initialization and branches to the first address of the user's code.

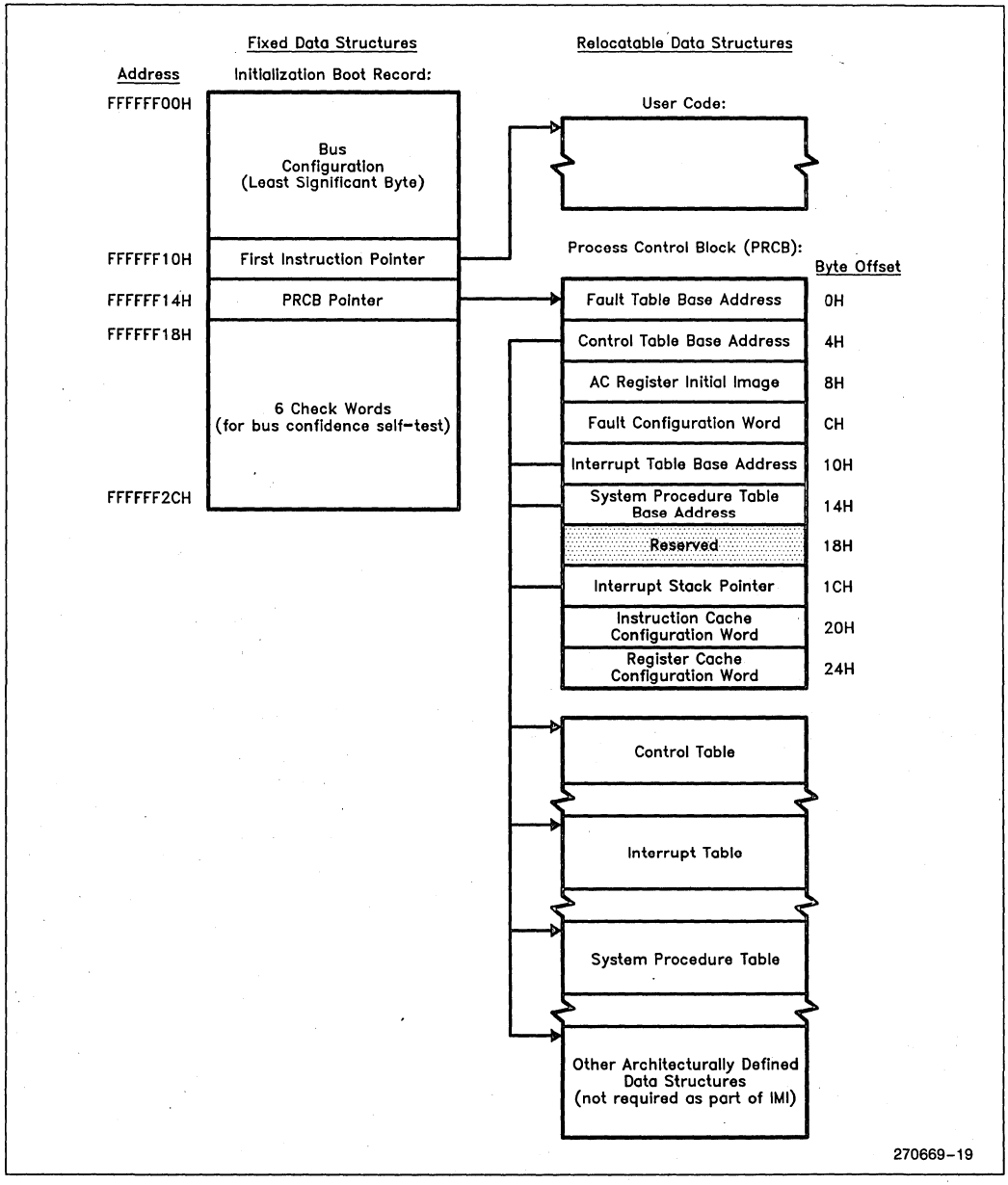


Figure 3-13. Initial Memory Image

4.0 80960CA SYSTEM IMPLEMENTATION

This section is an overview of the peripherals integrated with the 80960CA core. The features and operation of the Bus Controller, DMA Controller, Interrupt Controller, and the interfaces between these peripherals and the core are described.

4.1 Peripheral Interface

A program communicates with the on-chip peripherals by reading or modifying the special function registers (SFRs) or by loading control registers. The SFRs generally serve to transfer status information and data between a peripheral and the core, and the control registers serve to configure the peripherals. SFRs are accessed directly as instruction operands. The control registers are loaded by using the system control (**sysctl**) instruction.

4.2 Bus Controller Unit

The Bus Controller Unit (BCU) manages the data and instruction path between the 80960CA and external memory. Data operations and instruction fetches share a 32-bit data bus. Memory addresses are output on a separate 32-bit address bus. The BCU incorporates several advanced features to simplify the bus interface to external memory. A programmable *memory region configuration table* allows the characteristics of the external bus to be programmed differently for 16 separate regions in memory. The attributes of the external bus which are programmable include wait states and external ready control, data bus width (8, 16, or 32 bits), burst mode, address pipelining, and byte ordering. The region programmable bus options are described in this section.

4.2.1 BUS TRANSFERS, ACCESSES, AND REQUESTS

The distinction between *transfer, bus access,* and *bus request,* as these terms apply to the 80960CA, must be presented before beginning a discussion of the BCU.

Transfer—A *bus transfer* is defined simply as a movement of code or data between a memory system and the 80960CA. A write transfer occurs when the memory system is the destination of a data movement. A read transfer occurs when the 80960CA is the destination for a data or a code fetch from memory.

Bus Access—A *bus access* is defined as an address cycle and one or more transfers. In burst mode, an access can consist of a single address cycle and 1 to 4 transfers.

Bus Request—A *bus request* is issued by the core and directed to the Bus Controller. A bus request is sent to the BCU when a load, store, or an atomic instruction is executed, or when an instruction fetch is needed. Bus requests are also issued by the core to perform DMA transfers. A bus request can consist of one or more bus accesses. For example, an aligned word (32-bit) request to an 8-bit memory region will result in four byte-length accesses.

4.2.2 BUS CONTROL COPROCESSOR

The 80960CA's peripherals are often referred to as coprocessors, since their operation is decoupled from the execution of the instruction stream. As an integrated coprocessor, the BCU receives bus requests and independently carries out the action of moving data or code between the processor and external memory. The BCU uses a three deep queue to store pending bus requests. The queue decouples the core from the BCU, since a series of adjacent requests may be issued faster than the BCU can service each request. Two of the three queue entries store requests from a user's program (loads, stores, fetches, etc.). The third queue entry is used by requests originating from a DMA operation. This queue entry takes user requests when the DMA is turned off. The 80960CA alternates service of requests issued by the user program and requests issued by a DMA operation.

4.2.3 SIGNAL DESCRIPTIONS

The external bus signals consist of 30 address signals, 4 byte enables, 32 data lines, and various control signals.

D31–D0 32-bit Data Bus (bi-directional)—32-, 16-, and 8-bit values are transmitted and received on these lines. The 8- and 16-bit quantities are transferred on the low order data lines when a memory region is configured respectively for an 8- or 16-bit bus.

A31–A2 30-bit Address (outputs)—The 30-bit address bus identifies all external addresses to word (4-byte) boundaries. The byte enable lines indicate the selected byte in each word.

$\overline{BE3}$–$\overline{BE0}$ Byte Enables (outputs)—The byte enables select which of 4 addressed bytes are active in a memory access. When a memory region is configured for an 8-bit bus width, $\overline{BE1}$ and $\overline{BE0}$ act as the lower two bits of the address. For a 16-bit memory region, $\overline{BE1}$, $\overline{BE3}$, and $\overline{BE0}$ are encoded to provide A1, $\overline{BHE}$, and $\overline{BLE}$ respectively.

W/$\overline{R}$ Write or Read (output)—This signal is low for read accesses and high for write accesses.

$\overline{ADS}$ Address Strobe (output)—Indicates valid address and the start of a new bus access.

DT/$\overline{R}$ Data Transmit or Receive (output)—Direction control for data transceivers; similar to W/$\overline{R}$.

$\overline{DEN}$ Data Enable (output)—Low during a bus request after the first address cycle. This signal is used to control data transceivers and to indicate the end of a bus request.

$\overline{WAIT}$ Wait (output)—Indicates that wait states are being inserted by the internal wait state generator.

$\overline{READY}$ Ready (input)—Signals that data is valid for a read transfer or ends data hold for a write transfer. This function can be disabled for a memory region.

$\overline{BTERM}$ Burst Terminate (input)—Terminates a burst access. Another address is generated to complete the request when the signal is deasserted. This function can be disabled for a memory region.

D/C Data or Code (output)—Indicates a data transfer or a code fetch.

$\overline{DMA}$ DMA Access (output)—Indicates that a bus request was initiated by either the user program or the DMA.

$\overline{SUP}$ Supervisor Access (output)—Indicates that a bus access originated from a bus request issued in supervisor mode. This signal can be used to protect system data structures, or peripherals from errant modification by the user code.

$\overline{LOCK}$ Lock (output)—Indicates that an atomic memory operation is in progress. This signal can be used to inhibit external agents from modifying memory which is atomically accessed.

$\overline{BLAST}$ Burst Last (output)—Indicates the last transfer in a burst access.

HOLD Hold (input)—HOLD can be used by a bus requester to request access to the bus. The processor asserts HLDA after the current bus request or locked requests have completed.

HOLDA Hold Acknowledge (output)—Indicates to a bus requester that the processor has relinquished control of the bus.

BREQ Bus Request (output)—Indicates that requests are queued in the bus controller and are waiting to be serviced. BREQ can be used for external bus arbitration logic in conjunction with HOLD and HLDA to regain bus mastership.

Figure 4-1 shows the timing for a simple, non-burst, non-pipelined read and write access. The timing relations for the key control signals are shown in this figure.

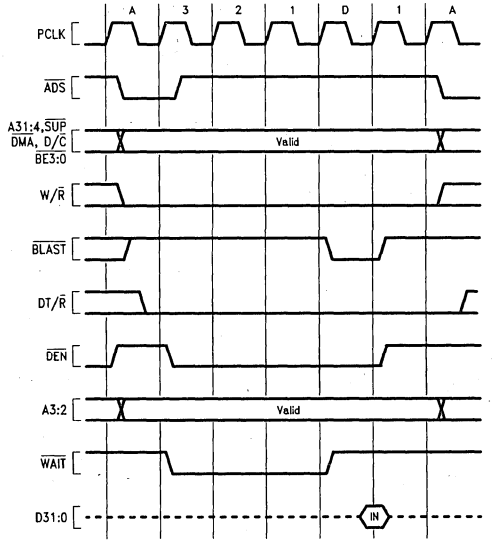


Figure 4-1. Basic Read and Write Request

4.2.4 MEMORY REGION CONFIGURATION TABLE

The BCU can be configured differently for 16 separate sections (referred to as regions) of the address space. The four most significant bits of a memory address define the location of each region in memory. The bus characteristics in a region are specified in the *memory region configuration table.* When a bus request is serviced, the BCU accesses the configuration table entry for the region addressed and services the request based on the bus characteristics programmed for that region. The characteristics programmed for each region are listed below:

- Burst Mode (on/off)
- Wait States (5 parameters)
- Bus Width (8-, 16-, or 32-bit)
- Ready Inputs (on/off)
- Address Pipelining (on/off)
- Byte Ordering (Big/Little Endian)

The flexibility of region programming simplifies the bus interface in applications where a memory system is made up of a variety of sub-systems, such as SRAM, DRAM, ROM, and memory mapped peripherals. Each memory sub-system can be mapped into a different region in memory, and that region can be configured specifically for the requirements of the particular memory sub-system.

The configuration table is made up of 16 on-chip control registers (Figure 4-2). Each register is programmed with the configuration information for a single region. Since the region table is located on-chip, access to region information does not affect the performance of the bus.

4.2.4.1 Burst Accesses

The 80960CA BCU is capable of burst accesses to memory systems which are designed to support this feature. Burst mode is intended to get the most performance from low cost memory systems. A burst access is a single address cycle followed by successive data or instruction transfers. The transfers reference data or instructions at sequential addresses starting at the address which began the burst access (Figure 4-3). In a burst memory system, the upper 28 bits of an address remain fixed while the lower two bits A2 and A3 increment to access subsequent locations.

Wait state timing for the first access of a burst request is controlled independently from the timing for subsequent accesses. A memory sub-system using static column mode or page mode DRAMs, for example, can take advantage of the short column access times for these devices by using burst mode. Interleaved ROM or EPROM systems can also be constructed which simultaneously access several words and then use burst mode to multiplex the multi-word array onto the data bus.

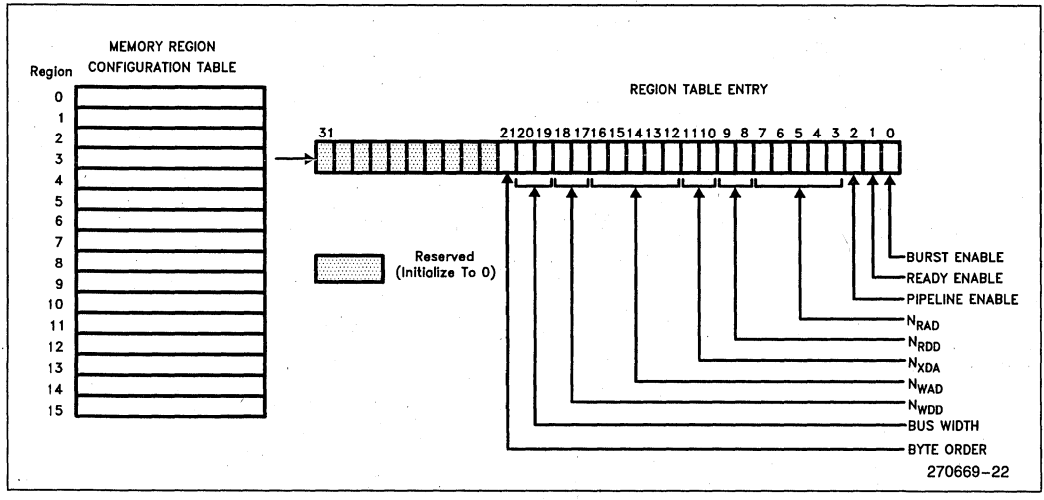


Figure 4-2. Memory Region Configuration Table

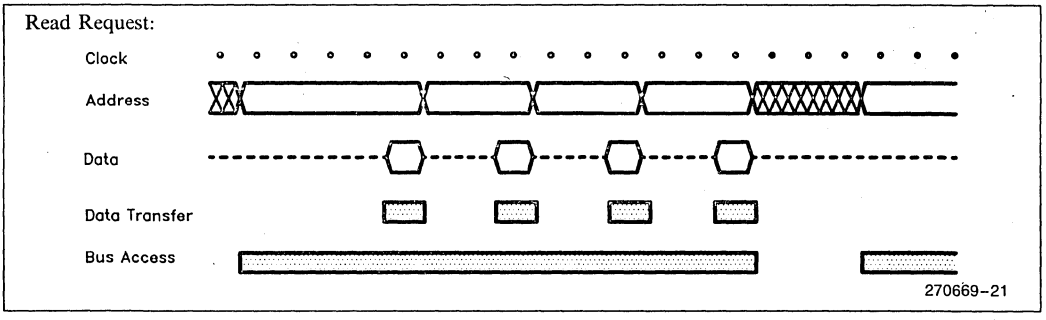


Figure 4-3. Burst Memory Request

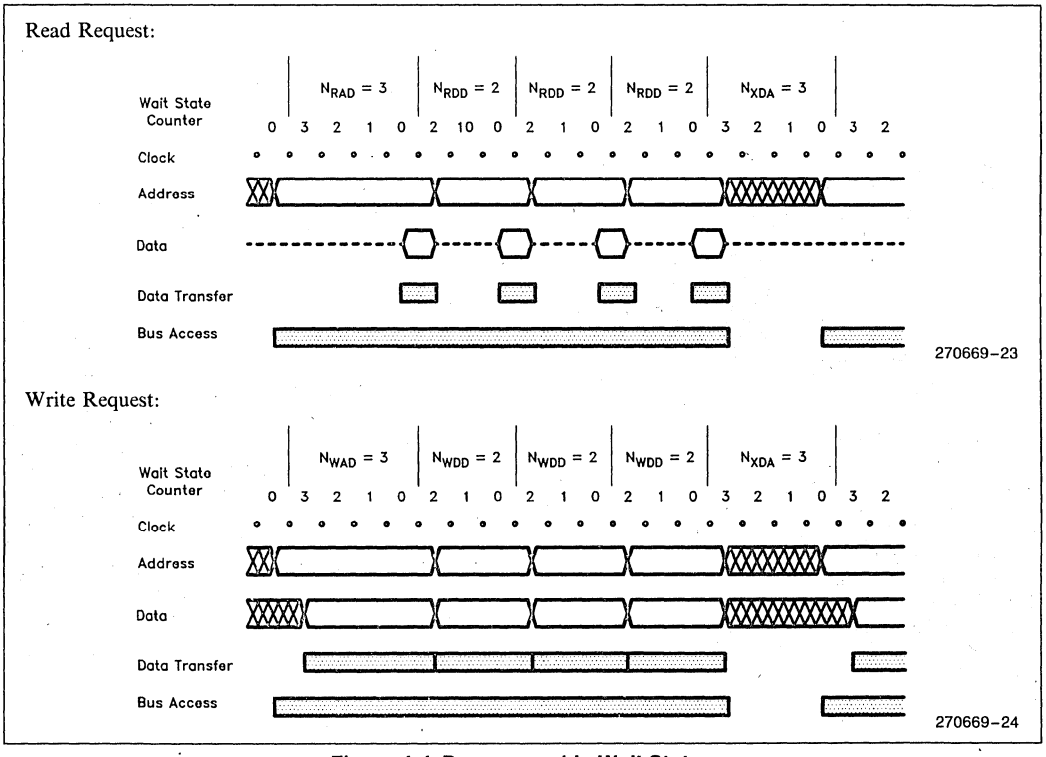


Figure 4-4. Programmable Wait States

4.2.4.2 Programmable Wait State Generation

The 80960CA may be interfaced with a variety of memory sub-systems and peripherals with a minimum system cost and complexity. To achieve this interface flexibility, the 80960CA implements an internal programmable wait state generator. Internally generated wait states eliminate the potential system delays which come from generating wait states with external logic.

Wait states are programmed for each region in the memory region configuration table. The number of wait states is programmable over a range which allows efficient control of memory devices ranging from ultra-fast SRAMs to slow peripherals. An external ready signal is also provided for external wait state control.

The wait states which can be generated by the 80960CA are shown in Figure 4-4. In this table N is the number of wait states inserted. The wait states for read accesses and for write accesses are described by three parameters each. For read accesses, N_{RAD} is the number of states between the address cycle and the first data cycle and N_{RDD} is the number of states between consecutive data cycles in a burst access. For writes, N_{WAD} is the number of states that data is held after an address cycle, and N_{WDD} is the number of states that data is held for consecutive data cycles in a burst write. For both reads and writes, N_{XDA} is the number of dead cycles after the last data cycle and before the next address.

4.2.4.3 READY Control

The memory region configuration table allows the ready input ($\overline{READY}$) to be enabled or disabled for each region. If the ready input is disabled, the external input has no effect on the wait states generated for a memory access; all wait states are generated internally. If the ready input is enabled, it works in conjunction with the programmable wait state generator. In this case, the ready input has no effect until the number of programmed wait states has expired. When the wait state counter reaches 0, the ready input is sampled, and wait states continue or are terminated based on the value of the ready input. In order to gain complete external control over wait states, all wait state parameters for a region can be set to 0.

4.2.4.4 Pipelined Reads

The 80960CA BCU provides an address pipelining mode (Figure 4-5) to optimize the performance of instruction and data fetches from external memory. When the pipelined read mode is enabled, an address cycle overlaps with the last data cycle in each access, effectively reducing the total time needed for each access. Pipelining mode is selected in each region by programming the memory region configuration table.

4.2.4.5 Byte Ordering

One of two configurations for byte ordering, often referred to as little endian or big endian, is selected for each region by programming the memory region configuration table. The byte ordering options make the 80960CA capable of sharing memory with a processor which uses either byte ordering scheme. Byte ordering refers to the way that the 80960CA relates internal data to the way that data is stored or fetched from memory. The little endian configuration orders the bytes in a short-word or word so that the least significant byte of the quantity is positioned at the lowest address and the most significant byte at the highest address in memory. Conversely, for the big endian configuration, the least significant byte is positioned at the highest address, and the most significant byte at the lowest address. For example, for little endian ordering, byte 0 for word data would be found in memory at an address of the form XXXX XXX0H and, for big endian, at address XXXX XXX3H.

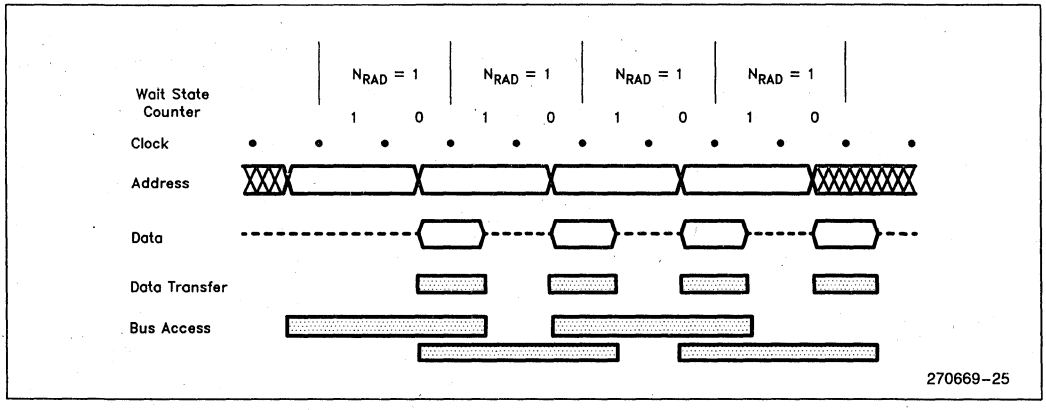


Figure 4-5. Pipelined Read Request

4.2.4.6 Data Alignment

The 80960CA can service any aligned or non-aligned bus request. Aligned requests are directed to their natural boundary in memory. In other words, the addresses for aligned requests are even multiples of the length of the data transferred. Non-aligned requests are not serviced directly by the BCU but are assisted by microcode. Microcode automatically breaks non-aligned requests into multiple aligned requests which are then reissued to the BCU. Depending on the degree of non-alignment and the length of the original request, the resulting requests by microcode will consist of a combination of byte, short-word, and double-word requests. The BCU is able to generate an operation-unaligned fault when a non-aligned bus request is first received. This fault can be selectively masked at initialization.

4.3 DMA Controller

The DMA controller is a high-performance, full-functioned integrated peripheral. The DMA controller can manage 4 channels of DMA transfer concurrent with program execution. Separate external control for each channel is provided. Each channel supports high-performance memory to memory transfers where the source and destination can be any combination of internal data RAM or external memory. The DMA Controller supports various types of transfers such as high-speed fly-by transfers and data chaining with the use of linked descriptor lists in memory.

The 80960CA's DMA controller is implemented using dedicated hardware and microcode. Because of the efficiency of the core, it is possible for the microcode to execute DMA transfers at high speeds. DMA transfers are performed by the core concurrently with execution of the user's program. Internal DMA logic is used for sampling requests, synchronizing transfers with external devices, and handling the service of multiple active channels.

4.3.1 SIGNAL DESCRIPTIONS

Twelve pins are dedicated to the DMA controller. Three pins are associated with each DMA channel. These pins are described below. In this description, the pin number corresponds to the channel number. For example, the $\overline{\text{DREQ0}}$ pin is the request pin for channel 0.

$\overline{\text{DREQ3}}$–$\overline{\text{DREQ0}}$ DMA Request (input)—This input indicates that an external device is requesting a DMA transfer. A DMA transfer refers to the complete transfer of one byte, short-word, word, or quad-word, depending on the transfer data width selected for the channel.

$\overline{\text{DACK3}}$–$\overline{\text{DACK0}}$ DMA Acknowledge (output)—This output becomes active when the requesting device is accessed.

$\overline{\text{EOP3/TC3}}$–$\overline{\text{EOP0/TC0}}$ End of Process (input) or Terminal Count (output)—This pin functions either as an input ($\overline{\text{EOPx}}$) or as an output ($\overline{\text{TCx}}$). When programmed as an output, the pin is driven active for one clock after byte count reaches zero and a DMA terminates. When programmed as an input, an external device can cause the DMA operation to terminate.

4.3.2 DMA TRANSFERS

The 80960CA DMA controller supports a variety of transfer modes and variations of these modes, allowing the DMA to adapt to a number of hardware systems and the performance requirements of these systems.

4.3.2.1 Standard Block and Demand Mode Transfers

A standard DMA transfer is made up of multiple bus requests. Loads from a source address are followed by stores to a destination address. The DMA controller issues the proper combination of these bus requests to execute the DMA transfer. For example, a typical DMA transfer between memory and an 8-bit peripheral could appear as a single byte load request directed to the source memory, followed by a single byte store request directed to the 8-bit peripheral.

The DMA controller has two basic transfer modes: block mode (unsynchronized) and demand mode (synchronized). Any DMA transfer will be serviced by one of these basic transfer modes.

A block mode DMA is initiated by software. Block mode DMAs are generally between memory. Block mode DMA transfers are not synchronized with any type of request from an external device. Once the DMA begins, it will continue until the entire block is complete or until it is suspended. The source and destination addresses for block mode transfers can be incremented or held constant for a DMA.

A demand mode DMA is controlled by an external device. Demand mode DMAs are generally between an external device and memory. In demand mode, each individual DMA transfer can be synchronized with a request. The request is signalled when an external device activates a DMA channel request pin ($\overline{\text{DREQ3}}$–$\overline{\text{DREQ0}}$). The DMA controller acknowledges this request with the DMA acknowledge pin ($\overline{\text{DACK3}}$–$\overline{\text{DACK0}}$) when the requesting device is accessed. A demand mode transfer may be synchronized with either the source or the destination device.

4.3.2.2 Fly-by Transfers

A fly-by transfer mode is provided for the most performance-critical DMA applications. Fly-by mode also makes very efficient use of the external bus during a DMA. Standard DMA transfers involve multiple bus requests: load requests directed to the source and a store request directed to the destination. Fly-by transfers only require a single bus request. For a fly-by transfer, memory sees a load or a store on the bus while the requesting device is selected by the DMA acknowledge pin. The data is never actually read from or written to the 80960CA. For memory to device transfers, the processor issues a load, and, while reading the memory, accesses the external device with the DMA acknowledge pin. The data is then written directly to the destination device with a single bus request. For a device to memory transfer, the reverse operation is performed. The DMA issues a store, and, while writing the memory, accesses the source device with the DMA acknowledge pin. In this case, the processor floats the data bus and the device's data is written directly into memory.

4.3.2.3 Data Chaining

Each DMA channel can be programmed in a data chaining mode. In this mode, all transfer information is taken from a linked-list descriptor in memory (Figure 4-6). Data chaining is started by specifying a pointer to a descriptor in memory. The transfer continues until the number of bytes in the byte count field in the descriptor is transferred. At this time, another linked-list descriptor may be executed. The next descriptor is specified by the next-pointer field in the current description. Data chaining continues until a null pointer is encountered in the next-pointer field. Data chaining can be designated as source chaining, destination chaining, or both.

In data chaining mode, an option exists which allows chaining descriptors to be updated while the DMA is running. When this option is enabled, the DMA sets a bit in the DMA's special function register after loading a descriptor and then checks this bit before loading the next descriptor. If the bit has been cleared by the user, the DMA continues; otherwise, the DMA waits for the next descriptor to be set up and for the user to clear the bit. An interrupt can be generated when each buffer is complete or when the DMA is terminated with a null pointer or the EOP pin.

4.3.3 TRANSFER CHARACTERISTICS

The DMA controller provides the programmer with a number of options for configuring the characteristics of a DMA transfer. Intelligent selection of transfer characteristics works to balance DMA performance and functionality with performance of the user program when the DMA is in progress.

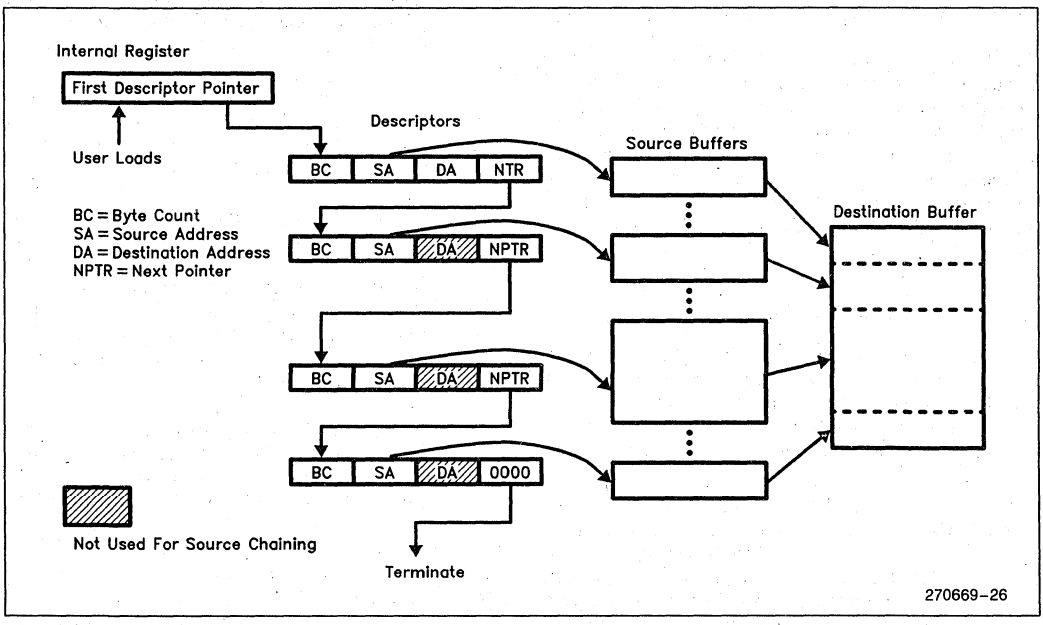


Figure 4-6. Source Data Chaining

The DMA controller provides features to optimize transfers by moving a maximum amount of data for each bus request issued. This is controlled by specifying the width of the source and destination directed bus requests for a DMA transfer, and by on-chip assembly or disassembly of the transfer when source and destination are not of equal widths.

Data alignment is performed automatically by the DMA controller when the source and destination of a transfer are not aligned. The alignment algorithm is optimized for many transfers, providing a performance comparable to the aligned transfer cases.

4.3.3.1 Transfer Data Length

The transfer data length specifies the length of bus requests directed to the source and destination in a standard DMA transfer. Byte, short, word, or quad-word loads and stores are selected for either source or destination when a DMA channel is set up. Assembly and disassembly of data is automatically performed when the source and destination widths are different. This feature provides the most efficient use of the bus when DMA transfers occur between a source and a destination with different external bus widths.

The DMA controller provides the option of using quad word transfers to enhance DMA performance. When quad transfers are specified, the DMA will request a four-word load request and four-word store request for each DMA transfer. The trade-off for the added DMA performance is latency on the external bus, preventing requests by the core, or by another DMA channel from being immediately serviced.

4.3.3.2 Data Alignment

The DMA controller supports transfer of source and destination data aligned to different byte boundaries in memory. The DMA implements microcode algorithms to transfer some non-aligned data with a performance level approaching that for aligned transfers. The DMA accomplishes this by attempting to issue the maximum number of aligned bus requests during a DMA (Figure 4-7). As shown, most of the overhead due to non-aligned DMAs is incurred at the beginning and end of the DMA. DMAs with low byte counts, therefore, do not benefit as much from the data alignment features of the DMA. The alignment feature is optimized for 8-bit to 8-bit, 32-bit to 32-bit and for 8-bit and 32-bit combinations of source and destination lengths.

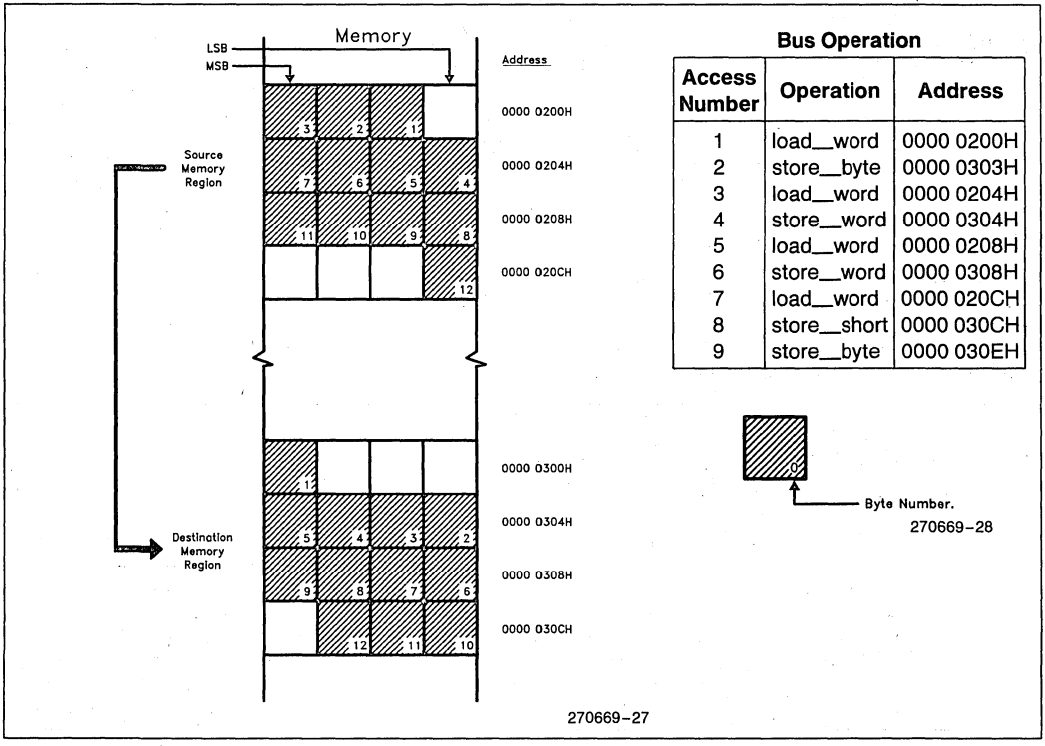


Figure 4-7. DMA Data Alignment

4.3.3.3 Channel Priority

The DMA controller arbitrates the priority of the 4 DMA channels. If multiple DMA channels are enabled, the DMA controller will determine in which order each channel is serviced.

The DMA controller can be configured in one of two priority modes, fixed mode or rotating mode. The fixed mode assumes a fixed priority for each channel with channel 0 having the highest priority, followed by channels 1, 2, and 3, with channel 3 having the lowest priority. The rotating mode updates a channel's priority to the lowest priority after that channel's DMA is made. This insures that a single channel is never locked out by other active channels. The priority sequence is always in the same order, with priority rotating from the low channel numbers to the high channel numbers.

4.3.3.4 Performance and Latency Considerations

DMA operations and the user program share the resources of the core and of the external bus. DMA performance and the performance of the user program are coupled directly to the balance of load sharing between these two processes. The core resources necessary to perform a DMA transfer vary depending on the way a channel has been configured. For example, byte assembly and disassembly requires more processor overhead per byte of transfer than does a transfer in which the source and destination transfer lengths are equal. The performance of a DMA is also tightly coupled to the user program's use of the external bus. If the user program does not make frequent bus requests, the requests by the DMA controller will be serviced with little or no delay.

The user can enhance performance of the DMA with trade-offs in system complexity and flexibility. Aligned transfers eliminate the microcode overhead needed to perform the internal alignments. DMAs between regions of equal transfer widths eliminate overhead for assembly and disassembly. Source or destination memory configured as burst memory will provide the most efficient use of the DMA controller when the quad-transfer feature is enabled. Using the fly-by mode reduces the number of bus requests needed for a DMA since fly-by mode uses only a single load or a single store request for each transfer.

4.3.4 DMA CONTROL AND CONFIGURATION

The DMA Controller uses an SFR register, the DMA command (DMAC) register, and the setup DMA (**sdma**) instruction for configuration and control of a DMA. The **sdma** instruction is used to configure each DMA channel. Transfer widths, byte count, source and destination addresses for a DMA are specified in this instruction.

The DMAC register (Figure 4-8) is described below.

The *channel enable field* enables a DMA once the channel is set up. Clearing these bits will also cause a DMA transfer to be suspended.

The *terminal count field* signals that byte count has reached zero and a DMA has ended.

The *channel active field* indicates that a channel is idle or active. If set, this bit indicates that the channel is active. This implies that the channel is servicing a transfer or has a request pending. The active bits are status information only.

The *channel done field* indicates that a DMA operation is complete. The done bits are status information only.

The *channel wait field* is used for handshaking with a user program in data chaining mode. The DMA sets these bits when a new linked-list descriptor is read. The DMA will not read the next descriptor until this bit is cleared by the user. The user can set up the next descriptor and then clear the channel wait bits to dynamically change descriptors.

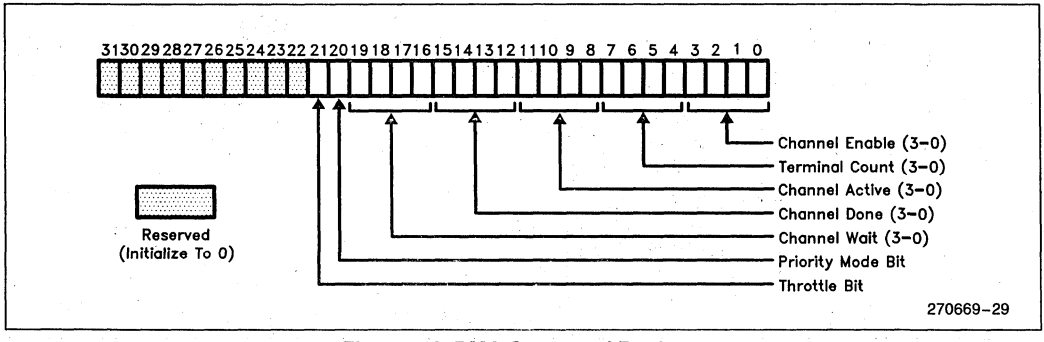


Figure 4-8. DMA Command Register

A *priority mode bit* selects rotating or fixed priority mode.

The *throttle bit* selects the maximum amount of core resources that the DMA microcode will receive in relation to the execution of the user program.

4.3.5 DMA INTERRUPTS

The DMA controller is the source of 4 hardware interrupts in the 80960CA. The DMA Controller can be programmed to request an interrupt when a DMA is complete, or when a buffer transfer is completed in chaining mode. Each channel requests a different interrupt.

4.4 Interrupt Controller

The 80960CA *Interrupt Controller* manages interrupts which are requested by external agents or by the DMA Controller. The interrupt controller manages 4 internal DMA interrupt sources, a single NMI (Non-Maskable Interrupt) pin, and 8 external interrupt pins. Up to 248 external interrupt sources can be supported by the interrupt controller. The interrupt controller handles the prioritization of software interrupts, hardware interrupts, and the process priority, and signals the core when interrupts are to be serviced. The interrupt controller provides the low-latency interrupt service featured on the 80960CA.

4.4.1 EXTERNAL INTERRUPTS

The 80960CA provides 8 interrupt pins and one NMI pin for detecting external requests. The interrupt controller allows the 8 interrupt pins to be configured as dedicated inputs capable of requesting 8 interrupts, or as a vectored input capable of requesting up to 248 interrupts. The NMI pin is always a dedicated input. The interrupt controller pins are described below.

$\overline{\text{XINT7}}$–$\overline{\text{XINT0}}$ External Interrupts (inputs)—These pins can be used as dedicated inputs, or acting together as an 8-bit number, request any interrupt. The inputs are edge or level detected, and are optionally debounced internally.

$\overline{\text{NMI}}$ Non-Maskable Interrupt (input)—NMI requests the highest priority interrupt. NMI is always taken and is not maskable (as the name implies), and not interruptable.

4.4.2 INTERRUPT MODES

The 8 external interrupt pins can be configured in one of three modes: dedicated mode, expanded mode, or mixed mode (Figure 4-9).

4.4.2.1 Dedicated Mode Interrupts

In dedicated mode, each of the 8 interrupt pins acts as a dedicated input. When an external event is detected on an interrupt pin, a unique interrupt is requested for that pin. It is possible to map each dedicated pin to one of a number of possible interrupt vectors. This is accomplished by programming the interrupt map (IMAP) control registers with an interrupt vector number for each pin. (Recall that interrupt vector numbers are 8-bit values which reference the 248 vectors in the interrupt table.)

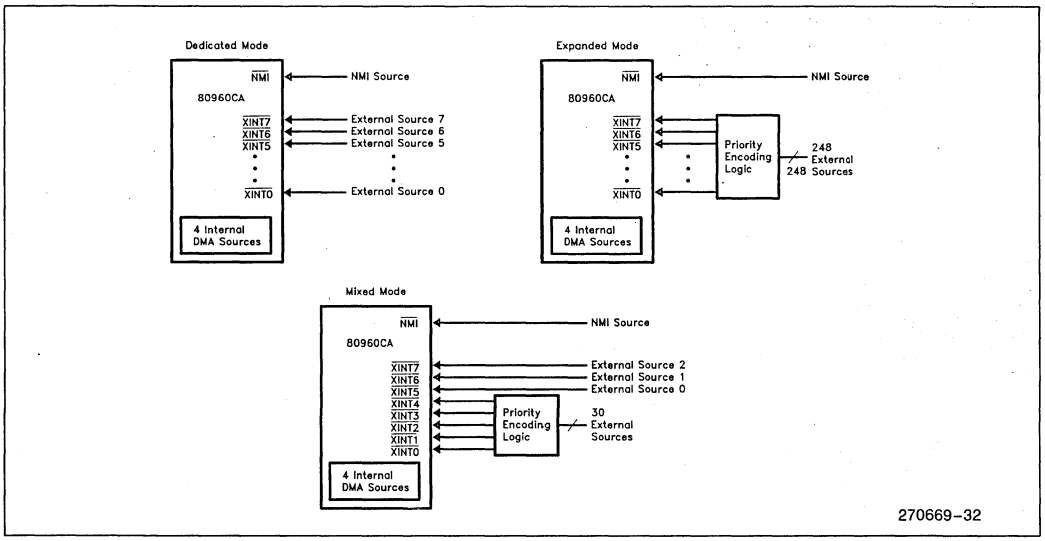


Figure 4-9. Interrupt Modes

Only the upper four bits of the vector number can be programmed for a dedicated mode interrupt. The lower four bits are fixed at the value 0010_2. With four programmable bits, one of 15 interrupt vectors is available for each dedicated pin. These interrupt vectors span the even priority levels from priority 2 to 30. The vector at priority 0 is not defined.

The 15 interrupt vectors available to dedicated sources can be cached in internal data RAM. If this interrupt vector caching feature is selected, the processor will automatically fetch the vector from data RAM, eliminating the latency caused by a bus request for a vector in external memory.

The DMA Controller can request four interrupts to signal the end of a DMA for each of four channels. The four interrupt signals from the DMA are handled by the interrupt controller in the same way as an interrupt pin configured as a dedicated input. Each of the four DMA sources may request one of 15 interrupts by programming the IMAP for that source.

4.4.2.2 Expanded Mode Interrupts

In expanded mode, external hardware considers the interrupt pins ($\overline{XINT0}$–$\overline{XINT7}$) as an 8-bit binary number. This number is used directly as the interrupt vector number. Each of the 248 possible interrupt vectors can be referenced in this way, allowing a separate external source for each vector. External hardware is responsible for recognizing individual hardware sources and then driving the interrupt vector number corresponding to that source onto the interrupt pins.

4.4.2.3 Mixed Mode Interrupts

In mixed mode, the 8 interrupt pins are divided into two functional sets. One set functions in dedicated mode, the other in expanded mode. In mixed mode, three pins are dedicated interrupt pins ($\overline{XINT7}$–$\overline{XINT5}$). A programmable vector number is associated with each of these pins. The remaining five interrupt pins ($\overline{XINT4}$–$\overline{XINT0}$) are treated as the most significant five bits of the expanded mode vector number. The lower order bits are internally forced to 010_2 to form the full 8-bit value for the vector number.

4.4.3 INTERRUPT CONTROLLER SETUP

The interrupt controller uses two special function registers to manage interrupt requests by hardware sources. The hardware interrupt pending register (IPND) and the hardware interrupt mask register (IMSK) are addressed as sf0 and sf1 respectively. A single bit in each register corresponds to each of the 8 possible external sources and 4 DMA sources for hardware interrupts. The IMSK register performs the function of masking hardware interrupts and the IPND register implements posting of interrupts requested by hardware. When configured for expanded or mixed mode interrupts, bit 0 of the IMSK register globally masks the expanded mode interrupts.

4.4.4. NON-MASKABLE INTERRUPT

In addition to the maskable hardware interrupts, a single *Non-Maskable Interrupt (NMI)* is provided. A dedicated NMI pin is used to request this interrupt. NMI is defined as a higher priority than any hardware interrupt, software interrupt, or process priority. The NMI procedure, therefore, can never be interrupted and must execute the return instruction before other procedures can execute. The NMI procedure is entered through vector 248. This vector is cached in internal data RAM at initialization to reduce latency for the NMI.

APPENDIX A
80960CA CORE IMPLEMENTATION

The 80960CA Core is a high-performance implementation of the 80960 Core Architecture. This section briefly describes the microarchitecture of the 80960CA core and the key constructs used to achieve parallel instruction execution.

The 80960CA core can be divided into the 6 main subunits listed below.

— Instruction Sequencer
— Register File
— Execution Unit
— Multiply and Divide Unit
— Address Generation Unit
— Static Data RAM and Local Register Cache

Figure A-1 is a simple block diagram of the 80960CA. The nucleus of the processor is the Instruction Sequencer and Register File. The other subunits of the core, referred to as coprocessors, radiate from these units, connecting to either the register (REG) side or the memory (MEM) side of the processor. The Instruction Sequencer issues directives, via the REG and MEM interfaces, which target a specific coprocessor. That coprocessor then executes an express function virtually decoupled from the IS and the other coprocessors. The REG and MEM data busses shown in Figure A-1 are used to transfer data between the common Register File and the coprocessors.

A.1 Instruction Sequencer

The *Instruction Sequencer (IS)* decodes the instruction stream and drives the decoded instruction stream onto the coprocessor interfaces. In a single clock, the IS decodes up to 4 instruction and issues up to three of these instructions to the on-chip coprocessors or to the IS itself. One register (REG) format, one memory (MEM) format, and one control or control and branch (CTRL or COBR) format instruction can be issued at one time. These instructions are directed respectively to the REG coprocessors, the MEM coprocessors, or to the IS. The ability to issue multiple instructions in parallel can result in the simultaneous execution of many instructions at once. An optimizing compiler or hand optimization of assembly code can easily produce an instruction stream which takes full advantage of the parallel execution of the core.

A technique known as *resource scoreboarding* is used to manage the parallel execution of instructions and the common resources of the processor. A coprocessor, for example, can scoreboard itself, indicating that it cannot

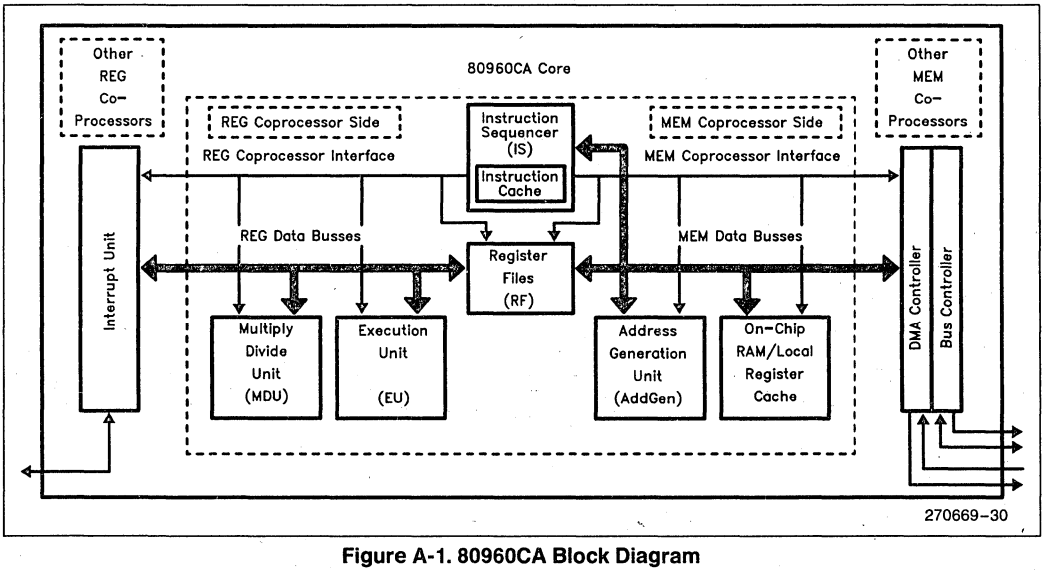


Figure A-1. 80960CA Block Diagram

act on another instruction until an instruction currently executing on that coprocessor is completed. A specific form of resource scoreboarding is referred to as *register scoreboarding*. When the computation stage of an instruction takes more than one clock, the destination register or registers for the result are scoreboarded as busy. A subsequent operation needing that particular register will be delayed until the multi-clock operation is completed. Instructions which do not use the scoreboarded registers can be executed in parallel.

The IS manages a three stage parallel instruction pipeline (Figure A-2). In the first stage of the pipeline (pipe 0), the address of the next instruction is calculated. This address may be the next sequential instruction, the target of a branch, or a location in microcode. In the second stage of the pipeline (pipe 1), the instructions are issued to the rest of the machine. In the third stage (pipe 2), the instruction computation is started, and for single cycle instructions, a result is returned.

Several microarchitectural features of the core are designed to minimize performance loss due to pipeline breaks.

Branch Prediction—To minimize pipeline breaks due to branching, the user can specify the direction that a conditional branch instruction will usually follow. The processor will execute along the specified instruction path with no pipeline break. If the branch direction specified was the direction actually selected by execution of the conditional branch, no pipeline break occurs. The direction of the branch guess is determined by a bit value in the CTRL format instructions.

Register Bypassing—Register bypassing is a feature which forwards the result of an instruction for immediate use as the source of another instruction. This forwarding occurs at the same time that the value is written to its destination register. Bypassing the register file saves the one clock cycle break which would otherwise occur while waiting for the value to be written to the register file and the register scoreboard to be cleared.

On-chip Cache—The on-chip instruction cache and local register cache eliminate many pipeline breaks which will occur if the IS is forced to wait for code or data to be moved between the 80960CA and external memory.

Register File Access—The Register File allows multiple instructions to gain access to the register set simultaneously. This eliminates pipeline breaks which would be caused by a loss of access to the register set by any coprocessor.

A.1.1 INSTRUCTION CACHE

The IS includes a 1 Kbyte two-way set associative instruction cache capable of delivering up to four instructions each clock to the Instruction Sequencer. The cache allows inner loops of code to execute with no external instruction fetches.

A.1.2 MICROCODE ROM

The 80960CA uses *microcode ROM* to implement complex instructions and functions. This includes calls, returns, DMA transfers, and initialization sequences. Microcode provides an inexpensive and simple method for implementing complex instructions in the mostly RISC environment of the 80960CA. When the IS encounters a microcoded instruction, it automatically branches to the microcode routine. The 80960CA performs this microcode branch in 0 clocks.

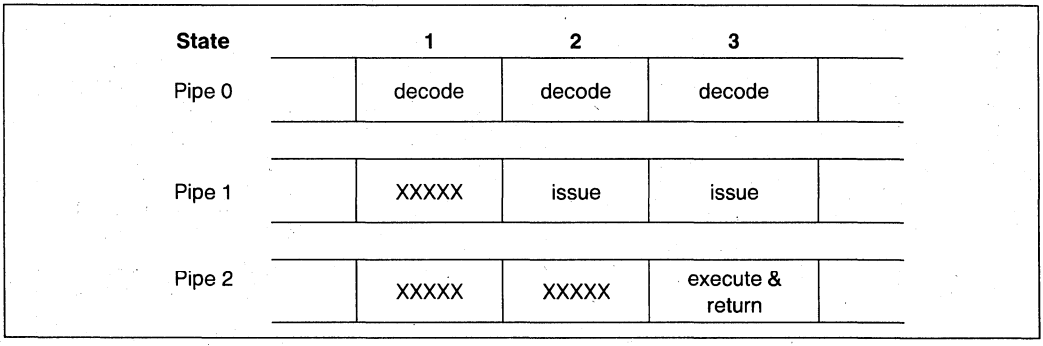


Figure A-2. Instruction Pipeline

A.2 Register File

The *Register File (RF)* contains the 16 local and 16 global registers. The register file has six ports (Figure A-3), allowing parallel access of the register set by several 80960CA coprocessors. This parallel access results in an ability to execute one simple logic or arithmetic instruction, one memory operation (load/store), and one address calculation per clock.

MEM coprocessors interface to the RF with a 128-bit wide load bus and a 128-bit wide store bus. These busses enable movement of up to 4 words per clock to and from the RF. These busses also allow LOAD data from a previous read access and STORE data from a current write access to be processed in the register file simultaneously. An additional 32-bit port allows an address or address reduction operand to be simultaneously fetched by the Address Generation Unit.

REG coprocessors interface to the RF with two 64-bit source busses and a single 64-bit destination bus. With this bus structure, two source operands are simultaneously issued to a REG coprocessor when an instruction is issued. A 64-bit destination bus allows the result from the previous operation to be written to the RF at the same time that the current operation's source operands are issued.

A.3 Execution Unit

The Execution Unit is the 32-bit Arithmetic and Logic Unit of the 80960CA Core. The EU can be viewed as a self-contained REG coprocessor with its own instruction set. As such, the EU is responsible for executing or supporting the execution of all the integer and ordinal arithmetic instructions, the logic and shift instructions, the move instructions, the bit and bit field instructions, and the compare operations. The EU performs any arithmetic or logical instructions in a single clock.

A.4 Multiply Divide Unit

The *Multiply and Divide Unit (MDU)* is a REG coprocessor which performs integer and ordinal multiply, divide, remainder, and modulo operations. The MDU detects integer overflow and divide by zero errors. The MDU is optimized for multiplication, performing 32-bit multiplies in 4 clocks. The MDU performs multiplies and divides in parallel with the main execution unit.

A.5 Address Generation Unit

The *Address Generation Unit (AGU)* is a MEM coprocessor which computes the effective addresses for memory operations. It directly executes the load address instruction (**lda**) and calculates addresses for loads and stores based on the addressing mode specified in these instructions. The address calculations are performed in parallel with the main execution unit (EU).

A.6 Data RAM and Local Register Cache

The *Data RAM and Local Register Cache* is part of a 1.5 Kbyte block of on-chip Static RAM (SRAM). 1 Kbyte of this SRAM is mapped into the 80960CA's address space from location 00000000H to 000003FFH. A portion of the remaining 512 bytes is dedicated to the Local Register Cache. This part of internal SRAM is not directly visible to the user. Loads and Stores, including quad-word accesses, to the internal SRAM are typically performed in only one clock. The complete local register set, therefore, can be moved to the local register cache in only four clocks.

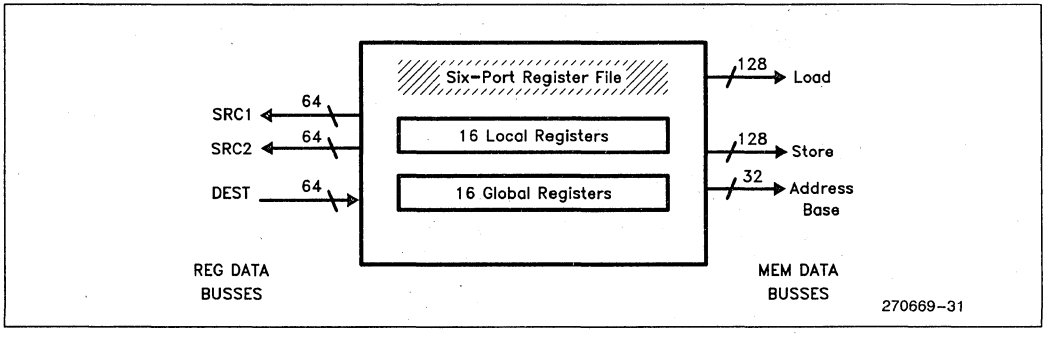
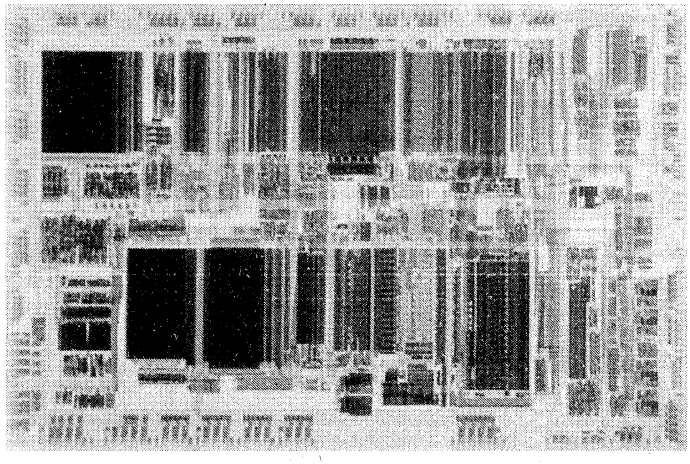


Figure A-3. Six-Port Register File

80960CA-33, -25, -16
32-BIT HIGH PERFORMANCE EMBEDDED PROCESSOR

- Two Instructions/Clock Sustained Execution
- Four 59 Mbytes/s DMA Channels with Data Chaining
- Demultiplexed 32-Bit Burst Bus with Pipelining

- 32-bit Parallel Architecture
  - Two Instructions/clock Execution
  - Load/Store Architecture
  - 16, 32-bit Global Registers
  - 16, 32-bit Local Registers
  - Manipulate 64-Bit Bit Fields
  - 11 Addressing Modes
  - Full Parallel Fault Model
  - Supervisor Protection Model
- Fast Procedure Call/Return Model
  - Full Procedure Call in 4 clocks
  - RISC Call in 2 clocks (BAL)
- On-Chip Register Cache
  - Caches Registers on Call/Ret
  - Minimum of 6 Frames provided
  - Number of Frames Programmable, up to 15
- On-Chip Instruction Cache
  - 1 Kbyte Two-Way Set Associative
  - 128-bit Path to Instruction Sequencer
  - Cache-Lock Modes
  - Cache-Off Mode
- High Bandwidth On-Chip Data Ram
  - 1 Kbytes On-chip RAM for Data
  - Sustain 128-bits per clock access
- Four On-Chip DMA Channels
  - 59 Mbytes/s Fly-by Transfers
  - 32 Mbytes/s Two-Cycle Transfers
  - Data Chaining
  - Data Packing/Unpacking
  - Programmable Priority Method
- 32-Bit Demultiplexed Burst Bus
  - 128-Bit Internal Data Paths to *and* from Registers
  - Burst Bus for DRAM Interfacing
  - Address Pipelining Option
  - Fully Programmable Wait States
  - Supports 8, 16 or 32-bit Bus Widths
  - Supports Unaligned Accesses
  - Supervisor Protection Pin
- High-Speed Interrupt Controller
  - Up to 248 External Interrupts
  - 32 Fully Programmable Priorities
  - Multi-mode 8-bit Interrupt Port
  - Four Internal DMA Interrupts
  - Separate, Non-maskable Interrupt Pin
  - Context Switch in 750 ns Typical



270727-1

Figure 1. 80960CA Die Photo

November 1991
Order Number: 270727-004

80960CA-33, -25, -16
32-Bit High Performance Embedded Processor

CONTENTS

CONTENTS

PAGE

FIGURES

3

CONTENTS

CONTENTS

1.0 PURPOSE

This document provides a preview of the electrical characteristics expected of the 33, 25 and 16 MHz versions of the 80960CA. For a detailed description of any 80960CA functional topic, other than parametric performance, consult the latest 80960CA Product Overview (Order No. 270669), or the 80960CA User's Manual (Order No. 270710).

2.0 80960CA OVERVIEW

The 80960CA is the second-generation member of the 80960 Family of embedded processors. The 80960CA is object code compatible with the 32-bit 80960 Core Architecture while including Special Function Register extensions to control on-chip peripherals, and instruction set extensions to shift 64-bit operands and configure on-chip hardware. Multiple 128-bit internal busses, on-chip instruction caching and a sophisticated instruction scheduler allow the processor to sustain execution of two instructions every clock, and peak at execution of 3 instructions per clock.

A 32-bit demultiplexed and pipelined burst bus provides a 132 Mbyte/s bandwidth to a system's high-speed external memory sub-system. In addition, the 80960CA's on-chip caching of instructions, procedure context and critical program data substantially decouples system performance from the wait states associated with accesses to the system's slower, cost sensitive, main memory sub-system.

The 80960CA bus controller also integrates full wait state and bus width control for highest system performance with minimal system design complexity. Unaligned access and Big Endian byte order support reduces the cost of porting existing applications to the 80960CA.

The processor also integrates four complete data-chaining DMA channels and a high-speed interrupt controller on-chip. The DMA channels perform: single-cycle or two-cycle transfers, data packing and unpacking, and data chaining. Block transfers, in addition to source or destination synchronized transfers are provided.

The interrupt controller provides full programmability of 248 interrupt sources into 32 priority levels with a typical interrupt task switch ("latency") time of 750 ns.

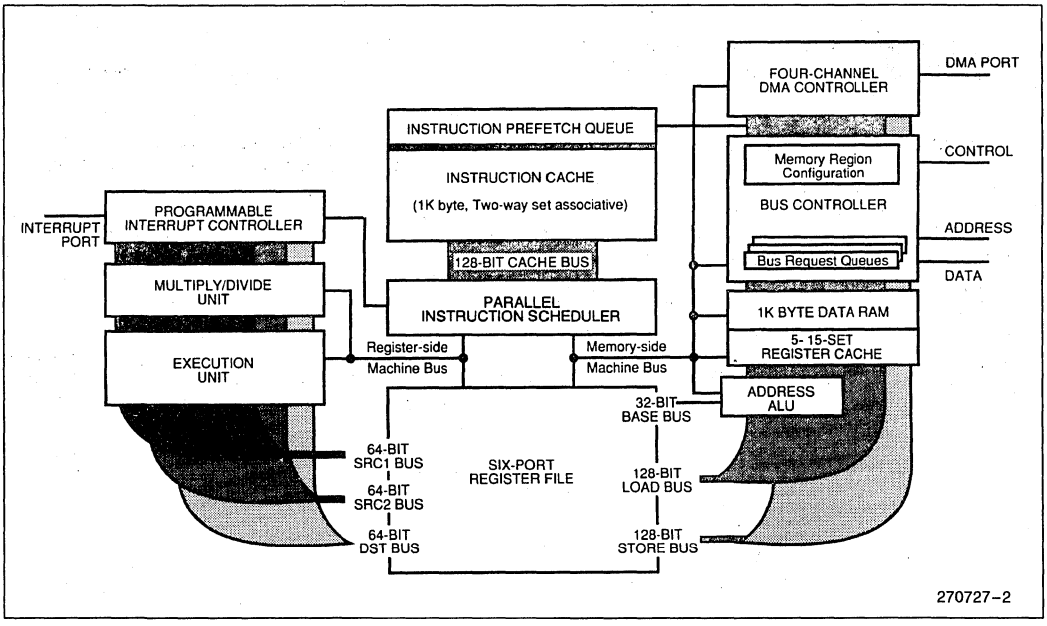


Figure 2. 80960CA Block Diagram

2.1. The C-Series Core

The C-Series core is a very high performance microarchitectural implementation of the 80960 Core Architecture. The C-Series core can sustain execution of two instructions per clock (66 MIPs at 33 MHz). To achieve this level of performance, Intel has incorporated state-of-the-art silicon technology and innovative microarchitectural constructs into the implementation of the C-Series core. Factors that contribute to the core's performance include:

— Parallel instruction decoding allows issue of up to three instructions per clock.

— Most instructions execute in a single clock.

— Parallel instruction decode allows sustained, simultaneous execution of two single-clock instructions every clock cycle.

— Efficient instruction pipeline is designed to minimize pipeline break losses.

— Register and resource scoreboarding allow simultaneous multi-clock instruction execution.

— Branch look-ahead and prediction allows many branches to execute with no pipeline break.

— Local Register Cache integrated on-chip caches Call/Return context.

— Two-way set associative, 1Kbyte integrated instruction cache

— 1Kbyte integrated Data RAM sustains a four-word (128-bit) access every clock cycle.

2.2. Pipelined, Burst Bus

A 32-bit high performance bus controller interfaces the 80960CA to external memory and peripherals. The Bus Control Unit features a maximum transfer rate of 132 Mbytes per second (at 33 MHz). Internally programmable wait states and 16 separately configurable memory regions allow the processor to interface with a variety of memory subsystems with a minimum of system complexity, and a maximum of performance. The Bus Controller's main features include:

— Demultiplexed, Burst Bus to exploit most efficient DRAM access modes

— Address Pipelining to reduce memory cost while maintaining performance

— 32-, 16- and 8-bit modes for I/O interfacing ease.

— Full internal wait state generation to reduce system cost

— Little and Big Endian support to ease application development

— Unaligned access support for code portability

— Three-deep request queue to decouple the bus from the core

— Direct interface to Intel's 27C960 Burst EPROM and 82596 Ethernet Controller.

2.3. Flexible DMA Controller

A four channel DMA controller provides high speed DMA control for data transfers involving peripherals and memory. The DMA provides advanced features such as data chaining, byte assembly and disassembly, and a high performance fly-by mode capable of transfer speed of up to 59 Mbytes per second at 33 MHz. The DMA controller features a performance and flexibility which is only possible by integrating the DMA controller and the 80960CA core.

2.4. Priority Interrupt Controller

A programmable-priority interrupt controller manages up to 248 external sources through the 8-bit external interrupt port. The Interrupt Unit also handles the 4 internal sources from the DMA controller, and a single non-maskable interrupt input. The 8-bit interrupt port can also be configured to provide individual interrupt sources that are level, or edge triggered.

Interrupts in the 80960CA are prioritized and signaled within 270 ns of the request. If the interrupt is of higher priority than the processor priority, the context switch to the interrupt routine typically is complete in another 480 ns. The interrupt unit provides the mechanism for the low latency and high throughput interrupt service which is essential for embedded applications.

2.5. Instruction Set Summary

The following table summarizes the 80960CA instruction set by logical groupings. See the 80960CA User's Manual for a complete description of the instruction set.

Data Movement	Arithmetic	Logical	Bit, Bit Field and Byte
Load Store Move Load Address	Add Subtract Multiply Divide Remainder Modulo Shift *Extended Shift Extended Multiply Extended Divide Add with Carry Subtract with Carry Rotate	And Not And And Not Or Exclusive Or Not Or Or Not Nor Exclusive Nor Not Nand	Set Bit Clear Bit Not Bit Alter Bit Scan for Bit Span over Bit Extract Modify Scan Byte for Equal
Comparison	**Branch**	**Call and Return**	**Fault**
Compare Conditional Compare Compare and Increment Compare and Decrement Condition Test Check Bit	Unconditional Branch Conditional Branch Compare and Branch	Call Call Extended Call System Return Branch and Link	Conditional Fault Synchronize Faults
Debug	**Processor Management**	**Atomic**	
Modify Trace Controls Mark Force Mark	Modify Process Controls Modify Arithmetic Controls *System Control *DMA Control Flush Local Registers	Atomic Add Atomic Modify	

NOTE:
Instructions marked by (*) are 80960CA extensions to the 80960 instruction set.

3.0 PACKAGE INFORMATION

3.1. Package Introduction

This section describes the pins, pinouts and thermal characteristics for the 80960CA in the 168-pin Ceramic Pin Grid Array (PGA) package and the 196 pin Plastic Quad Flat Package (PQFP). For complete package specifications and information, see the Intel Packaging Specification (Order # 231369).

3.2. Pin Descriptions

The 80960CA pins are described in this section. Table 1 presents the legend for interpreting the pin descriptions in the following tables.

The pins associated with the 32-bit demultiplexed processor bus are described in Table 2. The pins associated with basic processor configuration and control are described in Table 3. The pins associated with the 80960CA DMA Controller and Interrupt Unit are described in Table 4.

Figure 3 provides an example pin description table entry. The "I/O" signifies that the data pins are input-output. The "S" indicates the pins are synchronous to PCLK2:1. The "H(Z)" indicates that these pins float while the processor bus is in a Hold Acknowledge state. The "R(Z)" notation indicates that the pins also float while $\overline{\text{RESET}}$ is low.

All pins float while the processor is in the ONCE™ mode.

Table 1. Pin Description Nomenclature

Symbol	Description
I	Input only pin
O	Output only pin
I/O	Pin can be either an input or output
-	Pins "must be" connected as described
S(. . .)	Synchronous. Inputs must meet setup and hold times relative to PCLK2:1 for proper operation of the processor. All outputs are synchronous to PCLK2:1. S(E) Edge sensitive input S(L) Level sensitive input
A(. . .)	Asynchronous. Inputs may be asynchronous to PCLK2:1. A(E) Edge sensitive input A(L) Level sensitive input
H(. . .)	While the processor's bus is in the Hold Acknowledge or Bus Backoff state, the pin: H(1) is driven to V_{CC} H(0) is driven to V_{SS} H(Z) floats H(Q) continues to be a valid output
R(. . .)	While the processor's $\overline{\text{RESET}}$ pin is low, the pin R(1) is driven to V_{CC} R(0) is driven to V_{SS} R(Z) floats R(Q) continues to be a valid output

Name	Type	Description
D31:0	**I/O** S(L) H(Z) R(Z)	**DATA BUS** carries 32, 16 or 8-bit data quantities depending on bus width configuration. The least significant bit of the data is carried on D0 and the most significant on D31. When the bus is configured for 8 bit data, the lower 8 data lines, D7:0 are used. For 16 bit data widths, D15:0 are used. For 32 bit data the full data bus is used.

Figure 3. Example Pin Description Entry

Table 2. 80960CA Pin Description—External Bus Signals

Name	Type	Description
A31:2	**O** S H(Z) R(Z)	**ADDRESS BUS** carries the upper 30 bits of the physical address. A31 is the most significant address bit and A2 is the least significant. During a bus access, A31:2 identify all external addresses to word (4-byte) boundaries. The byte enable signals indicate the selected byte in each word. During burst accesses, A3 and A2 increment to indicate successive data cycles.
D31:0	**I / O** S(L) H(Z) R(Z)	**DATA BUS** carries 32, 16 or 8-bit data quantities depending on bus width configuration. The least significant bit of the data is carried on D0 and the most significant on D31. When the bus is configured for 8 bit data, the lower 8 data lines, D7:0 are used. For 16 bit bus widths, D15:0 are used. For 32 bit bus widths the full data bus is used.
$\overline{BE3}$ **$\overline{BE2}$** **$\overline{BE1}$** **$\overline{BE0}$**	**O** S H(Z) R(I)	**BYTE ENABLES** select which of the four bytes addressed by A31:2 are active during an access to a memory region configured for a 32-bit data-bus width. $\overline{BE3}$ applies to D31:24; $\overline{BE2}$ applies to D23:16; $\overline{BE1}$ applies to D15:8; and $\overline{BE0}$ applies to D7:0. 32-bit bus: $\overline{BE3}$ –Byte Enable 3 –enable D31:24 $\overline{BE2}$ –Byte Enable 2 –enable D23:16 $\overline{BE1}$ –Byte Enable 1 –enable D15:8 $\overline{BE0}$ –Byte Enable 0 –enable D7:0 For accesses to a memory region configured for a 16-bit data-bus width, the processor directly encodes $\overline{BE3}$, $\overline{BE1}$ and $\overline{BE0}$ to provided $\overline{BHE}$, A1 and $\overline{BLE}$ respectively. 16-bit bus: $\overline{BE3}$ –Byte High Enable ($\overline{BHE}$) –enable D15:8 $\overline{BE2}$ –Not used (is driven high or low) $\overline{BE1}$ –Address Bit 1 (A1) $\overline{BE0}$ –Byte Low Enable ($\overline{BLE}$) –enable D7:0 For accesses to a memory region configured for an 8-bit data bus width, the processor directly encodes $\overline{BE1}$ and $\overline{BE0}$ to provide A1 and A0 respectively. 8-bit bus: $\overline{BE3}$ –Not used (is driven high or low) $\overline{BE2}$ –Not used (is driven high or low) $\overline{BE1}$ –Address Bit 1 (A1) $\overline{BE0}$ –Address Bit 0 (A0)
W/$\overline{R}$	**O** S H(Z) R(0)	**WRITE/READ** is low (0) for read requests and high (1) for write requests. The W/$\overline{R}$ signal changes in the same clock cycle as $\overline{ADS}$. It remains valid for the entire access in non-pipelined regions. In pipelined regions, W/$\overline{R}$ may not be valid in the last cycle of a read access.
$\overline{ADS}$	**O** S H(Z) R(1)	**ADDRESS STROBE** indicates valid address and the start of a new bus access. $\overline{ADS}$ is asserted for the first clock of a bus access.
$\overline{READY}$	**I** S(L) H(Z) R(Z)	**READY** is an input which signals the termination of a data transfer. $\overline{READY}$ is used to indicate that read data on the bus is valid, or that a write-data transfer has completed. The $\overline{READY}$ signal works in conjunction with the internally programmed wait-state generator. If $\overline{READY}$ is enabled in a region, the pin is sampled after the programmed number of wait-states has expired. If the $\overline{READY}$ pin is deasserted high, wait states will continue to be inserted until $\overline{READY}$ becomes asserted low. This is true for the N_{RAD}, N_{RDD}, N_{WAD}, and N_{WDD} wait states. The N_{XDA} wait states cannot be extended.

Table 2. 80960CA Pin Description—External Bus Signals (Continued)

Name	Type	Description
$\overline{\text{BTERM}}$	I S(L) H(Z) R(Z)	**BURST TERMINATE**—The burst terminate signal breaks up a burst access and causes another address cycle to occur. The $\overline{\text{BTERM}}$ signal works in conjunction with the internally programmed wait-state generator. If $\overline{\text{READY}}$ and $\overline{\text{BTERM}}$ are enabled in a region, the $\overline{\text{BTERM}}$ pin is sampled after the programmed number of wait states has expired. When $\overline{\text{BTERM}}$ is asserted, additional wait states are inserted until $\overline{\text{BTERM}}$ is deasserted. When $\overline{\text{BTERM}}$ is deasserted, a new $\overline{\text{ADS}}$ signal is generated and the access is completed. The $\overline{\text{READY}}$ input is ignored when $\overline{\text{BTERM}}$ is asserted. $\overline{\text{BTERM}}$ must be externally synchronized to satisfy the $\overline{\text{BTERM}}$ setup and hold times.
$\overline{\text{WAIT}}$	O S H(Z) R(1)	**WAIT** indicates the status of the internal wait state generator. $\overline{\text{WAIT}}$ is active when wait states are being caused by the internal wait state generator and not by the $\overline{\text{READY}}$ or $\overline{\text{BTERM}}$ inputs. $\overline{\text{WAIT}}$ can be used to derive a write-data strobe. $\overline{\text{WAIT}}$ can also be thought of as a $\overline{\text{READY}}$ output that the processor provides when it is inserting wait states.
$\overline{\text{BLAST}}$	O S H(Z) R(0)	**BURST LAST** indicates the last transfer in a bus access. $\overline{\text{BLAST}}$ is asserted in the last data transfer of burst and non-burst accesses after the wait state counter reaches zero. $\overline{\text{BLAST}}$ remains active until the clock following the last cycle of the last data transfer of a bus access. If the $\overline{\text{READY}}$ or $\overline{\text{BTERM}}$ input is used to extend wait states, the $\overline{\text{BLAST}}$ signal remains active until $\overline{\text{READY}}$ or $\overline{\text{BTERM}}$ terminates the access.
DT/$\overline{\text{R}}$	O S H(Z) R(0)	**DATA TRANSMIT/RECEIVE** indicates direction for data transceivers. DT/$\overline{\text{R}}$ is used in conjunction with $\overline{\text{DEN}}$ to provide control for data transceivers attached to the external bus. When DT/$\overline{\text{R}}$ is low (0), the signal indicates that the processor will receive data. Conversely, when high (1) the processor will send data. DT/$\overline{\text{R}}$ will change only while $\overline{\text{DEN}}$ is high.
$\overline{\text{DEN}}$	O S H(Z) R(1)	**DATA ENABLE** indicates data cycles in a bus access. $\overline{\text{DEN}}$ is asserted (low) at the start of the first data cycle of a bus request and is deasserted (high) at the end of the last data cycle. $\overline{\text{DEN}}$ is used in conjunction with DT/$\overline{\text{R}}$ to provide control for data transceivers attached to the external bus. $\overline{\text{DEN}}$ remains asserted for sequential reads from pipelined memory regions. $\overline{\text{DEN}}$ is high when DT/$\overline{\text{R}}$ changes.
$\overline{\text{LOCK}}$	O S H(Z) R(1)	**BUS LOCK** indicates that an atomic read-modify-write operation is in progress. $\overline{\text{LOCK}}$ may be used to prevent external agents from accessing memory which is currently involved in an atomic operation. $\overline{\text{LOCK}}$ is asserted (0) in the first clock of an atomic operation, and deasserted in the clock cycle following the last bus access for the atomic operation. To allow the most flexibility for a memory system enforcement of locked accesses, the processor will acknowledge a bus hold request when $\overline{\text{LOCK}}$ is asserted. The processor will perform DMA transfers while $\overline{\text{LOCK}}$ is active.
HOLD	I S(L) H(Z) R(Z)	**HOLD REQUEST** signals that an external agent requests access to the external bus. The processor asserts HOLDA after completing the current bus request. HOLD, HOLDA, and BREQ are used together to arbitrate access to the processor's external bus by external bus agents.
$\overline{\text{BOFF}}$	I S(L) H(Z) R(Z)	**$\overline{\text{BOFF}}$ BUS BACKOFF** —The backoff pin, when asserted (0), suspends the current access and causes the bus pins to float. When the pin is deasserted (1), the $\overline{\text{ADS}}$ signal is asserted on the next clock cycle and the access is resumed.

Table 2. 80960CA Pin Description—External Bus Signals (Continued)

Name	Type	Description
HOLDA	O S H(1) R(Q)	**HOLD ACKNOWLEDGE** indicates to a bus requestor that the processor has relinquished control of the external bus. When HOLDA is asserted, the external address bus, data bus, and bus control signals are floated. HOLD, $\overline{\text{BOFF}}$, HOLDA and BREQ are used together to arbitrate access to the processor's external bus by external bus agents. Since the processor will grant HOLD requests and enter the Hold Acknowledge state even while $\overline{\text{RESET}}$ is active, the state of the HOLDA pin will be independent of the $\overline{\text{RESET}}$ pin.
BREQ	O S H(Q) R(0)	**BUS REQUEST** indicates that the processor wishes to perform a bus request. BREQ can be used by external bus arbitration logic in conjunction with HOLD and HOLDA to determine when to return mastership of the external bus to the processor.
D/$\overline{\text{C}}$	O S H(Z) R(Z)	**DATA OR CODE** indicates that a bus request is a data request (1) or a instruction request (0). D/$\overline{\text{C}}$ has the same timing as W/$\overline{\text{R}}$
$\overline{\text{DMA}}$	O S H(Z) R(Z)	**DMA ACCESS** indicates whether the bus request was initiated by the DMA controller. $\overline{\text{DMA}}$ will be asserted (low) for any DMA request. $\overline{\text{DMA}}$ will be deasserted (high) for all other requests.
$\overline{\text{SUP}}$	O S H(Z) R(Z)	**SUPERVISOR ACCESS** indicates whether the bus request is issued while in supervisor mode. $\overline{\text{SUP}}$ will be asserted (low) when the request has supervisor privileges, and will be deasserted (high) otherwise. $\overline{\text{SUP}}$ can be used to isolate supervisor code and data structures from non-supervisor requests.

Table 3. 80960CA Pin Description—Processor Control Signals

Name	Type	Description
$\overline{\text{RESET}}$	I A(L) H(Z) R(Z) N(Z)	**RESET** causes the chip to reset. When $\overline{\text{RESET}}$ is asserted (low), all external signals return to the reset state. When $\overline{\text{RESET}}$ is deasserted, initialization begins. When the two-x clock mode is selected, $\overline{\text{RESET}}$ must remain asserted for 16 PCLK2:1 cycles before being deasserted in order to guarantee correct initialization of the processor. When the one-x clock mode is selected, $\overline{\text{RESET}}$ must remain asserted for 10,000 PCLK2:1 cycles before being deasserted in order to guarantee correct initialization of the processor. The CLKMODE pin selects one-x or two-x input clock division of the CLKIN pin. The processor's Hold Acknowledge bus state functions while the chip is reset. If the processor's bus is in the Hold Acknowledge state when $\overline{\text{RESET}}$ is activated, the processor will internally reset, but will maintain the Hold Acknowledge state on external pins until the Hold request is removed. If a hold request is made while the processor is in the reset state, the processor bus will grant HOLDA and enter the Hold Acknowledge state.
$\overline{\text{FAIL}}$	O S H(Q) R(0)	**FAIL** indicates failure of the processor's self-test performed at initialization. When $\overline{\text{RESET}}$ is deasserted and the processor begins initialization, the $\overline{\text{FAIL}}$ pin is asserted (0). An internal self-test is performed as part of the initialization process. If this self-test passes, the $\overline{\text{FAIL}}$ pin is deasserted (1) otherwise it remains asserted. The $\overline{\text{FAIL}}$ pin is reasserted while the processor performs and external bus self-confidence test. If this self-test passes, the processor deasserts the $\overline{\text{FAIL}}$ pin and branches to the users initialization routine, otherwise the $\overline{\text{FAIL}}$ pin remains asserted. Internal self-test and the use of the $\overline{\text{FAIL}}$ pin can be disabled with the STEST pin.

Table 3. 80960CA Pin Description—Processor Control Signals (Continued)

Name	Type	Description
STEST	I S(L) H(Z) R(Z)	**SELF TEST** causes the processor's internal self-test feature to be enabled or disabled at initialization. STEST is read on the rising edge of $\overline{\text{RESET}}$. When asserted (high) the processor's internal self-test and external bus confidence tests are performed during processor initialization. When deasserted (low), only the internal self-test is not performed during initialization.
$\overline{\text{ONCE}}$™	I A(L) H(Z) R(Z)	**ON CIRCUIT EMULATION** causes all outputs to be floated when asserted (low). $\overline{\text{ONCE}}$ is continuously sampled while $\overline{\text{RESET}}$ is low, and is latched on the rising edge of $\overline{\text{RESET}}$. To place the processor in the ONCE state: (1) assert $\overline{\text{RESET}}$ and $\overline{\text{ONCE}}$ (order does not matter) (2) wait for at least 16 CLKIN periods in two-x mode, or 10,000 CLKIN periods in one-x mode, after V_{CC} and CLKIN are within operating specifications (3) deassert $\overline{\text{RESET}}$ (4) wait at least 32 CLKIN periods (The processor will now be latched in the ONCE state as long as $\overline{\text{RESET}}$ is high.) To exit the ONCE state, bring V_{CC} and CLKIN to operating conditions, then assert $\overline{\text{RESET}}$ and bring $\overline{\text{ONCE}}$ high prior to deasserting $\overline{\text{RESET}}$. CLKIN must operate within the specified operating conditions of the processor until step 4 above has been completed. The CLKIN may then be changed to DC to achieve the lowest possible ONCE mode leakage current. $\overline{\text{ONCE}}$ can be used by emulator products or for board testers to effectively make an installed processor transparent in the board.
CLKIN	I A(E) H(Z) R(Z)	**CLOCK INPUT** is an input for the external clock needed to run the processor. The external clock is internally divided as prescribed by the CLKMODE pin to produce PCLK2:1.
CLKMODE	I A(L) H(Z) R(Z)	**CLOCK MODE** selects the division factor applied to the external clock input (CLKIN). When CLKMODE is high (1), CLKIN is divided by one to create PCLK2:1 and the processor's internal clock. When CLKMODE is low (0), CLKIN is divided by two to create PCLK2:1 and the processor's internal clock. CLKMODE should be tied high, or low in a system, as the clock mode is not latched by the processor. If left unconnected, the processor will internally pull the CLKMODE pin low (0), enabling the two-x clock mode.
PCLK2 **PCLK1**	O S H(Q) R(Q)	**PROCESSOR OUTPUT CLOCKS** provide a timing reference for all inputs and outputs of the processor. All inputs and output timings are specified in relation to PCLK2 and PCLK1. PCLK2 and PCLK1 are identical signals. Two output pins are provided to allow flexibility in the system's allocation of capacitive loading on the clock. PCLK2:1 may also be connected at the processor to form a single clock signal.
V_{SS}	–	**GROUND** connections consist of 24 pins which must be connected externally to a V_{SS} board plane.
V_{CC}	–	**POWER** connections consist of 24 pins which must be connected externally to a V_{CC} board plane.
N/C	–	**NO CONNECT** pins must not be connected in a system.

Table 4. 80960CA Pin Description—DMA and Interrupt Unit Control Signals

Name	Type	Description
DREQ3 DREQ2 DREQ1 DREQ0	I A(L) H(Z) R(Z)	**DMA REQUEST** causes a DMA transfer to be requested. Each of the four signals requests a transfer on a single channel. DREQ0 requests channel 0, DREQ1 requests channel 1, etc. When two or more channels are requested simultaneously, the channel with the highest priority is serviced first. The channel priority mode is programmable.
DACK3 DACK2 DACK1 DACK0	O S H(1) R(1)	**DMA ACKNOWLEDGE** indicates that a DMA transfer is being executed. Each of the four signals acknowledges a transfer for a single channel. DACK0 acknowledges channel 0, DACK1 acknowledges channel 1, etc. DACK3:0 are active (0) when the requesting device of a DMA is accessed.
EOP3/TC3 EOP2/TC2 EOP1/TC1 EOP0/TC0	I / O A(L) H(Z/Q) R(Z)	**END OF PROCESS/TERMINAL COUNT** can be programmed as either an input (EOP3:0) or as an output (TC3:0), but not both. Each pin is individually programmable. When programmed as an input, EOPx causes the termination of a current DMA transfer for the channel corresponding to the EOPx pin. EOP0 corresponds to channel 0, EOP1 corresponds to channel 1, etc. When a channel is configured for source *and* destination chaining, the EOP pin for that channel causes termination of only the current buffer transferred and causes the next buffer to be transferred. EOP3:0 are asynchronous inputs. When programmed as an output, the channel's TCx pin indicates that the channel byte count has reached 0 and a DMA has terminated. TCx is driven with the same timing as DACKx during the last DMA transfer for a buffer. If the last bus request is executed as multiple bus accesses, TCx will stay asserted for the entire bus request.
XINT7 XINT6 XINT5 XINT4 XINT3 XINT2 XINT1 XINT0	I A(E/L) H(Z) R(Z)	**EXTERNAL INTERRUPT PINS** cause interrupts to be requested. These pins can be configured in three modes. In the Dedicated Mode, each pin is a dedicated external interrupt source. Dedicated inputs can be individually programmed to be level (low) or edge (falling) activated. In the Expanded Mode, the 8 pins act together as an 8-bit vectored interrupt source. The interrupt pins in this mode are level activated. Since the interrupt pins are active low, the vector number requested is the one's complement of the positive logic value place on the port. This eliminates glue logic to interface to combinational priority encoders which output negative logic. In the Mixed Mode, XINT7:5 are dedicated sources and XINT4:0 act as the 5 most significant bits of an expanded mode vector. The least significant bits are set to 010 internally.
NMI	I A(E) H(Z) R(Z)	**NON-MASKABLE INTERRUPT** causes a non-maskable interrupt event to occur. NMI is the highest priority interrupt recognized. NMI is an edge (falling) activated source.

3.3. 80960CA Pinout

3.3.1 80960CA CPGA PINOUT

Tables 5 and 6 list the 80960CA pin names with package location. Figure 4-a depicts the complete 80960CA pinout as viewed from the top side of the component (i.e., pins facing down). Figure 4b shows the complete 80960CA pinout as viewed from the pin-side of the package (i.e., pins facing up). See **Section 4.0, Electrical Specifications** for specifications and recommended connections.

Table 5. PGA Pin Name with Package Location (Signal Order)

Address Bus	Data Bus	Bus Control	Processor Control	I/O
Name . . Location	*Name . . Location*	*Name . . Location*	*Name . . . . Location*	*Name . . Location*
A31 . . . . . S15	D31 . . . . . R03	$\overline{BE3}$ S05	$\overline{RESET}$ A16	$\overline{DREQ3}$ A07
A30 . . . . . Q13	D30 . . . . . Q05	$\overline{BE2}$ S06		$\overline{DREQ2}$ B06
A29 . . . . . R14	D29 . . . . . S02	$\overline{BE1}$ S07	$\overline{FAIL}$ A02	$\overline{DREQ1}$ A06
A28 . . . . . Q14	D28 . . . . . Q04	$\overline{BE0}$ R09		$\overline{DREQ0}$ B05
A27 . . . . . S16	D27 . . . . . R02		STEST B02	
A26 . . . . . R15	D26 . . . . . Q03	W/$\overline{R}$ S10		$\overline{DACK3}$ A10
A25 . . . . . S17	D25 . . . . . S01		$\overline{ONCE}$ C03	$\overline{DACK2}$ A09
A24 . . . . . Q15	D24 . . . . . R01	$\overline{ADS}$ R06		$\overline{DACK1}$ A08
A23 . . . . . R16	D23 . . . . . Q02		CKLIN C13	$\overline{DACK0}$ B08
A22 . . . . . R17	D22 . . . . . P03	$\overline{READY}$ S03	CLKMODE C14	
A21 . . . . . Q16	D21 . . . . . Q01	$\overline{BTERM}$ R04	PCLK1 . . . . . B14	EOP/$\overline{TC0}$ . . . A11
A20 . . . . . P15	D20 . . . . . P02		PCLK2 . . . . . B13	EOP/$\overline{TC1}$ . . . A12
A19 . . . . . P16	D19 . . . . . P01	$\overline{WAIT}$ S12		EOP/$\overline{TC2}$ . . . A13
A18 . . . . . Q17	D18 . . . . . N02	$\overline{BLAST}$ S08	**V_{SS}**	EOP/$\overline{TC3}$ . . . A14
A17 . . . . . P17	D17 . . . . . N01		*Location*	
A16 . . . . . N16	D16 . . . . . M01	DT/$\overline{R}$ S11	C07, C08, C09,	$\overline{XINT7}$ C17
A15 . . . . . N17	D15 . . . . . L01	$\overline{DEN}$ S09	C10, C11, C12,	$\overline{XINT6}$ C16
A14 . . . . . M17	D14 . . . . . L02		F15, G03, G15,	$\overline{XINT5}$ B17
A13 . . . . . L16	D13 . . . . . K01	$\overline{LOCK}$ S14	H03, H15, J03,	$\overline{XINT4}$ C15
A12 . . . . . L17	D12 . . . . . J01		J15, K03, K15,	$\overline{XINT3}$ B16
A11 . . . . . K17	D11 . . . . . H01	HOLD R05	L03, L15, M03,	$\overline{XINT2}$ A17
A10 . . . . . J17	D10 . . . . . H02	HOLDA S04	M15, Q07, Q08,	$\overline{XINT1}$ A15
A9 . . . . . H17	D9 . . . . . G01	BREQ R13	Q09, Q10, Q11	$\overline{XINT0}$ B15
A8 . . . . . G17	D8 . . . . . F01		**V_{CC}**	
A7 . . . . . G16	D7 . . . . . E01	D/$\overline{C}$ S13	*Location*	$\overline{NMI}$ D15
A6 . . . . . F17	D6 . . . . . F02	$\overline{DMA}$ R12	B07, B09, B10,	
A5 . . . . . E17	D5 . . . . . D01	$\overline{SUP}$ Q12	B11, B12, C06,	
A4 . . . . . E16	D4 . . . . . E02		E15, F03, F16,	
			G02, H16, J02,	
			J16, K02, K16, M02,	
			M16, N03, N15,	
			Q06, R07, R08,	
			R10, R11	
A3 . . . . . D17	D3 . . . . . C01	$\overline{BOFF}$ B01	**No Connect**	
A2 . . . . . D16	D2 . . . . . D02		*Location*	
	D1 . . . . . C02		A01, A03, A04, A05,	
			B03, B04, C04, C05,	
			D03	
	D0 . . . . . E03			

Table 6. PGA Pin Name with Package Location (Pin Order)

Address Bus	Data Bus	Bus Control	Processor Control	I/O
Location . . Name	*Location . . Name*	*Location . . Name*	*Location . . . . Name*	*Location . . Name*
A01 . . . . . NC	C01 . . . . . D3	G01 . . . . . D9	M01 . . . . . D16	R01 . . . . . D24
A02 . . . . . $\overline{FAIL}$	C02 . . . . . D1	G02 . . . . . V_{CC}	M02 . . . . . V_{CC}	R02 . . . . . D27
A03 . . . . . NC	C03 . . . . . $\overline{ONCE}$	G03 . . . . . V_{SS}	M03 . . . . . V_{SS}	R03 . . . . . D31
A04 . . . . . NC	C04 . . . . . NC	G15 . . . . . V_{SS}	M15 . . . . . V_{SS}	R04 . . . . . $\overline{BTERM}$
A05 . . . . . NC	C05 . . . . . NC	G16 . . . . . A7	M16 . . . . . V_{CC}	R05 . . . . . HOLD
A06 . . . . . $\overline{DREQ1}$	C06 . . . . . V_{CC}	G17 . . . . . A8	M17 . . . . . A14	R06 . . . . . $\overline{ADS}$
A07 . . . . . $\overline{DREQ3}$	C07 . . . . . V_{SS}			R07 . . . . . V_{CC}
A08 . . . . . $\overline{DACK1}$	C08 . . . . . V_{SS}	H01 . . . . . D11	N01 . . . . . D17	R08 . . . . . V_{CC}
A09 . . . . . $\overline{DACK2}$	C09 . . . . . V_{SS}	H02 . . . . . D10	N02 . . . . . D18	R09 . . . . . $\overline{BE0}$
A10 . . . . . $\overline{DACK3}$	C10 . . . . . V_{SS}	H03 . . . . . V_{SS}	N03 . . . . . V_{CC}	R10 . . . . . V_{CC}
A11 . . . $\overline{EOP/TC0}$	C11 . . . . . V_{SS}	H15 . . . . . V_{SS}	N15 . . . . . V_{CC}	R11 . . . . . V_{CC}
A12 . . . $\overline{EOP/TC1}$	C12 . . . . . V_{SS}	H16 . . . . . V_{CC}	N16 . . . . . A16	R12 . . . . . $\overline{DMA}$
A13 . . . $\overline{EOP/TC2}$	C13 . . . . . CLKIN	H17 . . . . . A9	N17 . . . . . A15	R13 . . . . . BREQ
A14 . . . $\overline{EOP/TC3}$	C14 . . CLKMODE			R14 . . . . . A29
A15 . . . . . $\overline{XINT1}$	C15 . . . . . $\overline{XINT4}$	J01 . . . . . D12	P01 . . . . . D19	R15 . . . . . A26
A16 . . . . . $\overline{RESET}$	C16 . . . . . $\overline{XINT6}$	J02 . . . . . V_{CC}	P02 . . . . . D20	R16 . . . . . A23
A17 . . . . . $\overline{XINT2}$	C17 . . . . . $\overline{XINT7}$	J03 . . . . . V_{SS}	P03 . . . . . D22	R17 . . . . . A22
		J15 . . . . . V_{SS}	P15 . . . . . A20	
B01 . . . . . $\overline{BOFF}$	D01 . . . . . D5	J16 . . . . . V_{CC}	P16 . . . . . A19	S01 . . . . . D25
B02 . . . . . STEST	D02 . . . . . D2	J17 . . . . . A10	P17 . . . . . A17	S02 . . . . . D29
B03 . . . . . NC	D03 . . . . . NC			S03 . . . . . $\overline{READY}$
B04 . . . . . NC	D15 . . . . . $\overline{NMI}$	K01 . . . . . D13	Q01 . . . . . D21	S04 . . . . . HOLDA
B05 . . . . . $\overline{DREQ0}$	D16 . . . . . A2	K02 . . . . . V_{CC}	Q02 . . . . . D23	S05 . . . . . $\overline{BE3}$
B06 . . . . . $\overline{DREQ2}$	D17 . . . . . A3	K03 . . . . . V_{SS}	Q03 . . . . . D26	S06 . . . . . $\overline{BE2}$
B07 . . . . . V_{CC}		K15 . . . . . V_{SS}	Q04 . . . . . D28	S07 . . . . . $\overline{BE1}$
B08 . . . . . $\overline{DACK0}$	E01 . . . . . D7	K16 . . . . . V_{CC}	Q05 . . . . . D30	S08 . . . . . $\overline{BLAST}$
B09 . . . . . V_{CC}	E02 . . . . . D4	K17 . . . . . A11	Q06 . . . . . V_{CC}	S09 . . . . . $\overline{DEN}$
B10 . . . . . V_{CC}	E03 . . . . . D0		Q07 . . . . . V_{SS}	S10 . . . . . W/$\overline{R}$
B11 . . . . . V_{CC}	E15 . . . . . V_{CC}	L01 . . . . . D15	Q08 . . . . . V_{SS}	S11 . . . . . DT/$\overline{R}$
B12 . . . . . V_{CC}	E16 . . . . . A4	L02 . . . . . D14	Q09 . . . . . V_{SS}	S12 . . . . . $\overline{WAIT}$
B13 . . . . . PCLK2	E17 . . . . . A5	L03 . . . . . V_{SS}	Q10 . . . . . V_{SS}	S13 . . . . . D/$\overline{C}$
B14 . . . . . PCLK1		L15 . . . . . V_{SS}	Q11 . . . . . V_{SS}	S14 . . . . . $\overline{LOCK}$
B15 . . . . . $\overline{XINT0}$	F01 . . . . . D8	L16 . . . . . A13	Q12 . . . . . $\overline{SUP}$	S15 . . . . . A31
B16 . . . . . $\overline{XINT3}$	F02 . . . . . D6	L17 . . . . . A12	Q13 . . . . . A30	S16 . . . . . A27
B17 . . . . . $\overline{XINT5}$	F03 . . . . . V_{CC}		Q14 . . . . . A28	S17 . . . . . A25
	F15 . . . . . V_{SS}		Q15 . . . . . A24	
	F16 . . . . . V_{CC}		Q16 . . . . . A21	
	F17 . . . . . A6		Q17 . . . . . A18	

3

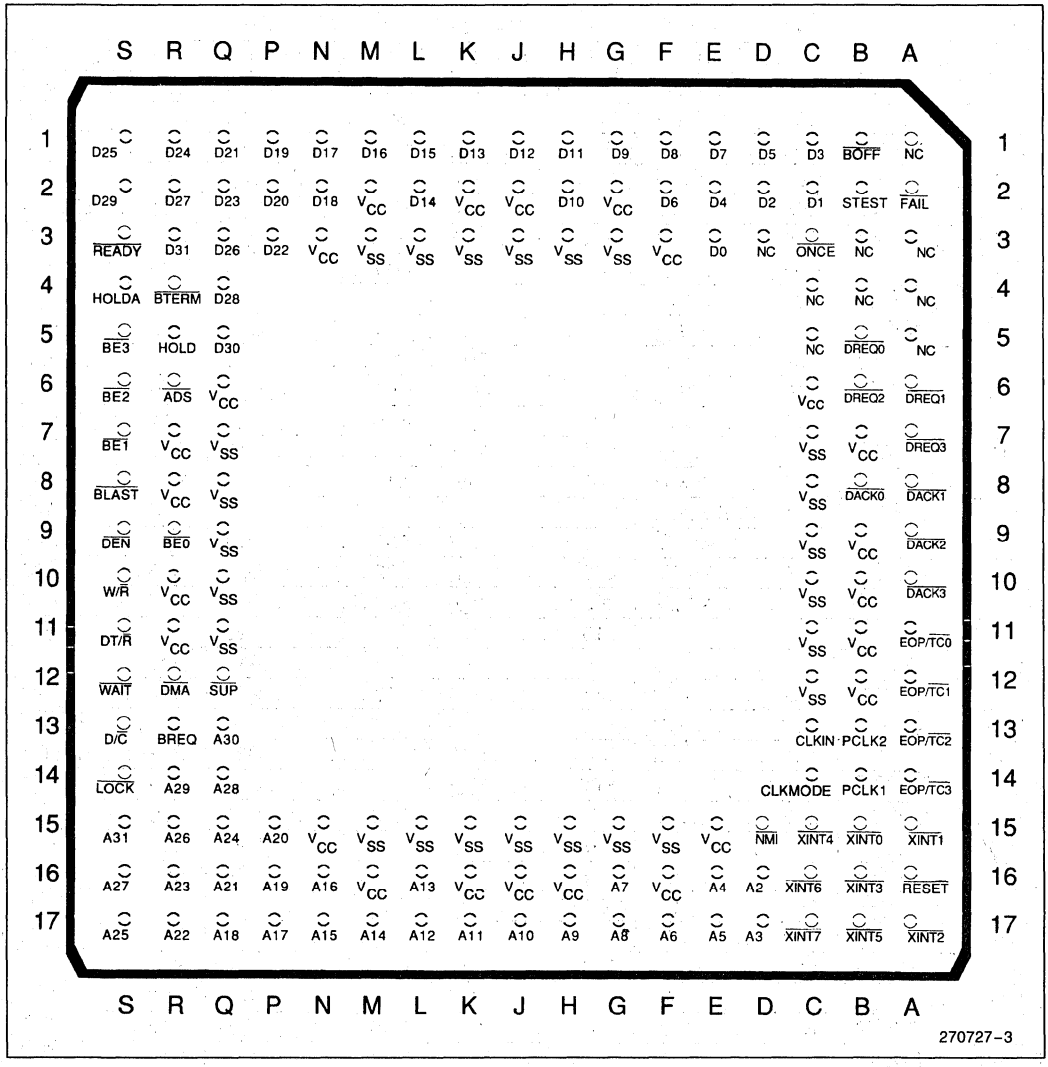


Figure 4a. 80960CA PGA Pinout (View from Top Side)

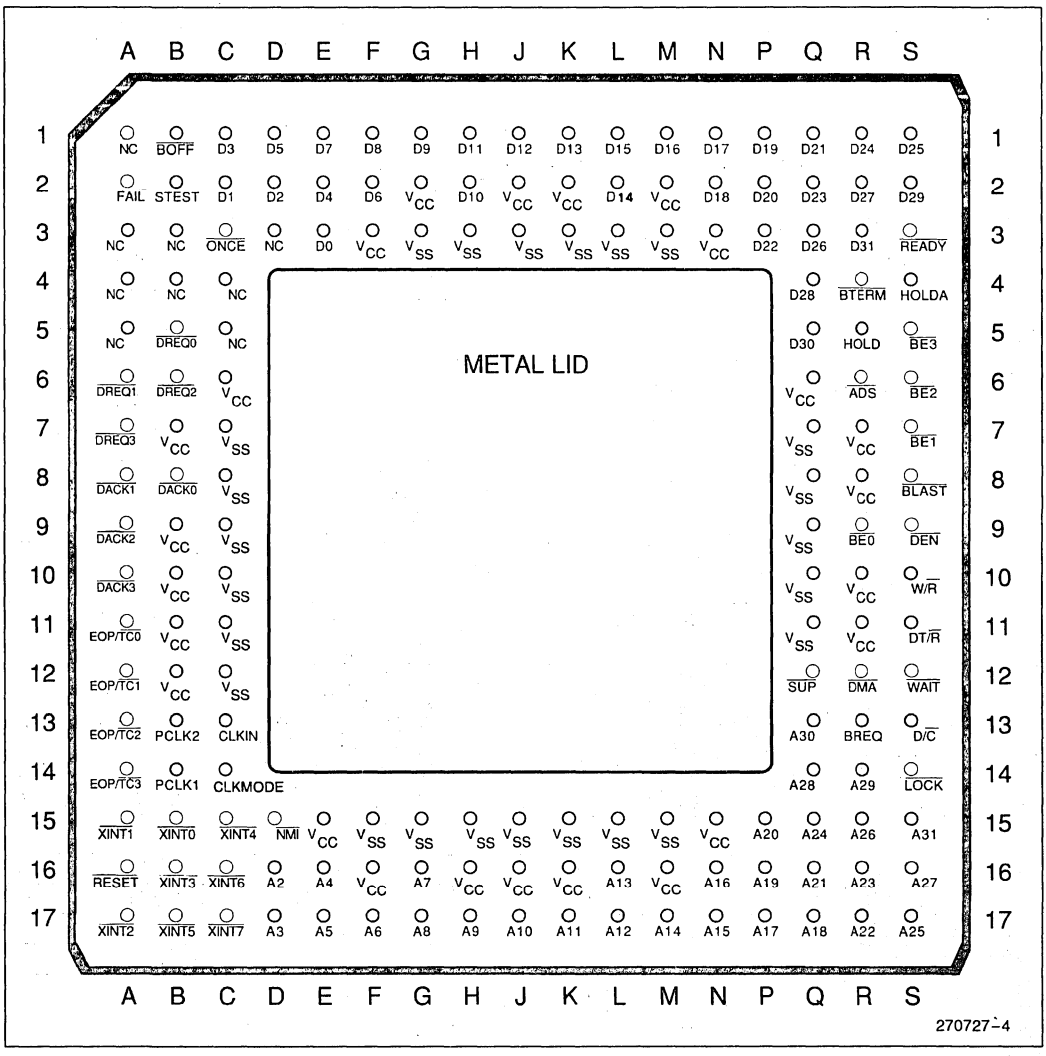


Figure 4b. 80960CA PGA Pinout (View from Bottom Side)

3.3.2 80960CA PQFP Pinout

Tables 7 and 8 list the 80960CA pin names with package location.

See **Section 4.0, Electrical Specifications** for specifications and recommended connections.

Table 7. PQFP Pin Name with Package Location (Pin Order)

Address Bus	Data Bus	Bus Control	Processor Control	I/O
Name . . Location	*Name . . Location*	*Name . . Location*	*Name . . . . . Location*	*Name . . Location*
A31 153	D31 186	$\overline{BE3}$ 176	$\overline{RESET}$ 091	$\overline{DREQ3}$ 060
A30 152	D30 187	$\overline{BE2}$ 175		$\overline{DREQ2}$ 059
A29 151	D29 188	$\overline{BE1}$ 172	$\overline{FAIL}$ 045	$\overline{DREQ1}$ 058
A28 145	D28 189	$\overline{BE0}$ 170		$\overline{DREQ0}$ 057
A27 144	D27 191		STEST 046	
A26 143	D26 192	W/$\overline{R}$ 164		$\overline{DACK3}$ 065
A25 142	D25 194		$\overline{ONCE}$ 043	$\overline{DACK2}$ 064
A24 141	D24 195	$\overline{ADS}$ 178		$\overline{DACK1}$ 063
A23 139	D23 003		CLKIN 087	$\overline{DACK0}$ 062
A22 138	D22 004	$\overline{READY}$ 182	CLKMODE 085	
A21 137	D21 005	$\overline{BTERM}$ 184	PCLK1 078	$\overline{EOP}$/$\overline{TC3}$ . . . 069
A20 136	D20 006		PCLK2 074	$\overline{EOP}$/$\overline{TC2}$ . . . 068
A19 134	D19 008	$\overline{WAIT}$ 162		$\overline{EOP}$/$\overline{TC1}$ . . . 067
A18 133	D18 009	$\overline{BLAST}$ 169	**V_{SS}**	$\overline{EOP}$/$\overline{TC0}$ . . . 066
A17 132	D17 010		*Location*	
A16 130	D16 011	DT/$\overline{R}$ 163	2, 7, 16, 24, 30, 38, 39, 49, 56, 70, 75, 77, 81, 83, 88, 89, 92, 98, 105, 109, 110, 121, 125, 131, 135, 147, 150, 161, 165, 173, 174, 185, 196	$\overline{XINT7}$ 107
A15 129	D15 013	$\overline{DEN}$ 167		$\overline{XINT6}$ 106
A14 128	D14 014			$\overline{XINT5}$ 102
A13 124	D13 015	$\overline{LOCK}$ 156		$\overline{XINT4}$ 101
A12 123	D12 017			$\overline{XINT3}$ 100
A11 122	D11 018	HOLD 181		$\overline{XINT2}$ 095
A10 120	D10 019	HOLDA 179	**V_{CC}**	$\overline{XINT1}$ 094
A9 119	D9 021	BREQ 155	*Location*	$\overline{XINT0}$ 093
A8 118	D8 022		1, 12, 20, 28, 32, 37, 44, 50, 61, 71, 72, 79, 82, 96, 99, 103, 115, 127, 140, 148, 154, 168, 171, 180, 190	
A7 117	D7 023	D/$\overline{C}$ 159		$\overline{NMI}$ 108
A6 116	D6 025	$\overline{DMA}$ 160		
A5 114	D5 026	$\overline{SUP}$ 158		
A4 113	D4 027			
A3 112	D3 033	$\overline{BOFF}$ 040	**No Connect**	
A2 111	D2 034		*Location*	
	D1 035		29, 41, 42, 47, 48, 51, 52, 53, 54, 55, 73, 76, 80, 84, 86, 90, 97, 104, 126, 146, 149, 157, 166, 177, 183, 193	
	D0 036			

Table 8. PQFP Pin Name with Package Location (Pin Order)

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	V_{CC}	50	V_{CC}	99	V_{CC}	148	V_{CC}
2	V_{SS}	51	NC	100	$\overline{XINT3}$	149	NC
3	D23	52	NC	101	$\overline{XINT4}$	150	V_{SS}
4	D22	53	NC	102	$\overline{XINT5}$	151	A29
5	D21	54	NC	103	V_{CC}	152	A30
6	D20	55	NC	104	NC	153	A31
7	V_{SS}	56	V_{SS}	105	V_{SS}	154	V_{CC}
8	D19	57	$\overline{DREQ0}$	106	$\overline{XINT6}$	155	BREQ
9	D18	58	$\overline{DREQ1}$	107	$\overline{XINT7}$	156	$\overline{LOCK}$
10	D17	59	$\overline{DREQ2}$	108	$\overline{NMI}$	157	NC
11	D16	60	$\overline{DREQ3}$	109	V_{SS}	158	$\overline{SUP}$
12	V_{CC}	61	V_{CC}	110	V_{SS}	159	D/$\overline{C}$
13	D15	62	$\overline{DACK0}$	111	A2	160	$\overline{DMA}$
14	D14	63	$\overline{DACK1}$	112	A3	161	V_{SS}
15	D13	64	$\overline{DACK2}$	113	A4	162	$\overline{WAIT}$
16	V_{SS}	65	$\overline{DACK3}$	114	A5	163	DT/$\overline{R}$
17	D12	66	$\overline{EOP0}$/$\overline{TC0}$	115	V_{CC}	164	W/$\overline{R}$
18	D11	67	$\overline{EOP1}$/$\overline{TC1}$	116	A6	165	V_{SS}
19	D10	68	$\overline{EOP2}$/$\overline{TC2}$	117	A7	166	NC
20	V_{CC}	69	$\overline{EOP3}$/$\overline{TC3}$	118	A8	167	$\overline{DEN}$
21	D9	70	V_{SS}	119	A9	168	V_{CC}
22	D8	71	V_{CC}	120	A10	169	$\overline{BLAST}$
23	D7	72	V_{CC}	121	V_{SS}	170	$\overline{BE0}$
24	V_{SS}	73	NC	122	A11	171	V_{CC}
25	D6	74	PCLK2	123	A12	172	$\overline{BE1}$
26	D5	75	V_{SS}	124	A13	173	V_{SS}
27	D4	76	NC	125	V_{SS}	174	V_{SS}
28	V_{CC}	77	V_{SS}	126	NC	175	$\overline{BE2}$
29	NC	78	PCLK1	127	V_{CC}	176	$\overline{BE3}$
30	V_{SS}	79	V_{CC}	128	A14	177	NC
31	NC	80	NC	129	A15	178	$\overline{ADS}$
32	V_{CC}	81	V_{SS}	130	A16	179	HLDA
33	D3	82	V_{CC}	131	V_{SS}	180	V_{CC}
34	D2	83	V_{SS}	132	A17	181	HOLD
35	D1	84	NC	133	A18	182	$\overline{READY}$
36	D0	85	CLKMODE	134	A19	183	NC
37	V_{CC}	86	NC	135	V_{SS}	184	$\overline{BTERM}$
38	V_{SS}	87	CLKIN	136	A20	185	V_{SS}
39	V_{SS}	88	V_{SS}	137	A21	186	D31
40	$\overline{BOFF}$	89	V_{SS}	138	A22	187	D30
41	NC	90	NC	139	A23	188	D29
42	NC	91	$\overline{RESET}$	140	V_{CC}	189	D28
43	$\overline{ONCE}$	92	V_{SS}	141	A24	190	V_{CC}
44	V_{CC}	93	$\overline{XINT0}$	142	A25	191	D27
45	$\overline{FAIL}$	94	$\overline{XINT1}$	143	A26	192	D26
46	STEST	95	$\overline{XINT2}$	144	A27	193	NC
47	NC	96	V_{CC}	145	A28	194	D25
48	NC	97	NC	146	NC	195	D24
49	V_{SS}	98	V_{SS}	147	V_{SS}	196	V_{SS}

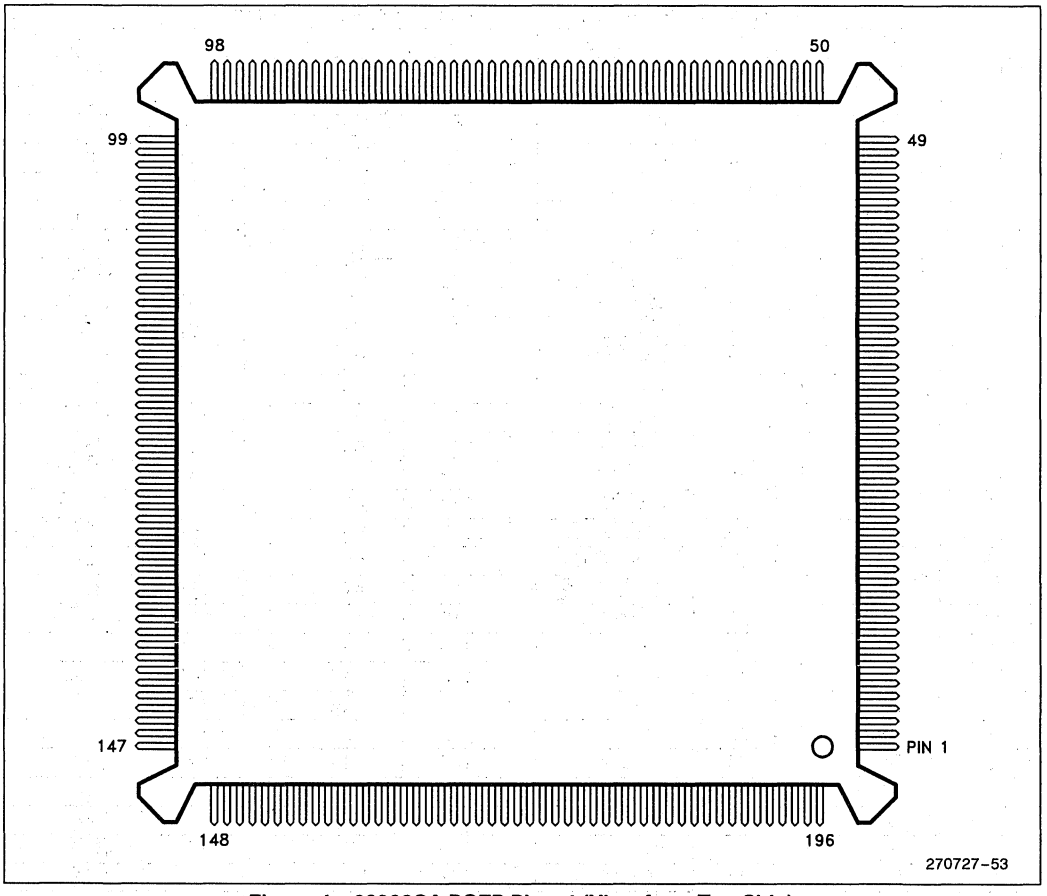
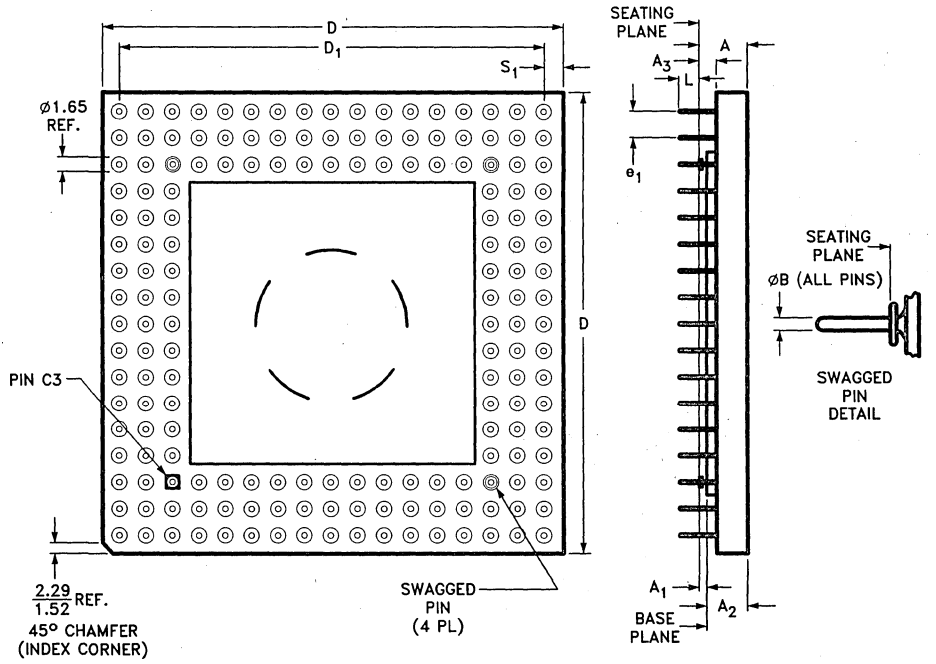


Figure 4c. 80960CA PQFP Pinout (View from Top Side)

3.4. Mechanical Data

3.4.1 CERAMIC PGA PACKAGE



Family: Ceramic Pin Grid Array Package						
Symbol	Millimeters			Inches		
	Min	Max	Notes	Min	Max	Notes
A	3.56	4.57		0.140	0.180	
A_1	0.64	1.14	SOLID LID	0.025	0.045	SOLID LID
A_2	23	0.30	SOLID LID	0.110	0.140	SOLID LID
A_3	1.14	1.40		0.045	0.055	
B	0.43	0.51		0.017	0.020	
D	44.07	44.83		1.735	1.765	
D_1	40.51	40.77		1.595	1.605	
e_1	2.29	2.79		0.090	0.110	
L	2.54	3.30		0.100	0.130	
N	168			168		
S_1	1.52	2.54		0.060	0.100	
ISSUE	IWS REV X 7/15/88					

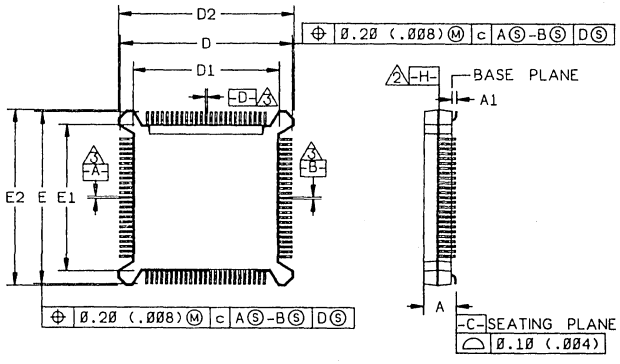
Figure 5. 168-Lead Ceramic PGA Package Dimensions

Table 9. Ceramic PGA Package Dimension Symbols

Letter or Symbol	Description of Dimensions
A	Distance from seating plane to highest point of body
A_1	Distance between seating plane and base plane (lid)
A_2	Distance from base plane to highest point of body
A_3	Distance from seating plane to bottom of body
B	Diameter of terminal lead pin
D	Largest overall package dimension of length
D_1	A body length dimension, outer lead center to outer lead center
e_1	Linear spacing between true lead position centerlines
L	Distance from seating plane to end of lead
S_1	Other body dimension, outer lead center to edge of body

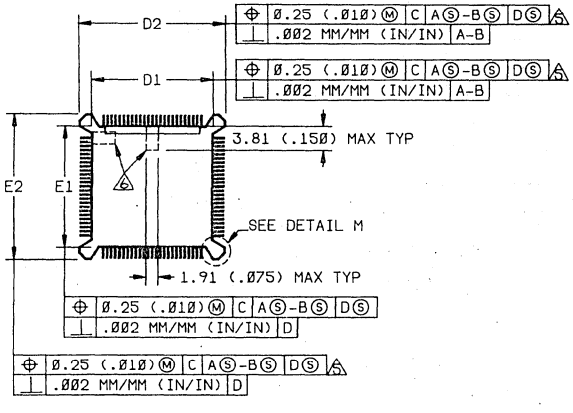
NOTES:
1. Controlling dimension: millimeter.
2. Dimension "e_1" ("e") is non-cumulative.
3. Seating plane (standoff) is defined by P.C. board hole size: 0.0415–0.0430 inch.
4. Dimensions "B", "B_1" and "C" are nominal.
5. Details of Pin 1 identifier are optional.

3.4.2 PLASTIC QUAD FLAT PACKAGE



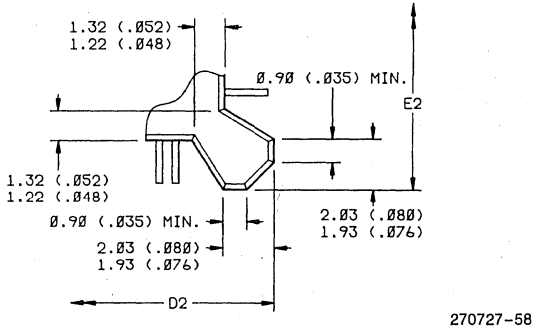
mm (inch)

Figure 6. Principal Dimensions and Datums



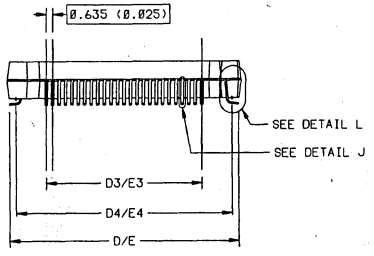
mm (inch)

Figure 7. Molded Details



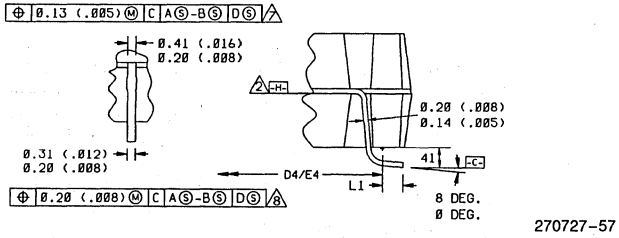
mm (inch)

Figure 8. Detail M



mm (inch)

Figure 9. Terminal Details



mm (inch)

Figure 10. Typical Lead

Table 10. PQFP Package Dimension Symbols

Symbol	Description	Min	Max	Min	Max
N	Leadcount	196		196	
A	Package Height	0.160	0.170	4.06	4.32
A1	Standoff	0.020	0.030	0.51	0.76
D, E	Terminal Dimension	1.475	1.485	37.47	37.72
D1, E1	Package Body	1.347	1.353	34.21	34.37
D2, E2	Bumper Distance	1.497	1.503	38.02	38.18
D3, E3	Lead Dimension	1.200 REF		30.48 REF	
D4, E4	Foot Radius Location	1.423	1.437	36.14	36.49
L1	Foot Length	0.020	0.030	0.51	0.76
Dimension			INCH		mm

NOTES:
1. All dimensions and tolerances conform to ANSI Y14.5M-1982.
2. Datum plane -H- located at the mold parting line and coincident with the bottom of the lead where lead exits plastic body.
3. Datums A-B and -D- to be determined where center leads exit plastic body at datum plane -H-.
4. Controlling Dimension, Inch.
5. Dimensions D1, D2, E1 and E2 are measured at the mold parting line. D1 and E1 do not include an allowable mold protrusion of 0.18 mm (0.007 in) per side. D2 and E2 do not include a total allowable mold protrusion of 0.18 mm (0.007 in) at maximum package size.
6. Pin 1 identifier is located within one of the two zones indicated.
7. Measured at datum plane -H-.
8. Measured at seating plane datum -C-.

3.5. Package Thermal Specifications

The 80960CA is specified for operation when T_C (the case temperature) is within the range of 0°C–100°C. T_C may be measured in any environment to determine whether the 80960CA is within specified operating range. The case temperature should be measured at the center of the top surface, opposite the pins. Refer to Figure 13.

T_A (the ambient temperature) can be calculated from θ_{CA} (thermal resistance from case to ambient) with the following equation:

$$T_A = T_C - P*\theta_{CA}$$

Table 11 shows the maximum T_A allowable (without exceeding T_C) at various airflows and operating frequencies (f_{PCLK}).

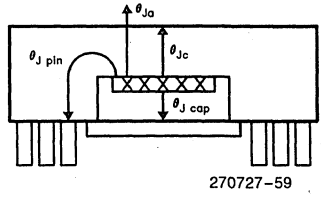
Note that T_A is greatly improved by attaching fins or a heat sink to the package. P (the maximum power consumption) is calculated by using the typical I_{CC} as tabulated in Section 4.4, **DC Characteristics**, and V_{CC} of 5V.

Table 11. Maximum T_A at Various Airflows In °C (PGA Package Only)

		Airflow-ft/min (m/sec)					
	f_{PCLK} (MHz)	0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
T_A with Heat Sink*	33	51	66	79	81	85	87
	25	61	73	83	85	88	89
	16	74	82	89	90	92	93
T_A without Heat Sink	33	36	47	59	66	73	75
	25	49	58	67	73	78	80
	16	66	72	78	82	86	87

* 0.285" high unidirectional heat sink (Al alloy 6061, 50 mil fin width, 150 mil center-to-center fin spacing).

PGA Thermal Resistance—°C/Watt						
Parameter	Airflow—ft./min (m/sec)					
	0 (0)	200 (1.01)	400 (2.03)	600 (3.07)	800 (4.06)	1000 (5.07)
θ Junction-to-Case (Case Measured as shown in Figure 13)	1.5	1.5	1.5	1.5	1.5	1.5
θ Case-to-Ambient (No Heatsink)	17	14	11	9	7.1	6.6
θ Case-to-Ambient (with Unidirectional) Heatsink)*	13	9	5.5	5.0	3.9	3.4



270727-59

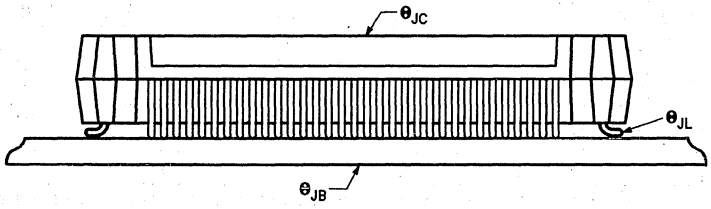
NOTES:
1. This table applies to 80960CA PGA plugged into socket or soldered directly into board.
2. $\theta_{JA} = \theta_{JC} + \theta_{CA}$.
3. $\theta_{J\text{-}CAP}$ = 4°C/W (approx.)
   $\theta_{J\text{-}PIN}$ = 4°C/W (inner pins) (approx.)
   $\theta_{J\text{-}PIN}$ = 8°C/W (outer pins) (approx.)

* 0.285" high unidirectional heat sink (Al alloy 6061, 50 mil fin width, 150 mil center-to-center fin spacing).

Figure 11. 80960CA PGA Package Thermal Characteristics

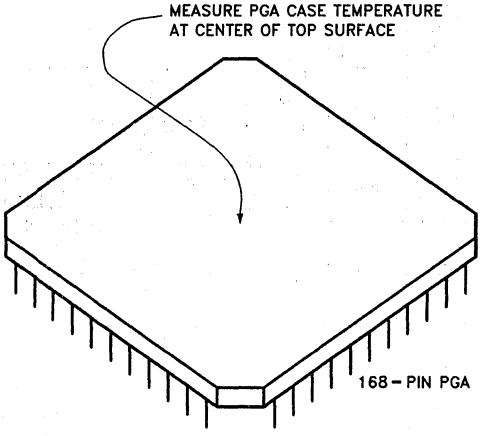
PQFP Thermal Resistance—°C/Watt							
Parameter	Airflow—ft./min (m/sec)						
	0 (0)	50 (0.25)	100 (0.50)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)
θ Junction-to-Case (Case Measured) as shown in Figure 13)	5	5	5	5	5	5	5
θ Case-to-Ambient (No Heatsink)	19	18	17	15	12	10	9

NOTES:
1. This table applies to 80960CA PQFP soldered directly into board.
2. $\theta_{JA} = \theta_{JC} + \theta_{CA}$.
3. $\theta_{JL} = 18°C/Watt$
   $\theta_{JB} = 18°C/Watt$

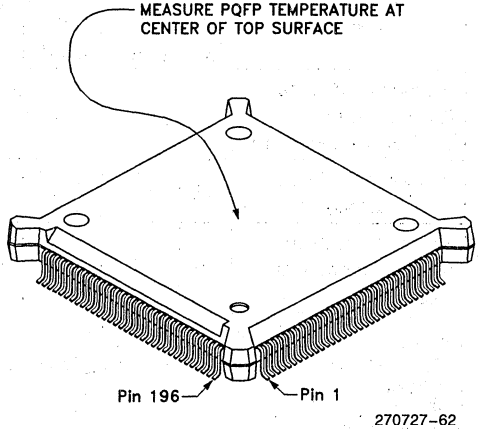


270727-60

Figure 12. 80960CA PQFP Package Thermal Characteristics



270727-61



270727-62

Figure 13. Measuring 80960CA PGA and PQFP Case Temperature

3.6 Stepping Register Information

Upon Reset, Register G0 contains die stepping information. The following figure shows how G0 is configured. The most significant byte contains an ASCII 0. The upper middle byte contains an ASCII C. The lower middle byte contains an ASCII A. The least significant byte contains the stepping number in ASCII. G0 retains this information until it is written over by the user program.

Table 12 contains a cross reference of the number in the least significant byte of register G0 to the die stepping number.

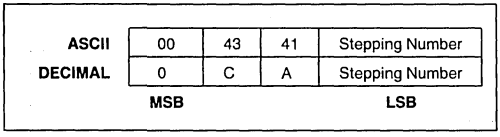


Figure 14. Register G0

Table 12. Die Stepping Cross Reference

G0 Least Significant Byte	Die Stepping
01	B
02	C-1
03	C-2
04	D

3.7 Suggested Sources for 80960CA Accessories

The following are some suggested sources of accessories for the 80960CA. They are not an endorsement of any kind, nor a warranty of the performance of any of the listed products and/or companies.

Sockets

1. 3M Textool Test and Interconnection Products Department
   P.O. Box 2963
   Austin, TX 78769-2963

2. Augat, Inc.
   Interconnection Products Group
   33 Perry Avenue
   P.O. Box 779
   Attleboro, MA 02703
   (508) 222-2202

3. Concept Manufacturing Inc.
   (Decoupling Sockets)
   43024 Christy Street
   Fremont, CA 94538
   (415) 651-3804

Heat Sinks/Fins

1. Thermalloy, Inc.
   2021 West Valley View Lane
   Dallas, TX 75381-0839
   (214) 243-4321

2. E G & G Division
   60 Audubon Road
   Wakefield, MA 01880
   (617) 245-5900

4.0 ELECTRICAL SPECIFICATIONS

NOTICE: This is a production data sheet. The specifications are subject to change without notice.

4.1 Absolute Maximum Ratings

Parameter	Maximum Rating
Storage Temperature	−65 °C to +150 °C
Case Temperature Under Bias	−65 °C to +110 °C
Supply Voltage wrt. V_{SS}	−0.5V to +6.5V
Voltage on Other pins wrt V_{SS}	−0.5V to V_{CC} +0.5V

**WARNING: Stressing the device beyond the "Absolute Maximum Ratings" may cause permanent damage. These are stress ratings only. Operation beyond the "Operating Conditions" is not recommended and extended exposure beyond the "Operating Conditions" may affect device reliability.*

4.2. Operating Conditions

Operating Conditions (80960CA-33, -25, -16)

Symbol	Parameter		Min	Max	Units	Notes
V_{CC}	Supply Voltage	80960CA-33	4.75	5.25	V	
		80960CA-25	4.50	5.50		
		80960CA-16	4.50	5.50		
f_{CLK2x}	Input Clock Frequency (2-x Mode)	80960CA-33	0	66	MHz	
		80960CA-25	0	50	MHz	
		80960CA-16	0	32	MHz	
f_{CLK1x}	Input Clock Frequency (1-x Mode)	80960CA-33	8	33	MHz	(1)
		80960CA-25	8	25	MHz	
		80960CA-16	8	16	MHz	
T_C	Case Temperature Under Bias	PGA Package	0	100	°C	
	80960CA-33, -25, -16	196-Pin PQFP	0	100		

NOTE:
(1) When in the 1-x input clock mode, CLKIN is an input to an internal phase-locked loop and must maintain a minimum frequency of 8 MHz for proper processor operation. However, in the 1-x Mode, CLKIN may still be stopped when the processor either is in a reset condition or is reset. If CLKIN is stopped, the specified $\overline{\text{RESET}}$ low time must be provided once CLKIN restarts and has stabilized.

4.3 Recommended Connections

Power and ground connections must be made to multiple V_{CC} and V_{SS} (GND) pins. Every 80960CA-based circuit board should include power (V_{CC}) and ground (V_{SS}) planes for power distribution. Every V_{CC} pin must be connected to the power plane, and every V_{SS} pin must be connected to the ground plane. Pins identified as "N.C." **must not** be connected in the system.

Liberal decoupling capacitance should be placed near the 80960CA. The processor can cause transient power surges when its numerous output buffers transition, particularly when connected to large capacitive loads.

Low inductance capacitors and interconnects are recommended for best high frequency electrical performance. Inductance can be reduced by shortening the board traces between the processor and decoupling capacitors as much as possible. Capacitors specifically designed for PGA packages will offer the lowest possible inductance.

For reliable operation, always connect unused inputs to an appropriate signal level. In particular, any unused interrupt ($\overline{\text{XINT}}$, $\overline{\text{NMI}}$) or DMA ($\overline{\text{DREQ}}$) input should be connected to V_{CC} through a pull-up resistor, as should $\overline{\text{BTERM}}$ if not used. Pull-up resistors should be in the range of 20 KΩ for each pin tied high. If $\overline{\text{READY}}$ or HOLD are not used, the unused input should be connected to ground. **N.C. pins must always remain unconnected.** Refer to the 80960CA User's Manual for more information.

4.4. DC Specifications

DC Characteristics

(80960CA-33, -25, -16 under the conditions described in **Section 4.2, Operating Conditions.**)

Symbol	Parameter	Min	Max	Units	Notes
V_{IL}	Input Low Voltage for all pins except $\overline{RESET}$	−0.3	0.8	V	
V_{IH}	Input High Voltage for all pins except $\overline{RESET}$	2.0	V_{CC} + 0.3	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 5 mA
V_{OH}	Output High Voltage I_{OH} = −1mA I_{OH} = −200μA	2.4 V_{CC} − 0.5		V V	
V_{ILR}	Input Low Voltage for $\overline{RESET}$	− 0.3	1.5	V	
V_{IHR}	Input High Voltage for $\overline{RESET}$	3.5	V_{CC} + 0.3	V	
I_{LI1}	Input Leakage Current for each pin *except*: $\overline{BTERM}$, $\overline{ONCE}$, $\overline{DREQ3:0}$, STEST, $\overline{EOP3:0}$/$\overline{TC3:0}$, $\overline{NMI}$, $\overline{XINT7:0}$, $\overline{READY}$, HOLD, $\overline{BOFF}$, CLKMODE		±15	μA	0V≤V_{IN}≤V_{CC} (1)
I_{LI2}	Input Leakage Current for: $\overline{BTERM}$, $\overline{ONCE}$, $\overline{DREQ3:0}$, STEST, $\overline{EOP3:0}$/$\overline{TC3:0}$, $\overline{NMI}$, $\overline{XINT7:0}$, $\overline{BOFF}$	0	−300	μA	V_{IN} = 0.45V (2)
I_{LI3}	Input Leakage Current for: $\overline{READY}$, HOLD, CLKMODE	0	500	μA	V_{IN} = 2.4V (3)
I_{LO}	Output Leakage Current		±15	μA	0.45V≤V_{OUT}≤V_{CC}
I_{CC}	Supply Current (80960CA-33) I_{CC} Max I_{CC} Typ		 900 750	 mA	 (4) (5)
I_{CC}	Supply Current (80960CA-25) I_{CC} Max I_{CC} Typ		 750 600	 mA	 (4) (5)
I_{CC}	Supply Current (80960CA-16) I_{CC} Max I_{CC} Typ		 550 400	 mA	 (4) (5)
I_{ONCE}	ONCE-mode Supply Current		100	mA	
C_{IN}	Input Capacitance for: CLKIN, $\overline{RESET}$, $\overline{ONCE}$, $\overline{READY}$, HOLD, $\overline{DREQ3:0}$, $\overline{BOFF}$ $\overline{XINT7:0}$, $\overline{NMI}$, $\overline{BTERM}$, CLKMODE	0	12	pF	F_C = 1 MHz
C_{OUT}	Output Capacitance of each output pin		12	pF	F_C = 1 MHz, (6)
$C_{I/O}$	I/O Pin Capacitance		12	pF	F_C = 1 MHz

NOTES:
(1) No Pull-up or pull-down.
(2) These pins have internal pullup resistors.
(3) These pins have internal pulldown resistors.
(4) Measured at worst case frequency, V_{CC} and temperature, with device operating and outputs loaded to the test conditions described in **Section 4.5.1, AC Test Conditions.**
(5) I_{CC} Typical is not tested.
(6) Output Capacitance is the capacitive load of a floating output.
(7) CLKMODE pin has a pull down resistor only when $\overline{ONCE}$ pin is deasserted.

4.5 AC Specification

AC Characteristics — 80960CA-33

(80960CA-33 only, under the conditions described in **Section 4.2, Operating Conditions** and **Section 4.5.1, AC Test Conditions**.)

Symbol	Parameter		Min	Max	Units	Notes
	INPUT CLOCK(10)					
T_F	CLKIN Frequency		0	66	MHz	(1)
T_C	CLKIN Period	In One-X Mode (f_{CLK1x})	30.3	125	ns	(1,12)
		In Two-X Mode (f_{CLK2x})	15.15	∞	ns	(1)
T_{CS}	CLKIN Period Stability	In One-X Mode (f_{CLK1x})		±0.1%	Δ	(1,13)
T_{CH}	CLKIN High Time	In One-X Mode (f_{CLK1x})	6	62.5	ns	(1,12)
		In Two-X Mode (f_{CLK2x})	6	∞	ns	(1)
T_{CL}	CLKIN Low Time	In One-X Mode (f_{CLK1x})	6	62.5	ns	(1,12)
		In Two-X Mode (f_{CLK2x})	6	∞	ns	(1)
T_{CR}	CLKIN Rise Time		0	6	ns	(1)
T_{CF}	CLKIN Fall Time		0	6	ns	(1)
	OUTPUT CLOCKS(9)					
T_{CP}	CLKIN to PCLK2:1 Delay	In One-X Mode (f_{CLK1x})	−2	2	ns	(1,3,13,14)
		In Two-X Mode (f_{CLK2x})	2	25	ns	(1,3)
T	PCLK2:1 Period	In One-X Mode (f_{CLK1x})	T_C		ns	(1,13)
		In Two-X Mode (f_{CLK2x})	$2T_C$		ns	(1,3)
T_{PH}	PCLK2:1 High Time		(T/2) − 2	T/2	ns	(1,13)
T_{PL}	PCLK2:1 Low Time		(T/2) − 2	T/2	ns	(1,13)
T_{PR}	PCLK2:1 Rise Time		1	4	ns	(1,3)
T_{PF}	PCLK2:1 Fall Time		1	4	ns	(1,3)
	SYNCHRONOUS OUTPUTS(10)					
T_{OV} T_{OH}	Output Valid Delay, Output Hold					(6, 11)
	T_{OV1}, T_{OH1}	A31:2	3	14	ns	
	T_{OV2}, T_{OH2}	$\overline{BE3:0}$	3	16	ns	
	T_{OV3}, T_{OH3}	$\overline{ADS}$	6	18	ns	
	T_{OV4}, T_{OH4}	W/$\overline{R}$	3	18	ns	
	T_{OV5}, T_{OH5}	D/$\overline{C}$, $\overline{SUP}$, $\overline{DMA}$	4	16	ns	
	T_{OV6}, T_{OH6}	$\overline{BLAST}$, $\overline{WAIT}$	5	16	ns	
	T_{OV7}, T_{OH7}	$\overline{DEN}$	3	16	ns	
	T_{OV8}, T_{OH8}	HOLDA, BREQ	4	16	ns	
	T_{OV9}, T_{OH9}	$\overline{LOCK}$	4	16	ns	
	T_{OV10}, T_{OH10}	$\overline{DACK3:0}$, EOP3:0/$\overline{TC3:0}$	3	18	ns	
	T_{OV11}, T_{OH11}	D31:0	3	16	ns	
	T_{OV12}, T_{OH12}	DT/$\overline{R}$	T/2 + 3	T/2 + 14	ns	
	T_{OV13}, T_{OH13}	$\overline{FAIL}$	2	14	ns	(6, 11)
T_{OF}	Output Float for all outputs		3	22	ns	(6)
	SYNCHRONOUS INPUTS(10)					
T_{IS}	Input Setup					
	T_{IS1}	D31:0	3		ns	(1,11)
	T_{IS2}	$\overline{BOFF}$	17		ns	(1,11)
	T_{IS3}	$\overline{BTERM}$/$\overline{READY}$	7		ns	(1,11)
	T_{IS4}	HOLD	7		ns	(1,11)
T_{IH}	Input Hold					
	T_{IH1}	D31:0	5		ns	(1,11)
	T_{IH2}	$\overline{BOFF}$	5		ns	(1,11)
	T_{IH3}	$\overline{BTERM}$/$\overline{READY}$	2		ns	(1,11)
	T_{IH4}	HOLD	3		ns	(1,11)

AC Characteristics — 80960CA-33

80960CA-33 only, under the conditions described in **Section 4.2, Operating Conditions** and **Section 4.5.1, AC Test Conditions.**) (Continued)

Symbol	Parameter	Min	Max	Units	Notes
	RELATIVE OUTPUT TIMINGS(9,7)				
T_{AVSH1}	A31:2 Valid to $\overline{ADS}$ Rising	T − 4	T + 4	ns	
T_{AVSH2}	$\overline{BE3:0}$, W/$\overline{R}$, $\overline{SUP}$, D/$\overline{C}$, $\overline{DMA}$, $\overline{DACK3:0}$ Valid to $\overline{ADS}$ Rising	T − 6	T + 6	ns	
T_{AVEL1}	A31:2 Valid to $\overline{DEN}$ Falling	T − 4	T + 4	ns	
T_{AVEL2}	$\overline{BE3:0}$, W/$\overline{R}$, $\overline{SUP}$, $\overline{INST}$, $\overline{DMA}$, $\overline{DACK3:0}$ Valid to $\overline{DEN}$ Falling	T − 6	T + 6	ns	
T_{NLQV}	$\overline{WAIT}$ Falling to Output Data Valid	±4		ns	
T_{DVNH}	Output Data Valid to $\overline{WAIT}$ Rising	N*T − 4	N*T + 4	ns	(4)
T_{NLNH}	$\overline{WAIT}$ Falling to $\overline{WAIT}$ Rising	N*T ± 4		ns	(4)
T_{NHQX}	Output Data Hold after $\overline{WAIT}$ Rising	(N + 1) * T − 4	(N + 1) * T + 4	ns	(5)
T_{EHTV}	DT/$\overline{R}$ Hold after $\overline{DEN}$ High	T/2 − 4	∞	ns	(6)
T_{TVEL}	DT/$\overline{R}$ Valid to $\overline{DEN}$ Falling	T/2 − 4	T/2 + 4	ns	(7)
	RELATIVE INPUT TIMINGS(7)				
T_{IS5}	$\overline{RESET}$ Input Setup	6		ns	(15)
T_{IH5}	$\overline{RESET}$ Input Hold	5		ns	(15)
T_{IS6}	$\overline{DREQ3:0}$ Input Setup	12		ns	(8)
T_{IH6}	$\overline{DREQ3:0}$ Input Hold	7		ns	(8)
T_{IS7}	$\overline{XINT7:0}$, $\overline{NMI}$ Input Setup	7		ns	(15)
T_{IH7}	$\overline{XINT7:0}$, $\overline{NMI}$ Input Hold	3		ns	(15)

NOTES:
(1) See **Section 4.5.2, AC Timing Waveforms** for waveforms and definitions.
(2) See Figure 22 for capacitive derating information for output delays and hold times.
(3) See Figure 23 for capacitive derating information for rise and fall times.
(4) Where N is the number of N_{RAD}, N_{RDD}, N_{WAD}, or N_{WDD} wait states that are programmed in the Bus Controller Region Table. When there are no wait states in an access, $\overline{WAIT}$ never goes active.
(5) N = Number of wait states inserted with $\overline{READY}$.
(6) Output Data and/or DT/$\overline{R}$ may be driven indefinitely following a cycle if there is no subsequent bus activity.
(7) See Notes 1, 2 and 3.
(8) Since asynchronous inputs are synchronized internally by the 80960CA they have no required setup or hold times in order to be recognized and for proper operation. However, in order to guarantee recognition of the input at a particular rising edge of PCLK2:1 the setup times shown must be met. Asynchronous inputs must be active for at least two consecutive PCLK2:1 rising edges to be seen by the processor.
(9) These specifications are guaranteed by the processor.
(10) These specifications must be met by the system for proper operation of the processor.
(11) This timing is dependent upon the loading of PCLK2:1. Use the derating curves of **Section 4.5.3** to adjust the timing for PCLK2:1 loading.
(12) In the One-x input clock mode the maximum input clock period is limited to 125 ns while the processor is operating. When the processor is in reset, the input clock may stop even in One-x mode.
(13) When in the One-x input clock mode, these specifications assume a stable input clock with a period variation of less than ±0.1% between adjacent cycles.
(14) This parameter is not tested.
(15) Since asynchronous inputs are synchronized internally by the 80960CA, they have no required setup or hold times in order to be recognized and for proper operation. However, in order to guarantee recognition of the input at a particular falling edge of PCLK2:1 the setup times shown must be met. Asynchronous inputs must be active for at least two consecutive PCLK2:1 falling edges to be seen by the processor.

AC Characteristics — 80960CA-25

(80960CA-25 only, under the conditions described in **Section 4.2, Operating Conditions** and **Section 4.5.1, AC Test Conditions**.)

Symbol	Parameter		Min	Max	Units	Notes
	INPUT CLOCK(10)					
T_F	CLKIN Frequency		0	50	MHz	(1)
T_C	CLKIN Period	In One-X Mode (f_{CLK1x})	40	125	ns	(1,12)
		In Two-X Mode (f_{CLK2x})	20	∞	ns	(1)
T_{CS}	CLKIN Period Stability	In One-X Mode (f_{CLK1x})		±0.1%	Δ	(1,13)
T_{CH}	CLKIN High Time	In One-X Mode (f_{CLK1x})	8	62.5	ns	(1,12)
		In Two-X Mode (f_{CLK2x})	8	∞	ns	(1)
T_{CL}	CLKIN Low Time	In One-X Mode (f_{CLK1x})	8	62.5	ns	(1,12)
		In Two-X Mode (f_{CLK2x})	8	∞	ns	(1)
T_{CR}	CLKIN Rise Time		0	6	ns	(1)
T_{CF}	CLKIN Fall Time		0	6	ns	(1)
	OUTPUT CLOCKS(9)					
T_{CP}	CLKIN to PCLK2:1 Delay	In One-X Mode (f_{CLK1x})	−2	2	ns	(1,3,13,14)
		In Two-X Mode (f_{CLK2x})	2	25	ns	(1,3)
T	PCLK2:1 Period	In One-X Mode (f_{CLK1x})	T_C		ns	(1,13)
		In Two-X Mode (f_{CLK2x})	$2T_C$		ns	(1,3)
T_{PH}	PCLK2:1 High Time		(T/2) − 3	T/2	ns	(1,13)
T_{PL}	PCLK2:1 Low Time		(T/2) − 3	T/2	ns	(1,13)
T_{PR}	PCLK2:1 Rise Time		1	4	ns	(1,3)
T_{PF}	PCLK2:1 Fall Time		1	4	ns	(1,3)
	SYNCHRONOUS OUTPUTS(10)					
T_{OV} T_{OH}	Output Valid Delay, Output Hold					(6, 11)
	T_{OV1}, T_{OH1}	A31:2	3	16	ns	
	T_{OV2}, T_{OH2}	$\overline{BE3:0}$	3	18	ns	
	T_{OV3}, T_{OH3}	$\overline{ADS}$	6	20	ns	
	T_{OV4}, T_{OH4}	W/$\overline{R}$	3	20	ns	
	T_{OV5}, T_{OH5}	D/$\overline{C}$,$\overline{SUP}$,$\overline{DMA}$	4	18	ns	
	T_{OV6}, T_{OH6}	$\overline{BLAST}$, $\overline{WAIT}$	5	18	ns	
	T_{OV7}, T_{OH7}	$\overline{DEN}$	3	18	ns	
	T_{OV8}, T_{OH8}	HOLDA, BREQ	4	18	ns	
	T_{OV9}, T_{OH9}	$\overline{LOCK}$	4	18	ns	
	T_{OV10}, T_{OH10}	$\overline{DACK3:0}$, EOP3:0/$\overline{TC3:0}$	4	20	ns	
	T_{OV11}, T_{OH11}	D31:0	3	18	ns	
	T_{OV12}, T_{OH12}	DT/$\overline{R}$	T/2 + 3	T/2 + 16	ns	
	T_{OV13}, T_{OH13}	$\overline{FAIL}$	2	16	ns	(6, 11)
T_{OF}	Output Float for all outputs		3	22	ns	(6)
	SYNCHRONOUS INPUTS(10)					
T_{IS}	Input Setup					
	T_{IS1}	D31:0	5		ns	(1,11)
	T_{IS2}	$\overline{BOFF}$	19		ns	(1,11)
	T_{IS3}	$\overline{BTERM}$/$\overline{READY}$	9		ns	(1,11)
	T_{IS4}	HOLD	9		ns	(1,11)
T_{IH}	Input Hold					
	T_{IH1}	D31:0	5		ns	(1,11)
	T_{IH2}	$\overline{BOFF}$	7		ns	(1,11)
	T_{IH3}	$\overline{BTERM}$/$\overline{READY}$	2		ns	(1,11)
	T_{IH4}	HOLD	5		ns	(1,11)

AC Characteristics — 80960CA-25

(80960CA-25 only, under the conditions described in **Section 4.2, Operating Conditions** and **Section 4.5.1, AC Test Conditions.**) (Continued)

Symbol	Parameter	Min	Max	Units	Notes
	RELATIVE OUTPUT TIMINGS(9,7)				
T_{AVSH1}	A31:2 Valid to $\overline{ADS}$ Rising	T − 4	T + 4	ns	
T_{AVSH2}	$\overline{BE3:0}$, W/$\overline{R}$, $\overline{SUP}$, D/$\overline{C}$, $\overline{DMA}$, $\overline{DACK3:0}$ Valid to $\overline{ADS}$ Rising	T − 6	T + 6	ns	
T_{AVEL1}	A31:2 Valid to $\overline{DEN}$ Falling	T − 4	T + 4	ns	
T_{AVEL2}	$\overline{BE3:0}$, W/$\overline{R}$, $\overline{SUP}$, $\overline{INST}$, $\overline{DMA}$, $\overline{DACK3:0}$ Valid to $\overline{DEN}$ Falling	T − 6	T + 6	ns	
T_{NLQV}	$\overline{WAIT}$ Falling to Output Data Valid	±4		ns	
T_{DVNH}	Output Data Valid to $\overline{WAIT}$ Rising	N*T − 4	N*T + 4	ns	(4)
T_{NLNH}	$\overline{WAIT}$ Falling to $\overline{WAIT}$ Rising	N*T ± 4		ns	(4)
T_{NHQX}	Output Data Hold after $\overline{WAIT}$ Rising	(N + 1) * T − 4	(N + 1) * T + 4	ns	(5)
T_{EHTV}	DT/$\overline{R}$ Hold after $\overline{DEN}$ High	T/2 − 4	∞	ns	(6)
T_{TVEL}	DT/$\overline{R}$ Valid to $\overline{DEN}$ Falling	T/2 − 4	T/2 + 4	ns	(7)
	RELATIVE INPUT TIMINGS(7)				
T_{IS5}	$\overline{RESET}$ Input Setup	8		ns	(15)
T_{IH5}	$\overline{RESET}$ Input Hold	7		ns	(15)
T_{IS6}	$\overline{DREQ3:0}$ Input Setup	14		ns	(8)
T_{IH6}	$\overline{DREQ3:0}$ Input Hold	9		ns	(8)
T_{IS7}	$\overline{XINT7:0}$, $\overline{NMI}$ Input Setup	9		ns	(15)
T_{IH7}	$\overline{XINT7:0}$, $\overline{NMI}$ Input Hold	5		ns	(15)

NOTES:
(1) See **Section 4.5.2, AC Timing Waveforms** for waveforms and definitions.
(2) See Figure 22 for capacitive derating information for output delays and hold times.
(3) See Figure 23 for capacitive derating information for rise and fall times.
(4) Where N is the number of N_{RAD}, N_{RDD}, N_{WAD}, or N_{WDD} wait states that are programmed in the Bus Controller Region Table. When there are no wait states in an access, $\overline{WAIT}$ never goes active.
(5) N = Number of wait states inserted with $\overline{READY}$.
(6) Output Data and/or DT/$\overline{R}$ may be driven indefinitely following a cycle if there is no subsequent bus activity.
(7) See Notes 1, 2 and 3.
(8) Since asynchronous inputs are synchronized internally by the 80960CA they have no required setup or hold times in order to be recognized and for proper operation. However, in order to guarantee recognition of the input at a particular rising edge of PCLK2:1 the setup times shown must be met. Asynchronous inputs must be active for at least two consecutive PCLK2:1 rising edges to be seen by the processor.
(9) These specifications are guaranteed by the processor.
(10) These specifications must be met by the system for proper operation of the processor.
(11) This timing is dependent upon the loading of PCLK2:1. Use the derating curves of **Section 4.5.3** to adjust the timing for PCLK2:1 loading.
(12) In the One-x input clock mode the maximum input clock period is limited to 125 ns while the processor is operating. When the processor is in reset, the input clock may stop even in One-x mode.
(13) When in the One-x input clock mode, these specifications assume a stable input clock with a period variation of less than ±0.1% between adjacent cycles.
(14) This parameter is not tested.
(15) Since asynchronous inputs are synchronized internally by the 80960CA, they have no required setup or hold times in order to be recognized and for proper operation. However, in order to guarantee recognition of the input at a particular falling edge of PCLK2:1 the setup times shown must be met. Asynchronous inputs must be active for at least two consecutive PCLK2:1 falling edges to be seen by the processor.

AC Characteristics — 80960CA-16

(80960CA-16 only, under the conditions described in **Section 4.2, Operating Conditions** and **Section 4.5.1, AC Test Conditions.**) (Continued)

Symbol	Parameter		Min	Max	Units	Notes
	INPUT CLOCK(10)					
T_F	CLKIN Frequency		0	32	MHz	(1)
T_C	CLKIN Period	In One-X Mode (f_{CLK1X})	62.5	125	ns	(1,12)
		In Two-X Mode (f_{CLK2X})	31.25	∞	ns	(1)
T_{CS}	CLKIN Period Stability	In One-X Mode (f_{CLK1X})		±0.1%	Δ	(1,13)
T_{CH}	CLKIN High Time	In One-X Mode (f_{CLK1X})	10	62.5	ns	(1,12)
		In Two-X Mode (f_{CLK2X})	10	∞	ns	(1)
T_{CL}	CLKIN Low Time	In One-X Mode (f_{CLK1X})	10	62.5	ns	(1,12)
		In Two-X Mode (f_{CLK2X})	10	∞	ns	(1)
T_{CR}	CLKIN Rise Time		0	6	ns	(1)
T_{CF}	CLKIN Fall Time		0	6	ns	(1)
	OUTPUT CLOCKS(9)					
T_{CP}	CLKIN to PCLK2:1 Delay	In One-X Mode (f_{CLK1X})	−2	2	ns	(1,3,13,14)
		In Two-X Mode (f_{CLK2X})	2	25	ns	(1,3)
T	PCLK2:1 Period	In One-X Mode (f_{CLK1X})	T_C		ns	(1,13)
		In Two-X Mode (f_{CLK2X})	$2T_C$		ns	(1,3)
T_{PH}	PCLK2:1 High Time		(T/2) − 4	T/2	ns	(1,13)
T_{PL}	PCLK2:1 Low Time		(T/2) − 4	T/2	ns	(1,13)
T_{PR}	PCLK2:1 Rise Time		1	4	ns	(1,3)
T_{PF}	PCLK2:1 Fall Time		1	4	ns	(1,3)
	SYNCHRONOUS OUTPUTS(10)					
T_{OV}	Output Valid Delay, Output Hold					(6, 11)
T_{OH}	T_{OV1}, T_{OH1}	A31:2	3	18	ns	
	T_{OV2}, T_{OH2}	$\overline{BE3:0}$	3	20	ns	
	T_{OV3}, T_{OH3}	$\overline{ADS}$	6	22	ns	
	T_{OV4}, T_{OH4}	W/$\overline{R}$	3	22	ns	
	T_{OV5}, T_{OH5}	D/$\overline{C}$, $\overline{SUP}$, $\overline{DMA}$	4	20	ns	
	T_{OV6}, T_{OH6}	$\overline{BLAST}$, $\overline{WAIT}$	5	20	ns	
	T_{OV7}, T_{OH7}	$\overline{DEN}$	3	20	ns	
	T_{OV8}, T_{OH8}	HOLDA, BREQ	4	20	ns	
	T_{OV9}, T_{OH9}	$\overline{LOCK}$	4	20	ns	
	T_{OV10}, T_{OH10}	$\overline{DACK3:0}$, EOP3:0/$\overline{TC3:0}$	4	22	ns	
	T_{OV11}, T_{OH11}	D31:0	3	20	ns	
	T_{OV12}, T_{OH12}	DT/$\overline{R}$	T/2 + 3	T/2 + 18	ns	
	T_{OV13}, T_{OH13}	$\overline{FAIL}$	2	18	ns	(6, 11)
T_{OF}	Output Float for all outputs		3	22	ns	(6)
	SYNCHRONOUS INPUTS(10)					
T_{IS}	Input Setup					
	T_{IS1}	D31:0	5		ns	(1,11)
	T_{IS2}	$\overline{BOFF}$	21		ns	(1,11)
	T_{IS3}	$\overline{BTERM}$/$\overline{READY}$	9		ns	(1,11)
	T_{IS4}	HOLD	9		ns	(1,11)
T_{IH}	Input Hold					
	T_{IH1}	D31:0	5		ns	(1,11)
	T_{IH2}	$\overline{BOFF}$	7		ns	(1,11)
	T_{IH3}	$\overline{BTERM}$/$\overline{READY}$	2		ns	(1,11)
	T_{IH4}	HOLD	5		ns	(1,11)

AC Characteristics — 80960CA-16

(80960CA-16 only, under the conditions described in **Section 4.2, Operating Conditions** and **Section 4.5.1, AC Test Conditions**.) (Continued)

Symbol	Parameter	Min	Max	Units	Notes
	RELATIVE OUTPUT TIMINGS(9,7)				
T_{AVSH1}	A31:2 Valid to $\overline{ADS}$ Rising	T − 4	T + 4	ns	
T_{AVSH2}	$\overline{BE3:0}$, W/$\overline{R}$, $\overline{SUP}$, D/$\overline{C}$, $\overline{DMA}$, $\overline{DACK3:0}$ Valid to $\overline{ADS}$ Rising	T − 6	T + 6	ns	
T_{AVEL1}	A31:2 Valid to $\overline{DEN}$ Falling	T − 6	T + 6	ns	
T_{AVEL2}	$\overline{BE3:0}$, W/$\overline{R}$, $\overline{SUP}$, $\overline{INST}$, $\overline{DMA}$, $\overline{DACK3:0}$ Valid to $\overline{DEN}$ Falling	T − 6	T + 6	ns	
T_{NLQV}	$\overline{WAIT}$ Falling to Output Data Valid	±4		ns	
T_{DVNH}	Output Data Valid to $\overline{WAIT}$ Rising	N*T − 4	N*T + 4	ns	(4)
T_{NLNH}	$\overline{WAIT}$ Falling to $\overline{WAIT}$ Rising	N*T ± 4		ns	(4)
T_{NHQX}	Output Data Hold after $\overline{WAIT}$ Rising	(N + 1) * T − 4	(N + 1) * T + 4	ns	(5)
T_{EHTV}	DT/$\overline{R}$ Hold after $\overline{DEN}$ High	T/2 − 4	∞	ns	(6)
T_{TVEL}	DT/$\overline{R}$ Valid to $\overline{DEN}$ Falling	T/2 − 4	T/2 + 4	ns	(7)
	RELATIVE INPUT TIMINGS(7)				
T_{IS5}	$\overline{RESET}$ Input Setup	10		ns	(15)
T_{IH5}	$\overline{RESET}$ Input Hold	9		ns	(15)
T_{IS6}	$\overline{DREQ3:0}$ Input Setup	16		ns	(8)
T_{IH6}	$\overline{DREQ3:0}$ Input Hold	11		ns	(8)
T_{IS7}	$\overline{XINT7:0}$, $\overline{NMI}$ Input Setup	9		ns	(15)
T_{IH7}	$\overline{XINT7:0}$, $\overline{NMI}$ Input Hold	5		ns	(15)

NOTES:
(1) See **Section 4.5.2, AC Timing Waveforms** for waveforms and definitions.
(2) See Figure 22 for capacitive derating information for output delays and hold times.
(3) See Figure 23 for capacitive derating information for rise and fall times.
(4) Where N is the number of N_{RAD}, N_{RDD}, N_{WAD}, or N_{WDD} wait states that are programmed in the Bus Controller Region Table. When there are no wait states in an access, $\overline{WAIT}$ never goes active.
(5) N = Number of wait state inserted with $\overline{READY}$.
(6) Output Data and/or DT/$\overline{R}$ may be driven indefinitely following a cycle if there is no subsequent bus activity.
(7) See Notes 1, 2 and 3.
(8) Since asynchronous inputs are synchronized internally by the 80960CA they have no required setup or hold times in order to be recognized and for proper operation. However, in order to guarantee recognition of the input at a particular rising edge of PCLK2:1 the setup times shown must be met. Asynchronous inputs must be active for at least two consecutive PCLK2:1 rising edges to be seen by the processor.
(9) These specifications are guaranteed by the processor.
(10) These specifications must be met by the system for proper operation of the processor.
(11) This timing is dependent upon the loading of PCLK2:1. Use the derating curves of Figure 22 to adjust the timing for PCLK2:1 loading.
(12) In the One-x input clock mode the maximum input clock period is limited to 125 ns while the processor is operating. When the processor is in reset, the input clock may stop even in One-x mode.
(13) When in the One-x input clock mode, these specifications assume a stable input clock with a period variation of less than ±0.1% between adjacent cycles.
(14) This parameter is not tested.
(15) Since asynchronous inputs are synchronized internally by the 80960CA, they have no required setup or hold times in order to be recognized and for proper operation. However, in order to guarantee recognition of the input at a particular falling edge of PCLK2:1 the setup times shown must be met. Asynchronous inputs must be active for at least two consecutive PCLK2:1 falling edges to be seen by the processor.

4.5.1. AC TEST CONDITIONS

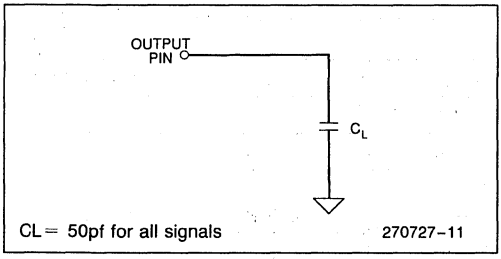


Figure 15. AC Test Load

The AC Specifications in Section 4.5 are tested with the 50 pf load shown in Figure 15. See Figure 22 to see how timings vary with load capacitance.

Specifications are measured at the 1.5V crossing point, unless otherwise indicated. Input waveforms are assumed to have a rise-and-fall time of ≤ 2 ns from 0.8V to 2.0V. See **Section 4.5.2, AC Timing Waveforms**, for AC spec definitions, test points, and illustrations.

4.5.2. AC TIMING WAVEFORMS

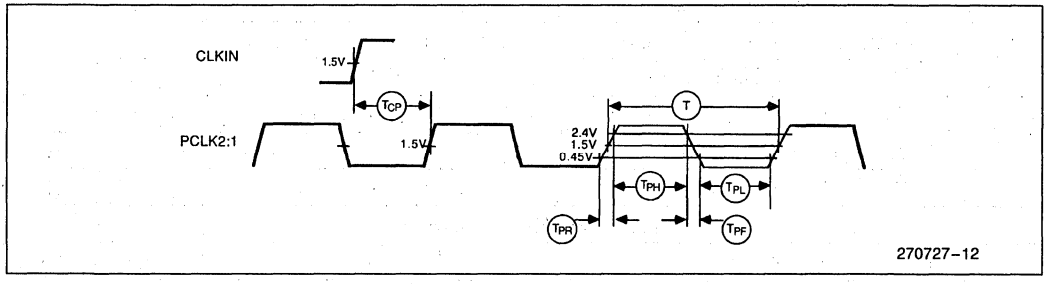


Figure 16a. Input and Output Clocks Waveform

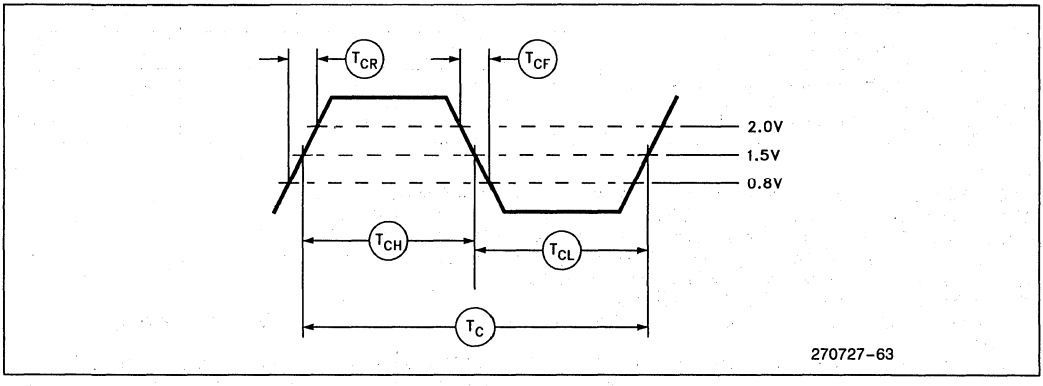


Figure 16b. CLKIN Waveform

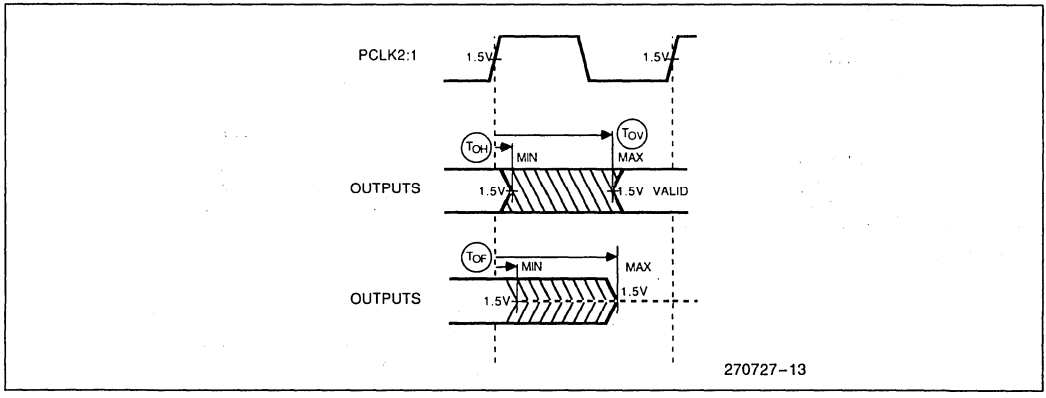


Figure 17. Output Delay and Float Waveform

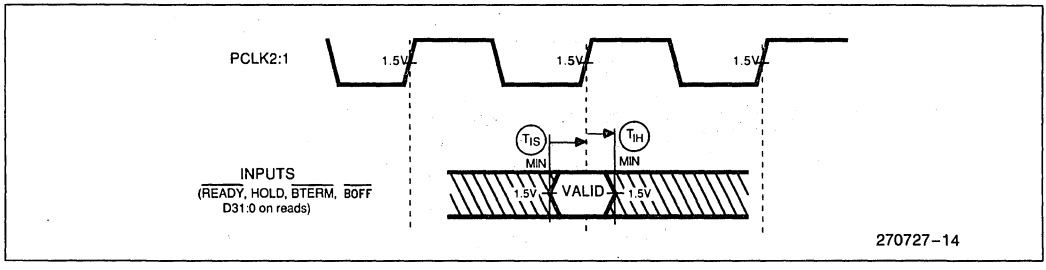


Figure 18a. Input Setup and Hold Waveform

T_{OV} T_{OH} — OUTPUT DELAY — The maximum output delay is referred to as the Output Valid Delay (T_{OV}). The minimum output delay is referred to as the Output Hold (T_{OH}).

T_{OF} — OUTPUT FLOAT DELAY — The output float condition occurs when the maximum output current becomes less than I_{LO} in magnitude.

T_{IS} T_{IH} — INPUT SETUP AND HOLD — The input setup and hold requirements specify the sampling window during which synchronous inputs must be stable for correct processor operation.

270727-64

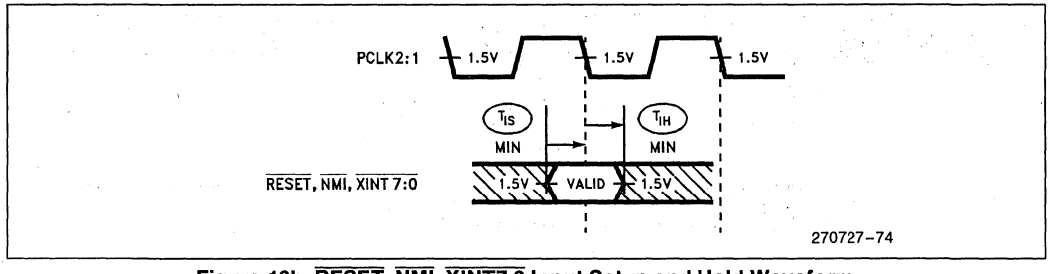
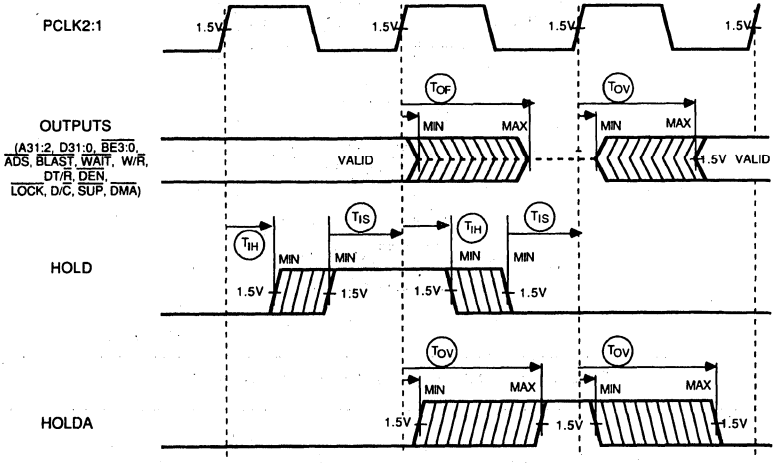


Figure 18b. RESET, NMI, XINT7:0 Input Setup and Hold Waveform



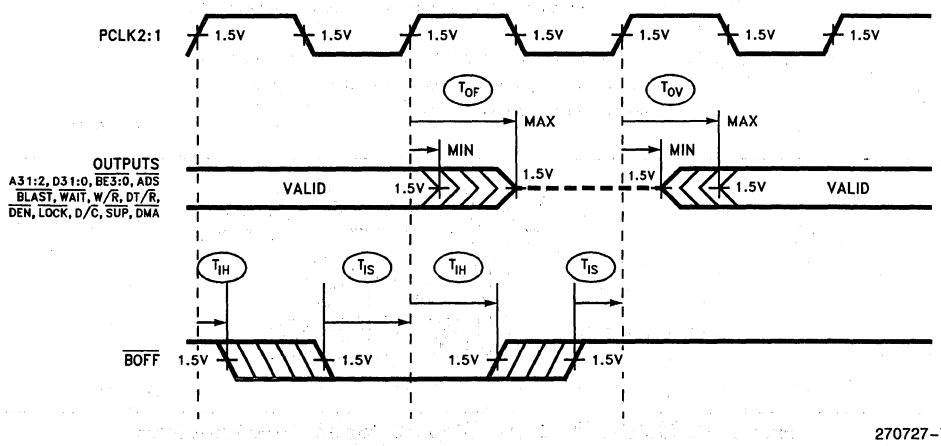
TOV TOH — OUTPUT DELAY — The maximum output delay is referred to as the Output Valid Delay (T_{OV}). The minimum output delay is referred to as the Output Hold (T_{OH}).

TOF — OUTPUT FLOAT DELAY — The output float condition occurs when the maximum output current becomes less than I_{LO} in magnitude.

TIS TIH — INPUT SETUP AND HOLD — The input setup and hold requirements specify the sampling window during which synchronous inputs must be stable for correct processor operation.

270727-15

Figure 19. Hold Acknowledge Timings



270727-75

Figure 20. Bus Back-Off ($\overline{BOFF}$) Timings

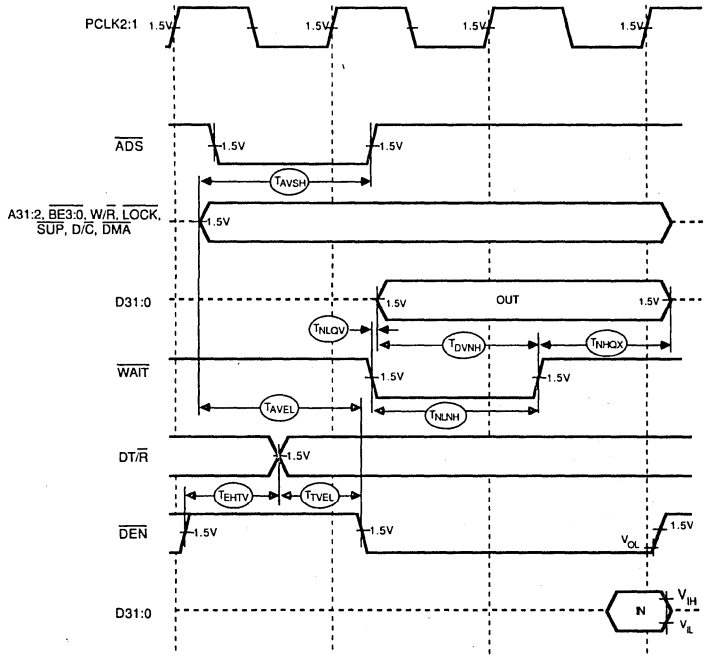
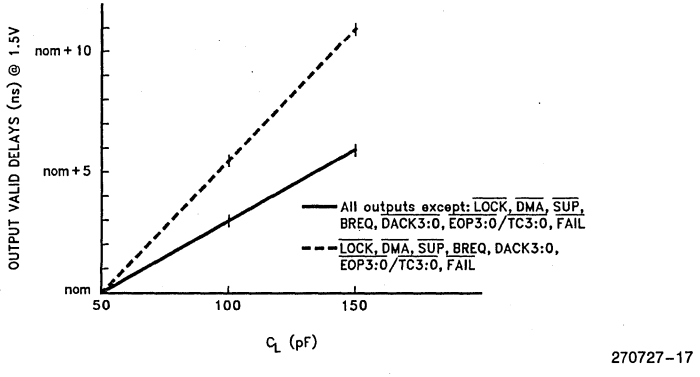


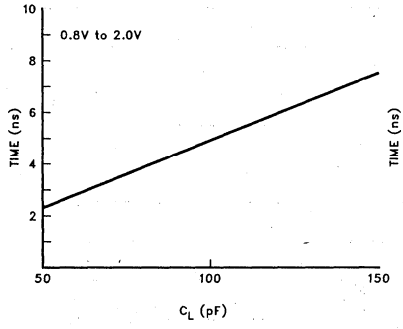
Figure 21. Relative-timings Waveforms

DERATING CURVES

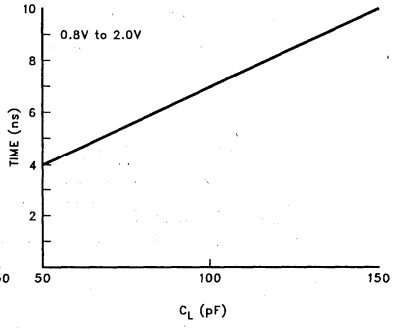


NOTE:
PCLK Load = 50 pF

Figure 22. Output Delay or Hold vs Load Capacitance



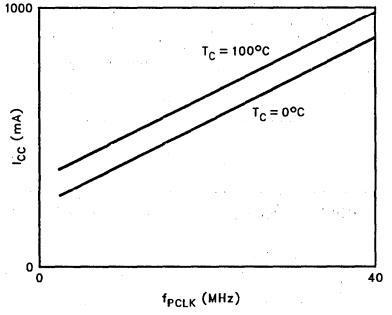
(a) All outputs except: $\overline{LOCK}$, $\overline{DMA}$, $\overline{SUP}$, HOLDA, BREQ, $\overline{DACK3:0}$, $\overline{EOP3:0}/\overline{TC3:0}$, $\overline{FAIL}$



(b) $\overline{LOCK}$, $\overline{DMA}$, $\overline{SUP}$, HOLDA, BREQ, $\overline{DACK3:0}$, $\overline{EOP3:0}/\overline{TC3:0}$, $\overline{FAIL}$

270727-18

Figure 23. Rise and Fall Time Derating at Highest Operating Temperature and Minimum V_{CC}



270727-19

I_{CC}—I_{CC} under test conditions

Figure 24. I_{CC} vs Frequency and Temperature

5.0 RESET, BACKOFF AND HOLD ACKNOWLEDGE

The following table lists the condition of each processor output pin while RESET is asserted (low).

Table 13. Reset Conditions

Pins	State During Reset (HOLDA inactive)[1]
A31:A2	Floating
D31:D0	Floating
BE3:0	Driven high (Inactive)
W/R	Driven low (Read)
ADS	Driven high (Inactive)
WAIT	Driven high (Inactive)
BLAST	Driven low (Active)
DT/R	Driven low (Receive)
DEN	Driven high (Inactive)
LOCK	Driven high (Inactive)
BREQ	Driven low (Inactive)
D/C	Floating
DMA	Floating
SUP	Floating
FAIL	Driven low (Active)
DACK3	Driven high (Inactive)
DACK2	Driven high (Inactive)
DACK1	Driven high (Inactive)
DACK0	Driven high (Inactive)
EOP/TC3	Floating (set to input mode)
EOP/TC2	Floating (set to input mode)
EOP/TC1	Floating (set to input mode)
EOP/TC0	Floating (set to input mode)

NOTE:
(1) With regard to bus output pin state only, the Hold Acknowledge state takes precedence over the reset state. Although asserting the RESET pin will internally reset the processor, the processor's bus output pins will not enter the reset state if it has granted Hold Acknowledge to a previous HOLD request (HOLDA is active). Furthermore, the processor will grant new HOLD requests and enter the Hold Acknowledge state even while in reset.
For example, if HOLDA is not active and the processor is in the reset state, then HOLD is asserted, the processor's bus pins will enter the Hold Acknowledge state and HOLDA will be granted. The processor will not be able to perform memory accesses until the HOLD request is removed, even if the RESET pin is brought high. This operation is provided to simplify boot-up synchronization among multiple processors sharing the same bus.

The following table lists the condition of each processor output pin while HOLDA is asserted (low).

Table 14. Hold Acknowledge and Backoff Conditions

Pins	State During HOLDA
A31:A2	Floating
D31:D0	Floating
BE3:0	Floating
W/R	Floating
ADS	Floating
WAIT	Floating
BLAST	Floating
DT/R	Floating
DEN	Floating
LOCK	Floating
BREQ	Driven (high or low)
D/C	Floating
DMA	Floating
SUP	Floating
FAIL	Driven high (Inactive)
DACK3	Driven high (Inactive)
DACK2	Driven high (Inactive)
DACK1	Driven high (Inactive)
DACK0	Driven high (Inactive)
EOP/TC3	Driven if output
EOP/TC2	Driven if output
EOP/TC1	Driven if output
EOP/TC0	Driven if output

6.0 BUS WAVEFORMS

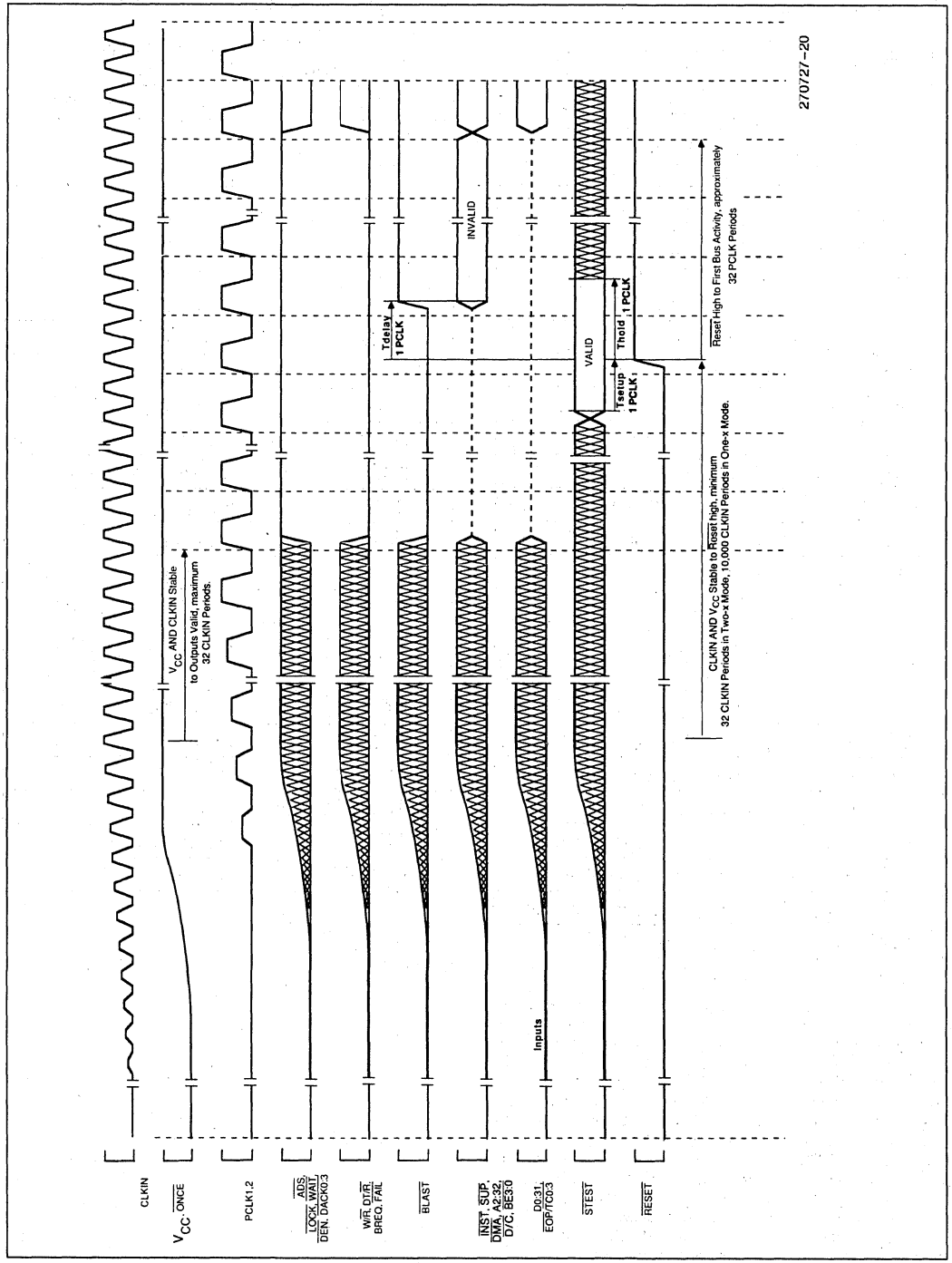


Figure 25. Cold Reset Waveform

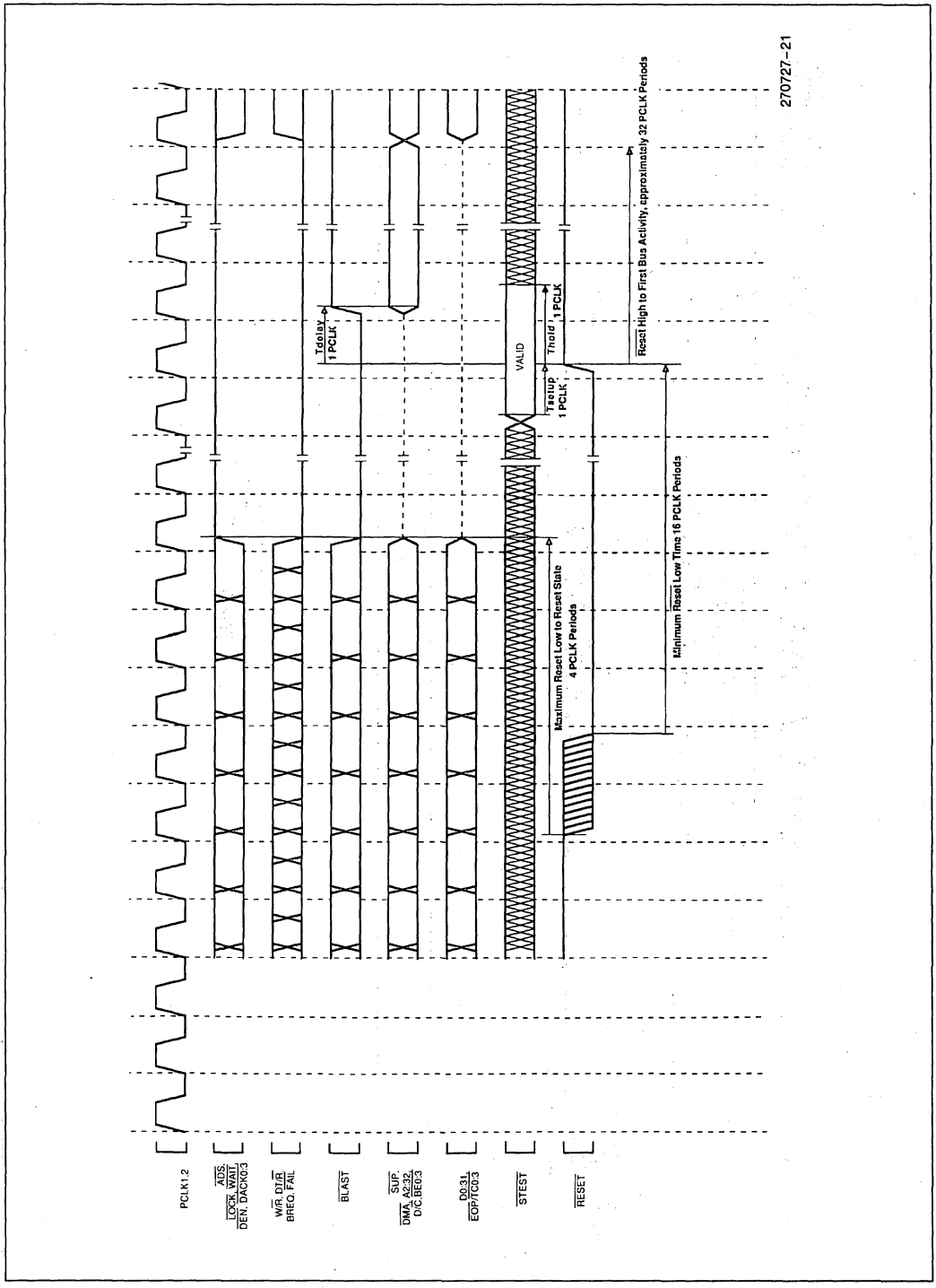


Figure 26. Warm Reset Waveform

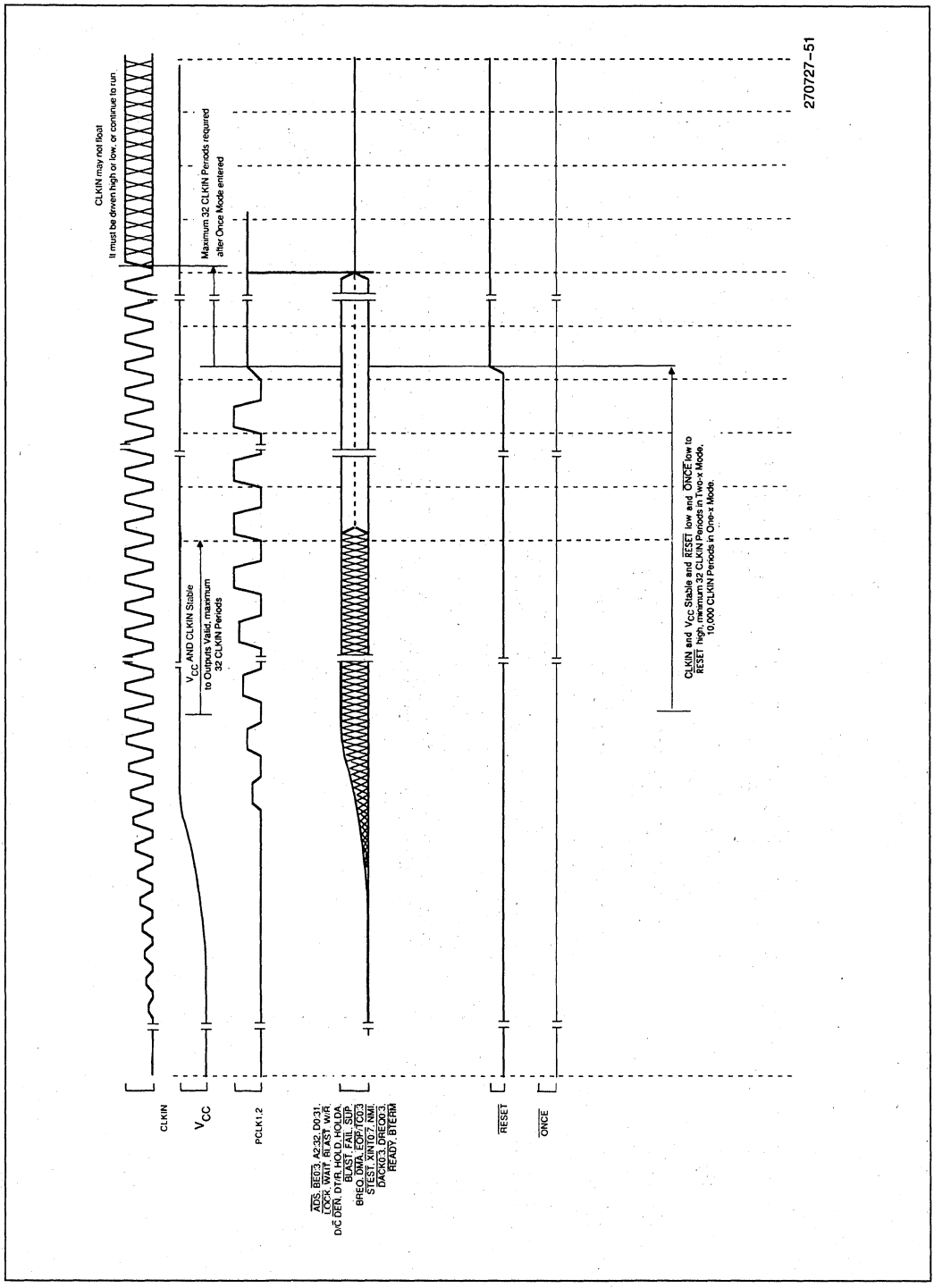
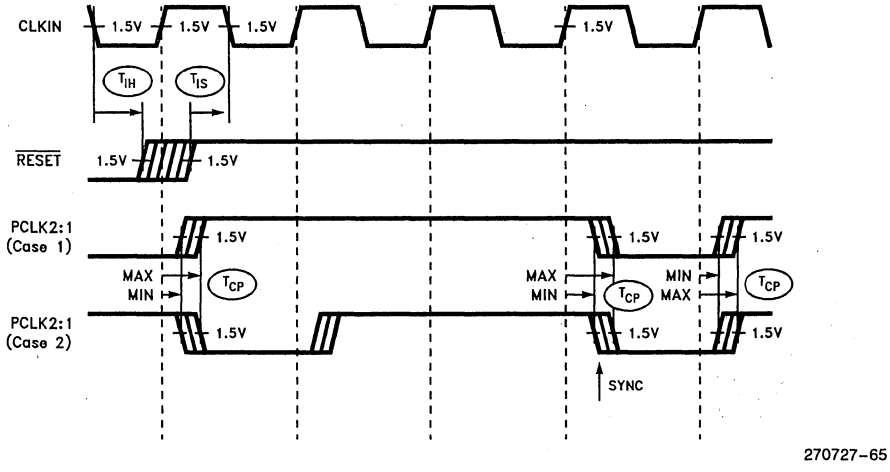


Figure 27. Entering the ONCE™ State



NOTE:
Case 1 and Case 2 show two possible polarities of PCLK2:1.

Figure 28. Clock Synchronization in the 2x Clock Mode

3

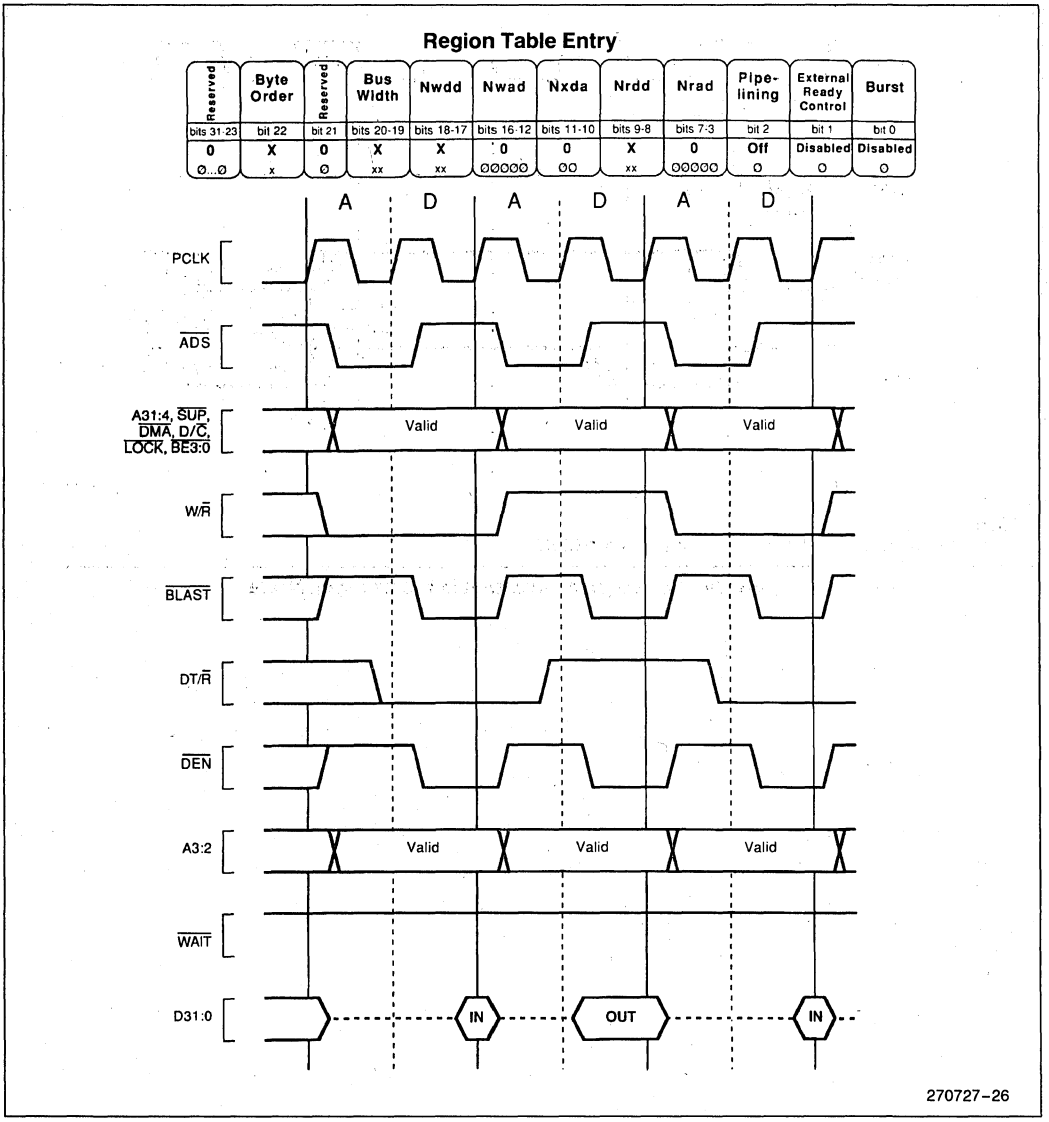
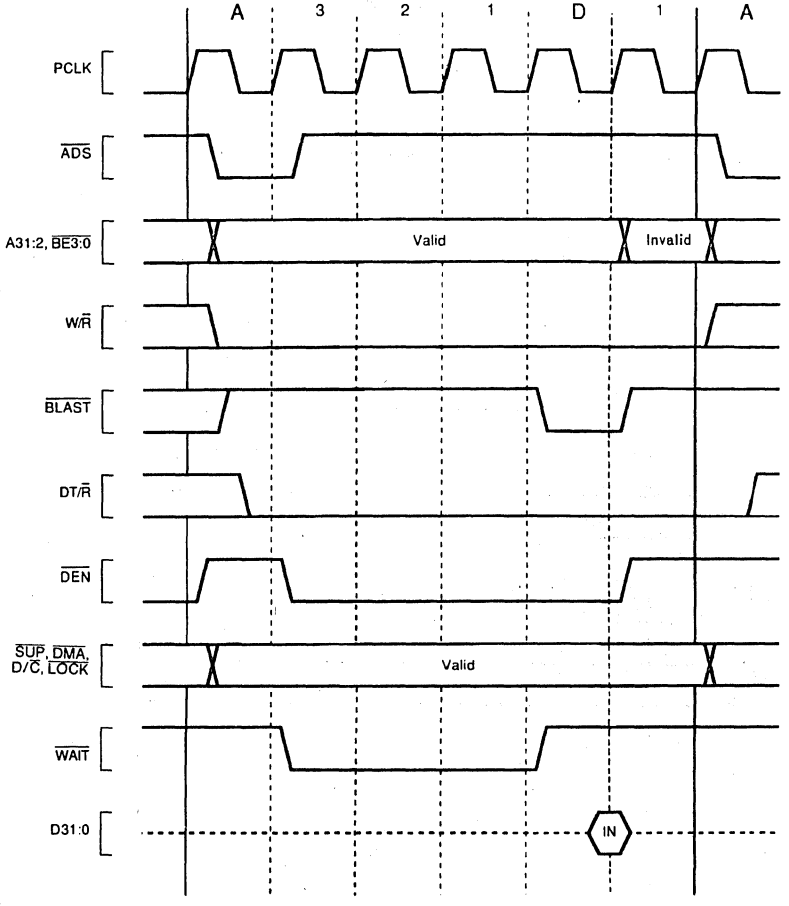


Figure 29. Non-Burst, Non-Pipelined Accesses without wait states

Region Table Entry

Reserved	Byte Order	Reserved	Bus Width	Nwdd	Nwad	Nxda	Nrdd	Nrad	Pipe-lining	External Ready Control	Burst
bits 31-23	bit 22	bit 21	bits 20-19	bits 18 17	bits 16-12	bits 11 10	bits 9-8	bits 7 3	bit 2	bit 1	bit 0
0	X	0	X	X	X	1	X	3	Off	Disabled	Disabled
Ø...Ø	x	Ø	xx	xx	xxxxx	Ø1	xx	ØØØ11	Ø	Ø	Ø



270727-27

Figure 30. Non-Burst, Non-Pipelined Read with wait states

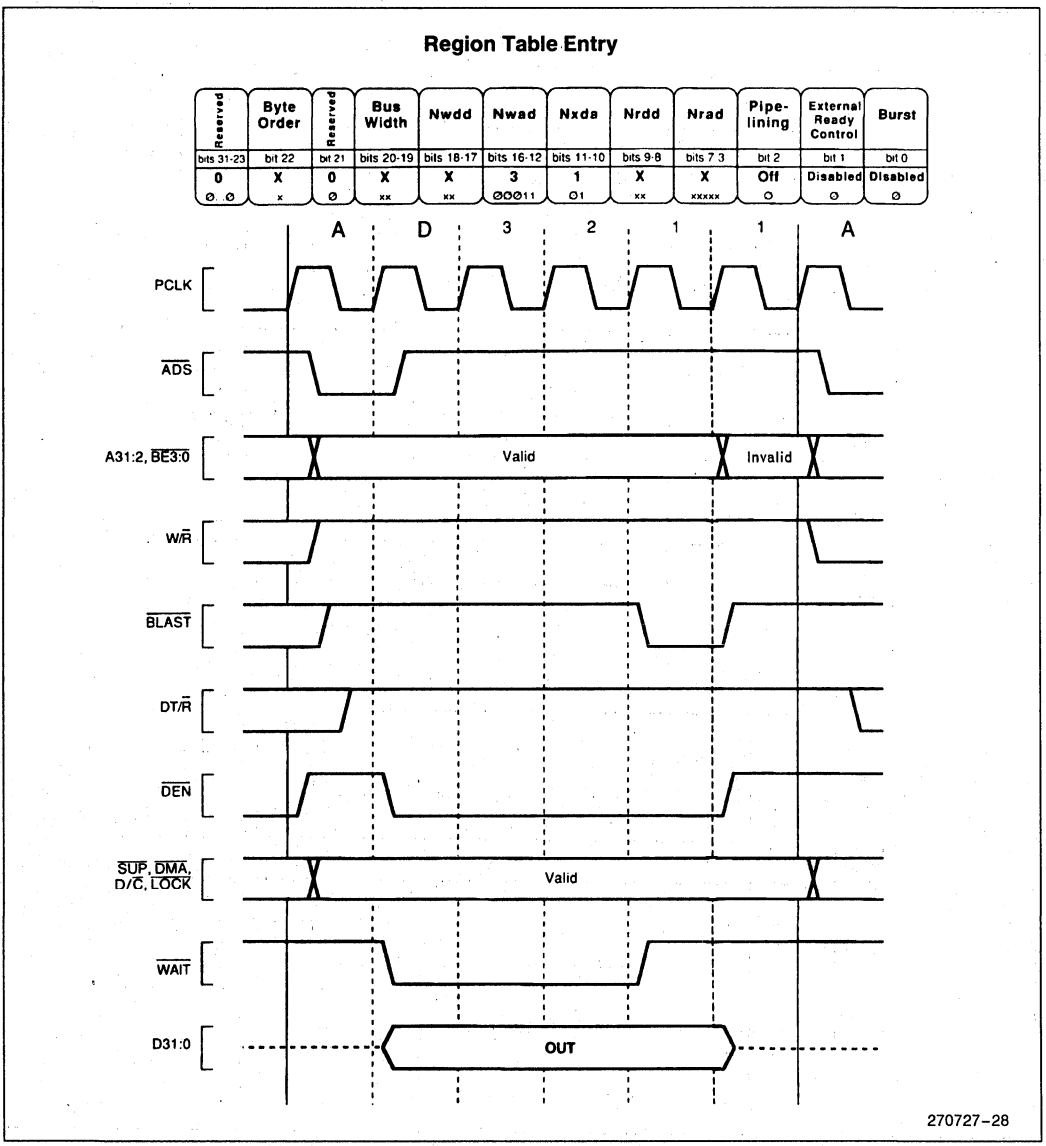


Figure 31. Non-Burst, Non-Pipelined Write with wait states

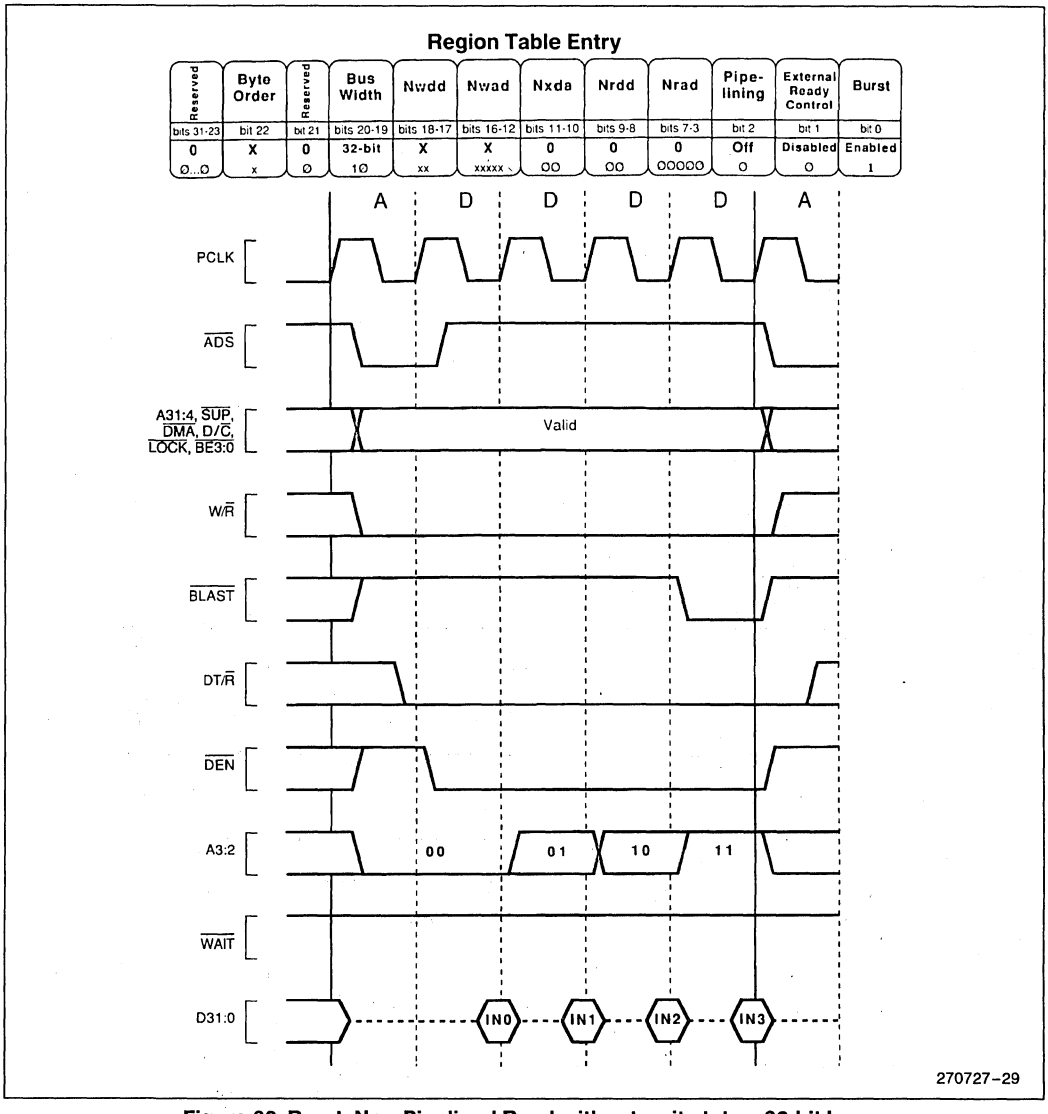


Figure 32. Burst, Non-Pipelined Read without wait states, 32-bit bus

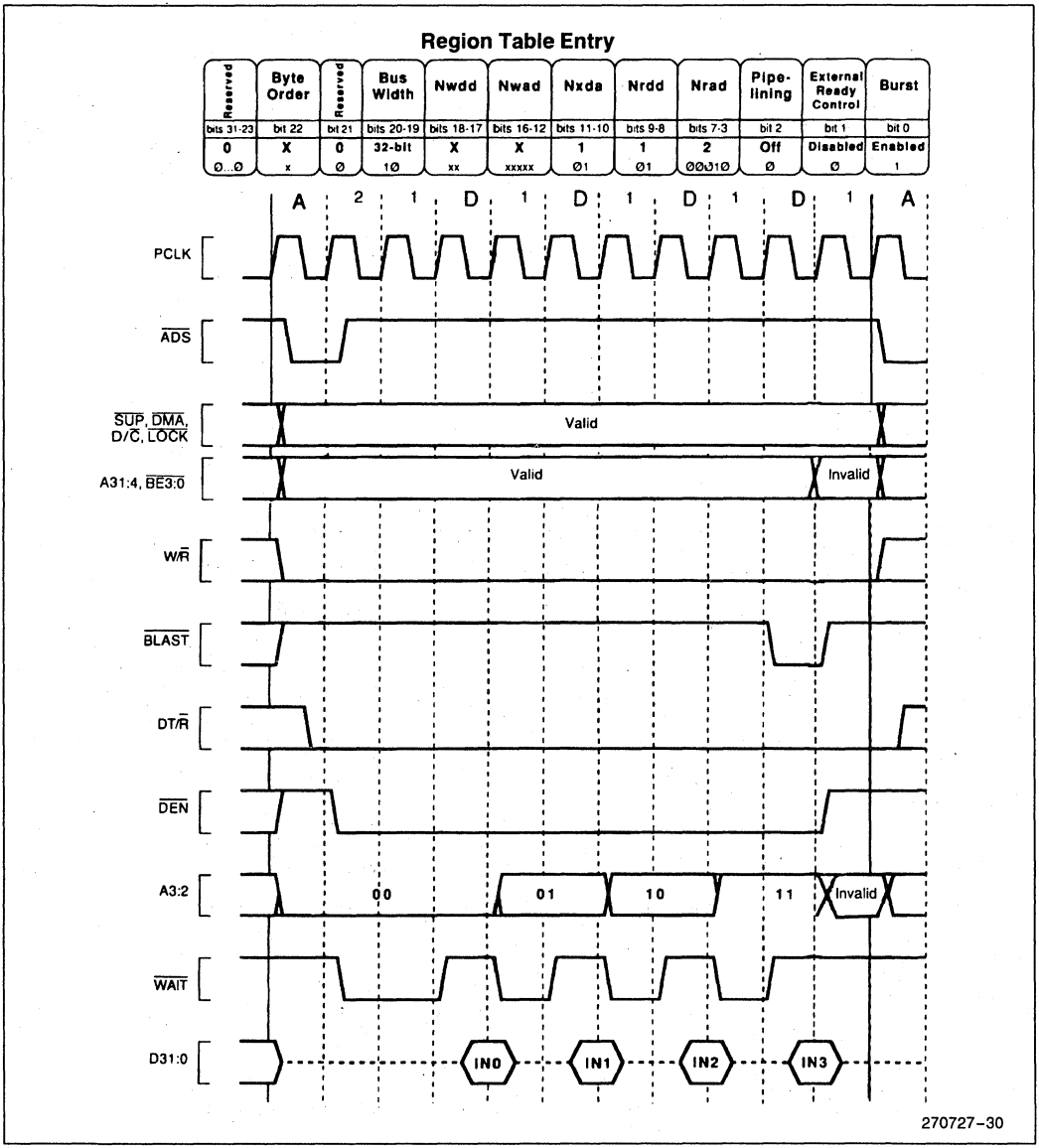
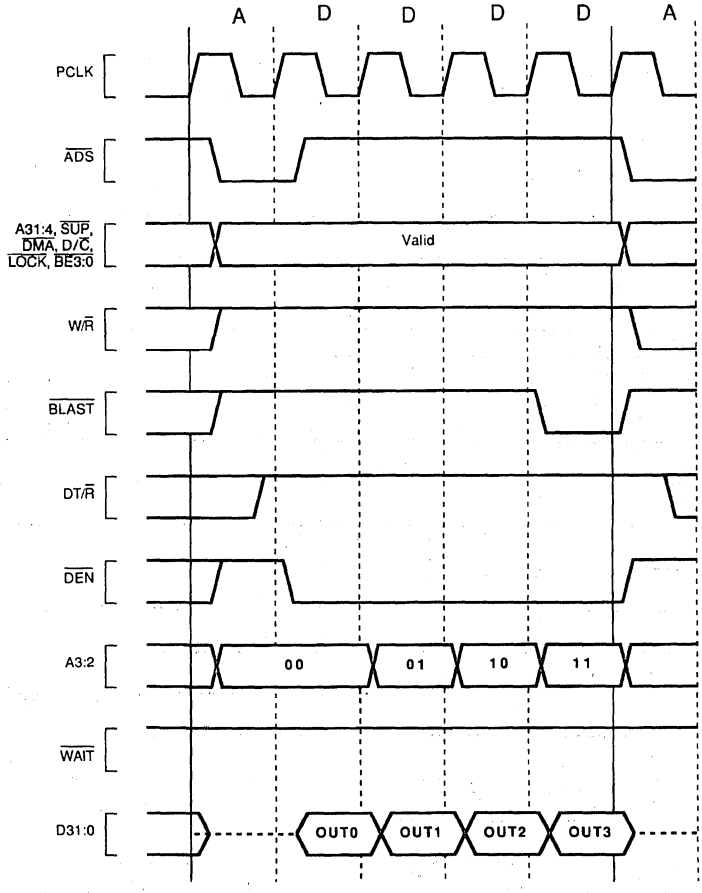


Figure 33. Burst, Non-Pipelined Read with wait states, 32-bit bus

Region Table Entry

Reserved	Byte Order	Reserved	Bus Width	Nwdd	Nwad	Nxda	Nrdd	Nrad	Pipe-lining	External Ready Control	Burst
bits 31-23	bit 22	bit 21	bits 20-19	bits 18-17	bits 16-12	bits 11-10	bits 9-8	bits 7-3	bit 2	bit 1	bit 0
0 Ø...Ø	X x	0 Ø	32-bit 1Ø	0 ØØ	0 ØØØØØ	0 ØØ	X xx	X xxxxx	Off O	Disabled O	Enabled 1



270727-31

Figure 34. Burst, Non-Pipelined Write without wait states, 32-bit bus

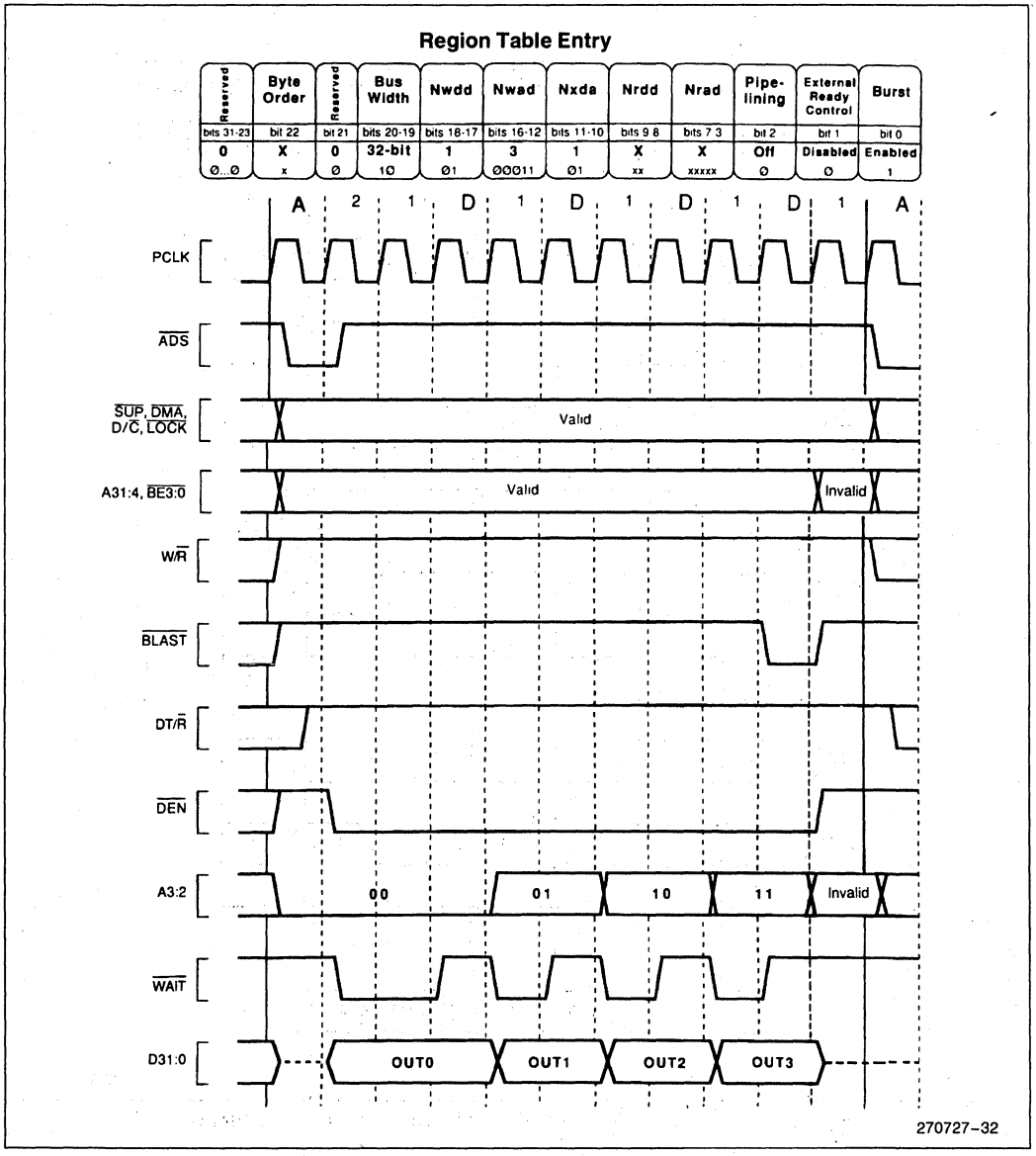


Figure 35. Burst, Non-Pipelined Write with wait states, 32-bit bus

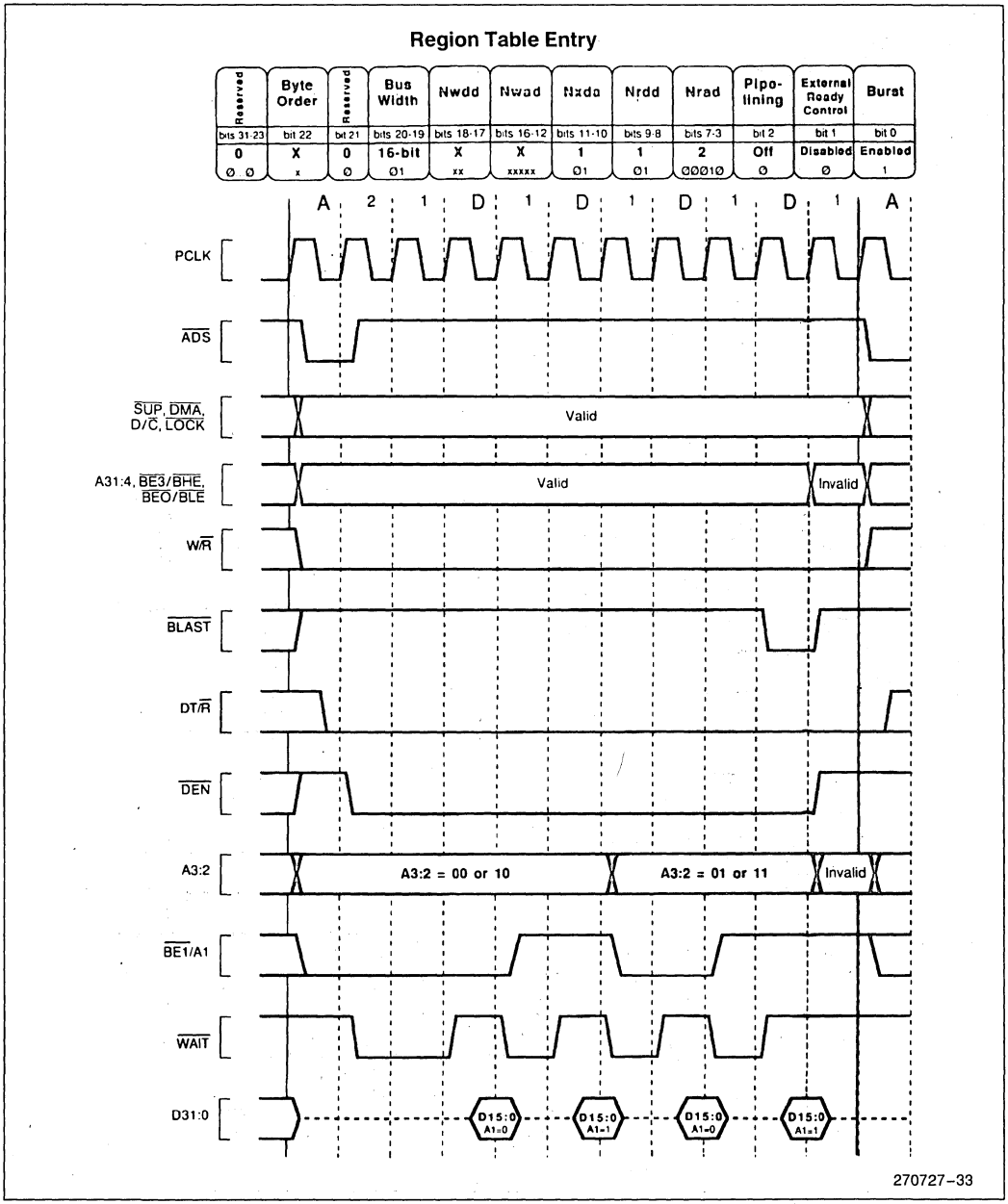


Figure 36. Burst, Non-Pipelined Read with wait states, 16-bit bus

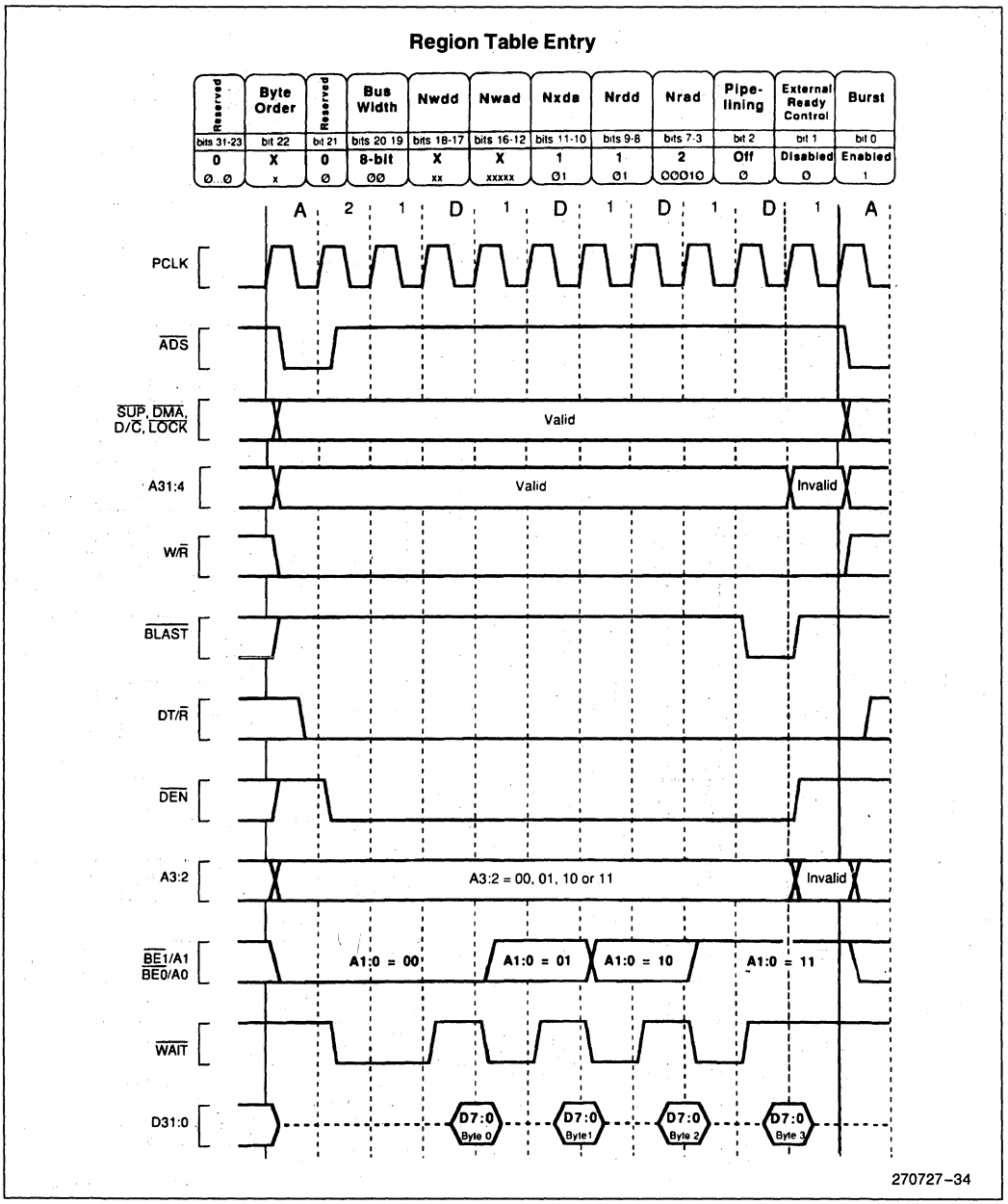
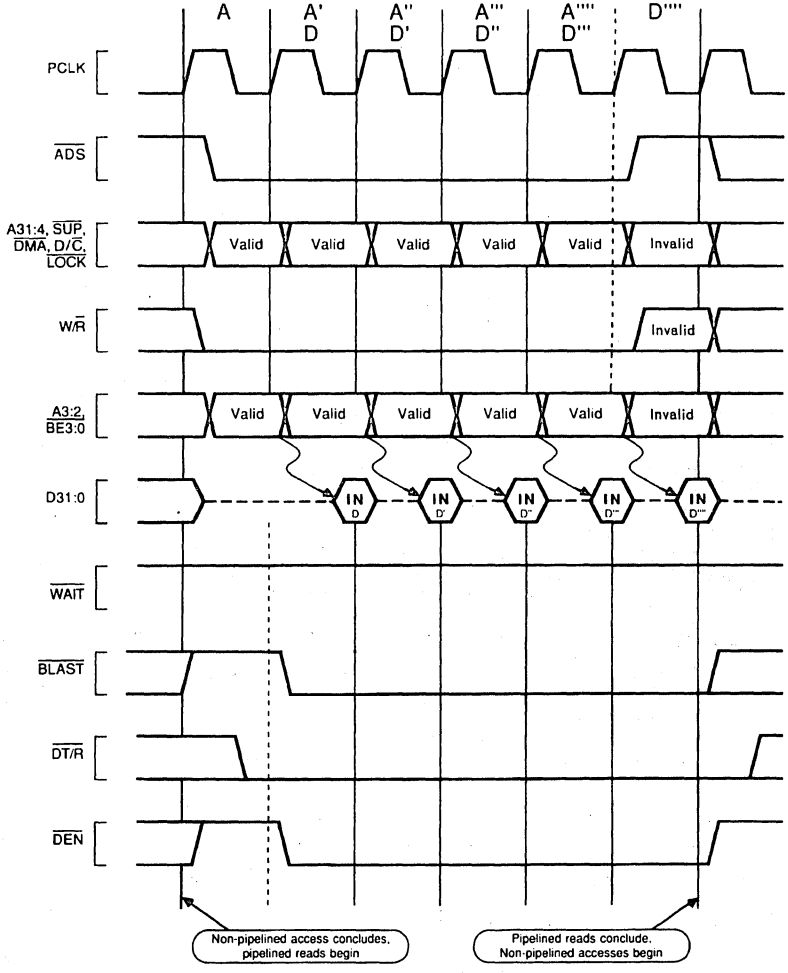


Figure 37. Burst, Non-Pipelined Read with wait states, 8-bit bus

Region Table Entry

Reserved	Byte Order	Reserved	Bus Width	Nwdd	Nwad	Nxda	Nrdd	Nrad	Pipe-lining	External Ready Control	Burst
bits 31-23	bit 22	bit 21	bits 20-19	bits 18-17	bits 16-12	bits 11-10	bits 9-8	bits 7-3	bit 2	bit 1	bit 0
0	X	0	X	X	X	X	X	0	On	X	Disabled
Ø...Ø	x	Ø	xx	xx	xxxxx	xx	xx	OOOOO	1	x	O



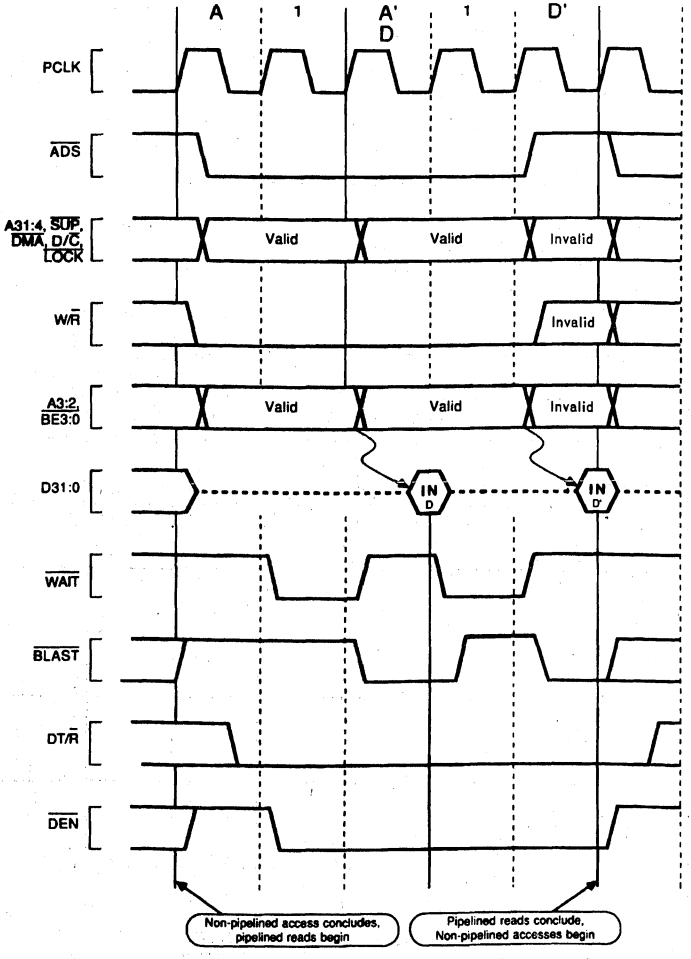
270727-35

Figure 38. Non-Burst, Pipelined Read without wait states, 32-bit bus

3

Region Table Entry

Reserved	Byte Order	Reserved	Bus Width	Nwdd	Nwad	Nxda	Nrdd	Nrad	Pipe-lining	External Ready Control	Burst
bits 31-23	bit 22	bit 21	bits 20-19	bits 18-17	bits 16-12	bits 11-10	bits 9-8	bits 7-3	bit 2	bit 1	bit 0
0	X	0	X	X	X	X	X	1	On	X	Disabled
Ø..Ø	x	Ø	xx	xx	xxxxx	xx	xx	ØØØ01	1	x	Ø



270727-36

Figure 39. Non-Burst, Pipelined Read with wait states, 32-bit bus

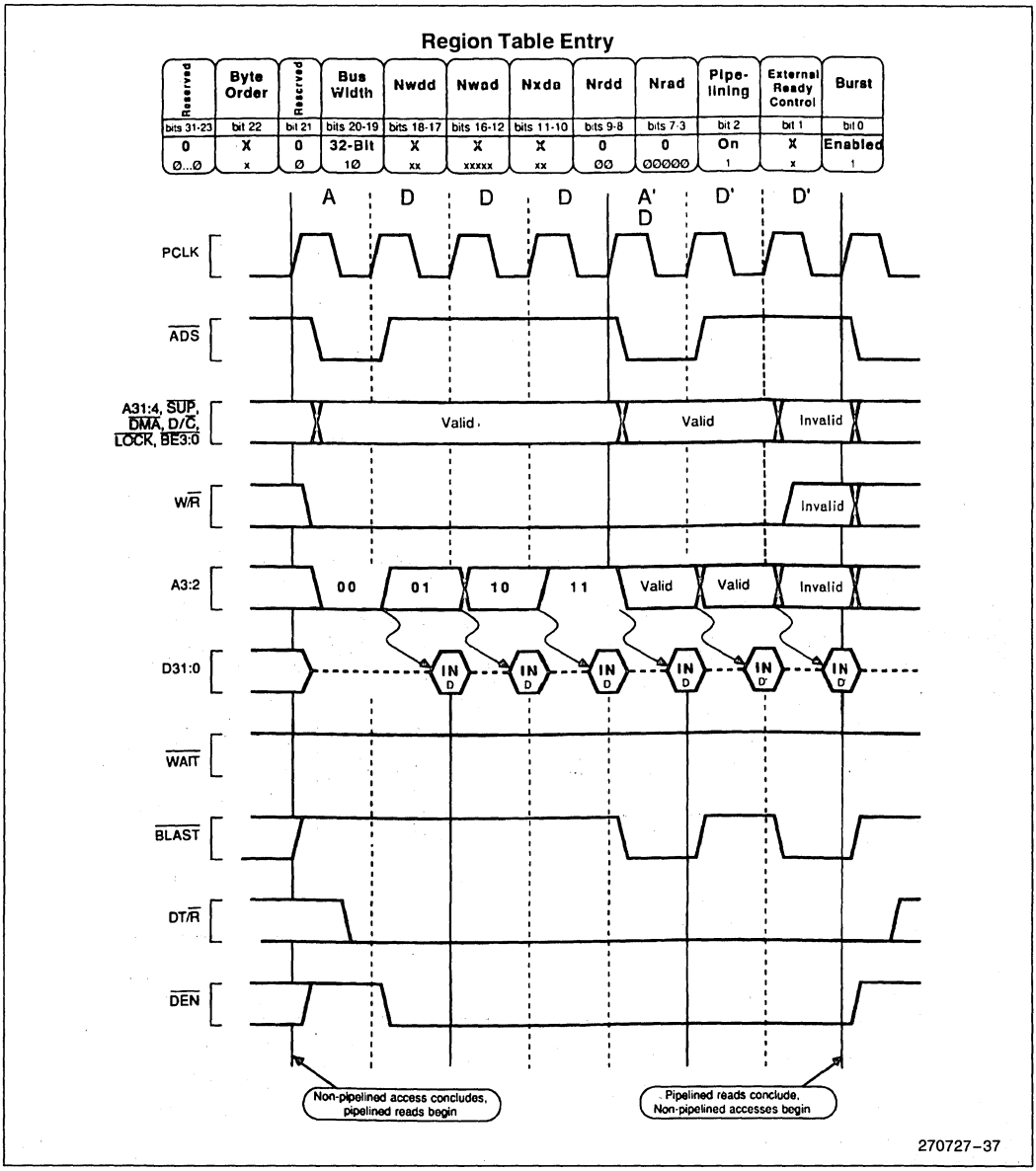


Figure 40. Burst, Pipelined Read without wait states, 32-bit bus

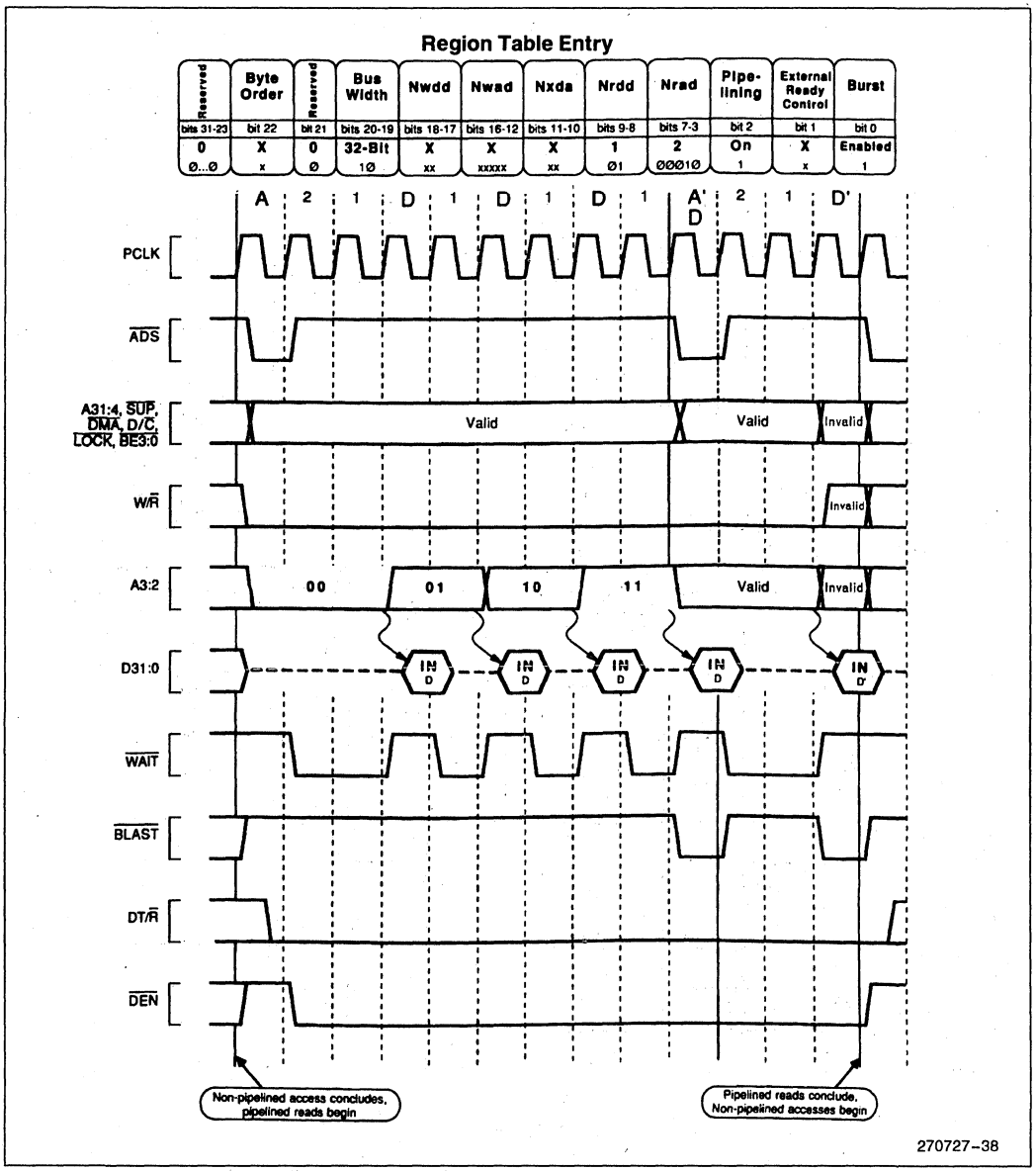


Figure 41. Burst, Pipelined Read with wait states, 32-bit bus

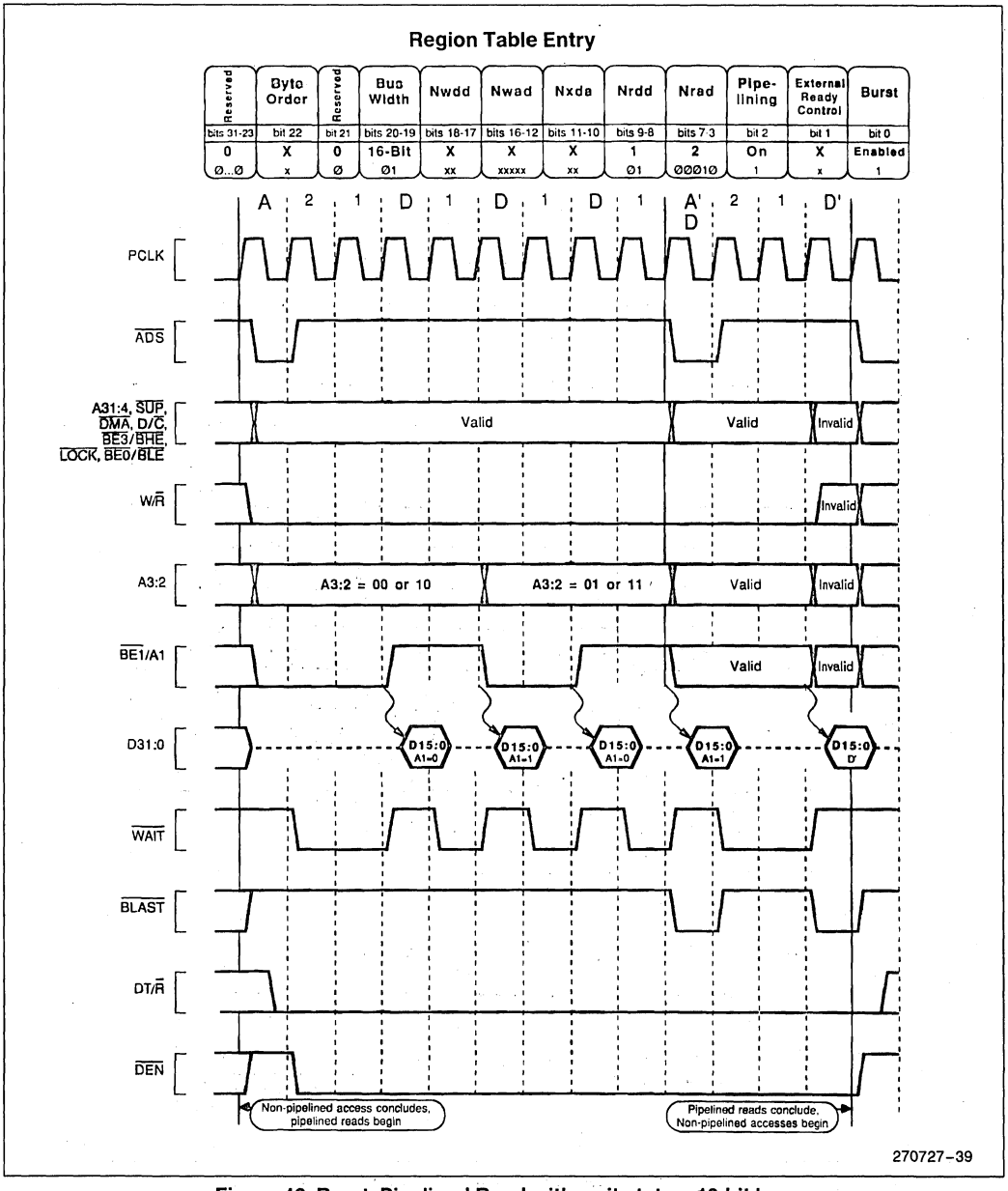


Figure 42. Burst, Pipelined Read with wait states, 16-bit bus

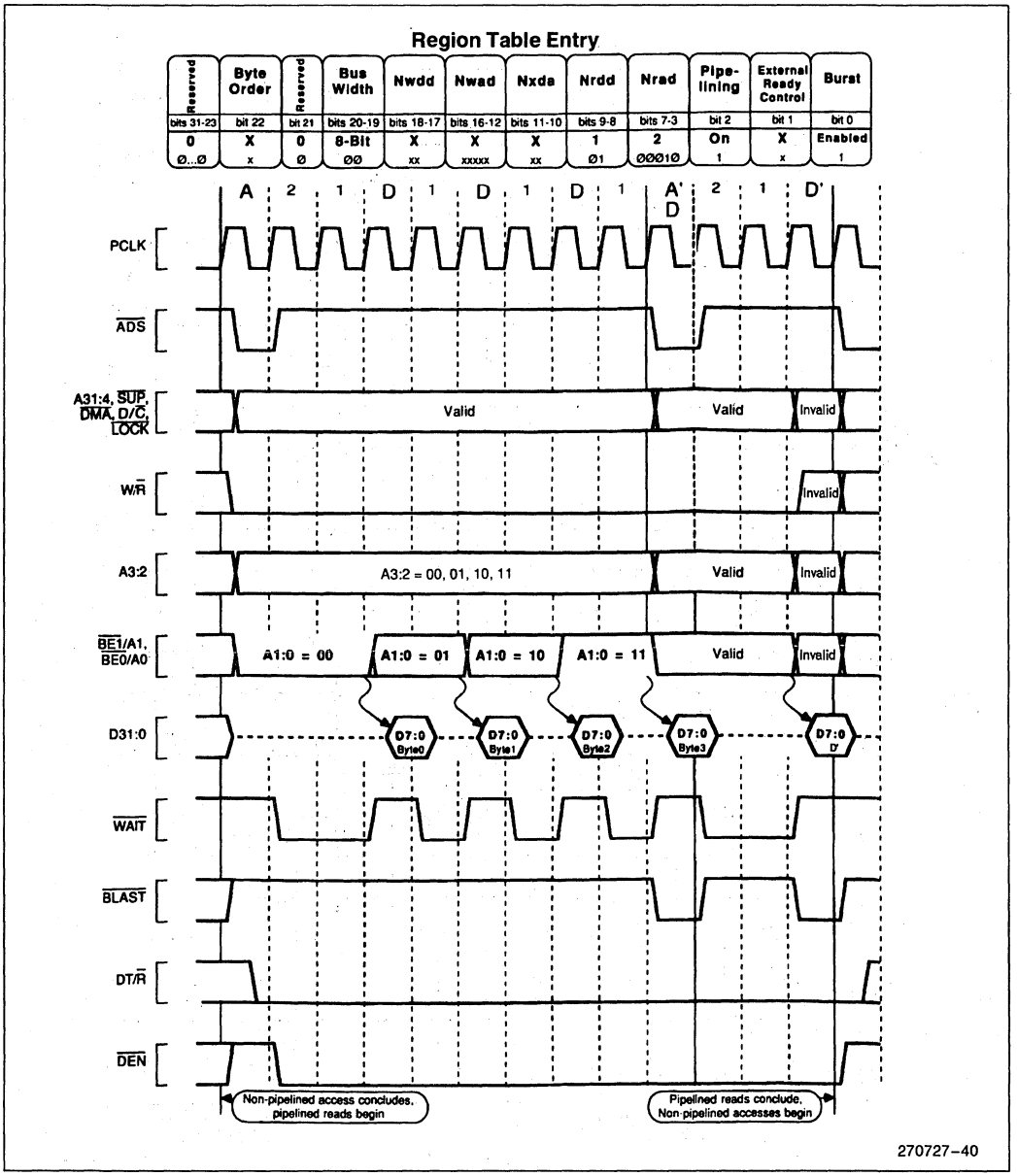


Figure 43. Burst, Pipelined Read with wait states, 8-bit bus

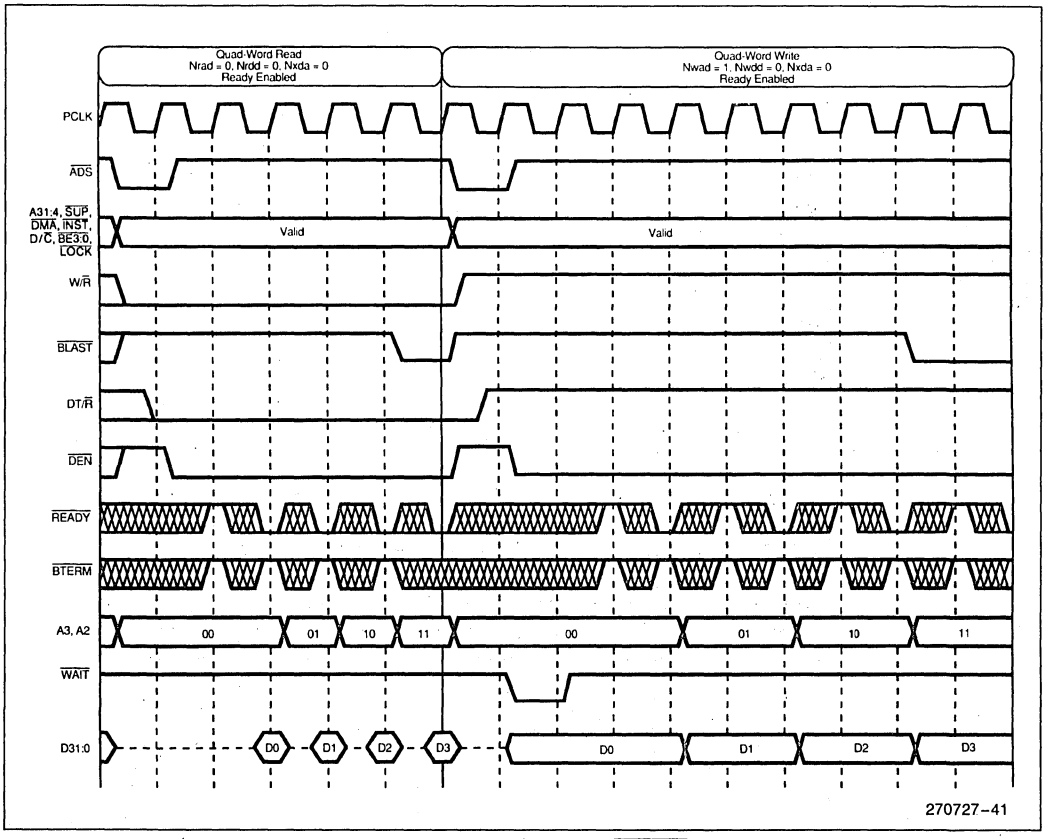
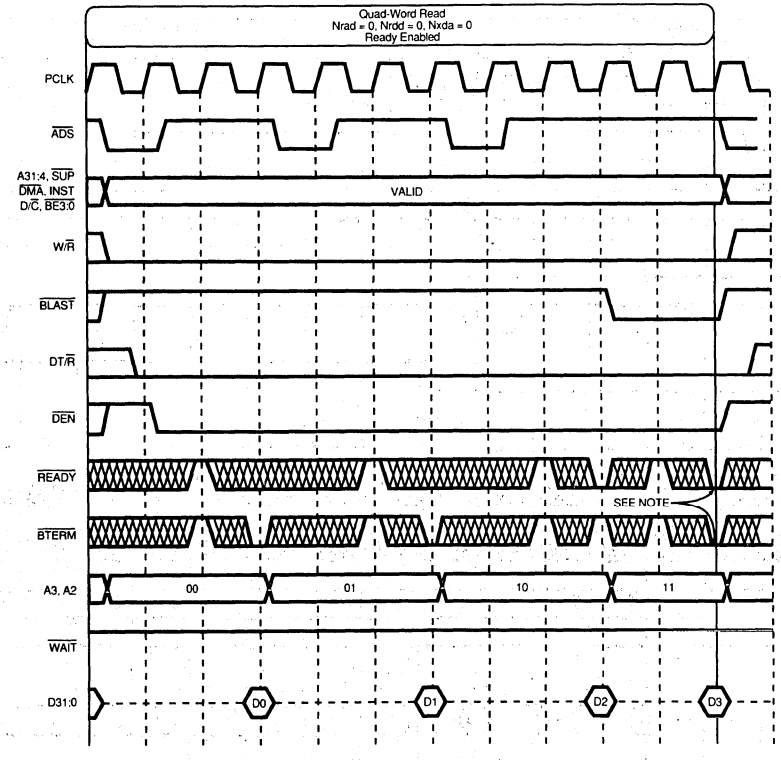


Figure 44. Using External READY



NOTE:
READY adds memory access time to data transfers, whether or not the bus access is a burst access. BTERM interrupts a bus access, whether or not the bus access has more data transfers pending. Either the READY signal or the BTERM signal will terminate a bus access if the signal is asserted during the last (or only) data transfer of the bus access.

Figure 45. Terminating a Burst with BTERM

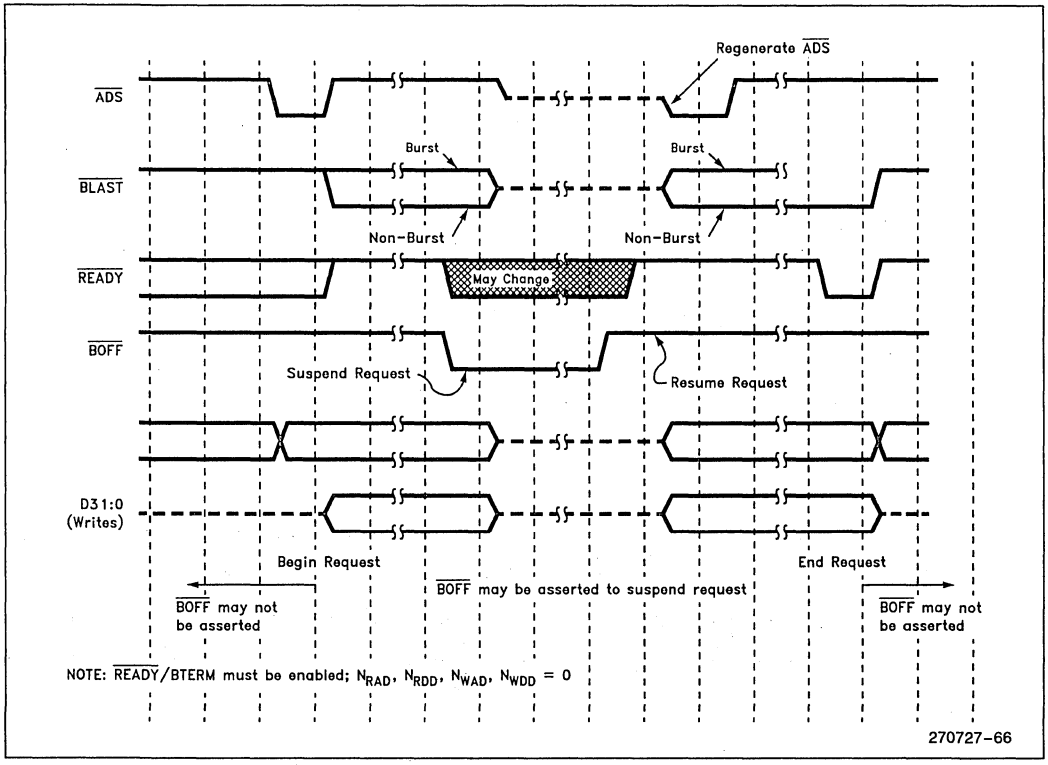


Figure 46. BOFF Functional Timing

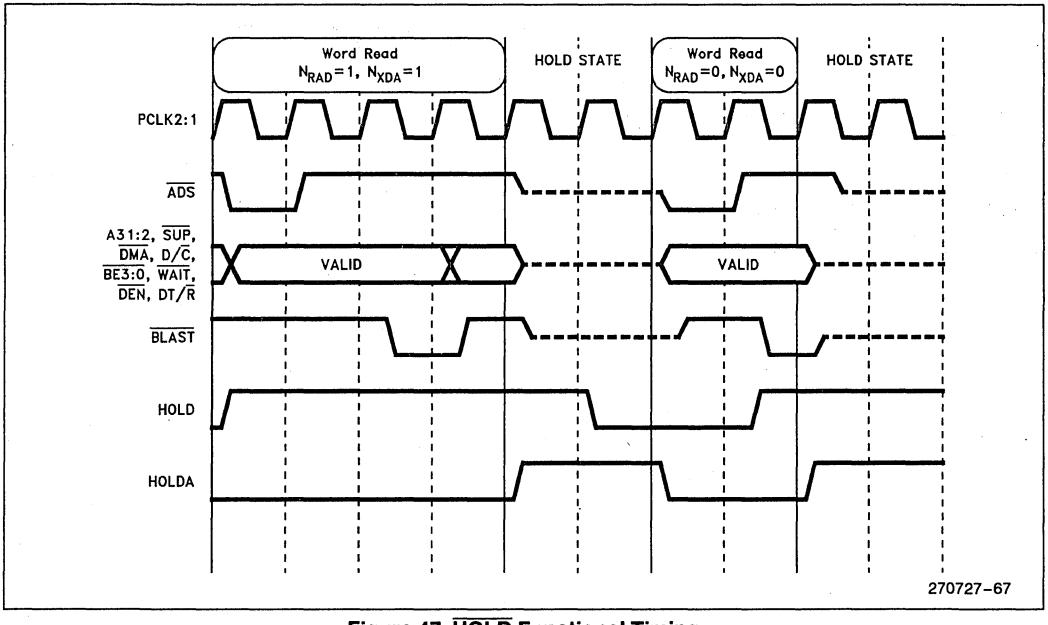
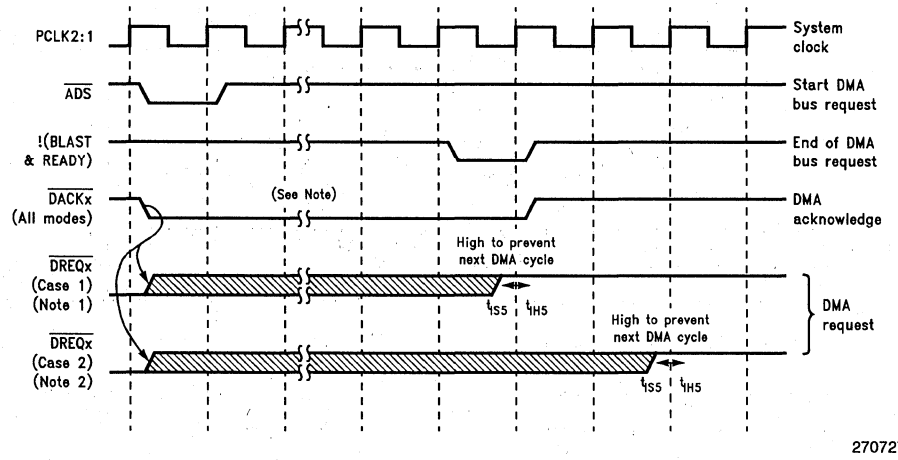
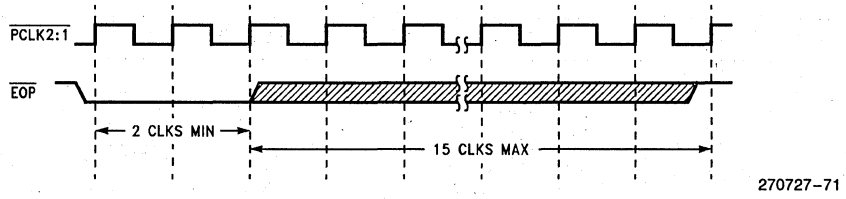


Figure 47. HOLD Functional Timing



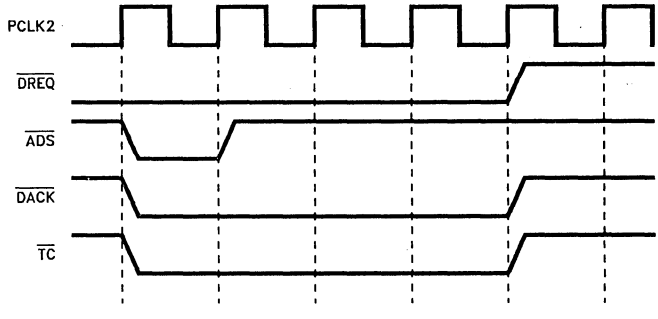
NOTES:
1. Case 1: DREQ must deassert before DACK deasserts. Applications are Fly-by and some packing and unpacking modes, adjacent load-stores or store-loads, loads followed by loads, and stores followed by stores.
2. Case 2: DREQ must be deasserted by the second clock (rising edge) after DACK is driven high. Applications are non fly-by transfers and adjacent load-stores or store-loads.
3. DACKx is asserted for the duration of a DMA bus request. The request may consist of multiple bus accesses (defined by ADS and BLAST. Refer to User's Manual for "access", "request" definition.

Figure 48. DREQ and DACK Functional Timing



NOTE:
EOP has the same AC Timing Requirements as DREQ to prevent unwanted DMA requests.
EOP is NOT edge triggered. EOP must be held for a minimum of 2 clock cycles then EOP must be deasserted within 15 clock cycles.

Figure 49. EOP Functional Timing



NOTE:
Terminal Count becomes active during the last bus request of a buffer transfer. If the last LOAD/STORE bus request is executed as multiple bus accesses, the TC will be active for the entire bus request. Refer to the User's Manual for further information.

Figure 50. Terminal Count Functional Timing

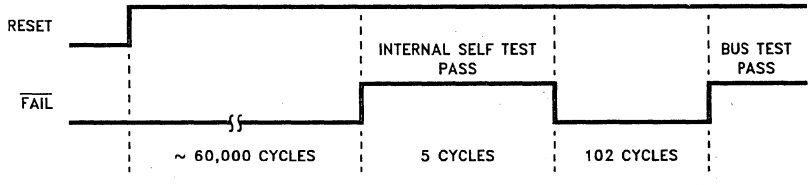


Figure 51. FAIL Functional Timing

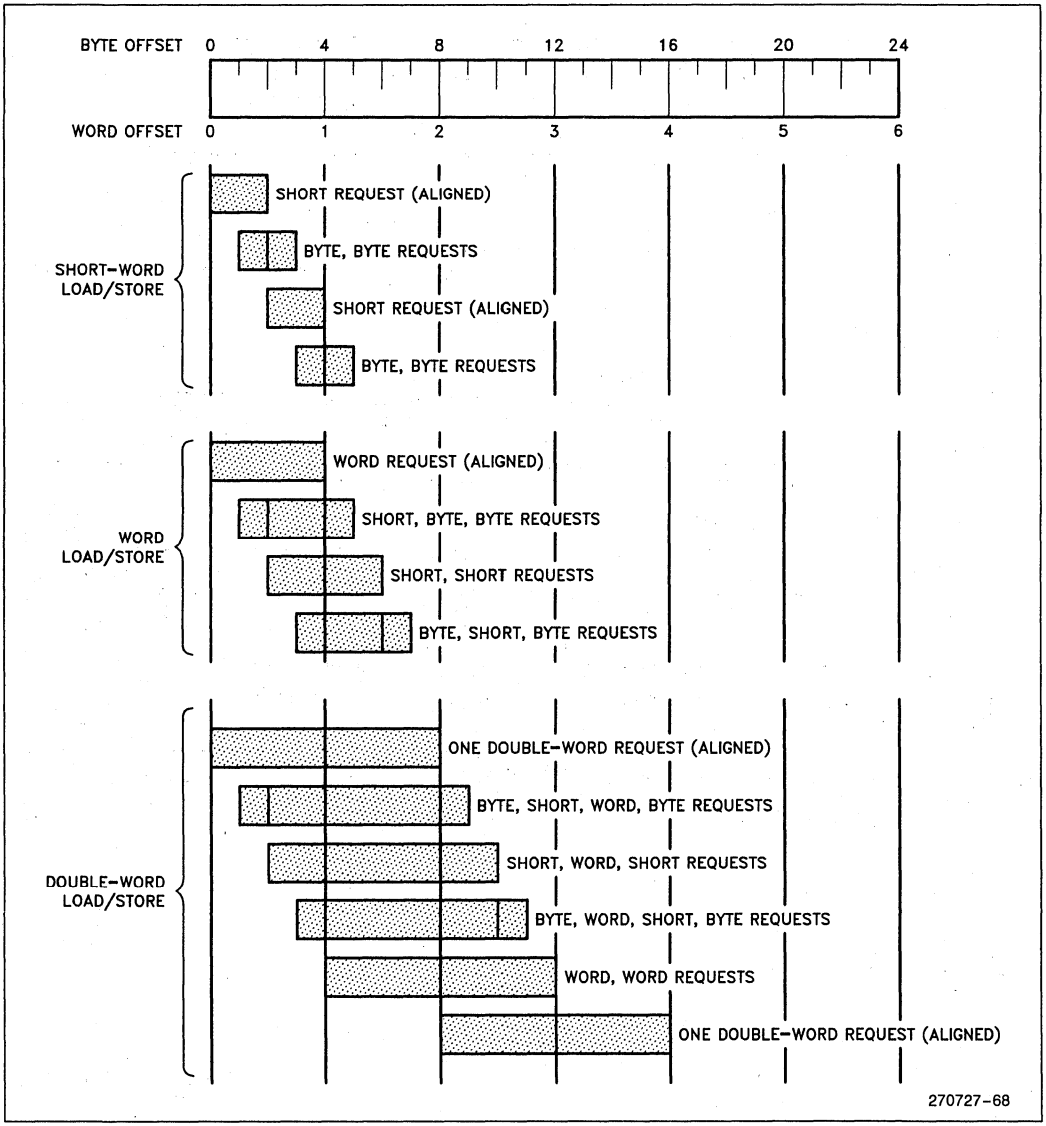


Figure 52. A Summary of Aligned and Unaligned Transfers for Little Endian Regions

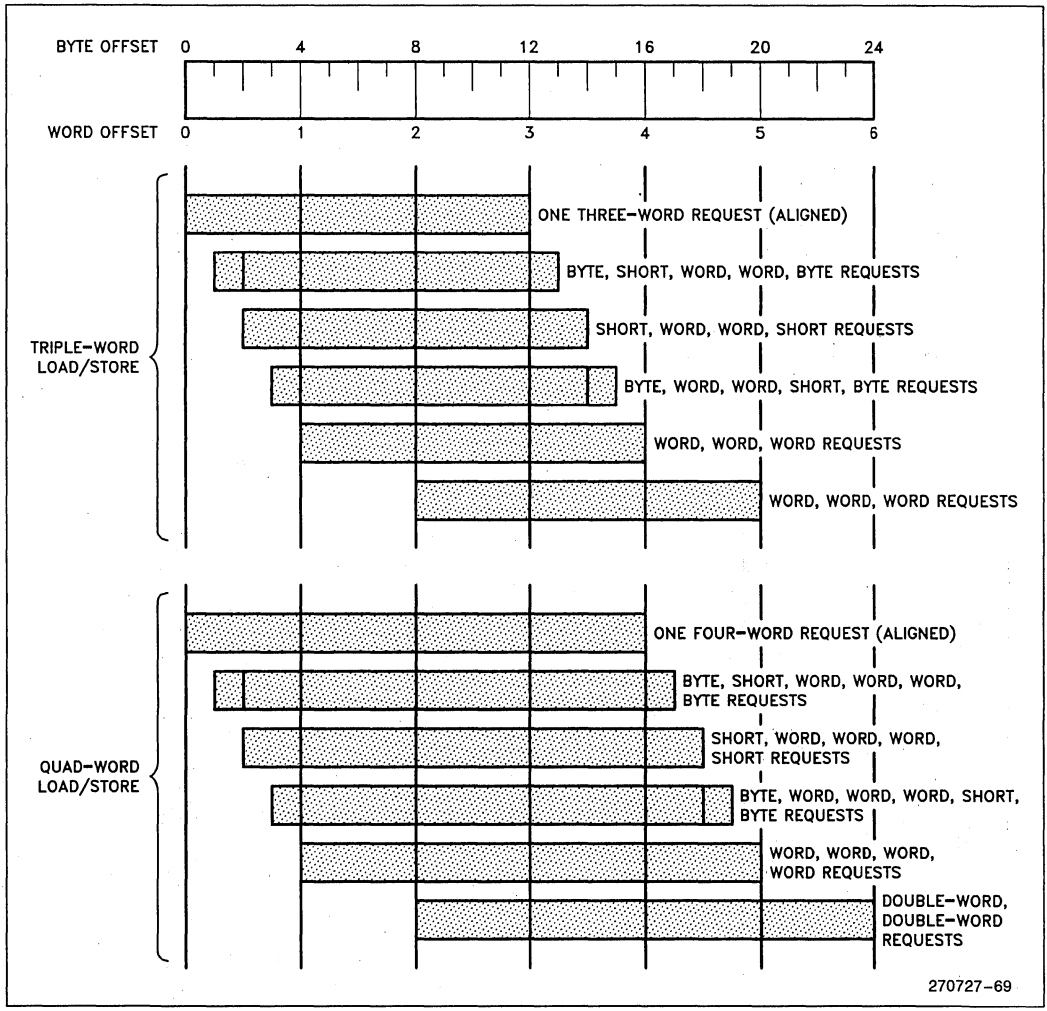


Figure 53. A Summary of Aligned and Unaligned Transfers for Little Endian Regions (Continued)

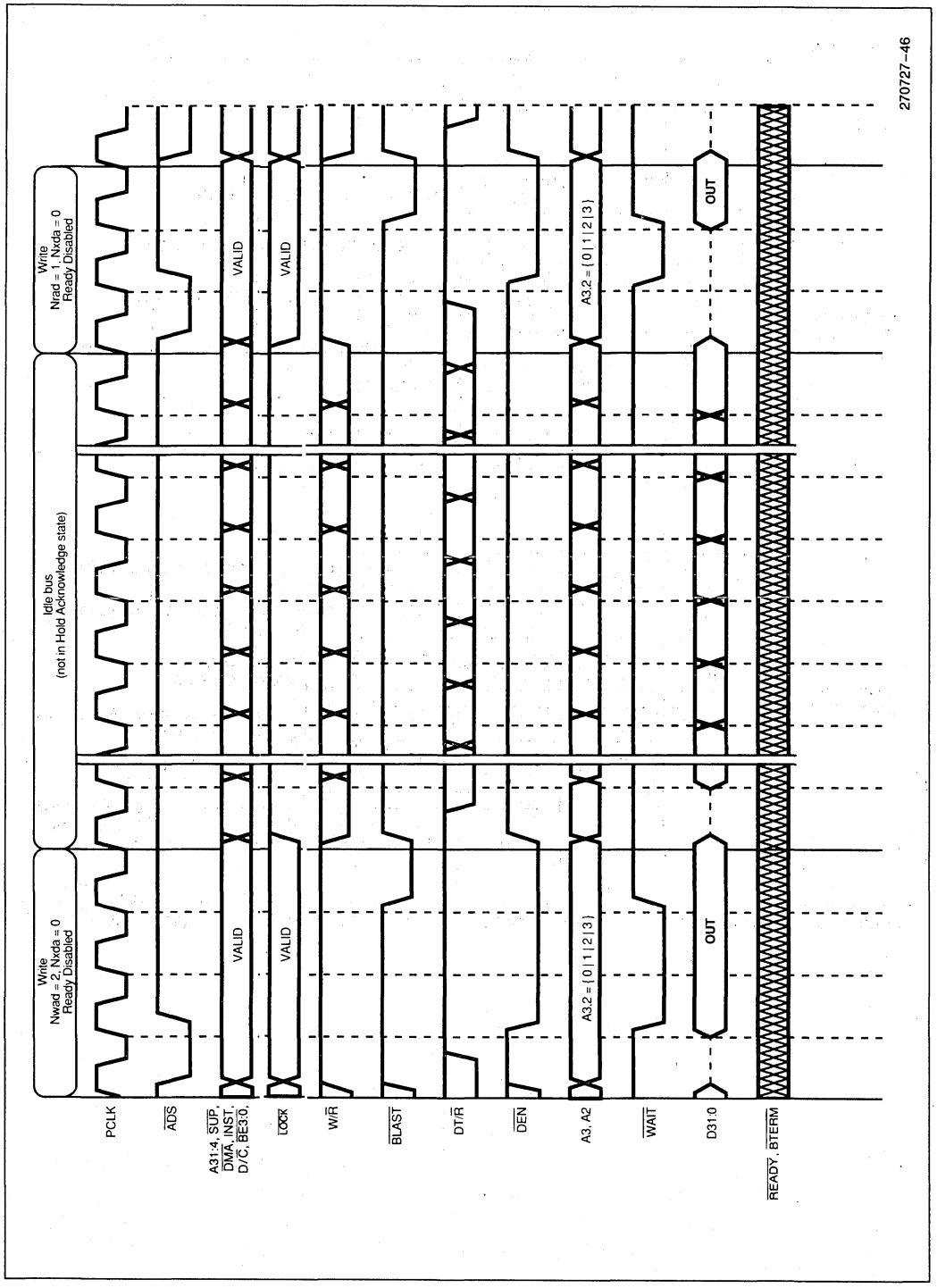


Figure 54. Idle Bus Operation

i960™ MC PROCESSOR PRODUCT OVERVIEW

This chapter provides an overview of the architecture of the i960 MC processor.

The i960 MC processor is the military-grade member of a new family of processors from Intel. This processor family is based on a new 32-bit architecture called the i960 architecture. The i960 architecture has been designed specifically to meet the needs of embedded applications such as avionics, aerospace, weapons systems, robotics and instrumentation, where high reliability is critical. It represents a renewed commitment from Intel to provide reliable, high-performance processors and controllers for the embedded processor marketplace.

The i960 architecture can best be characterized as a high-performance computing engine. It features high-speed instrumentation execution and ease of programming. It is also easily extensible, allowing processors and controllers based on this architecture to be conveniently customized to meet the needs of specific processing and control appplications.

Some of the important attributes of the i960 architecture include:

- full 32-bit registers
- high-speed, pipelined instruction execution
- a convenient program execution environment with 32 general-purpose registers and a versatile set of special-function registers
- a highly optimized procedure call mechanism that features on-chip caching of local variables and parameters
- extensive facilities for handling interrupts and faults
- extensive tracing facilities to support efficient program debugging and monitoring
- register scoreboarding and write buffering to permit efficient operation with lower performance memory subsystems

The i960 MC processor implements the i960 architecture, plus it offers several extensions to the architecture. Some of these extensions, such as on-chip support for floating-point arithmetic, virtual memory management and multitasking, are designed to enhance overall system performance. Several other extensions are designed to enhance system reliability and robustness. These extensions include facilities for hardware enforced protection of software modules and for creating fault tolerant systems through the use of redundant processors.

The following sections describe those features of the i960 architecture that are provided to streamline code execution and simplify programming. The extensions to this architecture provided in the i960 MC processor are described at the end of the chapter.

HIGH PERFORMANCE PROGRAM EXECUTION

Much of the design of the i960 architecture has been aimed at maximizing the processor's computational and data processing speed through increased parallelism. The following paragraphs describe several of the mechanisms and techniques used to accomplish this goal, including:

- an efficient load and store memory-access model
- caching of code and procedural data
- overlapped execution of instructions
- many one or two clock-cycle instructions

Load and Store Model

One of the more important features of the i960 architecture is that most of its operations are performed on operands in registers, rather than in memory. For example, all the arithmetic, logical, comparison, branching and bit operations are performed with registers and literals.

This feature provides two benefits. First, it increases program execution speed by minimizing the number of memory accesses required to execute a program. Second, it reduces memory latency encountered when using slower, lower-cost memory parts.

To support this concept, the architecture provides a generous supply of general-purpose registers. For each procedure, 32 registers are available (28 of which are available for general use). These registers are divided into two types: global and local. Both these types of registers can be used for general storage of operands. The only difference is that global registers retain their contents across procedure boundaries, whereas the processor allocates a new set of local registers each time a new procedure is called.

Order Number: 272031-001

The architecture also provides a set of fast, versatile load and store instructions. These instructions allow burst transfers of 1, 2, 4, 8, 12 or 16 bytes of information between memory and the registers.

On-Chip Caching of Code and Data

To further reduce memory accesses, the architecture offers two mechanisms for caching code and data on chip: an instruction cache and multiple sets of local registers. The instruction cache allows prefetching of blocks of instruction from memory, which helps insure that the instruction execution pipeline is supplied with a steady stream of instructions. It also reduces the number of memory accesses required when performing iterative operations such as loops. (The size of the instruction cache can vary. With the i960 MC processor, it is 512 bytes.)

To optimize the architecture's procedure call mechanism, the processor provides multiple sets of local registers. This allows the processor to perform most procedure calls without having to write the local registers out to the stack in memory.

(The number of local-register sets provided depends on the processor implementation. The i960 MC processor provides four sets of local registers.)

Overlapped Instruction Execution

Another technique that the i960 architecture employs to enhance program execution speed is overlapping the execution of some instructions. This is accomplished through two mechanisms: register scoreboarding and branch prediction.

Register scoreboarding permits instruction execution to continue while data is being fetched from memory. When a load instruction is executed, the processor sets one or more scoreboard bits to indicate the target registers to be loaded. After the target registers are loaded, the scoreboard bits are cleared. While the target registers are being loaded, the processor is allowed to execute other instructions that do not use these registers. The processor uses the scoreboard bits to insure that target registers are not used until the loads are complete. (The checking of scoreboard bits is transparent to software.) The net result of using this technique is that code can often be optimized in such a way as to allow some instructions to be executed parallel.

Single-Clock Instructions

It is the intent of the i960 architecture that a processor be able to execute commonly used instructions such as move, add, subtract, logical operations, compare and branch in a minimum number of clock cycles (preferably one clock cycle). The architecture supports this concept in several ways. For example, the load and store model described earlier in this chapter (with its concentration on register-to-register operations) allows simple operations to be performed without the overhead of memory-to-memory operations.

Also, all the instructions in the i960 architecture are 32 bits or 64 bits long and aligned on 32-bit boundaries. This feature allows instructions to be decoded in one clock cycle. It also eliminates the need for an instruction-alignment stage in the pipeline.

The design of the i960 MC processor takes full advantage of these features of the architecture, resulting in more than 50 instructions that can be executed in a single clock-cycle.

Efficient Interrupt Model

The i960 architecture provides an efficient mechanism for servicing interrupts from external sources. To handle interrupts, the processor maintains an interrupt table of 248 interrupt vectors (240 of which are available for general use). When an interrupt is signaled, the processor uses a pointer from the interrupt table to perform an implicit call to an interrupt handler procedure. In performing this call, the processor automatically saves the state of the processor prior to receiving the interrupt; performs the interrupt routine; and then restores the state of the processor. A separate interrupt stack is also provided to segregate interrupt handling from application programs.

The interrupt handling facilites also feature a method of prioritizing interrupts. Using this technique, the processor is able to store interrupts that are lower in priority than the task the processor is currently working on in a pending interrupt section of the interrupt table. At certain defined times, the processor checks the pending interrupts and services them.

SIMPLIFIED PROGRAMMING ENVIRONMENT

Partly as a side benefit of its streamlined execution environment and partly by design, processors based on the i960 architecture are particularly easy to program. For example, the large number of general-purpose registers allows relatively complex algorithms to be executed with a minimum number of memory accesses. The following paragraphs describe some of the other features that simplify programming.

Highly Efficient Procedure Call Mechanism

The procedure call mechanism makes procedure calls and parameter passing between procedures simple and compact. Each time a call instruction is issued, the processor automatically saves the current set of local registers and allocates a new set of local registers for the called procedure. Likewise, on a return from a procedure, the current set of local registers is deallocated and the local registers for the procedure being returned to are restored. On a procedure call, the program thus never has to explicitly save and restore those local variables and parameters that are stored in local registers.

Versatile Instruction Set and Addressing

The selection of instructions and addressing modes also simplifies programming. The architecture offers a full set of load, store, move, arithmetic, comparison and branch instructions, with operations on both integer and ordinal data types. It also provides a complete set of Boolean and bit-field instructions, to simplify operations on bits and bit strings.

The addressing modes are efficient and straightforward, while at the same time providing the necessary indexing and scaling modes required to address complex arrays and record structures.

The large 4-gigabyte address space provides ample room to store programs and data. The availability of 32 addressing lines allows some address lines to be memory-mapped to control hardware functions.

Extensive Fault Handling Capability

To aid in program development, the i960 architecture defines a wide selection of faults that the processor detects, including arithmetic faults, invalid operands, invalid operations and machine faults. When a fault is detected, the processor makes an implicit call to a fault handler routine, using a mechanism similar to that described above for interrupts. The information collected for each fault allows program developers to quickly correct faulting code. It also allows automatic recovery from some faults.

Debugging and Monitoring

To support debugging systems, the i960 architecture provides a mechanism for monitoring processor activity by means of trace events. The processor can be configured to detect as many as seven different trace events, including branches, calls, supervisor calls, returns, prereturns, breakpoints and the execution of any instruction. When the processor detects a trace event, it signals a trace fault and calls a fault handler. Intel provides several tools that use this feature, including an in-circuit emulator (ICE™) device.

SUPPORT FOR ARCHITECTURAL EXTENSIONS

The i960 architecture described earlier in this chapter provides a high-performance computing engine for use as the computational and data-processing core of embedded processor or controllers. The architecture also provides several features that enable processors based on this architecture to be easily customized to meet the needs of specific embedded applications, such as signal processing, array processing or graphics processing.

The most important of these features is a set of 32 special-function registers. These registers provide a convenient interface to circuitry in the processor or to pins that can be connected to external hardware. They can be used to control timers, to perform operations on special data types or to perform I/O functions.

The special-function registers are similar to the global registers. They can be addressed by all the register-access instructions.

EXTENSIONS INCLUDED IN THE 80960MC PROCESSOR

The extensions to the i960 architecture included in the i960 MC processor are built on top of the processor's core computing engine. These extensions are aimed at improving the efficiency and reliability of embedded systems.

On-Chip Floating Point

The i960 MC processor provides a complete implementation of the IEEE standard for binary floating-point arithmetic (IEEE 754-185). This implementation includes a full set of floating-point operations, including add, subtract, multiply, divide, trigonometric functions and logarithmic functions. These operations are performed on single precision (32-bit), double precision (64-bit) and extended precision (80-bit) real numbers.

One of the benefits of this implementation is that the floating-point handling facilities are completely integrated into the normal instruction execution environment. Single- and double-precision floating-point values are stored in the same registers as non-floating point values. Also, four 80-bit floating-point registers are provided to hold extended-precision values.

String and Decimal Operations

The i960 MC processor provides several instructions for moving, filling and comparing byte strings in memory. These instructions speed up string operations and reduce the amount of code required to handle strings.

The decimal instructions perform move, add with carry and subtract with carry operations on binary-coded decimal (BCD) strings.

Virtual-Memory Support

Another of the i960 MC processor's important features is support for virtual-memory management. When using the processor in virtual-memory mode, the processor provides each process (or task) with an address space of up to 2^{32} bytes. This address space is paged into physical memory in 4 Kbyte pages. On-chip memory-management facilities handle virtual-to-physical address translation. A translation look-aside buffer (TLB) speeds address translation by storing virtual-to-physical address translations for frequently accessed parts of memory, such as the location of the page tables and the location of often used system data structures.

Protection

The i960 MC processor offers two mechanisms for protecting critical data structures or software modules. The first is the ability to use page rights bits to restrict access to individual pages. Page rights allow various levels of access to be assigned to a page, ranging from no access to read only to read-write.

The second protection mechanism is a user/supervisor protection model. This two-level protection model provides hardware enforced protection of kernel procedures and data structures. When using this protection mechanism, priviledged procedures and data are placed in protected pages of memory. These pages can then be accessed only through a procedure table, which provides a tightly controlled interface to kernel functions.

Multitasking

The i960 MC processor offers a variety of process management facilities to support concurrent execution of multiple tasks. These facilities can be divided into two groups: process scheduling and interprocess communications.

The process scheduling facilities consist of a set of general-purpose data structures and instructions, which are designed to support several different multitasking schemes. For example, the processor provides a set of instructions that allow the kernel to explicitly dispatch a task (bind it to the processor) and to suspend a task (save the current state of a task so that another task can be bound to the processor). These instructions can be used within kernel procedures to schedule, dispatch and preempt multiple tasks.

The processor also provides a unique feature called *self dispatching*. Here, the kernel schedules tasks by queuing them to a dispatch port. Thereafter, the processor handles the dispatching, preempting and rescheduling of the tasks automatically, independent of the kernel. When using this mechanism, tasks can be scheduled by priority, with up to 32 priority levels to choose from.

The processor's interprocess communication facilities include support for semaphores and communication ports. These facilities allow synchronization of interdependent tasks and asynchronous communication between tasks.

Multiprocessing

The i960 MC processor provides several mechanisms designed to simplify the design of multiple-processor systems, allowing several processors to run in parallel, using shared memory resources. One of these mechanisms is the self-dispatching capability described above. Here, two or more processors can schedule and dispatch processes from a single dispatch port, with each processor equally sharing the processing load.

The processor also provides an inter-agent communication (IAC) mechanism that allows processors to exchange messages among themselves on the bus. This mechanism operates similarly to the interrupt mechanism, except that IAC messages are passed through dedicated sections of memory. The IAC mechanism can be used to preempt processes running on another processor, to manage interrupt handling or to initialize and synchronize several processors.

A set of atomic instructions are also provided to synchronize memory accesses. Multiple processors can then access shared memory without inserting inaccuracies and ambiguities into shared data structures.

Fault Tolerance

The i960 family of components supports fault-tolerant system design through the use of the M82965 Bus Extension Unit component. The M82965 allows two processors to be operated in tandem to form a self-checking module. The two M82965s check the outputs of two processors (a master and a checker) cycle-by-cycle. If the checking M82965 detects a difference between outputs, it signals an error. A software recovery procedure can then be initiated.

This fault detection mechanism supports several fault detection and recovery techniques, including self healing, and continuous-operation (non-stop) systems.

LOOK FOR MORE IN THE FUTURE

The i960 architecture offers exceptional performance, plus a wealth of useful features to help in the design of efficient and reliable embedded systems. But equally important, it offers lots of room to grow. The i960 MC processor provides average instruction processing rates of 7.5 million instructions per second (7.5 MIPS) at 20 MHz clock rate and 10 MIPS at a 25 MHz clock rate(1).

However, the i960 MC processor is only the beginning. With improvements in VLSI technology, future implementations of the i960 architecture will offer even greater performance. They will also offer a variety of useful extensions to solve specific control and monitoring needs in the field of embedded applications.

1. 1 MIP is equivalent to the performance of a Digital Equipment Corp. VAX 11/780.

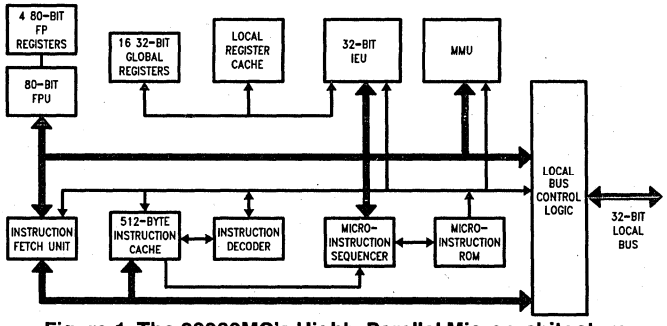
80960MC
EMBEDDED 32-BIT MICROPROCESSOR WITH INTEGRATED FLOATING-POINT UNIT AND MEMORY MANAGEMENT UNIT

Military

- **High-Performance Embedded Architecture**
  - **— 25 MIPS Burst Execution at 25 MHz**
  - **— 9.4 MIPS* Sustained Execution at 25 MHz**
- **On-Chip Floating-Point Unit**
  - **— Supports IEEE 754 Floating-Point Standard**
  - **— Full Transcendental Support**
  - **— Four 80-Bit Registers**
  - **— 5.2 Million Whetstones/Second at 25 MHz**
- **512-Byte On-Chip Instruction Cache**
  - **— Direct Mapped**
  - **— Parallel Load/Decode for Uncached Instructions**
- **Multiple Register Sets**
  - **— Sixteen Global 32-Bit Registers**
  - **— Sixteen Local 32-Bit Registers**
  - **— Four Local Register Sets Stored On-Chip (Sixteen 32-Bit Registers per Set)**
  - **— Register Scoreboarding**
- **On-Chip Memory Management Unit**
  - **— 4 Gigabyte Virtual Address Space per Task**
  - **— 4 Kbyte Pages with Supervisor/User Protection**
- **Built-In Interrupt Controller**
  - **— 32 Priority Levels**
  - **— 248 Vectors**
  - **— Supports M8259A**
  - **— 3.4 μs Latency**
- **Easy to Use, High Bandwidth 32-Bit Bus**
  - **— 66.7 MBytes/s Burst**
  - **— Up to 16-Bytes Transferred per Burst**
- **Multitasking and Multiprocessor Support**
  - **— Automatic Task Dispatching**
  - **— Prioritized Task Queues**
- **Advanced Package Technology**
  - **— 132 Lead Ceramic Pin Grid Array**
  - **— 164 Lead Ceramic Quad Flatpack**
- **Military Temperature Range**
  - **— $-55°C$ to $+125°C$ (T_C)**

The 80960MC is the enhanced military member of Intel's new 32-bit microprocessor family, the 960 series, which is designed especially for embedded applications. It is based on the family's high performance, common core architecture, and includes a 512-byte instruction cache, a built-in interrupt controller, an integrated floating-point unit and a memory management unit. The 80960MC has a large register set, multiple parallel execution units, and a high-bandwidth, burst bus. Using advanced RISC technology, this high performance processor can respond to interrupts in under 3.4 μs and is capable of execution rates in excess of 9.4 million instructions per second.* The 80960MC is well-suited for a wide range of military and other high reliability applications, including avionics, airborne radar, navigation, and instrumentation.

*Relative to Digital Equipment Corporation's VAX-11/780** at 1 MIPS



271080-1

Figure 1. The 80960MC's Highly Parallel Microarchitecture

**VAX-11™ is a trademark of Digital Equipment Corporation.

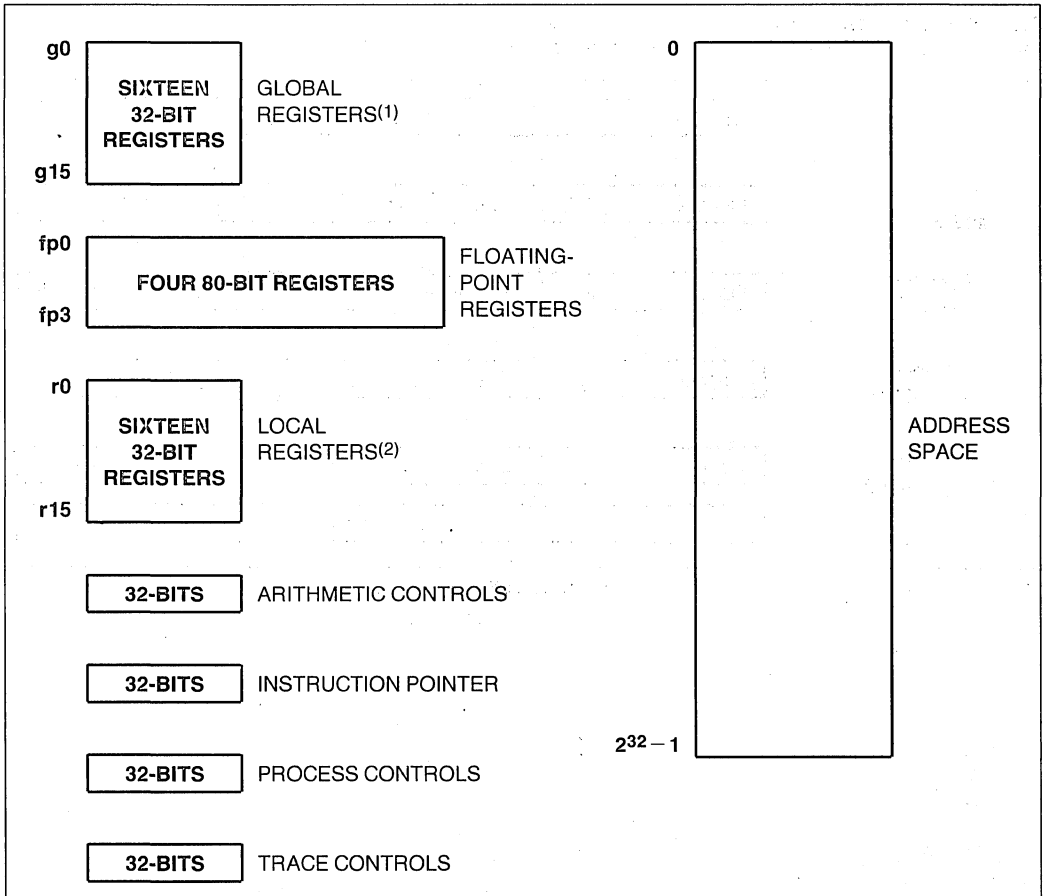
February 1991
Order Number: 271080-007

THE 960 SERIES

The 80960MC is the enhanced military member of a new family of 32-bit microprocessors from Intel known as the 960 Series. This series was especially designed to serve the needs of embedded applications. The embedded market includes applications as diverse as industrial automation, avionics, image processing, graphics, robotics, telecommunications, and automobiles. These types of applications require high integration, low power consumption, quick interrupt response times, and high performance. Since time to market is critical, embedded microprocessors need to be easy to use in both hardware and software designs.

All members of the 80960 series share a common core architecture which utilizes RISC technology so that, except for special functions, the family members are object code compatible. Each new processor in the series will add its own special set of functions to the core to satisfy the needs of a specific application or range of applications in the embedded market. For example, future processors may include a DMA controller, a timer, or an A/D converter.

The 80960MC includes an integrated Floating Point Unit (FPU), a Memory Management Unit (MMU), multitasking support, and multiprocessor support. There are also two commercial members of the family: the 80960KB processor with integrated FPU and the 80960KA without floating-point.



NOTES:
1. Register g15 is reserved for stack management functions.
2. Registers r0, r1, and r2 are reserved for stack management functions.

Figure 2. Register Set

KEY PERFORMANCE FEATURES

The 80960MC's architecture is based on the most recent advances in RISC technology and is grounded in Intel's long experience in designing embedded controllers. Many features contribute to the 80960MC's exceptional performance:

1. Large Register Set. Having a large number of registers reduces the number of times that a processor needs to access memory. Modern compilers can take advantage of this feature to optimize execution speed. For maximum flexibility, the 80960MC provides thirty-two 32-bit registers (sixteen local and sixteen global) and four 80-bit floating-point global registers. (See Figure 2.)

2. Fast Instruction Execution. Simple functions make up the bulk of instructions in most programs, so that execution speed can be greatly improved by ensuring that these core instructions execute in as short a time as possible. The most-frequently executed instructions such as register-register moves, add/subtract, logical operations, and shifts execute in one to two cycles (Table 1 contains a list of instructions.)

3. Load/Store Architecture. Like other processors based on RISC technology, the 80960MC has a Load/Store architecture, only the LOAD and STORE instructions reference memory; all other instructions operate on registers. This type of architecture simplifies instruction decoding and is used in combination with other techniques to increase parallelism.

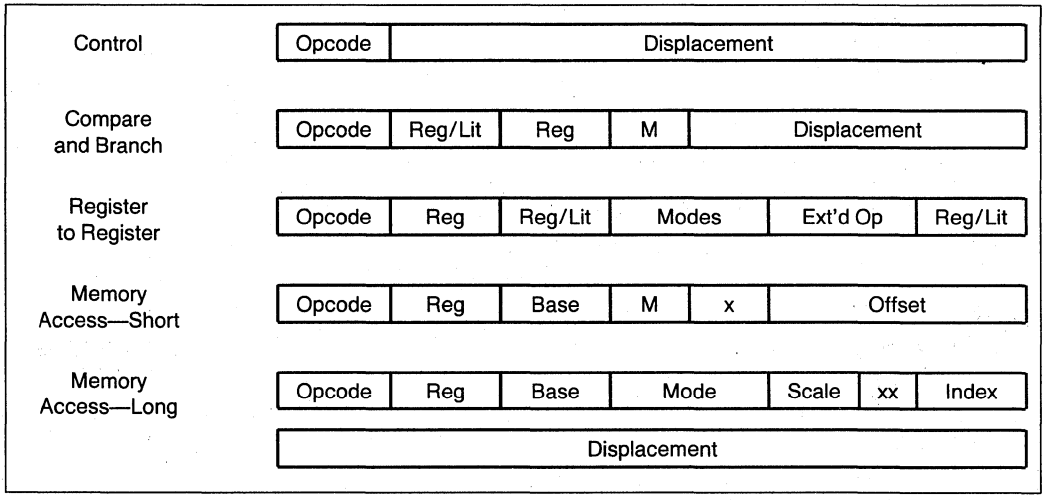


Figure 3. Instruction Formats

Table 1. 80960MC Instruction Set

Data Movement	Arithmetic	Floating Point	Logical
Load Store Move Load Address Load Physical Address	Add Subtract Multiply Divide Remainder Modulo Shift	Add Subtract Multiply Divide Remainder Scale Round Square Root Sine Cosine Tangent Arctangent Log Log Binary Log Natural Exponent Classify Copy Real Extended Compare	And Not And And Not Or Exclusive Or Not Or Or Not Nor Exclusive Nor Not Nand Rotate
Comparison	**Branch**	**Bit and Bit Field**	**String**
Compare Conditional Compare Compare and Increment Compare and Decrement	Unconditional Branch Conditional Branch Compare and Branch	Set Bit Clear Bit Not Bit Check Bit Alter Bit Scan for Bit Scan over Bit Extract Modify	Move String Move Quick String Fill String Compare String Scan Byte for Equal
Conversion	**Decimal**	**Call/Return**	**Process Management**
Convert Real to Integer Convert Integer to Real	Move Add with Carry Subtract with Carry	Call Call Extended Call System Return Branch and Link	Schedule Process Saves Process Resume Process Load Process Time Modify Process Controls Wait Conditional Wait Signal Receive Conditional Receive Send Send Service Atomic Add Atomic Modify
Fault	**Debug**	**Miscellaneous**	
Conditional Fault Synchronize Faults	Modify Trace Controls Mark Force Mark	Flush Local Registers Inspect Access Modify Arithmetic Controls Test Condition Code	

4. Simple Instruction Formats. All instructions in the 80960MC are 32-bits long and must be aligned on word boundaries. This alignment makes it possible to eliminate the instruction-alignment stage in the pipeline. To simplify the instruction decoder further, there are only five instruction formats and each instruction uses only one format. (See Figure 3.)

5. Overlapped Instruction Execution. A load operation allows execution of subsequent instructions to continue before the data has been returned from memory, so that these instructions can overlap the load. The 80960MC manages this process transparently to software through the use of a register scoreboard. Conditional instructions also make use of a scoreboard so that subsequent unrelated instructions can be executed while the conditional instruction is pending.

6. Integer Execution Optimization. When the result of an operation is used as an operand in a subsequent calculation, the value is sent immediately to its destination register. Yet at the same time, the value is put back on a bypass path to the ALU, thereby saving the time that otherwise would be required to retrieve the value for the next operation.

7. Bandwidth Optimizations. The 80960MC gets optimal use of its memory bus bandwidth because the bus is tuned for use with the cache: the line size of the instruction cache matches the maximum burst size for instruction fetches. The 80960MC automatically fetches four words in a burst and stores them directly in the cache. Due to the size of the cache and the fact that it is continually filled in anticipation of needed instructions in the program flow, the 80960MC is exceptionally insensitive to memory wait states. In fact, each wait state causes only a 7% degradation in system perfomance. The benefit is that the 80960MC will deliver outstanding performance even with a low cost memory system.

8. Cache Bypass. If there is a cache miss, the processor fetches the needed instruction, then sends it on to the instruction decoder at the same time it updates the cache. Thus, no extra time is taken to load and read the cache.

Memory Space and Addressing Modes

The 80960MC allows each task (process) to address a logical memory space of up to 4 Gbytes. In turn, each task's address space is divided into four 1-Gbyte regions and each region can be mapped to physical addresses by zero, one, or two levels of page tables. The region with the highest addresses (Region 3) is common to all tasks.

In keeping with RISC design principles, the number of addressing modes has been kept to a minimum but includes all those necessary to ensure efficient execution of high-level languages such as Ada, C, and Fortran. Table 2 lists the memory addressing modes.

Data Types

The 80960MC recognizes the following data types:

Numeric:
- 8-, 16-, 32- and 64-bit ordinals
- 8-, 16, 32- and 64-bit integers
- 32-, 64- and 80-bit real numbers

Non-Numeric:
- Bit
- Bit Field
- Triple-Word (96 bits)
- Quad-Word (128 bits)

Large Register Set

The programming environment of the 80960MC includes a large number of registers. In fact, 36 registers are available at any time. The availability of this many registers greatly reduces the number of memory accesses required to execute most programs, which leads to greater instruction processing speed.

There are two types of general-purpose registers: local and global. The 20 global registers consist of sixteen 32-bit registers (G0 through G15) and four 80-bit registers (FP0 through FP3). These registers perform the same function as the general-purpose registers provided in other popular microprocessors. The term global refers to the fact that these registers retain their contents across procedure calls.

The local registers, on the other hand, are procedure specific. For each procedure call, the 80960MC allocates 16 local registers (R0 through R15). Each local register is 32 bits wide. Any register can also be used for floating-point operations; the 80-bit floating-point registers are provided for extended precision.

Multiple Register Sets

To further increase the efficiency of the register set, multiple sets of local registers are stored on-chip. This cache holds up to four local register frames, which means that up to three procedure calls can be made without having to access the procedure stack resident in memory.

Table 2. Memory Addressing Modes

- 12-Bit Offset
- 32-Bit Offset
- Register-Indirect
- Register + 12-Bit Offset
- Register + 32-Bit Offset
- Register + (Index-Register × Scale-Factor)
- Register × Scale Factor + 32-Bit Displacement
- Register + (Index-Register × Scale-Factor) + 32-Bit Displacement

Scale-Factor is 1, 2, 4, 8 or 16

Although programs may have procedure calls nested many calls deep, a program typically oscillates back and forth between only two or three levels. As a result, with four stack frames in the cache, the probability of there being a free frame on the cache when a call is made is very high. In fact, runs of representative C-language programs show that 80% of the calls are handled without needing to access memory.

If there are four or more active procedures and a new procedure is called, the processor moves the oldest set of local registers in the register cache to a procedure stack in memory to make room for a new set of registers. Global register G15 is used by the processor as the frame pointer (FP) for the procedure stack.

Note that the global and floating-point registers are not exchanged on a procedure call, but retain their contents, making them available to all procedures for fast parameter passing. An illustration of the register cache is shown in Figure 4.

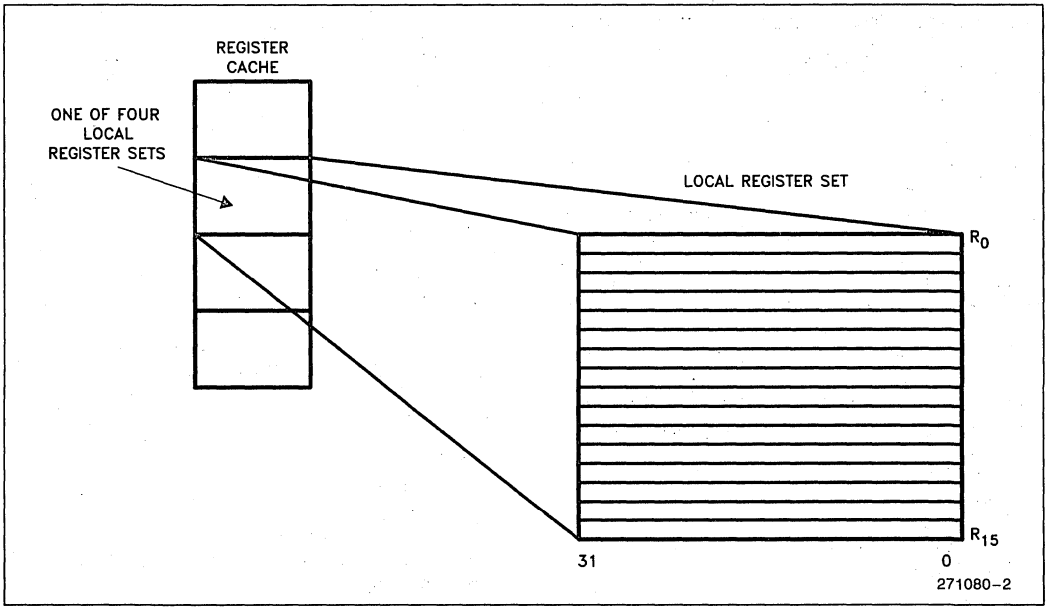


Figure 4. Multiple Register Sets Are Stored On-Chip

Instruction Cache

To further reduce memory accesses, the 80960MC includes a 512-byte on-chip instruction cache. The instruction cache is based on the concept of locality of reference; that is, most programs are not usually executed in a steady stream but consist of many branches and loops that lead to jumping back and forth within the same small section of code. Thus, by maintaining a block of instructions in a cache, the number of memory references required to read instructions into the processor can be greatly reduced.

To load the instruction cache, instructions are fetched in 16-byte blocks, so that up to four instructions can be fetched at one time. An efficient prefetch algorithm increases the probability that an instruction will already be in the cache when it is needed.

Code for small loops will often fit entirely within the cache, leading to a great increase in processing speed since further memory references might not be necessary until the program exits the loop. Similarly, when calling short procedures, the code for the calling procedure is likely to remain in the cache, so it will be there on the procedure's return.

Register Scoreboarding

The instruction decoder has been optimized in several ways. One of these optimizations is the ability to do instruction overlapping by means of register scoreboarding.

Register scoreboarding occurs when a LOAD instruction is executed to move a variable from memory into a register. When the instruction is initiated, a scoreboard bit on the target register is set. When the register is actually loaded, the bit is reset. In between, any reference to the register contents is accompanied by a test of the scoreboard bit to insure that the load has completed before processing continues. Since the processor does not have to wait for the LOAD to be completed, it can go on to execute additional instructions placed in between the LOAD instruction and the instruction that uses the register contents, as shown in the following example:

```
LOAD R4, address 1
LOAD R5, address 2
Unrelated instruction
Unrelated instruction
ADD R4, R5, R6
```

In essence, the two unrelated instructions between the LOAD and ADD instructions are executed for free (i.e., take no apparent time to execute) because they are executed while the register is being loaded. Up to three LOAD instructions can be pending at one time with three corresponding scoreboard bits set. By exploiting this feature, system programmers and compilers have a useful tool for optimizing execution speed.

Memory Management and Protection

The 80960MC will be especially useful for multitasking applications that require software protection and a very large address space. To ensure the highest level of performance possible, the memory management unit and translation look-aside buffer (TLB) are contained on-chip.

The 80960MC supports a conventional form of demand-paged virtual memory in which the address space is divided into 4 Kbyte pages. Studies have shown that a 4 Kbyte page is the optimum size for a broad range of applications.

Each page table entry includes a 2-bit page rights field that specifies whether the page is a no-access, read-only, or read-write page. This field is interpreted differently depending on whether the current task (process) is executing in user or supervisor mode, as shown below:

Rights	User	Supervisor
00	No Access	Read-Only
01	No Access	Read-Write
10	Read-Only	Read-Write
11	Read-Write	Read-Write

Floating-Point Arithmetic

In the 80960MC, floating-point arithmetic has been made an integral part of the architecture. Having the floating-point unit integrated on-chip provides two advantages. First, it improves the performance of the chip for floating-point applications, since no additional bus overhead is associated with floating-point calculations, thereby leaving more time for other bus operations such as I/O. Second, the cost of using floating-point operations is reduced because a separate coprocessor chip is not required.

The 80960MC floating-point (real number) data types include single-precision (32-bit), double-precision (64-bit), and extended precision (80-bit) floating-point numbers. Any register may be used to execute floating-point operations.

The processor provides hardware support for both mandatory and recommended portions of IEEE Standard 754 for floating-point arithmetic, including all arithmetic, exponential, logarithmic, and other transcendental functions. Table 3 shows execution times for some representative instructions.

Table 3. Sample Floating-Point Execution Times (μs) at 25 MHz

	32-Bit	64-Bit
Add	0.4	0.5
Subtract	0.4	0.5
Multiply	0.7	1.3
Divide	1.3	2.9
Square Root	3.7	3.9
Arctangent	10.1	13.1
Exponent	11.3	12.5
Sine	15.2	16.6
Cosine	15.2	16.6

Multitasking Support

Multitasking programs commonly involve the monitoring and control of an external operation, such as the activities of a process controller or the movements of a machine tool. These programs generally consist of a number of processes that run independently of one another, but share a common database or pass data among themselves.

The 80960MC offers several hardware functions designed to support multitasking systems. One unique feature, called self-dispatching, allows a processor to switch itself automatically among scheduled tasks. When self-dispatching is used, all the operating system is required to do is place the task in the scheduling queue.

When the processor becomes available, it dispatches the task from the beginning of the queue and then executes it until it becomes blocked, interrupted, or until its time-slice expires. It then returns the task to the end of the queue (i.e., automatically reschedules it) and dispatches the next ready task.

During these operations, no communication between the processor and the operating system is necessary until the running task is complete or an interrupt is issued.

Synchronization and Communication

The 80960MC also offers instructions to set up and test semaphores to ensure that concurrent tasks remain synchronized and no data inconsistency results. Special data structures, known as communication ports, provide the means for exchanging parameters and data structures. Transmission of information by means of communication ports is asynchronous and automatically buffered by the processor.

Communication between tasks by means of ports can be carried out independently of the operating system. Once the ports have been set up by the programmer, the processor handles the message passing automatically.

3

High Bandwidth Local Bus

An 80960MC CPU resides on a high-bandwidth address/data bus known as the local bus (L-Bus). The L-Bus provides a direct communication path between the processor and the memory and I/O subsystem interfaces. The processor uses the local bus to fetch instructions, manipulate memory, and respond to interrupts. Its features include:

- 32-bit multiplexed address/data path
- Four-word burst capability, which allows transfers from 1 to 16 bytes at a time
- High bandwidth reads and writes at 66.7 MBytes per second
- Special signal to indicate whether a memory transaction can be cached

Figure 5 identifies the groups of signals which constitute the L-Bus. Table 4 lists the function of the L-Bus and other processor-support signals, such as the interrupt lines.

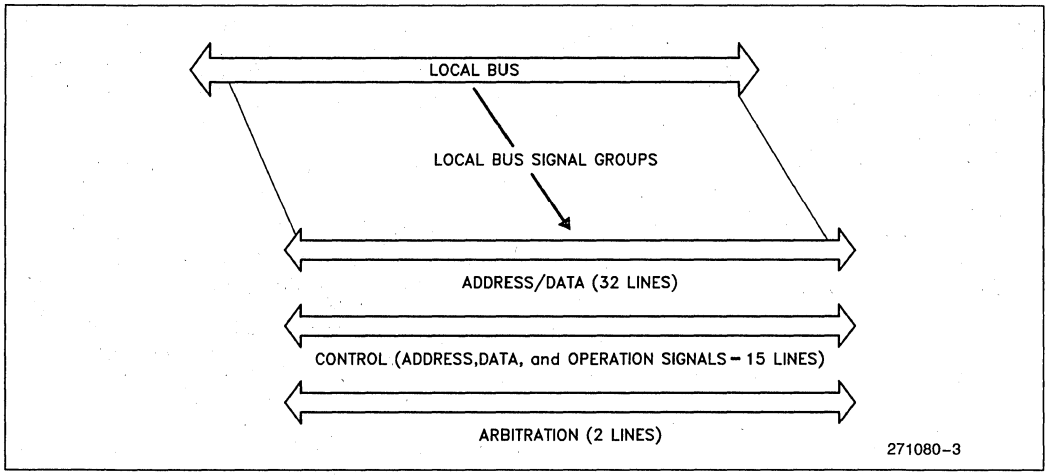


Figure 5. Local Bus Signal Groups

Multiple Processor Support

One means of increasing the processing power of a system is to run two or more processors in parallel. Since microprocessors are not generally designed to run in tandem with other processors, designing such a system is usually difficult and costly.

The 80960MC solves this problem by offering a number of functions to coordinate the actions of multiple processors. First, messages can be passed between processors to initiate actions such as flushing a cache, stopping or starting another processor, or preempting a task. The messages are passed on the bus and allow multiple processors to run together smoothly, with rare need to lock the bus or memory.

Second, a set of synchronization instructions help maintain the coherency of memory. These instructions permit several processors to modify memory at the same time without inserting inaccuracies or ambiguities into shared data structures.

The self-dispatching mechanism, in addition to being used in single-processor systems, provides the means to increase the performance of a system merely by adding processors. Each processor can either work on the same pool of tasks (sharing the same queue with other processors) or can be restricted to its own queue.

When processors perform system operations, they synchronize themselves by using atomic operations and sending special messages between each other. And changing the number of processors in a system never requires a software change. Software will execute correctly regardless of the number of processors in the system; systems with more processors simply execute faster.

Interrupt Handling

The 80960MC can be interrupted in one of two ways: by the activation of one of four interrupt pins or by sending a message on the processor's data bus.

The 80960MC is unusual in that it automatically handles interrupts on a priority basis and tracks pending interrupts through its on-chip interrupt controller. Two of the interrupt pins can be configured to provide M8259A handshaking for expansion beyond four interrupt lines.

An interrupt message is made up of a vector number and an interrupt priority. If the interrupt priority is greater than that of the currently running task, the processor accepts the interrupt and uses the vector as an index into the interrupt table. If the priority of the interrupt message is below that of the current task, the processor saves the information in a section of the interrupt table reserved for pending interrupts.

Debug Features

The 80960MC has built-in debug capabilities. There are two types of breakpoints and six different trace modes. The debug features are controlled by two

internal 32-bit registers, the Process-Controls Word and the Trace-Controls Word. By setting bits in these control words, a software debug monitor can closely control how the processor responds during program execution.

The 80960MC has both hardware and software breakpoints. It provides two hardware breakpoint registers on-chip which can be set by a special command to any value. When the instruction pointer matches the value in one of the breakpoint registers, the breakpoint will fire, and a breakpoint handling routine is called automatically.

The 80960MC also provides software breakpoints through the use of two instructions, MARK and FMARK. These instructions can be placed at any point in a program and will cause the processor to halt execution at that point and call the breakpoint handling routine. The breakpoint mechanism is easy to use and provides a powerful debugging tool.

Tracing is available for instructions (single-step execution), calls and returns, and branching. Each different type of trace may be enabled separately by a special debug instruction. In each case, the 80960MC executes the instruction first and then calls a trace handling routine (usually part of a software debug monitor). Further program execution is halted until the trace routine is completed. When the trace event handling routine is completed, instruction execution resumes at the next instruction. The 80960MC's tracing mechanisms, which are implemented completely in hardware, greatly simplify the task of testing and debugging software.

FAULT DETECTION

The 80960MC has an automatic mechanism to handle faults. There are ten fault types including trace, arithmetic, and floating-point faults. When the processor detects a fault, it automatically calls the appropriate fault handling routine and saves the current instruction pointer and necessary state information to make efficient recovery possible. The processor posts diagnostic information on the type of fault to a Fault Record. Like interrupt handling routines, fault handling routines are usually written to meet the needs of a specific application and are often included as part of the operating system or kernel.

For each of the ten fault types, there are numerous subtypes that provide specific information about a fault. For example, a floating-point fault may have its subtype set to an Overflow or Zero-Divide fault. The fault handler can use this specific information to respond correctly to the fault.

Interagent Communications (IAC)

In order to coordinate their actions, processors in a multiple processor system need a means for communicating with each other. The 80960MC does this through a mechanism known as Interagent Communication messages or IACs.

IAC messages cause a variety of actions including starting and stopping processors, flushing instruction caches and TLBs, and sending interrupts to other processors in the system. The upper 16 Mbytes of the processor's physical memory space is reserved for sending and receiving IAC messages.

BUILT-IN TESTABILITY

Upon reset, the 80960MC automatically conducts an exhaustive internal test of its major blocks of logic.

Then, before executing its first instruction, it does a zero check sum on the first eight words in memory to ensure that the system has been loaded correctly. If a problem is discovered at any point during the self-test, the 80960MC will assert its FAILURE pin and will not begin program execution. The self-test takes approximately 47,000 cycles to complete.

System manufacturers can use the 80960MC's self-test feature during incoming parts inspection. No special diagnostic programs need to be written, and the test is both thorough and fast. The self-test capability helps ensure that defective parts will be discovered before systems are shipped, and once in the field, the self-test makes it easier to distinguish between problems caused by processor failure and problems resulting from other causes.

COMPATIBILITY WITH 80960K-SERIES

Application programs written for the 80960K-Series microprocessors can be run on the 80960MC without modification. The 80960K-Series instruction set forms the core of the 80960MC's instructions, so binary compatibility is assured.

CHMOS

The 80960MC is fabricated using Intel's CHMOS IV (Complementary High Speed Metal Oxide Semiconductor) process. This advanced technology eliminates the frequency and reliability limitations of older CMOS processes and opens a new era in microprocessor performance. It combines the high performance capabilities of Intel's industry-leading HMOS technology with the high density and low power characteristics of CMOS. The 80960MC is available at 16, 20 and 25 MHz.

Table 4a. 80960MC Pin Description: L-Bus Signals

Symbol	Type	Name and Function
CLK2	I	**SYSTEM CLOCK** provides the fundamental timing for 80960MC systems. It is divided by two inside the 80960MC to generate the internal processor clock. CLK2 is shown in Figure 9.
LAD_{31} $-LAD_0$	I/O T.S.	**LOCAL ADDRESS/DATA BUS** carries 32-bit physical addresses and data to and from memory. During an address (T_a) cycle, bits 2–31 contain a physical word address (bits 0–1 indicate SIZE; see below). During a data (T_d) cycle, bits 0–31 contain read or write data. The LAD lines are active HIGH and float to a high impedance state when not active. **SIZE**, which is comprised of bits 0–1 of the LAD lines during a T_a cycle, specifies the size of a transfer in words for a burst transaction. $\underline{LAD_1}$ $\underline{LAD_0}$ 0 0 1 Word 0 1 2 Words 1 0 3 Words 1 1 4 Words
$\overline{ALE}$	O T.S.	**ADDRESS-LATCH ENABLE** indicates the transfer of a physical address. ALE is asserted during a T_a cycle and deasserted before the beginning of the T_d state. It is active LOW and floats to a high impedance state when the processor is idle or is at the end of any bus access.
$\overline{ADS}$	O O.D.	**ADDRESS STATUS** indicates an address state. ADS is asserted every T_a state and deasserted during the the following T_d state. For a burst transaction, $\overline{ADS}$ is asserted again every T_d state where $\overline{READY}$ was asserted in the previous cycle.
W/$\overline{R}$	O O.D.	**WRITE/READ** specifies, during a T_a cycle, whether the operation is a write or read. It is latched on-chip and remains valid during T_d and T_w states.
DT/$\overline{R}$	O O.D.	**DATA TRANSMIT/RECEIVE** indicates the direction of data transfer to and from the L-Bus. It is low during T_a, T_w and T_d cycles for a read or interrupt acknowledgement; it is high during T_a, T_w and T_d cycles for a write. DT/$\overline{R}$ never changes state when $\overline{DEN}$ is asserted (see Timing Diagrams).
$\overline{DEN}$	O O.D.	**DATA ENABLE** is asserted during T_d and T_w cycles and indicates transfer of data on the LAD bus lines.
$\overline{READY}$	I	**READY** indicates that data on LAD lines can be sampled or removed. If $\overline{READY}$ is not asserted during a T_d cycle, the T_d cycle is extended to the next cycle by inserting wait states (T_w), and $\overline{ADS}$ is not asserted in the next cycle.
$\overline{LOCK}$	I/O O.D.	**BUS LOCK** prevents other bus masters from gaining control of the L-Bus following the current cycle (if they would assert $\overline{LOCK}$ to do so). $\overline{LOCK}$ is used by the processor or any bus agent when it performs indivisible Read/Modify/Write (RMW) operations. For a read that is designated as a RMW-read, $\overline{LOCK}$ is examined. if asserted, the processor waits until it is not asserted; if not asserted, the processor asserts $\overline{LOCK}$ during the T_a cycle and leaves it asserted. A write that is designated as an RMW-write deasserts $\overline{LOCK}$ in the T_a cycle.

I/O = Input/Output, O = Output, I = Input, O.D. = Open-Drain, T.S. = three state
$T_a = T_{Address}$, $T_d = T_{Data}$, $T_W = T_{Wait}$, $T_r = T_{Recovery}$, $T_i = T_{Idle}$, $T_h = T_{Hold}$

Table 4a. 80960MC Pin Description: L-Bus Signals (Continued)

Symbol	Type	Name and Function
$\overline{BE_3}$–$\overline{BE_0}$	O O.D.	**BYTE ENABLE LINES** specify which data bytes (up to four) on the bus take part in the current bus cycle. $\overline{BE_3}$ corresponds to LAD_{31}–LAD_{24} and $\overline{BE_0}$ corresponds to LAD_7–LAD_0. The byte enables are provided in advance of data. The byte enables asserted during T_a specify the bytes of the first data word. The byte enables asserted during T_d specify the bytes of the next data word (if any), that is, the word to be transmitted following the next assertion of $\overline{READY}$. The byte enables during the T_d cycles preceding the last assertion of $\overline{READY}$ are undefined. The byte enables are latched on-chip and remain constant from one T_d cycle to the next when $\overline{READY}$ is not asserted. For reads, the byte enables specify the byte(s) that the processor will actually use. 80960MC's will assert only adjacent byte enables (e.g., asserting just $\overline{BE_0}$ and $\overline{BE_2}$ is not permitted), and are required to assert at least one byte enable. Accesses must also be naturally aligned (e.g., asserting $\overline{BE_1}$ and $\overline{BE_2}$ is not allowed even though they are adjacent). To produce address bits A_0 and A_1 externally, they can be decoded from the byte enables.
HOLD (HLDAR)	I	**HOLD** indicates a request from a secondary bus master to acquire the bus. If the processor is initialized as the primary bus master this input will be interpreted as HOLD. When the processor receives HOLD and grants another master control of the bus, it floats its three-state bus lines, asserts HOLD ACKNOWLEDGE, and enters the T_h state. When HOLD is deasserted, the processor will deassert HOLD ACKNOWLEDGE and go to either the T_i or T_a state. **HOLD ACKNOWLEDGE RECEIVED** indicates that the processor has acquired the bus. If the processor is initialized as the secondary bus master this input is interpreted as HLDAR. HOLD timing is shown in Figure 11.
HLDA (HOLDR)	O T.S.	**HOLD ACKNOWLEDGE** relinquishes control of the bus to another bus master. If the processor is initialized as the primary bus master this output will be interpreted as HLDA. When HOLD is deasserted, the processor will deassert HLDA and go to either the T_i or T_a state. **HOLD REQUEST** indicates a request to acquire the bus. If the processor is initialized as the secondary bus master this output will be interpreted as HOLDR. HOLD timing is shown in Figure 11.
CACHE	O T.S.	**CACHE** indicates if an access is cacheable during a T_a cycle. The CACHE signal floats to a high impedance state when the processor is idle.

I/O = Input/Output, O = Output, I = Input, O.D. = Open-Drain, T.S. = three state
$T_a = T_{Address}$, $T_d = T_{Data}$, $T_W = T_{Wait}$, $T_r = T_{Recovery}$, $T_i = T_{Idle}$, $T_h = T_{Hold}$

Table 4b. 80960MC Pin Description: Module Support Signals

Symbol	Type	Name and Function
$\overline{BADAC}$	I	**BAD ACCESS,** if asserted in the cycle following the one in which the last $\overline{READY}$ of a transaction is asserted, indicates that an unrecoverable error has occurred on the current bus transaction, or that a synchronous load/store instruction has not been acknowledged. **STARTUP:** During system reset, the $\overline{BADAC}$ signal is interpreted differently. If the signal is high, it indicates that this processor will perform system initialization. If it is low, another processor in the system will perform system initialization instead.
RESET	I	**RESET** clears the internal logic of the processor and causes it to re-initialize. During RESET assertion, the input pins are ignored (except for $\overline{BADAC}$ and $\overline{IAC}/\overline{INT_0}$), the tri-state output pins are placed in a high impedance state, and other output pins are placed in their non-asserted state. RESET must be asserted for at least 41 CLK2 cycles for a predictable RESET. The HIGH to LOW transition of RESET should occur after the rising edge of both CLK2 and the external bus CLK, and before the next rising edge of CLK2. RESET timing is shown in Figure 10.
$\overline{FAILURE}$	O O.D.	**INITIALIZATION FAILURE** indicates that the processor has failed to initialize correctly. After RESET is deasserted and before the first bus transaction begins, $\overline{FAILURE}$ is asserted while the processor performs a self-test. If the self-test completes successfully, then $\overline{FAILURE}$ is deasserted. Next, the processor performs a zero checksum on the first eight words of memory. If it fails, $\overline{FAILURE}$ is asserted for a second time and remains asserted; if it passes, system initialization continues and $\overline{FAILURE}$ remains deasserted.
N.C.	N/A	**NOT CONNECTED** indicates pins should not be connected. Never connect any pin marked N.C.
$\overline{IAC}$ ($\overline{INT_0}$)	I	**INTERAGENT COMMUNICATION REQUEST/INTERRUPT 0** indicates either that there is a pending IAC message for the processor or an interrupt. The bus interrupt control register determines in which way the signal should be interpreted. To signal an interrupt or IAC request in a synchronous system, this pin (as well as the other interrupt pins) must be enabled by being deasserted for at least one bus cycle and then asserted for at least one additional bus cycle; in an asynchronous system, the pin must remain deasserted for at least two bus cycles and then be asserted for at least two more bus cycles. **LOCAL PROCESSOR NUMBER:** This signal is interpreted differently during system reset. If the signal is at a high voltage level, it indicates that this processor is a primary bus master (Local Processor Number = 0); if it is at a low voltage level, it indicates that this processor is a secondary bus master (Local Processor Number = 1).
INT_1	I	**INTERRUPT 1,** like $\overline{INT_0}$, provides direct interrupt signaling.
INT_2 (INTR)	I	**INTERRUPT 2/INTERRUPT REQUEST:** The bus control registers determines how this pin is interpreted. If INT_2, it has the same interpretation as the $\overline{INT_0}$ and INT1 pins. If INTR, it is used to receive an interrupt request from an external interrupt controller.
$\overline{INT_3}$ ($\overline{INTA}$)	I/O O.D.	**INTERRUPT 3/INTERRUPT ACKNOWLEDGE:** The bus interrupt control register determines how this pin is interpreted. If $\overline{INT_3}$, it has the same interpretation as the $\overline{INT_0}$, INT_1, and INT_2 pins. If $\overline{INTA}$, it is used as an output to control interrupt-acknowledge bus transactions. The $\overline{INTA}$ output is latched on-chip and remains valid during T_d cycles; as an output, it is open-drain.

I/O = Input/Output, O = Output, I = Input, O.D. = Open-Drain, T.S. = three state
$T_a = T_{Address}$, $T_d = T_{Data}$, $T_W = T_{Wait}$, $T_r = T_{Recovery}$, $T_i = T_{Idle}$, $T_h = T_{Hold}$

ELECTRICAL SPECIFICATIONS

Power and Grounding

The 80960MC is implemented in CHMOS III technology and has modest power requirements. Its high clock frequency and numerous output buffers (address/data, control, error and arbitration signals) can cause power surges as multiple output buffers drive new signal levels simultaneously. For clean on-chip power distribution at high frequency, 12 V_{CC} and 13 V_{SS} pins separately feed functional units of the 80960MC.

Power and ground connections must be made to all power and ground pins of the 80960MC. On the circuit board, all V_{CC} pins must be strapped closely together, preferably on a power plane. Likewise, all V_{SS} pins should be strapped together, preferably on a ground plane.

Power Decoupling Recommendations

Liberal decoupling capacitance should be placed near the 80960MC. The processor can cause transient power surges when driving the L-Bus, particularly when it is connected to a large capacitive load.

Low inductance capacitors and interconnects are recommended for best high frequency electrical performance. Inductance can be reduced by shortening the board traces between the processor and decoupling capacitors as much as possible.

Connection Recommendations

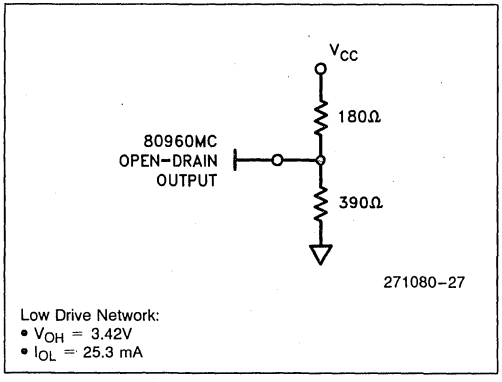
For reliable operation, always connect unused inputs to an appropriate signal level. In particular, if one or more interrupt lines are not used, they should be pulled up or down to their respective deasserted states. No inputs should ever be left floating.

All open-drain outputs require a pullup device. While in some cases a simple pullup resistor will be adequate, we recommend a network of pullup and pulldown resistors biased to a valid V_{IH} ($\geq$ 3.4V) and terminated in the characteristic impedance of the circuit board. Figure 6 shows our recommendations for the resistor values for both a low and high current drive network, which assumes that the circuit board has a characteristic impedance of 100Ω. The advantage of terminating the output signals in this fashion is that it limits signal swing and reduces AC power consumption.

Characteristic Curves

Figure 7 shows the typical supply current requirements over the operating temperature range of the processor at supply voltage (V_{CC}) of 5V. Figure 8 shows the typical power supply current (I_{CC}) required by the 80960MC at various operating frequencies when measured at three input voltage (V_{CC}) levels.

Figure 9 shows the typical capacitive derating curve for the 80960MC measured from 1.5V on the system clock (CLK) to 0.8V on the falling edge and 2.0V on the rising edge of the L-Bus address/data (LAD) signals.



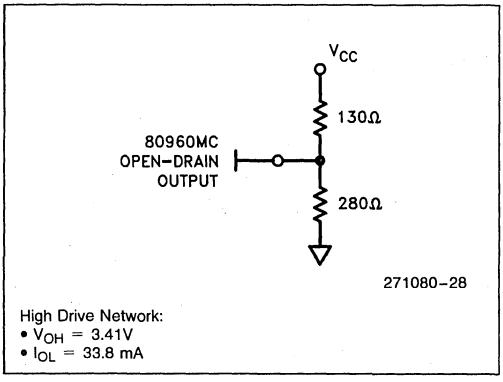


Figure 6. Connection Recommendations for Low and High Current Drive Networks

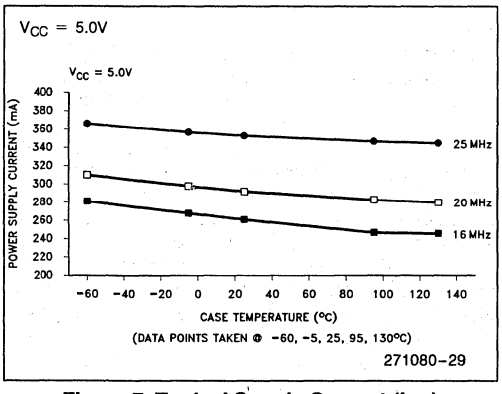


Figure 7. Typical Supply Current (I_{CC})

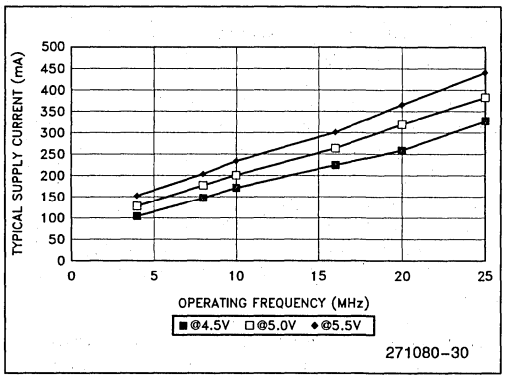


Figure 8. Typical Current vs Frequency

Test Load Circuit

Figure 10 illustrates the load circuit used to test the 80960MC's tristate pins, and Figure 11 shows the load circuit used to test the open drain outputs. The open drain test uses an active load circuit in the form of a matched diode bridge. Since the open-drain outputs sink current, only the I_{OL} legs of the bridge are necessary and the I_{OH} legs are not used. When the 80960MC driver under test is turned off, the output pin is pulled up to V_{REF} (i.e., V_{OH}). Diode D_1 is turned off and the I_{OL} current source flows through diode D_2.

When the 80960MC open-drain driver under test is on, diode D_1 is also on, and the voltage on the pin being tested drops to V_{OL}. Diode D_2 turns off and I_{OL} flows through diode D_1.

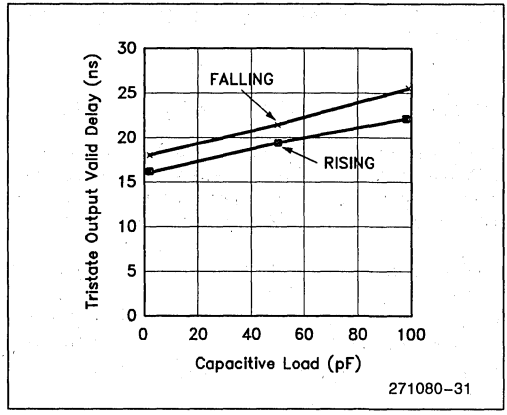


Figure 9. Capacitive Derating Curve

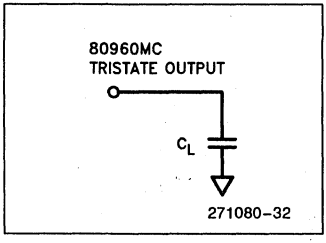


Figure 10. Test Load Circuit for TRI-STATE Output Pins

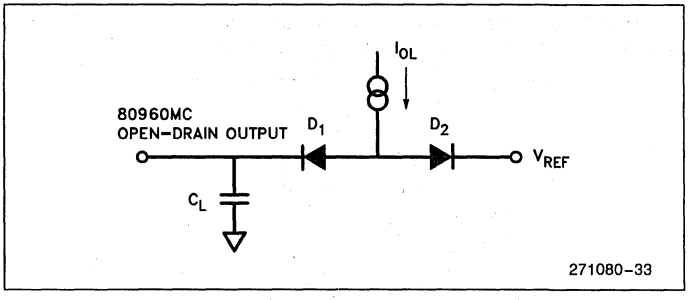


Figure 11. Test Load Circuit for Open-Drain Output Pins

ABSOLUTE MAXIMUM RATINGS*

Case Temperature under Bias[7] ..... −55°C to +125°C
Storage Temperature ..... −65°C to +150°C
Voltage on Any Pin ..... −0.5V to V_{CC} + 0.5V
Power Dissipation ..... 2.6W (25 MHz)

NOTICE: This data sheet contains information on products in the sampling and initial production phases of development. The specifications are subject to change without notice. Verify with your local Intel Sales office that you have the latest data sheet before finalizing a design.

WARNING: Stressing the device beyond the "Absolute Maximum Ratings" may cause permanent damage. These are stress ratings only. Operation beyond the "Operating Conditions" is not recommended and extended exposure beyond the "Operating Conditions" may affect device reliability.

D.C. CHARACTERISTICS

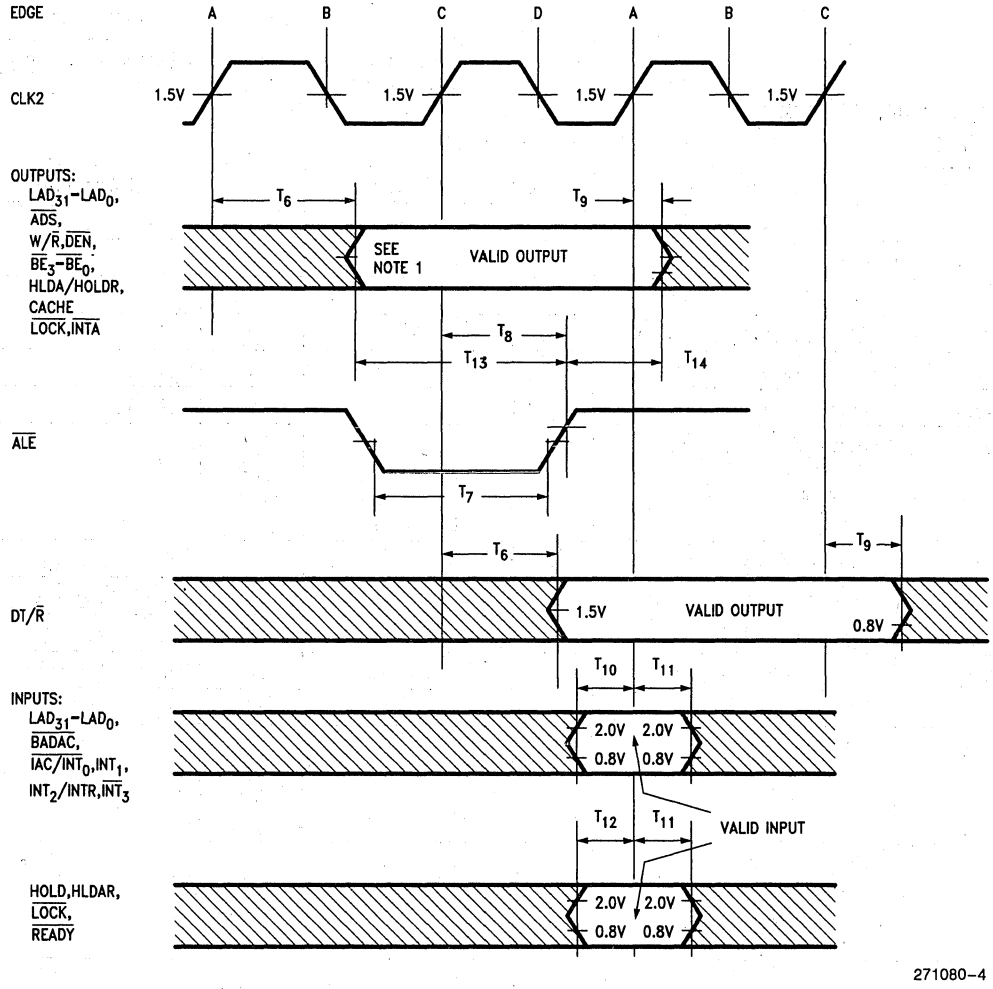
80960MC: T_{CASE}[6] = −55°C to +125°C, V_{CC} = 5V ± 5%

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.3	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.3	V	
V_{CL}	CLK2 Input Low Voltage	−0.3	+0.8	V	
V_{CH}	CLK2 Input High Voltage	0.55 V_{CC}	V_{CC} + 0.3	V	
V_{OL}	Output Low Voltage		0.45	V	(1, 5)
V_{OH}	Output High Voltage	2.4		V	(2, 4)
I_{CC}	Power Supply Current: 16 MHz 20 MHz 25 MHz		 375 420 480	 mA mA mA	
I_{LI}	Input Leakage Current		±15	μA	$0 \le V_O \le V_{CC}$
I_{LO}	Output Leakage Current		±15	μA	$0.45 \le V_O \le V_{CC}$
C_{IN}	Input Capacitance		10	pF	f_C = 1 MHz[3]
C_O	I/O or Output Capacitance		12	pF	f_C = 1 MHz[3]
C_{CLK}	Clock Capacitance		10	pF	f_C = 1 MHz[3]
θ_{JA}	Thermal Resistance (Junction-to-Ambient) Pin Grid Array Ceramic Quad Flatpack		 21 29	 °C/W °C/W	
θ_{JC}	Thermal Resistance (Junction-to-Case) Pin Grid Array Ceramic Quad Flatpack		 4 8	 °C/W °C/W	

NOTES:
1. For three-state outputs, this parameter is measured at:
   Address/Data ..... 4.0 mA
   Controls ..... 5.0 mA
2. This parameter is measured at:
   Address/Data ..... −1.0 mA
   Controls ..... −0.9 mA
   ALE ..... −5.0 mA
3. Input, output, and clock capacitance are not tested.
4. Not measured on open-drain outputs.
5. For open-drain outputs ..... 25 mA
6. Case temperatures are "instant on".

AC SPECIFICATIONS

This section describes the AC specifications for the 80960MC pins. All input and output timings are specified relative to the 1.5V level of the rising edge of CLK2, and refer to the time at which the signal reaches (for output delay and input setup) or leaves (for hold time) the TTL levels of LOW (0.8V) or HIGH (2.0V). All AC testing should be done with input voltages of 0.4V and 2.4V, except for the clock (CLK2), which should be tested with input voltages of 0.45V and 0.55 V_{CC}.



NOTE 1:
For Tri-State pins, T_6 and T_9 are measured at 1.5V.
For Open-Drain pins, T_6 is measured at 1.5V, T_9 at 0.8V.

Figure 12. Drive Levels and Timing Relationships for 80960MC Signals

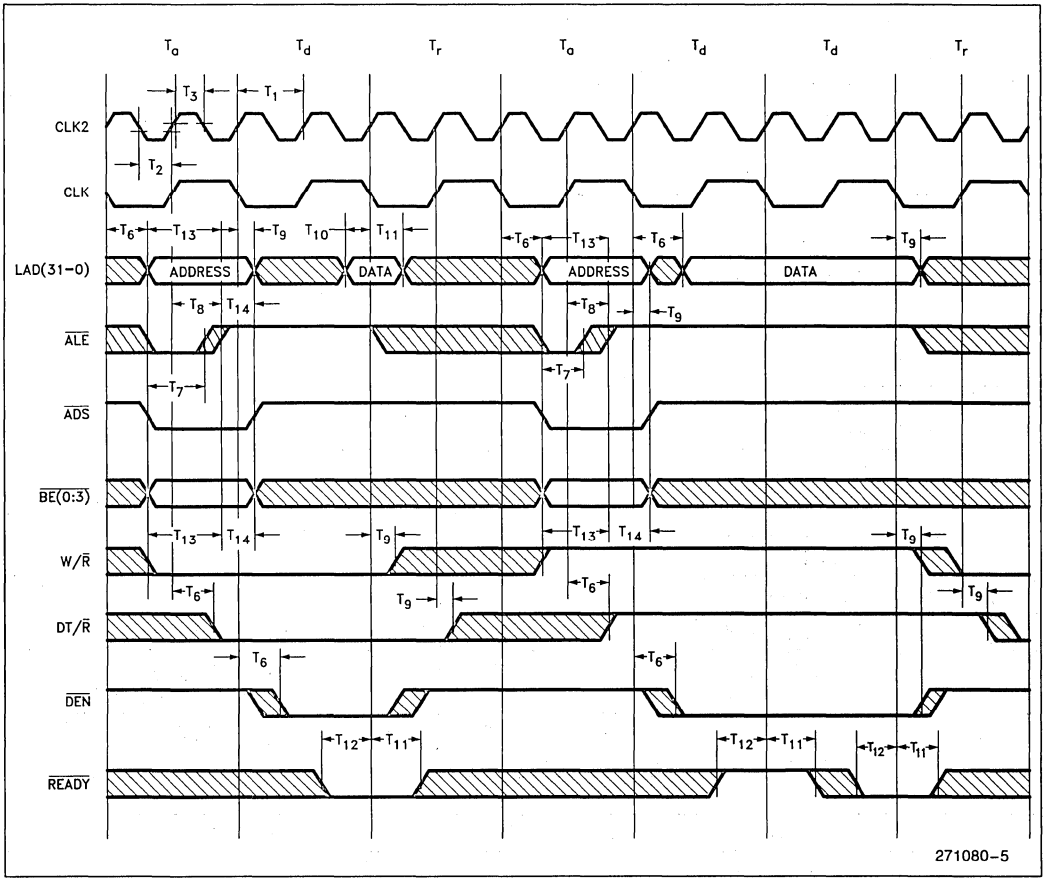


Figure 13. Timing Relationship of L-Bus Signals

A.C. Specification Tables

80960MC A.C. Characteristics (16 MHz)
T_{CASE}[3] = −55°C to +125°C, V_{CC} = 5V ±5%

Symbol	Parameter	Min	Max	Units	Test Conditions
T_1	Processor Clock Period (CLK2)	31.25	125	ns	V_{IN} = 1.5V
T_2	Processor Clock Low Time (CLK2)	8		ns	V_{IL} = 10% Point = 1.2V
T_3	Processor Clock High Time (CLK2)	8		ns	V_{IH} = 90% Point = 0.1V + 0.5 V_{CC}
T_4	Processor Clock Fall Time (CLK2)		10	ns	V_{IN} = 90% Point to 10% Point
T_5	Processor Clock Rise Time (CLK2)		10	ns	V_{IN} = 10% Point to 90% Point
T_6	Output Valid Delay	2	25	ns	C_L = 100 pF (LAD) C_L = 75 pF (Controls)
T_{6H}	HOLDA Output Valid Delay	4	31	ns	C_L = 75 pF
T_7	ALE Width	15		ns	C_L = 75 pF
T_8	ALE Invalid Delay	0	20	ns	C_L = 75 pF[2]
T_9	Output Float Delay	2	20	ns	C_L = 100 pF (LAD) C_L = 75 pF (Controls)[2]
T_{9H}	HOLDA Output Float Delay	4	20	ns	C_L = 75 pF
T_{10}	Input Setup 1	3		ns	(Note 1)
T_{11}	Input Hold	5		ns	(Note 1)
T_{11H}	HOLD Input Hold	4		ns	
T_{12}	Input Setup 2	8		ns	
T_{13}	Setup to ALE Inactive	10		ns	C_L = 100 pF (LAD) C_L = 75 pF (Controls)
T_{14}	Hold after ALE Inactive	8		ns	C_L = 100 pF (LAD) C_L = 75 pF (Controls)
T_{15}	Reset Hold	3		ns	
T_{16}	Reset Setup	5		ns	
T_{17}	Reset Width	1281		ns	41 CLK2 Periods Minimum

NOTES:
1. IAC/INT_0, INT_1, INT_2/INTR, INT_3 can be asynchronous.
2. A float condition occurs when the maximum output current becomes less than I_{LO}. Float delay is not tested, but should be no longer than the valid delay.
3. Case temperatures are "instant on".

A.C. Specification Tables (Continued)

80960MC A.C. Characteristics (20 MHz)

T_{CASE}[3] = −55°C to +125°C, V_{CC} = 5V ±5%

Symbol	Parameter	Min	Max	Units	Test Conditions
T_1	Processor Clock Period (CLK2)	25	125	ns	V_{IN} = 1.5V
T_2	Processor Clock Low Time (CLK2)	6		ns	V_{IL} = 10% Point = 1.2V
T_3	Processor Clock High Time (CLK2)	6		ns	V_{IH} = 90% Point = 0.1V + 0.5 V_{CC}
T_4	Processor Clock Fall Time (CLK2)		10	ns	V_{IN} = 90% Point to 10% Point
T_5	Processor Clock Rise Time (CLK2)		10	ns	V_{IN} = 10% Point to 90% Point
T_6	Output Valid Delay	2	20	ns	C_L = 60 pF (LAD) C_L = 50 pF (Controls)
T_{6H}	HOLDA Output Valid Delay	4	26	ns	C_L = 50 pF
T_7	ALE Width	12		ns	C_L = 50 pF
T_8	ALE Invalid Delay	0	20	ns	C_L = 50 pF[2]
T_9	Output Float Delay	2	20	ns	C_L = 60 pF (LAD) C_L = 50 pF (Controls)[2]
T_{9H}	HOLDA Output Float Delay	4	20	ns	C_L = 50 pF
T_{10}	Input Setup 1	3		ns	(Note 1)
T_{11}	Input Hold	5		ns	(Note 1)
T_{11H}	HOLD Input Hold	4		ns	
T_{12}	Input Setup 2	7		ns	
T_{13}	Setup to ALE Inactive	10		ns	C_L = 60 pF (LAD) C_L = 50 pF (Controls)
T_{14}	Hold after ALE Inactive	8		ns	C_L = 60 pF (LAD) C_L = 50 pF (Controls)
T_{15}	Reset Hold	3		ns	
T_{16}	Reset Setup	5		ns	
T_{17}	Reset Width	1025		ns	41 CLK2 Periods Minimum

NOTES:
1. IAC/INT_0, INT_1, INT_2/INTR, INT_3 can be asynchronous.
2. A float condition occurs when the maximum output current becomes less than I_{LO}. Float delay is not tested, but should be no longer than the valid delay.
3. Case temperatures are "instant on".

A.C. Specification Tables (Continued)

80960MC A.C. Characteristics (25 MHz)

T_{CASE}[3] = −55°C to +125°C, V_{CC} = 5V ±5%

Symbol	Parameter	Min	Max	Units	Test Conditions
T_1	Processor Clock Period (CLK2)	20	125	ns	V_{IN} = 1.5V
T_2	Processor Clock Low Time (CLK2)	5		ns	V_{IL} = 10% Point = 1.2V
T_3	Processor Clock High Time (CLK2)	5		ns	V_{IH} = 90% Point = 0.1V + 0.5 V_{CC}
T_4	Processor Clock Fall Time (CLK2)		10	ns	V_{IN} = 90% Point to 10% Point
T_5	Processor Clock Rise Time (CLK2)		10	ns	V_{IN} = 10% Point to 90% Point
T_6	Output Valid Delay	2	19	ns	C_L = 60 pF (LAD) C_L = 50 pF (Controls)
T_{6H}	HOLDA Output Valid Delay	4	24	ns	C_L = 50 pF
T_7	ALE Width	12		ns	C_L = 50 pF
T_8	ALE Invalid Delay	0	20	ns	C_L = 50 pF[2]
T_9	Output Float Delay	2	19	ns	C_L = 60 pF (LAD) C_L = 50 pF (Controls)[2]
T_{9H}	HOLDA Output Float Delay	4	20	ns	C_L = 50 pF
T_{10}	Input Setup 1	3		ns	(Note 1)
T_{11}	Input Hold	5		ns	(Note 1)
T_{11H}	HOLD Input Hold	4		ns	
T_{12}	Input Setup 2	7		ns	
T_{13}	Setup to ALE Inactive	8		ns	C_L = 60 pF (LAD) C_L = 50 pF (Controls)
T_{14}	Hold after ALE Inactive	8		ns	C_L = 60 pF (LAD) C_L = 50 pF (Controls)
T_{15}	Reset Hold	3		ns	
T_{16}	Reset Setup	5		ns	
T_{17}	Reset Width	820		ns	41 CLK2 Periods Minimum

NOTES:
1. IAC/INT_0, INT_1, INT_2/INTR, INT_3 can be asynchronous.
2. A float condition occurs when the maximum output current becomes less than I_{LO}. Float delay is not tested, but should be no longer than the valid delay.
3. Case temperatures are "instant on".

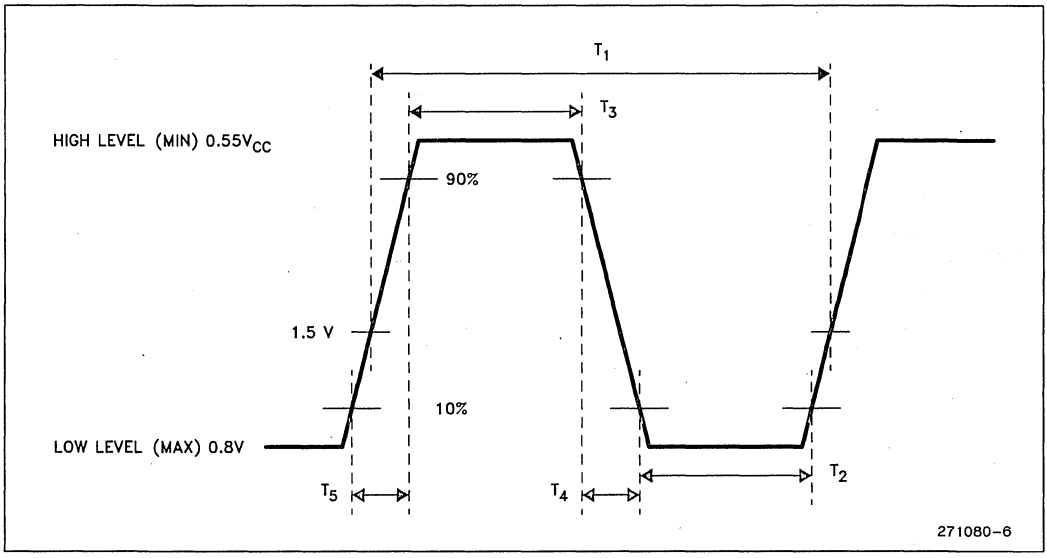


Figure 14. Processor Clock Pulse (CLK2)

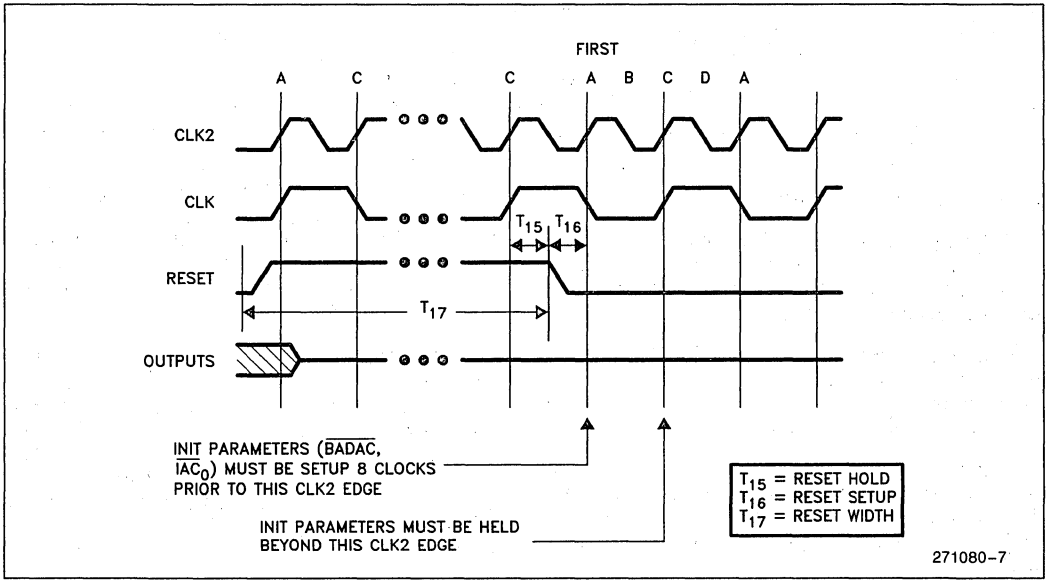


Figure 15. RESET Signal Timing

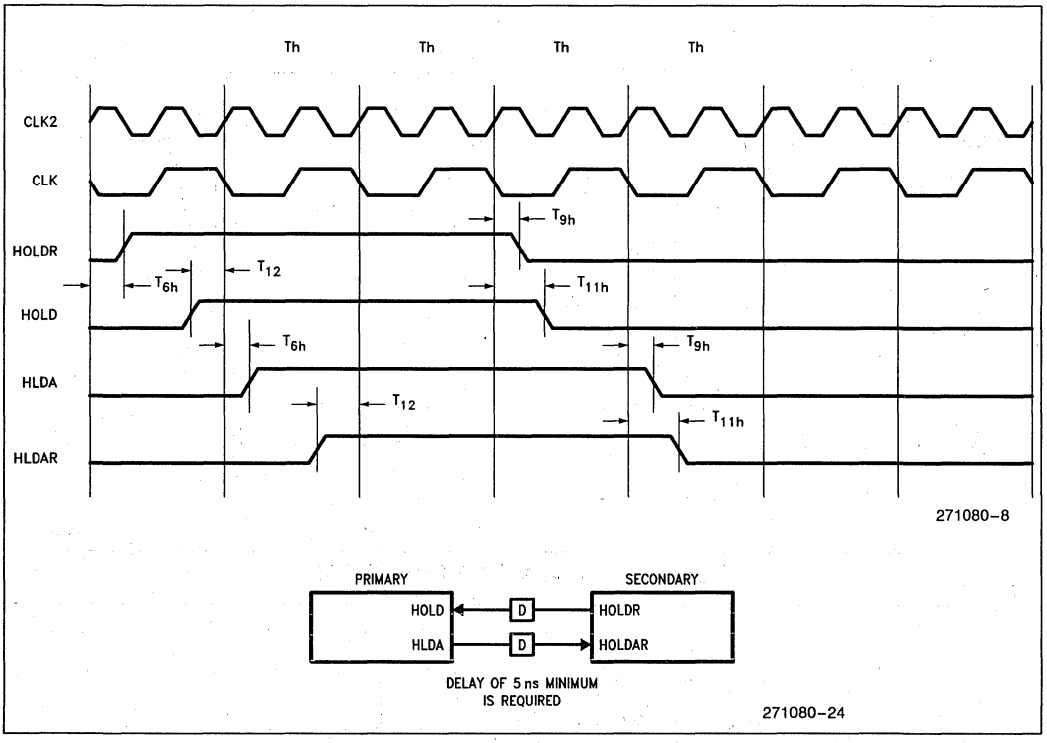


Figure 16. Hold Timing

Design Considerations

Input hold times can be disregarded by the designer whenever the input is removed because a subsequent output from the processor is deasserted (e.g., $\overline{DEN}$ becomes deasserted).

In other words, whenever the processor generates an output that indicates a transition into a subsequent state, the processor must have sampled any inputs for the previous state.

Similarly, whenever the processor generates an output that indicates a transition into a subsequent state, any outputs that are specified to be three stated in this new state are guaranteed to be three stated.

Designing for the ICE-960MC

The 80960MC In-Circuit Emulator assists in debugging 80960MC hardware and software designs. The product consists of a probe module, cable, and control unit. Because of the high operating frequency of 80960MC systems, the probe module connects directly to the 80960MC socket.

When designing an 80960MC hardware system that uses the ICE-960MC to debug the system, several electrical and mechanical characteristics should be considered. These considerations include capacitive loading, drive requirement, power requirement, and physical layout.

The ICE-960MC probe module increases the load capacitance of each line by up to 25 pF. It also adds one standard Schottky TTL load on the CLK2 line, up to one advanced low-power Schottky TTL load for each control signal line, and one advanced low-power Schottky TTL load for each address/data and byte enable line. These loads originate from the probe module and are driven by the 80960MC processor.

To achieve high noise immunity, the ICE-960MC probe is powered by the user's system. The high-speed probe circuitry draws up to 1.1A plus the maximum current (I_{CC}) of the 80960MC processor.

The mechanical considerations are shown in Figure 17, which illustrates the lateral clearance requirements for the ICE-960MC probe as viewed from above the socket of the 80960MC processor.

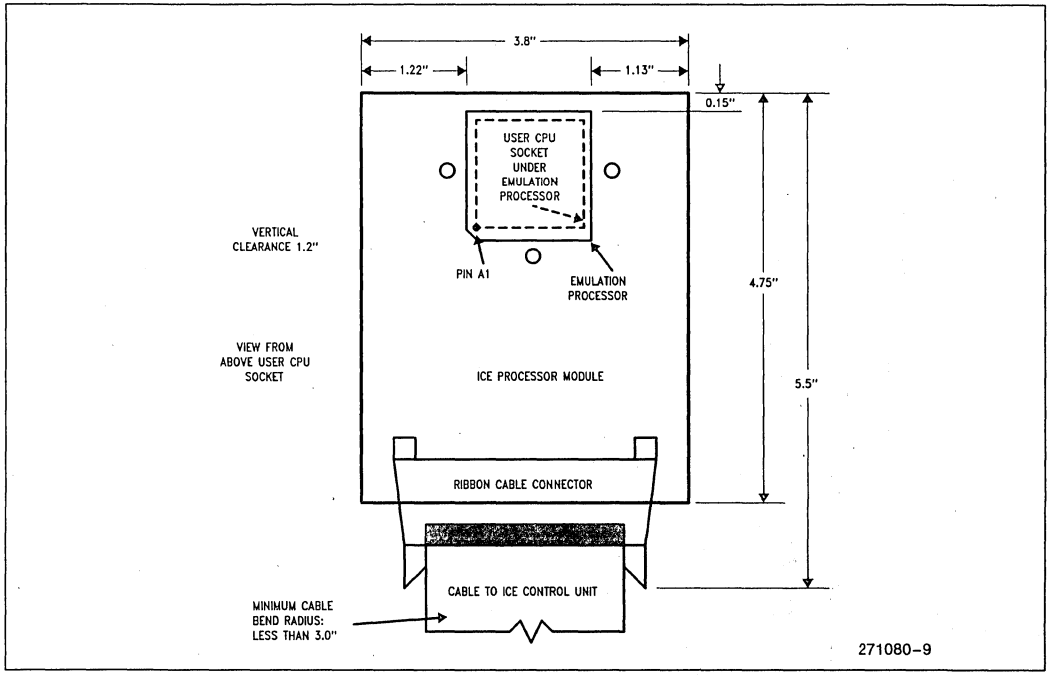


Figure 17. ICE-960MC Lateral Clearance Requirements

3

MECHANICAL DATA

Pin Assignment

The 80960MC is packaged in a 132-lead ceramic pin grid array and a 164-lead ceramic quad flatpack. The 80960MC pin grid array pinout as viewed from the substrate side of the component is shown in Figure 18 and from the pin side in Figure 19. The 80960MC ceramic quad flatpack pinout as viewed from the top of the package is shown in Figure 20.

V_{CC} and GND connections must be made to multiple V_{CC} and GND pins. Each V_{CC} and GND pin must be connected to the appropriate voltage or ground and externally strapped close to the package. Preferably, the circuit board should include power and ground planes for power distribution. Tables 5, 6, 7 and 8 list the function of each pin.

NOTE:
Pins identified as N.C., "No Connect," should never be connected under any circumstances.

Package Dimensions and Mounting

Pins in the pin grid array package are arranged 0.100 inch (2.54mm) center-to-center, in a 14 by 14 matrix, three rows around. (See Figure 21.)

A wide variety of available sockets allow low-insertion or zero-insertion force mountings, and a choice of terminals such as soldertail, surface mount, or wire wrap. Several applicable sockets are shown in Figure 22.

Package Thermal Specification

The 80960MC is specified for operation when its case temperature is within the range of −55°C to +125°C. The PGA case temperature should be measured at the center of the top surface opposite the pins as shown in Figure 23. The ceramic quad flatpack case temperature should be measured at the center of the lid on the top surface of the package.

WAVEFORMS

Figures 24 through 30 show the waveforms for various transactions on the 80960MC's local bus.

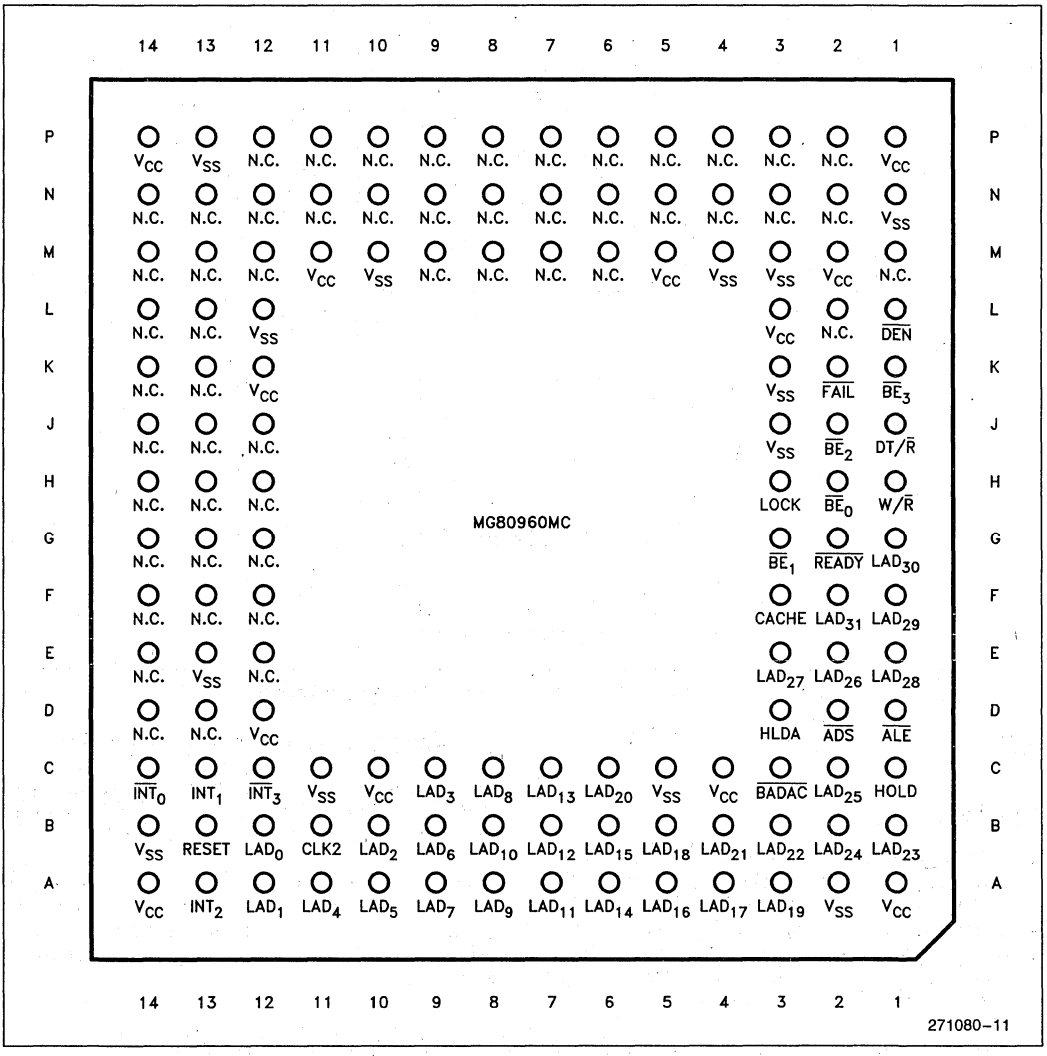


Figure 18. MG80960MC Pinout—View from Top (Pins Facing Down)

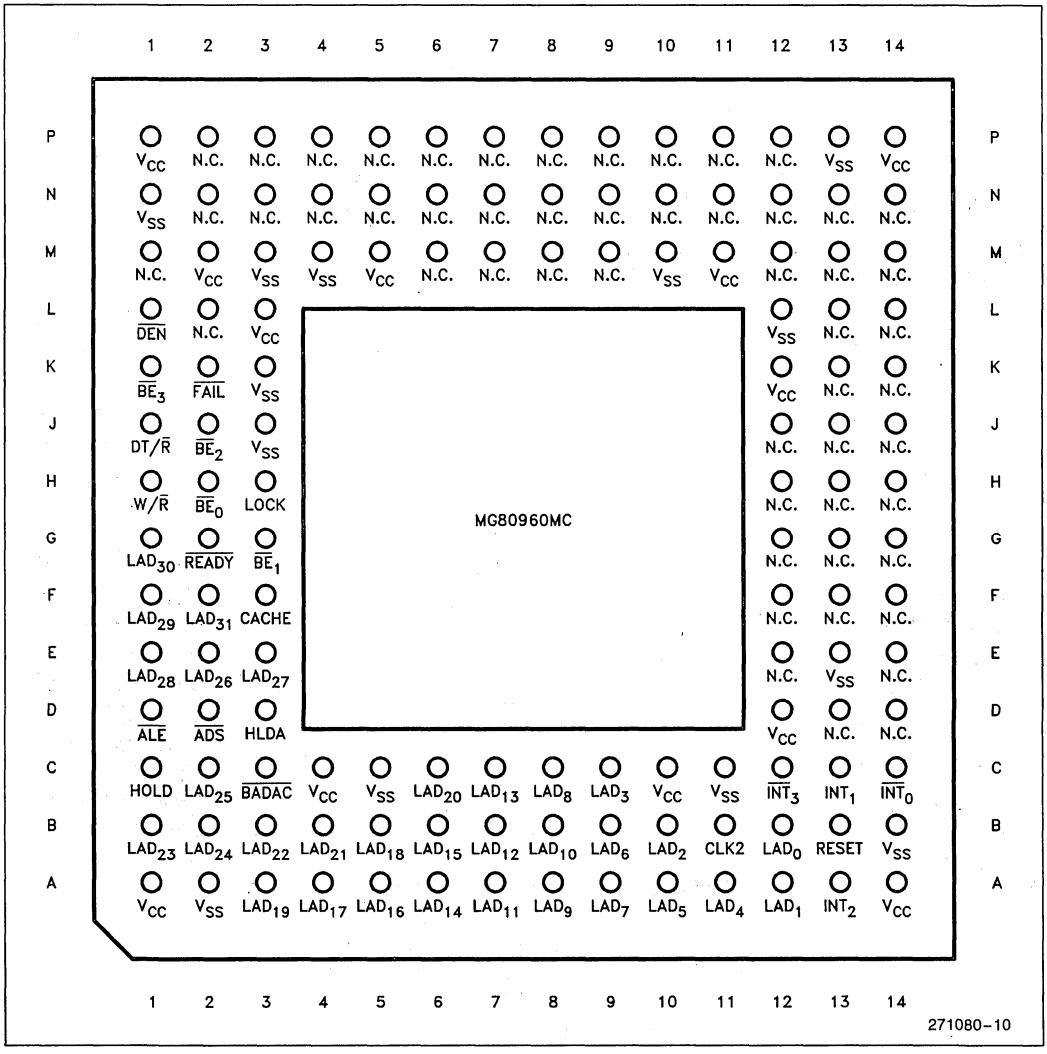
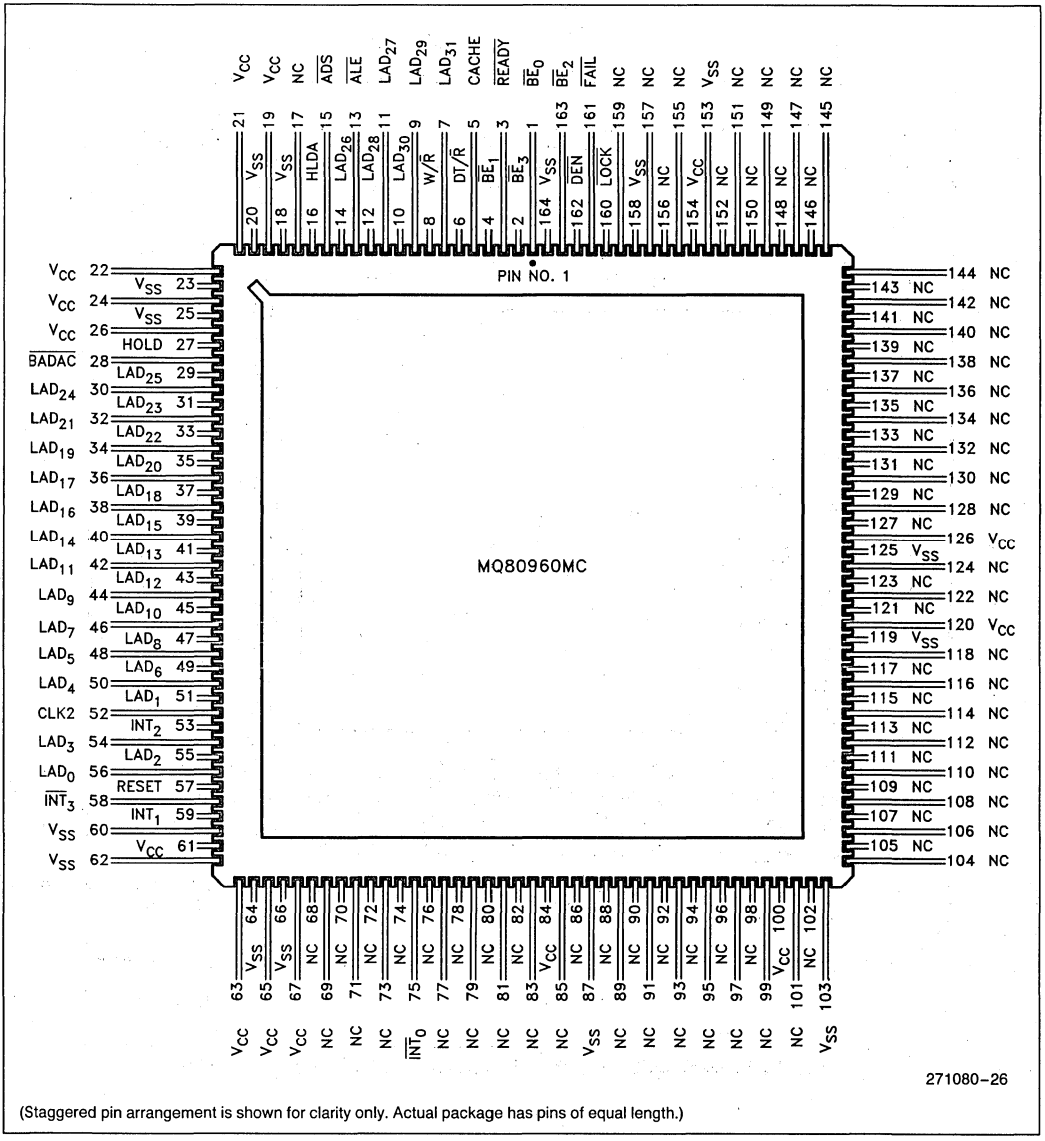


Figure 19. MG80960MC Pinout—View from Bottom (Pins Facing Up)



(Staggered pin arrangement is shown for clarity only. Actual package has pins of equal length.)

Figure 20. MQ80960MC Pinout—View from Top of Package

Table 5. MG80960MC (PGA) Pinout—In Pin Order

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
A1	V_{CC}	C6	LAD_{20}	H1	$W/\overline{R}$	M10	V_{SS}
A2	V_{SS}	C7	LAD_{13}	H2	$\overline{BE_0}$	M11	V_{CC}
A3	LAD_{19}	C8	LAD_8	H3	$\overline{LOCK}$	M12	N.C.
A4	LAD_{17}	C9	LAD_3	H12	N.C	M13	N.C.
A5	LAD_{16}	C10	V_{CC}	H13	N.C.	M14	N.C.
A6	LAD_{14}	C11	V_{SS}	H14	N.C.	N1	V_{SS}
A7	LAD_{11}	C12	$\overline{INT_3}/\overline{INTA}$	J1	$DT/\overline{R}$	N2	N.C.
A8	LAD_9	C13	INT_1	J2	$\overline{BE_2}$	N3	N.C.
A9	LAD_7	C14	$\overline{IAC}/\overline{INT_0}$	J3	V_{SS}	N4	N.C.
A10	LAD_5	D1	$\overline{ALE}$	J12	N.C	N5	N.C.
A11	LAD_4	D2	$\overline{ADS}$	J13	N.C.	N6	N.C.
A12	LAD_1	D3	HLDA/HLDR	J14	N.C.	N7	N.C.
A13	INT_2/INTR	D12	V_{CC}	K1	$\overline{BE_3}$	N8	N.C.
A14	V_{CC}	D13	N.C.	K2	$\overline{FAILURE}$	N9	N.C.
B1	LAD_{23}	D14	N.C.	K3	V_{SS}	N10	N.C.
B2	LAD_{24}	E1	LAD_{28}	K12	V_{CC}	N11	N.C.
B3	LAD_{22}	E2	LAD_{26}	K13	N.C.	N12	N.C.
B4	LAD_{21}	E3	LAD_{27}	K14	N.C.	N13	N.C.
B5	LAD_{18}	E12	N.C.	L1	$\overline{DEN}$	N14	N.C.
B6	LAD_{15}	E13	V_{SS}	L2	N.C.	P1	V_{CC}
B7	LAD_{12}	E14	N.C.	L3	V_{CC}	P2	N.C.
B8	LAD_{10}	F1	LAD_{29}	L12	V_{SS}	P3	N.C.
B9	LAD_6	F2	LAD_{31}	L13	N.C.	P4	N.C.
B10	LAD_2	F3	CACHE	L14	N.C.	P5	N.C.
B11	CLK2	F12	N.C.	M1	N.C.	P6	N.C.
B12	LAD_0	F13	N.C.	M2	V_{CC}	P7	N.C.
B13	RESET	F14	N.C.	M3	V_{SS}	P8	N.C.
B14	V_{SS}	G1	LAD_{30}	M4	V_{SS}	P9	N.C.
C1	HOLD/HLDAR	G2	$\overline{READY}$	M5	V_{CC}	P10	N.C.
C2	LAD_{25}	G3	$\overline{BE_1}$	M6	N.C.	P11	N.C.
C3	$\overline{BADAC}$	G12	N.C.	M7	N.C.	P12	N.C.
C4	V_{CC}	G13	N.C.	M8	N.C.	P13	V_{SS}
C5	V_{SS}	G14	N.C.	M9	N.C.	P14	V_{CC}

NOTE:
Pins identified as N.C. ("No Connect") should never be connected under any circumstances.

Table 6. MG80960MC (PGA) Pinout—In Signal Order

Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin
$\overline{ADS}$	D2	LAD_{15}	B6	N.C.	J14	N.C.	P9
$\overline{ALE}$	D1	LAD_{16}	A5	N.C.	K13	N.C.	P10
$\overline{BADAC}$	C3	LAD_{17}	A4	N.C.	K14	N.C.	P11
$\overline{BE_0}$	H2	LAD_{18}	B5	N.C.	L13	N.C.	P12
$\overline{BE_1}$	G3	LAD_{19}	A3	N.C.	L14	N.C.	L2
$\overline{BE_2}$	J2	LAD_{20}	C6	N.C.	M1	$\overline{READY}$	G2
$\overline{BE_3}$	K1	LAD_{21}	B4	N.C.	M6	RESET	B13
CACHE	F3	LAD_{22}	B3	N.C.	M7	V_{CC}	A1
CLK2	B11	LAD_{23}	B1	N.C.	M8	V_{CC}	A14
$\overline{DEN}$	L1	LAD_{24}	B2	N.C.	M9	V_{CC}	C4
DT/$\overline{R}$	J1	LAD_{25}	C2	N.C.	M12	V_{CC}	C10
$\overline{FAILURE}$	K2	LAD_{26}	E2	N.C.	M13	V_{CC}	D12
HLDA/HOLDR	D3	LAD_{27}	E3	N.C.	M14	V_{CC}	K12
HOLD/HLDAR	C1	LAD_{28}	E1	N.C.	N2	V_{CC}	L3
$\overline{IAC}/\overline{INT_0}$	C14	LAD_{29}	F1	N.C.	N3	V_{CC}	M2
INT_1	C13	LAD_{30}	G1	N.C.	N4	V_{CC}	M5
INT_2/INTR	A13	LAD_{31}	F2	N.C.	N5	V_{CC}	M11
$\overline{INT_3}/\overline{INTA}$	C12	$\overline{LOCK}$	H3	N.C.	N6	V_{CC}	P1
LAD_0	B12	N.C.	D13	N.C.	N7	V_{CC}	P14
LAD_1	A12	N.C.	D14	N.C.	N8	V_{SS}	A2
LAD_2	B10	N.C.	E12	N.C.	N9	V_{SS}	B14
LAD_3	C9	N.C.	E14	N.C.	N10	V_{SS}	C5
LAD_4	A11	N.C.	F12	N.C.	N11	V_{SS}	C11
LAD_5	A10	N.C.	F13	N.C.	N12	V_{SS}	E13
LAD_6	B9	N.C.	F14	N.C.	N13	V_{SS}	J3
LAD_7	A9	N.C.	G12	N.C.	N14	V_{SS}	K3
LAD_8	C8	N.C.	G13	N.C.	P2	V_{SS}	L12
LAD_9	A8	N.C.	G14	N.C.	P3	V_{SS}	M3
LAD_{10}	B8	N.C.	H12	N.C.	P4	V_{SS}	M4
LAD_{11}	A7	N.C.	H13	N.C.	P5	V_{SS}	M10
LAD_{12}	B7	N.C.	H14	N.C.	P6	V_{SS}	N1
LAD_{13}	C7	N.C.	J12	N.C.	P7	V_{SS}	P13
LAD_{14}	A6	N.C.	J13	N.C.	P8	W/$\overline{R}$	H1

NOTE:
Pins identified as N.C. ("No Connect") should never be connected under any circumstances.

Table 7. MQ80960MC (CQP) Pinout—In Pin Order

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	$\overline{BE_0}$	42	LAD_{11}	83	N.C.	124	N.C.
2	$\overline{BE_3}$	43	LAD_{12}	84	V_{CC}	125	V_{SS}
3	$\overline{READY}$	44	LAD_9	85	N.C.	126	V_{CC}
4	$\overline{BE_1}$	45	LAD_{10}	86	N.C.	127	N.C.
5	CACHE	46	LAD_7	87	V_{SS}	128	N.C.
6	DT/$\overline{R}$	47	LAD_8	88	N.C.	129	N.C.
7	LAD_{31}	48	LAD_5	89	N.C.	130	N.C.
8	W/$\overline{R}$	49	LAD_6	90	N.C.	131	N.C.
9	LAD_{29}	50	LAD_4	91	N.C.	132	N.C.
10	LAD_{30}	51	LAD_1	92	N.C.	133	N.C.
11	LAD_{27}	52	CLK_2	93	N.C.	134	N.C.
12	LAD_{28}	53	INT_2	94	N.C.	135	N.C.
13	$\overline{ALE}$	54	LAD_3	95	N.C.	136	N.C.
14	LAD_{26}	55	LAD_2	96	N.C.	137	N.C.
15	$\overline{ADS}$	56	LAD_0	97	N.C.	138	N.C.
16	HLDA	57	RESET	98	N.C.	139	N.C.
17	N.C.	58	$\overline{INT_3}$	99	N.C.	140	N.C.
18	V_{SS}	59	INT_1	100	V_{CC}	141	N.C.
19	V_{CC}	60	V_{SS}	101	N.C.	142	N.C.
20	V_{SS}	61	V_{CC}	102	N.C.	143	N.C.
21	V_{CC}	62	V_{SS}	103	V_{SS}	144	N.C.
22	V_{CC}	63	V_{CC}	104	N.C.	145	N.C.
23	V_{SS}	64	V_{SS}	105	N.C.	146	N.C.
24	V_{CC}	65	V_{CC}	106	N.C.	147	N.C.
25	V_{SS}	66	V_{SS}	107	N.C.	148	N.C.
26	V_{CC}	67	V_{CC}	108	N.C.	149	N.C.
27	HOLD	68	N.C.	109	N.C.	150	N.C.
28	$\overline{BADAC}$	69	N.C.	110	N.C.	151	N.C.
29	LAD_{25}	70	N.C.	111	N.C.	152	N.C.
30	LAD_{24}	71	N.C.	112	N.C.	153	V_{SS}
31	LAD_{23}	72	N.C.	113	N.C.	154	V_{CC}
32	LAD_{21}	73	N.C.	114	N.C.	155	N.C.
33	LAD_{22}	74	N.C.	115	N.C.	156	N.C.
34	LAD_{19}	75	$\overline{INT_0}$	116	N.C.	157	N.C.
35	LAD_{20}	76	N.C.	117	N.C.	158	V_{SS}
36	LAD_{17}	77	N.C.	118	N.C.	159	N.C.
37	LAD_{18}	78	N.C.	119	V_{SS}	160	$\overline{LOCK}$
38	LAD_{16}	79	N.C.	120	V_{CC}	161	$\overline{FAIL}$
39	LAD_{15}	80	N.C.	121	N.C.	162	$\overline{DEN}$
40	LAD_{14}	81	N.C.	122	N.C.	163	$\overline{BE_2}$
41	LAD_{13}	82	N.C.	123	N.C.	164	V_{SS}

NOTE:
Pins identified as N.C. ("No Connect") should never be connected under any circumstances.

Table 8. MQ80960MC (CQP) Pinout—In Signal Order

Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin
$\overline{ADS}$	15	LAD_{23}	31	N.C.	102	N.C.	148
$\overline{ALE}$	13	LAD_{24}	30	N.C.	104	N.C.	149
$\overline{BADAC}$	28	LAD_{25}	29	N.C.	105	N.C.	150
$\overline{BE_0}$	1	LAD_{26}	14	N.C.	106	N.C.	151
$\overline{BE_1}$	4	LAD_{27}	11	N.C.	107	N.C.	152
$\overline{BE_2}$	163	LAD_{28}	12	N.C.	108	N.C.	155
$\overline{BE_3}$	2	LAD_{29}	9	N.C.	109	N.C.	156
CACHE	5	LAD_{30}	10	N.C.	110	N.C.	157
CLK2	52	LAD_{31}	7	N.C.	111	N.C.	159
$\overline{DEN}$	162	$\overline{LOCK}$	160	N.C.	112	$\overline{READY}$	3
$DT/\overline{R}$	6	N.C.	17	N.C.	113	RESET	57
$\overline{FAILURE}$	161	N.C.	68	N.C.	114	V_{CC}	19
HLDA/HOLDR	16	N.C.	69	N.C.	115	V_{CC}	21
HOLD/HLDAR	27	N.C.	70	N.C.	116	V_{CC}	22
$\overline{IAC}/\overline{INT_0}$	75	N.C.	71	N.C.	117	V_{CC}	24
INT_1	59	N.C.	72	N.C.	118	V_{CC}	26
INT_2/INTR	53	N.C.	73	N.C.	121	V_{CC}	61
$\overline{INT_3}/\overline{INTA}$	58	N.C.	74	N.C.	122	V_{CC}	63
LAD_0	56	N.C.	76	N.C.	123	V_{CC}	65
LAD_1	51	N.C.	77	N.C.	124	V_{CC}	67
LAD_2	55	N.C.	78	N.C.	127	V_{CC}	84
LAD_3	54	N.C.	79	N.C.	128	V_{CC}	100
LAD_4	50	N.C.	80	N.C.	129	V_{CC}	120
LAD_5	48	N.C.	81	N.C.	130	V_{CC}	126
LAD_6	49	N.C.	82	N.C.	131	V_{CC}	154
LAD_7	46	N.C.	83	N.C.	132	V_{SS}	18
LAD_8	47	N.C.	85	N.C.	133	V_{SS}	20
LAD_9	44	N.C.	86	N.C.	134	V_{SS}	23
LAD_{10}	45	N.C.	88	N.C.	135	V_{SS}	25
LAD_{11}	42	N.C.	89	N.C.	136	V_{SS}	60
LAD_{12}	43	N.C.	90	N.C.	137	V_{SS}	62
LAD_{13}	41	N.C.	91	N.C.	138	V_{SS}	64
LAD_{14}	40	N.C.	92	N.C.	139	V_{SS}	66
LAD_{15}	39	N.C.	93	N.C.	140	V_{SS}	87
LAD_{16}	38	N.C.	94	N.C.	141	V_{SS}	103
LAD_{17}	36	N.C.	95	N.C.	142	V_{SS}	119
LAD_{18}	37	N.C.	96	N.C.	143	V_{SS}	125
LAD_{19}	34	N.C.	97	N.C.	144	V_{SS}	153
LAD_{20}	35	N.C.	98	N.C.	145	V_{SS}	158
LAD_{21}	32	N.C.	99	N.C.	146	V_{SS}	164
LAD_{22}	33	N.C.	101	N.C.	147	$W/\overline{R}$	8

NOTE:
Pins identified as N.C. ("No Connect") should never be connected under any circumstances.

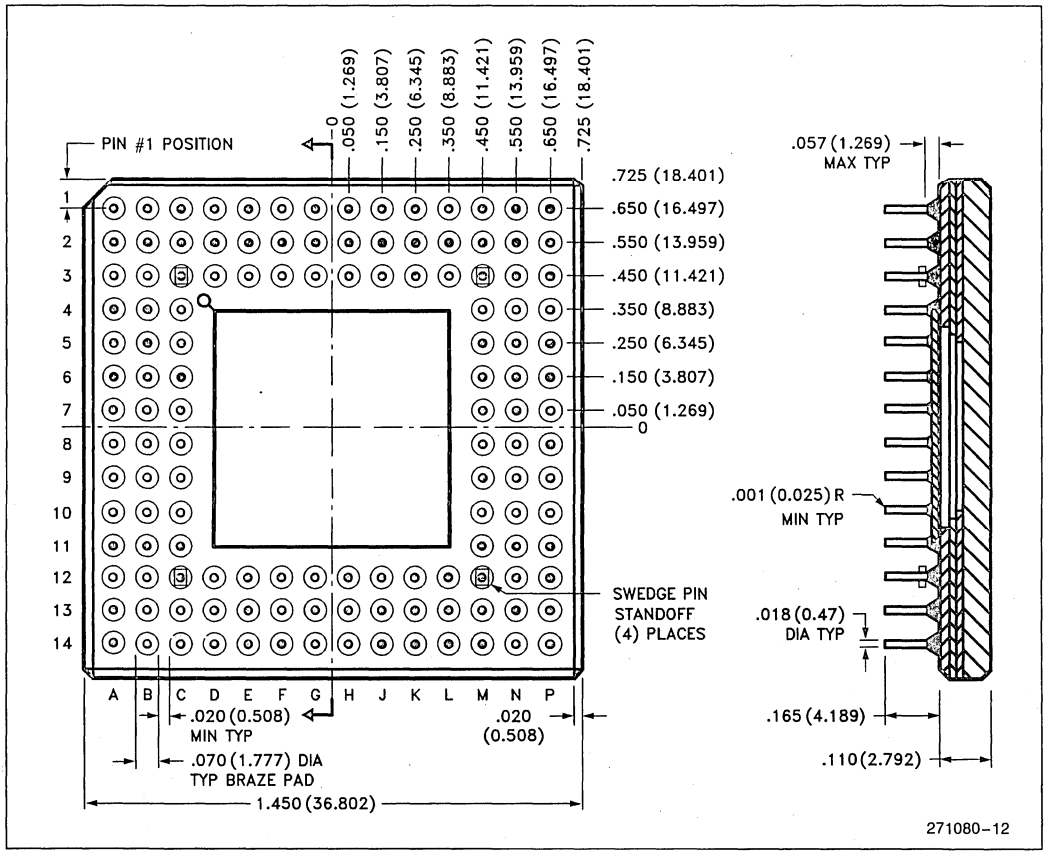


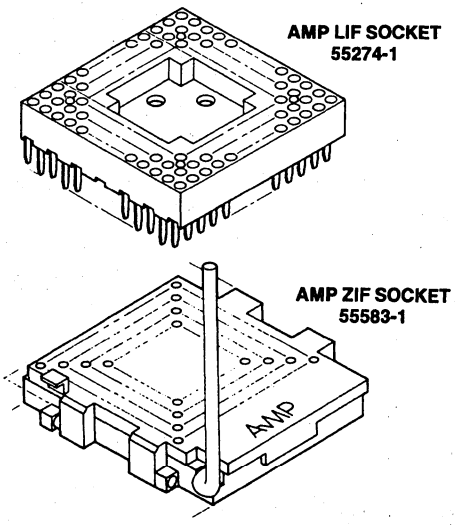
Figure 21. A 132-Lead Pin-Grid Array (PGA) Used to Package the MG80960MC

- Low insertion force (LIF) soldertail 55274-1
- Amp tests indicate 50% reduction in insertion force compared to machined sockets

Other socket options

- Zero insertion force (ZIF) soldertail 55583-1
- Zero insertion force (ZIF) Burn-in version 55573-2

Amp Incorporated
(Harrisburg, PA 17105 U.S.A
Phone 717-564-0100)



271080-13

Cam handle locks in low profile position when MG80960MC is installed (handle UP for open and DOWN for closed positions).

Courtesy Amp Incorporated

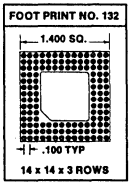
Peel-A-Way* Mylar and Kapton Socket Terminal Carriers

- Low insertion force surface mount CS132-37TG
- Low insertion force soldertail CS132-01TG
- Low insertion force wire-wrap CS132-02TG (two-level) CS132-03TG (thee-level)
- Low insertion force press-fit CS132-05TG

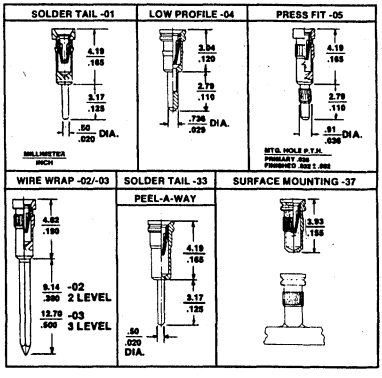
Advanced Interconnections
(5 Division Street)
Warwick, RI 02818 U.S.A.
Phone 401-885-0485)

Peel-A-Way Carrier No. 132:
Kapton Carrier is KS132
Mylar Carrier is MS132

Molded Plastic Body KS132 is shown below:



271080-14



271080-15

Courtesy Advanced Interconnections
(Peel-A-Way Terminal Carriers
U.S. Patent No. 4442938)

*Peel-A-Way is a trademark of Advanced Interconnections.

Figure 22. Several Socket Options for Mounting the MG80960MC

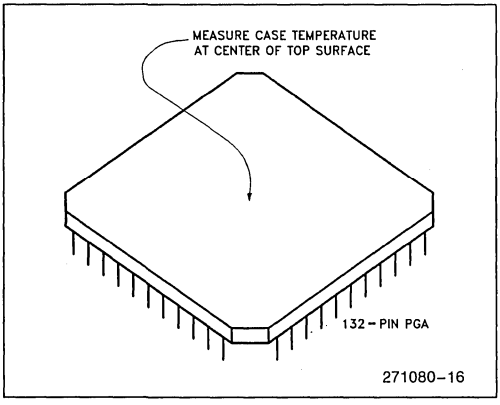


Figure 23. Measuring MG80960MC PGA Case Temperature (T_C)

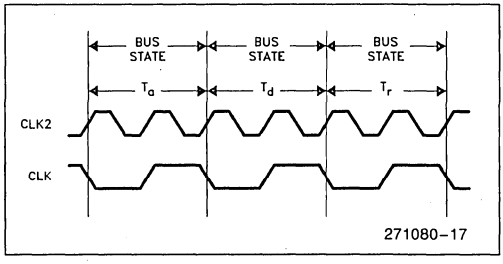


Figure 24. System and Processor Clock Relationship

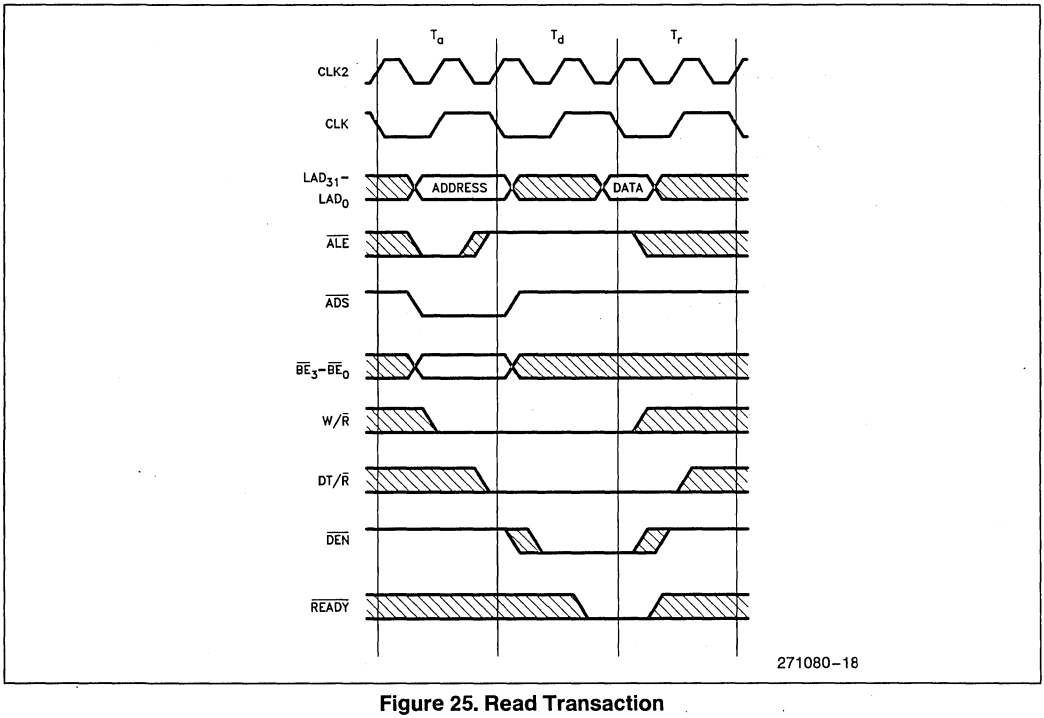


Figure 25. Read Transaction

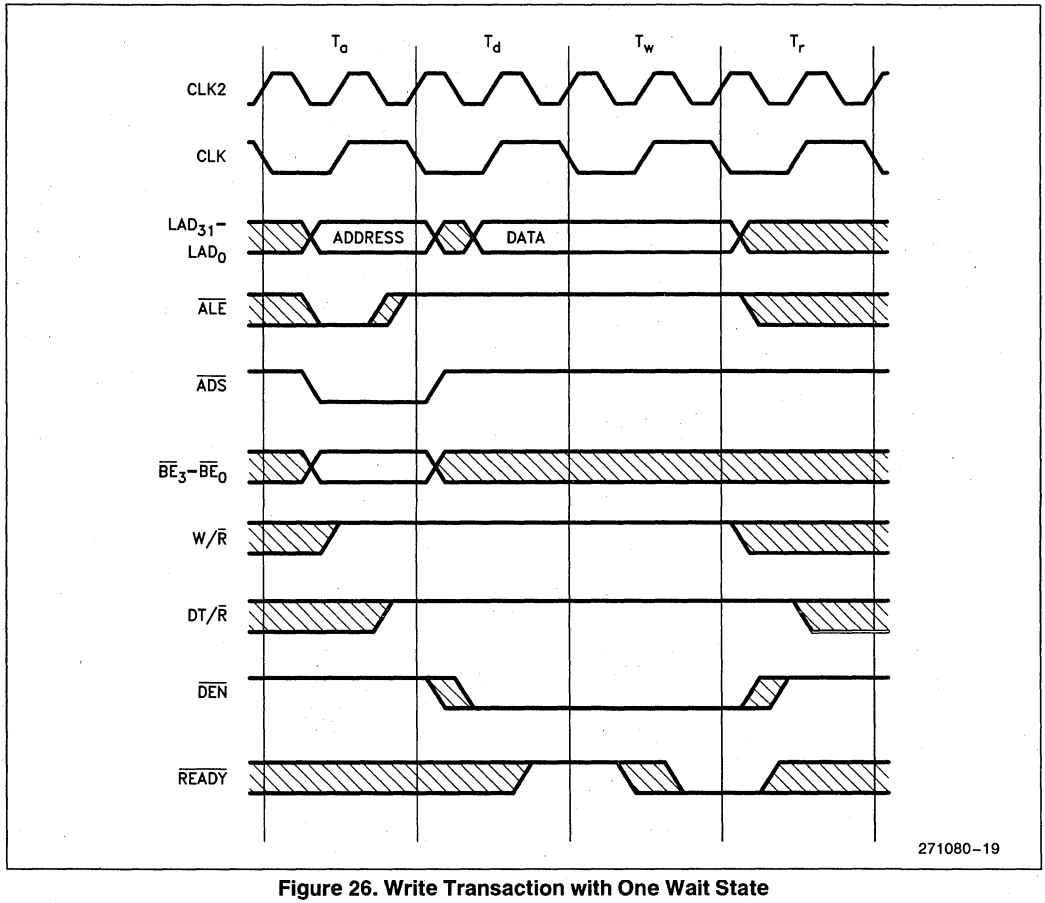


Figure 26. Write Transaction with One Wait State

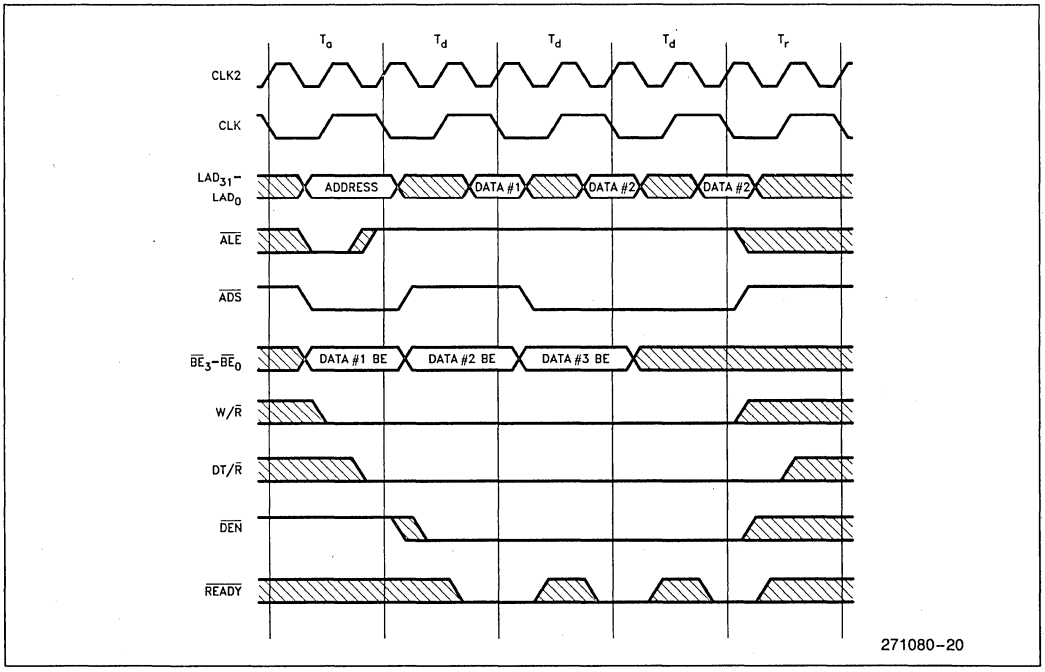


Figure 27. Burst Read Transaction

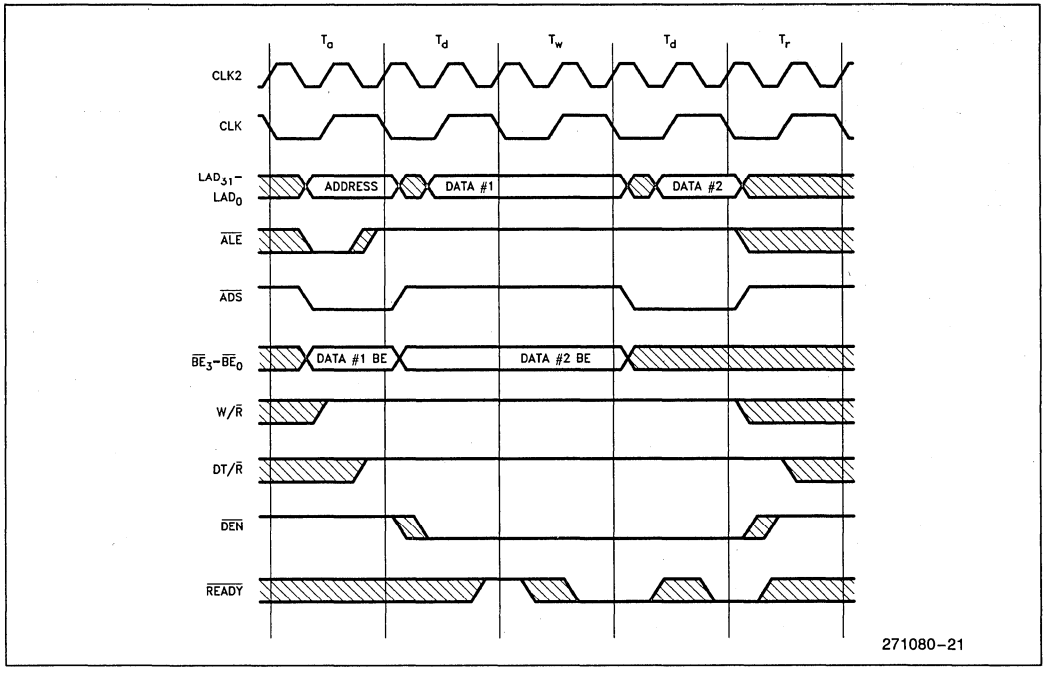


Figure 28. Burst Write Transaction with One Wait State

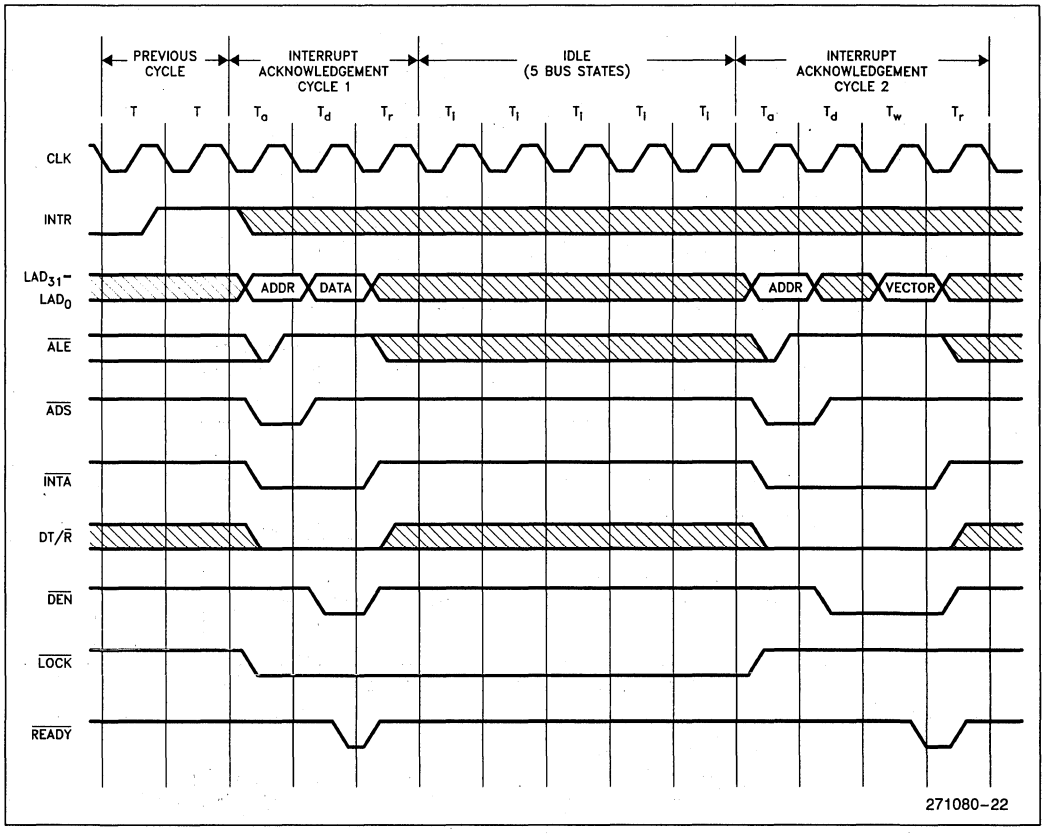


Figure 29. Interrupt Acknowledge Transaction

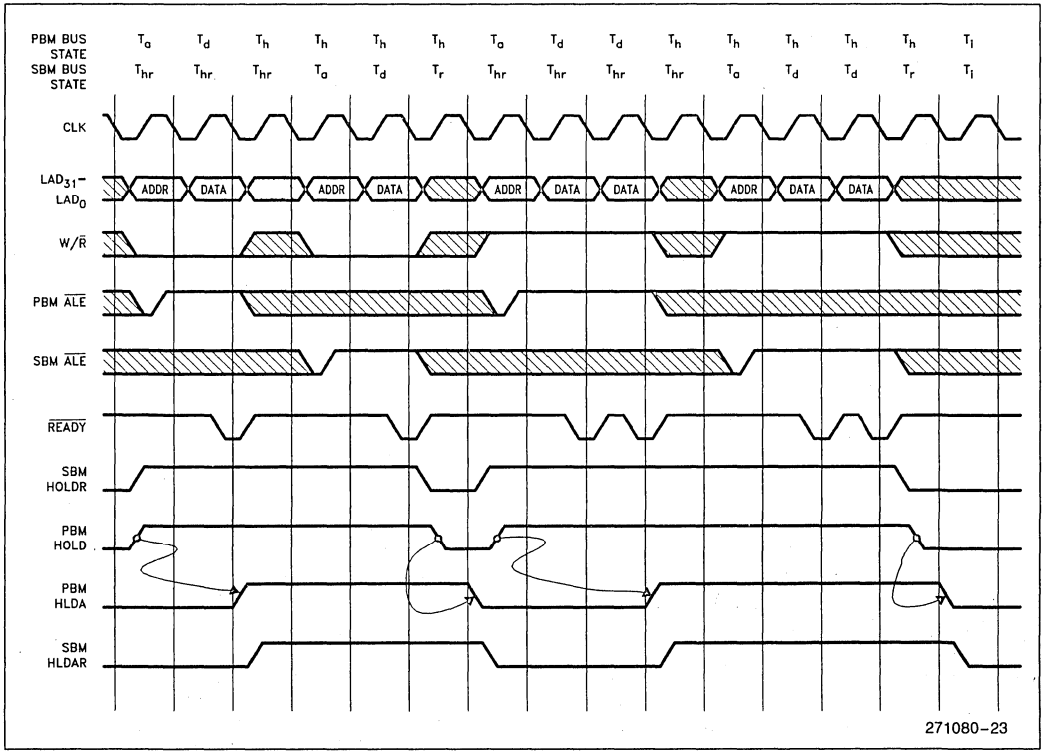


Figure 30. Bus Exchange Transaction (PBM = Primary Bus Master, SBM = Secondary Bus Master)

Revision History

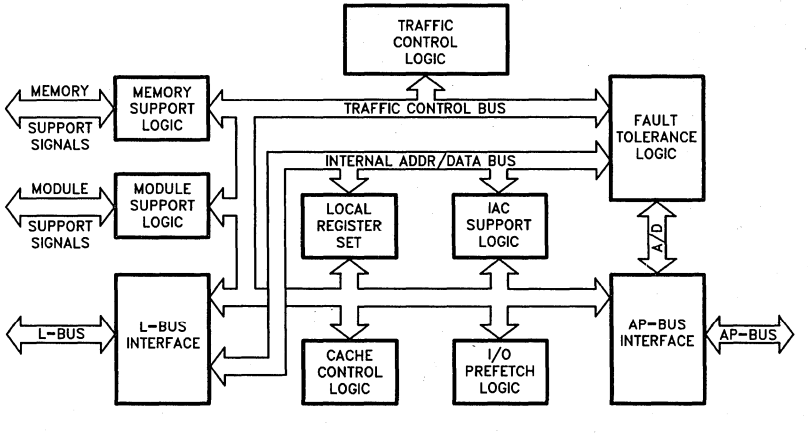
1. 20 MHz timing specifications were added.
2. Pin 158, ceramic quad pack, (see Figure 20) changed from NC (No Connect) to V_{SS}.

M82965
FAULT TOLERANT BUS EXTENSION UNIT

Military

- **Multiprocessor Support**
  - **— Connect up to 32 Processor and Memory Modules in a Single System**
- **Multiple Bus Support with No External Logic**
  - **— Connect up to Four 32-Bit Buses for High-Bandwidth Access to Interleaved Memory**
- **Software-Transparent Fault Tolerance**
  - **— Recover from a Single-Point Failure in a Module or Bus without Affecting Program Execution**
- **Cache Control Support**
  - **— Provides Directory, Coherency Logic, and Control Signals for a Two-Way Set-Associative Cache**
  - **— Single BXU Supports 16 Kbytes**
  - **— Combine up to Four BXUs to Support 64 Kbytes**
- **Message Passing**
  - **— Supports Interagent Communication**
  - **— Redundant Error Reporting Network**
- **Two I/O Prefetch Channels**
  - **— Provides High-Bandwidth, Low Latency Access to Memory or I/O for Sequential Transfers**
- **Memory Module Support**
  - **— Interfaces Discrete Memory Controller and DRAM Array to AP-Bus**
- **Advanced CHMOS III Technology**
- **Advanced Package Technology**
  - **— 132 Lead Ceramic Pin Grid Array**
  - **— 164 Lead Ceramic Quad Flatpack**
- **Military Temperature Range: −55°C to +125°C (T_C)**

The M82965 Bus Extension Unit (BXU) is the key to building multiprocessor and fault-tolerant systems with the 80960MC 32-bit microprocessor. BXUs connect to each other in an expandable matrix that can support up to 32 processor and memory modules in a single, high-performance system. No external interface logic is required. The BXU increases overall system performance by providing hardware support for local caches, I/O prefetch, message passing, and multiprocessor arbitration. Through redundant modules, fault-tolerant systems based on the BXU can sustain a single-point failure and then reconfigure themselves automatically, while application programs continue undisrupted. Truly a VLSI building block, the M82965 BXU supports a wide range of fault tolerance and performance options to meet a diverse set of cost, performance, and reliability needs.



271082-1

Figure 1. M82965 Block Diagram

January 1990
Order Number: 271082-003

FUNCTIONAL OVERVIEW

The M82965 Bus Extension Unit (BXU) is the key component in building multiprocessor and fault-tolerant system designs with the 80960MC 32-bit microprocessor. Its primary function is to connect the Local bus (L-Bus) of a system module to a system-wide bus called the Advanced Processor Bus (AP-Bus), allowing the system to expand incrementally as each new module or AP-Bus is added.

Several important features are provided within the BXU which streamline 80960MC multiprocessor system operation. To increase the available system bus bandwidth, multiple BXUs can be employed within each system module to support up to four AP-Buses. To reduce AP-Bus traffic, BXU components can directly support a two-way set-associative cache. I/O prefetch channels are incorporated within each BXU to reduce the time necessary to transfer large blocks of data from shared system memory or I/O. BXUs support processor-to-processor communication by recognizing, storing, and exchanging Interagent Communication (IAC) messages with other BXUs along the AP-Bus. Requests for access to the AP-Bus are resolved through BXU arbitration logic which ensures that no system modules will suffer from resource starvation.

BXUs support fault tolerant system operation through several mechanisms used to detect, isolate and recover from hardware errors. Paired BXUs monitor each other's operation on a cycle-by-cycle basis through a method called Functional Redundancy Checking (FRC). Errors on the AP-Bus are detected through interlaced parity bits on the address/data and control lines, signal duplication on the transaction control lines, and a bus timer used to monitor the bus for non-response to a request. Recovery mechanisms include the capability to marry FRC modules in a primary-shadow pair (Quad Modular Redundancy), so that if either fails, the surviving spouse can take over operations immediately. Transient errors on the AP-Bus are automatically retried, and in the case of permanent errors, the failed bus is disabled and all memory accesses switched to a backup bus.

MULTIPROCESSOR SUPPORT

A multiprocessor 80960MC system is composed of a set of modules connected to an AP-Bus. Figure 2 shows the three possible types of modules: active, passive, and the combination of both an active and passive module. Active modules contain up to two 80960MC processors, cache or private memory, and a BXU. Passive modules contain a memory array and controller and a BXU. Active/Passive modules contain either processors and global memory, or master and slave I/O devices.

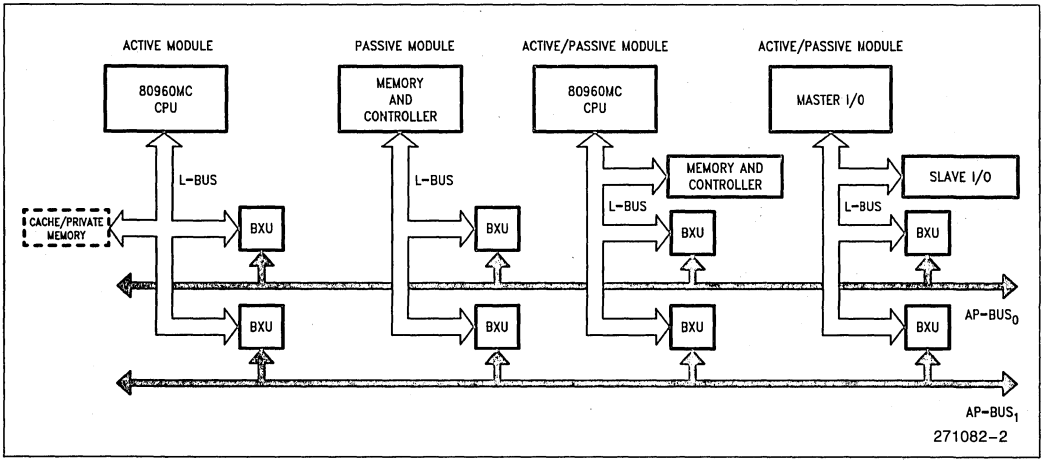


Figure 2. Types of Modules

Local Bus

In a multiprocessor system each module has its own Local Bus (L-Bus), which is typically confined to a single board. The L-Bus is provided to interconnect components within a module. It is a 32-bit multiplexed, synchronous bus with a maximum bandwidth of 43 Mbytes per second at 16 MHz. It has been designed to interface with standard support components using minimal glue logic. The L-Bus uses HOLD/HOLDA for arbitration with bus slaves and LOCK for signaling indivisible operations. A $\overline{\text{READY}}$ signal can be used to lengthen bus transactions.

Local Bus protocol permits both primary and secondary bus masters to coexist on the bus (often a processor and a DMA, or occasionally two processors). A secondary bus master must obtain use of the L-Bus from the bus master through the use of HOLDR/HOLDAR. A BXU is always used as a master in a memory module and is generally used as a slave in a processor module. Fifty BXU pins are dedicated to L-Bus and module support operations (including cache control). The L-Bus control registers are shown in Table 1.

Table 1. L-Bus Control Registers

Register	Description
Physical-ID (Local)	This register contains a unique identifier for a specific BXU on the L-Bus. It corresponds to the AP-Bus Physical-ID register.
Logical-ID (Local)	This register holds the Logical-ID of the BXU. It corresponds to the AP-Bus Logical-ID register.
LBI Control	This is the major control register for BXU functions on the L-Bus. It is used to set the interleaving factor for the cache, determines if the BXU should act as a master on the L-Bus, and indicates whether the BXU is in memory or processor mode.
System Bus ID	This register uniquely identifies the BXU as attached to one of four AP-Buses.
Local-Bus Test	This register allows system diagnostics to check on the type of recognition that was done on the previous L-Bus request.
Match 0	The contents of this register determine which bits in the L-Bus address should be recognized by the BXU. This register provides a base address for a partition of memory recognized by the BXU.
Mask 0	The contents of this register determine if certain bits in the Match 0 register should be ignored (i.e., marked "don't care") during address recognition.
Match 1	Same function as Match Register 0.
Mask 1	Same function as Mask Register 0.
Match 2	Same function as Match Register 0.
Mask 2	Same function as Mask Register 0.
Private Memory Match	Private memory address recognizer.
Private Memory Mask	Private memory mask register.

Advanced Processor Bus

A highly optimized multiprocessing bus called the Advanced Processor Bus (AP-Bus) interconnects 80960MC system modules. The AP-Bus is synchronous, in that all components in the system, including processors and BXUs, are driven by the same clock edge. It is a 32-bit multiplexed bus with a maximum bandwidth of 43 Mbytes per second at 16 MHz.

Transactions over the AP-Bus are encoded into pairs of request and reply packets. A request packet defines the operation, amount of data, and the location (or address) where the transaction will occur. In the case of a write request, the packet will also include data. The reply packet indicates whether or not the action completed successfully, and in the case of read replies, will also include the requested data. Table 2 lists the various types of AP-Bus operations.

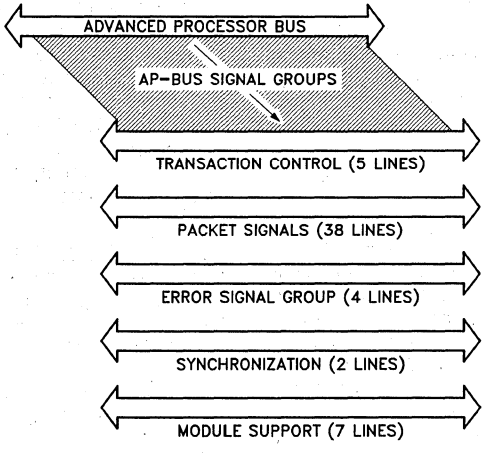
The AP-Bus supports a pipelining feature that allows up to three requests to be pending at any time. Reply packets are returned in the order requested unless deferred, but requests and replies may be intermixed. For example, two requests may be made, followed by a single reply packet, then another request packet, before being completed by two reply packets.

The AP-Bus consists of 47 bi-directional signals, a clock signal, a RESET signal, and five module support signals which are used to interface system modules to the AP-Bus (see Figure 3). The BXU is the only component that attaches to the AP-Bus.

BXUs connect to each other in the form of a matrix to allow orderly growth in the system by the addition of buses or modules. An 80960MC multiprocessing system allows up to 32 modules and four AP-Buses. In practice, the number of modules in a system will be somewhat less in order to meet the AP-Bus's timing and electrical specifications; a practical limit may be 20 to 25 connections to an AP-Bus. Table 3 contains a summary of the functions of the AP-Bus Interface Registers.

Table 2. Types of AP-Bus Operations

Packet Type	Base Action	Specific Operation
Request	Write	Write Word(s)
		RMW Write Word(s)
	Read	Read Word(s)
		RMW Read Word(s)
Reply	Accepted	Read Reply Word(s)
		Acknowledge (Write Reply)
	Refused	Reissue
		Not Acknowledged (NACK)
		Bad Access



Transaction Control
- Arbitration: ARB (3..0)
- Reply Ordering: RPYDEF

Packet Signals
- Specification: SPEC (5..0)
- Address/Data: AD (31..0)

Error Signal Group
- Check Signal: CHK (1..0)
- Bus Error: (1..0)

Synchronization and Initialization Group
- System Clock: CLK2
- Initialization: RESET

Module Support Group
- Identification: INITID
- Module Check: MODCHK
- Bus Output Control: BOUT
- Communication: COM
- Voltage Reference: V_{REF}
- Pop Queue: POPQUE
- Subsystem Busy: SSBUSY

Figure 3. Advanced Processor Bus

Table 3. AP-Bus Interface Registers

Register	Description
Physical ID	This register contains a unique identifier for a specific BXU (or FRC pair of BXUs) on an AP-Bus.
Logical ID	This register holds the logical ID for the BXU. In every case, all BXUs in the same module will share the same logical ID. When two modules are married in a QMR configuration, they will also share the same logical ID.
Component Specifier	The contents of this read-only register are fixed at manufacture and specify the type and stepping of the component.
Arbitration ID	When the BXU needs to issue a request on the AP-Bus, it must actively arbitrate for the bus. The time and order in which a BXU arbitrates is determined by the contents of this write-only register.
Com	This register is used for loading external information, such as the type of board the BXU resides on, into the BXU. The register is useful for both initialization and diagnostics.
AP-Bus Control	This register is the general control and status register for the BXU's AP-Bus interface.
FT1	Most of the BXU fault-tolerant capabilities can be selectively enabled by altering control bits in this register.
Maxtime	The value in this register determines the length of time that BXUs will remain quiescent following the beginning of an error report.
FRC Splitting Control	Writing to this register allows a master/checker pair of BXUs to be split into separately functioning components.
FRC Register	The contents of this register determine of a BXU is part of a master/checker pair and how the component responds if it is part of a QMR module.
Test Detection	Bits in this register enable parity logic and other internal self testing diagnostic features.
AP Match	Bits in this register are compared against the corresponding bits in the AP-Bus address cycle and determine which partition of the address space is recognized by this BXU.
AP Mask	If a bit in this register is cleared, it will cause the corresponding bit position in the Address Match register to be ignored during comparisons.

Memory addressing over the AP-Bus is divided into 16-byte blocks. The location of a bus transaction is defined by a 32-bit address. Each address points to a single byte that is part of a larger 16-byte block. All transactions are performed on a single block or portion of a block, and do not overlap multiple blocks.

Modes of Operation

The BXU operates in either Processor or Memory mode. Processor mode provides support for Active or Active/Passive modules, while Memory mode supports Passive modules. The functions of several BXU signals are dependent on the operating mode of the BXU.

In Processor mode, the BXU supports cache, I/O prefetch and IAC message functions. The BXU can act as either a master or slave on the L-Bus and requests can flow in either direction between the AP-Bus and the L-Bus. The assumption is, however, that most traffic will flow from the L-Bus out onto the AP-Bus. In a processor-only module, there is no need for the BXU to participate in arbitration for the L-Bus, since it will operate only as a slave.

In Memory mode, the BXU always operates as a master on the L-Bus and no requests are ever accepted from the L-Bus. All requests flow from the AP-Bus into the module. In this mode, the BXU supports memory functions and signaling, but does not provide caching or I/O prefetch.

Read-Modify-Write Transactions

Read-Modify-Write (RMW) operations are provided to give BXUs the ability to read and modify a location as a single indivisible action. A RMW-Read operation initiates the indivisible action by asserting the LOCK signal on the L-bus. A RMW-Write operation is used to terminate the action.

When an RMW-Read transaction occurs, the block of memory addressed is marked by the BXU controlling that portion of memory as locked (the lock covers a fixed address space based on address bits 4 and 6). Once locked, any other RMW-Reads to this block will be rejected, but the block remains available for other types of memory operations.

When an RMW-Read is issued, the BXU controlling the affected memory will either respond with data in a normal Read Reply (and set the appropriate lock), or it will respond with a Reissue Reply indicating that the requested block is already locked. If refused, the requesting BXU will wait a short interval and then put the RMW-Read request back into the arbitration process and try again.

RMW-Writes are equivalent to Write Word(s) except that it resets the lock for that memory location. The only valid reply packet is the Ack (Write Reply).

Interagent Communications (IAC) Support

Bus Extension Units and 80960MC processors communicate by sending Interagent Communication (IAC) messages, which are a set of memory-mapped addresses recognized by all BXUs. These messages are used for such system functions as initialization, cache flushing, access to error logs and interrupts. The upper 16 Mbytes of the 80960MC's 4 Gigabyte address range are reserved for IAC communications.

IAC requests fall into two major groups: messages and register requests. Messages are sent between processors to cause a processor to perform a specific action (e.g., start, stop, flush cache, etc.) and are held in the IAC message support registers; Table 4 summarizes the function of these four registers. Register requests are used by software to read and write to BXU registers in order to control the system operation or configuration.

An IAC message always originates on an L-Bus and usually from a processor. From the originator, the request flows to the BXU where it may be handled internally or propagated on to the AP-Bus. If the IAC is sent on to the AP-Bus, the final destination of the IAC (another BXU) must reside on that bus. The IAC will not be propagated onto another L-Bus or AP-Bus. IAC messages can be one to four words long.

Although each L-Bus (processor or memory module) may be connected to as many as four AP-Buses, at any point in time only one bus will be designated as the message bus. All IAC messages will flow over that bus. The BXUs on the message bus are responsible for handling the IAC message traffic on behalf of the processors residing on their L-Bus (an L-Bus may support one or two processors).

AP-Bus 0 normally serves as the message bus. If AP-Bus 0 is not functional, then AP-Bus 1 serves as the message bus, completely transparent to the software. Processors are unaware of which bus is actually acting as the message bus.

I/O Prefetch Support

The BXU offers two I/O prefetch channels to provide high bandwidth, low latency access to memory for sequential transfers. Each channel buffers 32 bytes of data in two 16-byte blocks. As data is requested from the buffers, the BXU automatically prefetches the next data block. The BXU can take

Table 4. IAC Support Registers

Register	Description
Processor 0 Priority	This register holds the priority of the task (process) which Processor 0 on the BXU's L-Bus is currently executing.
Processor 0 Message	This register buffers four words of data from an IAC message for Processor 0.
Processor 1 Priority	This register holds the priority of the task (process) which Processor 1 on the BXU's L-Bus is currently executing.
Processor 1 Message	This register buffers four words of data from an IAC message for Processor 1.

advantage of the three-deep AP-Bus pipeline to quickly fill the buffers if it ever gets behind because of momentary surges in AP-Bus traffic. In this way, the prefetch logic acts to provide stable, bounded response times, even in large multiprocessor configurations.

Because the normal operation of the BXU hides the latency of write requests by replying immediately on the L-Bus, the prefetch unit operates only for read requests. On a read request from the L-Bus, the prefetch logic returns the amount of data requested. Any processor or intelligent device used with the BXU must guarantee that it will split all memory requests that cross 16-byte boundaries into two requests.

Cache Support

The main function of a cache is to provide local high speed storage for frequently accessed memory locations. Storing the information locally, the cache intercepts memory references and handles them directly without transferring the request to the AP-Bus. This action results in lower traffic on the AP-Bus and decreased latency on the L-Bus, leading to improved performance for a processor on the L-Bus. It also increases potential system performance in a multiprocessor system by reducing each processor's demand for AP-Bus bandwidth, thereby allowing more processors in a system.

The BXU provides cache directory, coherency logic, and control signals, while external SRAM is used for data storage. A CACHE signal output from the 80960MC processor indicates to the BXU whether a request is cacheable. The operation of the BXU cache is not dependent on the size of the data transfer and therefore can support partial writes. Both data and instructions can be contained within the local cache.

The BXU supports a two-way, set associative cache with 64 sets. The (read address) tag field is 20 bits long and consists of LAD lines 31–12. There are eight bits that indicate if a line is valid (a line is 16 bytes). The control bits in the cache control registers can be used to mask some of these bits to change cache configurations. All entries in the directory can be invalidated by sending an INVALIDATE CACHE Command to each BXU in the module. Figure 4 shows one example of a BXU cache directory and its relation to L-Bus addresses.

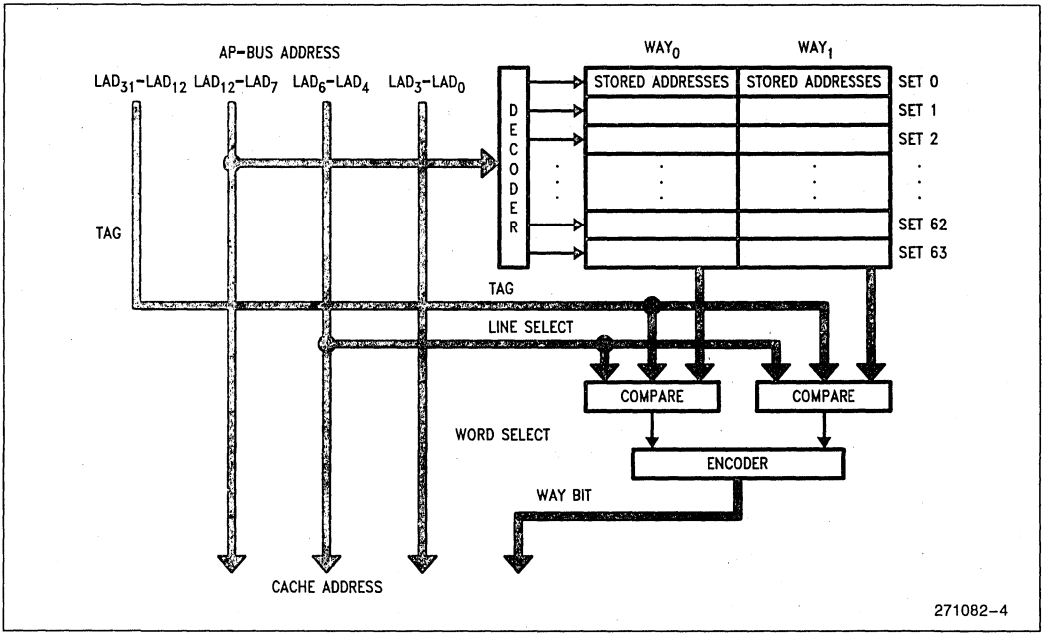


Figure 4. Example of a Cache Directory Array

A single BXU supports 16 Kbytes of cache. When a processor module uses multiple BXUs (and therefore multiple buses), the BXUs cooperate to provide a larger directory and addressing for a larger cache. The best way to view this larger directory is to think of it as having an increased number of sets. Thus a cache managed by two BXUs will have a directory consisting of 128 sets instead of 64. The maximum size cache is 64 Kbytes (four BXUs supporting four AP-Buses per processor module).

The cache is managed using a write-through policy that guarantees that the shared system memory will always have the most recent copy of all data; BXU caches never contain the only copy of revised data. Any time a processor updates a cache entry, it always causes a write request on the AP-Bus, so that there are never any hidden updates. In addition, all BXUs monitor AP-Bus traffic to detect if an update is being made to a location which they are storing in their own cache. If so, that line in the cache directory is marked invalid. This procedure guarantees that a BXU cache will always return correct data even when a system uses multiple caches, when multiple processors treat a single data item differently (some caching, some not), or when two processors are used on a single L-Bus.

An example of an SRAM control design using a single BXU is shown in Figure 5. The BXU supplies six memory control signals to interface the directory and control logic with an external cache composed of static RAM: Cache Read ($\overline{CR}$), Cache Write ($\overline{CW}$), Way0 ($\overline{WY0}$), Way1 ($\overline{WY1}$), Word0 ($\overline{WD0}$), and Word1 ($\overline{WD1}$). SRAM control also requires use of the L-Bus byte enable ($\overline{BE3}$–$\overline{BE0}$) signals and certain address lines. To simplify latching the byte enable signals, the BXU asserts $\overline{READY}$ on all address and recovery cycles as well as when it is transferring data.

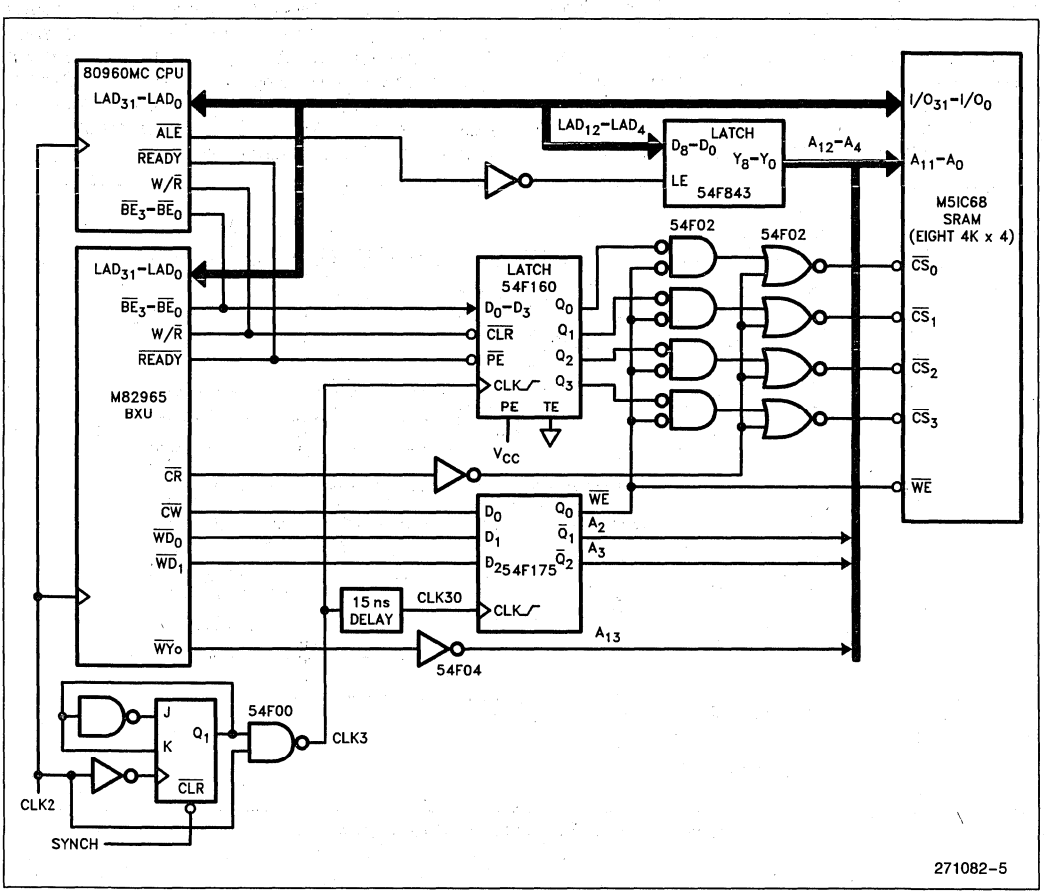


Figure 5. Sample Cache SRAM Control Design Using a BXU

The tight timing specifications of SRAMs require a small amount of external logic to interface a static RAM cache to a BXU. Since all BXU cache signals have a relatively wide clock to data valid specification (T_{cd}), external flip-flops are used to achieve tighter resolution of the Cache Write and Word edges. The address bits are latched using ALE from the processor. Way0 selects between the two "ways" in the cache directory, and Way1 selects between the cache and private memory (if present on the L-Bus).

In order to ensure that the cache is filled properly, the byte enable latch is cleared on read requests. If the processor made a read request for two bytes that missed the cache, the BXU would first write the entire 16-byte block, then return the requested information to the processor. If the byte enable latches weren't set, then the write into the cache wouldn't work correctly because not all byte enables would be asserted. Byte enable information does not need to be held on reads because data is always returned in full words and the processor selects the portion of the word that it needs internally. Signal timings are shown in Figures 6–10.

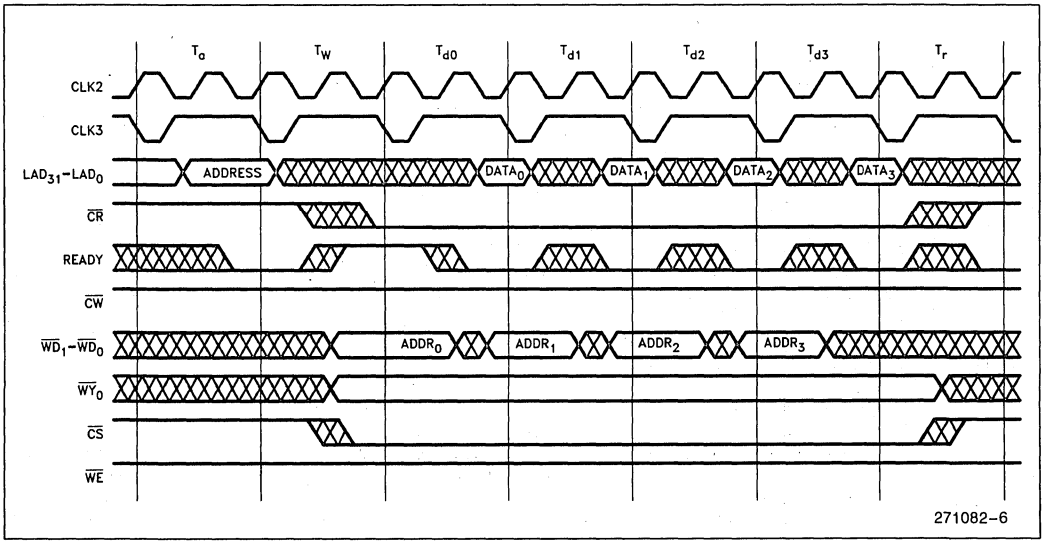


Figure 6. Cache Read Signal Timing for 35 ns SRAMs

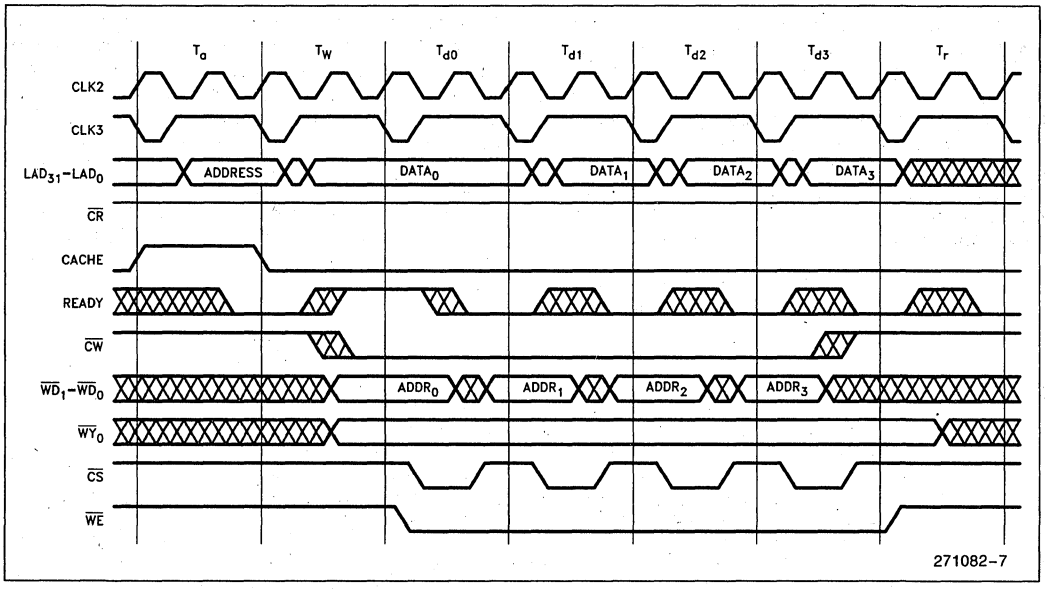


Figure 7. Cache Write Signal Timing for 35 ns SRAMs

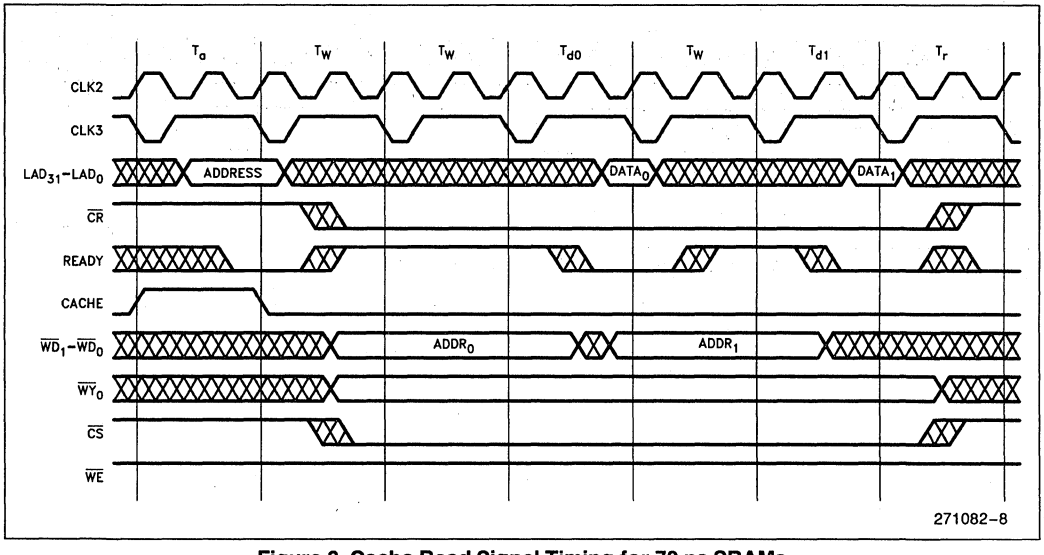


Figure 8. Cache Read Signal Timing for 70 ns SRAMs

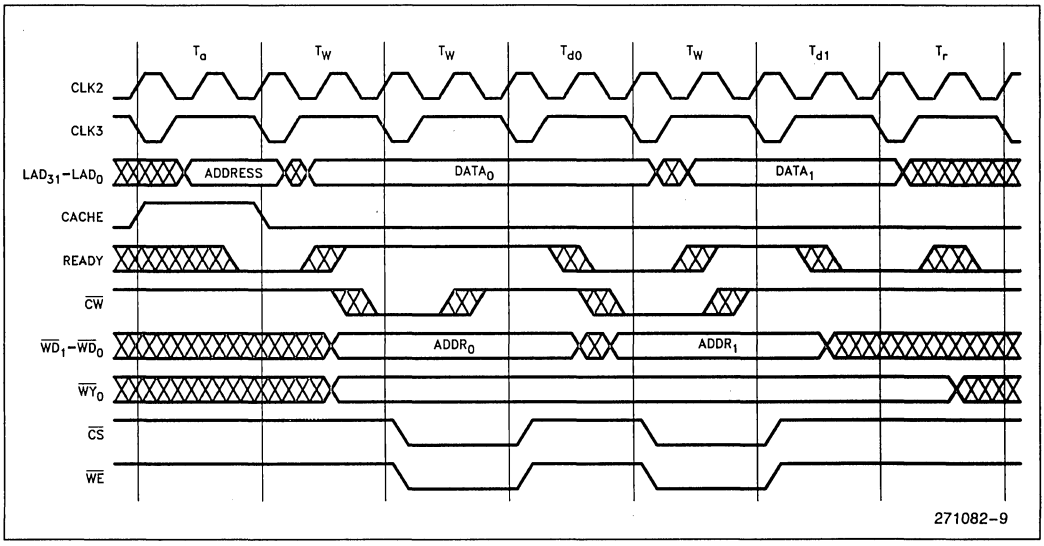


Figure 9. Cache Write Signal Timing for 70 ns SRAMs

3

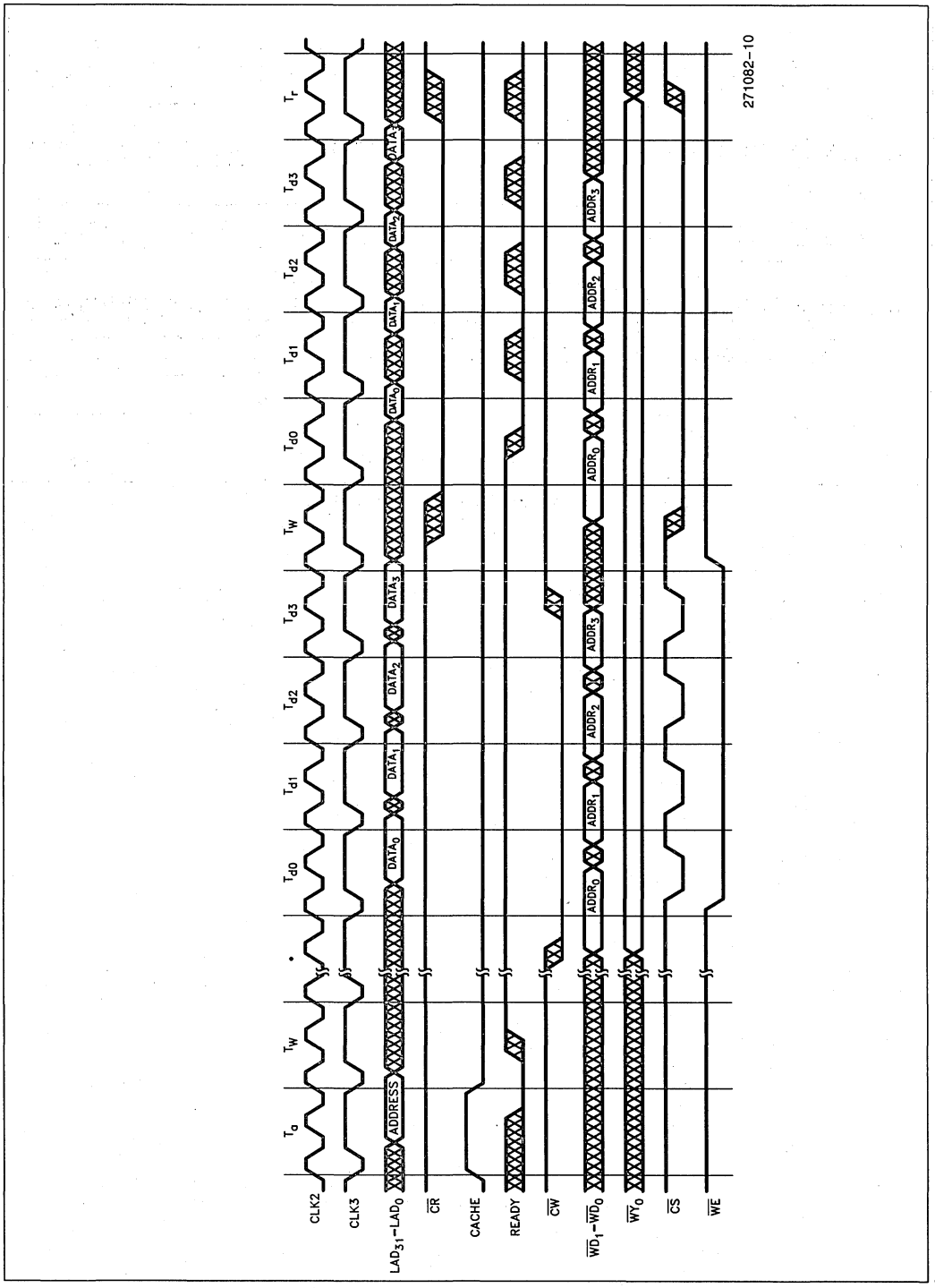


Figure 10. Cache Signal Timing for a 4-Word Read with a Cache Fill for 35 ns SRAMs

The BXU has four memory address recognizers for the L-Bus plus an additional recognizer for initialization RAM. Three of the memory address recognizers (Mask2–0 and Match2–0) map to shared system memory, while the fourth address recognizer maps requests to SRAM on the local bus, called private memory. The INIT-RAM recognizer serves two functions: it enables bootstrap software to use the SRAM cache as a scratch pad during system initialization, and it provides the means for executing a memory test on the SRAM cache. The private memory recognizer allows SRAM to be used on the local bus as normal memory in addition to a cache. Private memory is not accessable by other modules on the AP-Bus.

Memory Module Support

When operating in Memory mode, the BXU is a Local Bus master and only handles requests inbound from the AP-Bus. The cache control logic is disabled since it is unnecessary in a memory module.

A read request received by an idle BXU will be seen on the L-Bus 1.5 clock cycles after it was received on the AP-Bus. BXUs offer two reply speed options for inbound Read requests. The high-performance option, called the "fast reply" mode, allows data to flow onto the AP-Bus with only a half-cycle delay through the BXU. This option requires the L-Bus memory controller to be able to supply data on every clock cycle. In the "slow reply" mode, the BXU buffers the entire AP-Bus reply packet before sending it onto the AP-Bus. This option permits the use of slower, less costly memory.

Write requests are fully buffered before being passed to the L-Bus. Once the BXU has received an error-free packet, it initiates the L-Bus transaction. When the last data word has been accepted on the L-Bus, the BXU generates a reply on the AP-Bus.

In memory mode, the BXU provides two or four Ready-Modify-Write locks with timeouts. Four locks are available if the module is not interleaved with other modules, two locks if it is interleaved. When interleaving occurs, address bit 4 is used as part of the address recognition for the module, which thereby restricts a module to use either locks 0 and 2, or 1 and 3. This approach ensures that if a bus switch occurs, the locks that may have been allocated on the failed bus will not overlap with locks that are currently allocated on the surviving bus (since all traffic is rerouted to the surviving bus).

FAULT TOLERANCE

Three basic tenets form the basis for the implementation of 80960MC fault tolerant systems. First, fault tolerant functions are achieved through the replication of VLSI components. Second, the system is partitioned into a set of confinement areas which form the basis of error detection and recovery. Third, only bus-oriented communication paths are used to provide system communication.

The BXU is unique in that it provides all the functions necessary to detect, isolate, and recover from a failure in any single system module or AP-Bus. Unlike many other fault tolerant system designs, 80960MC systems do not rely on voter components for fault detection, thereby eliminating one potential source of single-point failures. Although the BXU registers must be initialized by software, all the fault tolerant mechanisms are built into the hardware, and correct fault recovery of a system built using the BXU does not depend on software intervention.

The purpose of a confinement area is to inhibit damage from error propagation and to isolate the faulty area for subsequent recovery and repair. A confinement area is defined as a unit (system module or AP-Bus) that has a limited number of tightly controlled interfaces. Figure 11 shows the confinement areas within a small system. Detection mechanisms exist at every interface to ensure that no inconsistent data can leave the confinement area and corrupt other confinement areas. When a fault occurs in the system, it is immediately isolated to a confinement area. The fault is known to be in that confinement area, and all other confinement areas are known to be fault-free. All intermodule communication in an 80960MC system occurs over buses. There are no point-to-point or daisy-chained signals.

This arrangement makes modular growth and online repair possible since no signal definition is dependent on the number of resources in the system. The presence or absence of any module cannot prevent communication between any other modules. The AP-Bus provides a uniform communications matrix that allows multiprocessor and fault-tolerant systems to expand modularly.

In 80960MC systems, there are three distinct steps in responding to an error. First, the error is detected and isolated to a confinement area. Next, the error is reported to all the modules in the system. This action prevents the incorrect data from propagating into another confinement area and provides all the modules with the information required to perform recovery. Finally, the faulty confinement area is isolated from the system. Recovery occurs through the application of redundant resources available in the system. Table 5 describes the fault-tolerant control registers.

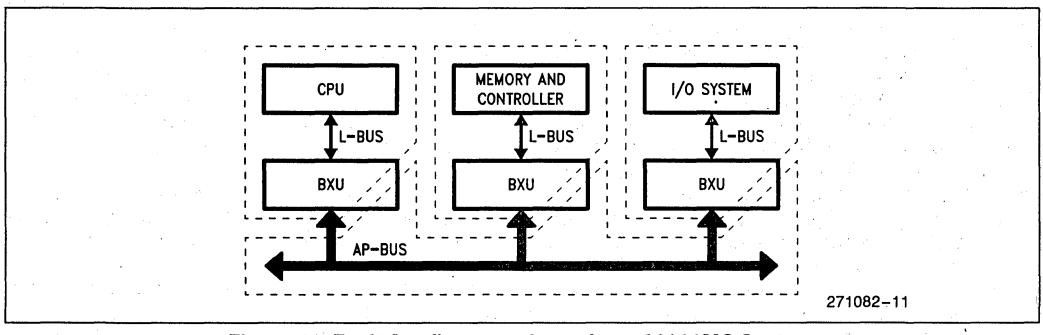


Figure 11. Fault Confinement Areas in an 80960MC System

Table 5. Fault Tolerance Support Registers and Commands

Register	Description
Test Type	The Test Report command instructs the BXU to test the error reporting network. The type of error report generated is determined by the content of this register.
Spouse ID	In a QMR module, this register holds the module ID of the FRC module to which this module is married.
QMR	The contents of this register determine if a module is part of a QMR pair, and if it should function as the primary or shadow in the pair.
Module Error ID	Identifies the BXU as part of a specific module confinement area.
Bus Error ID	Determines the Bus ID contents in an error report.
Error Log	Records the type of the most recent error report received and the number of errors that have occurred since the last Terminate Permanant Error Window command.
Error Record	Holds the contents of the previous error report.
FT2	Holds additional fault-tolerant control parameters.
Test Report Command	The Test Report command instructs the BXU to test the error reporting network. The type of error report generated is determined by the contents of the Test Type Register.
Primary Catastrophe Command	A write to this register causes a Primary Catastrophe error report, usually indicating a primary module power failure.
Shadow Catastrophe Command	A write to this register causes a Shadow Catastrophe error report, usually indicating a shadow module power failure.
Terminate Permanent Error Window Command	A write to this register closes the permanent error window, so that a reoccurance of a previous error is not recorded as permanent.
Attach Bus Command	A write to this register causes the identified bus to be attached to the system and become active.
Detach Bus Command	A write to this register causes the identified bus to be detached from the system and become inactive.
Sync Refresh Command	A write to this register causes BXUs in memory mode to assert their ForceRef pin and enables AP-Bus address matching.

Functional Redundancy Checking

BXU components can be paired together to compare their outputs to ensure that they agree. This detection mechanism is called Functional Redundancy Checking (FRC) because identical components are used to check operations.

At initialization time, one component in the BXU pair is selected to be the "Master", while the other is designated the "Checker". The Master BXU is responsible for carrying out the normal operation of the system and behaves as it would if it were operating in a non-fault tolerant system. The Checker BXU, in contrast, disables its AP-Bus outputs and instead monitors the AP-Bus pins of the Master (see Figure 12). The Checker BXU is responsible for duplicating the operation of the Master and using its internal comparison circuitry to detect any inconsistency between its result and the output of the Master.

The Master and Checker BXUs run in lock step, comparing operations cycle-by-cycle. If at any point the Master or Checker disagree, an FRC error will be signaled and an error reporting cycle will begin.

When using the FRC mechanism, the BXU pins comprising the electrical connection to the AP-Bus must be connected together. A BXU provides FRC coverage on the AD, SPEC, BOUT and MODCHK pins.

Failures in the Checker's AP-Bus drivers can be detected by reversing the role of the Master and Checker BXU. When Master/Checker Toggling is enabled, the Roles of the Master and Checker are switched after each bus cycle.

Parity, Duplication and Timeouts

In order to prevent incorrect AP-Bus operation for passing corrupted data to the BXU (and onto the Local Bus), the BXU uses parity, signal duplication, and bus timeouts to check for errors. Specifically, the AP-Bus has interlaced parity bits covering the AD and SPEC signals, signal duplication is used on both arbitration and RPYDEF, and a bus timer is set to monitor the bus for non-response to a request.

The BXU calculates two separate parity bits across alternate AD and SPEC signals, which are indicated by the CHK0 and CHK1 pins. CHK0 is even parity across the even AD and SPEC pins, and CHK1 is even parity across the odd pins. Since the arbitration and RPYDEF lines are driven independently by multiple bus agents (BXUs), parity cannot be used for error detection, rather the detection of errors is done by duplicating each set of lines, one set for Masters, the other set for Checkers. Consequently, each BXU connects to only one arbitration network. If there is a disagreement between the two sets of signals on

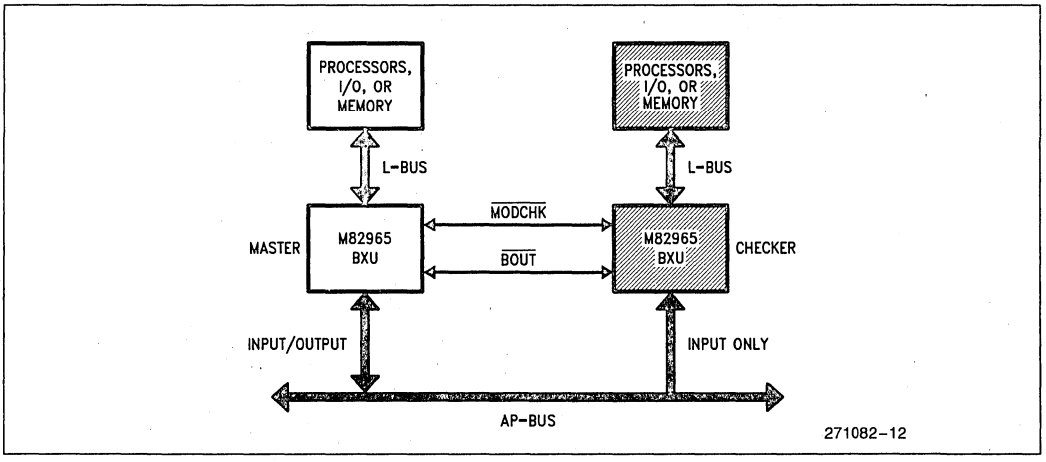


Figure 12. Functional Redundancy Checking (FRC)

the AP-Bus, it will be detected through an FRC disagreement. The BXU uses a timer to determine if no response has been received and too long a period has elapsed since the bus request was made. During normal operation the timer is active whenever the bus pipeline is not empty. The timer is reset on every bus reply or deferral. If the BXU was the source of the requests and a timeout occurs, it signals a Bad Access Reply on the AP-Bus. The timer is nominally 64 clocks.

Error Reporting

The error reporting network is the backbone of fault isolation and recovery. When an error is detected, the BXU detecting the error reports its type and location to all other nodes in the system. The error reporting network is designed so that, independent of an error in the system, each node not only receives an error report, but is guaranteed to receive the same error report. Each BXU in the system uniformly logs each error report, and is able to use this information to proceed independently with the appropriate recovery procedure.

The BXU has two serial Error Reporting Lines associated with each bus interface (BERLs for the AP-Bus and LERLs for the Local Bus). An indentical serial error report is sent over each pair of lines associated with each bus.

An AP-Bus error reporting cycle consists of five phases: Reporting, Partner Communications, Transient Waiting Period, Retry, and the Permanent Error Window (see Figure 13). The reporting phase lasts 256 cycles from the beginning of the first report received on the BXU's error reporting lines. The BXU becomes quiescent as soon as it detects the start bit of an error report and remains quiescent through the Transient Waiting Period.

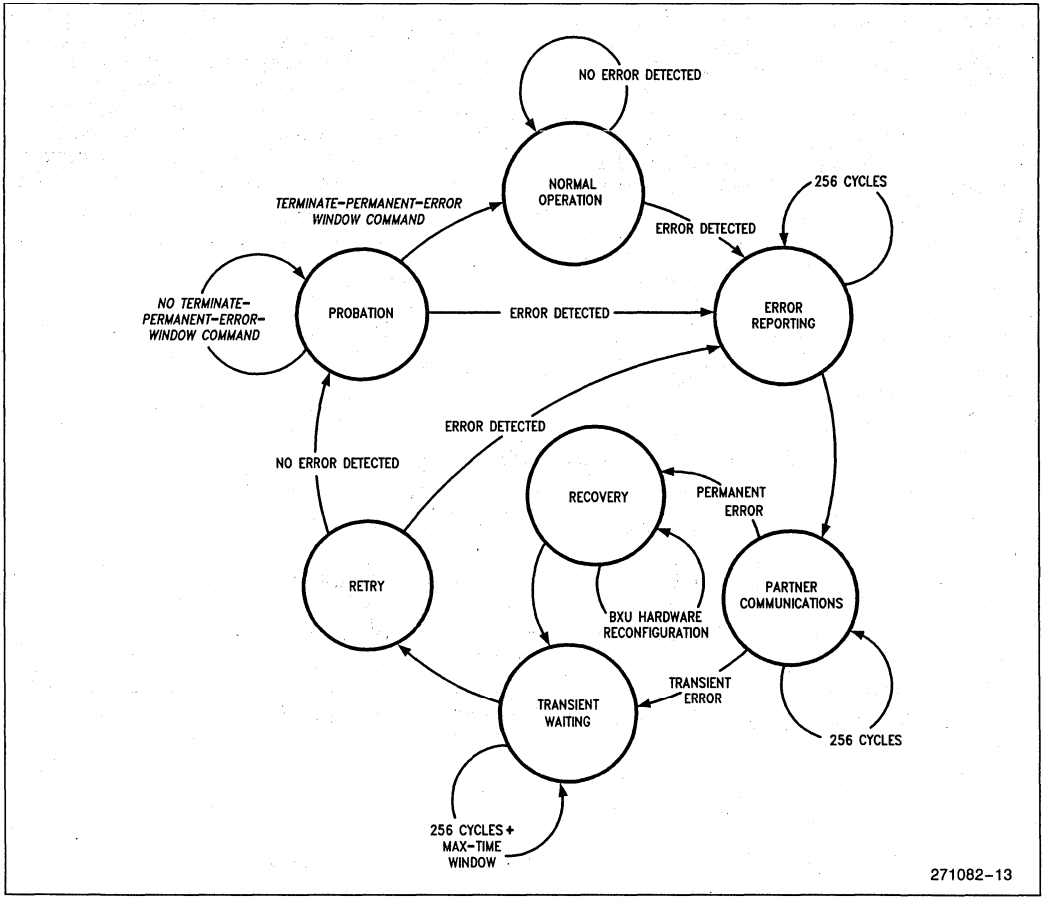


Figure 13. Error Reporting Cycle

During partner communications, BXUs communicate with each other via their POPQUE lines to determine whether to retry accesses in the case that one of the AP-Buses is removed from the system. Partner ordering lasts 256 cycles.

Transient waiting enables the system to sustain disturbances from mechanical vibrations and brief electrical transients without needing to permanently reconfigure the system. The BXUs simply wait a predetermined time for the transient to subside. The duration of the Transient Waiting Period is adjustable and can be set by software (16 μs to 500 ms at 16 MHz). During this period, the BXU completes its internal recovery mechanisms (if the error is permanent). Since the transient waiting mechanism on the buses depends on all buses moving to the retry state at the same time, all BXUs must have identical values for the Transient Waiting Period.

During the RETRY phase, all accesses that were pending at the time that the error report was received will be retried. At the same time as RETRY begins, the BXU enters the Permanent Error Window. During this interval, the BXU watches for the error to reoccur.

Each BXU has two registers that are used for logging error reports. The ERROR LOG register contains the current error report and the ERROR RECORD register contains the previous error report. When a error report is received, the contents of the ERROR LOG register are copied into the ERROR RECORD register. Both registers are accessible by software and are the primary means by which the software routines responsible for system management communicate with the hardware fault handling mechanisms. Table 6 lists the types of errors that can be reported.

Table 6. Error Types Reported

Error Type	Description
Unsafe Confinement Area	This type of report is issued when an error is detected that would make a retry dangerous.
Primary Catastophe	Generated in response to a Primary Catastrophe Command from software. The command is usually issued when all primary modules are about to fail because of a loss of power.
Shadow Catastophe	Generated in response to a Shadow Catastrophe Command from software. The command is usually issued when all shadow modules are about to fail because of a loss of power.
Error Reporting Error	The report indicates that a BXU has detected a failure on one of its error reporting lines.
Bus Arbitration	This report is issued when an FRC error is detected on the $\overline{\text{BOUT}}$ pin of the BXU indicating a bus arbitration error.
Bus Parity	Indicates that a parity error has been detected on the AP-Bus.
Component	Indicates that a checker has detected an FRC error while its master was driving the AP-Bus.
Uncorrectable Array Error	An uncorrectable error has been detected in one of the memory arrays.
Correctable ECC	A correctable error has been detected in one of the memory arrays.
COM Altered	This error report occurs when the COM input is toggled (two cycles high, followed by two cycles low) and may be used by external circuits to notify the system of an external fault.
Attach Bus	Issued in response to an Attach Bus command, this report is used to reactivate a bus that was previously out of service.
Detach Bus	Issued in response to a Detach Bus command, this report is used to remove a faulty bus from the system.
Terminate Permanent Error Window	Receiving this report signifies the end of the Permanent Error Window.
Sync Refresh	Used to synchronize memory modules that are being married to form a Primary/Shadow Pair.

The BXU's hardware compares the contents of the two error reporting registers to determine if a bus retry has resulted in a repeat of the previous error (which therefore must be considered a permanent error). Software can clear the two registers by sending a Terminate Permanent Error Window command. The registers allow software to monitor the health of the system and to respond appropriately in case of hardware problems. The availability of this information simplifies diagnostic routines.

The ERROR LOG register is handled independently by hardware and software; hardware always responds immediately to an error report so that it is never lost by failure of software to respond. During normal system operation, software should never write to this register, since it is both read and written by hardware. The ERROR LOG register is cleared on a cold start, but its contents are retained across a warm start.

RECOVERY MECHANISMS

Module Shadowing

Automatic recovery from permanent single-point failures in a module is accomplished through module shadowing, or what is more formally called Quad Modular Redundancy (QMR). Using this technique, two FRC pairs (master/checker) of the same type are logically linked to form a primary/shadow pair (see Figure 14). The marriage of the two modules is performed by software which sets the logical ID of the two modules equal and restarts them in lock step (or synchronous operation). There is no direct electrical connection between a primary/shadow pair. They are usually on separate boards so that either can be removed in the case of a failure in that module.

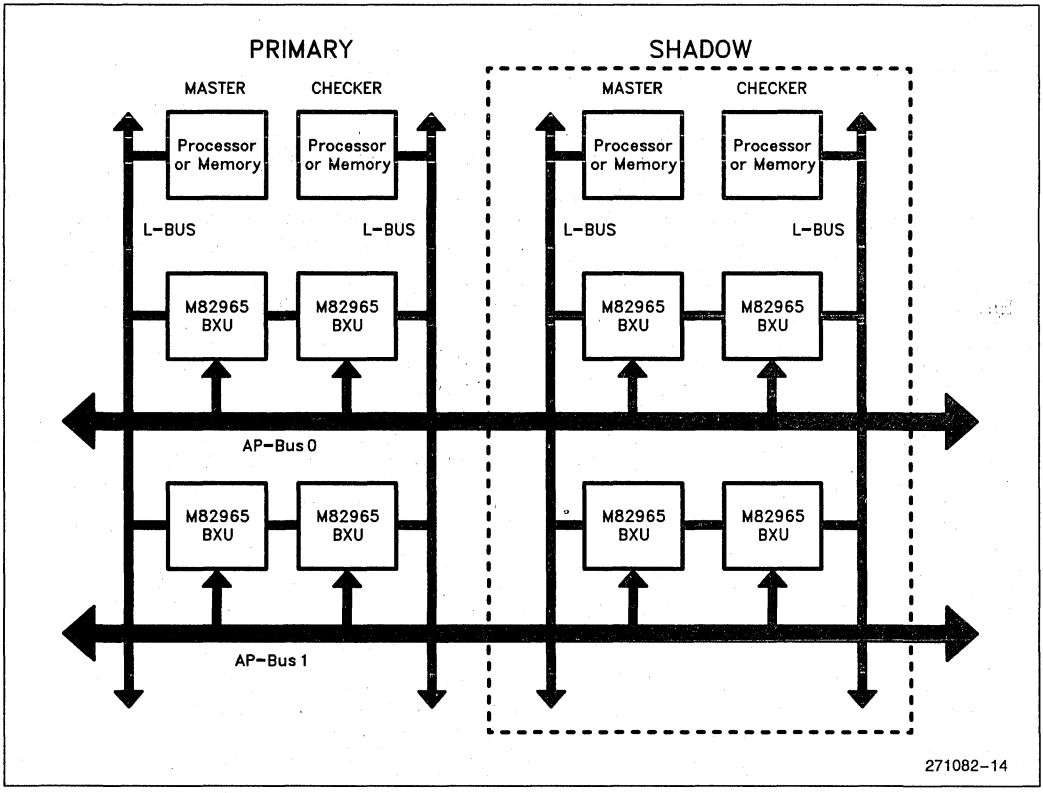


Figure 14. In Quad Modular Redundancy (QMR), Self-Checking Modules are Paired

The primary/shadow pair operate in lock step so that there is always a complete and current backup for an FRC pair. At any point in time, one FRC pair will be active (i.e., sending its output to the AP-Bus) while the other will be passive (i.e., its outputs will be disabled). Initially, the primary FRC pair is active and is responsible for issuing requests or replies to the AP-Bus. Data leaves only by means of the active FRC pair.

As an option, the roles of active and passive modules are switched after every second bus cycle. (In contrast, master/checker pairs are toggled every cycle). This ping-pong action exercises all of the logic in both primary and shadow modules. Any latent failure that exists in the AP-Bus drivers will be detected immediately. All of the logic to perform this lock step operation is contained in the BXU and neither the processors nor any discrete logic contained in a module is aware that the module is participating as one-half of a primary/shadow pair.

Each physical FRC pair (primary and shadow) remains a self-checking pair. Whether in an active or passive module, all detection mechanisms remain enabled and continuously check the operation of that module. Neither the primary nor the shadow check the operation of the other; FRC is used for fault detection, while module shadowing (Quad Modular Redundancy) is used to ensure immediate recovery.

Automatic Module Recovery

If a permanent error is detected in either a primary or a shadow FRC pair, the faulty pair will immediately be disabled as all BXUs in the pair shutdown. The surviving spouse then separates itself from the faulty FRC pair and operates as an active pair on every bus cycle. At that point, recovery is complete.

Hardware recovery is autonomous and requires no software intervention to complete. The operating system can be informed that a hardware reconfiguration has taken place by tying an error report line to one of the processor's interrupt pins. Then when a fault occurs, a processor can examine the error report log to discover what has happened and then reexamine the system configuration. Figure 15 shows an example of module recovery.

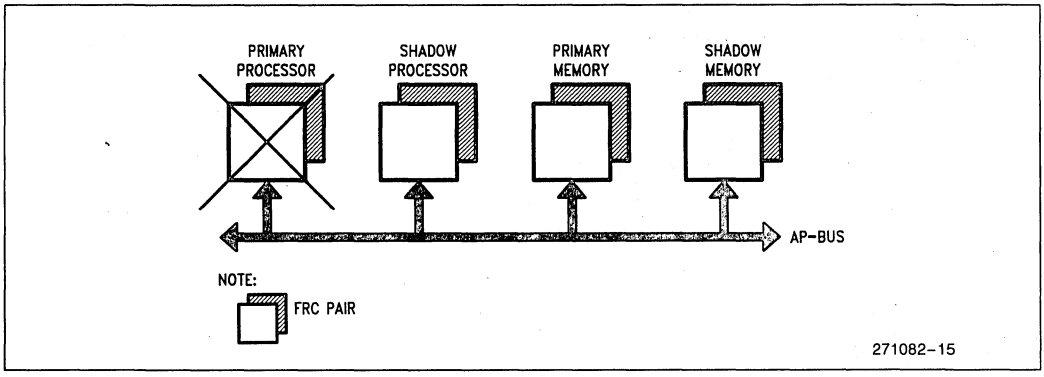


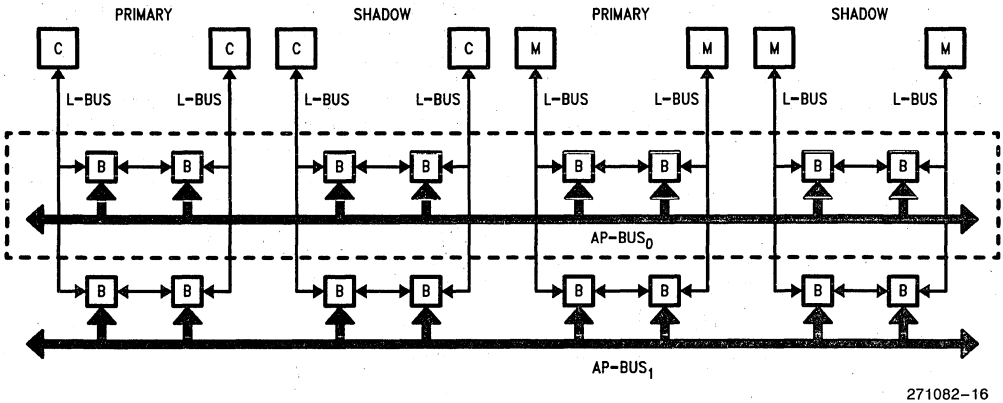
Figure 15. Faulty Modules are Automatically Disabled

Bus Switching

All AP-Buses in an 80960MC system are physically identical, but when a system is operational each bus handles a unique address range. The BXU has been designed so that it is possible to pair together two AP-Busses and have them act as redundant or alternate resources for each other. AP-Bus 0 is paired with AP-Bus 1 and AP-Bus 2 is paired with AP-Bus 3. In order for an FRC pair to have an additional bus, it must also have another pair of Master/Checker BXUs. Normally the memory addresses will be interleaved between the two (or four) buses, but this isn't necessary for bus switching.

Since the AP-Bus does not hold state information (as do processors and memory), all buses in the system may be used during normal operation. There is no degradation of throughput to achieve bus redundancy. Each bus is fully operational.

When a permanent error has been detected on an AP-Bus, all BXUs on the faulty bus disable themselves. L-Bus requests for the failed bus will be ignored by the disabled BXUs and picked up instead by the BXUs attached to the backup bus. If a BXU has a cache, the BXU invalidates its cache directory since the directory must be reorganized to match the new (and larger) address space, including a new interleaving factor. Figure 16 shows an example of bus switching.



Legend:
C = CPU
B = BXU
M = Memory Array
Hardware automatically reconfigures to bypass the faulty bus (AP-Bus$_0$).
AP-Bus$_1$ takes over the address space of AP-Bus$_0$.

Figure 16. If a Bus Fails, Its Backup Bus Takes Over Immediately

Self-Healing Systems

In some applications it is important to guarantee the integrity of the data, but momentary interruptions in processing can occur without seriously affecting operations or jeopardizing human lives. For these applications, a cost effective approach may be to use self-healing systems.

Self-healing systems use Functional Redundancy Checking to ensure that all errors are detected and that faults are confined within a module. Fault recovery is not automatic; recovery and reconfiguration is done by software following error detection. Self-healing systems are less costly than fully fault-tolerant systems because fewer components are necessary.

Self-healing systems do not operate continuously in the case of a hardware failure. Program execution cannot proceed after detection of a permanent error until the system has been reconfigured. Transient errors will still be taken care of by the hardware components. Upon detection of a permanent error, the system will cease operation, however FRC ensures that no data will have been corrupted.

After the system stops, it must be reset and a diagnostic program run which reads the BXU errors logs and determines the most appropriate action to take. Recovery and reconfiguration may be complete and the system back on-line within a few seconds to several minutes, depending on the nature of the fault.

Self-healing systems are not appropriate for real-time applications where program delays longer than a few milliseconds cannot be tolerated. In these critical applications, an interruption in system operation might result in damage to expensive material and equipment, or endangerment of human lives. The 80960MC system fault tolerant architecture provides the means for building systems that will recover automatically within 48 μs.

BXU Registers

Initialization and control of the BXU is done by reading and writing the BXU's internal registers. The registers are mapped to the upper 16 Mbytes of the 80960MC processor's physical address space.

Initialization of a system using BXUs occurs in three stages. In the first stage which immediately follows RESET, all registers (except for the registers containing error report information) are loaded with 0 or with values sampled off a set of pins.

During this stage the BXU's System Bus ID and mode of operation are established. In the second stage, software assigns logical, physical, and arbitration IDs to each BXU. Then in the third stage, the COM pin can be used to load board-specific information into the BXU and software can change the default values of any of the registers.

Once software has established the initial configuration of the system, no further interaction between the system software and the BXU may be necessary except for testing the error reporting functions and for making on-line changes to the system's initial configuration.

This Advance Information Data Sheet contains a functional description for each of the BXU's major register groups. For more specific details on controlling each of the registers, please consult the *80960MC Hardware Designer's Reference Manual.*

SIGNAL DESCRIPTIONS

Tables 7 through 11 describe the function of each of the BXU signals. Many of the pins are multiplexed and have different interpretations depending on whether the BXU is in Processor or Memory mode.

Table 7. M82965 BXU L-Bus Signals

Symbol	Type	Name and Function
LAD_{31} $-LAD_0$	I/O T.S.	**LOCAL ADDRESS/DATA BUS:** Carries 32-bit physical addresses and data to and from a processor or memory. During an address (T_a) cycle, bits 2–31 contain a physical word addres (bits 0–1 indicate SIZE; see below). During a data (T_d) cycle, bits 0–31 contain read or write data. The LAD lines are active HIGH and float to a three-state OFF when the bus is not acquired. **SIZE:** Which is comprised of bits 0–1 of the LAD bus during a T_a cycle, specifies the size of a transfer in words. **LAD_1 LAD_0** 0 0 1 Word 0 1 2 Words 1 0 3 Words 1 1 4 Words
$\overline{ALE}$	O T.S.	**ADDRESS-LATCH ENABLE:** Indicates the transfer of a physical address. $\overline{ALE}$ is asserted during a T_a cycle,and deasserted during T_d cycles and the second half of T_a cycles. It is active LOW and floats to a three-state OFF when the L-Bus is not acquired.
$\overline{ADS}$	I/O O.D.	**ADDRESS STATUS:** Is used to detect address cycles and additional data cycles.
CACHE	I	**CACHEABLE:** During a T_a cycle, specifies whether data is cacheable. **When operating in the MEMORY mode this pin should be tied to ground through a 10 kΩ resistor.**
W/$\overline{R}$	I/O O.D.	**WRITE/READ:** specifies, during a T_a cycle, whether the operation is a write or read. It is latched on-chip and remains valid during T_d cycles.
$\overline{CW}/\overline{DEN}$	O O.D.	**CACHE WRITE:** (Defined only when the BXU is in PROCESSOR mode). This signal indicates that the cache SRAM should be written with data from the L-Bus and is used to generate the chip select, and write enable signals required by the SRAM. The signal is open drain so it can be shared among multiple BXUs controlling a single set of SRAMs. **DATA ENABLE:** (Defined only when the BXU is in MEMORY mode). Is asserted during T_D cycles and indicates transfer of data on the local AD bus lines.
$\overline{CR}$/ DT/$\overline{R}$	O O.D.	**CACHE READ:** (Defined only when the BXU is in PROCESSOR mode). This signal indicates that the cache SRAM should drive data onto the L-Bus in response to a read request and is used to generate the chip select and output enable signals required by the SRAM. This signal is open drain so it can be shared among multiple BXUs controlling a single cache. **DATA TRANSMIT/RECEIVE:** (Defined only when the BXU is in MEMORY mode). Indicates the direction of data transfer. It is low during T_a and T_d cycles for a read or interrupt acknowledgement; it is high during T_a and T_d cycles for a write. DT/$\overline{R}$ never changes state when $\overline{DEN}$ is asserted.
$\overline{LOCK}$	I	**BUS LOCK:** Is used by the BXU to distinguish between normal reads and RMW-reads, normal writes and RMW-writes. An 80960MC processor asserts $\overline{LOCK}$ at the beginning of an RMW cycle, and the BXU recognizes it as an RMW-read. If the read operation is accepted by the module serving memory, the processor drops $\overline{LOCK}$, and executes an RMW-write. $\overline{LOCK}$ is also held asserted during an interrupt-acknowledge transaction.
$\overline{READY}$	I/O O.D	**READY:** Indicates that data on LAD lines can be sampled or removed. If $\overline{READY}$ is not asserted during a T_d cycle, the T_d cycle is extended to the next cycle, and $\overline{ADS}$ is not asserted in the next cycle. $\overline{READY}$ is driven on T_a, T_r and T_i cycles.

NOTES:
I/O = Input/Output, I = Input, O = Output, O.D. = Open Drain, T.S. = three-state

Table 7. M82965 BXU L-Bus Signals (Continued)

Symbol	Type	Name and Function
$\overline{BE}_3$–$\overline{BE}_0$	I/O O.D.	**BYTE ENABLES:** Specify which data bytes on the local bus will take part in the next bus cycle. $\overline{BE}_3$ corresponds to LAD_{24}–LAD_{31} and $\overline{BE}_0$ corresponds to LAD_0–LAD_7.
HOLD/ HOLDAR	I	**HOLD:** Indicates that a master I/O peripheral requests control of the bus. When the BXU receives HOLD and grants the peripheral control of the bus, it floats the bus lines and then asserts HLDA and enters the T_h state. When HOLD is deasserted, the BXU will deassert HLDA and go to either the T_i or T_a state.
		HOLD ACKNOWLEDGE REQUEST: Is an input to the secondary bus master that the primary bus master has relinquished control of the bus.
HLDA/ HOLDR	O	**HOLD ACKNOWLEDGE:** Relinquishes control of the bus to a master I/O peripheral. **HOLD REQUST:** Is used by a Secondary Bus Master to request use of the bus from the Primary Bus Master.

Table 8. M82965 BXU L-Bus Module Support Signals

Symbol	Type	Name and Function
$\overline{BADAC}$	O O.D.	**BAD ACCESS:** If asserted in the cycle following the one in which the last $\overline{READY}$ of a transaction is asserted as a result of a bad access, it indicates that the transaction has exceeded the AP-Bus time-out period.
$\overline{IAC}_0/\overline{ERR}$	I/O O.D.	**INTERAGENT COMMUNICATION: PROCESSOR 0:** (Defined only when the BXU is in PROCESSOR mode). Is an open-drain output that indicates that there is a pending IAC message for Processor 0 on the BXU's local bus. **EXTERNAL ERROR:** (Defined only when the BXU is in MEMORY mode). Is an input that indicates that an error has been detected in external logic (e.g., a failure in a discrete memory controller).
$\overline{IAC}_1/\overline{FRF}$	O O.D.	**INTERAGENT COMMUNICATION: PROCESSOR 1:** (Defined only when the BXU is in PROCESSOR mode). Is an open-drain output that indicates that there is a pending IAC message for Processor 1 on the BXU's local bus. **FORCE REFRESH:** (Defined only when the BXU is in memory mode). Is an open-drain output that tells the external memory controller to immediately execute a refresh operation.
$\overline{PFETCH}$	I	**PREFETCH:** Is used in conjunction with the Cache and Write/Read (W/$\overline{R}$) signals to define the type of request being issued (0 = LO, 1 = HI): **$\overline{PFETCH}$ CACHE W/$\overline{R}$** 0 0 0 Read using Prefetch Channel 0 0 0 1 Start for Prefetch Channel 0 0 1 0 Read using Prefetch Channel 1 0 1 1 Start for Prefetch Channel 1 1 0 0 Noncacheable Read 1 0 1 Noncacheable Write 1 1 0 Cacheable Read 1 1 1 Cacheable Write

NOTES:
I/O = Input/Output, I = Input, O = Output, O.D. = Open Drain

Table 9. M82965 BXU AP-Bus Signals

Symbol	Type	Name and Function
$\overline{AD}_{31}$–$\overline{AD}_0$	I/O O.D.	**SYSTEM ADDRESS/DATA LINES:** Carry 32-bit addresses and data between modules (BXUs) on an AP-Bus. The content of the AD lines is defined by the SPEC encoding during the same bus cycle.
$\overline{SPEC}_5$–$\overline{SPEC}_0$	I/O O.D.	**PACKET SPECIFICATION:** Signals define the packet type and the parameters required for the transaction: **$SPEC_5$: REQUEST:** Is asserted if the packet is a request packet. **$SPEC_4$: MULTICYCLE:** Is asserted if the packet consists of more than one bus cycle. **$SPEC_3$–$SPEC_2$: CYCLE COUNT:** These two bits are used in conjunction with Request and Multicycle signals to specify the length of the packet (in bus cycles) and the data length (in words). **$SPEC_1$–$SPEC_0$: OPERATION/STATUS TYPE:** These two bits identify the specific operation or status conveyed by the packet.
$\overline{CHK}_1$–$\overline{CHK}_0$	I/O O.D.	**CHECK SIGNALS:** Provide interlaced parity for the $\overline{SPEC}$ and $\overline{AD}$ lines.
$\overline{ARB}_3$–$\overline{ARB}_0$	I/O O.D.	**ARBITRATION:** Signals are used by the bus agents (BXUs) to determine which agent has access to the bus next. These signals have a timing that is one-half cycle out of phase with the $\overline{AD}$ lines.
$\overline{RPYDEF}$	I/O O.D.	**REPLY DEFER:** Signal allows an agent to give up its "slot" on the bus temporarily if its access is going to take a long time. This action reorders the pipeline, moving the deferred request to the bottom of the queue, resets the bus time-out counter and permits another agent to use the bus.
$\overline{BERL}_1$–$\overline{BERL}_0$	I/O O.D.	**BUS ERROR REPORT LINES:** Is used to signal errors from bus transactions or from within modules connected to the bus.

NOTES:
I/O = Input/Output, I = Input, O = Output, O.D. = Open Drain

Table 10. M82965 BXU AP-Bus (Local Agent) Support Signals

Symbol	Type	Name and Function
CLK2	I	**SYSTEM CLOCK:** Provides the base timing and synchronization for all agents (BXUs) in the system. It is sourced to all agents from a central clock and is twice the frequency of the bus cycle. **NOTE:** The clock skew over the AP-Bus for a typical system should be no greater than 6 ns for correct system operation.
$\overline{BOUT}$	I/O O.D.	**BUS OUTPUT CONTROL:** Is asserted whenever a component is driving the AP-Bus. Functional Redundancy Checks on $\overline{BOUT}$ can be used to detect arbitration failures.
$\overline{MODCHK}$	I/O O.D.	**MODULE CHECK:** Is connected between Master/Checker pairs, allowing a Functional Redundancy Check to be performed on internal states.
$\overline{INITID}$	I	**INITIALIZE ID:** Is connected to one of the 32 $\overline{AD}$ lines and is used in conjunction with the IDENTIFY DEVICE IAC to provide a unique address for each BXU at initialization time.
V_{REF}	I	**VOLTAGE REFERENCE:** Provides a stable voltage reference for the input buffers of components connected to the AP-Bus. External hardware must provide a V_{REF}/W voltage (see Table 14) on the V_{REF} pin during normal operation of the component. The V_{REF} pin is also used to distinguish between a warm start (system memory and the Error Record register retain their state) and a cold start (system memory and BXU registers are cleared).
RESET	I	**RESET:** Forces all agents on the bus to reset and synchronize. The bus cycle begins the first CLK2 period after RESET is deasserted. The RESET signal is the way a BXU is synchronized to the rest of the system.
$\overline{COM}$	I/O O.D.	**COMMUNICATION:** Can be used to load information into a component as part of the initialization sequence or to inform external logic that the component has failed. The BXU will asserted $\overline{COM}$ if it has shut itself off due to a failure in its module. The $\overline{COM}$ signal is not involved in any aspect of AP-Bus operation, but can be used to load board-dependent information into the BXU or to signal the rest of the system that an external error has occurred.

NOTES:
I/O = Input/Output, I = Input, O = Output, O.D. = Open Drain

Table 11. M82965 BXU Module Support Signals

Symbol	Type	Name and Function
$\overline{WY_0}/\overline{COR}$	O O.D.	**WAY_0:** (When the BXU is in processor mode). Indicates which one of the two "ways" in a directory set had a cache hit. The line is intended to drive the SRAM address pins and will remain stable throughout the length of a cache access. **CORRECT:** (When the BXU is in memory mode). Is used by the BXU to tell an external ECC controller to correct the memory data as it flows onto the local bus. If this signal is not asserted, then the memory data may flow directly onto the local bus with only error checking, but no correction.
$\overline{WY_1}$/MEM	O O.D.	**WAY_1:** (Defined only when the BXU is in PROCESSOR mode). Indicates if the access is for the cache or private memory half of the SRAM. The line is intended to drive the SRAM address lines directly and will remain stable throughout the length of a cache access. **MEMORY/REGISTER REQUEST:** (Defined only when the BXU is in MEMORY mode). This signal allows mapping some of the BXU's register space out to the registers in an external controller. If the signal is high, the associated L-Bus request is a memory request; otherwise, the L-Bus request is to an external register on the board.
$\overline{WD_0}/\overline{UNC}$	I/O O.D.	**$WORD_0$:** (Only defined when the BXU is in PROCESSOR mode). Provides the low order bit of the word address for the SRAM. Together with $WORD_1$, the two bits indicate which of the four words within an address line should be addressed. Because SRAM timing is critical, an external latch could be required. The signals change for each word of data transferred. **UNCORRECTABLE ECC:** (Only defined when the BXU is in MEMORY mode). Is an input used by the external ECC logic to signal to the BXU that it has detected an uncorrectable memory error.
$\overline{WD_1}/\overline{ECC}$	I/O O.D.	**$WORD_1$:** (Defined only when the BXU is in PROCESSOR mode). Provides the high order bit of the word address for the SRAM. Together with $WORD_0$, the two bits indicate which of the four words within an address line should be addressed. Because SRAM timing is critical, an external latch will be required. The signals change for each word of data transferred. **ECC ERROR:** (Defined only when the BXU is in MEMORY mode). Is an input used by the external ECC logic to signal to the BXU that it has detected a memory error. The signal will be asserted even though external logic may be correcting the error and providing correct data on the L-Bus. If the BXU is asserting its $\overline{CORRECT}$ signal, the ECC ERROR signal will be ignored. Only the $\overline{UNC}$ pin will be checked for an error indication under these conditions.
$\overline{SSBUSY}$	I/O O.D.	**SUBSYSTEM BUSY:** Connects together all BXUs in a module that are in the same subsystem. When the signal is pulled low (BUSY), the BXUs will accept a request address, but will not continue with the data cycles. This signal is used to ensure that the BXUs always handle RMW-writes, Interagent Communication messages, and retries correctly. An external signal is needed because BXUs can generate AP-Bus requests internally because of the prefetcher, or their internal logic can be tied up handling an IAC request from the AP-Bus.

Table 11. M82965 BXU Module Support Signals (Continued)

Symbol	Type	Name and Function
$\overline{POPQUE}$	I/O O.D.	**POP QUEUE:** Is used by the two BXUs acting as bus backups for each other to communicate status on the completion of outstanding L-Bus requests. Usually, this signal is asserted when the oldest write in the queue has completed. During the partner ordering period, a different protocol is used to convey the status of all write requests outstanding.
$\overline{LERL_1}$–$\overline{LERL_0}$	I/O O.D.	**LOCAL ERROR REPORTING LINES:** Are identical to the BERL signals defined for the AP-Bus, but are used on the module side to connect all BXUs on a single L-Bus.

NOTES:
I/O = Input/Output, I = Input, O = Output, O.D. = Open Drain

MECHANICAL DATA

Pin Assignment

The MG82965 BXU (PGA package) pinout as viewed from the top side of the component (pins down) is shown in Figure 17 and from the bottom side (pins up) in Figure 18.

V_{CC} and GND connections must be made to multiple V_{CC} and GND pins. Each V_{CC} and GND pin must be connected to the appropriate voltage or ground and externally strapped close to the package. Preferably, the circuit board should include power and ground planes for power distribution. Table 12 lists the function of each pin.

Many of the signals are multiplexed and several signals have different interpretations depending on whether the BXU is used in Processor or Memory mode.

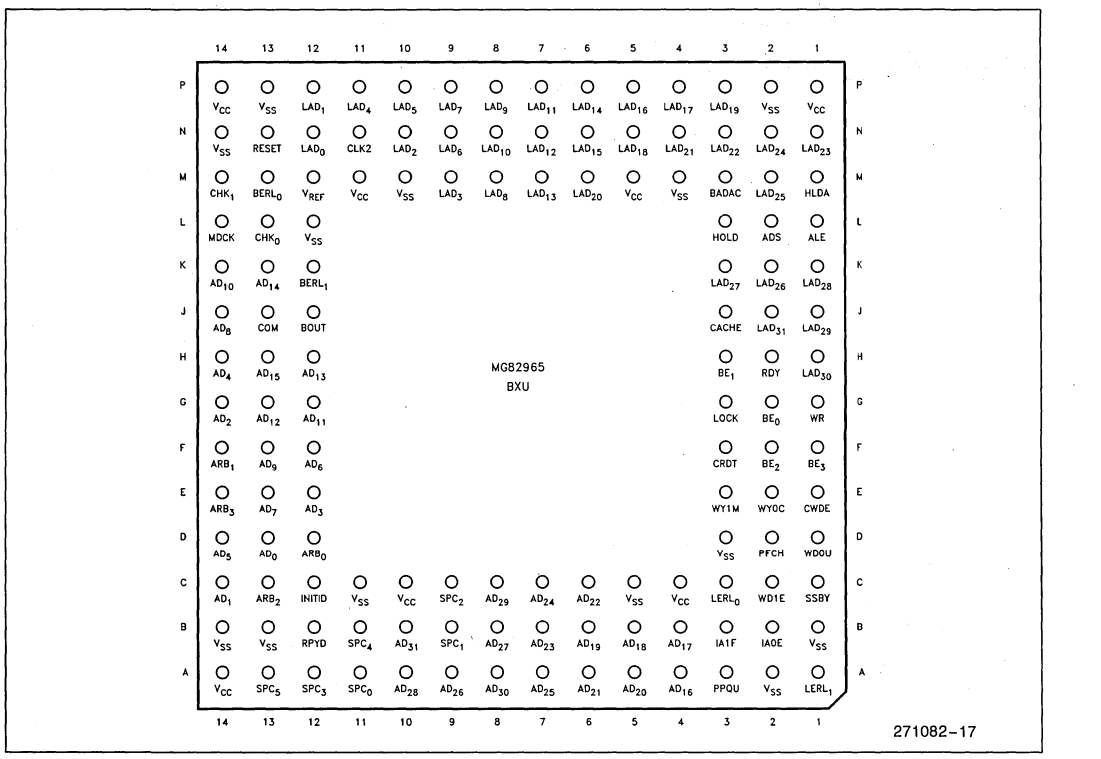


Figure 17. MG82965 BXU Pinout—View from Top Side (Pins Down)

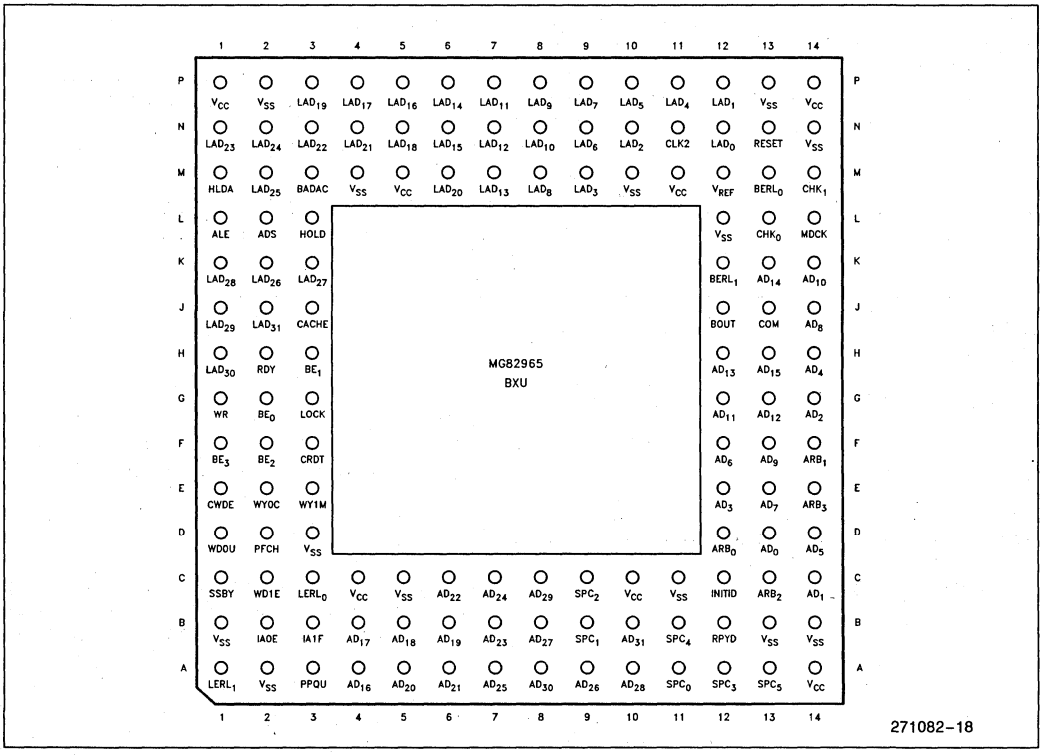


Figure 18. MG82965 BXU Pinout—View from Bottom Side (Pins Up)

Table 12. M82965 PGA Pinout—In Pin Order

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
A1	$\overline{LERL_1}$	C6	$\overline{AD_{22}}$	H1	LAD_{30}	M10	V_{SS}
A2	V_{SS}	C7	$\overline{AD_{24}}$	H2	$\overline{READY}$	M11	V_{CC}
A3	$\overline{POPQUE}$	C8	$\overline{AD_{29}}$	H3	$\overline{BE_1}$	M12	V_{REF}
A4	$\overline{AD_{16}}$	C9	$\overline{SPEC_2}$	H12	$\overline{AD_{13}}$	M13	$\overline{BERL_0}$
A5	$\overline{AD_{20}}$	C10	V_{CC}	H13	$\overline{AD_{15}}$	M14	$\overline{CHK_1}$
A6	$\overline{AD_{21}}$	C11	V_{SS}	H14	$\overline{AD_4}$	N1	LAD_{23}
A7	$\overline{AD_{25}}$	C12	$\overline{INITID}$	J1	LAD_{29}	N2	LAD_{24}
A8	$\overline{AD_{30}}$	C13	$\overline{ARB_2}$	J2	LAD_{31}	N3	LAD_{22}
A9	$\overline{AD_{26}}$	C14	$\overline{AD_1}$	J3	CACHE	N4	LAD_{21}
A10	$\overline{AD_{28}}$	D1	$\overline{WD_0/UNC}$	J12	$\overline{BOUT}$	N5	LAD_{18}
A11	$\overline{SPEC_0}$	D2	$\overline{PFETCH}$	J13	$\overline{COM}$	N6	LAD_{15}
A12	$\overline{SPEC_3}$	D3	V_{SS}	J14	$\overline{AD_8}$	N7	LAD_{12}
A13	$\overline{SPEC_5}$	D12	$\overline{ARB_0}$	K1	LAD_{28}	N8	LAD_{10}
A14	V_{CC}	D13	$\overline{AD_0}$	K2	LAD_{26}	N9	LAD_6
B1	V_{SS}	D14	$\overline{AD_5}$	K3	LAD_{27}	N10	LAD_2
B2	$\overline{IAC_0/ERR}$	E1	$\overline{CW/DEN}$	K12	$\overline{BERL_1}$	N11	CLK2
B3	$\overline{IAC_1/FRF}$	E2	$\overline{WY_0/COR}$	K13	$\overline{AD_{14}}$	N12	LAD_0
B4	$\overline{AD_{17}}$	E3	$\overline{WY_1}$/MEM	K14	$\overline{AD_{10}}$	N13	RESET
B5	$\overline{AD_{18}}$	E12	$\overline{AD_3}$	L1	$\overline{ALE}$	N14	V_{SS}
B6	$\overline{AD_{19}}$	E13	$\overline{AD_7}$	L2	$\overline{ADS}$	P1	V_{CC}
B7	$\overline{AD_{23}}$	E14	$\overline{ARB_3}$	L3	HOLD	P2	V_{SS}
B8	$\overline{AD_{27}}$	F1	$\overline{BE_3}$	L12	V_{SS}	P3	LAD_{19}
B9	$\overline{SPEC_1}$	F2	$\overline{BE_2}$	L13	$\overline{CHK_0}$	P4	LAD_{17}
B10	$\overline{AD_{31}}$	F3	$\overline{CR}$/DT/$\overline{R}$	L14	$\overline{MODCHK}$	P5	LAD_{16}
B11	$\overline{SPEC_4}$	F12	$\overline{AD_6}$	M1	HLDA	P6	LAD_{14}
B12	$\overline{RPYDEF}$	F13	$\overline{AD_9}$	M2	LAD_{25}	P7	LAD_{11}
B13	V_{SS}	F14	$\overline{ARB_1}$	M3	$\overline{BADAC}$	P8	LAD_9
B14	V_{SS}	G1	W/$\overline{R}$	M4	V_{SS}	P9	LAD_7
C1	$\overline{SSBUSY}$	G2	$\overline{BE_0}$	M5	V_{CC}	P10	LAD_5
C2	$\overline{WD_1/ECC}$	G3	$\overline{LOCK}$	M6	LAD_{20}	P11	LAD_4
C3	$\overline{LERL_0}$	G12	$\overline{AD_{11}}$	M7	LAD_{13}	P12	LAD_1
C4	V_{CC}	G13	$\overline{AD_{12}}$	M8	LAD_8	P13	V_{SS}
C5	V_{SS}	G14	$\overline{AD_2}$	M9	LAD_3	P14	V_{CC}

3

Table 13. M82965 Pinout—In Signal Order

Signal	PGA Pin	Signal	PGA Pin	Signal	PGA Pin	Signal	PGA Pin
$\overline{AD}_0$	D13	$\overline{ALE}$	L1	LAD_8	M8	$\overline{SPEC}_0$	A11
$\overline{AD}_1$	C14	$\overline{ARB}_0$	D12	LAD_9	P8	$\overline{SPEC}_1$	B9
$\overline{AD}_2$	G14	$\overline{ARB}_1$	F14	LAD_{10}	N8	$\overline{SPEC}_2$	C9
$\overline{AD}_3$	E12	$\overline{ARB}_2$	C13	LAD_{11}	P7	$\overline{SPEC}_3$	A12
$\overline{AD}_4$	H14	$\overline{ARB}_3$	E14	LAD_{12}	N7	$\overline{SPEC}_4$	B11
$\overline{AD}_5$	D14	$\overline{BADAC}$	M3	LAD_{13}	M7	$\overline{SPEC}_5$	A13
$\overline{AD}_6$	F12	$\overline{BE}_0$	G2	LAD_{14}	P6	$\overline{SSBUSY}$	C1
$\overline{AD}_7$	E13	$\overline{BE}_1$	H3	LAD_{15}	N6	V_{CC}	A14
$\overline{AD}_8$	J14	$\overline{BE}_2$	F2	LAD_{16}	P5	V_{CC}	C4
$\overline{AD}_9$	F13	$\overline{BE}_3$	F1	LAD_{17}	P4	V_{CC}	C10
$\overline{AD}_{10}$	K14	$\overline{BERL}_0$	M13	LAD_{18}	N5	V_{CC}	M5
$\overline{AD}_{11}$	G12	$\overline{BERL}_1$	K12	LAD_{19}	P3	V_{CC}	M11
$\overline{AD}_{12}$	G13	$\overline{BOUT}$	J12	LAD_{20}	M6	V_{CC}	P1
$\overline{AD}_{13}$	H12	CACHE	J3	LAD_{21}	N4	V_{CC}	P14
$\overline{AD}_{14}$	K13	$\overline{CHK}_0$	L13	LAD_{22}	N3	V_{REF}	M12
$\overline{AD}_{15}$	H13	$\overline{CHK}_1$	M14	LAD_{23}	N1	V_{SS}	A2
$\overline{AD}_{16}$	A4	CLK2	N11	LAD_{24}	N2	V_{SS}	B1
$\overline{AD}_{17}$	B4	$\overline{COM}$	J13	LAD_{25}	M2	V_{SS}	B13
$\overline{AD}_{18}$	B5	$\overline{CR}$/DT/$\overline{R}$	F3	LAD_{26}	K2	V_{SS}	B14
$\overline{AD}_{19}$	B6	CW/$\overline{DEN}$	E1	LAD_{27}	K3	V_{SS}	C5
$\overline{AD}_{20}$	A5	HLDA	M1	LAD_{28}	K1	V_{SS}	C11
$\overline{AD}_{21}$	A6	HOLD	L3	LAD_{29}	J1	V_{SS}	D3
$\overline{AD}_{22}$	C6	$\overline{IAC}_0$/$\overline{ERR}$	B2	LAD_{30}	H1	V_{SS}	L12
$\overline{AD}_{23}$	B7	$\overline{IAC}_1$/$\overline{FRF}$	B3	LAD_{31}	J2	V_{SS}	M4
$\overline{AD}_{24}$	C7	$\overline{INITID}$	C12	$\overline{LERL}_0$	C3	V_{SS}	M10
$\overline{AD}_{25}$	A7	LAD_0	N12	$\overline{LERL}_1$	A1	V_{SS}	N14
$\overline{AD}_{26}$	A9	LAD_1	P12	$\overline{LOCK}$	G3	V_{SS}	P2
$\overline{AD}_{27}$	B8	LAD_2	N10	$\overline{MODCHK}$	L14	V_{SS}	P13
$\overline{AD}_{28}$	A10	LAD_3	M9	$\overline{PFETCH}$	D2	$\overline{WD}_0$/$\overline{UNC}$	D1
$\overline{AD}_{29}$	C8	LAD_4	P11	$\overline{POPQUE}$	A3	$\overline{WD}_1$/$\overline{ECC}$	C2
$\overline{AD}_{30}$	A8	LAD_5	P10	$\overline{READY}$	H2	W/$\overline{R}$	G1
$\overline{AD}_{31}$	B10	LAD_6	N9	RESET	N13	$\overline{WY}_0$/$\overline{COR}$	E2
$\overline{ADS}$	L2	LAD_7	P9	$\overline{RPYDEF}$	B12	$\overline{WY}_1$/MEM	E3

Package Dimensions and Mounting

The MG82965 BXU is packaged in either a 132-lead ceramic pin-grid array (PGA) or a 164-pin CQP package. (Contact factory for details on CQP availability.) Pins in the PGA package are arranged 0.100 inch (2.54 mm) center-to-center, in a 14 by 14 matrix, three rows around. See Figure 19.

A wide variety of available sockets allows low-insertion or zero-insertion force mountings, and a choice of terminals such as soldertail, surface mount, or wire-wrap. Figure 20 shows several applicable sockets.

Package Thermal Specification

The M82965 BXU is specified for operation when its case temperature is within the range of −55°C to +125°C. The PGA case temperature should be measured at the center of the top surface opposite the pins as shown in Figure 21.

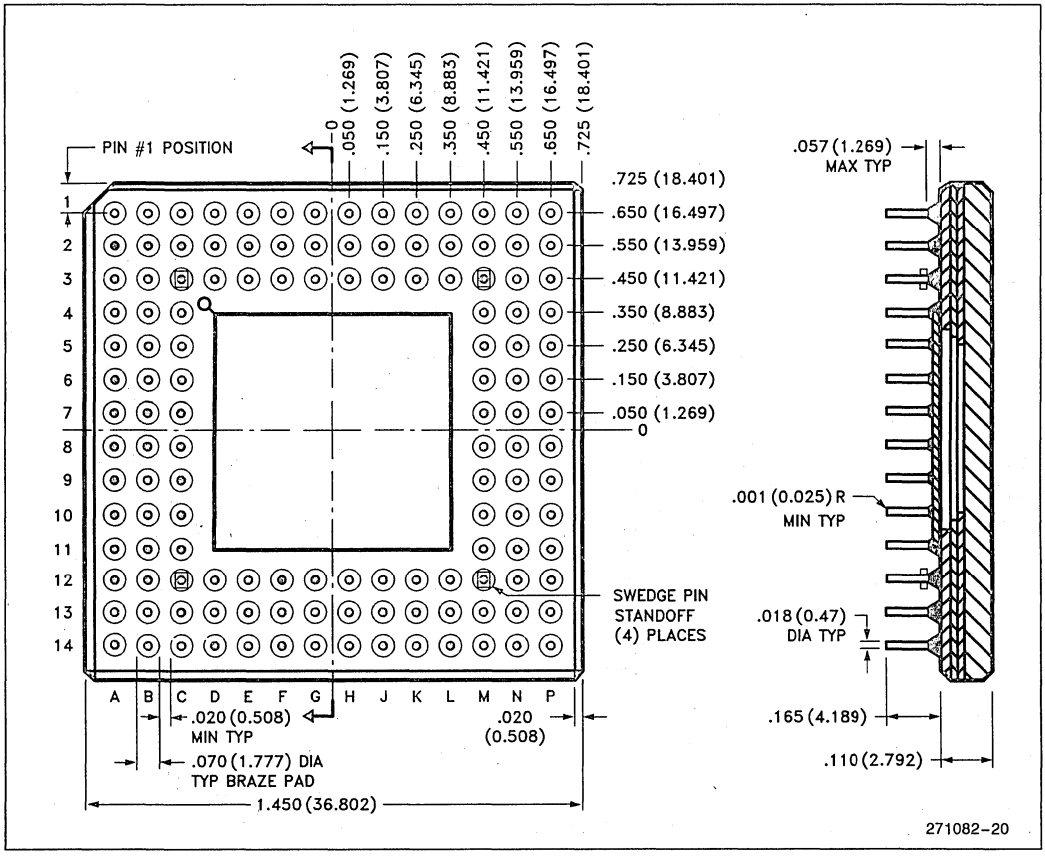


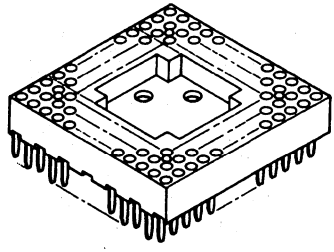
Figure 19. A 132-Lead Pin-Grid Array (PGA) Used to Package the MG82965 BXU

- Low insertion force (LIF) soldertail 55274-1
- Amp tests indicate 50% reduction in insertion force compared to machined sockets

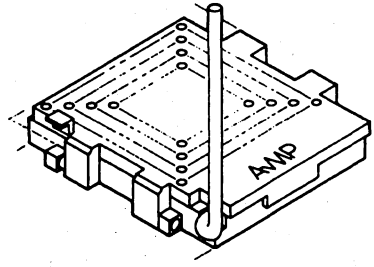
Other socket options

- Zero insertion force (ZIF) soldertail 55583-1
- Zero insertion force (ZIF) Burn-in version 55573-2

Amp Incorporated
(Harrisburg, PA 17105 U.S.A
Phone 717-564-0100)



Amp LIF Socket
55274-1



Amp LIF Socket

271082-21

Cam handle locks in low profile position when MG82965 is installed (handle UP for open and DOWN for closed positions).

Courtesy Amp Incorporated

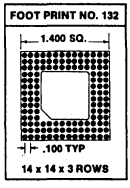
Peel-A-Way* and Kapton Socket Terminal Carriers

- Low insertion force surface mount CS132-37TG
- Low insertion force soldertail CS132-01TG
- Low insertion force wire-wrap CS132-02TG (two-level) CS132-03TG (three-level)
- Low insertion force press-fit CS132-05TG

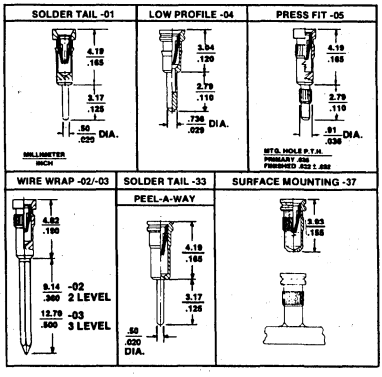
Advanced Interconnections
(5 Division Street)
Warwick, RI 02818 U.S.A.
Phone 401-885-0485)

Peel-A-Way Carrier No. 132:
Kapton Carrier is KS132
Mylar Carrier is MS132

Molded Plastic Body KS132 is shown below:



271082-22



271082-23

Courtesy Advanced Interconnections
(Peel-A-Way Terminal Carriers
U.S. Patent No. 4442938)

*Peel-A-Way is a trademark of Advanced Interconnections.

Figure 20. Several Socket Options for Mounting the M82965 BXU

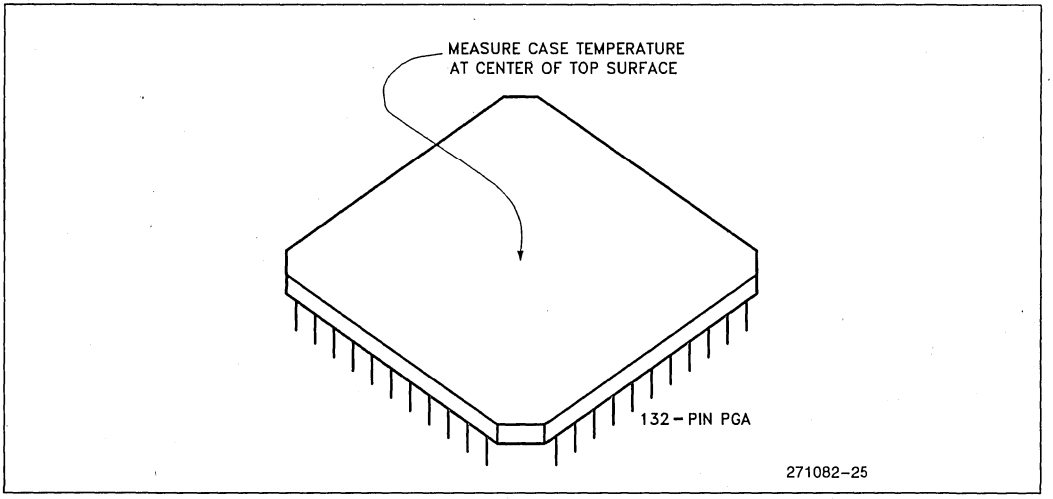


Figure 21. Measuring MG82965 Case Temperature

ELECTRICAL SPECIFICATIONS

Power and Grounding

The M82965 is implemented in CHMOS III technology and has modest power requirements. Its high clock frequency and numerous output buffers (address/data, control, error, and arbitration signals) can cause power surges as multiple output buffers drive new signal levels simultaneously. For clean on-chip power distribution at high frequency, seven V_{CC} and thirteen V_{SS} pins separately feed functional units of the M82965.

Power and ground connections must be made to all V_{CC} and V_{SS} pins of the M82965. On the circuit board, all V_{CC} pins must be strapped closely together, preferably on a V_{CC} plane. Likewise, all V_{SS} pins should be strapped together, preferably on a ground plane.

Power Decoupling Recommendations

Liberal decoupling capacitance should be placed near the M82965. The BXU when driving its two 32-bit address/data buses (AP-Bus and L-Bus) can cause transient power surges, particularly when driving large capacitive loads.

Low inductance capacitors and interconnects are recommended for best high frequency electrical performance. Inductance can be reduced by shortening the board traces between the BXU and decoupling capacitors as much as possible.

Connection Recommendations

For reliable operation, always connect unused inputs to an appropriate signal level. In particular, if $\overline{PFETCH}$ or $\overline{LERL}_{0-1}$ are not used, they should be pulled up and if the CACHE input is not used (i.e., BXU operating in the Memory mode) it should be tied low through a 10 kΩ resistor. No inputs should ever be left floating.

All open-drain outputs require a pullup device. While in most cases a simple pullup resistor will be adequate, a network of pullup and pulldown resistors biased to a valid V_{IH} (e.g., 3.5V) will limit noise and AC power consumption, especially on the AP-Bus.

ABSOLUTE MAXIMUM RATINGS*

Case Temperature
under Bias(1) ..... −55°C to +125°C Case
Storage Temperature ..... −65°C to +150°C
Voltage on Any Pin ..... −0.5V to V_{CC} + 0.5V
Power Dissipation 2.5W

NOTICE: This data sheet contains information on products in the sampling and initial production phases of development. The specifications are subject to change without notice. Verify with your local Intel Sales office that you have the latest data sheet before finalizing a design.

**WARNING: Stressing the device beyond the "Absolute Maximum Ratings" may cause permanent damage. These are stress ratings only. Operation beyond the "Operating Conditions" is not recommended and extended exposure beyond the "Operating Conditions" may affect device reliability.*

Operating Conditions

Symbol	Description	Min	Max	Units
T_C	Case Temperature (Instant On)	−55	+125	°C
V_{CC}	Digital Supply Voltage	4.75	5.25	V

Table 14. D.C. Characteristics (Over Specified Operating Conditions)

Symbol	Parameter	Min	Max	Units	Comments
V_{IL}	Input Low Voltage	−0.3	+0.8	V	
V_{ILA}	Input Low Voltage: AP-Bus	−0.5	+1.0	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.3	V	
V_{IHA}	Input High Voltage: AP-Bus	2.0	V_{CC}	V	
$V_{REF/C}$	V_{REF} Trip Point Cold Start	V_{CC} − 0.7		V	
$V_{REF/W}$	V_{REF} Trip Point Warm Start	1.7	1.8	V	
V_{CL}	CLK2 Input Low Voltage	−0.3	+1.0	V	
V_{CH}	CLK2 Input High Voltage	0.55 V_{CC}	V_{CC}	V	
V_{OL}	Output Low Voltage: I_{OL} = 4 mA: LAD Lines I_{OL} = 5 mA: Controls(2) I_{OL} = 25 mA: L-Bus Open-Drain Outputs I_{OL} = 80 mA: AP-Bus Open-Drain Outputs		 0.45 0.45 0.45 0.70	 V V V V	
V_{OH}	Output High Voltage: I_{OH} = 1 mA: LAD Lines I_{OH} = 0.9 mA: Controls(2) I_{OH} = 5.0 mA: ALE	 2.4 2.4 2.4		 V V V	
I_{CC}	Power Supply Current		450	mA	
I_{LI}	Input Leakage Current		±15	μA	$0V \le V_O \le V_{CC}$
I_{LO}	Output Leakage Current		±15	μA	$0.45V \le V_O \le V_{CC}$
C_{IN}	Input Capacitance		10	pF	Note 1
C_O	I/O or Output Capacitance		12	pF	Note 1
C_{CLK}	Clock Capacitance		12	pF	Note 1

NOTES:
1. Test frequency = 1 MHz, T_C = 25°C, unmeasured pins at GND.
2. "Controls" include all L-Bus I/O pins not otherwise specified.

A.C. SPECIFICATIONS

This section describes the A.C. specifications for the M82965 pins. All input and output timings are specified relative to the 1.5V level of the rising edge of CLK2, and refer to the time at which the signal reaches (for output delay and input setup) or leaves (for hold time) the TTL levels of LOW (0.8V) or HIGH (2.0V).

All A.C. testing should be done with input voltages of 0.45V and 2.4V.

Maximum output hold times are the same as minimum output delays. Tri-state signals have no resistive load or termination.

The Output Delay specified for open-drain signals includes both the low to high and high to low transitions. The float delay is the amount of time that the pulldown transistor may remain active. This specification is provided to help system designers calculate propagation delay for terminations other than the one used for testing.

Table 15. M82965 A.C. Timing Specifications (Over Specified Operating Conditions)

Symbol	Parameter	Min	Max	Units	Comments
T_1	Clock Period	31.25	125	ns	V_{IN} = 1.5V
T_2	Clock Low Time	11		ns	V_{IN} = 10% Point = 1.2V
T_3	Clock High Time	11		ns	V_{IN} = 90% Point = 0.1V + 0.5 V_{CC}
T_4	Clock Fall Time		10	ns	V_{IN} = 90% Point to 10% Point
T_5	Clock Rise Time		10	ns	V_{IN} = 10% Point to 90% Point
T_6	Output Valid Delay:				
	LAD	4	35	ns	C_L = 100 pF
	$\overline{WY}$	4	35	ns	C_L = 125 pF
	$\overline{CW}$, $\overline{WD}$, SS Busy	4	30	ns	C_L = 75 pF
	$\overline{CR}$	4	45	ns	C_L = 75 pF
	Controls(1)	2	35	ns	C_L = 75 pF
T_7	$\overline{ALE}$ Width	15		ns	C_L = 75 pF
T_8	$\overline{ALE}$ Invalid Delay		20	ns	C_L = 75 pF
T_9	Output Float Delay:				
	LAD	5	20	ns	C_L = 100 pF
	$\overline{WY}$	5	22	ns	C_L = 125 pF
	Controls(1)	5	22	ns	C_L = 75 pF
T_{10}	Input Setup Time:				
	$\overline{LOCK}$, HOLD, HOLDAR, $\overline{READY}$	8		ns	10% Point
	$\overline{ECC}$, $\overline{UNC}$	15		ns	10% Point
	Controls(1)	3		ns	10% Point
T_{11}	Input Data Hold	10		ns	90% Point
T_{12}	Setup to $\overline{ALE}$ Inactive	10		ns	C_L = 100 pF (LAD) C_L = 75 pF (Controls)
T_{13}	Hold after $\overline{ALE}$ Inactive	8		ns	C_L = 100 pF (LAD) C_L = 75 pF (Controls)
T_{14}	RESET Hold	5		ns	
T_{15}	RESET Setup	8		ns	
T_{16}	RESET Width	1250		ns	40 CLK2 Periods Minimum
T_{17}	Clock to Data Valid (AP-Bus)		17	ns	C_L = 50 pF I_{OL} = 50 mA
T_{18}	Clock to High Impedance (AP-Bus)		14	ns	
T_{19}	Output Hold (AP-Bus)	5		ns	C_L = 50 pF I_{OL} = 50 mA
T_{20}	Input Setup (AP-Bus)	7		ns	
T_{21}	Input Hold (AP-Bus)	10		ns	

NOTE:
1. "Controls" include all L-Bus I/O pins not otherwise specified.

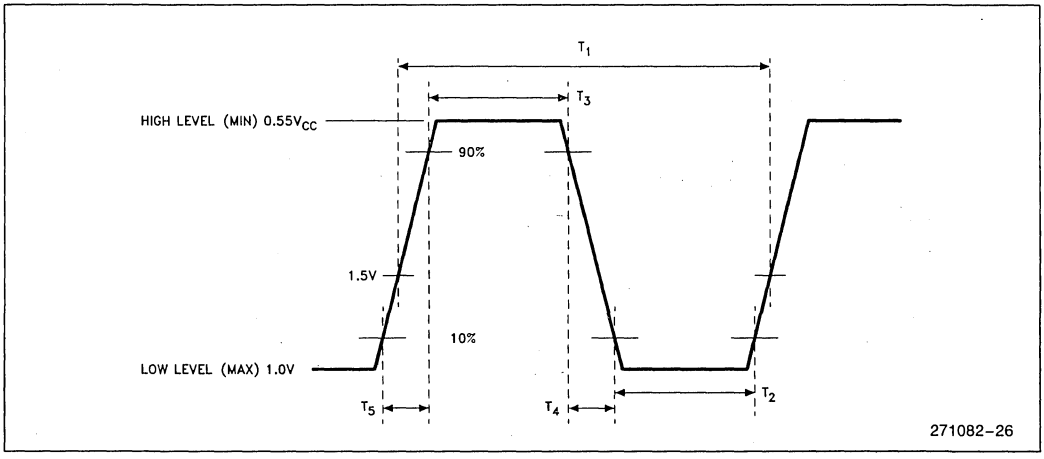
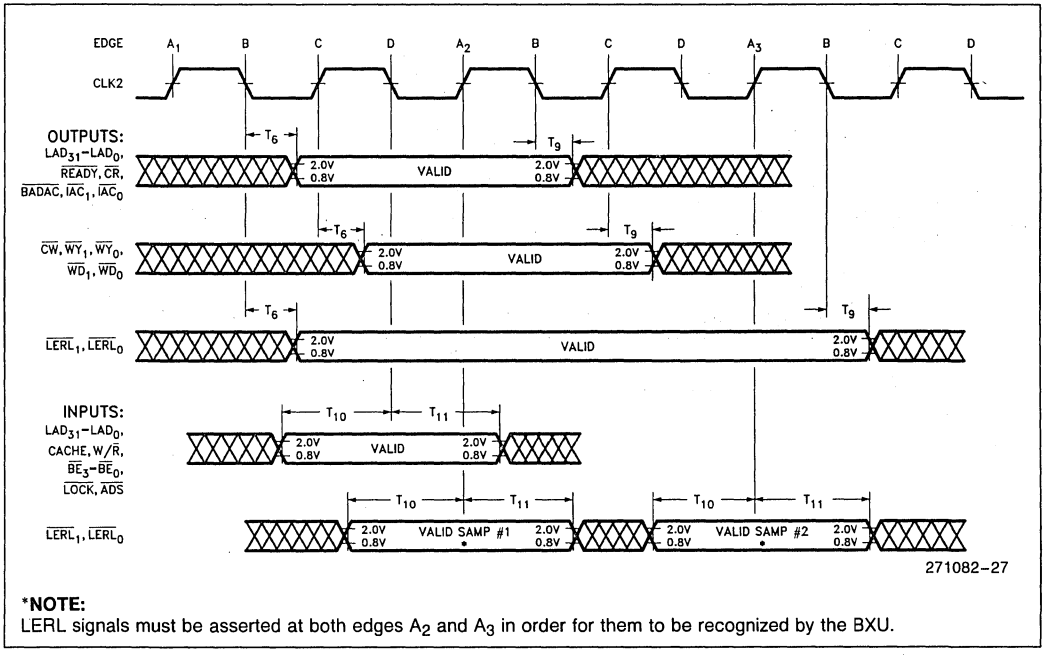
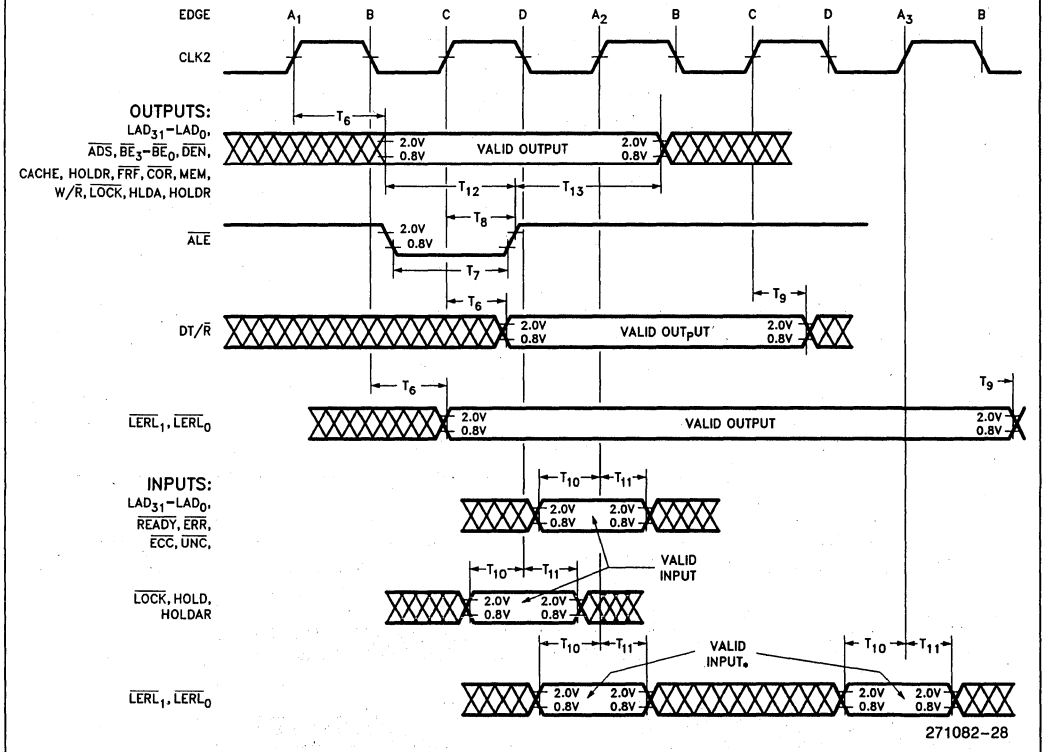


Figure 22. CLK2 Timing



***NOTE:**
LERL signals must be asserted at both edges A_2 and A_3 in order for them to be recognized by the BXU.

Figure 23. Drive Levels and Measurement Points for A.C. Specifications. L-Bus Timings for the BXU as a Bus Slave



***NOTE:**
LERL signals must be asserted at both edges A_2 and A_3 in order for them to be recognized by the BXU.

Figure 24. Drive Levels and Measurement Points for A.C. Specifications. L-Bus Timings for the BXU as a Bus Master

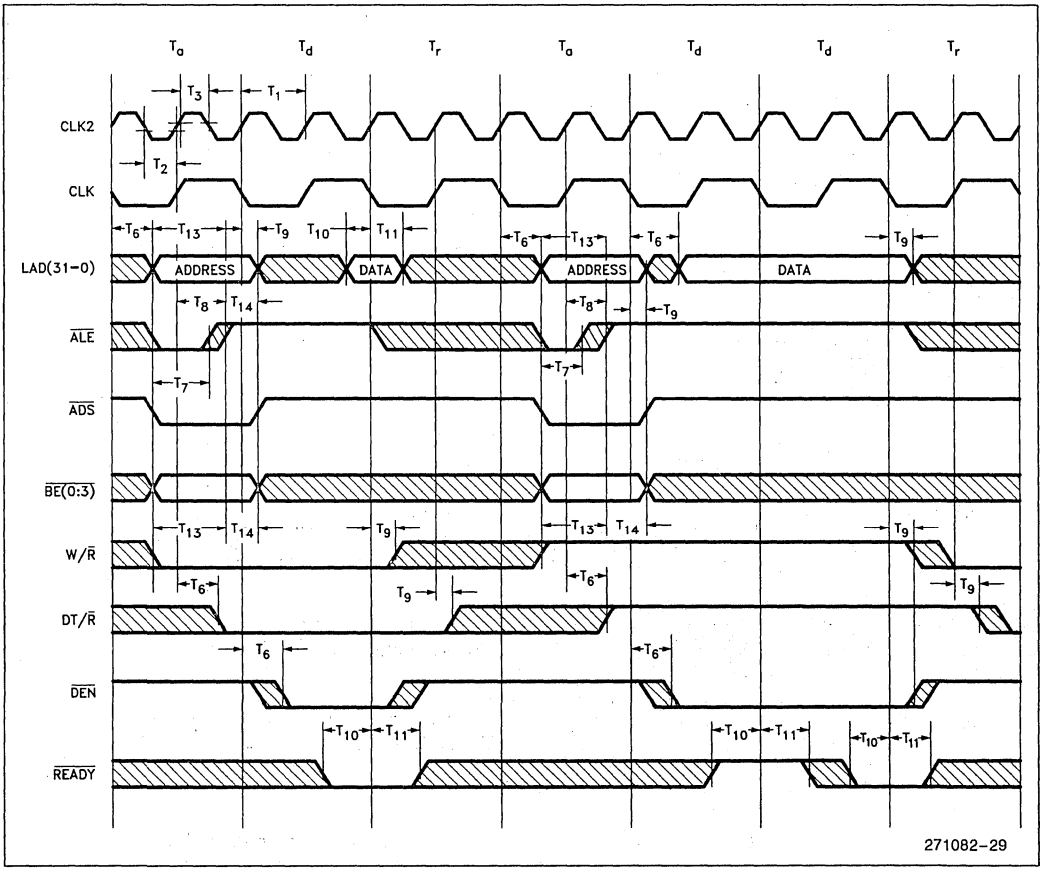
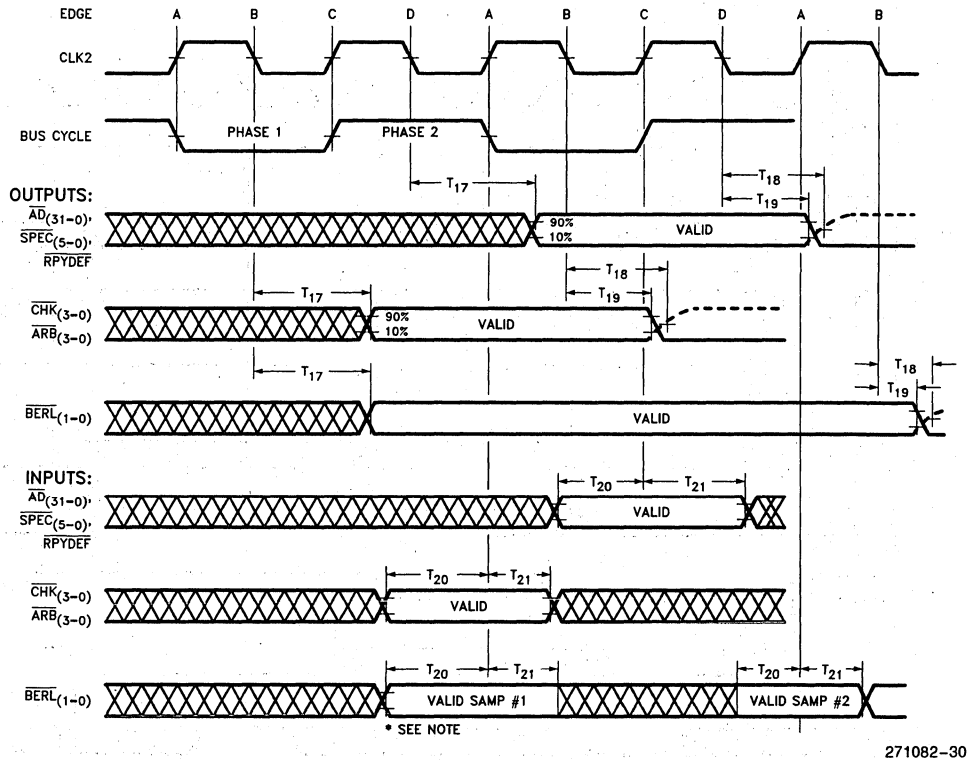


Figure 25. Relative Timing for L-Bus Signals



*NOTE:
BERL signals must be asserted at both edges A_2 and A_3 in order for them to be recognized by the BXU.

Figure 26. Relative Timing for AP-Bus Signals

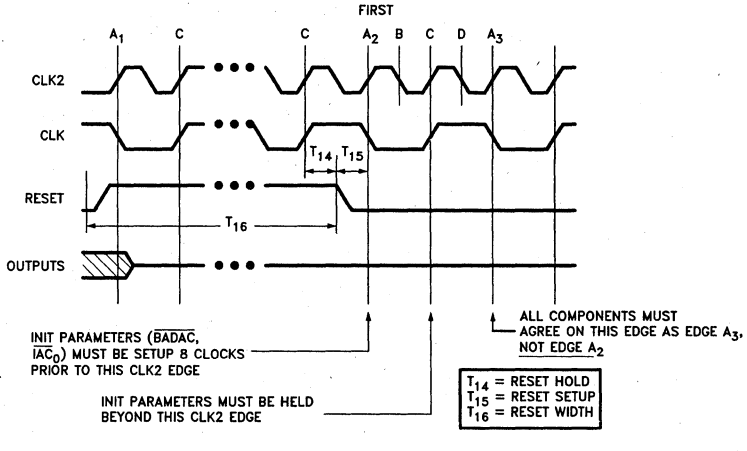


Figure 27. RESET Setup and HOLD Timing

L-BUS DESIGN CONSIDERATIONS

Input hold times can be disregarded by the designer whenever the input is removed because of a subsequent output from the BXU (e.g., $\overline{DEN}$ becomes deasserted). In other words, whenever the BXU generates an output that indicates a transition into a subsequent state, the BXU will have sampled any inputs for the previous state.

As an example, in the recovery (T_r) cycle following a read, the minimum time ($t_{6\ Min}$) that $\overline{DEN}$ becomes asserted is specified to be less than the minimum hold time on the data ($t_{11\ Min}$). When $\overline{DEN}$ is asserted, however, the data is guaranteed to have been sampled.

Similarly, whenever the BXU generates an output that indicates a transition to a subsequent state, any outputs that are specified to be tri-stated in this new state will be tri-stated.

For example, in the data (T_d) cycle following an address (T_a) cycle for a read, the minimum output delay ($t_{6\ Min}$) of $\overline{DEN}$ is specified to be less than the maximum float time of LAD ($t_{9\ Max}$). When $\overline{DEN}$ is asserted, however, the LAD outputs are guaranteed to have been tri-stated.

AP-BUS SIGNAL TIMING CONSIDERATIONS

The AP-Bus uses three-quarter cycle signaling for data transmission. Data is driven on edge D and sampled on edge C. This approach allows three-quarters of the bus cycle to be used for data transmission.

The remaining (one-quarter) time allows for clock skew and signal hold time. All AP-Bus signals except for the ARB, CHK, and BERL signals use this timing. The relationship of the AP-Bus signals is shown in Figure 28.

The CHK signals (interlaced parity) are delayed by one-half cycle or one phase to allow for generation of parity from the internal data that is being transmitted. The CHK lines are sampled one phase after the data has been sampled and compared against the parity generated for the received data.

Most input signals on the AP-Bus are sampled on the rising edge of CLK2 at edge C. The exceptions are the error signals CHK, BERL and ARB, which are sampled on the rising edge of CLK2 at edge A. Regardless of the edge, the setup and hold times are the same.

All outputs on the AP-Bus are driven relative to the falling edge of CLK2 at the middle of phase 2, except CHK, BERL and ARB, which transition on the falling edge of CLK2 at the middle of phase 1.

When designing a system based on the AP-Bus, the system topology will be limited by the available propagation time for signals in the system. The propagation time must allow for settling of ringing, ground shift, and crosstalk, all of which are dependent on board and system materials and design.

The following equation gives the propagation time available, given a specific clock implementation and frequency:

$$T_{PROP} = 2T_1 - (T_3 + T_4 + T_5 + (T_{18} \text{ or } T_{19}) + T_{10a} + T_{skew})$$

Where T_{skew} is the worst case clock skew between BXUs (clock skew is the time delay between any two clocks in the system due to physical distribution limits).

In AP-Bus systems, this skew is defined as follows:

$$T_{skew} \leq T_3 + T_{20} - T_{11}$$

L-Bus Waveforms

Figures 30 through 36 illustrate the relationship of L-Bus signals during a variety of bus transactions. For a detailed discussion of the operation of the L-Bus, consult the *80960MC Hardware Designer's Reference Manual.*

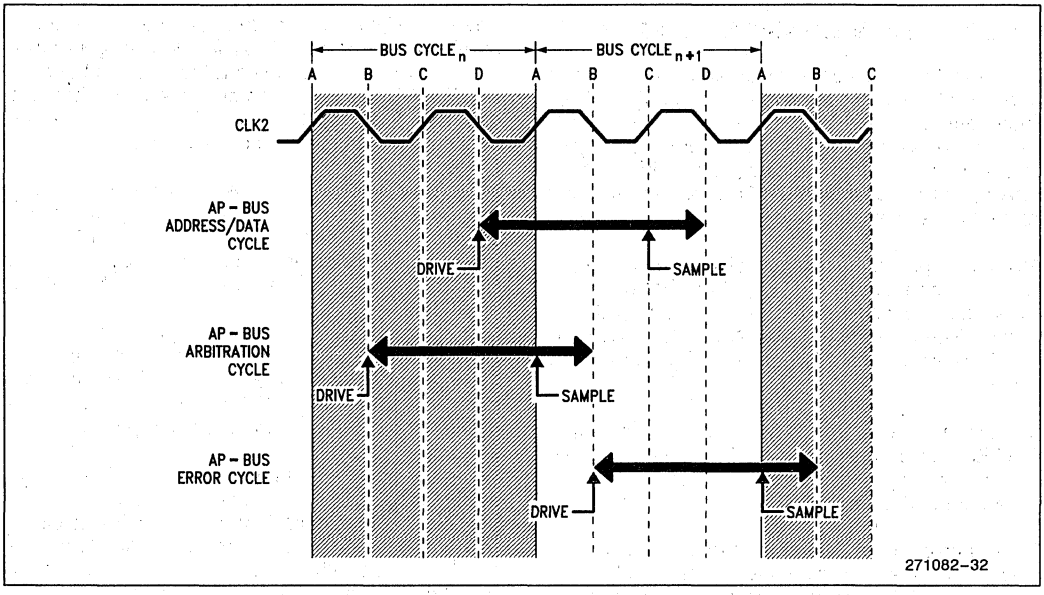


Figure 28. AP-Bus Signal Timing

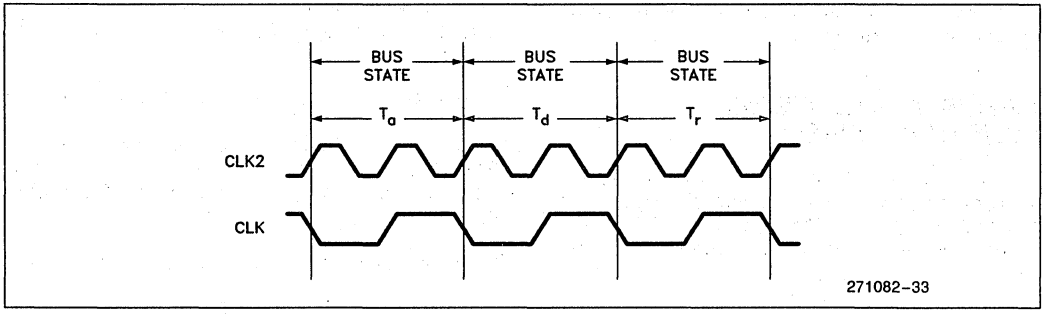


Figure 29. System and Processor Clock Relationship

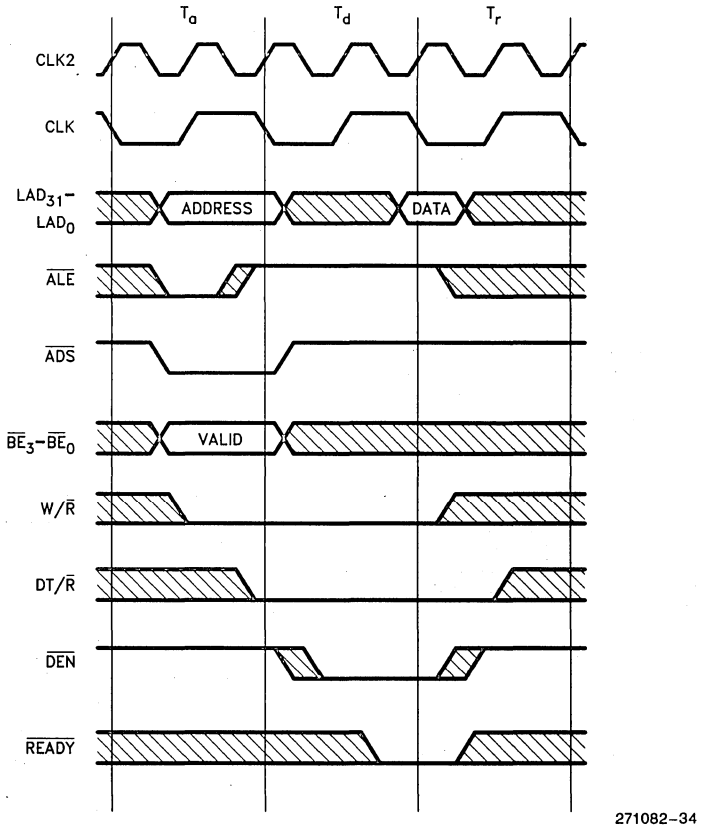


Figure 30. L-Bus Read Transaction

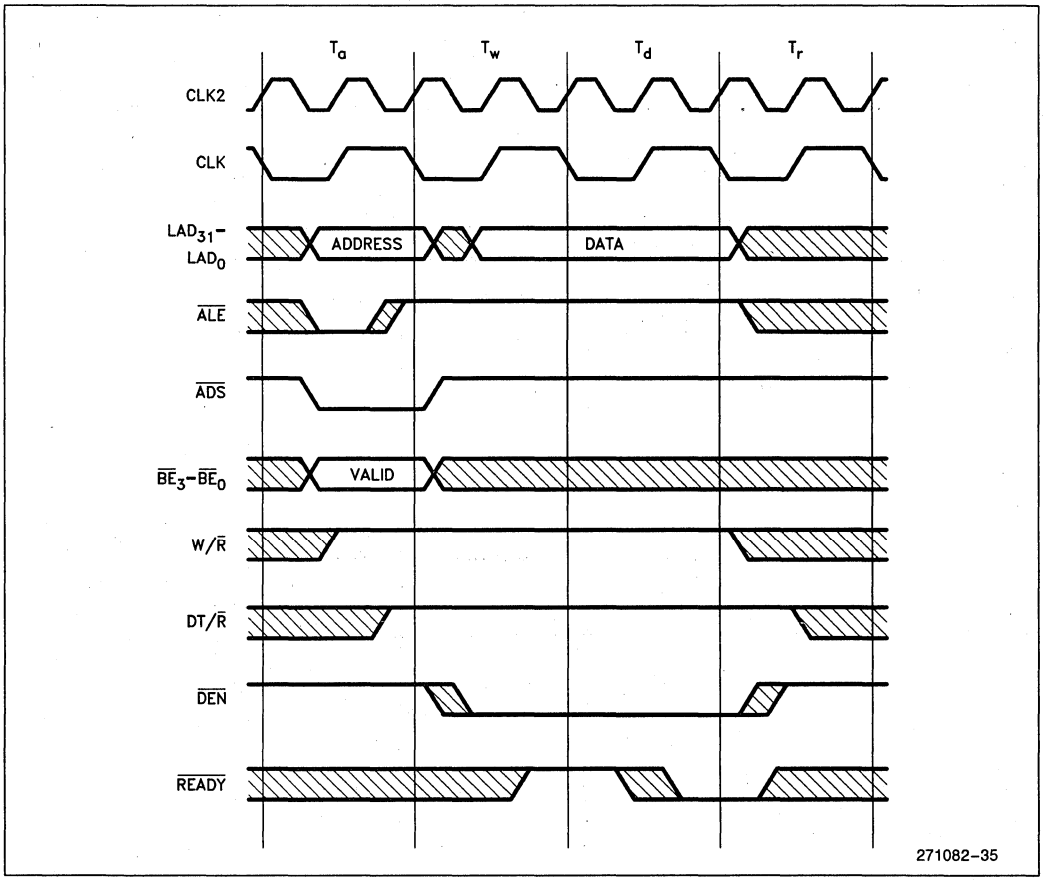


Figure 31. L-Bus Write Transaction

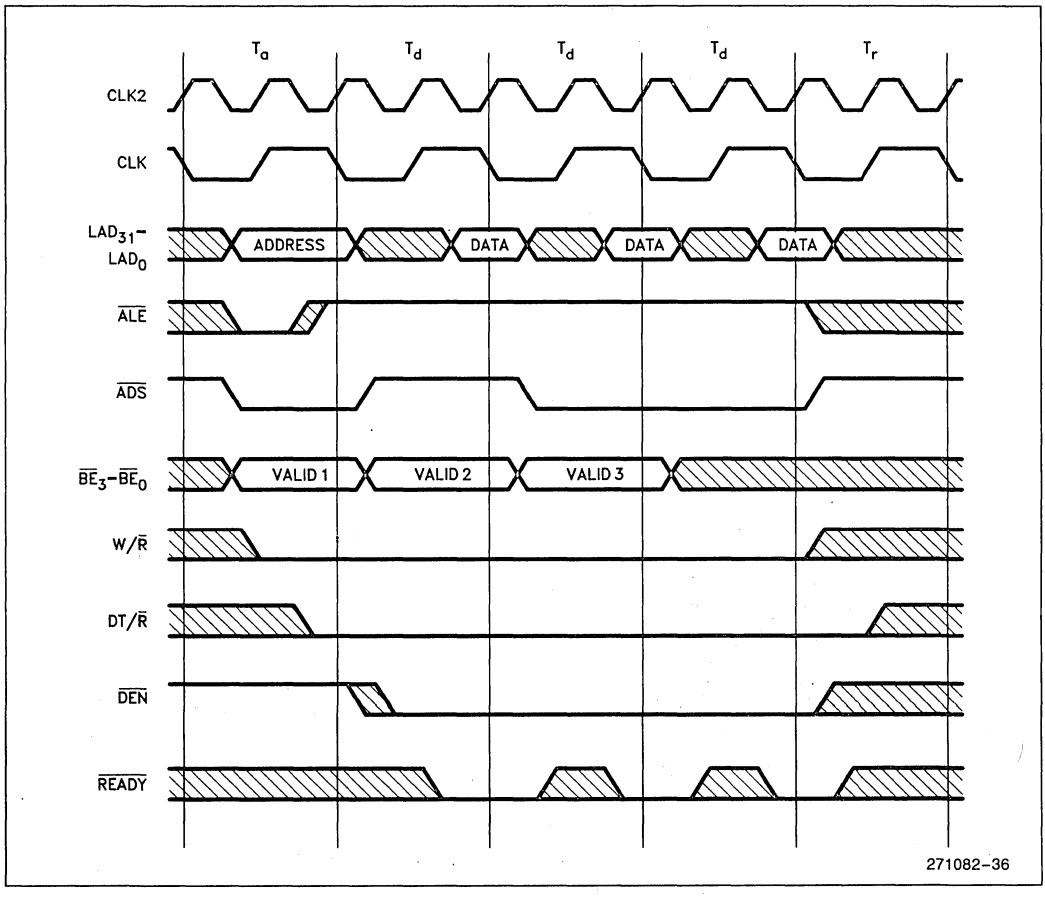


Figure 32. L-Bus Burst Read Transaction

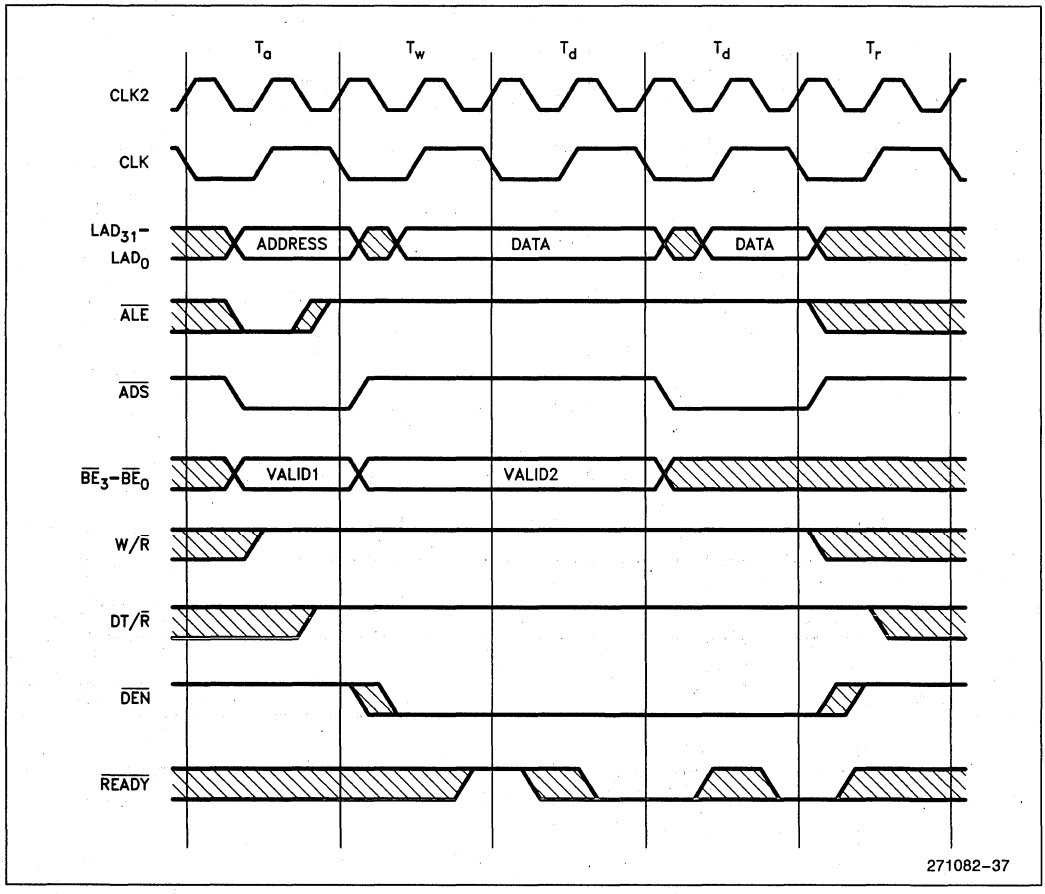
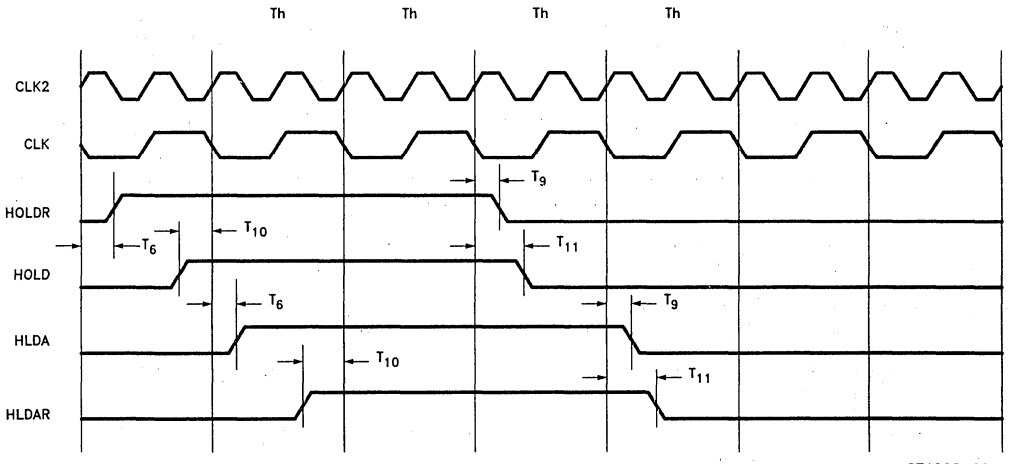
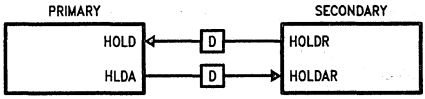


Figure 33. L-Bus Burst Write Transaction with One Wait State





DELAY OF 5 ns MINIMUM
IS REQUIRED

271082-39

Figure 34. Hold Timing

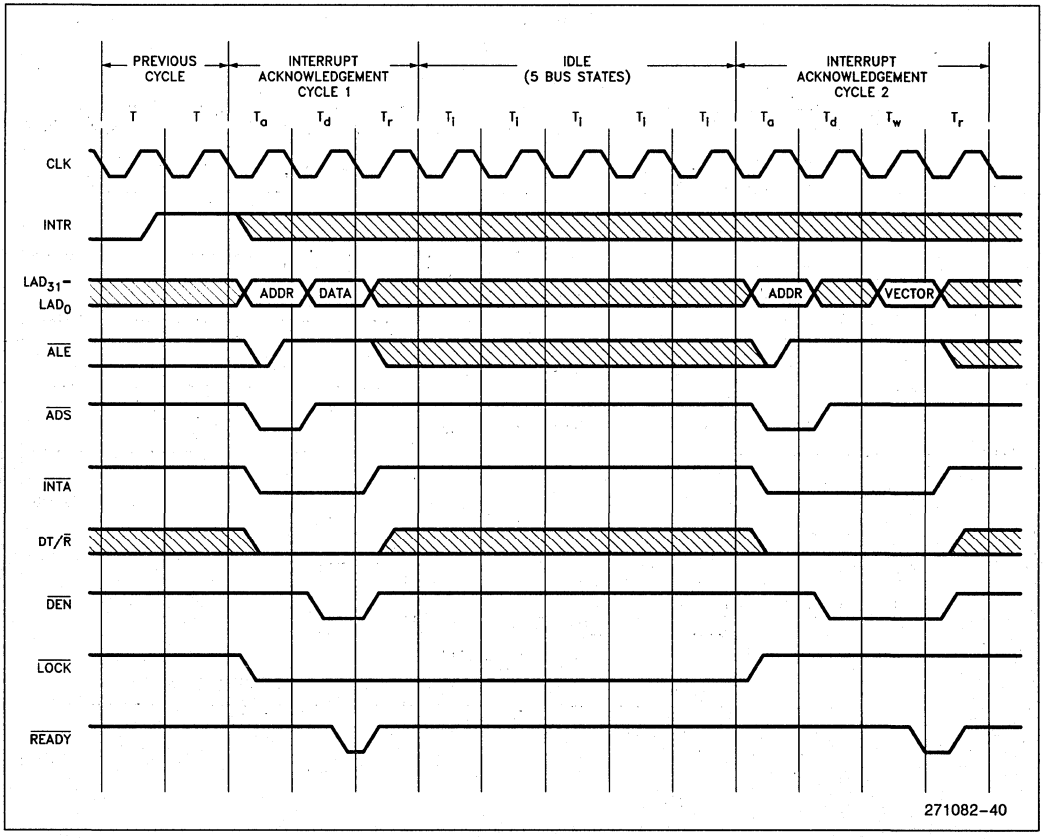


Figure 35. Interrupt Acknowledge Transaction

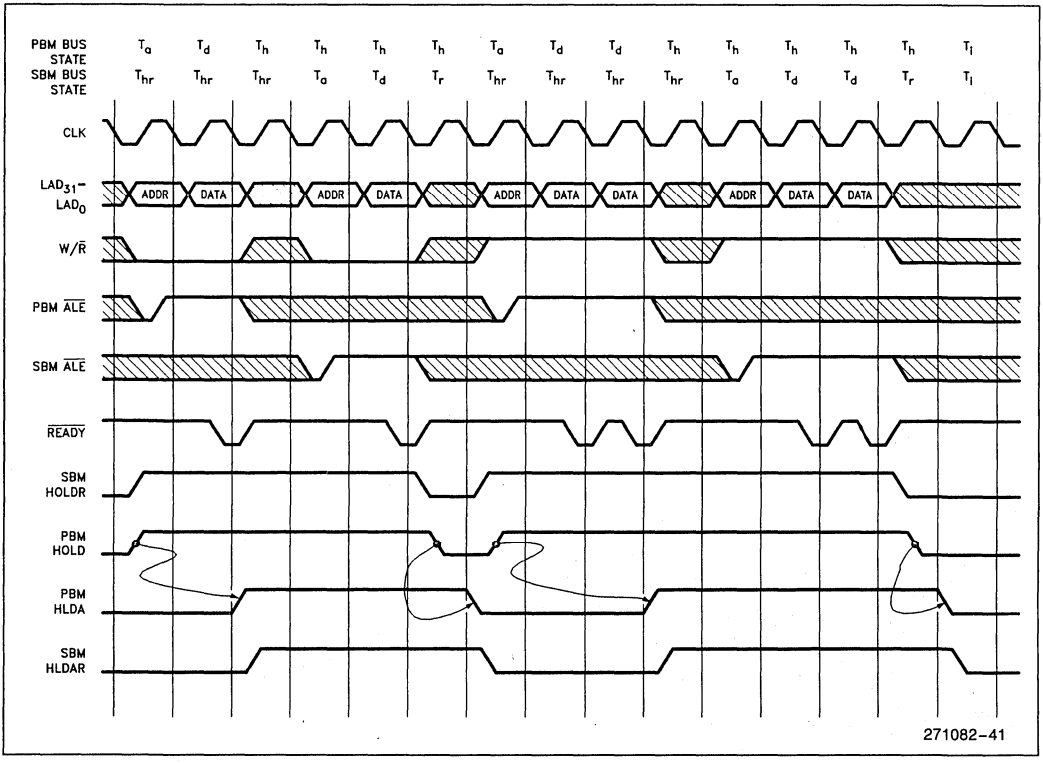


Figure 36. Bus Exchange Transaction (PBM = Primary Bus Master, SBM = Secondary Bus Master)

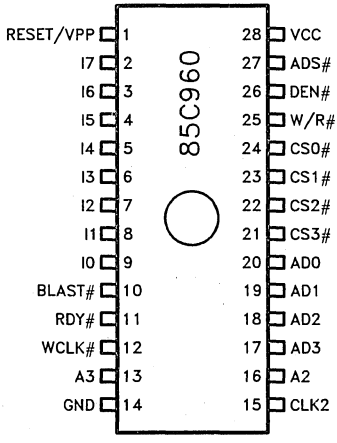
Memories and Peripherals

4

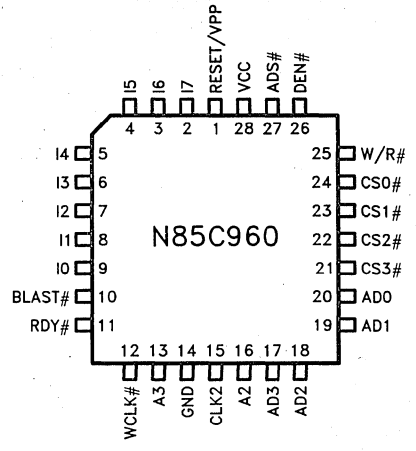
85C960
1-MICRON CHMOS
80960 K-SERIES BUS CONTROL μPLD

- Burst Logic, Ready Control, and Address Decode Support for 80960 KA/KB Embedded Controllers in Single Chip
- Burst Logic Supports Both Standard and New Generation "Burst Mode" Memories and Peripherals
- Ready/Timing Control Supports 0–15 Wait States across 8 Address Ranges, Read/Write Accesses, Burst Transactions
- 8 Dedicated Inputs Decoded into 8 Latched Chip Selects (4 External/ Internal; 4 Internal Only)
- Operates with 80960KA/KB at 20 MHz and 25 MHz
- I_{CC} = 50 mA Max.
- UV Erasable (CerDIP) or OTP™
- 100% Generically Testable Logic Array
- Based on Low Power CHMOS IIIE* Technology
- Available in 28-Pin 300-mil CerDIP and PDIP Packages and in 28-Pin PLCC Package
  (See Packaging Spec., Order Number #231369)

*CHMOS is a patented technology of Intel Corporation.



290192-1



290192-2

= Active Low Signals

Figure 1. Pinout Diagram

August 1990
Order Number: 290192-002

GENERAL DESCRIPTION

The Intel 85C960 is a single-chip burst/ready/decode μPLD (Microcomputer Programmable Logic Device) designed to interface 80960 KA/KB embedded controllers to system memory and I/O. The 85C960 provides programmable chip selects, a programmable read/write access wait state/ready generator, and burst address (A2, A3) cycling. Burst transaction cycling of A2, A3, and WCLK# (Write Clock) is also supported for intelligent peripherals on the bus.

For its programmable functions, the 85C960 uses advanced EPROM cells as logic array and wait-state table memory elements. Coupled with Intel's proprietary CHMOS IIIE technology, the result is a programmable device able to support Intel's 32-bit 80960 KA/KB embedded controllers at speeds up to 25 MHz.

ARCHITECTURE DESCRIPTION

The 85C960 μPLD integrates burst control, ready generation, and chip select decoding into a single device. Figure 2 shows the architecture of the 85C960. Table 1 lists and describes each signal on the device. The 85C960 replaces 6–10 separate PLD/discrete logic devices in small- and medium-sized 80960 systems. For medium- to large-sized systems, the 85C960 can be supplemented with an additional decoder, such as the 85C508, and a second 85C960. Figure 3 shows a single 85C960 in a typical application.

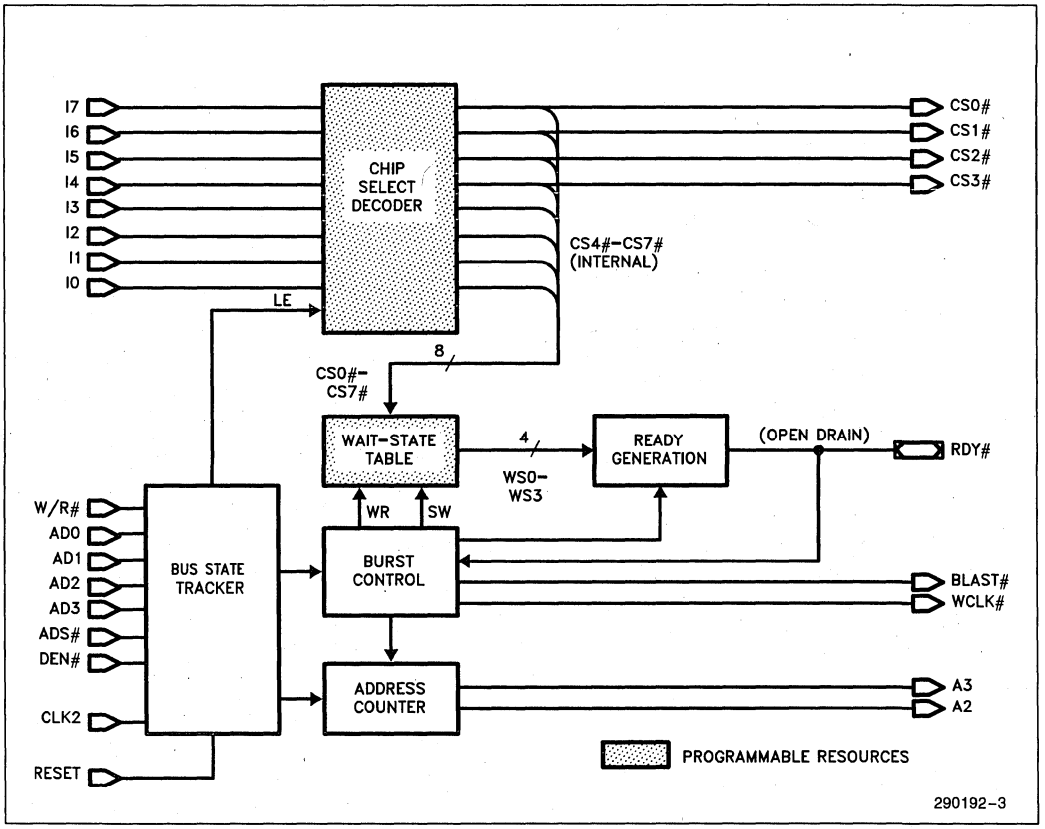


Figure 2. 85C960 Block Diagram

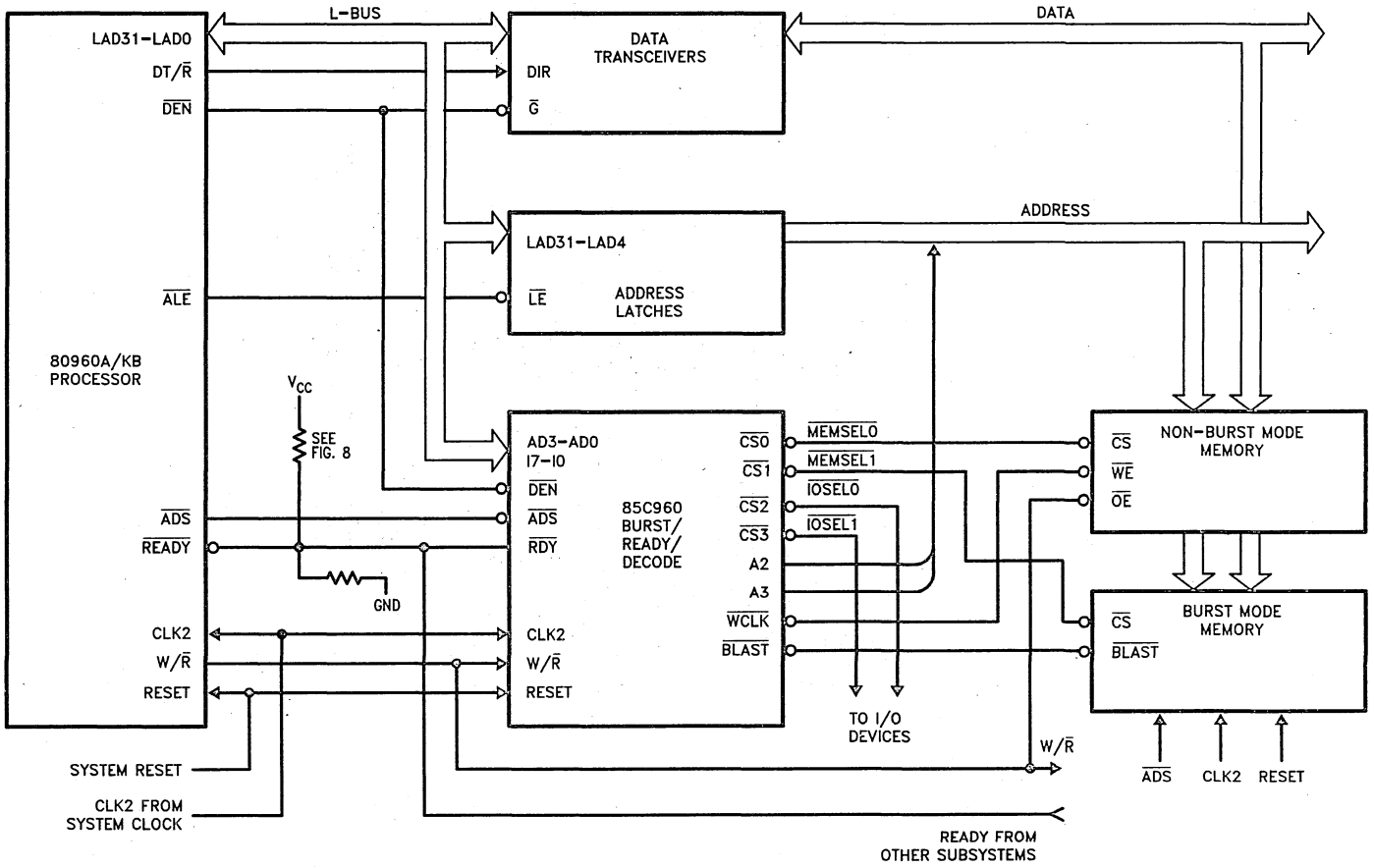


Figure 3. 85C960 in an 80960 System

Table 1. 85C960 Pin Descriptions

Symbol	Type	Name and Function
RESET	I	**RESET.** When RESET is high for a minimum of four CLK2 cycles, internal circuits are reset to a known state.
I7–I0	I	**INPUT 7–INPUT 0.** These are the address range inputs to the programmable decode logic array.
CLK2	I	**SYSTEM CLOCK.** This input, which connects to the 80960 CLK2 signal, provides the timing reference for all 85C960 operations.
AD3–AD0	I	**ADDRESS IN 3–ADDRESS IN 0.** These inputs are driven by LAD0–LAD3 from the Local Bus (L-Bus) to provide addressing and burst access decode information.
W/R#	I	**WRITE/READ.** Write/Read from controller. When low, indicates that the current access is a read. When high, indicates that the current access is a write.
DEN#	I	**DATA ENABLE.** This input from the controller indicates that data is present on the L-Bus.
ADS#	I	**ADDRESS/DATA STROBE.** This input from the 80960 indicates whether address or data information is currently on the L-Bus. When low, address information is changing. The 85C960 chip select timing is based in part on ADS# low during Ta states.
BLAST#	O	**BURST LAST.** This signal, when low, indicates that the current read/write access is the last access in a burst transaction. BLAST# is not cycled if RDY# is generated off-chip.
WCLK#	O	**WRITE CLOCK.** This output provides a write enable strobe to memories that do not support burst mode access.
A3, A2	O	**ADDRESS OUT 3, 2.** These outputs cycle during burst transactions. Typically connected to lowest memory address signals.
CS3#–CS0#	O	**CHIP SELECT 3–CHIP SELECT 0.** Single p-term select outputs that are driven active (low) for the programmed address condition on I7–I0.
RDY#	I/O	**READY.** RDY# is an active low, bidirectional, open-drain signal that should be connected to the controller's Ready input. As an output, RDY# goes high to cause the controller to extend the current access. RDY# goes low to indicate that the data on the L-Bus bus may be sampled (read) or removed (write). RDY# is controlled by the 85C960 Ready Generation and Wait-State Logic. The open-drain output allows RDY# to be OR-tied to other circuitry that may drive the controller's Ready input. As a bidirectional input, RDY# allows the 85C960 to provide Ready timing and burst cycling for intelligent peripherals that do not generate these signals themselves.

80960 L-Bus (Local Bus) cycles are monitored by the **Bus State Tracker** to synchronize the functional blocks in the 85C960 to the L-Bus. CLK2 provides the timing reference for all 85C960 operations.

Four external chip selects (CS0#–CS3#) are generated by the programmable **Chip Select Decoder.** These four signals provide decoded selects to memory and I/O devices and are routed to the programmable **Wait-State Table** so that the 85C960 can generate RDY# at the appropriate time. Four additional selects are decoded (internal only) and routed to the Wait-State Table so that the 85C960 can generate RDY# for up to four additional address ranges.

The **Ready Generation** block generates RDY# to the controller under control of the **Wait-State Table.** Depending on the contents programmed into this table and the current type of access, from 0–15 wait states can be introduced into each bus cycle. An independent wait state value can be chosen for each select and each access type. Four access types are possible: read first, read subsequent, write first, and write subsequent.

The **Burst Control** and **Address Counter** blocks control burst transaction timing to memory and I/O. Note that the RDY# pin is sampled by the Burst Control block to allow the 85C960 to generate burst transaction timing for other bus peripherals. WCLK# provides a write enable strobe for memory and I/O that do not support burst mode. BLAST# informs burst-mode devices that the current access is the last one in a burst transaction. A2 and A3 are cycled to select the address location for each access.

FUNCTIONAL DESCRIPTION

The following paragraphs provide a detailed description of each functional block in the 85C960 μPLD.

Chip Select Decoder

The Chip Select Decoder, shown in Figure 4, is a high speed, single p-term (product-term) latched decoder circuit with eight inputs (I0–I7) and eight latched outputs. Each output goes low when its associated product term is true. Four of these outputs (CS0#–CS3#) are available externally to be used as device selects. The remaining four outputs (CS4#–CS7#) are available internally so that the 85C960 can provide ready and burst timing for four more device selects. (The actual selects for these four additional devices/resources must be generated by external logic.)

The input to each latch is a single NAND p-term that can be connected to the dedicated inputs. The true and complements of all inputs (I7–I0) are available to all eight NAND p-terms.

Each intersecting point in the logic array is connected or not connected based on the value programmed in the EPROM array. Initially (EPROM erased state), no connections exist between any p-term and any input. Connections can be made by programming the appropriate EPROM cells. Since p-terms are implemented as NANDs, a true condition on a p-term drives the output low. Current consumption is higher when both true and complement p-terms for the same input are programmed.

Selects are latched on the falling edge of an internal Latch Enable (LE), which is generated from ADS#, DEN#, and CLK2. The proper combination of these signals occurs during an 80960 address state (Ta). Figure 5 shows the relationship of the internal LE and external chip selects to the three signals at the end of a Ta state. All selects are cleared to an inactive high state at the start of a recovery state. (Tr). All eight selects (four external and four internal) are routed to the Wait-State Table.

Wait State Table

Chip selects, WR (Write/Read), and SW (Subsequent Word) feed the Wait-State Table. Each chip select points to a set of four wait state values while WR and SW determine which of the four values to route to the Ready Generation block (see Figure 6). The four values are grouped into read and write groups with each group having a value for the first access and subsequent access (second through fourth). The four-bit wait-state value is sent to the Ready Generation block (via WS0#–WS3#) to be used as an initial count value. If two selects are active, the resulting count value is the logical bit AND of the two individual values. If more than two selects are active and the individual count values are not the same, the resulting count value is indeterminate. If no select is active, no count value is loaded (and the Ready Generation circuit is disabled).

Ready Generation

RDY# is high at the start of each burst transaction. The RDY Generator begins to count down from the wait state value, decrementing the counter at the start of each wait state. When the internal counter reaches 0000, RDY# is pulled low (CLK2c during the data state). On the next CLK2c edge (for a wait state), RDY# is released, allowing an external resistor to pull RDY# high. Figure 7 shows the timing for a four-word burst write transaction with 1 wait state for the first access and 0 wait states for the remaining three accesses (Burst Write 1-0-0-0).

RDY# is an open-drain I/O pin, which must be connected to pullup and pulldown resistors as shown in Figure 8. During a wait-state access, RDY# is pulled high to cause the controller to extend the current access so that the memory or peripheral chip has time to present data to the bus (read), or sample data on the bus (write). RDY# is released on the CLK2a edge of a Tr state. If a Read or Write access occurs without a chip select having been decoded on-chip, the RDY# output buffer is disabled and RDY# is sampled as an input. This allows the 85C960 to cycle A2, A3, and WCLK# to provide burst transaction timing for other bus controllers. RDY# may be OR-tied with other bus controllers so they can access the processor Ready signal.

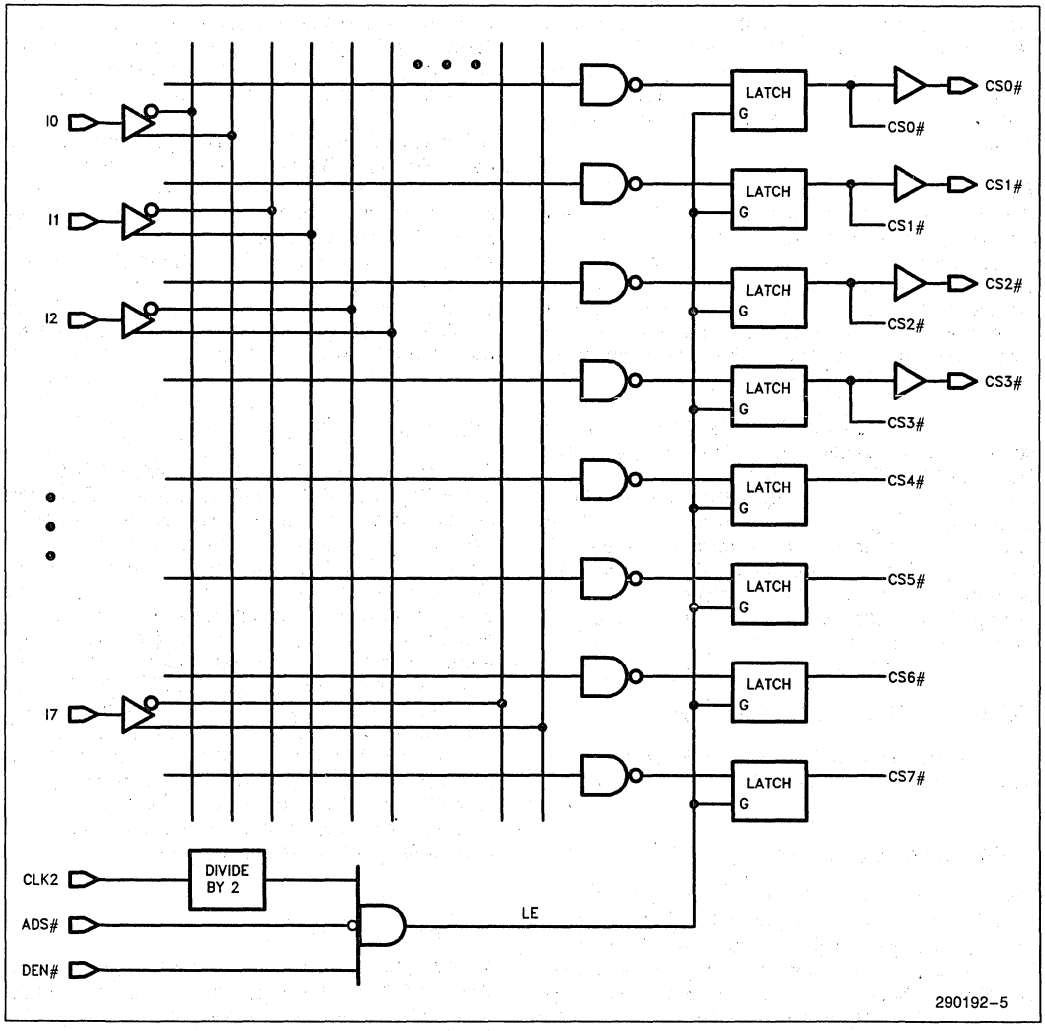
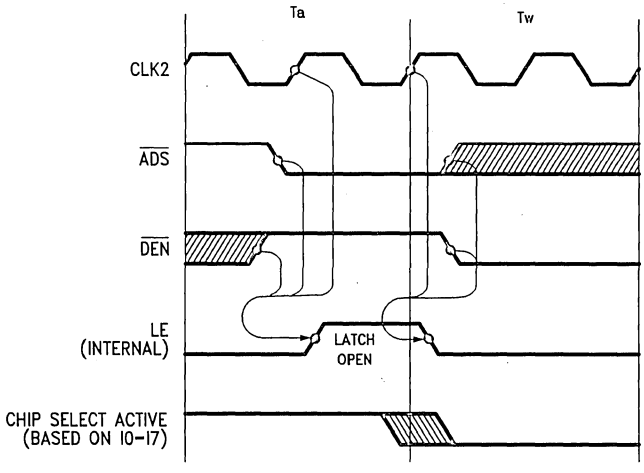


Figure 4. 85C960 Chip Select Decoder Block



Latch opens when CLK2 and DEN# go high and ADS# goes low.
Latch closes when DEN# goes low or ADS# or CLK2 go high.

Figure 5. Internal LE and External Chip Select Timing

Burst Transactions

AD3, AD2 are latched to indicate the starting address of a burst transaction. The 85C960 places these two signals out on A3 and A2, respectively, then cycles the two addresses upward until the last access of the burst. The 85C960 assumes that the processor handles splitting of the burst transaction when a 16-byte boundary is crossed.

AD0 and AD1 specify the size of the burst transfer in double-words as shown in Table 2.

Table 2. AD0–AD1 vs Burst Size

AD1	AD0	No. of Words Transferred
0	0	1
0	1	2
1	0	3
1	1	4

WCLK#, BLAST# Generation

WCLK# is the write enable signal for writing to non-burst mode memories. When low, address outputs A2 and A3 are valid. Its trailing edge (low-to-high transition) can be used to latch data into non-burst mode memories. WCLK# is only provided during writes; during reads, WCLK# remains high.

BLAST# indicates that the current access is the last access in a burst transaction. BLAST# is used by burst-mode memories to reset internal address counters. BLAST# is not cycled when RDY# is generated off-chip.

POWER-ON CHARACTERISTICS

85C960 inputs and outputs begin responding 1 μs (max.) after V_{CC} power-up (V_{CC} = 4.75V) or after a power-loss/power-up sequence. RESET must be synchronous to CLK2 and must be held high for a minimum of 4 clock cycles after V_{CC} reaches 4.75 V. After 4 clock cycles, A2 and A3 are high, CS0# – CS3# (and CS4# – CS7#), BLAST#, WCLK# are high, and the open drain RDY# signal is inactive.

Select CS0f#	Write/Read	
	WR = 0 (Read)	WR = 1 (Write)
SW=0 (First Word)	msb lsb 0000	msb lsb 0000
SW=1 (Subsequent Word)	msb lsb 0011	msb lsb 0010

msb = most significant bit
lsb = least significant bit

Figure 6. Example Wait-State Entries for CS0f#

ERASURE CHARACTERISTICS

Erasure time for the 85C960 is 20 minutes at 12,000 μWsec/cm^2 with a 2537Å UV lamp.

Erasure characteristics of the device are such that erasure begins to occur upon exposure to light with wavelengths shorter than approximately 4000Å. It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000Å–4000Å range. Data shows that constant exposure to room level fluorescent lighting could erase the typical 85C960 in approximately two years, while it would take approximately two weeks to erase the device when exposed to direct sunlight. If the device is to be exposed to these lighting conditions for extended periods of time, conductive opaque labels should be placed over the device window to prevent unintentional erasure.

The recommended erasure procedure for the 85C960 is exposure to shortwave ultraviolet light with a wavelength of 2537Å. The integrated dose (i.e., UV intensity $\times$ exposure time) for erasure should be a minimum of fifteen (15) Wsec/cm^2. The erasure time with this dosage is approximately 20 minutes using an ultraviolet lamp with a 12,000 μW/cm^2 power rating. The device should be placed within 1 inch of the lamp tubes during exposure. The maximum integrated dose the 85C960 can be exposed to without damage is 7258 Wsec/cm^2 (1 week at 12,000 μW/cm^2). Exposure to high intensity UV light for longer periods may cause permanent damage to the device.

LATCH-UP IMMUNITY

All of the input, output, and clock pins of the device have been designed to resist latch-up which is inherent in inferior CMOS processes. The 85C960 is designed with Intel's proprietary 1-micron CHMOS EPROM process. Thus, each of the pins will not experience latch-up with currents up to $\pm$100 mA and voltages ranging from -0.5V to (V_{CC} + 0.5V). The programming pin is designed to resist latch-up to the 13.5V max. device limit.

DESIGN RECOMMENDATIONS

For proper operation, it is recommended that all input and output pins be constrained to the voltage range GND $\leq$ (V_{IN} or V_{OUT}) $\leq$ V_{CC}. All unused inputs should be tied high or low to minimize power consumption (do not leave them floating). Unused outputs may be left floating. A high-speed ceramic decoupling capacitor of at least 0.2 μF must be connected directly between the V_{CC} and GND pin.

As with all CMOS devices, ESD handling procedures should be used with the 85C960 to prevent damage to the device during programming, assembly, and test.

FUNCTIONAL TESTING

Since the programmable sections of the 85C960 are controlled by EPROM elements, the device is completely testable during the manufacturing process. Each programmable EPROM bit controlling the internal logic is tested using application independent test patterns. EPROM cells in the device are 100% tested for programming and erasure. After testing, the devices are erased before shipments to the customers. No post-programming tests of the EPROM array are required.

The testability and reliability of EPROM-based programmable logic devices is an important feature over similar devices based on fuse technology. Fuse-based programmable logic devices require a user to perform post-programming tests to insure device functionality. During the manufacturing process, tests on fuse-based parts can only be performed in very restricted ways in order to avoid pre-programming the array.

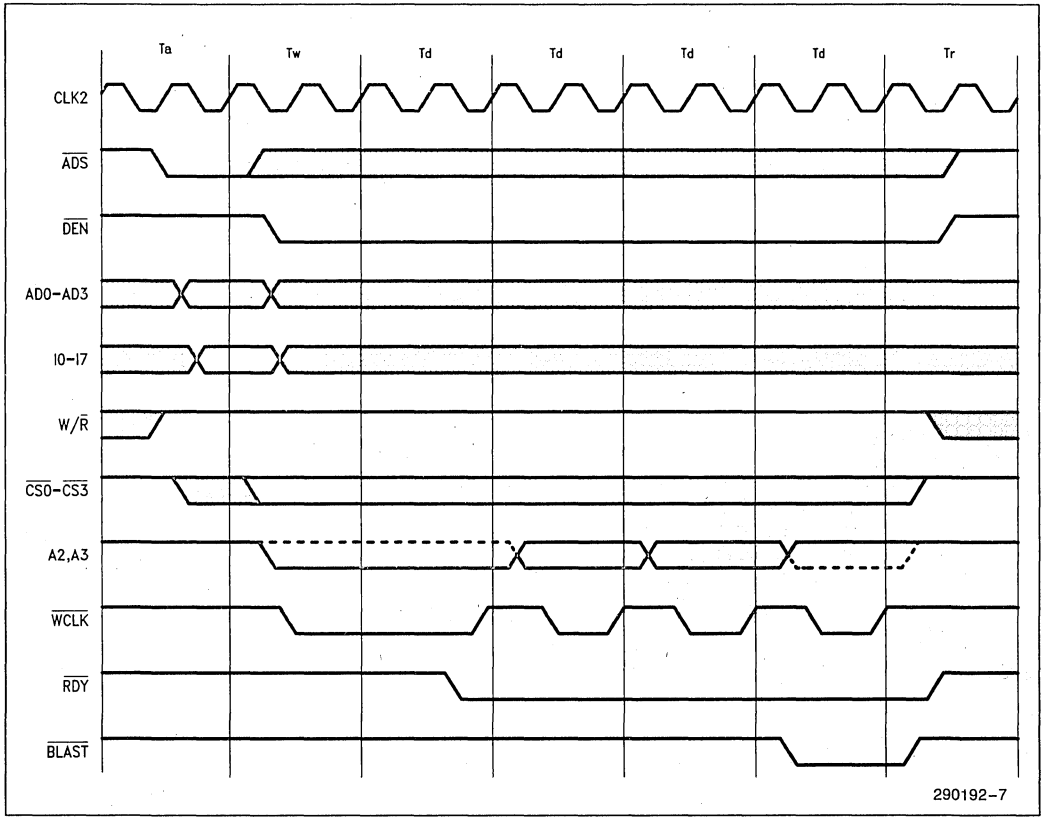


Figure 7. Burst Write Transaction (1-0-0-0)

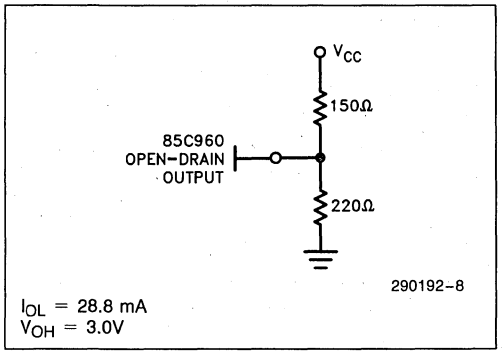


Figure 8. RDY# Pullup/Pulldown Resistors

IN-CIRCUIT RECONFIGURATION

The 85C960 allows in-circuit configuration changes after the device has powered up. At power-up, the device is configured according to the information programmed into the EPROM cells. After power-up, new information can be shifted in on select pins to alter device configuration. The new configuration is retained until the device is powered down or until the information is overwritten by another configuration change.

Note that in-circuit configuration changes allow "on-the-fly" changes to be made, but do not alter EPROM cell data. At the next power-up, the device will be configured according to the original data programmed into the EPROM cells. In-circuit reconfiguration requires additional circuitry external to the 85C960. For details on in-circuit configuration changes, refer to AP-337, *In-Circuit Reconfiguration of 85C960 and 85C508 μPLDs,* order number 292072.

DESIGN SOFTWARE

Software support is provided by version 2.1 (or later) of iPLS II (Intel Programmable Logic Software II). Programming is supported on the iUP-PC PC-based programmer or iUP-200A/201A Universal Programmer via the GUPI base module and the GUPI 85EPLD28 programming adaptor.

For detailed information on iPLS II, refer to the iPLDS II Data Sheet, order number: 290134. The tools section of the *Programmable Logic* handbook contains a complete listing of all design tools for Intel EPLDs.

ORDERING INFORMATION

80960KA/KB Clock Frequency	μPLD Order Code	Package	Operating Range
20 MHz	*D85C960-20	CERDIP	Commercial
	N85C960-20	PLCC	
25 MHz	*D85C960-25	CERDIP	Commercial
	N85C960-25	PLCC	

*Only windowed CERDIP allows UV-erase.

ABSOLUTE MAXIMUM RATINGS*

Supply Voltage (V_{CC})(1) ..... −2.0V to +7.0V

Programming Supply Voltage (V_{PP})(1) ..... −2.0V to +13.5V

D.C. Input Voltage (V_I)(1, 2) ... −0.5V to V_{CC} + 0.5V

Storage Temperature (T_{stg}) ..... −65°C to +150°C

Ambient Temperature (T_A)(3) ..... −10°C to +85°C

NOTES:
1. Voltages with respect to GND.
2. Minimum D.C. input is −0.5V. During transitions, the inputs may undershoot to −2.0V or overshoot to +7.0V for periods of less than 20 ns under no load conditions.
3. Under bias. Extended Temperature versions are also available.

NOTICE: This is a production data sheet. The specifications are subject to change without notice.

**WARNING: Stressing the device beyond the "Absolute Maximum Ratings" may cause permanent damage. These are stress ratings only. Operation beyond the "Operating Conditions" is not recommended and extended exposure beyond the "Operating Conditions" may affect device reliability.*

RECOMMENDED OPERATING CONDITIONS

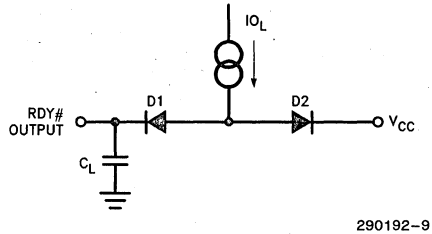
Symbol	Parameter	Min	Max	Units
V_{CC}	Supply Voltage	4.75	5.25	V
V_{IN}	Input Voltage	0	V_{CC}	V
V_O	Output Voltage	0	V_{CC}	V
T_A	Operating Temperature	0	+70	°C

D.C. CHARACTERISTICS (T_A = 0°C to +70°C, V_{CC} = 5.0V ± 5%)

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
V_{IH1}(4)	High Level Input Voltage (All Inputs except for ADS#, AD0–AD3, DEN#, and W/R#)	2.0		V_{CC} + 0.3	V	
V_{IH2}(4)	High Level Input Voltage for ADS#, AD0–AD3, DEN#, and W/R#	2.2			V	
V_{IL}(4)	Low Level Input Voltage	−0.3		0.8	V	
V_{OH}	High Level Output Voltage	2.4			V	I_{OH} = −4.0 mA D.C., V_{CC} = Min.
V_{OL1}	Low Level Output Voltage			0.4	V	I_{OL} = 4.0 mA D.C., V_{CC} = Min., C_L = 30 pF
V_{OL2}	Low Level Output Voltage for A2, A3			0.45	V	I_{OL} = 24 mA D.C., V_{CC} = Min., C_L = 60 pF
V_{OL3}	Low Level Output Voltage for Open Drain (RDY#)			0.5	V	I_{OL} = 30 mA D.C., V_{CC} = Min., C_L = 30 pF
I_I	Input Leakage Current			±10	μA	V_{CC} = Max., GND ≤ V_{IN} ≤ V_{CC}
I_{OZ}	Output Leakage Current			±10	μA	V_{CC} = Max., GND ≤ V_{OUT} ≤ V_{CC}
I_{SC}(5)	Output Short Circuit Current	−30		−90	mA	V_{CC} = Max., V_{OUT} = 0.5V
I_{CC}	Power Supply Current		10	50	mA	V_{CC} = Max., V_{IN} = V_{CC} or GND, No Load, CLK2 = 50 MHz

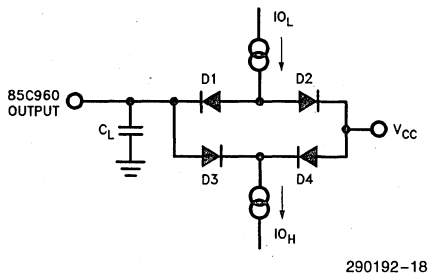
NOTES:
4. Absolute values with respect to device GND; all over and undershoots due to system or tester noise are included.
5. Not more than 1 output should be tested at a time. Duration of that test should not exceed 1 second.

A.C. TESTING LOAD CIRCUIT (RDY#)



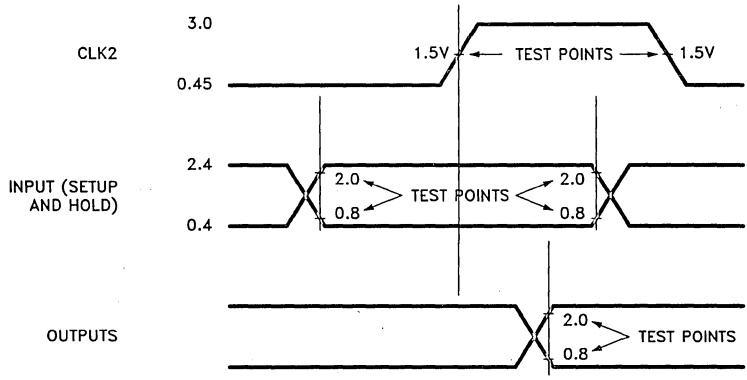
See D.C. Characteristics Table for Current and Capacitance Specifications.
D1 and D2 are matched.

A.C. TESTING LOAD CIRCUIT (ALL OUTPUTS EXCEPT RDY#)



See D.C. Characteristics Table for Current and Capacitance Specifications.
D1 and D2 are matched
D3 and D4 are matched

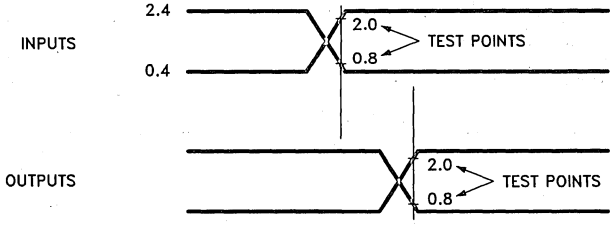
A.C. TESTING WAVEFORM—SYNCHRONOUS INPUTS AND OUTPUTS



290192-10

A.C. Testing: Inputs are driven at 2.4V for a Logic "1" and 0.4V for a Logic "0". CLK2 is driven at 3.0V for a Logic "1" and 0.45V for a Logic "0". Timing Measurements made relative to CLK2 are made from 1.5V on CLK2. Inputs and outputs are measured at 2.0V for a high and 0.8V for a low. Device input rise and fall times are less than 3 ns.

A.C. TESTING WAVEFORM—ASYNCHRONOUS INPUTS AND OUTPUTS



290192-11

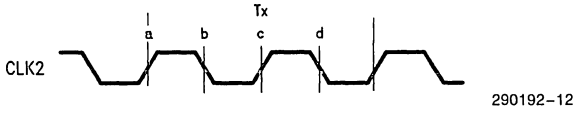
A.C. Testing: Inputs are driven at 2.4V for a Logic "1" and 0.4V for a Logic "0". Input timing is measured at 1.5V for high-to-low and low-to-high transitions. Outputs are measured at 2.0V for a high and 0.8V for a low. Device input rise and fall times are less than 3 ns.

A.C. CHARACTERISTICS (T_A = 0°C to +70°C, V_{CC} = 5.0V ±5%)

Symbol	Parameter	85C960-25		85C960-20		Units
		Min	Max	Min	Max	
t_1(6)	Input Setup to CLK2a	12		15		ns
t_2(6)	Input Hold from CLK2a	2		2		ns
t_3	CLK2a to A2, A3 Valid Delay	0	8	0	10	ns
t_4	CLK2c to RDY# Output Low Delay		10		15	ns
t_5(7)	CLK2c to RDY# Output High Delay		10		15	ns
t_6	CLK2a to CS0#–CS3# High Delay	5	40	5	50	ns
t_7	CLK2a to BLAST# Low Delay		20		20	ns
t_8	CLK2a to BLAST# High Delay	5		5		ns
t_9(8)	CLK2b to WCLK# Low Delay	0	10	0	12	ns
t_{10}(8)	CLK2d to WCLK# High Delay	0	10	0	12	ns
t_{11}(9)	ADS# Low to CS0#–CS3# Low Delay		10		12	ns
t_{12}(9)	CLK2c to CS0#–CS3# Low Delay		12		15	ns
t_{13}(10)	I0–I7 Setup to CLK2a	5		7		ns
t_{14}(10)	I0–I7 Hold from CLK2a	2		2		ns
t_{15}(11)	I0–I7 Valid to CS0#–CS3# Valid Delay–(t_{PD})		10		12	ns
t_{16}	RDY# Input Setup to CLK2d (Write)	7.5		10		ns
t_{17}	RDY# Input Setup to CLK2a (Read)	9		9		ns
t_{18}	RDY# Input Hold after CLK2a (Read/Write)	5		10		ns
t_{19}(12)	RESET High Setup to CLK2 ↑	0		0		ns
t_{20}(13)	RESET High Hold from CLK2 ↑	3		3		ns
t_{21}(12)	RESET Low Setup to CLK2a	5		5		ns

NOTES:
6. Applies to ADS#, DEN#, W/R#, and AD0–AD3. DEN# is high during the entire Ta state in 80960 KA/KB systems.
7. RDY# is an open-drain output. Specified time includes RDY# output float delay and pull-up/pull-down resistors (Figure 8). RDY# remains low for a minimum of 10 ns at the start of a Tr state and goes high by CLK2a of the next Tx state.
8. Minimum WCLK# pulse width is one clock period minus 3 ns. For example, at 25 MHz: 20 ns − 3 ns = a 17 ns minimum WCLK# pulse.
9. Chip Select Decoder latches are transparent flow-through types. Latches open when ADS# is low, DEN# is high, and CLK2 goes high during the middle of a Tx state (CLK2c). Since DEN# is high during the entire Ta state in 80960 KA/KB systems, only CLK2c and ADS# are specified.
10. Chip Select Decoder latches are transparent flow-through types. Latches close when ADS# is high or DEN# is low, or when CLK2 goes high at the start of a Tx state (CLK2a) after the latches have opened. Since ADS# is low and DEN# is high at the end of a Ta in 80960 KA/KB systems, setup and hold times are specified with reference to CLK2a only.
11. Propagation delay while latches are open (transparent); one output switching (high-to-low).
12. RESET must be held high for a minimum of 4 CLK2 cycles (80960 specifies 41 CLK2 cycles minimum).
13. RESET must hold after the low-to-high transition immediately prior to CLK2a. CLK2a is defined as the first low-to-high transition after RESET goes low.

CLK2 EDGES

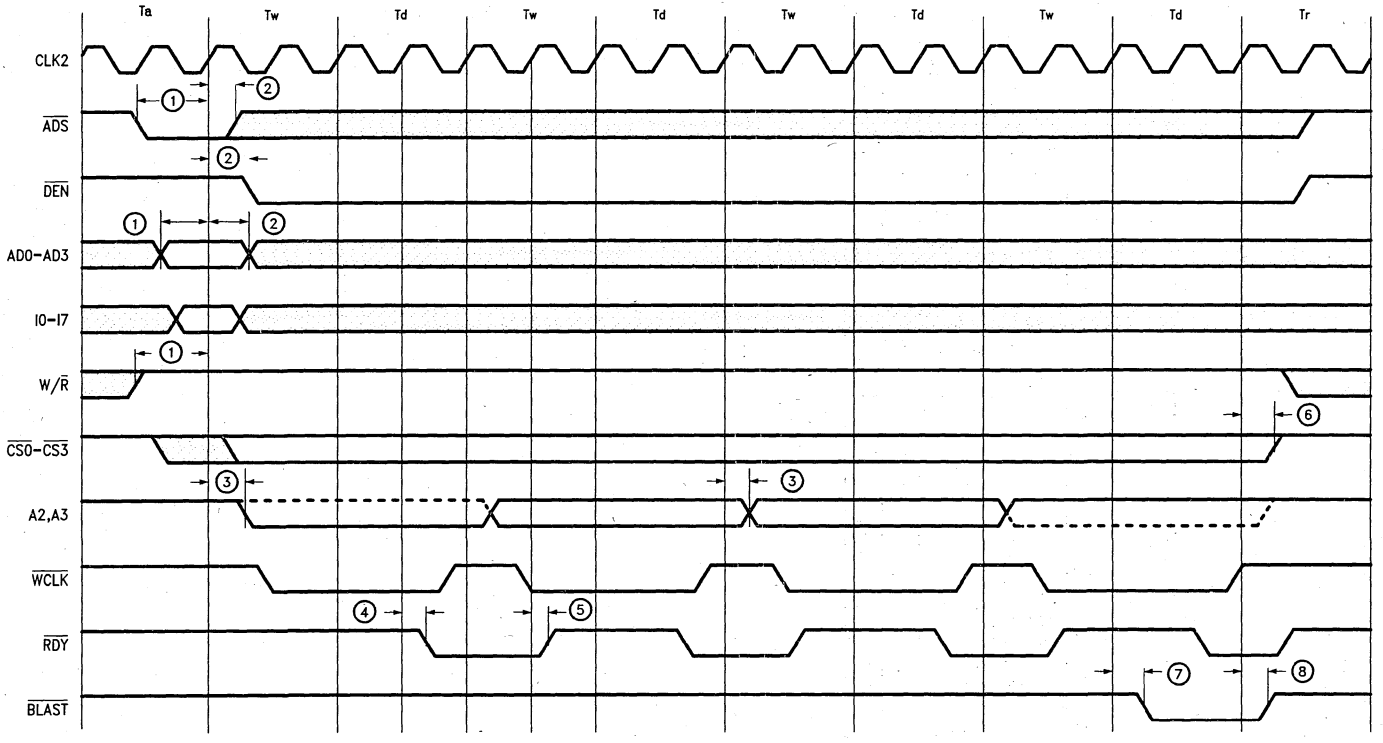


NOTE:
Minimum CLK2 high and low times are 8 ns measured from 1.5V to 1.5V.

CAPACITANCE (T_A = 0°C to +70°C; V_{CC} = 5.0V ± 5%)

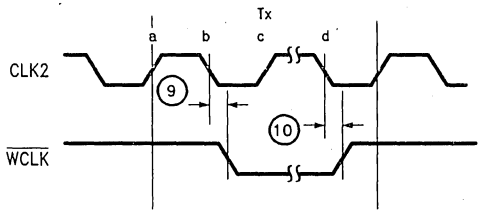
Symbol	Parameter	Min	Typ	Max	Unit	Conditions
C_{IN}	Input Capacitance		6	10	pF	V_{IN} = 0V, f = 1.0 MHz
C_{OUT}	Output Capacitance		6	10	pF	V_{OUT} = 0V, f = 1.0 MHz
C_{CLK}	CLK2 Capacitance		6	10	pF	V_{IN} = 0V, f = 1.0 MHz
C_{VPP}	V_{PP} Pin Capacitance		10	25	pF	V_{PP} on Pin 1 (RESET)
C_{RDY}	RDY# Capacitance		6	10	pF	V_{OUT} = 0V, f = 1.0 MHZ

4 Word Burst Write with 1 Wait State on Each Access
RDY# is Generated by the 85C960
(Same Timing for Read Cycle, Except WCLK# Remains High)



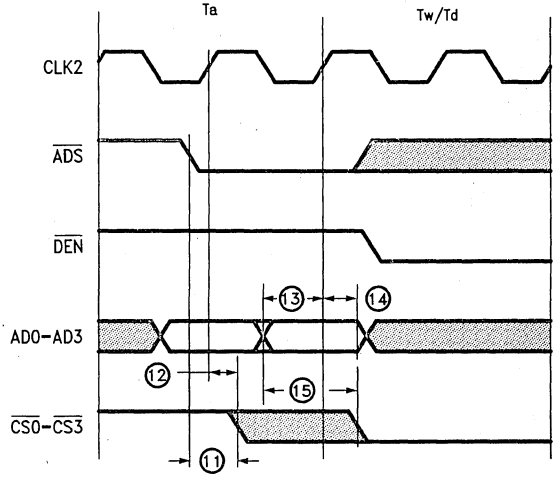
290192-13

WCLK# TIMING



290192-14

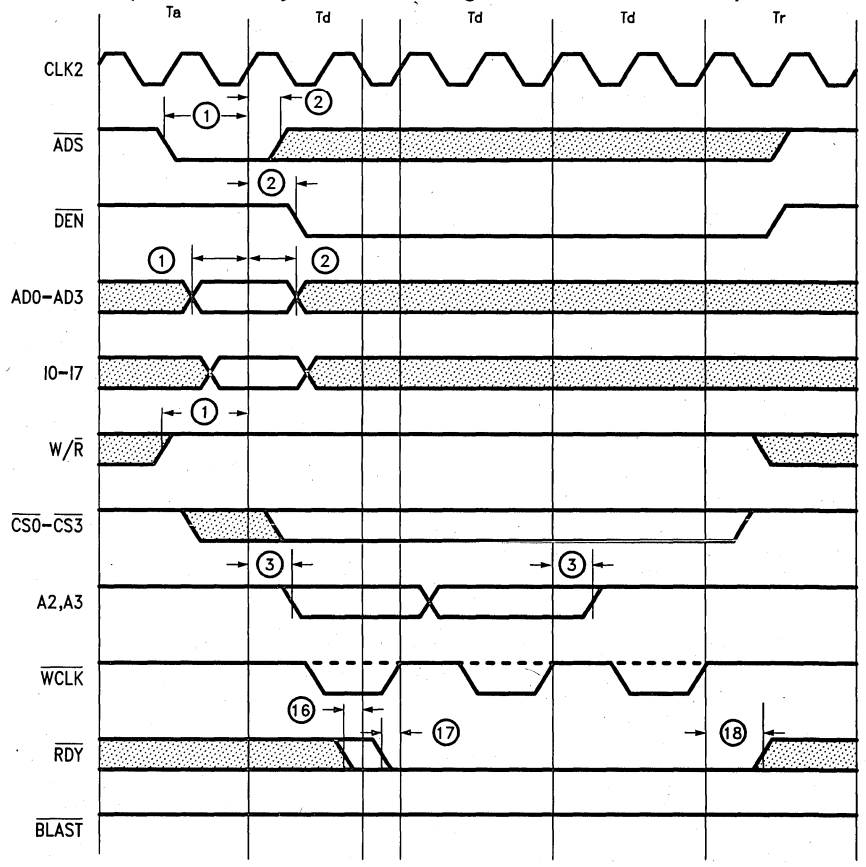
I0-I7 AND CS0#-CS3# TIMING



290192-15

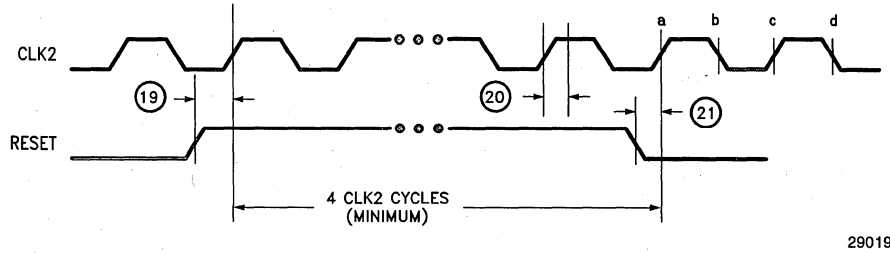
NOTE:
CLK2, ADS#, and DEN# generate internal latch enable. See Figure 7 for details.

3 Word Burst with 0 Wait States on Each Access
RDY# is Generated Externally
(WCLK# is Only Generated During Burst Write Transactions)



290192-16

RESET INPUT TIMING



290192-17

27960CX
PIPELINED BURST ACCESS 1M (128K x 8) CHMOS EPROM

- **Synchronous 4 Byte Data Burst Access**
- **No Glue Interface to 80960CA**
- **High Performance Clock to Data Out**
  - **— Zero Wait State Data to Data Burst**
  - **— Up to 33 MHz 80960CA Performance**
- **Asynch Microcontroller Reset Function**
  - **— Returns to Known State with High-Z Outputs**
- **Pipelined Addressing for Optimal Bus Bandwidth on 80960CA**
  - **— Next Addressing Overlaps Last Data Byte**
- **CHMOS III-E for High Performance and Low Power**
  - **— 125 mA Active, 30 mA Standby**
  - **— TTL Compatible Inputs**
- **1 Mbit Density Configures as 128K x 8**

Intel's 27960CX is a 5V only, 1,048,576 bit, Erasable Programmable Read Only Memory, organized as 128K words of 8 bits.

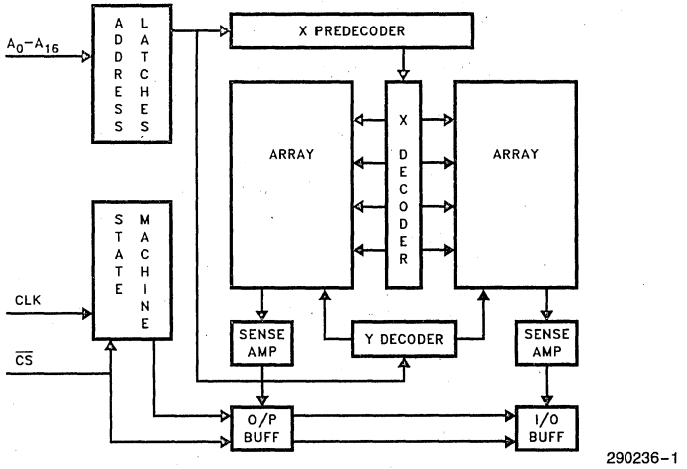
The 27960CX provides a no glue synchronous burst interface to the 80960CA bus. Internally the 27960CX is organized in 4 byte blocks, in which each byte is accessed sequentially. The internal state machine is factory configured to generate either 1 or 2 wait-states between the address and first data byte. High performance outputs provide zero wait-state data to data accesses at clock frequencies up to 33 MHz.

Pipelining capability allows addresses to overlap previous data, further optimizing bus bandwidth in 80960CA applications. An asynchronous microcontroller $\overline{\text{RESET}}$ feature puts the outputs in the high impedance state and takes the internal state machine to a known state where a new burst access can begin.

The 27960CX is available in 44-lead PLCC package, providing optimum cost effectiveness.

The 27960CX is manufactured on Intel's 1 micron CHMOS III-E technology. The Quick-Pulse Programming™ algorithm provides fast, reliable programming with throughput under 17 seconds for optimized equipment.

*CHMOS is a Patented Process of Intel Corporation.



290236-1

Figure 1. 27960CX Burst EPROM Block Diagram

September 1991
Order Number: 290236-006

27960CX BURST EPROM

EPROMs are established as the preferred code storage device in embedded applications. The non-volatile, flexible, reliable, cost effective EPROM makes a product easier to design, manufacture and service. Until recently, however, EPROMs could not match the performance needs of high-end systems. The 27960CX was designed to support the 80960CA embedded processor. It utilizes the burst interface to offer near zero wait-state performance without the high cost normally associated with this performance.

In embedded designs, board space and cost must be kept at a minimum without impacting performance and reliability. The 27960CX removes the need for expensive high-speed shadow RAM backed up by slow EPROM or ROM for non-volatile code storage. Code optimization concerns are reduced with "off-chip" code fetches no longer crippling to system performance. FONTs can be run directly out of these EPROMs at the same performance as high-speed DRAMs. With the 27960CX, the EPROM is the ideal code or FONT storage device for your 80960CA system.

Architecture

The 27960CX provides a no-glue, synchronous burst interface to the 80960CA's bus. It operates in pipelined or non-pipelined modes. Internally, the 27960CX is organized in 4 byte blocks which are accessed sequentially. A burst access begins on the first clock pulse after $\overline{ADS}$ and $\overline{CS}$ are asserted. The address of the 4 byte block is latched on the rising edge of clock following $\overline{ADS}$. After a preset number of wait-states (1 or 2), data is output one byte at a time on each subsequent clock cycle. A burst access is terminated on the rising edge of clock with $\overline{BLAST}$ asserted. High performance outputs provide zero wait-state data to data accesses at clock frequencies up to 33 MHz. Extra power and ground pins dedicated to the outputs reduce the effects of fast output switching on device performance.

The pipelining capability of the 27960CX allows the address to overlap the last data byte of the burst, further optimizing bus band width in 80960CA applications. In the pipelined mode, with a non-buffered interface, the 27960CX delivers 4 bytes of data in 6 clock cycles at 33 MHz. In a 32-bit configuration, this translates into a read bandwidth of 88 Mbytes/sec. Performance capability of the 27960CX in different 80960CA systems is given in Table I.

*CERQUAD is available in a socket only version.

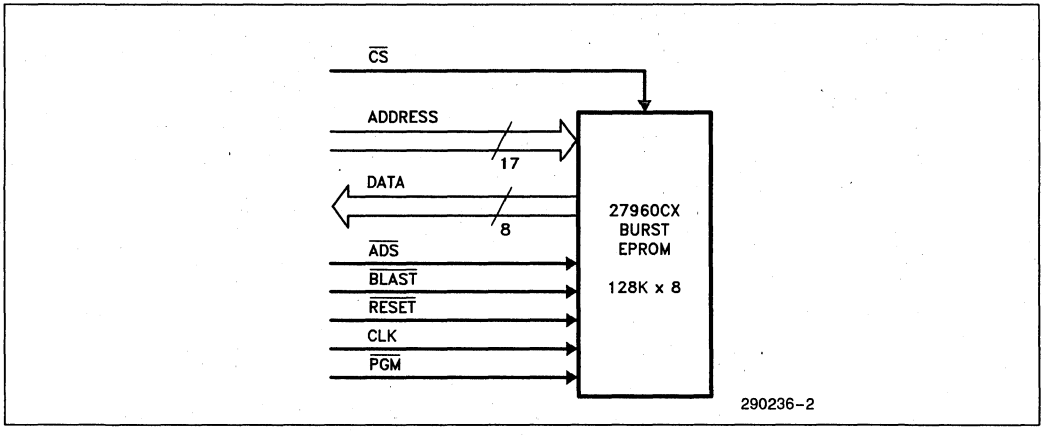


Figure 2. 27960CX Burst EPROM Signal Set

Table 1. Performance Capability

33 MHz 2 WS Non-Buffered: 4 Words/6 Clock Cycles → 88 Mbytes/Sec

ADDR	A_{00}	WS	WS	—	—	—	A_{01}	WS	WS	—	—	—	A_{02}	WS
DATA	—	—	—	D_{00}	D_{01}	D_{02}	D_{03}	—	—	D_{10}	D_{11}	D_{12}	D_{13}	—
PCLK	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_1	C_2	C_3	C_4	C_5	C_6	C_1

25 MHz 2 WS Buffered: 4 Words/6 Clock Cycles → 66 Mbytes/Sec

ADDR	A_{00}	WS	WS	—	—	—	A_{01}	WS	WS	—	—	—	A_{02}	WS
DATA	—	—	—	D_{00}	D_{01}	D_{02}	D_{03}	—	—	D_{10}	D_{11}	D_{12}	D_{13}	—
PCLK	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_1	C_2	C_3	C_4	C_5	C_6	C_1

16 MHz 1 WS Buffered: 4 Words/5 Clock Cycles → 51 Mbytes/Sec

ADDR	A_{00}	WS	—	—	—	A_{01}	WS	—	—	—	A_{02}	WS	
DATA	—	—	D_{00}	D_{01}	D_{02}	D_{03}	—	D_{10}	D_{11}	D_{12}	D_{13}	—	
PCLK	C_1	C_2	C_3	C_4	C_5	C_6	C_1	C_2	C_3	C_4	C_5	C_1	

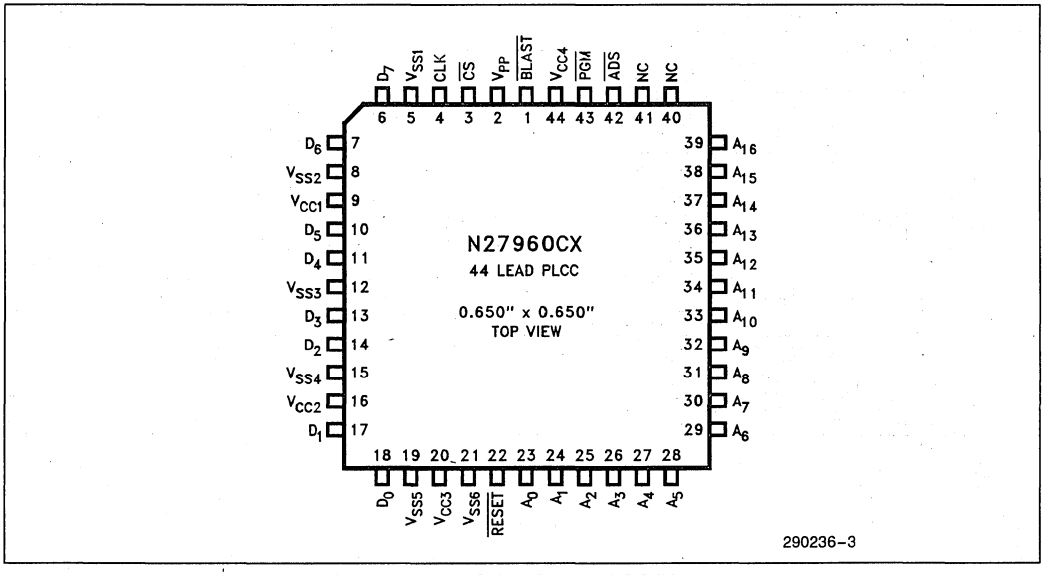


Figure 3. 27960CX 44 Lead PLCC Pinout

4

PIN DESCRIPTIONS

Symbol	Pin	Function
A_0–A_{16}	23–39	**ADDRESS INPUTS:** During a burst operation, A_2–A_{16} provides the base address pointing to a block of four consective bytes. A_0 and A_1 select the first byte of the burst access. The 27960CX latches addresses in the first clock cycle. An internal address generator increments addresses A_0 and A_1 for subsequent bytes of the burst.
D_0–D_7	18, 17, 14, 13, 11, 10, 7, 6	**DATA INPUTS/OUTPUTS**
$\overline{ADS}$	42	**ADDRESS STROBE:** Indicates the start of a new bus access. $\overline{ADS}$ is active low in the first clock cycle of a bus access.
$\overline{CS}$	3	**CHIP SELECT:** Master device enable. When asserted (active low) data can be written to and read from the device. In read mode, $\overline{CS}$ enables the state machine and the I/O circuitry. **NOTE:** 1. The address decode path is independent of $\overline{CS}$, i.e., X and Y decoding is always powered up. 2. For programming, $\overline{CS}$ should remain low for the entire cycle. Program and verify functions are done one byte at a time. 3. $\overline{CS}$ going high does not terminate a concurrent burst cycle.
$\overline{BLAST}$	1	**BURST LAST:** Terminates a concurrent burst data cycle at the rising edge of the CLK. It must be asserted by the fourth data byte.
$\overline{RESET}$	22	**RESET:** Resets the state machine into a known state, tri-states the outputs. $\overline{RESET}$ must be asserted for a minimum of 10 clock cycles. At least 5 clock cycles are required after deassertion of $\overline{RESET}$ before beginning the next cycle. $\overline{RESET}$ will abort a concurrent bus cycle.
$\overline{PGM}$	43	**PROGRAM-PULSE CONTROL INPUT**
V_{PP}	2	**PROGRAMMING POWER SUPPLY**
V_{SS}	5, 8, 12, 15, 19, 21	**GROUND**
V_{CC}	9, 16, 20, 44	**SUPPLY VOLTAGE INPUT**

INTERFACE EXAMPLE

Overview

This example illustrates 8-, 16- and 32-bit wide 27960CX interfaces to the 80960CA. The designs offer a simple "no-glue" interface.

A non-buffered 27960CX system organized as 256K x 32 is shown in Figure 4A. Since the 27960CX is capable of driving a 80 pF load, large, non-buffered systems can be implemented by stacking up to 2 banks of 4 EPROMs, resulting in a 256K x 32 memory subsystem. The input capacitive load seen on the address lines (due to the EPROM only) is 24 pF for a 128K x 32 system and 48 pF for a 256K x 32 system. The EPROM is specified at 6 pF for input capacitance (15 pF max) and 12 pF typical for output capacitance. Larger systems can be implemented with buffers (Figure 4B).

Chip Select Logic

High order address lines are decoded to provide $\overline{CS}$. Qualification with other signals is not required. The chip select logic can be implemented with standard asynchronous decoders, PAL's or PLD's (like Intel's 85C508).

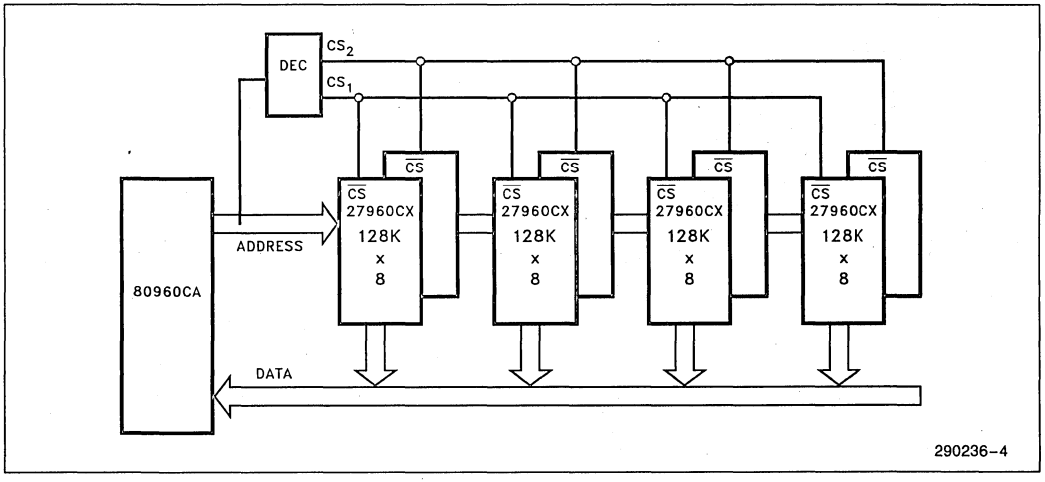


Figure 4A. 256K x 32 Non-Buffered Burst EPROM Memory System

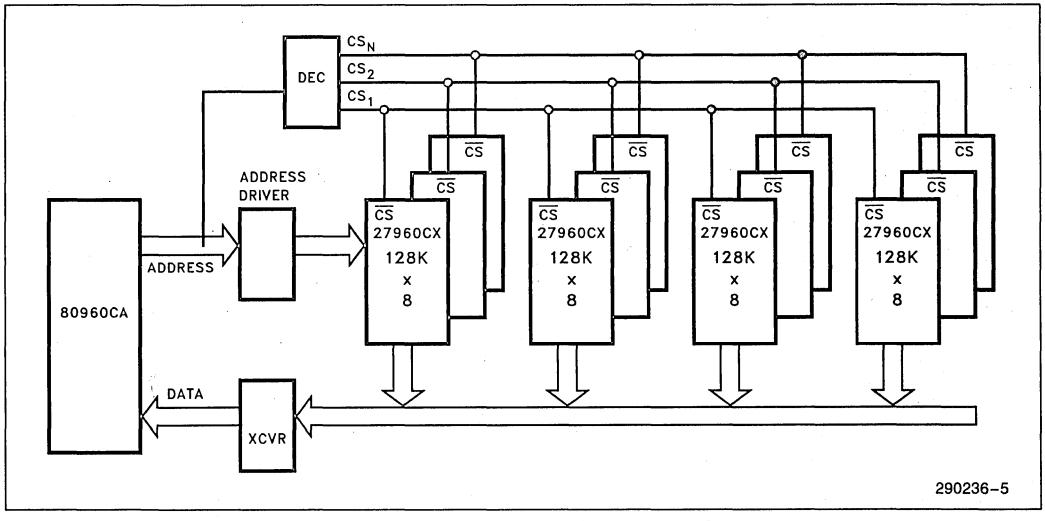


Figure 4B. Buffered Burst EPROM Memory System

Schematics

Figure 5 shows a non-buffered, 128K x 32 27960CX EPROM system.

Chip select logic, the only external logic that is required for this interface, can be derived from the global system chip select circuitry.

In a non-buffered, 16-bit system (Figure 6A) $\overline{BE1}$ and A_2 connect to the lower order address bits of the 27960CX. $\overline{BE1}$ connects to A_0 of both EPROMs, while A_2 connects to both A_1's.

In a non-buffered, 8-bit system (Figure 6B) $\overline{BE0}$ and $\overline{BE1}$ connect to A_0 and A_1 respectively.

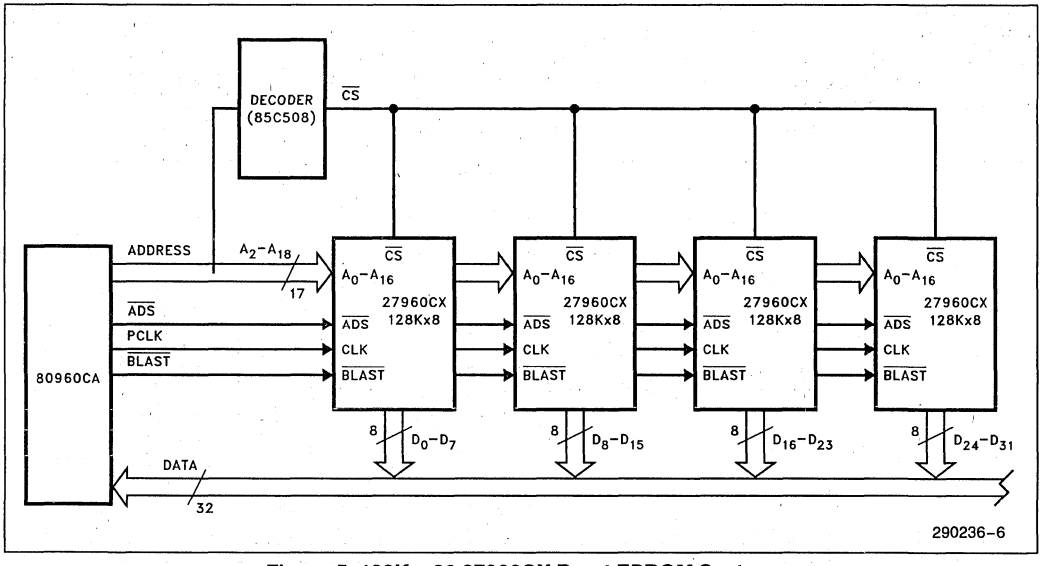


Figure 5. 128K x 32 27960CX Burst EPROM System

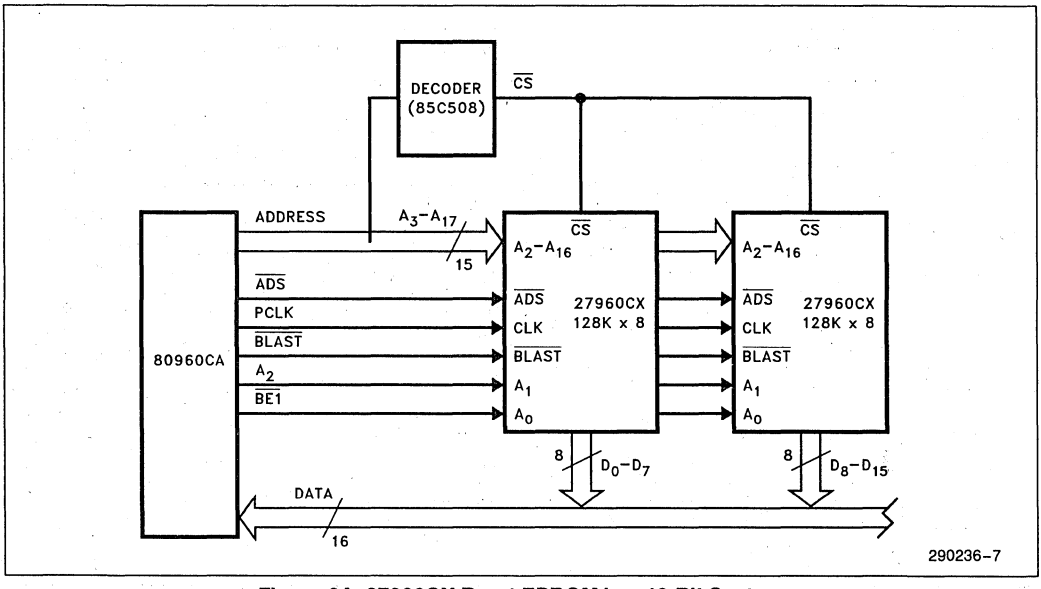


Figure 6A. 27960CX Burst EPROM in a 16-Bit System

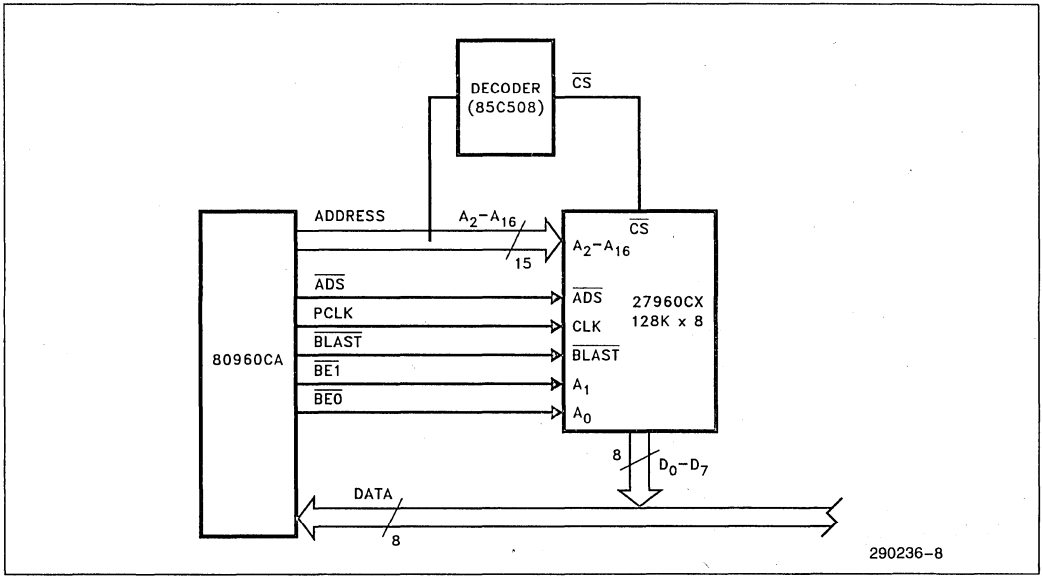


Figure 6B. 27960CX Burst EPROM in a 8-Bit System

Waveforms

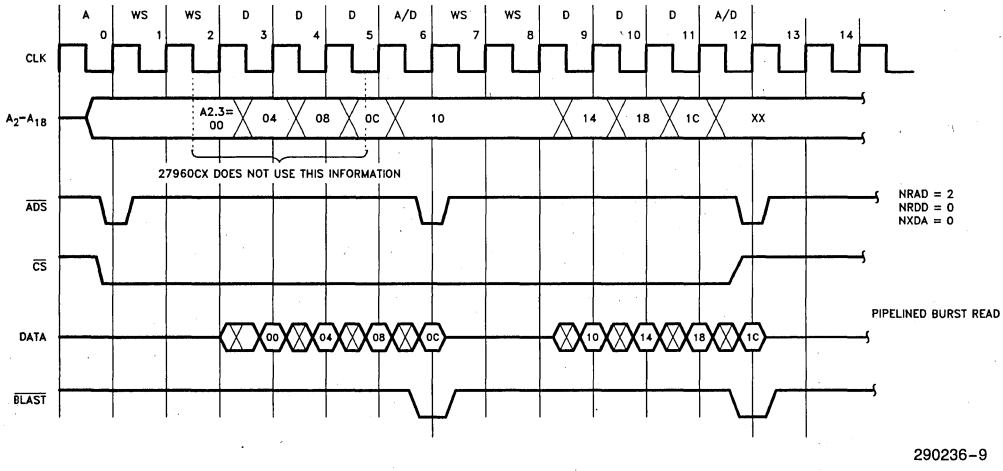
Figure 7 shows the timing waveforms of a 27960CX pipelined read in a 32-bit system.

$\overline{CS}$ Setup Time

$\overline{CS}$ setup time is the time between $\overline{CS}$ being asserted and the first CLK rising edge (during the address cycle). Since a memory access begins on the first CLK rising edge after $\overline{ADS}$ and $\overline{CS}$ are asserted, a minimum $\overline{CS}$ setup time of 7 ns (t_{SVCH}) at 33 MHz is required. With the 80960CA's maximum valid address delay of 14 ns at 33 MHz, 9 ns remains for $\overline{CS}$ decoding logic.

Bootup

The wait state configuration (1 or 2), of the 27960CX is programmed by the user into the 80960CA Region Table parameters of NRAD, NRDD, and NXDA. NRDD is always 0 for the 27960CX.



NOTES:
1. The EPROM can also operate in non pipelined mode i.e, next address and $\overline{ADS}$ can be asserted in the clock cycle following the last data word of the burst.
2. 2 – 0 – 0 – 0 Burst Read → 2 indicates the number of wait states to access the first word
0's indicate the number of wait states for subsequent data words:
0 in this case!

Figure 7. Two Cycles of a 27960CX 2 Wait State 4 Byte Read (2–0–0–0 Burst Read) in a 32 Bit System

During boot-up (Figure 8), the 80960CA picks up it's Region Table data from addresses FFFF FF00; FFFF FF04; FFFF FF08 and FFFF FF0C. Only the least significant byte of each of the above four 32-bit accesses is used to configure the Region Table. For boot-up, the wait-state parameters NRAD and NXDA default to 31 and 3 respectively. During boot-up, the 27960CX will wrap around the first word of the four-word burst and hold the first word until $\overline{BLAST}$ is asserted.

27960CX DEVICE NAMES

The device names on the 27960CX were derived as mnemonics that correspond to the number of wait states and expected operating frequency for the device. For example, the 25 MHz, 2 wait state 27960CX is named 27960C2-25.

AC TIMING DERIVATIONS

The AC timings for the 27960CX were generated specifically to meet the requirements of the 80960CA microprocessor. In each case the applicable 80960CA clock frequency and AC timing were taken together with an address buffer delay (if needed) and a typical 2 ns guardband to generate the 27960CX AC timing. Worst case timings were always assumed. On timings where the EPROM is faster than the microprocessor, we specified the time required by the EPROM and left the excess time as additional system guardband. The example below shows how the 27960C2-33 $tavc_0h$ timing was derived.

@33 MHz the clock cycle is ~30 ns.
t_{OV2} of the 80960CA is 3 ns – 14 ns.
Typical 2 ns guardband.

27960C2-33 $tavc_0h$ = 30 ns – 14 ns – 2 ns
= 14 ns

Decoders are needed for the systems chip select decoding. For the 27960CX timings we assumed a 10 ns chip select decoder for 16 MHz and a 7 ns decoder for 25 MHz and 33 MHz systems. The example below shows how the 27960C2-33 tsvch timing was derived.

@33 MHz the clock cycle is ~30 ns.
t_{OV2} of the 80960CA is 3 ns – 14 ns.
Decoder = 7 ns

27960C2-33 tsvch = 30 ns – 14 ns – 7 ns
= 9 ns

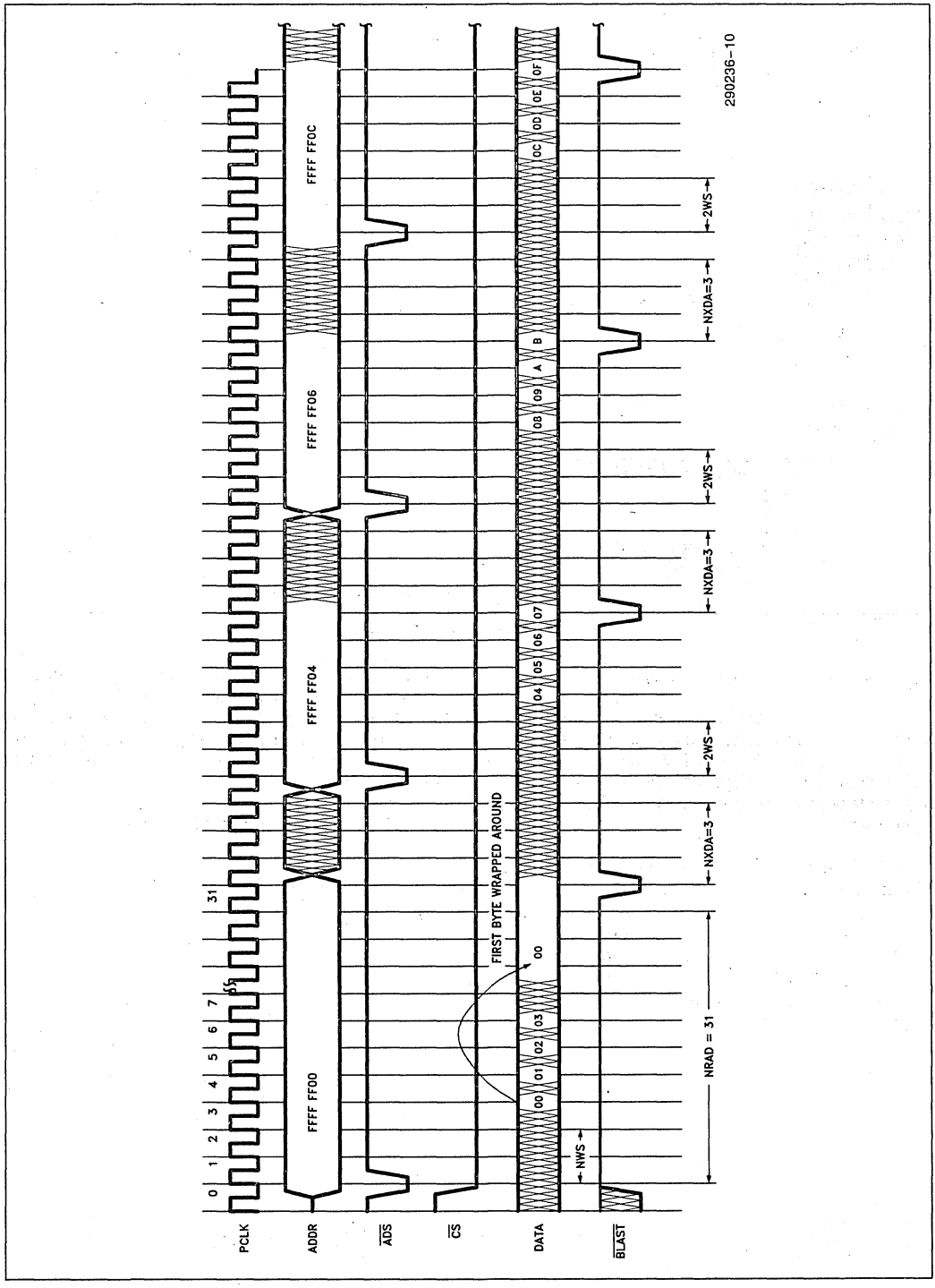


Figure 8. 27960CX/80960CA Bootup Timing

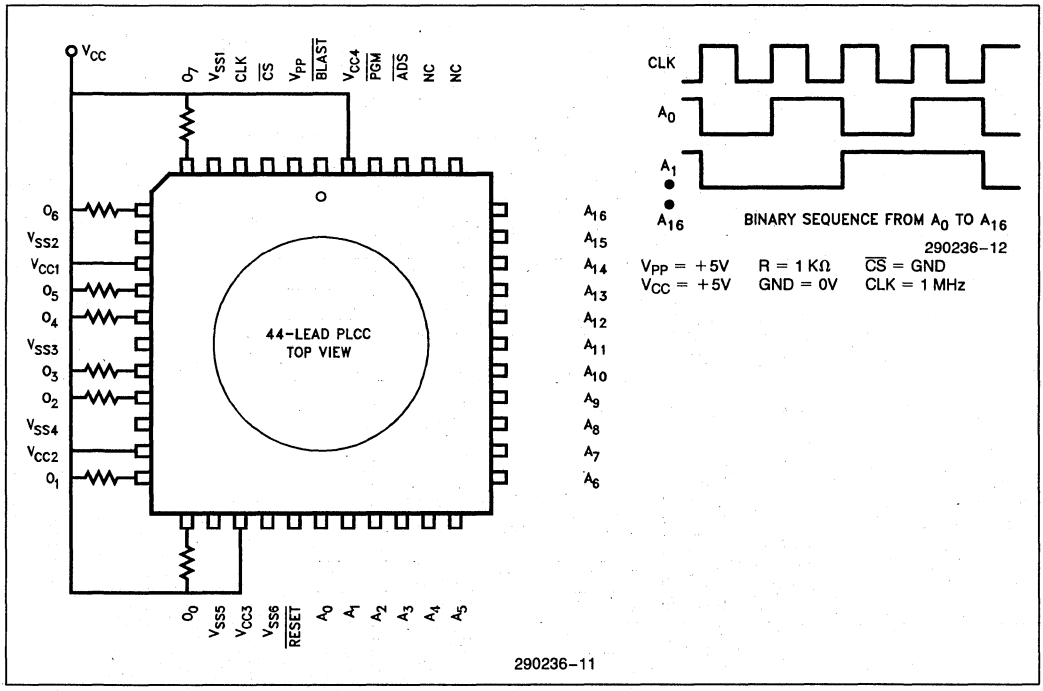


Figure 9. 27960CX Burn in Biasing Diagram

System Buffering Considerations

For large system applications buffering may be required between the microprocessor and memory devices. The 25 and 16 MHz 27960CX AC timings take this into account. For applications not requiring buffering these devices will provide additional system guardband.

The list below shows the buffers used in generating the 27960CX timings:

	Input Buffer	Output Buffer
25 MHz	8 ns	5 ns
16 MHz	10 ns	7 ns

Note that the 25 MHz buffers are slightly faster in keeping with the increased sensitivity for higher performance. Significantly faster buffers are available for applications requiring them. The example below shows the tchqv timing analysis for a buffered 27960C2-25.

@25 MHz the clock cycle is ~40 ns.

t_{IH1} of the 80960CA is 5 ns.

Output buffer for 25 MHz = 5 ns

27960C2-25 t_{CHQV} = 40 ns − 5 ns − 5 ns
= 30 ns

ABSOLUTE MAXIMUM RATINGS*

Read Operating Temperature . . . . . 0°C to +70°C(8)

Case Temperature Under Bias . . −10°C to +80°C(8)

Storage Temperature . . . . . −65°C to +125°C

All Input or Output Voltages with Respect to Ground . . . . . −0.6V to +6.5V(4)

Voltage on A_9 with Respect to Ground . . . . . −0.6V to +13.0V(4)

V_{PP} Supply Voltage with Respect to Ground . . . . . −0.6V to +14.0V(4)

V_{CC} Supply Voltage with Respect to Ground . . . . . −0.6V to +7.0V(4)

NOTICE: This data sheet contains preliminary information on new products in production. The specifications are subject to change without notice. Verify with your local Intel Sales office that you have the latest data sheet before finalizing a design.

**WARNING: Stressing the device beyond the "Absolute Maximum Ratings" may cause permanent damage. These are stress ratings only. Operation beyond the "Operating Conditions" is not recommended and extended exposure beyond the "Operating Conditions" may affect device reliability.*

READ OPERATION

DC CHARACTERISTICS $0°C < T_A +70°C$, $V_{CC} = 5V \pm 10\%$, TTL Inputs

Symbol	Parameter		Notes	Min	Max	Unit	Test Condition
I_{LI}	Input Load Current				1	μA	$V_{IN} = 5.5V$
I_{LO}	Output Leakage Current				10	μA	$V_{OUT} = 5.5V$
I_{PP}	V_{PP} Load Current Read				10	μA	$V_{PP} = 0$ to V_{CC}, $\overline{PGM} = V_{IH}$
I_{SB}	V_{CC} Standby	Switching	2		45	mA	$\overline{CS} = V_{IH}$, f = 33 MHz
		Stable	2		30	mA	$\overline{CS} = V_{IH}$
I_{CC}	V_{CC} Active Current		1, 3, 7		125	mA	$\overline{CS} = V_{IL}$, f = 33 MHz, $I_{OUT} = 0$ mA
V_{IL}	Input Low Voltage		4	−0.5	0.8	V	
V_{IH}	Input High Voltage			2.0	$V_{CC} + 1$	V	
V_{OL}	Output Low Voltage				0.45	V	$I_{OL} = 2.1$ mA
V_{OH}	Output High Voltage		5	$V_{CC} - 0.8$		V	$I_{OH} = -100$ μA
			5	2.4		V	$I_{OH} = -400$ μA
I_{OS}	Output Short Circuit		6		100	mA	

NOTES:
1. Maximum current is with outputs unloaded.
2. I_{CC} standby current assumes no output loading i.e., $I_{OH} = I_{OL} = 0$ mA.
3. I_{CC} is the sum of current through $V_{CC3} + V_{CC4}$ and does not include the current through V_{CC1} and V_{CC2}. (V_{CC1} and V_{CC2} supply power to the output drivers. V_{CC3} and V_{CC4} supply power to the reset of the device.)
4. Minimum DC input voltage on input and output pins is −0.5V. During transitions, this level may undershoot to −2.0V for periods less than 20 ns.
5. Maximum DC voltage on input and output pins is $V_{CC} + 0.5V$ which may overshoot to $V_{CC} + 2.0V$ for periods less than 20 ns.
6. One output shorted for no more than one second. I_{OS} is sampled but not 100% tested.
7. I_{CC} max measured with a 10.11 μF capacitor between V_{CC} and V_{SS}.
8. This specification defines commercial product operating temperatures.

EXPLANATION OF AC SYMBOLS

The nomenclature used for timing parameters are as per IEEE STD 662-1980 IEEE Standard Terminology for Semiconductor Memory.

Each timing symbol has five characters. The first is always a "t" (for time). The second character represents a signal name. e.g., (CLK, $\overline{ADS}$, etc.). The third character represents the signal's level (high or low) for the signal indicated by the second character. The fourth character represents a signal name at which a transition occurs marking the end of the time interval being specified.

The fifth character represents the signal level indicated for the fourth character. The list below shows character representations.

A:	Address	R:	$\overline{Reset}$
B:	$\overline{BLAST}$	Q:	Data
C:	Clock	S:	$\overline{CS}$
H:	Logic High Level	t:	Time
L:	$\overline{ADS}$/Logic Low Level	V:	Valid
P:	V_{PP} Programming Voltage	Z:	Tri-state Level
X:	No longer a valid "driven" logic level		

AC CHARACTERISTICS: READ OPERATION $0°C < T_A < +70°C, V_{CC} = 5V \pm 10\%$

Versions				27960C2-33		27960C2-25		27960C1-16		Unit
				33 MHz 2 Wait State		25 MHz 2 Wait State		16 MHz 1 Wait State		
No.	Symbol	Parameter	Notes	Min	Max	Min	Max	Min	Max	
1	t_{AVC_0H}	Address Valid to CLK High	CLK_0	12		10		22		ns
2	t_{C_NHAX}	CLK High to Address Invalid	2	0		0		0		ns
3	t_{LLCH}	$\overline{ADS}$ low to CLK High	CLK_0	8		8		22		ns
4	t_{CHLH}	CLK high to $\overline{ADS}$ High	5	6	22	6	32	6	40	ns
5	t_{SVCH}	$\overline{CS}$ Valid to CLK High	1	7		7		14		ns
6	t_{C_NHSX}	CLK High to $\overline{CS}$ Invalid	2	0		0		0		ns
7	t_{CHQV}	CLK High to Data Valid	7		27		30		40	ns
8	t_{CHQX}	CLK High to Data Invalid		5		5		5		ns
9	t_{CHQZ}	CLK High to Data High Z	6		25		30		30	ns
10	t_{BVCH}	$\overline{BLAST}$ Valid to CLK High		8		8		22		ns
11	t_{CHBX}	CLK High to $\overline{BLAST}$ Invalid	3	5	22	5	32	5	40	ns

NOTES:
1. Valid signal level is meant to be either a logic high or logic low.
2. The subscript N represents the number of wait states for this parameter. $\overline{CS}$ can be de-asserted (high) after the number of wait states (N) has expired and the EPROM will continue to burst out data for the current cycle.
3. BLAST# must be returned high before the next rising clock edge.
4. The sum of $t_{CHQV} + t_{AVCH} + N_{CLK}$ will not equal actual t_{AVQV} if independent test conditions are used to obtain t_{AVCH} and t_{CHQV} (N = number of wait states).
5. $\overline{ADS}$ must be returned high before the next rising clock edge.
6. Sampled, not 100% tested. The transition is measured ±500 mV from steady state voltage.
7. For capacitive loads above 80 pF, t_{CHQV} can be derated by 1 ns/20 pF.

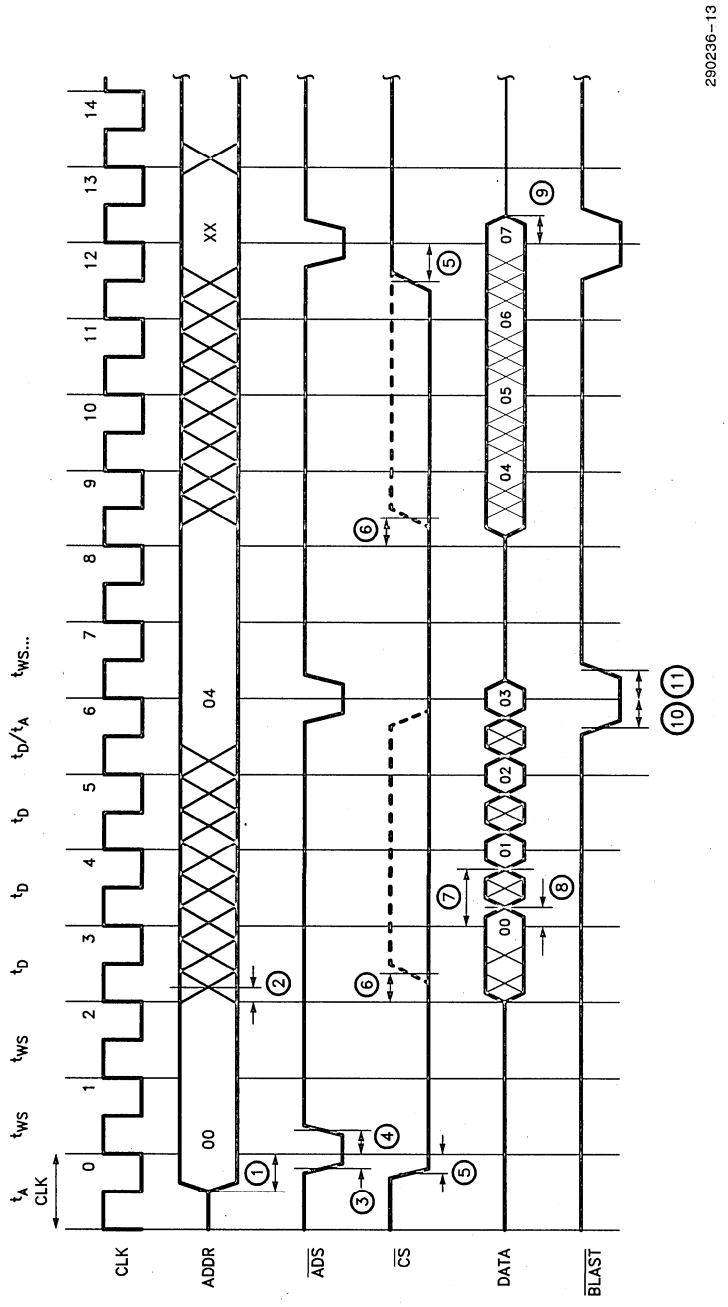


Figure 10. 27960CX Pipelined 2 Wait State AC Waveforms

AC CONDITIONS OF TEST

Input Rise and Fall Times (10% to 90%) ..... 4 ns
Input Pulse Levels ..... 0.45V to 2.4V
Input Timing Reference Level 1.5V
Output Timing Reference Level 1.5V

Table 2. Mode Table

Mode	$\overline{CS}$	$\overline{PGM}$	$\overline{BLAST}$	$\overline{ADS}$	$\overline{RESET}$	A_9	V_{PP}	V_{CC}	OUTPUT
Read	V_{IL}	V_{IH}	V_{IH}(1)	V_{IH}(2)	V_{IH}	X	V_{CC}	V_{CC}	D_{OUT}
Standby(6)	V_{IH}	X	X	X	V_{IH}	X	V_{CC} (5)	V_{CC}	High Z
Program	V_{IL}	V_{IL}	V_{IH}	V_{IH}(2)	V_{IH}	X	(3)	(3)	D_{IN}
Program Verify	V_{IL}	V_{IH}	V_{IH}(1)	V_{IH}	V_{IH}	X	(3)	(3)	D_{OUT}
Program Inhibit	V_{IH}	X	X	X	V_{IH}	X	(3)	(3)	High Z
ID Byte 0: Manufacturer	V_{IL}	V_{IH}	V_{IH}(1)	V_{IH}(2)	V_{IH}	V_{ID}(3)	V_{CC}	V_{CC}	89H
ID Byte 1: Part (27960)	V_{IL}	V_{IH}	V_{IH}(1)	V_{IH}(2)	V_{IH}	V_{ID}(3)	V_{CC}	V_{CC}	E0H
ID Byte 2: CX	V_{IL}	V_{IH}	V_{IH}(1)	V_{IH}(2)	V_{IH}	V_{ID}(3)	V_{CC}	V_{CC}	01B
ID Byte 3: 1 Wait State 2 Wait States	V_{IL}	V_{IH}	V_{IH}(1)	V_{IH}(2)	V_{IH}	V_{ID}(3)	V_{CC}	V_{CC}	01B 10B
Reset	X	X	X	X	V_{IL}	X	V_{CC}	V_{CC}	High Z

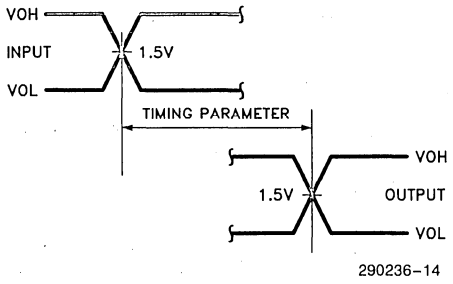
NOTES:
1. V_{IH} until data terminated at which time $\overline{BLAST}$ must go to V_{IL}.
2. Need to toggle from V_{IH} to V_{IL} to V_{IH}.
3. See DC Programming Characteristics for V_{CC}, V_{ID} and V_{PP} voltages.
4. X can be V_{IL} or V_{IH}.
5. $V_{PP} = V_{CC}$ to meet standy current specification. $V_{CC} > V_{PP} > V_{IL}$ will cause a slight increase in standby current.
6. The device must be in the idle state (by asserting $\overline{RESET}$ or using $\overline{BLAST}$) before going into standby.

CAPACITANCE(1) $T_A = 25°C$, f = 1.0 MHz

Symbol	Parameter	Typ	Max	Unit	Condition
C_{IN}	Input Capacitance	4	6	pF	$V_{IN} = 0V$
C_{OUT}	Output Capacitance	12	15	pF	$V_{OUT} = 0V$
C_{VPP}	V_{PP} Capacitance	40	45	pF	$V_{IN} = 0V$

NOTE:
1. Sampled. Not 100% tested.

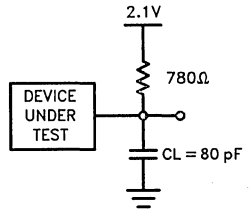
AC INPUT/OUTPUT REFERENCE WAVEFORMS



290236-14

Input and output timings are measured from 1.5V. Timing values are specified assuming maximum input and output rise and fall time = 4 ns.

AC TESTING LOAD CIRCUIT



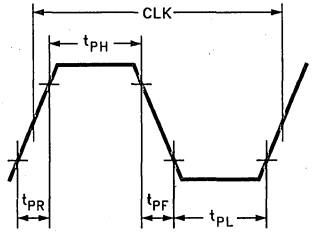
290236-15

CL includes jig capacitance
For t_{CHQZ} C_L = 5 pF and R_L = 405Ω

CLOCK CHARACTERISTICS

Versions		33 MHz		25 MHz		20 MHz		16 MHz		Units
Symbol	Parameter	Min	Max	Min	Max	Min	Max	Min	Max	
CLK	Period	30.3		40		50		62.5		ns
t_{PR}	Rise Time	1	4	1	4	1	4	1	4	ns
t_{PF}	Fall Time	1	4	1	4	1	4	1	4	ns
t_{PL}	Low Time	(t/2) − 2	t/2	(t/2) − 3	t/2	(t/2) − 4	t/2	(t/2) − 4	t/2	ns
t_{PH}	High Time	(t/2) − 2	t/2	(t/2) − 3	t/2	(t/2) − 4	t/2	(t/2) − 4	t/2	ns
Max Rise Time for Programming CLK = 100 ns										

CLOCK WAVEFORM



290236-16

Program/Program Verify

Initially, and after each erasure, all bits of the EPROM are in the "1's" state. Data is introduced by selectively programming "0's" into the desired bit locations. Although only "0's" can be programmed, both "1's" and "0's" can be present in the data word. Ultraviolet erasure is the only way to change "0's" to "1's".

Programming mode is entered when V_{PP} is raised to 12.75V. Program/Verify operation is synchronous with the clock and can only be initiated following an idle state. Program and Program Verify take place in 3 clock cycles. In the first clock cycle, addresses and data are input and programming occurs. Program Verify follows in the second clock cycle and the third clock cycle terminates synchronous Program/Verify operation, returning the state machine to the idle state with outputs at high impedance.

As in the Read mode, A_2–A_{16} point to a four byte block in the memory array. During programming, the internal address increment circuitry is disabled and the programmer must supply A_0 and A_1 to point to an individual byte within the four byte block that is to be programmed. Only one byte is programmed in each 3 cycle Program/Verify sequence.

Program Inhibit

The Program Inhibit mode allows parallel programming and verification of multiple devices with different data. With V_{PP} at 12.75V, a Program/Verify sequence is initiated for any device that receives a valid $\overline{ADS}$ pulse and rising clock edge while $\overline{CS}$ is asserted. A $\overline{PGM}$ pulse programs data in the first cycle of the sequence and data for Program Verify is output in the second cycle. The Program/Verify sequence is inhibited on any devices for which $\overline{CS}$ is not asserted. Data will not be programmed and the outputs will remain in their high impedance state.

inteligent Identifier™ Mode

The device's manufacturer, product type, and configuration are stored in a four byte block that can be accessed by using the inteligent Identifier™ mode.

The programmer can verify the device identifier and choose the programming algorithm that corresponds to the Intel 27960CX. The inteligent Identifier can also be used to verify that the product is configured with the desired Read mode options for wait states.

inteligent Identifier mode is entered when A_9 (pin 32) is raised to its high voltage (V_{ID}) level. The internal state machine is then set for intelligent Identifier Read operation. Reading the identifier is similar to a Read operation on a one wait state configured product. Up to four bytes can be read in a single burst access. inteligent Identifier read is terminated by a synchronous $\overline{BLAST}$ input, returning the state machine to the idle state with outputs at high impedance.

The four byte block code for the inteligent Identifier code is located at address 00H through 03H and is encoded as follows:

MEANING	(A1, A0)	DATA
Intel ID	Byte 00	89h
27960	Byte 01	E0h
CX	Byte 10	01b
1 Wait State	Byte 11	01b
2 Wait States	Byte 11	10b

RESET MODE

Due to the synchronous nature of the 27960CX, the various operating modes must be initiated from a known idle state. During normal operation, the internal state machine returns to an idle state at the termination of a bus access (after $\overline{BLAST}$ is asserted).

During initial device power up, the state machine is in an indeterminant state. The reset mode is provided to force operation into the idle state. Reset mode is entered when the $\overline{RESET}$ pin is asserted. Output pins are asynchronously set to the high impedance state and address latches are put into the flow through mode. A reset is successfully completed and the state machine set in an idle state when $\overline{RESET}$ has been asserted for a minimum of 10 clock cycles and deasserted for five clock cycles.

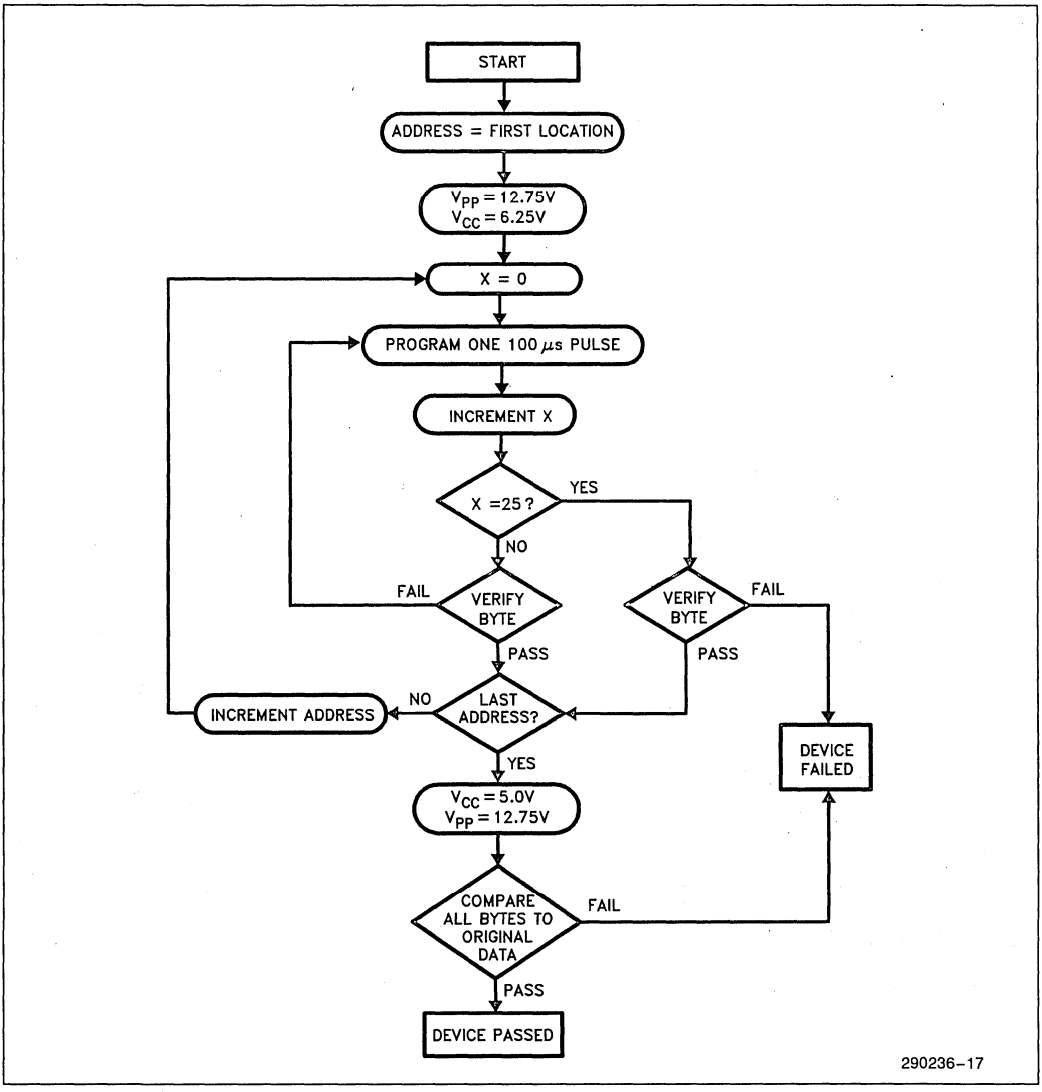


Figure 11. Quick-Pulse Programming™ Algorithm

QUICK-PULSE PROGRAMMING™ ALGORITHM

The Quick-Pulse Programming algorithm programs Intel's 27960CX. Developed to substantially reduce programming throughput time, this algorithm allows optimized equipment to program a 27960CX in under 17 seconds. Actual programming time depends on the programmer used.

The Quick-Pulse Programming algorithm uses a 100 μs pulse followed by a byte verification to determine when the addressed byte is correctly programmed. The algorithm terminates if 25 100 μs pulses fail to program a byte. Figure 11 shows the 27960CX Quick-Pulse Programming algorithm flowchart.

The entire program-pulse/byte-verify sequence is performed with V_{CC} = 6.25V and V_{PP} = 12.75V. The program equipment must establish V_{CC} before applying voltages to any other pins. When programming is complete, all bytes should be compared to the original data with V_{CC} = 5.0V and V_{PP} = 12.75V.

D.C. PROGRAMMING CHARACTERISTICS $T_A = 25° \pm 5°C$

Symbol	Parameter	Notes	Min	Max	Unit	Condition
I_{LI}	Input Load Current			10	μA	$V_{IN} = V_{IH}$ or V_{IL}
I_{CC}	V_{CC} Program Current	1		125	mA	$\overline{CS} = V_{IL}$
I_{PP}	V_{PP} Program Current	1		50	mA	$\overline{CS} = V_{IL}$
V_{IL}	Input Low Voltage		−0.5	0.8	V	
V_{IH}	Input High Voltage		2.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage(Verify)			0.40	V	I_{OL} = 2.1 mA
V_{OH}	Output High Voltage(Verify)		V_{CC} − 0.8		V	I_{OH} = −400 μA
V_{ID}	A_9 int$_e$ligent Identifier Voltage		11.5	12.5	V	
V_{CC}	Supply Voltage (Program)	2	6.0	6.5	V	
V_{PP}	Program Voltage	2	12.5	13.0	V	

NOTES:
1. The maximium current value is with outputs unloaded.
2. V_{CC} must be applied simultaneously or before V_{PP} and removed simultaneously or after V_{PP}.
3. During programming clock levels are V_{IH} and V_{IL}.

A.C. PROGRAMMING, RESET AND ID CHARACTERISTICS $T_A = 25°C \pm 5°C$

No.	Symbol	Parameter	Notes	Min	Max	Unit
1	t_{AVPL}	Address Valid to $\overline{PGM}$ Low		2		μs
2	t_{CHAX}	CLK High to Address Invalid		50		ns
3	t_{LLCH}	$\overline{ADS}$ Low to CLK High	1	50		ns
4	t_{CHLH}	CLK High to $\overline{ADS}$ High	2	50		ns
5	t_{SVCH}	$\overline{CS}$ Valid to CLK High		50		ns
6	t_{CHSX}	CLK High to $\overline{CS}$ Invalid	3			ns
7	t_{CHQV}	CLK High to D_{OUT} Valid		100		ns
8	t_{CHQX}	CLK High to D_{OUT} Invalid		0		ns
9	t_{BVCH}	$\overline{BLAST}$ Valid to CLK High		50		ns
10	t_{CHBX}	CLK High to $\overline{BLAST}$ Invalid	4	50		ns
11	t_{QVPL}	DATA Valid to $\overline{PGM}$ Low		2		μs
12	t_{PLPH}	$\overline{PGM}$ Program Pulse Width		95	105	μs
13	t_{PHQX}	$\overline{PGM}$ High to D_{IN} Invalid		2		μs
14	t_{CLPL}	CLK Low to $\overline{PGM}$ Low		50		ns
15	t_{QZCH}	D_{IN} Tri-State to CLK High		2		μs
16	t_{VCS}	V_{CC} Program Voltage to CLK High	7	2		μs
17	t_{VPS}	V_{PP} Program Voltage to CLK High	7	2		μs
18	t_{A_9HCH}	A_9 V_{ID} Voltage to CLK High		2		μs
19	t_{CHA_9X}	CLK High to A_9 Not V_{ID} Voltage		2		μs
20	t_{RVCH}	$\overline{RESET}$ Valid to CLK High	6	50		ns
21	t_{CHCL}	CLK High to CLK Low	5	100		ns
22	t_{CLCH}	CLK Low to CLK High	5	100		ns

NOTES:
1. If $\overline{CS}$ is low, $\overline{ADS}$ can go low no sooner than the falling edge of the previous CLK.
2. $\overline{ADS}$ must return high prior to the next rising edge of clock.
3. $\overline{CS}$ must remain low until after the rising edge of CLK1.
4. $\overline{BLAST}$ must return high prior to the next rising edge of CLK.
5. Max CLK rise/fall time is 100 ns.
6. $\overline{RESET}$ must be low for 10 clock cycles and high for 5 clock cycles.
7. V_{CC} must be applied simultaneously or before V_{PP} and removed simultaneously or after V_{PP}.

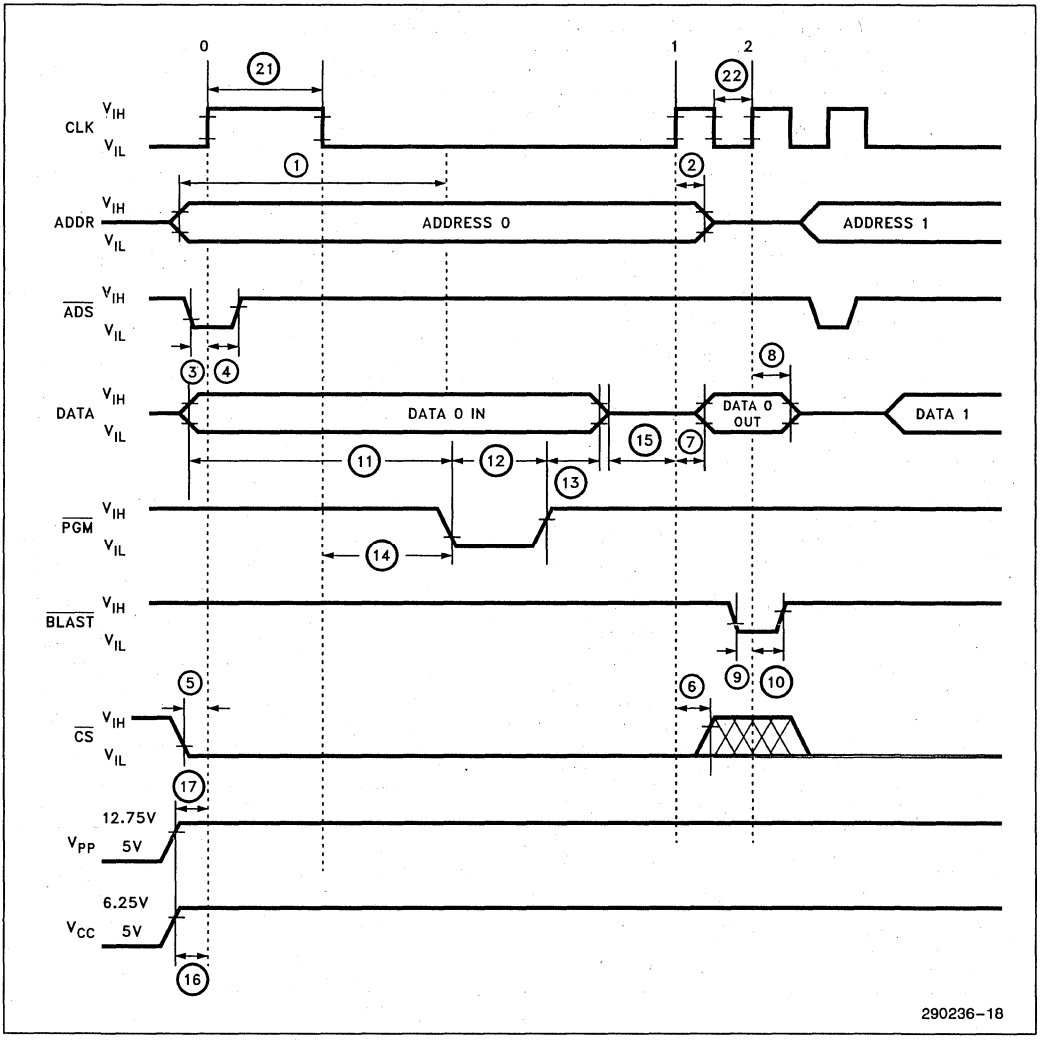


Figure 12. 27960CX Programming Waveforms

RESET and inteligent Identifier Waveforms

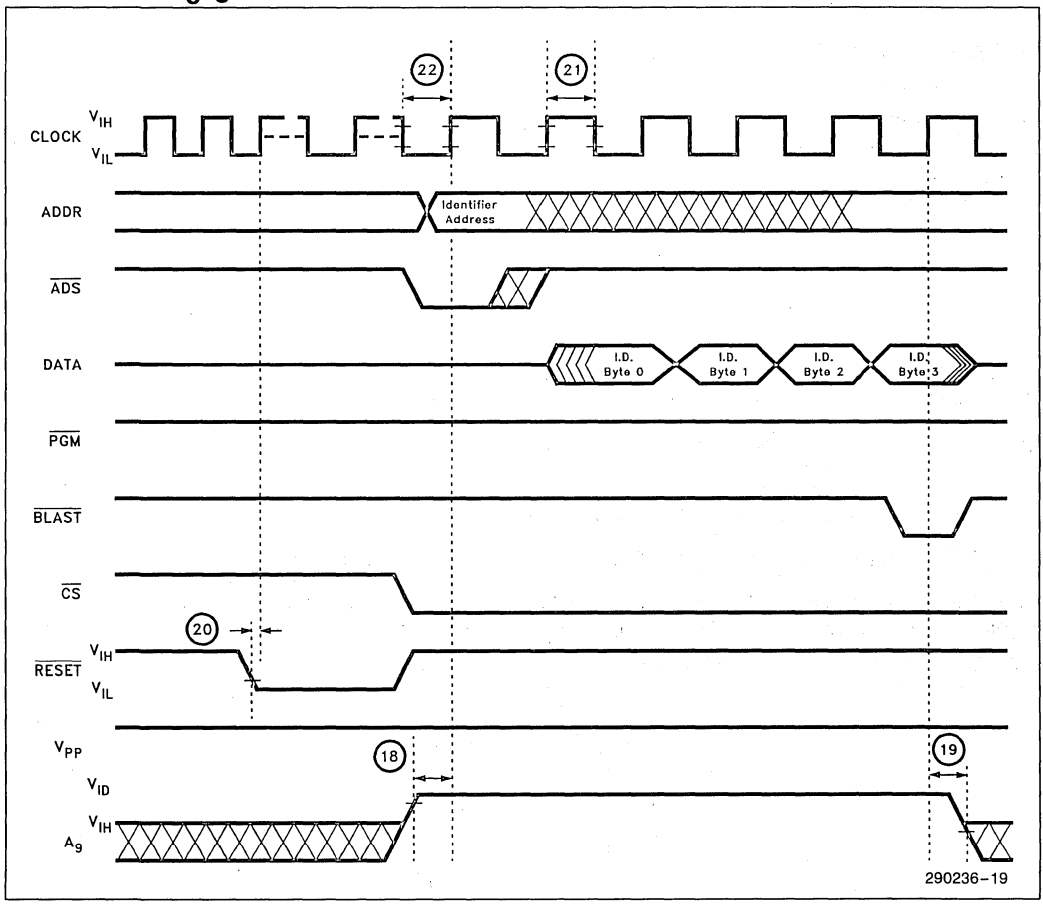


Figure 13. 27960CX RESET and ID Waveforms

27960KX
BURST ACCESS 1M (128K x 8) CHMOS EPROM

- **Synchronous 4-Byte Data Burst Access**
- **Simple Interface to the 80960KA/KB**
- **High Performance Clock to Data Out**
  - **— Zero Wait State Data-to-Data Burst**
  - **— Supports 16, 20 and 25 MHz 80960KA/KB Devices**
- **Asynch Microcontroller Reset Function**
  - **— Returns to Known State with High Z Outputs**
- **CHMOS* III-E for High Performance and Low Power**
  - **— 125 mA Active, 30 mA Standby**
  - **— TTL Compatible Inputs**
- **1 Mbit Density Configures as 128K x 8**

Intel's 27960KX is a 5V only, 1,048,576 bit, Erasable Programmable Read Only Memory, organized as 128K words of 8 bits.

The 27960KX provides a simple synchronous burst interface to the 80960KA/KB bus. Internally the 27960KX is organized in 4 byte blocks, in which each byte is accessed sequentially. The internal state machine is factory configured to generate either 1 or 2 wait-states between the address and first data byte. High performance outputs provide zero wait-state data to data accesses at clock frequencies up to 25 MHz.

An asynchronous microcontroller $\overline{\text{RESET}}$ feature puts the outputs in the high impedance state and takes the internal state machine to a known state where a new burst access can begin.

The 27960KX is available in 44 lead PLCC package, providing optimum cost effectiveness.

The 27960KX is manufactured on Intel's 1 micron CHMOS III-E technology. The Quick-Pulse Programming™ algorithm provides fast, reliable programming with throughput under 17 seconds for optimized equipment.

*CHMOS is a patented process of Intel Corporation.

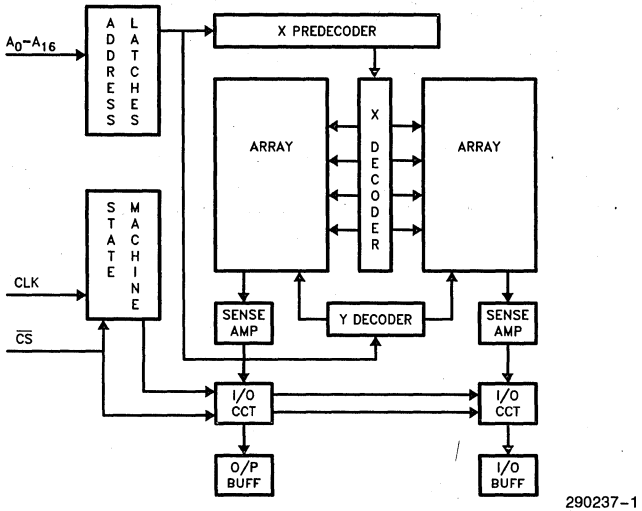


Figure 1. 27960KX Burst EPROM Block Diagram

September 1991
Order Number: 290237-006

27960KX BURST EPROM

EPROMs are established as the preferred code storage device in embedded applications. The non-volatile, flexible, reliable, cost effective EPROM makes a product easier to design, manufacture and service. Until recently, however, EPROMs could not match the performance needs of high-end systems. The 27960KX was designed to support the 80960KA/KB embedded processor. It utilizes the burst interface to offer near zero-wait state performance without the high cost normally associated with this performance.

In embedded designs, board space and cost must be kept at a minimum without impacting performance and reliability. The 27960KX removes the need for expensive high-speed shadow RAM backed up by slow EPROM or ROM for non-volatile code storage. Code optimization concerns are reduced with "off-chip" code fetches no longer crippling to system performance. FONTs can be run directly out of these EPROMs at the same performance as high-speed DRAMs. With the 27960KX, the EPROM is the ideal code or FONT storage device for your 80960KA/KB system.

Architecture

The 27960KX provides a simple, synchronous burst interface to the 80960KA/KB's bus. Internally, the 27960KX is organized in 4 byte blocks each byte is accessed sequentially. A burst access begins on the first clock pulse after $\overline{CS}$ is asserted. The address of the four byte block is latched by the rising edge of $\overline{ALE}$. After a preset number of wait-states (1 or 2), data is output one byte at a time on each subsequent clock cycle. A burst access is terminated on the rising edge of CLOCK if $\overline{BLAST}$ is asserted. High performance outputs provide zero wait-state data to data accesses at clock frequencies up to 25 MHz. Extra power and ground pins dedicated to the outputs reduce the effects of fast output switching on device performance.

The 27960KX delivers 4 bytes of data in 8 clock cycles at 25 MHz and 4 bytes of data in 7 clock cycles at 20 MHz. In a 32-bit configuration, this translates into a read bandwidth of 50 Mbytes/sec and 45 Mbytes/sec respectively. Performance capability of the 27960KX in different 80960KA/KB systems is given in Table 1.

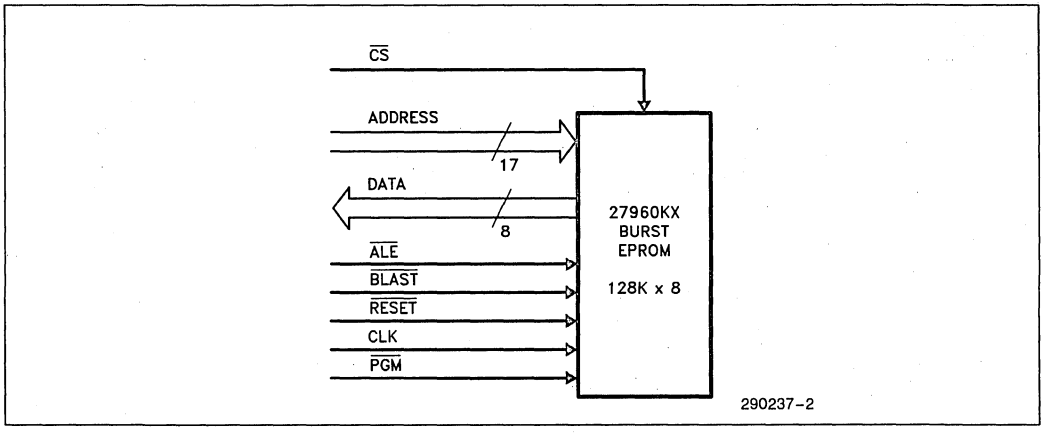


Figure 2. 27960KX Burst EPROM Signal Set

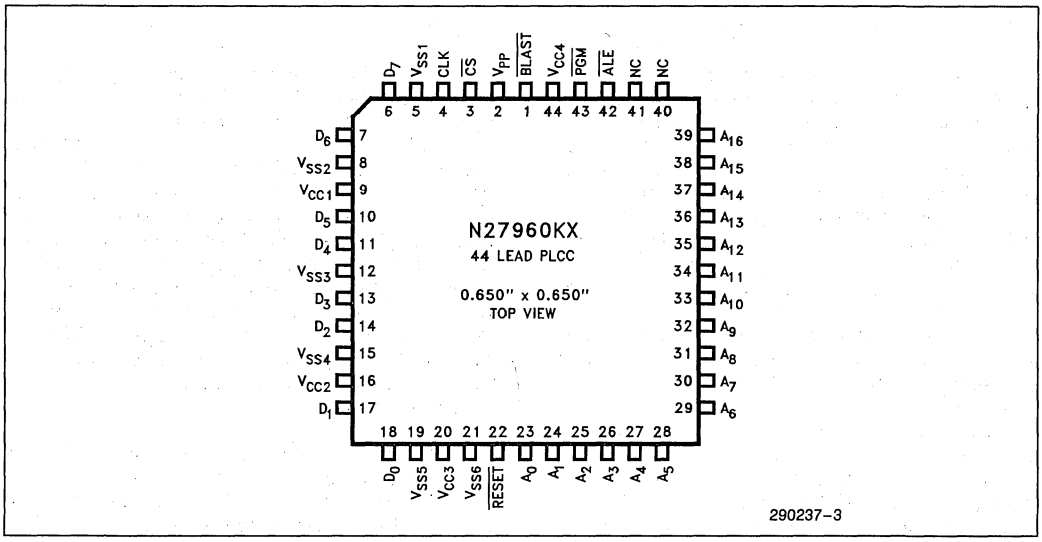


Figure 3. 27960KX 44-Lead PLCC Pinout

PIN DESCRIPTIONS

Symbol	Pin	Function
A_0–A_{16}:	23–39	**ADDRESS INPUTS:** During a burst operation, A_2 through A_{16} provide the base address pointing to a block of four consecutive bytes. A_0 and A_1 select the first byte of the burst access. The 27960KX latches valid addresses in the first clock cycle. An internal address generator increments addresses A_0 and A_1 for subsequent bytes of the burst.
D_0–D_7:	18, 17, 14, 13, 11, 10, 7, 6	**DATA INPUTS/OUTPUTS**
$\overline{ALE}$	42	**ADDRESS LATCH ENABLE:** Indicates the transfer of a physical address. $\overline{ALE}$ is an active low signal used to latch the addresses from the processor. Addresses are latched on the rising edge of $\overline{ALE}$. Valid addresses must be present at or before $\overline{ALE}$ becomes valid.
$\overline{CS}$	3	**CHIP SELECT:** Master device enable. When asserted (active low) data can be written to and read from the device. In read mode, $\overline{CS}$ enables the state machine and the I/O circuitry. **NOTES:** 1. The address decode path is independent of $\overline{CS}$, i.e., X and Y decoding is always powered up. 2. For programming, $\overline{CS}$ should remain low for the entire cycle. Program and verify functions are done one byte at a time. 3. $\overline{CS}$ going high does not terminate a concurrent burst cycle. 4. $\overline{CS}$ must be deasserted between bursts.
$\overline{BLAST}$	1	**BURST LAST:** Terminates a concurrent burst data cycle at the rising edge of the CLK. It must be asserted by the fourth data byte.
$\overline{RESET}$	22	**RESET:** Resets the state machine into a known state, tri-states the outputs. The duration of $\overline{RESET}$ should be 10 CLK cycles minimum. At least 5 clock cycles are required after deassertion of $\overline{RESET}$ before beginning the next cycle. Reset will abort a concurrent bus cycle.

PIN DESCRIPTIONS (Continued)

Symbol	Pin	Function
$\overline{PGM}$	43	PROGRAM-PULSE CONTROL INPUT
V_{PP}	2	PROGRAMMING POWER SUPPLY V_{PP}
V_{SS}	5, 8, 12, 15, 19, 21	GROUND
V_{CC}	9, 16, 20, 44	SUPPLY VOLTAGE INPUT

Table 1. Performance Capability

25/20 MHz 2 WS NON-BUFFERED : 4 WORDS/8 CLOCK CYCLES → 50/40 MBYTES/SEC

ADDR	A_{00}	WS	WS	-	-	-	-	RS	A_{01}	WS	WS	-	-	-	-	RS
DATA	-	-	-	D_{00}	D_{01}	D_{02}	D_{03}	-	-	-	-	D_{10}	D_{11}	D_{12}	D_{13}	
CLK	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8

20 MHz 1 WS NON-BUFFERED : 4 WORDS/7 CLOCK CYCLES → 45 MBYTES/SEC

ADDR	A_{00}	WS	-	-	-	-	RS	A_{01}	WS	-	-	-	-	RS	A_{03}	WS
DATA	-	-	D_{00}	D_{01}	D_{02}	D_{03}	-	-	-	D_{10}	D_{11}	D_{12}	D_{13}			
CLK	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_1	C_2	C_3	C_4	C_5	C_6	C_7		

16 MHz 1 WS BUFFERED : 4 WORDS/7 CLOCK CYCLES → 36 MBYTES/SEC

ADDR	A_{00}	WS	-	-	-	-	RS	A_{01}	WS	-	-	-	-	RS	A_{03}	WS
DATA	-	-	D_{00}	D_{01}	D_{02}	D_{03}	-	-	-	D_{10}	D_{11}	D_{12}	D_{13}			
CLK	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_1	C_2	C_3	C_4	C_5	C_6	C_7		

4

INTERFACE EXAMPLE

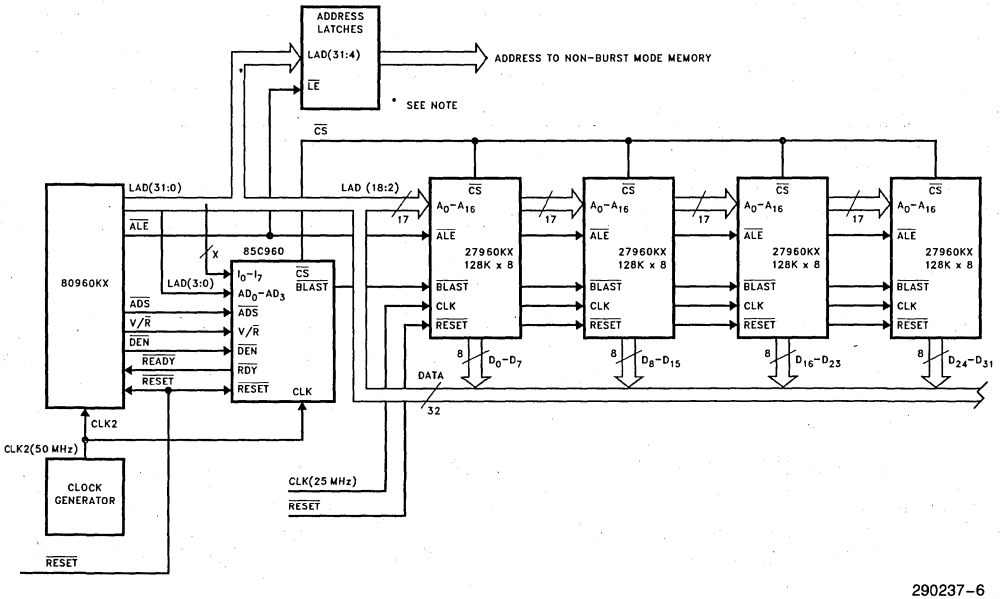
Overview

The following design offers a simple interface to the 80960KA/KB's bus.

A non-buffered 27960KX burst EPROM system is shown in Figure 4. Since the 27960KX is capable of driving a 120 pF load, large, non-buffered systems can be implemented by stacking up to 2 banks of 4 EPROMs, giving a memory size of 256K x 32. The input capacitive load seen on the address lines (due to the EPROM only) is 24 pF for a 128K x 32 system (shown) and 48 pF for a 256K x 32 system. The EPROM is specified at 4 pF for input capacitance and 12 pF typical for output capacitance. Larger systems can be implemented with buffers.

Chip Select Logic

High order address lines are decoded to provide $\overline{CS}$. Qualification with other signals is not required. The chip select logic can be implemented with standard asynchronous decoders, PAL's or PLD's (like Intel's 85C960).



NOTE:
27960KX does not require address latches

Figure 4. 128K x 32 Burst EPROM System

Waveforms

Figure 5 shows the timing waveforms of 27960KX reads in a 32-bit system.

$\overline{CS}$ setup time

$\overline{CS}$ setup time is the time between $\overline{CS}$ asserted and the first rising CLK edge of CLK (during the address cycle). Since a memory access begins on the first CLK rising edge after $\overline{CS}$ asserted, a minimum $\overline{CS}$ setup time of 5 ns (t_{SVCH}) at 25 MHz is required. With the 80960KA/KB's maximum valid address delay of 18 ns at 25 MHz, 13 ns remains for $\overline{CS}$ decoding logic.

$\overline{CS}$ Deassert between bursts

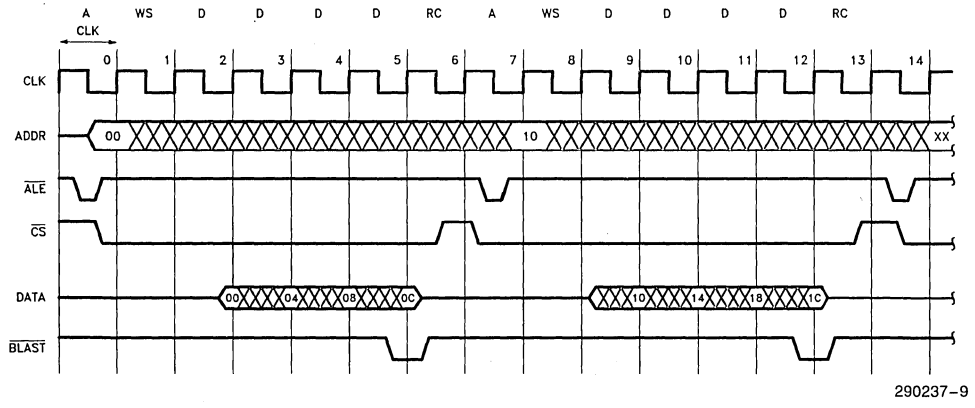
After every EPROM read (one to four words) $\overline{CS}$ must be deasserted.

Reset and $\overline{RESET}$

The 27960KX uses $\overline{RESET}$. The 80960 KA/KB RESET signal must be inverted for the 27960KX.

Clock Phase

The initial rising edge of CLK and CLK2 must be in phase with as small a skew as possible.



NOTES:
1. 1-0-0-0 Burst Read → 1 indicates the number of wait states to access the first word
   0's indicate the number of wait states for subsequent data words (0 in this case)
2. 27960KX latches addresses on the rising edge of ALE: it has an internal address generator which increments addresses for subsequent words of the burst.

Figure 5. Two Cycles of a 27960KX 1 Wait State, 4-Byte Read (1-0-0-0 Burst Read) in a 32-Bit System

27960KX DEVICE NAMES

The device names on the 27960KX were derived as mnemonics that correspond to the number of wait states and expected operating frequency for the device. For example, the 25 MHz, 2 wait state 27960KX is named 27960K2-25.

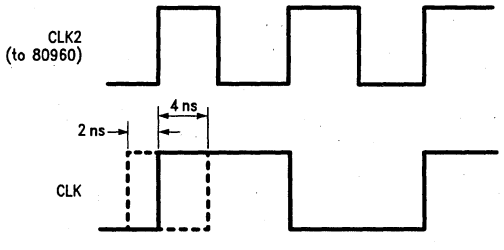
AC TIMING DERIVATIONS

The AC timings for the 27960KX were generated specifically to meet the requirements of the 80960KA/KB microprocessor. In each case the applicable 80960KA/KB clock frequency and AC timing were taken together with an address buffer delay (if needed) and a 4 ns positive clock skew or a 2 ns negative clock skew (see Figure 6A) guardband to generate the 27960KX AC timing. Worst case timings were always assumed. The example below shows how the 27960K1-20 $tavc_0h$ timing was derived.

@20 MHz the clock cycle is ~50 ns.
t_6 of the 80960KA/KB is 2–20 ns.
4 ns clock skew guardband.

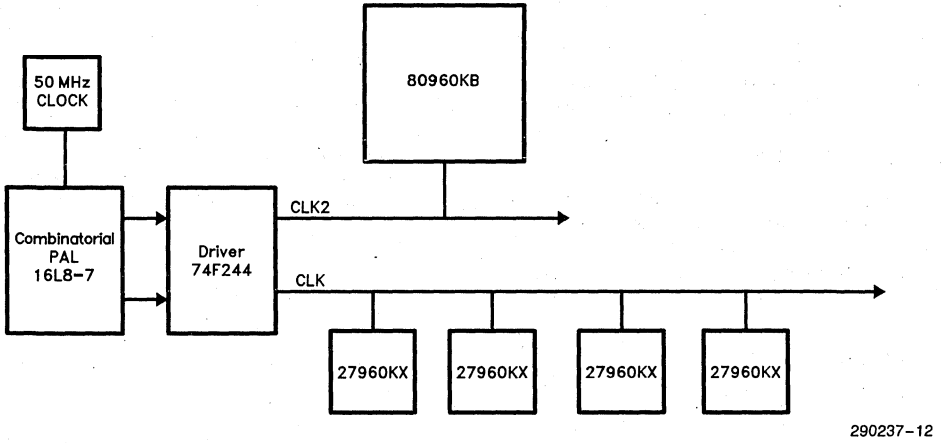
27960K1-20 $tavc_0h$ = 50 ns − 20 ns − 4 ns
= 26 ns

On timings such as this, where the EPROM is faster than the microprocessor, we specified the EPROM's timing leaving the excess time as system guardband.



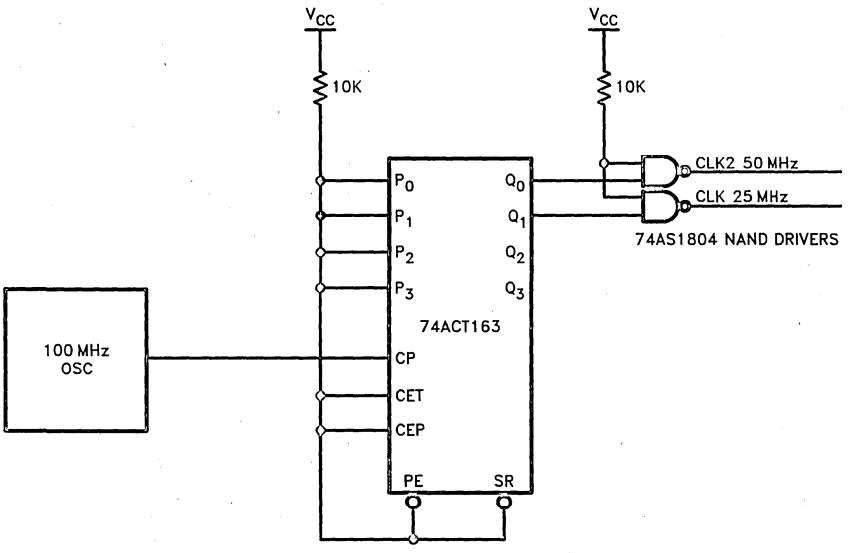
NOTE:
The 27960KX allows a positive clock skew (CLK2 leading CLK) of up to 4 ns and a negative clock skew (CLK2 lagging CLK) of up to 2 ns. The larger positive clock skew takes into account longer trace lengths and heavier loading on the 1x clock trace.

Figure 6A. Definition of Positive and Negative Clock Skew



NOTE:
CLK and CLK2 are generated by the same PAL. This minimizes skew between CLK and CLK2. Both PAL outputs are fed to a 74F244 driver. The EPROMs should be as close to the clock driver as possible.

Figure 6B. Example Clock Circuit with Minimum Skew



NOTE:
This clock generation circuit uses a 100 MHz oscillator. The EPROMs should be as close to the NAND drivers as possible.

Figure 6C. Example Clock Circuit Using a 100 MHz Oscillator

Decoders are needed for the systems address (chip select) decoding. For the 27960KX's timings we assumed a 5–10 ns chip select decoder for 16 MHz and 20 MHz frequencies and a 5–9 ns decoder for 25 MHz systems. The example below shows how the 27960K2-25 tsvch timing was derived.

@25 MHz the clock cycle is ~40 ns.
t_6 of the 80960KA/KB is 2–18 ns.
Decoder = 9 ns
4 ns clock skew guardband

27960K2-25 tsvch = 40 ns − 18 ns − 9 ns − 4 ns
= 9 ns

SYSTEM BUFFERING CONSIDERATIONS

For many large system applications buffering may be required between the microprocessor and memory devices. The 20 MHz − 2 WS and 16 MHz 27960KX AC timings take this into account. For applications at these frequencies not requiring buffering these devices will provide an additional 5–10 ns of system guardband.

The list below shows the buffers used in generating these timings:

	Input Buffer	Output Buffer
20 MHz	9 ns	5 ns
16 MHz	10 ns	7 ns

The 20 MHz buffers are slightly faster in keeping with the increased sensitivity for higher performance. We chose the above buffers because of their wide availability. Significantly faster buffers are available for applications requiring them. The example below shows tchqv for the 27960K2-20.

@20 MHz the clock cycle is ~50 ns.
t_{10} of the 80960KA/KB is 3 ns.
Output buffer for 20 MHz = 5 ns.
4 ns clock skew guardband

27960K2-20 tchqv = 50 ns − 5 ns − 3 ns − 4 ns
= 38 ns

ABSOLUTE MAXIMUM RATINGS*

Read Operating Temperature . . . . . 0°C to +70°C(8)

Case Temperature under Bias . . −10°C to +80°C(8)

Storage Temperature . . . . . −65°C to +125°C

All Input or Output Voltages . . . . . −0.6V to +6.5V(4) with Respect to Ground

Voltage on A_9 . . . . . −0.6V to +13.0V(4) with Respect to Ground

V_{PP} Supply Voltage . . . . . −0.6V to +14.0V(4) with Respect to Ground

V_{CC} Supply Voltage . . . . . −0.6V to +7.0V(4) with Respect to Ground

NOTICE: This data sheet contains preliminary information on new products in production. The specifications are subject to change without notice. Verify with your local Intel Sales office that you have the latest data sheet before finalizing a design.

**WARNING: Stressing the device beyond the "Absolute Maximum Ratings" may cause permanent damage. These are stress ratings only. Operation beyond the "Operating Conditions" is not recommended and extended exposure beyond the "Operating Conditions" may affect device reliability.*

DC CHARACTERISTICS: READ OPERATION

$0°C < T_A < +70°C$, $V_{CC} = 5V \pm 10\%$, TTL Inputs

Symbol	Parameter		Notes	Min	Max	Unit	Test Condition
I_{LI}	Input Load Current				1	μA	$V_{IN} = 5.5V$
I_{LO}	Output Leakage Current				10	μA	$V_{OUT} = 5.5V$
I_{PP}	V_{PP} Load Current Read				10	μA	$V_{PP} = 0$ to V_{CC}, $\overline{PGM} = V_{IH}$
I_{SB}	V_{CC} Standby	Switching	2		45	mA	$\overline{CS} = V_{IH}$, f = 25 MHz
		Stable	2		30	mA	$\overline{CS} = V_{IH}$
I_{CC}	V_{CC} Active Current		1, 3, 7		125	mA	$\overline{CS} = V_{IL}$, f = 25 MHz, $I_{OUT} = 0$ mA
V_{IL}	Input Low Voltage		4	−0.5	0.8	V	
V_{IH}	Input High Voltage			2.0	$V_{CC}+1$	V	
V_{OL}	Output Low Voltage				0.45	V	$I_{OL} = 2.1$ mA
V_{OH}	Output High Voltage		5	$V_{CC}-0.8$		V	$I_{OH} = -100$ μA
			5	2.4		V	$I_{OH} = -400$ μA
I_{OS}	Output Short Circuit		6		100	mA	

NOTES:

1. Maximum current is with outputs unloaded.
2. I_{CC} standby current assumes no output loading, i.e., $I_{OH} = I_{OL} = 0$ mA.
3. I_{CC} is the sum of current through $V_{CC3} + V_{CC4}$ and does not include the current through V_{CC1} and V_{CC2}. (V_{CC1} and V_{CC2} supply power to the output drivers. V_{CC3} and V_{CC4} supply power to the rest of the device.)
4. Minimum DC voltage on input and output pins is −0.5V. During transitions, this level may undershoot to −2.0V for periods less than 20 ns.
5. Maximum DC voltage on input and output pins is $V_{CC} + 0.5V$ which may overshoot to $V_{CC} + 2.0V$ for periods less than 20 ns.
6. One output shorted for no more than one second. I_{OS} is sampled but not 100% tested.
7. I_{CC} max measured with a 10.11 μF capacitor between V_{CC} and V_{SS}.
8. This specification defines commercial product operating temperatures.

EXPLANATION OF AC SYMBOLS

The nomenclature used for timing parameters are as per IEEE STD 662-1980 IEEE Standard Terminology for Semiconductor Memory.

Each timing symbol has five characters. The first is always a "t" (for time). The second character represents a signal name, e.g., (CLK, $\overline{ALE}$, etc.). The third character represents the signal's level (high or low) for the signal indicated by the second character. The fourth character represents a signal name at which a transition occurs marking the end of the time interval being specified.

The fifth character represents the signal level indicated for the fourth character. The list below shows character representations.

A:	Address	R:	$\overline{Reset}$
B:	$\overline{BLAST}$	Q:	Data
C:	Clock	S:	$\overline{CS}$
H:	Logic High Level	t:	Time
L:	$\overline{ALE}$/Logic Low Level	V:	Valid
P:	V_{PP} Programming Voltage	Z:	Tri-state level
X:	No longer a valid "driven" logic level		

AC CHARACTERISTICS: READ OPERATION $0\,°C < T_A < +70°C$, $V_{CC} = 5V \pm 10\%$

Versions				27960K2-25 25 MHz 2 Wait States		27960K1-20 20 MHz 1 Wait State		27960K2-20 20 MHz 2 Wait States		27960K1-16 16 MHz 1 Wait State		Unit
No	**Symbol**	**Characteristic**	**Notes**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**	
1	t_{AVC_0H}	Address Valid to CLK High	CLK 0	12		18		10		15		ns
2	t_{AVLH}	Address Valid to $\overline{ALE}$ High		10		10		10		10		ns
3	t_{LLLH}	$\overline{ALE}$ Low to $\overline{ALE}$ High		12		12		12		12		ns
4	t_{LHAX}	$\overline{ALE}$ High to Address Invalid		8		8		8		8		ns
5	t_{SVCH}	$\overline{CS}$ Valid to CLK High	1, 5	5		8		7		8		ns
6	t_{C_NHSX}	CLK High to $\overline{CS}$ Invalid	2	0		0		0		0		ns
7	t_{CHQV}	CLK High to Data Valid	7		33		43		38		45	ns
8	t_{CHQX}	CLK High to Data Invalid		7		7		7		7		ns
9	t_{CHQZ}	CLK High to Data High-Z	6		30		35		35		35	ns
10	t_{BVCH}	$\overline{BLAST}$ Valid to CLK High		15		15		15		15		ns
11	t_{CHBX}	CLK High to $\overline{BLAST}$ Invalid	3	5	35	5	45	5	45	5	45	ns

NOTES:
1. Valid signal level is meant to be either a logic high or logic low.
2. t_{C_NHSX}—The subscript N represents the number of wait states for this parameter. $\overline{CS}$ can be de-asserted (high) after the number of wait states (N) has expired. The EPROM will continue to burst out data for the current cycle.
3. $\overline{BLAST}$ must be returned high before the next rising clock edge.
4. The sum of t_{CHQV} + t_{AVCH} + NCLK will not equal actual t_{AVQV} if independent test conditions are used to obtain t_{AVCH} and t_{CHQV} (N = number of wait states).
5. $\overline{CS}$ must be deasserted after every burst read (see Figure 7).
6. Sampled, not 100% tested. The transition is measured ±500 mV from steady state voltage.
7. For capacitive loads above 120 pF, t_{CHQV} can be derated by 1 ns/20 pF.

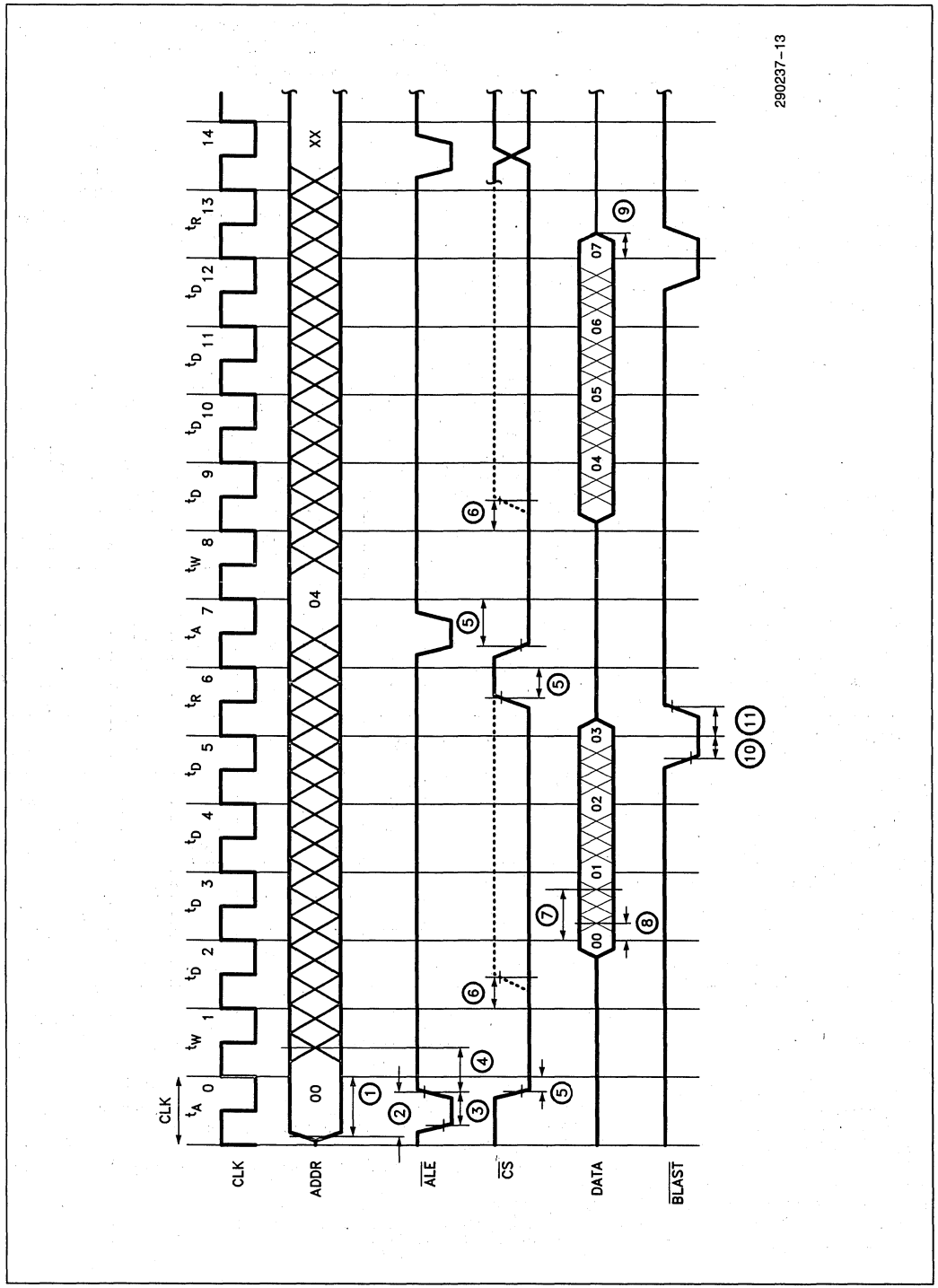


Figure 7. 27960KX 1 WS AC Read Waveforms

AC CONDITIONS OF TEST

Input Rise and Fall Times
(10% to 90%) ..... 4 ns
Input Pulse Levels ..... 0.45V to 2.4V
Input Timing Reference Level 1.5V
Output Timing Reference Level ..... 0.8V and 2.0V

Table 2. Mode Table

MODE	$\overline{CS}$	$\overline{PGM}$	$\overline{BLAST}$	$\overline{ALE}$	$\overline{RESET}$	A_9	V_{PP}	V_{CC}	OUTPUT
Read	V_{IL}	V_{IH}	V_{IH}(1)	V_{IH}(2)	V_{IH}	X(4)	V_{CC}	V_{CC}	D_{OUT}
Standby (6)	V_{IH}	X	X	X	V_{IH}	X	V_{CC}(5)	V_{CC}	High Z
Program	V_{IL}	V_{IL}	V_{IH}	V_{IH}(2)	V_{IH}	X	(3)	(3)	D_{IN}
Program Verify	V_{IL}	V_{IH}	V_{IH}(1)	V_{IH}	V_{IH}	X	(3)	(3)	D_{OUT}
Program Inhibit	V_{IH}	X	X	X	V_{IH}	X	(3)	(3)	High Z
ID Byte 0: Manufacturer	V_{IL}	V_{IH}	V_{IH}(1)	V_{IH}(2)	V_{IH}	V_{ID}(3)	V_{CC}	V_{CC}	89H
ID Byte 1: Part (27960)	V_{IL}	V_{IH}	V_{IH}(1)	V_{IH}(2)	V_{IH}	V_{ID}(3)	V_{CC}	V_{CC}	E0H
ID Byte 2: KX	V_{IL}	V_{IH}	V_{IH}(1)	V_{IH}(2)	V_{IH}	V_{ID}(3)	V_{CC}	V_{CC}	00B
ID Byte 3: 1 Wait-State 2 Wait-States	V_{IL}	V_{IH}	V_{IH}(1)	V_{IH}(2)	V_{IH}	V_{ID}(3)	V_{CC}	V_{CC}	01B 10B
Reset	X	X	X	X	V_{IL}	X	V_{CC}	V_{CC}	High Z

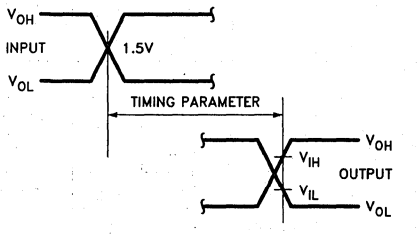
NOTES:
1. V_{IH} until data terminated at which time $\overline{BLAST}$ must go to V_{IL}.
2. Need to toggle from V_{IH} to V_{IL} to V_{IH} to latch address.
3. See DC Programming Characteristics for V_{CC}, V_{ID} and V_{PP} voltages.
4. X can be V_{IL} or V_{IH}.
5. $V_{PP} = V_{CC}$ to meet standby current specification. $V_{CC} > V_{PP} > V_{IL}$ will cause a slight increase in standby current.
6. The device must be in the idle state (by asserting $\overline{RESET}$ or using $\overline{BLAST}$) before going into standby.

CAPACITANCE(1) T_A = 25°C, f = 1.0 MHz

Symbol	Parameter	Typ	Max	Unit	Condition
C_{IN}	Input Capacitance	4	6	pF	V_{IN} = 0V
C_{OUT}	Output Capacitance	12	15	pF	V_{OUT} = 0V
C_{VPP}	V_{PP} Capacitance	40	45	pF	V_{IN} = 0V

NOTE:
1. Sampled, not 100% tested

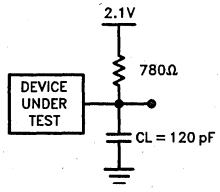
AC INPUT/OUTPUT REFERENCE WAVEFORMS



290237-14

AC test inputs are driven at 2.4V (V_{OH}) for a logic '1' and 0.45V (V_{OL}) for a logic '0'.
Input timing begins at 1.5V.
Output timing ends at V_{IH} (2.0V) and V_{IL} (0.8V)
Input Rise and fall times (10% to 90%) < 4.0 ns

AC TESTING LOAD CIRCUIT



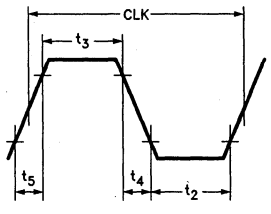
290237-15

For t_{CHQZ} C_L = 5 pF and R_L = 405Ω
C_L includes jig capacitance

CLOCK CHARACTERISTICS

Versions		25 MHz		20 MHz		16 MHz		Units
Symbol	Parameter	Min	Max	Min	Max	Min	Max	
CLK	Period	40		50		62.5		ns
T_5	Rise Time		10		10		10	ns
T_4	Fall Time		10		10		10	ns
T_2	Low Time	7		8		11		ns
T_3	High Time	7		8		11		ns
Max CLK Rise Time during Programming is 100 ns								

CLOCK WAVEFORM



290237-16

Program/Program Verify

Initially, and after each erasure, all bits of the EPROM are in the "1's" state. Data is introduced by selectively programming "0's" into the desired bit locations. Although only "0's" can be programmed, both "1's" and "0's" can be present in the data word. Ultraviolet erasure is the only way to change "0's" to "1's".

Program mode is entered when V_{PP} is raised to 12.75V. Program/Verify operation is synchronous with the clock and can only be initiated following an idle state. Program and Program Verify take place in 3 clock cycles. In the first clock cycle, addresses and data are input and programming occurs. Program Verify follows in the second clock cycle and the third clock cycle terminates synchronous Program/Verify operation, returning the state machine to the idle state with outputs at high impedance.

As in the Read mode, A_2–A_{16} point to a four byte block in the memory array. During Programming the internal address increment circuitry is disabled and the programmer must supply A_0 and A_1 to point to an individual byte within the four byte block that is to be programmed. Only one byte is programmed in each 3 cycle program/Verify sequence.

Program Inhibit

Program Inhibit mode allows parallel programming and verification of multiple devices with different data. With V_{PP} at 12.75V, a Program/Verify sequence is initiated for any device that receives a valid $\overline{ALE}$ pulse and rising clock edge while $\overline{CS}$ is asserted. A $\overline{PGM}$ pulse programs data in the first cycle of the sequence and data for Program Verify is output in the second cycle. The Program/Verify sequence is inhibited on any devices for which $\overline{CS}$ is not asserted during the first ($\overline{ALE}$) cycle. Data will not be programmed and the outputs will remain in their high impedance state.

inteligent Identifier™ Mode

The device's manufacturer, product type, and configuration are stored in a four byte block that can be accessed by using the inteligent Identifier™ mode. The programmer can verify the device identifier and choose the programming algorithm that corresponds to the Intel 27960KX. The inteligent Identifier can also be used to verify that the product is configured with the desired Read mode options for wait states.

Inteligent Identifier mode is entered when A_9 (pin 32) is raised to its high voltage (V_H) level. The internal state machine is then set for inteligent Identifier Read operation. Reading the Identifier is similar to a Read operation on a one wait state configured product. Up to four bytes can be read in a single burst access. inteligent Identifier read is terminated by a synchronous $\overline{BLAST}$ input, returning the state machine to the idle state with outputs at high impedance.

The four byte block code for the inteligent Identifier code is located at address 00H through 03H and is encoded as follows:

MEANING	(A_1, A_0)	DATA
Intel ID	Byte 00	89h
27960	Byte 01	E0h
KX	Byte 10	00b
1 wait state	Byte 11	01b
2 wait states	Byte 11	10b

RESET MODE

Due to the synchronous nature of the 27960KX, the various operating modes must be initiated from a known idle state. During normal operation, the internal state machine returns to an idle state at the termination of a bus access (after $\overline{BLAST}$ is asserted).

During initial device power up, the state machine is in an indeterminant state. The reset mode is provided to force operation in to the idle state. Reset mode is entered when the $\overline{RESET}$ pin is asserted. Output pins are asynchronously set to the high impedance state and address latches are put into the flow through mode. A reset is successfully completed and the state machine set in an idle state in the cycle after $\overline{RESET}$ has been asserted for a minimum of 10 clock cycles and deasserted for five clock cycles.

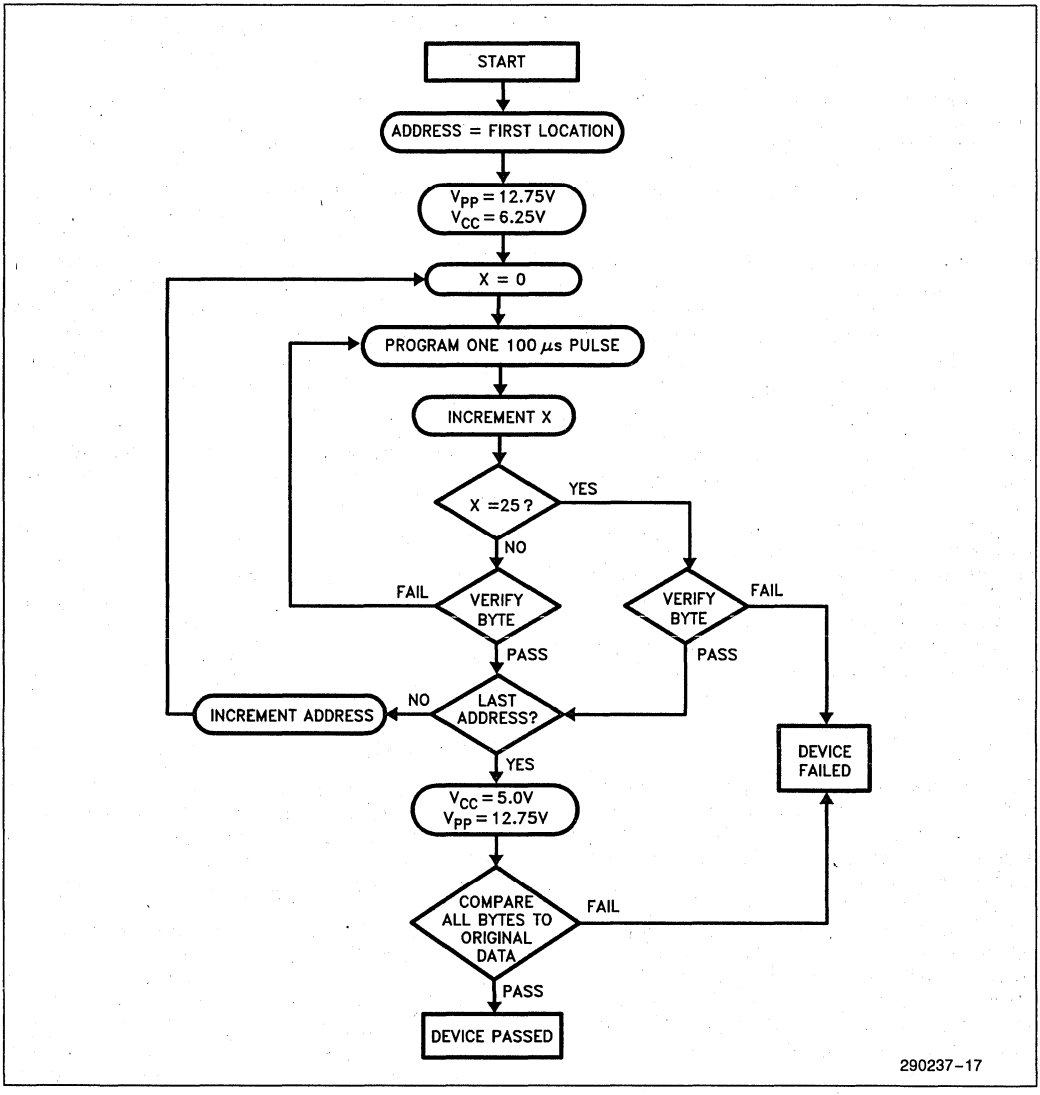


Figure 8. Quick-Pulse Programming™ Algorithm

QUICK-PULSE PROGRAMMING ALGORITHM

The Quick-Pulse Programming algorithm programs Intel's 27960KX. Developed to substantially reduce programming throughput time, this algorithm allows optimized equipment to program a 27960KX in under 17 seconds. Actual programming time depends on the programmer used.

The Quick-Pulse Programming algorithm uses a 100 μs pulse followed by a byte verfication to determine when the addressed byte is correctly programmed. The algorithm terminates if 25 100μs pulses fail to program a byte. Figure 8 shows the 27960KX Quick-Pulse Programming algorithm flowchart.

The entire program-pulse, byte-verify sequence is performed with V_{CC} = 6.25V and V_{PP} = 12.75V. The programming equipment must establish V_{CC} before applying voltages to any other pins. When programming is complete, all bytes should be compared to the original data with V_{CC} = 5.0V and V_{PP} = 12.75V.

D.C. PROGRAMMING CHARACTERISTICS T_A = 25°C ±5°C

Symbol	Parameter	Notes	Min	Max	Unit	Test Condition
I_{LI}	Input Load Current			10	μA	$V_{IN} = V_{IH}$ or V_{IL}
I_{CC}	V_{CC} Program Current	1		125	mA	$\overline{CS} = V_{IL}$
I_{PP}	V_{PP} Program Current	1		50	mA	$\overline{CS} = V_{IL}$
V_{IL}	Input Low Voltage		−0.5	0.8	V	
V_{IH}	Input High Voltage		2.0	V_{CC}+0.5	V	
V_{OL}	Output Low Voltage (Verify)			0.40	V	I_{OL} = 2.1 mA
V_{OH}	Output High Voltage (Verify)		V_{CC}−0.8		V	I_{OH} = −400 μA
V_{ID}	A_9 inteligent Identifier Voltage		11.5	12.5	V	
V_{CC}	Supply Voltage (Program)	2	6.0	6.5	V	
V_{PP}	Program Voltage	2	12.5	13.0	V	

NOTES:
1. The maximum current value is with outputs unloaded.
2. V_{CC} must be applied simultaneously or before V_{PP} and remove simultaneously or after V_{PP}.
3. During programming clock levels are V_{IH} and V_{IL}.

AC PROGRAMMING, RESET AND ID CHARACTERISTICS $T_A = 25°C \pm 5°C$

No	Symbol	Parameter	Notes	Min	Max	Units
1	t_{AVPL}	Address Valid to $\overline{PGM}$ Low		2		µs
2	t_{CHAX}	CLK High to Address Invalid		50		ns
3	t_{LLCH}	$\overline{ALE}$ Low to CLK High	1	50		ns
4	t_{CHLH}	CLK High to $\overline{ALE}$ High	2	50		ns
5	t_{SVCH}	$\overline{CS}$ Valid to CLK High		50		ns
6	t_{CHSX}	CLK High to $\overline{CS}$ Invalid	3			ns
7	t_{CHQV}	CLK High to D_{OUT} Valid			100	ns
8	t_{CHQX}	CLK High to D_{OUT} Invalid		0		ns
9	t_{BVCH}	$\overline{BLAST}$ Valid to CLK High		50		ns
10	t_{CHBX}	CLK High to $\overline{BLAST}$ Invalid	4	50		ns
11	t_{QVPL}	DATA Valid to $\overline{PGM}$ Low		2		µs
12	t_{PLPH}	$\overline{PGM}$ Program Pulse Width		95	105	µs
13	t_{PHQX}	$\overline{PGM}$ High to D_{IN} Invalid		2		µs
14	t_{CLPL}	CLK Low to $\overline{PGM}$ Low		50		ns
15	t_{QZCH}	D_{IN} in Tri-State to CLK High		2		µs
16	t_{VCS}	V_{CC} Program Voltage to CLK High	7	2		µs
17	t_{VPS}	V_{PP} Program Voltage to CLK High	7	2		µs
18	t_{A_9HCH}	A_9 V_{ID} Voltage to CLK High		2		µs
19	t_{CHA_9X}	CLK High to A9 not V_{ID} Voltage		2		µs
20	t_{RVCH}	$\overline{RESET}$ Valid to CLK High	6	50		ns
21	t_{CHCL}	CLK High to CLK Low	5	100		ns
22	t_{CLCH}	CLK Low to CLK High	5	100		ns

NOTES:
1. If $\overline{CS}$ is low, $\overline{ALE}$ can go low no sooner than the falling edge of the previous CLK.
2. $\overline{ALE}$ must return high prior to the next rising edge of clock.
3. $\overline{CS}$ must remain low until after the rising edge CLK1.
4. $\overline{BLAST}$ must return high prior to the next rising edge of CLK.
5. Max CLK rise/fall time is 100 ns.
6. $\overline{RESET}$ must be held low for 10 cycles and high for 5 cycles before performing a read.
7. V_{CC} must be applied simultaneously or before V_{PP} and removed simultaneously or after V_{PP}.

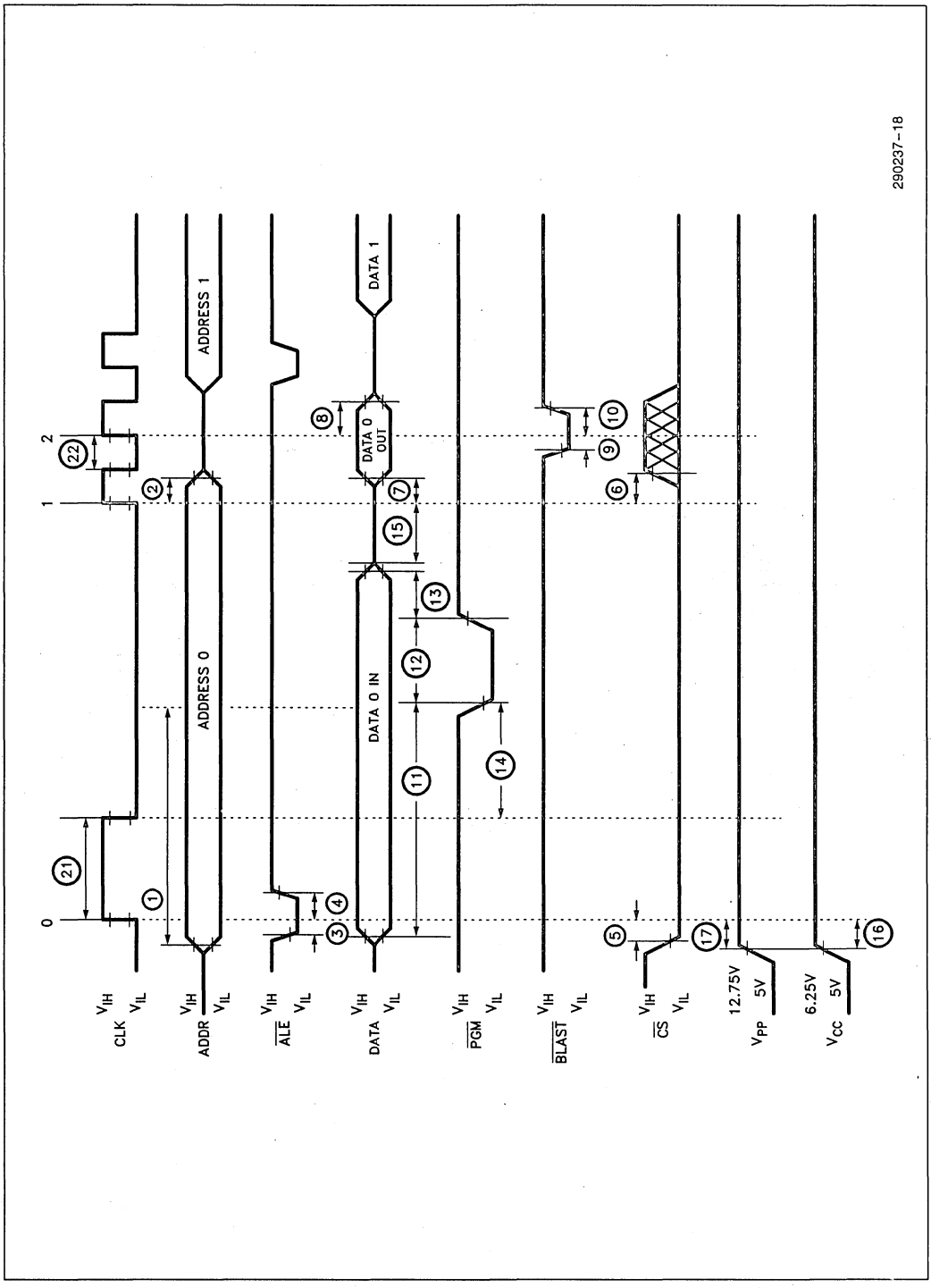


Figure 9. 27960KX Programming Waveforms

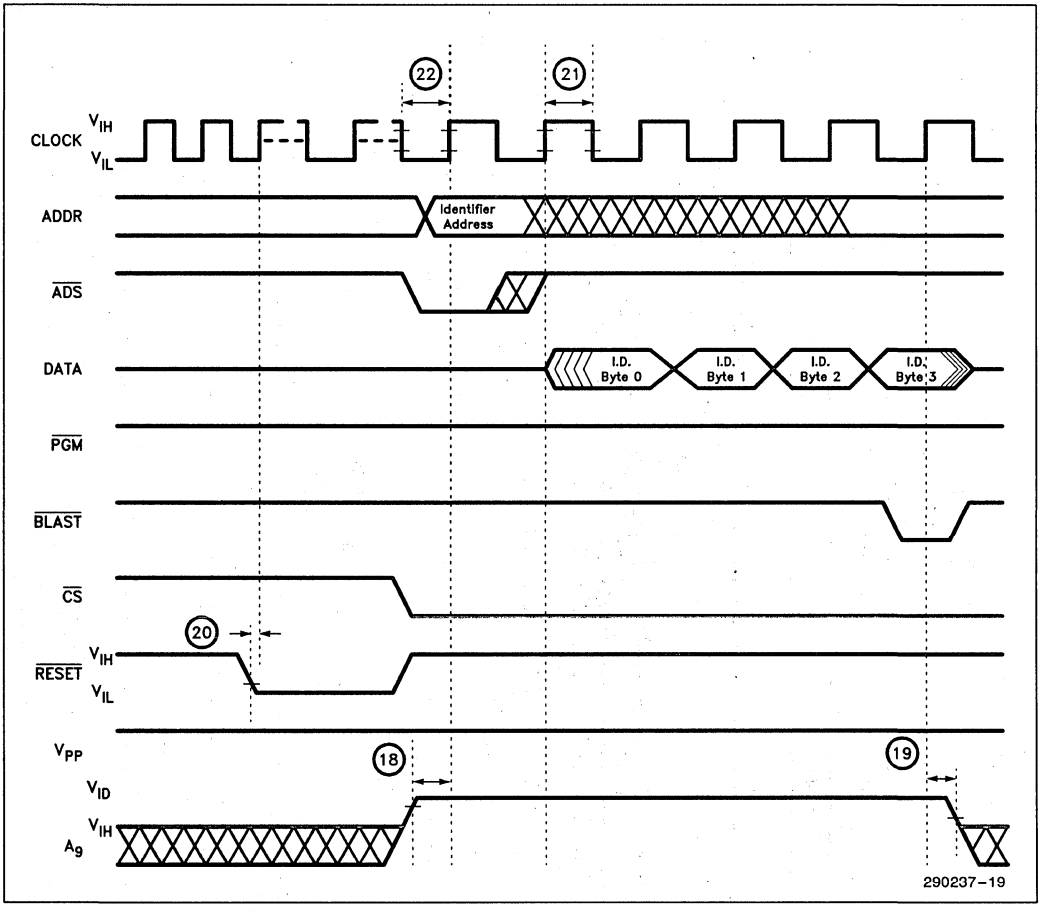


Figure 10. 27960KX RESET and ID Waveforms

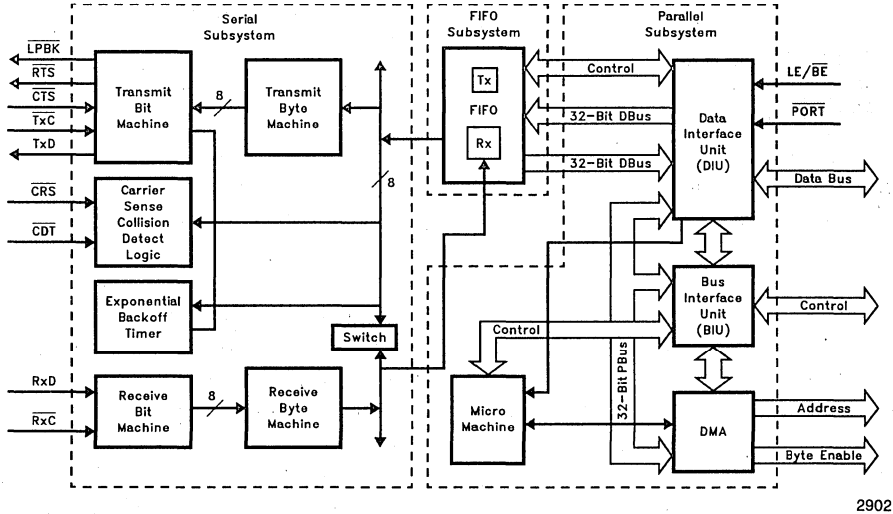
82596CA
HIGH-PERFORMANCE 32-BIT LOCAL AREA NETWORK COPROCESSOR

- Performs Complete CSMA/CD Medium Access Control (MAC) Functions—Independently of CPU
  - —IEEE 802.3 (EOC) Frame Delimiting
  - —HDLC Frame Delimiting
- Supports Industry Standard LANs
  - —IEEE TYPE 10BASE-T, IEEE TYPE 10BASE5 (Ethernet*), IEEE TYPE 10BASE2 (Cheapernet), IEEE TYPE 1BASE5 (StarLAN), and the Proposed Standard 10BASE-F
  - —Proprietary CSMA/CD Networks Up to 20 Mb/s
- On-Chip Memory Management
  - —Automatic Buffer Chaining
  - —Buffer Reclamation after Receipt of Bad Frames; Optional Save Bad Frames
  - —32-Bit Segmented or Linear (Flat) Memory Addressing Formats
- Network Management and Diagnostics
  - —Monitor Mode
  - —32-Bit Statistical Counters
- 82586 Software Compatible
- Optimized CPU Interface
  - —Optimized Bus Interface to Intel's i486™DX, i486™SX and 80960CA Processors
  - —Supports Big Endian and Little Endian Byte Ordering
- 32-Bit Bus Master Interface
  - —106 MB/s Bus Bandwidth
  - —Burst Bus Transfers
  - —Bus Throttle Timers
  - —Transfers Data at 100% of Serial Bandwidth
  - —128-Byte Receive FIFO, 64-Byte Transmit FIFO
- Self-Test Diagnostics
- Configurable Initialization Root for Data Structures
- High-Speed, 5V, CHMOS** IV Technology
- 132-Pin Plastic Quad Flat Pack (PQFP) and PGA Package

  (See Packaging Spec Order No. 240800-001, Package Type KU and A)

i486 is a trademark of Intel Corporation.
*Ethernet is a registered trademark of Xerox Corporation.
**CHMOS is a patented process of Intel Corporation.

4



290218-1

Figure 1. 82596CA Block Diagram

November 1991
Order Number: 290218-004

82596CA High-Performance 32-Bit Local Area Network Coprocessor

CONTENTS

CONTENTS

INTRODUCTION

The 82596CA is an intelligent, high-performance 32-bit Local Area Network coprocessor. The 82596CA implements the CSMA/CD access method and can be configured to support all existing IEEE 802.3 standards—TYPEs 10BASE-T, 10BASE5, 10BASE2, 1BASE5, and 10BROAD36. It can also be used to implement the proposed standard TYPE 10BASE-F. The 82596CA performs high-level commands, command chaining, and interprocessor communications via shared memory, thus relieving the host CPU of many tasks associated with network control. All time-critical functions are performed independently of the CPU, this increases network performance and efficiency. The 82596CA bus interfaces is optimized for Intel's i486™SX, i486™DX, 80960CA, and 80960KB processors.

The 82596CA implements all IEEE 802.3 Medium Access Control and channel interface functions, these include framing, preamble generation and stripping, source address generation, destination address checking, short-frame detection, and automatic length-field handling. Data rates up to 20 Mb/s are supported.

The 82596CA provides a powerful host system interface. It manages memory structures automatically, with command chaining and bidirectional data chaining. An on-chip DMA controller manages four channels, this allows autonomous transfer of data blocks (buffers and frames) and relieves the CPU of byte transfer overhead. Buffers containing errored or collided frames can be automatically recovered without CPU intervention. The 82596CA provides an upgrade path for existing 82586 software drivers by providing an 82586-software-compatible mode that supports the current 82586 memory structure. The 82586CA also has a Flexible memory structure and a Simplified memory structure. The 82596CA can address up to 4 gigabytes of memory. The 82596CA supports Little Endian and Big Endian byte ordering.

The 82596CA bus interface can achieve a burst transfer rate of 106 MB/s at 33 MHz. The bus interface employs bus throttle timers to regulate 82596CA bus use. Two large, independent FIFOs—128 bytes for Receive and 64 bytes for Transmit—tolerate long bus latencies and provide programmable thresholds that allow the user to optimize bus overhead for any worst-case bus latency. The high-performance bus is capable of back-to-back transmission and reception during the IEEE 802.3 9.6-μs Interframe Spacing (IFS) period.

The 82596CA provides a wide range of diagnostics and network management functions, these include internal and external loopback, exception condition tallies, channel activity indicators, optional capture of all frames regardless of destination address (promiscuous mode), optional capture of errored or collided frames, and time domain reflectometry for locating fault points on the network cable. The statistical counters, in 32-bit segmented and linear modes, are 32-bits each and include CRC errors, alignment errors, overrun errors, resource errors, short frames, and received collisions. The 82596CA also features a monitor mode for network analysis. In this mode the 82596CA can capture status bytes, and update statistical counters, of frames monitored on the link without transferring the contents of the frames to memory. This can be done concurrently while transmitting and receiving frames destined for that station.

The 82596CA can be used in both baseband and broadband networks. It can be configured for maximum network efficiency (minimum contention overhead) with networks of any length. Its highly flexible CSMA/CD unit supports address field lengths of zero through six bytes—configurable to either IEEE 802.3/Ethernet or HDLC frame delimitation. It also supports 16- or 32-bit cyclic redundancy checks. The CRC can be transferred directly to memory for receive operations, or dynamically inserted for transmit operations. The CSMA/CD unit can also be configured for full duplex operation for high throughput in point-to-point connections.

82596 B-Stepping

The 82956 B-Step incorporates new features compared to the 82596 A1 stepping. The following is a summary of the 82596 B-step new features.

- The 82596 B-step transmit buffers can now be byte aligned.
- In big endian mode, and when configured to Linear mode, the 82596 B-step treats 32-bit address pointers as big endian 32-bit entities. However, the SCB absolute address and statistical counters are still treated as two 16-bit big endian entities. This big endian 32-bit entity support is configured through the SYSBUS byte; not setting this mode will configure the 82596 B-step to be 100% compatible to the 82596 A1-step big endian mode.
- The 82596 B-step has improved performance on back-to-back frame transmission.
- The 82596 B-step can be configured to reread the next Command Block on the CB list upon receiving a CU RESUME Control Command.

The 82596CA is fabricated with Intel's reliable, 5-V, CHMOS IV (process 648.8) technology. It is available in a 132-pin PQFP or PGA package.

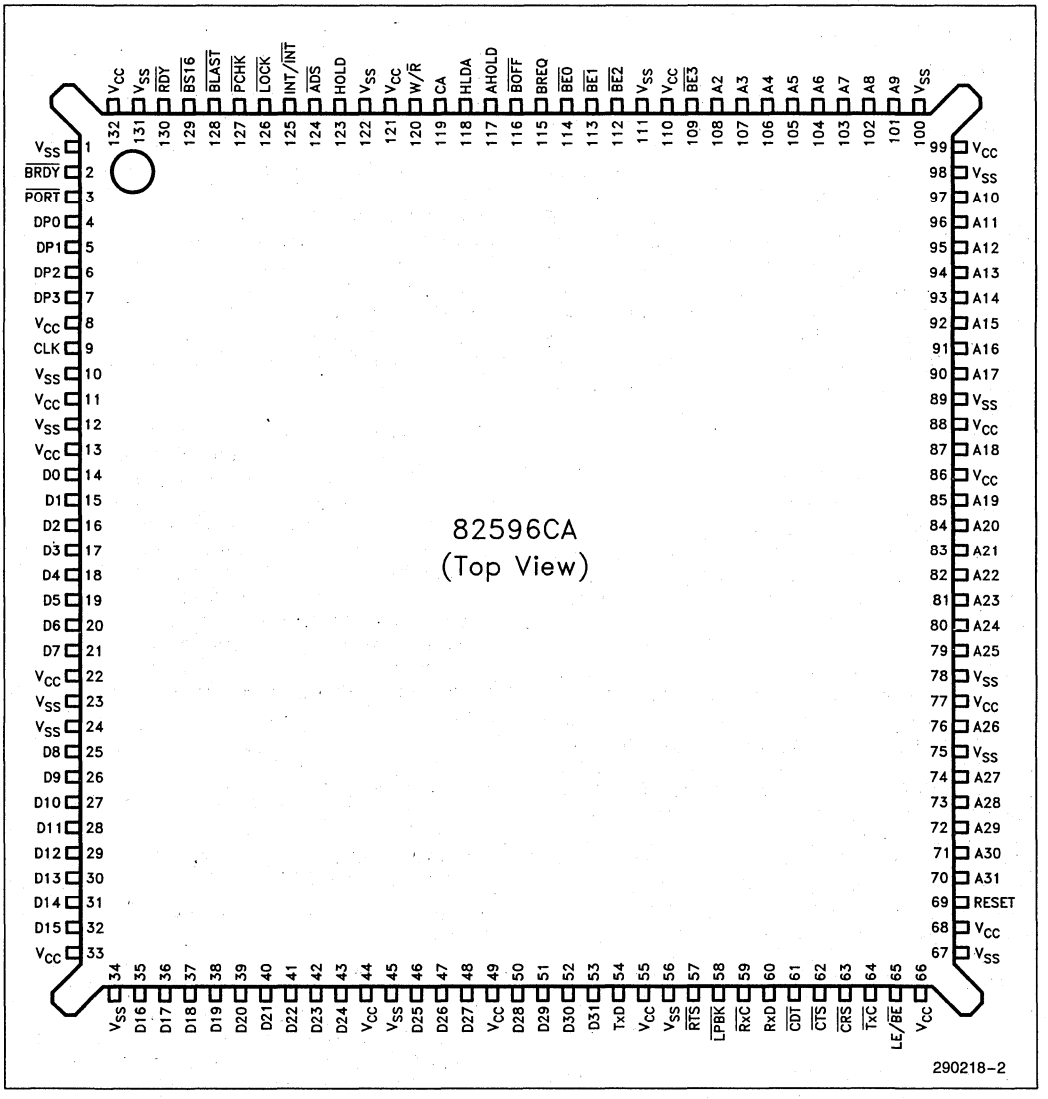


Figure 2. 82596CA PQFP Pin Configuration

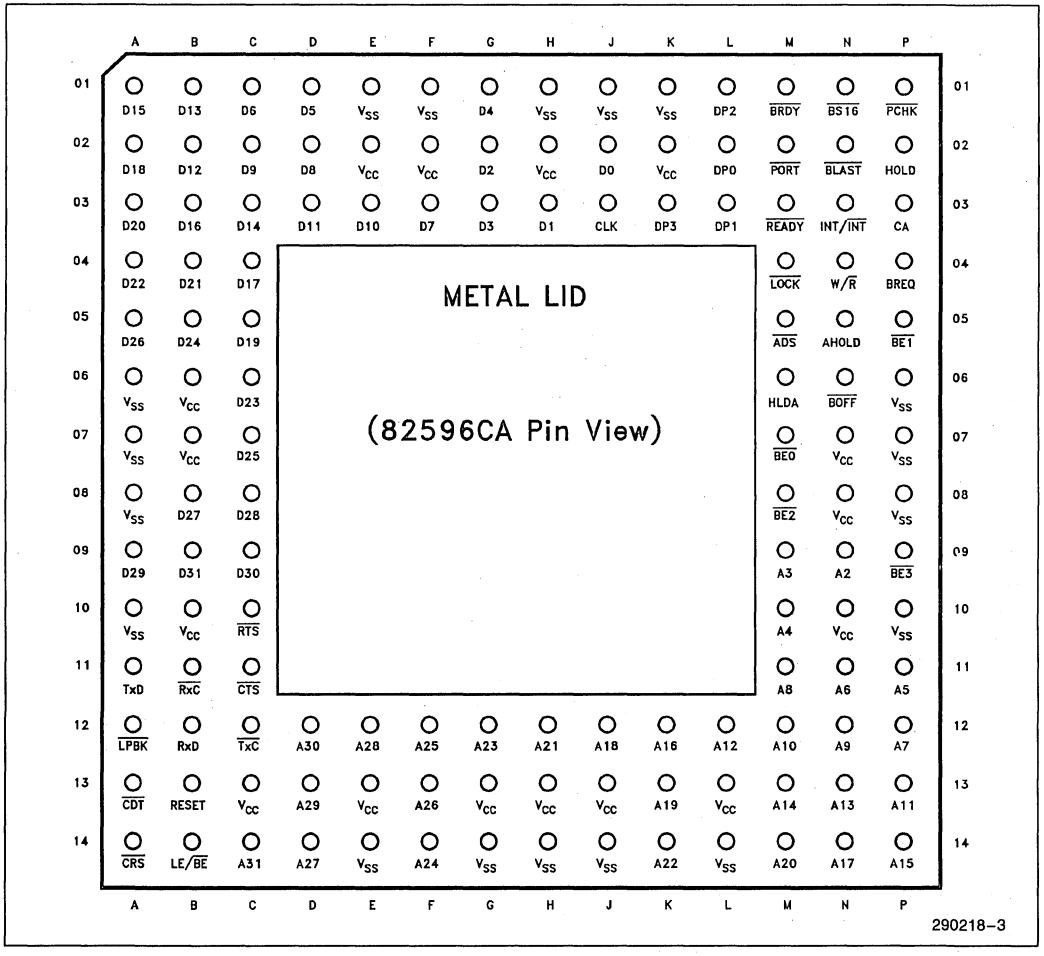


Figure 3. 82596CA PGA Pinout

82596CA PGA Cross Reference by Pin Name

Address		Data		Control		Serial Interface		V_{CC}	V_{SS}
Signal	Pin No.	Signal	Pin No.	Signal	Pin No.	Signal	Pin No.	Pin No.	Pin No.
A2	N9	D0	J2	$\overline{ADS}$	M5	$\overline{CDT}$	A13	B6	A6
A3	M9	D1	H3	AHOLD	N5	$\overline{CRS}$	A14	B7	A7
A4	M10	D2	G2	$\overline{BE0}$	M7	$\overline{CTS}$	C11	B10	A8
A5	P11	D3	G3	$\overline{BE1}$	P5	$\overline{LPBK}$	A12	C13	A10
A6	N11	D4	G1	$\overline{BE2}$	M8	$\overline{RTS}$	C10	E2	E1
A7	P12	D5	D1	$\overline{BE3}$	P9	$\overline{RxC}$	B11	E13	E14
A8	M11	D6	C1	$\overline{BLAST}$	N2	$\overline{RxD}$	B12	F2	F1
A9	N12	D7	F3	$\overline{BOFF}$	N6	$\overline{TxC}$	C12	G13	G14
A10	M12	D8	D2	$\overline{BRDY}$	M1	TxD	A11	H2	H1
A11	P13	D9	C2	BREQ	P4			H13	H14
A12	L12	D10	E3	$\overline{BS16}$	N1			J13	J1
A13	N13	D11	D3	CA	P3			K2	J14
A14	M13	D12	B2	CLK	J3			L13	K1
A15	P14	D13	B1	DP0	L2			N7	L14
A16	K12	D14	C3	DP1	L3			N8	P6
A17	N14	D15	A1	DP2	L1			N10	P7
A18	J12	D16	B3	DP3	K3				P8
A19	K13	D17	C4	HLDA	M6				P10
A20	M14	D18	A2	HOLD	P2				
A21	H12	D19	C5	INT/$\overline{INT}$	N3				
A22	K14	D20	A3	LE/$\overline{BE}$	B14				
A23	G12	D21	B4	$\overline{LOCK}$	M4				
A24	F14	D22	A4	$\overline{PCHK}$	P1				
A25	F12	D23	C6	$\overline{PORT}$	M2				
A26	F13	D24	B5	$\overline{READY}$	M3				
A27	D14	D25	C7	RESET	B13				
A28	E12	D26	A5	W/$\overline{R}$	N4				
A29	D13	D27	B8						
A30	D12	D28	C8						
A31	C14	D29	A9						
		D30	C9						
		D31	B9						

PIN DESCRIPTIONS

<table>
<tr><th>Symbol</th><th>PQFP Pin No.</th><th>Type</th><th>Name and Function</th></tr>
<tr><td>CLK</td><td>9</td><td>I</td><td>CLOCK. The system clock input provides the fundamental timing for the 82596. It is a 1X CLK input used to generate the 82596 clock and requires TTL levels. All external timing parameters are specified in reference to the rising edge of CLK.</td></tr>
<tr><td>D0–D31</td><td>14–53</td><td>I/O</td><td>DATA BUS. The 32 Data Bus lines are bidirectional, tri-state lines that provide the general purpose data path between the 82596 and memory. With the 82596 the bus can be either 16 or 32 bits wide; this is determined by the $\overline{BS16}$ signal. The 82596 always drives all 32 data lines during Write operations, even with a 16-bit bus. D31– D0 are floated after a Reset or when the bus is not acquired.
These lines are inputs during a CPU Port access; in this mode the CPU writes the next address to the 82596 through the data lines. During $\overline{PORT}$ commands (Relocatable SCP, Self-Test, Reset and Dump) the address must be aligned to a 16-byte boundary. This frees the D_3–D_0 lines so they can be used to distinguish the commands. The following is a summary of the decoding data.
<table>
<tr><th>D0</th><th>D1</th><th>D2</th><th>D3</th><th>D31–D4</th><th>Function</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0000</td><td>Reset</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>ADDR</td><td>Relocatable SCP</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td><td>ADDR</td><td>Self-Test</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>0</td><td>ADDR</td><td>Dump Command</td></tr>
</table></td></tr>
<tr><td>DP0–DP3</td><td>4–7</td><td>I/O</td><td>DATA PARITY. These are tri-stated data parity pins. There is one parity line for each byte of the data bus. The 82596 drives them with even-parity information during write operations having the same timing as data writes. Likewise, even-parity information, with the same timing as read information, must be driven back to the 82596 over these pins to ensure that the correct parity check status is indicated by the 82596.</td></tr>
<tr><td>$\overline{PCHK}$</td><td>127</td><td>O</td><td>PARITY CHECK. This pin is driven high one clock after $\overline{RDY}$ to inform Read operations of the parity status of data sampled at the end of the previous clock cycle. When driven low it indicates that incorrect parity data has been sampled. It only checks the parity status of enabled bytes, which are indicated by the Byte Enable and Bus Size signals. $\overline{PCHK}$ is only valid for one clock time after data read is returned to the 82596; i.e., it is inactive (high) at all other times.</td></tr>
<tr><td>A31-A2</td><td>70–108</td><td>O</td><td>ADDRESS LINES. These 30 tri-stated Address lines output the address bits required for memory operation. These lines are floated after a Reset or when the bus is not acquired.</td></tr>
<tr><td>$\overline{BE3}$–$\overline{BE0}$</td><td>109–114</td><td>O</td><td>BYTE ENABLE. These tri-stated signals are used to indicate which bytes are involved with the current memory access. The number of Byte Enable signals asserted indicates the physical size of the data being transferred (1, 2, 3, or 4 bytes).
• $\overline{BE0}$ indicates D7–D0
• $\overline{BE1}$ indicates D15–D8
• $\overline{BE2}$ indicates D23–D16
• $\overline{BE3}$ indicates D31–D24
These lines are floated after a Reset or when the bus is not acquired.</td></tr>
<tr><td>W/$\overline{R}$</td><td>120</td><td>O</td><td>WRITE/READ. This dual function pin is used to distinguish Write and Read cycles. This line is floated after a Reset or when the bus is not acquired.</td></tr>
</table>

PIN DESCRIPTIONS (Continued)

Symbol	PQFP Pin No.	Type	Name and Function
$\overline{ADS}$	124	O	**ADDRESS STATUS.** The 82596 uses this tri-state pin to indicate to indicate that a valid bus cycle has begun and that A31–A2, $\overline{BE3}$–$\overline{BE0}$, and W/$\overline{R}$ are being driven. It is asserted during t1 bus states. This line is floated after a Reset or when the bus is not acquired.
$\overline{RDY}$	130	I	**READY.** Active low. This signal is the acknowledgment from addressed memory that the transfer cycle can be completed. When high, it causes wait states to be inserted. It is ignored at the end of the first clock of the bus cycle's data cycle. This active-low signal does not have an internal pull-up resistor. This signal must meet the setup and hold times to operate correctly.
$\overline{BRDY}$	2	I	**BURST READY.** Active low. Burst Ready, like $\overline{RDY}$, indicates that the external system has presented valid data on the data pins in response to a Read, or that the external system has accepted the 82596 data in response to a Write request. Also, like $\overline{RDY}$, this signal is ignored at the end of the first clock in a bus cycle. If the 82596 can still receive data from the previous cycle, ADS will not be asserted in the next clock cycle; however, Address and Byte Enable will change to reflect the next data item expected by the 82596. $\overline{BRDY}$ will be sampled during each succeeding clock and if active, the data on the pins will be strobed to the 82596 or to external memory (read/write). $\overline{BRDY}$ operates exactly like READY during the last data cycle of a burst sequence and during nonburstable cycles.
$\overline{BLAST}$	128	O	**BURST LAST.** A signal (active low) on this tri-state pin indicates that the burst cycle is finished and when $\overline{BRDY}$ is next returned it will be treated as a normal ready; i.e., another set of addresses will be driven with $\overline{ADS}$ or the bus will go idle. $\overline{BLAST}$ is not asserted if the bus is not acquired.
AHOLD	117	I	**ADDRESS HOLD.** This hold signal is active high, it allows another bus master to access the 82596 address bus. In a system where an 82596 and an i486 processor share the local bus, AHOLD allows the cache controller to make a cache invalidation cycle while the 82596 holds the address lines. In response to a signal on this pin, the 82596 immediately (i.e. during the next clock) stops driving the entire address bus (A31–A2); the rest of the bus can remain active. For example, data can be returned for a previously specified bus cycle during Address Hold. The 82596 will not begin another bus cycle while AHOLD is active.
$\overline{BOFF}$	116	I	**BACKOFF.** This signal is active low, it informs the 82596 that another bus master requires access to the bus before the 82596 bus cycle completes. The 82596 immediately (i.e. during the next clock) floats its bus. Any data returned to the 82596 while $\overline{BOFF}$ is asserted is ignored. $\overline{BOFF}$ has higher priority than $\overline{RDY}$ or $\overline{BRDY}$; if two such signals are returned in the same clock period, $\overline{BOFF}$ is given preference. The 82596 remains in Hold until $\overline{BOFF}$ goes high, then the 82596 resumes its bus cycle by driving out the address and status, and asserting $\overline{ADS}$. $\overline{BOFF}$ should not be asserted during T1.
$\overline{LOCK}$	126	O	**LOCK.** This tri-state pin is used to distinguish locked and unlocked bus cycles. $\overline{LOCK}$ generates a semaphore handshake to the CPU. $\overline{LOCK}$ can be active for several memory cycles, it goes active during the first locked memory cycle (t1) and goes inactive at the last locked cycle (t2). This line is floated after a Reset or when the bus is not acquired. $\overline{LOCK}$ can be disabled via the sysbus byte in software.

PIN DESCRIPTIONS (Continued)

Symbol	PQFP Pin No.	Type	Name and Function
$\overline{BS16}$	129	I	**BUS SIZE.** This signal allows the 82596CA to work with either 16- or 32-bit bytes. Inserting $\overline{BS16}$ low causes the 82596 to perform two 16-bit memory accesses when transferring 32-bit data. In little endian mode the D15–D0 lines are driven when $\overline{BS16}$ is inserted, in Big Endian mode the D31–D16 lines are driven.
HOLD	123	O	**HOLD.** The HOLD signal is active high, the 82596 uses it to request local bus mastership. In normal operation HOLD goes inactive before HLDA. The 82596 can be forced off the bus by deasserting HLDA or if the bus throttle timers expire.
HLDA	118	I	**HOLD ACKNOWLEDGE.** The HLDA signal is active high, it indicates that bus mastership has been given to the 82596. HLDA is internally synchronized; after HOLD is detected low, the CPU drives HLDA low. **NOTE:** *Do not connect HLDA to V_{CC}—it will cause a deadlock.* A user wanting to give the 82596 permanent access to the bus should connect HLDA to HOLD. If HLDA goes inactive before HOLD, the 82596 will release the bus (by deasserting HOLD) within a maximum of within a specified number of bus cycles as specified in the 82596 User's Manual.
BREQ	115	I	**BUS REQUEST.** This signal, when configured to an externally activated mode, is used to trigger the bus throttle timers.
$\overline{PORT}$	3	I	**PORT.** When this signal is received, the 82596 latches the data on the data bus into an internal 32-bit register. When the CPU is asserting this signal it can write into the 82596 (via the data bus). This pin must be activated twice during all CPU Port access commands.
RESET	69	I	**RESET.** This active high, internally synchronized signal causes the 82596 to terminate current activity. The signal must be high for at least five system clock cycles. After five system clock cycles and four $\overline{TxC}$ clock cycles the 82596 will execute a Reset when it receives a high RESET signal. When RESET returns to low the 82596 waits for the first CA signal and then begins the initialization sequence.
LE/$\overline{BE}$	65	I	**LITTLE ENDIAN/BIG ENDIAN.** This dual-function pin is used to select byte ordering. When LE/$\overline{BE}$ is high, little endian byte ordering is used; when low, big endian byte ordering is used for data in frames (bytes) and for control (SCB, RFD, CBL, etc).
CA	119	I	**CHANNEL ATTENTION.** The CPU uses this pin to force the 82596 to begin executing memory resident Command blocks. The CA signal is internally synchronized. The signal must be high for at least one system clock. It is latched internally on the high to low edge and then detected by the 82596. The first CA after a Reset forces the 82596 into the initialization sequence beginning at location 00FFFFF6h or an SCP address written to the 82596 using CPU Port access. All subsequent CA signals cause the 82596 to begin executing new command sequences from the SCB.
INT/$\overline{INT}$	125	O	**INTERRUPT.** A high signal on this pin notifies the CPU that the 82596 is requesting an interrupt. This signal is an edge triggered interrupt signal, and can be configured to be active high or low.

PIN DESCRIPTIONS (Continued)

Symbol	PQFP Pin No.	Type	Name and Function
V_{CC}	17 Pins		**POWER.** +5 V ±10%.
V_{SS}	17 Pins		**GROUND.** 0 V.
TxD	54	O	**TRANSMIT DATA.** This pin transmits data to the serial link. It is high when not transmitting.
$\overline{TxC}$	64	I	**TRANSMIT CLOCK.** This signal provides the fundamental timing for the serial subsystem. The clock is also used to transmit data synchronously on the TxD pin. For NRZ encoding, data is transferred to the TxD pin on the high to low clock transition. For Manchester encoding, the transmitted bit center is aligned with the low to high transition. Transmit clock must always be running for proper device operation.
$\overline{LPBK}$	58	O	**LOOPBACK.** This TTL-level control signal enables the loopback mode. In this mode serial data on the TxD input is routed through the 82C501 internal circuits and back to the RxD output without driving the transceiver cable. To enable this signal, both internal and external loopback need to be set with the Configure command.
RxD	60	I	**RECEIVE DATA.** This pin receives NRZ serial data only. It must be high when not receiving.
$\overline{RxC}$	59	I	**RECEIVE CLOCK.** This signal provides timing information to the internal shifting logic. For NRZ data the state of the RxD pin is sampled on the high to low transition of the clock.
$\overline{RTS}$	57	O	**REQUEST TO SEND.** When this signal is low the 82596 informs the external interface that it has data to transmit. It is forced high after a Reset or when transmission is stopped.
$\overline{CTS}$	62	I	**CLEAR TO SEND.** An active-low signal that enables the 82596 to send data. It is normally used as an interface handshake to $\overline{RTS}$. Asserting $\overline{CTS}$ high stops transmission. $\overline{CTS}$ is internally synchronized. If $\overline{CTS}$ goes inactive, meeting the setup time to the $\overline{TxC}$ negative edge, the transmission will stop and $\overline{RTS}$ will go inactive within, at most, two $\overline{TxC}$ cycles.
$\overline{CRS}$	63	I	**CARRIER SENSE.** This signal is active low, it is used to notify the 82596 that traffic is on the serial link. It is only used if the 82596 is configured for external Carrier Sense. In this configuration external circuitry is required for detecting traffic on the serial link. $\overline{CRS}$ is internally synchronized. To be accepted, the signal must remain active for at least two serial clock cycles (for CRSF = 0).
$\overline{CDT}$	61	I	**COLLISION DETECT.** This active-low signal informs the 82596 that a collision has occurred. It is only used if the 82596 is configured for external Collision Detect. External circuitry is required for collision detection. $\overline{CDT}$ is internally synchronized. To be accepted, the signal must remain active for at least two serial clock cycles (for CDTF = 0).

82596 AND HOST CPU INTERACTION

The 82596CA and the host CPU communicate through shared memory. Because of its on-chip DMA capability, the 82596 can make data block transfers (buffers and frames) independently of the CPU; this greatly reduces the CPU byte transfer overhead.

The 82596 is a multitasking coprocessor that comprises two independent logical units—the Command Unit (CU) and the Receive Unit (RU). The CU executes commands from shared memory. The RU handles all activities related to frame reception. The independence of the CU and RU enables the 82596 to engage in both activities simultaneously—the CU can fetch and execute commands from memory while the RU is storing received frames in memory. The CPU is only involved with this process after the CU has executed a sequence of commands or the RU has finished storing a sequence of frames.

The CPU and the 82596 use the hardware signals Interrupt (INT) and Channel Attention (CA) to initiate communication with the System Control Block (SCB), see Figure 4. The 82596 uses INT to alert the CPU of a change in the contents of the SCB, the CPU uses CA to alert the 82596.

The 82596 has a CPU Port Access state that allows the CPU to execute certain functions without accessing memory. The 82596 $\overline{\text{PORT}}$ pin and data bus pins are used to enable this feature. The CPU can directly activate four operations when the 82596 is in this state.

- Write an alternative System Configuration Pointer (SCP). This can be used when the 82596 cannot use the default SCP address space.
- Write a different Dump Command Pointer and execute Dump. This can be used for troubleshooting No Response problems.
- The CPU can reset the 82596 via software without disturbing the rest of the system.
- A self-test can be used for board testing; the 82596 will execute a self-test and write the results to memory.

82596 BUS INTERFACE

The 82596CA has bus interface timings and pin definitions that are compatible with Intel's 32-bit i486™SX and i486™DX microprocessors. This eliminates the need for additional bus interface logic. Operating at 33 MHz, the 82596's bus bandwidth can be as high as 106 MB/s. Since Ethernet only requires 1.25 MB/s, this leaves a considerable amount of bandwidth for the CPU. The 82596 also has a bus throttle to regulate its use of the bus. Two timers can be programmed through the SCB: one controls the maximum time the 82596 can remain on the bus, the other controls the time the 82596 must stay off the bus (see Figure 5). The bus throttle can be programmed to trigger internally with HLDA or externally with BREQ. These timers can restrict the 82596 HOLD activation time and improve bus utilization.

82596 MEMORY ADDRESSING

The 82596 has a 32-bit memory address range, which allows addressing up to four gigabytes of memory. The 82596 has three memory addressing modes (see Table 1).

- **82586 Mode.** The 82596 has a 24-bit memory address range. The System Control Block, Command List, Receive Descriptor List, and Buffer Descriptors must reside in one 64-KB memory segment. Transmit and Receive buffers can reside in a 24-bit address space.
- **32-Bit Segmented Mode.** The 82596 has a 32-bit memory address range. The System Control Block, Command List, Receive Descriptor List, and Buffer Descriptors must reside in one 64-KB memory segment. Transmit and Receive buffers can reside in a 32-bit address space.
- **Linear Mode.** The 82596 has a 32-bit memory address range. Any memory structure can reside anywhere within the 32-bit memory address range.

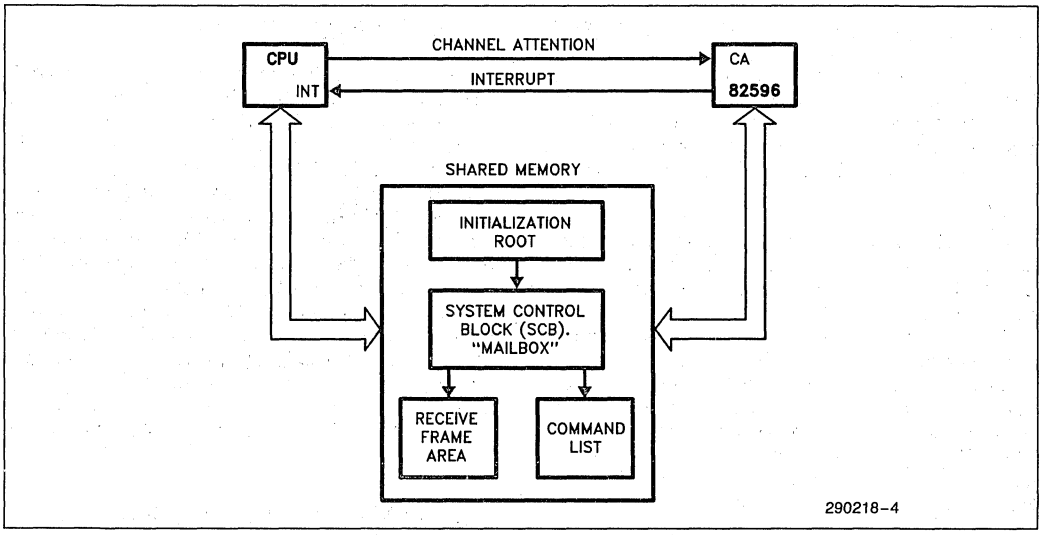


Figure 4. 82596 and Host CPU Intervention

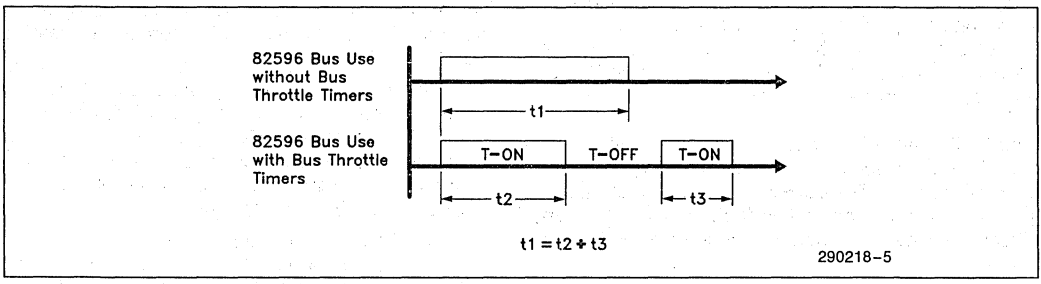


Figure 5. Bus Throttle Timers

Table 1. 82596 Memory Addressing Formats

Pointer or Offset	Operation Mode		
	82586	32-Bit Segmented	Linear
ISCP Address	24-Bit Linear	32-Bit Linear	32-Bit Linear
SCB Address	Base (24) + Offset (16)	Base (32) + Offset (16)	32-Bit Linear
Command Block Pointers	Base (24) + Offset (16)	Base (32) + Offset (16)	32-Bit Linear
Rx Frame Descriptors	Base (24) + Offset (16)	Base (32) + Offset (16)	32-Bit Linear
Tx Frame Descriptors	Base (24) + Offset (16)	Base (32) + Offset (16)	32-Bit Linear
Rx Buffer Descriptors	Base (24) + Offset (16)	Base (32) + Offset (16)	32-Bit Linear
Tx Buffer Descriptors	Base (24) + Offset (16)	Base (32) + Offset (16)	32-Bit Linear
Rx Buffers	24-Bit Linear	32-Bit Linear	32-Bit Linear
Tx Buffers	24-Bit Linear	32-Bit Linear	32-Bit Linear

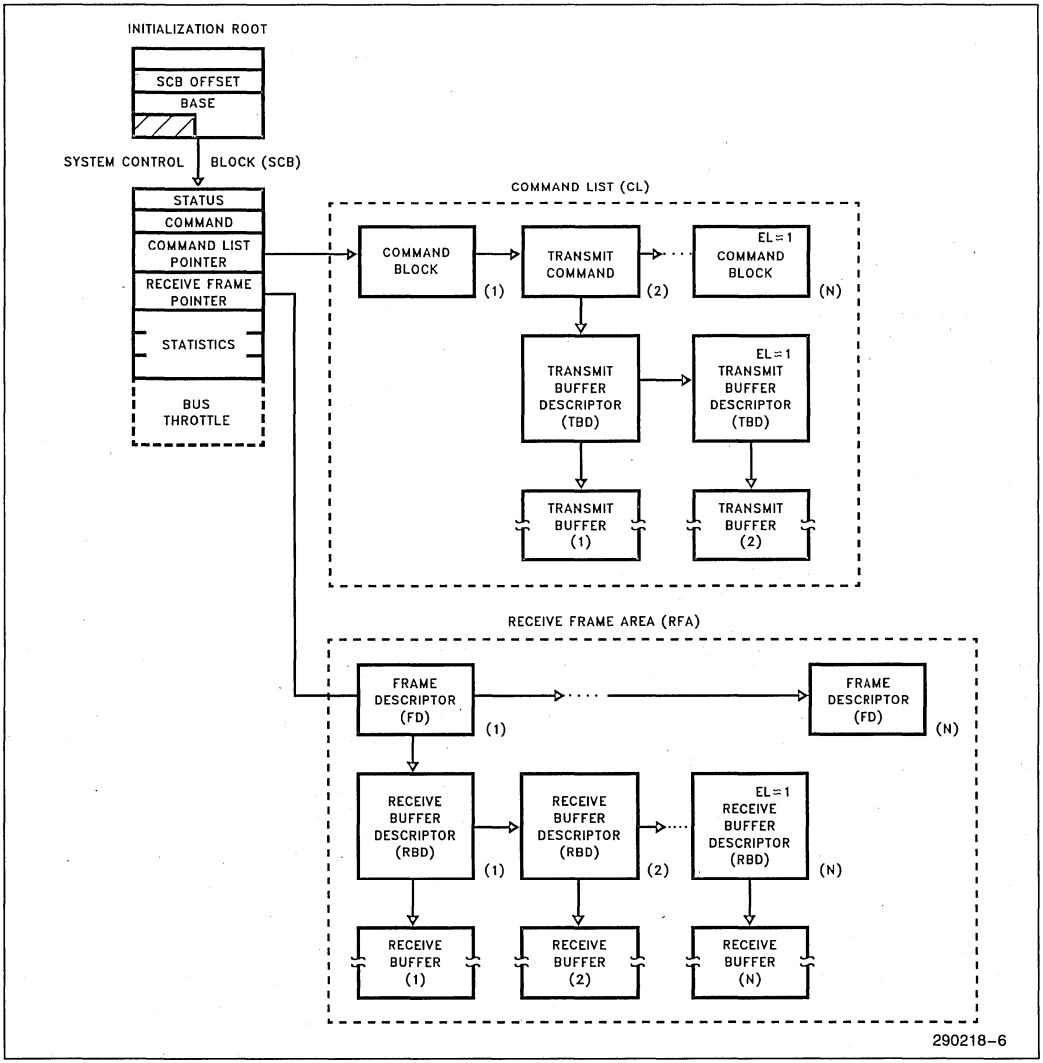


Figure 6. 82596 Shared Memory Structure

82596 SYSTEM MEMORY STRUCTURE

The Shared Memory structure consists of four parts: the Initialization Root, the System Control Block, the Command List, and the Receive Frame Area (see Figure 6).

The Initialization Root is in an established location known to the host CPU and the 82596 (00FFFFF6h). However, the CPU can establish the Initialization Root in another location by using the CPU Port access. This root is accessed during initialization, and points to the System Control Block.

The System Control Block serves as a bidirectional mail drop for the host CPU and the 82596 CU and RU. It is the central point through which the CPU and the 82596 exchange control and status information. The SCB has two areas. The first contains instructions from the CPU to the 82596. These include: control of the CU and RU (Start, Abort, Suspend, and Resume), a pointer to the list of CU commands, a pointer to the Receive Frame Area, a set of Interrupt Acknowledge bits, and the T-ON and T-OFF timers for the bus throttle. The second area contains status information the 82596 is sending to the CPU. Such as, the CU and RU states (Idle, Active

Ready, Suspended, No Receive Resources, etc.), interrupt bits (Command Completed, Frame Received, CU Not Ready, and RU Not Ready), and statistical counters.

The Command List functions as a program for the CU; individual commands are placed in memory units called Command Blocks (CBs). These CBs contain the parameters and status of specific high-level commands called Action Commands; e.g., Transmit or Configure.

Transmit causes the 82596 to transmit a frame. The Transmit CB contains the destination address, the length field, and a pointer to a list of linked buffers holding the frame that is to be constructed from several buffers scattered throughout memory. The Command Unit operates without CPU intervention; the DMA for each buffer, and the prefetching of references to new buffers, is performed in parallel. The CPU is notified only after a transmission is complete.

The Receive Frame Area is a list of Free Frame Descriptors (descriptors not yet used) and a list of user-prepared buffers. Frames arrive at the 82596 unsolicited; the 82596 must always be ready to receive and store them in the Free Frame Area. The Receive Unit fills the buffers when it receives frames, and reformats the Free Buffer List into received-frame structures. The frame structure is, for all practical purposes, identical to the format of the frame to be transmitted. The first Frame descriptor is referenced by the SCB. Unless the 82596 is configured to Save Bad Frames, the frame descriptor, and the associated buffer descriptor, which is wasted when a bad frame is received, are automatically reclaimed and returned to the Free Buffer List.

Receive buffer chaining (storing incoming frames in a linked buffer list) significantly improves memory utilization. Without buffer chaining, the user must allocate consecutive blocks of memory, each capable of containing a maximum frame (for Ethernet, 1518 bytes). Since an average frame is about 200 bytes, this is very inefficient. With buffer chaining, the user can allocate small buffers and the 82596 will only use those that are needed.

Figure 7 A–D illustrates how the 82596 uses the Receive Frame Area. Figure 7A shows an unused Receive Frame Area composed of Free Frame Descriptors and Free Receive Buffers prepared by the user. The SCB points to the first Frame Descriptor of the Frame Descriptor List. Figure 7B shows the same Receive Frame Area after receiving one frame. This first frame occupies two Receive Buffers and one Frame Descriptor—a valid received frame will only occupy one Frame Descriptor. After receiving this frame the 82596 sets the next Free Frame Descriptor RBD pointer to the next Free RBD. Figure 7C shows the RFA after receiving a second frame. In this example the second frame occupies only one Receive Buffer and one RFD. The 82596 again sets the RBD pointer. This process is repeated again in Figure 7D, showing the reception of another frame using one Receive Buffer; in this example there is an extra Frame Descriptor.

TRANSMIT AND RECEIVE MEMORY STRUCTURES

There are three memory structures for reception and transmission. The 82586 memory structure, the Flexible memory structure, and the Simplified memory structure. The 82586 mode is selected by configuring the 82596 during initialization. In this mode all the 82596 memory structures are compatible with the 82586 memory structures.

When the 82596 is not configured to the 82586 mode, the other two memory structures, Simplified and Flexible, are available for transmitting and receiving. These structures can be selected on a frame-by-frame basis by setting the S/F bit in the Transmit Command and the Receive Frame Descriptor (see Figures 29, 30, 41, and 42). The Simplified memory structure offers a simple structure for ease of programming (see Figure 8). All information about a frame is contained in one structure; for example, during reception the RFD and data field are contained in one structure.

The Flexible memory structure (see Figure 9) has a control field that allows the programmer to specify the amount of receive data the RFD will contain for receive operations and the amount of transmit data the Transmit Command Block will contain for transmit operations. For example, when the control field in the RFD is set to 20 bytes during a reception, the first 20 bytes of the data field are stored in the RFD (6 bytes of destination address, 6 bytes of source address, 2 bytes of length field, and 6 bytes of data) and the remainder of the data field is stored in the Receive Data Buffers. This is useful for capturing frame headers when header information is contained in the data field. The header information can then be automatically stored in the RFD partitioned from the Receive Data Buffer.

The control field can also be used for the Transmit Command when the Flexible memory structure is used. The quantity of data field bytes to be transmitted from the Transmit Command Block is specified by the variable control field.

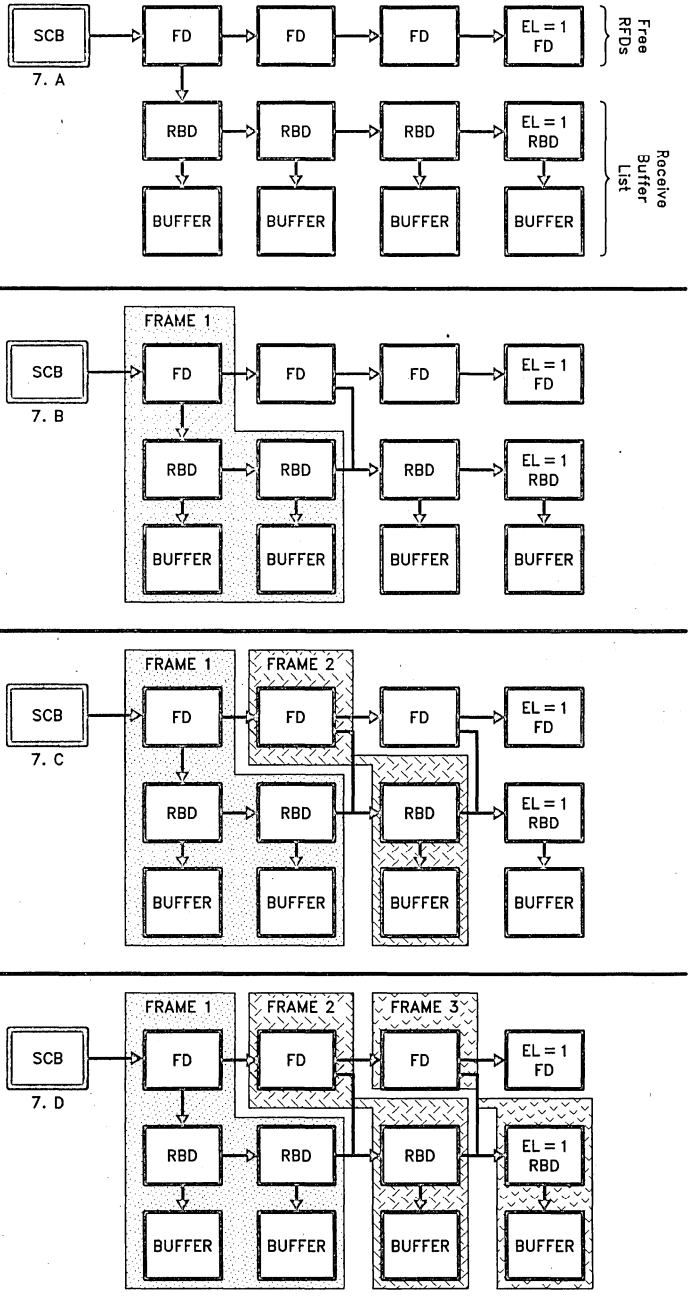


Figure 7. Frame Reception in the RFA

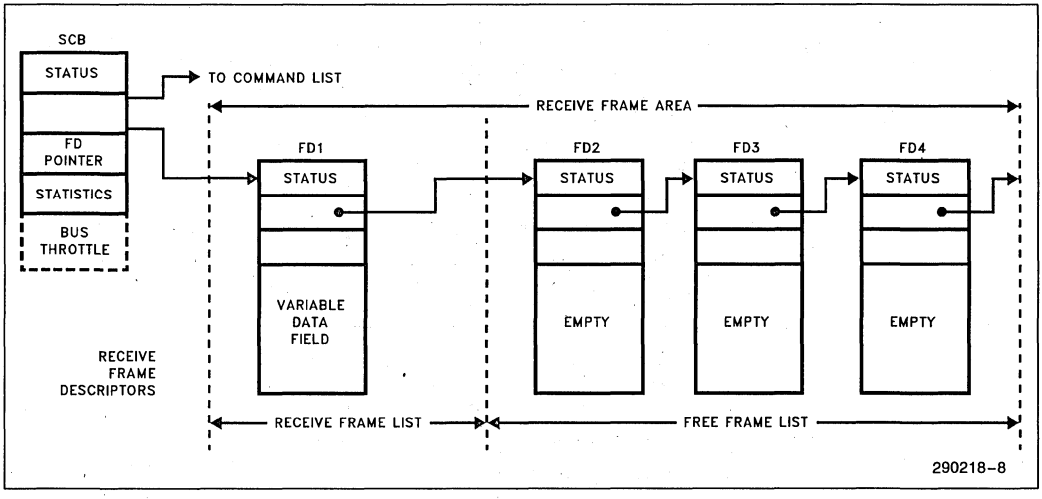


Figure 8. Simplified Memory Structure

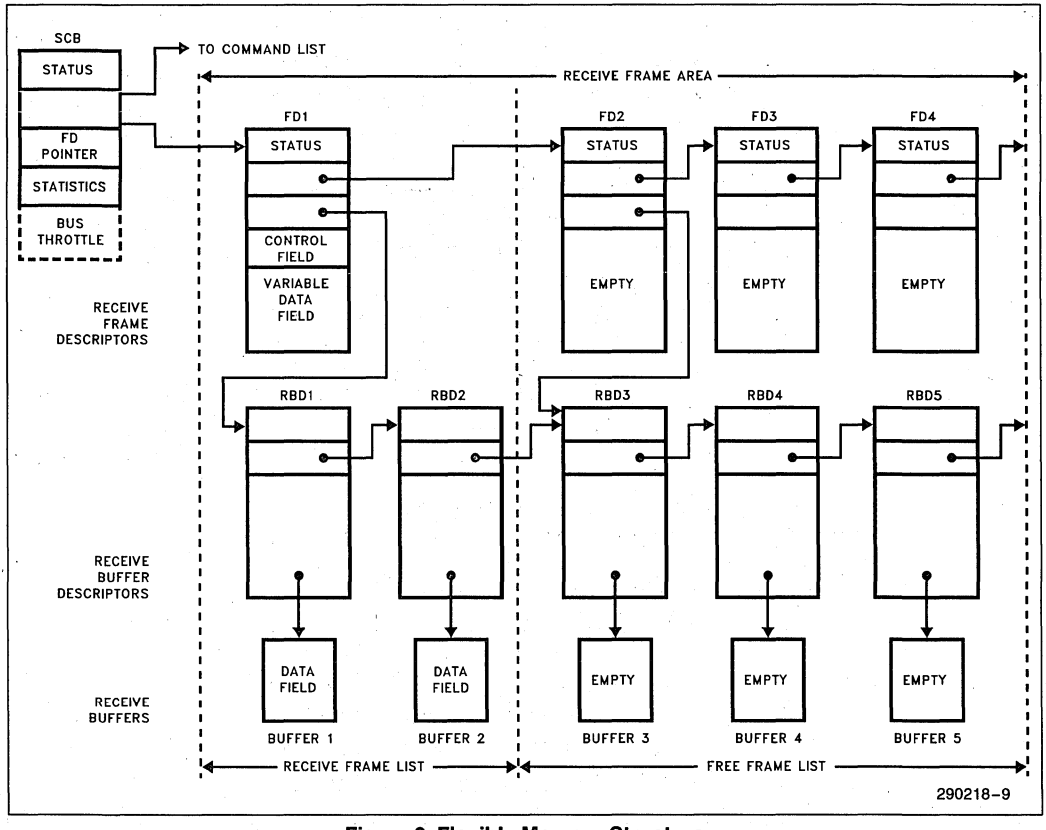


Figure 9. Flexible Memory Structure

TRANSMITTING FRAMES

The 82596 executes high-level Action Commands from the Command List in system memory. Action Commands are fetched and executed in parallel with the host CPU operation, thereby significantly improving system performance. The format of the Action Commands is shown in Figure 10. Figure 28 shows the 82586 mode, and Figures 29 and 30 show the command formats of the Linear and 32-bit Segmented modes.

A single Transmit command contains, as part of the command-specific parameters, the destination address and length field of the transmitted frame and a pointer to buffer area in memory containing the data portion of the frame. The data field is contained in a memory data structure consisting of a buffer descriptor (BD) and a data buffer—or a linked list of buffer descriptors and buffers—as shown in Figure 11.

Multiple data buffers can be chained together using the BDs. Thus, a frame with a long data field can be transmitted using several (shorter) data buffers chained together. This chaining technique allows the system designer to develop efficient buffer management.

The 82596 automatically generates the preamble (alternating 1s and 0s) and start frame delimiter, fetches the destination address and length field from the Transmit command, inserts its unique address as the source address, fetches the data field specified by the Transmit command, and computes and appends the CRC to the end of the frame (see Figure 12). In the Linear and 32-bit Segmented mode the CRC can be optionally inserted on a frame-by-frame basis by setting the NC bit in the Transmit Command Block (see Figures 29 and 30).

The 82596 can be configured to generate two types of start and end frame delimiters—End of Carrier (EOC) or HDLC. In EOC mode the start frame delimiter is 10101011 and the end frame delimiter is indicated by the lack of a signal after the last bit of the frame check sequence field has been transmitted. In EOC mode the 82596 can be configured to extend short frames by adding pad bytes (7Eh) during transmission, according to the length field. In HDLC mode the 82596 will generate the 01111110 flag for the start and end frame delimiters, and do standard bit stuffing and stripping. Furthermore, the 82596 can be configured to pad frames shorter than the specified minimum frame length by appending the appropriate number of flags to the end of the frame.

When a collision occurs, the 82596 manages the jam, random wait, and retry processes, reinitializing DMA pointers without CPU intervention. Multiple frames can be sent by linking the appropriate number of Transmit commands together. This is particularly useful when transmitting a message larger than the maximum frame size (1518 bytes for Ethernet).

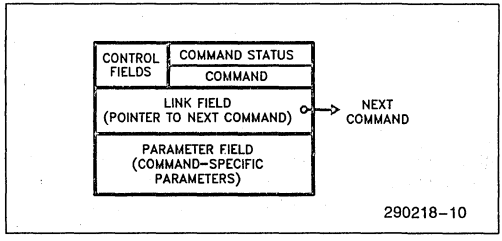


Figure 10. Action Command Format

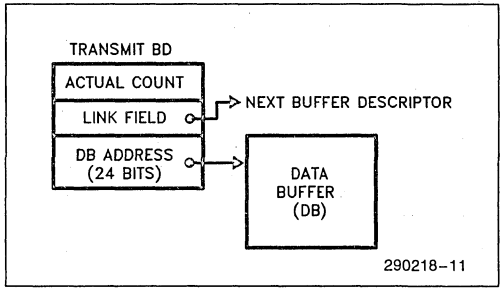


Figure 11. Data Buffer Descriptor and Data Buffer Structure

PREAMBLE	START FRAME DELIMITER	DESTINATION ADDRESS	SOURCE ADDRESS	LENGTH FIELD	DATA FIELD	FRAME CHECK SEQUENCE	END FRAME DELIMITER

Figure 12. Frame Format

RECEIVING FRAMES

To reduce CPU overhead, the 82596 is designed to receive frames without CPU supervision. The host CPU first sets aside an adequate receive buffer space and then enables the 82596 Receive Unit. Once enabled, the RU watches for arriving frames and automatically stores them in the Receive Frame Area (RFA). The RFA contains Receive Frame Descriptors, Receive Buffer Descriptors, and Data Buffers (see Figure 13). The individual Receive Frame Descriptors make up a Receive Descriptor List (RDL) used by the 82596 to store the destination and source addresses, the length field, and the status of each frame received (see Figure 14).

Once enabled, the 82596 checks each passing frame for an address match. The 82596 will recognize its own unique address, one or more multicast addresses, or the broadcast address. If a match is found the 82596 stores the destination and source addresses and the length field in the next available RFD. It then begins filling the next available Data Buffer on the FBL, which is pointed to by the current RFD, with the data portion of the incoming frame. As one Data Buffer is filled, the 82596 automatically fetches the next DB on the FBL until the entire frame is received. This buffer chaining technique is particularly memory efficient because it allows the system designer to set aside buffers to fit frames much shorter than the maximum allowable frame length. If AL-LOC = 1, or if the flexible memory structure is used, the addresses and length field can be placed in the Receive Buffer.

Once the entire frame is received without error, the 82596 does the following housekeeping tasks.

- The actual count field of the last Buffer Descriptor used to hold the frame just received is updated with the number of bytes stored in the associated Data Buffer.
- The next available Receive Frame Descriptor is fetched.
- The address of the next available Buffer Descriptor is written to the next available Receive Frame Descriptor.
- A frame received interrupt status bit is posted in the SCB.
- An interrupt is sent to the CPU.

If a frame error occurs, for example a CRC error, the 82596 automatically reinitializes its DMA pointers and reclaims any data buffers containing the bad frame. The 82596 will continue to receive frames without CPU help as long as Receive Frame Descriptors and Data Buffers are available.

82596 NETWORK MANAGEMENT AND DIAGNOSTICS

The behavior of data communication networks is normally very complex because of their distributed and asynchronous nature. It is particularly difficult to pinpoint a failure when it occurs. The 82596 has extensive diagnostic and network management functions that help improve reliability and testability. The 82596 reports on the following events after each frame is transmitted.

- Transmission successful.
- Transmission unsuccessful. Lost Carrier Sense.
- Transmission unsuccessful. Lost Clear to Send.
- Transmission unsuccessful. A DMA underrun occurred because the system bus did not keep up with the transmission.
- Transmission unsuccessful. The number of collisions exceeded the maximum allowed.
- Number of Collisions. The number of collisions experienced during the frame.
- Heartbeat Indicator. This indicates the presence of a heartbeat during the last Interframe Spacing (IFS) after transmission.

When configured to Save Bad Frames the 82596 checks each incoming frame and reports the following errors.

- CRC error. Incorrect CRC in a properly aligned frame.
- Alignment error. Incorrect CRC in a misaligned frame.
- Frame too short. The frame is shorter than the value configured for minimum frame length.
- Overrun. Part of the frame was not placed in memory because the system bus did not keep up with incoming data.
- Out of buffer. Part of the frame was discarded because of insufficient memory storage space.
- Receive collision. A collision was detected during reception.
- Length error. A frame not matching the frame length parameter was detected.

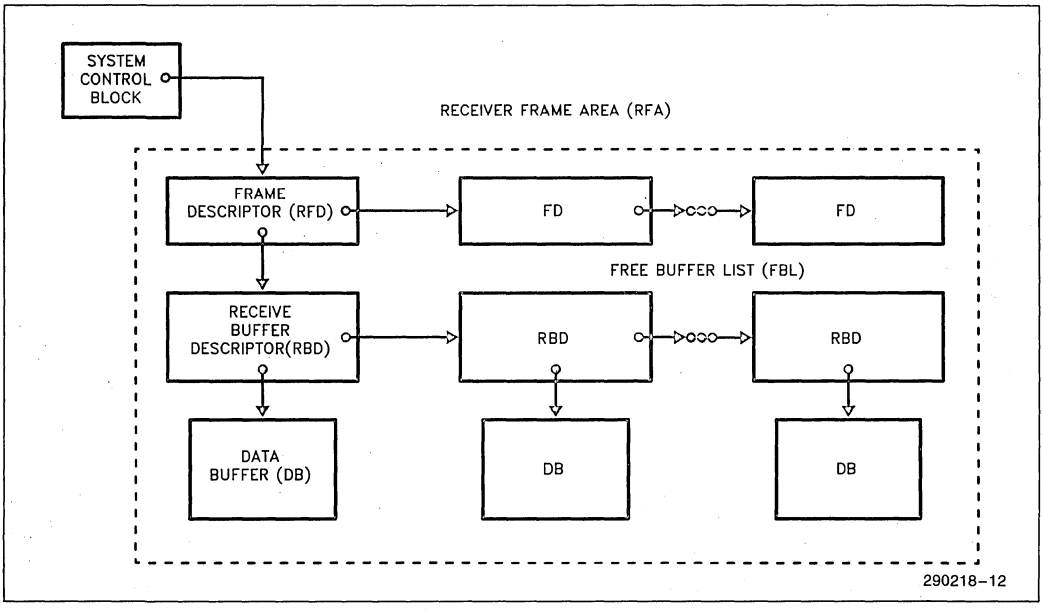


Figure 13. Receive Frame Area Diagram

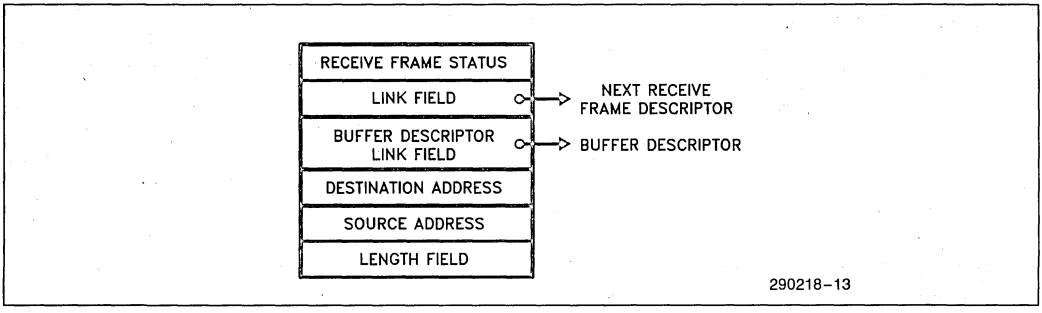


Figure 14. Receive Frame Descriptor

NETWORK PLANNING AND MAINTENANCE

To properly plan, operate, and maintain a communication network, the network management entity must accumulate information on network behavior. The 82596 provides a rich set of network-wide diagnostics that can serve as the basis for a network management entity.

Information on network activity is provided in the status of each frame transmitted. The 82596 reports the following activity indicators after each frame.

- Number of collisions. The number of collisions the 82596 experienced while attempting to transmit the frame.
- Deferred transmission. During the first transmission attempt the 82596 had to defer to traffic on the link.

The 82596 updates its 32-bit statistical counters after each received frame that both passes address filtering and is longer than the Minimum Frame Length configuration parameter. The 82596 reports the following statistics.

- CRC errors. The number of well-aligned frames that experienced a CRC error.
- Alignment errors. The number of misaligned frames that experienced a CRC error.
- No resources. The number of frames that were discarded because of insufficient resources for reception.
- Overrun errors. The number of frames that were not completely stored in memory because the system bus did not keep up with incoming data.
- Receive Collision counter. The number of collisions detected during receive.
- Short Frame counter. The number of frames that were discarded because they were shorter than the configured minimum frame length.

The 82596 can be configured to Promiscuous mode. In this mode it captures all frames transmitted on the network without checking the Destination Address. This is useful when implementing a monitoring station to capture all frames for analysis.

A useful method of capturing frame headers is to use the Simplified memory mode, configure the 82596 to Save Bad Frames, and configure the 82596 to Promiscuous mode with space in the RFD allocated for specific number of receive data bytes. The 82596 will receive all frames and put them in the RFD. Frames that exceed the available space in the RFD will be truncated, the status will be updated, and the 82596 will retrieve the next RFD. This allows the user to capture the initial data bytes of each frame (for instance, the header) and discard the remainder of the frame.

The 82596 also has a monitor mode for network analysis. During normal operation the receive function enables the 82596 to receive frames that pass address filtering. These frames must have the Start of Frame Delimiter (SFD) field and must be longer than the absolute minimum frame length of 5 bytes (6 bytes in case of Multicast address filtering). Contents and status of the received frames are transferred to memory. The monitor function enables the 82596 to simply evaluate the incoming frames. The 82596 can monitor the frames that pass or do not pass the address filtering. It can also monitor frames which do not have the SFD fields. The 82596 can be configured to only keep statistical information about monitor frames. Three options are available in the Monitor mode. These options are selected by the two monitor mode configuration bits available in the configuration command.

When the first option is selected, the 82596 receives good frames that pass address filtering and transfers them to memory while monitoring frames that do not pass address filtering or are shorter than the minimum frame size (these frames are not transferred to memory). When this option is used the 82596 updates six counters: CRC errors, alignment errors, no resource errors, overrun errors, short frames and total good frames received.

When the second option is selected, the receive function is completely disabled. The 82596 monitors only those frames that pass address filterings and meet the minimum frame length requirement. When this option is used the 82596 updates six counters: CRC errors, alignment errors, total frames (good and bad), short frames, collisions detected and total good frames.

When the third option is selected, the receive function is completely disabled. The 82596 monitors all frames, including frames that do not have a Start Frame Delimiter. When this option is used the 82596 updates six counters: CRC errors, alignment errors, total frames (good and bad), short frames, collisions detected and total good frames.

STATION DIAGNOSTICS AND SELF-TEST

The 82596 provides a large set of diagnostic and network management functions. These include internal and external loopback and time domain reflectometry for locating fault points in the network cable. The 82596 ensures software reliability by dumping the contents of the 82596 internal registers into system memory. The 82596 has a self-test mode that enables it to run an internal self-test and place the results in system memory.

82586 SOFTWARE COMPATIBILITY

The 82596 has a software-compatible state in which all its memory structures are compatible with the 82586 memory structure. This includes all the Action Commands, the Receive Frame Area (including the RFD, Buffer Descriptors, and Data Buffers), the System Control Block, and the initialization procedures. There are two minor differences between the 82596 in the 82586-Compatible memory structure and the 82586.

- When the internal and external loopback bits in the Configure command are set to 11 the 82596 is in external loopback and the $\overline{LPBK}$ pin is activated; in the 82586 this situation would produce internal loopback.
- During a Dump command both the 82596 and 82586 dump the same number of bytes; however, the data format is different.

INITIALIZING THE 82596

A Reset command is issued to the 82596 to prepare it for normal operation. The 82596 is initialized through two data structures that are addressed by two pointers, the System Configuration Pointer (SCP) and the Intermediate System Configuration Pointer (ISCP). The initialization procedure begins when a Channel Attention signal is asserted after RESET. The 82596 uses the address of the double word that contains the SCP as a default—00FFFFF4h. Before the CA signal is asserted this default address can be changed to any other available address by asserting the $\overline{PORT}$ pin and providing the desired address over the $D_{31}-D_4$ pins of the address bus. Pins D_3-D_0 must be 0010; i.e., any alternative address must be aligned to 16-byte boundaries. All addresses sent to the 82596 must be word aligned, which means that all pointers and memory structures must start on an even address (A_0 = zero).

SYSTEM CONFIGURATION POINTER (SCP)

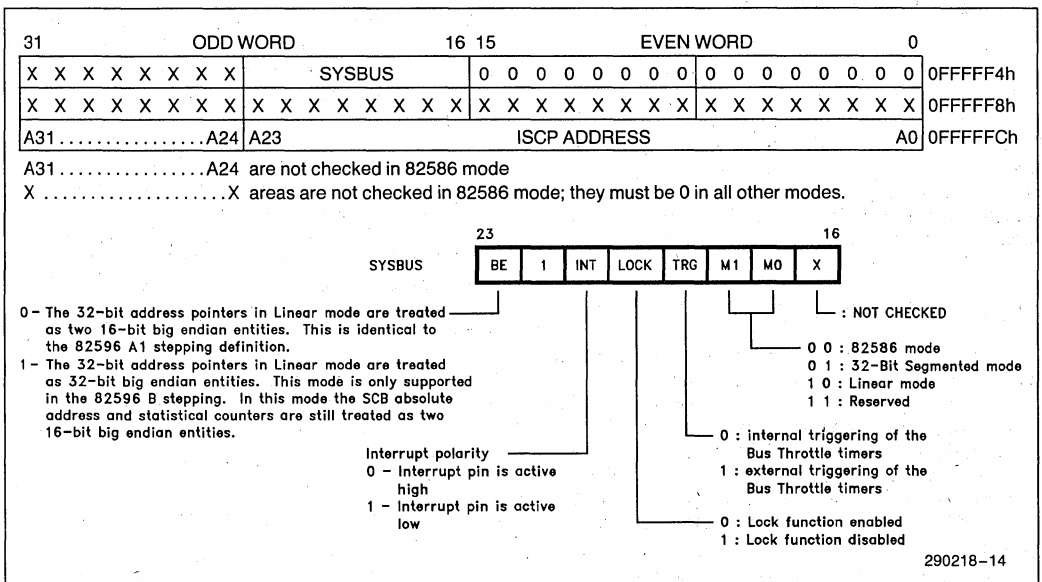
The SCP contains the sysbus byte and the location of the next structure of the initialization process, the ISCP. The following parameters are selected in the SYSBUS.

- The 82596 operation mode.
- The Bus Throttle timer triggering method.
- Lock enabled.
- Interrupt polarity.
- Big Endian 32-bit entity mode.

Byte ordering is determined by the LE/$\overline{BE}$ pin. LE/$\overline{BE}$ = 1 selects Little Endian byte ordering and LE/$\overline{BE}$ = 0 selects Big Endian byte ordering.

NOTE:
In the following, X indicates a bit not checked 82586 mode. This bit must be set to 0 in all other modes.

The following diagram illustrates the format of the SCP.



ISCP ADDRESS— The physical address of the ISCP. In the 82586 mode, bits A31–A24 are considered to be zero.

Figure 15. The System Configuration Pointer

Writing the Sysbus

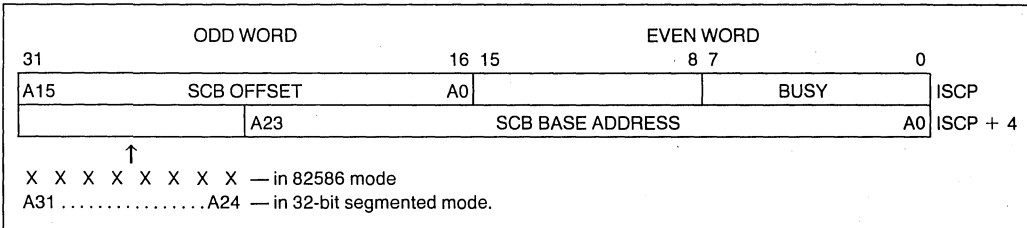
When writing the sysbus byte it is important to pay attention to the byte order.

- When a Little Endian processor is used, the sysbus byte is located at byte address 00FFFFF6h (or address $n+2$ if an alternative SCP address n was programmed).
- When a processor using Big Endian byte ordering is used, the sysbus, alternative SCP, and ISCP addresses will be different.
  - The sysbus byte is located at 00FFFFF5h.
  - If an alternative SCP address is programmed, the sysbus byte should be at byte address $n+1$.

INTERMEDIATE SYSTEM CONFIGURATION POINTER (ISCP)

The ISCP indicates the location of the System Control Block. Often the SCP is in ROM and the ISCP is in RAM. The CPU loads the SCB address (or an equivalent data structure) into the ISCP and asserts CA. This Channel Attention signal causes the 82596 to begin its initialization procedure and to get the SCB address from the ISCP and SCP. In 82586 and 32-bit Segmented modes the SCP base address is also the base address of all Command Blocks, Frame Descriptors, and Buffer Descriptors (but not buffers). All these data structures must reside in one 64-KB segment; however, in Linear mode no such limitation is imposed.

The following diagram illustrates the ISCP format.

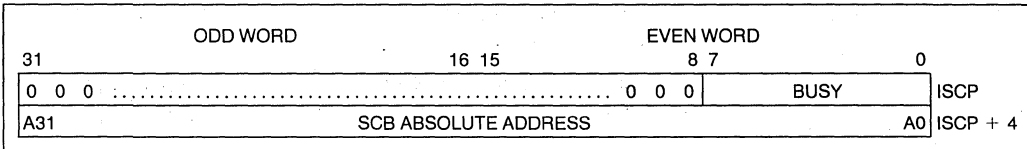


BUSY — Indicates that the 82596 is being initialized. The CPU sets the ISCP to 01h before it gives the first CA to the 82596. The ISCP is cleared by the 82596 after the SCB base and offset are read. Note that the most significant byte of the first word of the ISCP is not modified when BUSY is cleared.

SCB OFFSET— This 16-bit quantity specifies the offset portion of the address of the SCB.

SCB BASE — Specifies the base portion of the address of the SCB. The base of SCB is also the base of all 82596 Command Blocks, Frame Descriptors and Buffer Descriptors. In the 82586 mode, bits A31–A24 are considered to be zero.

Figure 16. The Intermediate System Configuration Pointer—82586 and 32-Bit Segmented Modes



BUSY — Indicates that the 82596 is being initialized. The ISCP is set to 01h by the CPU before its first CA to the 82596. It is cleared by the 82596 after the SCB address is read.

SCB ADDRESS— This 32-bit quantity specifies the physical address of the SCB.

Figure 17. The Intermediate System Configuration Pointer—Linear Mode.

INITIALIZATION PROCESS

The CPU sets up the SCP, ISCP, and the SCB structures, and, if desired, an alternative SCP address. It also sets BUSY to 01h. The 82596 is initialized when a Channel Attention signal follows a Reset signal, causing the 82596 to access the System Configuration Pointer. The sysbus byte, the operational mode, the bus throttle timer triggering method, the interrupt polarity, and the state of $\overline{\text{LOCK}}$ are read. After reset the Bus Throttle timers are essentially disabled—the T-ON value is infinite, the T-OFF value is zero. After the SCP is read, the 82596 reads the ISCP and saves the SCB address. In 82586 and 32-bit Segmented modes this address is represented as a base address plus the offset (this base address is also the base address of all the control blocks). In Linear mode the base address is also an absolute address. The 82596 clears BUSY, sets CX and CNR to equal 1 in the SCB, clears the SCB command word, sends an interrupt to the CPU, and awaits another Channel Attention signal. RESET configures the 82596 to its default state before CA is asserted.

CONTROLLING THE 82596CA

The host CPU controls the 82596 with the commands, data structures, and methods described in this section. The CPU and the 82596 communicate through shared memory structures. The 82596 contains two independent units: the Command Unit and the Receive Unit. The Command Unit executes commands from the CPU, and the Receive Unit handles frame reception. These two units are controlled and monitored by the CPU through a shared memory structure called the System Control Block (SCB). The CPU and the 82596 use the CA and INT signals to communicate with the SCB.

82596 CPU ACCESS INTERFACE ($\overline{PORT}$)

The 82596 has a CPU access interface that allows the host CPU to do four things.

- Write an alternative System Configuration Pointer address.
- Write an alternative Dump area pointer and perform Dump.
- Execute a software reset.
- Execute a self-test.

The following events initiate the CPU access state.

- Presence of an address on the $D_{31}-D_4$ data bus pins.
- The D_3-D_0 pins are used to select one of the four functions.
- The $\overline{PORT}$ input pin is asserted, as in a regular write cycle.

NOTE.
The SCP Dump and Self-Test addresses must be 16-byte aligned.

The 82596 requires two 16-bit write cycles for a port command. The first write holds the internal machines and reads the first 16 bits; the second activates the $\overline{PORT}$ command and reads the second 16 bits.

The $\overline{PORT}$ Reset is useful when only the 82596 needs to be reset. The CPU must wait for 10-system and 5-serial clocks before issuing another CA to the 82596; this new CA begins a new initialization process.

The Dump function is useful for troubleshooting No Response problems. If the chip is in a No Response state, the $\overline{PORT}$ Dump operation can be executed and a $\overline{PORT}$ Reset can be used to reinitialize the 82596 without disturbing the rest of the system.

The Self-Test function can be used for board testing; the 82596 will execute a self-test and write the results to memory.

Table 2. $\overline{PORT}$ Function Selection

	D31 ..... D4			D3	D2	D1	D0
Function	**Addresses and Results**						
Reset	A31	Don't Care	A4	0	0	0	0
Self-Test	A31	Self-Test Results Address	A4	0	0	0	1
SCP	A31	Alternative SCP Address	A4	0	0	1	0
Dump	A31	Dump Area Pointer	A4	0	0	1	1

MEMORY ADDRESSING FORMATS

The 82596 accesses memory by 32-bit addresses. There are two types of 32-bit addresses: linear and segmented. The type of address used depends on the 82596 operating mode and the type of memory structure it is addressing. The 82596 has three operating modes.

- 82586 Mode
  - A Linear address is a single 24-bit entity. Address pins $A_{31}-A_{24}$ are always zero.
  - A Segmented address uses a 24-bit base and a 16-bit offset.
- 32-bit Segmented Mode
  - A Linear address is a single 32-bit entity.
  - A Segmented address uses a 32-bit base and a 16-bit offset.

NOTE:
In the previous two memory addressing modes, each command header (CB, TBD, RFD, RBD, and SCB) must wholly reside within one segment. If the 82596 encounters a memory structure that does not follow this restriction, the 82596 will fetch the next contiguous location in memory (beyond the segment).

- Linear Mode
  - A Linear address is a single 32-bit entity.
  - There are no Segmented addresses.

Linear addresses are primarily used to address transmit and receive data buffers. In the 82586 and 32-bit Segmented modes, segmented addresses (base plus offset) are used for all Command Blocks, Buffer Descriptors, Frame Descriptors, and System Control Blocks. When using Segmented addresses, only the offset portion of the entity being addressed is specified in the block. The base for all offsets is the same—that of the SCB. See Table 1.

LITTLE ENDIAN AND BIG ENDIAN BYTE ORDERING

The 82596 supports both Little Endian and Big Endian byte ordering for its memory structures.

The 82596 A1 stepping supports Big Endian byte ordering for word and byte entities. Dword entities are not supported with 82596 A1 Big Endian byte ordering. This results in slightly different 82596A1 memory structures for Big Endian operation. These structures are defined in the *32 LAN Components Users Manual.*

The 82596 B stepping supports Big Endian byte ordering for Linear mode only. All 82596 B 32-bit address pointers are treated as 32-bit Big Endian entities, however, the SCB absolute address and statistical counters are treated as two 16-bit Big Endian entities. This 32-bit Big Endian entity support is configured through bit 7 in the SYSBUS byte.

NOTE:
All 82596 memory entities must be word or dword aligned, except the transmit buffers can be byte aligned for the 82596 B-Stepping.

An example of a dword entity is a frame descriptor command/status dword, whereas the raw data of the frame are byte entities. Both 32- and 16-bit buses are supported. When a 16-bit bus is used with Big Endian memory organization, data lines $D_{15}-D_0$ are used. The 82596 has an internal crossover that handles these swap operations.

COMMAND UNIT (CU)

The Command Unit is the logical unit that executes Action Commands from a list of commands very similar to a CPU program. A Command Block is associated with each Action Command. The CU is modeled as a logical machine that takes, at any given time, one of the following states.

- **Idle.** The CU is not executing a command and is not associated with a CB on the list. This is the initial state.
- **Suspended.** The CU is not executing a command; however, it is associated with a CB on the list.
- **Active.** The CU is executing an Action Command and pointing to its CB.

The CPU can affect CU operation in two ways: by issuing a CU Control Command or by setting bits in the Command word of the Action Command.

RECEIVE UNIT (RU)

The Receive Unit is the logical unit that receives frames and stores them in memory. The RU is modeled as a logical machine that takes, at any given time, one of the following states.

- **Idle.** The RU has no memory resources and is discarding incoming frames. This is the initial state.
- **No Resources.** The RU has no memory resources and is discarding incoming frames. This state differs from Idle in that the RU accumulates statistics on the number of discarded frames.
- **Suspended.** The RU has memory available for storing frames, but is discarding them. The suspend state can only be reached if the CPU forces this through the SCB or sets the suspend bit in the RFD.
- **Ready.** The RU has memory available and is storing incoming frames.

The CPU can affect RU operation in three ways: by issuing an RU Control Command, by setting bits in the Frame Descriptor Command word of the frame being received, or by setting the EL bit of the current buffer's Buffer Descriptor.

SYSTEM CONTROL BLOCK (SCB)

The SCB is a memory block that plays a major role in communications between the CPU and the 82596. Such communications include the following.

- Commands issued by the CPU
- Status reported by the 82596

Control commands are sent to the 82596 by writing them into the SCB and then asserting CA. The 82596 examines the command, performs the required action, and then clears the SCB command word. Control commands perform the following types of tasks.

- Operation of the Command Unit (CU). The SCB controls the CU by specifying the address of the Command Block List (CBL) and by starting, suspending, resuming, or aborting execution of CBL commands.
- Operation of the Bus Throttle. The SCB controls the Bus Throttle timers by providing them with new values and sending the Load and Start timer commands. The timers can be operated in both the 32-bit Segmented and Linear modes.
- Reception of frames by the Receive Unit (RU). The SCB controls the RU by specifying the address of the Receive Frame Area and by starting, suspending, resuming, or aborting frame reception.
- Acknowledgment of events that cause interrupts.
- Resetting the chip.

The 82596 sends status reports to the CPU via the System Control Block. The SCB contains four types of status reports.

- The cause of the current interrupts. These interrupts are caused by one or more of the following 82596 events.
  - The Command Unit completes an Action Command that has its I bit set.
  - The Receive Unit receives a frame.
  - The Command Unit becomes inactive.
  - The Receive Unit becomes not ready.
- The status of the Command Unit.
- The status of the Receive Unit.
- Status reports from the 82596 regarding reception of corrupted frames.

Events can be cleared only by CPU acknowledgment. If some events are not acknowledged by the ACK field the Interrupt signal (INT) will be reissued after Channel Attention (CA) is processed. Furthermore, if a new event occurs while an interrupt is set, the interrupt is temporarily cleared to trigger edge-triggered interrupt controllers.

The CPU uses the Channel Attention line to cause the 82596 to examine the SCB. This signal is trailing-edge triggered—the 82596 latches CA on the trailing edge. The latch is cleared by the 82596 before the SCB control command is read.

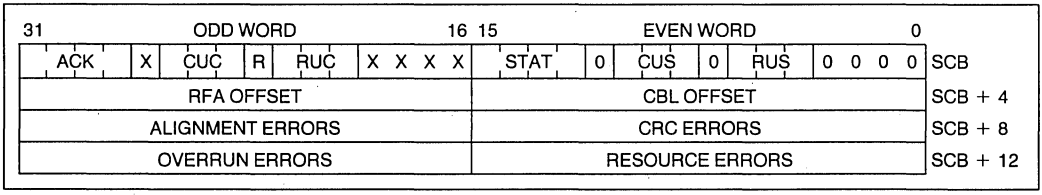
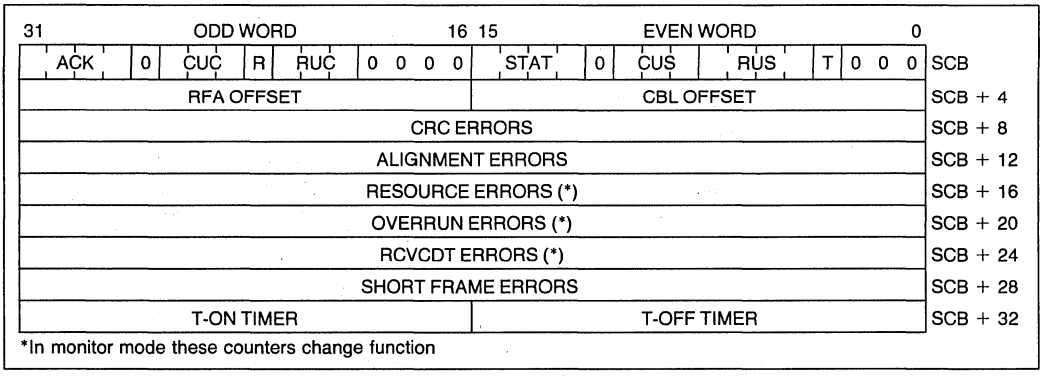
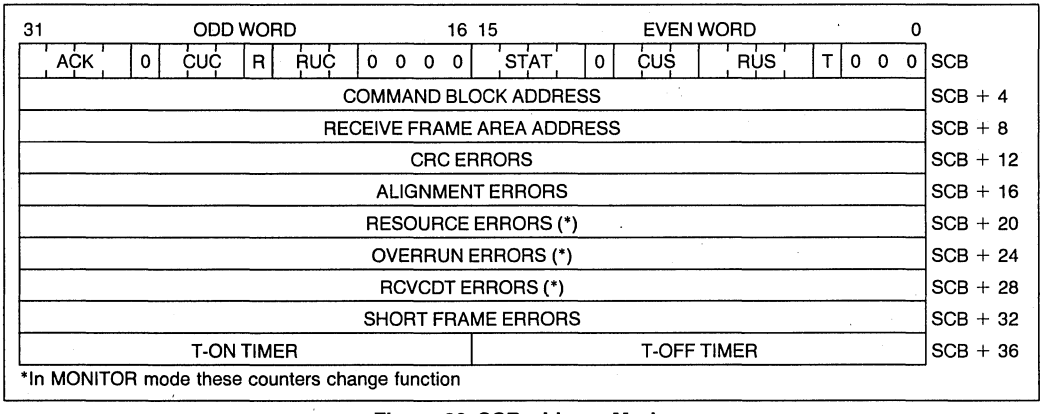


Figure 18. SCB—82586 Mode



*In monitor mode these counters change function

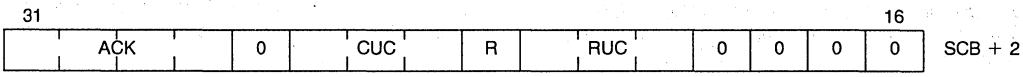
Figure 19. SCB—32-Bit Segmented Mode



*In MONITOR mode these counters change function

Figure 20. SCB—Linear Mode

Command Word



These bits specify the action to be performed as a result of a CA. This word is set by the CPU and cleared by the 82596. Defined bits are:

Bit 31 ACK-CX — Acknowledges that the CU completed an Action Command.

Bit 30 ACK-FR — Acknowledges that the RU received a frame.

Bit 29 ACK-CNA — Acknowledges that the Command Unit became not active.

Bit 28 ACK-RNR — Acknowledges that the Receive Unit became not ready.

Bits 24–26 CUC — (3 bits) This field contains the command to the Command Unit. Valid values are:

0 — NOP (does not affect current state of the unit).

1 — Start execution of the first command on the CBL. If a command is executing, complete it before starting the new CBL. The beginning of the CBL is in CBL OFFSET (address).

2 — Resume the operation of the Command Unit by executing the next command. This operation assumes that the Command Unit has been previously suspended.

3 — Suspend execution of commands on CBL after current command is complete.

4 — Abort current command immediately.

5 — Loads the Bus Throttle timers so they will be initialized with their new values after the active timer (T-ON or T-OFF) reaches Terminal Count. If no timer is active new values will be loaded immediately. This command is not valid in 82586 mode.

6 — Loads and immediately restarts the Bus Throttle timers with their new values. This command is not valid in 82586 mode.

7 — Reserved.

Bit 23 RESET — Reset chip (logically the same as hardware RESET).

Bits 20–22 RUC — (3 bits) This field contains the command to the Receive Unit. Valid values are:

0 — NOP (does not alter current state of unit).

1 — Start reception of frames. The beginning of the RFA is contained in the RFA OFFSET (address). If a frame is being received complete reception before starting.

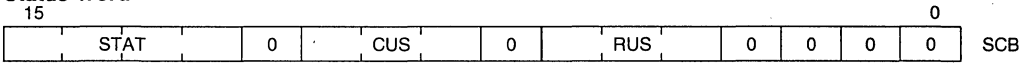
2 — Resume frame reception (only when in suspended state).

3 — Suspend frame reception. If a frame is being received complete its reception before suspending.

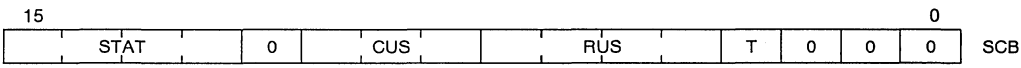
4 — Abort receiver operation immediately.

5–7 — Reserved.

Status Word



82586 mode



32-Bit Segmented and Linear mode.

Indicates the status of the 82596. This word is modified only by the 82596. Defined bits are:

Bit 15 CX — The CU finished executing a command with its I (interrupt) bit set.

Bit 14 FR — The RU finished receiving a frame.

Bit 13 CNA — The Command Unit left the Active state.

Bit 12 RNR — The Receive Unit left the Ready state.

Bits 8–10 CUS — (3 bits) This field contains the status of the command unit. Valid values are:

0 — Idle

1 — Suspended

2 — Active

3–7 — Not used

Bits 4–7 RUS — This field contains the status of the receive unit. Valid values are:

0h (0000) — Idle

1h (0001) — Suspended

2h (0010) — No Resources. This bit indicates both no resources due to lack of RFDs in the RDL and no resources due to lack of RBDs in the FBL.

4h (0100) — Ready

Ah (1010) — No resources due to no more RBDs (not in the 82586 mode).

Ch (1100) — No more RBDs (not in 82586 mode)

No other combinations are allowed

Bit 3 T — Bus Throttle timers loaded (not in 82586 mode).

4

SCB OFFSET ADDRESSES

CBL Offset (Address)

In 82586 and 32-bit Segmented modes this 16-bit quantity indicates the offset portion of the address for the first Command Block on the CBL. In Linear mode it is a 32-bit linear address for the first Command Block on the CBL. It is accessed only if CUC equals Start.

RFA Offset (Address)

In 82586 and 32-bit Segmented modes this 16-bit quantity indicates the offset portion of the address for the Receive Frame Area. In Linear mode it is a 32-bit linear address for the Receive Frame Area. It is accessed only if RUC equals Start.

SCB STATISTICAL COUNTERS

Statistical Counter Operation

- The CPU is responsible for clearing all error counters before initializing the 82596. The 82596 updates these counters by reading them, adding 1, and then writing them back to the SCB.
- The counters are wraparound counters. After reaching FFFFFFFFh the counters wrap around to zero.
- The 82596 updates the required counters for each frame. It is possible for more than one counter to be updated; multiple errors will result in all affected counters being updated.
- The 82596 executes the read-counter/increment/write-counter operation without relinquishing the bus (locked operation). This is to ensure that no logical contention exists between the 82596 and the CPU due to both attempting to write to the counters simultaneously. In the dual-port memory configuration the CPU should not execute any write operation to a counter if $\overline{LOCK}$ is asserted.
- The counters are 32-bits wide and their behavior is fully compatible with the IEEE 802.3 standard. The 82596 supports all relevant statistics (mandatory, optional, and desired) through the status of the transmit and receive header and directly through SCB statistics.

CRCERRS

This 32-bit quantity contains the number of aligned frames discarded because of a CRC error. This counter is updated, if needed, regardless of the RU state.

ALNERRS

This 32-bit quantity contains the number of frames that both are misaligned (i.e., where $\overline{CRS}$ deasserts on a nonoctet boundary) and contain a CRC error. The counter is updated, if needed, regardless of the RU state.

SHRTFRM

This 32-bit quantity contains the number of received frames shorter than the minimum frame length.

The last three counters change function in monitor mode.

RSCERRS

This 32-bit quantity contains the number of good frames discarded because there were no resources to contain them. Frames intended for a host whose RU is in the No Receive Resources state, fall into this category. This counter is updated only if the RU is in the No Resources state. When in Monitor mode this counter counts the total number of frames—good and bad.

OVRNERRS

This 32-bit quantity contains the number of frames known to be lost because the local system bus was not available. If the traffic problem lasts longer than the duration of one frame, the frames that follow the first are lost without an indicator, and they are not counted. This counter is updated, if needed, regardless of the RU state.

RCVCDT

This 32-bit quantity contains the number of collisions detected during frame reception. In Monitor mode this counter counts the total number of good frames.

ACTION COMMANDS AND OPERATING MODES

This section lists all the Action Commands of the Command Unit Command Block List (CBL). Each command contains the Command field, the Status and Control fields, the link to the next Action Command, and any command-specific parameters. There are three basic types of action commands: 82596 Configuration and Setup, Transmission, and Diagnostics. The following is a list of the actual commands.

- NOP
- Individual Address Setup
- Configure
- MC Setup
- Transmit
- TDR
- Dump
- Diagnose

The 82596 has three addressing modes. In the 82586 mode all the Action Commands look exactly like those of the 82586.

- **82586 Mode.** The 82596 software and memory structure is compatible with the 82586.
- **32-Bit Segmented Mode.** The 82596 can access the entire system memory and use the two new memory structures—Simplified and Flexible—while still using the segmented approach. This does not require any significant changes to existing software.
- **Linear Mode.** The 82596 operates in a flat, linear, 4 gigabyte memory space without segmentation. It can also use the two new memory structures.

In the 32-bit Segmented mode there are some differences between the 82596 and 82586 action commands, mainly in programming and activating new 82596 features. Those bits marked "don't care" in the compatible mode are not checked; however, we strongly recommend that those bits all be zeroes; this will allow future enchancements and extensions.

In the Linear mode all of the address offsets become 32-bit address pointers. All new 82596 features are accessible in this mode, and all bits previously marked "don't care" must be zeroes.

The Action Commands, and all other 82596 memory structures, must begin on even byte boundaries, i.e., they must be word aligned.

NOP

This command results in no action by the 82596 except for those performed in the normal command processing. It is used to manipulate the CBL manipulation. The format of the NOP command is shown in Figure 21.

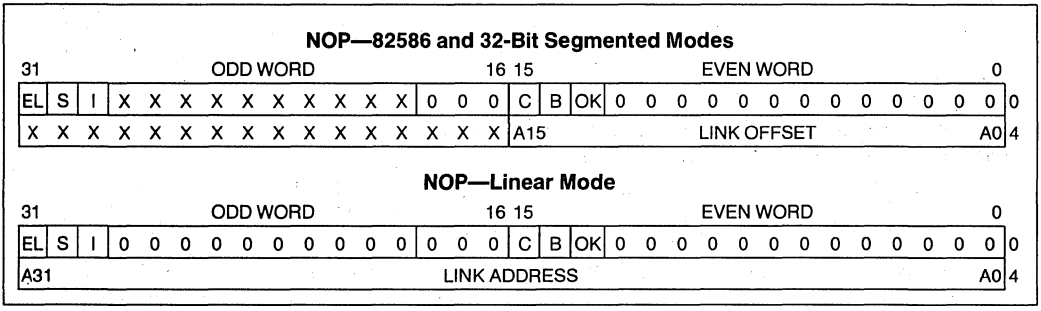


Figure 21

where:

LINK POINTER — In the 82586 or 32-bit Segmented modes this is a 16-bit offset to the next Command Block. In the Linear mode this is the 32-bit address of the next Command Block.

EL — If set, this bit indicates that this command block is the last on the CBL.

S — If set to one, suspend the CU upon completion of this CB.

I — If set to one, the 82596 will generate an interrupt after execution of the command is complete. If I is not set to one, the CX bit will not be set.

CMD (bits 16–18) — The NOP command. Value: 0h.

Bits 19–28 — Reserved (zero in the 32-bit Segmented and Linear modes).

C — This bit indicates the execution status of the command. The CPU initially resets it to zero when the Command Block is placed on the CBL. Following a command Completion, the 82596 will set it to one.

B — This bit indicates that the 82596 is currently executing the NOP command. It is initially reset to zero by the CPU. The 82596 sets it to one when execution begins and to zero when execution is completed. This bit is also set when the 82596 prefetches the command.

NOTE:

The C and B bits are modified in one operation.

OK — Indicates that the command was executed without error. If set to one no error occurred (command executed OK). If zero an error occured.

Individual Address Setup

This command is used to load the 82596 with the Individual Address. This address is used by the 82596 for inserting the Source Address during transmission and recognizing the Destination Address during reception. After RESET, and prior to Individual Address Setup Command execution, the 82596 assumes the Broadcast Address is the Individual Address in all aspects, i.e.:

- This will be the Individual Address Match reference.
- This will be the Source Address of a transmitted frame (for AL-LOC=0 mode only).

The format of the Individual Address Setup command is shown in Figure 22.

IA Setup—82586 and 32-Bit Segmented Modes

31 ODD WORD 16	15 EVEN WORD 0	
EL S I X X X X X X X X X X 0 0 1	C B OK A 0 0 0 0 0 0 0 0 0 0 0 0	0
INDIVIDUAL ADDRESS 1st byte	A15 LINK OFFSET A0	4
6th byte 5th byte	4th byte 3rd byte	8

IA Setup—Linear Mode

31 ODD WORD 16	15 EVEN WORD 0	
EL S I 0 0 0 0 0 0 0 0 0 0 0 0 1	C B OK A 0 0 0 0 0 0 0 0 0 0 0 0	0
A31 LINK ADDRESS A0		4
4th byte 3rd byte	INDIVIDUAL ADDRESS 1st byte	8
	6th byte 5th byte	C

Figure 22

where:

LINK ADDRESS, EL, B, C, I, S — As per standard Command Block (see the NOP command for details)

A — Indicates that the command was abnormally terminated due to CU Abort control command. If one, then the command was aborted, and if necessary it should be repeated. If this bit is zero, the command was not aborted.

Bits 19–28 — Reserved (zero in the 32-bit Segmented and Linear modes).

CMD (bits 16–18) — The Address Setup command. Value: 1h.

INDIVIDUAL ADDRESS — The individual address of the node, 0 to 6 bytes long.

The least significant bit of the Individual Address must be zero for Ethernet (see the Command Structure). However, no enforcement of 0 is provided by the 82596. Thus, an Individual Address with 1 as its least significant bit is a valid Individual Address in all aspects.

The default address length is 6 bytes long, as in 802.3. If a different length is used the IA Setup command should be executed after the Configure command.

Configure

The Configure command loads the 82596 with its operating parameters. It allows changing some of the parameters by specifying a byte count less than the maximum number of configuration bytes (11 in the 82586 mode, 14 in the 32-Bit Segmented and Linear modes). The 82596 configuration depends on its mode of operation. When configuring the 12th byte (Byte 11 undefined) in 82586 mode this byte should be all ones.

- In the 82586 mode the maximum number of configuration bytes is 12. Any number larger than 12 will be reduced to 12 and any number less than 4 will be increased to 4.
- The additional features of the serial side are disabled in the 82586 mode.
- In both the 32-Bit Segmented and Linear modes there are four additional configuration bytes, which hold parameters for additional 82596 features. If these parameters are not accessed, the 82596 will follow their default values.
- For more detailed information refer to the 32-Bit LAN Components User's Manual.

The format of the Configure command is shown in Figure 23, 24 and 25.

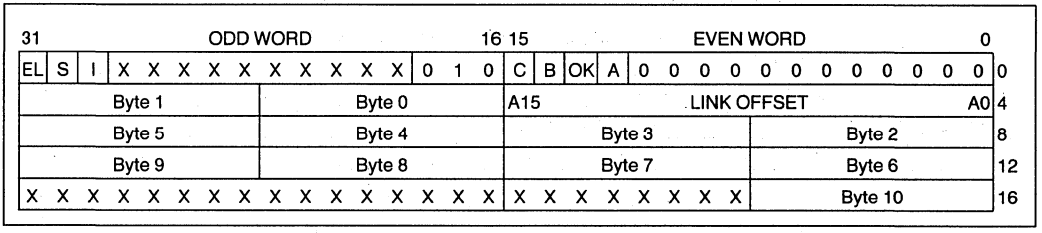


Figure 23. CONFIGURE—82586 Mode

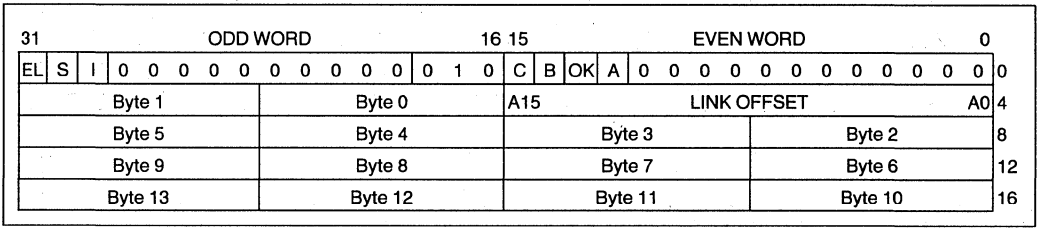


Figure 24. CONFIGURE—32-Bit Segmented Mode

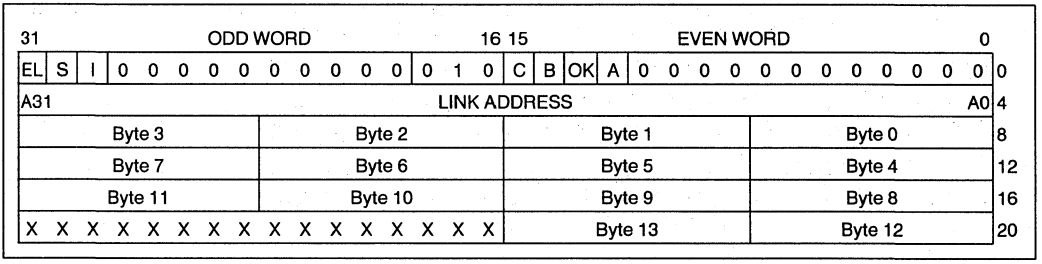


Figure 25. CONFIGURE—Linear Mode

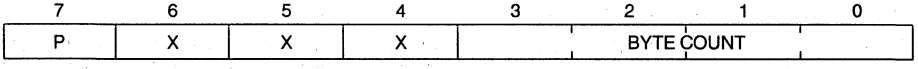
LINK ADDRESS, EL, B, C, I, S — As per standard Command Block (see the NOP command for details)

A — Indicates that the command was abnormally terminated due to a CU Abort control command. If 1, then the command was aborted and if necessary it should be repeated. If this bit is 0, the command was not aborted.

Bits 19–28 — Reserved (zero in the 32-Bit Segmented and Linear Modes)

CMD (bits 16–18) — The CONFIGURE command. Value: 2h.

The interpretation of the fields follows:



BYTE 0

BYTE CNT (Bits 0–3) — Byte Count. Number of bytes, including this one, that hold parameters to be configured.

PREFETCHED (Bit 7) — Enable the 82596 to write the prefetched bit in all prefetch RBDs.

NOTE:
The P bit is valid only in the new memory structure modes. In 82586 mode this bit is disabled (i.e., no prefetched mark).

7							0
MONITOR	X	X	FIFO LIMIT				

BYTE 1

FIFO Limit (Bits 0–3) — FIFO limit.

MONITOR# (Bits 6–7) — Receive monitor options. If the Byte Count of the configure command is less than 12 bytes then these Monitor bits are ignored.

DEFAULT: C8h

7							0
SAV BF	1	0	0	0	0	RESUME_RD	0

BYTE 2

SAV BF (Bit 7) — 0—Received bad frames are not saved in the memory.
1—Received bad frames are saved in the memory.

DEFAULT: 40h

RESUME_RD (Bit 1) — 0 — The 82596 does not reread the next CB on the list when a CU Resume Control Command is issued.
1 — The 82596 will reread the next CB on the list when a CU Resume Control Command is issued. This is available only on the 82596B stepping.

7				0
LOOP BACK MODE	PREAMBLE LENGTH	NO SRC ADD INS	ADDRESS LENGTH	

BYTE 3

ADR LEN (Bits 0–2) — Address length (any kind).

NO SCR ADD INS (Bit 3) — No Source Address Insertion.
In the 82586 this bit is called AL LOC.

PREAM LEN (Bits 4–5) — Preamble length.

LP BCK MODE (Bits 6–7) — Loopback mode.

DEFAULT: 26h

7				0
BOF METD	EXPONENTIAL PRIORITY	0	LINEAR PRIORITY	

BYTE 4

LIN PRIO (Bits 0–2) — Linear Priority.

EXP PRIO (Bits 4–6) — Exponential Priority.

BOF METD (Bit 7) — Exponential Backoff method.

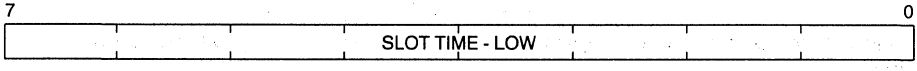
DEFAULT: 00h

7							0
INTER FRAME SPACING							

BYTE 5

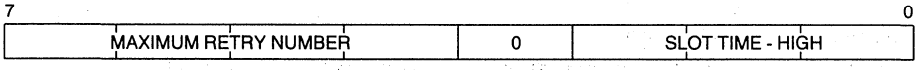
INTERFRAME SPACING — Interframe spacing.

DEFAULT: 60h



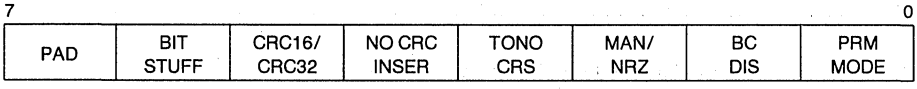
BYTE 6

SLOT TIME (L)	Slot time, low byte.
DEFAULT: 00h	



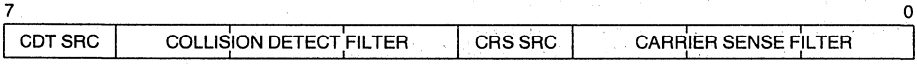
BYTE 7

SLOT TIME (H) (Bits 0–2)	Slot time, high part.
RETRY NUM (Bits 4–7)	Number of transmission retries on collision.
DEFAULT: F2h	



BYTE 8

PRM (Bit 0)	Promiscuous mode.
BC DIS (Bit 1)	Broadcast disable.
MANCH/NRZ (Bit 2)	Manchester or NRZ encoding. See specific timing requirements for TXC in Manchester mode.
TONO CRS (Bit 3)	Transmit on no CRS.
NOCRC INS (Bit 4)	No CRC insertion.
CRC-16/CRC-32 (Bit 5)	CRC type.
BIT STF (Bit 6)	Bit stuffing.
PAD (Bit 7)	Padding.
DEFAULT: 00h	



BYTE 9

CRSF (Bits 0–2)	Carrier Sense filter (length).
CRS SRC (Bit 3)	Carrier Sense source.
CDTF (Bits 4–6)	Collision Detect filter (length).
CDT SRC (Bit 7)	Collision Detect source.
DEFAULT: 00h	

7							0
MINIMUM FRAME LENGTH							

BYTE 10

MIN FRAME LEN — Minimum frame length.

DEFAULT: 40h

7							0
MONITOR		MC_ALL	CDBSAC	AUTOTX	CRCINM	LNGFLD	PRECRS

BYTE 11

PRECRS (Bit 0) — Preamble until Carrier Sense

LNGFLD (Bit 1) — Length field. Enables padding at the End-of-Carrier framing (802.3).

CRCINM (Bit 2) — Rx CRC appended to the frame in memory.

AUTOTX (Bit 3) — Auto retransmit when a collision occurs during the preamble.

CDBSAC (Bit 4) — Collision Detect by source address recognition.

MC_ALL (Bit 5) — Enable to receive all MC frames.

MONITOR (Bits 6–7) — Receive monitor options.

DEFAULT: FFH

7							0
0	FDX	0	0	0	0	0	0

BYTE 12

FDX (Bit 6) — Enables Full Duplex operation.

DEFAULT: 00h

7							0
DIS_BOF	MULT_IA	1	1	1	1	1	1

BYTE 13

MULT_IA (Bit 6) — Multiple individual address.

DIS_BOF (Bit 7) — Disable the backoff algorithm.

DEFAULT: 3Fh

A reset (hardware or software) configures the 82596 according to the following defaults.

Table 4. Configuration Defaults

	Parameter	Default Value	Units/Meaning
	ADDRESS LENGTH	**6	Bytes
	A/L FIELD LOCATION	0	Located in FD
*	AUTO RETRANSMIT	1	Auto Retransmit Enable
	BITSTUFFING/EOC	0	EOC
	BROADCAST DISABLE	0	Broadcast Reception Enabled
*	CDBSAC	1	Disabled
	CDT FILTER	0	Bit Times
	CDT SRC	0	External Collision Detection
*	CRC IN MEMORY	1	CRC Not Transferred to Memory
	CRC-16/CRC-32	**0	CRC-32
	CRS FILTER	0	0 Bit Times
	CRS SRC	0	External CRS
*	DISBOF	0	Backoff Enabled
	EXT LOOPBACK	0	Disabled
	EXPONENTIAL PRIORITY	**0	802.3 Algorithm
	EXPONENTIAL BACKOFF METHOD	**0	802.3 Algorithm
*	FULL DUPLEX (FDX)	0	CSMA/CD Protocol (No FDX)
	FIFO THRESHOLD	8	TX: 32 Bytes, RX: 64 Bytes
	INT LOOPBACK	0	Disabled
	INTERFRAME SPACING	**96	Bit Times
	LINEAR PRIORITY	**0	802.3 Algorithm
*	LENGTH FIELD	1	Padding Disabled
	MIN FRAME LENGTH	**64	Bytes
*	MC ALL	1	Disabled
*	MONITOR	11	Disabled
	MANCHESTER/NRZ	0	NRZ
*	MULTI IA	0	Disabled
	NUMBER OF RETRIES	**15	Maximum Number of Retries
	NO CRC INSERTION	0	CRC Appended to Frame
	PREFETCH BIT IN RBD	0	Disabled (Valid Only in New Modes)
	PREAMBLE LENGTH	**7	Bytes
*	Preamble Until CRS	1	Disabled
	PROMISCUOUS MODE	0	Address Filter On
	PADDING	0	No Padding
	SLOT TIME	**512	Bit Times
	SAVE BAD FRAME	0	Discards Bad Frames
	TRANSMIT ON NO CRS	0	Disabled

NOTES:
1. This configuration setup is compatible with the IEEE 802.3 specification.
2. The Asterisk "*" signifies a new configuration parameter not available in the 82586.
3. The default value of the Auto retransmit configuration parameter is enabled(1).
4. Double Asterisk "**" signifies IEEE 802.3 requirements.

Multicast-Setup

This command is used to load the 82596 with the Multicast-IDs that should be accepted. As noted previously, the filtering done on the Multicast-IDs is not perfect and some unwanted frames may be accepted. This command resets the current filter and reloads it with the specified Multicast-IDs. The format of the Multicast-addresses setup command is:

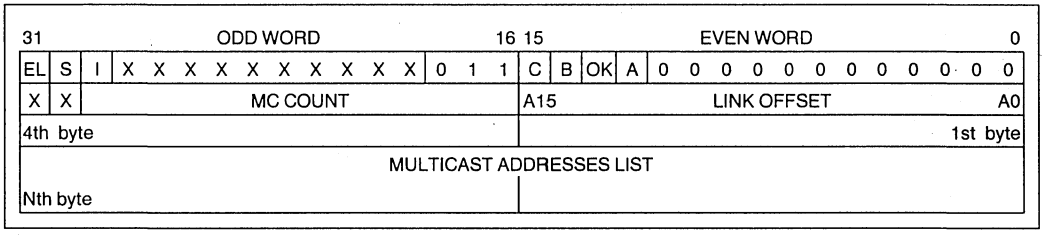


Figure 26. MC Setup—82586 and 32-Bit Segmented Modes

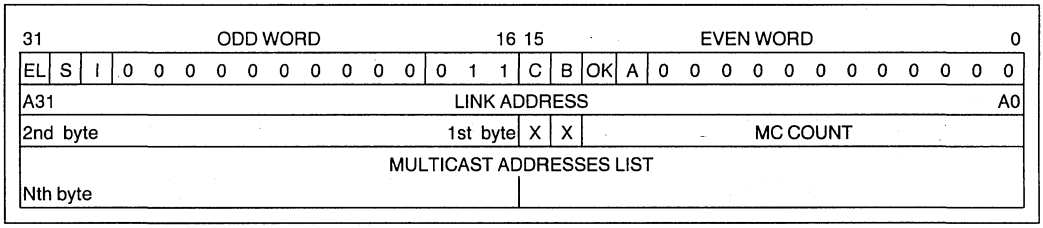


Figure 27. MC Setup—Linear Mode

where:

LINK ADDRESS, EL, B, C, I, S — As per standard Command Block (see the NOP command for details)

A — Indicates that the command was abnormally terminated due to a CU Abort control command. If one, then the command was aborted and if necessary it should be repeated. If this bit is zero, the command was not aborted.

Bits 19–28 — Reserved (0 in both the 32-Bit Segmented and Linear Modes).

CMD (bits 16–18) — The MC SETUP command value: 3h.

MC-CNT — This 14-bit field indicates the number of bytes in the MC LIST field. The MC CNT must be a multiple of the ADDR LEN; otherwise, the 82596 reduces the MC CNT to the nearest ADDR LEN multiple. MC CNT = 0 implies resetting the Hash table which is equivalent to disabling the Multicast filtering mechanism.

MC LIST — A list of Multicast Addresses to be accepted by the 82596. The least significant bit of each MC address must be 1.

NOTE:

The list is sequential; i.e., the most significant byte of an address is immediately followed by the least significant byte of the next address.

— When the 82596 is configured to recognize multiple Individual Address (Multi-IA), the MC-Setup command is also used to set up the Hash table for the individual address.

The least significant bit in the first byte of each IA address must be 0.

Transmit

This command is used to transmit a frame of user data onto the serial link. The format of a Transmit command is as follows.

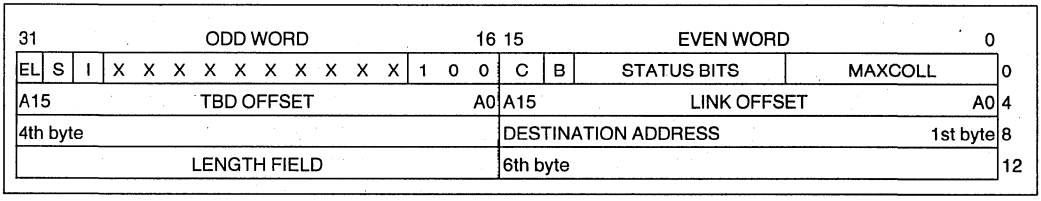


Figure 28. TRANSMIT—82586 Mode

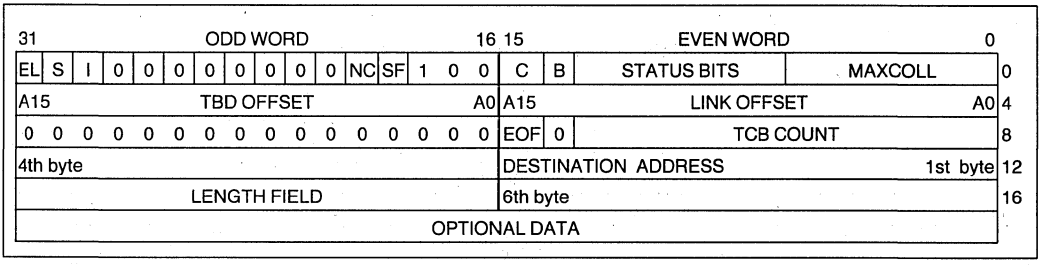


Figure 29. TRANSMIT—32-Bit Segmented Mode

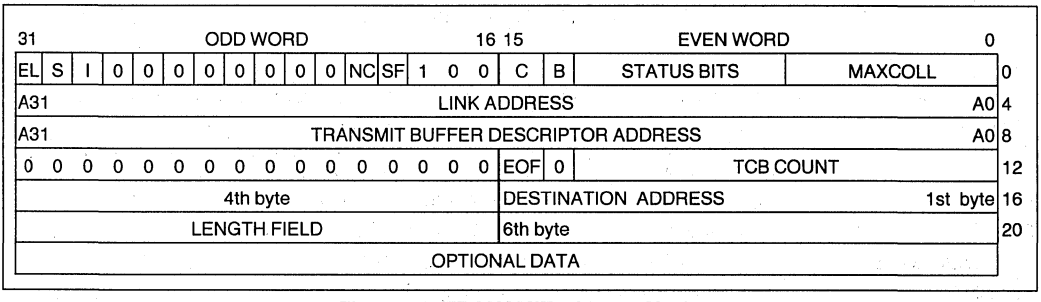
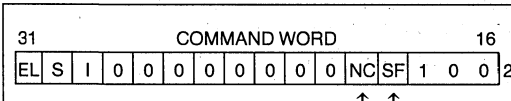


Figure 30. TRANSMIT—Linear Mode



0: No CRC Insertion disable; when the configure command is configured to not insert the CRC during transmission the NC bit has no effect.

1: No CRC Insertion enable; when the configure command is configured to insert the CRC during transmission the CRC will not be inserted when NC = 1.

0: Simplified Mode, all the Tx data is in the Transmit Command Block. The Transmit Buffer Descriptor Address field is all 1s.

1: Flexible Mode. Data is in the TCB and in a linked list of TBDs.

where:

EL, B, C, I, S — As per standard Command Block (see the NOP command for details).

OK (Bit 13) — Error free completion.

A (Bit 12) — Indicates that the command was abnormally terminated due to CU Abort control command. If 1, then the command was aborted, and if necessary it should be repeated. If this bit is 0, the command was not aborted.

Bits 19–28 — Reserved (0 in the 32-bit Segmented and Linear modes).

CMD (Bits 16–18) — The transmit command: 4h.

Status Bit 11 — Late collision. A late collision (a collision after the slot time is elapsed) is detected.

Status Bit 10 — No Carrier Sense signal during transmission. Carrier Sense signal is monitored from the end of Preamble transmission until the end of the Frame Check Sequence for TONOCRS = 1 (Transmit On No Carrier Sense mode) it indicates that transmission has been executed despite a lack of CRS. For TONOCRS = 0 (Ethernet mode), this bit also indicates unsuccessful transmission (transmission stopped when lack of Carrier Sense has been detected).

Status Bit 9 — Transmission unsuccessful (stopped) due to Loss of $\overline{CTS}$.

Status Bit 8 — Transmission unsuccessful (stopped) due to DMA Underrun; i.e., the system did not supply data for transmission.

Status Bit 7 — Transmission Deferred, i.e., transmission was not immediate due to previous link activity.

Status Bit 6 — Heartbeat Indicator, Indicates that after a previously performed transmission, and before the most recently performed transmission, (Interframe Spacing) the CDT signal was monitored as active. This indicates that the Ethernet Transceiver Collision Detect logic is performing properly. The Heartbeat is monitored during the Interframe Spacing period.

Status Bit 5 — Transmission attempt was stopped because the number of collisions exceeded the maximum allowable number of retries.

Status Bit 4 — 0 (Reserved).

MAX-COL (Bits 3–0) — The number of Collisions experienced during this frame. Max Col = 0 plus S5 = 1 indicates 16 collisions.

LINK OFFSET — As per standard Command Block (see the NOP Command for details)

TBD POINTER — In the 82586 and 32-bit Segmented modes this is the offset of the first Tx Buffer Descriptor containing the data to be transmitted. In the Linear mode this is the 32-bit address of the first Tx Buffer Descriptor on the list. If the TBD POINTER is all 1s it indicates that no TBD is used.

DEST ADDRESS — Contains the Destination Address of the frame. The least significant bit (MC) indicates the address type.

MC = 0: Individual Address.

MC = 1: Multicast or Broadcast Address.

If the Destination Address bits are all 1s this is a Broadcast Address.

LENGTH FIELD — The contents of this 2-byte field are user defined. In 802.3 it contains the length of the data field. It is placed in memory in the same order it is transmitted; i.e., most significant byte first, least significant byte second.

TCB COUNT — This 14-bit counter indicates the number of bytes that will be transmitted from the Transmit Command Block, starting from the third byte after the TCB COUNT field (address $n+12$ in the 32-bit Segmented mode, $N+16$ in the Linear mode). The TCB COUNT field can be any number of bytes (including an odd byte), this allows the user to transmit a frame with a header having an odd number of bytes. The TCB COUNT field is not used in the 82586 mode.

EOF Bit — Indicates that the whole frame is kept in the Transmit Command Block. In the Simplified memory model it must be always asserted.

The interpretation of what is transmitted depends on the No Source Address insertion configuration bit and the memory model being used.

NOTES:

1. The Destination Address and the Length Field are sequential. The Length Field immediately follows the most significant byte of the Destination Address.
2. In case the 82596 is configured with No Source Address insertion bit equal to 0, the 82596 inserts its configured Source Address in the transmitted frame.
   - In the 82586 mode, or when the Simplified memory model is used, the Destination and Length fields of the transmitted frame are taken from the Transmit Command Block.
   - If the FLEXIBLE memory model is used, the Destination and Length fields of the transmitted frame can be found either in the TCB or TBD, depending on the TCB COUNT.
3. If the 82596 is configured with the Address/Length Field Location equal to 1, the 82596 does not insert its configured Source Address in the transmitted frame. The first (2 × Address Length) + 2 bytes of the transmitted frame are interpreted as Destination Address, Source Address, and Length fields respectively. The location of the first transmitted byte depends on the operational mode of the 82596:
   - In the 82586 mode, it is always the first byte of the first Tx Buffer.
   - In both the 32-bit Segmented and Linear modes it depends on the SF bit and TCB COUNT:
     - — In the Simplified memory mode the first transmitted byte is always the third byte after the TCB COUNT field.
     - — In the Flexible mode, if the TCB COUNT is greater than 0 then it is the third byte after the TCB COUNT field. If TCB COUNT equals 0 then it is first byte of the first Tx Buffer.
   - Transmit frames shorter than six bytes are invalid. The transmission will be aborted (only in 82586 mode) because of a DMA Underrun.
4. Frames which are aborted during transmission are jammed. Such an interruption of transmission can be caused by any reason indicated by any of the status bits 8, 9, 10 and 12.

Jamming Rules

1. Jamming will not start before completion of preamble transmission.
2. Collisions detected during transmission of the last 11 bits will not result in jamming.

The format of a Transmit Buffer Descriptor is:

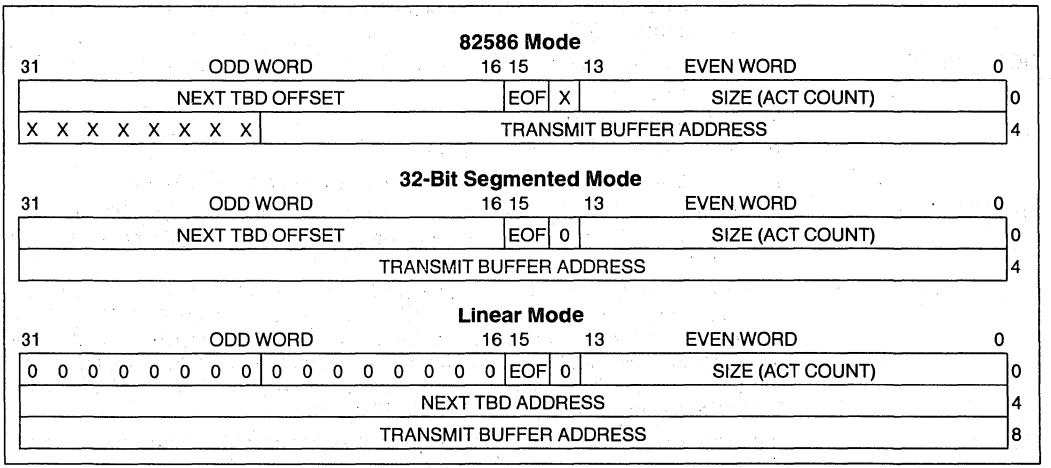


Figure 31

where:

EOF — This bit indicates that this TBD is the last one associated with the frame being transmitted. It is set by the CPU before transmit.

SIZE (ACT COUNT) — This 14-bit quantity specifies the number of bytes that hold information for the current buffer. It is set by the CPU before transmission.

NEXT TBD ADDRESS — In the 82586 and 32-bit Segmented modes, it is the offset of the next TBD on the list. In the Linear mode this is the 32-bit address of the next TBD on the list. It is meaningless if EOF = 1.

BUFFER ADDRESS — The starting address of the memory area that contains the data to be sent. In the 82586 mode, this is a 24-bit address (A31–A24 are considered to be zero). In the 32-bit Segmented and Linear modes this is a 32-bit address. This buffer can be byte aligned for the 82596 B step.

TDR

This operation activates Time Domain Reflectomet, which is a mechanism to detect open or short circuits on the link and their distance from the diagnosing station. The TDR command has no parameters. The TDR transmit sequence was changed, compared to the 82586, to form a regular transmission. The TDR bit stream is as follows.

— Preamble
— Source address
— Another Source address (the TDR frame is transmitted back to the sending station, so DEST ADR = SRC ADR).
— Data field containing 7Eh patterns.
— Jam Pattern, which is the inverse CRC of the transmitted frame.

Maximum length of the TDR frame is 2048 bits. If the 82596 senses collision while transmitting the TDR frame it transmits the jam pattern and stops the transmission. The 82596 then triggers an internal timer (STC); the timer is reset at the beginning of transmission and reset if CRS is returned. The timer measures the time elapsed from the start of transmission until an echo is returned. The echo is indicated by Collision Detect going active or a drop in the Carrier Sense signal. The following table lists the possible cases that the 82596 is able to analyze.

Conditions of TDR as Interpreted by the 82596

Transceiver Type / Condition	Ethernet	Non Ethernet
Carrier Sense was inactive for 2048-bit-time periods	Short or Open on the Transceiver Cable	NA
Carrier Sense signal dropped	Short on the Ethernet cable	NA
Collision Detect went active	Open on the Ethernet cable	Open on the Serial Link
The Carrier Sense Signal did not drop or the Collision Detect did not go active within 2048-bit time period	No Problem	No Problem

An Ethernet transceiver is defined as one that returns transmitted data on the receive pair and activates the Carrier Sense Signal while transmitting. A Non-Ethernet Transceiver is defined as one that does not do so.

The format of the Time Domain Reflectometer command is:

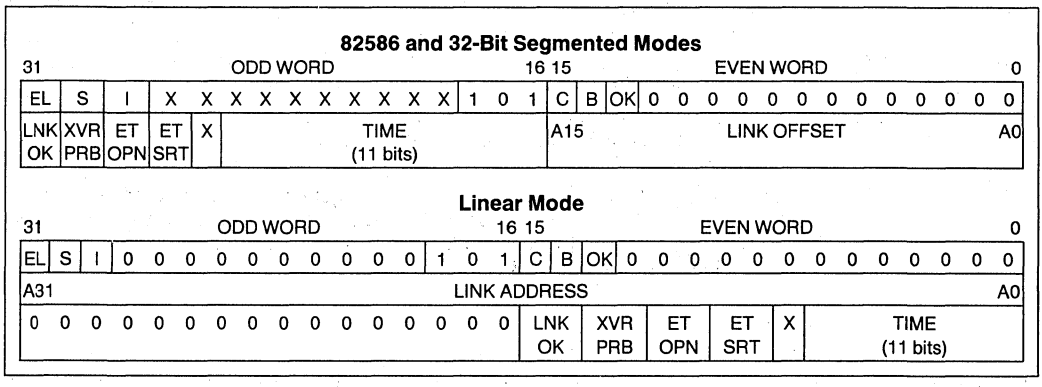


Figure 32. TDR

where:

LINK ADDRESS, EL, B, C, I, S — As per standard Command Block (see the NOP command for details).

A — Indicates that the command was abnormally terminated due to CU Abort control command. If one, then the command was aborted, and if necessary it should be repeated. If this bit is zero, the command was not aborted.

Bits 19–28 — Reserved (0 in the 32-bit Segmented and Linear Modes).

CMD (Bits 16–18) — The TDR command. Value: 5h.

TIME — An 11-bit field that specifies the number of TxC cycles that elapsed before an echo was observed. No echo is indicated by a reception consisting of "1s" only. Because the network contains various elements such as transceiver links, transceivers, Ethernet, repeaters etc., the TIME is not exactly proportional to the problems distance.

LNK OK (Bit 15) — No link problem identified. TIME = 7FFh.

XCVR PRB (Bit 14) — Indicates a Transceiver problem. Carrier Sense was inactive for 2048-bit time period. LNK OK = 0. TIME = 7FFh.

ET OPN (Bit 13) — The transmission line is not properly terminated. Collision Detect went active and LNK OK = 0.

ET SRT (Bit 12) — There is a short circuit on the transmission line. Carrier Sense Signal dropped and LNK OK = 0.

DUMP

This command causes the contents of various 82596 registers to be placed in a memory area specified by the user. It is supplied as a 82596 self-diagnostic tool, and to provide registers of interest to the user. The format of the DUMP command is:

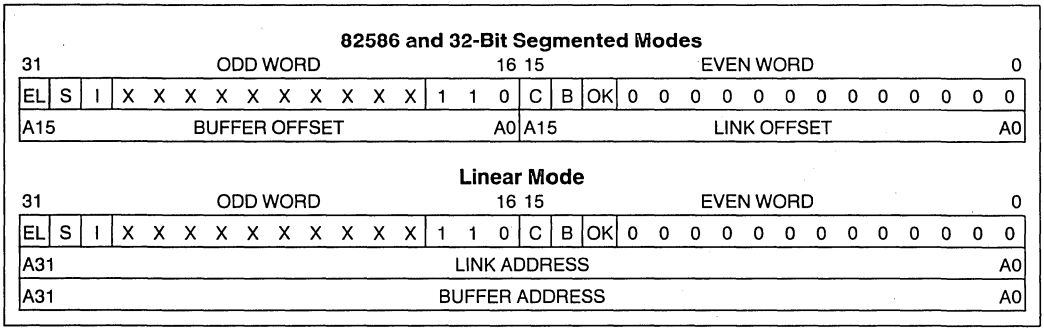


Figure 33. Dump

where:

LINK ADDRESS, EL, B, C, I, S — As per standard Command Block (see the NOP command for details).

OK — Indicates error free completion.

Bits 19–28 — Reserved (0 in the 32-bit Segmented and Linear Modes).

CMD (Bits 16–18) — The Dump command. Value: 6h.

BUFFER POINTER — In the 82586 and 32-bit Segmented modes this is the 16-bit-offset portion of the dump area address. In the Linear mode this is the 32-bit linear address of the dump area.

Dump Area Information Format

- The 82596 is not Dump compatible with the 82586 because of the 32-bit internal architecture. In 82586 mode the 82596 will dump the same number of bytes as the 82586. The compatible data will be marked with an asterisk.
- In 82586 mode the dump area is 170 bytes.
- The DUMP area format of the 32-bit Segmented and Linear modes is described in Figure 35.
- The size of the dump area of the 32-bit Segmented and Linear modes is 304 bytes.
- When the Dump is executed by the Port command an extra word will be appended to the Dump Area. The extra word is a copy of the Dump Area status word (containing the C, B, and OK Bits). The C and OK Bits are set when the 82596 has completed the Port Dump command.

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	
DMA CONTROL REGISTER	00
CONFIGURE BYTES* 3, 2	02
CONFIGURE BYTES* 5, 4	04
CONFIGURE BYTES* 7, 6	06
CONFIGURE BYTES* 9, 8	08
CONFIGURE BYTES* 10	0A
I.A. BYTES 1, 0*	0C
I.A. BYTES 3, 2*	0E
I.A. BYTES 5, 4*	10
LAST T.X. STATUS*	12
T.X. CRC BYTES 1, 0*	14
T.X. CRC BYTES 3, 2*	16
R.X. CRC BYTES 1, 0*	18
R.X. CRC BYTES 3, 2*	1A
R.X. TEMP MEMORY 1, 0*	1C
R.X. TEMP MEMORY 3, 2*	1E
R.X. TEMP MEMORY 5, 4*	20
LAST RECEIVED STATUS*	22
HASH REGISTER BYTES 1, 0*	24
HASH REGISTER BYTES 3, 2*	26
HASH REGISTER BYTES 5, 4*	28
HASH REGISTER BYTES 7, 6*	2A
SLOT TIME COUNTER*	2C
WAIT TIME COUNTER*	2E
MICRO MACHINE**	30
REGISTER FILE	.
60 BYTES	6A
MICRO MACHINE LFSR**	6C
MICRO MACHINE**	6E
FLAG ARRAY	.
14 BYTES	7A
QUEUE MEMORY**	7C
CU PORT	.
8 BYTES	82
MICRO MACHINE ALU**	84
RESERVED**	86
M.M. TEMP A ROTATE R**	88
M.M. TEMP A**	8A
T.X. DMA BYTE COUNT**	8C
M.M. INPUT PORT ADDRESS**	8E
T.X. DMA ADDRESS	90
M.M. OUTPUT PORT**	92
R.X. DMA BYTE COUNT**	94
M.M. OUTPUT PORT ADDRESS REGISTER**	96
R. DMA ADDRESS**	98
RESERVED**	9A
BUS THROTTLE TIMERS	9C
DIU CONTROL REGISTER**	9E
RESERVED**	A0
DMA CONTROL REGISTER**	A2
BIU CONTROL REGISTER**	A4
M.M. DISPATCHER REG.**	A6
M.M. STATUS REGISTER**	A8

*The 82596 is not Dump compatible with the 82586 because of the 32-bit internal architecture. In 82586 mode the 82596 will dump the same number of bytes as the 82586.

**These bytes are not user defined, results may vary from Dump command to Dump command.

Figure 34. Dump Area Format—82586 Mode

31	0	
CONFIGURE BYTES 5, 4, 3, 2		00
CONFIGURE BYTES 9, 8, 7, 6		04
CONFIGURE BYTES 13, 12, 11, 10		08
I.A. BYTES 1, 0	X X X X X X X X	0C
I.A. BYTES 5, 2		10
TX CRC BYTES 0, 1	LAST T.X. STATUS	14
RX CRC BYTES 0, 1	TX CRC BYTES 3, 2	18
RX TEMP MEMORY 1, 0	RX CRC BYTES 3, 2	1C
R.X. TEMP MEMORY 5, 2		20
HASH REGISTERS 1, 0	LAST R.X. STATUS	24
HASH REGISTER BYTES 5, 2		28
SLOT TIME COUNTER	HASH REGISTERS 7, 6	2C
RECEIVE FRAME LENGTH	WAIT-TIME COUNTER	30
MICRO MACHINE** REGISTER FILE 128 BYTES		34 . . . B0
MICRO MACHINE LFSR**		B4
MICRO MACHINE** FLAG ARRAY 28 BYTES		B8 . . . D0
M.M. INPUT PORT** 16 BYTES		D4 . E0
MICRO MACHINE ALU**		E4
RESERVED**		E8
M.M. TEMP A ROTATE R.**		EC
M.M. TEMP A**		F0
T.X. DMA BYTE COUNT**		F4
M.M. INPUT PORT ADDRESS REGISTER**		F8
T.X. DMA ADDRESS**		FC
M.M. OUTPUT PORT REGISTER**		100
R.X. DMA BYTE COUNT**		104
M.M. OUTPUT PORT ADDRESS REGISTER**		108
R.X. DMA ADDRESS REGISTER**		10C
RESERVED**		110
BUS THROTTLE TIMERS		114
DIU CONTROL REGISTER**		118
RESERVED**		11C
DMA CONTROL REGISTER**		120
BIU CONTROL REGISTER**		124
M.M. DISPATCHER REG.**		128
M.M. STATUS REGISTER**		12C

The 82596 is not Dump compatible with the 82586 because of the 32-bit internal architecture. In 82586 mode the 82596 will dump the same number of bytes as the 82586.
**These bytes are not user defined, results may vary from Dump command to Dump command.

Figure 35. Dump Area Format—Linear and 32-Bit Segmented Mode

Diagnose

The Diagnose Command triggers an internal self-test procedure that checks internal 82596 hardware, which includes:

- Exponential Backoff Random Number Generator (Linear Feedback Shift Register).
- Exponential Backoff Timeout Counter.
- Slot Time Period Counter.
- Collision Number Counter.
- Exponential Backoff Shift Register.
- Exponential Backoff Mask Logic.
- Timer Trigger Logic.

This procedure checks the operation of the Backoff block, which resides in the serial side and is not easily controlled. The Diagnose command is performed in two phases.

The format of the 82596 Diagnose command is:

82586 and 32-Bit Segmented Modes

31			ODD WORD										16	15							EVEN WORD										0
EL	S	I	X	X	X	X	X	X	X	X	X	X	1	1	1	C	B	OK	0	F	0	0	0	0	0	0	0	0	0	0	0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	A15					LINK OFFSET										A0

Linear Mode

31			ODD WORD										16	15							EVEN WORD										0
EL	S	I	0	0	0	0	0	0	0	0	0	0	1	1	1	C	B	OK	0	F	0	0	0	0	0	0	0	0	0	0	0
A31																					LINK ADDRESS										A0

Figure 36. Diagnose

where:

LINK ADDRESS, EL, B, C, I, S	— As per standard Command Block (see the NOP command for details).
Bits 19–28	— Reserved (0 in the 32-bit Segmented and Linear Modes).
CMD (bits 16–18)	— The Diagnose command. Value: 7h.
OK (bit 13)	— Indicates error free completion.
F (bit 11)	— Indicates that the self-test procedure has failed.

RECEIVE FRAME DESCRIPTOR

Each received frame is described by one Receive Frame Descriptor (see Figure 37). Two new memory structures are available for the received frames. The structures are available only in the Linear and 32-bit Segmented modes.

Simplified Memory Structure

The first is the Simplified memory structure, the data section of the received frame is part of the RFD and is located immediately after the Length Field. Receive Buffer Descriptors are not used with the Simplified structure, it is primarily used to make programming easier. If the length of the data area described in the Size Field is smaller than the incoming frame, the following happens.

1. The received frame is truncated.
2. The No Resource error counter is updated.
3. If the 82596 is configured to Save Bad Frames the RFD is not reused; otherwise, the same RFD is used to hold the next received frame, and the only action taken regarding the truncated frame is to update the counter.
4. The 82596 continues to receive the next frame in the next RFD.

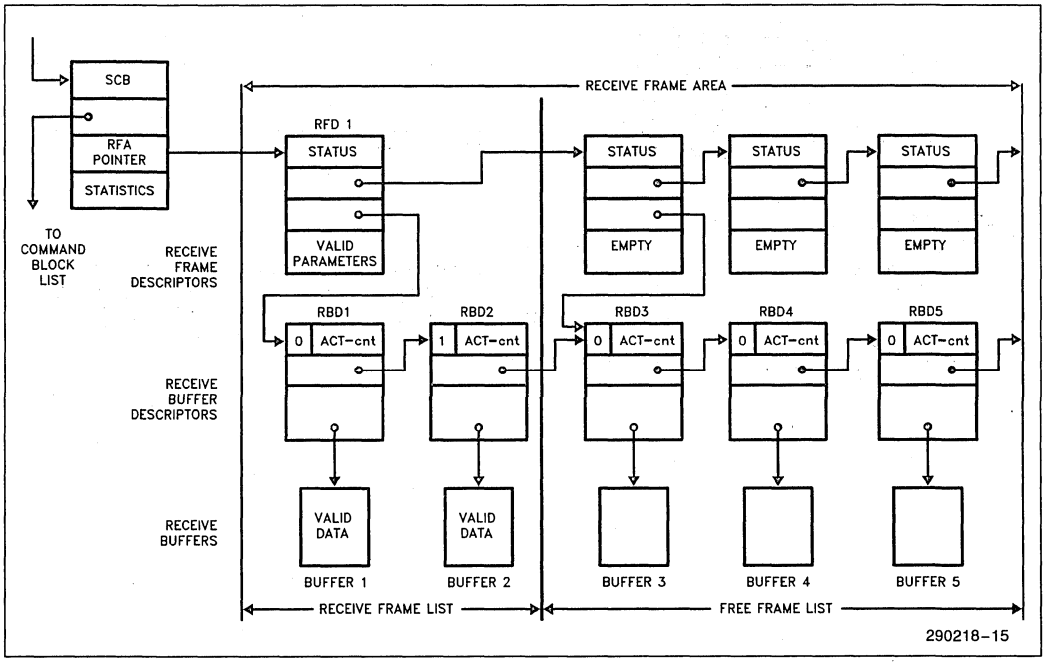


Figure 37. The Receive Frame Area

Note that this sequence is very useful for monitoring. If the 82596 is configured to Save Bad Frames, to receive in Promiscuous mode, and to use the Simplified memory structure, any programmed length of received data can be saved in memory.

The Simplified memory structure is shown in Figure 38.

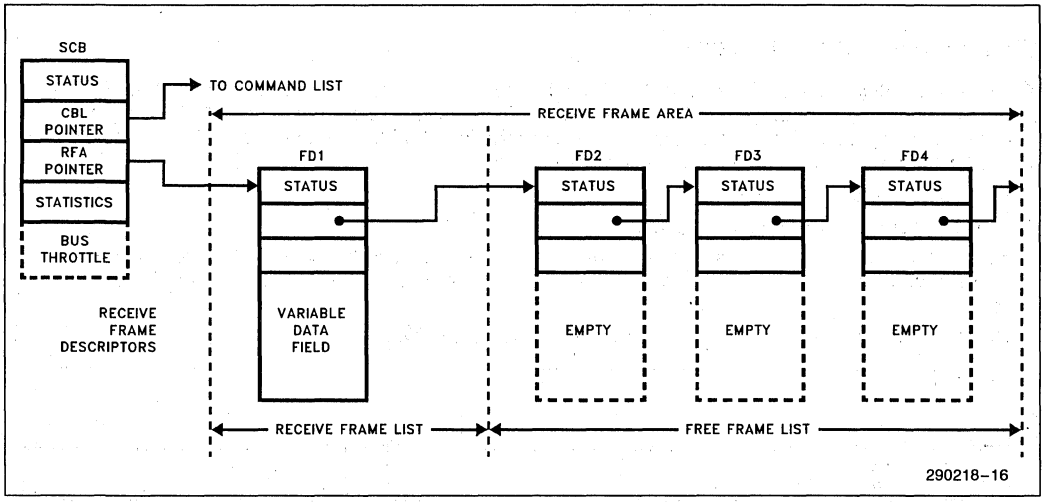


Figure 38. RFA Simplified Memory Structure

Flexible Memory Structure

The second structure is the Flexible memory structure, the data structure of the received frame is stored in both the RFD and in a linked list of Receive Buffers—Receive Buffer Descriptors. The received frame is placed in the RFD as configured in the Size field. Any remaining data is placed in a linked list of RBDs.

The Flexible memory structure is shown in Figure 39.

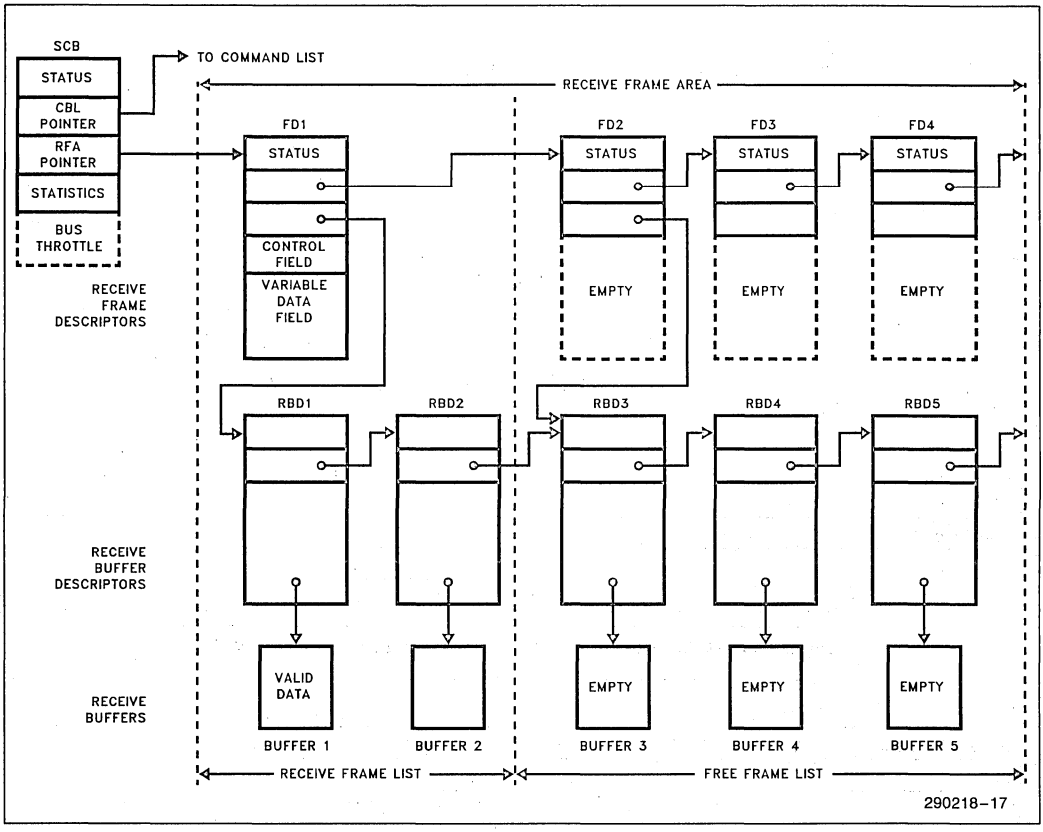


Figure 39. RFA Flexible Memory Structure

Buffers on the receive side can be different lengths. The 82596 will not place more bytes into a buffer than indicated in the associated RBD. The 82596 will fetch the next RBD before it is needed. The 82596 will attempt to receive frames as long as the FBL is not exhausted. If there are no more buffers, the 82596 Receive Unit will enter the No Resources state. Before starting the RU, the CPU must place the FBL pointer in the RBD pointer field of the first RFD. All remaining RBD pointer fields for subsequent RFDs should be "1s." If the Receive Frame Descriptor and the associated Receive Buffers are not reused (e.g., the frame is properly received or the 82596 is configured to Save Bad Frames), the 82596 writes the address of the next free RBD to the RBD pointer field of the next RFD.

Receive Buffer Descriptor (RBD)

The RBDs are used to store received data in a flexible set of linked buffers. The portion of the frame's data field that is outside the RFD is placed in a set of buffers chained by a sequence of RBDs. The RFD points to the first RBD, and the last RBD is flagged with an EOF bit set to 1. Each buffer in the linked list of buffers related to a particular frame can be any size up to 2^{14} bytes but must be word aligned (begin on an even numbered byte). This ensures optimum use of the memory resources while maintaining low overhead. All buffers in a frame are filled with the received data except for the last, in which the actual count can be smaller than the allocated buffer space.

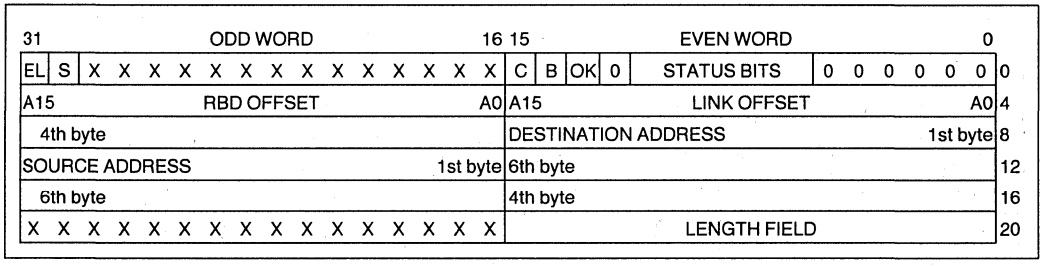


Figure 40. Receive Frame Descriptor—82586 Mode

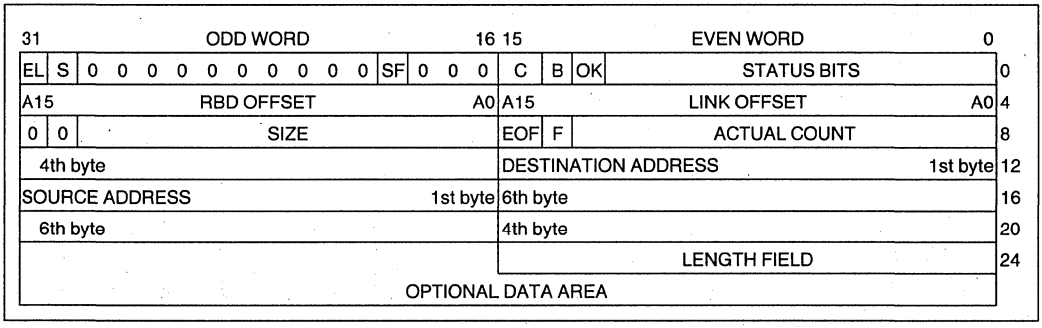


Figure 41. Receive Frame Descriptor—32-Bit Segmented Mode

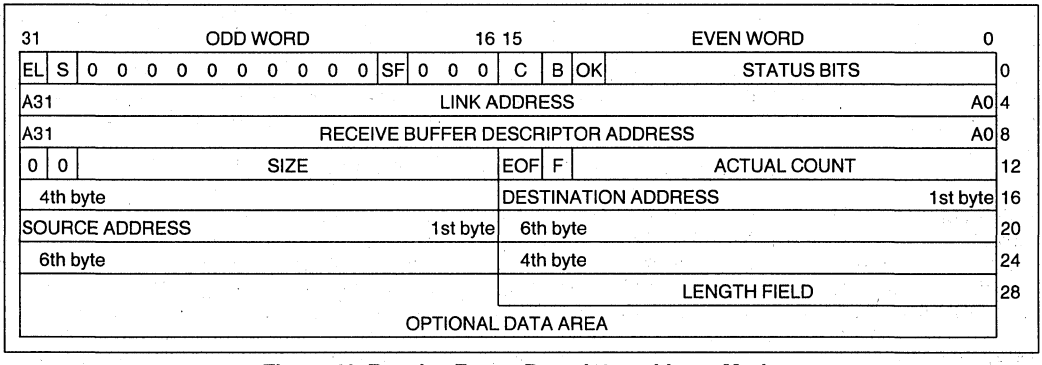


Figure 42. Receive Frame Descriptor—Linear Mode

where:

EL — When set, this bit indicates that this RFD is the last one on the RDL.

S — When set, this bit suspends the RU after receiving the frame.

SF — This bit selects between the Simplified or the Flexible mode.

0 — Simplified mode, all the RX data is in the RFD. RBD ADDRESS field is all "1s."

1 — Flexible mode. Data is in the RFD and in a linked list of Receive Buffer Descriptors.

C — This bit indicates the completion of frame reception. It is set by the 82596.

B — This bit indicates that the 82596 is currently receiving this frame, or that the 82596 is ready to receive the frame. It is initially set to 0 by the CPU. The 82596 sets it to 1 when reception set up begins, and to 0 upon completion. The C and B bits are set during the same operation.

OK (bit 13) — Frame received successfully, without errors. RFDs with bit 13 equal to 0 are possible only if the save bad frames, configuration option is selected. Otherwise all frames with errors will be discarded, although statistics will be collected on them.

STATUS — The results of the Receive operation. Defined bits are,

Bit 12: Length error if configured to check length

Bit 11: CRC error in an aligned frame

Bit 10: Alignment error (CRC error in misaligned frame)

Bit 9: Ran out of buffer space—no resources

Bit 8: DMA Overrun failure to acquire the system bus.

Bit 7: Frame too short.

Bit 6: No EOP flag (for Bit stuffing only)

Bit 5: When the SF bit equals zero, and the 82596 is configured to save bad frames, this bit signals that the receive frame was truncated. Otherwise it is zero.

Bits 2–4: Zeros

Bit 1: When it is zero, the destination address of the received frame matches the IA address. When it is a 1, the destination address of the received frame did not match the individual address. For example, a multicast address or broadcast address will set this bit to a 1.

Bit 0: Receive collision, a collision is detected during reception.

LINK ADDRESS — A 16-bit offset (32-bit address in the Linear mode) to the next Receive Frame Descriptor. The Link Address of the last frame can be used to form a cyclical list.

RBD POINTER — The offset (address in the Linear mode) of the first RBD containing the received frame data. An RBD pointer of all ones indicates no RBD.

EOF
F
SIZE
ACT COUNT — These fields are for the Simplified and Flexible memory models. They are exactly the same as the respective fields in the Receive Buffer Descriptor. See the next section for detailed explanation of their functions.

MC — Multicast bit.

DESTINATION ADDRESS — The contents of the destination address of the receive frame. The field is 0 to 6 bytes long.

SOURCE ADDRESS — The contents of the Source Address field of the received frame. It is 0 to 6 bytes long.

LENGTH FIELD — The contents of this 2-byte field are user defined. In 802.3 it contains the length of the data field. It is placed in memory in the same order it is received, i.e., most significant byte first, least significant byte second.

NOTES

1. The Destination address, Source address and Length fields are packed, i.e., one field immediately follows the next.
2. The affect of Address/Length Location (No Source Address Insertion) configuration parameter while receiving is as follows:

— 82586 Mode: The Destination address, Source address and Length field are not used, they are placed in the RX data buffers.

— 32-Bit Segmented and Linear Modes: when the Simplified memory model is used, the Destination address, Source address and Length fields reside in their respective fields in the RFD. When the Flexible memory strucrture is used the Destination address, Source address, and Length field locations depend on the SIZE field of the RFD. They can be placed in the RFD, in the RX data buffers, or partially in the RFD and the rest in the RX data buffers, depending on the SIZE field value.

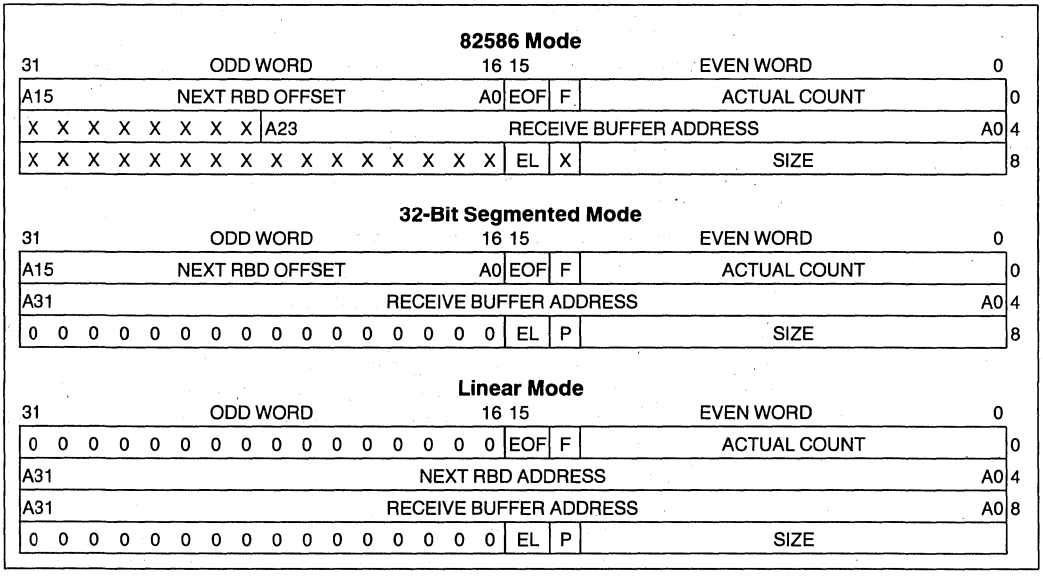


Figure 43. Receive Buffer Descriptor

where:

EOF — Indicates that this is the last buffer related to the frame. It is cleared by the CPU before starting the RU, and is written by the 82596 at the end of reception of the frame.

F — Indicates that this buffer has already been used. The Actual Count has no meaning unless the F bit equals one. This bit is cleared by the CPU before starting the RU, and is set by the 82596 after the associated buffer has been. This bit has the same meaning as the Complete bit in the RFD and CB.

ACT COUNT — This 14-bit quantity indicates the number of meaningful bytes in the buffer. It is cleared by the CPU before starting the RU, and is written by the 82596 after the associated buffer has already been used. In general, after the buffer is full, the Actual Count value equals the size field of the same buffer. For the last buffer of the frame, Actual Count can be less than the buffer size.

NEXT BD ADDRESS — The offset (absolute address in the Linear mode) of the next RBD on the list. It is meaningless if EL=1.

BUFFER ADDRESS — The starting address of the memory area that contains the received data. In the 82586 mode, this is a 24-bit address (with pins A24–A31=0). In the 32-bit Segmented and Linear modes this is a 32-bit address.

EL — Indicates that the buffer associated with this RBD is last in the FBL.

P — This bit indicates that the 82596 has already prefetched the RBDs and any change in the RBD data will be ignored. This bit is valid only in the new 82596 memory modes, and if this feature has been enabled during configure command. The 82596 Prefetches the RBDs in locked cycles; after prefetching the RBD the 82596 performs a write cycle where the P bit is set to one and the rest of the data remains unchanged. The CPU is responsible for resetting it in all RBDs. The 82596 will not check this bit before setting it.

SIZE — This 14-bit quantity indicates the size, in bytes, of the associated buffer. This quantity must be an even number.

PGA PACKAGE THERMAL SPECIFICATION

Parameter	Thermal Resistance
θ_{JC}	3°C/W
θ_{JA}	24°C/W

ELECTRICAL AND TIMING CHARACTERISTICS

Absolute Maximum Ratings

- Storage Temperature . . . . . −65°C to +150°C
- Case Temperature under Bias −65°C to +110°C
- Supply Voltage with Respect to V_{SS} . . . . . −0.5V to +6.5V
- Voltage on Other Pins . . . . −0.5V to V_{CC} + 0.5V

DC Characteristics

T_C = 0°C–85°C, V_{CC} = 5V ±10% LE/$\overline{BE}$ have MOS levels (see V_{MIL}, V_{MIH}).
All other signals have TTL levels (see V_{IL}, V_{IH}, V_{OL}, V_{OH}).

Symbol	Parameter	Min	Max	Units	Notes
V_{IL}	Input Low Voltage (TTL)	−0.3	+0.8	V	
V_{IH}	Input High Voltage (TTL)	2.0	V_{CC} + 0.3	V	
V_{MIL}	Input Low Voltage (MOS)	−0.3	+0.8	V	
V_{MIH}	Input High Voltage (MOS)	3.7	V_{CC} + 0.3	V	
V_{OL}	Output Low Voltage (TTL)		0.45	V	I_{OL} = 4.0 mA
V_{CIL}	$\overline{RXC}$, $\overline{TXC}$ Input Low Voltage	−0.5	0.6	V	
V_{CIH}	$\overline{RXC}$, $\overline{TXC}$ Input High Voltage	3.3	V_{CC}+0.5	V	
V_{OH}	Output High Voltage (TTL)	2.4		V	I_{OH} = 0.9 mA–1 mA
I_{LI}	Input Leakage Current		±15	μA	$0 \le V_{IN} \le V_{CC}$
I_{LO}	Output Leakage Current		±15	μA	$0.45 < V_{OUT} < V_{CC}$
C_{IN}	Capacitance of Input Buffer		10	pF	FC = 1 MHz
C_{OUT}	Capacitance of Input/Output Buffer		12	pF	FC = 1 MHz
C_{CLK}	CLK Capacitance		20	pF	FC = 1 MHz
I_{CC}	Power Supply		200	mA	At 25 MHz I_{CC} Typical = 100 mA
I_{CC}	Power Supply		300	mA	At 33 MHz I_{CC} Typical = 150 mA

AC Characteristics

82596CA INPUT/OUTPUT SYSTEM TIMINGS

T_C = 0°C–85°C, V_{CC} = 5V ±10%. These timing assume the C_L on all outputs is 50 pF unless otherwise specified. C_L can be 20 pF to 120 pF however timings must be derated. All timing requirements are given in nanoseconds.

Symbol	Parameter	25 MHz		Notes
		Min	Max	
	Operating Frequency	12.5 MHz	25 MHz	1X CLK Input
T1	CLK Period	40	80	
T1a	CLK Period Stability		0.1%	Adjacent CLK Δ
T2	CLK High	14		2.0V
T3	CLK Low	14		0.8V
T4	CLK Rise Time		4	0.8V to 2.0V
T5	CLK Fall Time		4	2.0V to 0.8V
T6	$\overline{BEn}$, $\overline{LOCK}$, and A2–A31 Valid Delay	3	22	
T6a	$\overline{BLAST}$, $\overline{PCHK}$ Valid Delay	3	27	
T7	$\overline{BEn}$, $\overline{LOCK}$, $\overline{BLAST}$, A2–A31 Float Delay	3	30	
T8	W/$\overline{R}$ and $\overline{ADS}$ Valid Delay	3	22	
T9	W/$\overline{R}$ and $\overline{ADS}$ Float Delay	3	30	
T10	D0–D31, DPn Write Data Valid Delay	3	22	
T11	D0–D31, DPn Write Data Float Delay	3	30	
T12	HOLD Valid Delay	3	22	
T13	CA and BREQ Setup Time	7		1, 2
T14	CA and BREQ Hold Time	3		1, 2
T15	$\overline{BS16}$ Setup Time	8		2
T16	$\overline{BS16}$ Hold Time	3		2
T17	$\overline{BRDY}$, $\overline{RDY}$ Setup Time	8		2
T18	$\overline{BRDY}$, $\overline{RDY}$ Hold Time	3		2
T19	D0–D31, DPn READ Setup Time	5		2
T20	D0–D31, DPn READ Hold Time	3		2
T21	AHOLD and HLDA Setup Time	10		1, 2
T22	AHOLD Hold Time	3		1, 2
T22a	HLDA Hold Time	3		1, 2
T23	RESET Setup Time	10		1, 2
T24	RESET Hold Time	3		1, 2
T25	INT/$\overline{INT}$ Valid Delay	1	26	
T26	CA and BREQ, $\overline{PORT}$ Pulse Width	2 T1		1, 2, 3
T27	D0–D31 CPU $\overline{PORT}$ Access Setup Time	5		2
T28	D0–D31 CPU $\overline{PORT}$ Access Hold Time	3		2
T29	$\overline{PORT}$ Setup Time	7		2
T30	$\overline{PORT}$ Hold Time	3		2
T31	$\overline{BOFF}$ Setup Time	10		2
T32	$\overline{BOFF}$ Hold Time	3		2

AC Characteristics (Continued)

82596CA INPUT/OUTPUT SYSTEM TIMINGS

T_C = 0°C–85°C, V_{CC} = 5V ±5%. These timing assume the C_L on all outputs is 50 pF unless otherwise specified. C_L can be 20 pF to 120 pF, however timings must be derated. All timing requirements are given in nanoseconds.

Symbol	Parameter	33 MHz		Notes
		Min	Max	
	Operating Frequency	12.5 MHz	33 MHz	1X CLK Input
T1	CLK Period	30	80	
T1a	CLK Period Stability		0.1%	Adjacent CLK Δ
T2	CLK High	11		2.0V
T3	CLK Low	11		0.8V
T4	CLK Rise Time		3	0.8V to 2.0V
T5	CLK Fall Time		3	2.0V to 0.8V
T6	$\overline{BEn}$, $\overline{LOCK}$, and A2–A31 Valid Delay	3	19	
T6a	$\overline{BLAST}$, $\overline{PCHK}$ Valid Delay	3	22	
T7	$\overline{BEn}$, $\overline{LOCK}$, $\overline{BLAST}$, A2–A31 Float Delay	3	20	
T8	W/$\overline{R}$ and $\overline{ADS}$ Valid Delay	3	19	
T9	W/$\overline{R}$ and $\overline{ADS}$ Float Delay	3	20	
T10	D0–D31, DPn Write Data Valid Delay	3	19	
T11	D0–D31, DPn Write Data Float Delay	3	20	
T12	HOLD Valid Delay	3	19	
T13	CA and BREQ Setup Time	7		1, 2
T14	CA and BREQ Hold Time	3		1, 2
T15	$\overline{BS16}$ Setup Time	6		2
T16	$\overline{BS16}$ Hold Time	3		2
T17	$\overline{BRDY}$, $\overline{RDY}$ Setup Time	6		2
T18	$\overline{BRDY}$, $\overline{RDY}$ Hold Time	3		2
T19	D0–D31, DPn READ Setup Time	5		2
T20	D0–D31, DPn READ Hold Time	3		2
T21	AHOLD and HLDA Setup Time	8		1, 2
T22	AHOLD Hold Time	3		1, 2

AC Characteristics (Continued)

82596CA INPUT/OUTPUT SYSTEM TIMINGS

C_L on all outputs is 50 pF unless otherwise specified.
All timing requirements are given in nanoseconds.

Symbol	Parameter	33 MHz		Notes
		Min	Max	
T22a	HLDA Hold Time	3		1, 2
T23	RESET Setup Time	8		1, 2
T24	RESET Hold Time	3		1, 2
T25	INT/$\overline{\text{INT}}$ Valid Delay	1	20	
T26	CA and BREQ, $\overline{\text{PORT}}$ Pulse Width	2T1		1, 2, 3
T27	D0–D31 CPU $\overline{\text{PORT}}$ Access Setup Time	5		2
T28	D0–D31 CPU $\overline{\text{PORT}}$ Access Hold Time	3		2
T29	$\overline{\text{PORT}}$ Setup Time	7		2
T30	$\overline{\text{PORT}}$ Hold Time	3		2
T31	$\overline{\text{BOFF}}$ Setup Time	8		2
T32	$\overline{\text{BOFF}}$ Hold Time	3		2

NOTES:
1. RESET, HLDA, and CA are internally synchronized. This timing is to guarantee recognition at next clock for RESET, HLDA and CA.
2. All set-up, hold and delay timings are at maximum frequency specification Fmax, and must be derated according to the following equation for operation at lower frequencies:
   Tderated = (Fmax/Fopr) × T
   where:
   Tderate = Specifies the value to derate the specification.
   Fmax = Maximum operating frequency.
   Fopr = Actual operating frequency.
   T = Specification at maximum frequency.
   This calculation only provides a rough estimate for derating the frequency. For more detailed information, contact your Intel Sales Office for the data sheet supplement.
3. CA pulse width need only be 1 T1 wide if the set up and hold times are met; BREQ must meet setup and hold times and need only be 1 T1 wide.

4

TRANSMIT/RECEIVE CLOCK PARAMETERS

Symbol	Parameter	20 MHz		Notes
		Min	Max	
T36	$\overline{\text{TxC}}$ Cycle	50		1, 3
T38	$\overline{\text{TxC}}$ Rise Time		5	1
T39	$\overline{\text{TxC}}$ Fall Time		5	1
T40	$\overline{\text{TxC}}$ High Time	19		1, 3
T41	$\overline{\text{TxC}}$ Low Time	18		1, 3
T42	TxD Rise Time		10	4
T43	TxD Fall Time		10	4
T44	TxD Transition	20		2, 4
T45	$\overline{\text{TxC}}$ Low to TxD Valid		25	4, 6
T46	$\overline{\text{TxC}}$ Low to TxD Transition		25	2, 4
T47	$\overline{\text{TxC}}$ High to TxD Transition		25	2, 4
T48	$\overline{\text{TxC}}$ Low to TxD High (At End of Transition)		25	4

TRANSMIT/RECEIVE CLOCK PARAMETERS (Continued)

Symbol	Parameter	20 MHz		Notes
		Min	Max	
$\overline{\text{RTS}}$ AND $\overline{\text{CTS}}$ PARAMETERS				
T49	$\overline{\text{TxC}}$ Low to $\overline{\text{RTS}}$ Low, Time to Activate $\overline{\text{RTS}}$		25	5
T50	$\overline{\text{CTS}}$ Low to $\overline{\text{TxC}}$ Low, $\overline{\text{CTS}}$ Setup Time		20	
T51	$\overline{\text{TxC}}$ Low to $\overline{\text{CTS}}$ Invalid, $\overline{\text{CTS}}$ Hold Time	10		7
T52	$\overline{\text{TxC}}$ Low to $\overline{\text{RTS}}$ High		25	5
RECEIVE CLOCK PARAMETERS				
T53	$\overline{\text{RXC}}$ Cycle	50		1, 3
T54	$\overline{\text{RXC}}$ Rise Time		5	1
T55	$\overline{\text{RXC}}$ Fall Time		5	1
T56	$\overline{\text{RXC}}$ High Time	19		1
T57	$\overline{\text{RXC}}$ Low Time	18		1
RECEIVED DATA PARAMETERS				
T58	RXD Setup Time	20		6
T59	RXD Hold Time	10		6
T60	RXD Rise Time		10	
T61	RXD Fall Time		10	
$\overline{\text{CRS}}$ AND $\overline{\text{CDT}}$ PARAMETERS				
T62	$\overline{\text{CDT}}$ Low to $\overline{\text{TXC}}$ HIGH External Collision Detect Setup Time	20		
T63	$\overline{\text{TXC}}$ High to $\overline{\text{CDT}}$ Inactive, $\overline{\text{CDT}}$ Hold Time	10		
T64	$\overline{\text{CDT}}$ Low to Jam Start			10
T65	$\overline{\text{CRS}}$ Low to $\overline{\text{TXC}}$ High, Carrier Sense Setup Time	20		
T66	$\overline{\text{TXC}}$ High to $\overline{\text{CRS}}$ Inactive, $\overline{\text{CRS}}$ Hold Time (Internal Collision Detect)	10		
T67	$\overline{\text{CRS}}$ High to Jamming Start,			12
T68	Jamming Period			11
T69	$\overline{\text{CRS}}$ High to $\overline{\text{RXC}}$ High, $\overline{\text{CRS}}$ Inactive Setup Time	30		
T70	$\overline{\text{RXC}}$ High to $\overline{\text{CRS}}$ High, $\overline{\text{CRS}}$ Inactive Hold Time	10		

TRANSMIT/RECEIVE CLOCK PARAMETERS (Continued)

Symbol	Parameter	20 MHz		Notes
		Min	Max	
INTERFRAME SPACING PARAMETERS				
T71	Interframe Delay			9
EXTERNAL LOOPBACK-PIN PARAMETERS				
T72	$\overline{TXC}$ Low to $\overline{LPBK}$ Low		T36	4
T73	$\overline{TXC}$ Low to $\overline{LPBK}$ High		T36	4

NOTES:
1. Special MOS levels. V_{CIL} = 0.9V and V_{CIH} = 3.0V.
2. Manchester only.
3. Manchester. Needs 50% duty cycle.
4. 1 TTL load + 50 pF.
5. 1 TTL load + 100 pF.
6. NRZ only.
7. Abnormal end of transmission—CTS expires before RTS.
8. Normal end to transmission.
9. Programmable value:
   T71 = N_{IFS} • T36
   where: N_{IFS} = the IFS configuration value
   (if N_{IFS} is less than 12 then N_{IFS} is forced to 12).
10. Programmable value:
    T64 = (N_{CDF} • T36) + x • T36
    (If the collision occurs after the preamble)
    where:
    N_{CDF} = the collision detect filter configuration value,
    and
    x = 12, 13, 14, or 15
11. T68 = 32 • T36
12. Programmable value:
    T67 = (N_{CSF} • T36) + x • T36
    where: N_{CSF} = the Carrier Sense Filter configuration value, and
    x = 12, 13, 14, or 15
13. To guarantee recognition on the next clock.

4

82596CA BUS OPERATION

The following figures show the 82596CA basic bus cycle and basic burst cycle.

Please refer to the *32-Bit LAN Component User's Manual.*

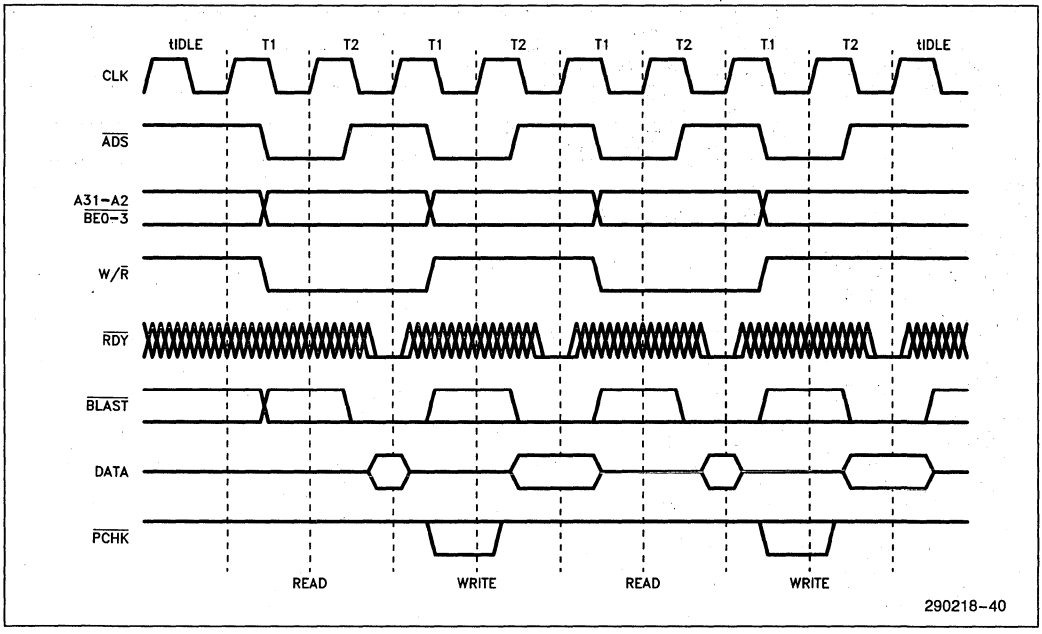


Figure 44. Basic 82596CA Bus Cycle

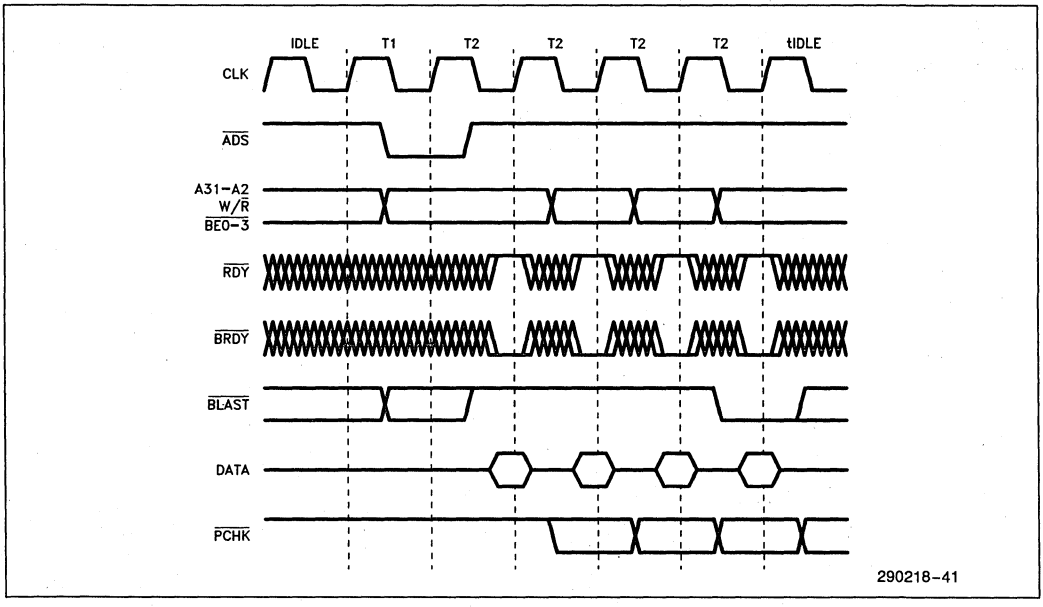


Figure 45. Basic 82596CA Burst Cycle

SYSTEM INTERFACE A.C. TIMING CHARACTERISTICS

The measurements should be done at:

- T_C = 0°C–85°C, V_{CC} = 5V ±10%, C = 50 pF unless otherwise specified.
- A.C. testing inputs are driven at 2.4V for a logic "1" and 0.45V for a logic "0".
- Timing measurements are made at 1.5V for both logic "1" and "0".
- Rise and Fall time of inputs and outputs signals are measured between 0.8V and 2.0V respectively unless otherwise specified.
- All timings are relative to CLK crossing the 1.5V level.
- All A.C. parameters are valid only after 100 μs from power up.

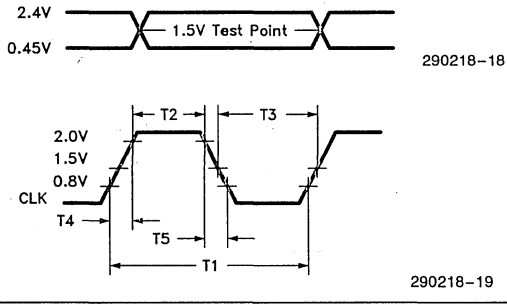
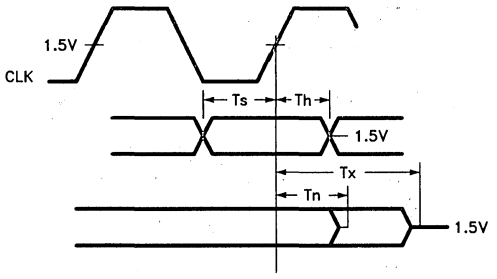


Figure 46. CLK Timings

Two types of timing specifications are presented below:

1. Input Timing—minimum setup and hold times.
2. Output Timings—output delays and float times from CLK rising edge.

Figure 47 defines how the measurements should be done:



LEGEND:
Ts = Input Setup Time
Th = Input Hold Time
Tn = Minimum output delay or Mininum float delay
Tx = Maximum output delay or Maximum float delay

Figure 47. Drive Levels and Measurements Points for A.C. Specifications

Ts = T13, T15, T17, T19, T21, T23, T27, T29, T31
Th = T14, T16, T18, T20, T22, T22a, T24, T28, T30, T32
Tn = T6, T6a, T7, T8, T9, T10, T11, T12, T25
Tx = T6, T6a, T7, T8, T9, T10, T11, T12, T25

INPUT WAVEFORMS

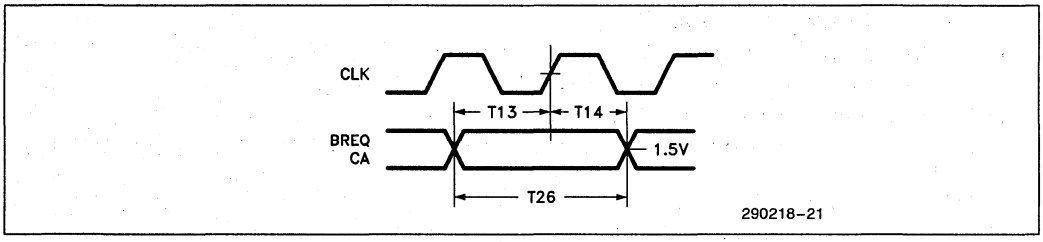


Figure 48. CA and BREQ Input Timing

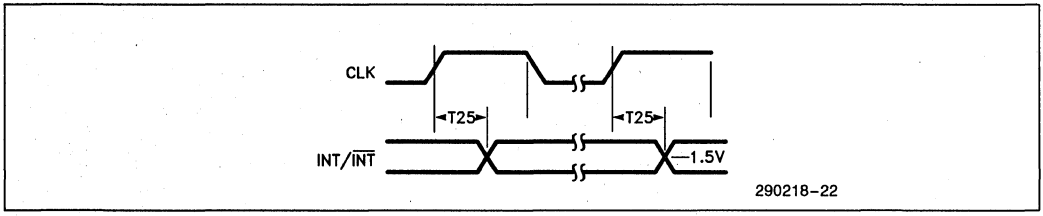


Figure 49. INT/INT Output Timing

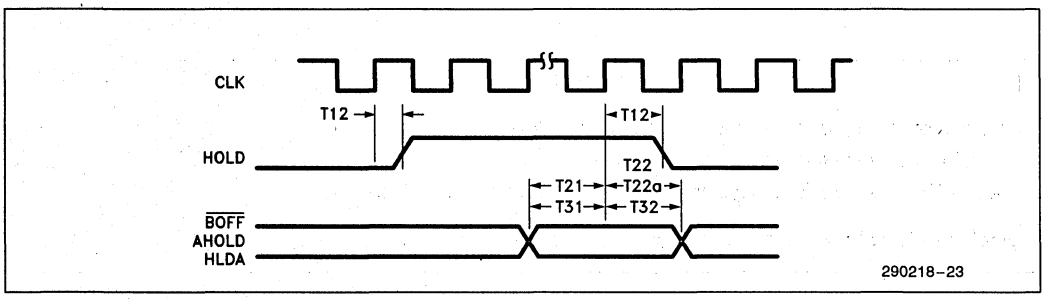


Figure 50. HOLD/HLDA Timings

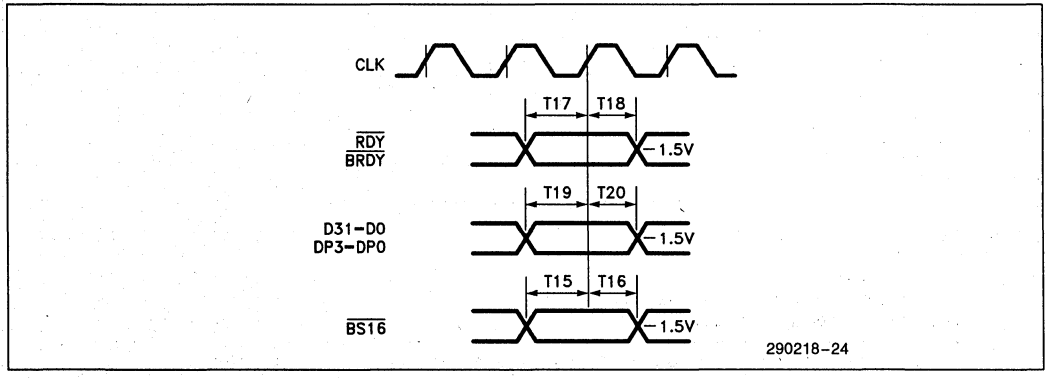


Figure 51. Input Setup and Hold Time

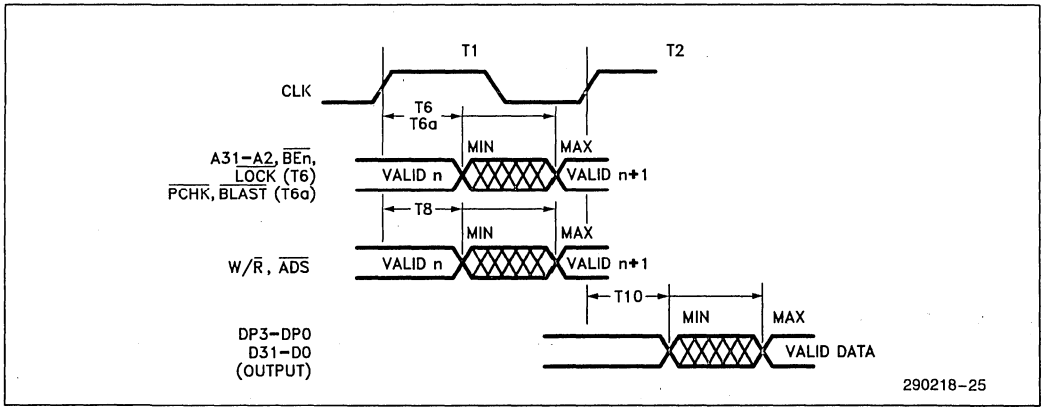


Figure 52. Output Valid Delay Timing

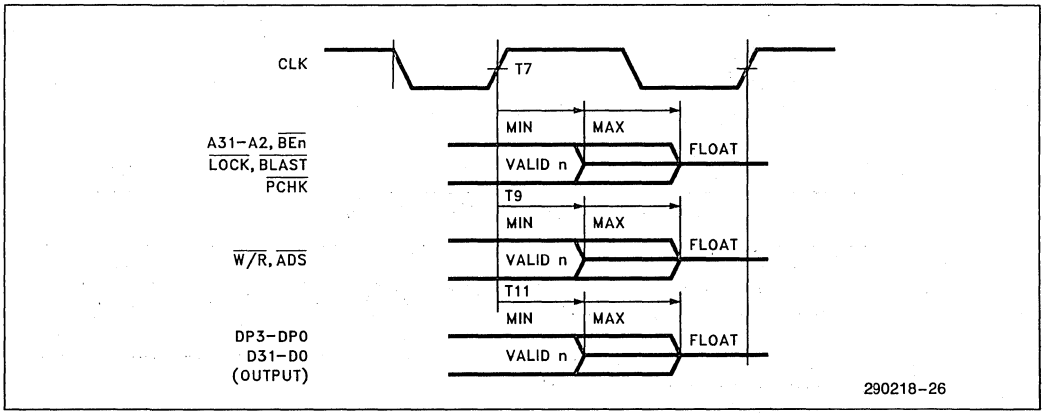


Figure 53. Output Float Delay Timing

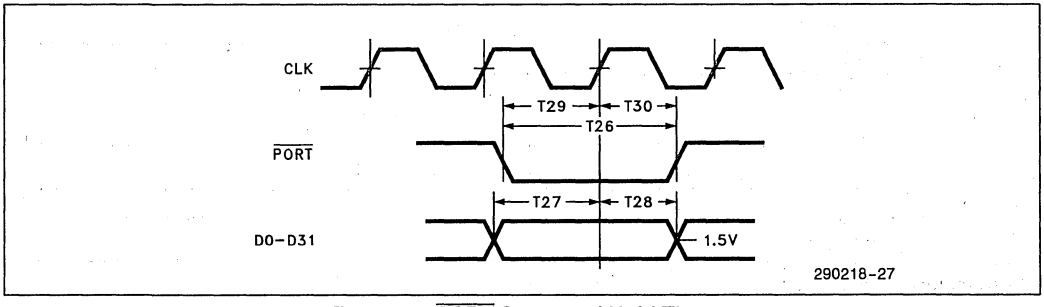


Figure 54. PORT Setup and Hold Time

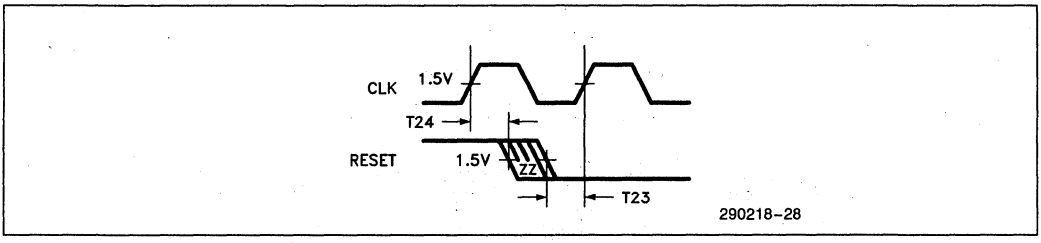


Figure 55. RESET Input Timing

SERIAL AC TIMING CHARACTERISTICS

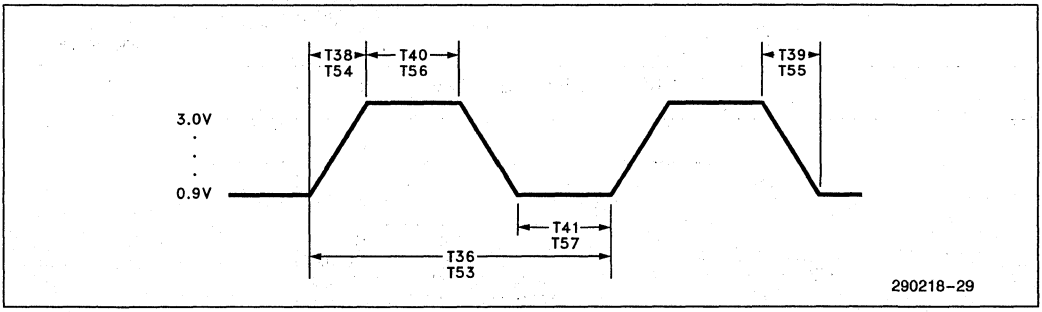


Figure 56. Serial Input Clock Timing

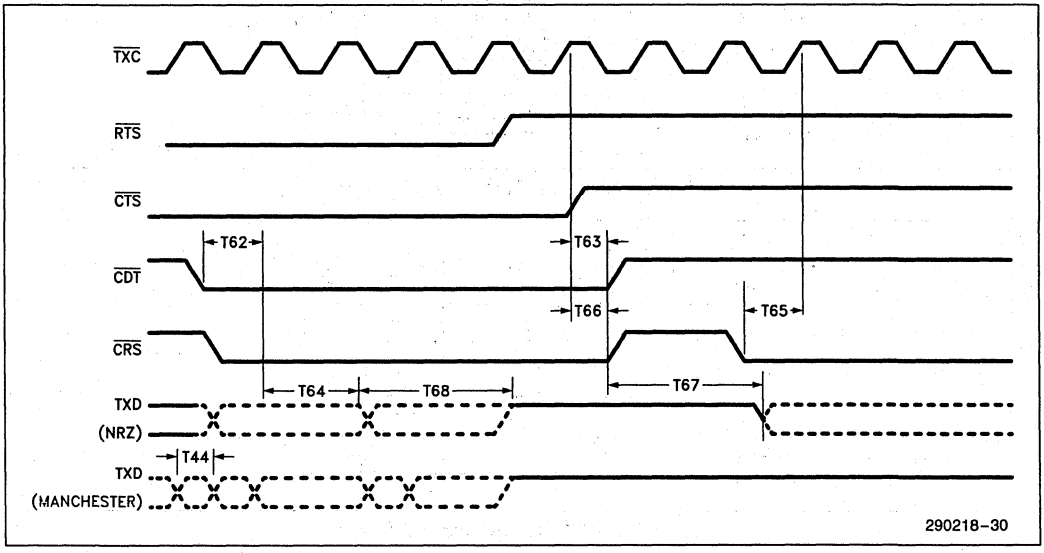


Figure 57. Transmit Data Waveforms

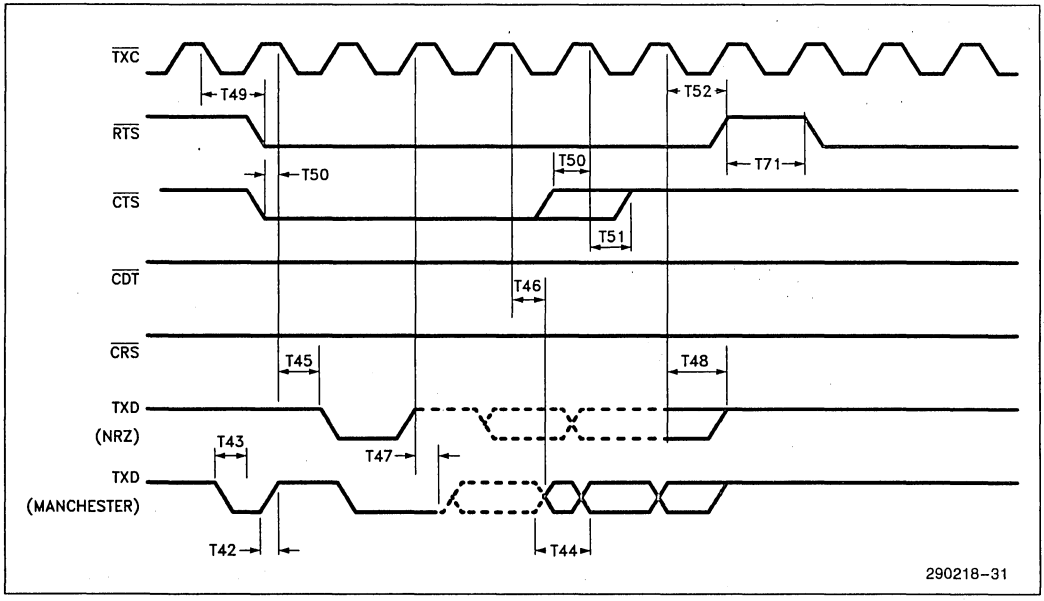


Figure 58. Transmit Data Waveforms

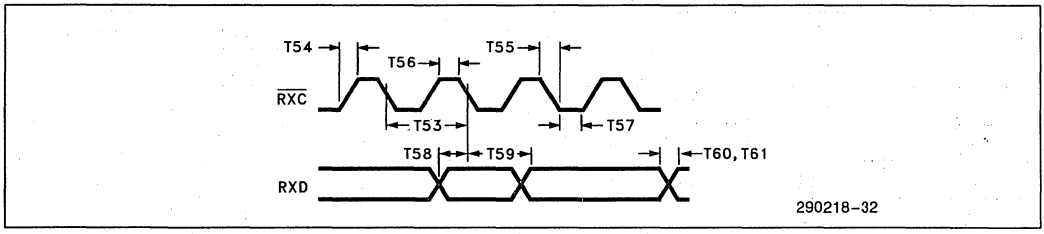


Figure 59. Receive Data Waveforms (NRZ)

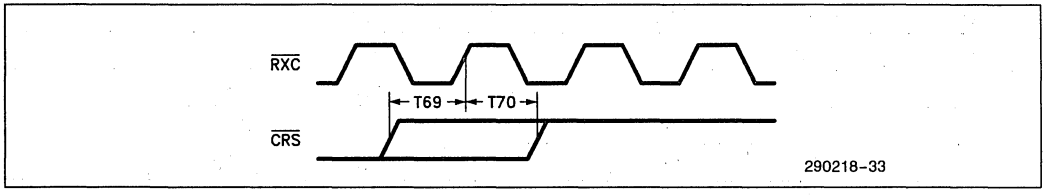
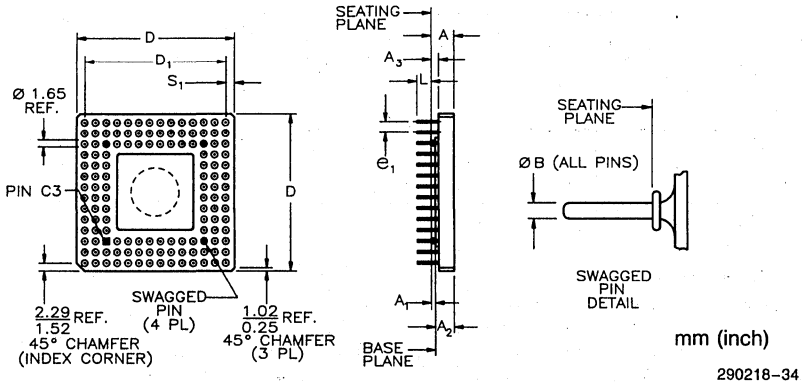


Figure 60. Receive Data Waveforms (CRS)

OUTLINE DIAGRAMS

132 LEAD CERAMIC PIN GRID ARRAY PACKAGE INTEL TYPE A



Family: Ceramic Pin Grid Array Package						
Symbol	Millimeters			Inches		
	Min	Max	Notes	Min	Max	Notes
A	3.56	4.57		0.140	0.180	
A_1	0.76	1.27	Solid Lid	0.030	0.050	Solid Lid
A_2	2.67	3.43	Solid Lid	0.105	0.135	Solid Lid
A_3	1.14	1.40		0.045	0.055	
B	0.43	0.51		0.017	0.020	
D	36.45	37.21		1.435	1.465	
D_1	32.89	33.15		1.295	1.305	
e_1	2.29	2.79		0.090	0.110	
L	2.54	3.30		0.100	0.130	
N	132			132		
S_1	1.27	2.54		0.050	0.100	
ISSUE	IWS 10/12/88					

Intel Case Outline Drawings
Plastic Quad Flat Pack (PQFP)
0.025 Inch (0.635mm) Pitch

Symbol	Description	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max
N	Leadcount	68		84		100		132		164		196	
A	Package Height	0.160	0.170	0.160	0.170	0.160	0.170	0.160	0.170	0.160	0.170	0.160	0.170
A1	Standoff	0.020	0.030	0.020	0.030	0.020	0.030	0.020	0.030	0.020	0.030	0.020	0.030
D, E	Terminal Dimension	0.675	0.685	0.775	0.785	0.875	0.885	1.075	1.085	1.275	1.285	1.475	1.485
D1, E1	Package Body	0.547	0.553	0.647	0.653	0.747	0.753	0.947	0.953	1.147	1.153	1.347	1.353
D2, E2	Bumper Distance	0.697	0.703	0.797	0.803	0.897	0.903	1.097	1.103	1.297	1.303	1.497	1.503
D3, E3	Lead Dimension	0.400 REF		0.500 REF		0.600 REF		0.800 REF		1.000 REF		1.200 REF	
D4, E4	Foot Radius Location	0.623	0.637	0.723	0.737	0.823	0.837	1.023	1.037	1.223	1.237	1.423	1.437
L1	Foot Length	0.020	0.030	0.020	0.030	0.020	0.030	0.020	0.030	0.020	0.030	0.020	0.030
Issue	IWS Preliminary 12/12/88												INCH

Symbol	Description	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max
N	Leadcount	68		84		100		132		164		196	
A	Package Height	4.06	4.32	4.06	4.32	4.06	4.32	4.06	4.32	4.06	4.32	4.06	4.32
A1	Standoff	0.51	0.76	0.51	0.76	0.51	0.76	0.51	0.76	0.51	0.76	0.51	0.76
D, E	Terminal Dimension	17.15	17.40	19.69	19.94	22.23	22.48	27.31	27.56	32.39	32.64	37.47	37.72
D1, E1	Package Body	13.89	14.05	16.43	16.59	18.97	19.13	24.05	24.21	29.13	29.29	34.21	34.37
D2, E2	Bumper Distance	17.70	17.85	20.24	20.39	22.78	22.93	27.86	28.01	32.94	33.09	38.02	38.18
D3, E3	Lead Dimension	10.16 REF		12.70 REF		15.24 REF		20.32 REF		25.40 REF		30.48 REF	
D4, E4	Foot Radius Location	15.82	16.17	18.36	18.71	21.25	21.25	25.89	26.33	31.06	31.41	36.14	36.49
L1	Foot Length	0.51	0.76	0.51	0.76	0.51	0.76	0.51	0.76	0.51	0.76	0.51	0.76
Issue	IWS Preliminary 12/12/88												mm

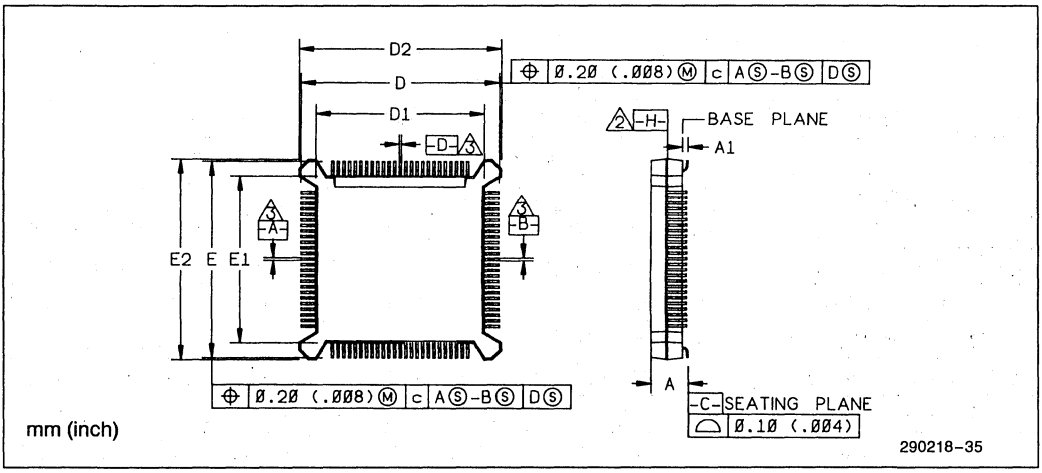


Figure 61. Principal Dimensions and Datums

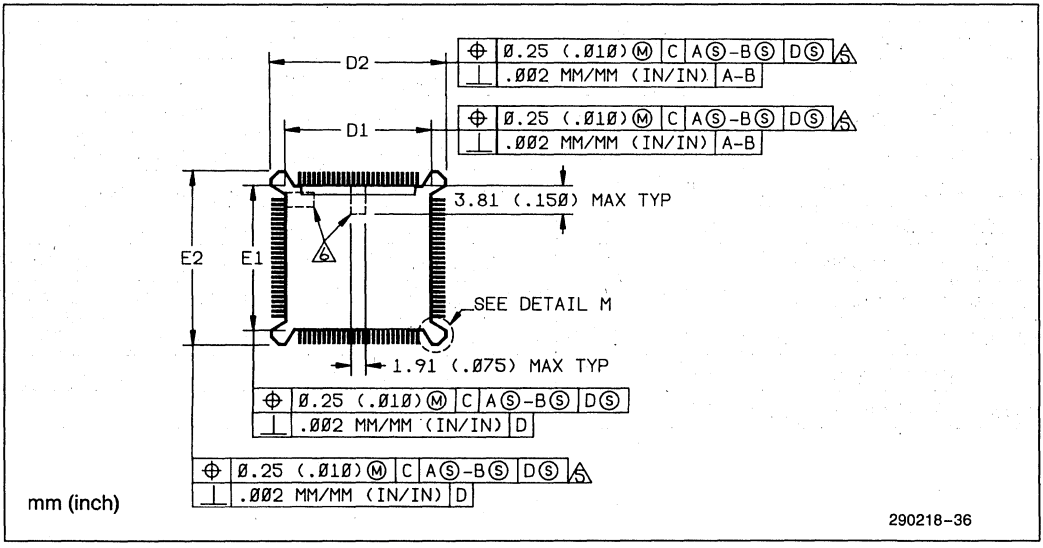


Figure 62. Molded Details

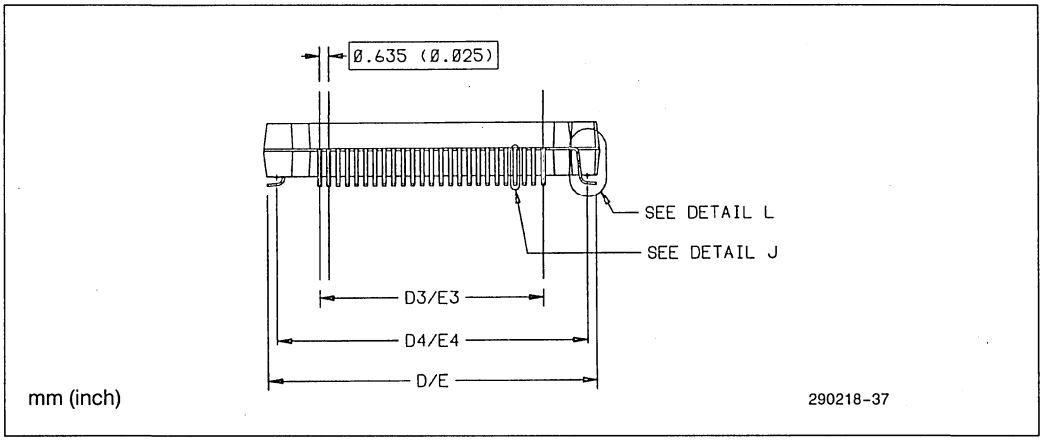


Figure 63. Terminal Details

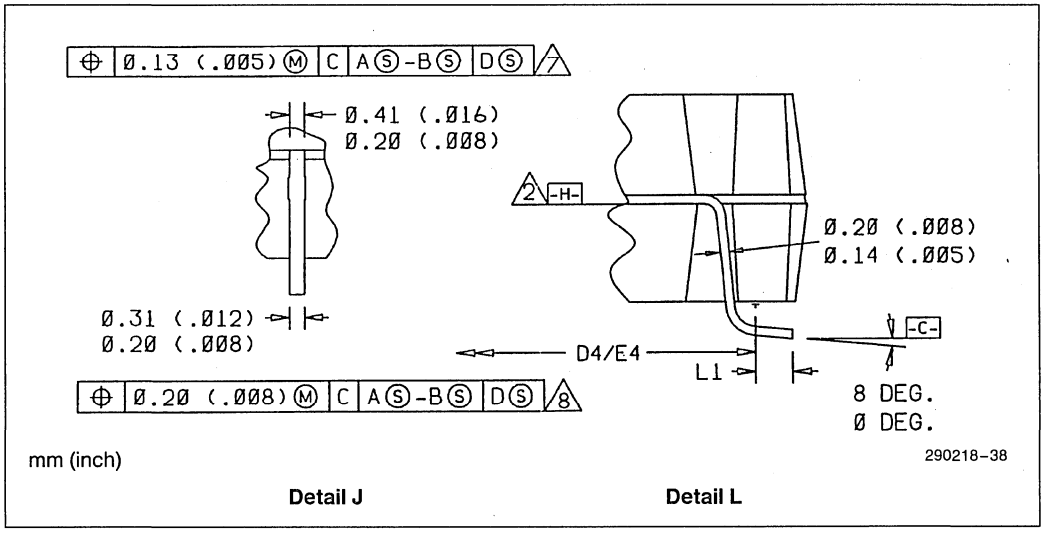
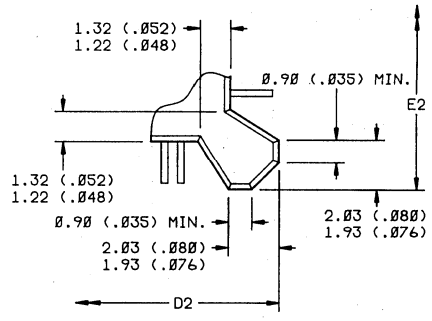


Figure 64. Typical Lead

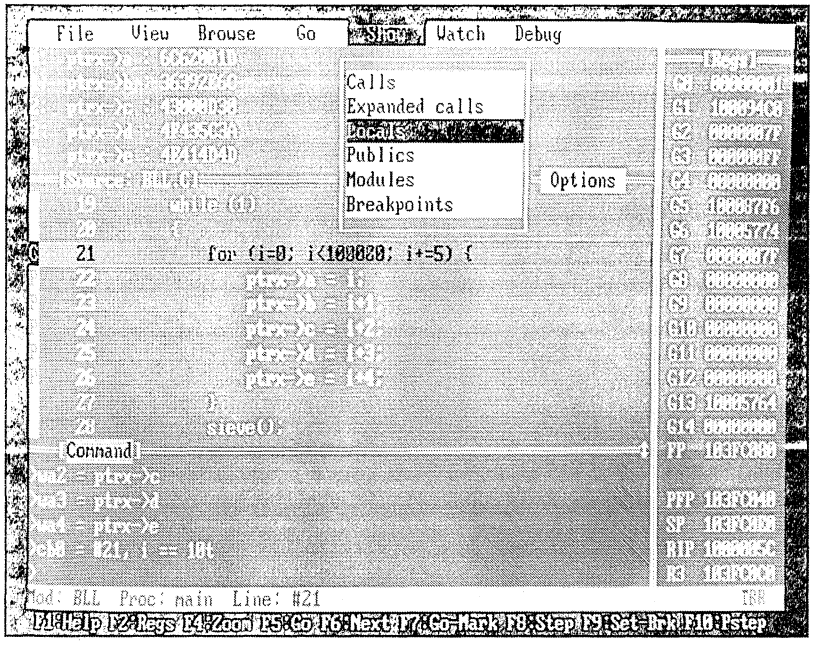


mm (inch)

Figure 65. Detail M

Development Support Tools 5

i960™ FAMILY OF SOFTWARE DEBUGGERS



280916-1

5

COMPREHENSIVE SOFTWARE DEBUG SUPPORT FOR i960™ EMBEDDED APPLICATIONS

Intel provides comprehensive software debug support for all members of the i960™ component architecture, including the newest members, the i960SA and i960SB. All Intel's i960 software debug products share the same high-level, windowed user interface emerging as the standard for all i960 tools from Intel. This innovative debug interface allows users to focus their efforts on finding bugs rather than spending time learning and manipulating the debug environment.

Intel's i960 software debug tools support a wide variety of debug environments, including code debug on a simulated target environment, a PC-based evaluation board, a serial-based Intel evaluation board, or a serial-based, customized target system.

GENERAL i960 SOFTWARE DEBUGGER FEATURES

- Windowed, pull down menu user interface shared by other i960 Development Tools
- Full symbolic debug with source level display allows C or assembly code debugging
- Debugging productivity enhanced by ability to quickly browse source code and view call stacks or symbol run-time values
- Breakpoints may be defined symbolically using module names, procedure names and line numbers
- Single step execution, code assembly/ disassembly, memory and register display/modification
- Run-time library support allows programs to access host files and perform I/O

*IBM, PC/AT, and Personal System/2 are registered trademarks of International Business Machines Corporation.
*Compaq is a registered trademark of the Compaq Corporation.
*Intel is a registered trademark of the Intel Corporation.

November 1991
Order Number: 280916-002

intel

FEATURES

EASY TO USE, POWERFUL USER INTERFACE

All i960 debuggers share the same high-level, powerful user interface as other i960 development tools. Utilizing pulldown menus, users have access to a color, windowed environment featuring source-level, symbolic debugging. Multiple, non-overlapping windows can be used to display source code, registers, variable values, and command line entries.

DEBUGGING FEATURES

High-level source or disassembled code can be displayed in the source window. Users can scroll through the source, browse from module to module in a program, scope to any executable point in the source, or instantaneously relocate from a symbol name to the location where it was defined (hyperscope operation). Symbol names in the source can be highlighted to inspect the current run-time value of program variables. Call stacks can be examined to trace execution flow.

A variety of breakpoints can be specified including source breakpoints, watch points, passpoints, or event-action breakpoints. Breakpoints can be defined symbolically using module names, procedure names and line numbers. Watch points allow users to observe a variable as it changes during program execution. Passpoints display a message when a specified instruction is executed, giving the user a non-realtime way to track execution of key code sequences without halting instruction flow. The event-action form allows complex breakpoint conditions to be set up, including data breakpoints (when supported by on-chip registers).

Users can step through program execution via a single assembly language instruction, a high-level language statement or a high-level function or procedure. Memory can be displayed or modified as common data types and all processor registers and system tables can be examined or changed.

Expressions involving symbol names, memory references, or both, can be defined as watch expressions whose values are monitored in a Watch window as a program executes. The i960 family of software debuggers also allows screen flipping between the debugger environment and the display output from the program.

Low level, run time libraries are provided that allow programs running on an i960 board to access the file system on the host or to perform I/O operations.

RETARGETABLE SOFTWARE DEBUGGER

Intel's DB-960 Retargetable Software Debugger is a combination application and system level debugger designed for use with the i960 family of embedded microprocessors. DB-960's retargetable monitor can be customized to a target system, allowing source-level, symbolic debug across a serial interface cable.

RETARGETABLE MONITOR

Utilizing a combination of object files and source code, a retargetable monitor is provided with DB-960 for users to customize and incorporate into their proprietary target systems. This retargetable monitor is designed to support all members of the i960 family. Most of the monitor code is provided in object code and does not need to be changed. Hardware-dependent source code is supplied for modification by users. Example code is provided for porting the monitor to the Intel EV80960CA and QT960 target boards. Both boards use an Intel 82510 UART serial controller chip and the Intel 82C54 Counter/Timer.

HARDWARE DEBUG

DB-960 takes advantage of on-chip debug registers like those found on the i960CA to provide two hardware execution address breakpoints and two data address breakpoints. Once the monitor has been retargeted to the target system, hardware designers can download initialization code, read/write to registers and examine memory or register contents.

HIGH SPEED SERIAL LINK

DB-960 communications between the host and target system is supported via RS232 and RS422 communication links. RS232 allows access to industry standard serial protocols while the RS422 interface provides higher speed communication (up to 115K baud) for faster code and data download. PC-AT bus-compatible RS422 communication boards are available from various third party vendors.

FEATURES

CUSTOMIZED ENVIRONMENT

Because the user has control over the target board and serial driver source code, a highly customized target environment can be developed. Serial communication functions can be modified to allow for parallel communication schemes, allowing faster download speeds.

LICENSING

There are no incorporation or royalty fees for customers shipping the retargeted DB-960 monitor with their product or system.

PC-BASED SOFTWARE DEBUGGER

The DB960KBDEVA Software Debugger is designed for debugging i960KA or i960KB code executing on an Intel EVA-960KB4MB Software Execution Vehicle plugged into PC-ATs or compatibles using DOS. DB960KBDEVA offers the same powerful debug user interface as other i960 softerware debuggers and utilizes I/O resources provided by the PC. Due to compatibility with the i960KA and i960KB, i960SA and i960SB code can be executed and debugged using the Intel EVA-960KB4MB Software Execution Vehicle in conjunction with the DB960KBDEVA Software Debugger.

SIMULATOR-BASED SOFTWARE DEBUGGER

The DBSIM960 Debug Simulator combines an i960 CA/KA/SA instruction-level simulator with the easy to use, powerful DB960 software debugger interface. Users can debug i960 applications without a hardware target system being available, allowing products to get to market sooner. For i960 CA designs, performance information is provided, with timing profiles accurate to plus or minus 5%.

Users can specify the target system's clock speed and wait-state information for each region of memory.* DBSIM960 uses this information to provide i960 CA performance statistics. DBSIM960 expects COFF executable files generated by Intel's CTOOLS960 compiler and assembler. Execution flow can be monitored by using a trace capability, which reports the 8 digit cycle address, 8 digit instruction pointer value, and the disassembled instruction for each operation.

Program execution statistics reported include:

- Total number of instructions executed
- Total time
- Number of times a call caused processor to write registers to external memory
- Current clock setting in cycles per second
- Current wait-state setting for each of the 16 memory regions
- Number of instruction words executed from cache rather than external memory
- Total number of cycles elapsed
- Number of stack frames or register sets cached on chip
- Number of times an unaligned load or store operation occurred
- Bus utilization
- Branch prediction efficiency
- Usage for load, store, call and branch cache instructions

Generally, DBSIM960 provides all the full symbolic, debug capabilities found in the i960 family of debug tools, while providing a complete benchmarking environment prior to target system availability.

*By being able to easily change the waitstate definition for their code, the user's hardware and software design can be optimized before any hardware development takes place.

IN-CIRCUIT DEBUG MONITOR

Intel's DB960CADIC in-circuit debug monitor hosted on extended DOS/386 allows users to debug high-speed, cached applications at the full speed of the i960CA target processor. DB960CADIC can be used by both hardware and software developers, at any stage of design. Early in the development process, DB960CADIC allows software debugging when inserted into an existing i960CA board such as the EV80960CA, or in the DB960CASAST stand-alone self-test unit. Later in the design cycle, DB960CADIC can be inserted into the user's target system, facilitating debug of hardware/software integration.

DB960CADIC offers the same, windowed debug user interface as other i960 software debuggers and is also available with an optional 4 MB standalone self test chassis to debug and test code before prototype hardware is available. For further information, see fact sheet #280900 from Intel.

intel

FEATURES

SOFTWARE COMPLETES THE SYSTEM

Intel provides a comprehensive software development environment to complement DB-960. This environment includes a C Compiler, an i960 Assembler, a system generator for automating the compilation process and instruction-level simulators. The languages support the entire range of i960 embedded processors.

WORLDWIDE SERVICE, SUPPORT, AND TRAINING

To augment its development tools, Intel offers a full array of seminars, classes, workshops, field application engineering expertise, hotline technical support, and on-site service.

Intel also offers a Software Support Contract which includes technical software information, automatic distributions of software and documentation updates, *iCOMMENTS* publication, remote diagnostic software, and a development tools troubleshooting guide.

Intel's 90-day Hardware Support package includes technical hardware information, telephone support, warranty on parts, labor, material, and on-site hardware support.

Intel Development Tools also offers a 30-day, money-back guarantee to customers who are not satisfied after purchasing any Intel development tool.

SPECIFICATIONS AND REQUIREMENTS

HOST SYSTEM REQUIREMENTS

Host system requirements to run Intel's i960 family of software debuggers include the following:

- DOS version 3.3 or later excluding DOS 4.0
- 640K bytes of RAM in conventional memory
- A fixed disk drive with at least 1.25M bytes of free disk space
- One disk drive capable of reading 5.25 inch, 360K byte disks
- RS232 serial port (COM1 or COM2)

Evaluated Systems include:

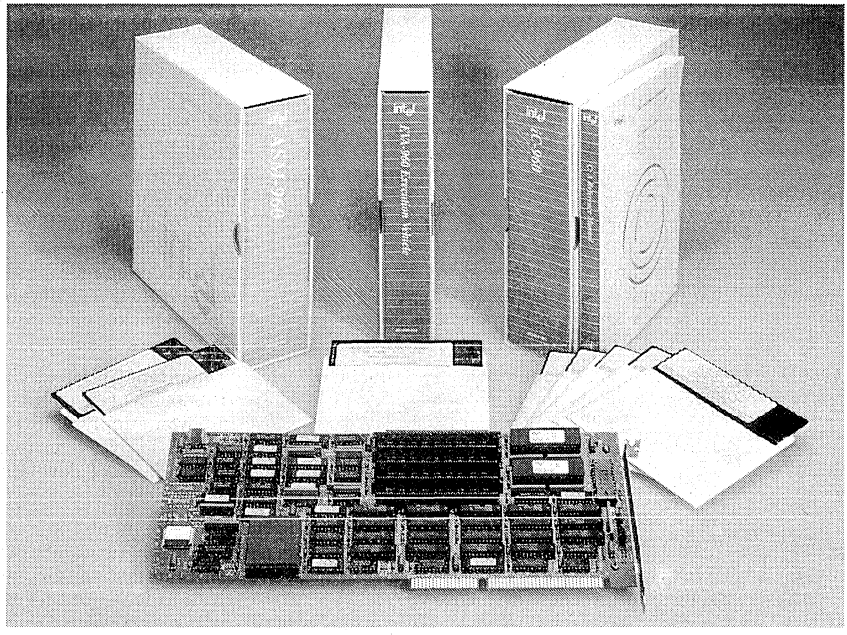
IBM PC-AT* with DOS 3.3
COMPAQ 386* with DOS 3.3
Intel 301/302* with DOS 3.3
IBM Personal System/2* Model 70/80 with DOS 4.01

ORDERING INFORMATION

DB960KBDEV	DOS-based, retargetable software debugger for the 960KA, i960KB, i960SA, i960SB and i960CA embedded microprocessors. Includes host debug software, retargetable monitor, host I/O libraries and documentation.
DB960KBDEVA	DOS-based source level debugger for the i960KA, i960KB, i960SA and i960SB embedded microprocessors. Requires EVA-960KB4MB Software Execution Vehicle and PC-AT compatible bus.
DBSIM960D	DOS/386-hosted debug simulator for the i960 CA, i960 KA and i960 SA which utilizes an i960 CA instruction-level simulator allowing code development and debug prior to hardware prototype availability.
DBSIM960S	UNIX System V/386-hosted debug simulator for the i960 CA, i960 KA and i960 SA which utilizes an i960 CA instruction-level simulator allowing code development and debug prior to hardware prototype availability.
DBSIM960R	IBM RS/6000-hosted debug simulator for the i960 CA, i960 KA and i960 SA which utilizes an i960 CA instruction-level simulator allowing code development and debug prior to hardware prototype availability.
DB960CADIC	DOS/386 hosted in-circuit debug monitor for i960CA only. Includes small board with i960CA processor, system debug monitor and serial interface. Plugs into i960CA socket on hardware prototype system.
DB960CASAST	Standalone Self Test Unit for DB960CADIC. Includes built-in power supply, self-test board, 4M byte of usable DRAM for code development and enclosure.

To order your Intel Development Tool product, for more information, or for the number of your nearest sales office or distributor, call 800-874-6835 (North America). For literature on other Intel products call 800-548-4725 (North America). Outside of North America, please contact your local Intel sales office or distributor for more information.

EXV-960MC EXECUTION VEHICLE



280879-1

80960MC-BASED TARGET SYSTEM SUPPORTING EARLY SOFTWARE DEVELOPMENT AND BENCHMARKING

EXV-960MC is a software execution vehicle designed to support 80960MC-based designs. Users can use the EXV-960MC board to execute and debug their application software before a functional hardware prototype is available. The EXV-960MC is also designed with programmable waitstate SRAM to support benchmarking activities. The EXV-960MC is supported by the complete set of Intel C, assembler and Ada code generation tools. Both of the VAX/VMS*-hosted 80960MC software debuggers, the SDM-960MC system debug monitor and the Ada-960MC source-level debugger, can be used for debugging software running on the EXV-960MC.

EXV-960MC includes a Multibus I form factor board and a set of SDM-960MC target monitor EPROMS. The SDM-960MC and the Ada-960MC debugger are preconfigured to support the EXV-960MC execution environment. Designers can select the software debugger best suited to their development needs. The Ada-960MC debugger is a source-level symbolic debugger which provides a productive debugging environment for Ada applications. The SDM-960MC debug monitor offers a complete debugging facility for applications written in C, assembler or Ada.

*VAX/VMS is a trademark of Digital Equipment Corp.

December 1990
Order Number: 280879-002

SDM-960MC RETARGETABLE SYSTEM DEBUG MONITOR

FEATURES

- 25 MHz 80960MC processor
- 256 Kbytes of (0,0,0,0) programmable wait-state SRAM
- 4 Mbytes dual-ported (3,1,1,1) wait-state DRAM
- iSBX™ interface
- Two serial ports, one bi-directional parallel port
- 8254 programmable interval timer
- 8259A programmable interrupt controller

ELECTRICAL CHARACTERISTICS

10 A @ +5V

50mA @ +12V

50mA @ −12V

ENVIRONMENTAL CHARACTERISTICS

Operating temperature: 0° to +60°C (32° to 140°F), 300 LFM

Operating Humidity: 10% to 90% non-condensing

SOFTWARE DEBUGGING SUPPORT

The SDM-960MC is a VAX/VMS*-hosted system debug monitor that provides a complete, flexible environment to execute and debug 80960MC-based applications. Users can tailor the execution environment as software development evolves. Initially, the application may require the full support of the system debug monitor to establish a run-time environment. As the application evolves, the SDM-960MC allows the application to take more of the responsibility for system functions.

The default execution environment of the SDM-960MC is the EXV-960MC execution vehicle. The VAX-hosted portion of the SDM-960MC debug monitor provides complete on-target debugging support through its interface with the target-resident portion of the SDM-960MC. To facilitate debugging on a user's custom target system, the SDM-960MC includes source and object files necessary to reconfigure the target monitor. SDM-960MC and other 80960MC development tools allow the developers to take full advantage of the 80960MC processor.

FEATURES

- assemble and disassemble 80960MC instructions
- single step program execution
- access to memory and processor resources
- support 64 execution breakpoints
- issue Interagent Communications (IACs)
- powerful execution trace
- serial download

HARDWARE REQUIREMENTS

- a serial interface
- 25 Kbytes of EPROM
- contiguous 50 Kbytes of RAM

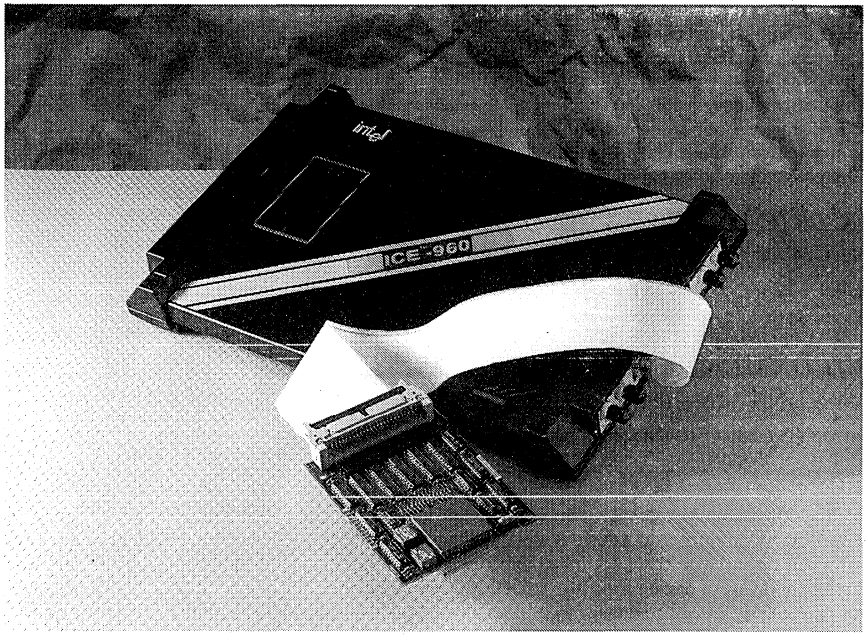
WORLDWIDE SERVICE AND SUPPORT

Intel augments its 80960 architecture family development tools with a full array of seminars, classes, and workshops; on-site consulting services and telephone support are available at all stages of development.

ORDERING INFORMATION

Product Code	Description
EXV960MC	80960MC execution vehicle (board and target EPROM)
SDM960MC	VAX, MicroVAX/VMS hosted System Debug Monitor, retargetable source is included

80960SA/SB DEVELOPMENT SUPPORT



280906-1

COMPREHENSIVE DEVELOPMENT SUPPORT FOR 80960SA/SB EMBEDDED APPLICATIONS

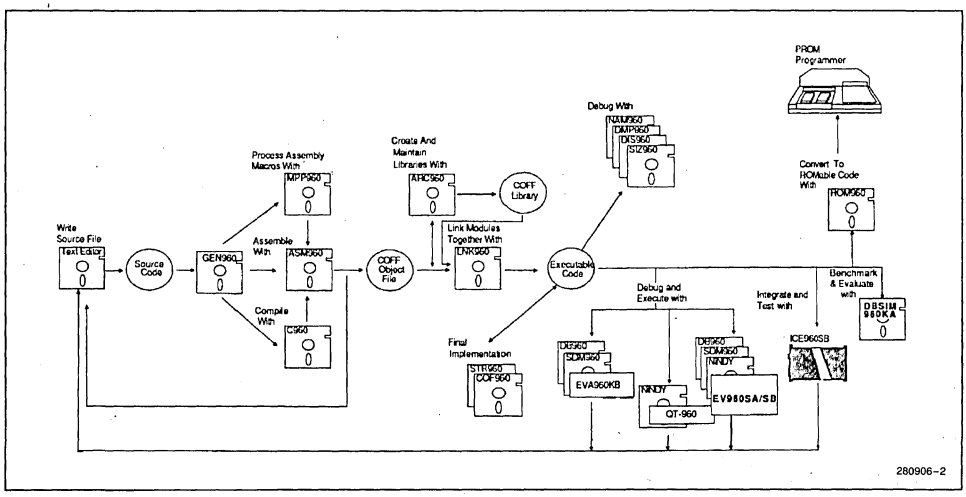
Intel provides comprehensive development support for the 80960 component architecture, including the newest members, the 80960SA and 80960SB. Tools range from compilers to simulators and from debuggers to emulators. All designed specifically for members of the 80960 family, allowing you to take full advantage of their RISC-based design while reducing time to market.

DEVELOPMENT TOOLS AVAILABLE:

- ASM-960 macro assembler for developing and tuning speed-critical code
- iC-960 highly optimizing C language compiler for high-level language software development
- GEN-960 system generator for initializing your design to take advantage of 80960 on-chip features
- DB/SIM960KA debug simulator for 80960KA and 80960SA applications
- Windowed, interactive, source-level DB-960 debugger which can be targeted to one of the evaluation and development boards below, or customized to your target system
- Evaluation and development boards including the EV960SB, the QT80960KB, and the EVA960KB
- ICE-960SA/SB offers a full featured in-circuit emulator for the 80960SA/SB components

November 1990
Order Number: 280906-001

80960SA/SB DEVELOPMENT SUPPORT



ASM-960 MACRO ASSEMBLER

The ASM-960 macro assembler is used to fine-tune sections of code for peak program execution speed on the 80960SA, 80960SB, 80960KA, 80960KB, 80960MC, and 80960CA. ASM-960 does this by giving you absolute control over program instructions. In addition to the assembler and macro preprocessor, ASM-960 includes several utilities for application program maintenance and debug:

- LINKER provides incremental program linking/locating and link-time optimization.
- ARCHIVER allows you to build reusable function libraries for applications.
- DISASSEMBLER produces assembly language from object files.
- SYMBOL DUMPER provides symbolic information from a program file for facilitating low-level debug.
- ROM IMAGE BUILDER produces a hex file suitable for PROM programmers.
- Macro preprocessor provides code generation flexibility and improves code readability, reducing maintenance costs.

A Floating Point Arithmetic Library (FPAL) is included for the 80960SA, 80960KA, and 80960CA components. It eliminates the need to develop your own floating point code.

GEN-960 SYSTEM GENERATOR

The 80960 System Generator (GEN-960) helps you set up data structures for standalone, embedded applications that use the on-chip features of the 80960 architecture. GEN-960 is used with other 80960 tools to generate and refine ROM or RAM code. GEN-960 supplies a set of command and template files containing assembly code and linker control commands to set up processor control blocks, inter-agent communication mechanisms, system procedure tables, and other requirements for initialization. The result is a batch file containing all the commands needed to compile, assemble and link the final target system.

- Improves engineering productivity by automating the compilation, assembly and linking process
- Supplies sample initialization code, reducing programming time
- Save engineering time by simplifying the task of initializing each processor for on-chip capabilities

5

80960SA/SB DEVELOPMENT SUPPORT

iC-960 COMPILER

iC-960 is a highly optimizing C language compiler for the 80960 family of microprocessors. iC-960 supports the full C language as described in the Kernighan and Ritchie book, The C Programming Language (Prentice-Hall, 1978). iC-960 includes standard ANSI extensions to the C language and is used in conjunction with ASM-960 for creating object files.

The iC-960 compiler supports a number of processor dependent optimizations including global register allocation, constant propagation, arithmetic identity folding, redundant load/store elimination, strength reduction and register allocation/scheduling of arguments. Processor independent optimizations include common sub-expression elimination, folding of constant expressions, elimination of superfluous branches, removing unreachable code, tail recursion and procedure incorporation.

iC-960 includes a standard C library with I/O functions and mathematical routines. A second library provides low level, environment-dependent routines emulating UNIX* system calls and supplies I/O routines for the EVA-960 Software Execution Vehicle.

iC-960 also includes the following enhancements for embedded application development:

Programs may be easily placed in ROM.
Memory-mapped I/O allows high-level language access to application-specific input and output.
In-line assembly simplifies the integration of C language and assembly code for speed-critical functions.

Floating point support produces in-line code to take full advantage of the floating point capability of the 80960SB, 80960KB and 80960MC.

Symbolic debugging of source code for iC-960 and ASM-960 is provided by the DB-960 Source Level Debugger, the DBSIM960KA debugging simulator, the DB960CADIC in-target debugger, and the ICE960SB and ICE960KB emulators.

DEBUGGING SIMULATOR

The DBSIM960KA simulator features an easy to use, pulldown menu user interface combined with an 80960SA/80960KA instruction simulator. DBSIM960KA facilitates debugging 80960SA and 80960KA applications by providing debugging capabilities before target hardware is available. DBSIM960KA's powerful, windowed, source-oriented interface allows you to focus your efforts on finding bugs rather than on learning and manipulating the debug environment.

Ease of learning. Drop-down menus make the debugger easy to learn for new or casual users. A command line interface allows direct command entry for solving more complex problems, improving productivity of knowledgeable users.

Extensive debug modes. You can set conditional breakpoints, pass points, and temporary breakpoints as needed.

See into your program. Using pull-down menus or function keys, you can browse source and Call stacks, monitor processor registers, view screen output, and watch the values of variables change.

Full debug symbolics for maximum productivity. You need not know whether a variable is an unsigned integer, a real, or a structure: the debugger displays program variables in their respective type formats.

80960SA/SB DEVELOPMENT SUPPORT

EVA-960KB4MB SOFTWARE EXECUTION VEHICLE

The EVA-960KB4MB is a software execution vehicle for the 80960KA/KB microprocessor. It is a single PC AT plug-in board which provides easy and convenient architecture evaluation and benchmarking, as well as software development. Since the board uses an 80960KB, 80960SA and 80960SB performance can be extrapolated. The EVA-960KB4MB contains the following:

- 4 MB or 16 MB (EVA960KB16MB) of one wait-state program memory (DRAM)
- 64 Kbytes of zero wait-state program memory (SRAM)
- Three-channel programmable interval timer
- Hosted debug monitor which supports two hardware and 64 software breakpoints, single-step program execution, register and memory access, program download and upload
- DOS access libraries that allow: screen display, keyboard input, read and write disk files, and the ability to spawn a DOS process that could communicate with serial or parallel I/O
- 20 MHz operation, allowing software to operate at full speed of 80960KB

EVA-960KB4MB also operates with the DB-960 Source Level Debugger for code development/debug prior to target system availability.

SOURCE-LEVEL DEBUGGER

The DB-960 Debugger with source-level debug capabilities is available for PC ATs equipped with DOS. DB-960 can debug 80960 code executing on an Intel EVA-960 Software Execution Vehicle or on a hardware target system via a serial interface. The EVA-960 targeted debugger uses I/O resources provided by the PC, while 80960 code executes at high speed on the EVA-960. Two serial versions of DB-960 are available. DB-960CADIC plugs directly into the 80960CA socket on your prototype, offering a "plug-in and go" debug environment. DB-960D is a serial, retargetable version of DB-960 whose system debug monitor can be customized for 80960SA/SB, 80960KA/KB, or 80960CA operation.

Ease of learning. Drop-down menus make the debugger easy to learn for new or casual users. A command line interface allows direct command entry for solving more complex problems, improving productivity of knowledgeable users.

Extensive debug modes. You can set conditional breakpoints, pass points, and temporary breakpoints as needed.

See into your program. Using pull-down menus or function keys, you can browse source and Call stacks, monitor processor registers, view screen output, and watch the values of variables change.

Full debug symbolics for maximum productivity. You need not know whether a variable is an unsigned integer, a real, or a structure: the debugger displays program variables in their respective type formats.

In-Target Debug. Porting the DB960D retargetable monitor to your target system allows the debugger to be used in-target, thus facilitating debugging of code dependent upon hardware interaction.

80960SA/SB DEVELOPMENT SUPPORT

ICE960SB IN-CIRCUIT EMULATOR

ICE960SB is a full featured in-circuit emulator for the 80960SA and 80960SB components. A separate ICE probe can be purchased to support 80960KA and 80960KB components. ICE960SB includes:

Full speed emulation of the 80960SA/SB components to 16 MHz

- Complete symbolic information when used with Intel 80960 compilers
- 1024 Frames Bus or Execution Trace with Time-Tags
- Comprehensive break capabilities including execution addresses, instruction type, bus read/write/access, data values, and external synch lines
- Qualification of break conditions based on a 8-state machine or an occurrence counter
- Fastbreaks to dynamically access memory or variables during emulation
- Examine and modify memory and 80960 registers
- Stand-Alone Self-Test module provides diagnostic circuitry and 256 Kbytes of memory for software development
- Optional 2 Mbyte of relocatable expansion memory
- Support for socketed and surface mounted 84 Pin PLCC components and surface mounted 80 Pin EIAJ components via ONCE mode
- DOS Hosting with support for RS232 and RS422 communication links

WORLDWIDE SERVICE, SUPPORT, AND TRAINING

To augment its development tools, Intel offers a full array of seminars, classes, and workshops, field application engineering expertise, hotline technical support, and on-site service.

Intel also offers a Software Support package which includes technical software information, telephone support, automatic distribution of software and documentation updates, access to the "ToolTalk" electronic bulletin board, "*iComments*" publication, remote diagnostic software, and a development tools troubleshooting guide.

Intel's Hardware Support package includes technical hardware information, telephone support, warranty on parts, labor, material, and on-site hardware support.

80960SA/SB DEVELOPMENT SUPPORT

80960SA/SB DEVELOPMENT TOOLS

ASM960	Assembler package containing the assembler, linker/loader, macro preprocessor, archiver, ROM image builder, other object file utilities, and the 80960SA/KA/CA floating point arithmetic library.
C960	Optimizing C Compiler, with ANSI extensions for embedded control applications; contains standard STDIO libraries and in-line assembly capability.
GEN960	80960 System Generation software automates the compilation, assembly and linking process. Simplifies usage of 80960 sophisticated features.
DBSIM960KA	Debugging Simulator software emulates the 80960SA and 80960KA instruction set allowing code development and debugging prior to hardware prototype availability.
DB960KBDEVA	Source Level Debugger software for the 80960KB/KA with powerful debug capabilities including conditional breakpoints, source and Call stack browsing, memory/register display and modification, and ability to watch variables change value. Requires EVA-960KB4MB Software Execution Vehicle. For PC AT hosted systems only.
DB960D	Source Level Debugger software for 80960SA/SB, 80960KA/KB, or CA processors resident on serially-interfaced hardware prototype systems. Includes customizable system debug monitor and serial interface protocol specifications. For PC AT hosted systems only.
EVA960KB4MB	Software Execution Vehicle for 80960SA/SB and 80960KA/KB components. Includes 4 Mbyte of on-board memory, system debug monitor and code download software. Code compatible with the 80960SA/SB components. Required by DB960KBDEVA.
EVA960KB16MB	Identical to EVA960KB4MB with 16 Mbyte of DRAM instead of 4 Mbyte.
ICE960SB	In-Circuit emulator for the 80960SA/SB components. Includes ICE base and probe, stand-alone self-test module, and your choice of PLCC or PQFP target adapters. Optional 2 Mbyte relocatable expansion memory option provides overlayable memory for software prototyping and hardware debugging.

5

80960SA/SB DEVELOPMENT SUPPORT

ARCHITECTURE EVALUATION STARTER KITS

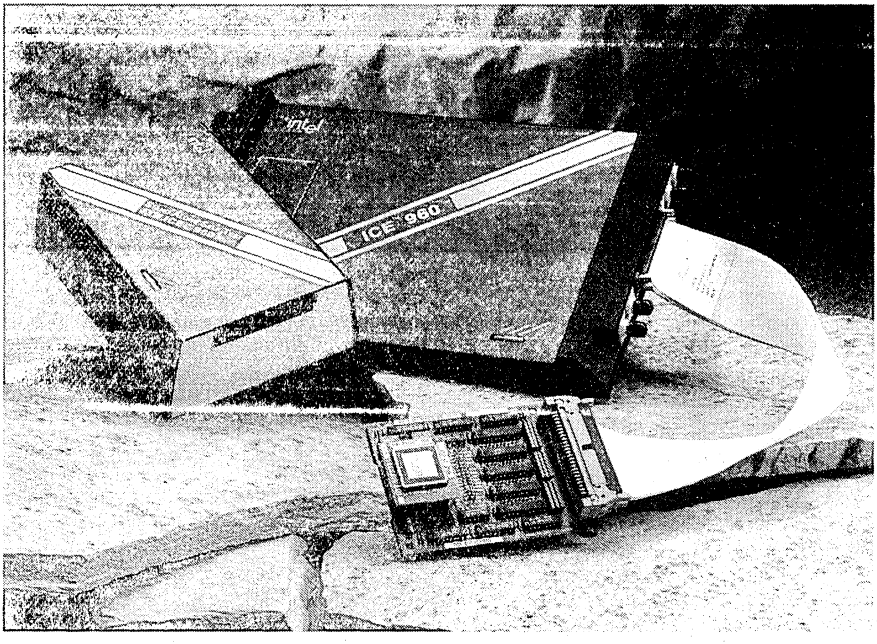
960SKit3 Contains ASM960D Assembler and iC960D Compiler

DB960KIT2 Kit contains DB-960KBDEVA (KB version of DB-960 used with EVA-960), EVA960KB4MB Software Execution Vehicle, ASM960D and C960E. Requires PC AT with 640K memory.

DB960KIT3 Kit contains DB-960D (serial version of DB-960 supporting the 80960SA/SB, 80960KA/KB and 80960CA components (operating on PC-AT/DOS), ASM960D and C960D. Requires PC AT with 640K memory.

Product Code to order, by Host							
Product Category	**PC-AT/DOS**	**UNIX-386 V.4**	**OS/2**	**Sun 3/ UNIX**	**HP9000/ HP-UX**	**VAX/ ULTRIX**	**μVAX/ ULTRIX**
Assembler	ASM960D	ASM960S	ASM960P	ASM960U	ASM960H	ASM960VX	ASM960MX
C Compiler	C960D	C960S	C960P	C960U	CP60H	C960VX	C960VX
System Gen	GEN960D	—	—	GEN960U	GEN960H	—	—
SX Debugger	DB960D	—	—	—	—	—	—
KX Debugger	DB960KBDEVA	—	—	—	—	—	—
	DB960D	—	—	—	—	—	—
CA Debugger	DB960CADIC	—	—	—	—	—	—
	DB960D	—	—	—	—	—	—
SA Simulator	—	DBSIM960KAS	—	—	—	—	—
CA Simulator	SIM960CAD	—	—	SIM960CAU	SIM960CAH	—	—
ICE960SB	ICE960SB	—	—	—	—	—	—
ICE960KB	ICE960KB	—	—	—	—	—	—

ICE™-960SB AND ICE-960KB IN-CIRCUIT EMULATOR



280852-1

5

INTERCHANGEABLE PROBES

The ICE™-960 in-circuit emulator delivers real-time hardware and software debugging capabilities for i960™ SA/SB and i960 KA/KB-based designs. Features include full-speed emulation of each of the microprocessors, powerful breakpoint specification, fastbreaks, optional relocatable expansion memory, two types of trace capability, large trace buffering, sophisticated human interface and high-speed communication links with the DOS host. The ICE-960 in-circuit emulator gives you unmatched control over all phases of hardware/software debug, including developing, integrating and testing, which improves development productivity and improves time to market.

FEATURES

- Real-Time Emulation of the i960 KA/KB microprocessors up to 25 MHz and emulation of the i960 SA/SB to 16 MHz
- Full symbolic integration with Intel ASM and C compilers
- Optional ICE960KBREM/ ICE960SBREM boards provide 2 Mbytes of ICE memory which can overlay user ROM or RAM.
- Examine and modify memory and the i960 registers
- Dynamically monitor and update program variables via fastbreaks
- Breakpoint capabilities include: execution address, instruction type, bus read/write/access, and data value. Qualification of events is based on an occurrence counter and an 8-state states-machine

October 1991
Order Number: 280852-003

FEATURES

- Hosted on IBM PC AT* or compatible and supporting RS232, RS422 and Ethernet operation
- 1024 frame trace buffer for execution and/or bus trace and time tags
- The on-chip cache does not effect collection of the execution trace
- 256 Kbytes of memory in standalone self-test (SAST) unit
- Real-time bus trace with time-tags for tracking code execution time
- Assembly and disassembly of code in i960 instruction mnemonics
- ICE to component interconnect includes support for surface-mounted and socketed 84-pin PLCCD and surface mounted 80-pin EIAJ QFP i960 SA/SB and 132-pin PGA for i960 KA/KB

The ICE-960 in-circuit emulator provides emulation of the i960 SA/SB at speeds to 16 MHz and the i960 KA/KB at speeds to 25 MHz, thus providing early detection of subtle timing problems that may arise at full speed. Intel's intimate knowledge of the component makes possible the tightest conceivable conformance between timing parameters of the emulator and the target microprocessor.

PROCESSOR/MEMORY EXAMINATION AND MODIFICATION

The i960 registers can be accessed mnemonically (e.g. g12, r5, fp3) with the ICE-960 emulator software. Data can be displayed or modified in hexadecimal, decimal, octal, or binary and by data type (byte, word, etc). Program memory contents can be modified as i960 assembly instruction mnemonics.

PROGRAM TRACING

The ICE-960 emulator can store 1024 frames of program execution history processor/address/data bus activity in the trace buffer. Each frame of program execution contains a discontinuity address (branch, call, return, etc) and a time-tag. This information can be used to reconstruct a history of the program execution. With the execution trace option enabled, the ICE-960 will run at less than full speed. Each trace frame of bus cycles contains one complete bus burst trace. Collection of trace information is controlled by a logic analyzer type moving trace window and by bus access type.

EVENT RECOGNITION (BREAKPOINT CONTROL) AND EMULATION CONTROL

ICE-960 provides comprehensive event recognition capabilities including: two hardware and thirty-two software breakpoints for instruction execution breakpoints, and use of the internal debug registers to recognize execution of certain instruction types such as branch or call instructions. Bus analysis logic provides recognition of external bus addresses qualified by read, write, or access type as well as data values. The data values may be entered as masked values and qualified by type. Two synchronization lines are provided for recognition of external events. ICE-960 also provides qualification of events based on an occurrence counter or by a recognition sequence of up to 8 events. Additionally, emulation can be automatically stopped when the trace buffer is full. Besides the ability to execute program code at full speed between specified points, the ICE-960 emulator provides the capability to single-step through program code.

RELOCATABLE EXPANSION MEMORY

An optional board provides ICE-960 with 2 Mbytes of relocatable expansion memory which allows users to develop applications either before the target system memory is working, or in place of ROM or EPROM to speed the debugging cycle. This memory can be mapped in two separate 1 Mbyte partitions on 1 Mbyte boundaries.

For the new ICE960KBREM board, the memory waitstate pattern is (3,1,1,1) when the users system does not return RDY# for accesses in the mapped area. For accesses where the user system does return RDY# for these areas, the waitstate pattern will be the larger of (3,1,1,1) or user waitstate pattern plus (2,2,2,2). For either board, the size and shape of the board is identical to the ICE probe and is installed between the probe and the user's target system when in use. The memory configuration can be mapped via an ICE MAP command.

The ICE960KBREM/ICE960SBREM cards add some constraints when used with the ICE in a users target system. First, users should qualify bus drivers/buffers with DEN# in order to eliminate potential bus conflict between the REM board and their target memory while

using the ICE. Second, the 1M Byte partition size can not be reduced and may effect the design of the users memory subsystem. Third, the REM boards delay the ADS# and DEN# signals by 5 ns (typical) and delays the RDY# signal by 4 ns (typical). Fourth, it adds loading, capacitance, and power requirements as shown in tables 3 and 4.

STANDALONE OPERATION

Product software can be developed and debugged prior to and independent of hardware availability with the Standalone Self Test unit (SAST), which contains 256 Kbytes of two wait-state program memory. The SAST also provides diagnostic testing to assure full functionality of the ICE-960 emulator.

VERSATILE AND POWERFUL HOST SOFTWARE

ICE-960 provides an easy-to-use human interface which utilizes color forms to complement a powerful command set. The software includes: an on-line help facility, a dynamic command entry and syntax guide, screen oriented editor, assembler and disassembler, input/output redirection, command piping, DOS command entry, and the ability to customize the command set via debug procedures and literal definitions.

DEBUG PROCEDURES AND LITERALS

Debug procedures (PROCs) are user-defined groups of ICE960 emulator commands. They can be stored on disk and recalled during later debugging sessions. PROCs can be used to simplify the process of debugging by grouping repetitive emulator commands, which can then be accessed by typing the name of the PROC. Literals are user-defined abbreviations for whole or partial ICE-960 emulator commands. Literals are a shorthand method of customizing the emulator commands to fit your needs and preferences.

ICE TO COMPONENT INTERCONNECT SYSTEM

Using the On-Circuit Emulation (ONCE) i960 SA/SB silicon feature, ICE960SB can be used in systems with surface-mounted i960 SA/SB components in either PLCC or EIAJ QFP packages. The hinge cable adapters included in the various ICE kits and pictured to the right, are placed directly on top of the surface mounted i960 SA/SB device. The circuitry necessary for the emulator to take control from the target processor is fully supported in the emulator. No additional circuitry is required.

Of course, socketed support for i960 SA/SB components in PLCC packages, or i960 KA/KB components in PGA packages are also supported. Please see Figures 1, 2, 3, and 4 for ICE Probe physical characteristics. Refer to Table 5 for hinge cable loading and delay characteristics.

WORLDWIDE SERVICE, SUPPORT, AND TRAINING

To augment its development tools, Intel offers a full array of seminars, classes, workshops, field application engineering expertise, hotline technical support, and on-site service.

Intel also offers a Software Support contract which includes technical software information, automatic distributions of software and documentation updates, *iCOMMENTS* publication, remote diagnostic software, and a development tools troubleshooting guide.

intel

FEATURES

HIGH-SPEED HOST-TO-ICE COMMUNICATIONS PROTOCOLS

ICE-960 supports RS232 and RS422 communications protocols to 115 KBaud and 1152 KBaud respectively depending upon the ability of the host to support the specific rate. Testing for these systems and the configurations involved are described in the following sections.

Intel's 90-day Hardware Support package includes technical hardware information, warranty on parts, labor, material, and on-site hardware support.

Intel Development Tools also offers a 30-day, money-back guarantee to customers who are not satisfied after purchasing any Intel development tool.

SPECIFICATIONS

HOST REQUIREMENTS

IBM PC-AT (minimum requirements) with 640 KBytes of conventional memory

1 MByte of RAM (Lotus, Intel, Microsoft expanded memory specification)

20 MByte Fixed Disk

At least one 5¼" or 3½" Floppy Disk drive

RS232 or RS422 Communication Interface

DOS Operating System (version 3.2 or 3.3)

TESTED HOST CONFIGURATIONS

IBM PC-AT with DOS 3.3. Tested with built-in RS232 and a Quatech DS202 Asynchronous RS422 Communications Board with 16550 Option

COMPAQ Deskpro 386* with DOS 3.3. Tested with built-in RS232 and Quatech DS202 Asynchronous RS422 Communications Board with 16550 Option

Systems Based on an Intel 301/302™ Box with DOS 3.3. Tested with built-in RS232 to 115.2 KBaud and a Quatech DS202 Asynchronous RS422 Communications Board with 16550 Option to 1.152 MBaud

IBM Personal System/2* with DOS 4.01. Tested with built-in RS232

REQUIRED SYSTEM RESOURCES

The ICE-960 emulator requires the following: a) exclusive use of the i960 SA/SB or i960 KA/KB's on-chip debug registers and b) a minimum of 256 bytes of target system RAM used to flush the i960 local registers.

MECHANICAL SPECIFICATIONS

TABLE 1. ICE-960 Emulator Physical Characteristics

Unit	Width		Height		Length		Weight	
	Inches	cm	Inches	cm	Inches	cm	lbs	kg
Control Unit	10.5	26.7	1.5	3.8	16.0	40.6	6.0	2.72
Processor Module*	3.8	9.6	1.5	3.8	5.0	12.7		
SAST	6.0	15.2	2.0	5.1	8.0	20.3	3.5	1.59
OIB	3.8	9.6	0.9	2.3	5.1	13.0		
Power Supply	2.8	7.1	4.2	10.7	11.0	27.9	4.7	2.14
User Cable					22.0	55.9		
Serial Cables					12.0'	3.66m		

*Measurement includes target adaptor

SPECIFICATIONS

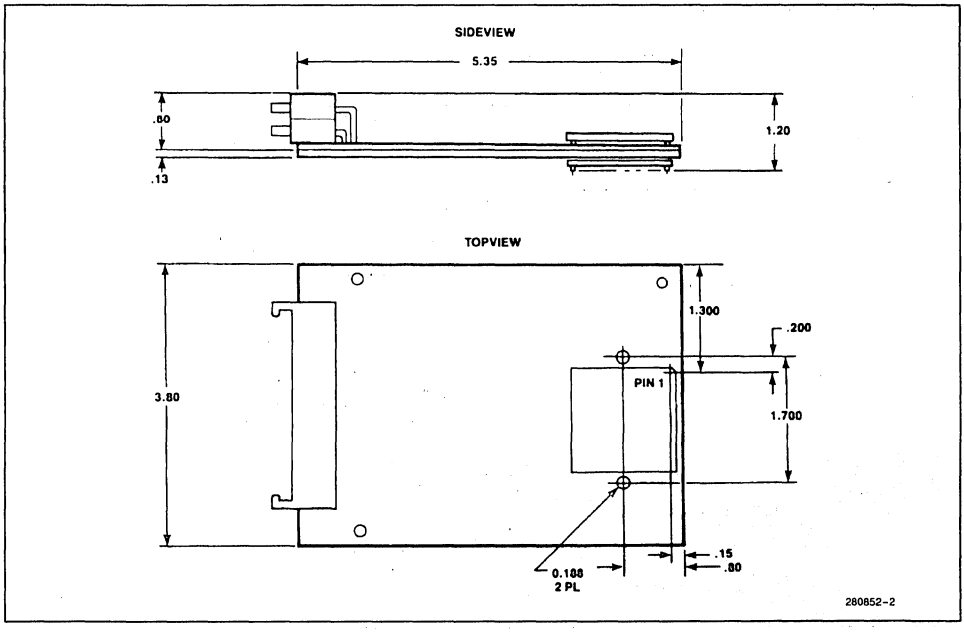


Figure 1: ICE960KB25 Processor Module

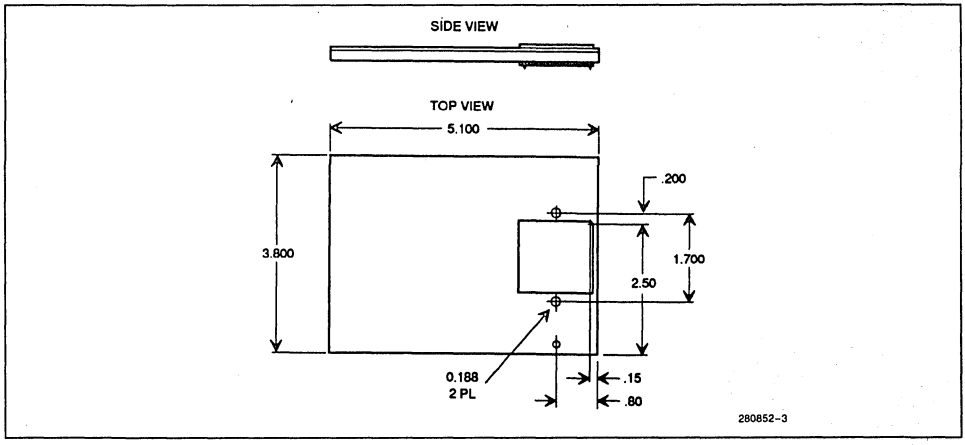


Figure 2: Optional Isolation Board

SPECIFICATIONS

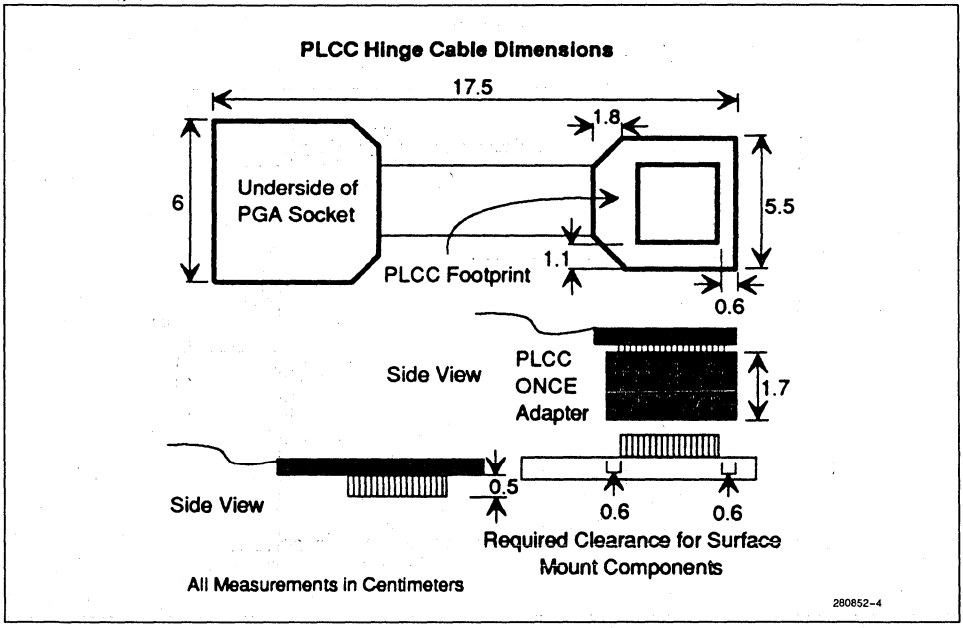


Figure 3: ICE960SB16C Adapter

SPECIFICATIONS

ELECTRICAL SPECIFICATIONS

SYNC Line Specification

The SYNCIN line must be valid for at least one instruction cycle because it is only sampled on bus access boundaries. The SYNCIN line is a standard TTL input. The SYNCOUT line is driven by a TTL open collector with a 4.75 KΩ pull-up resistor

AC/DC Specifications

The Optional Isolation Board (OIB) isolates the ICE-960 probe from an untested user target system. When the OIB is in use, the ICE-960 AC and DC specifications differ from the i960 microprocessor as shown below. When the OIB is not installed, the ICE-960KB timing specifications are identical to those of the i960 component.

TABLE 2. AC Specifications with the OIB Installed

Symbol*	Parameter	16 MHz 80960SB		25 MHz 80960KB	
		Min	Max	Min	Max
T1	Clock Period	32 ns	125 ns	20 ns	125 ns
T2	Clock Low Time	9 ns		6 ns	
T3	Clock High Time	9 ns		6 ns	
T4	Clock Fall Time		10 ns		10 ns
T5	Clock Rise		10 ns		10 ns
T6	Output Valid Delay A(2:3), BE#(0:1), BLAST#,* DEN#, DTR#, WR#** A/D Lines***		 40 ns 40 ns		 33 ns 33 ns
T6 AS	AS Valid Delay (AS#)		36 ns		33 ns
T7	ALE# Width	16 ns		12 ns	
T8	ALE# Valid Delay		36 ns		33 ns
T9	Output Float Delay A(2:3), BE#(0:1), BLAST#,* DEN#, DTR#, WR#** A/D Lines		 50 ns 50 ns		 35 ns 40 ns
T10	Input Setup 1 HLDA, INT0#, INT1, INT2, INT3#	 13 ns		 6 ns	
T11	Input Hold HLDA, INT0#, INT1, INT2, INT3# HOLD, READY#, LOCK#	 10 ns 10 ns		 13 ns 13 ns	
T12	Input Setup 2 HOLD, READY#, LOCK#	 17 ns		 11 ns	
T13	Setup to ALE# Inactive	7 ns		7 ns	
T14	Hold after ALE#	5 ns		5 ns	
T15	RESET# Hold	4 ns		4 ns	
T16	RESET# Setup	4 ns		4 ns	
T17	RESET# Width	1281 ns		820 ns	

*T_{PLH} dependent on termination for KB control signals
**OIB does not float A/D bus during T_r and T_i (between bus cycles)
***Output Valid Delay for control signals after HOLD ACKNOWLEDGE is deasserted 50 ns for 80960SB and 43 ns for 80960KB

SPECIFICATIONS

TABLE 3. ICE-960 Emulator DC Specifications

	ICE Probe	OIB	REM	Processor Speed
ICE960SB	1.4	0.4	0.5	16
ICE960KB	1.4	0.6	0.7	25

TARGET SYSTEM DESIGN CONSIDERATIONS

In addition to the mechanical, power consumption, and signal loading considerations for the ICE probe, the following points should be taken into account when the target system is being designed:

1) [SA/SB/KA/KB/MC]

   The AD bus should not be driven by an external source unless DEN# is asserted.

2) [SA/SB/KA/KB/MC]

   The LOCK# signal must be terminated as recommended in the 80960SA/SB component data sheet.

3) [SA/SB/KA/KB/MC]

   To guarantee timings, the ICE requires ±5% supply voltage to the target system (i.e., ICE probe power).

4) [SA/SB]

   To ensure correct bus trace the ICE requires a data hold time (T11) of 4 ns.

5) [SA/SB/KA/KB/MC]

   Each V_{CC} and GND pin of the processor must be connected to the appropriate voltage or ground and externally strapped close to the package.

6) [SA/SB/KA/KB/MC]

   Processor no connect (N.C.) pins must be left disconnected.

SPECIFICATIONS

TABLE 4. Additional DC Loading

Signal	(ICE Probe)		(OIB)		(KB REM)		(SB REM)	
	I_{IH} Max	I_{IL} Max	I_{IH} Max	I_{IL} Max	I_{IH} Max	I_{IL} Max	I_{IH} Max	I_{IL} Max
AD(0:31)	25 μA	25 μA	15 μA	−15 μA	120 μA	0.7 mA	20 μA	100 μA
ADS#	25 μA	25 μA	115 μA	−15 μA	Driven by 74AS760 w/ 4.7k Pull-Up		10 μA	10 μA
DEN#	25 μA	25 μA	115 μA	−15 μA				
W/R#	25 μA	25 μA	115 μA	−15 μA	150 μA	1.7 mA	10 μA	10 μA
CLK2	50 μA	500 μA	25 μA	−25 μA	130 μA	2.9 mA	20 μA	1600 μA
RESET	25 μA	250 μA	45 μA	−750 μA	250 μA	0.3 mA	10 μA	10 μA
BE(0:3)#	25 μA	25 μA	115 μA	−15 μA	10 μA	0.1 mA	10 μA	10 μA
READY#	25 μA	25 μA	45 μA	−750 μA	750 μA	0.8 mA	25 μA	260 μA
ALE#	25 μA	25 μA	15 μA	−15 μA	20 μA	0.5 mA	10 μA	1600 μA
DT/R#	25 μA	25 μA	115 μA	−15 μA				
INT(0:3)	25 μA	25 μA	15 μA	−565 μA				
BADAC#	25 μA	25 μA	15 μA	−565 μA				
LOCK#	25 μA	25 μA	140 μA	−500 μA				
HOLD	25 μA	25 μA	45 μA	−750 μA				
FAILURE#	25 μA	25 μA	20 μA	−1 mA				

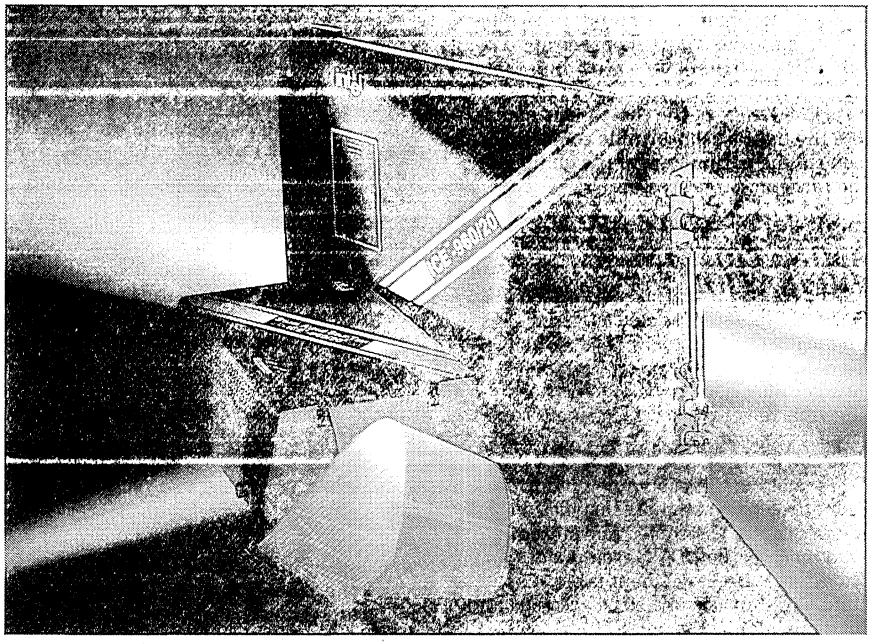
TABLE 5. 80960SB PLCC Hinge Cable Loading and Delay

Signal Loading	15 pF Typical
Signal Delay	Signals from Processor delayed 4 ns typical, Setup and Hold Timings unaffected.

ORDERING INFORMATION

Order Code	Description
ADPT80EIAJ	Hinge Cable Adapter for surface-mount i960SB EIAJ QFP packages. This adapter is included in the ICE960SB16J kit.
ADPT84PLCC	Hinge Cable Adapter for surface-mount and socketed i960SB PLCC packages. This adapter is included in the ICE960SB16C kit.
ICE960SB16C	ICE960 base, i960 SA/SB probe, 84-pin PLCC surface-mount and socketed target component interconnect, and RS232 and RS422 communication cables. (Shrink-Wrap license, Class 1)
ICE960SB16J	ICE960 base, i960 SA/SB probe, 80-pin EIAJ surface-mount target component interconnect, and RS232 and RS422 communication cables. (Shrink-Wrap license, Class 1)
ICE960KB25	ICE960 base, i960 KA/KB probe, 132-pin PGA target component interconnect, and RS232 and RS422 communication cables. (Shrink-Wrap license, Class 1)
ICE960SBREM	Optional 2 MByte Relocatable Expansion Memory Board for i960 SA/SB components.
ICE960KBREM	Optional 2 MByte Relocatable Expansion Memory Board for 80960KA/KB components.
PTOI960SB16	Probe and Software to convert ICE960KB25 to ICE960SB16. An ADPT80EIAJ or ADPT84PLCC adapter kit should also be ordered with this package to support the component packaging type of your choice. (Shrink-Wrap license, Class 1)
PTOI960KB25	Probe and Software to convert ICE960SB16C or ICE960SB16J to ICE960KB25. (Shrink-Wrap license, Class 1)

ICE™-960MC IN-CIRCUIT EMULATOR



280899-1

IN-CIRCUIT EMULATOR FOR THE 80960MC MICROPROCESSOR

The ICE™-960MC In-circuit Emulator delivers real-time hardware and software debugging capabilities for 80960MC based designs. Features include emulation of the 80960MC microprocessor, powerful breakpoint specification, fastbreaks, optional relocatable expansion memory, two types of trace capability, large trace buffering, support of virtual and physical component addressing modes, and sophisticated human interface. The ICE-960MC In-circuit Emulator gives you unmatched control over all phases of hardware/software debug, including developing, integrating and testing, which improves development productivity and speeds time to market.

FEATURES

- Real-Time Emulation of the 80960MC microprocessors up to 20 MHz (25 MHz optional)
- Full Symbolic Information Relating to Code. Data symbolics subject to some limitations in virtual addressing mode
- Optional ICE960KBREM Board Provides 2 Mbytes of ICE Memory Which Can Overlay User ROM or RAM.
- Zero wait-state operation from user memory
- Examine and modify Memory and the 80960 Registers
- Breakpoint Capabilities include: Execution Address, Instruction Type, Bus Read/Write/Access, and Data Value. Qualification of Events is Based on an Occurrence Counter and an 8 state State-Machine
- Hosted on IBM PC AT or compatible
- Dynamically monitor or update program variables or memory during emulation with Fastbreaks
- 1024 Frame Trace Buffer for execution and/or Bus Trace and time tags
- 256 Kbytes of Memory in Standalone Self-Test (SAST) Unit

November 1990
Order Number: 280899-001

ICE™-960MC IN-CIRCUIT EMULATOR

REAL-TIME EMULATION

The ICE-960MC In-circuit Emulator provides emulation of the 80960MC at speeds up to 20 MHz (25 MHz optional), thus providing early detection of subtle timing problems. Intel's intimate knowledge of the component makes possible the tightest conceivable conformance between timing parameters of the emulator and the target microprocessor.

PROCESSOR/MEMORY EXAMINATION AND MODIFICATION

The 80960MC registers can be accessed mnemonically (e.g. g12, r5, fp3) with the ICE-960MC emulator software. Data can be displayed or modified in one of four bases (hexadecimal, decimal, octal, or binary) and by data type (byte, word, etc). Program memory contents can be disassembled and displayed as 80960 assembly instruction mnemonics. Additionally, 80960 assembly instruction mnemonics can be assembled and stored into program memory. 80960MC system data structures such as the segment table, dispatch port, and page tables can also be accessed and modified mnemonically.

PROGRAM TRACING

The ICE-960MC emulator can store 1024 frames of program execution history or 1024 frames of the 80960MC address/data bus activity in the trace buffer. Each frame of program execution contains a discontinuity address (branch, call, return, etc) and a time-tag. This information can be used to reconstruct a history of the program execution. With the execution trace option enabled, the ICE-960MC will run at less than full speed. Each trace frame of bus cycles contains one complete bus burst trace. Collection of trace information is controlled by a logic analyzer type moving trace window and by bus access type.

EVENT RECOGNITION (BREAKPOINT CONTROL) AND EMULATION CONTROL

ICE-960MC provides comprehensive event recognition capabilities including: two hardware and thirty-two software breakpoints for instruction execution breakpoints, and use of the internal debug registers to recognize execution of certain instruction types such as branch or call instructions. Bus analysis logic provides recognition of external bus addresses qualified by read, write, or access type as well as data values which may be entered as masked values. Two synchronization lines are provided for recognition of external events. ICE-960MC also provides qualification of events based on an occurrence counter or by a recognition sequence of up to 8 events. Special additions for the 80960MC include the ability to recognize process binds. Additionally, emulation can be automatically stopped when the trace buffer is full. Besides the ability to execute program code at full speed between specified points, the ICE-960MC emulator provides the capability to single-step through program code.

RELOCATABLE EXPANSION MEMORY

An optional board provides ICE-960MC with 2 Mbytes of relocatable expansion memory which allows users to develop applications either before the target system memory is working, or in place of ROM or EPROM to speed the debugging cycle. This memory can be mapped in two separate 1 Mbyte partitions on 1 Mbyte boundaries. The memory waitstate pattern is (3,1,1,1) when the user's system does not return RDY# for accesses directed to the ICE960KBREM board. For accesses where the user system does return RDY# the waitstate pattern will be the larger of (3,1,1,1) or user waitstate pattern plus (2,2,2,2). The size and shape of the board is identical to the ICE probe and is installed between the probe and the user's target system when in use. The memory configuration can be mapped via either an ICE MAP command or via switches on the ICE960KBREM board.

The ICE-960KBREM card adds some constraints when used with the ICE in a user's target system. First, users should qualify bus drivers/buffers with DEN# in order to eliminate potential bus conflict between REM960 and their target memory. Second, the 1 Mbyte partition size can not be reduced and may effect the design of the user's memory subsystem. Third, ICE960KBREM delays the ADS# and DEN# signals by 5 nsec (typical) and delays the RDY# signal by 2 nsec (typical). Fourth, it adds loading, capacitance, and power requirements as shown in tables 3 and 4.

STANDALONE OPERATION

Product software can be developed and debugged prior to and independent of hardware availability with the Standalone Self Test unit (SAST), which contains 256 Kbytes of two wait-state program memory. The SAST also provides diagnostic testing to assure full functionality of the ICE-960MC emulator.

VERSATILE AND POWERFUL HOST SOFTWARE

ICE-960MC provides an easy-to-use human interface which utilizes color and pull-down menus to complement a powerful command set. The software includes: an on-line help facility, a dynamic command entry and syntax guide, screen oriented editor, assembler and disassembler, input/output redirection, command piping, DOS command entry, and the ability to customize the command set via debug procedures and literal definitions.

Special software commands are provided to display, interpret, and modify the 80960MC hardware data structures including the segment table, dispatch port, process control block, and the page tables and directories.

DEBUG PROCEDURES AND LITERALS

Debug procedures (PROCs) are user-defined groups of ICE-960MC emulator commands. They can be stored on disk and recalled during later debugging sessions. PROCs can be used to simplify the process of debugging by grouping repetitive emulator commands, which can then be accessed by typing the name of the PROC. Literals are user-defined abbreviations for whole or partial ICE-960MC emulator commands. Literals are a shorthand method of customizing the emulator commands to fit your needs and preferences.

ICE™-960MC IN-CIRCUIT EMULATOR

WORLDWIDE SERVICE, SUPPORT, AND TRAINING

To augment its development tools, Intel offers a full array of seminars, classes, and workshops, field application engineering expertise, hotline technical support, and on-site service.

Intel also offers a Software Support package which includes technical software information, telephone support, automatic distribution of software and documentation updates, access to the "ToolTalk" electronic bulletin board, "*iComments*" publication, remote diagnostic software, and a development tools troubleshooting guide.

Intel's Hardware Support package includes technical hardware information, telephone support, warranty on parts, labor, material, and on-site hardware support.

SPECIFICATIONS

HOST REQUIREMENTS

IBM PC AT (minimum requirements) with 640 KB of conventional memory

- 1 MB of RAM (Lotus, Intel, Microsoft expanded memory specification)
- 20 MB Fixed Disk
- At least one 5-¼" Floppy Disk drive
- A serial interface
- DOS Operating system (version 3.2 or later excluding 4.x)

REQUIRED SYSTEM RESOURCES

The ICE-960MC emulator requires the following: a) exclusive use of the 80960MC's on-chip debug registers and b) a minimum of 256 bytes of target system RAM used to flush the 80960 local registers.

Mechanical Specifications

TABLE 1. ICE-960MC Emulator Physical Characteristics

	Width		Height		Length		Weight	
Unit	**Inches**	**cm**	**Inches**	**cm**	**Inches**	**cm**	**lbs**	**kg**
Control unit	10.5	26.7	1.5	3.8	16.0	40.6	6.0	2.72
Processor module*	3.8	9.6	1.5	3.8	5.0	12.7		
SAST	6.0	15.2	2.0	5.1	8.0	20.3	3.5	1.59
OIB	3.8	9.6	.9	2.3	5.1	13.0		
Power supply	2.8	7.1	4.2	10.7	11.0	27.9	4.7	2.14
User cable					22.0	55.9		
Serial cable					12.0 ft	3.66m		

*measurement includes target adaptor

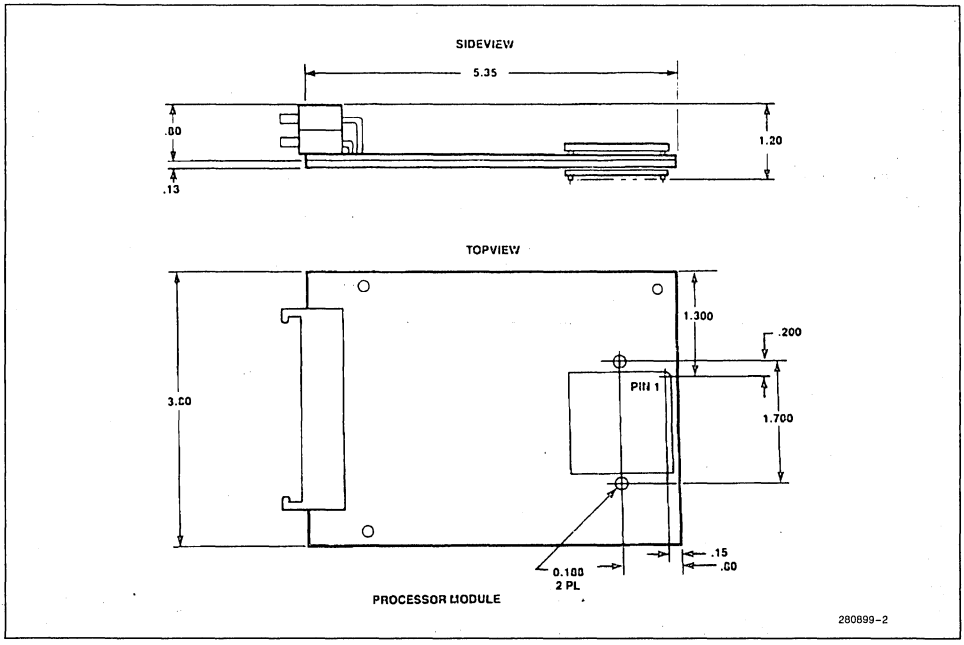


Figure 1: Processor Module

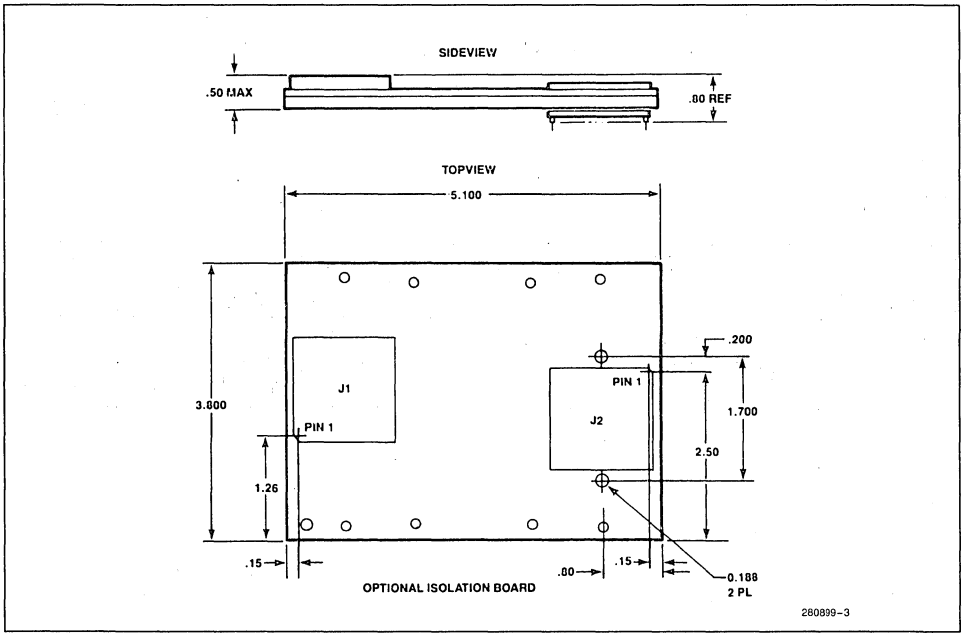


Figure 2: Optional Isolation

SPECIFICATIONS

ELECTRICAL SPECIFICATIONS

SYNC Line Specification

The SYNCIN line must be valid for at least one instruction cycle because it is only sampled on instruction boundaries. The SYNCIN line is a standard TTL input. The SYNCOUT line is driven by a TTL open collector with a 4.75K-ohm pull-up resistor.

AC/DC Specifications

The Optional Isolation Board (OIB) isolates the ICE-960MC probe from an untested user target system. When the OIB is in use, the ICE-960MC AC and DC specifications differ from the 80960MC microprocessor as shown below. When the OIB is not installed, the ICE-960MC specifications are identical to those of the 80960MC component.

TABLE 2. AC Specifications With The OIB Installed

Symbol*	Parameter	Minimum	Maximum
t2	clock low time	2+1nS	
t3	clock high time	3+1ns	
t6	output valid delay		
	A/D 0:31	6+8ns	t6+16Ns
	DT/R#, DEN#, BE0-3#, ADS#, W/R#	6+7nS	t6+14ns
	HLDA, CACHE, LOCK#, INTA#	6+6ns	t6+8nS
	ALE#	6+10nS	t6+20nS
t7	ALE# width	7−6.5nS	
t8	ALE# disable delay	8+nS	t8+14nS
t9	output float delay		
	A/D 0:31	t9+5nS	t9+22nS
	DT/R#, DEN#, BE0-3#, ADS#, W/R#	t9+7nS	t9+15ns
	HLDA, CACHE, LOCK#, INTA#	t9+6nS	t9+8nS
t10	input setup 1		
	A/D 0:31	t10+2nS	
	BADAC#, INT0-3# deassertion	t10+14nS	
t11	input hold		
	A/D 0:31, HOLD	t11+6nS	
	BADAC#, INT0-3#,		
	READY#	t11+7nS	
t16	reset setup time	16+6	

*symbol refers to 80960MC specification

TABLE 3. ICE-960MC Emulator DC Specifications

Symbol*	Parameter	Maximum
PM-Icc	Supply current with 80960KB-20	1400mA
OIB-Icc	Supply current	PM-Icc + 1100mA
REM-Icc	Supply current	PM-Icc + 1300mA (1700 Total Typical)

SPECIFICATIONS

TABLE 4. Additional DC Loading

Signal	(without OIB installed) Iih Maximum	(without OIB installed) Iil Maximum	(with OIB installed) Iih Maximum	(with OIB installed) Iil Maximum	(with REM installed) Iih Maximum	(with REM installed) Iil Maximum
AD (0:31)	100 uA	0.6 mA	20 uA	−1 mA	120 uA	0.7 mA
ADS#	140 uA	1.6 mA	20 uA	−1 mA	Driven by 74AS760	
DEN#	40 uA	1.0 mA	20 uA	−1 mA	w/ 4.7k pull-up	
W/R#	140 uA	1.6 mA	20 uA	−1 mA	150 uA	1.7 mA
CLK2	80 uA	2.2 mA	50 uA	−2 mA	130 uA	2.9 mA
RESET			50 uA	−2 mA	250 uA	0.3 mA
BE (0:3)#			20 uA	−1 mA	10 uA	0.1 mA
READY#			20 uA	−1 mA	750 uA	0.8 mA
ALE#			20 uA	−1 mA	20 uA	0.5 mA
DT/R#			20 uA	−1 mA		
INTO#, INT3#			20 uA	−1 mA		
INT1, INT2			20 uA	−1 mA		
BADAC#			20 uA	−1 mA		
LOCK#			20 uA	−1 mA		
HOLD			20 uA	−1 mA		
FAILURE#			20 uA	−1 mA		

intel

SPECIFICATIONS

POWER SUPPLY

100–120V or 220–240V (Selectable)
50–60 Hz
2 amps (AC Max) @ 120V
1 amp (AC Max) @ 240V

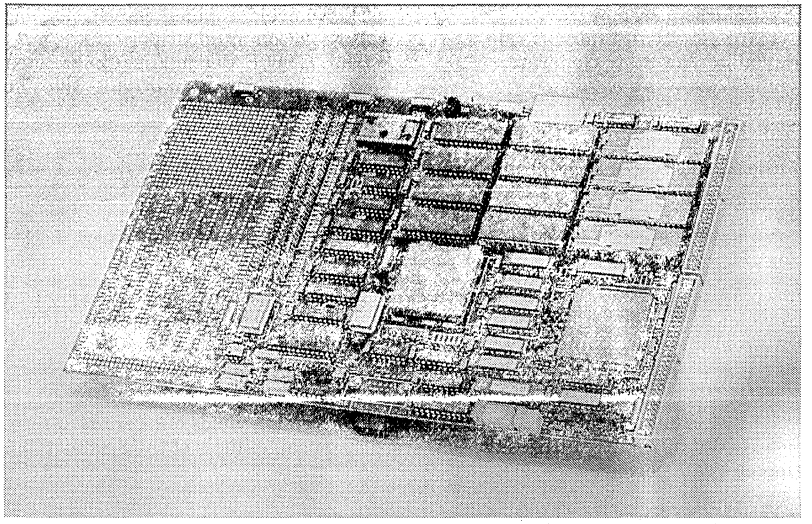
ENVIRONMENTAL CHARACTERISTICS

Operating Temperature	10°C to 40°C (50°F to 104°F)
Operating Humidity	Maximum 85% Relative Humidity, non-condensing

ORDERING INFORMATION

Order Code	Description
ICE960MC	The complete 20 MHz ICE-960MC emulator system including control unit, processor module, power supply, SAST, OIB, SAB, serial communications cable (SCOM4), IEDIT, V1.0 software. (Requires software license, Class I)
ICE960MC25P	25 MHz ICE960MC as described above
I960MCUPG	Conversion kit to convert ICE-960KB to ICE-960MC. Consists of new host and probe software, probe firmware, and manual. Requires ICE-960KB V2.0 hardware.
ICE960KBREM	Optional 2 Mbyte Relocatable Expansion Memory Board.

QT960 EVALUATION AND PROTOTYPING BOARD



270743-1

LOW COST EVALUATION TOOL

The QT960 products give you a 32-bit starter kit to begin software evaluation and hardware design at a low cost. The boards feature the 20 MHz 80960KB 32-bit embedded processor. The 80960KB has integrated floating point, instruction and register caches, and an on-chip interrupt controller. The 80960K-series are the first in a new architectural family of embedded processors from Intel built using Intel's CHMOS IV† process. These boards provide you with full access to the features of the 80960KB processor. A wire wrap prototyping area offers you easy access to board features to test your designs. Interleaved EPROM means fast execution of your code taking advantage of the 80960KB's burst bus. A programmable wait state generator simulates different memory environments useful in evaluating the performance of your code. These features make the QT960 boards useful low cost tools for the 32-bit embedded designer.

Once written, you can debug your program with NINDY, an EPROM resident debug monitor. NINDY enables you to download code, set seven different trace modes, display and modify memory or registers, and disassemble problem code sequences.

Available separately from Intel are the ASM-960 (assembly language) and iC-960 (high-level language) products which provide you with the code development environment for the QT960 boards.

The starter kit comes in two versions: the QT960F version has fast SRAM, high speed EPROM and Flash memory; the QT960E version has lower cost SRAM, Flash memory and no high speed EPROM. Each version has NINDY in either EPROM (QT960F) or Flash memory (QT960E), power supply cable, and the QT960 User Manual. Both versions also include the parts list, source code of the debug monitor, and the board data base (schematics) all on diskette. Armed with this starter kit you now have a system to evaluate and prototype your product ideas quickly and at low cost.

November 1990
Order Number: 270743-002

intel

FEATURES

QT960 FEATURES

- 20 MHz Execution Speed
- 128K Bytes to Zero Wait State EPROM‡
- 128K Bytes of Flash Memory
- 128K Bytes of Zero Wait State SRAM‡
- Programmable Wait State Generator
- Prototyping Wire Wrap Area
- Five Instruction Traces
- Two Hardware Breakpoints
- Display/Modify Memory and Registers
- Code Disassembly
- High Level Language Support
- RS-232 Communications Link
- The QT960E Version has 128K Bytes of Two Wait State SRAM and 128K Bytes Four Wait State Flash Memory

Product Order Codes: EVQT960F20 and EVQT960E20

†CHMOS IV is a patented Intel process.
‡QT960F Version only.

FAST AND EASY CODE UPDATES

128K Bytes of Intel's 28F256 Flash memory provides an easy and quick method of changing your code in nonvolatile memory. Flash memory may be conveniently reprogrammed without removing it from the board while software is under development.

FAST EPROM

Interleaved fast EPROM (Intel's 27C202) on the QT960F version yields one-zero-zero-zero wait state code access. It efficiently utilizes the four word burst capabilities of the 80960KB bus maximizing program performance.

PROTOTYPING SUPPORT

A prototyping wire wrap area is provided on board with access to the system's signals and buses. This area gives you access to the board's features and allows you to easily test design ideas. A system bus connector is also provided for off board prototyping.

PROGRAMMABLE WAIT STATE GENERATOR

A software programmable wait state generator enables you to quickly model various memory speeds. Under software control you can set over 16 different wait state combinations and evaluate the performance of your target system.

DMA

The board offers you eight DMA channels accessed through a NINDY library function using Intel's 82380. In addition, off board connectors provide DMA I/O capabilities.

FIVE INSTRUCTION TRACES AND TWO HARDWARE BREAKPOINTS

NINDY utilizes the built-in trace capabilities of the 80960KB to provide you with single step, supervisor, call, return, and branch instruction tracing offering you extensive debug capabilities for software examination and modification. Two hardware breakpoints enable you to break on and examine EPROM resident code.

FEATURES

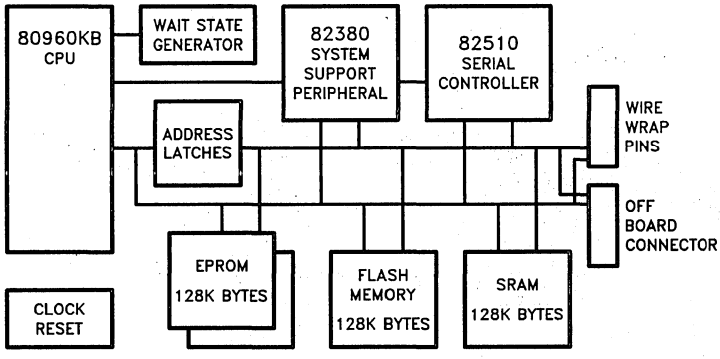
HIGH LEVEL LANGUAGE SUPPORT

NINDY is capable of downloading absolute object code generated by ASM-960 or iC-960. ASM-960 and iC-960 may be purchased separately from Intel.

COMMUNICATION AND SOFTWARE REQUIREMENTS

The QT960 boards communicate with the host through the RS-232 link using an Intel 82510 UART provided on board. The boards support five baud rates: 1200, 2400, 9600, 19200, and 38400. The default is 9600 baud. To communicate with the QT960 boards you must meet the following minimum software requirements:

- Terminal Emulator
- XMODEM Download Capabilities

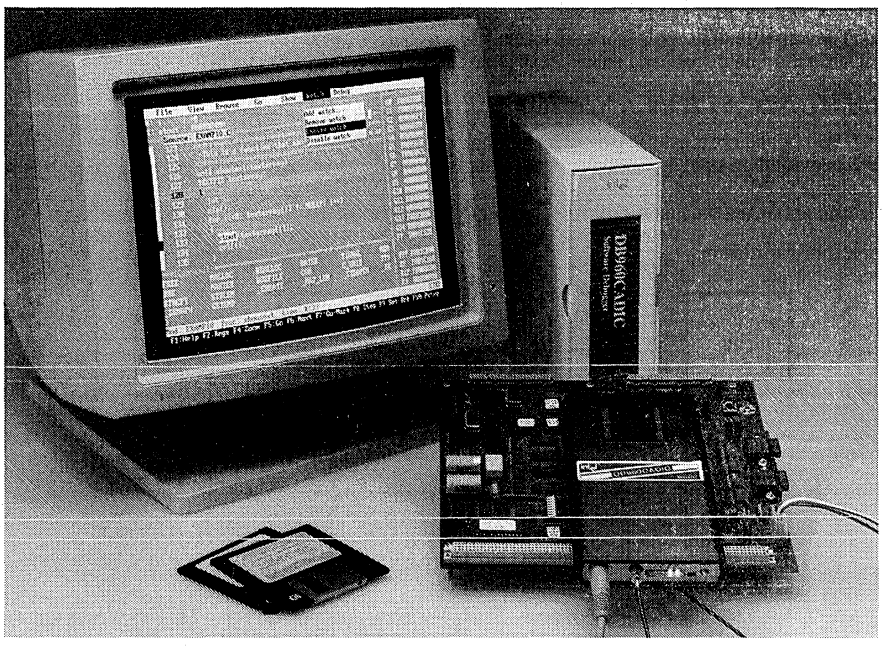


270743-2

Block Diagram of the QT960 Board

For information or the number of your nearest sales office call 800-548-4752 (U.S. and Canada).
Intel Corporaton, Literature Department, 3065 Bowers Avenue, Santa Clara CA 95051, United States. Tel: 408-987-8080.

DB960CADIC IN-CIRCUIT DEBUG MONITOR



280900-1

DB960CADIC

Intel's DB960CADIC, the in-circuit debug monitor for the 33 MHz i960CA embedded microprocessor, represents a new generation of development tool technology.

DB960CADIC allows users to debug high-speed, cached applications at the full speed of the i960CA target processor. Controlled by Intel's DB interface, DB960CADIC offers the user a tool with a powerful feature set at a fraction of the cost of traditional development tools. DB960CADIC is designed to improve productivity by allowing the user to debug software before and after the target system arrives, with minimal hardware intrusion.

Features

- Real-time emulation of the i960CA embedded microprocessor at speeds up to 33 MHz
- Full development and debug support for i960CA on-chip cache and RAM
- Minimal intrusive operation, allowing the user to debug the target system with minimal modification subject to initial design constraints
- Breakpoint capabilities include ten software breakpoints, two hardware execution address breakpoints, and two hardware data address breakpoints. The human interface supplements these breakpoints with the ability to break on data values, conditions, and a four-state state machine in non-real time.
- Low-Cost
- Source-Level, Symbolic Debugging in a Windowed Human Interface with pull down Menus (DB). This interface is consistent across i960CA tools.
- 128K Bytes User Memory
- Virtual I/O, the ability to perform I/O between the DB960CADIC unit and the host
- In-Circuit operation facilitates easy transition between target systems
- Optional Stand-Alone Self-Test (DB960CASAST) Module
- Optional Logic Analyzer Interface Board (LAI960CA)

June 1991
Order Number: 280900-002

DB960CADIC IN-CIRCUIT DEBUG MONITOR

Full-Speed Debug and Development

The DB960CADIC In-Circuit Debug Monitor provides sophisticated real-time hardware and software debug capabilities for i960CA embedded microprocessor-based designs. The user can run at the full speed of the target processor, ensuring that elusive timing bugs will be found. The DB960CADIC is jumpered to receive a clock pulse from either the user's target system, or from an internal 25 MHz clock.

Ideal for All Stages of Development

DB960CADIC can be used by both hardware and software developers, at any stage of design. Early in the development process, DB960CADIC allows software debugging when inserted into an existing i960CA board such as the DB960CASAST module or the EV80960CA board. Later in the design cycle, DB960CADIC can be inserted into the user's target system, thus facilitating debug of hardware/software integration.

Speed Development with Source Code, Symbolic Debugging

Using source code oriented debugging in a windowed, symbolic interface, software engineers can increase productivity by debugging in the medium they are familiar with, software source.

Commands can be entered via either function keys, pull-down menus which group logically related commands, or a supplementary command line which allows entry of complex conditions. In addition, source code symbolics can be used to examine and modify memory and registers. Optimal symbolic debugging can be achieved when using DB960CADIC with genuine Intel languages.

Powerful Break Capabilities

DB960CADIC provides complex emulation control by utilizing the on-chip debug registers within the i960CA. Real-time break capabilities include the ability to break on any two execution addresses or data access addresses in hardware. Software breakpoints are also used to supplement the hardware breakpoints for RAM-based memory subsystems. DB960CADIC extends these capabilities by providing the ability to break on data values, NOT data values, or combinations of the above in a four-state state machine. More complex conditions such as breaking when a variable is less than a certain value can be entered via a very flexible feature called conditional breakpoints.

128K Bytes User Memory

DB960CADIC provides the user with 128K bytes of memory in Region F of the i960CA target space. Since the debug monitor is also placed in Region F, the on-chip bus interface unit of the i960CA is configured to address region F as byte-wide memory with 5 waitstates and no burst accesses allowed.

Virtual Input/Output

DB960CADIC is shipped with documented library calls which provide users with a built-in mechanism of performing target I/O using the host system. These libraries provide the ability to simulate I/O operations in the target system before target hardware is available.

High Speed Serial Link

Communication between a host and the DB960CADIC module is supported via RS232 and RS422 communication links. RS232 allows access to industry standard serial protocols while the RS422 link provides a higher speed communication mechanism currently emerging in the development market. PC/AT Compatible RS422 communication boards are available from various third party vendors.

Optional Stand-Alone Self Test Chassis

An optional stand-alone self test chassis complements DB960CADIC by allowing the user to debug and test code before prototype hardware is available. The DB960CASAST includes self-test circuitry to ensure that the DB960CADIC unit is working correctly. It also provides 4 Megabyte of DRAM to be used for

DB960CADIC IN-CIRCUIT DEBUG MONITOR

developing applications. This memory has a (3,1,1,1) waitstate pattern at 25 MHz. This waitstate pattern is programmable using the bus controller unit in the i960CA. It also includes an 8254 programmable timer which can optionally interrupt the i960CA processor and provide the ability to time code sequences.

Optional Logic Analyzer Interface Board

The LAI960CA board provides access to i960CA pins by routing the signals to easily accessible stake pins while passing them through to the target system.

Software Completes the System

Intel provides a comprehensive software development environment to complement DB960CADIC. This environment includes C and ASM source languages, a retargetable debug monitor, and DB960CADIC. The languages support the entire range of 80960 embedded processors.

Worldwide Service, Support, and Training

To augment its development tools, Intel offers a full array of seminars, classes, workshops, field application engineering expertise, hotline technical support, and on-site service.

Intel also offers a Software Support contract which includes technical software information, telephone support, automatic distributions of software and documentation updates, *iCOMMENTS* publication, remote diagnostic software, and a development tools troubleshooting guide.

Intel's 90-day Hardware Support package includes technical hardware information, telephone support, warranty on parts, labor, material, and on-site hardware support.

Intel Development Tools also offers a 30-day, money-back guarantee to customers who are not satisfied after purchasing any Intel development tool.

DB960CADIC SPECIFICATIONS AND REQUIREMENTS

Host System Requirements

Host system requirements to run the in-circuit debugger include the following:

—DOS version 3.2 or later excluding DOS 4.0

—640 bytes of RAM in conventional memory

—A 20 MB hard disk

—An RS232 or RS422 Serial Port

Evaluated Systems include:

IBM PC-AT* with DOS 3.3

COMPAQ 386* with DOS 3.3

Intel 301/302* with DOS 3.3

IBM Personal System/2* Model 70/80 with DOS 4.01

Environment Characteristics

Operating Temperature:	+10°C to +40°C (50°F to 104°F)
Operating Humidity:	Maximum of 90% relative humidity, non-condensing.

DB960CADIC IN-CIRCUIT DEBUG MONITOR

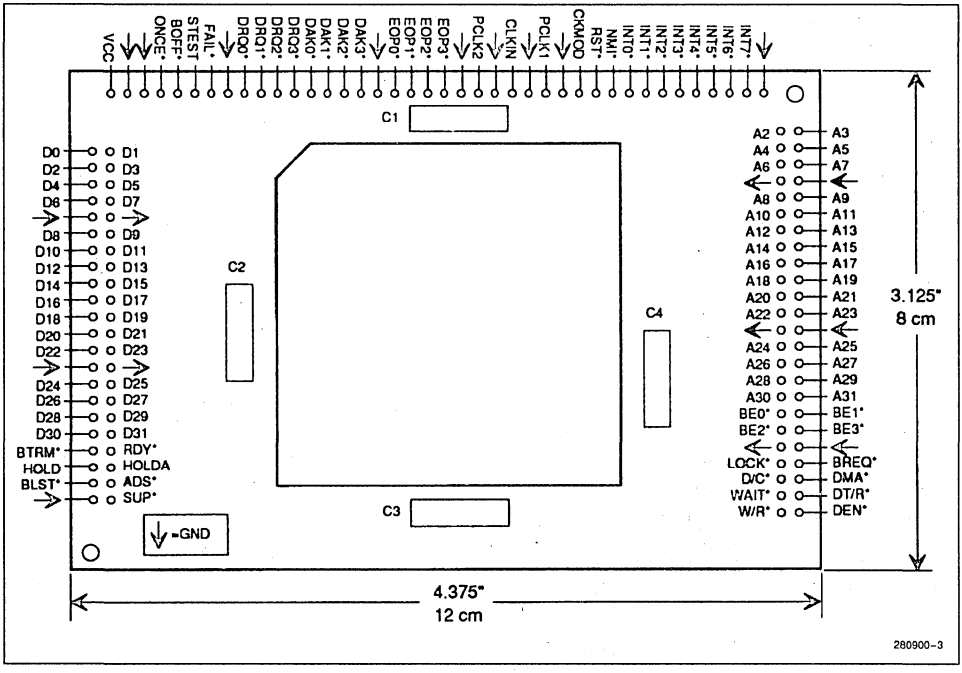


Figure 1

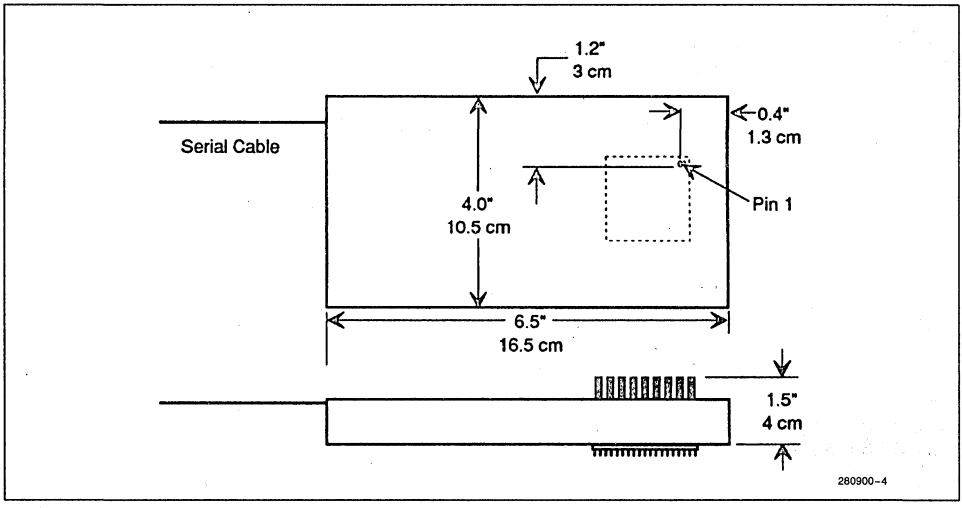


Figure 2

DB960CADIC IN-CIRCUIT DEBUG MONITOR

DB960CADIC Interface Considerations

Target systems intended to receive DB960CADIC must meet the following requirements:

- The target system must not respond to memory accesses in Region F (0F0000000-0FFFFFFFF) with DB960CADIC installed. DB960CADIC provides an ACTIVE out signal which can be used to qualify bus logic to prevent this occurrence when DB960CADIC is installed.
- The Target System must provide 1.3 Amps of power (worst case) .9 Amps average to power the DB960CADIC unit.
- Use of one of the nine directly accessible i960CA interrupts.
- Use of interrupt table entry 242 or 248.
- Additional Signal Loading as follows:

The DB960CADIC makes use of the PCLK outputs, D0 through D7, and some of the address and control signals of the processor. The following table lists the worst case loadings added by the presence of the DB960CADIC circuitry.

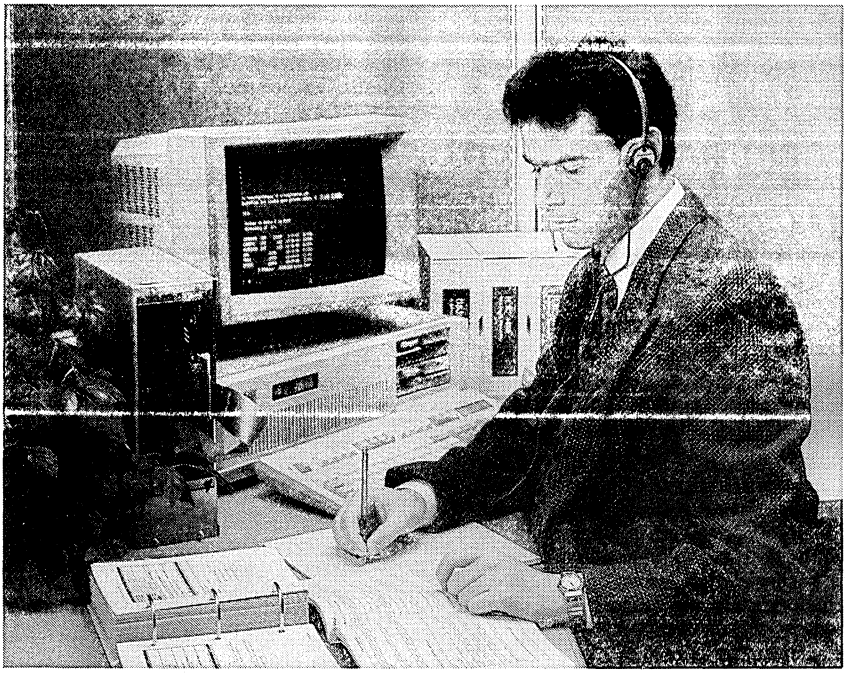
Signal Name	DC Load (μA)	Capacitive Load (pF)
PCLK1	+25/−250	8
PCLK2	+30/−255	17
CLKIN	+12/−12	13
D0:D7	+20/−600	10
A31:26	+25/−250	11
A2:A17	+20/−100	10
BE0*, BE1*	+20/−100	10
ADS*	+50/−500	13
W/R*	+50/−500	13
WAIT*	+25/−250	8
BLAST*	+25/−250	8
FAIL*	+20/−20	8
RESET*	+15/−15	25
INT0:7*	+20/−500	15
NMI*	+20/−500	15

Additional Loading Imposed on the Target by the DB960CADIC

Ordering Information

DB960CADIC	In-circuit debug monitor for the i960CA embedded microprocessor. Operates at speeds up to 33 MHz. Includes hardware debug module, RS232/RS422 serial cables, DOS host software, and documentation.
DB960CASAST	Stand-Alone Self Test Unit for DB960CADIC. Includes built-in power supply, self-test board, 4Mbyte of usable DRAM for code development, and enclosure.
DB960CAST	DB960CADIC and DB960CASAST as described above.
LAI960CA	Optional Logic Analyzer Interface Board for the i960CA system. Does not require DB960CADIC.

INTEL DEVELOPMENT TOOLS SOFTWARE SERVICES



280921-1

Intel is committed to providing high quality products and customer support. Our commitment to quality is demonstrated by a 30 day, money-back, unconditional refund to customers not satisfied with their purchase of an Intel Development Tools product.

Intel supports its customers by offering a 90-day software warranty and standard software support including free technical support over the phone.

Intel software is continuously undergoing improvement. For customers who desire the security of having the most current software and the convenience of having updates sent automatically, Intel offers inexpensive Software Support Contracts.

SOFTWARE WARRANTY

The standard software warranty is 90 days and entitles the customer to the following (provided the customer has registered their software by returning a completed Warranty Registration Card):

- Replacement of defective media
- Software product updates occurring within the 90 day warranty period.

STANDARD SOFTWARE SUPPORT

Standard Software Support, provided at no additional cost, offers the following additional benefits:

- Free Technical Information Phone Service ("TIPS")
- Timely response to Software Problem Reports

intel

INTEL DEVELOPMENT TOOLS SOFTWARE SERVICES

Software Support Contracts

Software Support Contracts cover products for one year from the date of purchase and are renewable annually. The following benefits are provided:

- Automatic Software Updates
- Standard Software Support
- Remote Diagnostic Software for DOS-based products.
- Monthly issues of *iCOMMENTS*, a technical support publication
- Quarterly issues of Troubleshooting Guides (host-specific)
- Quantity discounts

ORDERING INFORMATION

Ordering Procedures

For more information, call 1-800-468-3548 or your local Intel sales office. Similar support offerings are available outside of North America. Software Support Contracts are available for North American customers only.

All orders for contracts, including renewals, can be submitted through the local Intel sales office or directly to the **Development Tools Operation** by calling **1-800-874-6835**.

To order a Software Support Contract, a customer must have registered their product or provide proof of ownership. Customers must also have the most current version of the software, otherwise, they must order a product upgrade before a support contract may be purchased.

Pricing is a percentage of the List Price, based on the number of copies covered by the Software Support Contract. For emulators, the percentage will be applied to the identified list price of the software portion only, not the full list price of the emulator.

Pricing Information

Quantity discounts are:

product quantity	*pricing per copy*
1–10 copies	20% of List Price
11–25 copies	15% of List Price
26+ copies	10% of List Price

VAX and MicroVAX software not included. Please call **1-800-874-6835** for price quote.

Ordering Information

order code	*description*
SWSUPPORT51	Software Support Contract for 51 family
SWSUPPORT86	Software Support Contract for 86 family
SWSUPPORT96	Software Support Contract for 96 family
SWSUPPORT286	Software Support Contract for 286 family
SWSUPPORT386	Software Support Contract for 386 family
SWSUPPORT486	Software Support Contract for 486 family
SWSUPPORT960	Software Support Contract for 960 family

iRMK™ 960 REAL-TIME KERNEL

- 32-Bit Real-Time Multi-Tasking Kernel for the i960™ Microprocessor Family
- Flexible, Modular Design to Ease System Integration
- Fast Execution with Predictable Response Time for Time-Critical Applications
- Compact Code Size (14 Kbytes—Including All Optional Modules)
- Requires Only an i960 KA, KB or MC Embedded Processor
- Bus Independent
- Easy Customization and Add-On Enhancements
- Easily EPROMmable
- Comprehensive Development Tool Support

The iRMK 960 Real-Time Kernel is the 32-bit real-time executive developed and supported by Intel, the i960 architecture experts. The kernel is a small, fast and highly modular package of system control software. It contains the basic software building blocks that act as the foundation in using the key features of the i960 microprocessor. The iRMK 960 software is fully supported by an array of tools that work in the most popular development environments (i.e., DOS*, VAX/VMS*, SUN*).

The iRMK 960 Real-Time Kernel is available off-the-shelf. The kernel reduces the cost and risk of designing and maintaining software for numerous real-time applications such as, embedded control systems and dedicated real-time subsystems in multiprocessor environment. Use of the kernel can save man years that might otherwise be spent developing or porting another real-time kernel. This means reduced time to market for the user.

*DOS® is a registered trademark of Microsoft Corporation.
VAX/VMS™ is a trademark of Digital Equipment Corporation.
SUN™ is a trademark of Sun Microsystems.

October 1991
Order Number: 281006-001

ARCHITECTURAL OVERVIEW

At the heart of the architecture are the kernel core modules consisting of a scheduler, task manager, interrupt manager and time manager (See Figure 1). As additional building blocks, the kernel provides optional modules consisting of a mailbox manager, semaphore manager, memory manager, on-processor interrupt controller manager and fault handler manager. The optional device manager for the 82380 Integrated System Peripheral (ISP) and 8254 Programmable Interval Timer (PIT) complete the architecture.

FUNCTIONAL FEATURES

A Full Set of Real-Time Building Blocks

The kernel provides a full set of services for real-time applications including task management, time management, synchronization of and communications between tasks, and memory pool management.

TASK MANAGEMENT

The iRMK 960 kernel uses system calls to create, manage and schedule tasks in a multi-tasking environment. It provides pre-emptive priority scheduling combined with optional time-slice (round robin) scheduling.

The scheduling algorithm used by the kernel enables tasks to be rescheduled in a fixed amount of time regardless of the number of tasks. Applications may contain any number of tasks.

An application can integrate optional task handlers to customize task management. These handlers can execute on task creation, task switch, task deletion and task priority change. Task handlers can be used for a wide range of functions, including saving and restoring the state of coprocessor registers on task switch, masking interrupts based on task priority or implementing statistical and diagnostic monitors.

INTERRUPT MANAGEMENT

iRMK 960 interrupts are managed by immediately switching control to user-written interrupt handlers when an interrupt occurs.

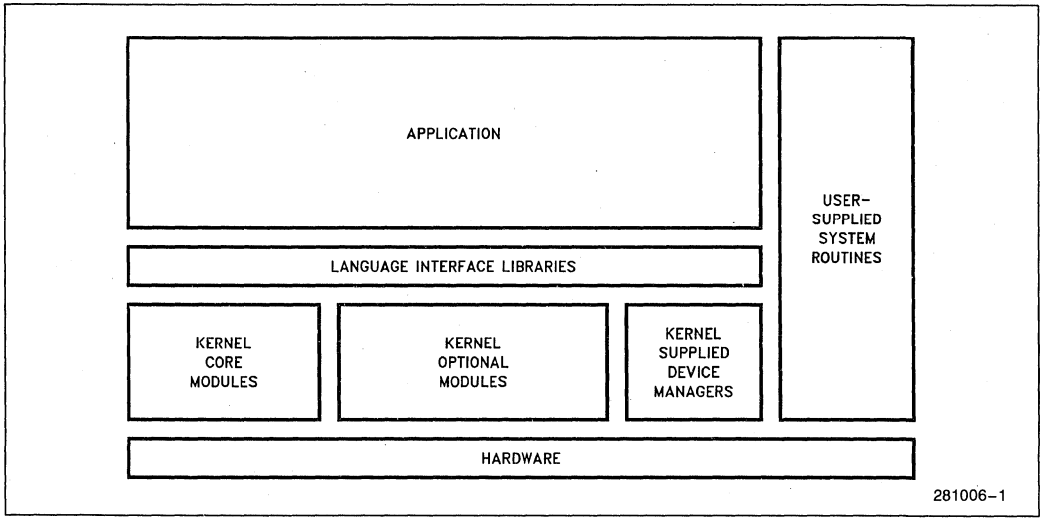


Figure 1. iRMK™ 960 Real-Time Architecture

Response to interrupts is both fast and predictable. Most of the kernel's system calls can be executed directly from interrupt handlers.

TIME MANAGEMENT

The time management features included in the kernel provide single-shot alarms, repetitive alarms and a real-time clock. In addition, alarms can be reset.

These time management facilities can solve a wide range of real-time programming problems. Single-shot alarms, for example, can be used to handle timeouts. If the timeout occurs, the alarm invokes a user-written handler; if the event occurs before the timeout, the application simply deletes the alarm. Other uses for the kernel's time management facilities including polling devices with repetitive alarms, putting tasks to sleep for specified periods of time, or implementing a time-of-day clock.

INTERTASK SYNCHRONIZATION AND COMMUNICATION

Semaphores, regions and mailboxes are the key mechanisms the kernel uses for synchronizing tasks and communicating between tasks.

Semaphores are objects used for intertask signaling and synchronization. Tasks exchange abstract "units" with semaphores as a means of becoming synchronized. A task **requests** a unit from a semaphore to gain access to a resource. If the resource is available, the semaphore will have a unit to give to the task, enabling the task to proceed. A task **sends** a unit to a semaphore to indicate that it has released a previously obtained resource.

A special binary type of semaphore is called a Region. Regions are used to ensure mutual exclusion, thus preventing deadlock when tasks contend for control of system resources. A task holding a region's unit runs at the priority of the highest priority task waiting in queue for the region's unit.

Mailboxes are queues that can hold any number of messages and are used to exchange data between tasks. Either data or pointers can be sent using mailboxes. The kernel allows mailbox messages to be of any length. High priority messages can be placed (jammed) at the front of the message queue to ensure that they are received and processed before other messages queued at the mailbox.

To ensure that high priority tasks are not blocked by lower priority tasks, the kernel allows tasks to queue at semaphores and mailboxes in priority order. The kernel also supports first-in, first-out task queueing.

MEMORY POOL MANAGEMENT

The iRMK 960 kernel uses the concept of memory pools to efficiently divide and manage blocks of memory. The memory pool manager provides for both fixed and variable block allocation.

Memory can be divided into any number of pools. Multiple memory pools might be created for different speed memories, or for allocating different size blocks. The times to allocate and de-allocate fixed-size areas from within a pool have a fixed upper bound.

The kernel-supplied memory manager works with flat memory architecture. Users can also write their own memory manager to provide different memory management policies or support virtual memory.

Hardware Requirements and Support

The kernel requires only an i960 microprocessor and sufficient memory for itself and its application. The kernel's design, however, recognizes that many systems use additional programmable peripheral devices and coprocessors. The kernel provides optional device managers for:

- The 82380 Integrated System Peripheral (ISP) chip
- The 8254 Programmable Interval Timer (PIT) chip

An application can supply managers for other devices and coprocessors in addition to or in replacement of the devices listed above.

The openness of the iRMK 960 kernel is a major benefit to the OEM. The kernel is designed to be programmed into PROM or EPROM, making it easy to use in embedded designs. In addition, it can be used with any system bus, including those of MULTIBUS I and MULTIBUS II bus architectures.

A Modular Architecture for Easy Customization

The kernel is designed for maximum flexibility. It can be customized for any application. Each major function, mailboxes for example, is implemented as a separate module. The kernel's modules have not been linked together and are supplied individually. (See Table 1 for the list of kernel modules, and their approximate sizes.)

The user links only the modules needed for his application. Any module not used does not need to be linked in, and does not increase the size of the kernel in your application. The user can also replace

any optional kernel module with one that implements specific features required by the application. For example, the user might want to replace the kernel's memory manager with one that supports virtual memory.

Table 1. iRMK™ 960 Kernel Modules and Approximate Sizes

Core Modules	Bytes
Task Manager	2600
Interrupt Manager	150
Time Manager	3000
Scheduler	1700
Initialization	50
Optional Modules	
Mailbox Manager	1250
Semaphore Manager	2900
Memory Manger	1260
Fault Handler Manager	50
Miscellaneous	300
Optional Device Manager	
82380 Integrated System Peripheral	4200
8254 Programmable Interval Timer	1200

Total size of the (entire) kernel (minus device managers) is about 13.5 Kbytes.

Developing with the iRMK™ 960 Real-Time Kernel

Kernel applications can be written using any language or compiler that produces code that executes on the i960 microprocessor. This independence is achieved by using an interface library. This library works with the idiosyncracies of a particular language—for example, the ordering of parameters. The interface library translates the calls provided by the language into a standard format expected by the kernel. Intel provides an interface library for our iC 960 compiler. The source code of this library is included, so that the user can modify it to support other compliers.

Because the kernel is supplied as unlinked object modules, applications can be developed on any system that hosts the development tools needed.

Comprehensive Development Tool Support

Intel provides a complete line of 80960 development tools for writing and debugging iRMK 960 applications.

These tools include:

Software: ASM 960 assembler iC 960 compiler

NOTE:
These tools are available for DOS, VAX/VMS*, MicroVAX*, SUN* and EVA960KB 4MB environment

Debuggers:
- ICE™ 960 — In-Circuit Emulator for the i960 microprocessor
- SMD™ 960 — System Debug Monitor for the i960 microprocessor

Evaluation Vehicles:
- EVA960KB — AT Bus-Compatible Board
- A960KB4MB — AT Bus-Compatible Board with 4 Mbytes of Memory
- QT960 — Standalone Evaluation Vehicle

Intel Support, Consulting and Training

With iRMK 960 kernel software, the developer has available the total Intel i960 architecture and real-time expertise of Intel's support engineers. Intel provides telephone support, on or off-site consulting, troubleshooting guides and updates. The kernel includes 90 days of Intel's Technical Information Phone Service (TIPS). Extended support and consulting are also available.

Contents of the iRMK™ 960 Kernel Development Package

The iRMK 960 Kernel comes in a comprehensive package including:

- Kernel object modules
- Source for the kernel supplied 82380 Integrated System Peripheral and 8254 PIT device managers
- Source for the iC 960 interface library
- Source for sample applications showing the following:
  - — Structure of kernel applications
  - — Use of the kernel with an application written in iC 960 language
  - — Compile, bind and build sequences
  - — Sample initialization code for the i960 microprocessor
  - — Applications written to execute in a flat memory space
- User reference guide
- 90 days of customer support

LICENSING

iRMK 960 software requires prior execution of the standard Intel Software License Agreement (SLA). A single development copy requires a Class I license and allows iRMK 960 software to be loaded and run on one single-processor system.

SPECIFICATIONS

System Calls

The following items are system calls arranged by type:

iRMK™ 960 KERNEL SYSTEM CALLS LISTING

KERNEL INITIALIZATION

KN_initialize	Initialize kernel

OBJECT MANAGEMENT

KN_token_to_ptr	Returns a pointer to the area holding object
KN_current_task	Returns a pointer for the current task

TASK MANAGEMENT

KN_create_task	Creates a task
KN_delete_task	Deletes a task
KN_suspend_task	Suspends a task
KN_resume_task	Resumes a task
KN_set_priority	Change priority of a task
KN_get_priority	Return priority of a task

INTERRUPT MANAGEMENT

KN_set_interrupt	Specify interrupt handler
KN_stop_scheduling	Suspend task switching
KN_start_scheduling	Resume task switching

TIME MANAGEMENT

KN_sleep	Put calling task to sleep
KN_create_alarm	Create and start virtual alarm clock
KN_reset_alarm	Reset an existing alarm
KN_delete_alarm	Delete alarm
KN_get_time	Get time
KN_set_time	Set time
KN_tick	Notify kernel that clock tick has occurred

INTERTASK COMMUNICATION AND SYNCHRONIZATION

KN_create_semaphore	Create a semaphore
KN_delete_semaphore	Delete a semaphore
KN_send_unit	Add a unit to a semaphore
KN_receive_unit	Receive a unit from a semaphore
KN_create_mailbox	Create a mailbox
KN_delete_mailbox	Delete a mailbox
KN_send_data	Send data to a mailbox
KN_send_priority_data	Place (jam) priority message at head of message queue
KN_receive_data	Request a message from a mailbox

MEMORY MANAGEMENT

KN_create_pool	Create a memory pool
KN_delete_pool	Delete a memory pool
KN_create_area	Create a memory area from a pool
KN_delete_area	Return a memory area to a memory pool
KN_get_pool_attributes	Get a memory pool's attributes

PROGRAMMABLE INTERRUPT CONTROLLER MANAGEMENT

KN_initialize_PICs	Initialize PIC's
KN_mask_slot	Mask out interrupts on a specified slot
KN_unmask_slot	Unmask interrupts on a specified slot
KN_send_EOI	Signal the PIC that the interrupt on a specified slot has been serviced
KN_new_masks	Change interrupt masks
KN_get_slot	Return the most important active interrupt slot
KN_get_interrupt	Get address of specified interrupt handler

PROGRAMMABLE INTERVAL CONTROLLER MANAGEMENT

KN_initialize_PIT	Initialize the PIT
KN_start_PIT	Start PIT counting
KN_get_PIT_interval	Return PIT interval

PROCESSOR RECOGNIZED FAULT HANDLING

KN_get_fault_handler	Get address of fault handler currently associated with specified fault type
KN_set_fault_handler	Establish address of fault handler for the specified fault type

PROCESSOR INTERRUPT CONTROLLER SUPPORT

KN_get_processor_ _priority	Returns value of the processor
KN_set_processor_ _priority	Change the value of the processor priority

PERFORMANCE

The figures listed below were derived from a test suite running on a EVA-960 evaluation vehicle using an 80960KB running at 20 MHz. The EVA-960 has what is known as 2-1-1-1 wait state memory; what this means is that the first instruction of a four instruction fetch takes two wait states, and each of the three successive instructions takes one wait state. The figures are the worst case values obtained from several sets of test runs. The code was generated using the iC 960 DOS hosted compiler, Version 1.1.

Action	Time (in μs)
Create Pool	18
Get Pool Attributes	36
Delete Pool	1
Create Area	35
Delete Area	32
Create Semaphore	6
Delete Semaphore	14
FIFO Semaphore Send Unit	7
FIFO Semaphore Receive Unit	7
Region Semaphore Send Unit	18
Region Semaphore Receive Unit	14
Create Mailbox	19
Delete Mailbox	23
Send Data	21
Receive Data	21
Create Alarm	29
Delete Alarm	30
FIFO Semaphore Send/Receive Unit with Task Switch	75
Suspend Task with Task Switch	70
Basic Task Switch	50
Create Task	62
Suspend Task	26
Resume Task	50
Delete Task	50
Get Priority	5
Set Priority	27
Set Interrupt	3
Get Interrupt	3

MANUALS

iRMK 960 User's Manual (Intel Order #463863-001).

TRAINING INFORMATION

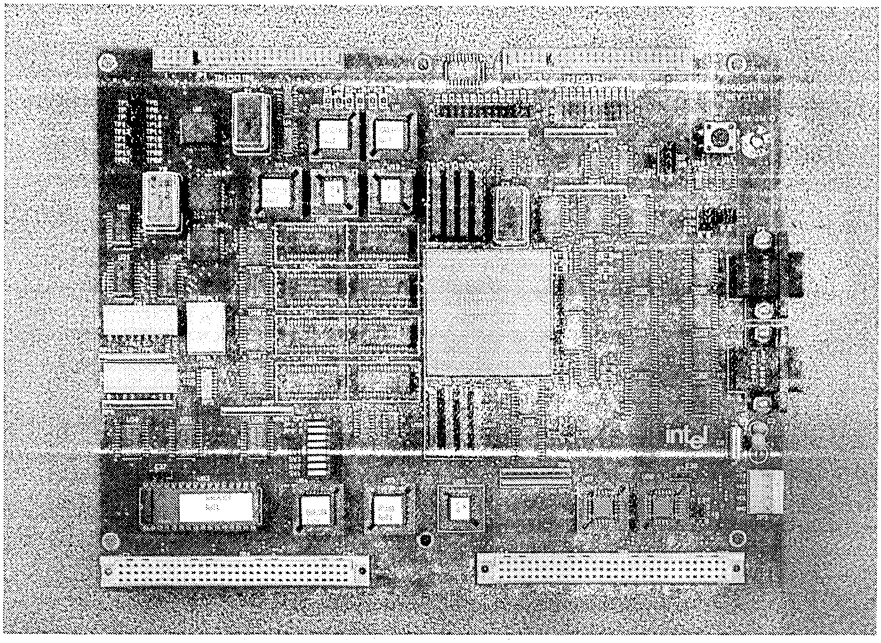
Intel Customer Service Training:

"80960 KA/KB Embedded Processor Training Course"

ORDERING INFORMATION

Ordering Code	Product Description
RMK960	iRMK 960 Real-Time Kernel

EV80960CA Evaluation Board



270870-1

Low Cost Processor Evaluation Tool

Intel's EV80960CA evaluation board provides a low-cost hardware environment for code execution and software debugging. The board features the 80960CA, the newest and highest performance member of Intel's family of 32-bit embedded microprocessors. The board allows a user's program to take full advantage of the power of the 80960CA and provides zero wait state execution of the user's code.

Popular features such as single line assembler/disassembler, single-step program execution and software breakpoints are standard on the EV80960CA's on-board monitor. Available separately, Intel offers a complete code development environment using the assembler (ASM-960) as well as high-level languages, such as Intel's iC-960 C compiler, to accelerate development schedules.

The EV80960CA evaluation board package features the 80960CA System Debug Monitor (SDM) in EPROM, a SDM host software floppy disk, a power supply cable, a 9-pin PC/AT serial connector for terminal and the EV80960CA User's Manual. The EV80960CA User's Manual includes schematics of the board, a part list and programmable logic (PLD) equations. The board is hosted on an IBM or BIOS-compatible PC/AT.

*The SRAM memory system provides zero wait state read (0-0-0-0-0) and one wait state write (1-1-1-1-0) performance.
**The DRAM memory system provides 2-1-1-1-1 reads and writes.

October 1991
Order Number: 270870-001

EV80960CA Evaluation Board

EV80960CA Features

- 25 MHz Execution Speed
- 32 Kbytes of EPROM for 80960CA SDM Target Operating Firmware
- 64 Kbytes of Zero Wait State Pipelined SRAM*
- 1 Mbyte of Static-Column Mode DRAM** expandable to 4 Mbytes
- Concurrent Interrogation of Memory and Registers
- Software Breakpoints
- Code Disassembly
- High-Level Language Support
- Two RS-232s for Host and User Communication
- Two iSBX I/O Connectors
- An Expansion Bus to Accommodate Eurocard Form-Factor Prototyping Boards

Fast Pipelined SRAM Memory System

The pipelined-read memory system of the EV80960CA provides true zero wait state read and one wait state performance. The memory design utilizes the internal wait state generator of the 80960CA.

Fast Static-Column Mode DRAM

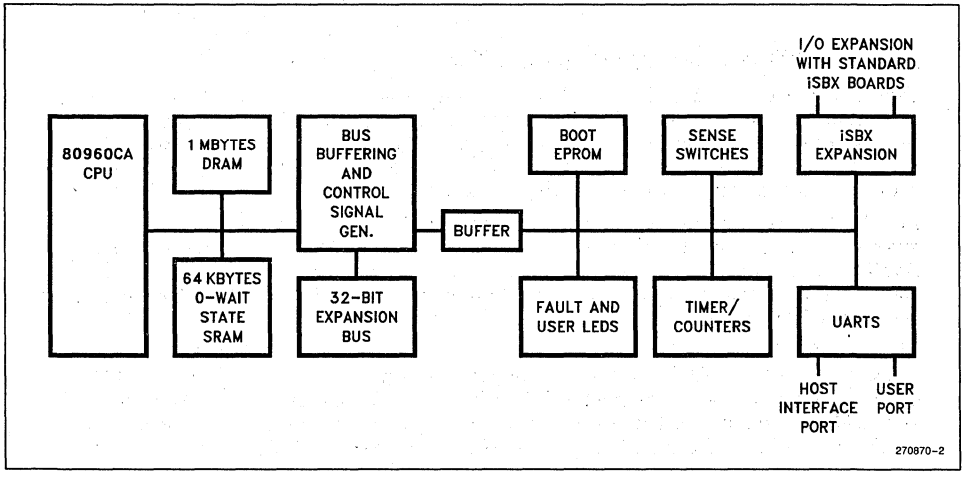
The memory design of the EV80960CA uses the 80960CA burst mode bus and static-column DRAM mode. The DRAM control PLDs are functionally isolated into interconnected state machines. The PLDs can be changed to allow alternative DRAM memory implementations with different DRAM access modes (static-column mode, nibble mode or fast-page mode).

Concurrent Interrogation of Memory and Registers

The 80960CA System Debug Monitor (SDM) for the EV80960CA allows the user to read and modify internal registers and external memory while the user's program is running on the board.

iSBX I/O Connectors and Expansion Interface

The EV80960CA evaluation board has two connectors to support both 8- and 16-bit standard iSBX Expansion Modules. The board also provides an expansion bus to accommodate Eurocard form-factor prototyping boards.



Block Diagram of the EV80960CA Board

EV80960CA Evaluation Board

Communication Link

The EV80960CA board communicates with the host through the RS-232 link using an Intel 82510 UART provided on board. The board supports seven baud rates: 300, 1200, 2400, 4800, 9600, 19200 and 38400.

Power Requirements

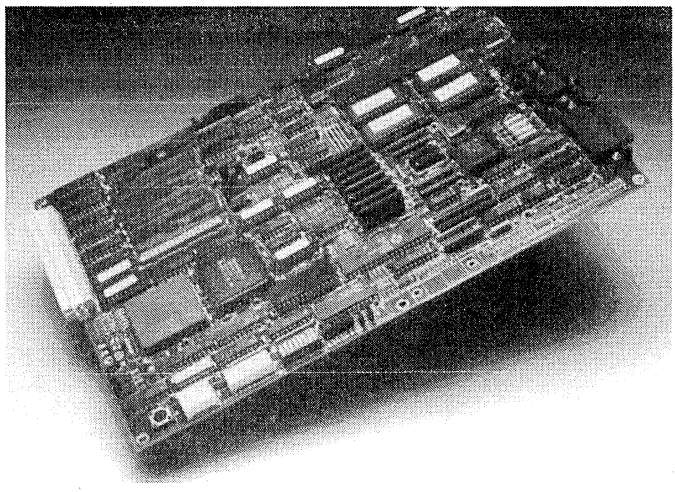
The EV80960CA Evaluation Board requires 5V at 2000 mA and ±12V at 25 mA.

Host System Requirements

The EV80960CA Evaluation Board is hosted on an IBM PC/AT or compatibles; a 386-based PC is recommended. The host system must meet the following minimum requirements:

- 512 Kbytes of Memory
- One 1.2 Mbyte Floppy Disk Drive
- PC-DOS 3.2 or Later
- A Serial Port (COM1 or COM2)

i960™ SA/SB EVALUATION BOARD



272033-1

i960™ SA/SB EVALUATION BOARD

The EV80960SX board is a general purpose evaluation tool for the i960™ SA/SB embedded processors. This evaluation board provides a high-performance DRAM subsystem, an interleaved EPROM subsystem, and a robust set of peripheral devices for benchmarking and debugging application code written for the i960 SA/SB embedded processors.

The EV80960SX is a great starter kit for your 32-bit application. The EV80960SX, NINDY debug environment, along with assembler and C-compiler (not provided) provide a seamless environment for developing code and evaluating the i960 SA/SB processors. The NINDY monitor provides code download capabilities from a number of popular development systems, including DOS-based PC's. Single step, breakpoints, register and memory display are among the full set of features provided by NINDY.

The board is provided with the following features:

- DRAM Subsystem operates at 1-0-0-0-0-0-0-0 wait states for read and write cycles in the burst mode. The DRAM subsystem runs at the maximum processor frequency of 16 MHz, using 100 ns fast page mode DRAMs. The DRAM subsystem can accommodate from 512 Kbytes to 4 Mbytes, using 4 or 8 ZIP-packaged DRAMs.
- Interleaved EPROM Subsystem executes burst program fetches with a 2-0-1-0-2-0-1-0 wait state performance.

The EPROM subsystem accommodates four, 32-pin or 28-pin 8-bit wide EPROMs with up to 150 ns access times.

- Flash EPROM Subsystem reads and writes two 8-bit wide Flash EPROMS.
- 8259A Interrupt Controller provides expanded interrupt capabilities using the i960 SA/SB's interrupt controller interface.
- Parallel Port Input allows fast downloads of code or data to the EV80960SX board. The parallel port provides auto-busy and interrupt capabilities, and is a full implementation of the Centronics standard.

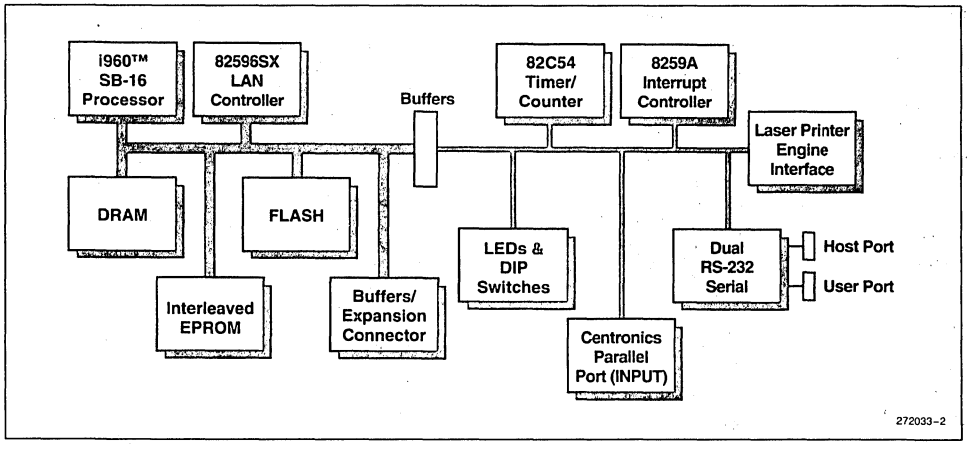
June 1991
Order Number: 272033-001

i960™ SA/SB EVALUATION BOARD

- Two serial ports provide queued and interrupt driven serial transfer at up to 128000 baud.
- 82C54 Timer/Counter provides a 32-bit counter and 16-bit counter, each with dedicated interrupts.
- Expansion/Prototype Bus (XBUS) allows expansion cards and prototype hardware direct access to the i960 SA/SB's bus and control signals. Optionally, a configurable wait state scheme provides a no glue interface to most peripherals attached to the XBUS.
- LEDs and Switches are user programmable. One 10-segment bar LED, a 7-segment LED and an 8-position switch are under program control.
- Local Area Networking (LAN) is implemented using an 82596SX LAN coprocessor.
- Laser Printer Control provides interfaces to TEC or Canon compatible laser engines.
- Monitor and Self-test diagnostics are provided for the EV80960SX in the EPROMs installed in the board.

The evaluation board comes complete with a design database included on diskette, the NINDY debug monitor on diskette and in EPROM, power and serial cables, schematics and user's manual.

The EV80960SX is a public domain design. The hardware is fully documented and provides working examples of popular memory and peripheral interfaces to the i960 SA/SB processor. The schematic and PLD database are provided with each board. The EV80960SX designs are easily duplicated and can be used directly as the building blocks for custom designs. Custom hardware can be prototyped using the expansion bus (XBUS) connector.



EV80960SX Evaluation Board

NORTH AMERICAN SALES OFFICES

ALABAMA

Intel Corp.
5015 Bradford Dr., #2
Huntsville 35805
Tel: (205) 830-4010
FAX: (205) 837-2640

ARIZONA

†Intel Corp.
410 North 44th Street
Suite 500
Phoenix 85008
Tel: (602) 231-0386
FAX: (602) 244-0446

CALIFORNIA

†Intel Corp.
21515 Vanowen Street
Suite 116
Canoga Park 91303
Tel: (818) 704-8500
FAX: (818) 340-1144

Intel Corp.
1 Sierra Gate Plaza
Suite 280C
Roseville 95678
Tel: (916) 782-8086
FAX: (916) 782-8153

†Intel Corp.
9665 Chesapeake Dr.
Suite 325
San Diego 92123
Tel: (619) 292-8086
FAX: (619) 292-0628

*†Intel Corp.
400 N. Tustin Avenue
Suite 450
Santa Ana 92705
Tel: (714) 835-9642
TWX: 910-595-1114
FAX: (714) 541-9157

*†Intel Corp.
San Tomas 4
2700 San Tomas Expressway
2nd Floor
Santa Clara 95051
Tel: (408) 986-8086
TWX: 910-338-0255
FAX: (408) 727-2620

COLORADO

Intel Corp.
4445 Northpark Drive
Suite 100
Colorado Springs 80907
Tel: (719) 594-6622
FAX: (303) 594-0720

*†Intel Corp.
600 S. Cherry St.
Suite 700
Denver 80222
Tel: (303) 321-8086
TWX: 910-931-2289
FAX: (303) 322-8670

CONNECTICUT

†Intel Corp.
301 Lee Farm Corporate Park
83 Wooster Heights Rd.
Danbury 06810
Tel: (203) 748-3130
FAX: (203) 794-0339

FLORIDA

†Intel Corp.
800 Fairway Drive
Suite 160
Deerfield Beach 33441
Tel: (305) 421-0506
FAX: (305) 421-2444

†Intel Corp.
5850 T.G. Lee Blvd.
Suite 340
Orlando 32822
Tel: (407) 240-8000
FAX: (407) 240-8097

GEORGIA

†Intel Corp.
20 Technology Parkway
Suite 150
Norcross 30092
Tel: (404) 449-0541
FAX: (404) 605-9762

ILLINOIS

*†Intel Corp.
Woodfield Corp. Center III
300 N. Martingale Road
Suite 400
Schaumburg 60173
Tel: (708) 605-8031
FAX: (708) 706-9762

INDIANA

†Intel Corp.
8910 Purdue Road
Suite 350
Indianapolis 46268
Tel: (317) 875-0623
FAX: (317) 875-8938

MARYLAND

*†Intel Corp.
10010 Junction Dr.
Suite 200
Annapolis Junction 20701
Tel: (301) 206-2860
FAX: (301) 206-3677
(301) 206-3678

MASSACHUSETTS

*†Intel Corp.
Westford Corp. Center
3 Carlisle Road
2nd Floor
Westford 01886
Tel: (508) 692-0960
TWX: 710-343-6333
FAX: (508) 692-7867

MICHIGAN

†Intel Corp.
7071 Orchard Lake Road
Suite 100
West Bloomfield 48322
Tel: (313) 851-8096
FAX: (313) 851-8770

MINNESOTA

†Intel Corp.
3500 W. 80th St.
Suite 360
Bloomington 55431
Tel: (612) 835-6722
TWX: 910-576-2867
FAX: (612) 831-6497

NEW JERSEY

*†Intel Corp.
Lincroft Office Center
125 Half Mile Road
Red Bank 07701
Tel: (908) 747-2233
FAX: (908) 747-0983

NEW YORK

*Intel Corp.
850 Crosskeys Office Park
Fairport 14450
Tel: (716) 425-2750
TWX: 510-253-7391
FAX: (716) 223-2561

*†Intel Corp.
2950 Express Dr., South
Suite 130
Islandia 11722
Tel: (516) 231-3300
TWX: 510-227-6236
FAX: (516) 348-7939

†Intel Corp.
300 Westage Business Center
Suite 230
Fishkill 12524
Tel: (914) 897-3860
FAX: (914) 897-3125

OHIO

*†Intel Corp.
3401 Park Center Drive
Suite 220
Dayton 45414
Tel: (513) 890-5350
TWX: 810-450-2528
FAX: (513) 890-8658

*†Intel Corp.
25700 Science Park Dr.
Suite 100
Beachwood 44122
Tel: (216) 464-2736
TWX: 810-427-9298
FAX: (804) 282-0673

OKLAHOMA

Intel Corp.
6801 N. Broadway
Suite 115
Oklahoma City 73162
Tel: (405) 848-8086
FAX: (405) 840-9819

OREGON

†Intel Corp.
15254 N.W. Greenbrier Pkwy.
Building B
Beaverton 97006
Tel: (503) 645-8051
TWX: 910-467-8741
FAX: (503) 645-8181

PENNSYLVANIA

*†Intel Corp.
925 Harvest Drive
Suite 200
Blue Bell 19422
Tel: (215) 641-1000
FAX: (215) 641-0785

*†Intel Corp.
400 Penn Center Blvd.
Suite 610
Pittsburgh 15235
Tel: (412) 823-4970
FAX: (412) 829-7578

PUERTO RICO

†Intel Corp.
South Industrial Park
P.O. Box 910
Las Piedras 00671
Tel: (809) 733-8616

TEXAS

†Intel Corp.
8911 N. Capital of Texas Hwy.
Suite 4230
Austin 78759
Tel: (512) 794-8086
FAX: (512) 338-9335

*†Intel Corp.
12000 Ford Road
Suite 400
Dallas 75234
Tel: (214) 241-8087
FAX: (214) 484-1180

*†Intel Corp.
7322 S.W. Freeway
Suite 1490
Houston 77074
Tel: (713) 988-8086
TWX: 910-881-2490
FAX: (713) 988-3660

UTAH

†Intel Corp.
428 East 6400 South
Suite 104
Murray 84107
Tel: (801) 263-8051
FAX: (801) 268-1457

WASHINGTON

†Intel Corp.
155 108th Avenue N.E.
Suite 386
Bellevue 98004
Tel: (206) 453-8086
TWX: 910-443-3002
FAX: (206) 451-9556

Intel Corp.
408 N. Mullan Road
Suite 102
Spokane 99206
Tel: (509) 928-8086
FAX: (509) 928-9467

WISCONSIN

Intel Corp.
330 S. Executive Dr.
Suite 102
Brookfield 53005
Tel: (414) 784-8087
FAX: (414) 796-2115

CANADA

BRITISH COLUMBIA

Intel Semiconductor of
Canada, Ltd.
4585 Canada Way
Suite 202
Burnaby V5G 4L6
Tel: (604) 298-0387
FAX: (604) 298-8234

ONTARIO

†Intel Semiconductor of
Canada, Ltd.
2650 Queensview Drive
Suite 250
Ottawa K2B 8H6
Tel: (613) 829-9714
FAX: (613) 820-5936

†Intel Semiconductor of
Canada, Ltd.
190 Attwell Drive
Suite 500
Rexdale M9W 6H8
Tel: (416) 675-2105
FAX: (416) 675-2438

QUEBEC

†Intel Semiconductor of
Canada, Ltd.
1 Rue Holiday
Suite 115
Tour East
Pt. Claire H9R 5N3
Tel: (514) 694-9130
FAX: 514-694-0064

†Sales and Service Office
*Field Application Location

NORTH AMERICAN DISTRIBUTORS

ALABAMA

Arrow Electronics, Inc.
1015 Henderson Road
Huntsville 35806
Tel: (205) 837-6955
FAX: (205) 721-1581

Hamilton/Avnet Electronics
4960 Corporate Drive, #135
Huntsville 35805
Tel: (205) 837-7210
FAX: (205) 721-0356

MTI Systems Sales
4950 Corporate Drive
Suite 120
Huntsville 35805
Tel: (205) 830-9526
FAX: (205) 830-9557

Pioneer/Technologies Group, Inc.
4835 University Square, #5
Huntsville 35805
Tel: (205) 837-9300
FAX: (205) 837-9358

ARIZONA

†Arrow Electronics, Inc.
4134 E. Wood Street
Phoenix 85040
Tel: (602) 437-0750
FAX: (602) 252-9109

Avnet Computer
30 South McKemy Avenue
Chandler 85226
Tel: (602) 961-6460
FAX: (602) 961-4787

Hamilton/Avnet Electronics
30 South McKemy Avenue
Chandler 85226
Tel: (602) 961-6403
FAX: (602) 961-1331

Wyle Distribution Group
4141 E. Raymond
Phoenix 85040
Tel: (602) 437-2088
FAX: (602) 437-2124

CALIFORNIA

Arrow Commercial System Group
1502 Crocker Avenue
Hayward 94544
Tel: (415) 489-5371
FAX: (415) 489-9393

Arrow Commercial System Group
14242 Chambers Road
Tustin 92680
Tel: (714) 544-0200
FAX: (714) 731-8438

†Arrow Electronics, Inc.
19748 Dearborn Street
Chatsworth 91311
Tel: (818) 701-7500
FAX: (818) 772-8930

†Arrow Electronics, Inc.
9511 Ridgehaven Court
San Diego 92123
Tel: (619) 565-4800
FAX: (619) 279-8062

†Arrow Electronics, Inc.
1180 Murphy Avenue
San Jose 95131
Tel: (408) 441-9700
FAX: (408) 453-4810

†Arrow Electronics, Inc.
2961 Dow Avenue
Tustin 92680
Tel: (714) 838-5422
FAX: (714) 838-4151

Avnet Computer
3170 Pullman Street
Costa Mesa 92626
Tel: (714) 641-4121
FAX: (714) 641-4170

Avnet Computer
1361B West 190th Street
Gardena 90248
Tel: (800) 345-3870
FAX: (213) 327-5389

Avnet Computer
755 Sunrise Blvd., #150
Roseville 95661
Tel: (916) 781-2521
FAX: (916) 781-3819

Avnet Computer
1175 Bordeaux Drive, #A
Sunnyvale 94089
Tel: (408) 743-3304
FAX: (408) 743-3348

Avnet Computer
21150 Califa Street
Woodland Hills 91376
Tel: (808) 345-3870
FAX: (818) 594-8333

†Hamilton/Avnet Electronics
3170 Pullman Street
Costa Mesa 92626
Tel: (714) 641-4100
FAX: (714) 754-6033

†Hamilton/Avnet Electronics
1175 Bordeaux Drive, #A
Sunnyvale 94089
Tel: (408) 743-3300
FAX: (408) 745-6679

†Hamilton/Avnet Electronics
4545 Viewridge Avenue
San Diego 92123
Tel: (619) 571-1900
FAX: (619) 571-8761

†Hamilton/Avnet Electronics
21150 Califa St.
Woodland Hills 91367
Tel: (818) 594-0403
FAX: (818) 594-8234

†Hamilton/Avnet Electronics
1361B West 190th Street
Gardena 90248
Tel: (213) 516-8600
FAX: (213) 217-6822

†Hamilton/Avnet Electronics
755 Sunrise Avenue, #150
Roseville 95661
Tel: (916) 925-2216
FAX: (916) 925-3478

Pioneer/Technologies Group, Inc.
134 Rio Robles
San Jose 95134
Tel: (408) 954-9100
FAX: 408-954-9113

†Wyle Distribution Group
124 Maryland Street
El Segundo 90245
Tel: (213) 322-8100
FAX: (213) 416-1151

Wyle Distribution Group
7431 Chapman Ave.
Garden Grove 92641
Tel: (714) 891-1717
FAX: (714) 891-1621

†Wyle Distribution Group
2951 Sunrise Blvd., Suite 175
Rancho Cordova 95742
Tel: (916) 638-5282
FAX: (916) 638-1491

†Wyle Distribution Group
9525 Chesapeake Drive
San Diego 92123
Tel: (619) 565-9171
FAX: (619) 365-0512

†Wyle Distribution Group
3000 Bowers Avenue
Santa Clara 95051
Tel: (408) 727-2500
FAX: (408) 727-5896

†Wyle Distribution Group
17872 Cowan Avenue
Irvine 92714
Tel: (714) 863-9953
FAX: (714) 263-0473

†Wyle Distribution Group
26010 Mureau Road, #150
Calabasas 91302
Tel: (818) 880-9000
FAX: (818) 880-5510

COLORADO

Arrow Electronics, Inc.
3254 C Frazer Street
Aurora 80011
Tel: (303) 373-5616
FAX: (303) 373-5760

†Hamilton/Avnet Electronics
9605 Maroon Circle, #200
Englewood 80112
Tel: (303) 799-7800
FAX: (303) 799-7801

†Wyle Distribution Group
451 E. 124th Avenue
Thornton 80241
Tel: (303) 457-9953
FAX: (303) 457-4831

CONNECTICUT

†Arrow Electronics, Inc.
12 Beaumont Road
Wallingford 06492
Tel: (203) 265-7741
FAX: (203) 265-7988

Avnet Computer
55 Federal Road, #103
Danbury 06810
Tel: (203) 797-2880
FAX: (203) 791-9050

†Hamilton/Avnet Electronics
55 Federal Road, #103
Danbury 06810
Tel: (203) 743-6077
FAX: (203) 791-9050

†Pioneer/Standard Electronics
112 Main Street
Norwalk 06851
Tel: (203) 853-1515
FAX: (203) 838-9901

FLORIDA

†Arrow Electronics, Inc.
400 Fairway Drive, #102
Deerfield Beach 33441
Tel: (305) 429-8200
FAX: (305) 428-3991

†Arrow Electronics, Inc.
37 Skyline Drive, #3101
Lake Mary 32746
Tel: (407) 333-9300
FAX: (407) 333-9320

Avnet Computer
3343 W. Commercial Blvd.
Bldg. C/D, Suite 107
Ft. Lauderdale 33309
Tel: (305) 979-9067
FAX: (305) 730-0368

Avnet Computer
3247 Tech Drive North
St. Petersburg 33716
Tel: (813) 573-5524
FAX: (813) 572-4324

†Hamilton/Avnet Electronics
5371 N.W. 33rd Avenue
Ft. Lauderdale 33309
Tel: (305) 484-5016
FAX: (305) 484-8369

†Hamilton/Avnet Electronics
3247 Tech Drive North
St. Petersburg 33716
Tel: (813) 573-3930
FAX: (813) 572-4329

†Hamilton/Avnet Electronics
7079 University Boulevard
Winter Park 32791
Tel: (407) 657-3300
FAX: (407) 678-1878

†Pioneer/Technologies Group, Inc.
337 Northlake Blvd., Suite 1000
Alta Monte Springs 32701
Tel: (407) 834-9090
FAX: (407) 834-0865

Pioneer/Technologies Group, Inc.
674 S. Military Trail
Deerfield Beach 33442
Tel: (305) 428-8877
FAX: (305) 481-2950

GEORGIA

Arrow Commercial System Group
3400 C. Corporate Way
Duluth 30136
Tel: (404) 623-8825
FAX: (404) 623-8802

†Arrow Electronics, Inc.
4250 E. Rivergreen Pkwy., #E
Duluth 30136
Tel: (404) 497-1300
FAX: (404) 476-1493

Avnet Computer
3425 Corporate Way, #G
Duluth 30136
Tel: (404) 623-5452
FAX: (404) 476-0125

Hamilton/Avnet Electronics
3425 Corporate Way, #G
Duluth 30136
Tel: (404) 446-0611
FAX: (404) 446-1011

Pioneer/Technologies Group, Inc.
4250 C. Rivergreen Parkway
Duluth 30136
Tel: (404) 623-1003
FAX: (404) 623-0665

ILLINOIS

†Arrow Electronics, Inc.
1140 W. Thorndale Rd.
Itasca 60143
Tel: (708) 250-0500

Avnet Computer
1124 Thorndale Avenue
Bensenville 60106
Tel: (708) 860-8573
FAX: (708) 773-7976

†Hamilton/Avnet Electronics
1130 Thorndale Avenue
Bensenville 60106
Tel: (708) 860-7700
FAX: (708) 860-8530

MTI Systems
1140 W. Thorndale Avenue
Itasca 60143
Tel: (708) 250-8222
FAX: (708) 250-8275

†Pioneer/Standard Electronics
2171 Executive Dr., Suite 200
Addison 60101
Tel: (708) 495-9680
FAX: (708) 495-9831

INDIANA

†Arrow Electronics, Inc.
7108 Lakeview Parkway West Dr.
Indianapolis 46268
Tel: (317) 299-2071
FAX: (317) 299-2379

Avnet Computer
485 Gradle Drive
Carmel 46032
Tel: (317) 575-8029
FAX: (317) 844-4964

Hamilton/Avnet Electronics
485 Gradle Drive
Carmel 46032
Tel: (317) 844-9333
FAX: (317) 844-5921

†Pioneer/Standard Electronics
9350 Priority Way West Dr.
Indianapolis 46250
Tel: (317) 573-0880
FAX: (317) 573-0979

†Certified VAD

NORTH AMERICAN DISTRIBUTORS (Contd.)

IOWA

Hamilton/Avnet Electronics
2335A Blairsferry Rd., N.E.
Cedar Rapids 52402
Tel: (319) 362-4757
FAX: (319) 393-7050

KANSAS

Arrow Electronics, Inc.
8208 Melrose Dr., Suite 210
Lenexa 66214
Tel: (913) 541-9542
FAX: (913) 541-0328

Avnet Computer
15313 W. 95th Street
Lenexa 61219
Tel: (913) 541-7989
FAX: (913) 541-7904

†Hamilton/Avnet Electronics
15313 W. 95th
Overland Park 66215
Tel: (913) 888-1055
FAX: (913) 541-7951

KENTUCKY

Hamilton/Avnet Electronics
805 A. Newtown Circle
Lexington 40511
Tel: (606) 259-1475
FAX: (606) 252-3238

MARYLAND

Arrow Commercial Systems Group
200 Perry Parkway
Gaithersburg 20877
Tel: (301) 670-1600
FAX: (301) 670-0188

†Arrow Electronics, Inc.
8300 Guilford Road, #H
Columbia 21046
Tel: (301) 995-6002
FAX: (301) 995-6201

Avnet Computer
7172 Columbia Gateway Dr., #G
Columbia 21045
Tel: (301) 995-0020
FAX: (301) 995-3515

†Hamilton/Avnet Electronics
7172 Columbia Gateway Dr., #F
Columbia 21045
Tel: (301) 995-3554
FAX: (301) 995-3515

†North Atlantic Industries
Systems Division
7125 Riverwood Dr.
Columbia 21046
Tel: (301) 290-3999

†Pioneer/Technologies Group, Inc.
15810 Gaither Road
Gaithersburg 20877
Tel: (301) 921-0660
FAX: (301) 670-6746

MASSACHUSETTS

Arrow Electronics, Inc.
25 Upton Dr.
Wilmington 01887
Tel: (508) 658-0900
FAX: (508) 694-1754

Avnet Computer
10 D Centennial Drive
Peabody 01960
Tel: (508) 532-9886
FAX: (508) 532-9660

†Hamilton/Avnet Electronics
10D Centennial Drive
Peabody 01960
Tel: (508) 531-7430
FAX: (508) 532-9802

†Pioneer/Standard Electronics
44 Hartwell Avenue
Lexington 02173
Tel: (617) 861-9200
FAX: (617) 863-1547

Wyle Distribution Group
15 Third Avenue
Burlington 01803
Tel: (617) 272-7300
FAX: (617) 272-6809

MICHIGAN

†Arrow Electronics, Inc.
19880 Haggerty Road
Livonia 48152
Tel: (313) 665-4100
FAX: (313) 462-2686

Avnet Computer
2876 28th Street, S.W., #5
Grandville 49418
Tel: (616) 531-9607
FAX: (616) 531-0059

Avnet Computer
41650 Garden Road
Novi 48375
Tel: (313) 347-1820
FAX: (313) 347-4067

Hamilton/Avnet Electronics
2876 28th Street, S.W., #5
Grandville 49418
Tel: (616) 243-8805
FAX: (616) 531-0059

Hamilton/Avnet Electronics
41650 Garden Brook Rd., #100
Novi 48375
Tel: (313) 347-4270
FAX: (313) 347-4021

†Pioneer/Standard Electronics
4505 Broadmoor S.E.
Grand Rapids 49512
Tel: (616) 698-1800
FAX: (616) 698-1831

†Pioneer/Standard Electronics
13485 Stamford
Livonia 48150
Tel: (313) 525-1800
FAX: (313) 427-3720

MINNESOTA

†Arrow Electronics, Inc.
10120A West 76th Street
Eden Prairie 55344
Tel: (612) 829-5588
FAX: (612) 942-7803

Avnet Computer
10000 West 76th Street
Eden Prairie 55344
Tel: (612) 829-0025
FAX: (612) 944-2781

†Hamilton/Avnet Electronics
12400 Whitewater Drive
Minnetonka 55343
Tel: (612) 932-0600
FAX: (612) 932-0613

†Pioneer/Standard Electronics
7625 Golden Triange Dr., #G
Eden Prairie 55344
Tel: (612) 944-3355
FAX: (612) 944-3794

MISSOURI

†Arrow Electronics, Inc.
2380 Schuetz Road
St. Louis 63141
Tel: (314) 567-6888
FAX: (314) 567-1164

Avnet Computer
739 Goddard Avenue
Chesterfield 63005
Tel: (314) 537-2725
FAX: (314) 537-4248

†Hamilton/Avnet Electronics
741 Goddard
Chesterfield 63005
Tel: (314) 537-1600
FAX: (314) 537-4248

NEW HAMPSHIRE

Avnet Computer
2 Executive Park Drive
Bedford 03102
Tel: (603) 624-6630
FAX: (603) 624-2402

NEW JERSEY

†Arrow Electronics, Inc.
4 East Stow Road
Unit 11
Marlton 08053
Tel: (609) 596-8000
FAX: (609) 596-9632

†Arrow Electronics, Inc.
6 Century Drive
Parsipanny 07054
Tel: (201) 538-0900
FAX: (201) 538-4962

Avnet Computer
1-B Keystone Ave., Bldg. 36
Cherry Hill 08003
Tel: (609) 424-8961
FAX: (609) 751-2502

Avnet Computer
10 Industrial Road
Fairfield 07006
Tel: (201) 882-2879
FAX: (201) 808-9251

†Hamilton/Avnet Electronics
1 Keystone Ave., Bldg. 36
Cherry Hill 08003
Tel: (609) 424-0110
FAX: (609) 751-2552

†Hamilton/Avnet Electronics
10 Industrial
Fairfield 07006
Tel: (201) 575-3390
FAX: (201) 575-5839

†MTI Systems Sales
6 Century Drive
Parsippany 07054
Tel: (201) 539-6496
FAX: (201) 539-6430

†Pioneer/Standard Electronics
14-A Madison Rd.
Fairfield 07006
Tel: (201) 575-3510
FAX: (201) 575-3454

NEW MEXICO

Alliance Electronics Inc.
10510 Research Avenue
Albuquerque 87123
Tel: (505) 292-3360
FAX: (505) 275-6392

Avnet Computer
7801 Academy Road
Bldg. 1, Suite 204
Albuquerque 87109
Tel: (505) 828-9725
FAX: (505) 828-0360

†Hamilton/Avnet Electronics
7801 Academy Rd. N.E.
Bldg. 1, Suite 204
Albuquerque 87108
Tel: (505) 765-1500
FAX: (505) 243-1395

NEW YORK

†Arrow Electronics, Inc.
3375 Brighton Henrietta Townline Rd.
Rochester 14623
Tel: (716) 427-0300
FAX: (716) 427-0735

Arrow Electronics, Inc.
20 Oser Avenue
Hauppauge 11788
Tel: (516) 231-1000
FAX: (516) 231-1072

Avnet Computer
933 Motor Parkway
Hauppauge 11788
Tel: (516) 231-9040
FAX: (516) 434-7426

Avnet Computer
2060 Townline
Rochester 14623
Tel: (716) 272-9306
FAX: (716) 272-9685

†Hamilton/Avnet Electronics
933 Motor Parkway
Hauppauge 11788
Tel: (516) 231-9800
FAX: (516) 434-7426

†Hamilton/Avnet Electronics
2060 Townline Rd.
Rochester 14623
Tel: (716) 292-0730
FAX: (716) 292-0810

Hamilton/Avnet Electronics
103 Twin Oaks Drive
Syracuse 13120
Tel: (315) 437-2641
FAX: (315) 432-0740

MTI Systems
50 Horseblock Road
Brookhaven 11719
Tel: (516) 924-9400
FAX: (516) 924-1103

MTI Systems
1 Penn Plaza
250 W. 34th Street
New York 10119
Tel: (212) 643-1280
FAX: (212) 643-1288

Pioneer/Standard Electronics
68 Corporate Drive
Binghamton 13904
Tel: (607) 722-9300
FAX: (607) 722-9562

†Pioneer/Standard Electronics
60 Crossway Park West
Woodbury, Long Island 11797
Tel: (516) 921-8700
FAX: (516) 921-2143

†Pioneer/Standard Electronics
840 Fairport Park
Fairport 14450
Tel: (716) 381-7070
FAX: (716) 381-5955

NORTH CAROLINA

†Arrow Electronics, Inc.
5240 Greensdairy Road
Raleigh 27604
Tel: (919) 876-3132
FAX: (919) 878-9517

Avnet Computer
2725 Millbrook Rd., #123
Raleigh 27604
Tel: (919) 790-1735
FAX: (919) 872-4972

Hamilton/Avnet Electronics
5250-77 Center Dr. #350
Charlotte 28217
Tel: (704) 527-2485
FAX: (704) 527-8058

†Hamilton/Avnet Electronics
3510 Spring Forest Drive
Raleigh 27604
Tel: (919) 878-0819

Pioneer/Technologies Group, Inc.
9401 L-Southern Pine Blvd.
Charlotte 28210
Tel: (704) 527-8188
FAX: (704) 522-8564

Pioneer Technologies Group, Inc.
2810 Meridian Parkway, #148
Durham 27713
Tel: (919) 544-5400
FAX: (919) 544-5885

OHIO

Arrow Commercial System Group
284 Cramer Creek Court
Dublin 43017
Tel: (614) 889-9347
FAX: (614) 889-9680

†Arrow Electronics, Inc.
6573 Cochran Road, #E
Solon 44139
Tel: (216) 248-3990
FAX: (216) 248-1106

Arrow Electronics, Inc.
8200 Washington Village Dr.
Centerville 45458
Tel: (513) 435-5563
FAX: (513) 435-2049

†Certified VAD

NORTH AMERICAN DISTRIBUTORS (Contd.)

OHIO (Contd.)

Avnet Computer
7764 Washington Village Dr.
Dayton 45459
Tel: (513) 439-6756
FAX: (513) 439-6719

Avnet Computer
30325 Bainbridge Rd., Bldg. A
Solon 44139
Tel: (216) 349-2505
FAX: (216) 349-1894

†Hamilton/Avnet Electronics
7760 Washington Village Dr.
Dayton 45459
Tel: (513) 439-6733
FAX: (513) 439-6711

†Hamilton/Avnet Electronics
30325 Bainbridge
Solon 44139
Tel: (800) 543-2984
FAX: (216) 349-1894

Hamilton/Avnet Electronics
2600 Corp Exchange Drive, #180
Columbus 43231
Tel: (614) 882-7004
FAX: (614) 882-8650

MTI Systems Sales
23404 Commerce Park Road
Beachwood 44122
Tel: (216) 464-6688
FAX: (216) 464-3564

†Pioneer/Standard Electronics
4433 Interpoint Boulevard
Dayton 45424
Tel: (513) 236-9900
FAX: (513) 236-8133

†Pioneer/Standard Electronics
4800 E. 131st Street
Cleveland 44105
Tel: (216) 587-3600
FAX: (216) 663-1004

OKLAHOMA

Arrow Electronics, Inc.
12111 East 51st Street, #101
Tulsa 74146
Tel: (918) 252-7537
FAX: (918) 254-0917

†Hamilton/Avnet Electronics
12121 E. 51st St., Suite 102A
Tulsa 74146
Tel: (918) 664-0444
FAX: (918) 250-8763

OREGON

†Almac Electronics Corp.
1885 N.W. 169th Place
Beaverton 97006
Tel: (503) 629-8090
FAX: 503-645-0611

Avnet Computer
9409 Southwest Nimbus Ave.
Beaverton 97005
Tel: (503) 627-0900
FAX: (503) 526-6242

†Hamilton/Avnet Electronics
9409 S.W. Nimbus Ave.
Beaverton 97005
Tel: (503) 627-0201
FAX: (503) 641-4012

Wyle
9640 Sunshine Court
Bldg. G, Suite 200
Beaverton 97005
Tel: (503) 643-7900
FAX: (503) 646-5466

PENNSYLVANIA

Avnet Computer
213 Executive Drive, #320
Mars 16046
Tel: (412) 772-1888
FAX: (412) 772-1890

Hamilton/Avnet Electronics
213 Executive, #320
Mars 16045
Tel: (412) 281-4152
FAX: (412) 772-1890

Pioneer/Technologies Group, Inc.
259 Kappa Drive
Pittsburgh 15238
Tel: (412) 782-2300
FAX: (412) 963-8255

†Pioneer/Technologies Group, Inc.
500 Enterprise Road
Keith Valley Business Center
Horsham 19044
Tel: (215) 674-4000
FAX: (215) 674-3107

TENNESSEE

Arrow Commercial System Group
3635 Knight Road, #7
Memphis 38118
Tel: (901) 367-0540
FAX: (901) 367-2081

TEXAS

Arrow Electronics, Inc.
3220 Commander Drive
Carrollton 75006
Tel: (214) 380-6464
FAX: (214) 248-7208

Avnet Computer
4004 Beltline, Suite 200
Dallas 75244
Tel: (214) 308-8181
FAX: (214) 308-8129

Avnet Computer
1235 North Loop West, #525
Houston 77008
Tel: (713) 867-7500
FAX: (713) 861-6851

†Hamilton/Avnet Electronics
1826-F Kramer Lane
Austin 78759
Tel: (800) 772-5668
FAX: (512) 832-4315

†Hamilton/Avnet Electronics
4004 Beltline, #200
Dallas 75244
Tel: (214) 308-8111
FAX: (214) 308-8109

†Hamilton/Avnet Electronics
1235 N. Loop West, #521
Houston 77008
Tel: (713) 240-7733
FAX: (713) 861-6541

†Pioneer/Standard Electronics
1826-D Kramer Lane
Austin 78758
Tel: (512) 835-4000
FAX: (512) 835-9829

†Pioneer/Standard Electronics
13765 Beta Road
Dallas 75244
Tel: (214) 386-7300
FAX: (214) 490-6419

†Pioneer/Standard Electronics
10530 Rockley Road, #100
Houston 77099
Tel: (713) 495-4700
FAX: (713) 495-5642

†Wyle Distribution Group
1810 Greenville Avenue
Richardson 75081
Tel: (214) 235-9953
FAX: (214) 644-5064

Wyle Distribution Group
4030 West Braker Lane, #330
Austin 78759
Tel: (512) 345-8853
FAX: (512) 345-9330

Wyle Distribution Group
11001 South Wilcrest, #100
Houston 77099
Tel: (713) 879-9953
FAX: (713) 879-6540

UTAH

Arrow Electronics, Inc.
1946 W. Parkway Blvd.
Salt Lake City 84119
Tel: (801) 973-6913

Avnet Computer
1100 E. 6600 South, #150
Salt Lake City 84121
Tel: (801) 266-1115
FAX: (801) 266-0362

Avnet Computer
17761 Northeast 78th Place
Redmond 98052
Tel: (206) 867-0160
FAX: (206) 867-0161

†Hamilton/Avnet Electronics
1100 East 6600 South, #120
Salt Lake City 84121
Tel: (801) 972-2800
FAX: (801) 263-0104

†Wyle Distribution Group
1325 West 2200 South, #E
West Valley 84119
Tel: (801) 974-9953
FAX: (801) 972-2524

WASHINGTON

†Almac Electronics Corp.
14360 S.E. Eastgate Way
Bellevue 98007
Tel: (206) 643-9992
FAX: (206) 643-9709

†Hamilton/Avnet Electronics
17761 N.E. 78th Place, #C
Redmond 98052
Tel: (206) 241-8555
FAX: (206) 241-5472

Wyle Distribution Group
15385 N.E. 90th Street
Redmond 98052
Tel: (206) 881-1150
FAX: (206) 881-1567

WISCONSIN

Arrow Electronics, Inc.
200 N. Patrick Blvd., Ste. 100
Brookfield 53005
Tel: (414) 792-0150
FAX: (414) 792-0156

Avnet Computer
20875 Crossroads Circle, #400
Waukesha 53186
Tel: (414) 784-8205
FAX: (414) 784-6006

†Hamilton/Avnet Electronics
28875 Crossroads Circle, #400
Waukesha 53186
Tel: (414) 784-4510
FAX: (414) 784-9509

Pioneer/Standard Electronics
120 Bishops Way #163
Brookfield 53005
Tel: (414) 784-3480

ALASKA

Avnet Computer
1400 West Benson Blvd.
Suite 400
Anchorage 99503
Tel: (907) 274-9899
FAX: (907) 277-2639

CANADA

ALBERTA

Avnet Computer
2816 21st Street Northeast
Calgary T2E 6Z2
Tel: (403) 291-3284
FAX: (403) 250-1591

Zentronics
6815 8th Street N.E., #100
Calgary T2E 7H
Tel: (403) 295-8838
FAX: (403) 295-8714

BRITISH COLUMBIA

†Hamilton/Avnet Electronics
8610 Commerce Court
Burnaby V5A 4N6
Tel: (604) 420-4101
FAX: (604) 420-5376

Zentronics
11400 Bridgeport Rd., #108
Richmond V6X 1T2
Tel: (604) 273-5575
FAX: (604) 273-2413

ONTARIO

Arrow Electronics, Inc.
36 Antares Dr., Unit 100
Nepean K2E 7W5
Tel: (613) 226-6903
FAX: (613) 723-2018

†Arrow Electronics, Inc.
1093 Meyerside, Unit 2
Mississauga L5T 1M4
Tel: (416) 670-7769
FAX: (416) 670-7781

Avnet Computer
Canada System Engineering Group
3688 Nashua Dr., Unit 6
Mississauga L4V 1M5
Tel: (416) 672-8638
FAX: (416) 677-5091

Avnet Computer
6845 Rexwood Road
Units 7–9
Mississauga L4V 1M4
Tel: (416) 672-8638
FAX: (416) 672-8650

Avnet Computer
190 Colonade Road
Nepean K2E 7J5
Tel: (613) 727-7529
FAX: (613) 226-1184

†Hamilton/Avnet Electronics
6845 Rexwood Rd., Units 3–5
Mississauga L4T 1R2
Tel: (416) 677-7432
FAX: (416) 677-0940

†Hamilton/Avnet Electronics
190 Colonade Road
Nepean K2E 7J5
Tel: (613) 226-1700
FAX: (613) 226-1184

†Zentronics
1355 Meyerside Drive
Mississauga L5T 1C9
Tel: (416) 564-9600
FAX: (416) 564-3127

†Zentronics
155 Colonade Rd., South
Unit 17
Nepean K2E 7K1
Tel: (613) 226-8840
FAX: (613) 226-6352

QUEBEC

Arrow Electronics Inc.
1100 St. Regis Blvd.
Dorval H9P 2T5
Tel: (514) 421-7411
FAX: (514) 421-7430

Arrow Electronics, Inc.
500 Boul. St-Jean-Baptiste Ave.
Quebec H2E 5R9
Tel: (418) 871-7500
FAX: (418) 871-6816

Avnet Computer
2795 Rue Halpern
St. Laurent H4S 1P8
Tel: (514) 335-2483
FAX: (514) 335-2481

†Hamilton/Avnet Electronics
2795 Halpern
St. Laurent H4S 1P8
Tel: (514) 335-1000
FAX: (514) 335-2481

†Zentronics
520 McCaffrey
St. Laurent H4T 1N3
Tel: (514) 737-9700
FAX: (514) 737-5212

†Certified VAD

EUROPEAN SALES OFFICES

ILAND

tel Finland OY
.uosilantie 2
/0390 Helsinki
Tel. (358) 0 544 644
FAX: (358) 0 544 030

FRANCE

Intel Corporation S.A.R.L.
1, Rue Edison-BP 303
78054 St. Quentin-en-Yvelines
Cedex
Tel: (33) (1) 30 57 70 00
FAX: (33) (1) 30 64 60 32

GERMANY

Intel GmbH
Dornacher Strasse 1
8016 Feldkirchen bei Muenchen
Tel: (49) 089/90992-0
FAX: (49) 089/9043948

ISRAEL

Intel Semiconductor Ltd.
Atidim Industrial Park-Neve Sharet
P.O. Box 43202
Tel-Aviv 61430
Tel: (972) 03 498080
FAX: (972) 03 491870

ITALY

Intel Corporation Italia S.p.A.
Milanofiori Palazzo E
20094 Assago
Milano
Tel: (39) (02) 89200950
FAX: (39) (2) 3498464

NETHERLANDS

Intel Semiconductor B.V.
Postbus 84130
3009 CC Rotterdam
Tel: (31) 10 407 11 11
FAX: (31) 10 455 4688

SPAIN

Intel Iberia S.A.
Zubaran, 28
28010 Madrid
Tel: (34) 308 25 52
FAX: (34) 410 7570

SWEDEN

Intel Sweden A.B.
Dalvagen 24
171 36 Solna
Tel: (46) 8 734 01 00
FAX: (46) 8 278085

UNITED KINGDOM

Intel Corporation (U.K.) Ltd.
Pipers Way
Swindon, Wiltshire SN3 1RJ
Tel: (44) (0793) 696000
FAX: (44) (0793) 641440

EUROPEAN DISTRIBUTORS/REPRESENTATIVES

AUSTRIA

Bacher Electronics GmbH
Rotenmuehlgasse 26
A-1120 Wien
Tel: 43 222 81356460
FAX: 43 222 834276

BELGIUM

Inelco Belgium S.A.
Oorlogskruisenlaan 94
B-1120 Bruxelles
Tel: 32 2 244 2811
FAX: 32 2 216 4301

FRANCE

Almex
48, Rue de l'Aubepine
B.P. 102
92164 Antony Cedex
Tel: 33 1 4096 5400
FAX: 33 1 4666 6028

Lex Electronics
Silic 585
60 Rue des Gemeaux
94663 Rungis Cedex
Tel: 33 1 4978 4978
FAX: 33 1 4978 0596

Metrologie
Tour d'Asnieres
4, Avenue Laurent Cely
92606 Asnieres Cedex
Tel: 33 1 4790 6240
FAX: 33 1 4790 5947

Tekelec-Airtronic
Cite Des Bruyeres
Rue Carle Vernet
BP 2
92310 Sevres
Tel: 33 1 4623 2425
FAX: 33 1 4507 2191

GERMANY

E2000 Vertriebs-AG
Stahlgruberring 12
8000 Muenchen 82
Tel: 49 89 420010
FAX: 49 89 42001209

Jermyn GmbH
Im Dachsstueck 9
6250 Limburg
Tel: 49 6431 5080
FAX: 49 6431 508289

Metrologie GmbH
Steinerstrasse 15
8000 Muenchen 70
Tel: 49 89 724470
FAX: 49 89 72447111

Proelectron Vertriebs GmbH
Max-Planck-Strasse 1-3
6072 Dreieich
Tel: 49 6103 304343
FAX: 49 6103 304425

Rein Electronik GmbH
Loetscher Weg 66
4054 Nettetal 1
Tel: 49 2153 7330
FAX: 49 2153 733513

GREECE

Pouliadis Associates Corp.
5 Koumbari Street
Kolonaki Square
10674 Athens
Tel: 30 1 360 3741
FAX: 30 1 360 7501

IRELAND

Micro Marketing
Tany Hall
Eglinton Terrace
Dundrum
Dublin
Tel: 0001 989 400
FAX: 0001 989 8282

ISRAEL

Eastronics Ltd.
Rozanis 11
P.O.B. 39300
Tel Baruch
Tel-Aviv 61392
Tel: 972 3 475151
FAX: 972 3 475125

ITALY

Celdis Spa
Via F.11i Gracchi 36
20092 Cinisello Balsamo
Milano
Tel: 39 2 66012003
FAX: 39 2 6182433

Intesi Div. Della Deutsche
Divisione ITT
Industries GmbH
P.I. 06550110156
Milanofiori Palazzo E5
20094 Assago (Milano)
Tel: 39 2 824701
FAX: 39 2 8242631

Lasi Elettronica S.p.A.
P.I. 00839000155
Viale Fulvio Testi, N.280
20126 Milano
Tel: 39 2 66101370
FAX: 39 2 66101385

Telcom s.r.l. – Divisione MDS
Via Trombetta
Zona Marconi
Strada Cassanese
Segrate – Milano
Tel: 39 2 2138010
FAX: 39 2 216061

NETHERLANDS

Koning en Hartman B.V.
Energieweg 1
2627 AP Delft
Tel: 31 15 609 906
FAX: 31 15 619 194

PORTUGAL

ATD Electronica LDA
Rua Dr. Faria de
Vasconcelos, 3a
1900 Lisboa
Tel: 351 1 8472200
FAX: 351 1 8472197

SPAIN

ATD Electronica
Plaza Ciudad de Viena, 6
28040 Madrid
Tel: 34 1 534 4000/09
FAX: 34 1 534 7663

Metrologia Iberica
Ctra De Fuencarral N.80
28100 Alcobendas
Madrid
Tel: 34 1 6538611
FAX: 34 1 6517549

SCANDINAVIA

OY Fintronic AB
Heikkilantie 2a
SF-00210 Helsinki
Tel: 358 0 6926022
FAX: 358 0 6821251

ITT Multikomponent A/S
Naverland 29
DK-2600 Glostrup
Denmark
Tel: 010 45 42 451822
FAX: 010 45 42 457624

Nordisk Elektronik A/S
Postboks 122
Smedsvingen 4
N-1364 Hvalstad
Norway
Tel: 47 2 846210
FAX: 47 2 846545

Nordisk Electronik AB
Box 36
Torshamnsgatan 39
S-16493 Kista
Sweden
Tel: 46 8 7034630
FAX: 46 8 7039845

SWITZERLAND

Industrade A.G.
Hertistrasse 31
CH-8304 Wallisellen
Tel: 41 1 8328111
FAX: 41 1 8307550

TURKEY

EMPA
80050 Sishane
Refik Saydam Cad No. 89/5
Istanbul
Tel: 90 1 143 6212
FAX: 90 1 143 6547

UNITED KINGDOM

Access Elect Comp Ltd.
Jubilee House
Jubilee Road
Letchworth
Hertfordshire
SG6 1QH
Tel: 0462 480888
FAX: 0462 682467

Bytech Components Ltd.
12a Cedarwood
Chineham Business Park
Crockford Lane
Basingstoke
Hants RG12 1RW
Tel: 0256 707107
FAX: 0256 707162

Bytech Systems
Unit 3
The Western Centre
Western Road
Bracknell
Berks RG12 1RW
Tel: 0344 55333
FAX: 0344 867270

Metrologie
Rapid House
Oxford Road
High Wycombe
Bucks
Herts HP11 2EE
Tel: 0494 474147
FAX: 0494 452144

Jermyn
Vestry Estate
Otford Road
Sevenoaks
Kent TN14 5EU
Tel: 0732 450144
FAX: 0732 451251

MMD
3 Bennet Court
Bennet Road
Reading
Berkshire RG2 0QX
Tel: 0734 313232
FAX: 0734 313255

Rapid Silicon
3 Bennet Court
Bennet Road
Reading
Berks RG2 0QX
Tel: 0734 752266
FAX: 0734 312728

Metro Systems
Rapid House
Oxford Road
High Wycombe
Bucks HP11 2EE
Tel: 0494 474171
FAX: 0494 21860

YUGOSLAVIA

H.R. Microelectronics Corp.
2005 de la Cruz Blvd.
Suite 220
Santa Clara, CA 95050
U.S.A.
Tel: (408) 988-0286
FAX: (408) 988-0306

INTERNATIONAL SALES OFFICES

AUSTRALIA

Intel Australia Pty. Ltd.
Unit 13
Allambie Grove Business Park
25 Frenchs Forest Road East
Frenchs Forest, NSW, 2086
Sydney
Tel: 61-2-975-3300
FAX: 61-2-975-3375

Intel Australia Pty. Ltd.
711 High Street
1st Floor
East Kw. Vic., 3102
Melbourne
Tel: 61-3-810-2141
FAX: 61-3-819 7200

BRAZIL

Intel Semicondutores do Brazil LTDA
Avenida Paulista, 1159-CJS 404/405
01311 - Sao Paulo - S.P.
Tel: 55-11-287-5899
TLX: 11-37-557-ISDB
FAX: 55-11-287-5119

CHINA/HONG KONG

Intel PRC Corporation
15/F, Office 1, Citic Bldg.
Jian Guo Men Wai Street
Beijing, PRC
Tel: (1) 500-4850
TLX: 22947 INTEL CN
FAX: (1) 500-2953

Intel Semiconductor Ltd.*
10/F East Tower
Bond Center
Queensway, Central
Hong Kong
Tel: (852) 844-4555
FAX: (852) 868-1989

INDIA

Intel Asia Electronics, Inc.
4/2, Samrah Plaza
St. Mark's Road
Bangalore 560001
Tel: 91-812-215773
TLX: 953-845-2646 INTEL IN
FAX: 091-812-215067

JAPAN

Intel Japan K.K.
5-6 Tokodai, Tsukuba-shi
Ibaraki, 300-26
Tel: 0298-47-8511
FAX: 0298-47-8450

Intel Japan K.K.*
Hachioji ON Bldg.
4-7-14 Myojin-machi
Hachioji-shi, Tokyo 192
Tel: 0426-48-8770
FAX: 0426-48-8775

Intel Japan K.K.*
Bldg. Kumagaya
2-69 Hon-cho
Kumagaya-shi, Saitama 360
Tel: 0485-24-6871
FAX: 0485-24-7518

Intel Japan K.K.*
Kawa-asa Bldg.
2-11-5 Shin-Yokohama
Kohoku-ku, Yokohama-shi
Kanagawa, 222
Tel: 045-474-7661
FAX: 045-471-4394

Intel Japan K.K.*
Ryokuchi-Eki Bldg.
2-4-1 Terauchi
Toyonaka-shi, Osaka 560
Tel: 06-863-1091
FAX: 06-863-1084

Intel Japan K.K.
Shinmaru Bldg.
1-5-1 Marunouchi
Chiyoda-ku, Tokyo 100
Tel: 03-3201-3621
FAX: 03-3201-6850

Intel Japan K.K.
Green Bldg.
1-16-20 Nishiki
Naka-ku, Nagoya-shi
Aichi 460
Tel: 052-204-1261
FAX: 052-204-1285

KOREA

Intel Korea, Ltd.
16th Floor, Life Bldg.
61 Yoido-dong, Youngdeungpo-Ku
Seoul 150-010
Tel: (2) 784-8186
FAX: (2) 784-8096

SINGAPORE

Intel Singapore Technology, Ltd.
101 Thomson Road #08-03/06
United Square
Singapore 1130
Tel: (65) 250-7811
FAX: (65) 250-9256

TAIWAN

Intel Technology Far East Ltd.
Taiwan Branch Office
8th Floor, No. 205
Bank Tower Bldg.
Tung Hua N. Road
Taipei
Tel: 886-2-5144202
FAX: 886-2-717-2455

INTERNATIONAL DISTRIBUTORS/REPRESENTATIVES

ARGENTINA

Dafsys S.R.L.
Chacabuco, 90-6 Piso
1069-Buenos Aires
Tel: 54-1-34-7726
FAX: 54-1-34-1871

AUSTRALIA

Email Electronics
15-17 Hume Street
Huntingdale, 3166
Tel: 011-61-3-544-8244
TLX: AA 30895
FAX: 011-61-3-543-8179

NSD-Australia
205 Middleborough Rd.
Box Hill, Victoria 3128
Tel: 03 8900970
FAX: 03 8990819

BRAZIL

Microlinear
Largo do Arouche, 24
01219 Sao Paulo, SP
Tel: 5511-220-2215
FAX: 5511-220-5750

CHILE

Sisteco
Vecinal 40 – Las Condes
Santiago
Tel: 562-234-1644
FAX: 562-233-9895

CHINA/HONG KONG

Novel Precision Machinery Co., Ltd.
Room 728 Trade Square
681 Cheung Sha Wan Road
Kowloon, Hong Kong
Tel: (852) 360-8999
TWX: 32032 NVTNL HX
FAX: (852) 725-3695

GUATEMALA

Abinitio
11 Calle 2 – Zona 9
Guatemala City
Tel: 5022-32-4104
FAX: 5022-32-4123

INDIA

Micronic Devices
Arun Complex
No. 65 D.V.G. Road
Basavanagudi
Bangalore 560 004
Tel: 011-91-812-600-631
011-91-812-611-365
TLX: 9538458332 MDBG

Micronic Devices
No. 516 5th Floor
Swastik Chambers
Sion, Trombay Road
Chembur
Bombay 400 071
TLX: 9531 171447 MDEV

Micronic Devices
25/8, 1st Floor
Bada Bazaar Marg
Old Rajinder Nagar
New Delhi 110 060
Tel: 011-91-11-5723509
011-91-11-589771
TLX: 031-63253 MDND IN

Micronic Devices
6-3-348/12A Dwarakapuri Colony
Hyderabad 500 482
Tel: 011-91-842-226748

S&S Corporation
1587 Kooser Road
San Jose, CA 95118
Tel: (408) 978-6216
TLX: 820281
FAX: (408) 978-8635

JAMAICA

MC Systems
10-12 Grenada Crescent
Kingston 5
Tel: (809) 929-2638
(809) 926-0188
FAX: (809) 926-0104

JAPAN

Asahi Electronics Co. Ltd.
KMM Bldg. 2-14-1 Asano
Kokurakita-ku
Kitakyushu-shi 802
Tel: 093-511-6471
FAX: 093-551-7861

CTC Components Systems Co., Ltd.
4-8-1 Dobashi, Miyamae-ku
Kawasaki-shi, Kanagawa 213
Tel: 044-852-5121
FAX: 044-877-4268

Dia Semicon Systems, Inc.
Flower Hill Shinmachi Higashi-kan
1-23 Shinmachi, Setagaya-ku
Tokyo 154
Tel: 03-3439-1600
FAX: 03-3439-1601

Okaya Koki
2-4-18 Sakae
Naka-ku, Nagoya-shi 460
Tel: 052-204-8315
FAX: 052-204-8380

Ryoyo Electro Corp.
Konwa Bldg.
1-12-22 Tsukiji
Chuo-ku, Tokyo 104
Tel: 03-3546-5011
FAX: 03-3546-5044

KOREA

J-Tek Corporation
Dong Sung Bldg. 9/F
158-24, Samsung-Dong, Kangnam-Ku
Seoul 135-090
Tel: (822) 557-8039
FAX: (822) 557-8304

Samsung Electronics
Samsung Main Bldg.
150 Taepyung-Ro-2KA, Chung-Ku
Seoul 100-102
C.P.O. Box 8780
Tel: (822) 751-3680
TWX: KORSST K 27970
FAX: (822) 753-9065

MEXICO

PSI S.A. de C.V.
Fco. Villa esq. Ajusco s/n
Cuernavaca, MOR 62130
Tel: 52-73-13-9412
52-73-17-5340
FAX: 52-73-17-5333

NEW ZEALAND

Email Electronics
36 Olive Road
Penrose, Auckland
Tel: 011-64-9-591-155
FAX: 011-64-9-592-681

SAUDI ARABIA

AAE Systems, Inc.
642 N. Pastoria Ave.
Sunnyvale, CA 94086
U.S.A.
Tel: (408) 732-1710
FAX: (408) 732-3095
TLX: 494-3405 AAE SYS

SINGAPORE

Electronic Resources Pte, Ltd.
17 Harvey Road
#03-01 Singapore 1336
Tel: (65) 283-0888
TWX: RS 56541 ERS
FAX: (65) 289-5327

SOUTH AFRICA

Electronic Building Elements
178 Erasmus St. (off Watermeyet St.)
Meyerspark, Pretoria, 0184
Tel: 011-2712-803-7680
FAX: 011-2712-803-8294

TAIWAN

Micro Electronics Corporation
12th Floor, Section 3
285 Nanking East Road
Taipei, R.O.C.
Tel: (886) 2-7198419
FAX: (886) 2-7197916

Acer Sertek Inc.
15th Floor, Section 2
Chien Kuo North Rd.
Taipei 18479 R.O.C.
Tel: 886-2-501-0055
TWX: 23756 SERTEK
FAX: (886) 2-5012521

URUGUAY

Interfase
Zabala 1378
11000 Montevideo
Tel: 5982-96-0490
5982-96-1143
FAX: 5982-96-2965

VENEZUELA

Unixel C.A.
4 Transversal de Monte Cristo
Edf. AXXA, Piso 1, of. 1&2
Centro Empresarial Boleita
Caracas
Tel: 582-238-6082
FAX: 582-238-1816

*Field Application Location

NORTH AMERICAN SERVICE OFFICES

ALASKA

Intel Corp.
c/o TransAlaska Network
1515 Lore Rd.
Anchorage 99507
Tel: (907) 522-1776

Intel Corp.
c/o TransAlaska Data Systems
c/o GCI Operations
520 Fifth Ave., Suite 407
Fairbanks 99701
Tel: (907) 452-6264

ARIZONA

*Intel Corp.
410 North 44th Street
Suite 500
Phoenix 85008
Tel: (602) 231-0386
FAX: (602) 244-0446

*Intel Corp.
500 E. Fry Blvd., Suite M-15
Sierra Vista 85635
Tel: (602) 459-5010

ARKANSAS

Intel Corp.
c/o Federal Express
1500 West Park Drive
Little Rock 72204

CALIFORNIA

*Intel Corp.
21515 Vanowen St., Ste. 116
Canoga Park 91303
Tel: (818) 704-8500

*Intel Corp.
300 N. Continental Blvd.
Suite 100
El Segundo 90245
Tel: (213) 640-6040

*Intel Corp.
1900 Prairie City Rd.
Folsom 95630-9597
Tel: (916) 351-6143

*Intel Corp.
9665 Chesapeake Dr., Suite 325
San Diego 92123
Tel: (619) 292-8086

**Intel Corp.
400 N. Tustin Avenue
Suite 450
Santa Ana 92705
Tel: (714) 835-9642

**Intel Corp.
2700 San Tomas Exp., 1st Floor
Santa Clara 95051
Tel: (408) 970-1747

COLORADO

*Intel Corp.
600 S. Cherry St., Suite 700
Denver 80222
Tel: (303) 321-8086

CONNECTICUT

*Intel Corp.
301 Lee Farm Corporate Park
83 Wooster Heights Rd.
Danbury 06811
Tel: (203) 748-3130

FLORIDA

**Intel Corp.
800 Fairway Dr., Suite 160
Deerfield Beach 33441
Tel: (305) 421-0506
FAX: (305) 421-2444

*Intel Corp.
5850 T.G. Lee Blvd., Ste. 340
Orlando 32822
Tel: (407) 240-8000

GEORGIA

*Intel Corp.
20 Technology Park, Suite 150
Norcross 30092
Tel: (404) 449-0541

5523 Theresa Street
Columbus 31907

HAWAII

**Intel Corp.
Honolulu 96820
Tel: (808) 847-6738

ILLINOIS

**†Intel Corp.
Woodfield Corp. Center III
300 N. Martingale Rd., Ste. 400
Schaumburg 60173
Tel: (708) 605-8031

INDIANA

*Intel Corp.
8910 Purdue Rd., Ste. 350
Indianapolis 46268
Tel: (317) 875-0623

KANSAS

*Intel Corp.
10985 Cody, Suite 140
Overland Park 66210
Tel: (913) 345-2727

KENTUCKY

Intel Corp.
133 Walton Ave., Office 1A
Lexington 40508
Tel: (606) 255-2957

Intel Corp.
896 Hillcrest Road, Apt. A
Radcliff 40160 (Louisville)

LOUISIANA

Hammond 70401
(serviced from Jackson, MS)

MARYLAND

**Intel Corp.
10010 Junction Dr., Suite 200
Annapolis Junction 20701
Tel: (301) 206-2860

MASSACHUSETTS

**Intel Corp.
Westford Corp. Center
3 Carlisle Rd., 2nd Floor
Westford 01886
Tel: (508) 692-0960

MICHIGAN

*Intel Corp.
7071 Orchard Lake Rd., Ste. 100
West Bloomfield 48322
Tel: (313) 851-8905

MINNESOTA

*Intel Corp.
3500 W. 80th St., Suite 360
Bloomington 55431
Tel: (612) 835-6722

MISSISSIPPI

Intel Corp.
c/o Compu-Care
2001 Airport Road, Suite 205F
Jackson 39208
Tel: (601) 932-6275

MISSOURI

*Intel Corp.
3300 Rider Trail South
Suite 170
Earth City 63045
Tel: (314) 291-1990

Intel Corp.
Route 2, Box 221
Smithville 64089
Tel: (913) 345-2727

NEW JERSEY

**Intel Corp.
300 Sylvan Avenue
Englewood Cliffs 07632
Tel: (201) 567-0821

*Intel Corp.
Lincroft Office Center
125 Half Mile Road
Red Bank 07701
Tel: (908) 747-2233

NEW MEXICO

Intel Corp.
Rio Rancho 1
4100 Sara Road
Rio Rancho 87124-1025
(near Albuquerque)
Tel: (505) 893-7000

NEW YORK

*Intel Corp.
2950 Expressway Dr. South
Suite 130
Islandia 11722
Tel: (516) 231-3300

Intel Corp.
300 Westage Business Center
Suite 230
Fishkill 12524
Tel: (914) 897-3860

Intel Corp.
5858 East Molloy Road
Syracuse 13211
Tel: (315) 454-0576

NORTH CAROLINA

*Intel Corp.
5800 Executive Center Drive
Suite 105
Charlotte 28212
Tel: (704) 568-8966

**Intel Corp.
5540 Centerview Dr., Suite 215
Raleigh 27606
Tel: (919) 851-9537

OHIO

**Intel Corp.
3401 Park Center Dr., Ste. 220
Dayton 45414
Tel: (513) 890-5350

*Intel Corp.
25700 Science Park Dr., Ste. 100
Beachwood 44122
Tel: (216) 464-2736

OREGON

**Intel Corp.
15254 N.W. Greenbrier Pkwy.
Building B
Beaverton 97006
Tel: (503) 645-8051

PENNSYLVANIA

*†Intel Corp.
925 Harvest Drive
Suite 200
Blue Bell 19422
Tel: (215) 641-1000
1-800-468-3548
FAX: (215) 641-0785

**†Intel Corp.
400 Penn Center Blvd., Ste. 610
Pittsburgh 15235
Tel: (412) 823-4970

*Intel Corp.
1513 Cedar Cliff Dr.
Camp Hill 17011
Tel: (717) 761-0860

PUERTO RICO

Intel Corp.
South Industrial Park
P.O. Box 910
Las Piedras 00671
Tel: (809) 733-8616

TEXAS

**Intel Corp.
Westech 360, Suite 4230
8911 N. Capitol of Texas Hwy.
Austin 78752-1239
Tel : (512) 794-8086

**†Intel Corp.
12000 Ford Rd., Suite 401
Dallas 75234
Tel: (214) 241-8087

**Intel Corp.
7322 SW Freeway, Suite 1490
Houston 77074
Tel: (713) 988-8086

UTAH

Intel Corp.
428 East 6400 South
Suite 104
Murray 84107
Tel: (801) 263-8051
FAX: (801) 268-1457

VIRGINIA

*Intel Corp.
9030 Stony Point Pkwy.
Suite 360
Richmond 23235
Tel: (804) 330-9393

WASHINGTON

**Intel Corp.
155 108th Avenue N.E., Ste. 386
Bellevue 98004
Tel: (206) 453-8086

CANADA

ONTARIO

**Intel Semiconductor of
Canada, Ltd.
2650 Queensview Dr., Ste. 250
Ottawa K2B 8H6
Tel: (613) 829-9714

**Intel Semiconductor of
Canada, Ltd.
190 Attwell Dr., Ste. 102
Rexdale (Toronto) M9W 6H8
Tel: (416) 675-2105

QUEBEC

**Intel Semiconductor of
Canada, Ltd.
1 Rue Holiday
Suite 115
Tour East
Pt. Claire H9R 5N3
Tel: (514) 694-9130
FAX: 514-694-0064

CUSTOMER TRAINING CENTERS

ARIZONA

2402 W. Beardsley Road
Phoenix 85027
Tel: (602) 869-4288
1-800-468-3548

SYSTEMS ENGINEERING OFFICES

MINNESOTA

3500 W. 80th Street
Suite 360
Bloomington 55431
Tel: (612) 835-6722

NEW YORK

2950 Expressway Dr., South
Islandia 11722
Tel: (506) 231-3300

*Carry-in locations
**Carry-in/mail-in locations

Multimedia and Supercomputing Processors

Intel Corporation's Multimedia and Supercomputing Components Group products enrich computerized information and exchange technologies in imaginative new ways never before possible. To learn more about Intel's problem-solving MSCG products: The i750® video processor, and the i860™ and i960™ microprocessor families, you will want to read this publication.

Printed in USA/0192/35K/RRD/DM
Order Number 272084-001

ISBN# 1-55512-149-7