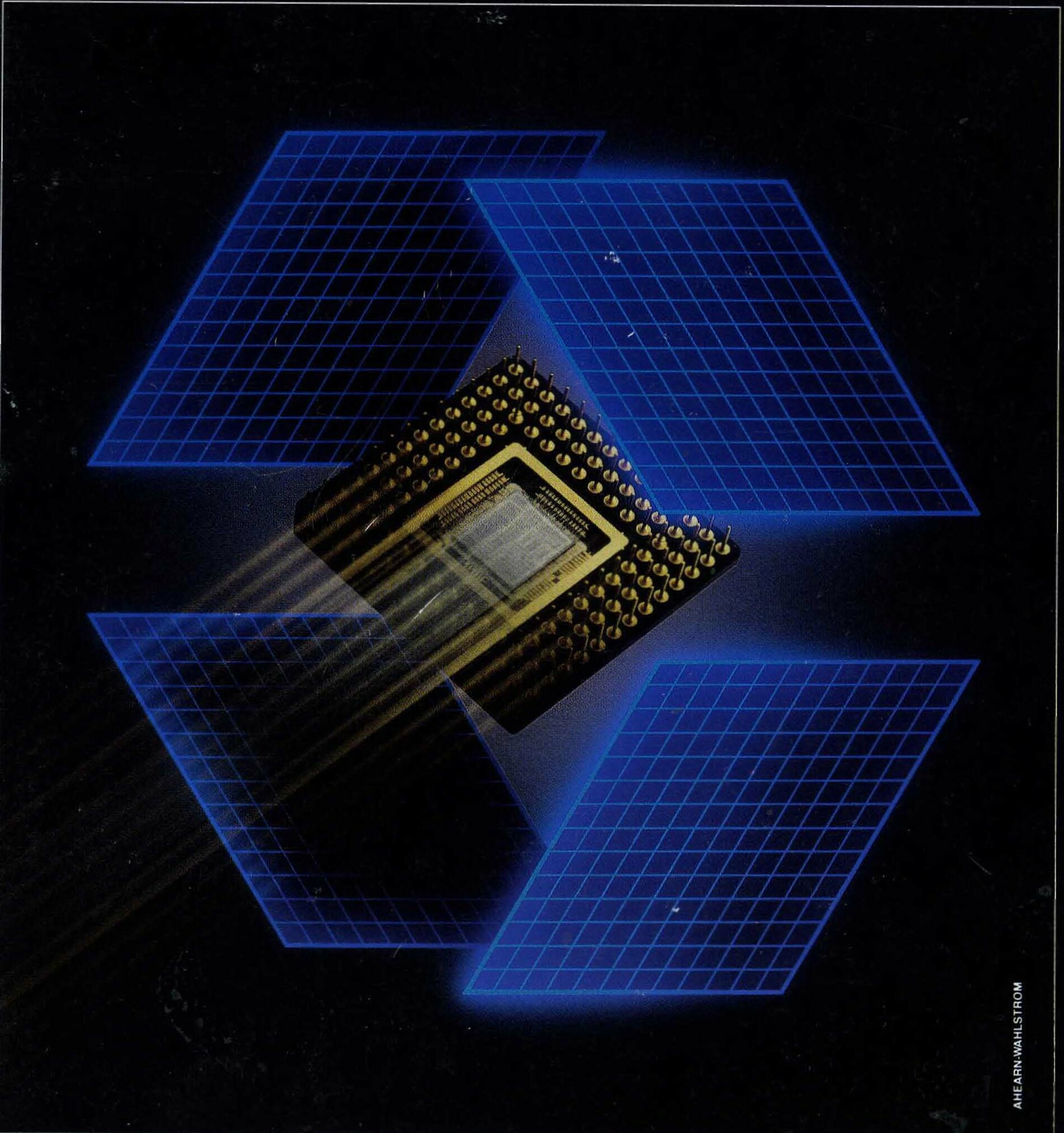
80386 Hardware Reference Manual


AHEARN-WAHLSTROM

Order Number: 231732-002

LITERATURE

To order Intel literature write or call:

Intel Literature Sales
P.O. Box 58130
Santa Clara, CA 95052-8130

Intel Literature:
(800) 548-4725

Use the order blank on the facing page or call our **Toll Free** Number listed above to order literature. Remember to add your local sales tax and a 10% postage charge for U.S. and Canada customers, 20% for outside U.S. customers. Prices subject to change.

1987 HANDBOOKS

Product line handbooks contain data sheets, application notes, article reprints and other design information.

NAME	ORDER NUMBER	*PRICE IN U.S. DOLLARS
COMPLETE SET OF 9 HANDBOOKS Save $50.00 off the retail price of $175.00	231003	$125.00
MEMORY COMPONENTS HANDBOOK	210830	$18.00
MICROCOMMUNICATIONS HANDBOOK	231658	$20.00
EMBEDDED CONTROLLER HANDBOOK (includes Microcontrollers and 8085, 80186, 80188)	210918	$18.00
MICROPROCESSOR AND PERIPHERAL HANDBOOK (2 Volume Set)	230843	$25.00
DEVELOPMENT TOOLS HANDBOOK	210940	$18.00
DOS DEVELOPMENT SOFTWARE CATALOG	280199	N/C
OEM BOARDS AND SYSTEMS HANDBOOK	280407	$18.00
MILITARY HANDBOOK	210461	$18.00
COMPONENTS QUALITY/RELIABILITY HANDBOOK	210997	$20.00
SYSTEMS QUALITY/RELIABILITY HANDBOOK	231762	$20.00
PROGRAMMABLE LOGIC HANDBOOK (Not included in Handbook Set)	296083	$18.00
PRODUCT GUIDE Overview of Intel's complete product lines	210846	N/C
LITERATURE PRICE LIST List of Intel Literature	210620	N/C
INTEL PACKAGING OUTLINES AND DIMENSIONS Packaging types, number of leads, etc.	231369	N/C

*These prices are for the U.S. and Canada only. In Europe and other international locations, please contact your local Intel Sales Office or Distributor for literature prices.

intel

LITERATURE SALES ORDER FORM

NAME: ____

COMPANY: ____

ADDRESS: ____

CITY: ____ STATE: ____ ZIP: ____

COUNTRY: ____

PHONE NO.: ( ) ____

ORDER NO.	TITLE	QTY.		PRICE		TOTAL
			×		=	
			×		=	
			×		=	
			×		=	
			×		=	
			×		=	
			×		=	
			×		=	
			×		=	
			×		=	

Subtotal ____

Must Add Your Local Sales Tax ____

Must add appropriate postage to subtotal (10% U.S. and Canada, 20% all other) → Postage ____

Total ____

Pay by Visa, MasterCard, American Express, Check, Money Order, or company purchase order payable to Intel Literature Sales. Allow 2-4 weeks for delivery.

☐ Visa ☐ MasterCard ☐ American Express Expiration Date ____

Account No. ____

Signature: ____

Mail To:

Intel Literature Sales
P.O. Box 58130
Santa Clara, CA
95052-8130

International Customers outside the U.S. and Canada should contact their local Intel Sales Office or Distributor listed in the back of most Intel literature.

Call Toll Free: (800) 548-4725 for phone orders

Prices good until 12/31/87.

Source HB

Mail To: Intel Literature Sales
P.O. Box 58130
Santa Clara, CA 95052-8130

intel®

80386
HARDWARE REFERENCE MANUAL

1987

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:

Above, BITBUS, COMMputer, CREDIT, Data Pipeline, FASTPATH, Genius, i, î, ICE, iCEL, iCS, iDBP, iDIS, I^{2}ICE, iLBX, i_m, iMDDX, iMMX, Inboard, Insite, Intel, int$_e$l, int$_e$lBOS, Intel Certified, Intelevision, int$_e$ligent Identifier, int$_e$ligent Programming, Intellec, Intellink, iOSP, iPDS, iPSC, iRMK, iRMX, iSBC, iSBX, iSDM, iSXM, KEPROM, Library Manager, MAPNET, MCS, Megachassis, MICROMAINFRAME, MULTIBUS, MULTICHANNEL, MULTIMODULE, MultiSERVER, ONCE, OpenNET, OTP, PC BUBBLE, Plug-A-Bubble, PROMPT, Promware, QUEST, QueX, Quick-Pulse Programming, Ripplemode, RMX/80, RUPI, Seamless, SLD, SugarCube, SupportNET, UPI, and VLSiCEL, and the combination of ICE, iCS, iRMX, iSBC, iSBX, iSXM, MCS, or UPI and a numerical suffix, 4-SITE.

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.

*MULTIBUS is a patented Intel bus.

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

Intel Corporation
Literature Distribution
Mail Stop SC6-59
3065 Bowers Avenue
Santa Clara, CA 95051

©INTEL CORPORATION 1987

CG-5/26/87

PREFACE

The Intel 80386 is a high-performance 32-bit microprocessor. This manual provides complete hardware reference information for 80386 system designs. It is written for system engineers and hardware designers who understand the operating principles of microprocessors and microcomputer systems. Readers of this manual should be familiar with the information in the *Introduction to the 80386* (Intel publication Order Number 231252).

RELATED PUBLICATIONS

In this manual, the 80386 is presented from a hardware perspective. Information on the software architecture, instruction set, and programming of the 80386 can be found in these related Intel publications:

- *80386 Programmer's Reference Manual*, Order Number 230985
- *80386 System Software Writer's Guide*, Order Number 231499
- *80386 Data Sheet*, Order Number 231630

The *80386 Data Sheet* contains device specifications for the 80386. Always consult the most recent version of this publication for specific 80386 parameter values.

Detailed device specifications on 80386 family components can be found in the following publications:

- *80387 Data Sheet*, Order Number 231920
- *82380 Data Sheet*, Order Number 290128

Together with the *80386 Hardware Reference Manual*, these publications provide a complete description of the 80386 system for hardware designers, software engineers, and all users of 80386 systems.

ORGANIZATION OF THIS MANUAL

The information in this manual is divided into 12 chapters and three appendices. The material begins with a description of the 80386 microprocessor and continues with discussions of hardware design information needed to implement 80386 system designs.

- Chapter 1, "System Overview." This chapter provides an overview of the 80386 and its supporting devices.
- Chapter 2, "Internal Architecture." This chapter describes the internal architecture of the 80386.
- Chapter 3, "Local Bus Interface." This chapter discusses the 80386 local bus interface. This chapter includes 80386 signal descriptions, memory and I/O organization, and local bus interface guidelines.

- Chapter 4, "Performance Considerations." This chapter explores the factors that affect the performance of an 80386 system.
- Chapter 5, "Coprocessor Hardware Interface." This chapter describes the interface between the 80386 and the 80287 and 80387 Numeric Coprocessors. Each of these coprocessors expands the floating-point numerical processing capabilities of the 80386.
- Chapter 6, "Memory Interfacing." This chapter discusses techniques for designing memory subsystems for the 80386.
- Chapter 7, "Cache Subsystems." This chapter describes cache memory subsystems, which provide higher performance at lower relative cost.
- Chapter 8, "I/O Interfacing." This chapter discusses techniques for connecting I/O devices to an 80386 system.
- Chapter 9, "MULTIBUS® I and 80386." This chapter describes the interface between an 80386 system and the Intel MULTIBUS I multi-master system bus.
- Chapter 10, "MULTIBUS® II and 80386." This chapter describes the interface between an 80386 system and the Intel MULTIBUS II multi-master system bus.
- Chapter 11, "Physical Design and Debugging." This chapter contains recommendations for constructing and debugging 80386 systems.
- Chapter 12, "Test Capabilities." This chapter describes 80386 test procedures.
- Appendix A contains descriptions of the components of the basic memory interface described in Chapter 6.
- Appendix B contains descriptions of the components of the 80387 emulator described in Chapter 5.
- Appendix C contains descriptions of the components of the dynamic RAM subsystem described in Chapter 6.

TABLE OF CONTENTS

CHAPTER 1 Page

SYSTEM OVERVIEW

1.1 Microprocessor 1-1
1.2 Coprocessors 1-3
1.3 Integrated System Peripheral 1-4
1.4 Cache Controller 1-5
1.5 Clock Generator 1-5
1.6 8086/80286 Family Components 1-5

CHAPTER 2

INTERNAL ARCHITECTURE

2.1 Bus Interface Unit 2-2
2.2 Code Prefetch Unit 2-3
2.3 Instruction Decode Unit 2-3
2.4 Execution Unit 2-3
2.5 Segmentation Unit 2-4
2.6 Paging Unit 2-4

CHAPTER 3

LOCAL BUS INTERFACE

3.1 Bus Operations 3-2
3.1.1 Bus States 3-4
3.1.2 Address Pipelining 3-5
3.1.3 32-Bit Data Bus Transfers and Operand Alignment 3-5
3.1.4 Read Cycle 3-10
3.1.5 Write Cycle 3-13
3.1.6 Pipelined Address Cycle 3-14
3.1.7 Interrupt Acknowledge Cycle 3-17
3.1.8 Halt/Shutdown Cycle 3-18
3.1.9 BS16 Cycle 3-19
3.1.10 16-Bit Byte Enables and Operand Alignment 3-20
3.2 Bus Timing 3-22
3.2.1 Read Cycle Timing 3-24
3.2.2 Write Cycle Timing 3-24
3.2.3 READY# Signal Timing 3-25
3.3 Clock Generation 3-26
3.3.1 82384 Clock Generator 3-26
3.3.2 Clock Timing 3-26
3.3.3 Crystal Oscillator Clock Generator 3-27
3.4 Interrupts 3-29
3.4.1 Non-Maskable Interrupt (NMI) 3-30
3.4.2 Maskable Interrupt (INTR) 3-31
3.4.3 Interrupt Latency 3-31

Page

3.5 Bus Lock 3-32
3.5.1 Locked Cycle Activators 3-32
3.5.2 Locked Cycle Timing 3-32
3.5.3 LOCK# Signal Duration 3-33
3.6 HOLD/HLDA (Hold Acknowledge) 3-34
3.6.1 HOLD/HLDA Timing 3-34
3.6.2 HOLD Signal Latency 3-36
3.6.3 HOLD State Pin Conditions 3-36
3.7 Reset 3-37
3.7.1 RESET Timing 3-37
3.7.2 80386 Internal States 3-38
3.7.3 80386 External States 3-38

CHAPTER 4
PERFORMANCE CONSIDERATIONS

4.1 Wait States and Pipelining 4-1

CHAPTER 5
COPROCESSOR HARDWARE INTERFACE

5.1 80287 Numeric Coprocessor Interface 5-2
5.1.1 80287 Connections 5-2
5.1.2 80287 Bus Cycles 5-2
5.1.3 80287 Clock Input 5-4
5.2 80387 Numeric Coprocessor Interface 5-4
5.2.1 80387 Connections 5-4
5.2.2 80387 Bus Cycles 5-6
5.2.3 80387 Clock Input 5-6
5.3 Local Bus Activity with the 80287/80387 5-7
5.4 Designing an Upgradable 80386 System 5-8
5.4.1 80287/80387 Recognition 5-8
5.4.1.1 Hardware Recognition of the NPX 5-8
5.4.1.2 Software Recognition of the NPX 5-8
5.4.2 80387 Emulator 5-10

CHAPTER 6
MEMORY INTERFACING

6.1 Memory Speed versus Performance and Cost 6-1
6.2 Basic Memory Interface 6-1
6.2.1 PAL Devices 6-2
6.2.2 Address Latch 6-3
6.2.3 Address Decoder 6-4
6.2.4 Data Transceiver 6-5
6.2.5 Bus Control Logic 6-6

Page

6.2.6 EPROM Interface 6-9
6.2.7 SRAM Interface 6-11
6.2.8 16-Bit Interface 6-14
6.3 Dynamic RAM (DRAM) Interface 6-15
6.3.1 Interleaved Memory 6-15
6.3.2 DRAM Memory Performance 6-15
6.3.3 DRAM Controller 6-16
6.3.3.1 3-CLK DRAM Controller 6-17
6.3.3.2 2-CLK DRAM Controller 6-21
6.3.4 DRAM Design Variations 6-23
6.3.5 Refresh Cycles 6-25
6.3.5.1 Distributed Refresh 6-25
6.3.5.2 Burst Refresh 6-26
6.3.5.3 DMA Refresh using the 82380 DRAM Refresh Controller 6-26
6.3.6 Initialization 6-27
6.3.7 Timing Analysis 6-28

CHAPTER 7
CACHE SUBSYSTEMS
7.1 Introduction to Caches 7-2
7.1.1 Program Locality 7-2
7.1.2 Block Fetch 7-2
7.2 Cache Organizations 7-3
7.2.1 Fully Associative Cache 7-3
7.2.2 Direct Mapped Cache 7-4
7.2.3 Set Associative Cache 7-6
7.3 Cache Updating 7-8
7.3.1 Write-Through System 7-8
7.3.2 Buffered Write-Through System 7-8
7.3.3 Write-Back System 7-9
7.3.4 Cache Coherency 7-10
7.4 Efficiency and Performance 7-12
7.5 Cache and DMA 7-12
7.6 Cache Example 7-13
7.6.1 Example Design 7-14
7.6.2 Example Cache Memory Organization 7-14
7.6.3 Example Cache Implementation 7-15

CHAPTER 8
I/O INTERFACING
8.1 I/O Mapping versus Memory Mapping 8-1
8.2 8-Bit, 16-Bit, and 32-Bit I/O Interfaces 8-1
8.2.1 Address Decoding 8-1

Page

8.2.2 8-Bit I/O 8-2
8.2.3 16-Bit I/O 8-4
8.2.4 32-Bit I/O 8-4
8.2.5 Linear Chip Selects 8-4
8.3 Basic I/O Interface 8-4
8.3.1 Address Latch 8-5
8.3.2 Address Decoder 8-6
8.3.3 Data Transceiver 8-8
8.3.4 Bus Control Logic 8-8
8.4 Timing Analysis for I/O Operations 8-9
8.5 Basic I/O Examples 8-14
8.5.1 8274 Serial Controller 8-14
8.5.2 82380 Programmable Interrupt Controller 8-15
8.5.2.1 Cascaded Interrupt Controllers to the 82380 PIC 8-15
8.5.3 8259A Interrupt Controller 8-15
8.5.3.1 Single Interrupt Controller 8-16
8.5.3.2 Cascaded Interrupt Controllers 8-16
8.5.3.3 Handling More Than 64 Interrupts 8-17
8.6 80286-Compatible Bus Cycles 8-17
8.6.1 A0/A1 Generator 8-18
8.6.2 S0#/S1# Generator 8-18
8.6.3 Wait-State Generator 8-19
8.6.4 Bus Controller and Bus Arbiter 8-20
8.6.5 82380 Integrated System Peripheral 8-21
8.6.6 82258 ADMA Controller 8-23
8.6.6.1 82258 as Bus Master 8-25
8.6.6.2 82258 as Peripheral 8-26
8.6.7 82586 LAN Coprocessor 8-27
8.6.7.1 Dedicated CPU 8-28
8.6.7.2 Decoupled Dual-Port Memory 8-29
8.6.7.3 Coupled Dual-Port Memory 8-30
8.6.7.4 Shared Bus 8-30

CHAPTER 9
MULTIBUS® I AND 80386

9.1 MULTIBUS® I (IEEE 796) 9-1
9.2 MULTIBUS® I Interface Example 9-2
9.2.1 Address Latches and Data Transceivers 9-2
9.2.2 Address Decoder 9-5
9.2.3 Wait-State Generator 9-5
9.2.4 Bus Controller and Bus Arbiter 9-7
9.3 Timing Analysis of MULTIBUS® I Interface 9-10
9.4 82289 Bus Arbiter 9-10

Page

9.4.1 Priority Resolution 9-11
9.4.2 82289 Operating Modes 9-11
9.4.3 MULTIBUS® I Locked Cycles 9-14
9.5 Other MULTIBUS® I Design Considerations 9-14
9.5.1 Interrupt-Acknowledge on MULTIBUS® I 9-14
9.5.2 Byte Swapping during MULTIBUS® I Byte Transfers 9-16
9.5.3 Bus Timeout Function for MULTIBUS® I Accesses 9-17
9.5.4 MULTIBUS® I Power Failure Handling 9-17
9.6 iLBX™ Bus Expansion 9-18
9.7 Dual-Port RAM with MULTIBUS® I 9-19
9.7.1 Avoiding Deadlock with Dual-Port RAM 9-20

CHAPTER 10
MULTIBUS® II AND 80386

10.1 MULTIBUS® II Standard 10-1
10.2 Parallel System Bus (iPSB) 10-1
10.2.1 iPSB Interface 10-2
10.2.1.1 BAC Signals 10-4
10.2.1.2 MIC Signals 10-6
10.3 Local Bus Extension (iLBX™ II) 10-7
10.4 Serial System Bus (iSSB) 10-7

CHAPTER 11
PHYSICAL DESIGN AND DEBUGGING

11.1 Power and Ground Requirements 11-1
11.1.1 Power and Ground Planes 11-1
11.1.2 Decoupling Capacitors 11-2
11.2 High-Frequency Design Considerations 11-3
11.2.1 Line Termination 11-4
11.2.2 Interference 11-5
11.2.3 Latchup 11-7
11.3 Clock Distribution and Termination 11-7
11.4 Thermal Characteristics 11-7
11.5 Debugging Considerations 11-10
11.5.1 Hardware Debugging Features 11-10
11.5.2 Bus Interface 11-11
11.5.3 Simplest Diagnostic Program 11-11
11.5.4 Building and Debugging a System Incrementally 11-12
11.5.5 Other Simple Diagnostic Software 11-14
11.5.6 Debugging Hints 11-14

CHAPTER 12
TEST CAPABILITIES

12.1 Internal Tests 12-1
12.1.1 Automatic Self-Test 12-1
12.1.2 Translation Lookaside Buffer Tests 12-2
12.2 Board-Level Tests 12-5

APPENDIX A
LOCAL BUS CONTROL PAL DESCRIPTIONS

PAL-1 Functions A-1
PAL-2 Functions A-2
PAL Equations A-2

APPENDIX B
80387 EMULATOR PAL DESCRIPTION

APPENDIX C
DRAM PAL DESCRIPTIONS

DRAM State PAL C-1
DRAM Control PAL C-13
Refresh Interval Counter PAL C-13
Refresh Address Counter PAL C-13
Timing Parameters C-25

Figures

Figure	Title	Page
1-1	80386 System Block Diagram	1-2
2-1	Instruction Pipelining	2-1
2-2	80386 Functional Units	2-2
3-1	CLK2 and CLK Relationship	3-5
3-2	80386 Bus States Timing Example	3-6
3-3	Bus State Diagram (Does Not Include Address Pipelining)	3-7
3-4	Non-Pipelined Address and Pipelined Address Differences	3-8
3-5	Consecutive Bytes in Hardware Implementation	3-9
3-6	Address, Data Bus, and Byte Enables for 32-Bit Bus	3-9
3-7	Misaligned Transfer	3-11
3-8	Non-Pipelined Address Read Cycles	3-12
3-9	Non-Pipelined Address Write Cycles	3-15
3-10	Pipelined Address Cycles	3-16
3-11	Interrupt Acknowledge Bus Cycles	3-18
3-12	Internal NA# and BS16# Logic	3-20
3-13	32-Bit and 16-Bit Bus Cycle Timing	3-21

Figure	Title	Page
3-14	32-Bit and 16-Bit Data Addressing	3-22
3-15	Connecting 82384 to 80386	3-27
3-16	Using CLK to Determine Bus Cycle Start	3-28
3-17	Clock Generator	3-28
3-18	ADS# Synchronizer	3-29
3-19	Error Condition Caused by Unlocked Cycles	3-33
3-20	LOCK# Signal during Address Pipelining	3-34
3-21	Bus State Diagram with HOLD State	3-35
3-22	Typical RC RESET Timing Circuit	3-37
3-23	RESET, CLK, and CLK2 Timing	3-38
5-1	80386 System with 80287 Coprocessor	5-3
5-2	80386 System with 80387 Coprocessor	5-5
5-3	Pseudo-Synchronous Interface	5-7
5-4	Software Routine to Recognize the 80287	5-9
5-5	80387 Emulator Schematic	5-11
6-1	Basic Memory Interface Block Diagram	6-2
6-2	PAL Equation and Implementation	6-4
6-3	PAL Naming Conventions	6-5
6-4	Bus Control Logic	6-7
6-5	Bus Control Signal Timing	6-8
6-6	150-Nanosecond EPROM Timing Diagram	6-10
6-7	100-Nanosecond SRAM Timing Diagram	6-12
6-8	3-CLK DRAM Controller Schematic	6-18
6-9	3-CLK DRAM Controller Cycles	6-20
6-10	2-CLK DRAM Controller Schematic	6-22
6-11	2-CLK DRAM Controller Cycles	6-24
6-12	Refresh Request Generation	6-27
7-1	Cache Memory System	7-1
7-2	Fully Associative Cache Organization	7-4
7-3	Direct Mapped Cache Organization	7-5
7-4	Two-Way Set Associative Cache Organization	7-7
7-5	Stale Data Problem	7-9
7-6	Bus Watching	7-10
7-7	Hardware Transparency	7-11
7-8	Non-Cacheable Memory	7-12
7-9	Example of Cache Memory Organization	7-15
8-1	32-Bit to 8-Bit Bus Conversion	8-3
8-2	Linear Chip Selects	8-5
8-3	Basic I/O Interface Block Diagram	8-6
8-4	Basic I/O Interface Circuit	8-7
8-5	Basic I/O Timing Diagram	8-11
8-6	8274 Interface	8-14
8-7	Single 8259A Interface	8-16

Figure	Title	Page
8-8	80286-Compatible Interface	8-19
8-9	A0, A1, and BHE# Logic	8-21
8-10	S0#/S1# Generator Logic	8-22
8-11	Wait-State Generator Logic	8-22
8-12	82288 and 82289 Connections	8-23
8-13	80386/82380 Interface	8-24
8-14	HOLD and HLDA Logic for 80386-82258 Interface	8-25
8-15	82258 Slave Mode Interface	8-26
8-16	LAN Station	8-28
8-17	Decoupled Dual-Port Memory Interface	8-29
8-18	Coupled Dual-Port Memory Interface	8-30
8-19	Shared Bus Interface	8-31
9-1	80386-MULTIBUS® I Interface	9-3
9-2	MULTIBUS® I Address Latches and Data Transceivers	9-4
9-3	Wait-State Generator Logic	9-6
9-4	MULTIBUS® Arbiter and Bus Controller	9-7
9-5	MULTIBUS® I Read Cycle Timing	9-8
9-6	MULTIBUS® I Write Cycle Timing	9-9
9-7	Bus Priority Resolution	9-12
9-8	Operating Mode Configurations	9-13
9-9	Bus-Select Logic for Interrupt Acknowledge	9-16
9-10	Byte-Swapping Logic	9-18
9-11	Bus-Timeout Protection Circuit	9-19
9-12	iLBX™ Signal Generation	9-20
10-1	iPSB Bus Cycle Timing	10-3
10-2	iPSB Bus Interface	10-4
11-1	Reducing Characteristic Impedance	11-2
11-2	Circuit without Decoupling	11-2
11-3	Decoupling Chip Capacitors	11-3
11-4	Decoupling Leaded Capacitors	11-4
11-5	Series Termination	11-5
11-6	Split Termination	11-5
11-7	Avoid Closed-Loop Signal Paths	11-6
11-8	CLK2 Series Termination	11-8
11-9	CLK2 Loading	11-9
11-10	CLK2 Waveforms	11-9
11-11	4-Byte Diagnostic Program	11-12
11-12	More Complex Diagnostic Program	11-15
11-13	Object Code for Diagnostic Program	11-16
12-1	80386 Self-Test	12-2
12-2	TLB Test Registers	12-3
A-1	PAL-1 State Listings	A-3
A-2	PAL-2 State Listings	A-9

Figure	Title	Page
A-3	PAL-1 Equations	A-14
A-4	PAL-2 Equations	A-15
B-1	80387 Emulator PAL Equations	B-1
C-1	PAL Sampling Edges	C-1
C-2	3-CLK DRAM State PAL Equations	C-3
C-3	2-CLK DRAM State PAL Equations	C-8
C-4	3-CLK DRAM Control PAL Equations	C-15
C-5	2-CLK DRAM Control PAL Equations	C-17
C-6	Refresh Interval Counter PAL Equations	C-20
C-7	Refresh Address Counter PAL Equations	C-23
C-8	DRAM Circuit Timing Diagram	C-26

Tables

Table	Title	Page
1-1	80386 System Components	1-1
3-1	Summary of 80386 Signal Pins	3-3
3-2	Bus Cycle Definitions	3-4
3-3	Possible Data Transfers on 32-Bit Bus	3-10
3-4	Misaligned Data Transfers on 32-Bit Bus	3-13
3-5	Generation of BHE#, BLE#, and A1 from Byte Enables	3-23
3-6	Byte Enables during BS16 Cycles	3-23
3-7	Output Pin States during RESET	3-39
4-1	80386 Performance with Wait States and Pipelining	4-1
4-2	Performance versus Wait States and Operating Frequency	4-3
6-1	Bus Cycles Generated by Bus Controller	6-6
6-2	DRAM Memory Performance	6-16
6-3	Designs for Six DRAM Types	6-28
7-1	Cache Hit Rates	7-13
8-1	Data Lines for 8-Bit I/O Addresses	8-2
8-2	Timings for Peripherals using Basic I/O Interface	8-13
8-3	A0, A1, and BHE# Truth Table	8-20
9-1	MULTIBUS® I Timing Parameters	9-10
C-1	DRAM State PAL Pin Description	C-2
C-2	DRAM Control PAL Pin Description	C-14
C-3	Refresh Interval Counter PAL Pin Description	C-19
C-4	Refresh Address Counter PAL Pin Description	C-22
C-5	DRAM Circuit Timing Parameters	C-27

CUSTOMER SUPPORT

CUSTOMER SUPPORT

Customer Support is Intel's complete support service that provides Intel customers with hardware support, software support, customer training, and consulting services. For more information contact your local sales offices.

After a customer purchases any system hardware or software product, service and support become major factors in determining whether that product will continue to meet a customer's expectations. Such support requires an international support organization and a breadth of programs to meet a variety of customer needs. As you might expect, Intel's customer support is quite extensive. It includes factory repair services and worldwide field service offices providing hardware repair services, software support services, customer training classes, and consulting services.

HARDWARE SUPPORT SERVICES

Intel is committed to providing an international service support package through a wide variety of service offerings available from Intel Hardware Support.

SOFTWARE SUPPORT SERVICES

Intel's software support consists of two levels of contracts. Standard support includes TIPS (Technical Information Phone Service), updates and subscription service (product-specific troubleshooting guides and COMMENTS Magazine). Basic support includes updates and the subscription service. Contracts are sold in environments which represent product groupings (i.e., iRMX environment).

CONSULTING SERVICES

Intel provides field systems engineering services for any phase of your development or support effort. You can use our systems engineers in a variety of ways ranging from assistance in using a new product, developing an application, personalizing training, and customizing or tailoring an Intel product to providing technical and management consulting. Systems Engineers are well versed in technical areas such as microcommunications, real-time applications, embedded microcontrollers, and network services. You know your application needs; we know our products. Working together we can help you get a successful product to market in the least possible time.

CUSTOMER TRAINING

Intel offers a wide range of instructional programs covering various aspects of system design and implementation. In just three to ten days a limited number of individuals learn more in a single workshop than in weeks of self-study. For optimum convenience, workshops are scheduled regularly at Training Centers worldwide or we can take our workshops to you for on-site instruction. 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 Overview

1

CHAPTER 1
SYSTEM OVERVIEW

The 80386 is a new 32-bit microprocessor that forms the basis for a high-performance 32-bit system. The 80386 incorporates multitasking support, memory management, pipelined architecture, address translation caches, and a high-speed bus interface all on one chip. The integration of these features speeds the execution of instructions and reduces overall chip count for a system. Paging and dynamic data bus sizing can each be invoked selectively, making the 80386 suitable for a wide variety of system designs and user applications.

While the 80386 represents a significant improvement over previous generations of microprocessors, substantial ties to the earlier processors are preserved. Software compatibility at the object-code level is provided, so that an existing investment in 8086 and 80286 software can be maintained. New software can be built upon existing routines, reducing the time to market for new products. Hardware compatibility is preserved through the dynamic bus-sizing feature.

The major components of an 80386 system and their functions are shown in Figure 1-1. Table 1-1 describes these components.

1.1 MICROPROCESSOR

The 80386 provides unprecedented performance. At 20 MHz, the 80386 is capable of executing at sustained rates of over five million instructions per second, a speed comparable to that of most super minicomputers. This achievement is made possible through a state-of-the-art design that includes a pipelined internal architecture, address translation caches, and a high-performance bus.

The 80386 features 32-bit wide internal and external data paths and eight general-purpose 32-bit registers. The instruction set offers 8-, 16-, and 32-bit data types, and the processor outputs 32-bit physical addresses directly, for a physical memory capacity of four gigabytes.

Table 1-1. 80386 System Components

Component	Description
80386 Microprocessor	32-bit high-performance microprocessor with on-chip memory management and protection
80287 or 80387 Numeric Coprocessor	Performs numeric instruction in parallel with 80386; expands instruction set
82380 Integrated System Peripheral	Provides 32-bit high-speed direct memory access, interrupt control, and interval timers
82385 Cache Controller	Provides cache directory and management logic
82384 Clock Generator	Generates system clock and RESET signal
8259A Programmable Interrupt Controller	Provides interrupt control and management
82258 Advanced DMA	Performs direct memory controller access (DMA)

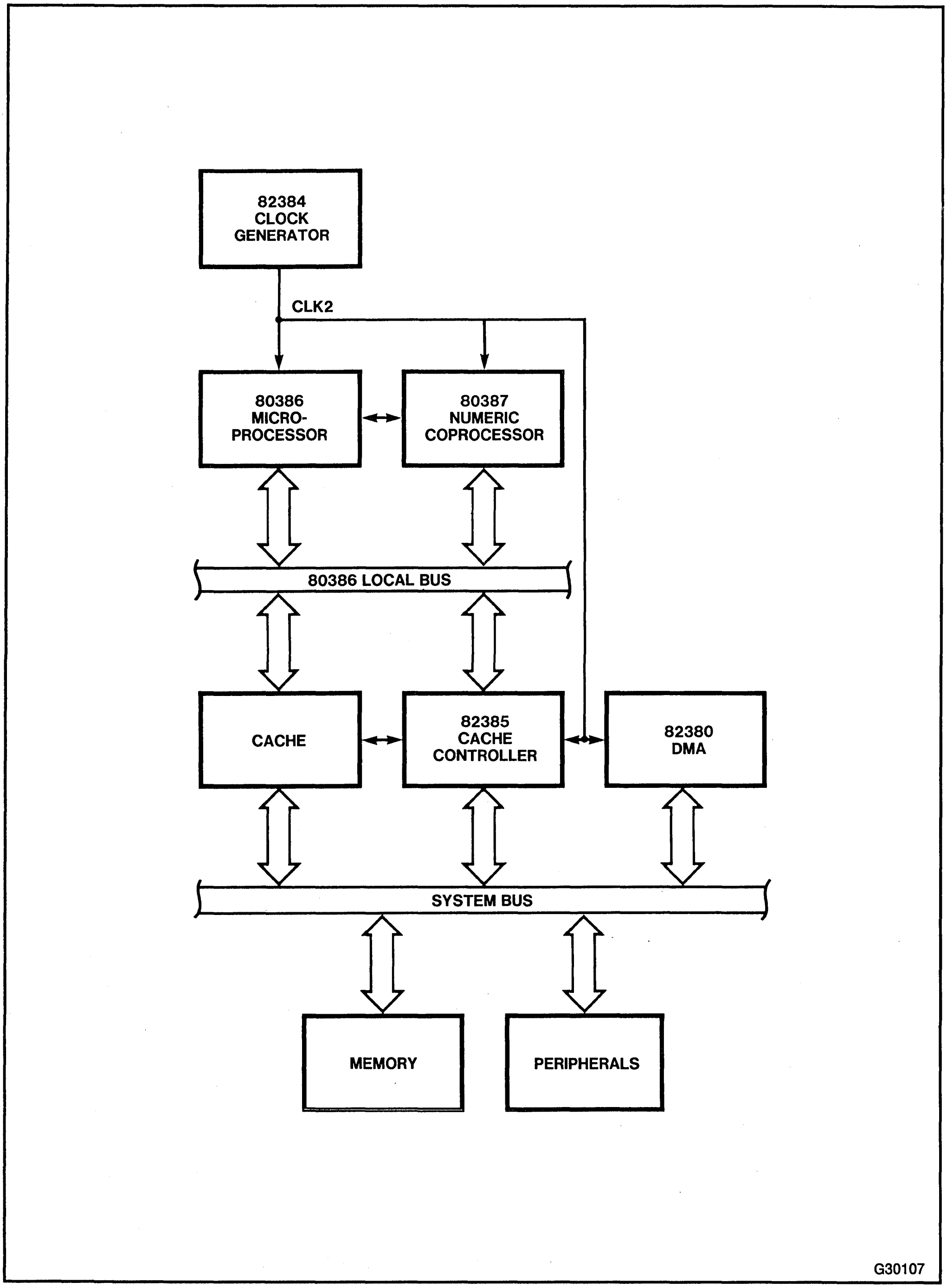

82384
CLOCK
GENERATOR
CLK2
80386
MICRO-
PROCESSOR
80387
NUMERIC
COPROCESSOR
80386 LOCAL BUS
CACHE
82385
CACHE
CONTROLLER
82380
DMA
SYSTEM BUS
MEMORY
PERIPHERALS
G30107

Figure 1-1. 80386 System Block Diagram

The 80386 has separate 32-bit data and address paths. A 32-bit memory access can be completed in only two clock cycles, enabling the bus to sustain a throughput of 40 megabytes per second (at 20 MHz). By making prompt transfers between the microprocessor, memory, and peripherals, the high-speed bus design ensures that the entire system benefits from the processor's increased performance.

Pipelined architecture enables the 80386 to perform instruction fetching, decoding, execution, and memory management functions in parallel. The six independent units that make-up the 80386 pipeline are described in detail in Chapter 2. Because the 80386 prefetches instructions and queues them internally, instruction fetch and decode times are absorbed in the pipeline; the processor rarely has to wait for an instruction to execute.

Pipelining is not unusual in modern microprocessor architecture; however, including the memory management unit (MMU) in the on-chip pipeline is a unique feature of the 80386. By performing memory management on-chip, the 80386 eliminates the serious access delays typical of implementations that use off-chip memory management units. The benefit is not only high performance but also relaxed memory-access time requirements, hence lower system cost.

The integrated memory management and protection mechanism translates logical addresses to physical addresses and enforces the protection rules necessary for maintaining task integrity in a multitasking environment. The paging function simplifies the operating-system swapping algorithms by providing a uniform mechanism for managing the physical structure of memory.

Task switching occurs frequently in real-time multitasking or multiuser systems. To perform task switching efficiently, the 80386 incorporates special high-speed hardware. Only a single instruction or an interrupt is needed for the 80386 to perform a complete task switch. A 20-MHz 80386 can save the state of one task (all registers), load the state of another task (all registers, even segment and paging registers if required), and resume execution in less than 14 microseconds (at 20 MHz). For less sophisticated task and interrupt handling, the latency can be as short as 2.9 microseconds (at 20 MHz).

1.2 COPROCESSORS

The performance of most applications can be enhanced by the use of specialized coprocessors. A coprocessor provides the hardware to perform functions that would otherwise be performed in software. Coprocessors extend the instruction set of the 80386.

The 80386 has a numeric coprocessor interface that can support one of two coprocessors: the 80387 or the 80287. For applications that benefit from high-precision integer and floating-point calculations, these numeric coprocessors provide full support for the IEEE standard for floating-point operations. Both the 80387 and 80287 are software compatible with the 8087, an earlier numeric coprocessor. At 16 MHz, the 80387 operates about eight times faster than a 5-MHz 80287. However, the 80287 is fast enough for many applications and is a cost-effective solution for many designs. The 80386 therefore offers the system designer the choice of low-cost or high-performance numeric solutions.

1.3 INTEGRATED SYSTEM PERIPHERAL

A DMA (Direct Memory Access) controller performs DMA transfers between main memory and an I/O device, typically a hard disk, floppy disk, or communications channel. In a DMA transfer, a large block of data can be copied from one place to another without the intervention of the CPU.

The 82380 Integrated System Peripheral is a multi-function 80386 companion chip. It integrates a 32-bit DMA Controller with other necessary processor support functions needed in an 80386 environment. The 82380 is optimized for use with the 80386 microprocessor. It enhances the overall 80386 system performance by providing high data throughput as well as efficient bus operation. The 32-bit DMA Controller provides eight independently programmable channels that can transfer data at the full bandwidth of the 80386 bus. Other features of the 82380 are listed as follows.

- High performance 32-bit DMA Controller
  - —8 independently programmable channels
  - —32 megabytes per second data transfer rate at 16 MHz
  - —40 megabytes per second data transfer rate at 20 MHz
  - —Capable of transferring data between devices with different bus widths
  - —Automatic byte assembly/disassembly capability for non-align data transfers
  - —Buffer chaining capability for transferring data into non-contiguous memory buffers
- 20-Source Interrupt Controller
  - —15 external, 5 internal interrupt requests
  - —82C59A susperset
  - —Individually programmable interrupt vectors
- Four 16-bit programmable Interval Timers
  - —82C54 compatible
- Programmable Wait State Generator
  - —0 to 15 wait states for memory and I/O access cycles
- DRAM Refresh Controller

  Refresh request always has the highest priority among the DMA requests
- 80386 Shutdown Detect and Reset Control
  - —Software and hardware reset
- Optimized for use with the 80386 microprocessor
  - —Resides on local bus for maximum bus bandwidth

1.4 CACHE CONTROLLER

A cache memory subsystem provides fast local storage for frequently accessed code and data. This results in faster memory access for the microprocessor and reduces the amount of traffic on the system bus.

The 82385 Cache Controller is a high performance peripheral designed specifically for the 80386. The 82385 allows the 80386 to reach its full performance potential by offering the following features:

- Supports a 32 kbyte cache memory organized as either 2-way set associative or direct mapped.
- Integrated cache directory and management logic.
- Utilizes posted writes for zero wait states on write cycles.
- Guarantees cache coherency by bus watching.
- Supports non-cacheable accesses.
- Presents an 80386 interface to system resources.
- Dhrystone benchmark shows an average hit rate of 95%.

1.5 CLOCK GENERATOR

The 82384 Clock Generator generates timing for the 80386 and its support components. The 82384 provides both the 80386 clock (CLK2) and a half-frequency clock (CLK) to indicate the internal phase of the 80386 and to drive 80286-compatible devices that may be included in the system. It can also be used to generate the RESET signal for the 80386 and other system components. Both CLK2 and CLK are used throughout this manual to describe execution times.

1.6 8086/80286 FAMILY COMPONENTS

With the appropriate interface, the 80386 can use 8086/80286 family components. These components include the 8259A and 82258.

The 8259A Programmable Interrupt Controller manages interrupts for an 80386 system. Interrupts from as many as eight external sources are accepted by one 8259A; as many as 64 requests can be accommodated by cascading several 8259A chips. The 8259A resolves priority between active interrupts, then interrupts the processor and passes a code to the processor to identify the interrupting source. Programmable features of the 8259A allow it to be used in a variety of ways to fit the interrupt requirements of a particular system.

The 82258 Advanced DMA (ADMA) Controller offers four channels and provides all the signals necessary to perform DMA transfers. Other features of the 82258 are as follows:

- Command chaining to perform multiple commands
- Data chaining to scatter data to separate memory locations (separate pages, for example) and gather data from separate locations
- Automatic assembly and disassembly to convert from 16-bit memory to 8-bit I/O, or vice versa
- Compare, translate, and verify functions
- The option to replace one of the four high-speed channels with as many as 32 lower-speed, multiplexed channels.

Internal Architecture

2

CHAPTER 2
INTERNAL ARCHITECTURE

The internal architecture of the 80386 consists of six functional units that operate in parallel. Fetching, decoding, execution, memory management, and bus accesses for several instructions are performed simultaneously. This parallel operation is called pipelined instruction processing. With pipelining, each instruction is performed in stages, and the processing of several instructions at different stages may overlap as illustrated in Figure 2-1. The six-stage pipelined processing of the 80386 results in higher performance and an enhanced throughput rate over non-pipelined processors.

The six functional units of the 80386 are identified as follows:

- Bus Interface Unit
- Code Prefetch Unit
- Instruction Decode Unit
- Execution Unit
- Segmentation Unit
- Paging Unit

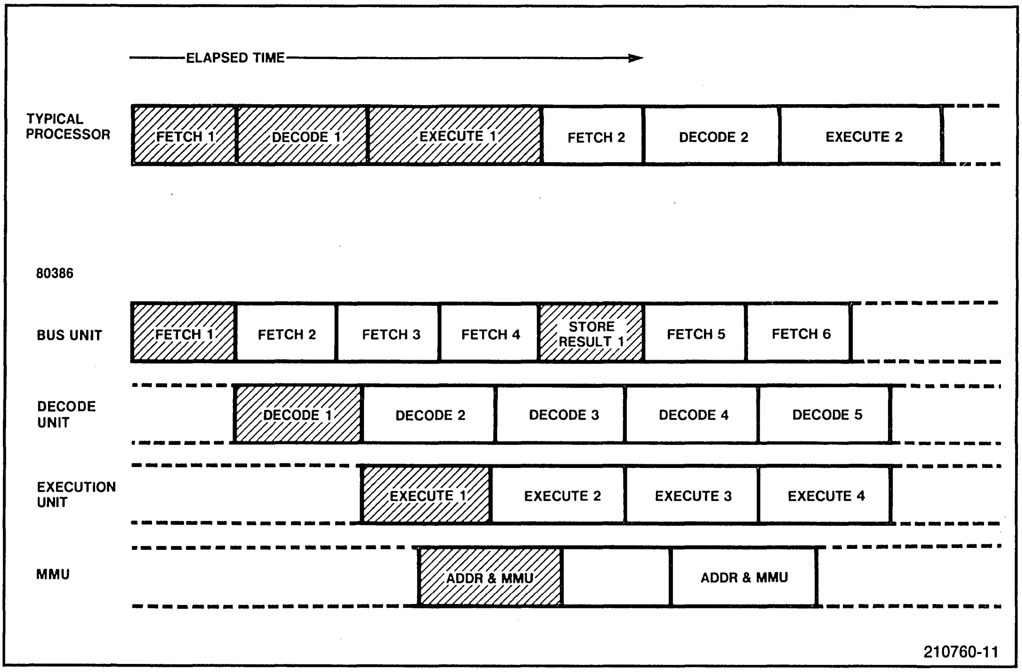

ELAPSED TIME
TYPICAL PROCESSOR
FETCH 1
DECODE 1
EXECUTE 1
FETCH 2
DECODE 2
EXECUTE 2
80386
BUS UNIT
FETCH 1
FETCH 2
FETCH 3
FETCH 4
STORE RESULT 1
FETCH 5
FETCH 6
DECODE UNIT
DECODE 1
DECODE 2
DECODE 3
DECODE 4
DECODE 5
EXECUTION UNIT
EXECUTE 1
EXECUTE 2
EXECUTE 3
EXECUTE 4
MMU
ADDR & MMU
ADDR & MMU
210760-11

Figure 2-1. Instruction Pipelining

The Execution Unit in turn consists of three subunits:

- Control Unit
- Data Unit
- Protection Test Unit

Figure 2-2 shows the organization of these units. This chapter describes the function of each unit, as well as interactions between units.

2.1 BUS INTERFACE UNIT

The Bus Interface Unit provides the interface between the 80386 and its environment. It accepts internal requests for code fetches (from the Code Prefetch Unit) and data transfers (from the Execution Unit), and prioritizes the requests. At the same time, it generates or

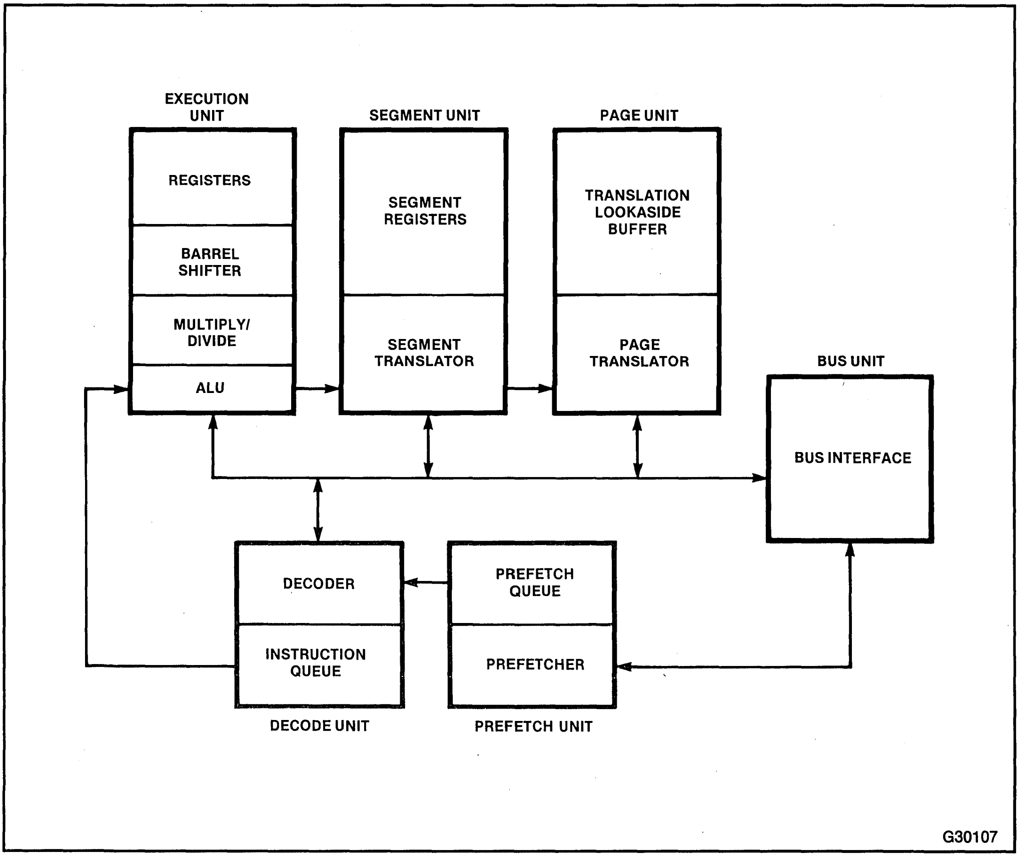

EXECUTION UNIT
SEGMENT UNIT
PAGE UNIT
REGISTERS
SEGMENT REGISTERS
TRANSLATION LOOKASIDE BUFFER
BARREL SHIFTER
MULTIPLY/ DIVIDE
SEGMENT TRANSLATOR
PAGE TRANSLATOR
BUS UNIT
ALU
BUS INTERFACE
DECODER
PREFETCH QUEUE
INSTRUCTION QUEUE
PREFETCHER
DECODE UNIT
PREFETCH UNIT
G30107

Figure 2-2. 80386 Functional Units

processes the signals to perform the current bus cycle. These signals include the address, data, and control outputs for accessing external memory and I/O. The Bus Interface Unit also controls the interface to external bus masters and coprocessors.

2.2 CODE PREFETCH UNIT

The Code Prefetch Unit performs the program look ahead function of the 80386. When the Bus Interface Unit is not performing bus cycles to execute an instruction, the Code Prefetch Unit uses the Bus Interface Unit to fetch sequentially along the instruction byte stream. These prefetched instructions are stored in the 16-byte Code Queue to await processing by the Instruction Decode Unit.

Code prefetches are given a lower priority than data transfers; assuming zero wait state memory access, prefetch activity never delays execution. On the other hand, if there is no data transfer requested, prefetching uses bus cycles that would otherwise be idle. Instruction prefetching reduces to practically zero the time that the processor spends waiting for the next instruction.

2.3 INSTRUCTION DECODE UNIT

The Instruction Decode Unit takes instruction stream bytes from the Prefetch Queue and translates them into microcode. The decoded instructions are then stored in a three-deep Instruction Queue (FIFO) to await processing by the Execution Unit. Immediate data and opcode offsets are also taken from the Prefetch Queue.

2.4 EXECUTION UNIT

The Execution Unit executes the instructions from the Instruction Queue and therefore communicates with all other units required to complete the instruction. The functions of its three subunits are as follows:

- The Control Unit contains microcode and special parallel hardware that speeds multiply, divide, and effective address calculation.
- The Data Unit contains the ALU, a file of eight 32-bit general-purpose registers, and a 64-bit barrel shifter (which performs multiple bit shifts in one clock). The Data Unit performs data operations requested by the Control Unit.
- The Protection Test Unit checks for segmentation violations under the control of the microcode.

To speed up the execution of memory reference instructions, the Execution Unit partially overlaps the execution of any memory reference instruction with the previous instruction. Because memory reference instructions are frequent, a performance gain of approximately nine percent is achieved.

2.5 SEGMENTATION UNIT

The Segmentation Unit translates logical addresses into linear addresses at the request of the Execution Unit. The on-chip Segment Descriptor Cache stores the currently used segment descriptors to speed this translation. At the same time it performs the translation, the Segmentation Unit checks for bus-cycle segmentation violations. (These checks are separate from the static segmentation violation checks performed by the Protection Test Unit.) The translated linear address is forwarded to the Paging Unit.

2.6 PAGING UNIT

When the 80386 paging mechanism is enabled, the Paging Unit translates linear addresses generated by the Segmentation Unit or the Code Prefetch Unit into physical addresses. (If paging is not enabled, the physical address is the same as the linear address, and no translation is necessary.) The Page Descriptor Cache stores recently used Page Directory and Page Table entries in its Translation Lookaside Buffer (TLB) to speed this translation. The Paging Unit forwards physical addresses to the Bus Interface Unit to perform memory and I/O accesses.

Local Bus Interface

3

CHAPTER 3
LOCAL BUS INTERFACE

Local bus operations are considered in this chapter. The 80386 performs a variety of bus operations in response to internal conditions and external conditions (interrupt servicing, for example). The function and timing of the signals that make up the local bus interface are described, as well as the sequences of particular local bus operations.

The high-speed bus interface of the 80386 provides high performance in any system. At the same time, the bus control inputs and status outputs of the 80386 allow for adaptation to a wide variety of system environments.

The 80386 communicates with external memory, I/O, and other devices through a parallel bus interface. This interface consists of a data bus, a separate address bus, five bus status pins, and three bus control pins as follows:

- The bidirectional data bus consists of 32 pins (D31-D0). Either 8, 16, 24, or 32 bits of data can be transferred at once.

- The address bus, which generates 32-bit addresses, consists of 30 address pins (A31-A2) and four byte-enable pins (BE3#-BE0#). Each byte-enable pin corresponds to one of four bytes of the 32-bit data bus. The address pins identify a 4-byte location, and the byte-enable pins select the active bytes within the 4-byte location.

- The bus status pins establish the type of bus cycle to be performed. These outputs indicate the following conditions:

  Address Status (ADS#)—address bus outputs valid
  Write/Read (W/R#)—write or read cycle
  Memory /I/O (M/IO#)—memory or I/O access
  Data/Control (D/C#)—data or control cycle
  LOCK#—locked bus cycle

- The bus control pins allow external logic to control the bus cycle on a cycle-by-cycle basis. These inputs perform the following functions:

  READY#—ends the current bus cycle; controls bus cycle duration

  Next Address (NA#)—allows address pipelining, that is, emitting address and status signals for the next bus cycle during the current cycle

  Bus Size 16 (BS16#)—activates 16-bit data bus operation; data is transferred on the lower 16 bits of the data bus, and an extra cycle is provided for transfers of more than 16 bits

The following pins are used to control the execution of instructions in the 80386 and to interface external bus masters. The 80386 provides both a standard interface to communicate with other bus masters and a special interface support a numerics coprocessor.

- The CLK2 input provides a double-frequency clock signal for synchronous operation. This signal is divided by two internally, so the 80386 fundamental frequency is half the CLK2 signal frequency. For example, a 20-MHz 80386 uses a 40-MHz CLK2 signal.
- The RESET input forces the 80386 to a known reset state.
- The HOLD signal can be generated by another bus master to request that the 80386 release control of the bus. The 80386 responds by activating the Hold Acknowledge (HLDA) signal as it relinquishes control of the local bus.
- The Maskable Interrupt (INTR) and Non-Maskable Interrupt (NMI) inputs cause the 80386 to interrupt its current instruction stream and begin execution of an interrupt service routine.
- The BUSY#, ERROR#, and Coprocessor Request (PEREQ) signals make up the interface to an external numeric coprocessor. BUSY# and ERROR# are status signals from the coprocessor; PEREQ allows the coprocessor to request data from the 80386. The 80386 can use either the 80287 or the 80387 coprocessor.

All of the 80386 bus interface pins are summarized in Table 3-1.

3.1 BUS OPERATIONS

There are seven types of bus operations:

- Memory read
- Memory write
- I/O read
- I/O write
- Instruction fetch
- Interrupt acknowledge
- Halt/shutdown

Each bus cycle is initiated when the address is valid on the address bus, and bus status pins are driven to states that correspond to the type of bus cycle, and ADS# is driven low. Status pin states that correspond to each bus cycle type are shown in Table 3-2. Notice that the signal combinations marked as invalid states may occur when ADS# is false (high). These combinations will never occur if the signals are sampled on the CLK2 rising edge when ADS# is low, and the 80386 internal CLK is high (as indicated by the CLK output of the 82384). Bus status signals must be qualified with ADS# is true (low) to identify the bus cycle.

Memory read and memory write cycles can be locked to prevent another bus master from using the local bus and allow for indivisible read-modify-write operations.

Table 3-1. Summary of 80386 Signal Pins

Signal Name	Signal Function	Active State	Input/ Output	Input Synch or Asynch to CLK2	Output High Impedance During HLDA?
CLK2	Clock	—	I	—	—
D0–D31	Data Bus	High	I/O	S	Yes
BE0#–BE3#	Byte Enables	Low	O	—	Yes
A2–A31	Address Bus	High	O	—	Yes
W/R#	Write-Read Indication	High	O	—	Yes
D/C#	Data-Control Indication	High	O	—	Yes
M/IO#	Memory-I/O Indication	High	O	—	Yes
LOCK#	Bus Lock Indication	Low	O	—	Yes
ADS#	Address Status	Low	O	—	Yes
NA#	Next Address Request	Low	I	S	—
BS16#	Bus Size 16	Low	I	S	—
READY#	Transfer Acknowledge	Low	I	S	—
HOLD	Bus Hold Request	High	I	S	—
HLDA	Bus Hold Acknowledge	High	O	—	No
PEREQ	Coprocessor Request	High	I	A	—
BUSY#	Coprocessor Busy	Low	I	A	—
ERROR#	Coprocessor Error	Low	I	A	—
INTR	Maskable Interrupt Request	High	I	A	—
NMI	Non-Maskable Intrpt Request	High	I	A	—
RESET	Reset	High	I	S	—

Table 3-2. Bus Cycle Definitions

M/IO#	D/C#	W/R#	Bus Cycle Type	Locked?
Low	Low	Low	INTERRUPT ACKNOWLEDGE	Yes
Low	Low	High	does not occur	—
Low	High	Low	I/O DATA READ	No
Low	High	High	I/O DATA WRITE	No
High	Low	Low	MEMORY CODE READ	No
High	Low	High	HALT: Address = 2 (BE0# High BE1# High BE2# Low BE3# High A2–A31 Low) SHUTDOWN: Address = 0 (BE0# Low BE1# High BE2# High BE3# High A2–A31 Low)	No
High	High	Low	MEMORY DATA READ	Some Cycles
High	High	High	MEMORY DATA WRITE	Some Cycles

3.1.1 Bus States

The 80386 uses a double-frequency clock input (CLK2) to generate its internal processor clock signal (CLK). As shown in Figure 3-1, each CLK cycle is two CLK2 cycles wide.

Notice that the internal 80386 matches the external 82384 CLK. The 82384 CLK is permitted to lag CLK2 slightly, but will never lead CLK2, so that it can be used reliably as a phase status indicator. All 80386 inputs are sampled at CLK2 rising edges. Many 80386 signals are sampled every other CLK2 rising edge; some are sampled on the CLK2 edge when CLK is high, while some are sampled on the CLK2 edge when CLK is low. The maximum data transfer rate for a bus operation, as determined by the 80386 internal clock, is 32 bits for every two CLK cycles, or 40 megabytes per second (CLK2 = 40 MHz, internal CLK = 20 MHz).

Each bus cycle is comprised of at least two bus states, T1 and T2. Each bus state in turn consists of two CLK2 cycles, which can be thought of as Phase 1 and Phase 2 of the bus state. Figure 3-2 shows bus states for some typical read and write cycles. During the first bus state (T1), address and bus status pins go active. During the second bus state (T2), external logic and devices respond. If the READY# input of the 80386 is sampled low at the end of the second CLK cycle, the bus cycle terminates. If READY# is high when sampled, the bus cycle continues for an additional T2 state, called a wait state, and READY# is sampled again. Wait states are added until READY# is sampled low.

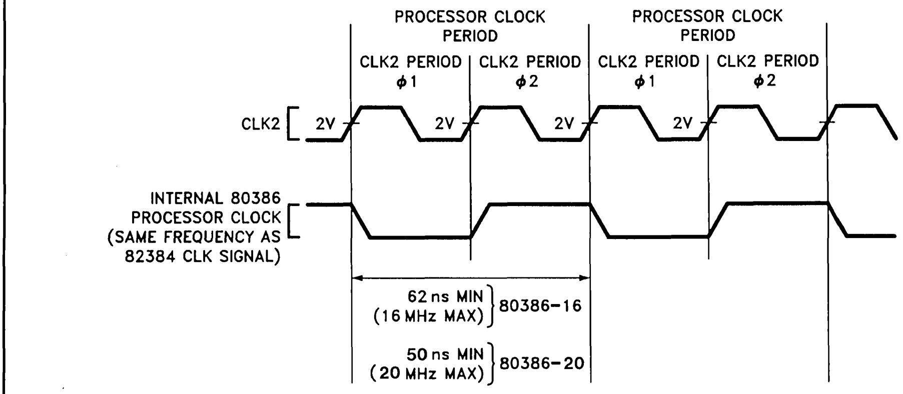

PROCESSOR CLOCK PERIOD
PROCESSOR CLOCK PERIOD
CLK2 PERIOD φ1
CLK2 PERIOD φ2
CLK2 PERIOD φ1
CLK2 PERIOD φ2
CLK2
2V
2V
2V
2V
INTERNAL 80386 PROCESSOR CLOCK (SAME FREQUENCY AS 82384 CLK SIGNAL)
62 ns MIN (16 MHz MAX) 80386-16
50 ns MIN (20 MHz MAX) 80386-20
231630-2

Figure 3-1. CLK2 and CLK Relationship

When no bus cycles are needed by the 80386 (no bus requests are pending, the 80386 remains in the idle bus state (TI). The relationship between T1, T2, and TI is shown in Figure 3-3.

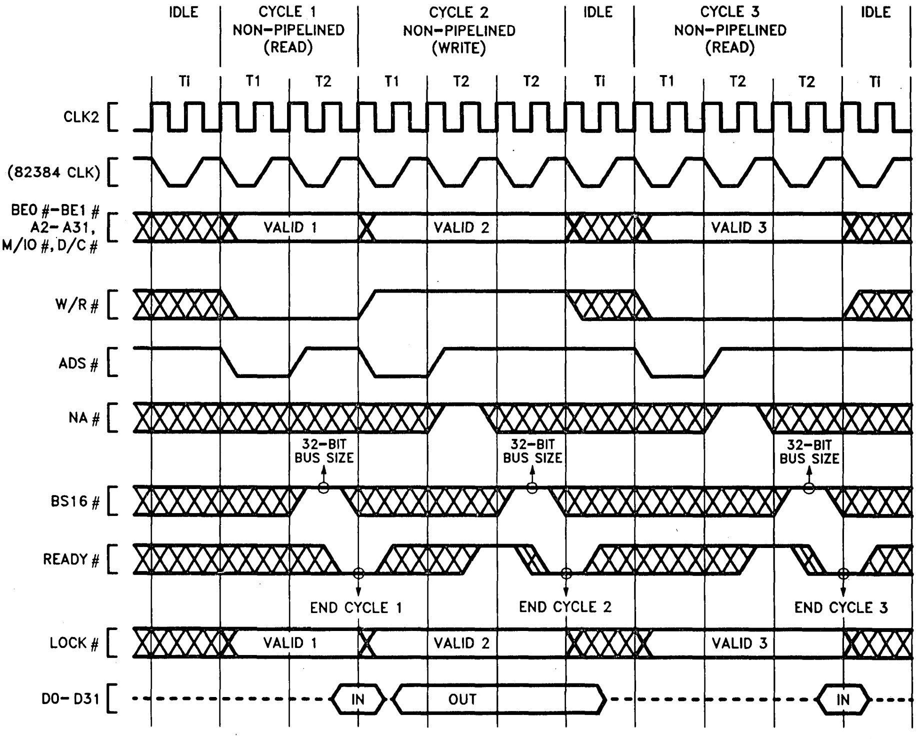
3.1.2 Address Pipelining

In the 80386, the timing address and status outputs can be controlled so the outputs become valid before the end of the previous bus cycle. This technique, which allows bus cycles to be overlapped, is called address pipelining. Figure 3-4 compares non-pipelined address cycles to pipelined address cycles.

Address pipelining increases bus throughput without decreasing allowable memory or I/O access time, thus allowing high bandwidth with relatively inexpensive components. In addition, using pipelining to address slower devices can yield the same throughput as addressing faster devices with no pipelining. A 20-MHz 80386 can transfer data at the maximum rate of 40 megabytes per second while allowing an address access time of three CLK cycles (150 nanoseconds at CLK = 20 MHz neglecting signal delays); without address pipelining, the access time is only two CLK cycles (100 nanoseconds at CLK = 20 MHz). When address pipeline is activated following an idle bus cycle, performance is decreased slightly because the first bus cycle cannot be pipelined. This condition is explained fully in Chapter 4.

3.1.3 32-Bit Data Bus Transfers and Operand Alignment

The 80386 can address up to four gigabytes (2^{32} bytes, addresses 00000000H-FFFFFFFFH) of physical memory and up to 64 kilobytes (2^{16} bytes, addresses 00000000H-0000FFFFH) of I/O. The 80386 maintains separate physical memory and I/O spaces.


IDLE
CYCLE 1
NON-PIPELINED
(READ)
CYCLE 2
NON-PIPELINED
(WRITE)
IDLE
CYCLE 3
NON-PIPELINED
(READ)
IDLE
Ti
T1
T2
T1
T2
T2
Ti
T1
T2
T2
Ti
CLK2
(82384 CLK)
BE0#-BE1#
A2-A31,
M/IO#, D/C#
VALID 1
VALID 2
VALID 3
W/R#
ADS#
NA#
32-BIT
BUS SIZE
BS16#
READY#
END CYCLE 1
END CYCLE 2
END CYCLE 3
LOCK#
D0-D31
IN
OUT

231630-16

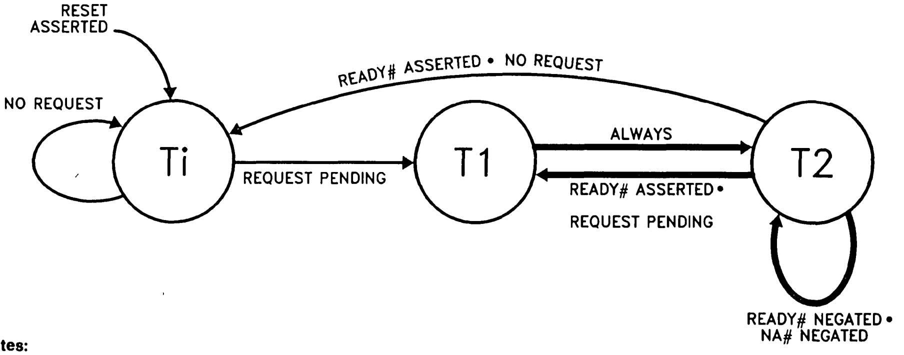
Idle states are shown here for diagram variety only. Write cycles are **not** always followed by an idle state. An active bus cycle can immediately follow the write cycle.

231630-16

Figure 3-2. 80386 Bus States Timing Example

The programmer views the address space (memory or I/O) of the 80386 as a sequence of bytes. Words consist of two consecutive bytes, and double words consist of four consecutive bytes. However, in the system hardware, address space is implemented in four sections. Each of the four 8-bit portions of the data bus (D0-D7, D8-D15, D16-D23, and D24-D31) connects to a section. When the 80386 reads a doubleword, it accesses one byte from each section. The 80386 automatically translates the programmers' view of consecutive bytes into this hardware implementation (see Figure 3-5).

The 80386 memory spaces and I/O space are organized physically as sequences of 32-bit doublewords (2^{30} 32-bit memory locations and 2^{14} 32-bit I/O ports maximum). Each doubleword starts at a physical address that is a multiple of four, and has four individually addressable bytes at consecutive addresses.


RESET ASSERTED
NO REQUEST
READY# ASSERTED • NO REQUEST
Ti
REQUEST PENDING
T1
ALWAYS
READY# ASSERTED • REQUEST PENDING
T2
READY# NEGATED • NA# NEGATED

Bus States:
T1—first clock of a non-pipelined bus cycle (80386 drives new address and asserts ADS#)
T2—subsequent clocks of a bus cycle when NA# has not been sampled asserted in the current bus cycle
Ti— idle state

The fastest bus cycle consists of two states: T1 and T2.

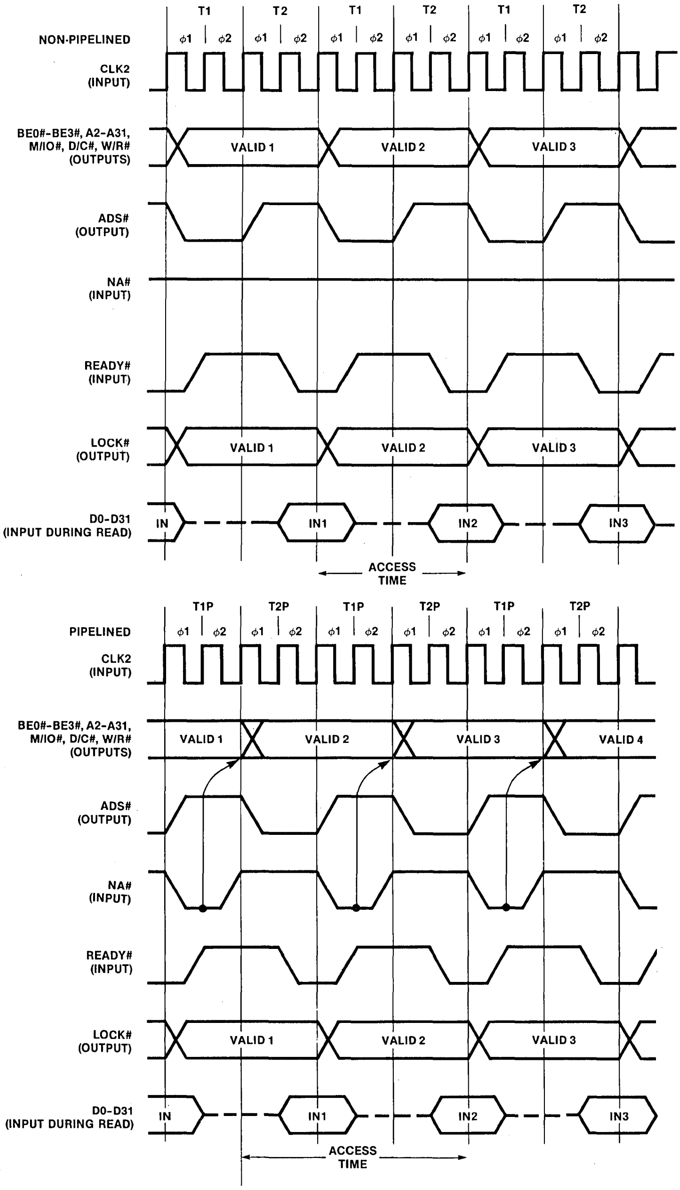
231630-17

Figure 3-3. Bus State Diagram (Does Not Include Address Pipelining)

Pins A31-A2 correspond to the most significant bits of the physical address; these pins address doublewords of memory. The two least significant bits of the physical address are used internally to activate the appropriate byte enable output (BE3#-BE0#). Figure 3-6 shows the relationship between physical address, doubleword location, data bus pins, and byte enables. This relationship holds for a 32-bit bus only; the organization for a 16-bit bus is described later in Section 3.1.10.

Data can be transferred in quantities of 32 bits, 24 bits, 16 bits, or 8 bits for each bus cycle of a data transfer. Table 3-3 shows which bytes of a 32-bit doubleword can be transferred in a single bus cycle. If a data transfer can be completed in a single cycle, the transfer is said to be aligned. For example, a word transfer involving D23-D8 and activating BE1# and BE2# is aligned.

Transfers of words and doublewords that overlap a doubleword boundary of the 80386 are called misaligned transfers. These transfers require two bus cycles, which are automatically generated by the 80386. For example, a word transfer at (byte) address 0003H requires two byte transfers: the first transfer activates doubleword address 0004H and uses D7-D0, and the second transfer activates doubleword address 0000H and uses D31-D24.


NON-PIPELINED
T1
T2
ϕ1
ϕ2
CLK2 (INPUT)
BE0#-BE3#, A2-A31, M/IO#, D/C#, W/R# (OUTPUTS)
VALID 1
VALID 2
VALID 3
ADS# (OUTPUT)
NA# (INPUT)
READY# (INPUT)
LOCK# (OUTPUT)
D0-D31 (INPUT DURING READ)
IN
IN1
IN2
IN3
ACCESS TIME
PIPELINED
T1P
T2P
VALID 4

G30107

Figure 3-4. Non-Pipelined Address and Pipelined Address Differences

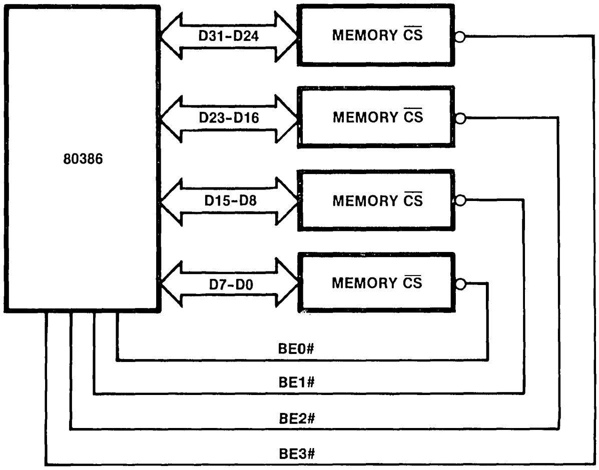
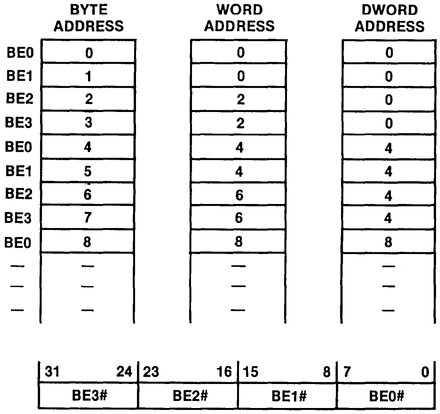

80386
D31-D24
D23-D16
D15-D8
D7-D0
MEMORY $\overline{CS}$
MEMORY $\overline{CS}$
MEMORY $\overline{CS}$
MEMORY $\overline{CS}$
BE0#
BE1#
BE2#
BE3#
G30107

Figure 3-5. Consecutive Bytes in Hardware Implementation



	BYTE ADDRESS	WORD ADDRESS	DWORD ADDRESS
BE0	0	0	0
BE1	1	0	0
BE2	2	2	0
BE3	3	2	0
BE0	4	4	4
BE1	5	4	4
BE2	6	6	4
BE3	7	6	4
BE0	8	8	8
—	—	—	—
—	—	—	—
—	—	—	—

31 24	23 16	15 8	7 0
BE3#	BE2#	BE1#	BE0#

Figure 3-6. Address, Data Bus, and Byte Enables for 32-Bit Bus

Table 3-3. Possible Data Transfers on 32-Bit Bus

Possible Data Transfers to 32-Bit Memory	
Size	**Byte Enables**
32 bits	3-2-1-0
24 bits	3-2-1 2-1-0
16 bits	3-2 2-1 1-0
8 bits	3 2 1 0

Figure 3-7 shows the steps required for a misaligned 32-bit transfer. In the first bus cycle, the physical address crosses over into the next doubleword location, and BE0# and BE1# are active. In the second bus cycle, the address is decremented to the previous doubleword, and BE2# and BE3# are active. After the transfer, the data bits are automatically assembled in the correct order.

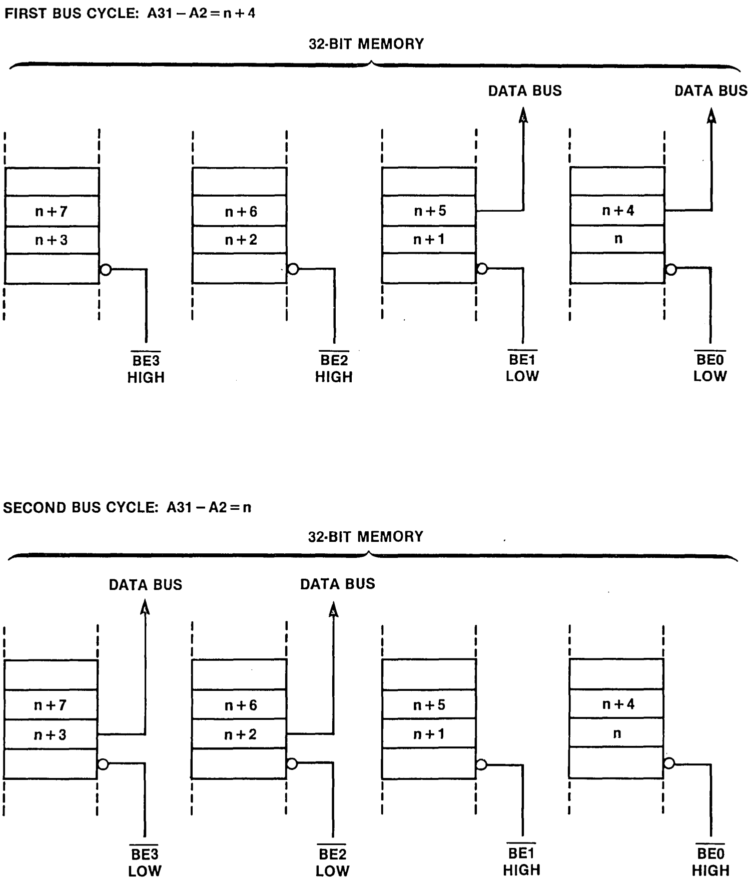
Table 3-4 shows the sequence of bus cycles for all possible misaligned transfers. Even though misaligned transfers are transparent to a program, they are slower than aligned transfers and should thus be avoided.

Because the 80386 operates on only bytes, words, and doublewords, certain combinations of BE3#-BE0# are never produced. For example, a bus cycle is never performed with only BE0# and BE2# active because such a transfer would be an operation on two noncontiguous bytes at the same time. A single 3-byte transfer will never occur, but a 3-byte transfer followed or preceded by a 1-byte transfer can occur for some misaligned doubleword transfers.

3.1.4 Read Cycle

Read cycles are of two types: pipelined address cycles and non-pipelined address cycles. In a non-pipelined address cycle, the address bus and bus status signals become valid during the first CLK period of the cycle. In a pipelined address cycle, the address bus and bus status signals are output before the beginning of cycle, in the previous bus cycle, to allow longer memory access times. Pipelined address cycles are described in Section 3.1.6.

The timing for two non-pipelined address read cycles (one with and one without a wait state) is shown in Figure 3-8.


FIRST BUS CYCLE: A31 – A2 = n + 4
32-BIT MEMORY
DATA BUS
DATA BUS
n + 7
n + 3
n + 6
n + 2
n + 5
n + 1
n + 4
n
$\overline{BE3}$ HIGH
$\overline{BE2}$ HIGH
$\overline{BE1}$ LOW
$\overline{BE0}$ LOW
SECOND BUS CYCLE: A31 – A2 = n
32-BIT MEMORY
DATA BUS
DATA BUS
n + 7
n + 3
n + 6
n + 2
n + 5
n + 1
n + 4
n
$\overline{BE3}$ LOW
$\overline{BE2}$ LOW
$\overline{BE1}$ HIGH
$\overline{BE0}$ HIGH

G30107

Figure 3-7. Misaligned Transfer

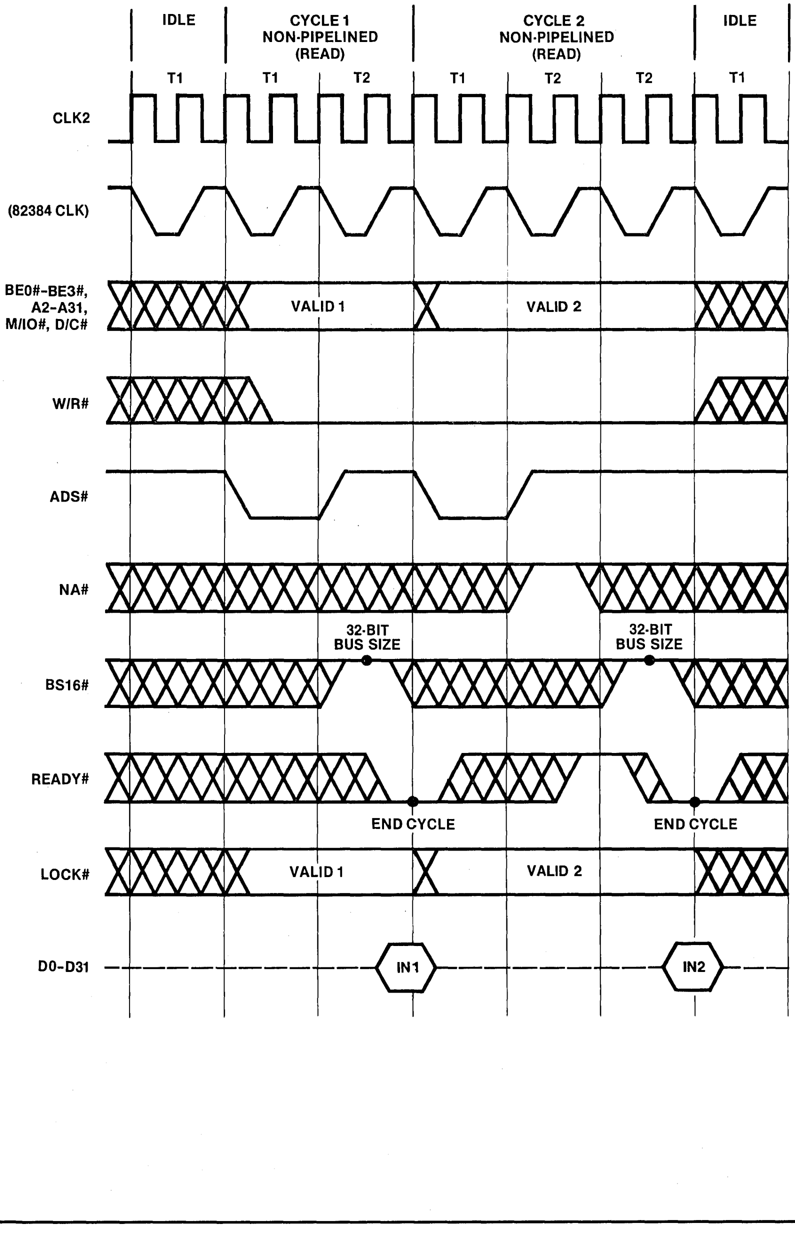

IDLE
CYCLE 1 NON-PIPELINED (READ)
CYCLE 2 NON-PIPELINED (READ)
IDLE
T1
T1
T2
T1
T2
T2
T1
CLK2
(82384 CLK)
BE0#-BE3#, A2-A31, M/IO#, D/C#
VALID 1
VALID 2
W/R#
ADS#
NA#
32-BIT BUS SIZE
32-BIT BUS SIZE
BS16#
READY#
END CYCLE
END CYCLE
LOCK#
VALID 1
VALID 2
D0-D31
IN1
IN2
G30107

Figure 3-8. Non-Pipelined Address Read Cycles

Table 3-4. Misaligned Data Transfers on 32-Bit Bus

		First Cycle:		Second Cycle:	
Transfer Type	**Physical Address**	**Address Bus**	**Byte Enables**	**Address Bus**	**Byte Enables**
Word	4N + 3	4N + 4	0	4N	3
Doubleword	4N + 1	4N + 4	0	4N	1–3
Doubleword	4N + 2	4N + 4	0–1	4N	2–3
Doubleword	4N + 3	4N + 4	0–2	4N	3

NOTE: 4N = Nth doubleword address

The sequence of signals for the non-pipelined read cycle is as follows:

- The 80386 initiates the cycle by driving ADS# low. The states of the address bus (A31-A2), byte enable pins (BE3#-BE0#), and bus status outputs (M/IO#, D/C#, W/R#, and LOCK#) at the CLK2 edge when ADS# is sampled low determine the type of bus cycle to be performed. For a read cycle,

  —W/R# is low

  —M/IO# is high for a memory read, low for an I/O read

  —For a memory read, D/C# is high if data is to be read, low if an instruction is to be read. Immediate data is included in an instruction.

  —LOCK# is low if the bus cycle is a locked cycle. In a read-modify-write sequence, both the memory data read cycle and the memory data write cycle are locked. No other bus master should be permitted to control the bus between two locked bus cycles.

  The address bus, byte enable pins, and bus status pins (with the exception of ADS#) remain active through the end of the read cycle.

- At the end of T2, READY# is sampled. If READY# is low, the 80386 reads the input data on the data bus.
- If READY# is high, wait states (one CLK cycle) are added until READY# is sampled low. READY# is sampled at the end of each wait state.
- Once READY# is sampled low, the 80386 reads the input data, and the read cycle terminates. If a new bus cycle is pending, it begins on the next CLK cycle.

3.1.5 Write Cycle

Write cycles, like read cycles, are of two types: pipelined address and non-pipelined address. Pipelined address cycles are described in Section 3.1.6.

Figure 3-9 shows two non-pipelined address write cycles (one with and one without a wait state. The sequence of signals for a non-pipelined write cycle is as follows:

- The 80386 initiates the cycle by driving ADS# low. The states of the address bus (A31-A2), byte enable pins (BE3#-BE0#), and bus status outputs (M/IO#, D/C#, W/R#, and LOCK#) at the CLK edge when ADS# is sampled low to determine the type of bus cycle to be performed. For a write cycle,

  —W/R# is high

  —M/IO# is high for a memory write, low for an I/O write

  —D/C# is high

  —LOCK# is low if the bus cycle is a locked cycle. In a read-modify-write sequence, both the memory data read cycle and the memory data write cycle are locked. No other bus master should be permitted to control the bus between two locked bus cycles.

  The address bus, byte enable pins, and bus status pins (with the exception of ADS#) remain active through the end of the write cycle.
- At the start of Phase 2 in T1, output data becomes valid on the data bus. This data remains valid until the start of Phase 2 in T1 of the next bus cycle.
- At the end of T2, READY# is sampled. If READY# is low, the write cycle terminates.
- If READY# is not low, wait states are added until READY# is sampled low. READY# is sampled at the end of each wait state.
- Once READY# is sampled low, the write cycle terminates. If a new bus cycle is pending, it begins on the next CLK cycle.

3.1.6 Pipelined Address Cycle

Address pipelining allows bus cycles to be overlapped, increasing the amount of time available for the memory or I/O device to respond. The NA# input of the 80386 controls address pipelining. NA# is generated by logic in the system to indicate that the address bus is no longer needed (for example, after the address has been latched). If the system is designed so that NA# goes active before the end of the cycle, address pipelining may occur.

NA# is sampled at the rising CLK2 edge of Phase 2 of each CLK cycle. Once NA# is sampled active, the address, byte enables, and bus status signals for the next bus cycle are output as soon as they are available internally. Once NA# is sampled active, it is not required again until the CLK cycle after ADS# goes active.

Figure 3-10 illustrates the effect of NA#. During the second CLK cycle (T2) of a non-pipelined address cycle, NA# is sampled low. The address, byte enables, and bus status signals for the next bus cycle are output in the third CLK cycle (the first wait state of the current bus cycle). Thereafter, NA# is sampled in the next CLK cycle after ADS# is valid (T1 of each bus cycle); if NA# is active, the address, byte enables and bus-status pins for the next cycle are output in T2 if another bus cycle is pending.

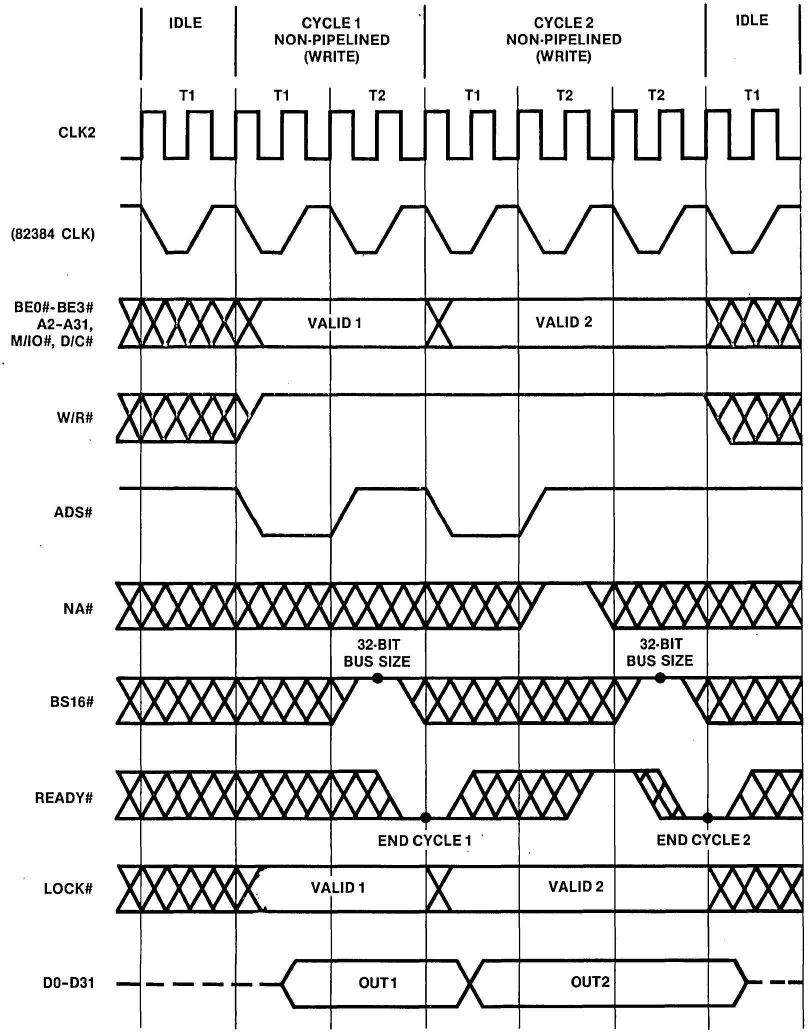
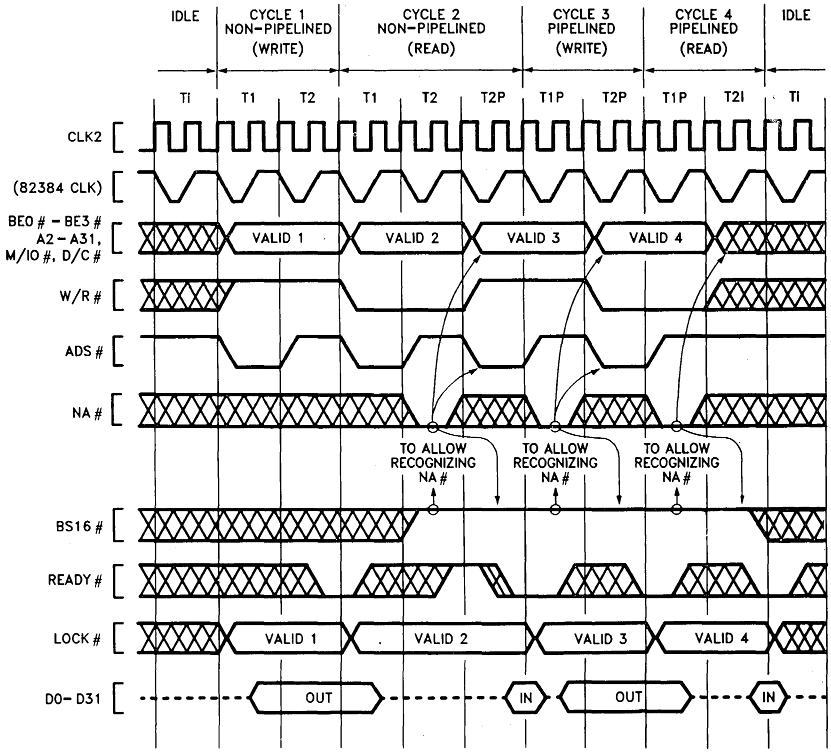

IDLE
CYCLE 1
NON-PIPELINED
(WRITE)
CYCLE 2
NON-PIPELINED
(WRITE)
IDLE
T1
T1
T2
T1
T2
T2
T1
CLK2
(82384 CLK)
BE0#-BE3#
A2-A31,
M/IO#, D/C#
VALID 1
VALID 2
W/R#
ADS#
NA#
32-BIT
BUS SIZE
32-BIT
BUS SIZE
BS16#
READY#
END CYCLE 1
END CYCLE 2
LOCK#
VALID 1
VALID 2
D0-D31
OUT1
OUT2
G30107

Figure 3-9. Non-Pipelined Address Write Cycles


IDLE
CYCLE 1 NON-PIPELINED (WRITE)
CYCLE 2 NON-PIPELINED (READ)
CYCLE 3 PIPELINED (WRITE)
CYCLE 4 PIPELINED (READ)
IDLE
Ti
T1
T2
T1
T2
T2P
T1P
T2P
T1P
T2I
Ti
CLK2
(82384 CLK)
BE0# - BE3#
A2 - A31,
M/IO#, D/C#
VALID 1
VALID 2
VALID 3
VALID 4
W/R#
ADS#
NA#
TO ALLOW RECOGNIZING NA#
TO ALLOW RECOGNIZING NA#
TO ALLOW RECOGNIZING NA#
BS16#
READY#
LOCK#
VALID 1
VALID 2
VALID 3
VALID 4
D0- D31
OUT
IN
OUT
IN

231630-20

Following any idle bus state (Ti), addresses are non-pipelined. Within non-pipelined bus cycles, NA# is only sampled during wait states. Therefore, to begin address pipelining during a group of non-pipelined bus cycles requires a non-pipelined cycle with at least one wait state (Cycle 2 above).

231630-20

Figure 3-10. Pipelined Address Cycles

The first bus cycle after an idle bus state is always non-pipelined. To initiate address pipelining, this cycle must be extended by at least one CLK cycle so that the address and status can be output before the end of the cycle. Subsequent cycles can be pipelined as long as no idle bus cycles occur.

NA# is sampled at the start of Phase 2 of any CLK cycle in which ADS# is not active, specifically,

- The second CLK cycle of a non-pipelined address cycle
- The first CLK cycle of a pipelined address cycle
- Any wait state of a non-pipelined address or pipelined address cycle unless NA# has already been sampled active

Once NA# is sampled active, it remains active internally throughout the current bus cycle. If NA# and READY# are active in the same CLK cycle, the state of NA# is irrelevant, because READY# causes the start of a new bus cycle; therefore, the new address and status signals are always output regardless of the state of NA#.

A complete discussion of the considerations for using address pipelining can be found in the *80386 Data Sheet* (Order Number 231630).

3.1.7 Interrupt Acknowledge Cycle

An unmasked interrupt causes the 80386 to suspend execution of the current program and perform instructions from another program called a service routine. Interrupts are described in detail in Section 3.4.

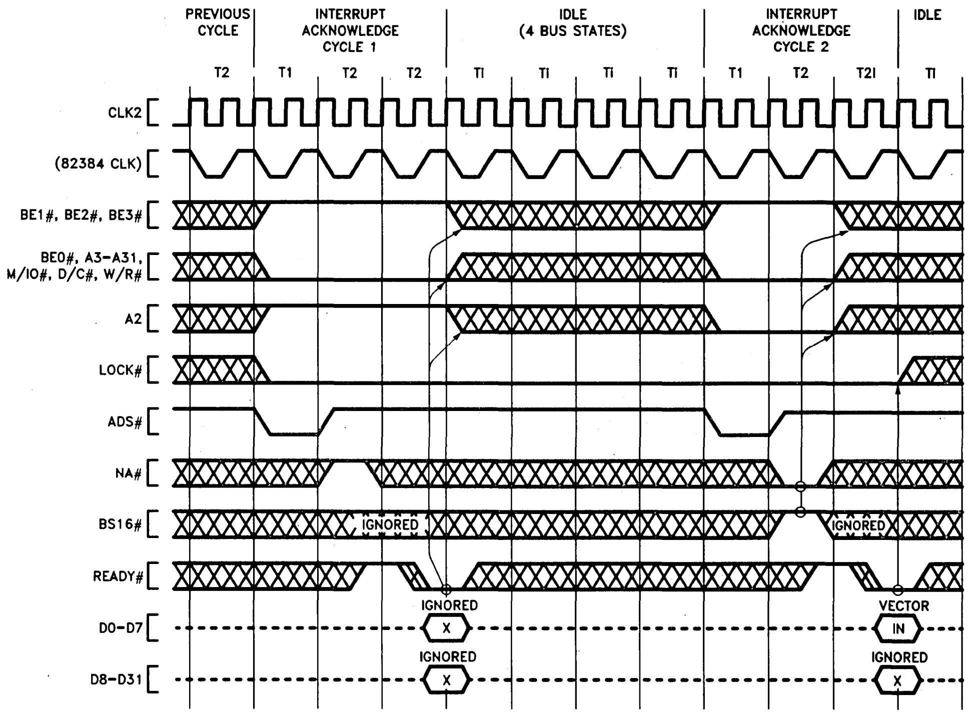
The 8259A Programmable Interrupt Controller is a system component that coordinates the interrupts of several devices (eight interrupts for a single 8259A; up to 64 interrupts with eight cascaded 8259As). When a device signals an interrupt request, the 8259A activates the INTR input of the 80386.

Interrupt acknowledge cycles are special bus cycles designed to activate the 8259A INTA input that enables the 8259A to output a service-routine vector on the data bus. The 80386 performs two back-to-back interrupt acknowledge cycles in response to an active INTR input (as long as the interrupt flag of the 80386 is enabled).

Interrupt acknowledge cycles are similar to regular bus cycles in that the 80386 bus outputs signals at the start of each bus cycle and an active READY# terminates each bus cycle. The cycles are shown in Figure 3-11.

- ADS# is driven low to start each bus cycle.
- Control signals M/IO#, D/C#, and W/R# are driven low to signal to interrupt-acknowledge bus cycles. These signals must be decoded to generate the INTA input signal for the 8259A. The decoding logic is usually included in the bus controller logic for the particular design. Bus controller designs are discussed in Chapters 6 and 8.
- LOCK# is active from the beginning of the first cycle to the end of the second. HOLD requests from other bus masters are not recognized until after the second interrupt acknowledge cycle.
- The address driven during the first cycle is 4; during the second cycle, the address is 0. BE3#, BE2#, and BE1# are high, BE0# is low, and A31-A3 are low for both cycles; A2 is high for the first cycle and low for the second.
- The 80386 floats D31-D0 for both cycles; however, at the end of the second cycle, the service routine vector at the 8259A outputs is read by the 80386 on pins D7-D0.
- READY# must go low to terminate each cycle.

System logic must delay READY# to extend the cycle to the minimum pulse-width requirement of the 8259A Programmable Interrupt Controller. In addition, the 80386 inserts at least 160 nanoseconds of bus idle time (four Ti states) between the two cycles to match the recovery time of the 8259A.


PREVIOUS CYCLE
INTERRUPT ACKNOWLEDGE CYCLE 1
IDLE (4 BUS STATES)
INTERRUPT ACKNOWLEDGE CYCLE 2
IDLE
T2
T1
T2
T2
TI
TI
TI
TI
T1
T2
T2I
TI
CLK2
(82384 CLK)
BE1#, BE2#, BE3#
BE0#, A3-A31, M/IO#, D/C#, W/R#
A2
LOCK#
ADS#
NA#
BS16#
READY#
D0-D7
D8-D31
IGNORED
IGNORED
X
IGNORED
X
IGNORED
VECTOR
IN
IGNORED
X

231630-26

Interrupt Vector (0-255) is read on D0-D7 at end of second Interrupt Acknowledge bus cycle.
Because each Interrupt Acknowledge bus cycle is followed by idle bus states, asserting NA# has no practical effect. Choose the approach which is simplest for your system hardware design.

231630-26

Figure 3-11. Interrupt Acknowledge Bus Cycles

3.1.8 Halt/Shutdown Cycle

The halt condition in the 80386 occurs in response to a HLT instruction. The shutdown condition occurs when the 80386 is processing a double fault and encounters a protection fault; the 80386 cannot recover and shuts down. Halt or shutdown cycles result from these conditions. Externally, a shutdown cycle differs from a halt cycle only in the resulting address bus outputs.

As with other bus cycles, a halt or shutdown cycle is initiated by activating ADS# and the bus status pins as follows:

- M/IO# and W/R# are driven high, and D/C# is driven low to indicate a halt cycle.

- All address bus outputs are driven low. For a halt condition, BE2# is active; for a shutdown condition, BE0# is active. These signals are used by external devices to respond to the halt or shutdown cycle.

READY# must be asserted to complete the halt or shutdown cycle. The 80386 will remain in the halt or shutdown condition until...

- NMI goes high; 80386 services the interrupt
- RESET goes high; 80386 is reinitialized

In the halt condition (but not in the shutdown condition), if maskable interrupts are enabled, an active INTR input will cause the 80386 to end the halt cycle to service the interrupt. The 80386 can service processor extension (PEREQ input) requests and HOLD (HOLD input) requests while in the halt or shutdown condition.

3.1.9 BS16 Cycle

The 80386 can perform data transfers for both 32-bit and 16-bit data buses. A control input, BS16#, allows the bus size to be specified for each bus cycle. This dynamic bus sizing gives the 80386 flexibility in using 16-bit components and buses.

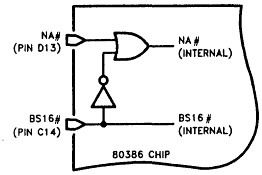
The BS16# input causes the 80386 to perform data transfers for a 16-bit data bus (using data bus signals D15-D0) rather than a 32-bit data bus. The 80386 automatically performs two or three cycles for data transfers larger than 16 bits and for misaligned (odd-addressed) 16-bit transfers.

BS16# must be supplied by external hardware, either through chip select decoding or directly from the addressed device. BS16# is sampled at the start of Phase 2 only in CLK cycle as long as ADS# is not active. If BS16# and READY# are sampled low in the same CLK cycle, the 80386 assumes a 16-bit data bus.

The BS16# control input affects the performance of a data transfer only for data transfers in which 1) BE0# or BE1# is active and 2) BE2# or BE3# is active at the same time. In these transfers, the 80386 must perform two bus cycles using only the lower half of the data bus.

If a BS16 cycle requires an additional bus cycle, the 80386 will retain the current address for the second cycle. Address pipelining cannot be used with BS16 cycles because address pipelining requires that the next address be generated on the bus before the end of the current bus cycle. Therefore, because both signals are sampled at the same sampling window, BS16# must be active before or at the same time as NA# to guarantee 16-bit operation. Once NA# is sampled active in a bus cycle and BS16# is not active at that time, BS16# must be negated for the remainder of the bus cycle.

If BS16# is asserted during the last clock of the bus cycle and NA# was not asserted previously in the bus cycle, then the processor performs a 16-bit bus cycle. This is true, even if NA# is asserted during the last clock of the bus cycle. Figure 3-12 illustrates this logic.



231630-22

Figure 3-12. Internal NA# and BS16# Logic

Figure 3-13 compares the signals for 32-bit and 16-bit bus cycles.

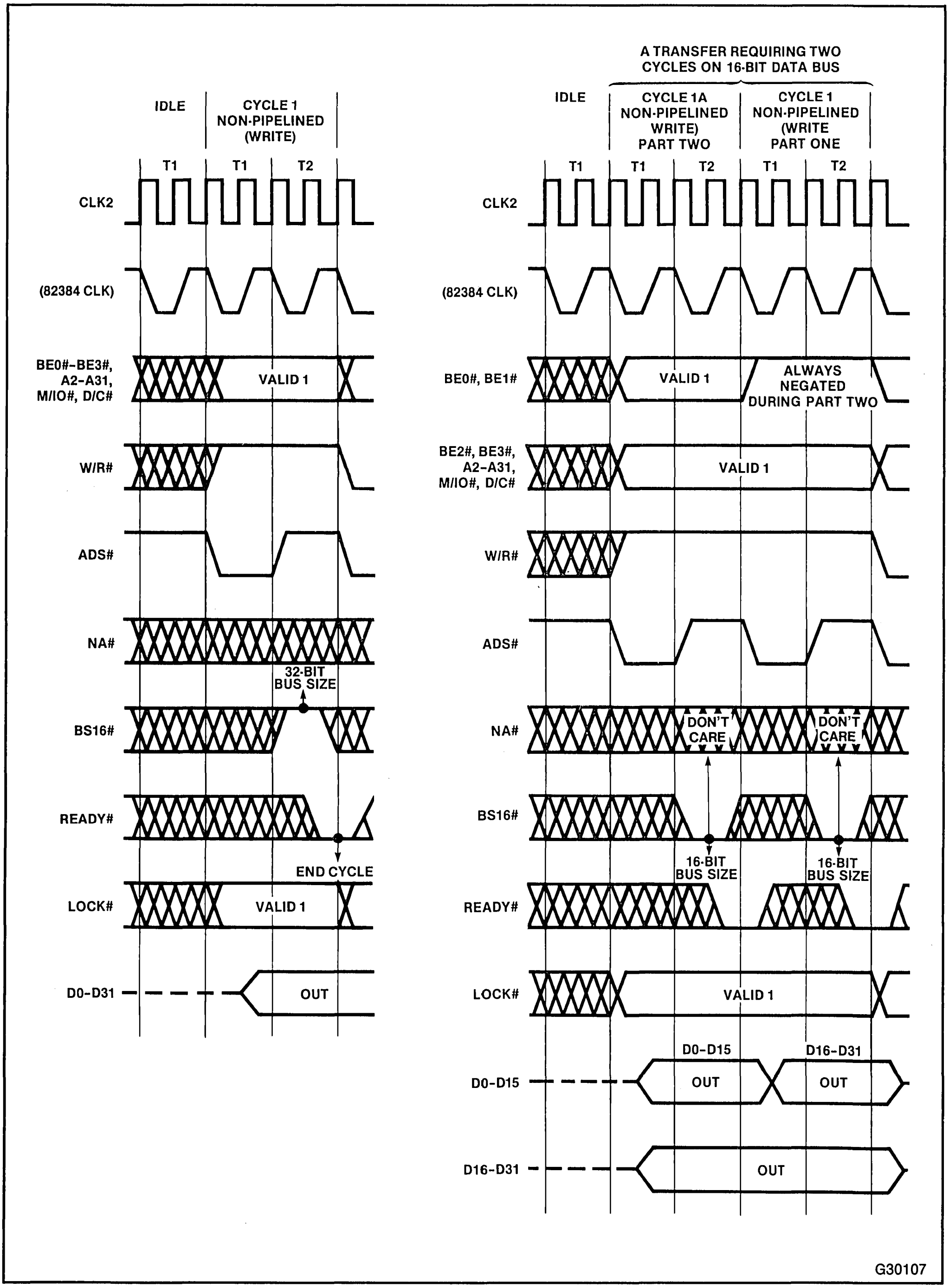
3.1.10 16-Bit Byte Enables and Operand Alignment

For a 16-bit data bus, the 80386 views memory and I/O as sequences of 16-bit words. For this configuration, the Bus High Enable (BHE#), A0 or Bus Low Enable (BLE#), and A1 signals are needed. BHE# and BLE# are byte enables that correspond to two banks of memory in the same way that BE3#-BE0# correspond to four banks. A1 is added to A31-A2 to generate the addresses of 2-byte locations instead of 4-byte locations. Figure 3-14 compares the addressing configurations of 32-bit and 16-bit data buses.

The BHE#, BLE#, and A1 signals can be generated from BE3#, BE2#, BE1#, and BE0# using just four external logic gates. Table 3-5 shows the truth table for this conversion. Note that certain combinations of BE3#-BE0# are never generated.

When BS16# is sampled active, the states of BE3#-BE0# determine how the 80386 responds:

- BS16# has no effect if activated for a bus cycle in which BE3# and BE2# are inactive.
- If BE0# and BE1# are both inactive during a BS16 cycle, and either BE2# or BE3# is active,
  - —For a write cycle, data on D31-D16 is duplicated on D15-D0, regardless of the state of BS16#. (This duplication occurs because BS16# is sampled late in the cycle but data must be available early).
  - —For a read cycle, data that would normally be read on D31-D24 is read on D15-D8, and data that would normally be read on D23-D16 is read on D7-D0.
- If BE0# or BE1# is active, and BE2# or BE3# is active, two bus cycles are required. The two cycles are identical except BE0# and BE1# are inactive in the second cycle and
  - —For a write cycle, the data that was on D31-D16 in the first cycle is copied onto D15-D0.
  - —For a read cycle, data that would normally be read on D31-D24 is read on D15-D8, and data that would normally be read on D23-D16 is read on D7-D0.



IDLE
CYCLE 1 NON-PIPELINED (WRITE)
T1
T1
T2
CLK2
(82384 CLK)
BE0#-BE3#, A2-A31, M/IO#, D/C#
VALID 1
W/R#
ADS#
NA#
32-BIT BUS SIZE
BS16#
READY#
END CYCLE
LOCK#
VALID 1
D0-D31
OUT
A TRANSFER REQUIRING TWO CYCLES ON 16-BIT DATA BUS
IDLE
CYCLE 1A NON-PIPELINED WRITE) PART TWO
CYCLE 1 NON-PIPELINED (WRITE PART ONE
T1
T1
T2
T1
T2
CLK2
(82384 CLK)
BE0#, BE1#
VALID 1
ALWAYS NEGATED DURING PART TWO
BE2#, BE3#, A2-A31, M/IO#, D/C#
VALID 1
W/R#
ADS#
NA#
DON'T CARE
DON'T CARE
BS16#
16-BIT BUS SIZE
16-BIT BUS SIZE
READY#
LOCK#
VALID 1
D0-D15
OUT
D16-D31
OUT
D0-D15
D16-D31
OUT
G30107

Figure 3-13. 32-Bit and 16-Bit Bus Cycle Timing

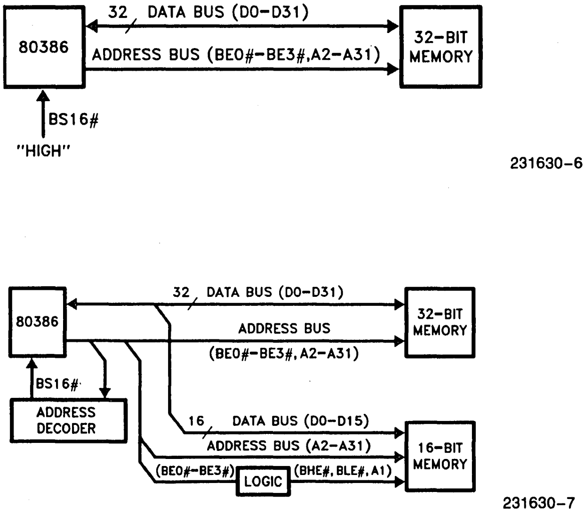

80386
32
DATA BUS (D0-D31)
ADDRESS BUS (BE0#-BE3#,A2-A31)
32-BIT MEMORY
BS16#
"HIGH"
231630-6
80386
32
DATA BUS (D0-D31)
ADDRESS BUS
(BE0#-BE3#, A2-A31)
32-BIT MEMORY
BS16#
ADDRESS DECODER
16
DATA BUS (D0-D15)
ADDRESS BUS (A2-A31)
(BE0#-BE3#)
LOGIC
(BHE#, BLE#, A1)
16-BIT MEMORY
231630-7

Figure 3-14. 32-Bit and 16-Bit Data Addressing

Table 3-6 shows which combinations of BE3#-BE0# require two bus cycles and the states of BE3#-BE0# for each cycle.

In some cases, 16-bit cycles may be performed without using BS16#. Address pipelining may be used as follows for these cycles.

- BS16# is not needed for cycles that use only D15-D0.
- BS16# is not needed for a word-aligned 16-bit write. For write cycles, all 32 bits of the data bus are driven regardless of bus size.

3.2 BUS TIMING

This section describes timing requirements for read cycles, write cycles, and the READY# signal.

All 80386 signals have setup and hold time requirements relative to CLK2. The timings of certain signals relative to one another depends on whether address pipelining is used. These facts must be considered when determining external logic needed to facilitate bus cycles.

Table 3-5. Generation of BHE#, BLE#, and A1 from Byte Enables

80386 Signals				16-Bit Bus Signals			Comments
BE3#	BE2#	BE1#	BE0#	A1	BHE#	BLE# (A0)	
H*	H*	H*	H*	x	x	x	x—no active bytes
H	H	H	L	L	H	L	
H	H	L	H	L	L	H	
H	H	L	L	L	L	L	
H	L	H	H	H	H	L	
H*	L*	H*	L*	x	x	x	x—not contiguous bytes
H	L	L	H	L	L	H	
H	L	L	L	L	L	L	
L	H	H	H	H	L	H	
L*	H*	H*	L*	x	x	x	x—not contiguous bytes
L*	H*	L*	H*	x	x	x	x—not contiguous bytes
L*	H*	L*	L*	x	x	x	x—not contiguous bytes
L	L	H	H	H	L	L	
L*	L*	H*	L*	x	x	x	x—not contiguous bytes
L	L	L	H	L	L	H	
L	L	L	L	L	L	L	

BLE# asserted when D0–D7 of 16-bit bus is active.
BHE# asserted when D8–D15 of 16-bit bus is active.
A1 low for all even words; A1 high for all odd words.

Key:
x = don't care
H = high voltage level
L = low voltage level
* = a non-occurring pattern of Byte Enables; either none are asserted, or the pattern has Byte Enables asserted for non-contiguous bytes

Table 3-6. Byte Enables during BS16 Cycles

First Cycle:				Second Cycle:			
BE3#	BE2#	BE1#	BE0#	BE3#	BE2#	BE1#	BE0#
High	High	High	Low	None			
High	High	Low	High	None			
High	High	Low	Low	None			
High	Low	High	High	None			
High	Low	Low	High	High	Low	High	High
High	Low	Low	Low	High	Low	High	High
Low	High	High	High	None			
Low	Low	High	High	None			
Low	Low	Low	High	Low	Low	High	High
Low	Low	Low	Low	Low	Low	High	High

The analyses that follow are based on the assumption that a 16-MHz 80386 is used. If the processor is operated at a different frequency, the timings will change accordingly. Example worst-case signal parameter values from the *80386 Data Sheet* (Order Number 231630) are used; **consult the most recent data sheet to confirm these values.** Also note that delay times and setup times must be factored into the timing of system response and interaction with the 80386 to ensure comfortable margins for all critical timings.

3.2.1 Read Cycle Timing

For read cycles, the minimum amount of time from the output of valid addresses to the reading of the data bus sets an upper limit on memory access times (including address decoding time). In a non-pipelined address cycle, this time is

Four CLK2 cycles	125 nanoseconds
− A31-A2 output delay (maximum)	−38 nanoseconds
− D31-D0 input setup (minimum)	−10 nanoseconds
	77 nanoseconds

With address pipelining and no wait states, the address is valid one CLK cycle earlier:

Non-pipelined value	77 nanoseconds
+ One CLK cycle (2 CLK2 cycles)	+62.5 nanoseconds
	139.5 nanoseconds

For both cases above, each wait state in the bus cycle adds 62.5 nanoseconds.

3.2.2 Write Cycle Timing

For write cycles, the elapsed time from the output of valid address to the end of the cycle determines how quickly the external logic must decode and latch the address. In a non-pipelined address cycle, this time is

Four CLK2 cycles	125 nanoseconds
− A31-A2 output delay (maximum)	−38 nanoseconds
	87 nanoseconds
(With address pipelining)	(+ 62.5 nanoseconds)
(With N wait states)	(+ N*62.5 nanoseconds)

The minimum amount of time from the output of valid write data by the access device to the end of the write cycle is the least amount of time external logic has to read the data. This setup time is

Three CLK2 cycles	93.75 nanoseconds
− D31-D0 output delay (maximum)	−50 nanoseconds
	43.75 nanoseconds
(With N wait states)	(+ N*62.5 nanoseconds)

Data outputs are valid beyond the end of the bus cycle. This data hold time is at least

One CLK2 cycle	31.25 nanoseconds
+D31-D0 hold time (minimum)	+ 1 nanoseconds
	32.25 nanoseconds

(Wait states do not affect this parameter)

Wait states add the same amounts of data-to-end-of-cycle time as they do for read cycles. (See Section 3.2.1.)

3.2.3 READY# Signal Timing

The amount of time from the output of valid address signals to the assertion of READY# to end a bus cycle determines how quickly external logic must generate the READY# signal. READY# must meet the 80386 setup time. In a nonpipelined address cycle, READY# signal timing is as follows:

Four CLK2 cycles	125 nanoseconds
− A31-A2 output delay (maximum)	−38 nanoseconds
−READY# setup (minimum)	−20 nanoseconds
	67 nanoseconds
(With address pipelining)	(+ 62.5 nanoseconds)
(With N wait states)	(+ N*62.5 nanoseconds)

Again, pipelining and wait states increase this amount of time.

Because the efficiency of a cache depends upon quick turnaround of cache hits (i.e., when requested data is found in the cache) the timing of the READY# signal is critical; therefore, READY# is typically generated combinationally from the cache hit comparator. If the READY# signal is returned too slowly, the speed advantage of the cache is lost.

3.3 CLOCK GENERATION

3.3.1 82384 Clock Generator

The 82384 Clock Generator is a multifunction component of the 80386 system that provides clocking for synchronous operation of the 80386 and its support components as follows:

- Both CLK2 (a double-frequency clock for the 80386 and some support devices) and CLK (a system clock for some 80386 support devices) are generated. The phase of the 82384 CLK matches that of the CLK signal generated internally by the 80386.
- The RESET signal for the 80386 and other system components is generated. The RES# input of the 82384 accepts an asynchronous input from a simple RC circuit or similar source, synchronizes the signal with CLK, and outputs the active-high RESET signal to the 80386 and other system components. The timing and function of the RESET signal with respect to the 80386 is discussed later in this chapter.
- The 82384 uses the ADS# output of the 80386 (which guarantees setup and hold time to CLK2) to generate an ADSO# signal, which is functionally equivalent to ADS# but guarantees setup and hold times with respect to CLK. Devices that are clocked with the CLK output of the 82384 can use ADSO#.

Figure 3-15 shows a typical circuit to connect the 82384 to the 80386.

Either an external frequency source or a third overtone crystal can be used to drive the 82384. The F/C# input indicates the signal source; if F/C# is pulled high, the 82384 recognizes the signal on its External Frequency Input (EFI) pin as its frequency source. If F/C# is tied low, the crystal connected to the X1 and X2 pins of the 82384 is its frequency source. In either case, the source frequency must equal the desired CLK2 frequency. For example, a 32 MHz crystal yields a 32 MHz CLK2 signal and a 16 MHz 80386 internal clock rate.

3.3.2 Clock Timing

The CLK2 and CLK outputs of the 82384 are both MOS-level outputs with output high voltage levels of $V_{CC}-0.6V$ and adequate drive for TTL inputs. CLK2 is twice the frequency of CLK.

The internal CLK signal of the 80386 is matched to the CLK output of the 82384 by the falling edge of the RESET signal. This operation is described with the RESET function in Section 3.8.

The skew between the 82384 CLK2 and CLK signals is maintained at 0-16 nanoseconds (regardless of clock frequency). For closely timed interfaces, peripheral devices must be timed by CLK2. Devices that cannot be operated at the double-clock frequency must use the CLK output of the 82384. The 80386 interface to these devices must allow for the CLK2-to-CLK skew.

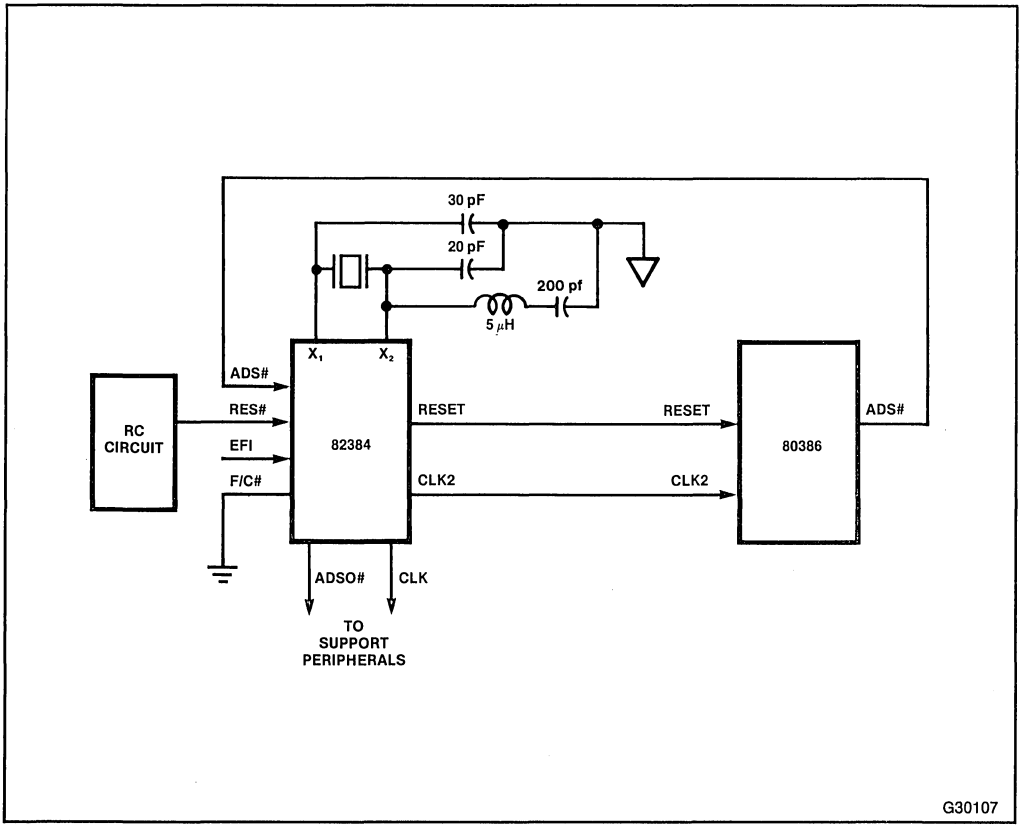


Figure 3-15. Connecting 82384 to 80386

The phase of the CLK output of the 82384 is useful for determining the beginning of a bus cycle. Because each CLK2 cycle is 31.25 nanoseconds (at CLK2 = 32 MHz), and bus status signal delays may be as much as 35 nanoseconds, it is impossible to tell from these status signals alone which CLK2 cycle begins the bus cycle, and therefore when to expect valid address signals. The phase of CLK can be used to make this determination. ADS# should be sampled on rising CLK2 transitions when CLK is high, i.e., at the end of phase 2 (see Figure 3-16).

3.3.3 Crystal Oscillator Clock Generator

Figure 3-17 shows a clock generator circuit for the 80386. It is implemented with TTL and CMOS components and is functionally similar to the 82384 clock generator. This circuit replaces the 82384 and is recommended for new designs. It provides the CLK2, CLK, and RESET signals needed by the 80386 and local bus controller. The CLK2 signal provides the fundamental timing for the 80386 and is generated by a CMOS crystal oscillator. This oscillator must have a CMOS output buffer guaranteed compatible with the 80386 CLK2 input.

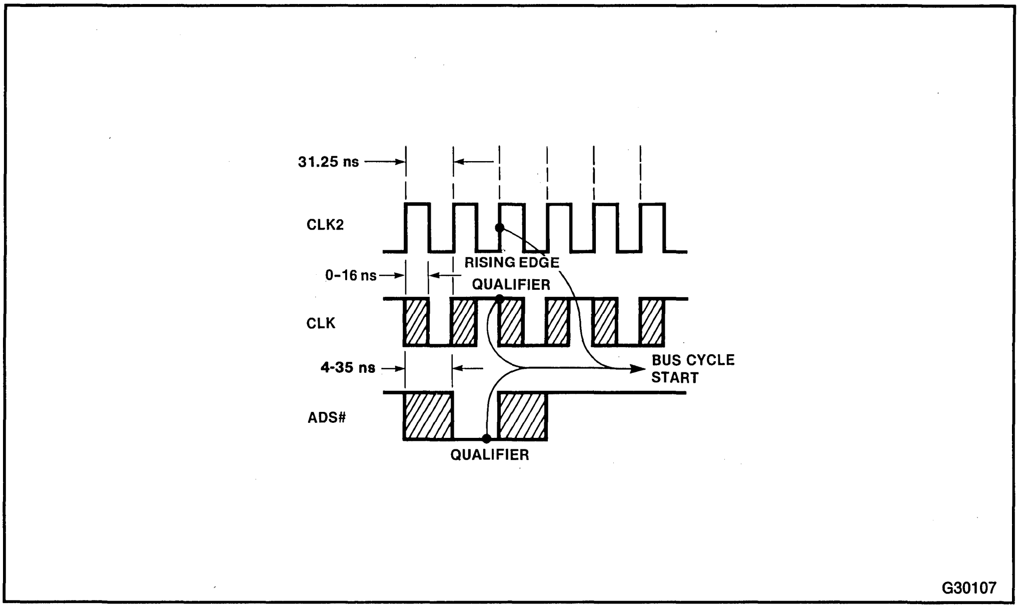

31.25 ns
CLK2
0-16 ns
RISING EDGE QUALIFIER
CLK
4-35 ns
BUS CYCLE START
ADS#
QUALIFIER
G30107

Figure 3-16. Using CLK to Determine Bus Cycle Start

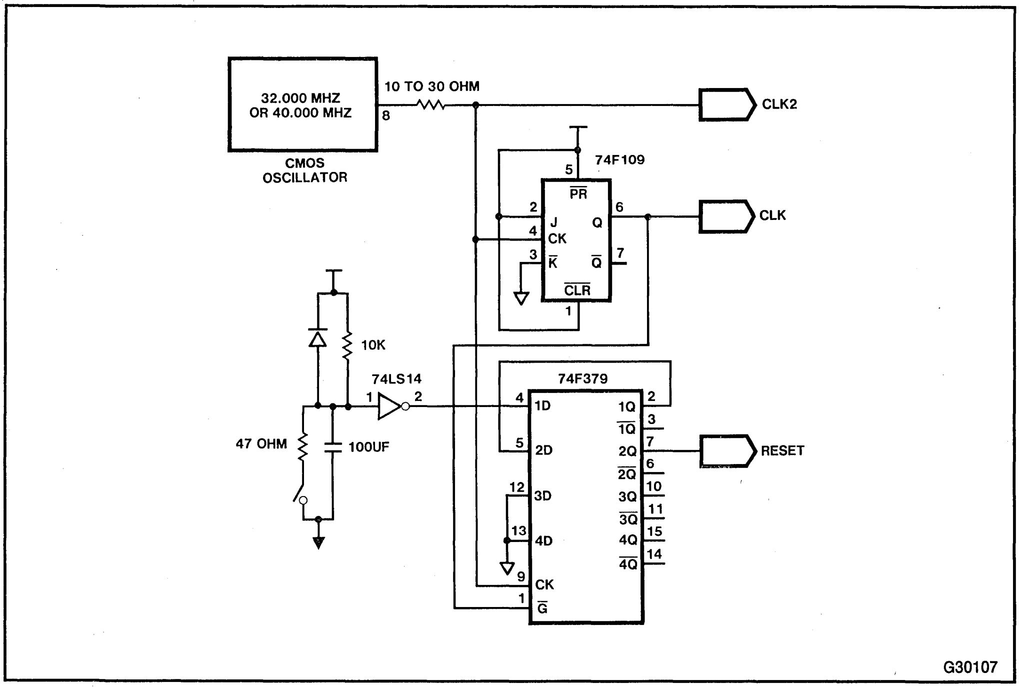

32.000 MHZ OR 40.000 MHZ
8
CMOS OSCILLATOR
10 TO 30 OHM
CLK2
74F109
5
PR
2
J
4
CK
3
K
Q
6
Q
7
CLR
1
CLK
10K
74LS14
1
2
47 OHM
100UF
74F379
4
1D
5
2D
12
3D
13
4D
9
CK
1
G
1Q
2
1Q
3
2Q
7
2Q
6
3Q
10
3Q
11
4Q
15
4Q
14
RESET
G30107

Figure 3-17. Clock Generator

A 74F109 flip-flop divides CLK2 by two to generate CLK. CLK has the same phase as the 80386 internal clock, going low during phase 1 and high during phase 2. A 74F379 generates a synchronous RESET output, ensuring that the falling edge of RESET occurs during phase 2.

The recommended circuit does not implement the ADS# synchronization feature of the 82384. The ADS# synchronizer is not necessary if all registered PALs in the local bus controller use CLK2. Clocking PALs with CLK2 (rather than CLK) is the preferred method since it minimizes skew between the processor and its surrounding logic. If it is necessary to have an address status signal synchronous to CLK, then an ADS# synchronizer may be built using a delay line and flip-flop, as shown in Figure 3-18.

An alternative method of generating CLK2 is to use a TTL oscillator coupled to a 74ACT244 buffer. Although a typical 74ACT244 datasheet does not guarantee an output compatible with the 80386 CLK2 input, some manufacturers devices have been observed to have sufficiently fast rise/fall times (4 ns at CL=80 pf), as well as the necessary swing, to meet the 80386 CLK2 requirements. This approach is recommended if the 80386 must run synchronously with an external clock (i.e. EFI). Similarly, if a CMOS level swing is required on CLK, a 74AC109 flip-flop may be used instead of the 74F109.

3.4 INTERRUPTS

Both hardware-generated and software-generated interrupts can alter the programmed execution of the 80386. A hardware-generated interrupt occurs in response to an active input on one of two 80386 interrupt request inputs (NMI or INTR). A software-generated interrupt occurs in response to an INT instruction or an exception (a software condition that

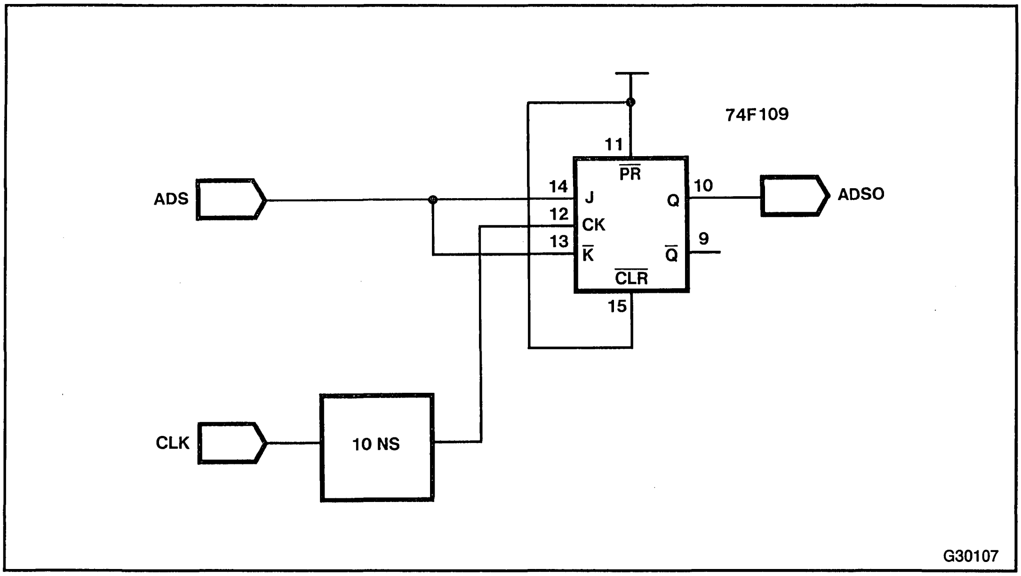


Figure 3-18. ADS# Synchronizer

requires servicing). For complete information on software-generated interrupts, see the *80386 Programmer's Reference Manual.*

In response to an interrupt request, the 80386 processes the interrupt (saves the processor state on the stack, plus task information if a task switch is required) and services the interrupt (transfers program execution to one of 256 possible interrupt service routines). Entry-point descriptors to service routines or interrupt tasks are stored in a table (Interrupt Descriptor Table or IDT) in memory. To access a particular service routine, the 80386 must obtain a vector, or index, to the table location that contains the corresponding descriptor. The source of this vector depends on the type of interrupt; if the interrupt is maskable (INTR input active), the vector is supplied by the 8259A Interrupt Controller. If the interrupt is nonmaskable (NMI input active), location 2 in the IDT is used automatically.

The NMI request and the INTR request differ in that the 80386 can be programmed to ignore INTR requests (by clearing the interrupt flag of the 80386). An NMI request always provokes a response from the 80386 unless the 80386 is already servicing a previous NMI request. In addition, an INTR request causes the 80386 to perform two interrupt-acknowledge bus cycles to fetch the service-routine vector. These bus cycles are not required for an NMI request, because the vector location for an NMI request is fixed.

Under the following two conditions a service routine will not be interrupted by an incoming interrupt:

- The incoming interrupt is an INTR request, and the 80386 is programmed to ignore maskable interrupts. (The 80386 is automatically programmed to ignore maskable interrupts when it receives any interrupt request. This condition may be changed by the interrupt service routine.) In this case, the INTR request will be serviced only if it is still active when maskable interrupts are reenabled.
- The incoming interrupt is an NMI, and the 80386 is servicing a previous NMI. In this case, the NMI is saved automatically to be processed after the IRET instruction in the NMI service routine has been executed. Only one NMI can be saved; any others that occur while the 80386 is servicing a previous NMI will not be recognized.

If neither of the above conditions is true, and an interrupt occurs while the 80386 is servicing a previous interrupt, the new interrupt is processed and serviced immediately. The 80386 then continues with the previous service routine. The last interrupt processed is the first one serviced.

If an NMI request and an INTR request arrive at the 80386 simultaneously, the NMI request is processed first. Multiple hardware interrupts arriving at the 8259A are processed according to their priority and are sent to the 80386 INTR input one at a time.

3.4.1 Non-Maskable Interrupt (NMI)

The NMI input of the 80386 generally signals a catastrophic event, such as an imminent power loss, a memory error, or a bus parity error. This input is edge-triggered (on a low-to-high transition) and asynchronous. A valid signal is low for eight CLK2 periods before the transition and high eight CLK2 periods after the transition. The NMI signal can be asynchronous to CLK2.

An NMI request automatically causes the 80386 to execute the service routine corresponding to location 2 in the IDT. The 80386 will not service subsequent NMI requests until the current request has been serviced. The 80386 disables INTR requests (although these can be reenabled in the service routine) in Real Mode. In Protected Mode, the disabling of INTR requests depends on the gate in IDT location 2.

3.4.2 Maskable Interrupt (INTR)

The INTR input of the 80386 allows external devices to interrupt 80386 program execution. To ensure recognition by the 80386, the INTR input must be held high until the 80386 acknowledges the interrupt by performing the interrupt acknowledge sequence. The INTR input is sampled at the beginning of every instruction; it must be high at least eight CLK2 periods prior to the instruction to guarantee recognition as a valid interrupt. This requirement reduces the possibility of false inputs from voltage glitches. In addition, maskable interrupts must enabled in software for interrupt recognition. The INTR input may be asynchronous to CLK2.

The INTR signal is usually supplied by the 8259A Programmable Interrupt Controller, which in turn is connected to devices that require interrupt servicing. The 8259A, which is controlled by commands from the 80386 (the 8259A appears as a set of I/O ports), accepts interrupt requests from devices connected to the 8259A, determines the priority for transmitting the requests to the 80386, activates the INTR input, and supplies the appropriate service routine vector when requested.

An INTR request causes the 80386 to execute two back-to-back interrupt acknowledge bus cycles, as described earlier in Section 3.1.4.

3.4.3 Interrupt Latency

The time that elapses before an interrupt request is serviced (interrupt latency) varies according to several factors. This delay must be taken into account by the interrupt source. Any of the following factors can affect interrupt latency:

- If interrupts are masked, an INTR request will not be recognized until interrupts are reenabled.
- If an NMI is currently being serviced, an incoming NMI request will not be recognized until the 80386 encounters the IRET instruction.
- If the 80386 is currently executing an instruction, the instruction must be completed. An interrupt request is recognized only on an instruction boundary. (However, Repeat String instructions can be interrupted after each iteration.)
- Saving the Flags register and CS:EIP registers (which contain the return address) requires time.
- If interrupt servicing requires a task switch, time must be allowed for saving and restoring registers.
- If the interrupt service routine saves registers that are not automatically saved by the 80386, these instructions also delay the beginning of interrupt servicing.

The longest latency occurs when the interrupt request arrives while the 80386 is executing a long instruction such as multiplication, division, or a task-switch in the Protected mode.

If the instruction loads the Stack Segment register, an interrupt is not processed until after the following instruction, which should be an ESP load. This allows the entire stack pointer to be loaded without interruption.

If an instruction sets the interrupt flag (thereby enabling interrupts), an interrupt is not processed until after the next instruction.

3.5 BUS LOCK

In a system in which more than one device may control the local bus, locked cycles must be used when it is critical that two or more bus cycles follow one another immediately. Otherwise, the cycles can be separated by a cycle from another bus master.

Any bus cycles that must be performed back-to-back without any intervening bus cycles by other bus masters should be locked. The use of a semaphore is one example of this precept. The value of a semaphore indicates a condition, such as the availability of a device. If the 80386 reads a semaphore to determine that a device is available, then writes a new value to the semaphore to indicate that it intends to take control of the device, the read cycle and write cycle should be locked to prevent another bus master from reading from or writing to the semaphore in between the two cycles. The erroneous condition that could result from unlocked cycles is illustrated in Figure 3-19.

The LOCK# output of the 80386 signals the other bus masters that they may not gain control of the bus. In addition, an 80386 with LOCK# asserted will not recognize a HOLD request from another bus master.

3.5.1 Locked Cycle Activators

The LOCK# signal is activated explicitly by the LOCK prefix on certain instructions. LOCK# is also asserted automatically for an XCHG instruction, a descriptor update, interrupt acknowledge cycles, and a page table update.

3.5.2 Locked Cycle Timing

LOCK# is activated on the CLK2 edge that begins the first locked bus cycle. LOCK# is deactivated when READY# is sampled low at the end of the last bus cycle to be locked.

LOCK# is activated and deactivated on these CLK2 edges whether or not address pipelining is used. If address pipelining is used, LOCK# will remain active until after the address bus and bus cycle status signals have been asserted for the pipelined cycle. Consequently, the LOCK# signal can extend into the next memory access cycle that does not need to be locked. (See Figure 3-20.) The result is that the use of the bus by another bus master is delayed by one bus cycle.

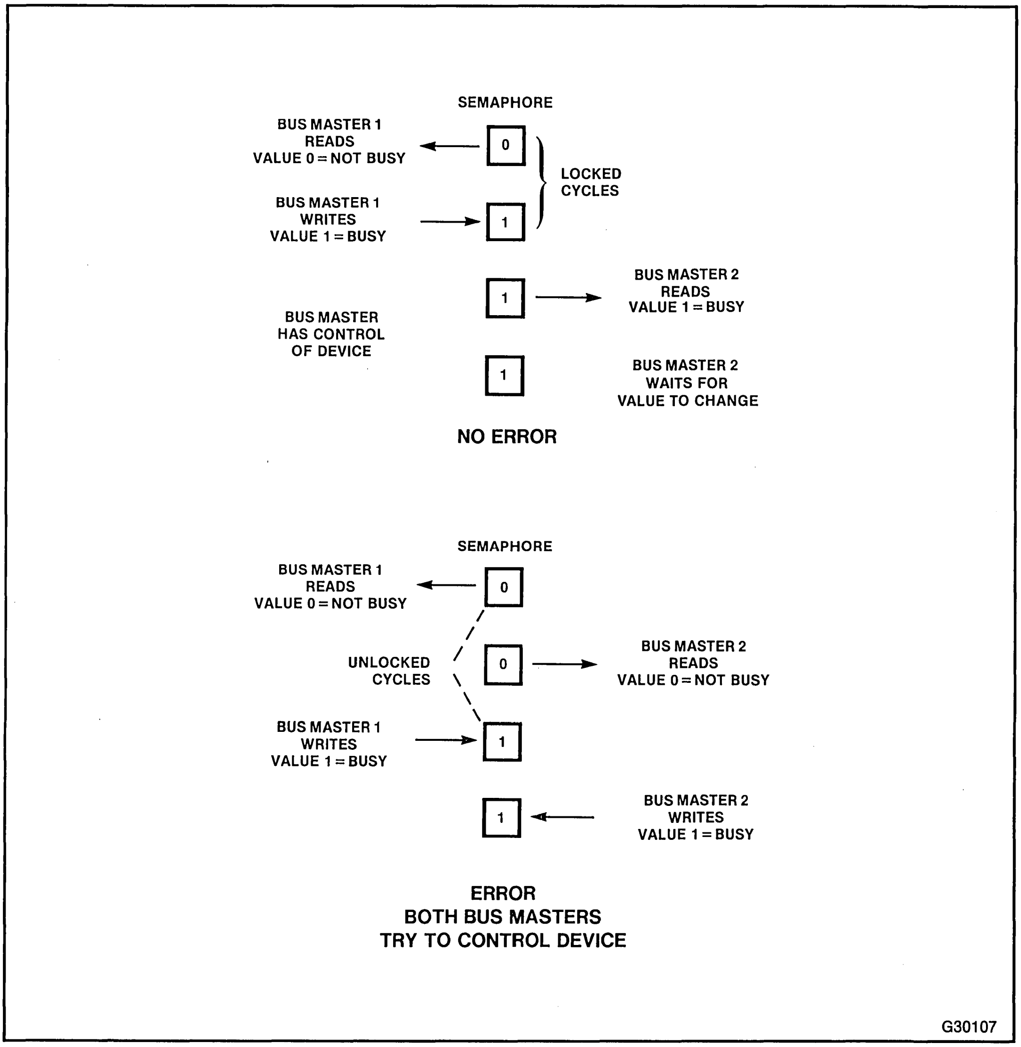

SEMAPHORE
BUS MASTER 1 READS VALUE 0 = NOT BUSY
0
LOCKED CYCLES
BUS MASTER 1 WRITES VALUE 1 = BUSY
1
1
BUS MASTER 2 READS VALUE 1 = BUSY
BUS MASTER HAS CONTROL OF DEVICE
1
BUS MASTER 2 WAITS FOR VALUE TO CHANGE
NO ERROR
SEMAPHORE
BUS MASTER 1 READS VALUE 0 = NOT BUSY
0
UNLOCKED CYCLES
0
BUS MASTER 2 READS VALUE 0 = NOT BUSY
BUS MASTER 1 WRITES VALUE 1 = BUSY
1
1
BUS MASTER 2 WRITES VALUE 1 = BUSY
ERROR BOTH BUS MASTERS TRY TO CONTROL DEVICE
G30107

Figure 3-19. Error Condition Caused by Unlocked Cycles

3.5.3 LOCK# Signal Duration

The maximum duration of the LOCK# signal affects the maximum HOLD request latency because HOLD is not recognized until LOCK# goes inactive. The duration of LOCK# depends on the instruction being executed and the number of wait states per cycle.

The longest duration of LOCK# in real mode is two bus cycles plus approximately two clocks. This occurs during the XCHG instruction and in LOCKed read-modify-write operations. The longest duration of LOCK# in protected mode is five bus cycles plus approximately fifteen clocks. This occurs when an interrupt (hardware or software interrupt) occurs

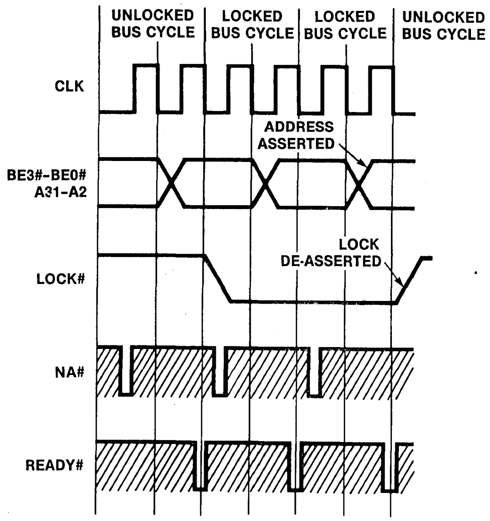

UNLOCKED BUS CYCLE
LOCKED BUS CYCLE
LOCKED BUS CYCLE
UNLOCKED BUS CYCLE
CLK
ADDRESS ASSERTED
BE3#-BE0#
A31-A2
LOCK DE-ASSERTED
LOCK#
NA#
READY#
G30107

Figure 3-20. LOCK# Signal during Address Pipelining

and the 80386 performs a LOCKed read of the gate in the IDT (8 bytes), a read of the target descriptor (8 bytes), and a write of the accessed bit in the target descriptor.

3.6 HOLD/HLDA (Hold Acknowledge)

The 80386 provides on-chip arbitration logic that supports a protocol for transferring control of the local bus to other bus masters. This protocol is implemented through the HOLD input and HLDA output.

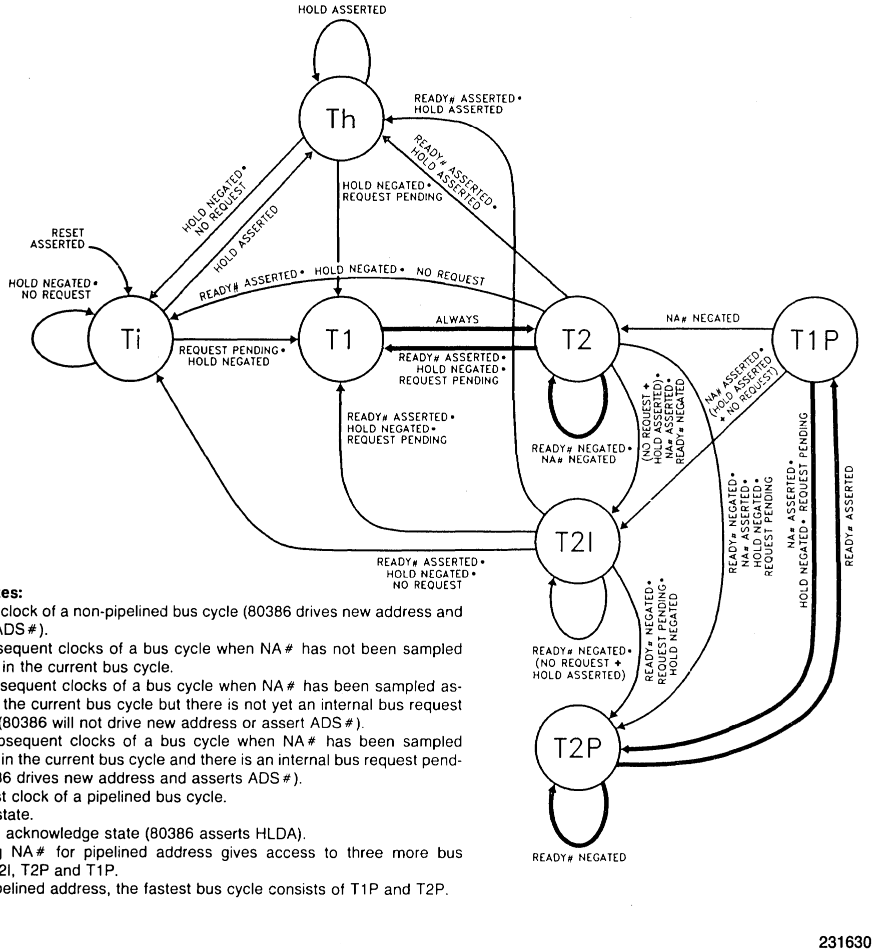
3.6.1 HOLD/HLDA Timing

To gain control of the local bus, the requesting bus master drives the 80386 HOLD input active. This signal must be synchronous to the CLK2 input of the 80386. The 80386 responds by completing its current bus cycle (plus a second locked cycle or a second cycle required by BS16#). Then the 80386 sets all outputs but HLDA to the three-state OFF condition to effectively remove itself from the bus and drives HLDA active to signal the requesting bus master that it may take control of the bus.

The requesting bus master must maintain HOLD active until it no longer needs the bus. When HOLD goes low, the 80386 drives HLDA low and begins a bus cycle (if one is pending).

For valid system operation, the requesting bus master must not take control of the bus before it receives the HLDA signal and must remove itself from the bus before de-asserting the HOLD signal. Setup and hold times relative to CLK2 for both rising and falling transitions of the HOLD signal must be met.

When the 80386 receives an active HOLD input, it completes the current bus cycle before relinquishing control of the bus. Figure 3-21 shows the state diagram for the bus including the HOLD state.


HOLD ASSERTED
Th
READY# ASSERTED • HOLD ASSERTED
READY# HOLD ASSERTED • ASSERTED
HOLD NEGATED • NO REQUEST
HOLD ASSERTED
HOLD NEGATED • REQUEST PENDING
RESET ASSERTED
HOLD NEGATED • NO REQUEST
READY# ASSERTED • HOLD NEGATED • NO REQUEST
Ti
T1
T2
T1P
ALWAYS
NA# NEGATED
REQUEST PENDING • HOLD NEGATED
READY# ASSERTED • HOLD NEGATED • REQUEST PENDING
READY# ASSERTED • HOLD NEGATED • REQUEST PENDING
READY# NEGATED • NA# NEGATED
(NO REQUEST + HOLD ASSERTED) • NA# ASSERTED • READY# NEGATED
NA# ASSERTED • (HOLD ASSERTED • NO REQUEST)
READY# NEGATED • NA# ASSERTED • HOLD NEGATED • REQUEST PENDING
NA# ASSERTED • HOLD NEGATED • REQUEST PENDING
READY# ASSERTED
T2I
READY# ASSERTED • HOLD NEGATED • NO REQUEST
READY# NEGATED • (NO REQUEST + HOLD ASSERTED)
READY# NEGATED • REQUEST PENDING • HOLD NEGATED
T2P
READY# NEGATED

Bus States:

T1—first clock of a non-pipelined bus cycle (80386 drives new address and asserts ADS#).

T2—subsequent clocks of a bus cycle when NA# has not been sampled asserted in the current bus cycle.

T2I—subsequent clocks of a bus cycle when NA# has been sampled asserted in the current bus cycle but there is not yet an internal bus request pending (80386 will not drive new address or assert ADS#).

T2P—subsequent clocks of a bus cycle when NA# has been sampled asserted in the current bus cycle and there is an internal bus request pending (80386 drives new address and asserts ADS#).

T1P—first clock of a pipelined bus cycle.

Ti—idle state.

Th—hold acknowledge state (80386 asserts HLDA).

Asserting NA# for pipelined address gives access to three more bus states: T2I, T2P and T1P.

Using pipelined address, the fastest bus cycle consists of T1P and T2P.

231630-24

Figure 3-21. Bus State Diagram with HOLD State

During the HOLD state, the 80386 can continue executing instructions in its Prefetch Queue. Program execution is delayed if a read cycle is needed while the 80386 is in the HOLD state. The 80386 can queue one write cycle internally, pending the return of bus access; if more than one write cycle is needed, program execution is delayed until HOLD is released and the 80386 regains control of the bus.

HOLD has priority over most bus cycles, but HOLD is not recognized between two interrupt acknowledge cycles, between two repeated cycles of a BS16 cycle, or during locked cycles. For the 80386, HOLD is recognized between two cycles required for misaligned data transfers; for the 8086 and 80286 HOLD it is not recognized. This difference should be considered if critical misaligned data transfers are not locked.

HOLD is not recognized while RESET is active, but is recognized during the time between the high-to-low transition of RESET and the first instruction fetch.

All inputs are ignored while the 80386 is in the HOLD state, except for the following:

- HOLD is monitored to determine when the 80386 may regain control of the bus.
- RESET takes precedence over the HOLD state. An active RESET input will reinitialize the 80386.
- One NMI request is recognized and latched. It is serviced after HOLD is released.

3.6.2 HOLD Signal Latency

Because other bus masters such as DMA controllers are typically used in time-critical applications, the amount of time the bus master must wait (latency) for bus access can be a critical design consideration.

The minimum possible latency occurs when the 80386 receives the HOLD input during an idle cycle. HLDA is asserted on the CLK2 rising edge following the HOLD active input (synchronous to CLK2).

Because a bus cycle must be terminated before HLDA can go active, the maximum possible latency occurs when a bus-cycle instruction is being executed. Wait states increase latency, and HOLD is not recognized between locked bus cycles, repeated cycles due to BS16#, and interrupt acknowledge cycles.

3.6.3 HOLD State Pin Conditions

LOCK#, M/IO#, D/C#, W/R#, ADS#, A31-A2, BE3#-BE0#, and D31-D0 enter the three-state OFF condition in the HOLD state. Note that external pullup resistors may be required on ADS#, LOCK# and other signals to guarantee that they remain inactive during transitions between bus masters.

3.7 RESET

RESET starts or restarts the 80386. When the 80386 detects a low-to-high transition on RESET, it terminates all activities. When RESET goes low again, the 80386 is initialized to a known internal state and begins fetching instructions from the reset address.

3.7.1 RESET Timing

The 82384 Clock Generator generates the RESET signal to initialize the 80386 and other system components. The 82384 has a Schmitt-trigger RES# input signal used to generate the RESET signal from an active-low pulse. The hysteresis on the RES# input prevents the RESET output from entering an indeterminate state, so a simple RC circuit can be used to generate the RES# input on power-up. Figure 3-22 shows an RC circuit that satisfies timing requirements for the RES# input.

The RESET input of the 80386 must remain high for at least 15 CLK2 periods to ensure proper initialization (at least 80 CLK2 periods if self-test is to be performed). The CLK output of the 82384 is initialized with the rising edge of RESET. When RESET goes low, the 80386 adjusts the falling edge of its internal clock (CLK) to coincide with the start of the first CLK2 cycle after the high-to-low transition of RESET. The 82384 times the high-to-low edge of RESET (synchronous to CLK2) so that the phase of the internal CLK of the 80386 matches the phase of the CLK output of the 82384. This relationship is shown in Figure 3-23.

On the high-to-low transition of RESET, the BUSY# pin is sampled. If BUSY# is low, the 80386 will perform a self-test lasting approximately $2^{20} + 60$ CLK2 cycles before it begins executing instructions. The 80386 continues with initialization after the test, regardless of the test results.

The 80386 fetches its first instruction from linear address 0FFFFFFF0H, sometime between 350 and 450 CLK2 cycles after the high-to-low transition of RESET (or, if self-test is

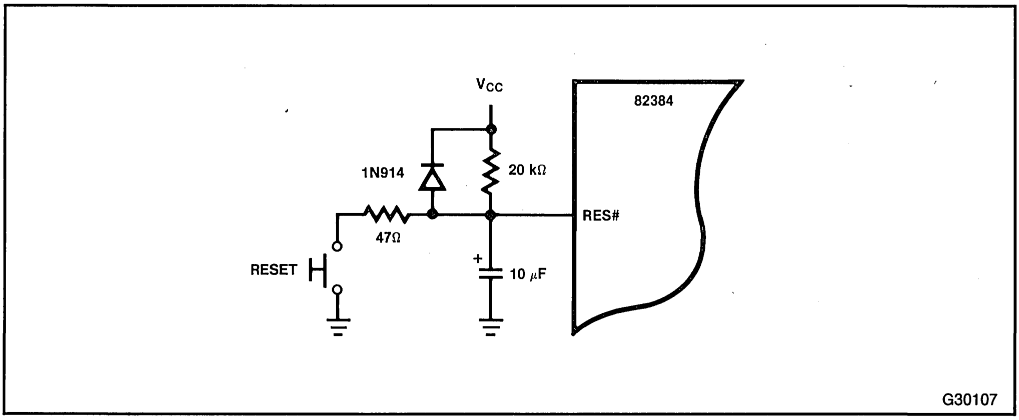


Figure 3-22. Typical RC RESET Timing Circuit

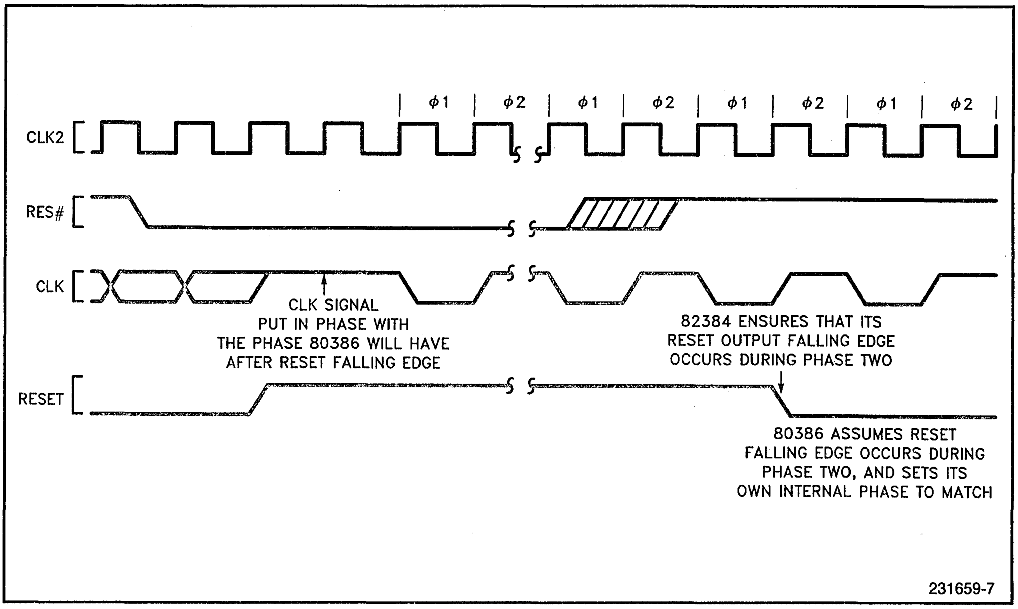

φ1
φ2
CLK2
RES#
CLK
CLK SIGNAL PUT IN PHASE WITH THE PHASE 80386 WILL HAVE AFTER RESET FALLING EDGE
82384 ENSURES THAT ITS RESET OUTPUT FALLING EDGE OCCURS DURING PHASE TWO
RESET
80386 ASSUMES RESET FALLING EDGE OCCURS DURING PHASE TWO, AND SETS ITS OWN INTERNAL PHASE TO MATCH
231659-7

Figure 3-23. RESET, CLK, and CLK2 Timing

performed, after completion of self-test). Because paging is disabled, linear address 0FFFFFFF0H is the same as physical address 0FFFFFFF0H. This location normally contains a JMP instruction to the beginning of the bootstrap program.

3.7.2 80386 Internal States

RESET should be kept high for at least one millisecond after V_{CC} and CLK2 have reached their DC and AC specifications.

The 80386 samples its ERROR# input during initialization to determine the type of processor extension present in the system. This sampling occurs at some time at least 20 CLK2 periods after the high-to-low transition of RESET and before the first instruction fetch. If the ERROR# input is low, the 80386 assumes that an 80387 numeric coprocessor is being used, and the programmer must issue a command (FINIT) to reset the ERROR# input after initialization. If ERROR# is high, an 80287 or no processor extension is assumed. In this case, a software test must be run to determine whether an 80287 is actually present and to set the corresponding flag in the 80386. This test is described in Chapter 5.

3.7.3 80386 External States

RESET causes the 80386 output pins to enter the states shown in Table 3-7. Data bus pins enter the three-state condition.

Prior to its first instruction fetch, the 80386 makes no internal requests to the bus, and, therefore, will relinquish bus control if it receives a HOLD request (see Section 3.7 for a complete description of HOLD cycles).

Interrupt requests (INTR and NMI) are not recognized before the first instruction fetch.

Table 3-7. Output Pin States during RESET

Pin Name	Pin State
LOCK#, D/C#, ADS#, A31-A2	High
W/R#, M/IO#, HLDA, BE3#-BE0#	Low
D31-D0	Three-State

Performance Considerations

4

CHAPTER 4
PERFORMANCE CONSIDERATIONS

System performance measures how fast a microprocessing system performs a given task or set of instructions. Through increased processing speed and data throughput, an 80386 operating at the heart of a system can improve overall performance immensely. The design of supporting logic and devices for efficient interaction with the 80386 is also important in optimizing system performance.

This chapter describes considerations for achieving high performance in 80386-based systems. A variety of examples illustrate the potential performance levels for a number of applications.

4.1 WAIT STATES AND PIPELINING

Because a system may include devices whose response is slow relative to the 80386 bus cycle, the overall system performance is often less than the potential performance of the 80386. Two techniques for accommodating slow devices are wait states and address pipelining. The designer must consider how to use one or both of these techniques to minimize the impact of device performance on system performance.

The impact of memory device speed on performance is generally much greater than that of I/O device speed because most programs require more memory accesses than I/O accesses. Therefore, the following discussion focuses on memory performance.

Wait states are extra CLK cycles added to the 80386 bus cycle. External logic generates wait states by delaying the READY# input to the 80386. For an 80386 operating at 16 MHz, one wait state adds 62.5 nanoseconds to the time available for the memory to respond. Each wait state increases the bus cycle time by 50 percent of the zero wait-state cycle time; however, overall system performance does not vary in direct proportion to the bus cycle increase. The second column of Table 4-1 shows the performance impact (based on an example simulation) for memory accesses requiring different numbers of wait states; one wait state results in an overall performance decrease of 19 percent.

Table 4-1. 80386 Performance with Wait States and Pipelining

Wait States When Address is Pipelined	Wait States When Address is Not Pipelined	Performance Relative to Non-Pipelined 0 Wait-State	Bus Utilization
0	0	1.00	73%
0	1	0.91	79%
1	1	0.81	86%
1	2	0.76	89%
2	2	0.66	91%
2	3	0.63	92%
3	3	0.57	93%

Unlike a wait state, address pipelining increases the time that a memory has to respond by one CLK cycle without lengthening the bus cycle. This extra CLK cycle eliminates the output delay of the 80386 address and status outputs. Address pipelining overlaps the address and status outputs of the next bus cycle with the end of the current bus cycle, lengthening the address access time by one or more CLK cycles from the point of view of the accessed memory device. An access that requires two wait states without address pipelining would require one wait state with address pipelining. The third column of Table 4-1 shows performance with pipelining for different wait-state requirements.

Address pipelining is advantageous for most bus cycles, but if the next address is not available before the current cycle ends,the 80386 cannot pipeline the next address, and the bus timing is identical to a non-pipelined bus cycle. Also, the first bus cycle after an idle bus must always be non-pipelined because there is no previous cycle in which to output the address early. If the next cycle is to be pipelined, the first cycle must be lengthened by at least one wait state so that the address can be output before the end of the cycle.

With the 80386, address pipelining is optional so that bus cycle timing can be closely tailored to the access time of the memory device; pipelining can be activated once the address is latched externally or not activated if the address is not latched.

The 80386 NA# input controls address pipelining. When the system no longer requires the 80386 to drive the address of the current bus cycle (in most systems, when the address has been latched), the system can activate the 80386 NA# input. The 80386 outputs the address and status signals for the next bus cycle on the next CLK cycle.

The system must activate the NA# signal without knowing which device the next bus cycle will access. In an optimal 80386 system, address pipelining should be used even for fast memory that does not require pipelining, because if a fast memory access is followed by a pipelined cycle to slower memory, one wait state is saved. If a fast memory access is followed by another fast memory access, the extra time is not used, and no processor time is lost. Therefore, all devices in a system must be able to accept both pipelined and non-pipelined cycles.

Consider a system in which a non-pipelined memory access requires one wait state and a non-pipelined I/O access requires four wait states. The bus control logic reads chip select signals from the address decoder to determine whether one or four wait states are required for the bus cycle. The bus control logic also determines whether the address has been pipelined, because a pipelined cycle requires one less wait state. The system includes logic for generating a Bus Idle signal that indicates whether the bus cycle has ended. The bus control logic can therefore detect that the address has been pipelined if the Address Status (ADS#) signal goes active while the Bus Idle signal is inactive.

Address pipelining is less effective for I/O devices requiring several wait states. The larger the number of wait states required, the less significant the elimination of one wait state through pipelining becomes. This fact coupled with the relative infrequency of I/O accesses means that address pipelining for I/O devices usually makes little difference to system performance.

A third and less common approach to accommodating memory speed is reducing the 80386 operating frequency. Because a slower clock frequency increases the bus cycle time, fewer wait states may be required for particular memory devices. At the same time, however, system performance depends directly on the 80386 clock frequency; execution time increases in direct proportion to the increase in clock period (reduction in clock frequency). A 12.5-MHz 80386 requires almost 33 percent more time to execute a program than a 16-MHz 80386 operating with the same number of wait states.

The design and application determine whether frequency reduction makes sense. In some instances, a slight reduction in clock frequency reduces the wait-state requirement and increases system performance. Table 4-2 shows that a 12.5-MHz 80386 operating with zero wait states yields better performance than a 16-MHz 80386 operating with two wait states.

Table 4-2. Performance versus Wait States and Operating Frequency

Number of Wait States	16 MHz Without Pipelining	16 MHz With Pipelining	12.5 MHz Without Pipelining	12.5 MHz With Pipelining
0	1.00	0.91	0.78	0.71
1	0.81	0.76	0.64	0.59
2	0.66	0.63	0.52	0.49
3	0.57	—	0.45	—

Coprocessor Hardware Interface 5

CHAPTER 5
COPROCESSOR HARDWARE INTERFACE

A numeric coprocessor enhances the performance of an 80386 system by performing numeric instructions in parallel with the 80386. The 80386 automatically passes on these instructions to the coprocessor as it encounters them.

Intel offers two numeric coprocessors:

- The 80287 performs 16-bit data transfers. With the proper interface, the 80386 supports the 80287.
- The 80387 performs 32-bit data transfers and interfaces directly with the 80386. The 80387 supports the instruction set of both the 80287 and the 8087, offering additional enhancements that include full compatibility with the IEEE Floating-Point Standard, draft 10. The performance of a 16-MHz 80387 is about eight times faster than that of a 5-MHz 80287.

Either an 80287 or an 80387 numeric coprocessor can be included in an 80386 system. The 80386 samples its ERROR# input during initialization to determine which coprocessor is present. As mentioned above, the 80287 and 80387 require different interfaces and therefore slightly different protocols. The 80287 data bus is 16 bits wide, whereas the 80387 data bus is 32 bits wide.

Data transfers to and from a coprocessor are accomplished through I/O addresses 800000F8H and 800000FCH; these addresses are automatically generated by the 80386 for coprocessor instructions and allow simple chip-select generation using A31 (high) and M/IO# (low). Because A31 is high for coprocessor cycles, the coprocessor addresses lie outside the range of the programmed I/O address space and are easy to distinguish from programmed I/O addresses. Coprocessor usage is independent of the I/O privilege level of the 80386.

The 80386 has three input signals for controlling data transfer to and from an 80287 or 80387 coprocessor: BUSY#, Coprocessor Request (PEREQ), and ERROR#. These signals, which are level-sensitive and may be asynchronous to the CLK2 input of the 80386, are described as follows:

- BUSY# indicates that the coprocessor is executing an instruction and therefore cannot accept a new one. When the 80386 encounters any coprocessor instruction except FNINIT and FNCLEX, the BUSY# input must be inactive (high) before the coprocessor accepts the instruction. A new instruction therefore cannot overrun the execution of the current coprocessor instruction. (Certain 80387 instructions can be transferred when BUSY# is active (low). These instructions are queued and do not interfere with the current instruction.)
- PEREQ indicates that the coprocessor needs to transfer data to or from memory. Because the coprocessor is never a bus master, all input and output data transfers are performed by the 80386. PEREQ always goes inactive before BUSY# goes inactive.

- ERROR# is asserted after a coprocessor math instruction results in an error that is not masked by the coprocessor's control register. The data sheets for the 80287 and 80387 describe these errors and explain how to mask them under program control. If an error occurs, ERROR# goes active before BUSY# goes inactive, so that the 80386 can take care of the error before performing another data transfer.

5.1 80287 NUMERIC COPROCESSOR INTERFACE

The 80287 is described in this section only as it relates to the 80386. For a complete functional description of the 80287, see the *80287 Data Sheet.*

5.1.1 80287 Connections

The connections between the 80386 and the 80287 are shown in Figure 5-1. These connections are made as follows:

- The 80287 BUSY#, ERROR#, and PEREQ outputs are connected to corresponding 80386 inputs.
- The 80287 RESET input is connected to the 82384 RESET output.
- The 80287 Numeric Processor Select chip-select inputs (NPS1# and NPS2) are connected to the latched M/IO# and A31 outputs, respectively. For coprocessor cycles, M/IO# is always low; A31, high.
- The 80287 Command inputs (CMD1 and CMD0) differentiate data from commands. These inputs are connected to ground and the latched A2 output, respectively. The 80386 outputs address 800000F8H when writing a command, address 800000FCH when writing or reading data.
- The lower half of the data bus connects to the 16 data bits of the 80287. The 80386 transfers data to and from the 80287 only over the D15-D0 lines.
- The 80287 Numeric Processor Read (NPRD#) and Numeric Processor Write (NPWR#) inputs are connected to I/O read and write signals from local bus control logic. The configuration of this logic depends on the overall system.
- The 80287 Processor Extension Acknowledge (PEACK#) input is pulled high. In an 80286 system, the 80286 generates PEACK# to disable the PEREQ output of the 80287 so that extra data is not transferred. Because the 80386 knows the length of the operand and will not transfer extra data, PEACK# is not needed or used in 80386 systems.

5.1.2 80287 Bus Cycles

When the 80386 encounters a coprocessor instruction, it automatically generates one or more I/O cycles to I/O addresses 800000F8H and 800000FCH. The 80386 performs all necessary bus cycles to memory and transfers data to and from the 80287 on the lower half of the

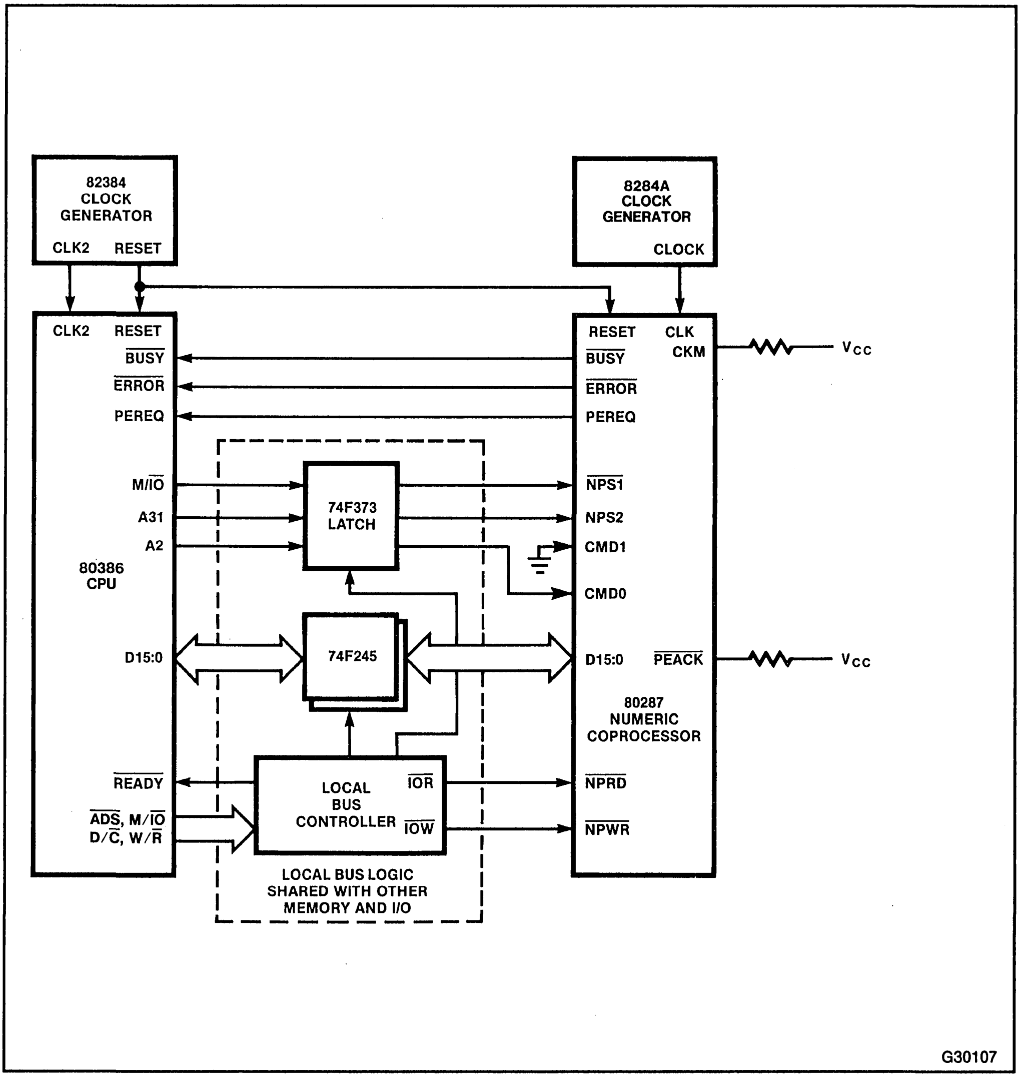

82384
CLOCK
GENERATOR
CLK2
RESET
8284A
CLOCK
GENERATOR
CLOCK
CLK2
RESET
BUSY
ERROR
PEREQ
M/IO
A31
A2
80386
CPU
D15:0
READY
ADS, M/IO
D/C, W/R
74F373
LATCH
74F245
LOCAL
BUS
CONTROLLER
IOR
IOW
LOCAL BUS LOGIC
SHARED WITH OTHER
MEMORY AND I/O
RESET
CLK
CKM
BUSY
ERROR
PEREQ
NPS1
NPS2
CMD1
CMD0
D15:0
PEACK
80287
NUMERIC
COPROCESSOR
NPRD
NPWR
Vcc
Vcc
G30107

Figure 5-1. 80386 System with 80287 Coprocessor

data bus. The 80386 automatically converts 32-bit memory transfers to 16-bit 80287 transfers and vice versa. Because the 80386 automatically performs 80287 transfers as 16 bits wide, the BS16# input of the 80386 does not need to be activated for transfers to and from the 80287.

Bus control logic must guarantee 80287 timing requirements, particularly the minimum command inactive time (T_{CMDI}). Depending on the design of the bus controller, command delays may be required.

5.1.3 80287 Clock Input

The 80287 can operate from the CLK or CLK2 output of the 82384 or a dedicated clock oscillator. To operate the 80287 from CLK or CLK2, the Clock Mode (CKM) pin of the 80287 must be tied to ground. In this configuration, the 80287 divides the system clock frequency internally by three.

The CKM pin of the 80287 is tied high to operate the 80287 from a dedicated MOS-level clock. In this configuration, the 80287 does not internally divide the clock frequency; it operates directly from the external clock. An 8284A clock driver and an appropriate crystal can be used to provide the 80287 with the desired clock frequency.

5.2 80387 NUMERIC COPROCESSOR INTERFACE

The 80387 achieves significant enhancements in performance and instruction capabilities over the 80287. It runs at internal clock rates of 16 or 20 MHz. To achieve maximum speed, the interface with the 80386 is synchronous and includes a full 32-bit data bus. Detailed information on other 80387 enhancements can be found in the *80387 Data Sheet.*

The 80387 is designed to run either fully synchronously or pseudosynchronously with the 80386. In the pseudosynchronous mode, the interface logic of the 80387 runs with the clock signal of the 80386, whereas internal logic runs with a different clock signal.

5.2.1 80387 Connections

The connections between the 80386 and the 80387 are shown in Figure 5-2 and are described as follows:

- The 80387 BUSY#, ERROR#, and PEREQ outputs are connected to corresponding 80386 inputs.
- The 80387 RESETIN input is connected to the 82384 RESET output.
- The 80387 Numeric Processor Select chip-select inputs (NPS1# and NPS2) are connected directly to the 80386 M/IO# and A31 outputs, respectively. For coprocessor cycles, M/IO# is always low; A31, high.
- The 80387 Command (CMD0#) input differentiates data from commands. This input is connected directly to the 80386 A2 output. The 80386 outputs address 800000F8H when writing a command or reading status, address 800000FCH when writing or reading data.
- All 32 bits (D31-D0) of the 80386 data bus connect directly to the data bus of the 80387. Because the data lines are connected directly, any local data bus transceivers must be disabled when the 80386 reads data from the 80387.
- The 80387 READY#, ADS#, and W/R# inputs are connected to the corresponding pins on the 80386. READY# and ADS# are used by the 80387 to track bus activity and determine when W/R#, NPS1#, NPS2, and Status Enable (STEN) can be sampled.

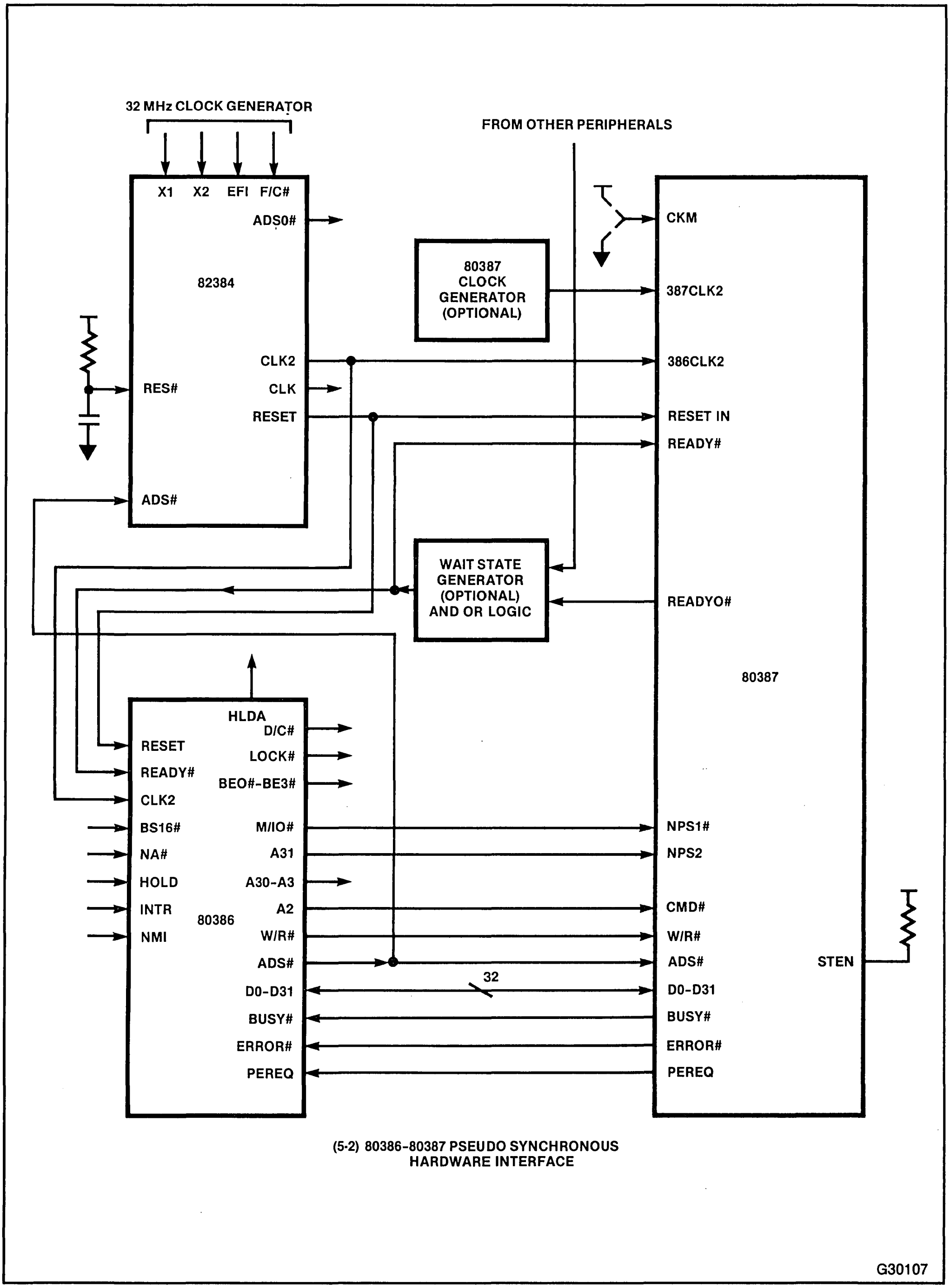

32 MHz CLOCK GENERATOR
X1
X2
EFI
F/C#
ADS0#
82384
RES#
CLK2
CLK
RESET
ADS#
FROM OTHER PERIPHERALS
80387 CLOCK GENERATOR (OPTIONAL)
CKM
387CLK2
386CLK2
RESET IN
READY#
WAIT STATE GENERATOR (OPTIONAL) AND OR LOGIC
READYO#
80387
HLDA
D/C#
RESET
LOCK#
READY#
BE0#-BE3#
CLK2
BS16#
M/IO#
NA#
A31
HOLD
A30-A3
INTR
A2
80386
NMI
W/R#
ADS#
D0-D31
BUSY#
ERROR#
PEREQ
32
NPS1#
NPS2
CMD#
W/R#
ADS#
STEN
D0-D31
BUSY#
ERROR#
PEREQ
(5-2) 80386-80387 PSEUDO SYNCHRONOUS HARDWARE INTERFACE
G30107

Figure 5-2. 80386 System with 80387 Coprocessor

- Status Enable (STEN) serves as a chip select for the 80387. This pin is high to enable the 80387, and may be driven low to float all 80387 outputs. STEN may be used to do onboard testing (using the overdrive method). STEN may also be used to activate one 80387 at a time, in systems with multiple 80387s. If not needed, STEN should be pulled high.

- Ready Out (READYO#) can be used to acknowledge 80387 bus cycles. The 80387 activates READYO# at such a time that write cycles are terminated after two clocks and read cycles are terminated after three clocks. READYO# can be connected to the 80386 READY# input through logic that ORs READY# signals from other devices. Alternatively, READYO# can be left disconnected, and external logic can be used to acknowledge 80387 bus cycles.

5.2.2 80387 Bus Cycles

When the 80386 encounters a coprocessor instruction, it automatically generates one or more I/O cycles to addresses 800000F8H and 800000FCH. The 80386 will perform all necessary bus cycles to memory and transfer data to and from the 80387. All 80387 transfers are 32 bits wide. If the memory subsystem is only 16 bits wide, the 80386 automatically performs the necessary conversion before transferring data to or from the 80387. Since the 80387 is a 32-bit device, BS16# must not be asserted during 80387 communication cycles.

Read cycles (transfers from the 80387 to the 80386) require at least one wait state, whereas write cycles to the 80387 require no wait states. This requirement is automatically reflected in the state of the READYO# output of the 80387, which can be used to generate the necessary wait state.

5.2.3 80387 Clock Input

The 80387 can be operated in two modes. In either mode, the CLK2 signal must be connected to the 386CLK2 input of the 80387 because the interface to the 80386 is always synchronous. The state of the 80387 CKM input determines its mode:

- In synchronous mode, CKM is high and the 387CLK2 input is not connected. The 80387 operates from the CLK2 signal. Operation of the 80387 is fully synchronous with that of the 80386.

- In pseudo-synchronous mode, CKM is low and a frequency source for the 387CLK2 input must be provided. Only the interface logic of the 80387 is synchronous with the 80386. The internal logic of the 80387 operates from the 387CLK2 clock source, whose frequency may be 10/16 to 16/10 times the speed of CLK2. Figure 5-3 depicts pseudo-synchronous operation.

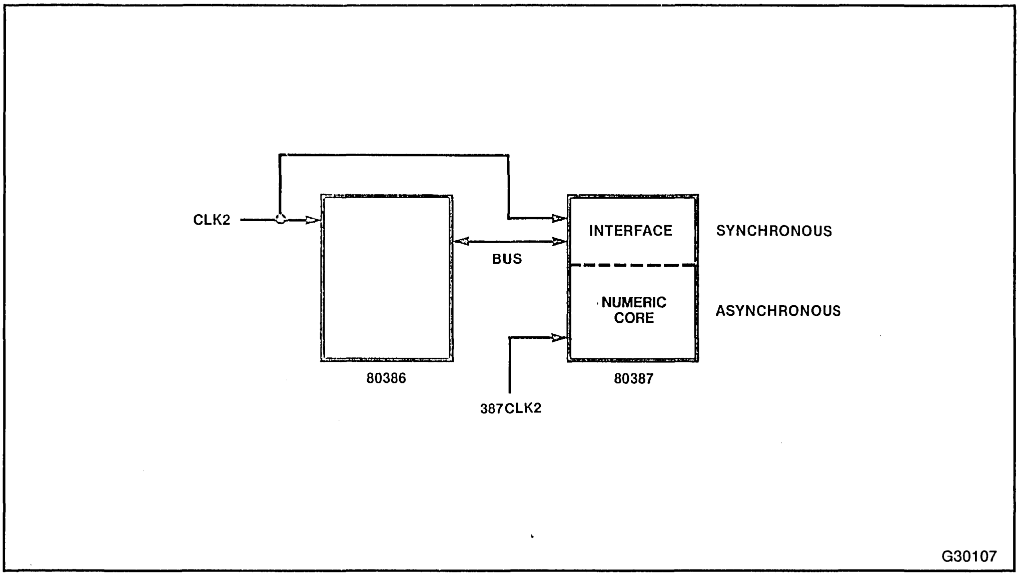


Figure 5-3. Pseudo-Synchronous Interface

5.3 LOCAL BUS ACTIVITY WITH THE 80287/80387

Both 80287 and 80387 coprocessors use two distinct methods to interact with the 80386:

- The 80386 initiates coprocessor operations during the execution of a coprocessor instruction (an ESC instruction). These interactions occur under program control.
- The coprocessor uses the PEREQ signal to request the 80386 to initiate operand transfers to or from system memory. These operand transfers occur when the 80287/80387 requests them; thus, they are asynchronous to the instruction execution of the 80386.

When the 80386 executes an ESC instruction that requires transfers of operands to or from the coprocessor, the 80386 automatically sets an internal memory address base register, memory address limit register, and direction flag. The coprocessor can then request operand transfers by driving PEREQ active. These requests occur only when the coprocessor is executing an instruction (when BUSY# is active).

Two, three, four or five bus cycles may be necessary for each operand transfer. These cycles include one coprocessor cycle plus one of the following:

- One memory cycle for an aligned operand
- Two memory cycles for a misaligned operand
- Two or three memory cycles for misaligned 32-bit operands to 16-bit memory
- Four memory cycles for misaligned 64-bit operands to 16-bit memory

Data transfers for the coprocessor have the same bus priority as programmed data transfers.

5.4 DESIGNING AN UPGRADABLE 80386 SYSTEM

It is relatively simple to design an 80386 system with an 80287 that may be later upgraded to an 80387. The advantage of such a system is that two performance levels may be addressed by one system at minimal additional cost.

5.4.1 80287/80387 Recognition

5.4.1.1 HARDWARE RECOGNITION OF THE NPX

The 80386 identifies the type of its coprocessor (80287 or 80387) by sampling its ERROR# input some time after the falling edge of RESET and before executing the first ESC instruction. The 80287 keeps its ERROR# output in inactive state after hardware reset; the 80387 keeps its ERROR# output in active state after hardware reset. The 80386 records this difference in the ET bit of control register zero (CR0). The 80386 subsequently uses ET to control its interface with the coprocessor. If ET is set, it employs the 32-bit protocol of the 80387; if ET is not set, it employs the 16-bit protocol of the 80287.

Systems software can (if necessary) change the value of ET. There are three reasons that ET may not be set:

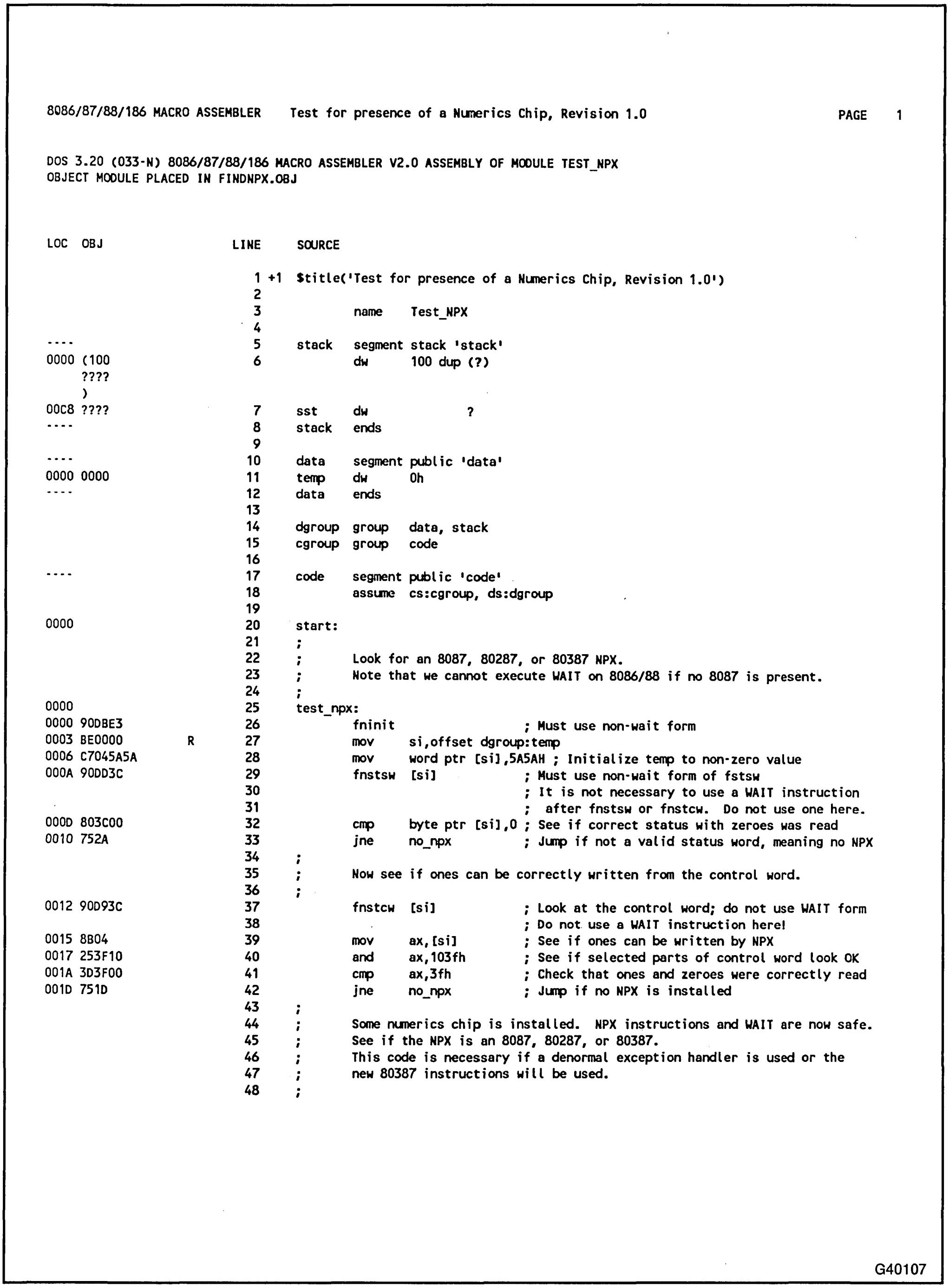
1. An 80287 is actually present.
2. No coprocessor is present.
3. An 80387 is present but it is connected in a nonstandard manner that does not trigger the setting of ET.

An example of case three is a PC/AT-compatible design. In such cases, initialization software may need to change the value of ET.

5.4.1.2 SOFTWARE RECOGNITION OF THE NPX

Figure 5-4 shows an example of a recognition routine that determines whether an NPX is present, and distinguishes between the 80387 and the 8087/80287. This routine can be executed on any 80386, 80286, or 8086 hardware configuration that has an NPX socket.

The example guards against the possibility of accidentally reading an expected value from a floating data bus when no NPX is present. Data read from a floating bus is undefined. By expecting to read a specific bit pattern from the NPX, the routine protects itself from the indeterminate state of the bus. The example also avoids depending on any values in reserved bits, thereby maintaining compatibility with future numerics coprocessors.



```
8086/87/88/186 MACRO ASSEMBLER    Test for presence of a Numerics Chip, Revision 1.0                 PAGE    1


DOS 3.20 (033-N) 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE TEST_NPX
OBJECT MODULE PLACED IN FINDNPX.OBJ


LOC  OBJ                  LINE     SOURCE

                             1 +1  $title('Test for presence of a Numerics Chip, Revision 1.0')
                             2
                             3             name    Test_NPX
                             4
----                         5     stack   segment stack 'stack'
0000 (100                    6             dw      100 dup (?)
     ????
     )
00C8 ????                    7     sst     dw              ?
----                         8     stack   ends
                             9
----                        10     data    segment public 'data'
0000 0000                   11     temp    dw      0h
----                        12     data    ends
                            13
                            14     dgroup  group   data, stack
                            15     cgroup  group   code
                            16
----                        17     code    segment public 'code'
                            18             assume  cs:cgroup, ds:dgroup
                            19
0000                        20     start:
                            21     ;
                            22     ;       Look for an 8087, 80287, or 80387 NPX.
                            23     ;       Note that we cannot execute WAIT on 8086/88 if no 8087 is present.
                            24     ;
0000                        25     test_npx:
0000 90DBE3                 26             fninit                  ; Must use non-wait form
0003 BE0000          R      27             mov     si,offset dgroup:temp
0006 C7045A5A               28             mov     word ptr [si],5A5AH ; Initialize temp to non-zero value
000A 90DD3C                 29             fnstsw  [si]            ; Must use non-wait form of fstsw
                            30                                     ; It is not necessary to use a WAIT instruction
                            31                                     ;  after fnstsw or fnstcw.  Do not use one here.
000D 803C00                 32             cmp     byte ptr [si],0 ; See if correct status with zeroes was read
0010 752A                   33             jne     no_npx          ; Jump if not a valid status word, meaning no NPX
                            34     ;
                            35     ;       Now see if ones can be correctly written from the control word.
                            36     ;
0012 90D93C                 37             fnstcw  [si]            ; Look at the control word; do not use WAIT form
                            38                                     ; Do not use a WAIT instruction here!
0015 8B04                   39             mov     ax,[si]         ; See if ones can be written by NPX
0017 253F10                 40             and     ax,103fh        ; See if selected parts of control word look OK
001A 3D3F00                 41             cmp     ax,3fh          ; Check that ones and zeroes were correctly read
001D 751D                   42             jne     no_npx          ; Jump if no NPX is installed
                            43     ;
                            44     ;       Some numerics chip is installed.  NPX instructions and WAIT are now safe.
                            45     ;       See if the NPX is an 8087, 80287, or 80387.
                            46     ;       This code is necessary if a denormal exception handler is used or the
                            47     ;       new 80387 instructions will be used.
                            48     ;
```

G40107

Figure 5-4. Software Routine to Recognize the 80287

```
8086/87/88/186 MACRO ASSEMBLER    Test for presence of a Numerics Chip, Revision 1.0                PAGE    2


LOC  OBJ                 LINE     SOURCE

001F 9BD9E8                49             fld1                     ; Must use default control word from FNINIT
0022 9BD9EE                50             fldz                     ; Form infinity
0025 9BDEF9                51             fdiv                     ; 8087/287 says +inf = -inf
0028 9BD9C0                52             fld     st               ; Form negative infinity
002B 9BD9E0                53             fchs                     ; 80387 says +inf <> -inf
002E 9BDED9                54             fcompp                   ; See if they are the same and remove them
0031 9BDD3C                55             fstsw   [si]             ; Look at status from FCOMPP
0034 8B04                  56             mov     ax,[si]
0036 9E                    57             sahf                     ; See if the infinities matched
0037 7406                  58             je      found_87_287     ; Jump if 8087/287 is present
                           59     ;
                           60     ;       An 80387 is present.  If denormal exceptions are used for an 8087/287,
                           61     ;       they must be masked.  The 80387 will automatically normalize denormal
                           62     ;       operands faster than an exception handler can.
                           63     ;
0039 EB0790                64             jmp     found_387
003C                       65     no_npx:
                           66     ;       set up for no NPX
                           67     ;           ...
                           68     ;
003C EB0490                69             jmp exit
003F                       70     found_87_287:
                           71     ;       set up for 87/287
                           72     ;           ...
                           73     ;
003F EB0190                74             jmp exit
0042                       75     found_387:
                           76     ;       set up for 387
                           77     ;           ...
                           78     ;
0042                       79     exit:
----                       80     code    ends
                           81             end     start,ds:dgroup,ss:dgroup:sst

ASSEMBLY COMPLETE, NO ERRORS FOUND
                                                                                                     G40107
```

Figure 5-4. Software Routine to Recognize the 80287 (Cont'd.)

5.4.2 80387 Emulator

The 80387 emulator circuit makes a 10-MHz 80287 appear to the 80386 as an 80387. The schematic is shown in Figure 5-5. An 80387 socket (80387 PGA Adapter) is connected to the 80386 (connections not shown) as described earlier in this chapter. Please note that this circuit is provided to illustrate concepts only; it has not been tested.

The 80287 sends signals to the 80386 through the socket. The inputs coming to the socket from the 80386 are decoded by the PAL16L8 (Math Control) to provide the buffered CMD0, NPRD#, and NPWR# inputs for the 80287. The PAL equations for the Math Control PAL are listed in Appendix B of this manual.

The data lines of the 80287 are connected to the lower half of the 80386 data bus through the socket; the BUSY#, ERROR#, and PEREQ signals are returned to the 80386 through the socket as well. Note that while this circuit allows the 80386 to use an 80387 interface with the 80287, the 80287 still performs only 16-bit data transfers. Therefore, during initialization, the ERROR# output of the 80287 is high to indicate the presence of an 80287, not an 80387.

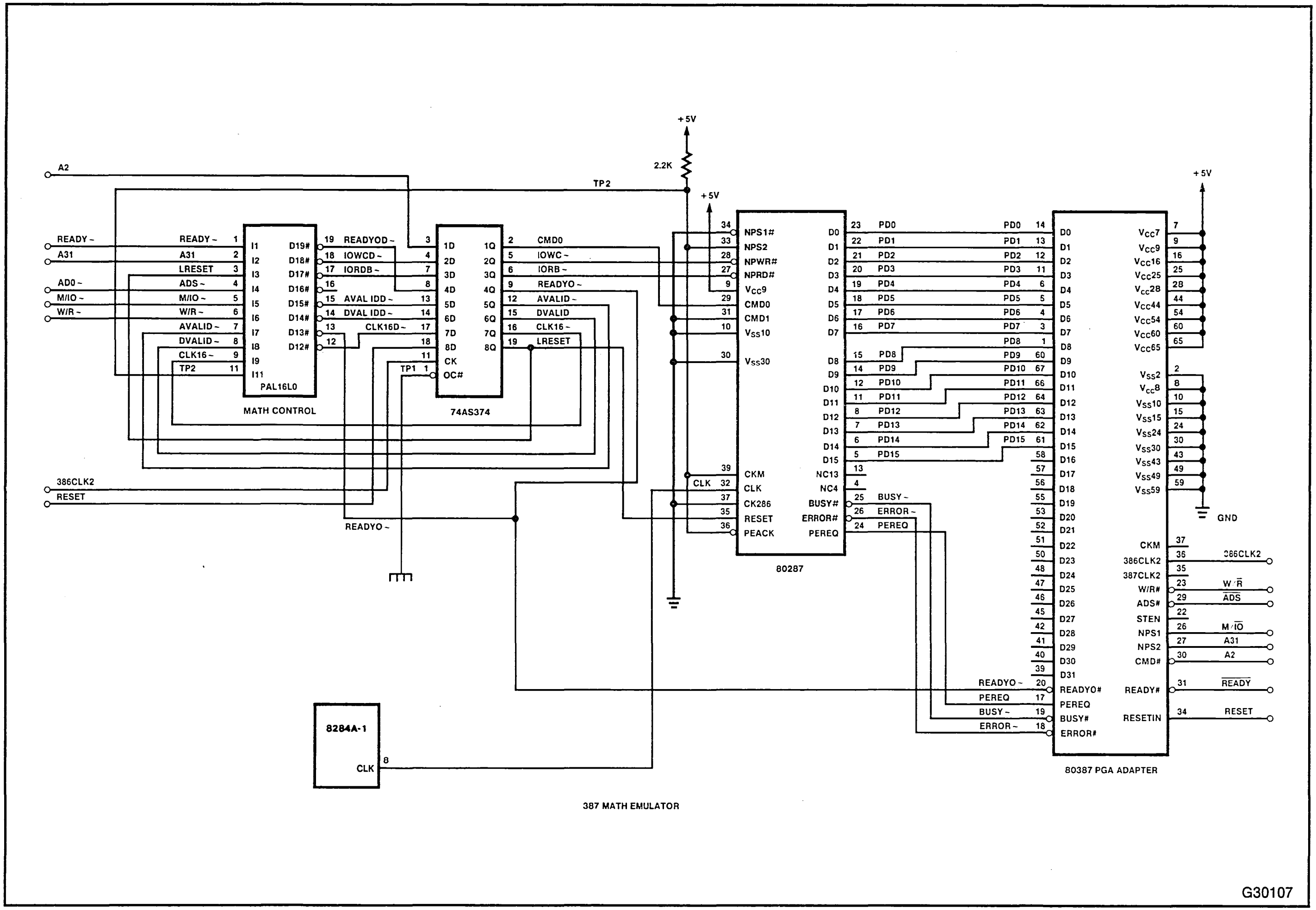

+5V
2.2K
TP2
+5V
A2
READY~
A31
AD0~
M/IO~
W/R~
386CLK2
RESET
READY~ 1
A31 2
LRESET 3
ADS~ 4
M/IO~ 5
W/R~ 6
AVALID~ 7
DVALID~ 8
CLK16~ 9
TP2 11
I1 I2 I3 I4 I5 I6 I7 I8 I9 I11
D19# D18# D17# D16# D15# D14# D13# D12#
PAL16L0
MATH CONTROL
19 READYOD~
18 IOWCD~
17 IORDB~
16
15 AVAL IDD~
14 DVAL IDD~
13 CLK16D~
12
1D 2D 3D 4D 5D 6D 7D 8D
CK
OC#
TP1 1
1Q 2Q 3Q 4Q 5Q 6Q 7Q 8Q
74AS374
2 CMD0
5 IOWC~
6 IORB~
9 READYO~
12 AVALID~
15 DVALID
16 CLK16~
19 LRESET
READYO~
NPS1#
NPS2
NPWR#
NPRD#
VCC9
CMD0
CMD1
VSS10
VSS30
CKM
CLK
CK286
RESET
PEACK
D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 D10 D11 D12 D13 D14 D15
NC13
NC4
BUSY#
ERROR#
PEREQ
80287
PD0 PD1 PD2 PD3 PD4 PD5 PD6 PD7 PD8 PD9 PD10 PD11 PD12 PD13 PD14 PD15
BUSY~
ERROR~
PEREQ
8284A-1
CLK 8
387 MATH EMULATOR
D0 … D31
READYO#
PEREQ
BUSY#
ERROR#
VCC7 VCC9 VCC16 VCC25 VCC28 VCC44 VCC54 VCC60 VCC65
VSS2 VCC8 VSS10 VSS15 VSS24 VSS30 VSS43 VSS49 VSS59
GND
CKM
386CLK2
387CLK2
W/R#
ADS#
STEN
NPS1
NPS2
CMD#
READY#
RESETIN
386CLK2
W/R
ADS
M/IO
A31
A2
READY
RESET
80387 PGA ADAPTER
G30107

Figure 5-5. 80387 Emulator Schematic

Memory Interfacing

6

CHAPTER 6
MEMORY INTERFACING

The 80386 high-speed bus interface has many features that contribute to high-performance memory interfaces. This chapter outlines approaches to designing memory systems that utilize these features, describes memory design considerations, and lists a number of useful examples. The concepts illustrated by these examples apply to a wide variety of memory system implementations.

6.1 MEMORY SPEED VERSUS PERFORMANCE AND COST

In a high-performance microprocessing system, overall system performance is linked to the performance of memory subsystems. Most bus cycles in a typical microprocessing system are used to access memory because memory is used to store programs as well as the data used in processing.

To realize the performance potential of the 80386, a system must use relatively fast memory. A high-performance processor coupled with low-performance memory provides no better throughput than a less expensive low-performance processor. Fast memory devices, however, cost more than slow memory devices.

The cost-performance tradeoff can be mediated by partitioning functions and using a combination of both fast and slow memories. If the most frequently used functions are placed in fast memory and all other functions are placed in slow memory, high performance for most operations can be achieved at a cost significantly less than that of a fast memory subsystem. For example, in a RAM-based system that uses read-only memory devices primarily during initialization, the PROM or EPROM can be slow (requiring three to four wait states) and yet have little effect on system performance. RAM memory can also be partitioned into fast local memory and slower system memory. Other performance considerations are described in detail in Chapter 4.

The relationship between memory subsystem performance and the speed of individual memory devices is determined by the design of the memory subsystem. Cache systems, which couple a small cache memory with a larger main memory, are described in Chapter 7. Basic memory interfaces are described in this chapter.

6.2 BASIC MEMORY INTERFACE

The high performance and flexibility of the 80386 local bus interface plus the availability of programmable and semi-custom logic (programmable logic arrays, for example) make it practical to design custom bus control logic that meets the requirements of a particular system. Standard logic components can generate the bus control signals needed to interface the 80386 with memory and I/O devices. The basic memory interface is discussed in this chapter; the basic I/O interface is presented in Chapter 8.

The block diagram of the basic memory interface is shown in Figure 6-1. The bus control logic provides the control signals for the address latches, data buffers, and memory devices; it also returns READY# active to end the 80386 bus cycle and NA# to control address pipelining. The address decoder generates chip-select signals and the BS16# signal based on the address outputs of the 80386. This interface is suitable for accessing ROMs, EPROMs, and static RAMs (SRAMs).

6.2.1 PAL Devices

Many design examples in this manual use PAL (Programmable Array Logic) devices, which can be programmed by the user to implement random logic. A PAL device can be used as a state machine or a signal decoder, for example. The advantages of PALs include the following:

- Programmability allows the PAL functions to be changed easily, simplifying prototype development.
- The designer determines PAL pinout and can simplify the board layout by moving signals as required.

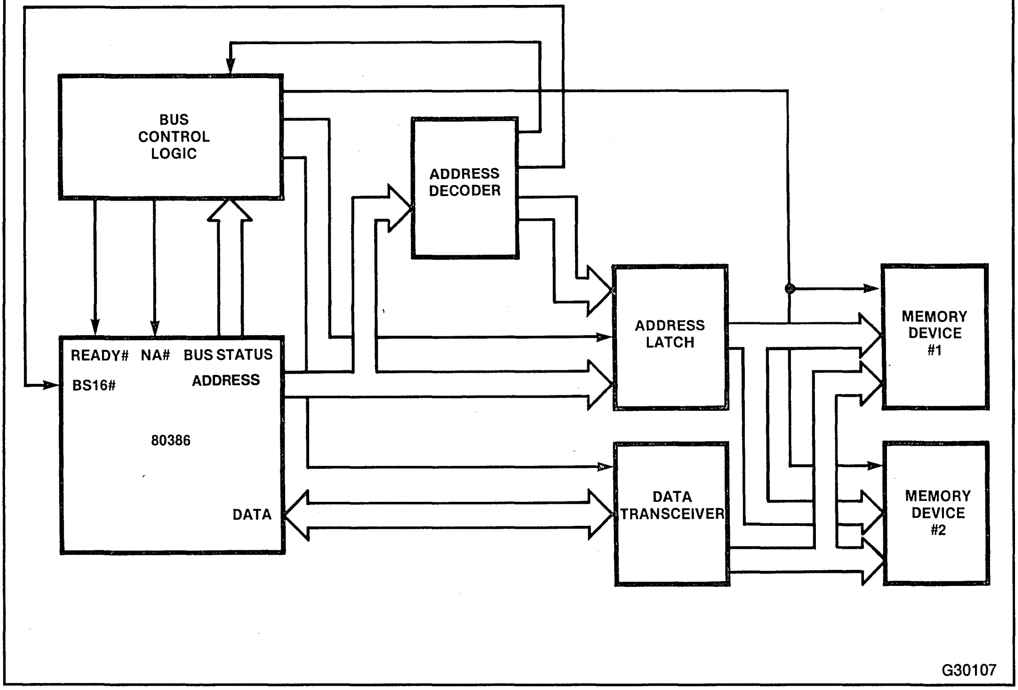

BUS CONTROL LOGIC
ADDRESS DECODER
ADDRESS LATCH
MEMORY DEVICE #1
READY# NA# BUS STATUS
BS16#
ADDRESS
80386
DATA
DATA TRANSCEIVER
MEMORY DEVICE #2
G30107

Figure 6-1. Basic Memory Interface Block Diagram

- PALs are inexpensive compared to dedicated bus controllers.
- Once a PAL design has been tested, Hard Array Logic (HAL) devices, which are mask-programmed PALs, can be used in production quantities (several thousand units).

PALs also have the following disadvantages:

- Pin counts or speeds of available PALs can restrict some designs.
- Most PALs do not have buried (not connected to outputs) state registers; therefore, in state-machine implementations, registered output pins must be used to store the current state.
- The drive capability of PALs may be insufficient for some applications. In these cases, buffering is required.

A PAL device consists logically of a programmable AND array whose output terms feed a fixed OR array. Any sum-of-products equation, within the limits of the number of PAL inputs, outputs, and equation terms, can be realized by specifying the correct AND array connections. Figure 6-2 shows an example of two PAL equations and the corresponding logic array. Note that every horizontal line in the AND array represents a multi-input AND gate; every vertical line represents a possible input to the AND gate. The X at the intersection of a horizontal line and a vertical line represents a connection from the input to the AND gate.

Programming a PAL device consists of determining where the Xs must be placed in the AND array. This task is simplified by the use of a PAL assembler program. Such a program accepts input in the form of sum-of-products equations. The assembled code is then applied to a PAL device using a standard PROM programmer equipped with a special PAL enhancement.

The following conventions apply to TTL and PAL devices described in this section:

- TTL devices are specified by number (function), but not by family (speed). Virtually any family of a device can be used if it meets the performance requirements of the application. For example, a 74x00 device might be implemented with a 74F00 or 74AS00. Generally, the F and AS families provide the highest performance.
- PAL devices are specified by part number. A PAL part number generally consists of a number, a letter, and a second number, and is interpreted as shown in Figure 6-3. PALs are manufactured for a number of performance levels. Standard PALs are suitable unless otherwise specified.

6.2.2 Address Latch

Latches maintain the address for the duration of the bus cycle and are necessary to pipeline addresses because the address for the next bus cycle appears on the address lines before the current cycle ends. In this example, 74x373 latches are used. Although the 80386 can be run without address pipelining to eliminate the need for address latching, the system will usually run less efficiently.

- **Sample equations:**

/Q = A · /B
+ A · /R
+ /A · B · R

/R := A · /R
+ /B

- **Sample block diagram:**

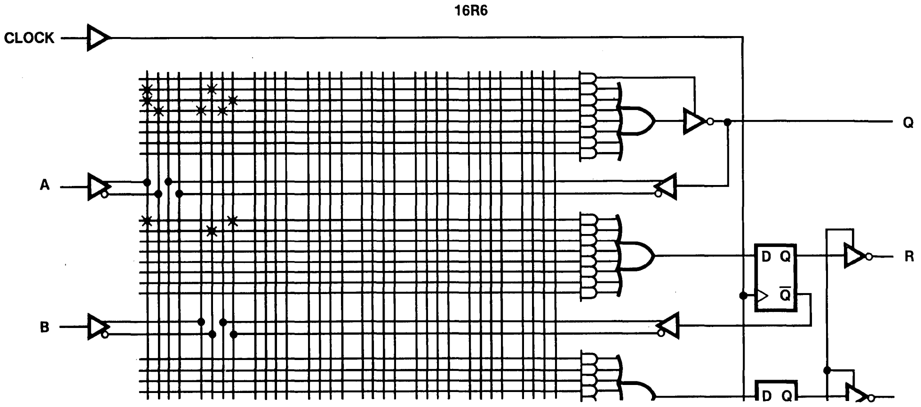
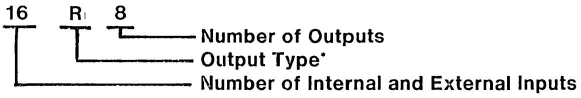


Figure 6-2. PAL Equation and Implementation

The 74x373 Latch Enable (LE) input is controlled by the Address Latch Enable (ALE or ALE#) signal from the bus control logic that goes active at the start of each bus cycle. The 74x373 Output Enable (OE#) is always active.

6.2.3 Address Decoder

In this example, the address decoder, which converts the 80386 address into chip-select signals, is located before the address latches. In general, the decoder may also be placed after the latches. If it is placed before the latches, the chip-select signal becomes valid as early as possible but must be latched along with the address. Therefore, the number of address latches needed is determined by the location of the address decoder as well as the number of address bits and chip-select signals required by the interface. The chip-select signals are routed to the bus control logic to set the correct number of wait states for the accessed device.



*Output types are designated as follows:

H	Active High
L	Active Low
C	Complementary
R	Registered
X	Exclusive-OR Registered
A	Arithmetic Registered

Figure 6-3. PAL Naming Conventions

The decoder consists of two one-of-four decoders, one for memory address decoding and one for I/O address decoding. In general, the number of decoders needed depends on the memory mapping complexity. In this basic example, the A31 output is sufficient to determine which memory device is to be selected.

6.2.4 Data Transceiver

Standard 8-bit transceivers (74x245, in this example) provide isolation and additional drive capability for the 80386 data bus. Transceivers are necessary to prevent the contention on the data bus that occurs if some devices are slow to remove read data from the data bus after a read cycle. If a write cycle follows a read cycle, the 80386 may drive the data bus before a slow device has removed its outputs from the bus, potentially causing reliability problems. Transceivers can be omitted only if the data float time of the device is short enough and the load on the 80386 data pins meets device specifications.

A bus interface must include enough transceivers to accommodate the device with the most inputs and outputs on the data bus. Normally, 32-bit-wide memories, which require four 8-bit transceivers, are used in 80386 systems.

The 74x245 transceiver is controlled through two input signals:

- Data Transmit/Receive (DT/R#)—When high, this input enables the transceiver for a write cycle. When low, it enables the transceiver for a read cycle. This signal is just a latched version of the 80386 W/R# output.
- Data Enable (DEN#)—When low, this input enables the transceiver outputs. This signal is generated by the bus control logic.

6.2.5 Bus Control Logic

Bus control logic is shown in Figure 6-4. The bus controller is implemented in two PALs. One PAL (PAL-1) follows the 80386 bus cycles and generates the overall bus cycle timing. The second PAL (PAL-2) generates most of the bus control signals. The equations for these PALs are listed in Appendix A of this manual.

The bus controller decodes the 80386 status outputs (W/R#, M/IO#, and D/C#) and activates a command signal for the type of bus cycle requested. The command signal corresponds to the bus cycle types (described in Chapter 3) as follows:

- Memory data read and memory code read cycles generate the Memory Read Command (MRDC#) output. MRDC# commands the selected memory device to output data.
- I/O read cycles generate the I/O Read Command (IORC#) output. IORC# commands the selected I/O device to output data.
- Memory write cycles generate the Memory Write Command (MWTC#) output. MWTC# commands the selected memory device to receive the data on the data bus.
- I/O write cycles generate the I/O Write Command (IOWC#) output. IOWC# commands the selected I/O device to receive the data on the data bus.
- Interrupt-acknowledge cycles generate the Interrupt Acknowledge (INTA#) output, which is returned to the 8259A Interrupt Controller. The second INTA cycle commands the 8259A to place the interrupt vector on the bus.

Figure 6-5 shows the timings of bus control signals for various memory accesses.

The bus controller also controls the READY# input to the 80386 that ends each bus cycle. The PAL-2 bus control PAL counts wait states and returns READY# after the number of wait states required by the accessed device. The design of this portion of the bus controller depends on the requirements of the system; relatively simple systems need less wait-state logic than more complex systems. The basic interface described here uses a PAL device to generate READY#; other designs may use counters and/or shift registers.

The bus controller generates one of two bus cycles depending on which memory is being accessed. Two inputs to PAL-1 (CS0WS and CS1WS) from the address decoder determine the bus control signal timings. Table 6-1 shows the number of wait states, the command delay time, and the data float time for each bus cycle.

Table 6-1. Bus Cycles Generated by Bus Controller

Chip-Select	Cycle Type	WAIT-STATES		Command Delay	Data-Float
		Pipelined	UnPipelined		
CS0WS	read write	0 1	1 2	1 CLK2 1 CLK2	2 CLK2 —
CS1WS	any	1	2	2 CLK2	4 CLK2

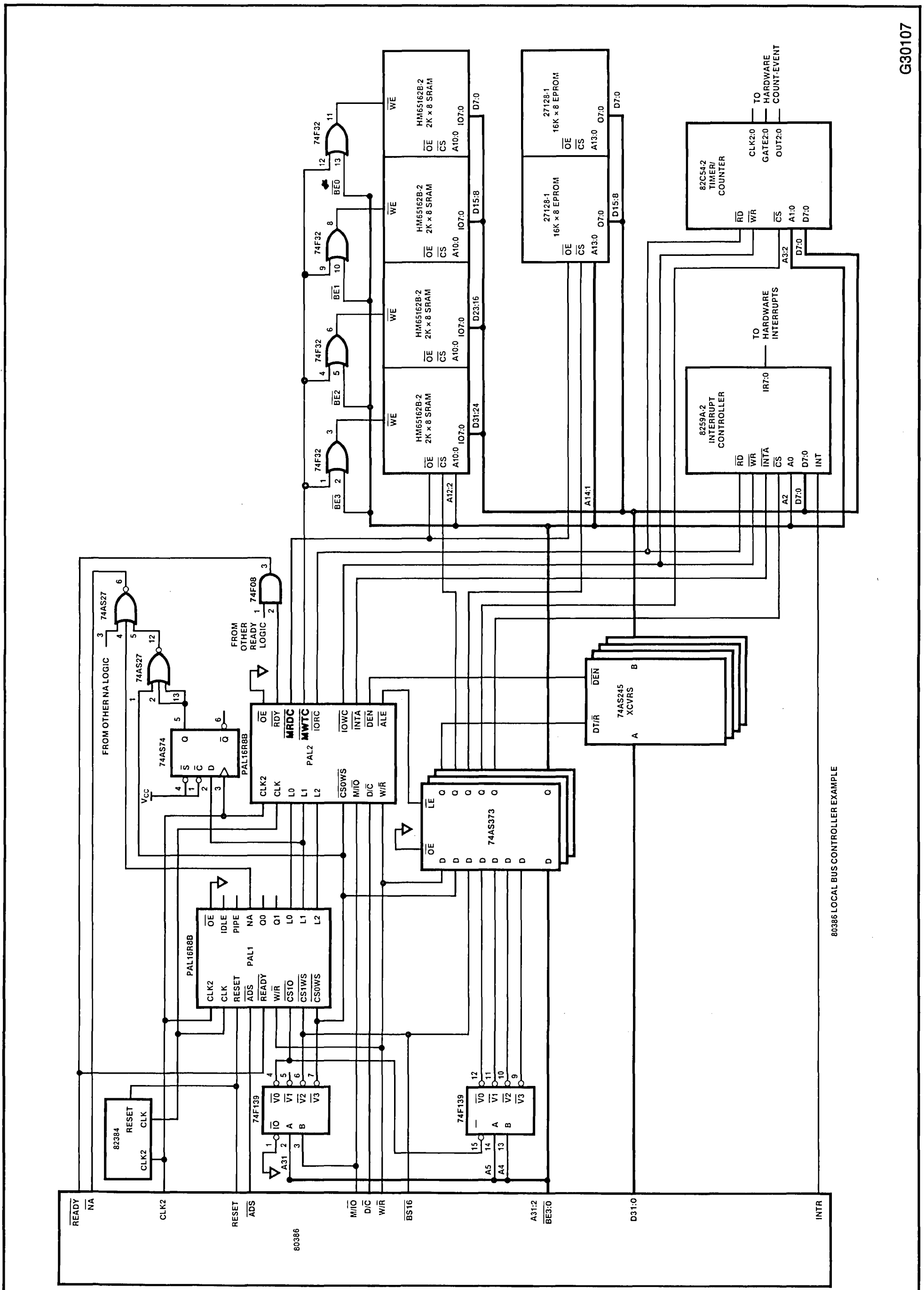

80386
READY
NA
CLK2
RESET
ADS
M/IO
D/C
W/R
BS16
A31:2
BE3:0
D31:0
INTR
82384
CLK2
RESET
CLK
74F139
PAL16R8B
PAL1
CLK2
CLK
RESET
ADS
READY
W/R
CS1O
CS1WS
CS0WS
OE
IDLE
PIPE
NA
Q0
Q1
L0
L1
L2
74AS74
74AS27
74F08
FROM OTHER NA LOGIC
FROM OTHER READY LOGIC
PAL2
RDY
MRDC
MWTC
IORC
IOWC
INTA
DEN
ALE
CS0WS
M/IO
D/C
W/R
74AS373
LE
74AS245 XCVRS
DT/R
74F32
BE0
BE1
BE2
BE3
HM65162B-2 2K × 8 SRAM
WE
OE
CS
A10:0
IO7:0
A12:2
D7:0
D15:8
D23:16
D31:24
27128-1 16K × 8 EPROM
A13:0
O7:0
A14:1
8259A-2 INTERRUPT CONTROLLER
RD
WR
INTA
CS
A0
D7:0
INT
IR7:0
TO HARDWARE INTERRUPTS
A2
82C54-2 TIMER/ COUNTER
CLK2:0
GATE2:0
OUT2:0
A1:0
A3:2
TO HARDWARE COUNT-EVENT
80386 LOCAL BUS CONTROLLER EXAMPLE
G30107

Figure 6-4. Bus Control Logic

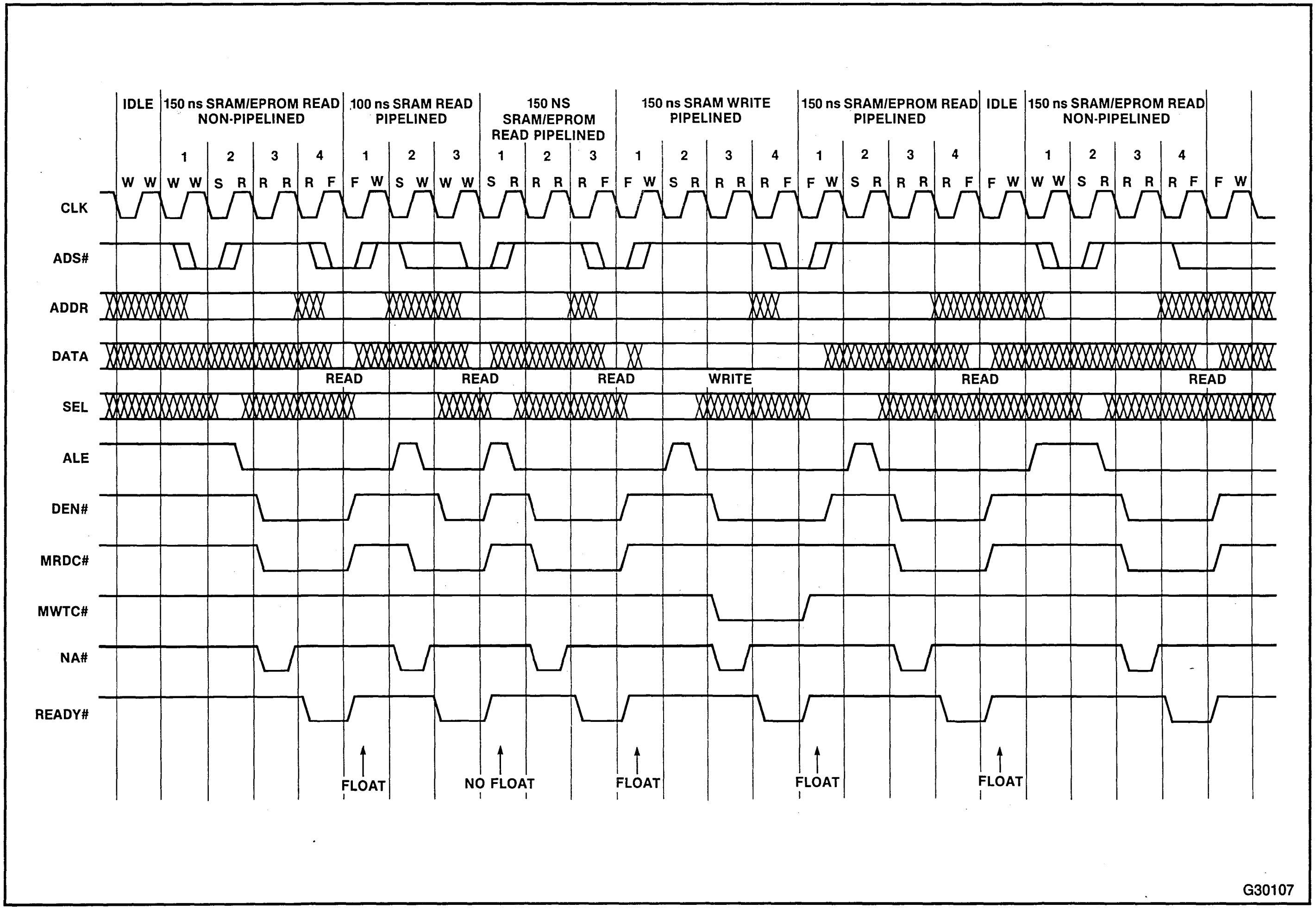

IDLE
150 ns SRAM/EPROM READ NON-PIPELINED
100 ns SRAM READ PIPELINED
150 NS SRAM/EPROM READ PIPELINED
150 ns SRAM WRITE PIPELINED
150 ns SRAM/EPROM READ PIPELINED
IDLE
150 ns SRAM/EPROM READ NON-PIPELINED
1 2 3 4 1 2 3 1 2 3 1 2 3 4 1 2 3 4 1 2 3 4
W W W W S R R R R F F W S W W W S R R R R F F W S R R R R F F W S R R R R F F W W W S R R R R F F W
CLK
ADS#
ADDR
DATA
SEL
ALE
DEN#
MRDC#
MWTC#
NA#
READY#
READ
READ
READ
WRITE
READ
READ
FLOAT
NO FLOAT
FLOAT
FLOAT
FLOAT
G30107

Figure 6-5. Bus Control Signal Timing

6.2.6 EPROM Interface

Figure 6-6 shows the signal timing for bus cycles from an 80386 operating at 16 MHz to a 27128-1 EPROM, which has a 150-nanosecond access time. Faster 110-nanosecond EPROMs are also available but in this design, they require the same number of wait states as the 150-nanosecond EPROMs require. Timings for a 150-nanosecond SRAM are included for comparison.

In the EPROM interface, the OE# input of each EPROM devices is connected directly to the MRDC# signal from the bus controller. The wait state requirement is calculated by adding up worst-case delays and comparing the total with the 80386 bus cycle time.

The bus cycle timings can be calculated from the waveforms in Figure 6-6. In the following example, the timings for I/O accesses are calculated for CLK2 = 32 MHz and B-series PALs. All times are in nanoseconds. Check the most recent *80386 Data Sheet* to confirm all parameter values.

tAR: Address stable before Read (MRDC# fall)

(1 x CLK2 period) − PAL RegOut Max − Latch Enable Max
− PAL RegOut Min
(1 x 31.25) − 12 − 11.5
+ 0

= 7.75 nanoseconds

tRR: Read (MRDC#) pulse width

(4 x CLK2 period) − PAL RegOut Max + PAL RegOut Min
(4 x 31.25) −12 + 0

= 113 nanoseconds

tRA: Address hold after Read (MRDC# rise)

(0 x CLK2 period) − PAL RegOut Max + PAL RegOut Min
+ Latch Enable Min
(0 x 31.25) − 12 + 0
+ 5

= −7 nanoseconds (This is acceptable because latched addresses are held for at least as long as the end of the bus cycle.)

tAD: Data delay from Address

(6 x CLK2 period) − PAL RegOut Max − Latch Enable Max
− xcvr. prop. Min − 80386 Data Setup Min
(6 x 31.25) − 12 − 11.5
− 6 − 10

= 148 nanoseconds

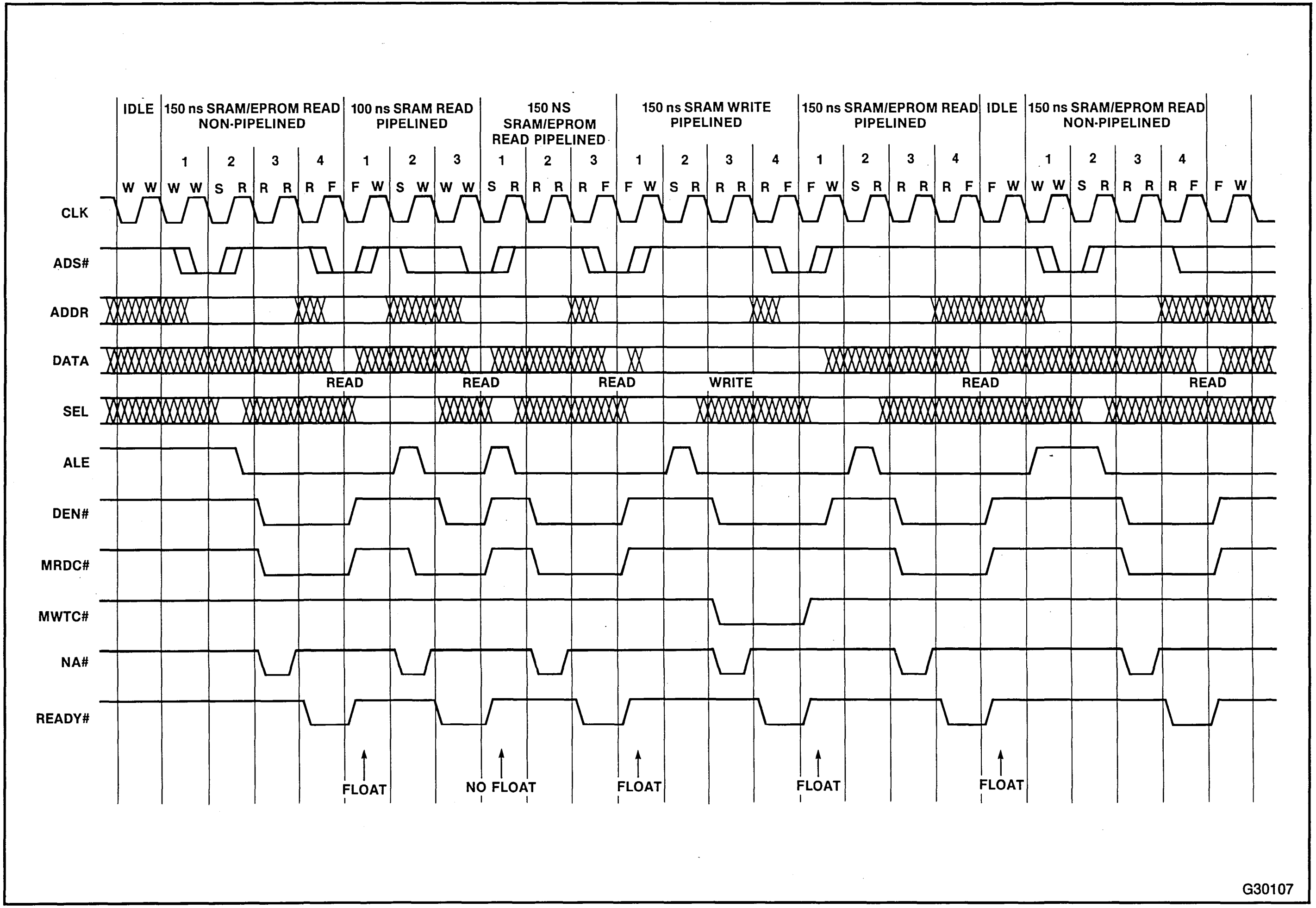

IDLE
150 ns SRAM/EPROM READ NON-PIPELINED
100 ns SRAM READ PIPELINED
150 NS SRAM/EPROM READ PIPELINED
150 ns SRAM WRITE PIPELINED
150 ns SRAM/EPROM READ PIPELINED
IDLE
150 ns SRAM/EPROM READ NON-PIPELINED
CLK
ADS#
ADDR
DATA
SEL
ALE
DEN#
MRDC#
MWTC#
NA#
READY#
READ
READ
READ
WRITE
READ
READ
FLOAT
NO FLOAT
FLOAT
FLOAT
FLOAT
G30107

Figure 6-6. 150-Nanosecond EPROM Timing Diagram tRD: Data delay from Read (MRDC#)

(4 x CLK2 period) − PAL RegOut Max − xcvr. prop Min
− 80386 Data Setup Min
(4 x 31.25) − 12 − 6
− 10

= 97 nanoseconds

tDF: read (MRDC# rise) to Data Float

(4 x CLK2 period) − PAL RegOut Max + PAL RegOut Min
+ xcvr. Enable Min
(4 x 31.25) − 12 + 0
+ 3

= 116 nanoseconds

To pipeline the address outputs for sequential EPROM accesses, the address decoder logic must generate the Next Address (NA#) signal. Note that the initial EPROM cycle following the idle cycle cannot be address pipelined because there is no previous bus cycle. In addition, the bus cycle following the last EPROM access is pipelined regardless of which device it accesses, because the address is output before the bus cycle destination can be determined.

6.2.7 SRAM Interface

In the SRAM interface, the OE# input of each SRAM device is connected to the MRDC# signal from the bus controller; the WE# input of each SRAM device is driven by the MWTC# signal. Because it is possible to write to only some bytes of a doubleword, the MWTC# signal cannot connect directly to the WE# inputs. Each WE# input must be qualified by the latched 80386 byte-enable signals (BE3#-BE0#).

Figure 6-7 shows the signal timing for bus cycles to an Intel SRAM, which has a 100-nanosecond access time. The timing is essentially the same as for an EPROM.

The bus cycle timings can be calculated from the waveforms in Figure 6-7. In the following example, the timings for I/O accesses are calculated for CLK2 = 32 MHz and B-series PALs. All times are in nanoseconds. Check the most recent *80386 Data Sheet* to confirm all parameter values.

tAR: Address stable before Read (MRDC# fall)
tAW: Address stable before Write (MWTC# fall)

(1 x CLK2 period) − PAL RegOut Max − Latch Enable Max
+ PAL RegOut Min
(1 x 31.25) − 12 − 11.5
+ 0

= 7.75 nanoseconds

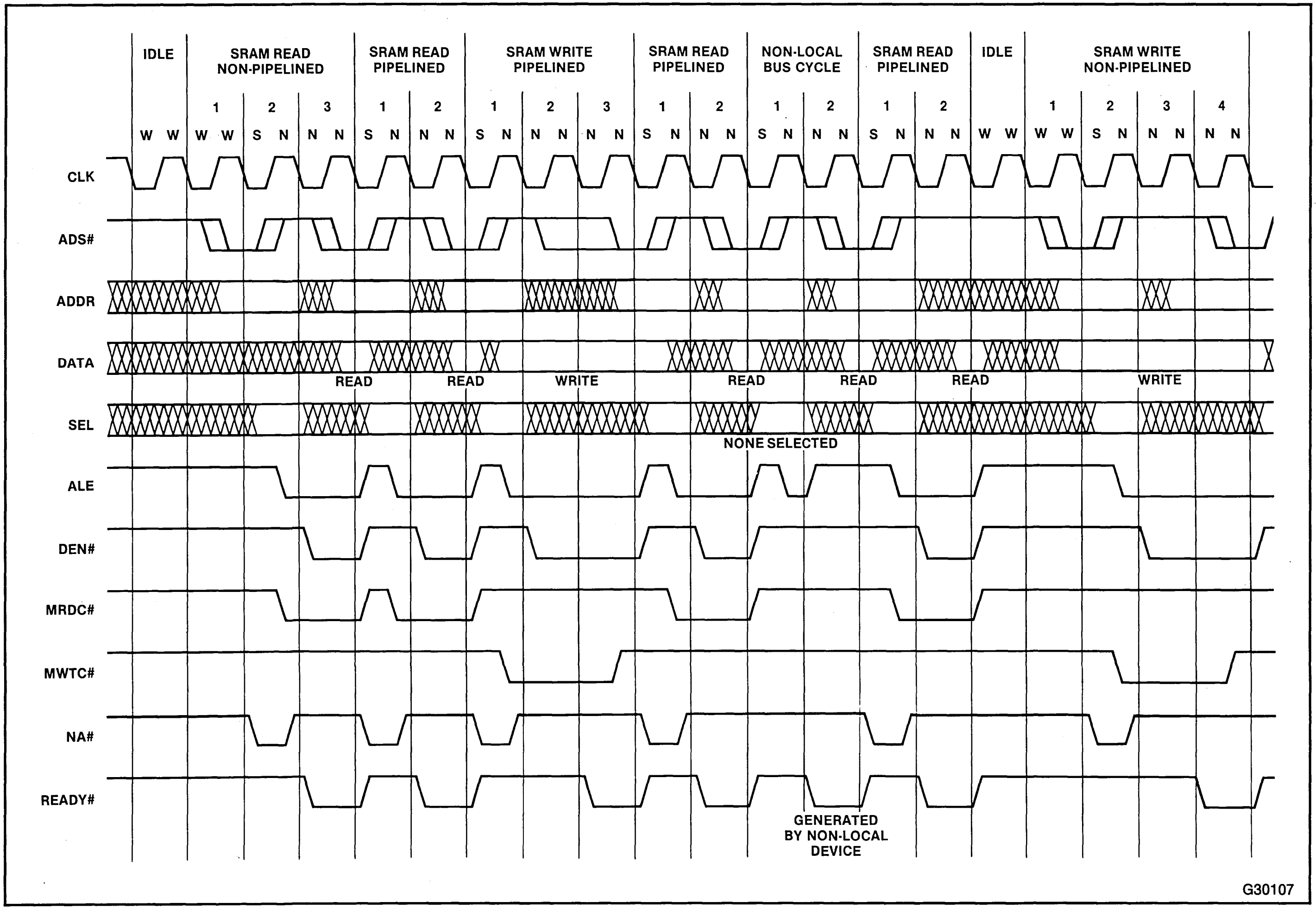

IDLE
SRAM READ NON-PIPELINED
SRAM READ PIPELINED
SRAM WRITE PIPELINED
SRAM READ PIPELINED
NON-LOCAL BUS CYCLE
SRAM READ PIPELINED
IDLE
SRAM WRITE NON-PIPELINED
1 2 3 1 2 1 2 3 1 2 1 2 1 2 1 2 3 4
W W W W S N N N S N N N S N N N N N S N N N S N N N S N N N W W W W S N N N N N
CLK
ADS#
ADDR
DATA
READ
READ
WRITE
READ
READ
READ
WRITE
SEL
NONE SELECTED
ALE
DEN#
MRDC#
MWTC#
NA#
READY#
GENERATED BY NON-LOCAL DEVICE
G30107

Figure 6-7. 100-Nanosecond SRAM Timing Diagram tRR: Read (MRDC#) pulse width

(3 x CLK2 period) − PAL RegOut Max + PAL RegOut Min
(3 x 31.25) − 12 + 0

= 81.75 nanoseconds

tWW: Write (MWTC#) pulse width

(4 x CLK2 period) − PAL RegOut Max + PAL RegOut Min
(4 x 31.25) − 12 + 0

= 113 nanoseconds

tRA: Address Hold after Read (MRDC# rise)

(0 x CLK2 period) − PAL RegOut Max + PAL RegOut Min
+ Latch Enable
(0 x 31.25) − 12 + 0
+ 5

= −7 nanoseconds (This is acceptable because latched addresses are held for at least as long as the end of the bus cycle.)

tWA: Address hold after Write (MWTC# rise)

(1 x CLK2 period) − PAL RegOut Max + PAL RegOut Min
+ Latch Enable Min
(1 x 31.25) − 12 + 0
+ 5

= 24.25 nanoseconds

tAD: Data delay from Address

(4 x CLK2 period) − PAL RegOut Max − Latch Enable Max
− xcvr. prop. Min − 80386 Data Setup Min
(4 x 31.25) − 12 − 11.5
− 6 − 10

= 85.5 nanoseconds

tRD: Data delay from Read (MRDC#)

(3 x CLK2 period) − PAL RegOut Max − xcvr. prop Min
− 80386 Data Setup Min
(3 x 31.25) − 12 − 6
− 10

= 65.75 nanoseconds

tDF: read (MRDC# rise) to Data Float

(2 x CLK2 period) − Pal RegOut Max + PAL RegOut Min
+ xcvr. Enable Min
(2 x 31.25) − 12 + 0
+ 3

= 47.5 nanoseconds

tDW: Data setup before write (MWTC# rise)

(3 x CLK2 period) − PAL RegOut Max − xcvr. Enable Max
+ PAL RegOut Min
(3 x 31.25) − 12 − 11
+ 0

= 70.75 nanoseconds

tWD: Data hold after write (MWTC# rise)

(1 x CLK2 period) − PAL RegOut Max + PAL RegOut Min
+ xcvr. Disable Min
(1 x 31.25) − 12 + 0 + 2

= 21.25 nanoseconds

6.2.8 16-Bit Interface

The use of a 16-bit data bus can be advantageous for some systems. Memory implemented as 16-bits wide rather than 32-bits wide reduces chip count. I/O addresses located at word boundaries rather than doubleword boundaries can be software compatible with some systems that use 16-bit microprocessors.

For example, if BS16# is asserted for EPROM accesses, only two byte-wide EPROMs are needed. Overall performance is reduced because 32-bit data accesses and all code prefetches from the EPROMs are slower (requiring two bus cycles instead of one). However, this reduction is acceptable in certain applications. A system that uses EPROMs only for power-on initialization and runs programs entirely from SRAM or DRAM has only a power-on time increase over the 32-bit EPROM system; its main programs run at the same speed as the 32-bit system.

The 80386 BS16# input directs the 80386 to perform data transfers on only the lower 16 bits of the data bus. In systems in which 16-bit memories are used, the address decoder logic must generate the BS16# signal for 16-bit accesses. Since NA# cannot be asserted during a bus cycle in which BS16# is asserted (because the current address may be needed for additional cycles), the decoder logic should also guarantee that the NA# signal is not generated. When the 80386 samples BS16# active and NA# inactive, it automatically performs any extra bus cycles necessary to complete a transfer on a 16-bit bus. The 80386 response is determined by the size and alignment of the data to be transferred, as described in Chapter 3.

6.3 DYNAMIC RAM (DRAM) INTERFACE

This section presents a dynamic RAM (DRAM) memory subsystem design that is both cost-effective and fast. The design can be adapted for a wide variety of speed and system requirements to provide high throughput at minimum cost. The DRAM design in this section is to illustrate concepts only; the actual circuit has not been tested.

6.3.1 Interleaved Memory

DRAMs provide relatively fast access times at a low cost per bit; therefore, large memory systems can be created at low cost. However, DRAMs have the disadvantage that they require a brief idle time between accesses to precharge; if this idle time is not provided, the data in the DRAM can be lost. If back-to-back accesses to the same bank of DRAM chips are performed, the second access must be delayed by the precharge time. To avoid this delay, memory should be arranged so that each subsequent memory access is most likely to be directed to a different bank. In this configuration, wait time between accesses is not required because while one bank of DRAMs performs the current access, another bank precharges and will be ready to perform the next access immediately.

Most programs tend to make subsequent accesses to adjacent memory locations during code fetches, stack operations, and array accesses, for example. If DRAMs are interleaved (i.e., arranged in multiple banks so that adjacent addresses are in different banks), the DRAM precharge time can be avoided for most accesses. With two banks of DRAMs, one for even 32-bit doubleword addresses and one for odd doubleword addresses, all sequential 32-bit accesses can be completed without waiting for the DRAMs to precharge.

Even if random accesses are made, two DRAM banks allow 50 percent of back-to-back accesses to be made without waiting for the DRAMs to precharge. The precharge time is also avoided when the 80386 has no bus accesses to be performed. During these idle bus cycles, the most recently accessed DRAM bank can precharge so that the next memory access to either bank can begin immediately.

The DRAM memory system design described here uses two interleaved banks of DRAMs. The DRAM controller keeps track of the most recently accessed bank in order to guarantee the precharge time for both banks while allowing memory accesses to begin as soon as possible.

6.3.2 DRAM Memory Performance

Table 6-2 shows the performance that can be obtained using this DRAM design with a variety of processor and DRAM speeds. Performance is indicated by the number of wait states per bus cycle (the number of CLK cycles in addition to the two-CLK minimum time required to complete the access).

The performance for each processor and DRAM speed combination is given for both the case of an access to the opposite bank of interleaved memories, in which no precharge time is required, and the case of an access to the same bank, in which the precharge time is factored in.

Table 6-2. DRAM Memory Performance

80386 Clock Rate	DRAM Access Time (Nanoseconds)	Bus Cycle Wait-States	
		Interleaved Piped:Unpiped	Same Bank
12 MHz	120	0 * : 1 *	1 *
12 MHz	200	1 : 2	3
16 MHz	80	0 * : 1 *	1 *
16 MHz	100	0 * : 1 *	1*
16 MHz	150	1 : 2	3

*Add one additional wait-state to these times for write accesses.

Note: The numbers for the 100-nanosecond DRAM are based on the assumption that no data transceivers are used.

The number of wait states required for interleaved accesses is based on the assumption that the address for the next access is pipelined. For cycles in which the address is not pipelined, one extra wait-state must be added to the number in Table 6-2. This requirement applies to all cycles that follow an idle bus state because these cycles can never be pipelined.

The number of wait states for same-bank accesses applies only to back-to-back cycles (without intervening idle bus time) to the same bank of DRAMs. Because the controller must allow the DRAMs to precharge before starting the access, address pipelining does not speed up the same-bank cycle; the number of wait states is identical with or without address pipelining.

The numbers in Table 6-2 are affected by DRAM refresh cycles. All DRAMs require periodic refreshing of each data cell to maintain the correct voltage levels. An access to a memory cell, called a refresh cycle, accomplishes the refresh. During one of these periodic refresh cycles, the DRAM cannot respond to processor requests.

Although the distributed DRAM refresh cycles occur infrequently, they can delay the current access so that the current access requires a total of up to four wait states (for the cases marked with an asterisk (*)) or eight wait states (for the other cases).

6.3.3 DRAM Controller

The performances shown in Table 6-2 are derived from two DRAM controller designs that differ in the number of CLKs they allow for read/refresh access and precharge times. The two designs are designated by the number of CLKs required for a read cycle. The 2-CLK design is used for the cases in Table 6-2 that are marked with an asterisk (*); the 3-CLK design is used for the other cases.

6.3.3.1 3-CLK DRAM CONTROLLER

Figure 6-8 shows a schematic of the 3-CLK DRAM controller. The DRAM array contains two banks of 32-bit-wide DRAMs. The top and bottom halves of the pictured array represent the two banks, which are each divided vertically along the four bytes for each doubleword.

The DRAM chips used to create the DRAM banks can be of any length (N), and they can be either one bit or four bits wide. If Nx1 DRAM chips are used, 64 chips are required for the two banks; if Nx4 DRAM chips are used, only 16 chips are required. The banks in Figure 6-8 are made from sixteen 64Kx4 DRAMs, but another type of DRAM can be substituted easily.

Two Row Address Strobe (RAS) signals are generated by the controller, one for each bank. The top bank is activated by RAS0# and contains the DRAM memory locations for which the 80386 address bit A2 is low. The bottom bank is activated by RAS1#, which corresponds to 80386 addresses for which A2 is high.

Four Column Address Strobe (CAS) signals are used, one for each byte of the 80386 data bus. These CAS signals are shared by both banks. The 80386 Byte Enable signals (BE3#-BE0#) map directly to the CAS signals (CAS3#-CAS0#). CAS0# is mapped directly from BE0# and enables the least-significant byte (D7-D0). Similarly, CAS3# is mapped directly from BE3# and enables the most-significant byte (D31-D24).

Each of the 32 data lines of the 80386 are connected to one DRAM chip from each bank. If Nx1 DRAMs are used, the corresponding data line is connected to both the Din and Dout pins. If Nx4 DRAMs are used, each data line is connected only to the corresponding I/O pin.

The Write Enable (WE#) signal and the multiplexed address signals are connected to every DRAM chip in both banks. Nx4 DRAMs also require an Output Enable (OE#) signal for every DRAM chip in both banks.

A single WE# control signal and four CAS control signals ensure that only those DRAM bytes selected for a write cycle are enabled. All other data bytes maintain their outputs in the high-impedance state. A common design error is to use a single CAS# control signal and four WE# control signals, using the WE# signals to write the DRAM bytes selectively in write cycles that use fewer than 32 bits. However, although the selected bytes are written correctly, the unselected bytes are enabled for a read cycle. These bytes output their data to the unselected bits of the data bus while the data transceivers output data to every bit of the data bus. When two devices simultaneously output data to the same bus, reliability problems and even permanent component damage can result. Therefore, a DRAM design should use CAS signals to enable bytes for a write cycle.

DRAMs require both the row and column addresses to be placed sequentially onto the multiplexed address bus. A set of 74F258 multiplexers accomplishes this function.

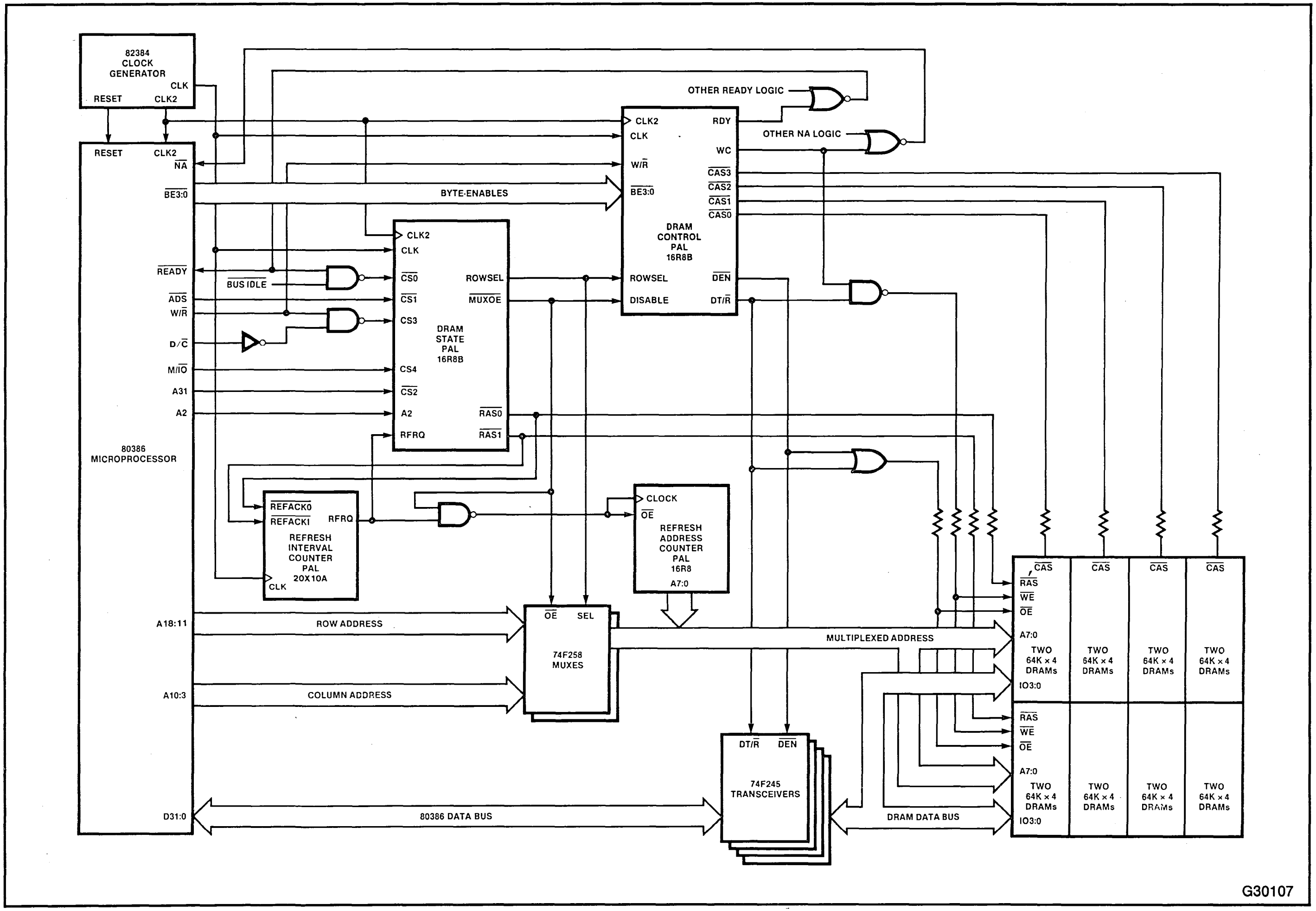

82384
CLOCK
GENERATOR
CLK
RESET
CLK2
RESET
CLK2
NA
BE3:0
BYTE-ENABLES
OTHER READY LOGIC
OTHER NA LOGIC
CLK2
CLK
W/R
BE3:0
RDY
WC
CAS3
CAS2
CAS1
CAS0
DRAM
CONTROL
PAL
16R8B
ROWSEL
DISABLE
DEN
DT/R
READY
BUS IDLE
ADS
W/R
D/C
M/IO
A31
A2
CLK2
CLK
CS0
CS1
CS3
CS4
CS2
A2
RFRQ
ROWSEL
MUXOE
DRAM
STATE
PAL
16R8B
RAS0
RAS1
80386
MICROPROCESSOR
REFACK0
REFACK1
RFRQ
REFRESH
INTERVAL
COUNTER
PAL
20X10A
CLK
CLOCK
OE
REFRESH
ADDRESS
COUNTER
PAL
16R8
A7:0
A18:11
ROW ADDRESS
OE
SEL
74F258
MUXES
A10:3
COLUMN ADDRESS
MULTIPLEXED ADDRESS
CAS
RAS
WE
OE
A7:0
TWO
64K x 4
DRAMs
IO3:0
CAS
CAS
CAS
RAS
WE
OE
A7:0
IO3:0
DT/R
DEN
74F245
TRANSCEIVERS
D31:0
80386 DATA BUS
DRAM DATA BUS
G30107

Figure 6-8. 3-CLK DRAM Controller Schematic

Four 74F245 octal transceivers buffer the DRAM from the data bus. Most DRAMs used in the 3-CLK design require these transceivers to meet the read-data float time. When a DRAM read cycle is followed immediately by an 80386 write cycle, the 80386 drives its data bus one CLK2 period after the read cycle completes. If the data transceivers are omitted, the RAS inactive delay plus the DRAM output buffer turn-off time (t-OFF) must be less than a CLK2 period to avoid data bus contention.

PALs are used to monitor the 80386 status signals and generate the appropriate control signals for the DRAM, multiplexer, and transceivers. PAL codes and pin descriptions for the 3-CLK design are listed in Appendix C of this manual.

The DRAM State PAL performs the following functions:

- Monitors the 80386 DRAM chip-select logic
- Receives DRAM refresh requests and responds with the necessary DRAM cycles
- Keeps track of DRAM banks requiring precharge time

A DRAM read or write access is requested when all the chip-select signals of the DRAM State PAL are sampled active simultaneously. These signals become active when all of the following conditions exist at once:

- M/IO#, W/R#, and D/C# outputs of the 80386 indicate either a memory read, memory write, or code fetch.
- The bus is idle or the current bus cycle is ending (READY# active).
- ADS# is active.
- A31 is low (in this design, the lower half (two gigabytes) of 80386 memory space is mapped to the DRAM controller).

If the DRAM controller is not already performing a cycle, it begins the access immediately. However, if the DRAM controller is performing a refresh cycle, or if it is waiting for the DRAM bank to precharge, the request is latched and performed when the controller is not busy.

The DRAM Control PAL generates the majority of the DRAM control signals. The Refresh Interval Counter PAL is a timer that generates refresh requests at the necessary intervals. The Refresh Address Counter PAL maintains the next refresh address. Both the Refresh Interval Counter PAL and the Refresh Address Counter PAL are simple enough to be replaced by TTL counter chips; however, the use of PALs reduces the total chip count. If there is a spare timer or counter in the system, it can be used to replace one or both of these PALs.

Figure 6-9 shows the timing of DRAM control signals for the 3-CLK design for the following five sequential DRAM cycles:

1. Read cycle
2. Write cycle to the opposite bank (no precharge)

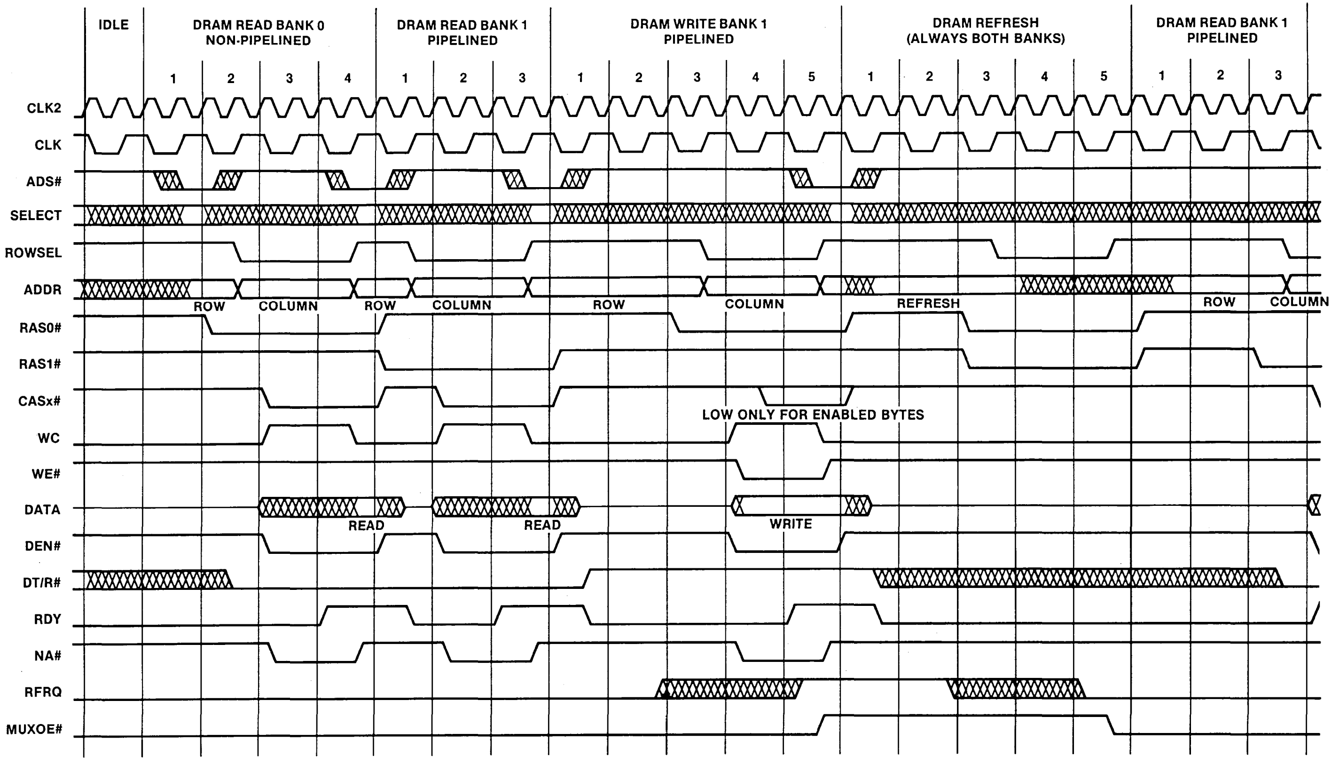

IDLE
DRAM READ BANK 0 NON-PIPELINED
DRAM READ BANK 1 PIPELINED
DRAM WRITE BANK 1 PIPELINED
DRAM REFRESH (ALWAYS BOTH BANKS)
DRAM READ BANK 1 PIPELINED
1 2 3 4 1 2 3 1 2 3 4 5 1 2 3 4 5 1 2 3
CLK2
CLK
ADS#
SELECT
ROWSEL
ADDR
ROW
COLUMN
ROW
COLUMN
ROW
COLUMN
REFRESH
ROW
COLUMN
RAS0#
RAS1#
CASx#
LOW ONLY FOR ENABLED BYTES
WC
WE#
DATA
READ
READ
WRITE
DEN#
DT/R#
RDY
NA#
RFRQ
MUXOE#
G30107

Figure 6-9. 3-CLK DRAM Controller Cycles

3. Read cycle to that same bank (requires precharge)
4. Refresh cycle (always requires precharge)
5. Read cycle (cycle after refresh always requires precharge)

During a normal DRAM access, only the RAS signal that corresponds to the selected bank is activated. During a refresh cycle, both RAS signals are activated. During write cycles, only the CAS signals corresponding to the enabled bytes are activated. During read cycles, all CAS signals are enabled.

6.3.3.2 2-CLK DRAM CONTROLLER

Figure 6-10 is a schematic of the 2-CLK design, which provides zero wait-state operation for pipelined interleaved accesses. The design differs from the 3-CLK controller in several ways.

Read and refresh cycles are completed in only two CLKs; write cycles require three CLKs to ensure that the 80386 write data is valid.

In general, the PALs that generate RAS and CAS signals can be either registered or combinatorial on the RAS and/or CAS outputs. If external registers are used, these PAL outputs must be combinatorial so that the output has time to set up the external register on the same CLK2 cycle. If the PAL outputs are used without external registers, the PAL outputs must be registered internally. When the CAS signals are registered internally, the DRAM Control PAL can sample and save the state of the Byte Enable (BE3#-BE0#) lines internally. When the CAS signals are registered externally, the BE3#-BE0# lines must be latched externally so that the DRAM Control PAL inputs maintain the valid byte enables.

For the 2-CLK design, the RAS and CAS signals are registered externally. The delay (from CLK2) for these signals is reduced, and more time is available for the DRAMs to respond. If more drive is required on these signals, multiple TTL registers can be used, each driving a small group of RAS or CAS lines. For example, in a design using Nx1 DRAMs, RAS0# must drive 32 DRAMs. To reduce the worst-case skew (caused by the heavy loading), RAS0# can be output on four register outputs, each of which drives eight DRAMs.

In the 3-CLK design, the column address does not need to be latched because the Next Address (NA#) signal is not activated until after the CAS signals go active, so the 80386 address remains valid for the memory access. Because a 2-CLK design has a shorter cycle time, NA# must be activated before the CAS signals go active to output the next address one CLK early. This early address necessitates a latch for the column addresses. In many cases, with a generalized LE control, this latch can be shared with the I/O subsystem, which usually must latch the address.

In the 2-CLK design, NA# is generated from the DRAM State PAL outputs and is therefore active in both the CLK2 cycle in which the 80386 samples NA# and the next CLK2 cycle. However, the 80386 does not sample NA# active twice. Once the 80386 outputs the next address, the address must be valid for at least two CLKs before NA# is sampled again.

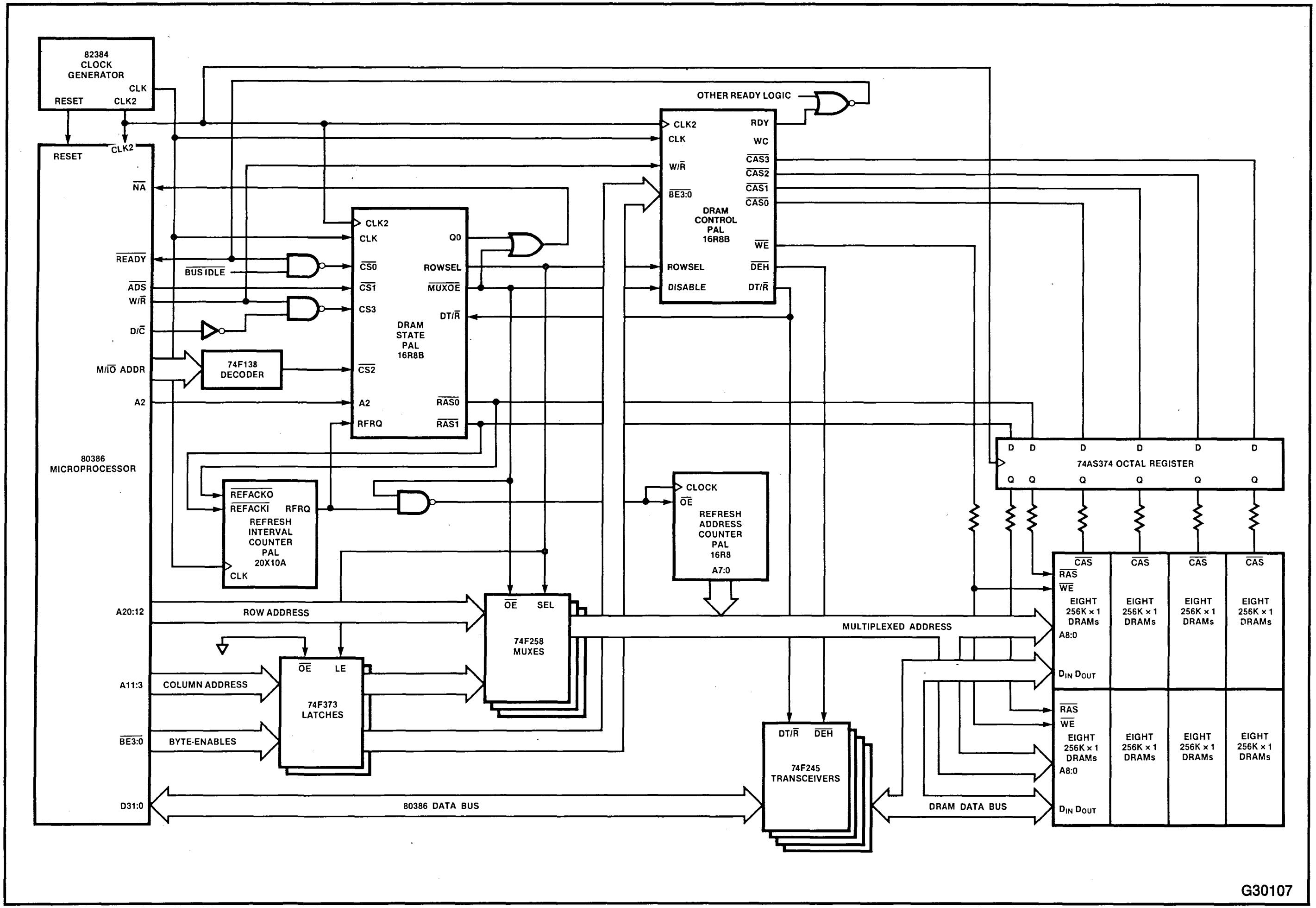

82384
CLOCK
GENERATOR
CLK
RESET
CLK2
RESET
CLK2
NA
READY
ADS
W/R
D/C
M/IO ADDR
A2
80386
MICROPROCESSOR
A20:12
A11:3
BE3:0
D31:0
BUS IDLE
74F138
DECODER
CLK2
CLK
CS0
CS1
CS3
CS2
A2
RFRQ
Q0
ROWSEL
MUXOE
DT/R
DRAM
STATE
PAL
16R8B
RAS0
RAS1
REFACK0
REFACK1
RFRQ
REFRESH
INTERVAL
COUNTER
PAL
20X10A
CLK
OTHER READY LOGIC
CLK2
CLK
W/R
BE3:0
ROWSEL
DISABLE
RDY
WC
CAS3
CAS2
CAS1
CAS0
DRAM
CONTROL
PAL
16R8B
WE
DEH
DT/R
CLOCK
OE
REFRESH
ADDRESS
COUNTER
PAL
16R8
A7:0
D
Q
74AS374 OCTAL REGISTER
ROW ADDRESS
OE
SEL
74F258
MUXES
COLUMN ADDRESS
OE
LE
74F373
LATCHES
BYTE-ENABLES
MULTIPLEXED ADDRESS
80386 DATA BUS
DT/R
DEH
74F245
TRANSCEIVERS
DRAM DATA BUS
CAS
RAS
WE
EIGHT
256K x 1
DRAMs
A8:0
D_{IN} D_{OUT}
G30107

Figure 6-10. 2-CLK DRAM Controller Schematic

In the 2-CLK design, the four data transceivers are optional because fast DRAMs with short read-data-float times are used. The DRAM data pins and can be connected directly to the 80386 data bus. The strong drive capability of the 80386 data bus can handle the load of one DRAM from each bank plus a transceiver load for the other peripherals.

PAL codes and pin descriptions for the 2-CLK design are listed in Appendix C of this manual. Figure 6-11 shows the timing of DRAM control signals for the 2-CLK design for the following five sequential DRAM cycles:

1. Read cycle
2. Write cycle to the opposite bank (no precharge)
3. Read cycle to that same bank (requires precharge)
4. Refresh cycle (always requires precharge)
5. Read cycle (cycle after refresh always requires precharge)

6.3.4 DRAM Design Variations

Some of the possible variations of the 2-CLK and 3-CLK designs are as follows:

- Both the 3-CLK and 2-CLK designs can use any length DRAM in Nx1 and Nx4 widths.
- Both 3-CLK or 2-CLK designs can use the internal PAL registers or external TTL registers on the RAS and/or CAS signals. A conservative design using Nx1 DRAMs might externally register each RAS (which drives 32 DRAMs) and internally register each CAS (which drives only 16 DRAMs).

Because internal registers have a greater maximum delay time and potentially less drive, the choice between registered PALs or external registers affects all of the DRAM timing parameters based on RAS and CAS. Some of the DRAM parameters are also affected by the minimum delay time of the internally registered PALs. Because PALs do not guarantee a minimum delay time, external TTL registers, which do guarantee a minimum delay time, can help meet these timing parameters, as well as provide greater drive capability.

- Data transceivers are optional for both designs. If a data transceiver is used, the DRAM read access must meet the 80386 read-data setup time. If no data transceiver is used, the DRAM read-data-float time must not interfere with the next 80386 cycle, particularly if it is a write cycle, and the 80386 data pin loading must not be exceeded.
- By including the column address latch and other circuitry, the DRAM controller can be adapted to run either 3-CLK or 2-CLK cycles depending on the speed (and cost) of DRAMs installed. To switch between the 3-CLK and 2-CLK controllers, the user should plug in a different set of DRAMs, a different DRAM State PAL, and a different DRAM Control PAL, and jumper the NA# logic.
- The choice of chip-select logic in both of the designs is arbitrary. Other DRAM memory-mapping schemes can be implemented by modifying the address decoding to the DRAM State PAL chip-selects.

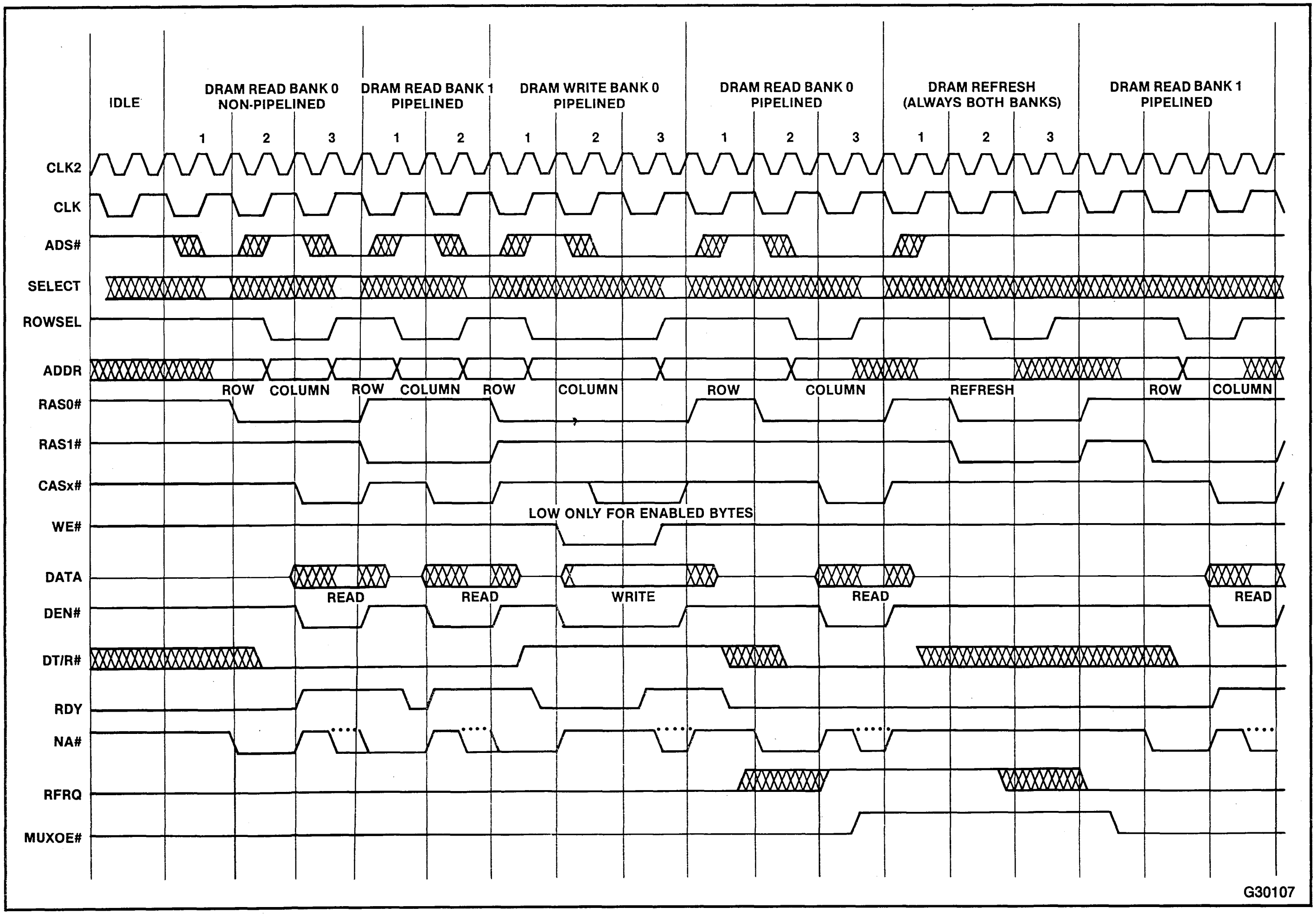

IDLE
DRAM READ BANK 0 NON-PIPELINED
DRAM READ BANK 1 PIPELINED
DRAM WRITE BANK 0 PIPELINED
DRAM READ BANK 0 PIPELINED
DRAM REFRESH (ALWAYS BOTH BANKS)
DRAM READ BANK 1 PIPELINED
1
2
3
1
2
1
2
3
1
2
3
1
2
3
CLK2
CLK
ADS#
SELECT
ROWSEL
ADDR
ROW
COLUMN
ROW
COLUMN
ROW
COLUMN
ROW
COLUMN
REFRESH
ROW
COLUMN
RAS0#
RAS1#
CASx#
LOW ONLY FOR ENABLED BYTES
WE#
DATA
READ
READ
WRITE
READ
READ
DEN#
DT/R#
RDY
NA#
RFRQ
MUXOE#
G30107

Figure 6-11. 2-CLK DRAM Controller Cycles

- For a single DRAM bank rather than two, the user should tie the DRAM State PAL A2 input low, leave RAS1# unconnected (only RAS0# is used), and feed the 80386 address bit A2 into the address multiplexer. The DRAM State PAL equations can be modified to change the RAS1# output to duplicate the RAS0# output for more drive capability, and the A2 input can be used as another chip-select input. When only one bank is used, no accesses can be interleaved, and back-to-back accesses run with one wait state with the 2-CLK design and three wait states with the 3-CLK design (independent of address pipelining).

6.3.5 Refresh Cycles

All DRAMs require periodic refreshing of their data. For most DRAMs, periodic activation of each of the row address signals internally refreshes the data in every column of the row. Almost all DRAMs allow a RAS-only refresh cycle, the timing of which is the same as a read cycle, except that only the RAS signals are activated (no CAS signals), and all of the data pins are in the high impedance state.

Both the 3-CLK and 2-CLK designs use RAS-only refresh. The address multiplexer is placed in the high impedance state, and the Refresh Address Counter PAL is enabled to output the address of the next row to be refreshed. Then the DRAM State PAL activates both RAS0# and RAS1# to refresh the selected row for both banks at once. After the refresh cycle is complete, the Refresh Address Counter PAL increments so that the next refresh cycle refreshes the next sequential row.

The frequency of refreshing and the number of rows to be refreshed depend on the type of DRAM. For most larger DRAMs (64KxN and larger), only the lower eight multiplexed address bits (A7-A0, 256 rows) must be supplied for the refresh cycle; the upper address bits are ignored. The Refresh Address Counter PAL must output only eight bits and only the lower eight bits of the address multiplexer must be placed in the high impedance state. The OE# signals of the higher order address multiplexers can be tied low. Larger DRAMs generally require refresh every 4 milliseconds. The following sections describe refresh specifically for larger DRAMs, although the concepts apply to smaller DRAMs.

6.3.5.1 DISTRIBUTED REFRESH

In distributed refresh, the 256 refresh cycles are distributed equally within the 4-millisecond interval. Every 15.625 microseconds (4 milliseconds/256), a single row refresh is performed. After 4 milliseconds all 256 rows have been refreshed, and the pattern repeats. Both the 3-CLK and 2-CLK designs use distributed refresh.

The Refresh Interval Counter PAL is programmed to request a single distributed refresh cycle at intervals slightly under 15.625 microseconds. The counter requests a new refresh cycle after a preset number of CLK cycles. This number is dependent on the CLK frequency and can be calculated as follows for a 16-MHz CLK signal:

$$\begin{aligned} 16 \text{ MHz} \times 15.625 \text{ microseconds} - 4/256 &= 249.98 \\ &= 249 \text{ CLK cycles} \end{aligned}$$

The term 4/256 is subtracted to allow for the time it takes the DRAM State PAL to respond to the request. Refresh requests are always given highest priority; however, if a DRAM access is already in progress, it must finish before the refresh cycle can start. The 3-CLK controller responds within 1-5 CLKs of the refresh request; the 2-CLK controller responds within 1-4 CLKs. The maximum latency (the difference between the longest and shortest responses) for either design is therefore 4 CLKs. This time is spread out among all 256 accesses, so 4/256 is subtracted in the above equations to account for the latency period. The counter immediately resets itself after it reaches the maximum count, regardless of this latency period.

Distributed refresh has two advantages over other types of refresh:

- Refresh cycles are spread out, guaranteeing that the 80386 access is never delayed very long for refresh cycles. Most programs execute in approximately the same time, regardless of when they are run with respect to DRAM refreshes.
- Distributed refresh hardware is typically simpler than hardware required for other types of refresh.

6.3.5.2 BURST REFRESH

Burst refreshes perform all 256 row refreshes consecutively once every 4 milliseconds rather than distributing them equally over the time period. Once a refresh is performed, the next 4-millisecond period is guaranteed free of refresh cycles. Time-critical sections of code can be executed during this time.

The 3-CLK and 2-CLK designs can be modified for burst refreshes by lengthening the maximum count of the Refresh Interval Counter to cover a 4-millisecond interval and holding the Refresh Request (RFRQ) signal active for 256 refresh cycles instead of a single refresh cycle. The completion of 256 refresh cycles can be determined by clearing the Refresh Address Counter PAL before the first refresh cycle and monitoring the outputs until they reach the zero address again. The Row Select (ROWSEL) signal can be used to clock the Refresh Address Counter PAL. The longer interval counter and extra logic requires another PAL device.

6.3.5.3 DMA REFRESH USING THE 82380 DRAM REFRESH CONTROLLER

The 82380 DRAM Refresh Controller can be used to perform refresh operations. The 82380 refresh logic provides a 24-bit Refresh Address Counter. Timer 1 is used to initiate refresh cycles. When the refresh function is enabled, the output of Timer 1, TOUT1/REF#, becomes the Refresh Request signal. The 82380 uses DMA operation to perform DRAM refresh. During a DRAM refresh cycle, TOUT1/REF# will be activated and a Refresh Address will be placed on the Address Bus. In order to ensure that no refresh cycles will be delayed, the Refresh Request is always arbitrated with the highest priority among the DMA requests.

DMA refresh can be used for both 3-CLK and 2-CLK designs. To activate both banks, the 82380's Refresh Request (TOUT1/REF#) is ANDed with the 80386's Hold Acknowledge (HLDA) to qualify for a valid refresh operation. The output of this ANDed signal is connected to the RFRQ input of the DRAM State PAL (see Figure 6-12). The DRAM

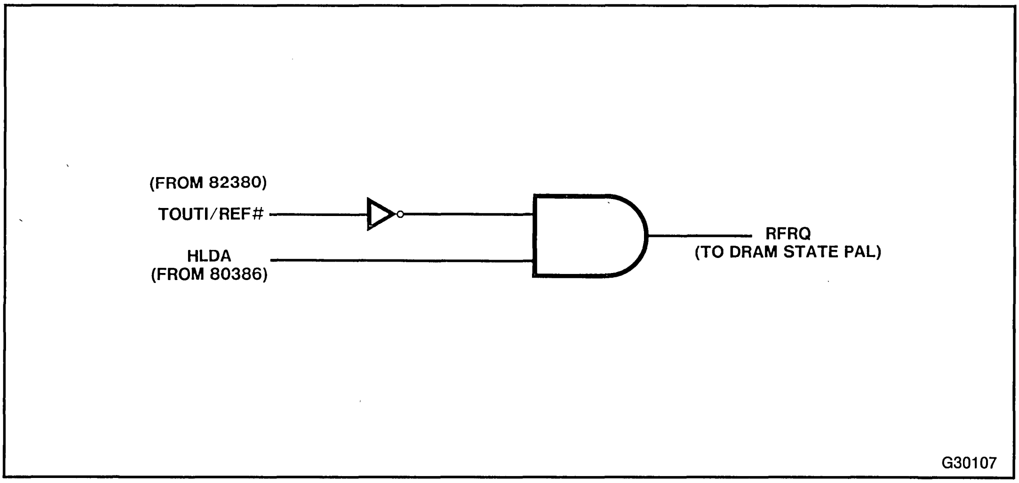


Figure 6-12. Refresh Request Generation

State PAL must be modified to ignore chip selects. This modification is needed to prevent the PAL from attempting to run a normal access cycle after the refresh cycle is complete.

In addition, the DRAM Control PAL must be modified so that the Ready (RDY) signal is generated on refresh accesses. Finally, the OE# input of the address multiplexer should be tied low so that it never enters the high impedance state, and the row address should include the least significant address bits (A10:3)

When using the 82380 DRAM Refresh Control to perform refresh, the Refresh Interval Counter PAL and the Refresh Address Counter PAL can be eliminated.

6.3.6 Initialization

Once the system is initialized, the integrity of the DRAM data and states is maintained, even during an 80386 halt or shutdown state or hardware reset, because all DRAM system functions are performed in hardware.

The controller PALs contain some state and counter information that is not implicitly reset during a power-up or hardware reset. The state machines are designed so that they enter the idle state within 18 CLK2 cycles regardless of whether they powerup in a valid state. The counters can start in any state. Thus, even though the state machines and counters can powerup into any state, they are ready for operation before the 80386 begins its first bus access.

Some DRAMs require a number of warm-up cycles before they can operate. Either method listed below can provide these cycles:

- Performing several dummy DRAM cycles as part of the 80386 initialization process. Setting up the 80386 registers and performing a REP LODS instruction is one way to perform these dummy cycles.
- Activating the RFRQ signal, using external logic, for a preset amount of time, causing the DRAM control hardware to run several refresh cycles.

6.3.7 Timing Analysis

The DRAM design (2-CLK or 3-CLK) for six combinations of DRAM speed and CLK frequency is listed in the Table 6-3. The table also indicates for each DRAM type whether data transceivers and/or external registers on the PAL outputs must be used with the design.

Appendix C of this manual contains a timing analysis of all 2-CLK and 3-CLK circuit parameters for all six DRAM types.

Table 6-3. Designs for Six DRAM Types

DRAM Access Time	386 CLK Rate	PAL Circuit	External Registers	Data xcvrs
80 nS	16 MHz	2-CLK	optional	optional
100 nS	16 MHz	2-CLK	optional	no
120 nS	12 MHz	2-CLK	optional	optional
150 nS	16 MHz	3-CLK	optional	yes
150 nS	16 MHz	3-CLK	yes	yes
200 nS	12 MHz	3-CLK	optional	yes

Cache Subsystems

7

CHAPTER 7
CACHE SUBSYSTEMS

Operating at 20 MHz, the 80386 can perform a complete bus cycle in only 100 nanoseconds, for a maximum bandwidth of 40 megabytes per second. To sustain this maximum speed, the 80386 must be matched with a high-performance memory system. The system must be fast enough to complete bus cycles with no wait states and large enough to allow the 80386 to execute large application programs.

Traditional memory systems have been implemented with dynamic RAMs (DRAMs), which provide a large amount of memory for a small amount of board space and money. However, no common low-cost DRAMs are available that can complete every bus cycle in 100 nanoseconds. Faster static RAMs (SRAMs) can meet the bus timing requirement, but they offer a relatively small amount of memory at a higher cost. Large SRAM systems can be prohibitively expensive.

A cache memory system contains a small amount of fast memory (SRAM) and a large amount of slow memory (DRAM). The system is configured to simulate a large amount of fast memory. Cache memory therefore provides the performance of SRAMs at a cost approaching that of DRAMs. A cache memory system (see Figure 7-1) consists of the following sections:

- Cache—fast SRAMs between the processor and the (slower) main memory
- Main memory—DRAMs
- Cache controller—logic to implement the cache

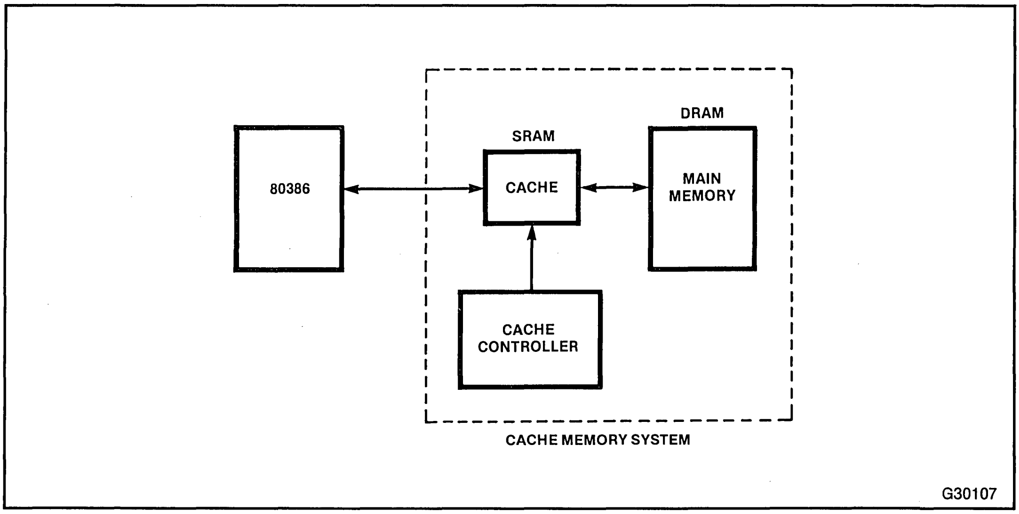


Figure 7-1. Cache Memory System

7.1 INTRODUCTION TO CACHES

In a cache memory system, all the data is stored in main memory and some data is duplicated in the cache. When the processor accesses memory, it checks the cache first. If the desired data is in the cache, the processor can access it quickly, because the cache is a fast memory. If the data is not in the cache, it must be fetched from the main memory.

A cache reduces average memory access time if it is organized so that the code and data that the processor needs most often is in the cache. Programs execute most quickly when most operations are transfers to and from the faster cache memory. If the requested data is found in the cache, the memory access is called a cache hit; if not, it is called a cache miss. The hit rate is the percentage of accesses that are hits; it is affected by the size and physical organization of the cache, the cache algorithm, and the program being run. The success of a cache system depends on its ability to maintain the data in the cache in a way that increases the hit rate. The various cache organizations presented in Section 7.2 reflect different strategies for achieving this goal.

7.1.1 Program Locality

Predicting the location of the next memory access would be impossible if programs accessed memory completely at random. However, programs usually access memory in the neighborhood of locations accessed recently. This principle is known as program locality or locality of reference.

Program locality makes cache systems possible. The same concept, on a larger scale, allows demand paging systems to work well. In typical programs, code execution usually proceeds sequentially or in small loops so that the next few accesses are nearby. Data variables are often accessed several times in succession. Stacks grow and shrink from one end so that the next few accesses are all near the top of the stack. Character strings and vectors are often scanned sequentially.

The principle of program locality pertains to how programs tend to behave, but it is not a law that all programs always obey. Jumps in code sequences and context switching between programs are examples of behavior that may not uphold program locality.

7.1.2 Block Fetch

The block fetch uses program locality to increase the hit rate of a cache. The cache controller partitions the main memory into blocks. Typical block sizes (also known as line size) are 2, 4, 8, or 16 bytes. A 32-bit processor usually uses two or four words per block. When a needed word is not in the cache, the cache controller moves not only the needed word from the main memory into the cache, but also the entire block that contains the needed word.

A block fetch can retrieve the data located before the requested byte (lookbehind), follows the requested byte (lookahead), or both. Generally, blocks are aligned (2-byte blocks on doubleword boundaries, 4-word blocks on doubleword boundaries). An access to any byte in the block copies the whole block into the cache. When memory locations are accessed in ascending order (code accesses, for example), an access to the first byte of a block in main memory results in a lookahead block fetch. When memory locations are accessed in descending order, the block fetch is look-behind.

Block size is one of the most important parameters in the design of a cache memory system. If the block size is too small, the lookahead and look-behind are reduced, and therefore the hit rate is reduced, particularly for programs that do not contain many loops. However, too large a block size has the following disadvantages:

- Larger blocks reduce the number of blocks that fit into a cache. Because each block fetch overwrites older cache contents, a small number of blocks results in data being overwritten shortly after it is fetched.
- As a block becomes larger, each additional word is further from the requested word, therefore less likely to be needed by the processor (according to program locality).
- Large blocks tend to require a wider bus between the cache and the main memory, as well as more static and dynamic memory, resulting in increased cost.

As with all cache parameters, the block size must be determined by weighing performance (as estimated from simulation) against cost.

7.2 CACHE ORGANIZATIONS

7.2.1 Fully Associative Cache

Most programs make reference to code segments, subroutines, stacks, lists, and buffers located in different parts of the address space. An effective cache must therefore hold several noncontiguous blocks of data.

Ideally, a 128-block cache would hold the 128 blocks most likely to be used by the processor regardless of the distance between these words in main memory. In such a cache, there would be no single relationship between all the addresses of these 128 blocks, so the cache would have to store the entire address of each block as well as the block itself. When the processor requested data from memory, the cache controller would compare the address of the requested data with each of the 128 addresses in the cache. If a match were found, the data for that address would be sent to the processor. This type of cache organization, depicted in Figure 7-2, is called fully associative.

A fully associative cache provides the maximum flexibility in determining which blocks are stored in the cache at any time. In the previous example, up to 128 unrelated blocks could be stored in the cache. Unfortunately, a 128-address compare is usually unacceptably slow, expensive, or both. One of the basic issues of cache organization is how to minimize the restrictions on which words may be stored in the cache while limiting the number of required address comparisons.

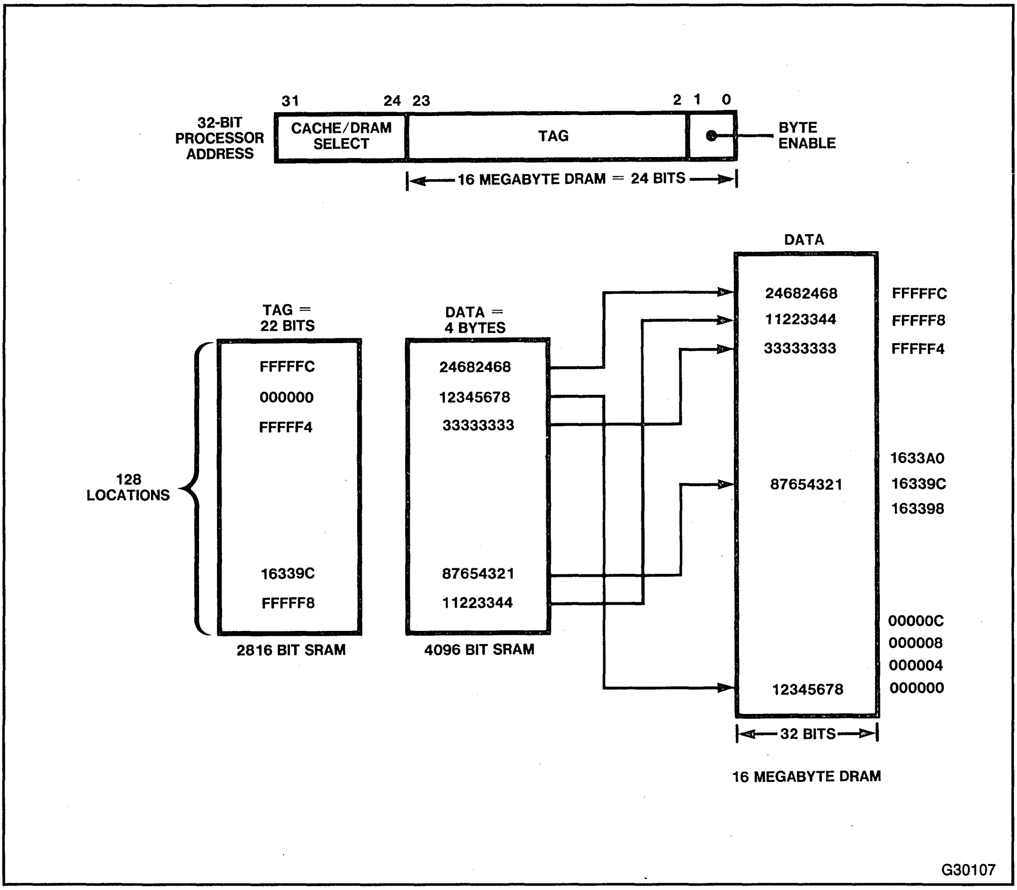

31
24 23
2 1 0
32-BIT PROCESSOR ADDRESS
CACHE/DRAM SELECT
TAG
BYTE ENABLE
16 MEGABYTE DRAM = 24 BITS
DATA
TAG = 22 BITS
DATA = 4 BYTES
128 LOCATIONS
FFFFFC
000000
FFFFF4
16339C
FFFFF8
24682468
12345678
33333333
87654321
11223344
2816 BIT SRAM
4096 BIT SRAM
24682468
11223344
33333333
87654321
12345678
FFFFFC
FFFFF8
FFFFF4
1633A0
16339C
163398
00000C
000008
000004
000000
32 BITS
16 MEGABYTE DRAM
G30107

Figure 7-2. Fully Associative Cache Organization

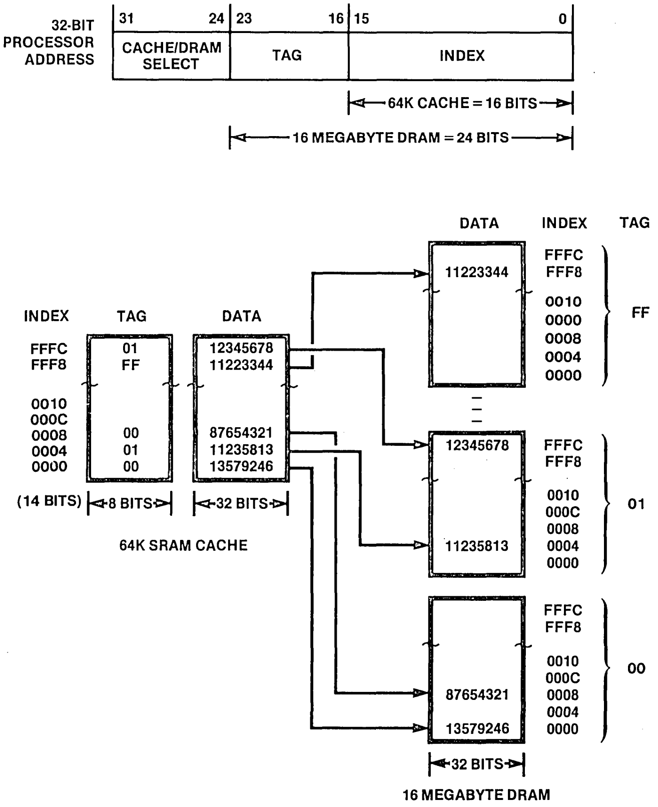
7.2.2 Direct Mapped Cache

In a direct mapped cache, unlike a fully associative cache, only one address comparison is needed to determine whether requested data is in the cache.

The many address comparisons of the fully associative cache are necessary because any block from the main memory can be placed in any location of the cache. Thus, every block of the cache must be checked for the requested address. The direct mapped cache reduces the number of comparisons needed by allowing each block from the main memory only one possible location in the cache.

Each direct mapped cache address has two parts. The first part, called the cache index field, contains enough bits to specify a block location within the cache. The second part, called the tag field, contains enough bits to distinguish a block from other blocks that may be stored at a particular cache location.

For example, consider a 64-kilobyte direct mapped cache that contains 16K 32-bit locations and caches 16 megabytes of main memory. The cache index field must include 14 bits to select one of the 16K blocks in the cache, plus 2 bits (or 4 byte Enables) to select a byte from the 4-byte block. The tag field must be 8 bits wide to identify one of the 256 blocks that can occupy the selected cache location. The remaining 8 bits of the 32-bit 80386 address are decoded to select the cache subsystem from among other memories in the memory space. The direct-mapped cache organization is shown in Figure 7-3.


32-BIT PROCESSOR ADDRESS
31
24
23
16
15
0
CACHE/DRAM SELECT
TAG
INDEX
64K CACHE = 16 BITS
16 MEGABYTE DRAM = 24 BITS
INDEX
TAG
DATA
FFFC
FFF8
0010
000C
0008
0004
0000
01
FF
00
01
00
12345678
11223344
87654321
11235813
13579246
(14 BITS)
8 BITS
32 BITS
64K SRAM CACHE
DATA
INDEX
TAG
11223344
FFFC
FFF8
0010
0000
0008
0004
0000
FF
12345678
11235813
FFFC
FFF8
0010
000C
0008
0004
0000
01
87654321
13579246
FFFC
FFF8
0010
000C
0008
0004
0000
00
32 BITS
16 MEGABYTE DRAM

G30107

Figure 7-3. Direct Mapped Cache Organization

In a system such as shown in Figure 7-3, a request for the data at the address 12FFE9H in the main memory is handled as follows:

1. The cache controller determines the cache location from the 14 most significant bits of the index field (FFE8H).
2. The controller compares the tag field (12H) with the tag stored at location FFE8H in the cache.
3. If the tag matches, the processor reads the least significant byte from the data in the cache.
4. If the tag does not match, the controller fetches the 4-byte block at address 12FFE8H in the main memory and loads it into location FFE8H of the cache, replacing the current block. The controller must also change the tag stored at location FFE8H to 12H. The processor then reads the least significant byte from the new block.

Any address whose index field is FFE8H can be loaded into the cache only at location FFE8H; therefore, the cache controller makes only one comparison to determine if the requested word is in the cache. Note that the address comparison requires only the tag field of the address. The index field need not be compared because anything stored in cache location FFE8H has an index field of FFE8H. The direct mapped cache uses direct addressing to eliminate all but one comparison operation.

The direct mapped cache, however, is not without drawbacks. If the processor in the example above makes frequent requests for locations 12FFE8H and 44FFE8H, the controller must access the main memory frequently, because only one of these locations can be in the cache at a time. Fortunately, this sort of program behavior is infrequent enough that the direct mapped cache, although offering poorer performance than a fully associative cache, still provides an acceptable performance at a much lower cost.

7.2.3 Set Associative Cache

The set associative cache compromises between the extremes of fully associative and direct mapped caches. This type of cache has several sets (or groups) of direct mapped blocks that operate as several direct mapped caches in parallel. For each cache index, there are several block locations allowed, one in each set. A block of data arriving from the main memory can go into a particular block location of any set. Figure 7-4 shows the organization for a 2-way set associative cache.

With the same amount of memory as the direct mapped cache of the previous example, the set associative cache contains half as many locations, but allows two blocks for each location. The index field is thus reduced to 15 bits, and the extra bit becomes part of the tag field.

Because the set associative cache has several places for blocks with the same cache index in their addresses, the excessive main memory traffic that is a drawback of a direct mapped cache is reduced and the hit rate increased. A set associative cache, therefore, performs more efficiently than a direct mapped cache.

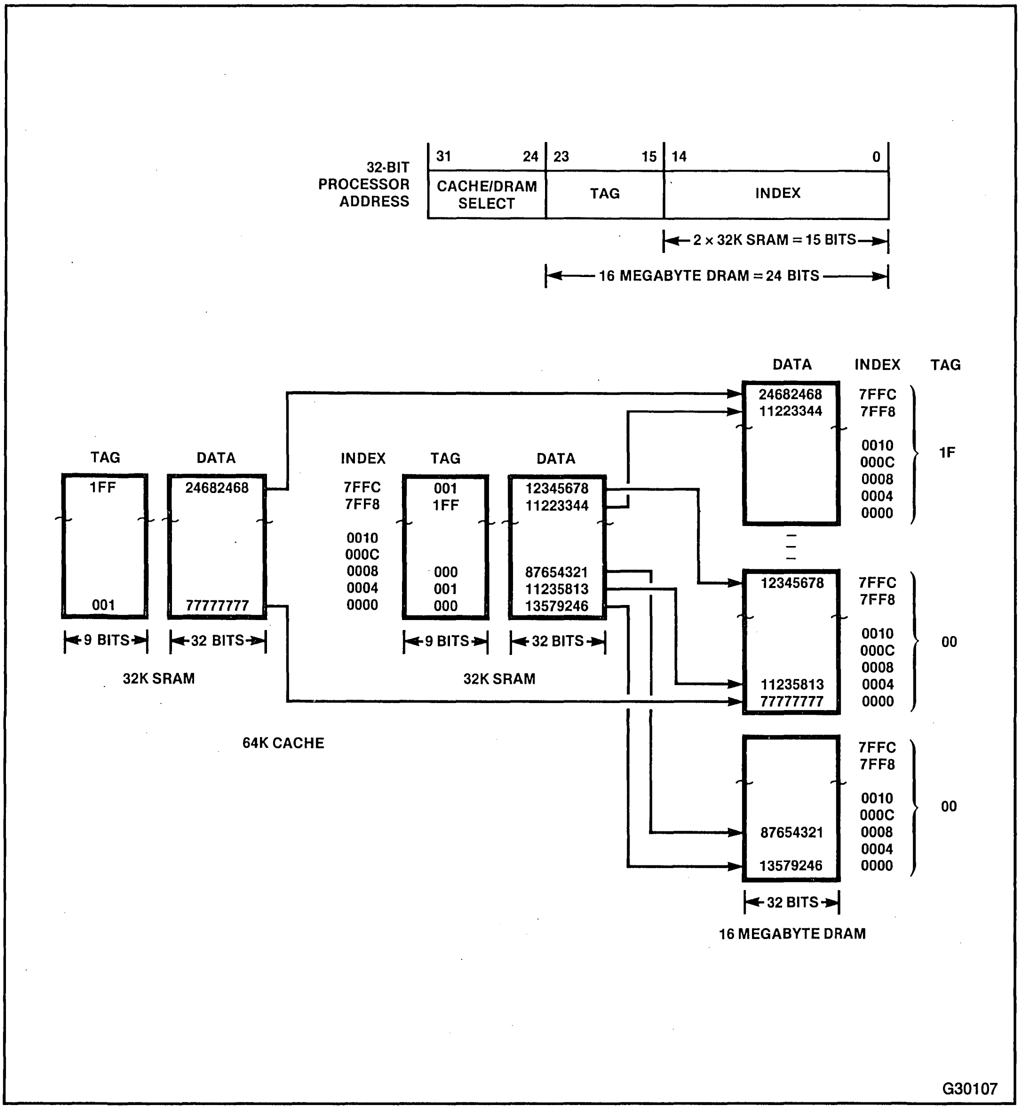

32-BIT PROCESSOR ADDRESS
31 24 23 15 14 0
CACHE/DRAM SELECT
TAG
INDEX
2 x 32K SRAM = 15 BITS
16 MEGABYTE DRAM = 24 BITS
TAG
DATA
1FF
24682468
001
77777777
9 BITS
32 BITS
32K SRAM
INDEX
7FFC
7FF8
0010
000C
0008
0004
0000
TAG
001
1FF
000
001
000
DATA
12345678
11223344
87654321
11235813
13579246
9 BITS
32 BITS
32K SRAM
64K CACHE
DATA
INDEX
TAG
24682468
11223344
7FFC
7FF8
0010
000C
0008
0004
0000
1F
12345678
11235813
77777777
00
87654321
13579246
00
32 BITS
16 MEGABYTE DRAM
G30107

Figure 7-4. Two-Way Set Associative Cache Organization

The set associative cache, however, is more complex than the direct mapped cache. In the 2-way set associative cache, there are two locations in the cache in which each block can be stored; therefore, the controller must make two comparisons to determine in which block, if any, the requested data is located. A set associative cache also requires a wider tag field, and thus a larger SRAM to store the tags, than a direct mapped cache with the same amount of cache memory and main memory. In addition, when information is placed into the cache, a decision must be made as to which block should receive the information.

The controller must also decide which block of the cache to overwrite when a block fetch is executed. There are several locations, rather than just one, in which the data from the main memory could be written. Three common approaches for choosing the block to overwrite are as follows:

- Overwriting the least recently accessed block. This approach requires the controller to maintain least-recently used (LRU) bits that indicate the block to overwrite. These bits must be updated by the cache controller on each cache transaction.
- Overwriting the blocks in sequential order (FIFO).
- Overwriting a block chosen at random.

The performance of each strategy depends upon program behavior. Any of the three strategies is adequate for most set associative cache designs; however, the LRU algorithm tends to provide the highest hit rate.

7.3 CACHE UPDATING

In a cache system, two copies of the same data can exist at once, one in the cache and one in the main memory. If one copy is altered and the other is not, two different sets of data become associated with the same address. A cache must contain an updating system to prevent old data values (called stale data) from being used. Otherwise, the situation shown in Figure 7-5 could occur. The following sections describe the write-through and write-back methods of updating the main memory during a write operation to the cache.

7.3.1 Write-Through System

In a write-through system, the controller copies write data to the main memory immediately after it is written to the cache. The result is that the main memory always contains valid data. Any block in the cache can be overwritten immediately without data loss.

The write-through approach is simple, but performance is decreased due to the time required to write the data to main memory and increased bus traffic (which is significant in multiprocessing systems).

7.3.2 Buffered Write-Through System

Buffered write-through is a variation of the write-through technique. In a buffered write-through system, write accesses to the main memory are buffered, so that the processor can begin a new cycle before the write cycle to the main memory is completed. If a write access is followed by a read access that is a cache hit, the read access can be performed while the main memory is being updated. The decrease in performance of the write-through system is thus avoided. However, because usually only a single write access can be buffered, two consecutive writes to the main memory will require the processor to wait. A write followed by a read miss will also require the processor to wait.

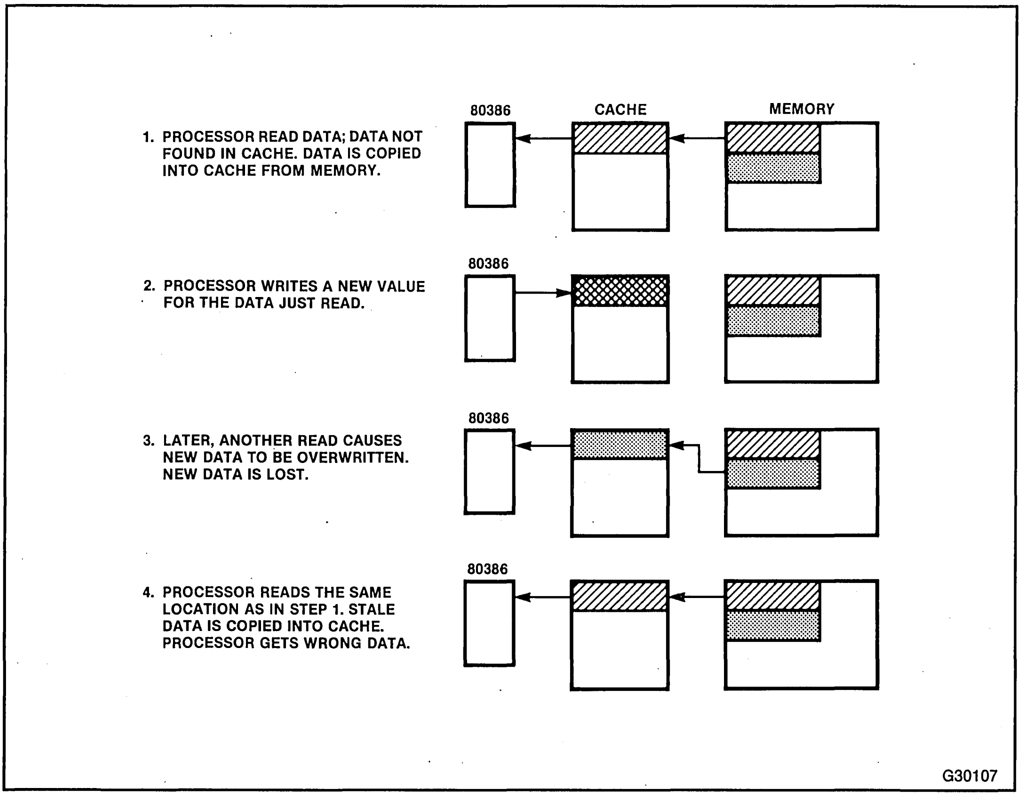

80386
CACHE
MEMORY
1. PROCESSOR READ DATA; DATA NOT FOUND IN CACHE. DATA IS COPIED INTO CACHE FROM MEMORY.
80386
2. PROCESSOR WRITES A NEW VALUE FOR THE DATA JUST READ.
80386
3. LATER, ANOTHER READ CAUSES NEW DATA TO BE OVERWRITTEN. NEW DATA IS LOST.
80386
4. PROCESSOR READS THE SAME LOCATION AS IN STEP 1. STALE DATA IS COPIED INTO CACHE. PROCESSOR GETS WRONG DATA.
G30107

Figure 7-5. Stale Data Problem

7.3.3 Write-Back System

In a write-back system, the tag field of each block in the cache includes a bit called the altered bit. This bit is set if the block has been written with new data and therefore contains data that is more recent than the corresponding data in the main memory. Before overwriting any block in the cache, the cache controller checks the altered bit. If it is set, the controller writes the block to main memory before loading new data into the cache.

Write-back is faster than write-through because the number of times an altered block must be copied into the main memory is usually less than the number of write accesses. However, write-back has these disadvantages:

- Write-back cache controller logic is more complex than write-through. When a write-back system must write an altered block to memory, it must reconstruct the write address from the tag and perform the write-back cycle as well as the requested access.
- All altered blocks must be written to the main memory before another device can access these blocks in main memory.

- In a power failure, the data in the cache is lost, so there is no way to tell which locations of the main memory contain stale data. Therefore, the main memory as well as the cache must be considered volatile and provisions must be made to save the data in the cache in the case of a power failure.

7.3.4 Cache Coherency

Write-through and write-back eliminate stale data in the main memory caused by cache write operations. However, if caches are used in a system in which more than one device has access to the main memory (multi-processing systems or DMA systems, for example), another stale data problem is introduced. If new data is written to main memory by one device, the cache maintained by another device will contain stale data. A system that prevents the stale cache data problem is said to maintain cache coherency. Four cache coherency approaches are described below:

- Bus Watching (Snooping) — The cache controller monitors the system address lines when other masters are accessing shared memory. If another master writes to a location in shared memory which also resides in the cache memory, the cache controller invalidates that cache entry. The 82385 uses snooping to maintain cache coherency in multi-master systems. Figure 7-6 illustrates bus watching.

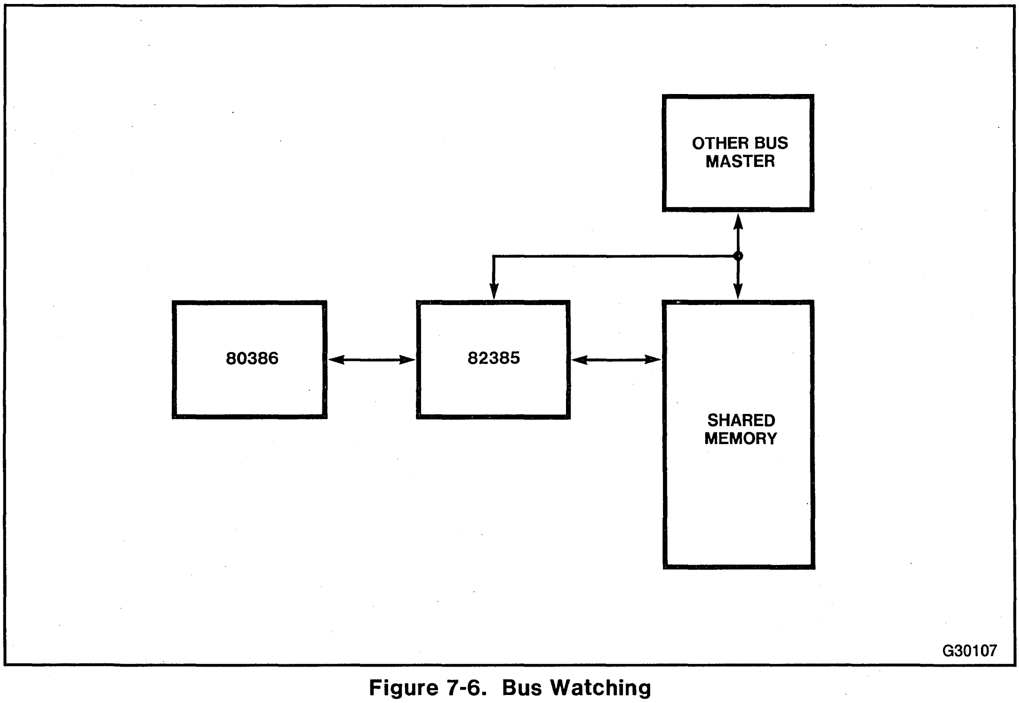


Figure 7-6. Bus Watching

- Hardware transparency—Hardware guarantees cache coherency by ensuring that all accesses to memory mapped by a cache are seen by the cache. This is accomplished either by routing the accesses of all devices to the main memory through the same cache or by copying all cache writes both to the main memory and to all other caches that share the same memory (a technique known as broadcasting). Hardware transparent systems are illustrated in Figure 7-7.

- Non-cacheable memory — Cache coherency is maintained by designating shared memory as non-cacheable. In such a system, all accesses to shared memory are cache misses, because the shared memory is never copied into the cache. The non-cacheable memory can be identified using chip-select logic or high-address bits. Figure 7-8 illustrates non-cacheable memory.

  Software can offset the reduction in the hit rate caused by non-cacheable memory by using the string move instruction (REP MOVS) to copy data between non-cacheable memory and cacheable memory and by mapping shared memory accesses to the cacheable locations. This technique is especially appropriate for systems in which copying is necessary for other reasons (as in some implementations of UNIX for example).

- Cache flushing — A cache flush writes any altered data to the main memory (if this has not been done with write-through) and clears the contents of the cache. If all the caches in the system are flushed before a device writes to shared memory, the potential for stale data in any cache is eliminated.

Combinations of various cache coherency techniques may offer the optimal solution for a particular system. For example, a system might use hardware transparency for time-critical I/O operations such as paging and non-cacheable memory for slower I/O such as printing.

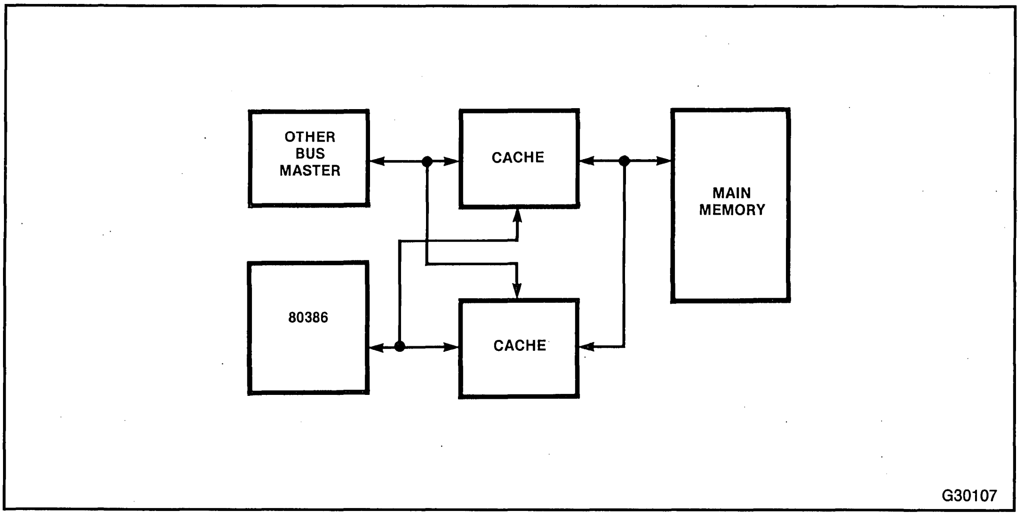


Figure 7-7. Hardware Transparency

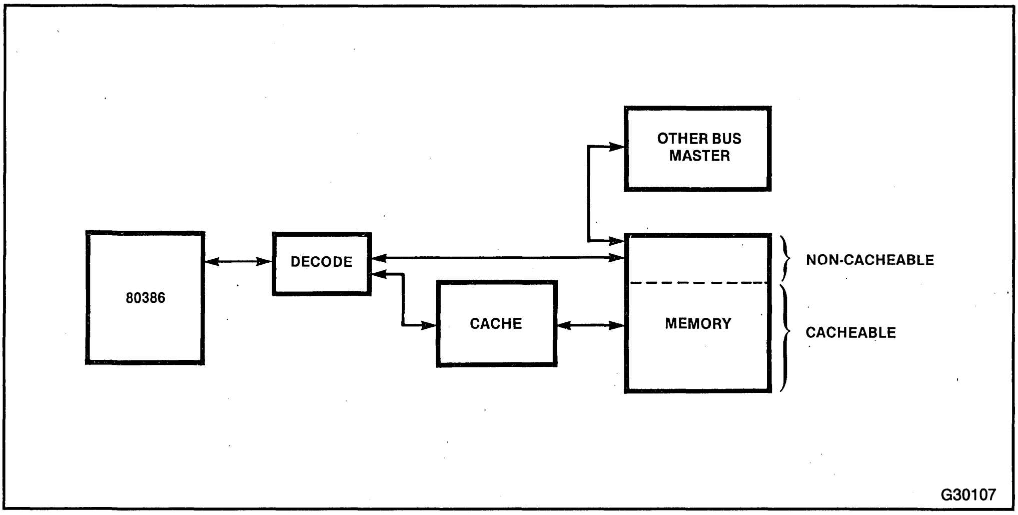


Figure 7-8. Non-Cacheable Memory

7.4 EFFICIENCY AND PERFORMANCE

The measurement of cache effectiveness is divided into two topics: efficiency and performance. Cache efficiency is its ability to maintain the most used code and data requested by the microprocessor. Efficiency is measured in terms of hit rate. Performance is a measurement of the speed in which a microprocessor can perform a given task, and is measured in effective waitstates. Hit rate is but one of many factors which affect performance. Write policy, update policy, and coherency methods are performance factors as well.

Hit rate data for various cache organizations is shown in Table 7-1. These statistics were computed by analyzing several mainframe traces, and selecting the one which produced the lowest hit rate. Thus, the numbers listed are a conservative estimate of cache efficiency. Note that hit rate statistics are not absolute quantities. The hit rate of a particular cache implementation can vary widely depending on software. Therefore, Table 7-1 should only be used to compare one cache configuration against another listed. The relative hit rates should be weighed against other considerations, such as hardware complexity, in selecting a cache organization.

7.5 CACHE AND DMA

Cache coherency is an issue one must consider when placing a DMA controller in an 80386 system. Because the DMA controller has access to main memory, it can potentially introduce stale data. Stale data can be avoided in the following ways:

- Implementing bus watching (snooping). In this approach, the DMA controller writes to main memory, and the cache controller monitors DMA cycles and automatically invalidates any cache location altered by DMA.

Table 7-1. Cache Hit Rates

Cache Configuration			Hit Rate
Size	Associativity	Line Size	
1K	direct	4 bytes	41%
8K	direct	4 bytes	73%
16K	direct	4 bytes	81%
32K	direct	4 bytes	86%
32K	2-way	4 bytes	87%
32K	direct	8 bytes	91%
64K	direct	4 bytes	88%
64K	2-way	4 bytes	89%
64K	4-way	4 bytes	89%
64K	direct	8 bytes	92%
64K	2-way	8 bytes	93%
128K	direct	4 bytes	89%
128K	2-way	4 bytes	89%
128K	direct	8 bytes	93%

- Implementing a transparent cache, in which memory accesses from both the 80386 and the DMA controller are directed through the cache.
- Restrict DMA cycles to non-cacheable areas of memory.

The first method has a distinct advantage: since the DMA controller does not access the cache directly, the 80386 can read from the cache while the DMA controller is moving data to the main memory. Although bus watching is difficult to implement in a discrete cache design, the 82385 integrates this function and performs zero waitstate bus watching. The overall memory bandwidth is increased since the 80386 can access its cache at the same time as the DMA controller accesses main memory.

The second approach has the advantage of requiring minimal hardware, but has the disadvantage that the 80386 must be placed in HOLD during DMA transfers. The third approach is useful if a separate, dual-ported memory can be used as the non-cacheable memory, and the DMA device is tightly coupled to this memory. In all approaches, the cache should be made software transparent, so that DMA cycles do not require special actions by software to insure cache coherency.

7.6 CACHE EXAMPLE

The cache system example described in this section illustrates some of the decisions a cache designer must make. The requirements of a particular system may result in different choices than the ones made here. However, the issues presented in this section will arise in the process of designing any cache system.

7.6.1 Example Design

The cache system uses a direct-mapped cache. In previous generations of computers, it was often practical to build a 2-way or 4-way associative cache. SRAMs had low memory capacity, so many of them were needed to construct a cache of reasonable size. However, today's SRAMs are more dense, cost less, and take up less space. It is now more economical to increase cache efficiency by increasing cache size (SRAMs) rather than associativity (control logic and comparators).

The main memory is updated using buffered write-through. Implementing buffered write-through is slightly more complicated than unbuffered write-through, but it has the advantage that the processor can continue to run while the DRAM write is taking place. In contrast, write-back is significantly more complicated, but may be beneficial if main memory traffic must be kept to a minimum (as in multiprocessor systems, for example).

The line size is four bytes, which is most convenient for the 32-bit data bus of the 80386. An 8-byte line size would transfer twice as much data for every DRAM access, but would require a wider bus as well as more SRAMs, DRAMs, and transceivers. In such cases, one must weigh the additional cost against the additional performance.

The cache in this example stores both code and data, rather than only code. Code-only caches are easier to implement because there are no write accesses. They can be useful if data accesses are infrequent. In general, however, most programs make frequent data accesses. The code prefetch function of the 80386 makes the access time for code less critical to overall performance, since opcodes returned to the processor more quickly may only reside in the code queue longer.

7.6.2 Example Cache Memory Organization

The example cache is organized as shown in Figure 7-9. The cache holds 64 kbytes (16K locations of 4-byte blocks) of data and code and requires 16K 16-bit tag locations. The main memory can hold up to 2 Gbytes.

The 32-bit address from the 80386 is divided into the following three fields:

- Select — Bit A31 is used to select the cache/DRAM subsystem.
- Tag — Bits A30-A16 identify which DRAM location currently is associated with each cache location.
- Index — Bits A15-A2 identify one of the 16,384 doubleword locations in the cache.

Each doubleword location of the cache can be occupied by one of the 32,768 blocks from main memory (one block from each 64-kilobyte section).

The 80386 bits A31-A2 are interpreted as follows:

1. Select bit A31 is low during cache/DRAM cycles.
2. Index bits A15-A2 select the cache location.

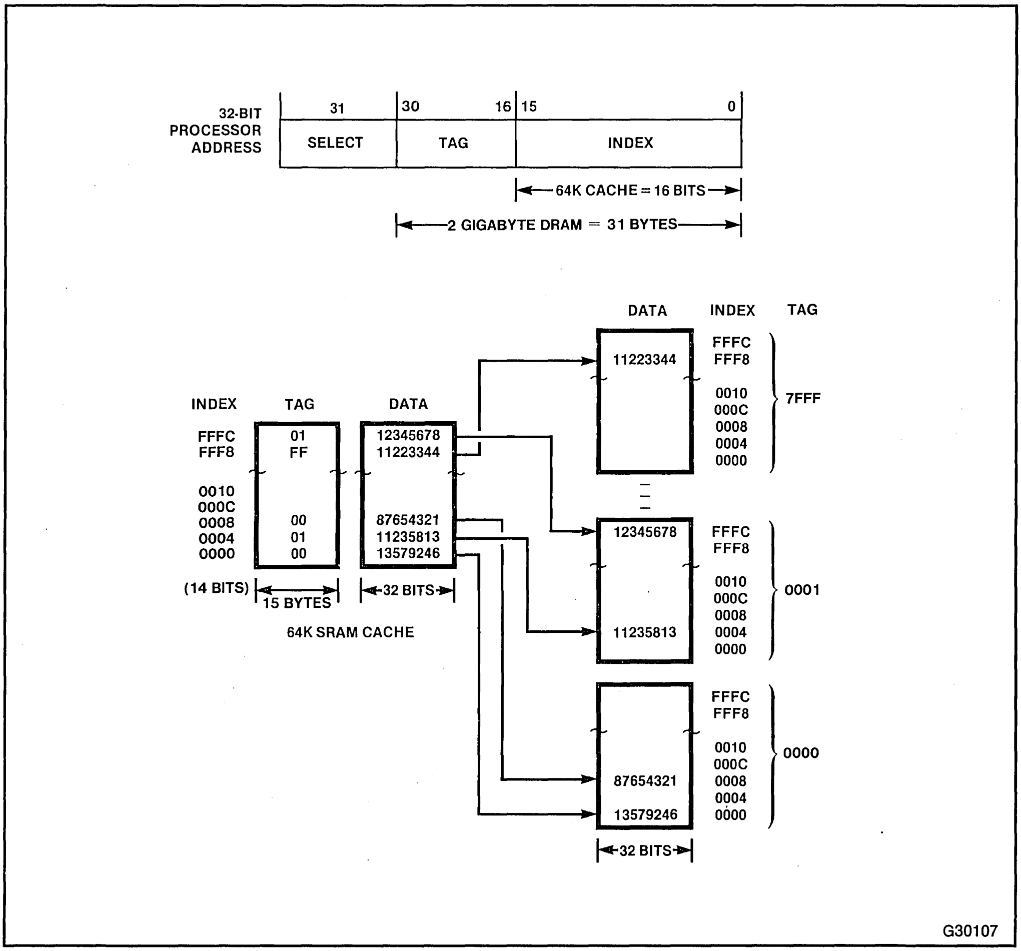

32-BIT PROCESSOR ADDRESS
31
30
16
15
0
SELECT
TAG
INDEX
64K CACHE = 16 BITS
2 GIGABYTE DRAM = 31 BYTES
DATA
INDEX
TAG
11223344
FFFC
FFF8
0010
000C
0008
0004
0000
7FFF
INDEX
TAG
DATA
FFFC
FFF8
0010
000C
0008
0004
0000
01
FF
00
01
00
12345678
11223344
87654321
11235813
13579246
(14 BITS)
15 BYTES
32 BITS
64K SRAM CACHE
12345678
11235813
0001
87654321
13579246
0000
32 BITS
G30107

Figure 7-9. Example of Cache Memory Organization

3. Tag bits A30-A16 are compared with the tag information stored in the cache to determine of the block in the cache is the block needed by the 80386.
   a. If the tag matches, the 80386 either reads the data in the cache or writes new data into the cache. In the case of a write, the data is also written to the main memory.
   b. If the tag does not match during a read cycle, a doubleword is read from main memory, stored in the cache, and simultaneously presented to the 80386. The new tag is also stored in the cache. During a write, if the tag does not match, the cache is not updated.

7.6.3 Example Cache Implementation

An implementation of a cache memory similar to the one described in this section is presented in application note AP-404, *80386 Cache Memory Example,* order number 231978.

I/O Interfacing

8

CHAPER 8
I/O INTERFACING

The 80386 supports 8-bit, 16-bit, and 32-bit I/O devices that can be mapped into either the 64-kilobyte I/O address space or the 4-gigabyte physical memory address space. This chapter presents the issues to consider when designing an interface to an I/O device. Mapping as well as timing considerations are described. Several examples illustrate the design concepts.

8.1 I/O MAPPING VERSUS MEMORY MAPPING

I/O mapping and memory mapping of I/O devices differ in the following respects:

- The address decoding required to generate chip selects for I/O-mapped devices is often simpler than that required for memory-mapped devices. I/O-mapped devices reside in the I/O space of the 80386 (64 kilobytes); memory-mapped devices reside in a much larger memory space (4 gigabytes) that makes use of more address lines.
- Memory-mapped devices can be accessed using any 80386 instruction, so I/O-to-memory, memory-to-I/O, and I/O-to-I/O transfers as well as compare and test operations can be coded efficiently. I/O-mapped devices can be accessed only through the IN, OUT, INS, and OUTS instructions. All I/O transfers are performed via the AL (8-bit), AX (16-bit), or EAX (32-bit) registers. The first 256 bytes of the I/O space are directly addressable. The entire 64-kilobyte I/O space is indirectly addressable through the DX register.
- Memory mapping offers more flexibility in protection than I/O mapping does. Memory-mapped devices are protected by memory management and protection features. A device can be inaccessible to a task, visible but protected, or fully accessible, depending on where the device is mapped in the memory space. Paging provides the same protection levels for individual 4-kilobyte pages and indicates whether a page has been written to. The I/O privilege level of the 80386 protects I/O-mapped devices by either preventing a task from accessing any I/O devices or by allowing a task to access all I/O devices. A virtual-8086-mode I/O permission bitmap can be used to select the privilege level for a combination of I/O bytes.

8.2 8-BIT, 16-BIT, AND 32-BIT I/O INTERFACES

The 80386 can operate with 8-bit, 16-bit, and 32-bit peripherals. The interface to a peripheral device depends not only upon data width, but also upon the signal requirements of the device and its location within the memory space or I/O space.

8.2.1 Address Decoding

Address decoding to generate chip selects must be performed whether I/O devices are I/O-mapped or memory-mapped. The decoding technique should be simple to minimize the amount of decoding logic.

One possible technique for decoding memory-mapped I/O addresses is to map the entire I/O space of the 80386 into a 64-kilobyte region of the memory space. The address decoding logic can be configured so that each I/O device responds to both a memory address and an I/O address. Such a configuration is compatible for both software that uses I/O instructions and software that assumes memory-mapped I/O.

Address decoding can be simplified by spacing the addresses of I/O devices so that some of the lower address lines can be omitted. For example, if devices are placed at every fourth address, the 80386 Byte Enable outputs (BE3#-BE0#) can be ignored for I/O accesses and each device can be connected directly to the same eight data lines. The 64-kilobyte I/O space is large enough to allow the necessary freedom in allocating addresses for individual devices.

Addresses can be assigned to I/O devices arbitrarily within the I/O space or memory space. Addresses for either I/O-mapped or memory-mapped devices should be selected to minimize the number of address lines needed.

8.2.2 8-Bit I/O

Eight-bit I/O devices can be connected to any of the four 8-bit sections of the data bus. Table 8-1 illustrates how the address assigned to a device determines which section of the data bus is used to transfer data to and from the device.

In a write cycle, if BE3# and/or BE2# is active but not BE1# or BE0#, the write data on the top half of the data bus is duplicated on the bottom half. If the addresses of two devices differ only in the values of BE3#-BE0# (the addresses lie within the same doubleword boundaries), BE3#-BE0# must be decoded to provide a chip select signal that prevents a write to one device from erroneously performing a write to the other. This chip select can be generated using an address decoder PAL device or TTL logic.

Another technique for interfacing with 8-bit peripherals is shown in Figure 8-1. The 32-bit data bus is multiplexed onto an 8-bit bus to accommodate byte-oriented DMA or block transfers to memory-mapped 8-bit I/O devices. The addresses assigned to devices connected to this interface can be closely spaced because only one 8-bit section of the data bus is enabled at a time.

Table 8-1. Data Lines for 8-Bit I/O Addresses

Address	4N + 3	4N + 2	4N + 1	4N
Byte	D31-D24	D23-D16	D15-D8	D7-D0
Word	D31-D16		D15-D0	
Doubleword	D31-D0			

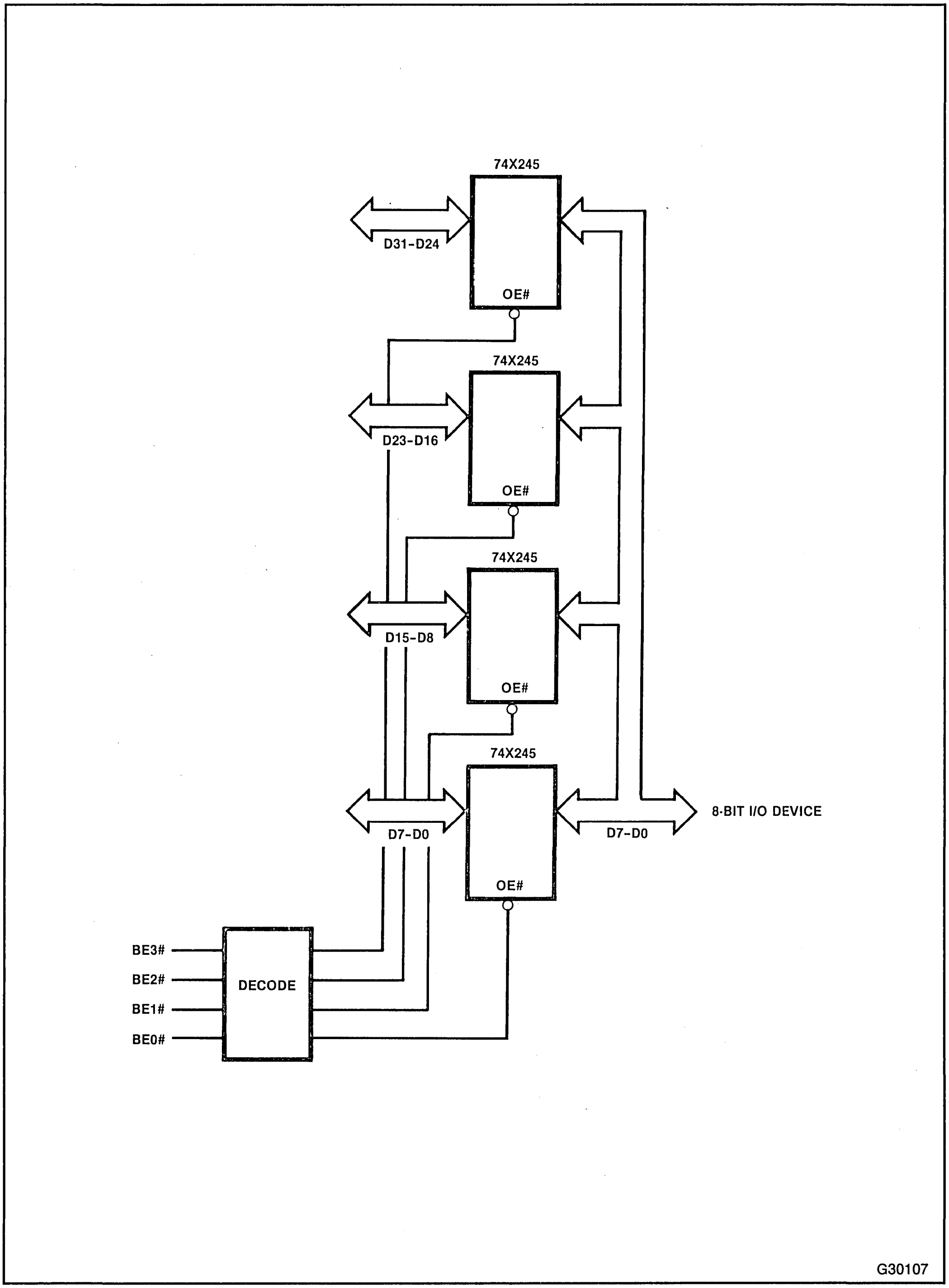

74X245
D31-D24
OE#
74X245
D23-D16
OE#
74X245
D15-D8
OE#
74X245
D7-D0
OE#
D7-D0
8-BIT I/O DEVICE
BE3#
BE2#
BE1#
BE0#
DECODE
G30107

Figure 8-1. 32-Bit to 8-Bit Bus Conversion

8.2.3 16-Bit I/O

To avoid extra bus cycles and to simplify device selection, 16-bit I/O devices should be assigned to even addresses. If I/O addresses are located on adjacent word boundaries, address decoding must generate the Bus Size 16 (BS16#) signal so that the 80386 performs a 16-bit bus cycle. If the addresses are located on every other word boundary (every doubleword address), BS16# is not needed.

8.2.4 32-Bit I/O

To avoid extra bus cycles and to simplify device selection, 32-bit devices should be assigned to addresses that are even multiples of four. Chip select for a 32-bit device should be conditioned by all byte enables (BE3#-BE0#) being active.

8.2.5 Linear Chip Selects

Systems with 14 or fewer I/O ports that reside only in the I/O space or that require more than one active select (at least one high active and one low active) can use linear chip selects to access I/O devices. Latched address lines A2-A15 connect directly to I/O device selects as shown in Figure 8-2.

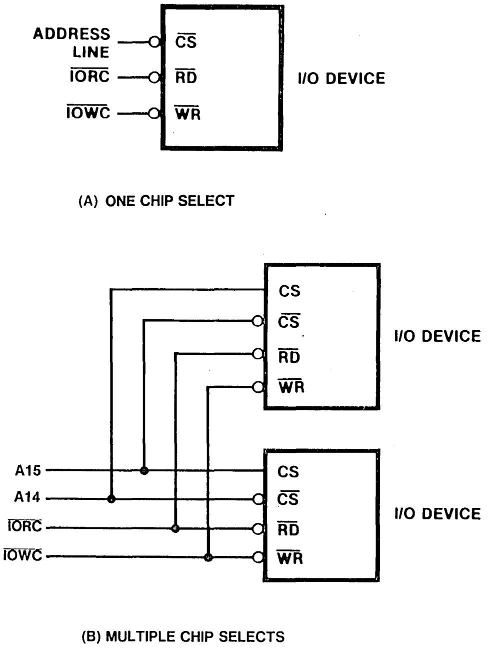
8.3 BASIC I/O INTERFACE

In a typical 80386 system design, a number of slave I/O devices can be controlled through the same local bus interface. Other I/O devices, particularly those capable of controlling the local bus, require more complex interfaces. This section presents a basic interface for slave peripherals.

The high performance and flexibility of the 80386 local bus interface plus the increased availability of programmable and semi-custom logic make it feasible to design custom bus control logic that meets the requirements of particular system.

The basic I/O interface shown in Figure 8-3 can be used to connect the 80386 to virtually all slave peripherals. The following list includes some common peripherals compatible with this interface:

8259A Programmable Interrupt Controller
8237 DMA Controller (remote mode)
82258 Advanced DMA Controller (remote mode)
8253, 8254 Programmable Interval Timer
8272 Floppy Disk Controller
82062, 82064 Fixed Disk Controller
8274 Multi-Protocol Serial Controller
8255 Programmable Peripheral Interface
8041, 8042 Universal Peripheral Interface


ADDRESS LINE
$\overline{IORC}$
$\overline{IOWC}$
$\overline{CS}$
$\overline{RD}$
$\overline{WR}$
I/O DEVICE
(A) ONE CHIP SELECT
CS
$\overline{CS}$
$\overline{RD}$
$\overline{WR}$
I/O DEVICE
A15
A14
$\overline{IORC}$
$\overline{IOWC}$
CS
$\overline{CS}$
$\overline{RD}$
$\overline{WR}$
I/O DEVICE
(B) MULTIPLE CHIP SELECTS

210760-97

Figure 8-2. Linear Chip Selects

The bus interface control logic presented here is identical to the one used in the basic memory interface described in Chapter 6. In most systems, the same control logic, address latches, and data buffers can be used to access both memory and I/O devices. The schematic of the interface is shown in Figure 8-4 and described in the following sections.

8.3.1 Address Latch

Latches maintain the address for the duration of the bus cycle. In this example, 74x373 latches are used.

The 74x373 Latch Enable (LE) input is controlled by the Address Latch Enable (ALE) signal from the bus control logic that goes active at the start of each bus cycle. The 74x373 Output Enable (OE#) is always active.

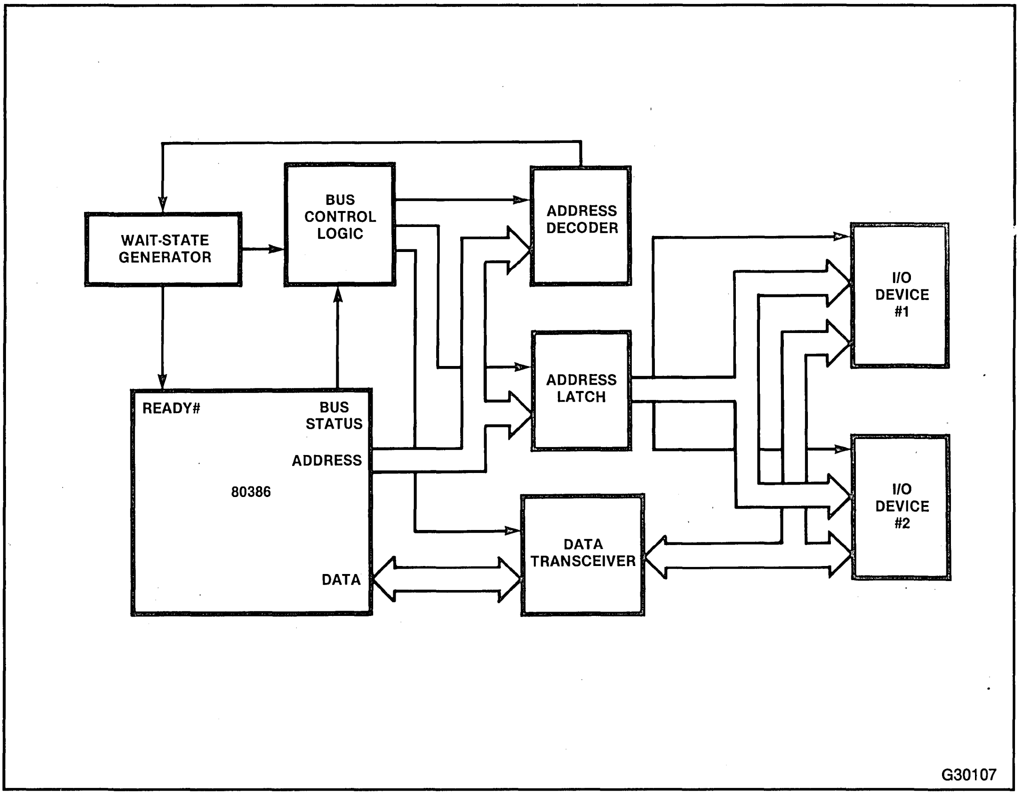

WAIT-STATE GENERATOR
BUS CONTROL LOGIC
ADDRESS DECODER
ADDRESS LATCH
DATA TRANSCEIVER
I/O DEVICE #1
I/O DEVICE #2
READY#
BUS STATUS
ADDRESS
80386
DATA
G30107

Figure 8-3. Basic I/O Interface Block Diagram

8.3.2 Address Decoder

In this example, the address decoder, which converts the 80386 address into chip-select signals, is located before the address latches. In general, the decoder may also be placed after the latches. If it is placed before the latches, the chip-select signal becomes valid as early as possible but must be latched along with the address. Therefore, the number of address latches needed is determined by the location of the address decoder as well as the number of address bits and chip-select signals required by the interface. The chip-select signals are routed to the bus control logic to set the correct number of wait states for the accessed device.

The decoder consists of two one-of-four decoders, one for memory address decoding and one for I/O address decoding. In general, the number of decoders needed depends on the memory mapping complexity. In this basic example, an output of the memory address decoder activates the I/O address decoder for I/O accesses. The addresses for the I/O devices are located so that only address bits A4 and A5 are needed to generate the correct chip-select signal.

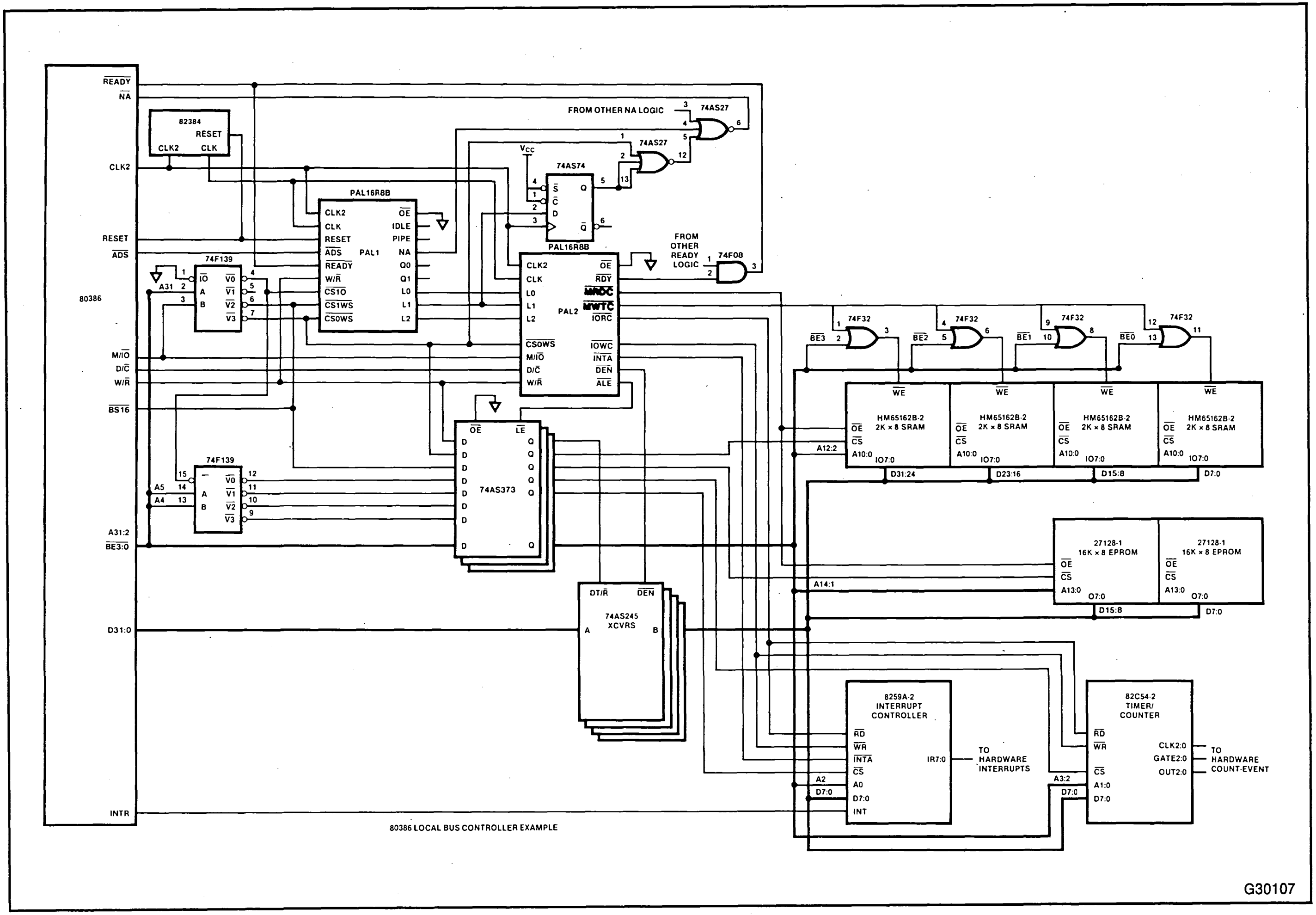

80386
READY
NA
CLK2
RESET
ADS
M/IO
D/C
W/R
BS16
A31:2
BE3:0
D31:0
INTR
82384
RESET
CLK2
CLK
74F139
PAL16R8B
PAL1
CLK2
CLK
RESET
ADS
READY
W/R
CS10
CS1WS
CS0WS
OE
IDLE
PIPE
NA
Q0
Q1
L0
L1
L2
74AS74
VCC
74AS27
FROM OTHER NA LOGIC
FROM OTHER READY LOGIC
74F08
PAL2
RDY
MRDC
MWTC
IORC
IOWC
INTA
DEN
ALE
74AS373
OE
LE
74AS245 XCVRS
DT/R
DEN
74F32
BE3
BE2
BE1
BE0
WE
HM65162B-2 2K × 8 SRAM
CS
A10:0
IO7:0
A12:2
D31:24
D23:16
D15:8
D7:0
27128-1 16K × 8 EPROM
A13:0
O7:0
A14:1
8259A-2 INTERRUPT CONTROLLER
RD
WR
INTA
CS
A0
D7:0
INT
IR7:0
TO HARDWARE INTERRUPTS
A2
82C54-2 TIMER/COUNTER
A1:0
CLK2:0
GATE2:0
OUT2:0
TO HARDWARE COUNT-EVENT
A3:2
80386 LOCAL BUS CONTROLLER EXAMPLE
G30107

Figure 8-4. Basic I/O Interface Circuit

8.3.3 Data Transceiver

Standard 8-bit transceivers (74x245, in this example) provide isolation and additional drive capability for the 80386 data bus. Transceivers are necessary to prevent the contention on the data bus that occurs if some devices are slow to remove read data from the data bus after a read cycle. If a write cycle follows a read cycle, the 80386 may drive the data bus before a slow device has removed its outputs from the bus, potentially causing bus contention problems. Transceivers can be omitted only if the data float time of the device is short enough and the load on the 80386 data pins meets device specifications.

A bus interface must include enough transceivers to accommodate the device with the most inputs and outputs on the data bus. If the widest device has 16 data bits and if the I/O addresses are located so that all devices are connected only to the lower half of the data bus, only two 8-bit transceivers are needed.

The 74x245 transceiver is controlled through two input signals:

- Data Transmit/Receive (DT/R#)—When high, this input enables the transceiver for a write cycle. When low, it enables the transceiver for a read cycle. This signal is just a latched version of the 80386 W/R# output.
- Data Enable (DEN#)—When low, this input enables the transceiver outputs. This signal is generated by the bus control logic.

Note that in a system using the 82380, the data transceivers must be disabled whenever the 80386 performs a read access to one of the internal registers of the 82380. Otherwise, both the 82380 and the data transceivers will be driving the local bus which causes data contention. This can be avoided by decoding the 82380 address space in the bus controller logic. Together with the bus cycle definition signals (W/R#, M/IO#), the data transceivers can be disabled by deactivating the DEN# signal.

8.3.4 Bus Control Logic

The bus control logic for the basic I/O interface is the same as the logic for the memory interface described in Section 6.2. The bus controller decodes the 80386 status outputs (W/R#, M/IO#, and D/C#) and activates a command signal for the type of bus cycle requested. The command signal corresponds to the bus cycle types (described in Chapter 3) as follows:

- Memory data read and memory code read cycles generate the Memory Read Command (MRDC#) output. MRDC# commands the selected memory device to output data.
- I/O read cycles generate the I/O Read Command (IORC#) output. IORC# commands the selected I/O device to output data.
- Memory write cycles generate the Memory Write Command (MWTC#) output. MWTC# commands the selected memory device to receive the data on the data bus.
- I/O write cycles generate the I/O Write Command (IOWC#) output. IOWC# commands the selected memory device to receive the data on the data bus.

Interrupt-acknowledge cycles generate the Interrupt Acknowledge (INTA#) output, which is returned to the 8259A Interrupt Controller.

The bus controller also controls the READY# input to the 80386 that ends each bus cycle. The PAL-2 bus control PAL counts wait states and returns READY# after the number of wait states required by the accessed device. The design of this portion of the bus controller depends on the requirements of the system; relatively simple systems need less wait-state logic than more complex systems. The basic interface described here uses a PAL device to generate READY#; other designs may use counters and/or shift registers.

If several I/O devices reside on the local bus, READY# logic can be simplified by combining into a single input the chip selects for devices that require the same number of wait states. The CSIO# input of PAL-1 generates the same number of wait states for all I/O accesses. Adding wait states to some devices to make the wait-state requirements of several devices the same does not significantly impact performance. If the response of the device is already slow (four wait states, for example), the additional wait state amounts to a relatively small delay. Typically, I/O devices are used infrequently enough that the access time is not critical.

8.4 TIMING ANALYSIS FOR I/O OPERATIONS

In this section, timing requirements for devices that use the basic I/O interface are discussed. The values of the various device specifications are examples only; **for correct timing analysis, always refer to the latest data sheet for the particular device.**

Timing for 80386 I/O cycles is identical to memory cycle timing in most respects; in particular, timing depends on the design of the interface. The worst-case timing values are calculated by assuming the maximum delay in the address latches, chip select logic, and command signals, and the longest propagation delay through the data transceivers (if used). These calculations yield the minimum possible access time for an I/O access for comparison with the access time of a particular I/O device. Wait states must be added to the basic worst-case values until read and write cycle times exceed minimum device access times.

The timing requirement for the address decoder dictates that the logic be combinational (not latched or registered) with a propagation delay less than the maximum delay calculated below.

The CS1WS signal requires a maximum decoder delay of 38.75 nanoseconds:

$$\begin{array}{lll}(3 \text{ x CLK2 period}) & - \text{ 80386 Addr Valid} & - \text{ PAL setup} \\ (3 \text{ x } 31.25) & - 38 & - 15\end{array}$$

$$\begin{array}{l}= 40.75 \text{ nanoseconds} \\ \quad (\text{CLK2} = 32 \text{ MHz})\end{array}$$

The CS0WS signal must be slightly faster in order to activate NA#:

(3 x CLK2 period) − 80386 Addr Valid − (2 x OR prop. delay)
− 80386 NA# setup
(3 x 31.25) − 38 − (2 x 6)
− 10

= 33.75 nanoseconds
(CLK2 = 32 MHz)

The timings of the other signals can be calculated from the waveforms in Figure 8-5. In the following example, the timings for I/O accesses are calculated for CLK2 = 32 MHz and B-series PALs. All times are in nanoseconds.

tAR: Address stable before Read (IORC# fall)
tAW: Address stable before Write (IOWC# fall)

(5 x CLK2 period) −PAL RegOut Max −Latch Enable Max
+ PAL RegOut Min
(5 x 31.25) − 12 − 11.5
+ 0

= 132.75 nanoseconds

tRR: Read (IORC#) pulse width

(9 x CLK2 period) − PAL RegOut Max + PAL RegOut Min
(9 x 31.25) − 12 + 0
= 269.25 nanoseconds

tWW: Write (IOWC#) pulse width

(10 x CLK2 period) − PAL RegOut Max + PAL RegOut Min
(10 x 31.25) − 12 + 0

= 300.5 nanoseconds

tRA: Address hold after Read (IORC# rise)

(6 x CLK2 period) − PAL RegOut Max + PAL RegOut Min
+ Latch Enable Min
(6 x 31.25) − 12 + 0
+ 5

= 180.5 nanoseconds

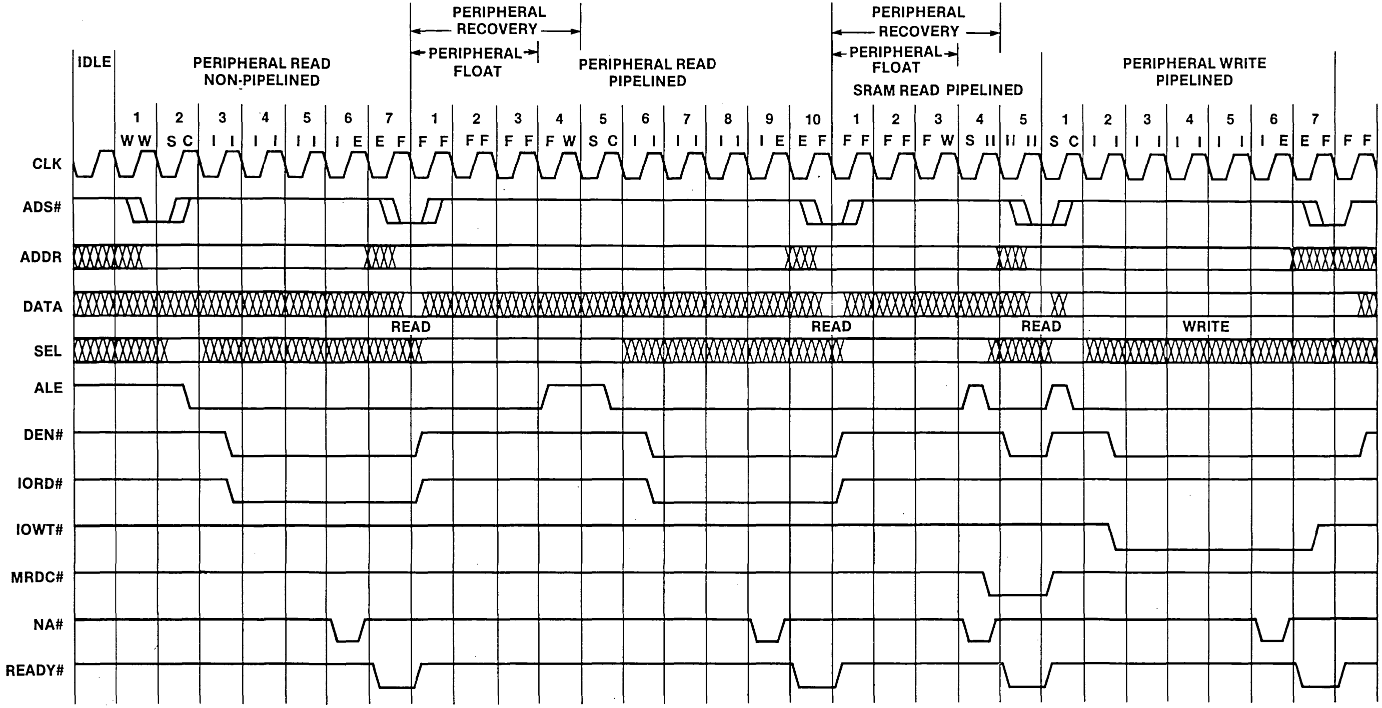

IDLE
PERIPHERAL READ NON-PIPELINED
PERIPHERAL RECOVERY
PERIPHERAL FLOAT
PERIPHERAL READ PIPELINED
PERIPHERAL RECOVERY
PERIPHERAL FLOAT
SRAM READ PIPELINED
PERIPHERAL WRITE PIPELINED
1 2 3 4 5 6 7 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 1 2 3 4 5 6 7
W W S C I I I I I I I E E F F F F F F F F W S C I I I I I I I E E F F F F F F W S II II II S C I I I I I I I I I E E F F F
CLK
ADS#
ADDR
DATA
READ
READ
READ
WRITE
SEL
ALE
DEN#
IORD#
IOWT#
MRDC#
NA#
READY#
G30107

Figure 8-5. Basic I/O Timing Diagram

tWA: Address hold after Write (IOWC# rise)

(7 x CLK2 period) − PAL RegOut Max + PAL RegOut Min
+ Latch Enable Min
(7 x 31.25) − 12 + 0
+ 5

= 211.75 nanoseconds

tAD: Data delay from Address

(12 x CLK2 period) − PAL RegOut Max + Latch Enable Max
− xcvr. prop Min − 80386 Data Setup Min
(12 x 31.25) − 12 − 11.5
− 6

− 10
= 335.5 nanoseconds

tRD: Data delay from Read (IORC#)

(9 x CLK2 period) − PAL RegOut Max − xcvr. prop Min
− 80386 Data Setup Min
(9 x 31.25) − 12 − 6
− 10

= 253.25 nanoseconds

tDF: read (IORC# rise) to Data Float

(8 x CLK2 period) − PAL RegOut Max + PAL RegOut Min
+ xcvr. Enable Min
(8 x 31.25) − 12 + 0
+ 3

= 241 nanoseconds

tDW: Data setup before write (IOWC# rise)

(10 x CLK2 period) − PAL RegOut Max − xcvr. Enable Max
+ PAL RegOut Min
(10 x 31.25) − 12 − 11
+ 0

= 289.5 nanoseconds

tWD: Data hold after write (IOWC# rise)

(2 x CLK2 period) − PAL RegOut + PAL RegOut
+ xcvr. Disable
(2 x 31.25) − 12 + 0
+ 2

= 52.5 nanoseconds

tRV: command recovery time

(11 x CLK2 period) − PAL RegOut Max + PAL RegOut Min
(11 x 31.25) − 12 + 0

= 331.75 nanoseconds

Many peripherals require a minimum recovery time between back-to-back accesses. This recovery time is usually provided in software by a series of NOP instructions. A JMP to the next instruction also provides a delay because it flushes the 80386 Prefetch Queue; this method has a more predictable execution time than the NOP method.

In 80386 systems, the instructions that provide recovery time are executed more quickly than in earlier systems. For software compatibility with earlier microprocessor generations, hardware must guarantee the recovery time. However, the circuitry to delay bus commands selectively for the specific instance of back-to-back accesses to a particular device is typically more complex than the frequency of such accesses justifies. Therefore, the preferred solution is to delay all I/O cycles by the minimum recovery time. Because most I/O accesses are relatively infrequent, performance is not degraded.

The I/O access timings of the basic interface are compatible with all of the currently available Intel peripherals. (In some cases, the high-speed versions of these peripherals are required). Table 8-2 compares several peripheral timings with the timings provided by bus controller.

Table 8-2. Timings for Peripherals using Basic I/O Interface

	tAR	tAW	tRR	tWW	tRA	tWA	tAD	tRD	tDF	tDW	tDWF	tWD	tRV
bus cntrlr	39	39	269	300	180	211	335	253	241	289	—	52	331
8259-2	0	0	160	190	0	0	200	120	85	160	—	0	190
8254-2	30	0	95	95	0	0	185	85	65	85	—	0	165
82C54-2	0	0	95	95	0	0	185	85	65	85	—	0	165
82C55-2	0	0	150	100	0	20	—	120	75	100	—	30	300
8272	0	0	250	250	0	0	—	200	100	150	—	5	—
82064	0	0	200	200	0	0	—	70	200	160	—	0	300
8041	0	0	250	250	0	0	225	225	100	150	—	0	2500
8042	0	0	160	160	0	0	130	130	85	130	—	0	1120
8251	0	0	250	250	0	0	—	200	100	150	—	20	—
8273-4	0	0	250	250	0	0	300	200	100	150	—	0	1920
8274	0	0	250	250	0	0	200	200	120	150	—	0	300
8291	0	0	140	170	0	0	250	100	60	130	—	0	—
8292	0	0	250	250	0	0	225	225	100	150	—	0	—

Only two peripherals do not meet the bus controller specifications: the 8041 and 8042 UPIs (Universal Peripheral Interface 8-bit Microcomputers). These intelligent peripherals meet all but the command recovery specification, so they can be used if this delay is implemented in software.

8.5 BASIC I/O EXAMPLES

In this section, two examples of the interface to slave I/O devices are presented. Typically, several of these devices exist on the 80386 local bus. The basic I/O interface presented above is used for both examples.

8.5.1 8274 Serial Controller

The 8274 Multi-Protocol Serial Controller (MPSC) is designed to interface high-speed serial communications lines using a variety of communications protocols, including asynchronous, IBM bisynchronous, and HDLC/SDLC protocols. The 8274 contains two independent full-duplex channels and can serve as a high-performance replacement for two 8251A Universal Synchronous/Asynchronous Receiver Transmitters (USARTs).

Figure 8-6 shows connections from the basic I/O interface through which the 80386 communicates with the 8274. The 8274 is accessed as a sequence of four 8-bit I/O addresses (I/O-mapped or memory-mapped). The Serial I/O (SERIO#) signal is a chip select generated by address decoding logic. RD# and WR# signals are provided by the bus control logic. DB7-DB0 inputs connect to the lower eight outputs of the data transceiver (D7-D0).

The 8274 A1 and A0 inputs are used for channel selection and data or command selection. These inputs are connected to two address lines that are determined by the 8274 addresses. The addresses must be chosen so that the A1 and A0 inputs receive the correct signals for addressing the 8274.

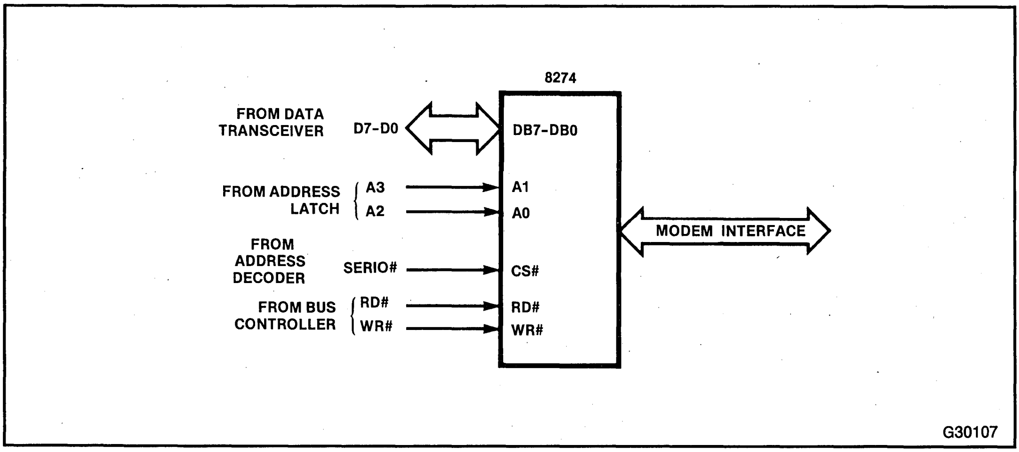


Figure 8-6. 8274 Interface

The 8274 requires a minimum recovery time between back-to-back accesses that is provided for in the basic I/O interface hardware.

8.5.2 82380 Programmable Interrupt Controller

The 82380 Programmable Interrupt Controller (PIC) can be used in interrupt-driven microcomputer systems. It has 15 external and 5 internal interrupt requests. Each of the external requests can be cascaded with an additional 82C59A Interrupt Controller to accommodate up to 120 external interrupt sources.

The 82380 PIC handles interrupt priority resolution and returns a preprogrammed service routine vector to the 80386 during an interrupt acknowledge cycle. It consists of three 82C59A compatible banks. The 82380 Data Sheet contains detailed information on the 82380 PIC.

When an interrupt occurs, the 82380 PIC activates its Interrupt (INT) output, which is connected to the Interrupt Request (INTR) input of the 80386. the 80386 automatically executes two back-to-back interrupt acknowledge cycles, as described in Chapter 3. The 82380 PIC will automatically terminate the interrupt acknowledge cycles by driving its READYO# signal. Each acknowledge cycle will be extended by five wait states. Also, four idle states are inserted by the 80386 between two consecutive interrupt acknowledge cycles.

8.5.2.1 CASCADED INTERRUPT CONTROLLERS TO THE 82380 PIC

Each of the external requests of the 82380 PIC can be cascaded with one 'slave' 82C59A Interrupt Controller. With all its external requests cascaded, the 82380 PIC can handle up to 120 external requests.

For a cascaded interrupt request, the 82380 PIC will output an 8-bit cascade address on the data bus during the first interrupt acknowledge cycle. A simple circuit can latch the 8-bit address and encode it to drive the CAS signals (CAS2# — CAS0#) of the slave controllers. During the second interrupt acknowledge cycle, the 82380 will not drive the data bus; instead, the selected slave controller will put the interrupt vector on the data bus for the 80386.

Chapter 9 describes the interface to slave controllers that reside on a MULTIBUS I system bus.

8.5.3 8259A Interrupt Controller

The 8259A Programmable Interrupt Controller is designed for use in interrupt-driven microcomputer systems. A single 8259A can process up to eight interrupts. Multiple 8259As can be cascaded to accommodate up to 64 interrupts. A technique to handle more than 64 interrupts is discussed at the end of this section.

The 8259A handles interrupt priority resolution and returns a preprogrammed service routine vector to the 80386 during an interrupt-acknowledge cycle. Intel *Application Note AP-59* contains detailed information on configurations of the 8259A.

8.5.3.1 SINGLE INTERRUPT CONTROLLER

Figure 8-7 shows the connections from the basic I/O interface used for the 80386 and a single 8259A. Programmable Interrupt Controller (PIC#) is a chip-select signal from the address decoding logic. INTA#, RD#, and WR# are generated by the bus control logic. BD7-BD0 are connected to the lower eight outputs of the data transceiver. The A2 bit, connected to the 8259A A0 input, is used by the 80386 to distinguish between the two interrupt acknowledge cycles; 8259A register addresses must therefore be located at two consecutive doubleword boundaries.

When an interrupt occurs, the 8259A activates its Interrupt (INT) output, which is connected to the Interrupt Request (INTR) input of the 80386. The 80386 automatically executes two back-to-back interrupt-acknowledge cycles, as described in Chapter 3. The 8259A timing requirements are as follows:

- Each interrupt-acknowledge cycle must be extended by at least one wait state. Wait-state generator logic must provide for this extension.
- Four idle bus cycles must be inserted between the two interrupt-acknowledge cycles. The 80386 automatically inserts these idle cycles.

8.5.3.2 CASCADED INTERRUPT CONTROLLERS

Several 8259As can be cascaded to handle up to 64 interrupt requests. In a cascaded configuration, one 8259A is designated as the master controller; it receives input from the other

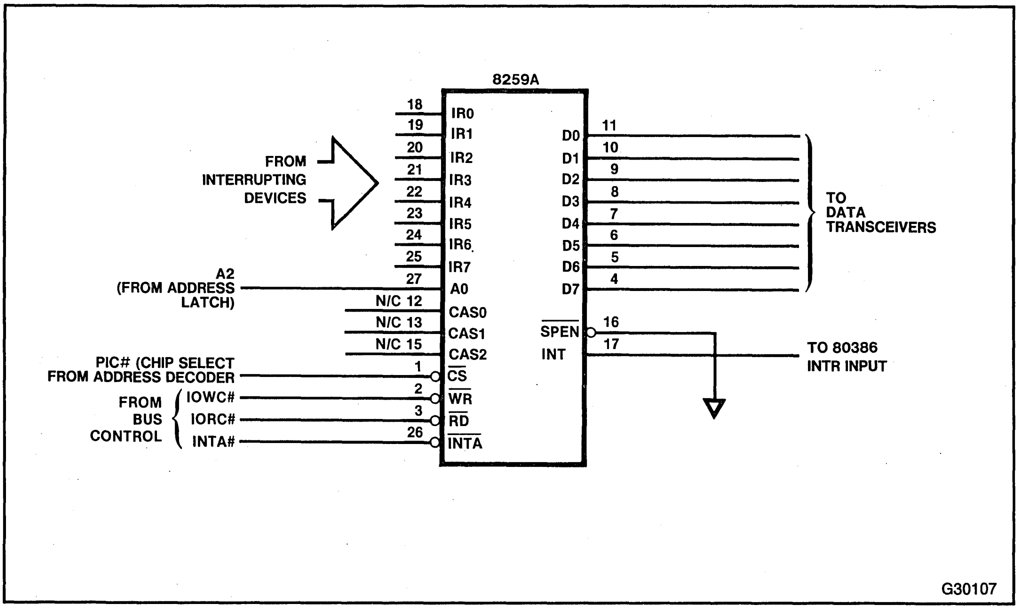

8259A
FROM INTERRUPTING DEVICES
18 IR0
19 IR1
20 IR2
21 IR3
22 IR4
23 IR5
24 IR6
25 IR7
A2 (FROM ADDRESS LATCH)
27 A0
N/C 12 CAS0
N/C 13 CAS1
N/C 15 CAS2
PIC# (CHIP SELECT FROM ADDRESS DECODER
1 CS
FROM BUS CONTROL
IOWC# 2 WR
IORC# 3 RD
INTA# 26 INTA
D0 11
D1 10
D2 9
D3 8
D4 7
D5 6
D6 5
D7 4
TO DATA TRANSCEIVERS
SPEN 16
INT 17
TO 80386 INTR INPUT
G30107

Figure 8-7. Single 8259A Interface

8259As, called slave controllers. The interface between the 80386 and multiple cascaded 8259As is an extension of the single-8259A interface with the following additions:

- The cascade address outputs (CAS2#-CAS0#) are output to provide address and chip-select signals for the slave controllers.
- The interrupt request lines (IR7-IR0) of the master controller are connected to the INT outputs of the slave controllers.

Each slave controller resolves priority between up to eight interrupt requests and transmits a single interrupt request to the master controller. The master controller, in turn, resolves interrupt priority between up to eight slave controllers and transmits a single interrupt request to the 80386.

The timing of the interface is basically the same as that of a single 8259A. During the first interrupt-acknowledge cycle, all the 8259As freeze the states of their interrupt request inputs. The master controller outputs the cascade address to select the slave controller that is generating the request with the highest priority. During the second interrupt-acknowledge cycle, the selected slave controller outputs an interrupt vector to the 80386.

Chapter 9 describes the interface to slave controllers that reside on a MULTIBUS I system bus.

8.5.3.3 HANDLING MORE THAN 64 INTERRUPTS

If an 80386 system requires more than 64 interrupt request lines, a third level of 8259As in polled mode can be added to the configuration described above. When a third-level controller receives an interrupt request, it drives one of the interrupt request inputs to a slave controller active. The slave controller sends an interrupt request to the master controller, and the master controller interrupts the 80386. The slave controller then returns a service-routine vector to the 80386. The service routine must include commands to poll the third level of interrupt controllers to determine the source of the interrupt request.

The only additional hardware required to handle more than 64 interrupts are the extra 8259As and the chip-select logic. For maximum performance, third-level interrupt controllers should be used only for noncritical, infrequently used interrupts.

8.6 80286-COMPATIBLE BUS CYCLES

Some devices (the 82258, for example) require an 80286-compatible interface in order to communicate with the 80386. An 80286-compatible interface must generate the following signals:

- Address bits A1 and A0, and Byte High Enable (BHE#) from the 80386 BE3#-BE0# outputs
- Bus cycle definition signals S0# and S1# from the 80386 M/IO#, W/R#, and D/C# outputs

- Address Latch Enable (ALE#), Device Enable (DEN), and Data Transmit/Receive (DT/R#) signals
- I/O Read Command (IORC#) and I/O Write Command (IOWC#) signals for I/O cycles
- Memory Read Command (MRDC#) and Memory Write Command (MWTC#) signals for memory cycles
- Interrupt Acknowledge (INTA#) signal for interrupt-acknowledge cycles

In the following example, the interface is constructed using the 80286-compatible bus controller (82288) and bus arbiter (82289). The 82289, along with the bus arbiters of other processing subsystems, coordinates control of the bus between the 80386 and other bus masters. The 82288 provides the control signals to perform bus cycles. Communication between the 80386 and these devices is accomplished through PALs that are programmed to perform all necessary signal translation and generation. Latching and buffering of the data and address buses is performed by TTL logic.

Figure 8-8 shows a block diagram of the interface, which consists of the following parts:

- A0/A1 generator—Generates the lower address bits from 80386 BE0#-BE3# outputs
- Address decoder—Determines the device the 80386 will access
- Address latches—Connect directly to 80386 address pins A19-A2 and the outputs of the A0/A1 generator
- Data transceivers—Connect directly to 80386 data pins D15-D0
- S0#/S1# generator—Translates 80386 outputs into the S0# and S1# signals
- Wait-state generator—Controls the length of the 80386 bus cycle through the READY# signal
- 82288 Bus Controller—Generates the bus command signals
- 82289 Bus Arbiter—Arbitrates contention for bus control between the 80386 and other bus masters

8.6.1 A0/A1 Generator

The A0, A1, and BHE# signals are 80286-compatible. These signals are generated from the 80386 byte enables (BE0#-BE3#) as shown in Table 8-3. The truth table can be implemented with the logic shown in Figure 8-9.

8.6.2 S0#/S1# Generator

S0# and S1# are 80286-compatible status signals that must be provided for the 82288 and 82289. The S0#/S1# logic in Figure 8-10 generates these signals from 80386 status outputs (D/C#, M/IO#, and W/R#) and wait-state generator outputs. WS1 and WS2 are wait-state generator outputs that correspond to the first and second wait states of the 80386 bus cycle. These signals ensure that S0# and S1# are valid for two CLK cycles.

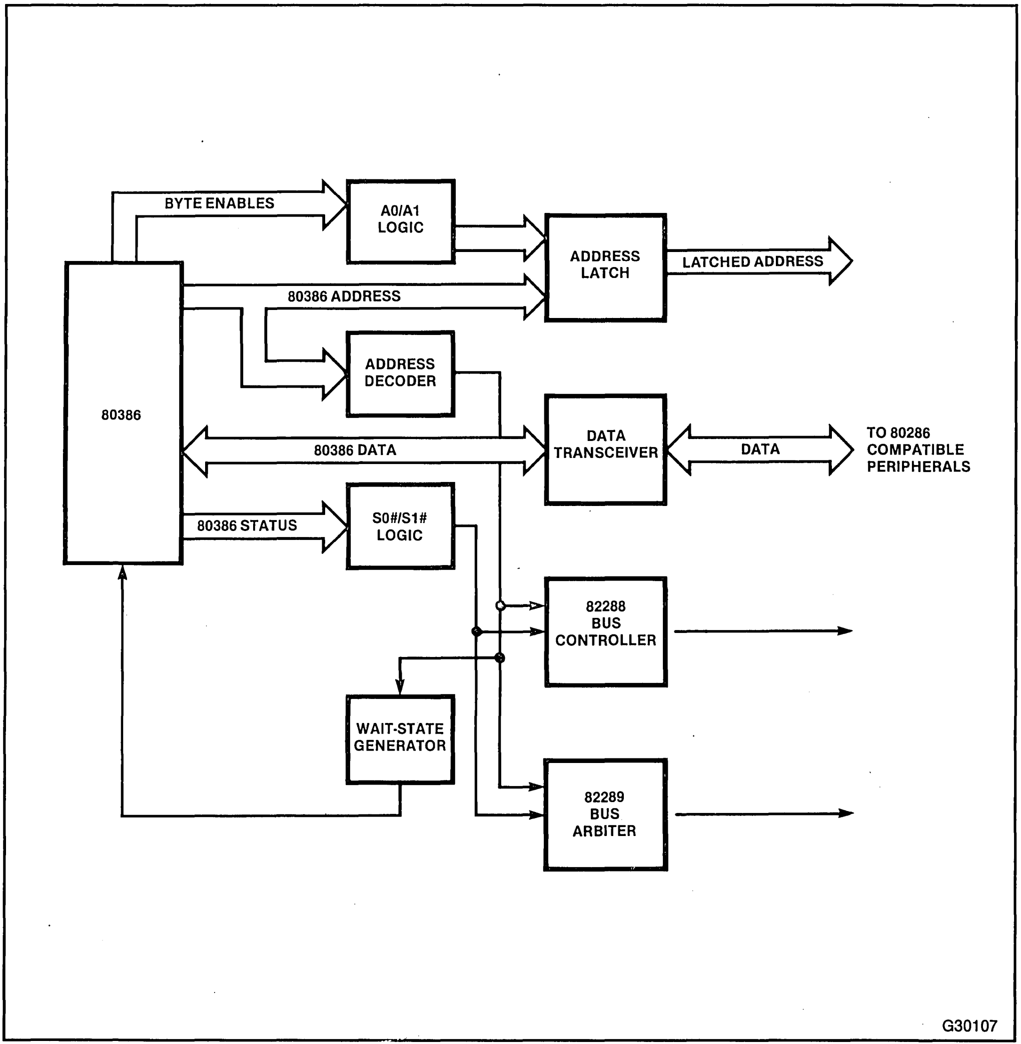

BYTE ENABLES
A0/A1 LOGIC
ADDRESS LATCH
LATCHED ADDRESS
80386 ADDRESS
ADDRESS DECODER
80386
DATA TRANSCEIVER
80386 DATA
DATA
TO 80286 COMPATIBLE PERIPHERALS
80386 STATUS
S0#/S1# LOGIC
82288 BUS CONTROLLER
WAIT-STATE GENERATOR
82289 BUS ARBITER
G30107

Figure 8-8. 80286-Compatible Interface

8.6.3 Wait-State Generator

The wait-state generator PAL shown in Figure 8-11 controls the READY# input of the 80386. For local bus cycles, the wait-state generator produces signal outputs that correspond to each wait state of the 80386 bus cycle, and the PAL READY# output uses these signals to set READY# active after the required number of wait states. Two of the wait-state signals, WS1 and WS2, are also used to generate S0# and S1#, as described above.

The PCLK signal, which necessary for producing 80286-compatible wait states, is generated by dividing the CLK signal from the 82384 by two.

Table 8-3. A0, A1, and BHE# Truth Table

80386 Signals				16-Bit Bus Signals			Comments
BE3#	**BE2#**	**BE1#**	**BE0#**	**A1**	**BHE#**	**BLE# (A0)**	
H*	H*	H*	H*	x	x	x	x—no active bytes
H	H	H	L	L	H	L	
H	H	L	H	L	L	H	
H	H	L	L	L	L	L	
H	L	H	H	H	H	L	
H*	L*	H*	L*	x	x	x	x—not contiguous bytes
H	L	L	H	L	L	H	
H	L	L	L	L	L	L	
L	H	H	H	H	L	H	
L*	H*	H*	L*	x	x	x	x—not contiguous bytes
L*	H*	L*	H*	x	x	x	x—not contiguous bytes
L*	H*	L*	L*	x	x	x	x—not contiguous bytes
L	L	H	H	H	L	L	
L*	L*	H*	L*	x	x	x	x—not contiguous bytes
L	L	L	H	L	L	H	
L	L	L	L	L	L	L	

BLE# asserted when D0–D7 of 16-bit bus is active.
BHE# asserted when D8–D15 of 16-bit bus is active.
A1 low for all even words; A1 high for all odd words.

Key:
x = don't care
H = high voltage level
L = low voltage level
* = a non-occurring pattern of Byte Enables; either none are asserted, or the pattern has Byte Enables asserted for non-contiguous bytes

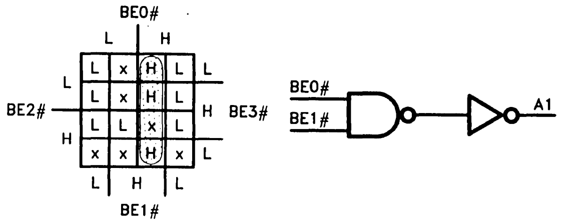
To meet the READY# input hold time requirement (25 nanoseconds) for the 82288 Bus Controller, the READY# signal must be two CLK cycles long. Therefore, two PAL equations are required to generate READY#. The first equation generates the Ready Pulse (RDYPLSE) output. RDYPLSE is fed into the READY# equation to extend READY# by an additional CLK cycle. These signals are gated by PCLK.

RDYPLSE := ARDY * PCLK

/READY := ARDY * PCLK + RDYPLSE

8.6.4 Bus Controller and Bus Arbiter

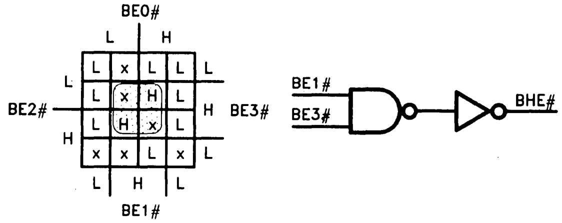
Connections for the 82288 and 82289 are shown in Figure 8-12. The 82288 MB input is tied low so that the 82288 operates in local-bus mode. Both the 82288 and the 82289 are selected by an output of the address decoder that selects 80286-compatible cycles. The AEN# signal from the 82289 enables the 82288 outputs.


BE0#
L
H
BE2#
L
H
BE3#
L
H
L
BE1#
L
H
L
BE0#
BE1#
A1

L	x	H	L
L	x	H	L
L	L	x	L
x	x	H	x

K-map for A1 signal (same as Figure 5-3)

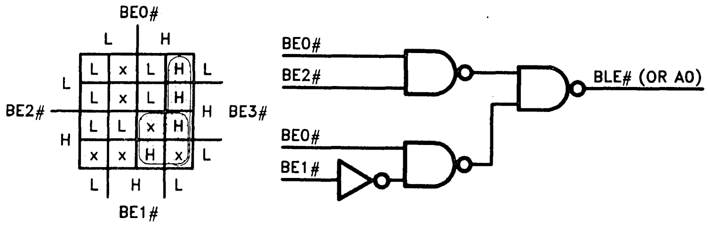
231630-8


BE0#
L
H
BE2#
L
H
BE3#
L
H
L
BE1#
L
H
L
BE1#
BE3#
BHE#

L	x	L	L
L	x	H	L
L	H	x	L
x	x	L	x

K-map for 16-bit BHE# signal

231630-9


BE0#
L
H
BE2#
L
H
BE3#
L
H
L
BE1#
L
H
L
BE0#
BE2#
BE0#
BE1#
BLE# (OR A0)

L	x	L	H
L	x	L	H
L	L	x	H
x	x	H	x

K-map for 16-bit BLE# signal (same as A0 signal in Figure 5-3)

231630-10

Figure 8-9. A0, A1, and BHE# Logic

8.6.5 82380 Integrated System Peripheral

The 82380 is designed for easy interface to the 80386 processor. It consists of a set of signals to interface directly to the 80386 local bus. Figure 8-13 depicts a typical system configuration with the 80386 processor.

When the 82380 switches from slave to master mode (or vice versa), there are idle states in which the ADS# signal is left floating. In order not to confuse the internal state machine of the 82380, a 10 K ohm pull-up resistor should be used on the ADS# signal to ensure that this line is inactive during these idle states.

As described in section 8.3.3, the data transceivers should be disabled during any read access to the 82380 in the slave mode to prevent bus contention.

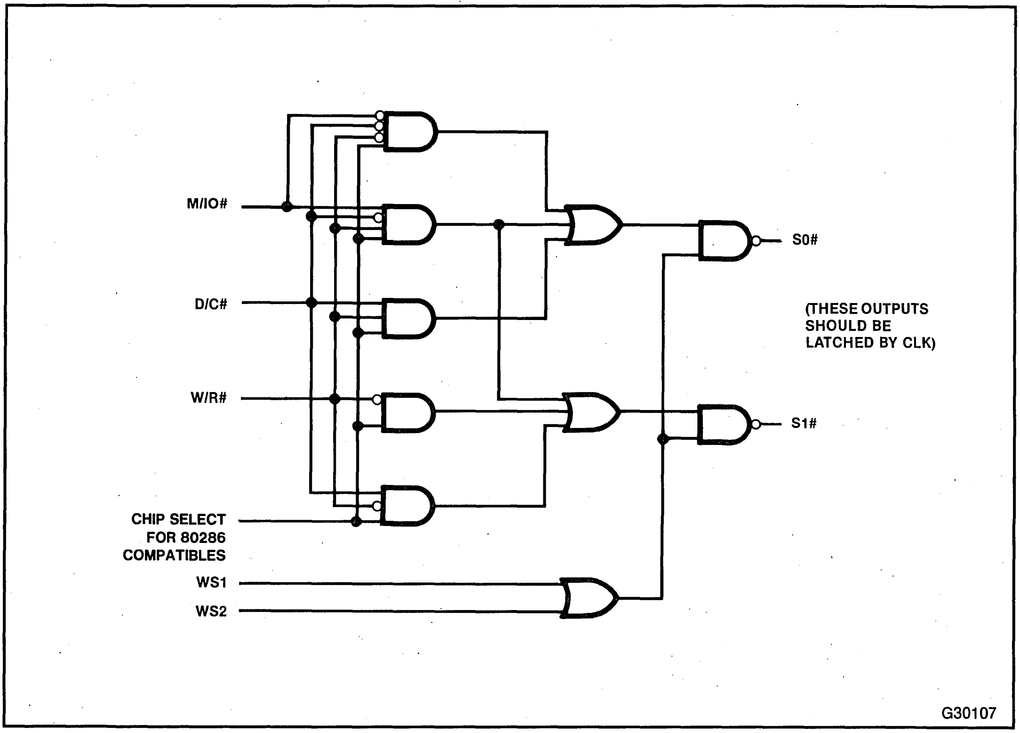

M/IO#
D/C#
W/R#
CHIP SELECT FOR 80286 COMPATIBLES
WS1
WS2
S0#
S1#
(THESE OUTPUTS SHOULD BE LATCHED BY CLK)
G30107

Figure 8-10. S0#/S1# Generator Logic

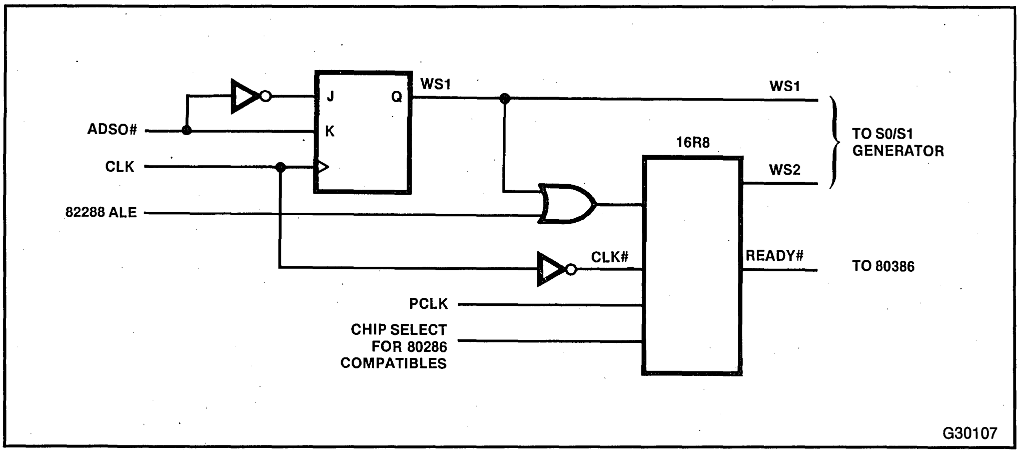

ADSO#
CLK
82288 ALE
J
K
Q
WS1
WS1
16R8
WS2
TO S0/S1 GENERATOR
CLK#
READY#
TO 80386
PCLK
CHIP SELECT FOR 80286 COMPATIBLES
G30107

Figure 8-11. Wait-State Generator Logic

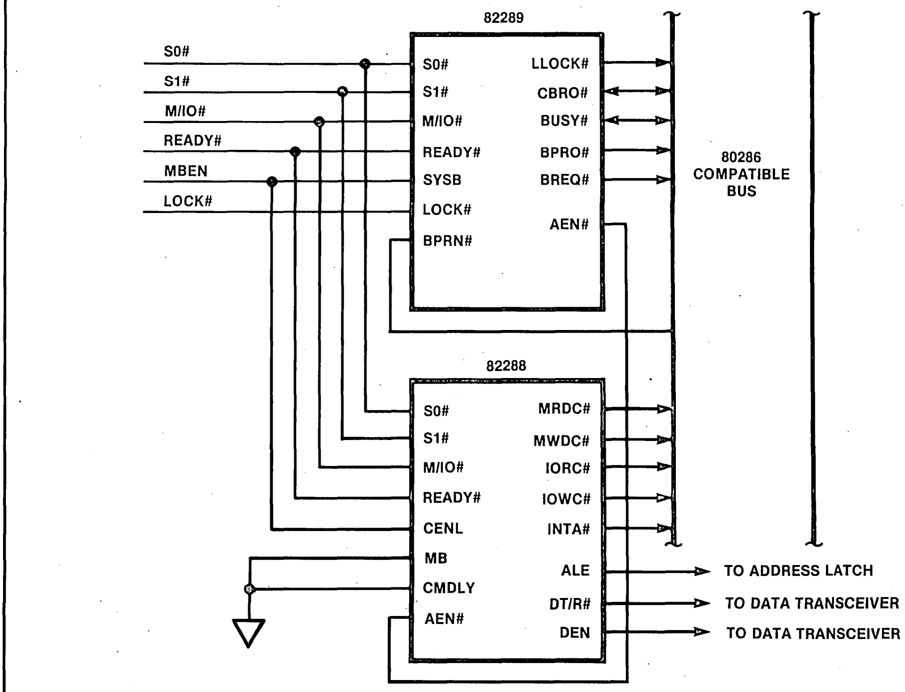

82289
S0#
S1#
M/IO#
READY#
MBEN
LOCK#
S0#
S1#
M/IO#
READY#
SYSB
LOCK#
BPRN#
LLOCK#
CBRO#
BUSY#
BPRO#
BREQ#
AEN#
80286 COMPATIBLE BUS
82288
S0#
S1#
M/IO#
READY#
CENL
MB
CMDLY
AEN#
MRDC#
MWDC#
IORC#
IOWC#
INTA#
ALE
DT/R#
DEN
TO ADDRESS LATCH
TO DATA TRANSCEIVER
TO DATA TRANSCEIVER
G30107

Figure 8-12. 82288 and 82289 Connections

8.6.6 82258 ADMA Controller

The 82258 Advanced Direct Memory Access (ADMA) controller performs DMA transfers between main memory and an I/O device, typically a magnetic disk or communications channel, without intervention from the 80386. Shifting the I/O processing function from the 80386 to the 82258 improves overall system performance because the 80386 doesn't have to switch context for every transfer.

The 82258 operates from the CLK output of the 82384 and provides the following advanced features that are not available in previous generations of DMA controllers:

- Command chaining to perform multiple commands sequentially
- Data chaining to scatter data to separate memory locations and to gather data from separate locations (this feature is useful for demand paging)

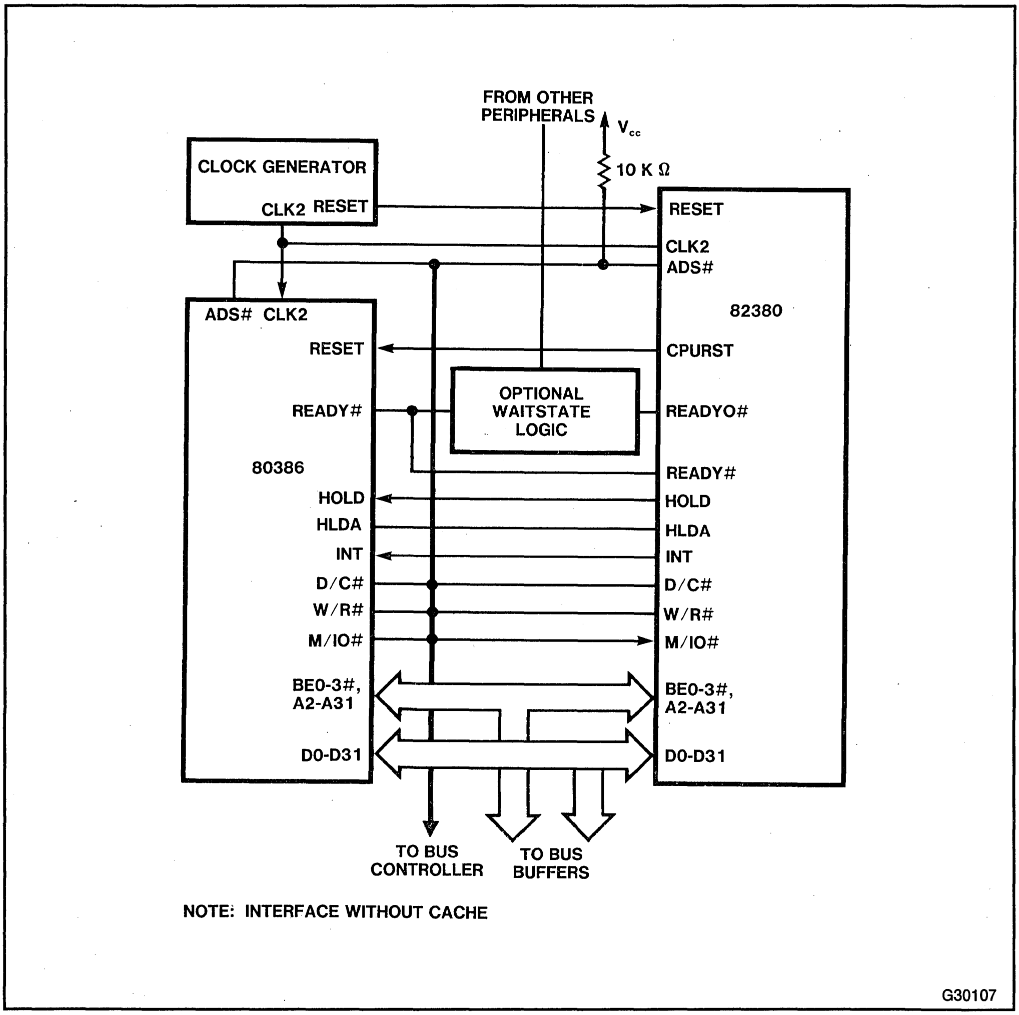

FROM OTHER PERIPHERALS
V_{cc}
10 K Ω
CLOCK GENERATOR
CLK2 RESET
ADS# CLK2
RESET
READY#
80386
HOLD
HLDA
INT
D/C#
W/R#
M/IO#
BE0-3#, A2-A31
D0-D31
OPTIONAL WAITSTATE LOGIC
RESET
CLK2
ADS#
82380
CPURST
READYO#
READY#
HOLD
HLDA
INT
D/C#
W/R#
M/IO#
BE0-3#, A2-A31
D0-D31
TO BUS CONTROLLER
TO BUS BUFFERS
NOTE: INTERFACE WITHOUT CACHE
G30107

Figure 8-13. 80386/82380 Interface

- Auto-assembly and disassembly to convert from 16-bit memory to 8-bit I/O or vice versa
- Compare, translate, and verify functions
- The option to use one of the four high-speed channels as a lower-speed, multiplexed channel. The 82258 gains up to 32 more channels through multiplexing

The 82258, paired with the 82288 Bus Controller, is capable of being a bus master on a common local bus and/or system bus with the 80386 and its bus controller. The 82258 requires both a local bus interface to act as bus master and an 80386 interface to communicate directly with the 80386.

8.6.6.1 82258 AS BUS MASTER

The 82258 operating mode determines the type of bus interface it generates. In the 80286 mode, the 82258 generates the signals for direct interface with an 80286. In this example, the 82258 is set to 80286 mode because the 80386 resembles the 80286 more than the 80186 or 8086. The 82258 is initialized to 80286 mode by holding its A23 pin high with a pullup resistor during reset.

The 82258 interface must include logic for sharing control of the local bus with the 80386. The HOLD and Hold Acknowledge (HLDA) pins on both the 80386 and the 82258 facilitate the transfer of bus control between the 80386 and the 82258.

Figure 8-14 shows the logic to transfer bus control. When the 82258 needs bus access, it asserts its HOLD request signal, which is synchronized to CLK2 to meet the synchronous setup and hold times of the 80386. The resulting Processor Hold (PHOLD) signal sets the 80386 HOLD input active.

When the 80386 recognizes the HOLD request, it completes the current bus cycle, then places all its outputs except HLDA in the high impedance state and asserts HLDA. External

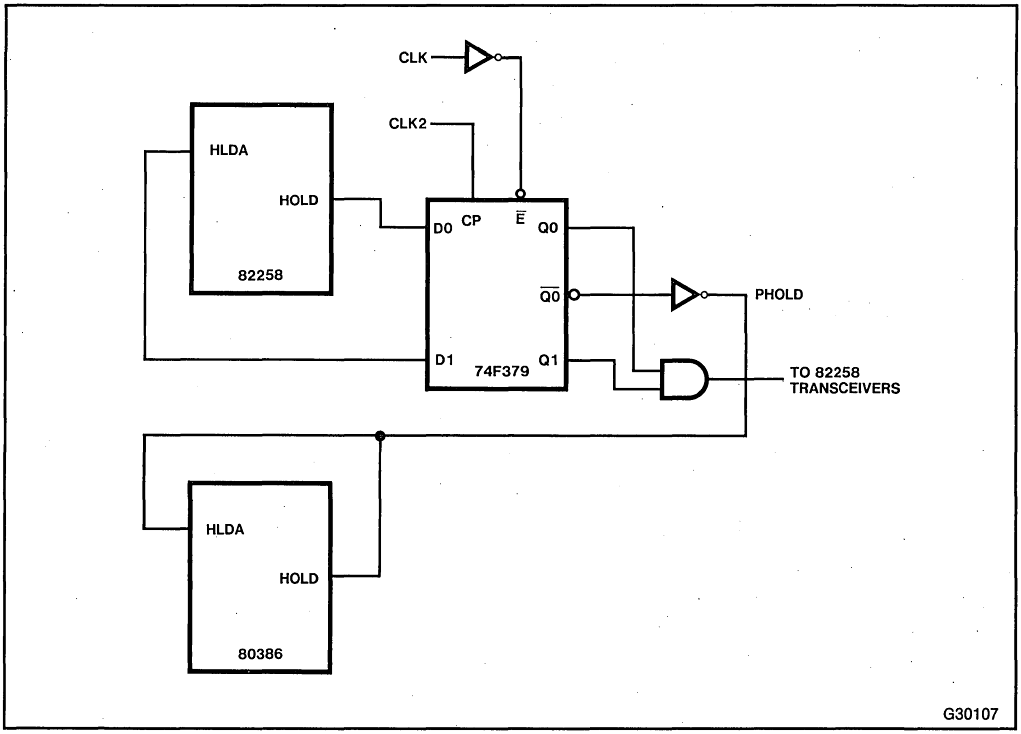


Figure 8-14. HOLD and HLDA Logic for 80386-82258 Interface

logic must use HLDA to disable the output buffers of the 80386 bus controller, data bus, and address bus, and enable the output buffers of the 82258 and its bus controller, data bus, and address bus.

The wait-state generator must be started with the ALE output of the 82288 Bus Controller. Although the 82258 can share wait-state generator logic with the 80386, the logic must be modified to support longer wait states for 82258 cycles. The 82258 divides its CLK input by two internally, so that its wait state requires two CLK cycles rather than one. In addition, the READY# output must meet the 82258 input hold time requirement of 25 nanoseconds.

When the 82258 completes its operation, its HOLD request goes inactive (low), disabling the 82258 output buffers and re-enabling the 80386 and its output buffer.

8.6.6.2 82258 AS PERIPHERAL

Although most of the communication between the 80386 and the 82258 is achieved indirectly through memory, the 80386 occasionally performs bus cycles to the 82258. For example, the 80386 performs a direct access to set the general mode register during initialization. The access occurs asynchronously over the slave mode interface of the 82258 that is shown in Figure 8-15. The interface consists of the following pins:

- Chip Select (CS#)—enables the slave mode interface
- Control (RD#, WR#)—indicates bus cycle type (asynchronous interface)
- Register Address (A7-A0)—selects an 82258 internal register
- Data (D15-D0)—transfers data to and from the 82258

With the 82258, asynchronous cycles must be used to allow time for the conversion of 80386 status signals to 80286-compatible inputs. The S0# and S1# inputs of the 82258 must therefore be inactive (high) during slave mode operations. Asynchronous cycles require an address hold time after a read command or write command, so the address outputs from the 80386

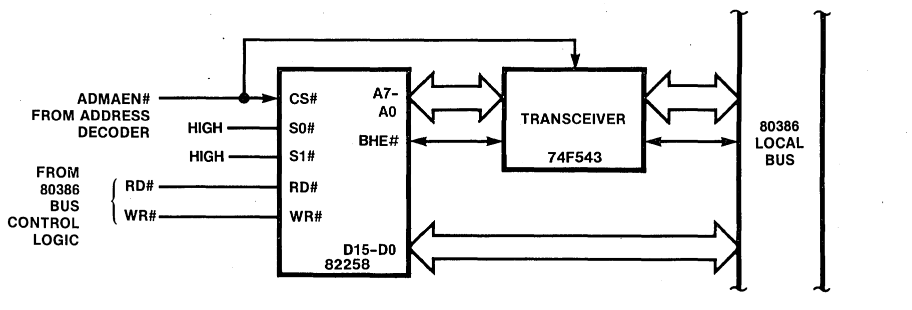


Figure 8-15. 82258 Slave Mode Interface

(A7-A2, and decoded A1, A0, and BHE#) that connect to corresponding inputs of the 82258 must be latched. Because A7-A0 and BHE# become 82258 outputs after initialization, registered transceivers (74F543) serve as these latches.

ADMA Enable (ADMAEN#) is a chip select generated by address decoding logic during 80386 accesses to the I/O addresses designated for the 82258. The RD# and WR# command signals from the 80386 bus control logic must be delayed to provide the proper setup time from chip select to command inputs. This delay can be generated using wait-state generator signals.

8.6.7 82586 LAN Coprocessor

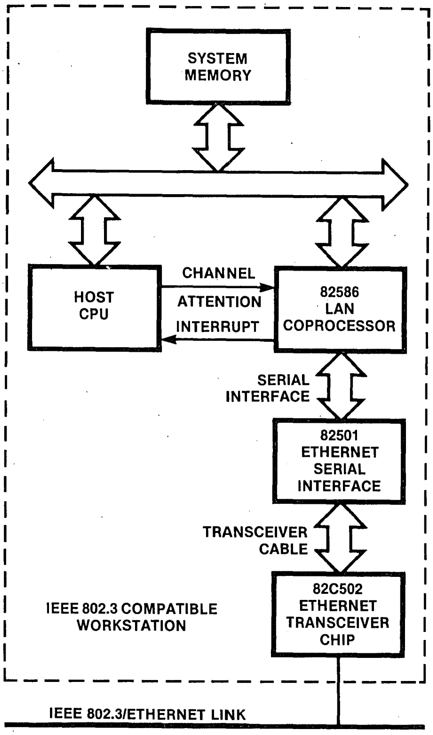
The 82586 is an intelligent, high-performance communications controller designed to perform most tasks required for controlling access to a local area network (LAN). In most applications, the 82586 is the communication manager for a station connected to a LAN. Such a station usually includes a host CPU, shared memory, a Serial Interface Unit, a transceiver, and a LAN link (see Figure 8-16). The 82586 performs all functions associated with data transfer between the shared memory and the LAN link, including:

- Framing
- Link management
- Address filtering
- Error detection
- Data encoding
- Network management
- Direct memory access (DMA)
- Buffer chaining
- High-level (user) command interpretation

The 82586 has two interfaces: a bus interface to the 80386 local bus and a network interface to the Serial Interface Unit. The bus interface is described here. For detailed information on using the 82586, refer to the *Local Area Networking (LAN) Component User's Manual.*

The 82586, which is a master on the 80386 local bus, communicates directly with the 80386 through the Channel Attention (CA) and interrupt (INT) signals. There are several ways to design an interface between the 82586 and the 80386. In general, higher performance interfaces (requiring less servicing time from the 80386) are more expensive. Four types of interfaces are described in this section:

- Dedicated CPU
- Decoupled dual-port memory
- Coupled dual-port memory
- Shared bus


SYSTEM MEMORY
HOST CPU
CHANNEL ATTENTION
INTERRUPT
82586 LAN COPROCESSOR
SERIAL INTERFACE
82501 ETHERNET SERIAL INTERFACE
TRANSCEIVER CABLE
82C502 ETHERNET TRANSCEIVER CHIP
IEEE 802.3 COMPATIBLE WORKSTATION
IEEE 802.3/ETHERNET LINK

G30107

Figure 8-16. LAN Station

8.6.7.1 DEDICATED CPU

Dedicating a CPU to control the 82586 results in a high-performance, high-cost interface. The CPU, typically an 80186, an 80188, or a microcontroller, executes the data link layer (a functional division) of software and sometimes the network, transport, and session layers as well. (For definitions of these layers, see the *Local Area Networking (LAN) Components User's Manual*). The dedicated CPU relieves the 80386 of these layers and provides a high-level, message-oriented interface that can be treated in software as a standard I/O device. In hardware, the interface is mapped into a dual-port memory.

8.6.7.2 DECOUPLED DUAL-PORT MEMORY

A decoupled dual-port memory interface, shown in Figure 8-17, contains two sections of memory:

- 80386 core memory—typically DRAM that provides executable memory space for the operating system
- 82586 communication channel memory—typically dual-ported SRAM that contains the commands and buffers of the 82586

Only the dual-ported SRAM is shared; the 82586 cannot access the 80386 core memory. The 80386 and 82586 operate in parallel except when both require access to the SRAM. In this instance, one processor must wait while the other completes its access. At all other times, the two devices are decoupled.

This interface requires at least one level of data copying to move data between the 80386 core memory and the 82586 communication channel memory. However, usually the data must be copied to separate the frame header information.

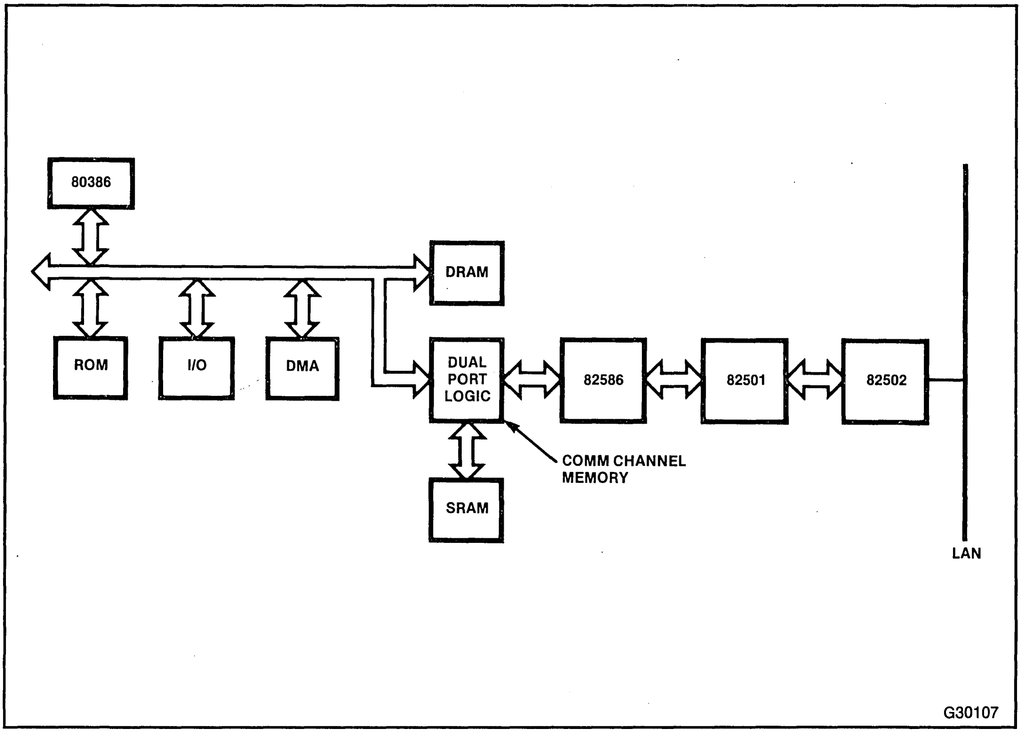


Figure 8-17. Decoupled Dual-Port Memory Interface

8.6.7.3 COUPLED DUAL-PORT MEMORY

In a coupled dual-port memory interface, the 80386 and the 82586 share a common memory space as illustrated in Figure 8-18. The 82586, with 24 address bits, can address up to 16 megabytes of memory. If the 80386 memory is larger than 16 megabytes, some memory is inaccessible to the 82586; this memory must be taken into account in the system design.

The advantage of coupled dual-port memory is that the 82586 can perform DMA transfers directly into the operating system memory. In this case, other logic must remove the frame header information from the data prior to the DMA transfer. Through the buffer-chaining feature of the 82586, the header information can be directed to a separate buffer, as long as the minimum buffer size requirements are met.

8.6.7.4 Shared Bus

In a shared bus interface (Figure 8-19), the 80386 and the 82586 share a common address and data bus. The HOLD and HLDA signals provide bus arbitration. When one device enters the hold state in response to the HOLD input, the other device can access the bus.

The shared bus interface is probably the simplest and least expensive interface. However, the performance of the 80386 may drop tremendously because the 80386 must wait for the 82586 to complete its bus operation before it can access the bus. This wait can be several hundred CLK cycles.

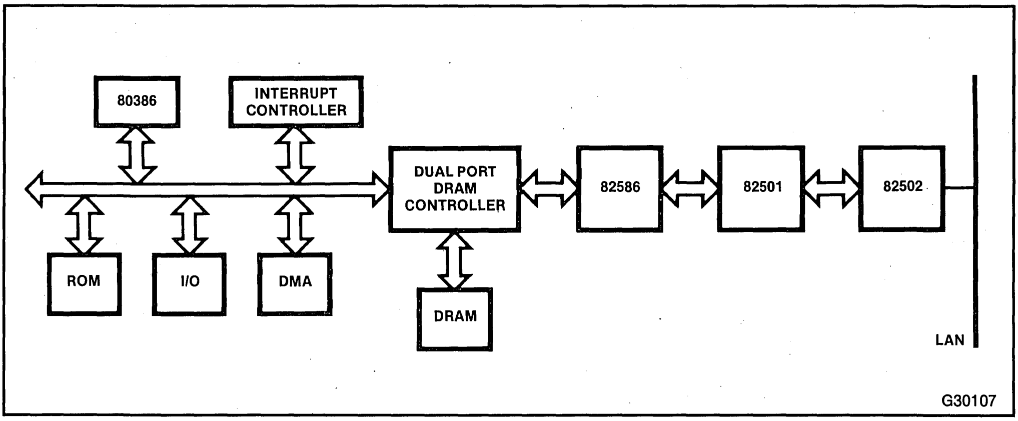


Figure 8-18. Coupled Dual-Port Memory Interface

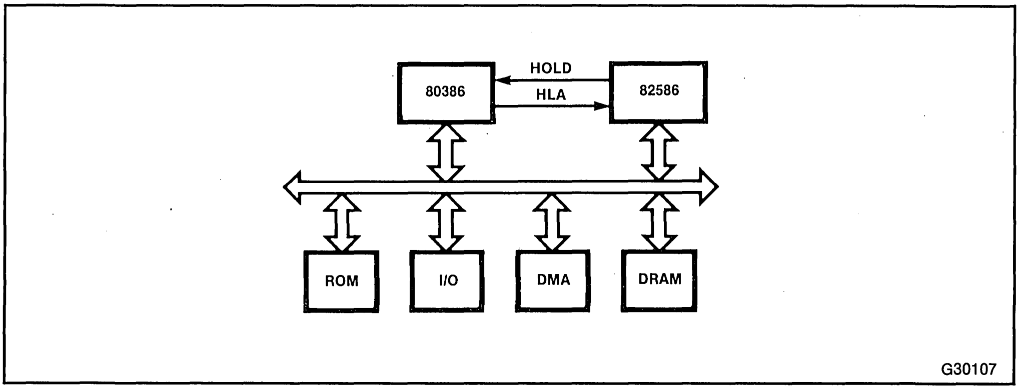

80386
HOLD
HLA
82586
ROM
I/O
DMA
DRAM
G30107

Figure 8-19. Shared Bus Interface

MULTIBUS® I and 80386

9

CHAPTER 9
MULTIBUS® I AND 80386

Previous chapters have presented single-bus systems in which a single 80386 connects to memory, I/O, and coprocessors. This chapter introduces the system bus, which connects several single-bus systems to create a powerful multiprocessing system. Two examples of multiprocessing system buses are the Intel MULTIBUS I, discussed in this chapter, and the Intel MULTIBUS II, discussed in Chapter 10.

A system bus connects several processing subsystems (each of which can include a local bus and private resources) and the resources that are shared between the processing subsystems. Because all the processing subsystems perform operations simultaneously on their respective local buses, such a multiprocessing system results in a significant increase in throughput over a single-bus system.

Another advantage of using a system bus is that the system can be expanded modularly. The system bus establishes the standard interface through which additional processing subsystems communicate with one another. Through this interface, components from different vendors can be integrated.

A central concern of any multiprocessing system is dividing resources between the system bus and the individual local buses; that is, determining which resources to share between all processors and which to keep for only one processor's use. These choices affect system reliability, integrity, throughput, and performance. The deciding factors are often the requirements of the particular target system.

Because local resources are isolated from failures occurring in other parts of the system, they enhance the overall reliability of the system. Also, because the processor does not have to contend with other processors for access to its local resources, bus cycles are performed quickly. However, local resources add to the system cost because each resource must be duplicated for each subsystem that requires it.

Resources used by more than one processing subsystem but not used frequently by any subsystem should be placed on the system bus. The system can minimize the idle time of such resources. However, this advantage must be weighed against the disadvantage of increased access time when more than one processor must use a system resource.

9.1 MULTIBUS® I (IEEE 796)

The Intel MULTIBUS I (IEEE 796 Standard) is a proven, industry-standard, 16-bit multiprocessing system bus. A wide variety of MULTIBUS I compatible I/O subsystems, memory boards, general purpose processing boards, and dedicated function boards are available from Intel. Designers who choose the MULTIBUS I protocols in their system bus have a ready supply of system components available for use in their products.

MULTIBUS I protocols are described in detail in the Intel *MULTIBUS® I Architecture Reference Book.*

One method of constructing an interface between the 80386 and the MULTIBUS I is to generate all MULTIBUS I signals using only TTL and PAL devices. A simpler method is to use the 80286-compatible interface described in Chapter 8. The latter option is described in the MULTIBUS I interface example in this chapter.

9.2 MULTIBUS® I INTERFACE EXAMPLE

The MULTIBUS I interface presented in the following example consists of the 80286-compatible 82289 Bus Arbiter and 82288 Bus Controller. The 82289, along with the bus arbiters of other processing subsystems, coordinates control of the MULTIBUS I; the 82288 provides the control signals to perform MULTIBUS I accesses. Communication between the 80386 and these devices is accomplished through PALs that are programmed to perform all necessary signal translation and generation. Latching and buffering of the data and address buses is performed by TTL logic.

Figure 9-1 shows a block diagram of the interface, which consists of the following parts:

- A0/A1 generator—Generates the lower address bits from 80386 BE0#-BE3# outputs
- Address decoder—Determines whether the bus cycle requires a MULTIBUS I access
- MULTIBUS I address latches—Connect directly to 80386 address pins A23-A2 and the outputs of the A0/A1 generator
- MULTIBUS I data latch/transceivers—Connect directly to 80386 data pins D15-D0
- S0#/S1# generator—Translates 80386 outputs into the S0# and S1# signals
- Wait-state generator—Controls the length of the 80386 bus cycle through the READY# signal
- 82288 Bus Controller—Generates the MULTIBUS I command signals
- 82289 Bus Arbiter—Arbitrates contention for bus control between the 80386 and other MULTIBUS I masters

These elements of the 80286-compatible interface are described in detail in Chapter 8. The block diagram in Figure 9-1 does not include the 80386 local bus interface and local resources. In a complete system, some logic (for example, the address decoder) is common to both MULTIBUS I and local bus interfaces. The following discussion includes only the logic necessary for the MULTIBUS I interface.

9.2.1 Address Latches and Data Transceivers

MULTIBUS I allows up to 24 address lines and 16 data lines. In this example, the MULTIBUS addresses are located in a 256-kilobyte range between F00000H and F3FFFFH, so that all 24 address lines are used. The 16 data lines correspond to the lower half of the 80386 data bus.

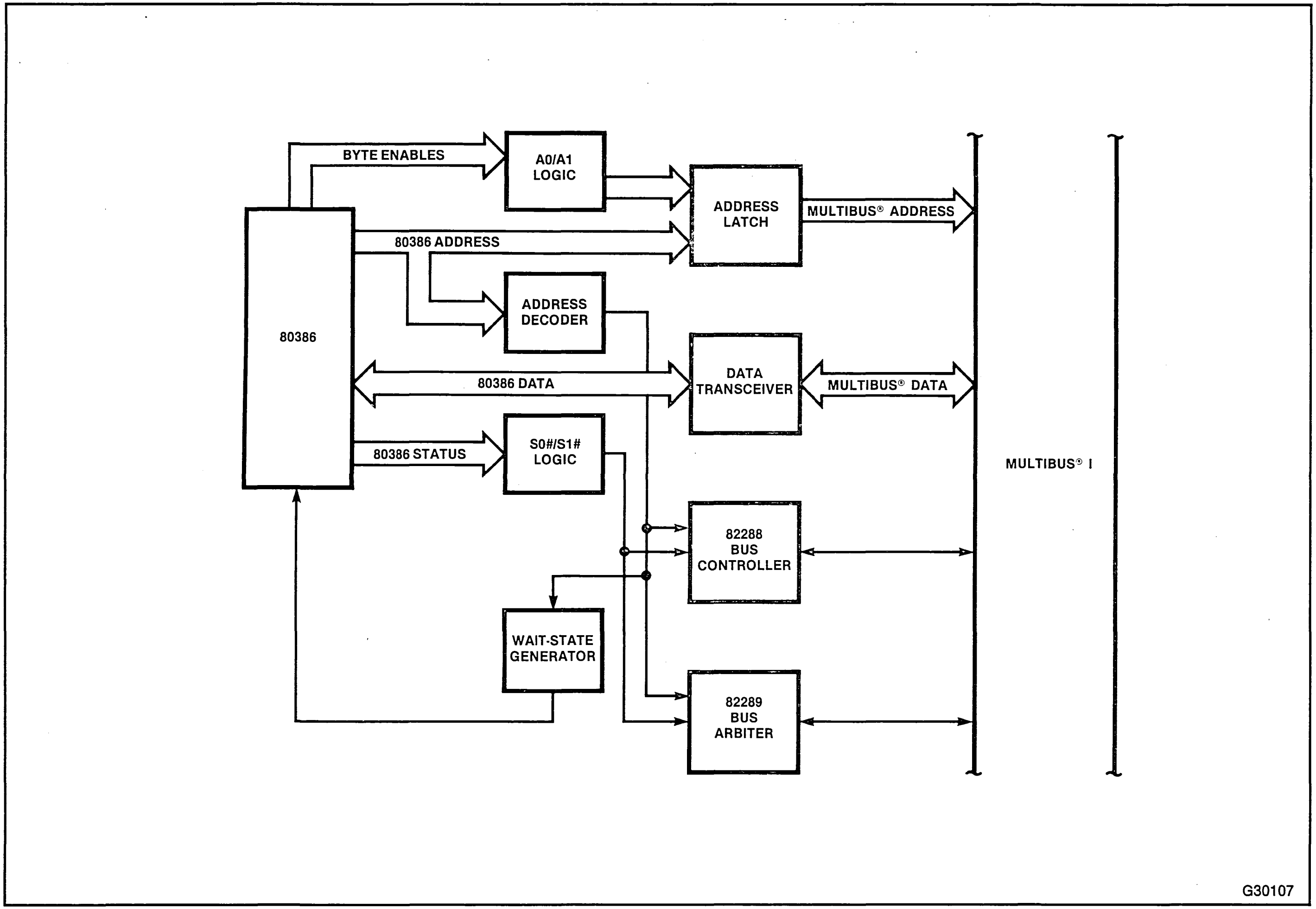

BYTE ENABLES
A0/A1 LOGIC
ADDRESS LATCH
MULTIBUS® ADDRESS
80386 ADDRESS
80386
ADDRESS DECODER
DATA TRANSCEIVER
80386 DATA
MULTIBUS® DATA
S0#/S1# LOGIC
80386 STATUS
MULTIBUS® I
82288 BUS CONTROLLER
WAIT-STATE GENERATOR
82289 BUS ARBITER
G30107

Figure 9-1. 80386-MULTIBUS® I Interface

Inverting address latches convert the 80386 address outputs to the active-low MULTIBUS I address bits. MULTIBUS I address bits are numbered in hexadecimal so that A23-A0 on the 80386 bus become ADR17#-ADR0# on the MULTIBUS I. The BHE# signal is latched to provide the MULTIBUS I BHEN# signal, as shown in Figure 9-2.

MULTIBUS I requires address outputs to be valid for at least 50 nanoseconds after the MULTIBUS I command goes inactive; therefore, the address on all bus cycles is latched. The Address Enable (AEN#) output of the 82289 Bus Arbiter, which goes active when the 82289 has control of the MULTIBUS I, is an output enable for the MULTIBUS I latches. The ALE# output of the 82288 latches the 80386 address for the MULTIBUS I, as shown in Figure 9-2.

Inverting latch/transceivers are needed to provide active-low MULTIBUS I data bits. MULTIBUS I data bits are numbered in hexadecimal, so D15-D0 convert to DATF#-DAT0#. Data is latched only on write cycles. For MULTIBUS I write cycles, the 82288 ALE#, DEN, and DT/R# inputs can control the address latches and data latch/transceivers. For MULTIBUS I read cycles, the local bus RD# signal can control the latch/transceivers. If DEN were used, data contention on the 80386 local bus would result when a MULTIBUS I read cycle immediately followed a local write cycle.

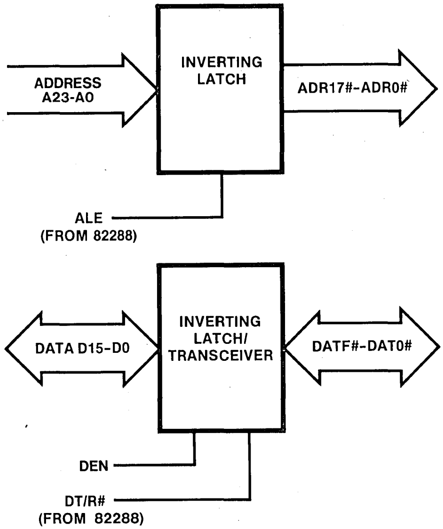


Figure 9-2. MULTIBUS® I Address Latches and Data Transceivers

9.2.2 Address Decoder

A MULTIBUS I system typically has both shared and local memory. I/O devices can also be located either on MULTIBUS I or a local bus. Therefore, the address space of the 80386 must be allocated between MULTIBUS I and the local bus, and address decoding logic must be used to select one bus or the other.

The following two signals are needed for MULTIBUS I selection:

- Bus Size 16 (BS16#) must be returned active to the 80386 to ensure a 16-bit bus cycle. Additional terms for other devices requiring a 16-bit bus can be added to the BS16# PAL equation.
- MULTIBUS Enable (MBEN) selects the 82288 Bus Controller and the 82289 Bus Arbiter on the MULTIBUS I interface. Other outputs of the decoder PAL are programmed to select memory and I/O devices on the local bus.

The decoding of addresses to select either the local bus or the MULTIBUS I is straight forward. In the following example, the system uses the first 64 megabytes of the 80386 memory address space, requiring 26 address lines. The MULTIBUS I memory is allocated to the addresses from F00000H to F3FFFFH. The same PAL equation generates the two PAL outputs BS16# and MBEN:

/A25 * /A24 * A23 * A22 * A21 * A20 * /A19 * /A18

I/O resources residing on MULTIBUS I can be memory-mapped into the memory space of the 80386 or I/O-mapped into the I/O address space independent of the physical location of the devices on MULTIBUS I. The addresses of memory-mapped I/O devices must be decoded to generate I/O read or I/O write commands for memory references that fall within the I/O-mapped regions of the memory space. This technique is discussed in Chapter 8 along with the tradeoffs between memory-mapped I/O and I/O-mapped I/O.

9.2.3 Wait-State Generator

The wait-state generator controls the READY# input of the 80386. For local bus cycles, the wait-state generator produces signal outputs that correspond to each wait state of the 80386 bus cycle, and the PAL READY# output uses these signals to set READY# active after the required number of wait states. Two of the wait-state signals, WS1 and WS2, are also used to generate S0# and S1#.

READY# generation for MULTIBUS I cycles is linked to the Transfer Acknowledge (XACK#) signal, which is returned active by the accessed device on MULTIBUS I when the MULTIBUS I cycle is complete. For a system containing a MULTIBUS I interface as well as a local bus, XACK# must be incorporated into the wait-state generator to produce the READY# signal. The necessary logic is shown in Figure 9-3.

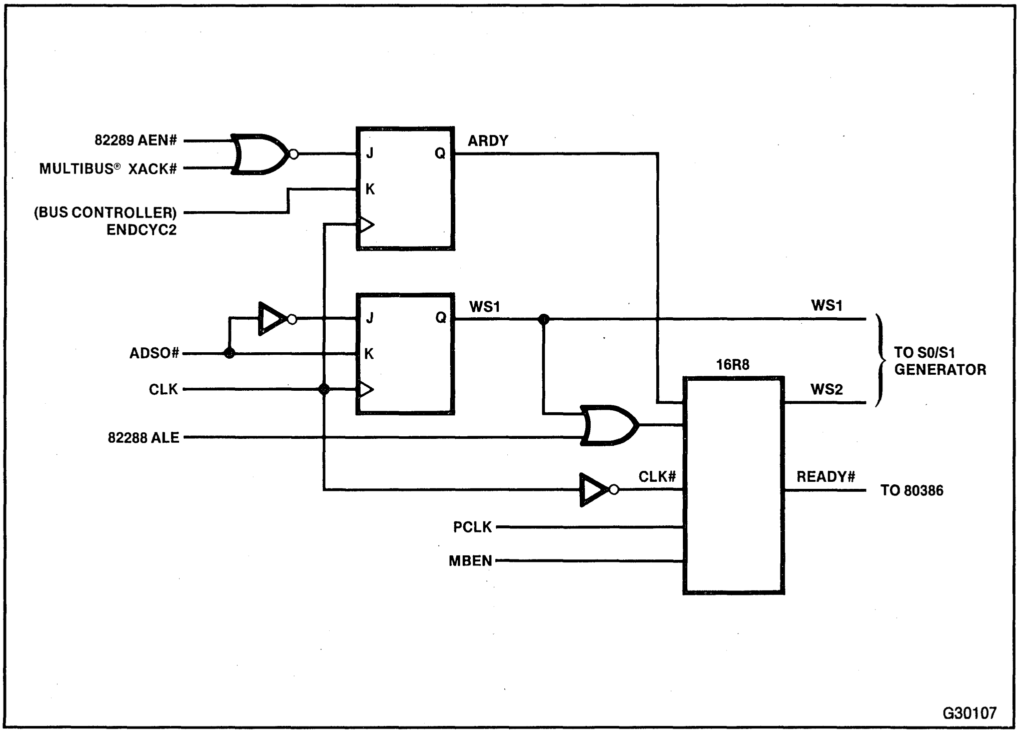

82289 AEN#
MULTIBUS® XACK#
(BUS CONTROLLER) ENDCYC2
J
K
Q
ARDY
ADSO#
CLK
82288 ALE
J
K
Q
WS1
WS1
16R8
WS2
TO S0/S1 GENERATOR
CLK#
READY#
TO 80386
PCLK
MBEN
G30107

Figure 9-3. Wait-State Generator Logic

For MULTIBUS I accesses, the wait-state generator is started by the ALE# signal from the 82288. When XACK# goes active, it is synchronized to CLK. The resulting Asynchronous Ready (ARDY) signal, incorporated into the PAL equation for the READY# signal, causes READY# to be output between two and three CLK cycles after ARDY goes active.

The PCLK signal, which is necessary for producing 80286-compatible wait states, is generated by dividing the CLK signal from the 82384 by two.

To meet the READY# input hold time requirement (25 nanoseconds) for the 82288 Bus Controller, the READY# signal for MULTIBUS I cycles must be two CLK cycles long. Therefore, two PAL equations are required to generate READY#. The first equation generates the Ready Pulse (RDYPLSE) output. RDYPLSE is fed into the READY# equation to extend READY# by an additional CLK cycle. These signals are gated by MBEN and PCLK.

RDYPLSE := ARDY * MBEN * PCLK

/READY := ARDY * MBEN * PCLK + RDYPLSE * MBEN

9.2.4 Bus Controller and Bus Arbiter

Connections for the 82288 and 82289 are shown in Figure 9-4. The 82288 can operate in either local-bus mode or MULTIBUS I mode; a pullup resistor on the 82288 MB input activates the MULTIBUS I mode. Both the 82288 and the 82289 are selected by the MBEN output of the address decoder PAL. The AEN# signal from the 82289 enables the 82288 outputs.

Timing diagrams for MULTIBUS I read and write cycles are shown in Figures 9-5 and 9-6. The only differences between the timings are that a read cycle controls the data latch/transceivers using RD# and outputs the MRDC# command signal, whereas a write cycle controls the data latch/transceivers using DEN and outputs the MWTC# command.

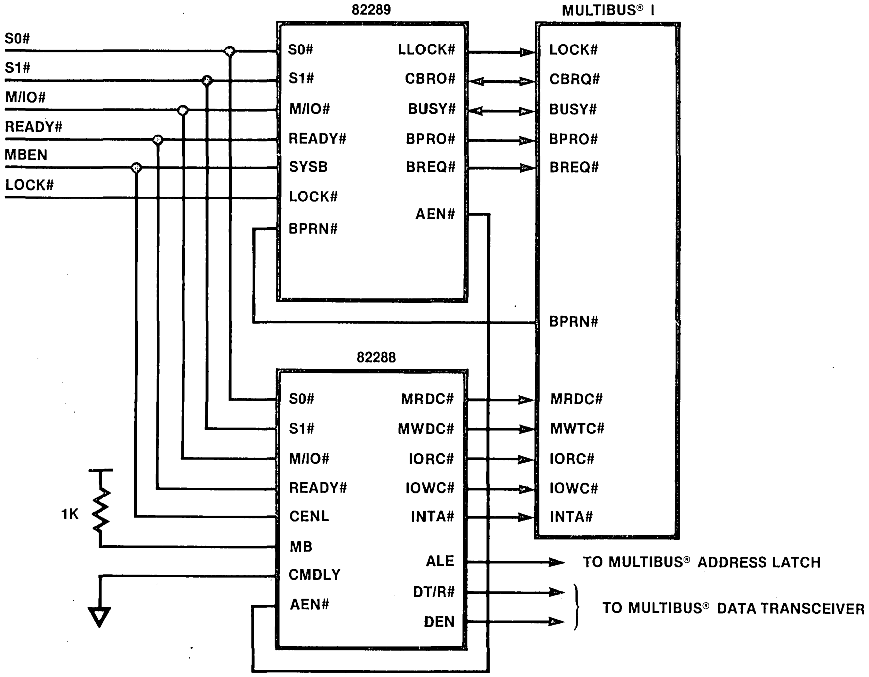

82289
MULTIBUS® I
S0#
S1#
M/IO#
READY#
MBEN
LOCK#
S0#
S1#
M/IO#
READY#
SYSB
LOCK#
BPRN#
LLOCK#
CBRO#
BUSY#
BPRO#
BREQ#
AEN#
LOCK#
CBRQ#
BUSY#
BPRO#
BREQ#
BPRN#
82288
S0#
S1#
M/IO#
READY#
CENL
MB
CMDLY
AEN#
MRDC#
MWDC#
IORC#
IOWC#
INTA#
ALE
DT/R#
DEN
MRDC#
MWTC#
IORC#
IOWC#
INTA#
1K
TO MULTIBUS® ADDRESS LATCH
TO MULTIBUS® DATA TRANSCEIVER
G30107

Figure 9-4. MULTIBUS® Arbiter and Bus Controller

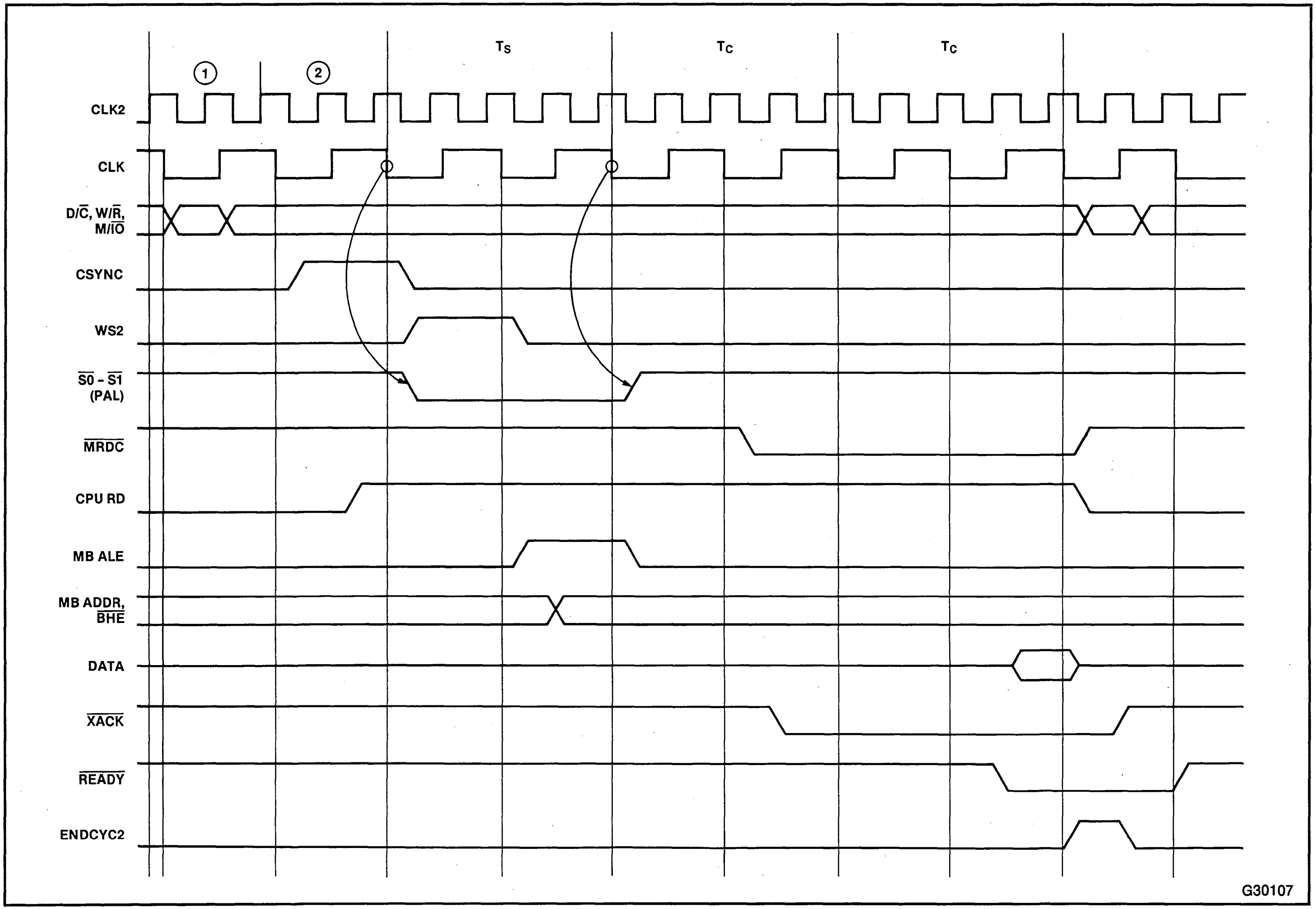

TS
TC
TC
①
②
CLK2
CLK
D/C̄, W/R̄, M/IŌ
CSYNC
WS2
S̄0 - S̄1 (PAL)
M̄R̄D̄C̄
CPU RD
MB ALE
MB ADDR, B̄H̄Ē
DATA
X̄ĀC̄K̄
R̄ĒĀD̄Ȳ
ENDCYC2
G30107

Figure 9-5. MULTIBUS® I Read Cycle Timing

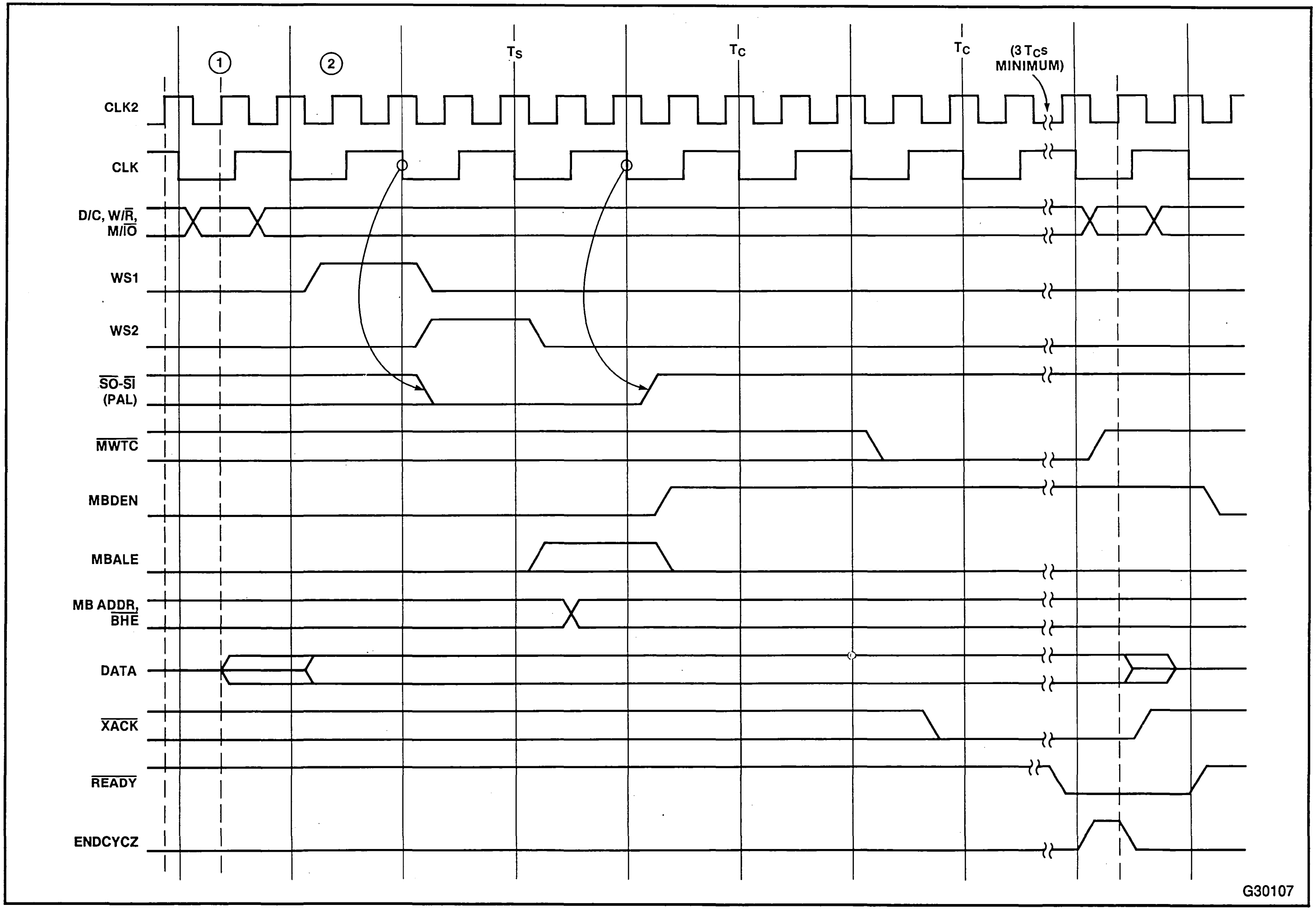

①
②
TS
TC
TC
(3 TCS MINIMUM)
CLK2
CLK
D/C, W/R̄, M/IO̅
WS1
WS2
S̄O̅-S̄I̅ (PAL)
M̄W̄T̄C̄
MBDEN
MBALE
MB ADDR, B̄H̄Ē
DATA
X̄ĀC̄K̄
R̄ĒĀD̄Ȳ
ENDCYCZ
G30107

Figure 9-6. MULTIBUS® I Write Cycle Timing

9.3 TIMING ANALYSIS OF MULTIBUS® I INTERFACE

The timing specifications for the MULTIBUS I are explained in the *MULTIBUS® I Specification*, Order Number 9800683. Table 9-1 lists the MULTIBUS I parameters that relate to the 80386 system. These calculations are based on the assumption that 74ALS580 latches and 74F544 transceivers are used for the MULTIBUS I address and data interface.

In addition to the parameters in Table 9-1, designers must allow for the following:

- To ensure sufficient access time for the slave device, bus operations must not be terminated until an XACK# signal is received from the slave device.
- Following an MRDC# or an IORC# command, the responding slave device must disable its data drivers within 125 nanoseconds after the return of the XACK# signal. All devices that meet the MULTIBUS I specification of 65 nanoseconds meet this requirement.

9.4 82289 BUS ARBITER

In a MULTIBUS I system, several processing subsystems contend for the use of shared resources. If one processor requests access to MULTIBUS I while another processor is using it, the requesting processor must wait. Bus arbitration logic controls access to MULTIBUS I for all processing subsystems.

Each processing subsystem contains its own 82289 Bus Arbiter. The Bus Arbiter directs its processor onto the bus and allows higher and lower priority bus masters to access the bus. Once the bus arbiter gains control of MULTIBUS I, the 80386 can access system resources. The bus arbiter handles bus contention in a manner that is transparent to the 80386.

Table 9-1. MULTIBUS® I Timing Parameters

Timing Parameter	MULTIBUS Specification	80386 System Timing
tAS Address setup before command active	50 ns minimum	125 ns (2 CLK cycles) − 20 ns (ALE max delay) − 22 ns (74ALS580 max. delay) + 3 ns (Command min. delay) 86 ns min.
tDS Write data setup before command active	50 ns minimum	125 ns (2 CLK cycles) − 30 ns (DEN max. delay) − 12 ns (74F544 max. delay) + 3 ns (Command min. delay) 86 ns min.
tAH Address hold after command inactive	50 ns minimum	187.5ns (3 CLK cycles) − 25 ns (Command inactive max. delay) + 3 ns (ALE max. delay) 165.5ns min

Each processor in the multiprocessing system initiates bus cycles as though it has exclusive use of MULTIBUS I. The bus arbiter keeps track of whether the subsystem has control of the bus and prevents the bus controller from accessing the bus when the subsystem does not control the bus.

When the bus arbiter receives control of MULTIBUS I, it enables the bus controller and address latches to drive MULTIBUS I. When the transfer is complete, MULTIBUS I returns the XACK# signal, which activates READY# to end the bus cycle.

9.4.1 Priority Resolution

Because a MULTIBUS I system includes many bus masters, logic must be provided to resolve priority between two bus masters that simultaneously request control of MULTIBUS I. Figure 9-7 shows two common methods for resolving priority: serial priority and parallel priority.

The serial priority technique is implemented by daisy-chaining the Bus Priority In (BPRN#) and Bus Priority Out (BPRO#) signals of all the bus arbiters in the system. Due to delays in the daisy chain, this technique accommodates only a limited number of bus arbiters.

The parallel priority technique requires external logic to recognize the BPRN# inputs from all bus arbiters and return the BPRO# signal active to the requesting bus arbiter that has the highest priority. The number of bus arbiters accommodated with this technique depends on the complexity of the decoding logic.

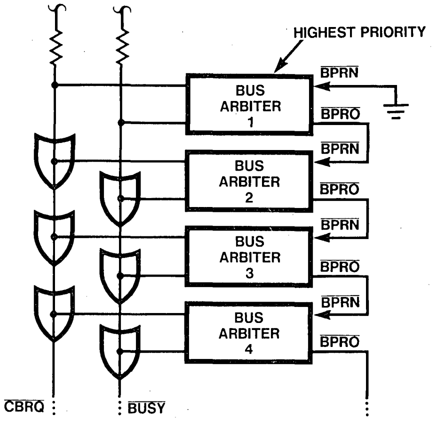
Priority resolution logic need not be included in the design of a single processing subsystem with a MULTIBUS I interface. The bus arbiter takes control of MULTIBUS I when the BPRN# signal goes active and relinquishes control when BPRN# goes inactive. As long as external logic exists to control the BPRN# inputs of all bus arbiters, a subsystem can be designed independent of the priority resolution circuit.

9.4.2 82289 Operating Modes

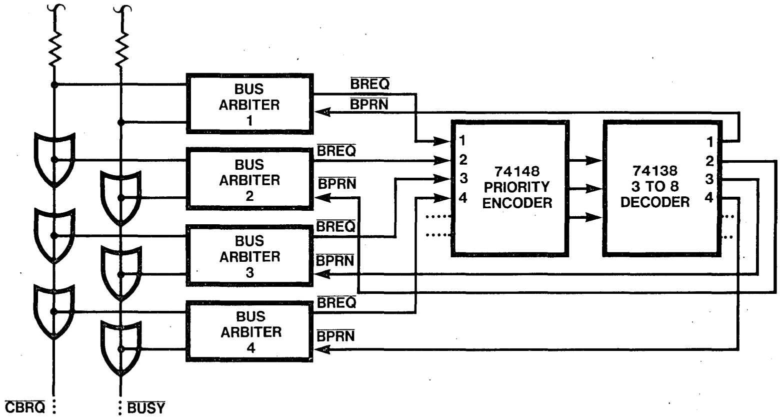
Following a MULTIBUS I cycle, the controlling bus arbiter can either retain bus control or release control so that another bus master can access the bus. Three modes for relinquishing bus control are as follows:

- Mode 1—The bus arbiter releases the bus at the end of each cycle.
- Mode 2—The bus arbiter retains control of the bus until another bus master (of any priority) requests control.
- Mode 3—The bus arbiter retains control of the bus until a higher priority bus master requests control.

In addition, the bus arbiter can switch between modes 2 and 3, based on the type of bus cycle.


HIGHEST PRIORITY
BUS ARBITER 1
BUS ARBITER 2
BUS ARBITER 3
BUS ARBITER 4
BPRN
BPRO
BPRN
BPRO
BPRN
BPRO
BPRN
BPRO
CBRQ
BUSY

SERIAL PRIORITY RESOLVING TECHNIQUE


BUS ARBITER 1
BUS ARBITER 2
BUS ARBITER 3
BUS ARBITER 4
BREQ
BPRN
BREQ
BPRN
BREQ
BPRN
BREQ
BPRN
1
2
3
4
74148 PRIORITY ENCODER
74138 3 TO 8 DECODER
1
2
3
4
CBRQ
BUSY

PARALLEL PRIORITY RESOLVING TECHNIQUE

210760-132

Figure 9-7. Bus Priority Resolution

Figure 9-8 shows the strapping configurations required to implement each of these four techniques.

The operating mode of one bus arbiter affects the throughput of both the individual subsystem as well as other subsystems on MULTIBUS I. This is because the delay required to transfer MULTIBUS I control from one bus arbiter to another affects all subsystems waiting to use MULTIBUS I. Therefore, the most efficient operating mode depends on how often a subsystem accesses MULTIBUS I and how this frequency compares to that of the other subsystems.

- Mode 1 is adequate for a subsystem that needs MULTIBUS I access only occasionally. By releasing MULTIBUS I after each bus cycle, the subsystem minimizes its impact on other subsystems that use MULTIBUS I.

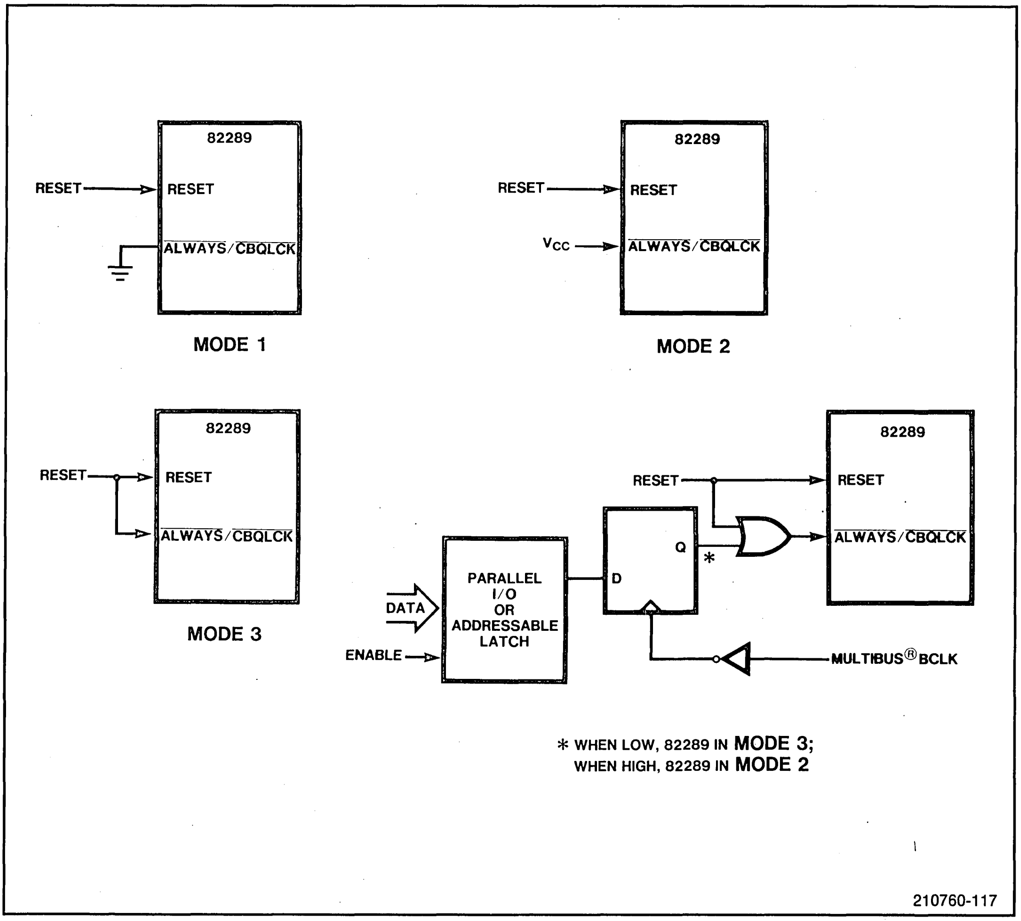

82289
RESET
RESET
ALWAYS/CBQLCK
MODE 1
82289
RESET
RESET
Vcc
ALWAYS/CBQLCK
MODE 2
82289
RESET
RESET
ALWAYS/CBQLCK
MODE 3
DATA
ENABLE
PARALLEL I/O OR ADDRESSABLE LATCH
D
Q
*
RESET
82289
RESET
ALWAYS/CBQLCK
MULTIBUS® BCLK
* WHEN LOW, 82289 IN MODE 3;
WHEN HIGH, 82289 IN MODE 2
210760-117

Figure 9-8. Operating Mode Configurations

- Mode 2 is suited for a subsystem that is one of several subsystems that are all equally likely to require MULTIBUS I. The performance decrease caused by the delay necessary to take control of MULTIBUS I is distributed evenly to all subsystems.
- Mode 3 should be used for a subsystem that uses MULTIBUS I frequently. The delay required for taking control of MULTIBUS I and the consequent performance decrease is shifted to subsystems that use MULTIBUS I less often.
- Switching between modes 2 and 3 is useful if the subsystem demand for MULTIBUS I is unknown or variable.

9.4.3 MULTIBUS® I Locked Cycles

Locked bus cycles for the local bus are described in Chapter 3. In locked bus cycles, the 80386 asserts the LOCK# signal to prevent another bus master from intervening between two bus cycles. In the same manner, an 80386 processing subsystem can assert the LLOCK# output of its bus arbiter to prevent other subsystems from gaining control of MULTIBUS I. A locked cycle overrides the normal operating mode of the bus arbiter (one of the four modes mentioned above).

Locked MULTIBUS I cycles are typically used to implement software semaphores (described in Chapter 3) for critical code sections or critical real-time events. Locked cycles can also be used for high-performance transfers within one instruction.

The 80386 initiates a locked MULTIBUS I cycle by asserting its LOCK# output to the 82289 bus arbiter. The bus arbiter outputs its LLOCK# signal to the MULTIBUS I LOCK# status line and holds LLOCK# active until the LOCK# signal from the 80386 goes inactive. The LLOCK# signal from the bus arbiter must be connected to the MULTIBUS I LOCK# status line through a tristate driver controlled by the AEN# output of the bus arbiter.

9.5 OTHER MULTIBUS® I DESIGN CONSIDERATIONS

Additional design considerations are presented in this section. These considerations include provisions for interrupt handling, 8-bit transfers, timeout protection, and power failure handling on MULTIBUS I.

9.5.1 Interrupt-Acknowledge on MULTIBUS® I

When an interrupt is received by the 80386, the 80386 generates an interrupt-acknowledge cycle (described in Chapter 3) to fetch an 8-bit interrupt vector from the 8259A Programmable Interrupt Controller. The 8259A can be located on either MULTIBUS I or a local bus.

Multiple 8259As can be cascaded (one master and up to eight slaves) to process up to 64 interrupts. Three configurations are possible for cascaded interrupt controllers:

- All of the interrupt controllers for one 80386 reside on the local bus of that processor, and all interrupt-acknowledge cycles are directed to the local bus.
- All slave interrupt controllers (those that connect directly to interrupting devices) reside on MULTIBUS I. The master interrupt controller may reside on either the local bus or MULTIBUS I. In this case, all interrupt-acknowledge cycles are directed to MULTIBUS I.
- Some slave interrupt controllers reside on local buses, and other slave interrupt controllers reside on MULTIBUS I. In this case, the appropriate bus for the interrupt-acknowledge cycle depends on the cascade address generated by the master interrupt controller.

In the first two configurations, no decoding is needed because all interrupt acknowledge cycles are directed to one bus. However, if a system contains a master interrupt controller residing on a local bus and at least one slave interrupt controller residing on MULTIBUS I, address decoding must select the bus for each interrupt-acknowledge cycle.

The interrupt-acknowledge cycle must be considered in the design of this decoding logic. The 80386 responds to an active INTR input by performing two bus cycles. During the first cycle, the master interrupt controller determines which, if any, of its slave controllers should return the interrupt vector and drive sits cascade address pins (CAS0#, CAS1#, CAS2#) to select that slave controller. During the second cycle, the 80386 reads an 8-bit vector from the selected interrupt controller and uses this vector to service the interrupt.

In a system that has slave controllers residing on MULTIBUS I, the circuit shown in Figure 9-9 can be used to decode the three cascade address pins from the master controller to select either MULTIBUS I or the local bus for the interrupt-acknowledge cycle. If MULTIBUS I is selected, the 82289 Bus Arbiter is enabled. The 82289 in turn requests control of MULTIBUS I and enables the address and data transceivers when the request is granted.

The bus-select signal must become valid for the second interrupt-acknowledge cycle. The master controller's cascade address outputs become valid within 565 nanoseconds after the INTA# output from the bus control logic goes active. Bus-select decoding requires 30 nanoseconds, for a total of 595 nanoseconds from INTA# to bus-select valid. The four idle bus cycles that the 80386 automatically inserts between the two interrupt-acknowledge cycles provides some of this time. The wait-state generator must add wait states to the first interrupt-acknowledge cycle to provide the rest of the time needed for the bus-select signal to become valid.

The cascade address outputs are gated onto A8, A9, and A10 of the address bus through three-state drivers during the second interrupt-acknowledge cycle. Bus control logic must generate a Master Cascade Enable (MCE) signal to enable these drivers. This signal must remain valid long enough for the cascade address to be captured in MULTIBUS I address latches; however it must be de-asserted before the 80386 drives the address bus.

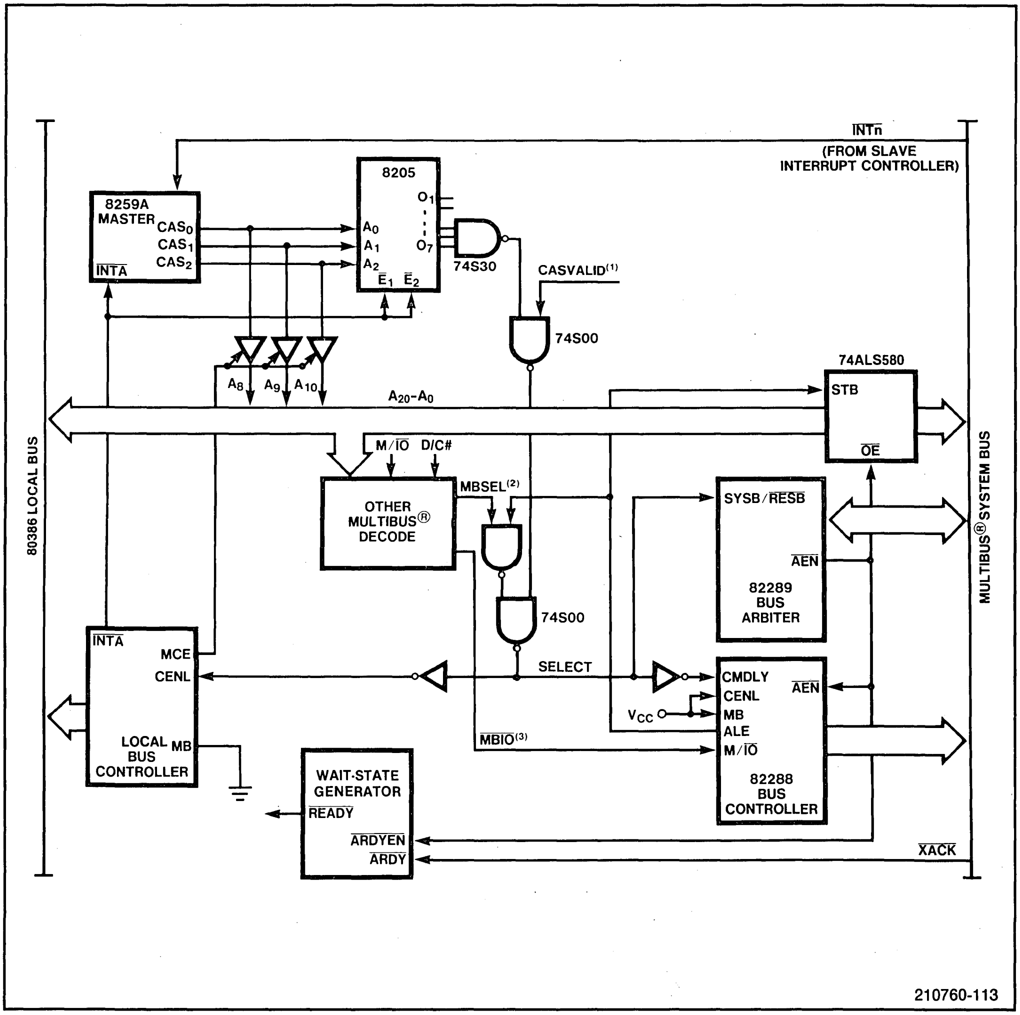

INTn
(FROM SLAVE INTERRUPT CONTROLLER)
8205
O1
O7
A0
A1
A2
E1 E2
74S30
CASVALID(1)
8259A MASTER
CAS0
CAS1
CAS2
INTA
74S00
74ALS580
STB
OE
A8
A9
A10
A20-A0
M/IO D/C#
MBSEL(2)
OTHER MULTIBUS® DECODE
SYSB/RESB
AEN
82289 BUS ARBITER
74S00
SELECT
INTA
MCE
CENL
LOCAL BUS CONTROLLER
MB
CMDLY
CENL
MB
ALE
M/IO
AEN
Vcc
82288 BUS CONTROLLER
MBIO(3)
WAIT-STATE GENERATOR
READY
ARDYEN
ARDY
XACK
80386 LOCAL BUS
MULTIBUS® SYSTEM BUS
210760-113

Figure 9-9. Bus-Select Logic for Interrupt Acknowledge

9.5.2 Byte Swapping during MULTIBUS® I Byte Transfers

The MULTIBUS I standard specifies that all byte transfers must be performed on the lower eight data lines (MULTIBUS I DAT0#-DAT7#), regardless of the address of the data. An 80386 subsystem must swap data from eight of its upper 24 data lines (D8-D15, D16-D23, or D24-D31) to its lower eight data lines (D0-D7) before transferring data to MULTIBUS I, and swap data from its lower data lines to the appropriate upper data lines when reading a byte from MULTIBUS I. This byte-swapping requirement maintains compatibility between 8-bit, 16-bit, and 32-bit systems sharing the same MULTIBUS I.

The BS16# signal is generated and returned to the 80386 for all MULTIBUS I cycles. The 80386 automatically swaps data between the lower half (D15-D0) and the upper half (D31-D16) of its data bus and adds an extra bus cycle as necessary to complete the data transfer. Therefore, only the logic to swap data from D15-D8 to D7-D0 is needed to meet the byte-swapping requirement of MULTIBUS I.

Figure 9-10 illustrates a circuit that performs the byte-swapping function. The Output Enable (OE#) inputs of the data latch/transceivers are conditioned by the states of the BHE# and A0 outputs of the address decoder.

9.5.3 Bus Timeout Function for MULTIBUS® I Accesses

The MULTIBUS I XACK# signal terminates an 80386 bus cycle by driving the wait-state generator logic. However, if the 80386 addresses a nonexistent device on MULTIBUS I, the XACK# signal is never generated. Without a bus-timeout protection circuit, the 80386 waits indefinitely for an active READY# signal and prevents other processors from using MULTIBUS I.

Figure 9-11 shows an implementation of a bus-timeout circuit that ensures that all MULTIBUS I cycles eventually end. The ALE# output of the bus controller activates a one-shot that outputs a 1-millisecond pulse. The rising edge of the pulse activates the TIMEOUT# signal if READY# does not go active within 1 millisecond to clear the TIMEOUT# flip-flop. The TIMEOUT# signal is input to the wait-state generator logic to activate the READY# signal. When READY# goes active, it is returned to clear the TIMEOUT# signal.

9.5.4 MULTIBUS® I Power Failure Handling

The MULTIBUS I interface includes a Power Fail Interrupt PFIN signal to signal an impending system power failure. Typically, PFIN# is connected to the non-maskable interrupt (NMI) request input of each 80386. The NMI service routine can direct the 80386 to save its environment immediately, before falling voltages and the MULTIBUS I Memory Protect (MPRO#) signal prevent any further memory activity. In systems with memory backup power or nonvolatile memory, the saved environment can be recovered on powerup.

The power-up sequence of the 80386 can check the state of the MULTIBUS I Power Fail Sense Latch (PFSN#) to see if a previous power failure has occurred. If this signal is active (low), the 80386 can branch to a power-up routine that resets the latch using the Power Fail Sense Reset signal (PFSR#), restores the previous 80386 environment, and resumes execution.

Further guidelines for designing 80386 systems with power failure features are contained in the Intel *MULTIBUS® I Specification.*

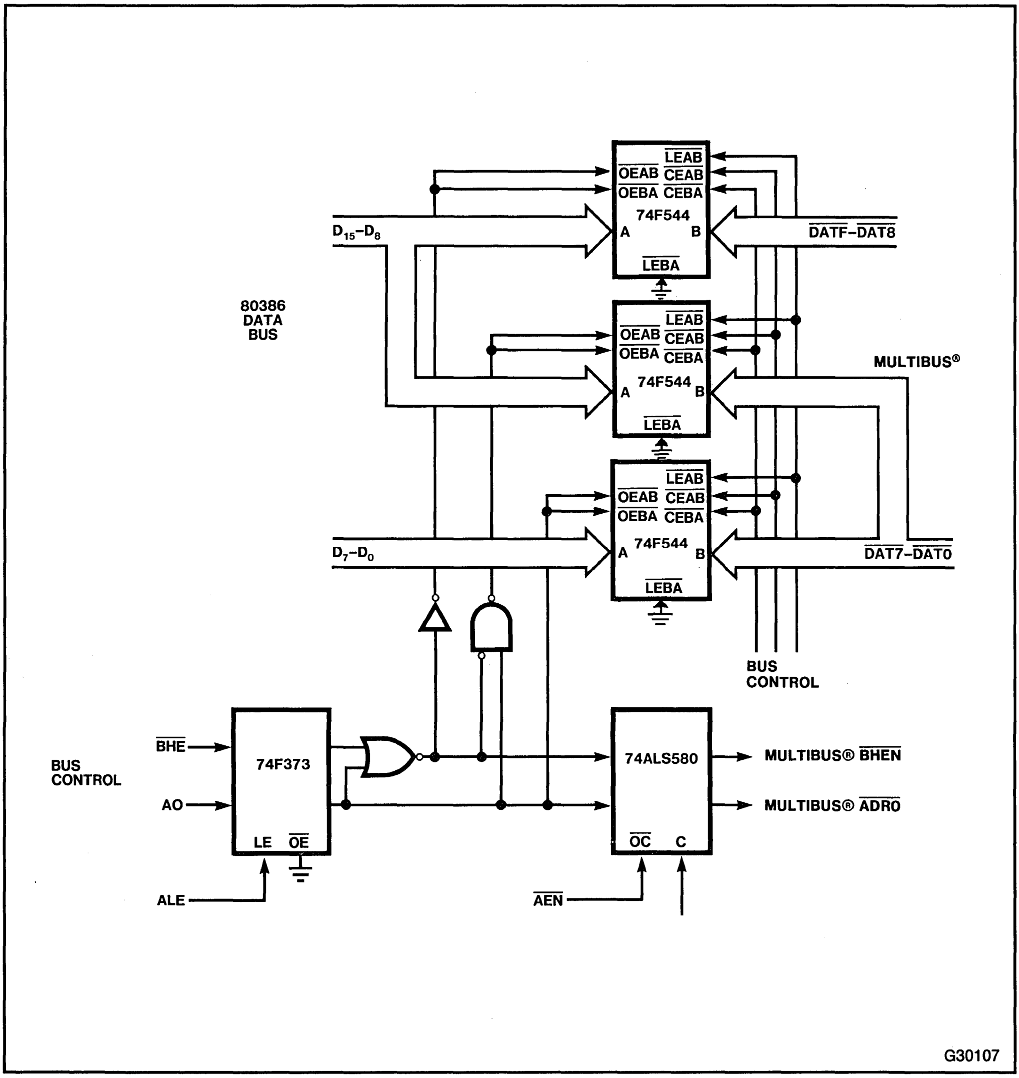

LEAB
OEAB CEAB
OEBA CEBA
74F544
A B
LEBA
D15-D8
DATF-DAT8
80386
DATA
BUS
MULTIBUS®
D7-D0
DAT7-DAT0
BUS
CONTROL
BHE
74F373
AO
LE OE
ALE
74ALS580
MULTIBUS® BHEN
MULTIBUS® ADRO
OC C
AEN
G30107

Figure 9-10. Byte-Swapping Logic

9.6 iLBX™ BUS EXPANSION

The iLBX (Local Bus Expansion) is a high-performance bus interface standard that permits the modular expansion of an 80386-based system. An iLBX interface links the 80386 system board with additional boards containing memory, I/O subsystems, and other peripheral devices or bus masters. Any board that conforms to the iLBX standard can be added to the system as the user's needs dictate. For a 16-MHz 80386-based system, a typical iLBX access cycle requires six wait states.

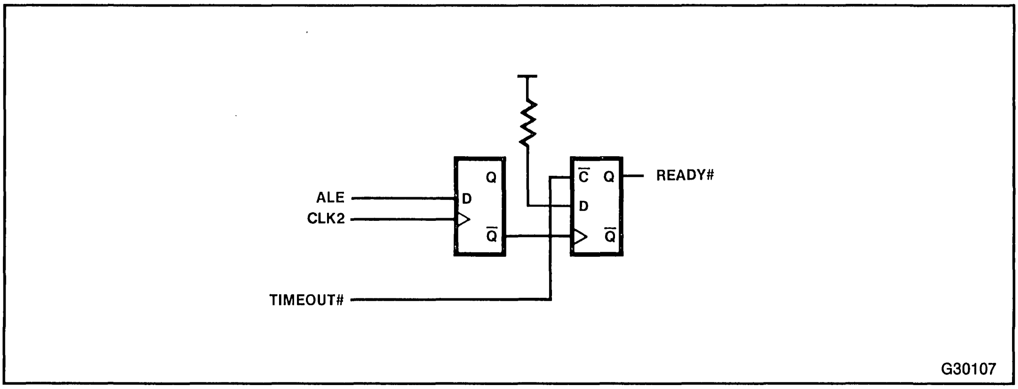


Figure 9-11. Bus-Timeout Protection Circuit

The *iLBX™ Bus Specification* describes the iLBX Local Bus Expansion standard in detail.

The iLBX bus interface requires the generation of A1, A0, and BHE# from the 80386 BE3#-BE0# outputs. The iLBX connector contains 24 address bits (AB23-AB0) and 16 data bits (DB15-DB0), which are taken from the buffered address lines (A23-A0), and data lines (D15-D0) of the 80386 local bus. BHE# is inverted and buffered to provide the Byte High Enable (BHEN) signal.

The Read/Write (R/W#), Data Strobe (DSTB#), and Address Strobe (ASTB#) controls are generated from local bus control signals using the logic shown in Figure 9-12. R/W# is a delayed, inverted version of the W/R# output of the 80386. DSTB# goes active when either RD# or WR# from the local bus control goes active. ASTB# and DSTB# are delayed to allow adequate setup time for BHEN. In this example, the WS2 signal, which is active during the third CLK cycle of the 80386 bus cycle, provides the delay.

A chip-select output of address decoding logic goes active for accesses to the memory and I/O locations allocated to the iLBX bus and selects the iLBX address and data buffers. Command signals from the local bus control logic enable the outputs of the iLBX transceivers.

When an iLBX cycle is complete, the Acknowledge (ACK#) signal is returned over the iLBX bus. This signal must be synchronized and incorporated into the wait-state generator logic to provide the READY# signal.

9.7 DUAL-PORT RAM WITH MULTIBUS® I

A dual-port RAM is a memory subsystem that can be accessed by both the 80386, through its local bus, and other processing subsystems, through the MULTIBUS I system bus. Dual-port RAM offers some of the advantages of both local resources and system resources. It is an effective solution when using only local memory or only system memory would decrease system cost and/or performance significantly.

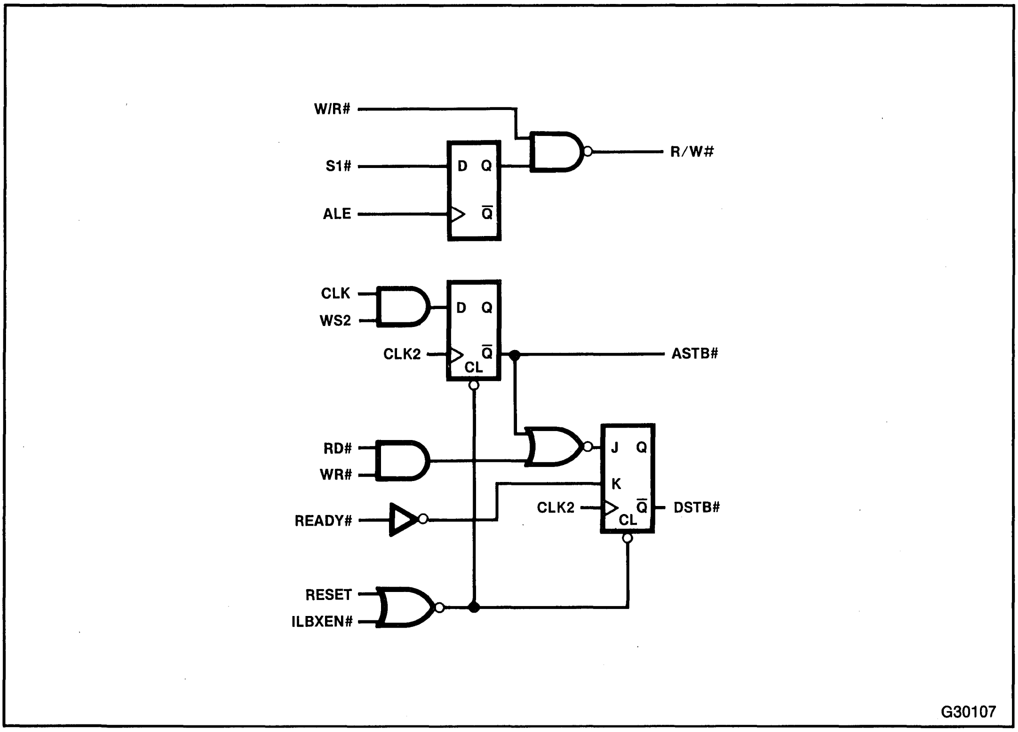


Figure 9-12. iLBX™ Signal Generation

The 80386 accesses dual-port RAM through its high-speed local bus, leaving MULTIBUS I free for other system operations. Other processing subsystems can pass data to and from the 80386 through the dual-port RAM using MULTIBUS I.

If necessary, dual-port RAM can be mapped to reserve address ranges for the exclusive use of the 80386. The 80386 and the other processing subsystems need not use the same address mapping for dual-port RAM.

The disadvantage of dual-port RAM is that its design is more complex than that of either local or system memory. Dual-port RAM requires arbitration logic to ensure that only one of the two buses gains access at one time.

9.7.1 Avoiding Deadlock with Dual-Port RAM

The MULTIBUS-LOCK# signal and the 80386 LOCK# signal mediate contention when both the 80386 and a MULTIBUS I device attempt to access dual-port RAM. However, locked cycles to dual-port RAM can potentially result in deadlock. Deadlock arises when the 80386 performs locked cycles to ensure back-to-back accesses to dual-port RAM and MULTIBUS I.

Suppose the 80386 locks an access to dual-port RAM followed by a MULTIBUS access, to ensure that the accesses are performed back-to-back. (This could happen only in protected mode during interrupt processing when the IDT is in the dual-port RAM and the target descriptor is in MULTIBUS RAM.) At the same time the 80386 performs the first locked cycle, another device gains control of MULTIBUS I for the purpose of accessing dual-port RAM. The 80386 cannot gain control of MULTIBUS I to complete the locked operation, and the other device cannot relinquish control of MULTIBUS I because it cannot complete its access to dual-port RAM. Each device therefore enters an interminable wait state.

Two approaches can be used to avoid deadlock:

- Requiring software to be free of locked accesses to dual-port RAM.
- Designing hardware to negate the LOCK# signal for transfers between dual-port RAM and MULTIBUS I. If this approach is used, software writers must be informed that such transfers will not be locked even though software dictates locked cycles.

MULTIBUS® II and 80386

10

CHAPTER 10
MULTIBUS® II AND 80386

Standard bus interfaces guarantee compatibility between existing and newly developed systems. This compatibility safeguards a user's hardware investment against obsolescence even in the face of rapidly advancing technology. The MULTIBUS I standard interface has proven its value in providing flexibility for the expansion of existing systems and the integration of new designs. The MULTIBUS II standard interface extends Intel's Open Systems design strategy into the world of 32-bit microprocessing systems.

10.1 MULTIBUS® II STANDARD

The MULTIBUS II standard is a processor-independent bus architecture that features a 32-bit parallel system bus with a maximum throughput of 40 megabytes per second, high-speed local bus access to off-board memory, a low-cost serial system bus, and full multiprocessing support. MULTIBUS II achieves these features through five specialized Intel buses:

- Parallel System Bus (iPSB)
- Local Bus Extension (iLBX II)
- Serial System Bus (iSSB)
- Multi-channel DMA I/O Bus
- System Expansion I/O Bus (iSBX)

The DMA I/O Bus and the iSBX are carried over directly from MULTIBUS I architecture. See the *MULTIBUS® I Architectural Specification* for a full description of these buses. The multiple bus structure provides the following important advantages over a single, generalized bus:

- Each bus is optimized for a specific function.
- The buses perform operations in parallel.
- Buses that are not needed for a particular system can be omitted, avoiding unnecessary costs.

10.2 PARALLEL SYSTEM BUS (iPSB)

The Parallel System Bus (iPSB) is optimized for interprocessor data transfer and communication. Its burst transfer capability provides a maximum sustained bandwidth of 40 megabytes per second for high-performance data transfers.

The iPSB supports four address spaces per bus agent (a board that encompasses a functional subsystem). The conventional I/O and memory address spaces are included, plus two other address spaces that support advanced functions:

- An 255 address message space supports message passing. Typically, a microprocessor performs interprocessor communications inefficiently. Message passing allows two bus agents to exchange a block of data at full bus bandwidth without supervision from a microprocessor. An intelligent bus interface capable of message passing shifts the burden of interprocessor communication away from the processor, thus enhancing overall system performance.
- An interconnect space allows geographic addressing, which is the identification of any bus agent (board) by slot number. Every MULTIBUS II system contains a Central Services Module (CSM) that provides system services, such as uniform initialization and bus timeout detection, for all bus agents residing on the iPSB bus. The CSM may use the registers of the interconnect space of each bus agent to configure the agent dynamically. Stake pin jumpers, DIP switches, and other hardware configuration devices can be eliminated.

Because the 80386 can access only memory space or I/O space, the message space and interconnect space may be mapped into the memory space or the I/O space. Decoding logic provides chip select signals for the devices implementing the message space and the interconnect space, as well as devices in the memory space and the I/O space.

Three types of bus cycles define activity on the iPSB bus:

- Arbitration Cycle—Determines the next owner of the bus. This cycle consists of a resolution phase, in which competing bus agents determine priority for bus control, and an acquisition phase, in which the agent with the highest priority initiates a transfer cycle.
- Transfer Cycle—Performs a data transfer between the bus owner and another bus agent. This cycle consists of a request phase, in which address control signals are driven, and a reply phase, in which the two agents perform a handshake to synchronize the data transfer. The reply phase is repeated and data transfers continue until the bus owner ends the transfer cycle.
- Exception Cycle—Indicates that an exception (error) has occurred during a transfer cycle. This cycle consists of a signal phase, in which an exception signal from one bus agent causes all other bus agents to terminate any arbitration and transfer cycles in progress, and a recovery phase, in which the exception signals go inactive. A new arbitration cycle can begin on the clock cycle after the recovery phase.

Figure 10-1 shows how the timing of these cycles overlap.

10.2.1 iPSB Interface

Each bus agent must provide a means of transferring data between its 80386, its interconnect registers, and the iPSB bus. The location of bus interface logic to meet this requirement is shown in Figure 10-2. A full-featured subsystem may also include provisions for the message passing protocols used by the iPSB bus.

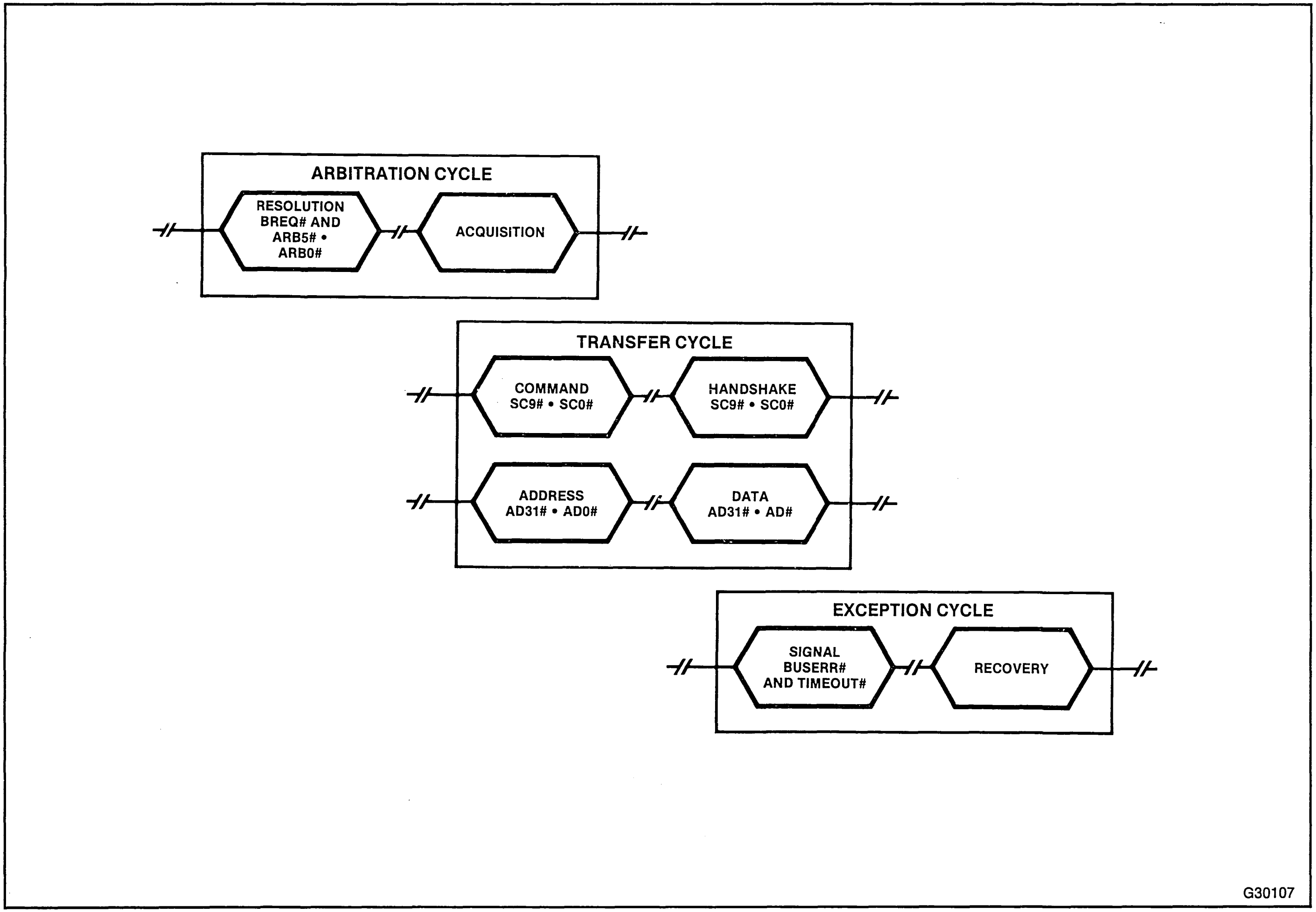

ARBITRATION CYCLE
RESOLUTION BREQ# AND ARB5# • ARB0#
ACQUISITION
TRANSFER CYCLE
COMMAND SC9# • SC0#
HANDSHAKE SC9# • SC0#
ADDRESS AD31# • AD0#
DATA AD31# • AD#
EXCEPTION CYCLE
SIGNAL BUSERR# AND TIMEOUT#
RECOVERY
G30107

Figure 10-1. iPSB Bus Cycle Timing

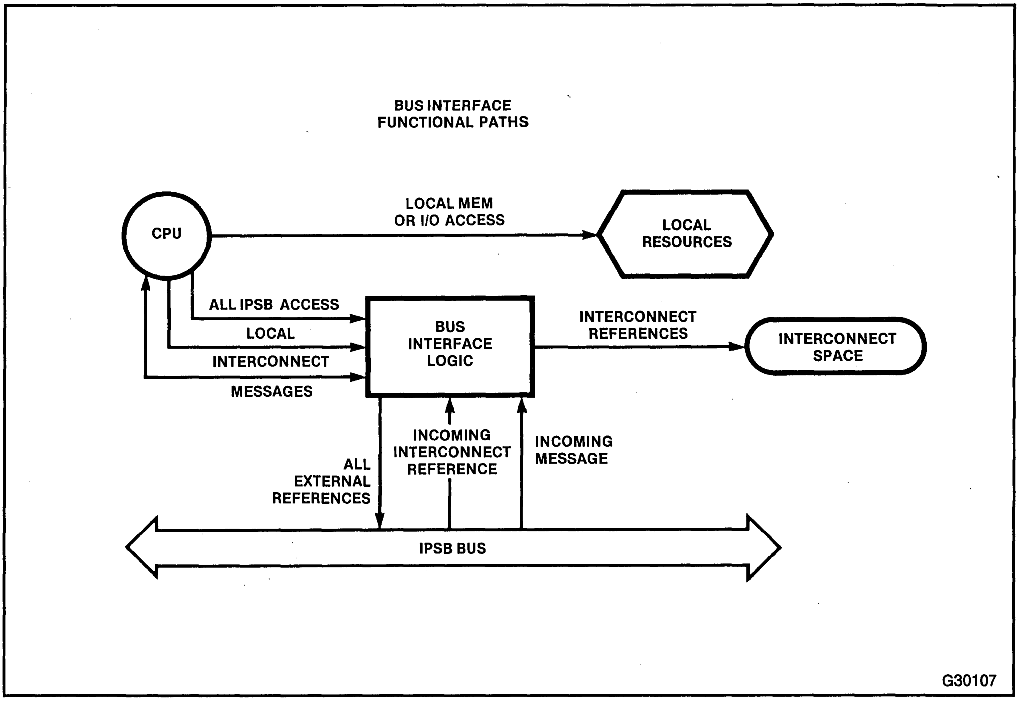

BUS INTERFACE
FUNCTIONAL PATHS
CPU
LOCAL MEM
OR I/O ACCESS
LOCAL
RESOURCES
ALL IPSB ACCESS
LOCAL
INTERCONNECT
MESSAGES
BUS
INTERFACE
LOGIC
INTERCONNECT
REFERENCES
INTERCONNECT
SPACE
ALL
EXTERNAL
REFERENCES
INCOMING
INTERCONNECT
REFERENCE
INCOMING
MESSAGE
IPSB BUS
G30107

Figure 10-2. iPSB Bus Interface

The iPSB interface may be conveniently implemented by a Bus Arbiter/Controller (BAC), a Message Interrupt Controller (MIC), and miscellaneous logic. The BAC coordinates direct interaction with the other devices on the iPSB bus, while the MIC works through the BAC to send and receive interrupt messages. Other logic is needed for address decoding, parity checking, and control signal generation.

The BAC and MIC are implemented in Intel gate arrays. In addition, Intel is developing an advanced CMOS device, the Message Passing Coprocessor (MPC), that integrates the functions of the BAC and the MIC plus parity checking and full message passing (solicited and unsolicited), all in one package called the BIC (Bus Interface Controller). Systems designed today with the available BAC and MIC can be upgraded to the MPC in the future.

10.2.1.1 BAC SIGNALS

The BAC provides arbitration and system control logic for the arbitration, transfer, and exception cycles defined by the MULTIBUS II architecture. Through the BAC, the bus agent functions as either a requestor or a replier in a transfer cycle. In all cases, the device requiring iPSB bus access (either the 80386 or the MIC) is completely isolated from the iPSB; the BAC provides all direct interaction.

The BAC signals can be divided into three functional groups:

- iPSB interface
- Local bus interface
- Register interface with the 80386

The iPSB interface signals perform mainly arbitration and system control. Five bidirectional Arbitration signals (ARB5-ARB0) are used during reset to read a cardslot ID and arbitration ID from the CSM, and during arbitration cycles to output the arbitration ID for priority resolution. Bus Request (BREQ#) is a bidirectional signal. Each bus agent asserts BREQ# to request control of the bus and samples BREQ# to determine if other agents are also contending for bus control.

Bus Error (BUSERR#) is a bidirectional signal that a bus agent outputs to all other bus agents when it detects a parity error during a transfer cycle. Bus Timeout (TIMOUT #) is output by the CSM to all bus agents when a bus cycle fails to end within a prescribed time period.

Ten System Control signals (SC9#-SC0#) coordinate transfer cycles. The *MULTIBUS® II Architectural Specification* defines each of these signals. Directional enables (SCOEH and SCOEL) are provided for transceivers to buffer these bidirectional signals. External logic checks byte parity on the multiplexed address and data bus (AD31-AD0) and sets the Parity inputs (PAR3-PAR0) accordingly.

Other iPSB signals are Reset (RST#), Reset-Not-Complete (RSTNC#), and ID Latch (LACHn#, n = slot number). These signals are used only during reset.

Local bus interface signals pertain to the communication between the BAC and the 80386 or between the BAC and the MIC. These signals indicate to the BAC when to request bus control and what type of bus cycle to drive when it gains bus control.

Four control signals are necessary for each of the two devices connected to the BAC. The signals that connect to the 80386 are REQUESTA, GRANTA, READYA, and SELECTA; those that connect to the MIC are REQUESTB, GRANTB, READYB, and SELECTB.

To request bus control, the 80386 or the MIC activates one of the REQUEST signals. The corresponding GRANT signal is returned by the BAC when it has bus control. Data width and address space selections are encoded on the WIDTH1#, WIDTH0#, SPACE1#, and SPACE0# inputs, while WR# dictates either a write cycle or a read cycle. These five inputs translate directly to SC6#-SC2# outputs during the request phase of a transfer cycle. READYA or READYB indicates that WIDTH0#, WIDTH1#, SPACE0#, SPACE1#, and WR# can be read by the BAC to drive the transfer cycle.

LASTINA or LASTINB controls the end-of-cycle signal for burst transfers. The LOCK# input is activated for locked transfers.

The bus agent that receives a transfer cycle from the bus owner must have its BAC enabled by an active SELECT input. Errors detected by the replying agent are encoded by its MIC on the AGERR2-AGERR0 inputs to its BAC so that the BAC can drive the SC7#-SC5# lines accordingly. If an error occurs, the requesting agent notifies the 80386 through the EINT signal.

The register interface signals control register operations between the 80386 and the BAC. Three 5-bit registers (Arbitration ID, Slot ID, and Error Port) are addressed through RSEL1 and RSEL0. Data is transferred on RIO4-RIO0; the direction of transfer is indicated by RRW.

10.2.1.2 MIC SIGNALS

The MIC coordinates interrupt handling for a bus agent on the iPSB bus. Interrupts are implemented as virtual interrupts in the message space. To send an interrupt message, the 80386 writes four bytes to the MIC to indicate the source, destination, and type of message. The MIC then coordinates the message transfer. The MIC of the receiving bus agent reads the 4-byte message and stores it in a 4-deep message queue to be read by the 80386.

The MIC signals are divided into three groups:

- iPSB interface
- Local bus interface
- BAC interface

The iPSB interface consists of the multiplexed address/data bus (AD31#-AD0#). Although the MIC gains access to the iPSB bus through the BAC, the MIC drives the address/data bus directly. As a requesting agent, the MIC drives the address and data at the appropriate times. As a receiving agent, the MIC monitors the address/data bus for its address. When it recognizes its address, the MIC selects its BAC to perform the required handshake and read the message into the message queue. Then, the MIC interrupts the 80386 to indicate that the message is pending in the queue. The 80386 reads the message and services the interrupt accordingly.

The local bus interface consists of seven register/ports, addressed through A2-A0, through which the MIC and the 80386 communicate. Data is transferred over D7-D0, and WR# and RD# determine the direction of transfer. Other signals include the MIC Chip Select (CS#), a WAIT# signal for adding wait states to the 80386 cycle, and a Message Interrupt (MINT) to signal an interrupt condition to the 80386.

The BAC interface includes REQUESTB, READYB, SELECTB, and GRANTB. These signals have already been described with the other BAC signals.

While the BAC and the MIC together provide the backbone for an iPSB interface, other logic provides buffering and control to round out the interface. An 8751 Microcontroller coordinates 80386 access to the interconnect space. An address decoder distinguishes between local, interconnect, and iPSB accesses. PALs control the buffering of signals between the 80386, BAC, MIC, 8751 Microcontroller, and iPSB bus.

10.3 LOCAL BUS EXTENSION (iLBX™ II)

The iLBX II bus extension is a high-speed execution bus designed for quick access to off-board memory. One iLBX II bus extension can support either two processing subsystems (called the primary requesting agent and the secondary requesting agent) plus four memory subsystems, or a single processing subsystem plus five memory subsystems. A MULTIBUS II system may contain more than one iLBX II bus extension to meet its memory requirements.

The iLBX II bus extension features a 26-bit address bus and a separate 32-bit data bus. Because these paths are separate, the extension allows pipelining of transfer cycles; the request phase of a transfer cycle can overlap the reply phase of the previous cycle.

Other features of the iLBX II bus extension are:

- A unidirectional handshake for fast data transfers
- Mutual exclusion capability to control multiported memory
- Interconnect space (for each bus agent) through which the primary requesting agent initializes and configures all other bus agents.

10.4 SERIAL SYSTEM BUS (iSSB)

The Serial System Bus (iSSB) provides a simple, low-cost alternative to the Parallel System Bus (iPSB) bus. In applications that do not require the high performance of the iPSB bus, the iSSB bus can provide some cost reduction. In systems containing both the iPSB bus and the iSSB bus, the iSSB bus provides an alternate path for interface control, diagnostics, or redundancy.

The iSSB bus can contain up to 32 bus agents distributed over a maximum of 10 meters. Bus control is determined through an access protocol called Carrier Sense Multiple Access with Collision Detection (CSMA/CD). This protocol allows agents to transmit data whenever they are ready. In case of simultaneous transmission by two or more bus agents, the iSSB invokes a deterministic collision resolution algorithm to grant fair access to all agents.

From the application point of view, the error detection capability of the iSSB bus, coupled with an intelligent bus agent interface (able to retransmit) makes the iSSB bus as reliable as the iPSB bus, even though the iSSB bus may be up to 10 meters long.

Physical Design and Debugging 11

CHAPTER 11
PHYSICAL DESIGN AND DEBUGGING

This chapter outlines recommendations for providing adequate power to the 80386, addressing high-frequency issues that do not exist for lower-frequency systems, achieving the proper thermal environment for the 80386, and building an 80386-based system successfully. The guidelines presented here allow the user to gain the full benefits of the high-performance 80386.

11.1 POWER AND GROUND REQUIREMENTS

The CHMOS III 80386 differs from previous HMOS microprocessors in that its power dissipation is primarily capacitive; there is almost no DC power dissipation. Power dissipation depends mostly on frequency.

Power dissipation can be distinguished as either internal (logic) power or I/O (bus) power. Internal power varies with operating frequency and to some extent with wait states and software. Internal power increases with supply voltage also. Process variations in manufacturing affect internal power, although to a lesser extent than with NMOS processes.

I/O power, which accounts for roughly one-fifth of the total power dissipation, varies with frequency and voltage. It also depends on capacitive bus load. Capacitive bus loadings for all output pins are specified in the 80386 data sheet. The 80386 output valid delays will increase if these loadings are exceeded. The addressing pattern of the software can affect I/O power by changing the effective frequency at the address pins. The variation in frequency at the data pins tends to be smaller; thus varying data patterns should not cause a significant change in power dissipation.

11.1.1 Power and Ground Planes

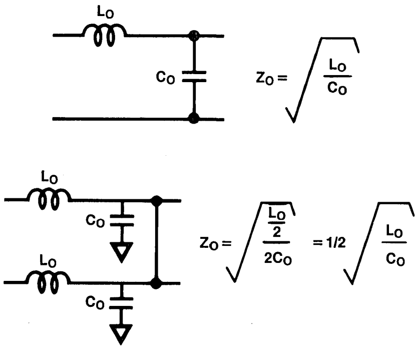
Power and ground planes must be used in 80386 systems to minimize noise. Power and ground lines have inherent inductance and capacitance, therefore an impedance $Z = (L/C)^{1/2}$. The total characteristic impedance for the power supply can be reduced by adding more lines. This effect is illustrated in Figure 11-1, which shows that two lines in parallel have half the impedance of one. To reduce the impedance even further, the user should add more lines. In the limit, an infinite number of parallel lines, or a plane, results in the lowest impedance. Planes also provide the best distribution of power and ground.

The 80386 has 20 V_{cc} pins and 21 V_{ss} (ground) pins. All power and ground pins must be connected to a plane. Ideally, the 80386 is located at the center of the board, to take full advantage of these planes.

Although the 80386 generally demands less power than the 80286, the possibility for power surges is increased due to higher frequency and pin count. Peak-to-peak noise on V_{cc} relative to V_{ss} should be maintained at no more than 400 mV, and preferably no more than 200 mV.

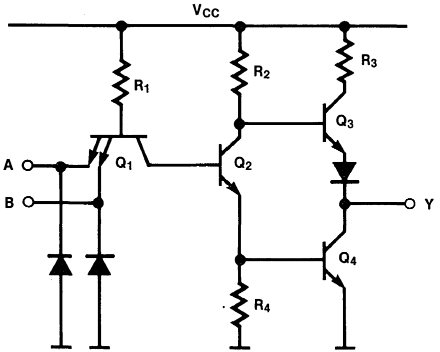
11.1.2 Decoupling Capacitors

The switching activity of one device can propagate to other devices through the power supply. For example, in the TTL NAND gate of Figure 11-2, both Q3 and Q4 transistors are on for a short time when the output is switching. This increased load causes a negative spike on V_{cc} and a positive spike on ground. In synchronous systems in which many gates switch simultaneously, the result is significant noise on the power and ground lines.


Lo
Co
$Z_O = \sqrt{\frac{L_O}{C_O}}$
Lo
Co
Lo
Co
$Z_O = \sqrt{\frac{\frac{L_O}{2}}{2C_O}} = 1/2 \sqrt{\frac{L_O}{C_O}}$

G30107

Figure 11-1. Reducing Characteristic Impedance


Vcc
R1
R2
R3
Q3
A
B
Q1
Q2
Y
Q4
R4

G30107

Figure 11-2. Circuit without Decoupling

Decoupling capacitors placed across the device between V_{cc} and ground reduce voltage spikes by supplying the extra current needed during switching. These capacitors should be placed close to their devices because the inductance of connection lines negates their effect.

When selecting decoupling capacitors, the user should provide 0.01 microfarads for each device and 0.1 microfarads for every 20 gates. Radio-frequency capacitors must be used; they should be distributed evenly over the board to be most effective. In addition, the board should be decoupled from the external supply line with a 2.2 microfarads capacitor.

Chip capacitors (surface-mount) are preferable because they exhibit lower inductance and require less total board space. They should be connected as in Figure 11-3. Leaded capacitors can also be used if the leads are kept as short as possible. Six leaded capacitors are required to match the effectiveness of one chip capacitor, but because only a limited number can fit around the 80386, the configuration in Figure 11-4 results.

11.2 HIGH-FREQUENCY DESIGN CONSIDERATIONS

At high signal frequencies, the transmission line properties of signal paths in a circuit must be considered. Reflections, interference, and noise become significant in comparison to the high-frequency signals. They can cause false signal transitions, data errors, and input voltage level violations. These errors can be transient and therefore difficult to debug. In this section, some high-frequency design issues are discussed; for more information, consult a reference book on high-frequency design.

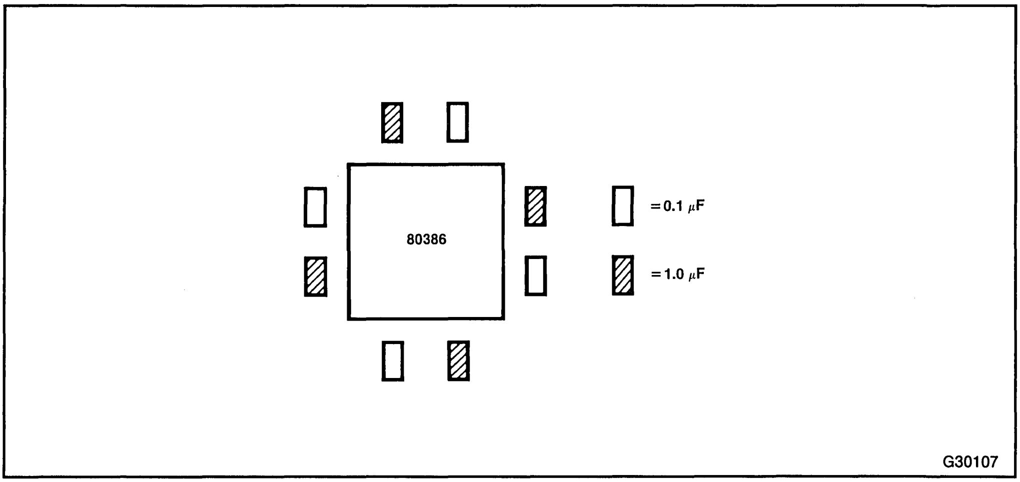


Figure 11-3. Decoupling Chip Capacitors

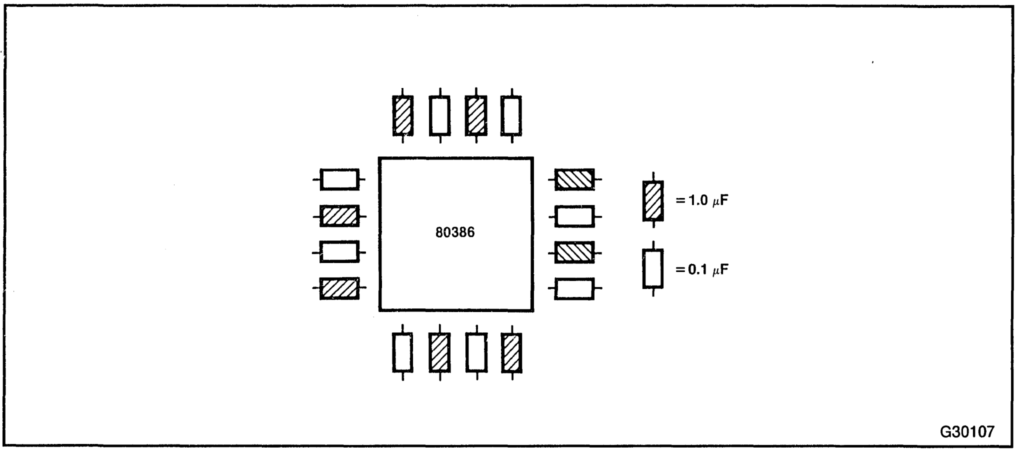


Figure 11-4. Decoupling Leaded Capacitors

11.2.1 Line Termination

Input voltage level violations are usually due to voltage spikes that raise input voltage levels above the maximum limit (overshoot) and below the minimum limit (undershoot). These voltage levels can cause excess current on input gates that results in permanent damage to the device. Even if no damage occurs, most devices are not guaranteed to function as specified if input voltage levels are exceeded.

Signal lines are terminated to minimize signal reflections and prevent overshoot and undershoot. If the round-trip signal path delay is greater than the rise time or fall time of the signal, terminate the line. If the line is not terminated, the signal reaches its high or low level before reflections have time to dissipate, and overshoot and undershoot occur.

There are two methods of termination: series and split. A series termination compensates for excess current before the signal travels down the line; a split termination adjusts the current at the end of the line.

Series termination decreases current flow in the signal path by adding a series resistor, as shown in Figure 11-5. The resistor increases the rise and fall times of the signal so that the change in current occurs over a longer period of time. Because the amount of voltage overshoot and undershoot depends on the change in current over time ($V = L\, di/dt$), the increased time reduces overshoot and undershoot. The series resistor should be placed as close as possible to the signal source.

Series termination is advantageous because less power is consumed than in split termination. However, series termination reduces signal rise and fall times, so it should not be used when these times are critical.

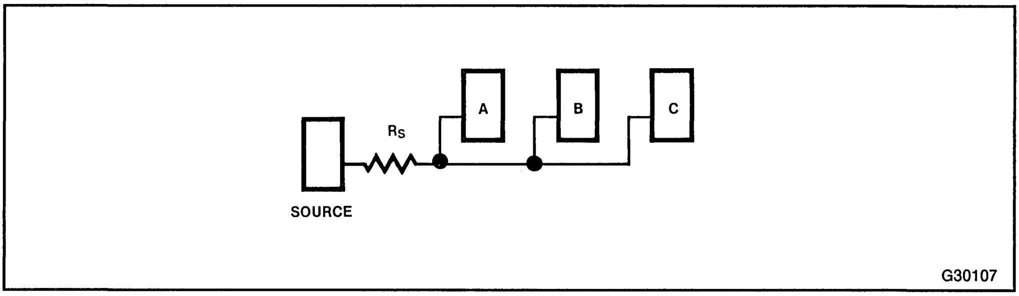


Figure 11-5. Series Termination

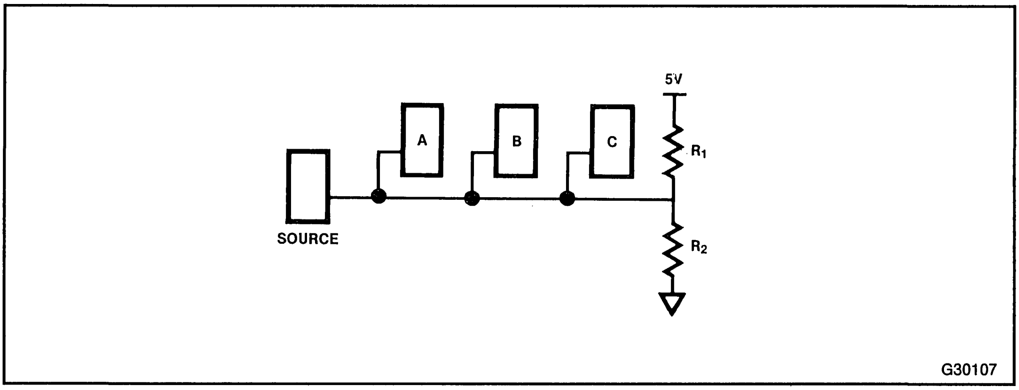


Figure 11-6. Split Termination

Split termination is effective in reducing signal reflection (ringing). Split termination is accomplished by the addition of two resistors, as shown in Figure 11-6. These two resistors "absorb" electrical energy at the end of the signal line. Split termination requires a line driver capable of supplying the large current drawn by R1 and R2.

11.2.2 Interference

Interference is the result of electrical activity in one conductor causing transient voltages to appear in another conductor. Interference increases with the following factors:

- Frequency-Interference is the result of changing currents and voltages. The more frequent the changes, the greater the interference.
- Closeness of the two conductors—Interference is due to electromagnetic and electrostatic fields whose effects are weaker further from the source.

There are two types of interference to consider in high frequency circuits: electromagnetic interference (EMI) and electrostatic interference (ESI).

EMI (also called crosstalk) is caused by the magnetic field that exists around any current-carrying conductor. The magnetic flux from one conductor can induce current in another conductor, resulting in transient voltage. Several precautions can minimize EMI:

- Running a ground line between two adjacent lines wherever they traverse a long section of the circuit board. The ground line should be grounded at both ends.
- Running ground lines between the lines of an address bus or a data bus if either of the following conditions exists:
  - The bus is on an external layer of the board.
  - The bus is on an internal layer but not sandwiched between power and ground planes that are at most 10 mils away.
- Avoiding closed loops in signal paths (see Figure 11-7). Closed loops cause excessive current and create inductive noise, especially in the circuitry enclosed by a loop.

ESI is caused by the capacitive coupling of two adjacent conductors. The conductors act as the plates of a capacitor; a charge built up on one induces the opposite charge on the other. The following steps reduce ESI:

- Separating signal lines so that capacitive coupling becomes negligible.
- Running a ground line between two lines to cancel the electrostatic fields.

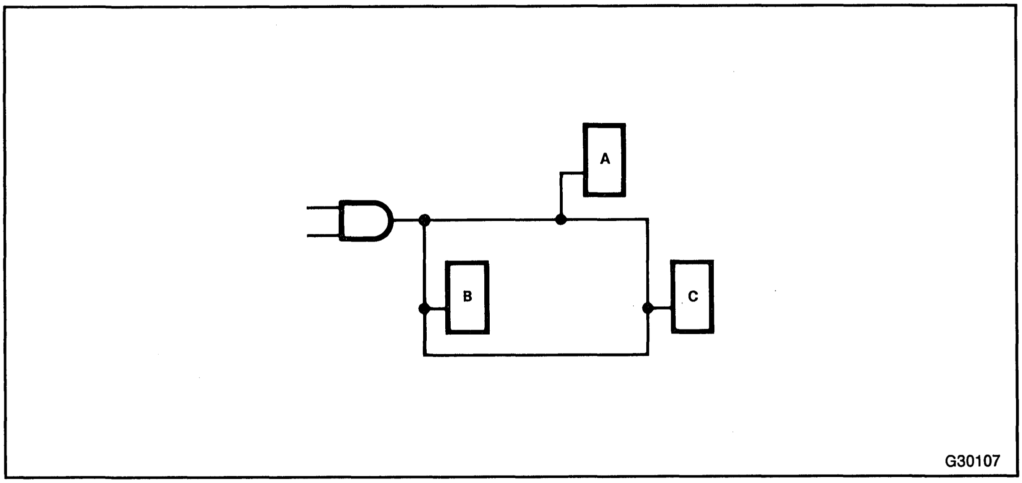


Figure 11-7. Avoid Closed-Loop Signal Paths

11.2.3 Latchup

Latchup is a condition in a CMOS circuit in which V_{cc} becomes shorted to V_{ss}. Intel's CHMOS III process is immune to latchup under normal operating conditions. Latchup can be triggered when the voltage limits on I/O pins are exceeded, causing internal PN junctions to become forward biased. The following guidelines help prevent latchup:

- Observing the maximum rating for input voltage on I/O pins.
- Never applying power to an 80386 pin or a device connected to an 80386 pin before applying power to the 80386 itself.
- Preventing overshoot and undershoot on I/O pins by adding line termination and by designing to reduce noise and reflection on signal lines.

11.3 CLOCK DISTRIBUTION AND TERMINATION

For performance at high frequencies, the clock signal (CLK2) for the 80386 must be free of noise and within the specifications listed in the 80386 data sheet. These requirements can be met by following these guidelines:

- Using the 82384 Clock Generator to provide both CLK2 and CLK signals. The 82384 is designed to match 80386 specifications.
- Terminating the CLK2 output with a series resistor to obtain a clean signal. The resistor value is calculated by measuring the total capacitive load on the CLK2 signal and referring to Figure 11-8. If the total capacitive load is less than 80 picofarads, the user should add capacitors to make up the difference. Because of the high frequency of CLK2, the terminating resistor must have low inductance; carbon resistors are recommended.
- Not putting more than two loads on a single trace to avoid signal reflection (see Figure 11-9 for example configurations).
- Using an oscilloscope to compare the CLK2 waveform with those in Figure 11-10.

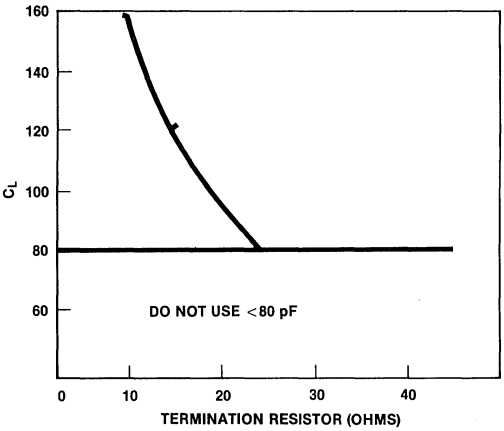
11.4 THERMAL CHARACTERISTICS

The thermal specification for the 80386 defines the maximum case temperature. This section describes how to ensure that an 80386 system meets this specification.

Thermal specifications for the 80386 are designed to guarantee a tolerable temperature at the surface of the 80386 chip. This temperature (called the junction temperature) can be determined from external measurements using the known thermal characteristics of the package. Two equations for calculating junction temperature are as follows:

$T_j = T_a + (\theta_{ja} * PD)$ and

$T_j = T_c + (\theta_{jc} * PD)$



- $C_L = C_{IN}(386) + C_{IN}(387) + C_{IN}(PALs) + \ldots + C_{BOARD}$.
  C_{BOARD} IS CALCULATED FROM LAYOUT AND BOARD PARAMETERS: THICKNESS, DIELECTRIC CONSTANT, DISTANCE TO GROUND/V_{CC} PLANES.
- TERMINATION RESISTOR MUST BE LOW INDUCTANCE TYPE. RECOMMEND CARBON FILLED TYPE.

G30107

Figure 11-8. CLK2 Series Termination

where

T_j = junction temperature

$_a$ = ambient temperature

T_c = case tempterature

θ_{ja} = junction-to-ambient temperature coefficient

θ_{jc} = junction-to-case temperature coefficient

PD = power dissipation (worst-case $I_{cc} * V_{cc}$

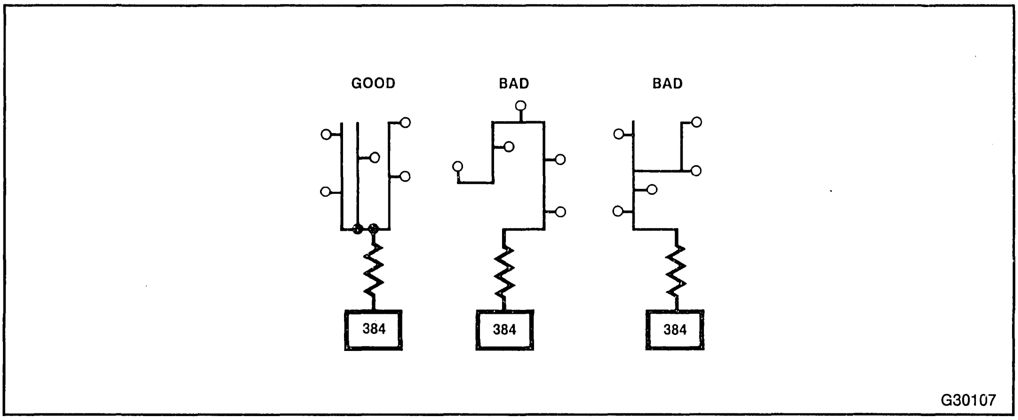


Figure 11-9. CLK2 Loading

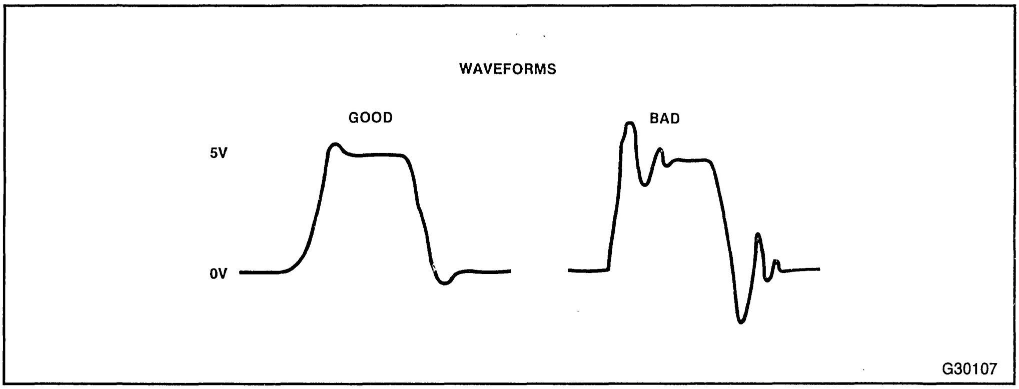


Figure 11-10. CLK2 Waveforms

Case temperature calculations offer several advantages over ambient temperature calculations:

- Case temperature is easier to measure accurately than ambient temperature because the measurement is localized to a single point (top center of the package).
- The worst-case junction temperature (T_j) is lower when calculated with case temperature for the following reasons:
  - The junction-to-case thermal coefficient (θ_{jc}) is lower than the junction-to-ambient thermal coefficient (θ_{ja}); therefore, calculated junction temperature varies less with power dissipation (PD).
  - θ_{jc} is not affected by air flow in the system; θ_{ja} varies with air flow.

With the case-temperature specification, the designer can either set the ambient temperature or use fans to control case temperature. Finned heat sinks or conductive cooling may also be used in environments where the use of fans is precluded. To approximate the case temperature for various environments, the two equations above should be combined by setting the junction temperature equal for both, resulting in this equation:

$$T_a = T_c - ((\theta_{ja} - \theta_{jc}) *PD)$$

The current data sheet should be consulted to determine the values of θ_{ja} (for the system's air flow) and ambient temperature that will yield the desired case temperature. Whatever the conditions are, the case temperature is easy to verify.

11.5 DEBUGGING CONSIDERATIONS

This section outlines an approach to building and debugging 80386 hardware incrementally. In a short time, a complete 80386-based system can be built and working. This approach does not have to be followed to the letter, but it provides several valuable debugging concepts and useful hints. Use these guidelines in conjunction with the 80386 data sheet, which contains detailed information about the 80386.

11.5.1 Hardware Debugging Features

Even before a system is built, debugging can be made easier by planning a suitable environment for the 80386. The 80386 board (whether it is a printed circuit board or a wire-wrap board) must have power and ground planes. The user should provide a decoupling capacitor between V_{cc} and GND next to each IC on the board. All 80386 V_{cc} and GND pins should be connected individually to the appropriate power or ground plane; multiple power or ground pins should not be daisy-chained.

Room in the system should be included for the following physical features to aid debugging:

- Two switches: one for generating the RESET signal to the 80386 and one for tying the READY# signal high (negated).
- Connections for a logic analyzer on major control signals:
  - Inputs to the 80386:
    - Ready (READY#)
    - Next Address (NA#)
    - Bus Size 16 (BS16#)
    - Data Bus (D0-D31)
  - Outputs from the 80386:
    - Address Strobe (ADS#)
      - Write/Read (W/R#), Data/Control (D/C#), Memory/IO (M/IO#), Lock (LOCK#)
    - Address Bus (A2-A31)
    - Byte Enable (BE0#-BE3#)

Logic analyzer connection points should be provided to all 80386 address outputs (A2-A31 and BE0#-BE3#) even if there are not enough logic analyzer inputs to accommodate all of them. Initially, only BE0#, BE1#, BE2#, BE3#, and the output of the address decoder circuit should be connected. The single output of an address decoder circuit represents many bits of address information. If the address decoder does not work as expected, more of the logic analyzer inputs should be moved to the 80386 address pins.

- Buffers and visual indicators (such as LEDs) for three or four of the critical 80386 control signals. A visual indicator for the ADS# output, for example, will light when the system is performing bus cycles.

11.5.2 Bus Interface

During initial debugging, bus-cycle operation should be simplified. The 80386 bus interface is flexible enough to be tested in stages. To simplify bus control, the initial testing should be performed with a non-pipelined address. The NA# input should be tied high (negated) to guarantee no address pipelining. The only signals that need to be controlled are the READY# input and the BS16# input.

The READY# input on the 80386 lets the user delay the end of any bus cycle for as long as necessary. For each CLK cycle after T2 that READY# is not sampled active, a wait state is added. READY# can be used to provide extra time (wait states) for slow memories or peripherals. Wait state requirements are a function of the device being addressed. Therefore, the address decoder must determine how many wait states, if any, to add to each bus cycle. The address decoder circuit (usually in conjunction with a shift register) must generate the READY# signal when it is time for the bus cycle to end. It is critical for the system to generate the READY# signal; if it does not, the 80386 will wait forever for the bus cycle to end.

EPROMs, static RAMs, and peripherals all interface in much the same way. The EPROM interface is the simplest because EPROMs are read-only devices. RAM interfaces must support byte addressability during RAM write cycles. Therefore, RAM write enables for each byte of the 32-bit data bus must be controlled separately.

The BS16# signal must be activated when the current bus cycle communicates over a 16-bit bus. An address decoder circuit can be used to determine if BS16# must be asserted during the current bus cycle.

11.5.3 Simplest Diagnostic Program

To start debugging 80386 hardware, the user should make a set of EPROMs containing a simple program, such as a 4-byte diagnostic that loops. Such a program is shown in Figure 11-11. Because the program is four bytes long, it will exercise all 32 bits of the data bus. This program tests only the code prefetch ability of the 80386.

In generating this program, the user should take into account the initial values of the 80386 CS register (F000H) and IP register (FFF0H) after reset. The software entry point (label START in Figure 11-11) must match the CS:IP location.

```
                    ASSUME CS:SIMPLEST_CODE

0000                SIMPLEST_CODE   SEGMENT
FFF0                                ORG OFFF0H

FFF0 90             START:          NOP
FFF1 90                             NOP
FFF2 EB FC                          JMP START

FFF4                SIMPLEST_CODE   ENDS
                                    END
```

Figure 11-11. 4-Byte Diagnostic Program

The 80386 is initially in Real Mode (the mode that emulates the 8086) after reset. With this simple diagnostic code, it will remain in Real Mode. In Real Mode, CS:IP generates the physical code fetch address directly, without any descriptors, by adding CS and IP in the following way:

$$\begin{array}{lr}
\text{(CS)} & \text{F000} \\
\text{(IP)} & +\ \text{FFF0} \\
\hline
 & \text{FFFF0}
\end{array}$$

Also, after reset (until the 80386 executes an intersegment JMP or CALL instruction), the physical base address of the code segment is set internally to FFFF0000H. Therefore, the physical address of the first code fetch after reset is always FFFFFFF0H. The simple diagnostic program must begin at this location.

11.5.4 Building and Debugging a System Incrementally

When designing an 80386 system, the designer plans the entire system. The core portions must be tested, however, before building the entire system. Beginning with only the 80386 and the 82384 Clock Generator, the following steps outline an approach that enables the designer to build up a system incrementally:

1. Install the 82384 and its crystal. Check that the CLK2 signal is clean. Connect the CLK2 signal to the 80386.
2. Connect the 82384 RESET output to the 80386 RESET input, and with CLK2 running, check that the state of the 80386 during RESET is correct.

3. Tie the 80386 INTR, NMI, and HOLD input pins low. Tie the READY# pin high so that the first bus cycle will not end. Reset the 80386, and check that the 80386 is emitting the correct signals to perform its first code fetch from physical address FFFFFFF0H. Connect the address latch, and verify that the address is driven at its outputs.
4. Connect the address decoding hardware to the 80386, and check that after reset, the 80386 is attempting to select the EPROM devices in which the initial code to be executed will be stored.
5. Connect the data transceiver to the system, and check that after reset, the transceiver control pins are being driven for a read cycle. Connect all address pins of the EPROM sockets, and check that after reset, they are receiving the correct address for the first code fetch cycle.

Intel's iPPS programmer for EPROMs supports dividing an object module into four EPROMs, as is necessary for a 32-bit data bus to EPROM. The programmer can also divide an object module into two EPROMs for a 16-bit data bus to the EPROMs. (In this case, the BS16# input to the 80386 must be asserted during all bus cycles communicating with the EPROMs).

When the 82384, crystal, 80386, address decoder, address latch, data transceiver, and READY# generation logic (including wait-state generation) are functioning, the 80386 is capable of running the software in the EPROMs. Now the simple debug program described above can be run to see whether the parts of the system work together.

After installing the EPROMs, the READY# line should be tied high (negated) so that the 80386 begins its first bus cycle after reset and then continues to add wait states. While the system is in this state, the circuit should be probed to verify signal states, using a voltmeter or oscilloscope probe.

The programmer should check whether the address latches have latched the first address and whether the address decoder is applying a chip-select signal to the EPROMs. The EPROMs should be emitting the first four opcode bytes of the first code to be executed (90H, 90H, EBH, FCH for the 4-byte program of Figure 11-11), and the opcode should be propagating through the data transceivers to the 80386 data pins.

Then the READY# input should be connected to the READY# generation logic, the 80386, and the results should be tested when the simple program runs. Because the program loops back on itself, it runs continuously. At this point, the system has progressed to running multiple bus cycles, so a logic analyzer is needed to observe the dynamic behavior of the system.

When the EPROMs programmed with the simple 4-byte diagnostic program are installed and the 80386 is executing the code, the LED indicator for ADS# (if included in the system) glows, because ADS# is generated for each bus cycle by the 80386. It is necessary to check that the EPROMs are selected for each code fetch cycle. After system operation is verified with the simple program, more complex programs can be run.

11.5.5 Other Simple Diagnostic Software

Other simple programs can be used to check the other operations the system must perform. The program described here is longer than the 4-byte program illustrated previously; it tests the abilities to write data into RAM and read the data back to the 80386.

This second diagnostic program, shown in Figure 11-12, is also suitable for placing into EPROMS. Because this diagnostic loops back to itself, the ADS# LED should glow continuously, just as it does when running the 4-byte program.

The program in Figure 11-12 is based on the assumption that hardware exists to report whether the data being read back from RAM is correct. This hardware consists of a writable output latch that can display a byte value written to it. The byte value written is a function of the RAM data comparison test. If the data is correct, the byte value written is AAH (10101010); if the data is incorrect, 55H (01010101) is written.

This diagnostic program is not comprehensive, but it does exercise EPROM, RAM, and an output latch to verify that the basic hardware works.

The program is short (45 bytes) to be easily understood. Because it is short and because it loops continuously, a logic analyzer or even an oscilloscope can be used to observe system activity.

This program can be written in ASM86 assembly language. Because the primary purpose of this program is to exercise the system hardware quickly, the 80386 is not tested extensively, and Protected Mode is not enabled.

The diagnostic software verifies the ability of the system to perform bus cycles. The 80386 fetches code from the EPROMs, implying that EPROM read cycles function correctly. Instructions in the program explicitly generate bus cycles to write and read RAM. The data value read back from RAM is checked for correctness, then a byte (AAH if the data is correct, 55H if it is not) is output to the 8-bit output latch. The program then loops back to its beginning and starts over.

After the source code is assembled, the resulting object code should be as shown in Figure 11-13.

11.5.6 Debugging Hints

The debugging approach described in this section is incremental; it lets the programmer debug the system piece by piece. If even the simple 4-byte program does not run, a logic analyzer can be used to determine where the problem is. At the very least, the 80386 should be initiating a code fetch cycle to EPROM.

```
                    PAGE    66,132
;
;         EQUATES
;
LATCH               EQU     0C8H              ;PRESUMES A HARDWARE
                                              ;LATCH IS AT I/O ADDR C8H
GOOD_SIGNAL         EQU     0AAH
BAD_SIGNAL          EQU     055H
;
;         CODE TO VERIFY ABILITY TO WRITE AND READ RAM CORRECTLY
;
                    ASSUME CS:INITIAL_CODE
INITIAL_CODE        SEGMENT

                    ORG     0F000H            ;THIS IS INTENDED TO BE LOCATED
                                              ;AT PHYSICAL ADDRESS FFFFF000H
TST_LOOP:           MOV     BX, 0000H         ;INITIALIZE BASE REGISTER TO 0
                    MOV     DS, BX            ;INITIALIZE DS REGISTER TO 0
                    MOV     [BX], 5473H       ;WRITE 5473H TO RAM ADDR 0 AND 1
                    MOV     [BX]+2, 2961H     ;WRITE 2961H TO RAM ADDR 2 AND 3
                    JMP     READ              ;JMP TO FORCE CPU TO BREAK
                                              ;PRE-FETCH QUEUE AND FETCH THE
                                              ;NEXT INSTRUCTION AGAIN.  THIS
                                              ;PREVENTS THE RAM DATA WRITTEN
                                              ;FROM JUST LINGERING ON THE DATA
                                              ;BUS UNTIL THE READ OCCURS
READ:               CMP     [BX], 5473H       ;READ DATA FROM RAM ADDR 0 AND 1
                                              ;AND COMPARE WITH VALUE WRITTEN
                    JNE     BADRAM            ;IF DATA DOESN´T MATCH, THEN JMP
                    CMP     [BX]+2, 2961H     ;READ DATA FROM RAM ADDR 2 AND 3
                                              ;AND COMPARE WITH VALUE WRITTEN
                    JNE     BADRAM            ;IF DATA DOESN´T MATCH, THEN JMP

                    MOV     AL, GOOD_SIGNAL
                    OUT     LATCH, AL         ;SIGNAL THAT DATA WAS CORRECT
                    JMP     TST_LOOP

BADRAM:             MOV     AL, BAD_SIGNAL
                    OUT     LATCH, AL         ;SIGNAL THAT DATA WAS BAD
                    JMP     TST_LOOP

                    ORG     0FFF0H            ;POSITION THE FOLLOWING INSTRUCTION
                                              ;AT OFFSET 0FFF0H
START:              JMP     TST_LOOP          ;INTRA-SEGMENT JUMP (WITHIN
                                              ;SEGMENT)
                                              ;THIS IS INTENDED TO BE THE FIRST
                                              ;INSTRUCTION EXECUTED, SO IT MUST
                                              ;BE LOCATED AT PHYSICAL ADDRESS
                                              ;FFFFFFF0H.
INITIAL_CODE        ENDS
                    END
```

Figure 11-12. More Complex Diagnostic Program

```
                                                 PAGE      66,132
                                   ;
                                   ;       EQUATES
                                   ;
= 00C8                             LATCH           EQU      0C8H
= 00AA                             GOOD_SIGNAL     EQU      0AAH
= 0055                             BAD_SIGNAL      EQU      055H

                                   ;
                                   ;       CODE TO VERIFY ABILITY TO WRITE
                                   ;       AND READ RAM CORRECTLY
                                   ;
                                                ASSUME CS:INITIAL_CODE
0000                               INITIAL_CODE SEGMENT

F000                                            ORG      0F000H

F000  BB 0000                      TST_LOOP:    MOV      BX, 0000H
F003  8E DB                                     MOV      DS, BX
F005  C7 07 5473                                MOV      [BX], 5473H
F009  C7 47 02 2961                             MOV      [BX]+2, 2961H
F00E  EB 01 90                                  JMP      READ

F011  81 3F 5473                   READ:        CMP      [BX], 5473H
F015  75 0D                                     JNE      BADRAM
F017  81 7F 02 2961                             CMP      [BX]+2, 2961H
F01C  75 06                                     JNE      BADRAM

F01E  B0 AA                                     MOV      AL, GOOD_SIGNAL
F020  E6 C8                                     OUT      LATCH, AL
F022  EB DC                                     JMP      TST_LOOP

F024  B0 55                        BADRAM:      MOV      AL, BAD_SIGNAL
F026  E6 C8                                     OUT      LATCH, AL
F028  EB D6                                     JMP      TST_LOOP

FFF0                                            ORG      0FFF0H
FFF0  E9 F000 R                    START:       JMP      TST_LOOP

FFF3                               INITIAL_CODE ENDS
                                                END

Warning Severe
Errors  Errors
0       0
```

Figure 11-13. Object Code for Diagnostic Program

The 80386 stops running only for one of three reasons:

- The READY# signal is never asserted to terminate the bus cycle.
- The HALT instruction is encountered, so the 80386 enters a HALT state.
- The 80386 encounters a shutdown condition. In Real Mode operation (as in the simple diagnostic program), a shutdown usually indicates that the 80386 is reading garbage on the data bus.

If the 80386 stops running, the cause can be determined easily if the system contains simple hardware decoders with associated LEDs to visually indicate halt and shutdown conditions. The 80386 emits specific codes on its W/R#, D/C#, M/IO#, and address outputs to indicate halt or shutdown. A circuit to decode these signals can be tested by executing a HLT instruction (F4H) to see if the halt LED is turned on. The shutdown LED cannot be tested in the same way, but its decoder is so similar to the halt decoder that if the halt decoder works, the shutdown decoder should also work.

If the shutdown LED comes on and the 80386 stops running, the data being read in during code fetch cycles is garbled. The programmer should check the EPROM contents, the wiring of the address path and data path, and the data transceivers. The 4-byte diagnostic program should be used to investigate the system. This program should work before more complex software is used.

If neither the halt LED nor the shutdown LED is on when the 80386 stops running, the READY# generation circuit has not activated READY# to complete the bus cycle. The 80386 is adding wait states to the cycle, waiting for the READY# signal to go active. The address at the address latch outputs and the states of the W/R#, D/C#, and M/IO# signals should be checked to narrow the investigation to a specific part of the READY# generation circuit. Then the circuit should be investigated with the logic analyzer.

Once the basic system is built and debugged, more software and further enhancements can be added to the system. The incremental approach described applies to these additions. Systematic, step-by-step testing and debugging is the surest way to build a reliable 80386-based system.

Test Capabilities

12

CHAPTER 12
TEST CAPABILITIES

The 80386 contains built-in features that enhance its testability. These features are derived from signature analysis, and proprietary test techniques. All the regular logic blocks of the 80386, or about half of all its internal devices, can be tested using these built-in features.

The 80386 testability features include aids for both internal and board-level testing. This chapter describes these features.

12.1 INTERNAL TESTS

Allowances have been made for two types of internal tests: automatic self-test and Translation Lookaside Buffer (TLB) tests. The automatic self-test is controlled completely by the 80386. The designer needs only to initiate the test and check the results. The TLB tests must be externally developed and applied. The 80386 provides an interface that makes this test development simple.

12.1.1 Automatic Self-Test

The 80386 can automatically verify the functionality of its three major Programmable Logic Arrays (PLAs) (the Entry Point, Control, and Test PLAs) and the contents of its Control ROM (CROM). The automatic self-test is initiated by setting the BUSY# input active during initialization (as described in Chapter 3). The test result is stored in the EAX register of the 80386.

The self-test progresses as follows (see Figure 12-1):

1. Normal PLA or CROM inputs are disabled.
2. A pseudo-random count sequence, generated by an internal Linear Feedback Shift register (LFSR), provides all possible combinations of PLA and CROM inputs.
3. PLA and CROM outputs for each input combinations are directed to a parallel-load LFSR.
4. Through the action of this LFSR, a signature of all output results is accumulated.
5. After all input combinations have been sequenced, the final contents of the LFSR are XORed with a signature constant stored in the 80386. If the LFSR contents match the signature constant, the result will be all zeroes, indicating functional PLA and CROM.
6. The result is loaded into the EAX register.

The self-test provides 100-percent coverage of single-bit faults, which statistically comprise a high percentage of total faults.

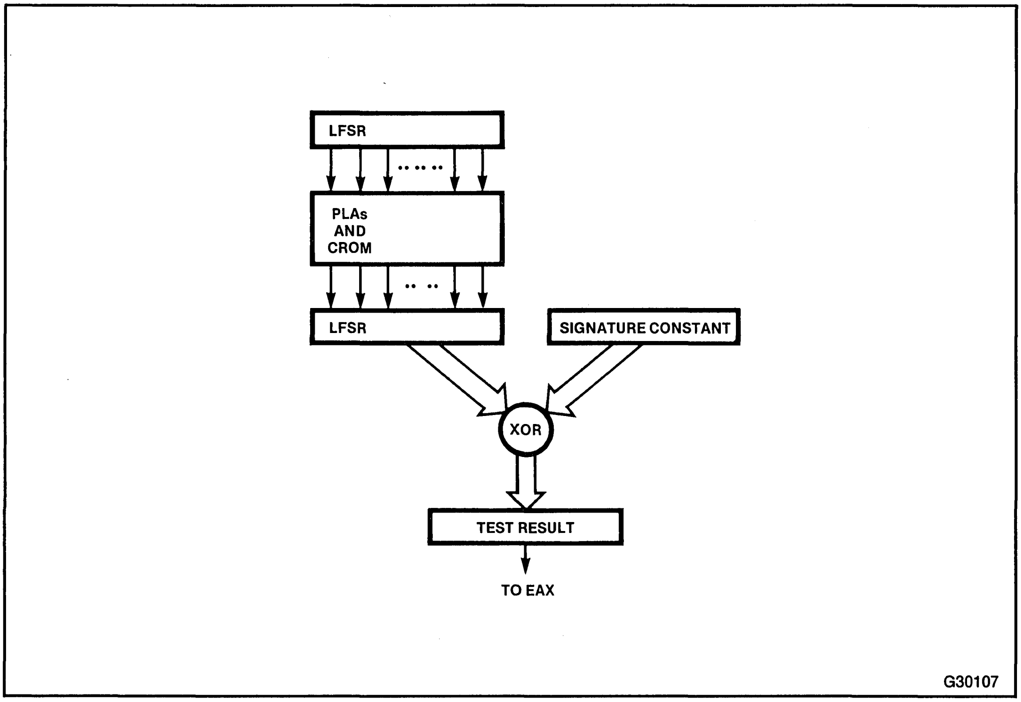


Figure 12-1. 80386 Self-Test

12.1.2 Translation Lookaside Buffer Tests

The on-chip Page Descriptor Cache of the 80386 stores its data in the TLB. (Cache operation is discussed fully in Chapter 7.) The linear-to-physical mapping values for the most recent memory accesses are stored in the TLB, thus allowing fast translation for subsequent accesses to those locations. The TLB consists of:

- Content-addressable memory (CAM)—holds 32 linear addresses (Page Directory and Page Table fields only) and associated tag bits (used for data protection and cache implementation)
- Random access memory (RAM)—holds the 32 physical addresses (upper 20 bits only) that correspond to the linear addresses in the CAM
- Logic-implements the four-way cache and includes a 2-bit replacement pointer that determines to which of the four sets a new entry is directed during a write to the TLB.

To translate a linear address to a physical address, the 83086 tries to match the Page Directory and Page Table fields of the linear address with an entry in the CAM. If a hit (a match) occurs, the corresponding 20 bits of physical address are retrieved from the RAM and added to the 12 bits of the Offset field of the linear address, creating a 32-bit physical address. If a miss (no match) occurs, the 80386 must bring the Page Directory and Page Table values into the TLB from memory.

The 80386 provides an interface through which to test the TLB. Two 32-bit test registers of the 80386 are used to write and read the contents of the TLB through the MOV TREG, reg and MOV reg, TREG instructions. An 80386 program can be used to generate test patterns which are applied to the TLB through automatic test machines or assembly language programs.

The paging mechanism of the 80386 must be disabled during a test of the TLB. The internal response is therefore not identical to that of normal operation, but the main functionality of the TLB can be verified.

Test register #6 is used as the command register for TLB accesses; test register #7 is used as the data register. Addresses and commands are written to the TLB through the command register. Data is read from or written to the TLB through the data register.

The two test operations that may be performed on the TLB are:

- Write the physical address contained in the data register and the linear address and tag bits contained in the command register into a TLB location designed by the data register.
- Look up a TLB entry using the linear address and tag bits contained in the command register. If a hit occurs, copy the corresponding physical address into the data register, and set the value of the hit/miss bit in the data register. If a miss occurs, clear the /hit/miss bit. In this case, the physical address in the data register is undefined.

A command is initiated by writing to the command register. The command register has the format shown in Figure 12-2 (top). The two possible commands are distinguished by the

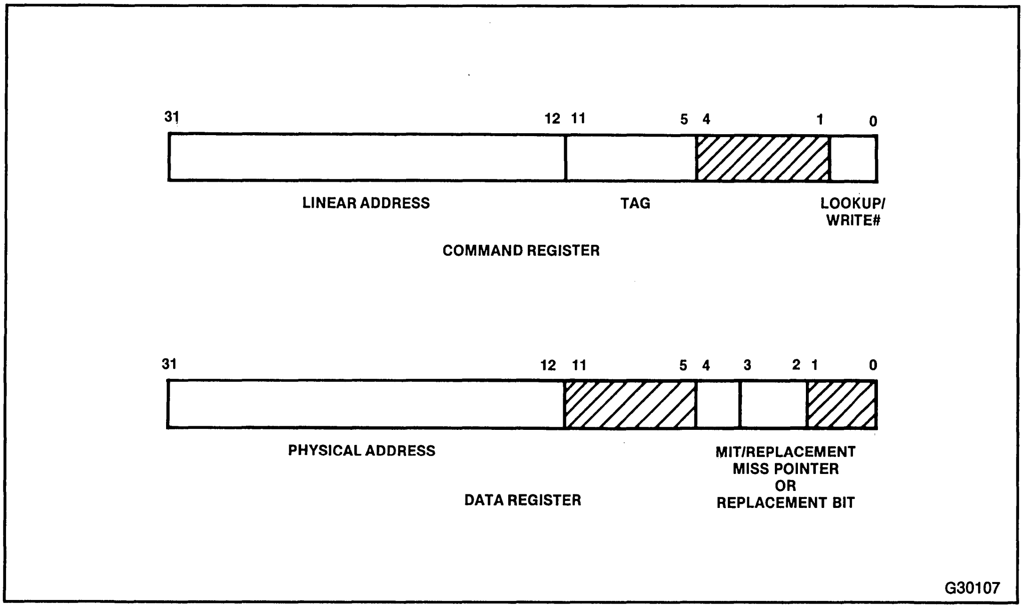


Figure 12-2. TLB Test Registers

state of bit 0 in the command register. If bit 0 = 1, a TLB lookup operation is performed. If bit 0 = 0, a TLB write is performed.

The tag bits (not including the linear address) consist of the following:

Bit	Name	Definition
11	Valid (V)	Entry is valid
10	Dirty (D)	Entry has been changed
9	Not Dirty (D#)	Entry has not been changed
8	User (U)	Entry is accessible to User privilege level
7	Not User (U#)	Entry is not accessible to User privilege level
6	Writable (W)	Entry may be changed
5	Not Writable (W#)	Entry may not be changed

The complement of the Dirty, User, and Writable bits are provided to force a hit or miss for TLB lookups. A lookup operation with a bit and its complement both low is forced to be a miss; if both bits are high, a hit is forced. A write operation must always be performed with a bit and its complement bit having opposite values.

The data register has the format shown in Figure 12-2 (bottom). The replacement pointer indicates which of the four sets of the TLB is to receive write data. Its value is changed according to a proprietary algorithm after every TLB hit. For testing, a TLB write may use the replacement pointer value that exists in the TLB, or it may use the value in bits 3 and 2 of the data register. If data register bit 4 = 0, the existing replacement pointer is used. If bit 4 = 1, bits 3 and 2 of the data register are used.

The TLB write operation progresses as follows:

1. The physical address, replacement bit, and replacement pointer value (optional) are written to the data register.
2. The linear address and tag values are written to the command register, as well as a 0 value for bit 0.

It is important not to write the same linear address to more than one TLB entry. Otherwise, hit information returned during a TLB lookup operation is undefined.

The TLB lookup operation progresses as follows:

- The linear address and tag values are written to the command register, as well as a 1 value for bit 0.
- New values for the hit/miss bit and replacement pointer are written to bits 4-2 in the data register. If the hit/miss bit (bit 4) is 1, bits 31-12 contain the physical address from the TLB. Otherwise, bits 31-12 are undefined.

For more information on how to write routines to test the TLB, refer to the *80386 Programmer's Reference Manual.*

12.2 BOARD-LEVEL TESTS

For board-level testing, it is often desirable to isolate areas of the board from the interactions of other devices. The 80386 can be forced to a state in which all but two of its pins are effectively removed from their circuits. This state is accomplished through the HOLD and HLDA pins.

When the HOLD input of the 80386 is asserted, the 80386 places all of its outputs except for HLDA in the three-state condition. HLDA is then driven high. The 80386 remains in this condition until HOLD is de-asserted. Note that RESET being asserted takes priority over HOLD requests.

The 80386 completes its current bus cycle before responding to the HOLD input. Detailed information on HOLD/HLDA response is given in Chapter 3.

Appendix A
Local Bus Control PAL Descriptions

APPENDIX A
LOCAL BUS CONTROL PAL DESCRIPTIONS

The bus controller is implemented in two PALs. One PAL (called PAL-1) follows the 80386 bus cycles and generates the overall bus cycle timings. The second PAL (PAL-2) generates most of the bus control signals.

The PALs are clocked by CLK2. They could also be clocked by CLK. Using CLK2 has the following advantages over using CLK:

- The skew from clock to command signal is reduced, so higher performance is possible with slower devices.
- The 80386 ADS# and READY# signals can be sampled directly. If CLK were used, it would be possible to sample ADSO# from the 82384 instead of ADS#, but the READY# generation logic would be more complex because READY# must be synchronized to CLK2.
- The PAL can provide delays in 31-nanosecond, rather than 62-nanosecond, increments.

The advantages of using CLK to clock the PALs are as follows:

- A slower PAL device could be used.
- One PAL input is saved because only CLK, rather than CLK and CLK2, is needed.

Because CLK2 is used to clock the PALs, the choice of PALs is currently limited to only 20-pin B-series PALs.

PAL-1 FUNCTIONS

PAL-1 is implemented as two main state machines. The BUSSTATE state machine, which is used to follow the 80386 bus cycles, is specified by the state of two signals, IDLE and PIPE, and follows the 80386 bus by sampling ADS# and READY#. IDLE and PIPE are often useful in implementing other 80386 subsystems (such as the DRAM controller described later in this book).

The LOCALSTATE state machine keeps track of the local bus state and is specified by signals L2, L1, and L0. Although the local bus state usually follows the 80386 bus state, so that the LOCALSTATE and BUSSTATE states are the same, there are times when the local bus cycle lags the 80386 bus cycle in order to handle data-float and peripheral recovery times correctly. Therefore, it is easier to implement this PAL using the two separate state machines. LOCALSTATE uses the 80386 W/R# signal and various chip-select inputs to determine what type of cycle to run.

A third, simpler state machine is also implemented in PAL-1. Q1 and Q0 comprise a SEQUENCE counter that is used to implement the various time delays required by the local bus state machine.

The NA# output of PAL-1 activates address pipelining for the I/O and 1-wait-state devices. For 0-wait-state devices, external logic generates NA#, because these devices require NA# sooner than PAL-1 can generate it.

PAL-2 FUNCTIONS

PAL-2 generates most bus control signals, including all five command signals, the READY# signal, and the latch and transceiver enable signals. PAL-2 inputs the three LOCALSTATE signals from PAL-1 and the three 80386 bus cycle definition pins (MIO#, DC#, and WR#) in order to follow the local bus state. PAL-2 also inputs the 0-wait-state chip-select signal in order to set output signals quickly enough for zero wait states.

Note that the transceiver direction enable (DT/R#) is simply a latched version of W/R#. This saves a PAL output and also guarantees that the transceiver direction does not change while DEN# is enabled.

PAL EQUATIONS

The equations for PAL-1 and PAL-2 are shown in Figures A-1 and A-2, respectively. These equations are shown in a high-level PAL language (ABEL, by Data I/O) that allows the PAL to be described as a series of states rather than equations. This language frees the designer of the tedious task of implementing the state machine and reducing the logical equations manually. The language saves time not only in the initial design, but also in debugging the state machines. The automated term reduction of the high-level PAL language allows the designer to explore many implementations quickly, which is a useful feature for complex PAL designs.

Figures A-3 and A-4 show the PAL equations for PAL-1 and PAL-2 using PALASM by Monolithic Memories.

```
module     Bus_Control_386_Pal_1  flag '-r3'

title
'80386 Local Bus Controller - pal 1    Intel Corp'

    BC386P1     device  'P16R8';        "use a 16R8 B-speed PAL for 16MHz 386

" Constants:

    ON          =     1;
    OFF         =     0;
    H           =     1;
    L           =     0;
    x           =     .X.;              " ABEL 'don't care' symbol
    c           =     .C.;              " ABEL 'clocking input' symbol

" State definitions for BUSSTATE (bus cycle state):

    IDLEBUS     =     ^b01;         "bus is idle or first CLK of unpipelined
    PIPEBUS     =     ^b10;         "first CLK of pipelined cycle
    ACTIVEBUS   =     ^b00;         "subsequent CLKs of active bus
    ILLEGALBUS  =     ^b11;         "unused

" State definitions for LOCALSTATE (local cycle state):

    WAITING     =     ^b101;        "waiting for next bus cycle
    SAMPLECS    =     ^b100;        "CLK2 before ALE falls and CS is sampled
    CMDDELAY    =     ^b000;        "delay before CMD active
    IO          =     ^b010;        "IO CMD active
    ENDIO       =     ^b110;        "IO CMD inactive
    MEMORY      =     ^b011;        "1WS CMD active
    FLOAT       =     ^b111;        "data bus float delay
    NOTLOCAL    =     ^b001;        "0WS cycle or bus cycle not to the local bus

" State definitions for SEQUENCE (local cycle sequence counter):

    SEQ0        =     ^b00;
    SEQ1        =     ^b01;
    SEQ2        =     ^b11;
    SEQ3        =     ^b10;


" Pin names:
                               " Input pins
    CLK         pin   2;           " 82384 CLK
    ADS         pin   3;           " 80386 ADS#
    READY       pin   4;           " 80386 READY#
    WR          pin   5;           " 80386 W/R#
    CS0WS       pin   6;           " chip-select for 0 wait-state (piped) devices
    CS1WS       pin   7;           " chip-select for 1 wait-state (piped) devices
    CSIO        pin   8;           " chip-select for peripheral devices
    RESET       pin   9;           " 80386/82384 RESET

    CLK2        pin   1;       " Clock pin - 82384 CLK2
    OE          pin   11;      " Output Enable pin

                               " Output pins
    NA          pin   19;          " NEXT ADDRESS (NA) for 1WS and IO devices
    IDLE        pin   18;          " bus state: IDLE or first CLK of unpiped cycle
    PIPE        pin   17;          " bus state: first CLK of pipelined cycle
    L2          pin   16;          " local cycle state
    L1          pin   15;          " local cycle state
    L0          pin   14;          " local cycle state
    Q1          pin   13;          " local cycle sequence counter
    Q0          pin   12;          " local cycle sequence counter

    BUSSTATE    =     [PIPE, IDLE];     " bus cycle state
    LOCALSTATE  =     [L2, L1, L0];     " local cycle state
    SEQUENCE    =     [Q1, Q0];         " local cycle sequence counter
```

Figure A-1. PAL-1 State Listings

```
" Macros:

    COUNTING    macro
        { (Q1 # Q0) };          " true until sequencer counts down to zero

    LOWCOUNTING macro
        { (     Q0) };          " true until sequencer counts down to zero
                                  " same as COUNTING except used to reduce PAL
                                  " terms and only works if count less than 3

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

state_diagram  BUSSTATE

  state IDLEBUS:                        "bus is idle or first CLK of unpipelined
    if RESET then IDLEBUS                  "reset to IDLEBUS
      else if !CLK # ADS then IDLEBUS      "remain til sample ADS active
                         else ACTIVEBUS;   "...then go ACTIVE

  state ACTIVEBUS:                      "subsequent CLKs of active bus
    if RESET then IDLEBUS                  "reset to IDLEBUS
      else if !CLK # READY then ACTIVEBUS  "remain til sample READY active
        else if !ADS then PIPEBUS          "...then next cycle either piped
                     else IDLEBUS;         "...or idle

  state PIPEBUS:                        "first CLK of pipelined cycle
    if RESET then IDLEBUS                  "reset to IDLEBUS
      else if !CLK then PIPEBUS              "remain for just one CLK
                   else ACTIVEBUS;           "...then go ACTIVE

    state ILLEGALBUS:                     "unused state - should never occur
      goto IDLEBUS;                          "if entered upon power-up...go IDLE

  """""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

  state_diagram  LOCALSTATE

    state WAITING:                        "waiting for next bus cycle
      NA   := OFF;
      if RESET then WAITING                  "reset to WAITING
       else if  !CLK
             #( ADS & IDLE)                        "remain while idle bus
             #( !CSIO & LOWCOUNTING) then WAITING  "..and remain for recovery time
                                     else SAMPLECS;"..else begin bus cycle

    state SAMPLECS:                       "CLK2 before ALE falls and CS is sampled
      NA   := OFF;
      if RESET then WAITING                  "reset to WAITING
        else if !CS1WS then MEMORY           "start 1WS access
          else if !CSIO  then CMDDELAY       "start IO access
                         else NOTLOCAL;      "start non-local access

    state CMDDELAY:                       "delay before CMD active
      NA   := OFF;
      if RESET then WAITING                  "reset to WAITING
               else IO;                      "only in state for 1 CLK2, then IO

    state IO:                             "IO CMD active
      NA   := (!COUNTING & CLK) # NA;        "activate NA after count down to zero
      if RESET then WAITING                  "reset to WAITING
        else if !NA then IO                  "remain until NA active
                    else ENDIO;              "...then ENDIO

    state ENDIO:                          "IO CMD active
      NA   := OFF;
      if RESET then WAITING                  "reset to WAITING
      else if LOWCOUNTING then ENDIO         "remain while COUNTING down
                          else FLOAT;        "...then FLOAT
```

Figure A-1. PAL-1 State Listings (Cont'd.)

```
  state MEMORY:                         "1WS CMD active
    NA    := (!NA & CLK) # (NA & !CLK);    "activate NA on second and third CLK2
    if RESET then WAITING                  "reset to WAITING
      else if LOWCOUNTING then MEMORY      "remain while COUNTING down
                          else FLOAT;      "...then FLOAT

  state FLOAT:                          "data bus float delay
    NA    := OFF;
    if RESET then WAITING                  "reset to WAITING
      else if !IDLE & !CLK & CS0WS & CS1WS & CSIO then NOTLOCAL
                                         "watch for non-local bus cycle to start
        else if COUNTING then FLOAT        "...else remain while COUNTING down
                         else WAITING;     "...then WAIT

  state NOTLOCAL:                   "0WS cycle or bus cycle not to the local bus
    NA    := OFF;
    if RESET then WAITING                  "reset to WAITING
      else if READY # !CLK then NOTLOCAL   "remain until bus cycle ends
        else if COUNTING then FLOAT      "...then finish FLOAT if still COUNTING
          else if !ADS then SAMPLECS       "...else SAMPLECS if next bus piped
                       else WAITING;       "...else WAIT

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

state_diagram  SEQUENCE     "counter for LOCALSTATE

  state SEQ3:
    if RESET then SEQ0                     "reset to SEQuence 0
      else if !CLK then SEQ3               "no change if CLK low
                   else SEQ2;              "count down every time CLK high

  state SEQ2:
    if RESET then SEQ0
      else if !CLK then SEQ2
                   else SEQ1;

  state SEQ1:
    if RESET then SEQ0
      else if !CLK then SEQ1
                   else SEQ0;

  state SEQ0:   "once count all the way down, figure out what to count next
    case
       RESET                                     : SEQ0; "reset to SEQuence 0
      !RESET & (LOCALSTATE == WAITING)           : SEQ0; "count not used
      !RESET & (LOCALSTATE == SAMPLECS) & !CS1WS : SEQ2; "set up for 1WS MEMORY
      !RESET & (LOCALSTATE == SAMPLECS) &  CS1WS : SEQ0; "other than 1WS MEMORY
      !RESET & (LOCALSTATE == CMDDELAY) &  WR    : SEQ3; "set up for IO write
      !RESET & (LOCALSTATE == CMDDELAY) & !WR    : SEQ2; "set up for IO read
      !RESET & (LOCALSTATE == IO)       & !NA    : SEQ0; "still in IO...remain
      !RESET & (LOCALSTATE == IO)       &  NA    : SEQ1; "set up for ENDIO
      !RESET & (LOCALSTATE == ENDIO)             : SEQ3; "set up for IO float
      !RESET & (LOCALSTATE == MEMORY)            : SEQ1; "set up for 1WS MEMORY
      !RESET & (LOCALSTATE == FLOAT)             : SEQ1; "set up for IO recovery
      !RESET & (LOCALSTATE == NOTLOCAL)          : SEQ0; "count not used
    endcase;
```

Figure A-1. PAL-1 State Listings (Cont'd.)

```
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

test_vectors    ( [CLK2, CLK, RESET, ADS, READY, WR, CS0WS, CS1WS, CSIO] ->
                  [LOCALSTATE, NA] )

            "[inputs] -> [outputs]

"C C  R  A R  W  C C C               N
"L L  E  D E  R  S S S               A
"K K  S  S A     0 1 I     LOCALSTATE
"2    E    D     W W O
"     T    Y     S S

[c,x, H, x,x, x, x,x,x] -> [ WAITING, L];"initialize to IDLE and WAITING state
[c,L, L, H,H, x, x,x,x] -> [ WAITING, L];
[c,H, L, H,H, x, x,x,x] -> [ WAITING, L];
[c,L, L, x,H, x, x,x,x] -> [ WAITING, L];
[c,H, L, L,H, x, x,x,x] -> [SAMPLECS, L];"100nS SRAM read
[c,L, L, x,H, L, L,H,H] -> [NOTLOCAL, L];"NA: activated externally
[c,H, L, H,H, x, x,x,x] -> [NOTLOCAL, L];
[c,L, L, x,L, x, x,x,x] -> [NOTLOCAL, L];
[c,H, L, L,L, x, x,x,x] -> [SAMPLECS, L];"100nS SRAM read
[c,L, L, x,H, L, L,H,H] -> [NOTLOCAL, L];"NA: activated externally
[c,H, L, H,H, x, x,x,x] -> [NOTLOCAL, L];
[c,L, L, x,L, x, x,x,x] -> [NOTLOCAL, L];
[c,H, L, L,L, x, x,x,x] -> [SAMPLECS, L];"100nS SRAM write
[c,L, L, x,H, H, L,H,H] -> [NOTLOCAL, L];"NA: activated externally
[c,H, L, H,H, x, x,x,x] -> [NOTLOCAL, L];
[c,L, L, x,H, x, x,x,x] -> [NOTLOCAL, L];
[c,H, L, x,H, x, x,x,x] -> [NOTLOCAL, L];
[c,L, L, x,L, x, x,x,x] -> [NOTLOCAL, L];
[c,H, L, L,L, x, x,x,x] -> [SAMPLECS, L];"100nS SRAM read
[c,L, L, x,H, L, L,H,H] -> [NOTLOCAL, L];"NA: activated externally
[c,H, L, H,H, x, x,x,x] -> [NOTLOCAL, L];
[c,L, L, x,L, x, x,x,x] -> [NOTLOCAL, L];
[c,H, L, L,L, x, x,x,x] -> [SAMPLECS, L];"non-local
[c,L, L, x,H, x, H,H,H] -> [NOTLOCAL, L];
[c,H, L, H,H, x, x,x,x] -> [NOTLOCAL, L];
[c,L, L, x,L, x, x,x,x] -> [NOTLOCAL, L];
[c,H, L, L,L, x, x,x,x] -> [SAMPLECS, L];"100nS SRAM read
[c,L, L, x,H, x, L,H,H] -> [NOTLOCAL, L];"NA: activated externally
[c,H, L, H,H, x, x,x,x] -> [NOTLOCAL, L];
[c,L, L, H,L, x, x,x,x] -> [NOTLOCAL, L];
[c,H, L, H,L, x, x,x,x] -> [ WAITING, L];"idle
[c,L, L, H,H, x, x,x,x] -> [ WAITING, L];
[c,H, L, H,H, x, x,x,x] -> [ WAITING, L];
[c,L, L, x,H, x, x,x,x] -> [ WAITING, L];
[c,H, L, L,H, x, x,x,x] -> [SAMPLECS, L];"100nS SRAM write
[c,L, L, x,H, H, L,H,H] -> [NOTLOCAL, L];"NA: activated externally
[c,H, L, H,H, x, x,x,x] -> [NOTLOCAL, L];
[c,L, L, x,H, x, x,x,x] -> [NOTLOCAL, L];
[c,H, L, H,H, x, x,x,x] -> [NOTLOCAL, L];
[c,L, L, x,L, x, x,x,x] -> [NOTLOCAL, L];
[c,H, L, H,L, x, x,x,x] -> [ WAITING, L];"idle

[c,L, L, H,H, x, x,x,x] -> [ WAITING, L];
[c,H, L, H,H, x, x,x,x] -> [ WAITING, L];
[c,L, L, x,H, x, x,x,x] -> [ WAITING, L];
[c,H, L, L,H, x, x,x,x] -> [SAMPLECS, L];"150nS ROM read
[c,L, L, x,H, L, H,L,H] -> [ MEMORY , L];
[c,H, L, H,H, x, x,x,x] -> [ MEMORY , H];
[c,L, L, H,H, x, x,x,x] -> [ MEMORY , H];
[c,H, L, H,H, x, x,x,x] -> [ MEMORY , L];
[c,L, L, x,L, x, x,x,x] -> [  FLOAT , L];
[c,H, L, L,L, x, x,x,x] -> [  FLOAT , L];
[c,L, L, x,H, x, L,x,x] -> [ WAITING, L];
[c,H, L, H,H, x, x,x,H] -> [SAMPLECS, L];"100nS SRAM read
[c,L, L, x,H, L, L,H,H] -> [NOTLOCAL, L];"NA: activated externally
[c,H, L, x,H, x, x,x,x] -> [NOTLOCAL, L];
```

Figure A-1. PAL-1 State Listings (Cont'd.)

```
[c,L, L, x,L, x, x,x,x] -> [NOTLOCAL, L];
[c,H, L, L,L, x, x,x,x] -> [SAMPLECS, L];"150nS ROM read
[c,L, L, x,H, L, H,L,H] -> [ MEMORY , L];
[c,H, L, H,H, x, x,x,x] -> [ MEMORY , H];
[c,L, L, H,H, x, x,x,x] -> [ MEMORY , H];
[c,H, L, H,H, x, x,x,x] -> [ MEMORY , L];
[c,L, L, x,L, x, x,x,x] -> [  FLOAT , L];
[c,H, L, L,L, x, x,x,x] -> [  FLOAT , L];
[c,L, L, x,H, x, x,L,x] -> [ WAITING, L];
[c,H, L, H,H, x, x,x,H] -> [SAMPLECS, L];"150nS SRAM write
[c,L, L, H,H, H, H,L,H] -> [ MEMORY , L];
[c,H, L, H,H, x, x,x,x] -> [ MEMORY , H];
[c,L, L, H,H, x, x,x,x] -> [ MEMORY , H];
[c,H, L, H,H, x, x,x,x] -> [ MEMORY , L];
[c,L, L, x,L, x, x,x,x] -> [  FLOAT , L];
[c,H, L, L,L, x, x,x,x] -> [  FLOAT , L];
[c,L, L, x,H, x, x,L,x] -> [ WAITING, L];
[c,H, L, H,H, x, x,x,H] -> [SAMPLECS, L];"150nS SRAM read
[c,L, L, H,H, L, H,L,H] -> [ MEMORY , L];
[c,H, L, H,H, x, x,x,x] -> [ MEMORY , H];
[c,L, L, H,H, x, x,x,x] -> [ MEMORY , H];
[c,H, L, H,H, x, x,x,x] -> [ MEMORY , L];
[c,L, L, H,L, x, x,x,x] -> [  FLOAT , L];
[c,H, L, H,L, x, x,x,x] -> [  FLOAT , L];
[c,L, L, H,H, x, x,x,x] -> [ WAITING, L];"idle
[c,H, L, H,H, x, x,x,x] -> [ WAITING, L];
[c,L, L, x,H, x, x,x,x] -> [ WAITING, L];
[c,H, L, L,H, x, x,x,x] -> [SAMPLECS, L];"150nS SRAM read
[c,L, L, x,H, L, H,L,H] -> [ MEMORY , L];
[c,H, L, H,H, x, x,x,x] -> [ MEMORY , H];
[c,L, L, x,H, x, x,x,x] -> [ MEMORY , H];
[c,H, L, L,H, x, x,x,x] -> [ MEMORY , L];
[c,L, L, L,L, x, x,x,x] -> [  FLOAT , L];
[c,H, L, H,L, x, x,x,x] -> [  FLOAT , L];
[c,L, L, H,H, x, x,x,x] -> [ WAITING, L];"idle
[c,H, L, H,H, x, x,x,x] -> [ WAITING, L];

[c,L, L, x,H, x, x,x,x] -> [WAITING , L];
[c,H, L, L,H, x, x,x,x] -> [SAMPLECS, L];"peripheral read
[c,L, L, x,H, L, H,H,L] -> [CMDDELAY, L];
[c,H, L, H,H, L, x,x,x] -> [   IO   , L];
[c,L, L, H,H, x, x,x,x] -> [   IO   , L];
[c,H, L, H,H, x, x,x,x] -> [   IO   , L];
[c,L, L, H,H, x, x,x,x] -> [   IO   , L];
[c,H, L, H,H, x, x,x,x] -> [   IO   , L];
[c,L, L, H,H, x, x,x,x] -> [   IO   , L];
[c,H, L, H,H, x, x,x,x] -> [   IO   , H];
[c,L, L, H,H, x, x,x,x] -> [ ENDIO  , H];
[c,H, L, H,H, x, x,x,x] -> [ ENDIO  , L];
[c,L, L, x,L, x, x,x,x] -> [ FLOAT  , L];
[c,H, L, L,L, x, x,x,x] -> [ FLOAT  , L];
[c,L, L, x,H, x, x,x,L] -> [ FLOAT  , L];
[c,H, L, H,H, x, x,x,x] -> [ FLOAT  , L];
[c,L, L, H,H, x, x,x,L] -> [ FLOAT  , L];
[c,H, L, H,H, x, x,x,x] -> [ FLOAT  , L];
[c,L, L, H,H, x, x,x,L] -> [WAITING , L];
[c,H, L, H,H, x, x,x,L] -> [WAITING , L];
[c,L, L, H,H, x, x,x,L] -> [WAITING , L];
[c,H, L, H,H, x, x,x,x] -> [SAMPLECS, L];"peripheral read
[c,L, L, H,H, L, H,H,L] -> [CMDDELAY, L];
[c,H, L, H,H, L, x,x,x] -> [   IO   , L];
[c,L, L, H,H, x, x,x,x] -> [   IO   , L];
[c,H, L, H,H, x, x,x,x] -> [   IO   , L];
[c,L, L, H,H, x, x,x,x] -> [   IO   , L];
[c,H, L, H,H, x, x,x,x] -> [   IO   , L];
[c,L, L, H,H, x, x,x,x] -> [   IO   , L];
```

Figure A-1. PAL-1 State Listings (Cont'd.)

```
[c,H, L, H,H, x, x,x,x] -> [   IO   , H];
[c,L, L, H,H, x, x,x,x] -> [ ENDIO  , H];
[c,H, L, H,H, x, x,x,x] -> [ ENDIO  , L];
[c,L, L, x,L, x, x,x,x] -> [ FLOAT  , L];
[c,H, L, L,L, x, x,x,x] -> [ FLOAT  , L];
[c,L, L, x,H, x, L,x,x] -> [ FLOAT  , L];
[c,H, L, H,H, x, x,x,x] -> [ FLOAT  , L];
[c,L, L, H,H, x, L,x,x] -> [ FLOAT  , L];
[c,H, L, H,H, x, x,x,x] -> [ FLOAT  , L];
[c,L, L, H,H, x, L,x,x] -> [WAITING , L];
[c,H, L, H,H, x, x,x,H] -> [SAMPLECS, L];"100nS SRAM read
[c,L, L, H,H, L, L,H,H] -> [NOTLOCAL, L];"NA: activated externally
[c,H, L, H,H, x, x,x,x] -> [NOTLOCAL, L];
[c,L, L, x,L, x, x,x,x] -> [NOTLOCAL, L];
[c,H, L, L,L, x, x,x,x] -> [SAMPLECS, L];"peripheral write
[c,L, L, x,H, H, H,H,L] -> [CMDDELAY, L];
[c,H, L, H,H, H, x,x,x] -> [   IO   , L];
[c,L, L, H,H, x, x,x,x] -> [   IO   , L];
[c,H, L, H,H, x, x,x,x] -> [   IO   , L];
[c,L, L, H,H, x, x,x,x] -> [   IO   , L];
[c,H, L, H,H, x, x,x,x] -> [   IO   , L];
[c,L, L, H,H, x, x,x,x] -> [   IO   , L];
[c,H, L, H,H, x, x,x,x] -> [   IO   , L];
[c,L, L, H,H, x, x,x,x] -> [   IO   , L];
[c,H, L, H,H, x, x,x,x] -> [   IO   , H];
[c,L, L, H,H, x, x,x,x] -> [ ENDIO  , H];
[c,H, L, H,H, x, x,x,x] -> [ ENDIO  , L];
[c,L, L, x,L, x, x,x,x] -> [ FLOAT  , L];
[c,H, L, x,L, x, x,x,x] -> [ FLOAT  , L];

end Bus_Control_386_Pal_1;
```

Figure A-1. PAL-1 State Listings (Cont'd.)

```
module     Bus_Control_386_Pal_2  flag '-r3'

title
'80386 Local Bus Controller - pal 2    Intel Corp'

    BC386P2     device  'P16R8';        "use a 16R8 B-speed PAL for 16MHz 386

" Constants:

    ON          =     1;
    OFF         =     0;
    H           =     1;
    L           =     0;
    x           =     .X.;              " ABEL 'don't care' symbol
    c           =     .C.;              " ABEL 'clocking input' symbol

" State definitions for LOCALSTATE (local cycle state):

    WAITING     =     ^b101;        "waiting for next bus cycle
    SAMPLECS    =     ^b100;        "CLK2 before ALE falls and CS is sampled
    CMDDELAY    =     ^b000;        "delay before CMD active
    IO          =     ^b010;        "IO CMD active
    ENDIO       =     ^b110;        "IO CMD inactive
    MEMORY      =     ^b011;        "1WS CMD active
    FLOAT       =     ^b111;        "data bus float delay
    NOTLOCAL    =     ^b001;       "0WS cycle or bus cycle not to the local bus

" Pin names:
                                " Input pins
    CLK         pin   2;            " 82384 CLK
    MIO         pin   3;            " 80386 M/IO#
    DC          pin   4;            " 80386 D/C#
    WR          pin   5;            " 80386 W/R#
    L0          pin   6;            " local cycle state (from PAL 1)
    L1          pin   7;            " local cycle state (from PAL 1)
    L2          pin   8;            " local cycle state (from PAL 1)
    CS0WS       pin   9;            " chip-select for 0 wait-state (piped) devices

    CLK2        pin   1;        " Clock pin - 82384 CLK2
    OE          pin   11;       " Output Enable pin

                                " Output pins
    MRDC        pin   19;           " Memory Read  Control signal
    MWTC        pin   18;           " Memory Write Control signal
    IORC        pin   17;           " I/O    Read  Control signal
    IOWC        pin   16;           " I/O    Write Control signal
    INTA        pin   15;           " Interrupt Acknowledge Control signal
    ALE         pin   14;           " Address Latch Enable    Control signal
    DEN         pin   13;           " Data Transceiver Enable Control signal
    RDY         pin   12;           " READY signal

    LOCALSTATE  =     [L2, L1, L0];     " local cycle state

" Macros:

    ifMEMORYREAD macro
        { ( MIO &       !WR) }; " true for memory data or code read

    ifMEMORYWRITE macro
        { ( MIO &  DC &  WR) }; " true for memory data write

    ifIOREAD macro
        { (!MIO &  DC & !WR) }; " true for I/O data read

    ifIOWRITE macro
        { (!MIO &  DC &  WR) }; " true for I/O data write

    ifINTACK macro
        { (!MIO & !DC & !WR) }; " true for interrupt acknowledge cycle
```

Figure A-2. PAL-2 State Listings

```
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
equations

!MRDC :=
    ((LOCALSTATE==WAITING)  & OFF)
   #((LOCALSTATE==SAMPLECS) & ifMEMORYREAD  & !CS0WS)    "activate if 0WS access
   #((LOCALSTATE==CMDDELAY) & OFF)
   #((LOCALSTATE==IO)       & ((ifMEMORYREAD  & DEN) # !MRDC))  "activate if IO
   #((LOCALSTATE==ENDIO)    & ((ifMEMORYREAD  & DEN) # !MRDC))  "remain if IO
   #((LOCALSTATE==MEMORY)   & ((ifMEMORYREAD  & DEN) # !MRDC))  "activate 1WS
   #((LOCALSTATE==FLOAT)    & OFF)
   #((LOCALSTATE==NOTLOCAL) & (!MRDC      & (RDY # !CLK)));     "remain if 0WS

!MWTC :=
   ((LOCALSTATE==WAITING)  & OFF)
  #((LOCALSTATE==SAMPLECS) & ifMEMORYWRITE & !CS0WS)
  #((LOCALSTATE==CMDDELAY) & OFF)
  #((LOCALSTATE==IO)       & ((ifMEMORYWRITE & DEN) # (!MWTC & RDY)))
  #((LOCALSTATE==ENDIO)    & ((ifMEMORYWRITE & DEN) # (!MWTC & RDY)))
  #((LOCALSTATE==MEMORY)   & ((ifMEMORYWRITE & DEN) # !MWTC))
  #((LOCALSTATE==FLOAT)    & OFF)
  #((LOCALSTATE==NOTLOCAL) & (!MWTC       &  RDY));

!IORC :=
   ((LOCALSTATE==WAITING)  & OFF)
  #((LOCALSTATE==SAMPLECS) & ifIOREAD      & !CS0WS)
  #((LOCALSTATE==CMDDELAY) & OFF)
  #((LOCALSTATE==IO)       & ((ifIOREAD      & DEN) # !IORC))
  #((LOCALSTATE==ENDIO)    & ((ifIOREAD      & DEN) # !IORC))
  #((LOCALSTATE==MEMORY)   & ((ifIOREAD      & DEN) # !IORC))
  #((LOCALSTATE==FLOAT)    & OFF)
  #((LOCALSTATE==NOTLOCAL) & (!IORC      & (RDY # !CLK)));

!IOWC :=
   ((LOCALSTATE==WAITING)  & OFF)
  #((LOCALSTATE==SAMPLECS) & ifIOWRITE     & !CS0WS)
  #((LOCALSTATE==CMDDELAY) & OFF)
  #((LOCALSTATE==IO)       & ((ifIOWRITE     & DEN) # (!IOWC & RDY)))
  #((LOCALSTATE==ENDIO)    & ((ifIOWRITE     & DEN) # (!IOWC & RDY)))
  #((LOCALSTATE==MEMORY)   & ((ifIOWRITE     & DEN) # !IOWC))
  #((LOCALSTATE==FLOAT)    & OFF)
  #((LOCALSTATE==NOTLOCAL) & (!IOWC       &  RDY));

!INTA :=
   ((LOCALSTATE==WAITING)  & OFF)
  #((LOCALSTATE==SAMPLECS) & ifINTACK      & !CS0WS)
  #((LOCALSTATE==CMDDELAY) & OFF)
  #((LOCALSTATE==IO)       & ((ifINTACK      & DEN) # !INTA))
  #((LOCALSTATE==ENDIO)    & ((ifINTACK      & DEN) # !INTA))
  #((LOCALSTATE==MEMORY)   & ((ifINTACK      & DEN) # !INTA))
  #((LOCALSTATE==FLOAT)    & OFF)
  #((LOCALSTATE==NOTLOCAL) & (!INTA      & (RDY # !CLK)));

ALE  :=
   ((LOCALSTATE==WAITING)  & ON)          "activate ALE while waiting
  #((LOCALSTATE==SAMPLECS) & OFF)
  #((LOCALSTATE==CMDDELAY) & OFF)
  #((LOCALSTATE==IO)       & OFF)
  #((LOCALSTATE==ENDIO)    & OFF)
  #((LOCALSTATE==MEMORY)   & OFF)
  #((LOCALSTATE==FLOAT)    & OFF)
  #((LOCALSTATE==NOTLOCAL) & ( (DEN & CS0WS)   "activate ALE if not-local
                               # ALE           "if active...remain active
                           #(!RDY & CLK))); "activate if last CLK2 of 0WS access
```

Figure A-2. PAL-2 State Listings (Cont'd.)

```
!DEN  :=
   ((LOCALSTATE==WAITING)   & OFF)
  #((LOCALSTATE==SAMPLECS)  & OFF)
  #((LOCALSTATE==CMDDELAY)  & OFF)
  #((LOCALSTATE==IO)        & ON)          "activate DEN while in IO
  #((LOCALSTATE==ENDIO)     & ON)          "activate DEN while in IO
  #((LOCALSTATE==MEMORY)    & ON)          "activate DEN while in 1WS access
  #((LOCALSTATE==FLOAT)     & (!MWTC # !IOWC))  "remain active 1 CLK2 if write
  #((LOCALSTATE==NOTLOCAL)  & ( (!ALE & !CSOWS) "activate DEN if OWS access
                               # !DEN)          "if active...remain active
                            & (RDY # !CLK));    "...until last CLK2 of OWS access

!RDY  :=
   ((LOCALSTATE==WAITING)   & OFF)
  #((LOCALSTATE==SAMPLECS)  & OFF)
  #((LOCALSTATE==CMDDELAY)  & OFF)
  #((LOCALSTATE==IO)        & OFF)
  #((LOCALSTATE==ENDIO)     & ON)          "activate RDY at end of IO
  #((LOCALSTATE==MEMORY)    & ((RDY & !DEN & CLK) # (!RDY & !CLK)))
                                        "activate RDY at end of 1WS access
  #((LOCALSTATE==FLOAT)     & OFF)
  #((LOCALSTATE==NOTLOCAL)  & ( ( RDY &  CLK & ( (!MRDC # !IORC # !INTA)
                                               #((!MWTC # !IOWC) & !DEN)))
                               #(!RDY & !CLK)));
                                        "activate RDY at end of OWS access
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

test_vectors    ( [CLK2, CLK, LOCALSTATE, MIO, DC, WR, CSOWS] ->
                  [MRDC, MWTC, IORC, IOWC, INTA, ALE, DEN, RDY] )

           "[inputs] -> [outputs]

"C C            M D W  C      M M I I I  A D R
"L L            I C R  S      R W O O N  L E D
"K K LOCALSTATE O      O      D T R W T  E N Y
"2                     W      C C C C A
"                      S

[c,L,      x    , x,x,x, x] -> [x,x,x,x,x, x,x,x];"initialize to IDLE and WAITING
[c,H,      x    , x,x,x, x] -> [x,x,x,x,x, x,x,x];
[c,L,      x    , x,x,x, x] -> [x,x,x,x,x, x,x,x];
[c,H,      x    , x,x,x, x] -> [x,x,x,x,x, x,x,x];
[c,L,      x    , x,x,x, x] -> [x,x,x,x,x, x,x,x];
[c,H,      x    , x,x,x, x] -> [x,x,x,x,x, x,x,x];
[c,L,      x    , x,x,x, x] -> [x,x,x,x,x, x,x,x];
[c,H,      x    , x,x,x, x] -> [x,x,x,x,x, x,x,x];
[c,L,   WAITING, x,x,x, x] -> [H,H,H,H,H, H,H,H];
[c,H,   WAITING, x,x,x, x] -> [H,H,H,H,H, H,H,H];
[c,L,   WAITING, x,x,x, x] -> [H,H,H,H,H, H,H,H];
[c,H,   WAITING, x,x,x, x] -> [H,H,H,H,H, H,H,H];"100nS SRAM read
[c,L, SAMPLECS, H,L,L, L] -> [L,H,H,H,H, L,H,H];
[c,H, NOTLOCAL, x,x,x, L] -> [L,H,H,H,H, L,L,L];
[c,L, NOTLOCAL, x,x,x, x] -> [L,H,H,H,H, L,L,L];
[c,H, NOTLOCAL, x,x,x, x] -> [H,H,H,H,H, H,H,H];"100nS SRAM read
[c,L, SAMPLECS, H,H,L, L] -> [L,H,H,H,H, L,H,H];
[c,H, NOTLOCAL, x,x,x, L] -> [L,H,H,H,H, L,L,L];
[c,L, NOTLOCAL, x,x,x, x] -> [L,H,H,H,H, L,L,L];
[c,H, NOTLOCAL, x,x,x, x] -> [H,H,H,H,H, H,H,H];"100nS SRAM write
[c,L, SAMPLECS, H,H,H, L] -> [H,L,H,H,H, L,H,H];
[c,H, NOTLOCAL, x,x,x, L] -> [H,L,H,H,H, L,L,H];
[c,L, NOTLOCAL, x,x,x, x] -> [H,L,H,H,H, L,L,H];
```

Figure A-2. PAL-2 State Listings (Cont'd.)

```
[c,H, NOTLOCAL, x,x,x, x] -> [H,L,H,H,H, L,L,L];
[c,L, NOTLOCAL, x,x,x, x] -> [H,H,H,H,H, L,L,L];
[c,H, NOTLOCAL, x,x,x, x] -> [H,H,H,H,H, H,H,H];"100nS SRAM read
[c,L, SAMPLECS, H,H,L, L] -> [L,H,H,H,H, L,H,H];
[c,H, NOTLOCAL, x,x,x, L] -> [L,H,H,H,H, L,L,L];
[c,L, NOTLOCAL, x,x,x, x] -> [L,H,H,H,H, L,L,L];
[c,H, NOTLOCAL, x,x,x, x] -> [H,H,H,H,H, H,H,H];"non-local
[c,L, SAMPLECS, x,x,x, H] -> [H,H,H,H,H, L,H,H];
[c,H, NOTLOCAL, x,x,x, H] -> [H,H,H,H,H, H,H,H];"READY: activated externally
[c,L, NOTLOCAL, x,x,x, x] -> [H,H,H,H,H, H,H,H];
[c,H, NOTLOCAL, x,x,x, x] -> [H,H,H,H,H, H,H,H];"100nS SRAM read
[c,L, SAMPLECS, H,L,L, L] -> [L,H,H,H,H, L,H,H];
[c,H, NOTLOCAL, x,x,x, L] -> [L,H,H,H,H, L,L,L];
[c,L, NOTLOCAL, x,x,x, x] -> [L,H,H,H,H, L,L,L];
[c,H, NOTLOCAL, x,x,x, x] -> [H,H,H,H,H, H,H,H];"idle
[c,L,  WAITING, x,x,x, x] -> [H,H,H,H,H, H,H,H];
[c,H,  WAITING, x,x,x, x] -> [H,H,H,H,H, H,H,H];
[c,L,  WAITING, x,x,x, x] -> [H,H,H,H,H, H,H,H];
[c,H,  WAITING, x,x,x, x] -> [H,H,H,H,H, H,H,H];"100nS SRAM write
[c,L, SAMPLECS, H,H,H, L] -> [H,L,H,H,H, L,H,H];
[c,H, NOTLOCAL, x,x,x, L] -> [H,L,H,H,H, L,L,H];
[c,L, NOTLOCAL, x,x,x, x] -> [H,L,H,H,H, L,L,H];
[c,H, NOTLOCAL, x,x,x, x] -> [H,L,H,H,H, L,L,L];
[c,L, NOTLOCAL, x,x,x, x] -> [H,H,H,H,H, L,L,L];
[c,H, NOTLOCAL, x,x,x, x] -> [H,H,H,H,H, H,H,H];"idle

[c,L,  WAITING, x,x,x, x] -> [H,H,H,H,H, H,H,H];
[c,H,  WAITING, x,x,x, x] -> [H,H,H,H,H, H,H,H];
[c,L,  WAITING, x,x,x, x] -> [H,H,H,H,H, H,H,H];
[c,H,  WAITING, x,x,x, x] -> [H,H,H,H,H, H,H,H];"150nS ROM read
[c,L, SAMPLECS, x,x,x, H] -> [H,H,H,H,H, L,H,H];
[c,H,  MEMORY , H,H,L, x] -> [L,H,H,H,H, L,L,H];
[c,L,  MEMORY , x,x,x, x] -> [L,H,H,H,H, L,L,H];
[c,H,  MEMORY , x,x,x, x] -> [L,H,H,H,H, L,L,L];
[c,L,  MEMORY , x,x,x, x] -> [L,H,H,H,H, L,L,L];
[c,H,   FLOAT , x,x,x, x] -> [H,H,H,H,H, L,H,H];
[c,L,   FLOAT , x,x,x, x] -> [H,H,H,H,H, L,H,H];
[c,H,  WAITING, x,x,x, x] -> [H,H,H,H,H, H,H,H];"100nS SRAM read
[c,L, SAMPLECS, H,L,L, L] -> [L,H,H,H,H, L,H,H];
[c,H, NOTLOCAL, x,x,x, L] -> [L,H,H,H,H, L,L,L];
[c,L, NOTLOCAL, x,x,x, x] -> [L,H,H,H,H, L,L,L];
[c,H, NOTLOCAL, x,x,x, x] -> [H,H,H,H,H, H,H,H];"150nS ROM read
[c,L, SAMPLECS, x,x,x, H] -> [H,H,H,H,H, L,H,H];
[c,H,  MEMORY , H,L,L, x] -> [L,H,H,H,H, L,L,H];
[c,L,  MEMORY , x,x,x, x] -> [L,H,H,H,H, L,L,H];
[c,H,  MEMORY , x,x,x, x] -> [L,H,H,H,H, L,L,L];
[c,L,  MEMORY , x,x,x, x] -> [L,H,H,H,H, L,L,L];
[c,H,   FLOAT , x,x,x, x] -> [H,H,H,H,H, L,H,H];
[c,L,   FLOAT , x,x,x, x] -> [H,H,H,H,H, L,H,H];
[c,H,  WAITING, x,x,x, x] -> [H,H,H,H,H, H,H,H];"150nS SRAM write
[c,L, SAMPLECS, x,x,x, H] -> [H,H,H,H,H, L,H,H];
[c,H,  MEMORY , H,H,H, x] -> [H,L,H,H,H, L,L,H];
[c,L,  MEMORY , x,x,x, x] -> [H,L,H,H,H, L,L,H];
[c,H,  MEMORY , x,x,x, x] -> [H,L,H,H,H, L,L,L];
[c,L,  MEMORY , x,x,x, x] -> [H,L,H,H,H, L,L,L];
[c,H,   FLOAT , x,x,x, x] -> [H,H,H,H,H, L,L,H];
[c,L,   FLOAT , x,x,x, x] -> [H,H,H,H,H, L,H,H];
[c,H,  WAITING, x,x,x, x] -> [H,H,H,H,H, H,H,H];"150nS SRAM read
[c,L, SAMPLECS, x,x,x, H] -> [H,H,H,H,H, L,H,H];
[c,H,  MEMORY , H,H,L, x] -> [L,H,H,H,H, L,L,H];
[c,L,  MEMORY , x,x,x, x] -> [L,H,H,H,H, L,L,H];
[c,H,  MEMORY , x,x,x, x] -> [L,H,H,H,H, L,L,L];
[c,L,  MEMORY , x,x,x, x] -> [L,H,H,H,H, L,L,L];
[c,H,   FLOAT , x,x,x, x] -> [H,H,H,H,H, L,H,H];
[c,L,   FLOAT , x,x,x, x] -> [H,H,H,H,H, L,H,H];"idle
[c,H,  WAITING, x,x,x, x] -> [H,H,H,H,H, H,H,H];
[c,L,  WAITING, x,x,x, x] -> [H,H,H,H,H, H,H,H];
[c,H,  WAITING, x,x,x, x] -> [H,H,H,H,H, H,H,H];"150nS SRAM read
```

Figure A-2. PAL-2 State Listings (Cont'd.)

```
[c,L, SAMPLECS, x,x,x, H] -> [H,H,H,H,H, L,H,H];
[c,H,  MEMORY , H,L,L, x] -> [L,H,H,H,H, L,L,H];
[c,L,  MEMORY , x,x,x, x] -> [L,H,H,H,H, L,L,H];
[c,H,  MEMORY , x,x,x, x] -> [L,H,H,H,H, L,L,L];
[c,L,  MEMORY , x,x,x, x] -> [L,H,H,H,H, L,L,L];
[c,H,   FLOAT , x,x,x, x] -> [H,H,H,H,H, L,H,H];
[c,L,   FLOAT , x,x,x, x] -> [H,H,H,H,H, L,H,H];"idle
[c,H,  WAITING, x,x,x, x] -> [H,H,H,H,H, H,H,H];
[c,L,  WAITING, x,x,x, x] -> [H,H,H,H,H, H,H,H];
[c,H,  WAITING, x,x,x, x] -> [H,H,H,H,H, H,H,H];"peripheral read
[c,L, SAMPLECS, x,x,x, H] -> [H,H,H,H,H, L,H,H];
[c,H, CMDDELAY, x,x,x, x] -> [H,H,H,H,H, L,H,H];
[c,L,    IO   , L,H,L, x] -> [H,H,L,H,H, L,L,H];
[c,H,    IO   , x,x,x, x] -> [H,H,L,H,H, L,L,H];
[c,L,    IO   , x,x,x, x] -> [H,H,L,H,H, L,L,H];
[c,H,    IO   , x,x,x, x] -> [H,H,L,H,H, L,L,H];
[c,L,    IO   , x,x,x, x] -> [H,H,L,H,H, L,L,H];
[c,H,    IO   , x,x,x, x] -> [H,H,L,H,H, L,L,H];
[c,L,    IO   , x,x,x, x] -> [H,H,L,H,H, L,L,H];
[c,H,  ENDIO  , x,x,x, x] -> [H,H,L,H,H, L,L,L];
[c,L,  ENDIO  , x,x,x, x] -> [H,H,L,H,H, L,L,L];
[c,H,  FLOAT  , x,x,x, x] -> [H,H,H,H,H, L,H,H];
[c,L,  FLOAT  , x,x,x, x] -> [H,H,H,H,H, L,H,H];
[c,H,  FLOAT  , x,x,x, x] -> [H,H,H,H,H, L,H,H];
[c,L,  FLOAT  , x,x,x, x] -> [H,H,H,H,H, L,H,H];
[c,H,  FLOAT  , x,x,x, x] -> [H,H,H,H,H, L,H,H];
[c,L,  FLOAT  , x,x,x, x] -> [H,H,H,H,H, L,H,H];
[c,H,  WAITING, x,x,x, x] -> [H,H,H,H,H, H,H,H];
[c,L,  WAITING, x,x,x, x] -> [H,H,H,H,H, H,H,H];
[c,H,  WAITING, x,x,x, x] -> [H,H,H,H,H, H,H,H];"peripheral interrupt ack.
[c,L, SAMPLECS, x,x,x, H] -> [H,H,H,H,H, L,H,H];
[c,H, CMDDELAY, x,x,x, x] -> [H,H,H,H,H, L,H,H];
[c,L,    IO   , L,L,L, x] -> [H,H,H,H,L, L,L,H];
[c,H,    IO   , x,x,x, x] -> [H,H,H,H,L, L,L,H];
[c,L,    IO   , x,x,x, x] -> [H,H,H,H,L, L,L,H];
[c,H,    IO   , x,x,x, x] -> [H,H,H,H,L, L,L,H];
[c,L,    IO   , x,x,x, x] -> [H,H,H,H,L, L,L,H];
[c,H,    IO   , x,x,x, x] -> [H,H,H,H,L, L,L,H];
[c,L,    IO   , x,x,x, x] -> [H,H,H,H,L, L,L,H];
[c,H,  ENDIO  , x,x,x, x] -> [H,H,H,H,L, L,L,L];
[c,L,  ENDIO  , x,x,x, x] -> [H,H,H,H,L, L,L,L];
[c,H,  FLOAT  , x,x,x, x] -> [H,H,H,H,H, L,H,H];
[c,L,  FLOAT  , x,x,x, x] -> [H,H,H,H,H, L,H,H];
[c,H,  FLOAT  , x,x,x, x] -> [H,H,H,H,H, L,H,H];
[c,L,  FLOAT  , x,x,x, x] -> [H,H,H,H,H, L,H,H];
[c,H,  FLOAT  , x,x,x, x] -> [H,H,H,H,H, L,H,H];
[c,L,  FLOAT  , x,x,x, x] -> [H,H,H,H,H, L,H,H];
[c,H,  WAITING, x,x,x, x] -> [H,H,H,H,H, H,H,H];"100nS SRAM read
[c,L, SAMPLECS, H,L,L, L] -> [L,H,H,H,H, L,H,H];
[c,H, NOTLOCAL, x,x,x, L] -> [L,H,H,H,H, L,L,L];
[c,L, NOTLOCAL, x,x,x, x] -> [L,H,H,H,H, L,L,L];
[c,H, NOTLOCAL, x,x,x, x] -> [H,H,H,H,H, H,H,H];"peripheral write
[c,L, SAMPLECS, x,x,x, H] -> [H,H,H,H,H, L,H,H];
[c,H, CMDDELAY, x,x,x, x] -> [H,H,H,H,H, L,H,H];
[c,L,    IO   , L,H,H, x] -> [H,H,H,L,H, L,L,H];
[c,H,    IO   , x,x,x, x] -> [H,H,H,L,H, L,L,H];
[c,L,    IO   , x,x,x, x] -> [H,H,H,L,H, L,L,H];
[c,H,    IO   , x,x,x, x] -> [H,H,H,L,H, L,L,H];
[c,L,    IO   , x,x,x, x] -> [H,H,H,L,H, L,L,H];
[c,H,    IO   , x,x,x, x] -> [H,H,H,L,H, L,L,H];
[c,L,    IO   , x,x,x, x] -> [H,H,H,L,H, L,L,H];
[c,H,    IO   , x,x,x, x] -> [H,H,H,L,H, L,L,H];
[c,L,    IO   , x,x,x, x] -> [H,H,H,L,H, L,L,H];
[c,H,  ENDIO  , x,x,x, x] -> [H,H,H,L,H, L,L,L];
[c,L,  ENDIO  , x,x,x, x] -> [H,H,H,H,H, L,L,L];
[c,H,  FLOAT  , x,x,x, x] -> [H,H,H,H,H, L,H,H];
[c,L,  FLOAT  , x,x,x, x] -> [H,H,H,H,H, L,H,H];

end Bus_Control_386_Pal_2;
```

Figure A-2. PAL-2 State Listings (Cont'd.)

```
PAL16R8                                          PAL DESIGN SPECIFICATIONS
PART NUMBER: 80386 LOCAL BUS CONTROLLER - PAL 1
80386 LOCAL BUS CONTROLLER : PAL 1 OF 2
INTEL, SANTA CLARA, CALIFORNIA
CLK2  CLK   ADS   READY WR    CS0WS CS1WS CSIO  RESET GND
OE    Q0    Q1    L0    L1    L2    PIPE  IDLE  NA    VCC

/PIPE :=   RESET
         + /CLK * /PIPE
         + CLK * PIPE
         + /PIPE * READY
         + IDLE
         + ADS * /PIPE

/IDLE :=   /CLK * /IDLE * /RESET
         + /IDLE * PIPE * /RESET
         + /IDLE * READY * /RESET
         + /ADS * CLK * /PIPE * /RESET

/NA :=   /L1
       + L2
       + /CLK * /NA
       + CLK * L0 * NA
       + /L0 * /NA * Q0
       + /L0 * /NA * Q1

/L2 :=   /CLK * /L1 * /L2 * /RESET
       + /L0 * /L2 * /NA * /RESET
       + /L1 * /L2 * READY * /RESET
       + L0 * L1 * /L2 * Q0 * /RESET
       + /L0 * /L1 * /RESET
       + /CLK * CS0WS * CS1WS * CSIO * /IDLE * L0 * L1 * L2 * /RESET

/L1 :=   RESET
       + /CLK * L0 * /L1
       + L0 * /L1 * READY
       + CS1WS * /L1 * L2
       + L0 * /L1 * L2
       + L0 * L2 * /Q0 * /Q1
       + L0 * /L1 * /Q0 * /Q1
       + /CLK * CS0WS * CS1WS * CSIO * /IDLE * L0 * L2

/L0 :=   /L0 * /L2 * /RESET
       + /L0 * L1 * Q0 * /RESET
       + CS1WS * /CSIO * /L0 * /L1 * /RESET
       + /ADS * CLK * CSIO * L0 * /L1 * L2 * /RESET
       + /ADS * CLK * L0 * /L1 * L2 * /Q0 * /RESET
       + CLK * CSIO * /IDLE * L0 * /L1 * L2 * /RESET
       + CLK * /IDLE * L0 * /L1 * L2 * /Q0 * /RESET
       + /ADS * CLK * /L1 * /L2 * /Q0 * /Q1 * /READY * /RESET

/Q1 :=   RESET
       + Q0 * /Q1
       + CLK * Q0
       + L0 * /Q1
       + CS1WS * /L1 * L2 * /Q1
       + L1 * /L2 * /Q1

/Q0 :=   RESET
       + /CLK * /Q0 * Q1
       + CLK * Q0 * /Q1
       + /L0 * L1 * L2 * /Q0 * /Q1
       + L0 * /L1 * /Q0 * /Q1
       + CS1WS * /L1 * L2 * /Q0 * /Q1
       + /L1 * /L2 * /Q0 * /Q1 * WR
       + /L0 * L1 * /NA * /Q0 * /Q1

DESCRIPTION

    This PAL is the first of two PALs that implement a 386 bus controller
```

Figure A-3. PAL-1 Equations

```
PAL16R8                                        PAL DESIGN SPECIFICATIONS
PART NUMBER: 80386 LOCAL BUS CONTROLLER - PAL 2
80386 LOCAL BUS CONTROLLER : PAL 2 OF 2
INTEL, SANTA CLARA, CALIFORNIA
CLK2  CLK   MIO   DC    WR    L0    L1    L2    CSOWS GND
OE    RDY   DEN   ALE   INTA  IOWC  IORC  MWTC  MRDC  VCC

/MRDC :=   /L0 * L1 * /MRDC
         + L1 * /L2 * /MRDC
         + /CLK * L0 * /L2 * /MRDC
         + L0 * /L2 * /MRDC * RDY
         + /CSOWS * /L0 * /L1 * L2 * MIO * /WR
         + DEN * /L0 * L1 * MIO * /WR
         + DEN * L1 * /L2 * MIO * /WR

/MWTC :=   L0 * L1 * /L2 * /MWTC
         + /L0 * L1 * /MWTC * RDY
         + L0 * /L2 * /MWTC * RDY
         + /CSOWS * DC * /L0 * /L1 * L2 * MIO * WR
         + DC * DEN * /L0 * L1 * MIO * WR
         + DC * DEN * L1 * /L2 * MIO * WR

/IORC :=   /IORC * /L0 * L1
         + /IORC * L1 * /L2
         + /CLK * /IORC * L0 * /L2
         + /IORC * L0 * /L2 * RDY
         + /CSOWS * DC * /L0 * /L1 * L2 * /MIO * /WR
         + DC * DEN * /L0 * L1 * /MIO * /WR
         + DC * DEN * L1 * /L2 * /MIO * /WR

/IOWC :=   /IOWC * L0 * L1 * /L2
         + /IOWC * /L0 * L1 * RDY
         + /IOWC * L0 * /L2 * RDY
         + /CSOWS * DC * /L0 * /L1 * L2 * /MIO * WR
         + DC * DEN * /L0 * L1 * /MIO * WR
         + DC * DEN * L1 * /L2 * /MIO * WR

/INTA :=   /INTA * /L0 * L1
         + /INTA * L1 * /L2
         + /CLK * /INTA * L0 * /L2
         + /INTA * L0 * /L2 * RDY
         + /CSOWS * /DC * /L0 * /L1 * L2 * /MIO * /WR
         + /DC * DEN * /L0 * L1 * /MIO * /WR
         + /DC * DEN * L1 * /L2 * /MIO * /WR

/ALE :=   /ALE * /CLK * /CSOWS * /L2
        + /ALE * /CLK * /DEN * /L2
        + /ALE * /CSOWS * /L2 * RDY
        + /L0
        + L1
        + /ALE * /DEN * /L2 * RDY

/DEN :=   /L0 * L1
        + L1 * /L2
        + /IOWC * L1
        + L1 * /MWTC
        + /CLK * /DEN * L0 * /L2
        + /DEN * L0 * /L2 * RDY
        + /ALE * /CLK * /CSOWS * L0 * /L2
        + /ALE * /CSOWS * L0 * /L2 * RDY

/RDY :=   /L0 * L1 * L2
        + /CLK * L0 * /L2 * /RDY
        + CLK * /DEN * L0 * L1 * /L2 * RDY
        + CLK * /INTA * L0 * /L1 * /L2 * RDY
        + CLK * /IORC * L0 * /L1 * /L2 * RDY
        + CLK * L0 * /L1 * /L2 * /MRDC * RDY
        + CLK * /DEN * /IOWC * L0 * /L2 * RDY
        + CLK * /DEN * L0 * /L2 * /MWTC * RDY

DESCRIPTION

    This PAL is the second of two PALs that implement a 386 bus controller
```

Figure A-4. PAL-2 Equations

Appendix B
80387 Emulator PAL Description

APPENDIX B
80387 EMULATOR PAL DESCRIPTION

This section describes the PAL equations for the Math Control PAL in the 80386 emulator circuit. These equations are listed in Figure B-1.

The primary function of the PAL is to decode the 80386 outputs and generate 80287 inputs. The CLK16D#, DVALID#, and AVALID# signals provide for the correct timing of the outputs. The TP2 input provides the ability to force the PAL outputs to the high impedance state. For normal operation, TP2 is pulled high.

```
PAL16L8B                                  PAL DESIGN SPECIFICATION
386/100          27 February 1986                 ED JACKS
PAL:             MATH CYCle                       MATHCYC
INTeL Corporation
/RDY A31 LRESET /ADS MIO /RD /AVALID /DVALID /CLK16 GND
TP2 /CLK16D /READYO /DVALIDD /AVALIDD NC /IORDD /IOWCD /READYOD VCC

IF (TP2) AVALIDD  = ADS * RDY * /LRESET * CLK16 * /AVALID
         + /RDY * A31 * /MIO * /LRESET * AVALID
         + A31 * /MIO * /LRESET * AVALID * DVALID
         + ADS * /LRESET * CLK16 * /AVALID * DVALID
         + /LRESET * /CLK16 * AVALID

IF (TP2) DVALIDD  = /LRESET * CLK16 * AVALID * DVALID
         + /RDY * /LRESET * DVALID
         + ADS * /LRESET * CLK16 * /DVALID
         + /LRESET * /CLK16 * DVALID

IF (TP2) IORDD   = /RDY * A31 * /MIO * /RD * AVALID * DVALID
         + A31 * /MIO * /CLK16 * RD * AVALID * DVALID

IF (TP2) IOWCD   = /RDY * A31 * /MIO * RD * AVALID * DVALID
         + A31 * /MIO * /CLK16 * /RD * AVALID * DVALID

IF (TP2) READYOD  = A31 * /MIO * /CLK16 * READYO * AVALID * DVALID
         + /RDY * A31 * /MIO * CLK16 * AVALID * DVALID

CLK16D       = /LRESET * /CLK16
```

Figure B-1. 80387 Emulator PAL Equations

Appendix DRAM PAL Descriptions

C

APPENDIX C
DRAM PAL DESCRIPTIONS

This section describes the inputs, outputs, and functions of each of the PALs in the DRAM design described in Chapter 6. The terms Start-Of-Phase and Middle-Of-Phase used to describe PAL input sampling times refer to the 80386 internal CLK phase and are defined in Figure C-1.

The setup, hold, and propagation delay times for each PAL input and output can be determined from the PAL data sheets. In a few cases, the setup and hold times during certain events must be violated; in these cases, the PAL equations mask these inputs so they are not sampled. Because the states are fully registered and because inputs are masked when their setup or hold times cannot be guaranteed, no hazards exist.

DRAM STATE PAL

The DRAM State PAL determines when to run a new DRAM cycle and tracks the state of the DRAM through the cycle. The inputs sample DRAM requests from the processor (or any other bus master) as well as requests for refresh. The outputs store state information and generate the two RAS signals and two multiplexer control signals. Table C-1 contains a description of the outputs and inputs.

The equations for the 3-CLK DRAM State PAL are shown in Figure C-2; those for the 2-CLK DRAM State PAL are shown in Figure C-3. The DRAM State PAL is implemented in a 16R8 PAL if the RAS signals are registered internally, or in a 16R6 PAL if external registers are used. For a 16-MHz system, B-series PAL speeds are required.

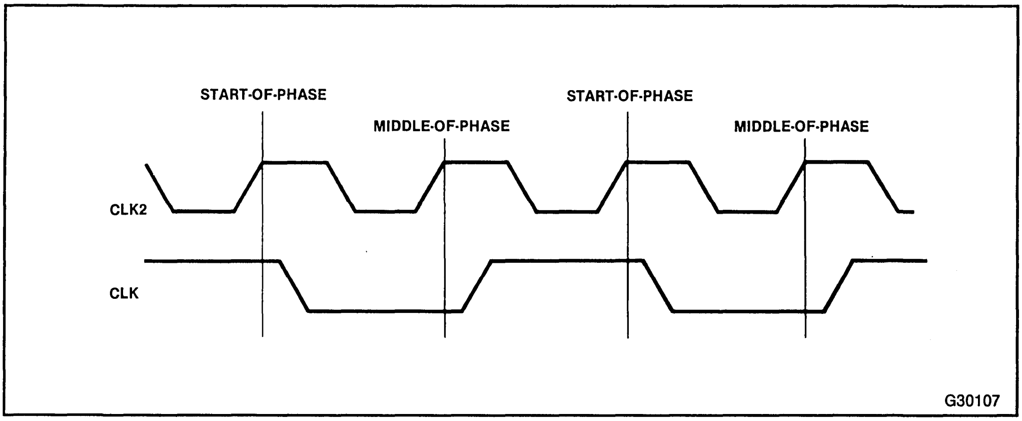


Figure C-1. PAL Sampling Edges

Table C-1. DRAM State PAL Pin Description

PAL CONTROLS

Name	Connects From	PAL Usage
CLK2	Systems CLK2	PAL register clock
OE	Tied low	Outputs always enabled

PAL INPUTS

Name	Connects From	PAL Usage	Sampled
CLK	System CLK	Indicates clock phase	Every CLK2
CS0# CS1# CS2# CS3# CS4#	Chip-Select Logic (uses Address, M/IO, W/R, D/C)	DRAM access is begun (or queued if another cycle is in progress) when all selects are sampled active	Start-Of-Phase (Queue cleared after first cycle of access)
DT/R#	DRAM CONTROL PAL DT/R# out	Indicates write/read Used only in 2-CLK	Start-Of-Phase on 2nd CLK of access
A2	System Address bit 2	Selects one of the two DRAM banks	Start-Of-Phase in which DRAM access starts
RFRQ	Refresh Interval Count	Starts refresh cycle as soon as possible	Middle-Of-Phase

PAL OUTPUTS

Name	Connects To	PAL Usage	Changes State
RAS0#	DRAM Bank 0	Controls DRAM RAS signals	Start-Of-Phase
RAS1#	DRAM Bank 1		
ROWSEL	Addr MUX select	Select DRAM row/column	Middle-Of-Phase
MUXOE#	Addr MUX enable	Disable MUX on refresh	Middle-Of-Phase
A2REG	Not connected	Store active DRAM bank	
DRAMSELECT	Not connected	Queue DRAM requests	
Q0	For NA in 2-CLK	Stores PAL State	
Q1	Not connected		

```
PAL16R8                                        PAL DESIGN SPECIFICATIONS
PART NUMBER: 3-CLK DRAM STATE PAL
DRAM STATE PAL OF INTERLEAVED DRAM CONTROLLER FOR 80386 SYSTEMS
INTEL, SANTA CLARA, CALIFORNIA
CLK2  CLK  A2  CS0  CS1  CS2  CS3  CS4  RFRQ  GND
OE  RAS0  ROWSEL  MUXOE  Q0  Q1  A2REG  DRAMSELECT  RAS1  VCC

/DRAMSELECT :=  CS0 * /DRAMSELECT
              + CS1 * /DRAMSELECT
              + CS2 * /DRAMSELECT
              + /CS3 * /DRAMSELECT
              + /CS4 * /DRAMSELECT
              + /CLK * /DRAMSELECT
              + /MUXOE * ROWSEL * /Q1 * /Q0 * /CLK

/ROWSEL :=  /ROWSEL * Q0 * CLK
          + /ROWSEL * /Q1
          + ROWSEL * /Q1 * /Q0 * /CLK
          + ROWSEL * /Q1 * Q0 * /CLK * MUXOE * RFRQ

/Q1:=  ROWSEL * /Q1 * /Q0 * /CLK
     + ROWSEL * Q0 * CLK
     + /ROWSEL * /Q0 * CLK
     + ROWSEL * Q1 * /Q0 * CLK * /CS0 * /CS1 * /CS2 * CS3 * CS4
            * /MUXOE *  A2 *  A2REG
     + ROWSEL * Q1 * /Q0 * CLK * /CS0 * /CS1 * /CS2 * CS3 * CS4
            * /MUXOE * /A2 * /A2REG
     + ROWSEL * /Q1 * Q0 * /CLK * /MUXOE
     + ROWSEL * /Q1 * Q0 * /CLK * /RFRQ

/Q0 :=  /ROWSEL * Q1 * Q0 * /CLK
      + /ROWSEL * /Q0 * Q1 * CLK
      + ROWSEL * /Q1 * /Q0 * /CLK
      + ROWSEL * Q1 * CLK * /CS0 * /CS1 * /CS2 * CS3 * CS4
            * /MUXOE *  A2 *  A2REG
      + ROWSEL * Q1 * CLK * /CS0 * /CS1 * /CS2 * CS3 * CS4
            * /MUXOE * /A2 * /A2REG
      + ROWSEL * /Q1 * Q0 * CLK * /CS0 * /CS1 * /CS2 * CS3 * CS4 * /MUXOE
      + ROWSEL * /Q1 * Q0 * CLK * DRAMSELECT * /MUXOE
      + ROWSEL * /Q1 * Q0 * /CLK * MUXOE * RFRQ

/RAS0 :=  ROWSEL * /Q1 * /Q0 * /CLK * /A2REG
        + ROWSEL * /Q1 * /Q0 * /CLK * MUXOE
        + /ROWSEL * /A2REG
        + /ROWSEL * MUXOE
        + ROWSEL * Q1 * CLK * /A2 * /A2REG * /CS0 * /CS1 * /CS2 * CS3 * CS4
            * /MUXOE
        + ROWSEL * /Q1 * Q0 * CLK * /A2 * /CS0 * /CS1 * /CS2 * CS3 * CS4
            * /MUXOE
        + ROWSEL * /Q1 * Q0 * CLK * /A2 * DRAMSELECT * /MUXOE

/RAS1 :=  ROWSEL * /Q1 * /Q0 * /CLK *  A2REG
        + ROWSEL * /Q1 * /Q0 * /CLK *  MUXOE
        + /ROWSEL *  A2REG
        + /ROWSEL *  MUXOE
        + ROWSEL * Q1 * CLK *  A2 *  A2REG * /CS0 * /CS1 * /CS2 * CS3 * CS4
            * /MUXOE
        + ROWSEL * /Q1 * Q0 * CLK *  A2 * /CS0 * /CS1 * /CS2 * CS3 * CS4
            * /MUXOE
        + ROWSEL * /Q1 * Q0 * CLK *  A2 * DRAMSELECT * /MUXOE

/MUXOE :=  /MUXOE * /Q0
         + /MUXOE * CLK
         + /MUXOE * /ROWSEL * /Q1
         + /RFRQ * ROWSEL * /Q1 * Q0 * /CLK
         + /MUXOE * /RFRQ * Q1 * Q0 * /CLK

/A2REG :=  /A2REG * /Q0
         + /A2REG * Q1 * CLK
         + /A2REG * ROWSEL * Q1
         + /A2REG * /ROWSEL * /Q1
         +  A2REG * /ROWSEL * Q1 * Q0 * /CLK
         + /A2 * ROWSEL * /Q1 * Q0
```

Figure C-2. 3-CLK DRAM State PAL Equations

```
FUNCTION TABLE

OE  CLK2  CLK  CS0  CS1  CS2  CS3  CS4  A2  RFRQ       ;inputs
ROWSEL  Q1  Q0  RAS0  RAS1  MUXOE  DRAMSELECT  A2REG   ;outputs
;OE
;| CLK2
;| | CLK                ROWSEL
;| | | /CS0             | Q1
;| | | | /CS1           | | Q0
;| | | | | /CS2         | | | /RAS0
;| | | | | | CS3        | | | | /RAS1
;| | | | | | | CS4      | | | | | /MUXOE
;| | | | | | | | A2     | | | | | | DRAMSELECT
;| | | | | | | | | RFRQ | | | | | | | A2REG
;| | | | | | | | | |    | | | | | | | |      STATE        COMMENTS
----------------------------------------------------------------------------
 L C H H X X X X X L    X X X X X X X X;     X     initialize to IDLE
 L C L H X X X X X L    X X X X X X X X;     X     initialize to IDLE
 L C H H X X X X X L    X X X X X X X X;     X     initialize to IDLE
 L C L H X X X X X L    X X X X X X X X;     X     initialize to IDLE
 L C H H X X X X X L    X X X X X X X X;     X     initialize to IDLE
 L C L H X X X X X L    X X X X X X X X;     X     initialize to IDLE
 L C H H X X X X X L    X X X X X X X X;     X     initialize to IDLE
 L C H H X X X X X L    L L H L H L L L;  ACCESS3  continue DRAM cycle
 L C L X X X X X X L    L H H L H L L L;  ACCESS4  continue DRAM cycle
 L C H H X X X X X L    L H H L H L L L;  ACCESS5  continue DRAM cycle
 L C L X X X X X X L    H H L L H L L H;  ACCESS6  continue DRAM cycle
 L C H H X X X X X L    H H H H H L L H;PRECHARGE1 no dram request pending
 L C L X X X X X X L    H H H H H L L H;PRECHARGE2 wait for precharge
 L C H L L L H H H L    H L L H L L H H;  ACCESS1  start DRAM cycle to other bank
 L C L X X X X X X L    L L L H L L L H;  ACCESS2  continue DRAM cycle
 L C H H X X X X X L    L L H H L L L H;  ACCESS3  continue DRAM cycle
 L C L X X X X X X L    L H H H L L L H;  ACCESS4  continue DRAM cycle
 L C H H X X X X X L    L H H H L L L H;  ACCESS5  continue DRAM cycle
 L C L X X X X X X L    H H L H L L L L;  ACCESS6  continue DRAM cycle
 L C H H X X X X X L    H H H H H L L L;PRECHARGE1 no dram request pending
 L C L X X X X X X L    H H H H H L L L;PRECHARGE2 wait for precharge
 L C H H X X X X X L    H L H H H L L X;   IDLE1   no dram request pending
 L C L X X X X X X L    H L H H H L L X;   IDLE2   wait for precharge
 L C H H X X X X X H    H L H H H L L X;   IDLE1   no dram request pending
 L C L X X X X X X H    H L H H H H L X;   IDLE2   refresh request sampled
 L C H L L L H H H H    H L H H H H H H;   IDLE1   can't start: refresh pending
 L C L X X X X X X H    L H L H H H H X; REFSTART2 refresh address set-up
 L C H H X X X X X H    H L L L L H H X;  ACCESS1  start refresh cycle
------------------------------------------------------------------------------
 L C L H X X X X X L    X X X X X X X X;     X     initialize to IDLE
 L C H H X X X X X L    X X X X X X X X;     X     initialize to IDLE
 L C L H X X X X X L    X X X X X X X X;     X     initialize to IDLE
 L C H H X X X X X L    X X X X X X X X;     X     initialize to IDLE
 L C L H X X X X X L    X X X X X X X X;     X     initialize to IDLE
 L C H H X X X X X L    H L H H H L L X;   IDLE1   remain in IDLE
 L C L X X X X X X L    H L H H H L L X;   IDLE2   remain in IDLE
 L C H X H X X X X L    H L H H H L L X;   IDLE1   remain in IDLE
 L C L X X X X X X L    H L H H H L L X;   IDLE2   remain in IDLE
 L C H X X H X X X L    H L H H H L L X;   IDLE1   remain in IDLE
 L C L X X X X X X L    H L H H H L L X;   IDLE2   remain in IDLE
 L C H L L L H H L L    H L L L H L H L;  ACCESS1  start DRAM cycle
 L C L X X X X X X L    L L L L H L L L;  ACCESS2  continue DRAM cycle
 L C H X X X L X X L    L L H L H L L L;  ACCESS3  continue DRAM cycle
 L C L X X X X X X L    L H H L H L L L;  ACCESS4  continue DRAM cycle
 L C H L L L H H X L    L H H L H L H L;  ACCESS5  continue DRAM cycle new request
 L C L X X X X X X L    H H L L H L H H;  ACCESS6  continue DRAM cycle
 L C H L L L H H H L    H L L H L L H H;  ACCESS1  start DRAM cycle to other bank
 L C L X X X X X X L    L L L H L L L H;  ACCESS2  continue DRAM cycle
 L C H X X X L X X L    L L H H L L L H;  ACCESS3  continue DRAM cycle
 L C L X X X X X X L    L H H H L L L H;  ACCESS4  continue DRAM cycle
 L C H X X X X L X L    L H H H L L L H;  ACCESS5  continue DRAM cycle
 L C L X X X X X X L    H H L H L L L L;  ACCESS6  continue DRAM cycle
 L C H L L L H H H L    H H H H H L H L;PRECHARGE1 can't start same bank cycle
```

Figure C-2. 3-CLK DRAM State PAL Equations (Cont'd.)

```
L C L X X X X X X L   H H H H H L H L;PRECHARGE2 wait for precharge
L C H L L L H H H L   H L H H H L H X;   IDLE1   can't start same bank cycle
L C L X X X X X X L   H L H H H L H X;   IDLE2   wait for precharge
L C H H X X X X H H   H L L H L L H H;  ACCESS1  start DRAM cycle to same bank
L C L X X X X X X H   L L L H L L L H;  ACCESS2  continue DRAM cycle
L C H H X X X X X H   L L H H L L L H;  ACCESS3  continue DRAM cycle
L C L X X X X X X H   L H H H L L L H;  ACCESS4  continue DRAM cycle
L C H L L L H H X H   L H H H L L H H;  ACCESS5  continue DRAM cycle new request
L C L X X X X X X H   H H L H L H H L;  ACCESS6  continue DRAM cycle refresh req
L C H L L L H H H H   H H H H H H H L;PRECHARGE1 can't start: refresh pending
L C L X X X X X X H   H H H H H H H L;PRECHARGE2 wait for precharge
L C H L L L H H H H   H L H H H H H X;   IDLE1   can't start: refresh pending
L C L X X X X X X H   L H L H H H H X; REFSTART2 wait for precharge
L C H H X X X X X H   H L L L L H H X;  ACCESS1  start refresh cycle
L C L X X X X X X H   L L L L L H H X;  ACCESS2  continue refresh cycle
L C H H X X X X X L   L L H L L H H X;  ACCESS3  continue refresh cycle
L C L X X X X X X L   L H H L L H H X;  ACCESS4  continue refresh cycle
L C H H X X X X X L   L H H L L H H X;  ACCESS5  continue refresh cycle
L C L X X X X X X L   H H L L L H H X;  ACCESS6  continue refresh cycle
L C H H X X X X X L   H H H H H H H X;PRECHARGE1 can't start: refresh precharge
L C L X X X X X X L   H H H H H H H X;PRECHARGE2 wait for precharge
L C H H X X X X X L   H L H H H H H X;   IDLE1   can't start: refresh precharge
L C L X X X X X X L   H L H H H L H X;   IDLE2   wait for precharge
L C H H X X X X L L   H L L L H L H L;  ACCESS1  start DRAM cycle
L C L X X X X X X L   L L L L H L L L;  ACCESS2  continue DRAM cycle


DESCRIPTION
*** NOTE - SOME VERSIONS OF PALASM WILL CRASH IF THE FILE IS TOO LONG ***
*** IF YOURS DOES, DELETE THIS DESCRIPTION (FROM HERE TO END-OF-FILE) ***

This PAL implements the main state machine of the DRAM controller.
The state machine is described below.

For brevity, the following keywords are used

SELECT = (/CS0 * /CS1 * /CS2 * CS3 * CS4 * CLK)
                            ;chip selects and clock must be active to select

SELECTED = (SELECT + DRAMSELECT)  ;true if DRAM is now or has been selected

STARTACCESS = (SELECTED * /MUXOE) ;start dram access cycle from idle

The states are defined below and indicated by [ROWSEL:Q1:Q0:CLK].
The 4-bit binary number following the state name represents these four signals.

/=======================================================\
| state REFSTART2 = 0101  ;cycle preceding refresh      |
|   /RAS0 :=  ON    ;next cycle is first RAS for refresh|
|   /RAS1 :=  ON                                        |---+
|   MUXOE :=  MUXOE           ;maintain MUXOE state     |   |always
\=======================================================/   |
                        ^                                   |
                        |MUXOE * RFRQ                       |
                        |                                   |
/=======================================================\   |
| state IDLE1 = 1010  ;waiting for access or refresh    |   |
|   /RAS0 := OFF              ;both RAS's idle          |<--|--------+
|   /RAS1 := OFF                                        |   |        |
|   MUXOE := RFRQ             ;sample refresh request   |   |        |
\=======================================================/   |        |
            |                   ^                           |        |
            |/(MUXOE * RFRQ)    |/STARTACCESS               |        |
            v                   |                           |        |
```

Figure C-2. 3-CLK DRAM State PAL Equations (Cont'd.)

```
/========================================================\   |        |
| state IDLE2 = 1011  ;waiting for access or refresh     |   |        |
|   /RAS0 := (/A2 * STARTACCESS)   ;start access         |   |        |
|   /RAS1 := ( A2 * STARTACCESS)                         |   |        |
|   A2REG :=  A2                   ;sample A2 state      |   |        |
|   MUXOE :=  MUXOE                ;maintain MUXOE state |   |        |
\========================================================/   |        |
                                                             |        |
                        |                                    |        |
                        |STARTACCESS                         |        |
                        v                                    |        |
/========================================================\   |        |
| state ACCESS1 = 1000 ;first cycle of access or refresh |<--+        |
|   /RAS0 := /A2REG + MUXOE   ;RAS corresponding to A2   |            |
|   /RAS1 :=  A2REG + MUXOE              ;or refresh     |<--------+  |
|   A2REG :=  A2REG        ;maintain state of sampled A2 |         |  |
|   MUXOE :=  MUXOE           ;maintain MUXOE state      |<-----+  |  |
\========================================================/      |  |  |
                        |                                       |  |  |
                        |always                                 |  |  |
                        v                                       |  |  |
/========================================================\      |  |  |
| state ACCESS2 = 0001;second cycle of access or refresh |      |  |  |
|   /RAS0 := /A2REG + MUXOE   ;RAS corresponding to A2   |      |  |  |
|   /RAS1 :=  A2REG + MUXOE              ;or refresh     |      |  |  |
|   A2REG :=  A2REG        ;maintain state of sampled A2 |      |  |  |
|   MUXOE :=  MUXOE           ;maintain MUXOE state      |      |  |  |
\========================================================/      |  |  |
                        |                                       |  |  |
                        |always                                 |  |  |
                        v                                       |  |  |
/========================================================\      |  |  |
| state ACCESS3 = 0010 ;third cycle of access or refresh |      |  |  |
|   /RAS0 := /A2REG + MUXOE   ;RAS corresponding to A2   |      |  |  |
|   /RAS1 :=  A2REG + MUXOE              ;or refresh     |      |  |  |
|   A2REG :=  A2REG        ;maintain state of sampled A2 |      |  |  |
|   MUXOE :=  MUXOE           ;maintain MUXOE state      |      |  |  |
\========================================================/      |  |  |
                        |                                       |  |  |
                        |always                                 |  |  |
                        v                                       |  |  |
/========================================================\      |  |  |
| state ACCESS4 = 0111;fourth cycle of access or refresh |      |  |  |
|   /RAS0 := /A2REG + MUXOE   ;RAS corresponding to A2   |      |  |  |
|   /RAS1 :=  A2REG + MUXOE              ;or refresh     |      |  |  |
|   A2REG :=  A2REG        ;maintain state of sampled A2 |      |  |  |
|   MUXOE :=  MUXOE           ;maintain MUXOE state      |      |  |  |
\========================================================/      |  |  |
                        |                                       |  |  |
                        |always                                 |  |  |
                        v                                       |  |  |
/========================================================\      |  |  |
| state ACCESS5 = 1110 ;fifth cycle of access or refresh |      |  |  |
|   /RAS0 := /A2REG + MUXOE   ;RAS corresponding to A2   |      |  |  |
|   /RAS1 :=  A2REG + MUXOE              ;or refresh     |      |  |  |
|   A2REG := /A2REG          ;invert state of sampled A2 |      |  |  |
|   MUXOE :=  MUXOE + RFRQ    ;sample refresh request    |      |  |  |
\========================================================/      |  |  |
                        |                                       |  |  |
                  always|       (STARTACCESS * (( A2 *  A2REG)  |  |  |
                        v                      +(/A2 * /A2REG)))|  |  |
/========================================================\      |  |  |
| state ACCESS6 = 1101 ;sixth cycle of access or refresh |      |  |  |
|   /RAS0 := /A2 * /A2REG * STARTACCESS;start next...    |      |  |  |
|   /RAS1 :=  A2 *  A2REG * STARTACCESS;interleave acces |------+  |  |
|   A2REG :=  A2REG     ;maintain state of interleave A2 |         |  |
|   MUXOE :=  MUXOE           ;maintain MUXOE state      |         |  |
\========================================================/         |  |
                        |                                          |  |
                        |/(STARTACCESS * (( A2 *  A2REG)           |  |
                        v                 +(/A2 * /A2REG)))        |  |
```

Figure C-2. 3-CLK DRAM State PAL Equations (Cont'd.)

```
/=======================================================\         |  |
| state PRECHARGE1 = 1110 ;first precharge after access |         |  |
|   /RAS0 := OFF              ;both RAS’s idle          |         |  |
|   /RAS1 := OFF                                        |         |  |
|   A2REG :=  A2REG     ;maintain state of interleave A2|         |  |
|   MUXOE :=  MUXOE + RFRQ    ;sample refresh request   |         |  |
\=======================================================/         |  |
                        |                                         |  |
                  always|         (STARTACCESS * (( A2 *  A2REG)  |  |
                        v                        +(/A2 * /A2REG)))|  |
/=======================================================\         |  |
| state PRECHARGE2 = 1111 ;second precharge after access|         |  |
|   /RAS0 := /A2 * /A2REG * STARTACCESS;start next...   |         |  |
|   /RAS1 :=  A2 *  A2REG * STARTACCESS;interleave acces|---------+  |
|   A2REG :=  A2REG     ;maintain state of interleave A2|            |
|   MUXOE :=  MUXOE             ;maintain MUXOE state   |            |
\=======================================================/            |
/(STARTACCESS * (( A2 *  A2REG)  |                                   |
                +(/A2 * /A2REG)))+-----------------------------------+
```

Finally, the karnaugh maps for the following signals are:

```
ROWSEL\Q0
     Q1\CLK  00     01     11     10
        \_______________________________            MUXOE
      00 |          MUX           MUX      key:
      01 |          MUX    MUX    M+R        MUX = MUXOE
      11 |          MUX    MUX    M+R        M+R = MUXOE + RFRQ
      10 |   MUX           MUX    RFR        RFR = RFRQ

ROWSEL\Q0
     Q1\CLK  00     01     11     10
        \_______________________________            A2REG
      00 |          A2R           A2R      key:
      01 |          A2R    A2R   /A2R        A2; = A2
      11 |          A2R    A2R    A2R        A2R = A2REG
      10 |   A2R           A2;    A2;

ROWSEL\Q0
     Q1\CLK  00     01     11     10
        \_______________________________            RAS signals
      00 |        /AM AM        /AM AM     key:  [RAS0:RAS1]
      01 |         ON ON /AM AM /AM AM       AM = A2REG + MUXOE
      11 |        /AI AI /AI AI OFFOFF       AS = A2 * STARTACCESS
      10 | /AM AM        /AS AS OFFOFF       AI = A2 * STARTACCESS * interleave

ROWSEL\Q0
     Q1\CLK  00     01     11     10
        \______________________________             ROWSEL state circled
      00 |         0010          0111      key:   [ROWSEL:Q1:Q0:CLK]
      01 |         1000   0110   1101        M = MUXOE * RFRQ
      11 |         1ii0   10i0   1111        S = STARTACCESS
      10 |  0001          10s0   mMm1        I = STARTACCESS * interleave

ROWSEL\Q0
     Q1\CLK  00     01     11     10
        \______________________________             Q1 state circled
      00 |         0010          0111
      01 |         1000   0110   1101
      11 |         1ii0   10i0   1111
      10 |  0001          10s0   mMm1

ROWSEL\Q0
     Q1\CLK  00     01     11     10
        \______________________________             Q0 state circled
      00 |         0010          0111
      01 |         1000   0110   1101
      11 |         1ii0   10i0   1111
      10 |  0001          10s0   mMm1
```

Figure C-2. 3-CLK DRAM State PAL Equations (Cont’d.)

```
PAL16R6                                        PAL DESIGN SPECIFICATIONS
PART NUMBER: 2-CLK DRAM STATE PAL
DRAM STATE PAL OF INTERLEAVED DRAM CONTROLLER FOR 80386 SYSTEMS
INTEL, SANTA CLARA, CALIFORNIA
CLK2  CLK  A2  CS0  CS1  CS2  CS3  DT%R  RFRQ  GND
OE  RAS0  ROWSEL  MUXOE  Q0  Q1  A2REG  DRAMSELECT  RAS1  VCC

/DRAMSELECT :=  CS0 * /DRAMSELECT
              + CS1 * /DRAMSELECT
              + CS2 * /DRAMSELECT
              + /CS3 * /DRAMSELECT
              + /CLK * /DRAMSELECT
              + /MUXOE * ROWSEL * /Q1 * /Q0 * /CLK

/ROWSEL :=  /ROWSEL * Q0 * CLK
          + /ROWSEL * /Q1
          + ROWSEL * /Q1 * /Q0 * /CLK
          + ROWSEL * /Q1 * Q0 * /CLK * MUXOE * RFRQ

/Q1:=  ROWSEL * /Q1 * /Q0 * /CLK
     + ROWSEL * Q0 * CLK
     + Q1 * /Q0 * CLK
     + /ROWSEL * /Q1 * /Q0 * CLK * DT%R * /MUXOE
     + ROWSEL * /Q1 * Q0 * /CLK * /MUXOE
     + ROWSEL * /Q1 * Q0 * /CLK * /RFRQ

/Q0 :=  /ROWSEL * Q1 * Q0 * /CLK
      + /ROWSEL * /Q0 * Q1 * CLK
      + ROWSEL * /Q1 * /Q0 * /CLK
      + ROWSEL * Q1 * CLK * /CS0 * /CS1 * /CS2 * CS3
            * /MUXOE *  A2 *  A2REG
      + ROWSEL * Q1 * CLK * /CS0 * /CS1 * /CS2 * CS3
            * /MUXOE * /A2 * /A2REG
      + ROWSEL * /Q1 * Q0 * CLK * /CS0 * /CS1 * /CS2 * CS3 * /MUXOE
      + ROWSEL * /Q1 * Q0 * CLK * DRAMSELECT * /MUXOE
      + ROWSEL * /Q1 * Q0 * /CLK * MUXOE * RFRQ

/RAS0  =  ROWSEL * /Q1 * /Q0 * /CLK * /A2REG
        + ROWSEL * /Q1 * /Q0 * /CLK * MUXOE
        + /ROWSEL * /A2REG
        + /ROWSEL * MUXOE
        + ROWSEL * Q1 * CLK * /A2 * /A2REG * /CS0 * /CS1 * /CS2 * CS3 * /MUXOE
        + ROWSEL * /Q1 * Q0 * CLK * /A2 * /CS0 * /CS1 * /CS2 * CS3 * /MUXOE
        + ROWSEL * /Q1 * Q0 * CLK * /A2 * DRAMSELECT * /MUXOE

/RAS1  =  ROWSEL * /Q1 * /Q0 * /CLK *  A2REG
        + ROWSEL * /Q1 * /Q0 * /CLK *  MUXOE
        + /ROWSEL *  A2REG
        + /ROWSEL *  MUXOE
        + ROWSEL * Q1 * CLK *  A2 *  A2REG * /CS0 * /CS1 * /CS2 * CS3 * /MUXOE
        + ROWSEL * /Q1 * Q0 * CLK *  A2 * /CS0 * /CS1 * /CS2 * CS3 * /MUXOE
        + ROWSEL * /Q1 * Q0 * CLK *  A2 * DRAMSELECT * /MUXOE

/MUXOE :=  /MUXOE * /Q0
         + /MUXOE * CLK
         + /MUXOE * /ROWSEL * /Q1
         + /RFRQ * ROWSEL * /Q1 * Q0 * /CLK
         + /MUXOE * /RFRQ * Q1 * Q0 * /CLK

/A2REG :=  /A2REG * /Q0
         + /A2REG * Q1 * CLK
         + /A2REG * ROWSEL * Q1
         + /A2REG * /ROWSEL * /Q1
         +  A2REG * /ROWSEL * Q1 * Q0 * /CLK
         + /A2 * ROWSEL * /Q1 * Q0
```

Figure C-3. 2-CLK DRAM State PAL Equations

```
FUNCTION TABLE

OE  CLK2  CLK  CS0  CS1  CS2  CS3  CS4  A2  RFRQ        ;inputs
ROWSEL  Q1  Q0  RAS0  RAS1  MUXOE  DRAMSELECT  A2REG    ;outputs
;OE
;| CLK2
;| | CLK                ROWSEL
;| | | /CS0             | Q1
;| | | | /CS1           | | Q0
;| | | | | /CS2         | | | /RAS0
;| | | | | | CS3        | | | | /RAS1
;| | | | | | | DT%R     | | | | | /MUXOE
;| | | | | | | | A2     | | | | | | DRAMSELECT
;| | | | | | | | | RFRQ | | | | | | | A2REG
;| | | | | | | | | |    | | | | | | | |    STATE      COMMENTS
---------------------------------------------------------------------------
 L C L H X X X X X L    X X X X X X X X;     X     initialize to IDLE
 L C H H X X X X X L    X X X X X X X X;     X     initialize to IDLE
 L C L H X X X X X L    X X X X X X X X;     X     initialize to IDLE
 L C H H X X X X X L    X X X X X X X X;     X     initialize to IDLE
 L C L H X X X X X L    X X X X X X X X;     X     initialize to IDLE
 L C H H X X X X X L    X X X X X X X X;     X     initialize to IDLE
 L C L H X X X X X L    X X X X X X X X;     X     initialize to IDLE
 L C H H X X X X X L    X X X X X X X X;     X     initialize to IDLE
 L C L H X X X X X L    X X X X X X X X;     X     initialize to IDLE
 L C H H X X X X X L    X X X X X X X X;     X     initialize to IDLE
 L C L H X X X X X L    X X X X X X X X;     X     initialize to IDLE
 L C H H X X X X X L    H L H H H L L X;   IDLE1   remain in IDLE
 L C L X X X X X X L    H L H H H L L X;   IDLE2   remain in IDLE
 L C H X H X X X X L    H L H H H L L X;   IDLE1   remain in IDLE
 L C L X X X X X X L    H L H H H L L X;   IDLE2   remain in IDLE
 L C H X X H X X X L    H L H H H L L X;   IDLE1   remain in IDLE
 L C L X X X X X X L    H L H H H L L X;   IDLE2   remain in IDLE
 L C H L L L H X L L    H L L L H L H L;  ACCESS1  start DRAM cycle
 L C L X X X X X X L    L L L L H L L L;  ACCESS2  continue DRAM cycle
 L C H X X X L H X L    L L H L H L L L;  ACCESS3  continue DRAM cycle: it's write
 L C L X X X X X X L    L H H L H L L L;  ACCESS4  continue DRAM cycle
 L C H L L L H X X L    L H H L H L H L;  ACCESS5  continue DRAM cycle new request
 L C L X X X X X X L    H H L L H L H H;  ACCESS6  continue DRAM cycle
 L C H L L L H H H L    H L L H L L H H;  ACCESS1  start DRAM cycle to other bank
 L C L X X X X X X L    L L L H L L L H;  ACCESS2  continue DRAM cycle
 L C H X X X L L X L    L H H H L L L H;  ACCESS5  continue DRAM cycle: it's read
 L C L X X X X X X L    H H L H L L L L;  ACCESS6  continue DRAM cycle
 L C H L L L H H H L    H L H H H L H X;   IDLE1   can't start same bank cycle
 L C L X X X X X X L    H L H H H L H X;   IDLE2   wait for precharge
 L C H H X X X X H L    H L L H L L H H;  ACCESS1  start DRAM cycle to same bank
 L C L X X X X X X L    L L L H L L L H;  ACCESS2  continue DRAM cycle
 L C H H X X X H X L    L L H H L L L H;  ACCESS3  continue DRAM cycle: it's write
 L C L X X X X X X L    L H H H L L L H;  ACCESS4  continue DRAM cycle
 L C H L L L H X X L    L H H H L L H H;  ACCESS5  continue DRAM cycle new request
 L C L X X X X X X L    H H L H L L H L;  ACCESS6  continue DRAM cycle
 L C H L L L H X H L    H L H H H L H X;   IDLE1   can't start same bank cycle
 L C L X X X X X X L    H L H H H L H X;   IDLE2   wait for precharge
 L C H H X X X X H H    H L L H L L H H;  ACCESS1  start DRAM cycle to same bank
 L C L X X X X X X H    L L L H L L L H;  ACCESS2  continue DRAM cycle
 L C H H X X X H X H    L L H H L L L H;  ACCESS3  continue DRAM cycle: it's write
 L C L X X X X X X H    L H H H L L L H;  ACCESS4  continue DRAM cycle
 L C H L L L H X X H    L H H H L L H H;  ACCESS5  continue DRAM cycle new request
 L C L X X X X X X H    H H L H L H H L;  ACCESS6  continue DRAM cycle refresh req
 L C H L L L H X H H    H L H H H H H X;   IDLE1   can't start: refresh pending
 L C L X X X X X X H    L H L H H H H X; REFSTART2 wait for precharge
 L C H H X X X X X H    H L L L L H H X;  ACCESS1  start refresh cycle
 L C L X X X X X X H    L L L L L H H X;  ACCESS2  continue refresh cycle
 L C H H X X X X X L    L H H L L H H X;  ACCESS5  continue refresh cycle
 L C L X X X X X X L    H H L L L H H X;  ACCESS6  continue refresh cycle
 L C H H X X X X X L    H L H H H H H X;   IDLE1   can't start: refresh precharge
 L C L X X X X X X L    H L H H H L H X;   IDLE2   wait for precharge
```

Figure C-3. 2-CLK DRAM State PAL Equations (Cont'd.)

```
L C H H X X X X L L    H L L L H L H L;  ACCESS1  start DRAM cycle
L C L X X X X X X L    L L L L H L L L;  ACCESS2  continue DRAM cycle
L C H L L L H L X L    L H H L H L H L;  ACCESS5  continue DRAM cycle: it's read
L C L X X X X X X L    H H L L H L H H;  ACCESS6  continue DRAM cycle
L C H L L L H X H L    H L L H L L H H;  ACCESS1  start DRAM cycle to other bank
L C L X X X X X X L    L L L H L L L H;  ACCESS2  continue DRAM cycle
L C H H X X X L X L    L H H H L L L H;  ACCESS5  continue DRAM cycle: it's read
L C L X X X X X X L    H H L H L L L L;  ACCESS6  continue DRAM cycle
L C H H X X X X X L    H L H H H L L X;   IDLE1   no dram request pending
L C L X X X X X X L    H L H H H L L X;   IDLE2   wait for precharge
L C H H X X X X X H    H L H H H L L X;   IDLE1   no dram request pending
L C L X X X X X X H    H L H H H H L X;   IDLE2   refresh request sampled
L C H L L L H X H H    H L H H H H H X;   IDLE1   can't start: refresh pending
L C L X X X X X X H    L H L H H H H X; REFSTART2 refresh address set-up
L C H H X X X X X H    H L L L L H H X;  ACCESS1  start refresh cycle
------------------------------------------------------------------------------


DESCRIPTION
*** NOTE - SOME VERSIONS OF PALASM WILL CRASH IF THE FILE IS TOO LONG ***
*** IF YOURS DOES, DELETE THIS DESCRIPTION (FROM HERE TO END-OF-FILE) ***

This PAL implements the main state machine of the DRAM controller.
The state machine is described below.

For brevity, the following keywords are used

SELECT = (/CS0 * /CS1 * /CS2 * CS3 * CLK)
                            ;chip selects and clock must be active to select

SELECTED = (SELECT + DRAMSELECT)  ;true if DRAM is now or has been selected

STARTACCESS = (SELECTED * /MUXOE) ;start dram access cycle from idle

The states are defined below and indicated by ROWSEL:Q1:Q0:CLK.
The 4-bit binary number following the state name represents these four signals.

/=======================================================\
| state REFSTART2 = 0101  ;cycle preceding refresh      |
|  /RAS0 :=  ON     ;next cycle is first RAS for refresh|
|  /RAS1 :=  ON                                         |---+
|  MUXOE :=  MUXOE           ;maintain MUXOE state      |   |always
\=======================================================/   |
                        ^                                   |
                        |MUXOE * RFRQ                       |
                        |                                   |
/=======================================================\   |
| state IDLE1 = 1010  ;waiting for access or refresh    |   |
|  /RAS0 := OFF              ;both RAS's idle           |<--|--------+
|  /RAS1 := OFF                                         |   |        |
|  MUXOE := RFRQ             ;sample refresh request    |   |        |
\=======================================================/   |        |
            |                   ^                           |        |
            |/(MUXOE * RFRQ)    |/STARTACCESS               |        |
            v                   |                           |        |
/=======================================================\   |        |
| state IDLE2 = 1011  ;waiting for access or refresh    |   |        |
|  /RAS0 := (/A2 * STARTACCESS)    ;start access        |   |        |
|  /RAS1 := ( A2 * STARTACCESS)                         |   |        |
|  A2REG :=  A2                    ;sample A2 state     |   |        |
|  MUXOE :=  MUXOE                 ;maintain MUXOE state|   |        |
\=======================================================/   |        |
                        |                                   |        |
                        |STARTACCESS                        |        |
                        v                                   |        |
```

Figure C-3. 2-CLK DRAM State PAL Equations (Cont'd.)

```
/=======================================================\   |       |
| state ACCESS1 = 1000 ;first cycle of access or refresh|<--+       |
|   /RAS0 := /A2REG + MUXOE   ;RAS corresponding to A2  |           |
|   /RAS1 :=  A2REG + MUXOE             ;or refresh     |           |
|   A2REG :=  A2REG       ;maintain state of sampled A2|           |
|   MUXOE :=  MUXOE           ;maintain MUXOE state     |<-----+    |
\=======================================================/      |    |
                        |                                      |    |
                        |always                                |    |
                        v                                      |    |
/=======================================================\      |    |
| state ACCESS2 = 0001;second cycle of access or refresh|      |    |
|   /RAS0 := /A2REG + MUXOE   ;RAS corresponding to A2  |      |    |
|   /RAS1 :=  A2REG + MUXOE             ;or refresh     |--+   |    |
|   A2REG :=  A2REG       ;maintain state of sampled A2|  |   |    |
|   MUXOE :=  MUXOE           ;maintain MUXOE state     |  |   |    |
\=======================================================/  |   |    |
                        |                     /DT%R + MUXOE|   |    |
                        |/(/DT%R + MUXOE)                  |   |    |
                        v                                  |   |    |
/=======================================================\  |   |    |
| state ACCESS3 = 0010 ;third cycle of access or refresh|  |   |    |
|   /RAS0 := /A2REG + MUXOE   ;RAS corresponding to A2  |  |   |    |
|   /RAS1 :=  A2REG + MUXOE             ;or refresh     |  |   |    |
|   A2REG :=  A2REG       ;maintain state of sampled A2|  |   |    |
|   MUXOE :=  MUXOE           ;maintain MUXOE state     |  |   |    |
\=======================================================/  |   |    |
                        |                                  |   |    |
                        |always                            |   |    |
                        v                                  |   |    |
/=======================================================\  |   |    |
| state ACCESS4 = 0111;fourth cycle of access or refresh|  |   |    |
|   /RAS0 := /A2REG + MUXOE   ;RAS corresponding to A2  |  |   |    |
|   /RAS1 :=  A2REG + MUXOE             ;or refresh     |  |   |    |
|   A2REG :=  A2REG       ;maintain state of sampled A2|  |   |    |
|   MUXOE :=  MUXOE           ;maintain MUXOE state     |  |   |    |
\=======================================================/  |   |    |
                        |                                  |   |    |
                        |always                            |   |    |
                        v                                  |   |    |
/=======================================================\  |   |    |
| state ACCESS5 = 1110 ;fifth cycle of access or refresh|  |   |    |
|   /RAS0 := /A2REG + MUXOE   ;RAS corresponding to A2  |  |   |    |
|   /RAS1 :=  A2REG + MUXOE             ;or refresh     |<-+   |    |
|   A2REG := /A2REG         ;invert state of sampled A2 |      |    |
|   MUXOE :=  MUXOE + RFRQ    ;sample refresh request   |      |    |
\=======================================================/      |    |
                        |                                      |    |
                  always|      (STARTACCESS * (( A2 *  A2REG)  |    |
                        v                     +(/A2 * /A2REG)))|    |
/=======================================================\      |    |
| state ACCESS6 = 1101 ;sixth cycle of access or refresh|      |    |
|   /RAS0 := /A2 * /A2REG * STARTACCESS;start next...   |      |    |
|   /RAS1 :=  A2 *  A2REG * STARTACCESS;interleave acces|------+    |
|   A2REG :=  A2REG     ;maintain state of interleave A2|           |
|   MUXOE :=  MUXOE           ;maintain MUXOE state     |           |
\=======================================================/           |
/(STARTACCESS * (( A2 *  A2REG)  |                                  |
                +(/A2 * /A2REG)))+----------------------------------+
```

Figure C-3. 2-CLK DRAM State PAL Equations (Cont'd.)

```
Finally, the karnaugh maps for the following signals are:

ROWSEL\Q0
     Q1\CLK  00     01     11     10
        \_______________________________            MUXOE
      00 |         MUX           MUX    key:
      01 |         MUX    MUX    M+R      MUX = MUXOE
      11 |         MUX                    M+R = MUXOE + RFRQ
      10 |  MUX          MUX    RFR       RFR = RFRQ

ROWSEL\Q0
     Q1\CLK  00     01     11     10
        \_______________________________            A2REG
      00 |         A2R           A2R    key:
      01 |         A2R    A2R    A2R      A2; = A2
      11 |         A2R                    A2R = A2REG
      10 |  A2R          A2;    A2;

ROWSEL\Q0
     Q1\CLK  00     01     11     10
        \_______________________________            RAS signals
      00 |        /AM AM        /AM AM   key:  [RAS0:RAS1]
      01 |         ON ON /AM AM /AM AM     AM = A2REG + MUXOE
      11 |        /AI AI                   AS = A2 * STARTACCESS
      10 | /AM AM        /AS AS OFFOFF     AI = A2 * STARTACCESS * interleave

ROWSEL\Q0
     Q1\CLK  00     01     11     10
        \______________________________             ROWSEL state circled
      00 |         0W10          0111   key:   [ROWSEL:Q1:Q0:CLK]
      01 |         1000   0110   1101     M = MUXOE * RFRQ
      11 |         10i0                   S = STARTACCESS
      10 |  0001          10s0   mMm1     I = STARTACCESS * interleave
                                          W = W%R + MUXOE
ROWSEL\Q0
     Q1\CLK  00     01     11     10
        \______________________________             Q1 state circled
      00 |         0W10          0111
      01 |         1000   0110   1101
      11 |         10i0
      10 |  0001          10s0   mMm1

ROWSEL\Q0
     Q1\CLK  00     01     11     10
        \______________________________             Q0 state circled
      10 |         0W10          0111
      11 |         1000   0110   1101
      01 |         10i0
      00 |  0001          10s0   mMm1
```

Figure C-3. 2-CLK DRAM State PAL Equations (Cont'd.)

DRAM CONTROL PAL

The DRAM Control PAL generates the majority of the control signals for the DRAM circuit. The inputs sample the W/R# and byte-enable outputs of the 80386 as well as status signals from the DRAM State PAL. The outputs generate the four CAS signals, two transceiver control signals, and the signals for the 80386 READY# and Next Address (NA#) logic. Table C-2 contains a description of the outputs and inputs.

The equations for the 3-CLK DRAM Control PAL are shown in Figure C-4; those for the 2-CLK DRAM Control PAL are shown in Figure C-5. A 16R8 PAL is needed to register the CAS signals internally. A 16R4 PAL is needed when external registers drive the CAS signals. For a 16-MHz system, B-series PAL speeds are required.

REFRESH INTERVAL COUNTER PAL

The Refresh Interval Counter PAL, which periodically generates refresh requests to the DRAM State PAL, operates as a counter decremented every CLK cycle. Once the counter reaches a preset value, it resets its value to 255 and activates its RFRQ (refresh request) output. This output remains active until both REFACK (refresh acknowledge) inputs are sampled simultaneously active.

Setup and hold times for RFRQ to the DRAM State PAL are guaranteed even with a large CLK2-to-CLK skew because the Refresh Interval Counter PAL is clocked by the rising edge of CLK, and the RFRQ output is only sampled by the DRAM State PAL at the middle-of-phase CLK2 edge. However, the CLK2-to-CLK and output delays can add up so that the setup and hold times for the REFACK inputs are not met. Therefore, the REFACK inputs are activated for a minimum of four CLK2 periods to ensure deactivation of RFRQ. The exact CLK in which RFRQ is deactivated is not critical.

Table C-3 shows the inputs and outputs of the Refresh Interval Counter PAL. Figure C-6 shows its PAL equations. The same equations are used for both the 3-CLK and 2-CLK designs. A 20X10 PAL is used to implement this counter. For 16-MHz systems, A-series PAL speeds are sufficient.

REFRESH ADDRESS COUNTER PAL

The Refresh Address Counter PAL maintains the address of the next DRAM row to be refreshed. After every refresh cycle, the PAL increments this address. Table C-4 shows the inputs and outputs of the Refresh Address Counter PAL.

PAL equations are shown in Figure C-7. Both the 3-CLK and the 2-CLK design use the same equations. Most DRAMs require only 8-bits or fewer for the refresh row address, so a 16R8 PAL can be used. If necessary, 10 bits of row address can be provided using a 20X10 PAL. For a system operating at any speed, standard-PAL speeds are sufficient.

Table C-2. DRAM Control PAL Pin Description

PAL CONTROLS		
Name	**Connects From**	**PAL Usage**
CLK2	System CLK2	PAL register clock
OE	Tied low	Outputs always enabled

PAL INPUTS			
Name	**Connects From**	**PAL Usage**	**Sampled**
CLK	System CLK	Indicates clock phase	Every CLK2
BE0# BE1# BE2# BE3#	System Byte-Enables	Used to enable the DRAM CAS signals corresponding to the active bytes	Start-Of-Phase for internal reg. Every CLK2 with external reg
W/R#	System W/R#	Select write/read	Every CLK2
ROWSEL	DRAM STATE PAL	Initiate DRAM access	Middle-Of-Phase
DISABLE	DRAM STATE PAL MUXOE	Disable controls during refresh	Middle-Of-Phase

PAL OUTPUTS			
Name	**Connects To**	**PAL Usage**	**Changes State**
CAS0#	DRAM Byte 0	Controls DRAM CAS signals (Separate controls for writes to individual bytes)	Start-Of-Phase for read active and read/write inactive Middle-Of-Phase for write active
CAS1#	DRAM Byte 1		
CAS2#	DRAM Byte 2		
CAS3#	DRAM Byte 3		
DEN#	Transceiver	Control xcvr enable	Start-Of-Phase
DT/R#	Transceiver	Control xcvr direction	Any time DEN# off
RDY	System Ready Logic	Control system ready	Rise: Start-Phase Fall: Middle-Phase
WC	DRAM WE# and System NA# logic	Stores PAL state used only in 3-CLK	Rise: Start-Phase Fall: Middle-Phase
WE#	DRAM WE#	Control DRAM WE# used only in 2-CLK	Rise: Start-Phase Fall: Middle-Phase

```
PAL16R8                                        PAL DESIGN SPECIFICATIONS
PART NUMBER: 3-CLK DRAM CONTROL PAL
DRAM CONTROL PAL OF INTERLEAVED DRAM CONTROLLER FOR 80386 SYSTEMS
INTEL, SANTA CLARA, CALIFORNIA
CLK2  CLK  BE0  BE1  BE2  BE3  W%R  ROWSEL  DISABLE  GND
OE  CAS0  CAS1  DT%R  DEN  RDY  WC  CAS2  CAS3  VCC

/CAS0 :=  /ROWSEL * CLK * /DT%R * /DISABLE   ;drop CAS on read
        + /ROWSEL * WC * RDY * /CLK * /BE0   ;drop on write
        + /ROWSEL * /CAS0                    ;maintain throughout cycle
                                        ;BEx can disappear after CAS drop
                ;DISABLE must be maintained through last /ROWSEL * CLK

/CAS1 :=  /ROWSEL * CLK * /DT%R * /DISABLE   ;drop CAS on read
        + /ROWSEL * WC * RDY * /CLK * /BE1   ;drop on write
        + /ROWSEL * /CAS1                    ;maintain throughout cycle

/CAS2 :=  /ROWSEL * CLK * /DT%R * /DISABLE   ;drop CAS on read
        + /ROWSEL * WC * RDY * /CLK * /BE2   ;drop on write
        + /ROWSEL * /CAS2                    ;maintain throughout cycle

/CAS3 :=  /ROWSEL * CLK * /DT%R * /DISABLE   ;drop CAS on read
        + /ROWSEL * WC * RDY * /CLK * /BE3   ;drop on write
        + /ROWSEL * /CAS3                    ;maintain throughout cycle

/DT%R :=  ROWSEL * DEN * /W%R          ;sample W%R when /ROWSEL * DEN
        + /ROWSEL * /DT%R                ;otherwise: maintain state
        + /DEN * /DT%R

/DEN :=   CLK * /ROWSEL * /DISABLE        ;when CLK: sample ROWSEL
       + /CLK * /DEN                    ;otherwise: maintain state

/WC :=   DISABLE                        ;keep low if DISABLE
       + ROWSEL                         ;         or /ROWSEL
       + /RDY                           ;         or /RDY
       + /WC * /CLK                     ;         or already low * /CLK

/RDY :=   WC  * CLK                     ;drop RDY
       + /RDY * /DEN                    ;maintain RDY
```

Figure C-4. 3-CLK DRAM Control PAL Equations

```
FUNCTION TABLE

OE  CLK2  CLK  BE0  BE1  BE2  BE3  W%R  ROWSEL  DISABLE  ;inputs
CAS0  CAS1  CAS2  CAS3  DT%R  DEN  RDY  WC               ;outputs

;OE
;| CLK2
;| | CLK                   CAS0
;| | | BE0                 | CAS1
;| | | | BE1               | | CAS2
;| | | | | BE2             | | | CAS3
;| | | | | | BE3           | | | | DT%R
;| | | | | | | W%R         | | | | | DEN
;| | | | | | | | ROWSEL    | | | | | | RDY
;| | | | | | | | | DISABLE | | | | | | | WC
;| | | | | | | | | |       | | | | | | | |           COMMENTS
------------------------------------------------------------------------------
 L C H X X X X X H L       X X X X X H X L  ; initialize to IDLE
 L C L X X X X L H L       H H H H L H H L  ; IDLE: DT%R tracking W%R
 L C H X X X X H H L       H H H H H H H L  ; IDLE: DT%R tracking W%R
 L C L X X X X L H L       H H H H L H H L  ; IDLE: DT%R tracking W%R
 L C H X X X X X L L       L L L L L L H H  ; begin read: assert all CAS's
 L C L X X X X X L L       L L L L L L H H  ; continue read:
 L C H X X X X X L L       L L L L L L L H  ; continue read: RDY active
 L C L X X X X X L L       L L L L L L L L  ; last read cycle
 L C H X X X X X H L       H H H H L H L L  ; CAS's and DEN rise
 L C L X X X X H H L       H H H H H H H L  ; DT%R and RDY rises
 L C H X X X X X L L       H H H H H L H H  ; begin write: assert DEN and WE
 L C L H L L H X L L       H L L H H L H H  ; continue write: assert valid CAS's
 L C H X X X X X L L       H L L H H L L H  ; continue write: RDY active
 L C L X X X X X L L       H L L H H L L L  ; continue write:
 L C H X X X X X H L       H H H H H H L L  ; CAS's and DEN rise
 L C L X X X X L H L       H H H H L H H L  ; RDY rises
 L C H X X X X X L L       L L L L L L H H  ; begin read: assert all CAS's
 L C L X X X X X L L       L L L L L L H H  ; continue read:
 L C H X X X X X L L       L L L L L L L H  ; continue read: RDY active
 L C L X X X X X L L       L L L L L L L L  ; last read cycle
 L C H X X X X X H L       H H H H L H L L  ; CAS's and DEN rise
 L C L X X X X X H L       H H H H X H H L  ; RDY rises
 L C H X X X X X L H       H H H H X H H L  ; begin refresh
 L C L X X X X X L H       H H H H X H H L  ; continue refresh
 L C H X X X X X L H       H H H H X H H L  ; continue refresh
 L C L X X X X X L L       H H H H X H H L  ; last refresh cycle
 L C H X X X X X H L       H H H H X H H L  ; IDLE
 L C L X X X X X H L       H H H H X H H L  ; IDLE
------------------------------------------------------------------------------

DESCRIPTION

This PAL implements most of the control signals of the DRAM controller.
```

Figure C-4. 3-CLK DRAM Control PAL Equations (Cont'd.)

```
PAL16R4                                        PAL DESIGN SPECIFICATIONS
PART NUMBER: 2-CLK DRAM CONTROL PAL
DRAM CONTROL PAL OF INTERLEAVED DRAM CONTROLLER FOR 80386 SYSTEMS
INTEL, SANTA CLARA, CALIFORNIA
CLK2  CLK  BE0  BE1  BE2  BE3  W%R  ROWSEL  DISABLE  GND
OE  CAS0  CAS1  DT%R  DEN  RDY  WE  CAS2  CAS3  VCC

/CAS0 =   /ROWSEL * /DT%R * CLK * /DISABLE  ;drop CAS on read
        + /ROWSEL * /DT%R * /DEN            ;maintain CAS on read
        + /ROWSEL * /DEN * /BE0 * /DISABLE  ;activate CAS on write
                                        ;BEx must be maintained throughout
        ;DISABLE must be maintained through last /ROWSEL * CLK

/CAS1 =   /ROWSEL * /DT%R * CLK * /DISABLE  ;drop CAS on read
        + /ROWSEL * /DT%R * /DEN            ;maintain CAS on read
        + /ROWSEL * /DEN * /BE1 * /DISABLE  ;activate CAS on write

/CAS2 =   /ROWSEL * /DT%R * CLK * /DISABLE  ;drop CAS on read
        + /ROWSEL * /DT%R * /DEN            ;maintain CAS on read
        + /ROWSEL * /DEN * /BE2 * /DISABLE  ;activate CAS on write

/CAS3 =   /ROWSEL * /DT%R * CLK * /DISABLE  ;drop CAS on read
        + /ROWSEL * /DT%R * /DEN            ;maintain CAS on read
        + /ROWSEL * /DEN * /BE3 * /DISABLE  ;activate CAS on write

/DT%R :=  ROWSEL * DEN * /W%R         ;sample W%R when /ROWSEL * DEN
        + /ROWSEL * /DT%R               ;otherwise: maintain state
        + /DEN * /DT%R

/DEN :=   CLK * /ROWSEL * /DISABLE      ;when CLK: sample ROWSEL
       + /CLK * /DEN                    ;otherwise: maintain state

/WE :=   /ROWSEL * DT%R * RDY * /DISABLE  ;only drops for writes

/RDY :=  /ROWSEL * /DT%R * /DISABLE * CLK ;drop RDY immediately for read
       + /WE  * CLK                     ;drop RDY later for write
       + /RDY * /DEN                    ;maintain RDY
```

Figure C-5. 2-CLK DRAM Control PAL Equations

```
FUNCTION TABLE

OE  CLK2  CLK  BE0  BE1  BE2  BE3  W%R  ROWSEL  DISABLE  ;inputs
CAS0  CAS1  CAS2  CAS3  DT%R  DEN  RDY  WE               ;outputs

;OE
;| CLK2
;| | CLK                   CAS0
;| | | BE0                 | CAS1
;| | | | BE1               | | CAS2
;| | | | | BE2             | | | CAS3
;| | | | | | BE3           | | | | DT%R
;| | | | | | | W%R         | | | | | DEN
;| | | | | | | | ROWSEL    | | | | | | RDY
;| | | | | | | | | DISABLE | | | | | | | WE
;| | | | | | | | | |       | | | | | | | |               COMMENTS
----------------------------------------------------------------------------------
 L C H X X X X X H L       X X X X X H X H  ; initialize to IDLE
 L C L X X X X L H L       H H H H L H H H  ; IDLE: DT%R tracking W%R
 L C H X X X X H H L       H H H H H H H H  ; IDLE: DT%R tracking W%R
 L C L X X X X L H L       H H H H L H H H  ; IDLE: DT%R tracking W%R
 L C H X X X X X L L       L L L L L L L H  ; begin read: assert all CAS's
 L C L X X X X X L L       L L L L L L L H  ; last read cycle
 L C H X X X X X H L       H H H H L H L H  ; CAS's and DEN rise
 L C L X X X X H H L       H H H H H H H H  ; DT%R and RDY rises
 L C H H L L H X L L       H L L H H L H L  ; begin write: assert DEN and WE
 L C L H L L H X L L       H L L H H L H L  ; continue write: assert valid CAS's
 L C H H L L H X L L       H L L H H L L L  ; continue write: RDY active
 L C L H L L H X L L       H L L H H L L H  ; continue write:
 L C H X X X X X H L       H H H H H H L H  ; CAS's and DEN rise
 L C L X X X X L H L       H H H H L H H H  ; RDY rises
 L C H X X X X X L L       L L L L L L L H  ; begin read: assert all CAS's
 L C L X X X X X L L       L L L L L L L H  ; last read cycle
 L C H X X X X X H L       H H H H L H L H  ; CAS's and DEN rise
 L C L X X X X X H L       H H H H X H H H  ; RDY rises
 L C H X X X X X L H       H H H H X H H H  ; begin refresh
 L C L X X X X X L H       H H H H X H H H  ; continue refresh
 L C H X X X X X L H       H H H H X H H H  ; continue refresh
 L C L X X X X X L L       H H H H X H H H  ; last refresh cycle
 L C H X X X X X H L       H H H H X H H H  ; IDLE
 L C L X X X X X H L       H H H H X H H H  ; IDLE
----------------------------------------------------------------------------------

DESCRIPTION

This PAL implements most of the control signals of the DRAM controller.
```

Figure C-5. 2-CLK DRAM Control PAL Equations (Cont'd.)

Table C-3. Refresh Interval Counter PAL Pin Description

PAL CONTROLS			
Name	**Connects From**	**PAL Usage**	
CLK	Systems CLK	PAL register clock	
OE	Tied low	Outputs always enabled	
PAL INPUTS			
Name	**Connects From**	**PAL Usage**	**Sampled**
REFACK0# REFACK1#	DRAM RAS0# DRAM RAS1#	Indicates when refresh starts: turns off RFRQ	Every CLK that RFRQ is active
NC0 NC1 NC2 NC3 NC4 NC5 NC6 NC7	Not connected	Not used	Never
PAL OUTPUTS			
Name	**Connects To**	**PAL Usage**	**Changes State**
RFRQ	DRAM STATE RFRQ	Latch refresh request	Any CLK
Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8	Not connected	Implements up to 9-bit modulo counter	Any CLK

```
PAL20X10                                       PAL DESIGN SPECIFICATIONS
PART NUMBER: 16 MHz REFRESH INTERVAL COUNTER PAL
REFRESH INTERVAL PAL OF INTERLEAVED DRAM CONTROLLER FOR 80386 SYSTEMS
INTEL, SANTA CLARA, CALIFORNIA
CLK  REFACK0  REFACK1  NC  NC  NC  NC  NC  NC  NC  NC  GND
OE  RFRQ  NC  Q0  Q1  Q2  Q3  Q4  Q5  Q6  Q7  VCC

/RFRQ :=     /RFRQ * Q7 * Q6 * Q5 * Q4 * Q3 * Q2 * Q1 * Q0 ;raise at 255
            + RFRQ * /REFACK0 * /REFACK1            ;clear when both ACKs low
        :+:  /RFRQ                                  ;else: don't change state

/Q0  :=       /Q0                      ;least-significant bit of counter
            + /Q7 * /Q6 * /Q5 * /Q4 * /Q3       ;set at 7 or less
        :+:   VCC                               ;else decrement

/Q1  :=       /Q1
            + /Q7 * /Q6 * /Q5 * /Q4 * /Q3       ;set at 7 or less
        :+:   /Q7 * /Q6 * /Q5 * /Q4 * /Q3       ;set at 7 or less
            + /Q0                               ;else decrement

/Q2  :=       /Q2
            + /Q7 * /Q6 * /Q5 * /Q4 * /Q3       ;set at 7 or less
        :+:   /Q7 * /Q6 * /Q5 * /Q4 * /Q3       ;set at 7 or less
            + /Q1 * /Q0                         ;else decrement

/Q3  :=       /Q3
            + /Q7 * /Q6 * /Q5 * /Q4 * /Q3       ;set at 7 or less
        :+:   /Q7 * /Q6 * /Q5 * /Q4 * /Q3       ;set at 7 or less
            + /Q2 * /Q1 * /Q0                   ;else decrement

/Q4  :=       /Q4
            + /Q7 * /Q6 * /Q5 * /Q4 * /Q3       ;set at 7 or less
        :+:   /Q7 * /Q6 * /Q5 * /Q4 * /Q3       ;set at 7 or less
            + /Q3 * /Q2 * /Q1 * /Q0             ;else decrement

/Q5  :=       /Q5
            + /Q7 * /Q6 * /Q5 * /Q4 * /Q3       ;set at 7 or less
        :+:   /Q7 * /Q6 * /Q5 * /Q4 * /Q3       ;set at 7 or less
            + /Q4 * /Q3 * /Q2 * /Q1 * /Q0       ;else decrement

/Q6  :=       /Q6
            + /Q7 * /Q6 * /Q5 * /Q4 * /Q3       ;set at 7 or less
        :+:   /Q7 * /Q6 * /Q5 * /Q4 * /Q3       ;set at 7 or less
            + /Q5 * /Q4 * /Q3 * /Q2 * /Q1 * /Q0  ;else decrement

/Q7  :=       /Q7                      ;most-significant bit of counter
            + /Q7 * /Q6 * /Q5 * /Q4 * /Q3       ;set at 7 or less
        :+:   /Q7 * /Q6 * /Q5 * /Q4 * /Q3       ;set at 7 or less
            + /Q6 * /Q5 * /Q4 * /Q3 * /Q2 * /Q1 * /Q0  ;else decrement
```

Figure C-6. Refresh Interval Counter PAL Equations

```
FUNCTION TABLE

OE  CLK  REFACK0  REFACK1                                   ;inputs
RFRQ  Q7  Q6  Q5  Q4  Q3  Q2  Q1  Q0                        ;outputs

;                           Q7
;                           | Q6
;                           | | Q5
;                           | | | Q4
;                           | | | | Q3
;          REFACK0          | | | | | Q2
;OE        | REFACK1        | | | | | | Q1
;| CLK     | |     RFRQ     | | | | | | | Q0
;| |       | |     |        | | | | | | | |           COMMENTS
-------------------------------------------------------------------------------
 L C       H X     H        L L L L H L H H; initialize(ignore errors on vector)
 L C       X H     H        L L L L H L H L; decrement
 L C       L L     L        L L L L H L L H; decrement
 L C       X X     L        L L L L H L L L; decrement
 L C       X X     L        L L L L L H H H; decrement to 7
 L C       X X     L        H H H H H H H H; reset to 255
 L C       X X     H        H H H H H H H L; decrement, activate RFRQ
 L C       H X     H        H H H H H H L H; decrement, sample REFACKs
 L C       X H     H        H H H H H H L L; decrement, sample REFACKs
 L C       L L     L        H H H H H L H H; decrement, both REFACKs: clear RFRQ
 L C       X X     L        H H H H H L H L; decrement
-------------------------------------------------------------------------------

DESCRIPTION

    This  PAL  implements  the counter  to determine  when distributed
refresh cycles  should be run.  This  counter counts intervals of 249
clocks which is just under 15 uS at 16 MHz.

    The  counter counts backwards from 255 to 7.  The clock after the
counter  reaches 7, the counter is set to 255 and will then continues
to  decrement.   Also when  7 is  hit,  RFRQ  is activated  until both
REFACK0 and REFACK1 are simultaneously sampled low.
```

Figure C-6. Refresh Interval Counter PAL Equations (Cont'd.)

Table C-4. Refresh Address Counter PAL Pin Description

PAL CONTROLS			
Name	**Connects From**	**PAL Usage**	
CLOCK	RFRQ & MUXOE#	PAL register clock	
OE	RFRQ & MUXOE#	outputs enable on refresh	
PAL INPUTS			
Name	**Connects From**	**PAL Usage**	**Sampled**
NC0	Not connected	Not used	Never
NC1			
NC2			
NC3			
NC4			
NC5			
NC6			
NC7			
PAL OUTPUTS			
Name	**Connects To**	**PAL Usage**	**Changes State**
Q0	Muxed Addr 0	Implements 8-bit counter	Any Clock
Q1	Muxed Addr 1		
Q2	Muxed Addr 2		
Q3	Muxed Addr 3		
Q4	Muxed Addr 4		
Q5	Muxed Addr 5		
Q6	Muxed Addr 6		
Q7	Muxed Addr 7		

```
PAL16R8                                             PAL DESIGN SPECIFICATIONS
PART NUMBER: REFRESH ADDRESS COUNTER PAL
REFRESH ADDRESS PAL OF INTERLEAVED DRAM CONTROLLER FOR 80386 SYSTEMS
INTEL, SANTA CLARA, CALIFORNIA
CLOCK  NC  NC  NC  NC  NC  NC  NC  NC  GND
OE  A0  A1  A2  A3  A4  A5  A6  A7  VCC

/A0  :=   A0               ;least significant bit of 8-bit counter

/A1  :=   A1 * A0
        +/A1 */A0

/A2  :=   A2 * A1 * A0
        +/A2 */A1
        +/A2 */A0

/A3  :=   A3 * A2 * A1 * A0
        +/A3 */A2
        +/A3 */A1
        +/A3 */A0

/A4  :=   A4 * A3 * A2 * A1 * A0
        +/A4 */A3
        +/A4 */A2
        +/A4 */A1
        +/A4 */A0

/A5  :=   A5 * A4 * A3 * A2 * A1 * A0
        +/A5 */A4
        +/A5 */A3
        +/A5 */A2
        +/A5 */A1
        +/A5 */A0

/A6  :=   A6 * A5 * A4 * A3 * A2 * A1 * A0
        +/A6 */A5
        +/A6 */A4
        +/A6 */A3
        +/A6 */A2
        +/A6 */A1
        +/A6 */A0

/A7  :=   A7 * A6 * A5 * A4 * A3 * A2 * A1 * A0;most-significant bit of counter
        +/A7 */A6
        +/A7 */A5
        +/A7 */A4
        +/A7 */A3
        +/A7 */A2
        +/A7 */A1
        +/A7 */A0
```

Figure C-7. Refresh Address Counter PAL Equations

```
FUNCTION TABLE

OE  CLOCK                                             ;inputs
A7  A6  A5  A4  A3  A2  A1  A0                        ;outputs

;         A7
;         | A6
;         | | A5
;         | | | A4
;         | | | | A3
;         | | | | | A2
;OE       | | | | | | A1
;| CLOCK  | | | | | | | A0
;| |      | | | | | | | |          COMMENTS
----------------------------------------------------------------------------
 L C      H H H H H H H H; initialize (ignore any errors on this vector)
 L C      L L L L L L L L; increment
 L C      L L L L L L L H; increment
 L C      L L L L L L H L; increment
 L C      L L L L L L H H; increment
 L C      L L L L L H L L; increment
 H H      Z Z Z Z Z Z Z Z; high-impedence state
----------------------------------------------------------------------------

DESCRIPTION

    This PAL implements a simple  8-bit  counter  which  is  used  to
generate the refresh row address by the DRAM controller.
```

Figure C-7. Refresh Address Counter PAL Equations (Cont'd.)

TIMING PARAMETERS

Figure C-8 shows the timing of signals for DRAM read and write cycles. Table C-5 displays the worst-case timing parameters for six DRAM circuits, each of which uses a different type of DRAM.

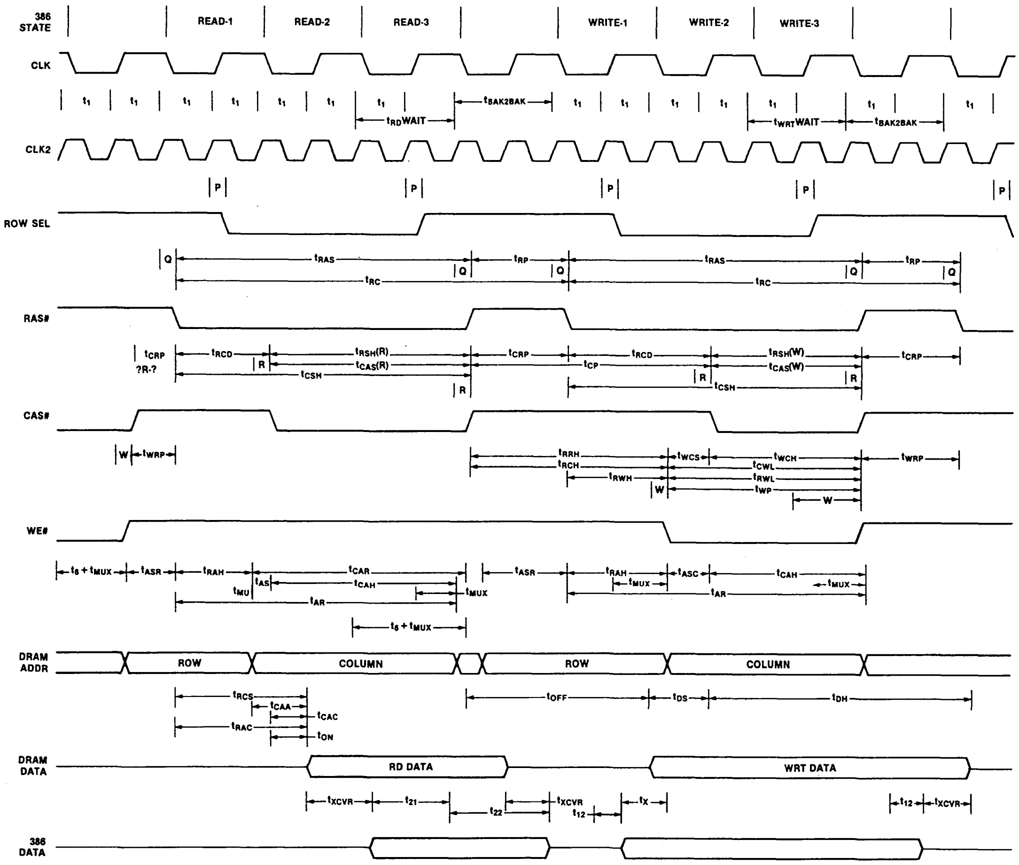

386 STATE
READ-1
READ-2
READ-3
WRITE-1
WRITE-2
WRITE-3
CLK
t1
tBAK2BAK
tRDWAIT
tWRTWAIT
CLK2
P
ROW SEL
Q
tRAS
tRP
tRC
RAS#
tCRP
?R-?
tRCD
tRSH(R)
tCAS(R)
tCSH
R
tCP
tRSH(W)
tCAS(W)
CAS#
W
tWRP
tRRH
tRCH
tRWH
tWCS
tWCH
tCWL
tRWL
tWP
WE#
t6 + tMUX
tASR
tRAH
tCAR
tAS
tCAH
tMU
tMUX
tAR
tASC
DRAM ADDR
ROW
COLUMN
tRCS
tCAA
tCAC
tRAC
tON
tOFF
tDS
tDH
DRAM DATA
RD DATA
WRT DATA
tXCVR
t21
t22
t12
tX
386 DATA
G30107

Figure C-8. DRAM Circuit Timing Diagram

```
                        386 DRAM Controller

                    T I M I N G   P A R A M E T E R S


Chip      Symbol   IO Description                  From      To         Min  Max  Min  Max  Min  Max  Min  Max  Min  Max  Min  Max
                                                                        51C64-8   51C64-10  51C256-12 51C256-15  2164-15  51C256-20

82384     t1        O CLK2 period                  CLK2   /  CLK2   /   31   32   31   32   40   42   31   32   31   32   40   42
                                                   CLK2   /  CLK2   /
                                                   CLK2   /  CLK2   /
                                                   CLK2   /  CLK2   /
                                                   CLK2   /  CLK2   /
                                                   CLK2   /  CLK2   /
                                                   CLK2   /  CLK2   /
                                                   CLK2   /  CLK2   /
                                                   CLK2   /  CLK2   /
                                                   CLK2   /  CLK2   /
                                                   CLK2   /  CLK2   /
                                                   CLK2   /  CLK2   /
                                                   CLK2   /  CLK2   /
                                                   CLK2   /  CLK2   /
                                                   CLK2   /  CLK2   /
82384     trdWAIT   O CLKperiod(s) rd WaitState    CLK2   /  CLK2   /    0    0    0    0    0    0   62   64   62   64   80   84
82384     twrtWAIT  O CLKperiod(s) wt WaitState    CLK2   /  CLK2   /   62   64   62   64   80   84   62   64   62   64   80   84
82384     tBAK2BAK  O CLKperiod(s) back-to-back    CLK2   /  CLK2   /   62   64   62   64  160  168  124  128  124  128  160  168
                                                   CLK2   /  CLK2   /
386       t12       O write data out-delay         CLK2   / 386 Data<    1   50    1   50    1   50    1   50    1   50    1   50
                                                   CLK2   / 386 Data>
386     * t21       O read data set-up            386 Data<  CLK2   /    0   10    0   10    0   10    0   10    0   10    0   10
386       t22       O read data hold               CLK2   / 386 Data>    2  999    2  999    2  999    2  999    2  999    2  999
386+MUX   t6+tMUX   O addr fr 386 thru LatchMux    CLK2   /      row<    3   46    3   46    3   46    3   46    3   46    3   46
                                                   CLK2   /      row<
PAL       P         O clock to PAL outputs         CLK2   /  Row Sel\    0   12    0   12    0   12    0   12    0   12    0   12
                                                   CLK2   /  Row Sel/
                                                   CLK2   /  Row Sel\
                                                   CLK2   /  Row Sel/
                                                   CLK2   /  Row Sel\
PALorREG  Q         O clock to PAL or REG outpt    CLK2   /  RAS#   \    0   12    0   12    0   12    0   12    4   10    0   12
                                                   CLK2   /  RAS#   /
                                                   CLK2   /  RAS#   \
                                                   CLK2   /  RAS#   /
                                                   CLK2   /  RAS#   \
Register  R         O clock to register output     CLK2   /  CAS#   /    0   12    0   12    0   12    0   12    4   10    0   12
                                                   CLK2   /  CAS#   \
                                                   CLK2   /  CAS#   /
                                                   CLK2   /  CAS#   \
                                                   CLK2   /  CAS#   /
                                                   CLK2   /  CAS#   \
PAL+NAND  W         O pal and write logic delay    CLK2   /   WE#   /    2   18    2   18    2   18    2   18    2   18    2   18
                                                   CLK2   /   WE#   \
                                                   CLK2   /   WE#   /
Transcvr  tXCVR     O transcvr prop in-to-out      rd data< 386 Data<    2    7    0    0    2    7    2    7    2    7    2    7
                                                  DramData> 386 Data>
                                                  386 Data< wrt data<
                                                  386 Data> DramData>
```

Table C-5. DRAM Circuit Timing Parameters

```
                      386 DRAM Controller

                  T I M I N G   P A R A M E T E R S


Chip      Symbol  IO Description              From      To        Min  Max   Min  Max   Min  Max   Min  Max   Min  Max   Min  Max
                                                                  51C64-8    51C64-10   51C256-12  51C256-15   2164-15   51C256-20

DRAM      tRAS     I RAS# pulse width         RAS#   \  RAS#   /   80 9999   100 9999   120 9999   150 9999   150 9999   200 9999
                                              RAS#   \  RAS#   /
DRAM      tRC      I random read/write cycle  RAS#   \  RAS#   \  140 9999   160 9999   200 9999   245 9999   260 9999   315 9999
                                              RAS#   \  RAS#   \
DRAM      tRP      I RAS# precharge time      RAS#   /  RAS#   \   50 9999    50 9999    70 9999    85 9999   100 9999   105 9999
                                              RAS#   /  RAS#   \
DRAM      tCSH     I CAS# hold time           RAS#   \  CAS#   /   80 9999   100 9999   120 9999   150 9999   150 9999   200 9999
                                              RAS#   \  CAS#   /
DRAM      tCAS(R)  I CAS# pulse width(rd cycl) CAS#  \  CAS#   /   15 9999    20 9999    25 9999    30 9999    85 9999    35 9999
DRAM      tCAS(W)  I CAS# pulse width(wrt cyc) CAS#  \  CAS#   /   25 9999    30 9999    25 9999    30 9999    85 9999    35 9999
DRAM      tWRP     I write to RAS# precharge   WE#   /  RAS#   \  -30 9999   -30 9999    10 9999    10 9999   -30 9999    10 9999
                                               WE#   /  RAS#   \
DRAM      tRWH     I RAS# to write hold time  RAS#   \   WE#   \    0 9999     0 9999    15 9999    20 9999     0 9999    25 9999
DRAM      tASR     I row address set-up time      row<  RAS#   \    0 9999     0 9999     0 9999     0 9999     0 9999     0 9999
                                                  row<  RAS#   \
DRAM      tRAH     I row address hold time    RAS#   \   column<   15 9999    15 9999    15 9999    20 9999    20 9999    25 9999
                                              RAS#   \   column<
DRAM      tCP      I CAS# precharge           CAS#   /  CAS#   \   10 9999    10 9999    10 9999    10 9999    25 9999    10 9999
                                              CAS#   /  CAS#   \
DRAM      tCRP     I CAS# to RAS# precharge   CAS#   /  RAS#   \  -20 9999   -20 9999   -20 9999   -20 9999   -20 9999   -20 9999
                                              CAS#   /  RAS#   \
                                              CAS#   /  RAS#   \
DRAM    # tRCD     I RAS# to CAS# delay       RAS#   \  CAS#   \   30 9999    30 9999    30 9999    35 9999    30 9999    40 9999
                                              RAS#   \  CAS#   \
DRAM      tASC     I column address set-up     column<  CAS#   \    0 9999     0 9999     5 9999     5 9999     0 9999     5 9999
                                               column<  CAS#   \
DRAM      tCAH     I column address hold      CAS#   \ DramAddr<   10 9999    10 9999    15 9999    20 9999    25 9999    25 9999
                                              CAS#   \ DramAddr<
DRAM      tAR      I column addr hold fr RAS# RAS#   \ DramAddr<   40 9999    40 9999    60 9999    70 9999    90 9999    80 9999
                                              RAS#   \ DramAddr<
DRAM    * tON      I output buffer turn on    CAS#   \  rd data<   20 9999    20 9999    25 9999    30 9999    85 9999    35 9999
DRAM    * tOFF     I output buffer turn off   CAS#   / wrt data<   20 9999    20 9999    20 9999    25 9999    30 9999    30 9999
DRAM    * tRAC     I access time from RAS#    RAS#   \  rd data<   80 9999   100 9999   120 9999   150 9999   150 9999   200 9999
DRAM    * tCAC     I access time from CAS#    CAS#   \  rd data<   20 9999    20 9999    25 9999    30 9999    85 9999    35 9999
DRAM    * tCAA     I access time fr column adr column<  rd data<   45 9999    55 9999    55 9999    70 9999    85 9999    90 9999
DRAM      tRSH(R)  I RAS# hold time (rd cycle) CAS#  \  RAS#   /   10 9999    10 9999    10 9999    10 9999    85 9999    10 9999
DRAM      tRCS     I read command set-up time RAS#   \  rd data<    0 9999     0 9999     0 9999     0 9999     0 9999     0 9999
DRAM      tCAR     I column address to RAS#    column<  RAS#   /   45 9999    55 9999    55 9999    70 9999    85 9999    90 9999
DRAM      tRCH     I read com hold ref to CAS# CAS#  /   WE#   \    0 9999     0 9999     0 9999     0 9999     5 9999     0 9999
DRAM      tRRH     I read com hold ref to RAS# RAS#  /   WE#   \   10 9999    10 9999    10 9999    10 9999    20 9999    10 9999
DRAM      tRSH(W)  I RAS# hold time (wrt cycl) CAS#  \  RAS#   /   35 9999    35 9999    25 9999    30 9999    85 9999    35 9999
DRAM      tRWL     I write command to RAS#     WE#   \  RAS#   /   25 9999    30 9999    25 9999    30 9999    40 9999    35 9999
DRAM      tCWL     I write command to CAS#     WE#   \  CAS#   /   25 9999    30 9999    25 9999    30 9999    40 9999    35 9999
DRAM      tWP      I write command pulse width WE#   \   WE#   /   20 9999    20 9999    20 9999    25 9999    30 9999    30 9999
DRAM      tWCS     I write command set-up time WE#   \  CAS#   \    0 9999     0 9999     0 9999     0 9999   -10 9999     0 9999
DRAM      tWCH     I write command hold time  CAS#   \   WE#   /   25 9999    30 9999    25 9999    30 9999    30 9999    35 9999
DRAM      tDS      I data-in set-up time    wrt data<  CAS#   \    0 9999     0 9999     0 9999     0 9999     0 9999     0 9999
DRAM      tDH      I data-in hold time        CAS#   \ DramData>   20 9999    20 9999    20 9999    25 9999    30 9999    30 9999
```

Table C-5. DRAM Circuit Timing Parameters (Cont'd.)

```
                              386 DRAM Controller

                       T I M I N G   C A L C U L A T I O N S

Chip     Symbol  IO Description                      From      To         Min  Max   Min  Max   Min  Max   Min  Max   Min  Max   Min  Max
                                                                          51C64-8   51C64-10  51C256-12 51C256-15   2164-15  51C256-20

DRAM     tRAS     RAS# pulse width                                         80 9999  100 9999  120 9999  150 9999  150 9999  200 9999
+Q       +trdWAIT +t1       +t1       +t1       +t1      -Q
                                                     RAS#   \  RAS#    /  112  140  112  140  148  180  174  204  180  198  228  264
+Q       +twrtWAIT+t1       +t1       +t1       +t1      -Q
                                                     RAS#   \  RAS#    /  174  204  174  204  228  264  174  204  180  198  228  264


DRAM     tRC      random read/write cycle                                 140 9999  160 9999  200 9999  245 9999  260 9999  315 9999
+Q       +tBAK2BAK+trdWAIT +t1        +t1       +t1       +t1      -Q
                                                     RAS#   \  RAS#    \  174  204  174  204  308  348  298  332  304  326  388  432
+Q       +tBAK2BAK+twrtWAIT+t1        +t1       +t1       +t1      -Q
                                                     RAS#   \  RAS#    \  236  268  236  268  388  432  298  332  304  326  388  432


DRAM     tRP      RAS# precharge time                                      50 9999   50 9999   70 9999   85 9999  100 9999  105 9999
+Q       +tBAK2BAK-Q                                 RAS#   /  RAS#    \   50   76   50   76  148  180  112  140  118  134  148  180
+Q       +tBAK2BAK-Q                                 RAS#   /  RAS#    \   50   76   50   76  148  180  112  140  118  134  148  180


DRAM     tCSH     CAS# hold time                                           80 9999  100 9999  120 9999  150 9999  150 9999  200 9999
+R       +trdWAIT +t1       +t1       +t1       +t1      -Q
                                                     RAS#   \  CAS#    /  112  140  112  140  148  180  174  204  180  198  228  264
+R       +twrtWAIT+t1       +t1       +t1       +t1      -Q
                                                     RAS#   \  CAS#    /  174  204  174  204  228  264  174  204  180  198  228  264


DRAM     tCAS(R)  CAS# pulse width(rd cycl)                                15 9999   20 9999   25 9999   30 9999   85 9999   35 9999
+R       +trdWAIT +t1       +t1       -R             CAS#   \  CAS#    /   50   76   50   76   68   96  112  140  118  134  148  180


DRAM     tCAS(W)  CAS# pulse width(wrt cyc)                                25 9999   30 9999   25 9999   30 9999   85 9999   35 9999
+R       +twrtWAIT+t1       -R                       CAS#   \  CAS#    /   81  108   81  108  108  138   81  108   87  102  108  138


DRAM     tWRP     write to RAS# precharge                                 -30 9999  -30 9999   10 9999   10 9999  -30 9999   10 9999
+Q       +t1      -W                                  WE#   /  RAS#    \   13   42   13   42   22   52   13   42   17   40   22   52
+Q       +tBAK2BAK+twrtWAIT-t1        -W              WE#   /  RAS#    \   74  107   74  107  180  222  136  171  140  169  180  222


DRAM     tRWH     RAS# to write hold time                                   0 9999    0 9999   15 9999   20 9999    0 9999   25 9999
+W       +t1      +t1       -Q                       RAS#   \   WE#    \   52   82   52   82   70  102   52   82   54   78   70  102


DRAM     tASR     row address set-up time                                   0 9999    0 9999    0 9999    0 9999    0 9999    0 9999
+Q       +t1      +t1       -t6+tMUX                     row<  RAS#    \   16   73   16   73   34   93   16   73   20   71   34   93
+Q       +tBAK2BAK+trdWAIT  -t6+tMUX                     row<  RAS#    \   16   73   16   73  114  177  140  201  144  199  194  261


DRAM     tRAH     row address hold time                                    15 9999   15 9999   15 9999   20 9999   20 9999   25 9999
+tMUX    +P       +t1       -Q                       RAS#   \  column<     23   55   23   55   32   65   23   55   25   51   32   65
+tMUX    +P       +t1       -Q                       RAS#   \  column<     23   55   23   55   32   65   23   55   25   51   32   65
```

Table C-5. DRAM Circuit Timing Parameters (Cont'd.)

```
                           386 DRAM Controller
                    T I M I N G   C A L C U L A T I O N S

Chip     Symbol   IO Description                      From      To            Min  Max    Min  Max    Min  Max    Min  Max    Min  Max    Min  Max
                                                                            51C64-8   51C64-10  51C256-12  51C256-15    2164-15  51C256-20

DRAM     tCP      CAS# precharge                                              10 9999     10 9999     10 9999     10 9999     25 9999     10 9999
+R       +t1      +t1      +t1      +tBAK2BAK-R
                                                      CAS#   /  CAS#      \  143  172    143  172    268  306    205  236    211  230    268  306
+R       +t1      +t1      +tBAK2BAK-R                CAS#   /  CAS#      \  112  140    112  140    228  264    174  204    180  198    228  264


DRAM     tCRP     CAS# to RAS# precharge                                     -20 9999    -20 9999    -20 9999    -20 9999    -20 9999    -20 9999
+Q       -R                                           CAS#   /  RAS#      \  -12   12    -12   12    -12   12    -12   12     -6    6    -12   12
+Q       +tBAK2BAK-R                                  CAS#   /  RAS#      \   50   76     50   76    148  180    112  140    118  134    148  180
+Q       +tBAK2BAK-R                                  CAS#   /  RAS#      \   50   76     50   76    148  180    112  140    118  134    148  180


DRAM   # tRCD     RAS# to CAS# delay                                          30 9999     30 9999     30 9999     35 9999     30 9999     40 9999
+R       +t1      +t1      -Q                         RAS#   \  CAS#      \   50   76     50   76     68   96     50   76     56   70     68   96
+R       +t1      +t1      +t1      -Q                RAS#   \  CAS#      \   81  108     81  108    108  138     81  108     87  102    108  138


DRAM     tASC     column address set-up                                        0 9999      0 9999      5 9999      5 9999      0 9999      5 9999
+R       +t1      -P       -tMUX                      column<   CAS#      \    8   40      8   40     17   50      8   40     12   38     17   50
+R       +t1      +t1      -P       -tMUX             column<   CAS#      \   39   72     39   72     57   92     39   72     43   70     57   92


DRAM     tCAH     column address hold                                         10 9999     10 9999     15 9999     20 9999     25 9999     25 9999
+tMUX    +P       +t1      +t1      +t1      -R
                                                      CAS#   \  DramAddr<     85  119     85  119    112  149     85  119     87  115    112  149
+tMUX    +P       +t1      +t1      -R                CAS#   \  DramAddr<     54   87     54   87     72  107     54   87     56   83     72  107


DRAM     tAR      column addr hold fr RAS#                                    40 9999     40 9999     60 9999     70 9999     90 9999     80 9999
+tMUX    +P       +t1      +t1      +t1      +t1      +t1      -Q
                                                      RAS#   \  DramAddr<    147  183    147  183    192  233    147  183    149  179    192  233
+tMUX    +P       +t1      +t1      +t1      +t1      +t1      -Q
                                                      RAS#   \  DramAddr<    147  183    147  183    192  233    147  183    149  179    192  233


DRAM   * tON      output buffer turn on                                       20 9999     20 9999     25 9999     30 9999     85 9999     35 9999
-tXCVR   -t21     +trdWAIT +t1      +t1      -R
                                                      CAS#   \  rd data<      33   62     40   64     51   82     95  126     97  122    131  166


DRAM   * tOFF     output buffer turn off                                      20 9999     20 9999     20 9999     25 9999     30 9999     30 9999
+tXCVR   +t12     +t1      +tBAK2BAK-R                CAS#   /  wrt data<     84  153     82  146    191  267    146  217    148  213    191  267


DRAM   * tRAC     access time from RAS#                                       80 9999    100 9999    120 9999    150 9999    150 9999    200 9999
-tXCVR   -t21     +trdWAIT +t1      +t1      +t1      +t1      -Q
                                                      RAS#   \  rd data<      95  126    102  128    131  166    157  190    159  186    211  250


DRAM   * tCAC     access time from CAS#                                       20 9999     20 9999     25 9999     30 9999     85 9999     35 9999
-tXCVR   -t21     +trdWAIT +t1      +t1      -R
                                                      CAS#   \  rd data<      33   62     40   64     51   82     95  126     97  122    131  166
```

Table C-5. DRAM Circuit Timing Parameters (Cont'd.)

```
                              386 DRAM Controller

                   T I M I N G   C A L C U L A T I O N S

Chip     Symbol   IO Description                  From     To           Min  Max   Min  Max   Min  Max   Min  Max   Min  Max   Min  Max
                                                                     51C64-8  51C64-10 51C256-12 51C256-15   2164-15 51C256-20

DRAM   * tCAA     access time fr column adr                         45 9999    55 9999    55 9999    70 9999    85 9999    90 9999
-tXCVR   -t21     +trdWAIT +t1     +t1     +t1     -P       -tMUX
                                                  column<  rd data<    53   90    60   92    80  120   115  154   115  154   160  204

DRAM     tRSH(R)  RAS# hold time (rd cycle)                         10 9999    10 9999    10 9999    10 9999    85 9999    10 9999
+Q       +trdWAIT +t1     +t1      -R             CAS#   \ RAS#   /    50   76    50   76    68   96   112  140   118  134   148  180

DRAM     tRCS     read command set-up time                           0 9999     0 9999     0 9999     0 9999     0 9999     0 9999
-tXCVR   -t21     +trdWAIT +t1     +t1     +t1     +t1      -Q
                                                  RAS#   \ rd data<    95  126   102  128   131  166   157  190   159  186   211  250

DRAM     tCAR     column address to RAS#                            45 9999    55 9999    55 9999    70 9999    85 9999    90 9999
+Q       +trdWAIT +t1     +t1     +t1     -P       -tMUX
                                                  column<  RAS#   /    70  104    70  104    97  134   132  168   136  166   177  218

DRAM     tRCH     read com hold ref to CAS#                          0 9999     0 9999     0 9999     0 9999     5 9999     0 9999
+W       +t1      +t1      +tBAK2BAK-R            CAS#   / WE#    \   114  146   114  146   230  270   176  210   178  206   230  270

DRAM     tRRH     read com hold ref to RAS#                         10 9999    10 9999    10 9999    10 9999    20 9999    10 9999
+W       +t1      +t1      +tBAK2BAK-Q            RAS#   / WE#    \   114  146   114  146   230  270   176  210   178  206   230  270

DRAM     tRSH(W)  RAS# hold time (wrt cycl)                         35 9999    35 9999    25 9999    30 9999    85 9999    35 9999
+Q       +twrtWAIT+t1      -R                     CAS#   \ RAS#   /    81  108    81  108   108  138    81  108    87  102   108  138

DRAM     tRWL     write command to RAS#                             25 9999    30 9999    25 9999    30 9999    40 9999    35 9999
+Q       +twrtWAIT+t1      +t1      -W            WE#    \ RAS#   /   106  138   106  138   142  178   106  138   110  136   142  178

DRAM     tCWL     write command to CAS#                             25 9999    30 9999    25 9999    30 9999    40 9999    35 9999
+R       +twrtWAIT+t1      +t1      -W            WE#    \ CAS#   /   106  138   106  138   142  178   106  138   110  136   142  178

DRAM     tWP      write command pulse width                         20 9999    20 9999    20 9999    25 9999    30 9999    30 9999
+W       +t1      +t1      +t1      -W            WE#    \ WE#    /    77  112    77  112   104  142    77  112    77  112   104  142

DRAM     tWCS     write command set-up time                          0 9999     0 9999     0 9999     0 9999   -10 9999     0 9999
+R       +t1      -W                              WE#    \ CAS#   \    13   42    13   42    22   52    13   42    17   40    22   52

DRAM     tWCH     write command hold time                           25 9999    30 9999    25 9999    30 9999    30 9999    35 9999
+W       +t1      +t1      -R                     CAS#   \ WE#    /    52   82    52   82    70  102    52   82    54   78    70  102
```

Table C-5. DRAM Circuit Timing Parameters (Cont'd.)

```
                              386 DRAM Controller

                         T I M I N G   C A L C U L A T I O N S

Chip    Symbol  IO Description                From     To          Min  Max   Min  Max   Min  Max   Min  Max   Min  Max   Min  Max
                                                                    51C64-8   51C64-10 51C256-12 51C256-15   2164-15 51C256-20

DRAM    tDS      data-in set-up time                                 0 9999    0 9999    0 9999    0 9999    0 9999    0 9999
+R      +t1      +t1      -t12     -tXCVR    wrt data< CAS#   \      5   73   12   75   23   93    5   73    9   71   23   93

DRAM    tDH      data-in hold time                                  20 9999   20 9999   20 9999   25 9999   30 9999   30 9999
+tXCVR  +t12     +t1      +twrtWAIT+t1     -R
                                              CAS#    \ DramData>  115  185  113  178  151  225  115  185  117  181  151  225
```

Table C-5. DRAM Circuit Timing Parameters (Cont'd.)

DOMESTIC SALES OFFICES

ALABAMA

†Intel Corp.
5015 Bradford Dr., #2
Huntsville 35805
Tel: (205) 830-4010

ARIZONA

†Intel Corp.
11225 N. 28th Dr., #D214
Phoenix 85029
Tel: (602) 869-4980

Intel Corp.
1161 N. El Dorado Place
Suite 301
Tucson 85715
Tel: (602) 299-6815

CALIFORNIA

†Intel Corp.
21515 Vanowen Street
Suite 116
Canoga Park 91303
Tel: (818) 704-8500

†Intel Corp.
2250 E. Imperial Highway
Suite 218
El Segundo 90245
Tel: (213) 640-6040

Intel Corp.
1510 Arden Way, Suite 101
Sacramento 95815
Tel: (916) 920-8096

†Intel Corp.
4350 Executive Drive
Suite 105
San Diego 92121
Tel: (619) 452-5880

Intel Corp.*
400 N. Tustin Avenue
Suite 450
Santa Ana 92705
Tel: (714) 835-9642
TWX: 910-595-1114

†Intel Corp.*
San Tomas 4
2700 San Tomas Expressway
Santa Clara, CA 95051
Tel: (408) 986-8086
TWX: 910-338-0255

COLORADO

Intel Corp.
4445 Northpark Drive
Suite 100
Colorado Springs 80907
Tel: (303) 594-6622

†Intel Corp.*
650 S. Cherry St., Suite 915
Denver 80222
Tel: (303) 321-8086
TWX: 910-931-2289

CONNECTICUT

†Intel Corp.
26 Mill Plain Road
Danbury 06811
Tel: (203) 748-3130
TWX: 710-456-1199

FLORIDA

†Intel Corp.
242 N. Westmonte Dr.
Suite 105
Altamonte Springs 32714
Tel: (305) 869-5588

Intel Corp.
6363 N.W. 6th Way, Suite 100
Ft. Lauderdale 33309
Tel: (305) 771-0600
TWX: 510-956-9407

Intel Corp.
11300 4th Street North
Suite 170
St. Petersburg 33702
Tel: (813) 577-2413

GEORGIA

†Intel Corp.
3280 Pointe Parkway
Suite 200
Norcross 30092
Tel: (404) 449-0541

ILLINOIS

Intel Corp.*
300 N. Martingale Road, Suite 400
Schaumburg 60173
Tel: (312) 310-8031

INDIANA

†Intel Corp.
8777 Purdue Road
Suite 125
Indianapolis 46268
Tel: (317) 875-0623

IOWA

Intel Corp.
St. Andrews Building
1930 St. Andrews Drive N.E.
Cedar Rapids 52402
Tel: (319) 393-5510

KANSAS

†Intel Corp.
8400 W. 110th Street
Suite 170
Overland Park 66210
Tel: (913) 345-2727

MARYLAND

Intel Corp.*
7321 Parkway Drive South
Suite C
Hanover 21076
Tel: (301) 796-7500
TWX: 710-862-1944

Intel Corp.
5th Floor
7833 Walker Drive
Greenbelt 20770
Tel: (301) 441-1020

MASSACHUSETTS

†Intel Corp.*
Westford Corp. Center
3 Carlisle Road
Westford 01886
Tel: (617) 692-3222
TWX: 710-343-6333

MICHIGAN

†Intel Corp.
7071 Orchard Lake Road
Suite 100
West Bloomfield 48033
Tel: (313) 851-8096

MINNESOTA

Intel Corp.
3500 W. 80th St., Suite 360
Bloomington 55431
Tel: (612) 835-6722
TWX: 910-576-2867

MISSOURI

Intel Corp.
4203 Earth City Expressway
Suite 131
Earth City 63045
Tel: (314) 291-1990

NEW JERSEY

Intel Corp.*
Parkway 109 Office Center
328 Newman Springs Road
Red Bank 07701
Tel: (201) 747-2233

Intel Corp.
280 Corporate Center
75 Livingston Avenue
First Floor
Roseland 07068
Tel: (201) 740-0111

NEW MEXICO

Intel Corp.
8500 Menual Boulevard N.E.
Suite B 295
Albuquerque 87112
Tel: (505) 292-8086

NEW YORK

Intel Corp.
127 Main Street
Binghamton 13905
Tel: (607) 773-0337

Intel Corp.*
850 Cross Keys Office Park
Fairport 14450
Tel: (716) 425-2750
TWX: 510-253-7391

Intel Corp.*
300 Motor Parkway
Hauppauge 11787
Tel: (516) 231-3300
TWX: 510-227-6236

Intel Corp.
Suite 2B Hollowbrook Park
15 Myers Corners Road
Wappinger Falls 12590
Tel: (914) 297-6161
TWX: 510-248-0060

NORTH CAROLINA

Intel Corp.
5700 Executive Center Drive
Suite 213
Charlotte 28212
Tel: (704) 568-8966

†Intel Corp.
2700 Wycliff Road
Suite 102
Raleigh 27607
Tel: (919) 781-8022

OHIO

Intel Corp.*
3401 Park Center Drive
Suite 220
Dayton 45414
Tel: (513) 890-5350
TWX: 810-450-2528

Intel Corp.*
25700 Science Park Dr., Suite 100
Beachwood 44122
Tel: (216) 464-2736
TWX: 810-427-9298

OKLAHOMA

Intel Corp.
6801 N. Broadway
Suite 115
Oklahoma City 73116
Tel: (405) 848-8086

OREGON

†Intel Corp.
15254 N.W. Greenbrier Parkway, Bldg. B
Beaverton 97006
Tel: (503) 645-8051
TWX: 910-467-8741

PENNSYLVANIA

Intel Corp.
1513 Cedar Cliff Drive
Camp Hill 17011
Tel: (717) 737-5035

Intel Corp.*
455 Pennsylvania Avenue
Fort Washington 19034
Tel: (215) 641-1000
TWX: 510-661-2077

Intel Corp.*
400 Penn Center Blvd., Suite 610
Pittsburgh 15235
Tel: (412) 823-4970

PUERTO RICO

Intel Microprocessor Corp.
South Industrial Park
P.O. Box 910
Las Piedras 00671
Tel: (809) 733-8616

TEXAS

†Intel Corp.
313 E. Anderson Lane
Suite 314
Austin 78752
Tel: (512) 454-3628

†Intel Corp.*
12300 Ford Road
Suite 380
Dallas 75234
Tel: (214) 241-8087
TWX: 910-860-5617

Intel Corp.*
7322 S.W. Freeway
Suite 1490
Houston 77074
Tel: (713) 988-8086
TWX: 910-881-2490

UTAH

Intel Corp.
5201 Green Street
Suite 290
Murray 84123
Tel: (801) 263-8051

VIRGINIA

†Intel Corp.
1603 Santa Rosa Road
Suite 109
Richmond 23288
Tel: (804) 282-5668

WASHINGTON

Intel Corp.
155-108 Avenue N.E.
Suite 386
Bellevue 98004
Tel: (206) 453-8086
TWX: 910-443-3002

Intel Corp.
408 N. Mullan Road
Suite 102
Spokane 99206
Tel: (509) 928-8086

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
TLX: 053-4115

†Intel Semiconductor of Canada, Ltd.
190 Attwell Drive
Suite 500
Rexdale M9W 6H8
Tel: (416) 675-2105
TLX: 06983574
FAX: (416) 675-2438

QUEBEC

†Intel Semiconductor of Canada, Ltd.
620 St. Jean Boulevard
Pointe Claire H9R 3K3
Tel: (514) 694-9130
TWX: 514-694-9134

†Sales and Service Office
*Field Application Location

CG-6/29/87

DOMESTIC DISTRIBUTORS

ALABAMA

Arrow Electronics, Inc.
1015 Henderson Road
Huntsville 35805
Tel: (205) 837-6955

†Hamilton/Avnet Electronics
4940 Research Drive
Huntsville 35805
Tel: (205) 837-7210
TWX: 810-726-2162

Pioneer/Technologies Group Inc.
4825 University Square
Huntsville 35805
Tel: (205) 837-9300
TWX: 810-726-2197

ARIZONA

†Hamilton/Avnet Electronics
505 S. Madison Drive
Tempe 85281
Tel: (602) 231-5100
TWX: 910-950-0077

Kierulff Electronics, Inc.
4134 E. Wood Street
Phoenix 85040
Tel: (602) 437-0750
TWX: 910-951-1550

Wyle Distribution Group
17855 N. Black Canyon Highway
Phoenix 85023
Tel: (602) 866-2888

CALIFORNIA

Arrow Electronics, Inc.
19748 Dearborn Street
Chatsworth 91311
Tel: (818) 701-7500
TWX: 910-493-2086

Arrow Electronics, Inc.
1502 Crocker Avenue
Hayward 94544
Tel: (408) 487-4600

Arrow Electronics, Inc.
9511 Ridgehaven Court
San Diego 92123
Tel: (619) 565-4800
TLX: 888064

†Arrow Electronics, Inc.
521 Weddell Drive
Sunnyvale 94086
Tel: (408) 745-6600
TWX: 910-339-9371

Arrow Electronics, Inc.
2961 Dow Avenue
Tustin 92680
Tel: (714) 838-5422
TWX: 910-595-2860

†Avnet Electronics
350 McCormick Avenue
Costa Mesa 92626
Tel: (714) 754-6051
TWX: 910-595-1928

Hamilton/Avnet Electronics
1175 Bordeaux Drive
Sunnyvale 94086
Tel: (408) 743-3300
TWX: 910-339-9332

†Hamilton/Avnet Electronics
4545 Viewridge Avenue
San Diego 92123
Tel: (619) 571-7500
TWX: 910-595-2638

†Hamilton/Avnet Electronics
20501 Plummer Street
Chatsworth 91311
Tel: (818) 700-6271
TWX: 910-494-2207

†Hamilton/Avnet Electronics
4103 Northgate Boulevard
Sacramento 95834
Tel: (916) 920-3150

†Hamilton/Avnet Electronics
3002 G Street
Ontario 91311
Tel: (714) 989-9411

Hamilton/Avnet Electronics
19515 So. Vermont Avenue
Torrance 90502
Tel: (213) 615-3909
TWX: 910-349-6263

Hamilton Electro Sales
9650 De Soto Avenue
Chatsworth 91311
Tel: (818) 700-6500

CALIFORNIA (Cont'd)

†Hamilton Electro Sales
10950 W. Washington Blvd.
Culver City 90230
Tel: (213) 558-2458
TWX: 910-340-6364

Hamilton Electro Sales
1361 B West 190th Street
Gardena 90248
Tel: (213) 558-2131

†Hamilton Electro Sales
3170 Pullman Street
Costa Mesa 92626
Tel: (714) 641-4150
TWX: 910-595-2638

Kierulff Electronics, Inc.
10824 Hope Street
Cypress 90430
Tel: (714) 220-6300

†Kierulff Electronics, Inc.
1180 Murphy Avenue
San Jose 95131
Tel: (408) 971-2600
TWX: 910-379-6430

†Kierulff Electronics, Inc.
14101 Franklin Avenue
Tustin 92680
Tel: (714) 731-5711
TWX: 910-595-2599

†Kierulff Electronics, Inc.
5650 Jillson Street
Commerce 90040
Tel: (213) 725-0325
TWX: 910-580-3666

Wyle Distribution Group
26560 Agoura Street
Calabasas 91302
Tel: (818) 880-9000
TWX: 818-372-0232

†Wyle Distribution Group
124 Maryland Street
El Segundo 90245
Tel: (213) 322-8100
TWX: 910-348-7140 or 7111

†Wyle Distribution Group
17872 Cowan Avenue
Irvine 92714
Tel: (714) 863-9953
TWX: 910-595-1572

Wyle Distribution Group
11151 Sun Center Drive
Rancho Cordova 95670
Tel: (916) 638-5282

†Wyle Distribution Group
9525 Chesapeake Drive
San Diego 92123
Tel: (619) 565-9171
TWX: 910-335-1590

†Wyle Distribution Group
3000 Bowers Avenue
Santa Clara 95051
Tel: (408) 727-2500
TWX: 910-338-0296

Wyle Military
18910 Teller Avenue
Irvine 92750
Tel: (714) 851-9958
TWX: 310-371-9127

Wyle Systems
7382 Lampson Avenue
Garden Grove 92641
Tel: (714) 891-1717
TWX: 910-595-2642

COLORADO

Arrow Electronics, Inc.
1390 S. Potomac Street
Suite 136
Aurora 80012
Tel: (303) 696-1111

†Hamilton/Avnet Electronics
8765 E. Orchard Road
Suite 708
Englewood 80111
Tel: (303) 740-1017
TWX: 910-935-0787

†Wyle Distribution Group
451 E. 124th Avenue
Thornton 80241
Tel: (303) 457-9953
TWX: 910-936-0770

CONNECTICUT

†Arrow Electronics, Inc.
12 Beaumont Road
Wallingford 06492
Tel: (203) 265-7741
TWX: 710-476-0162

Hamilton/Avnet Electronics
Commerce Industrial Park
Commerce Drive
Danbury 06810
Tel: (203) 797-2800
TWX: 710-456-9974

†Pioneer Northeast Electronics
112 Main Street
Norwalk 06851
Tel: (203) 853-1515
TWX: 710-468-3373

FLORIDA

†Arrow Electronics, Inc.
350 Fairway Drive
Deerfield Beach 33441
Tel: (305) 429-8200
TWX: 510-955-9456

Arrow Electronics, Inc.
1001 N.W. 62nd St., Ste. 108
Ft. Lauderdale 33309
Tel: (305) 776-7790
TWX: 510-955-9456

†Arrow Electronics, Inc.
50 Woodlake Drive W., Bldg. B
Palm Bay 32905
Tel: (305) 725-1480
TWX: 510-959-6337

†Hamilton/Avnet Electronics
6801 N.W. 15th Way
Ft. Lauderdale 33309
Tel: (305) 971-2900
TWX: 510-956-3097

Hamilton/Avnet Electronics
3197 Tech Drive North
St. Petersburg 33702
Tel: (813) 576-3930
TWX: 810-863-0374

Hamilton/Avnet Electronics
6947 University Boulevard
Winterpark 32792
Tel: (305) 628-3888
TWX: 810-853-0322

†Pioneer Electronics
337 N. Lake Blvd., Ste. 1000
Alta Monte Springs 32701
Tel: (305) 834-9090
TWX: 810-853-0284

Pioneer Electronics
674 S. Military Trail
Deerfield Beach 33442
Tel: (305) 428-8877
TWX: 510-955-9653

GEORGIA

†Arrow Electronics, Inc.
3155 Northwoods Parkway
Suite A
Norcross 30071
Tel: (404) 449-8252
TWX: 810-766-0439

Hamilton/Avnet Electronics
5825 D. Peachtree Corners
Norcross 30092
Tel: (404) 447-7500
TWX: 810-766-0432

Pioneer Electronics
3100 F. Northwoods Place
Norcross 30071
Tel: (404) 448-1711
TWX: 810-766-4515

ILLINOIS

†Arrow Electronics, Inc.
2000 E. Alonquin Street
Schaumberg 60195
Tel: (312) 397-3440
TWX: 910-291-3544

†Hamilton/Avnet Electronics
1130 Thorndale Avenue
Bensenville 60106
Tel: (312) 860-7780
TWX: 910-227-0060

Kierulff Electronics, Inc.
1140 W. Thorndale
Itasca 60143
Tel: (312) 250-0500

ILLINOIS (Cont'd)

MTI Systems Sales
1100 West Thorndale
Itasca 60143
Tel: (312) 773-2300

†Pioneer Electronics
1551 Carmen Drive
Elk Grove Village 60007
Tel: (312) 437-9680
TWX: 910-222-1834

INDIANA

†Arrow Electronics, Inc.
2495 Directors Row, Suite H
Indianapolis 46241
Tel: (317) 243-9353
TWX: 810-341-3119

Hamilton/Avnet Electronics
485 Gradle Drive
Carmel 46032
Tel: (317) 844-9333
TWX: 810-260-3966

†Pioneer Electronics
6408 Castleplace Drive
Indianapolis 46250
Tel: (317) 849-7300
TWX: 810-260-1794

KANSAS

†Hamilton/Avnet Electronics
9219 Quivera Road
Overland Park 66215
Tel: (913) 888-8900
TWX: 910-743-0005

Pioneer Electronics
10551 Lackman Rd.
Lenexa 66215
Tel: (913) 492-0500

KENTUCKY

Hamilton/Avnet Electronics
1051 D. Newton Park
Lexington 40511
Tel: (606) 259-1475

MARYLAND

Arrow Electronics, Inc.
8300 Gulford Road #H
Rivers Center
Columbia 21046
Tel: (301) 995-0003
TWX: 710-236-9005

†Hamilton/Avnet Electronics
6822 Oak Hall Lane
Columbia 21045
Tel: (301) 995-3500
TWX: 710-862-1861

†Mesa Technology Corp.
9720 Patuxentwood Dr.
Columbia 21046
Tel: (301) 720-5020
TWX: 710-828-9702

†Pioneer Electronics
9100 Gaither Road
Gaithersburg 20877
Tel: (301) 921-0660
TWX: 710-828-0545

MASSACHUSETTS

†Arrow Electronics, Inc.
1 Arrow Drive
Woburn 01801
Tel: (617) 933-8130
TWX: 710-393-6770

†Hamilton/Avnet Electronics
10D Centennial Drive
Peabody 01960
Tel: (617) 532-3701
TWX: 710-393-0382

Kierulff Electronics, Inc.
13 Fortune Dr.
Billerica 01821
Tel: (617) 667-8331

MTI Systems Sales
13 Fortune Drive
Billerica 01821

Pioneer Northeast Electronics
44 Hartwell Avenue
Lexington 02173
Tel: (617) 861-9200
TWX: 710-326-6617

MICHIGAN

Arrow Electronics, Inc.
755 Phoenix Drive
Ann Arbor 48104
Tel: (313) 971-8220
TWX: 810-223-6020

†Hamilton/Avnet Electronics
32487 Schoolcraft Road
Livonia 48150
Tel: (313) 522-4700
TWX: 810-242-8775

Hamilton/Avnet Electronics
2215 29th Street S.E.
Space A5
Grand Rapids 49508
Tel: (616) 243-8805
TWX: 810-273-6921

Pioneer Electronics
4505 Broadmoor Ave. S.E.
Grand Rapids 49508
Tel: (616) 555-1800

†Pioneer Electronics
13485 Stamford
Livonia 48150
Tel: (313) 525-1800
TWX: 810-242-3271

MINNESOTA

†Arrow Electronics, Inc.
5230 W. 73rd Street
Edina 55435
Tel: (612) 830-1800
TWX: 910-576-3125

Hamilton/Avnet Electronics
12400 White Water Drive
Minnetonka 55343
Tel: (612) 932-0600
TWX: (910) 576-2720

†Pioneer Electronics
10203 Bren Road East
Minnetonka 55343
Tel: (612) 935-5444
TWX: 910-576-2738

MISSOURI

†Arrow Electronics, Inc.
2380 Schuetz
St. Louis 63141
Tel: (314) 567-6888
TWX: 910-764-0882

†Hamilton/Avnet Electronics
13743 Shoreline Court
Earth City 63045
Tel: (314) 344-1200
TWX: 910-762-0684

Kierulff Electronics, Inc.
11804 Borman Dr.
St. Luis 63146
Tel: (314) 997-4956

NEW HAMPSHIRE

†Arrow Electronics, Inc.
3 Perimeter Road
Manchester 03103
Tel: (603) 668-6968
TWX: 710-220-1684

Hamilton/Avnet Electronics
444 E. Industrial Drive
Manchester 03104
Tel: (603) 624-9400

NEW JERSEY

†Arrow Electronics, Inc.
6000 Lincoln East
Marlton 08053
Tel: (609) 596-8000
TWX: 710-897-0829

†Arrow Electronics, Inc.
2 Industrial Road
Fairfield 07006
Tel: (201) 575-5300
TWX: 710-998-2206

†Hamilton/Avnet Electronics
1 Keystone Avenue
Bldg. 36
Cherry Hill 08003
Tel: (609) 424-0110
TWX: 710-940-0262

---

†Microcomputer System Technical Distributor Centers

DOMESTIC DISTRIBUTORS

NEW JERSEY (Cont'd)

†Hamilton/Avnet Electronics
10 Industrial
Fairfield 07006
Tel: (201) 575-3390
TWX: 701-734-4388

†Pioneer Northeast Electronics
45 Route 46
Pinebrook 07058
Tel: (201) 575-3510
TWX: 710-734-4382

†MTI Systems Sales
383 Route 46 W
Fairfield 07006
Tel: (201) 227-5552

NEW MEXICO

Alliance Electronics Inc.
11030 Cochiti S.E.
Albuquerque 87123
Tel: (505) 292-3360
TWX: 910-989-1151

Hamilton/Avnet Electronics
2524 Baylor Drive S.E.
Albuquerque 87106
Tel: (505) 765-1500
TWX: 910-989-0614

NEW YORK

Arrow Electronics, Inc.
25 Hub Drive
Melville 11747
Tel: (516) 694-6800
TWX: 510-224-6126

†Arrow Electronics, Inc.
3375 Brighton-Henrietta Townline Rd.
Rochester 14623
Tel: (716) 427-0300
TWX: 510-253-4766

Arrow Electronics, Inc.
7705 Maltage Drive
Liverpool 13088
Tel: (315) 652-1000
TWX: 710-545-0230

Arrow Electronics, Inc.
20 Oser Avenue
Hauppauge 11788
Tel: (516) 231-1000
TWX: 510-227-6623

Hamilton/Avnet Electronics
333 Metro Park
Rochester 14623
Tel: (716) 475-9130
TWX: 510-253-5470

†Hamilton/Avnet Electronics
103 Twin Oaks Drive
Syracuse 13206
Tel: (315) 437-2641
TWX: 710-541-1560

†Hamilton/Avnet Electronics
933 Motor Parkway
Hauppauge 11788
Tel: (516) 231-9800
TWX: 510-224-6166

†MTI Systems Sales
38 Harbor Park Drive
P.O. Box 271
Port Washington 11050
Tel: (516) 621-6200
TWX: 510-223-0846

†Pioneer Northeast Electronics
1806 Vestal Parkway East
Vestal 13850
Tel: (607) 748-8211
TWX: 510-252-0893

†Pioneer Northeast Electronics
60 Crossway Park West
Woodbury, Long Island 11797
Tel: (516) 921-8700
TWX: 510-221-2184

NEW YORK (Cont'd)

†Pioneer Northeast Electronics
840 Fairport Park
Fairport 14450
Tel: (716) 381-7070
TWX: 510-253-7001

NORTH CAROLINA

†Arrow Electronics, Inc.
5240 Greendairy Road
Raleigh 27604
Tel: (919) 876-3132
TWX: 510-928-1856

†Hamilton/Avnet Electronics
3510 Spring Forest Drive
Raleigh 27604
Tel: (919) 878-0819
TWX: 510-928-1836

Pioneer Electronics
9801 A-Southern Pine Blvd.
Charlotte 28210
Tel: (704) 527-8188
TWX: 810-621-0366

OHIO

Arrow Electronics, Inc.
7620 McEwen Road
Centerville 45459
Tel: (513) 435-5563
TWX: 810-459-1611

†Arrow Electronics, Inc.
6238 Cochran Road
Solon 44139
Tel (216) 248-3990
TWX: 810-427-9409

Hamilton/Avnet Electronics
777 Brookedge Blvd.
Westerville 43081
Tel: (614) 882-7004

†Hamilton/Avnet Electronics
954 Senate Drive
Dayton 45459
Tel: (513) 433-0610
TWX: 810-450-2531

†Hamilton/Avnet Electronics
4588 Emery Industrial Parkway
Warrensville Heights 44128
Tel: (216) 831-3500
TWX: 810-427-9452

†Pioneer Electronics
4433 Interpoint Blvd.
Dayton 45424
Tel: (513) 236-9900
TWX: 810-459-1622

†Pioneer Electronics
4800 E. 131st Street
Cleveland 44105
Tel: (216) 587-3600
TWX: 810-422-2211

OKLAHOMA

Arrow Electronics, Inc.
4719 S. Memorial Drive
Tulsa 74145
Tel: (918) 665-7700

OREGON

†Almac Electronics Corporation
1885 N.W. 169th Place
Beaverton 97006
Tel: (503) 629-8090
TWX: 910-467-8743

†Hamilton/Avnet Electronics
6024 S.W. Jean Road
Bldg. C, Suite 10
Lake Oswego 97034
Tel: (503) 635-7848
TWX: 910-455-8179

OREGON (Cont'd)

Wyle Distribution Group
5250 N.E. Elam Young Parkway
Suite 600
Hillsboro 97124
Tel: (503) 640-6000
TWX: 910-460-2203

PENNSYLVANIA

Arrow Electronics, Inc.
650 Seco Road
Monroeville 15146
Tel: (412) 856-7000

Hamilton/Avnet Electronics
2800 Liberty Ave., Bldg. E
Pittsburg 15238
Tel: (412) 281-4150

Pioneer Electronics
259 Kappa Drive
Pittsburgh 15238
Tel: (412) 782-2300
TWX: 710-795-3122

†Pioneer Electronics
261 Gibralter Road
Horsham 19044
Tel: (215) 674-4000
TWX: 510-665-6778

TEXAS

†Arrow Electronics, Inc.
3220 Commander Drive
Carrollton 75006
Tel: (214) 380-6464
TWX: 910-860-5377

†Arrow Electronics, Inc.
10899 Kinghurst
Suite 100
Houston 77099
Tel: (713) 530-4700
TWX: 910-880-4439

†Arrow Electronics, Inc.
10125 Metropolitan
Austin 78758
Tel: (512) 835-4180
TWX: 910-874-1348

†Hamilton/Avnet Electronics
2401 Rutland
Austin 78758
Tel: (512) 837-8911
TWX: 910-874-1319

†Hamilton/Avnet Electronics
2111 W. Walnut Hill Lane
Irving 75062
Tel: (214) 659-4100
TWX: 910-860-5929

†Hamilton/Avnet Electronics
4850 Wright Road #190
Stafford 77477
Tel: (713) 780-1771
TWX: 910-881-5523

Kierulff Electronics, Inc.
9610 Skillman
Dallas 75243
Tel: (214) 343-2400

†Pioneer Electronics
1826 D. Kramer Lane
Austin 78758
Tel: (512) 835-4000
TWX: 910-874-1323

†Pioneer Electronics
13710 Omega Road
Dallas 75234
Tel: (214) 386-7300
TWX: 910-850-5563

†Pioneer Electronics
5853 Point West Drive
Houston 77036
Tel: (713) 988-5555
TWX: 910-881-1606

UTAH

†Hamilton/Avnet Electronics
1585 West 2100 South
Salt Lake City 84119
Tel: (801) 972-2800
TWX: 910-925-4018

Kierulff Electronics, Inc.
1946 W. Parkway Blvd.
Salt Lake City 84119
Tel: (801) 973-6913

Wyle Distribution Group
1325 West 2200 South
Suite E
Salt Lake City 84119
Tel: (801) 974-9953

WASHINGTON

†Almac Electronics Corp.
14360 S.E. Eastgate Way
Bellevue 98007
Tel: (206) 643-9992
TWX: 910-444-2067

Arrow Electronics, Inc.
14320 N.E. 21st Street
Bellevue 98007
Tel: (206) 643-4800
TWX: 910-444-2017

Hamilton/Avnet Electronics
14212 N.E. 21st Street
Bellevue 98005
Tel: (206) 453-5874
TWX: 910-443-2469

Wyle Distribution Group
1750 132nd Ave., N.E.
Bellvue 98005
Tel: (206) 453-8300

WISCONSIN

†Arrow Electronics, Inc.
430 W. Rausson Avenue
Oakcreek 53154
Tel: (414) 764-6600
TWX: 910-262-1193

Hamilton/Avnet Electronics
2975 Moorland Road
New Berlin 53151
Tel: (414) 784-4510
TWX: 910-262-1182

Kierulff Electronics, Inc.
2238-E W. Bluemound Rd.
Waukeshaw 53186
Tel: (414) 784-8160

CANADA

ALBERTA

Hamilton/Avnet Electronics
2816 21st Street N.E.
Calgary T2E 6Z2
Tel: (403) 250-9380
TWX: 03-827-642

Hamilton/Avnet Electronics
6845 Rexwood Road Unit 6
Mississauga, Ontario L4V1R2
Tel: (416) 677-0484

Zentronics
6815 8th Street, N.E., Ste. 100
Calgary T2E 7H7
Tel: (403) 295-8838

BRITISH COLUMBIA

Hamilton/Avnet Electronics
105-2550 Boundary Road
Burnaby V5M 3Z3
Tel: (604) 437-6667

BRITISH COLUMBIA (Cont'd)

Zentronics
108-11400 Bridgeport Road
Richmond V6X 1T2
Tel: (604) 273-5575
TWX: 04-5077-89

MANITOBA

Zentronics
60-1313 Border Street
Winnipeg R3H 0X4
Tel: (204) 694-1957
FAX: (204) 633-9255

ONTARIO

Arrow Electronics Inc.
24 Martin Ross Avenue
Downsview M3J 2K9
Tel: (416) 661-0220
TLX: 06-218213

Arrow Electronics Inc.
148 Colonnade Road
Nepean K2E 7J5
Tel: (613) 226-6903

†Hamilton/Avnet Electronics
6845 Rexwood Road
Units G & H
Mississauga L4V 1R2
Tel: (416) 677-7432
TWX: 610-492-8867

†Hamilton/Avnet Electronics
210 Colonnade Road South
Nepean K2E 7L5
Tel: (613) 226-1700
TWX: 05-349-71

†Zentronics
8 Tilbury Court
Brampton L6T 3T4
Tel: (416) 451-9600
TWX: 06-976-78

Zentronics
564/10 Weber Street North
Waterloo N2L 5C6
Tel: (519) 884-5700

†Zentronics
155 Colonnade Road
Unit 17
Nepean K2E 7K1
Tel: (613) 225-8840
TWX: 06-976-78

SASKATCHEWAN

Zentronics
173-1222 Alberta Avenue
Saskatoon S7K 1R4
Tel: (306) 955-2202, 2207
FAX: (306) 244-3731

QUEBEC

†Arrow Electronics Inc.
4050 Jean Talon Quest
Montreal H4P 1W1
Tel: (514) 735-5511
TLX: 05-25596

Arrow Electronics Inc.
909 Charest Blvd.
Quebec 61N 269
Tel: (418) 687-4231
TLX: 05-13388

Hamilton/Avnet Electronics
2795 Rue Halpern
St. Laurent H4S 1P8
Tel: (514) 335-1000
TWX: 610-421-3731

Zentronics
505 Locke Street
St. Laurent H4T 1X7
Tel: (514) 735-5361
TWX: 05-827-535

---

†Microcomputer System Technical Distributor Centers

intel

DOMESTIC SERVICE OFFICES

ALABAMA

Intel Corp.
5015 Bradford Drive, #2
Huntsville 35805
Tel: (205) 830-4010

ARIZONA

Intel Corp.
11225 N. 28th Dr., #D214
Phoenix 85029
Tel: (602) 869-4980

Intel Corp.
500 E. Fry Blvd., Suite M-15
Sierra Vista 85635
Tel: (602) 459-5010

ARKANSAS

Intel Corp.
P.O. Box 206
Ulm 72170
Tel: (501) 241-3264

CALIFORNIA

Intel Corp.
21515 Vanowen St.
Suite 116
Canoga Park 91303
Tel: (818) 704-8500

Intel Corp.
2250 E. Imperial Highway
Suite 218
El Segundo 90245
Tel: 1-800-468-3548

Intel Corp.
1900 Prairie City Rd.
Folsom 95630-9597
Tel: (916) 351-6143

Intel Corp.
2000 E. 4th Street
Suite 110
Santa Ana 92705
Tel: (714) 835-5789
TWX: 910-595-2475

Intel Corp.
2700 San Tomas Expressway
Santa Clara 95051
Tel: (408) 970-1740

Intel Corp.
4350 Executive Drive
Suite 105
San Diego 92121
Tel: (619) 452-5880

COLORADO

Intel Corp.
650 South Cherry
Suite 915
Denver 80222
Tel: (303) 321-8086
TWX: 910-931-2289

CONNECTICUT

Intel Corp.
26 Mill Plain Road
Danbury 06811
Tel: (203) 748-3130
TWX: 710-456-1199

FLORIDA

Intel Corp.
1500 N.W. 62, Suite 104
Ft. Lauderdale 33309
Tel: (305) 771-0600
TWX: 510-956-9407

Intel Corp.
242 N. Westmonte Drive
Suite 105
Altamonte Springs 32714
Tel: (305) 869-5588

GEORGIA

Intel Corp.
3280 Pointe Parkway
Suite 200
Norcross 30092
Tel: (404) 441-1171

ILLINOIS

Intel Corp.
300 N. Martingale Rd.
Suite 300
Schaumburg 60194
Tel: (312) 310-5733

INDIANA

Intel Corp.
8777 Purdue Rd., #125
Indianapolis 46268
Tel: (317) 875-0623

KANSAS

Intel Corp.
8400 W. 110th Street
Suite 170
Overland Park 66210
Tel: (913) 345-2727

KENTUCKY

Intel Corp.
3525 Tatescreek Road, #51
Lexington 40502
Tel: (606) 272-6745

MARYLAND

Intel Corp.
5th Floor
7833 Walker Drive
Greenbelt 20770
Tel: (301) 441-1020

MASSACHUSETTS

Intel Corp.
Westford Corp. Center
3 Carlisle Road
Westford 01886
Tel: (617) 692-1060

MICHIGAN

Intel Corp.
7071 Orchard Lake Road
Suite 100
West Bloomfield 48033
Tel: (313) 851-8905

MISSOURI

Intel Corp.
4203 Earth City Expressway
Suite 143
Earth City 63045
Tel: (314) 291-2015

NEW JERSEY

Intel Corp.
385 Sylvan Avenue
Englewood Cliffs 07632
Tel: (201) 567-0821
TWX: 710-991-8593

Intel Corp.
Raritan Plaza III
Raritan Center
Edison 08817
Tel: (201) 225-3000

NORTH CAROLINA

Intel Corp.
2306 W. Meadowview Road
Suite 206
Greensboro 27407
Tel: (919) 294-1541

Intel Corp.
2700 Wycliff Rd., Suite 102
Raleigh 27607
Tel: (919) 781-8022

OHIO

Intel Corp.
Chagrin-Brainard Bldg.
Suite 305
28001 Chagrin Boulevard
Cleveland 44122
Tel: (216) 464-6915
TWX: 810-427-9298

Intel Corp.
6500 Poe
Dayton 45414
Tel: (513) 890-5350

OREGON

Intel Corp.
15254 N.W. Greenbrier Parkway, Bldg. B
Beaverton 01886
Tel: (503) 645-8051
TWX: 910-467-8741

Intel Corp.
5200 N.E. Elam Young Parkway
Hillsboro 97123
Tel: (503) 681-8080

PENNSYLVANIA

Intel Corp.
201 Penn Center Boulevard
Suite 301 W
Pittsburgh 15235
Tel: (313) 354-1540

TEXAS

Intel Corp.
313 E. Anderson Lane
Suite 314
Austin 78752
Tel: (512) 454-3628
TWX: 910-874-1347

Intel Corp.
12300 Ford Road
Suite 380
Dallas 75234
Tel: (214) 241-2820
TWX: 910-860-5617

Intel Corp.
8815 Dyer St., Suite 225
El Paso 79904
Tel: (915) 751-0186

VIRGINIA

Intel Corp.
1603 Santa Rosa Rd., #109
Richmond 23288
Tel: (804) 282-5668

WASHINGTON

Intel Corp.
110 110th Avenue N.E.
Suite 510
Bellevue 98004
Tel: 1-800-468-3548
TWX: 910-443-3002

WISCONSIN

Intel Corp.
330 S. Executive Dr.
Suite 102
Brookfield 53005
Tel: (414) 784-8087

CANADA

Intel Corp.
190 Attwell Drive, Suite 500
Rexdale, Ontario
Canada M9W 6H8
Tel: (416) 675-2105

Intel Corp.
620 St. Jean Blvd.
Pointe Claire, Quebec
Canada H9R 3K3
Tel: (514) 694-9130

Intel Corp.
2650 Queensview Drive, #250
Ottawa, Ontario,
Canada K2B 8H6
Tel: (613) 829-9714

CUSTOMER TRAINING CENTERS

CALIFORNIA

2700 San Tomas Expressway
Santa Clara 95051
Tel: (408) 970-1700

ILLINOIS

300 N. Martingale, #300
Schaumburg 60173
Tel: (312) 310-5700

MASSACHUSETTS

3 Carlisle Road
Westford 01886
Tel: (617) 692-1000

MARYLAND

7833 Walker Dr., 4th Floor
Greenbelt 20770
Tel: (301) 220-3380

SYSTEMS ENGINEERING OFFICES

CALIFORNIA

2700 San Tomas Expressway
Santa Clara 95051
Tel: (408) 986-8086

ILLINOIS

300 N. Martingale, #300
Schaumburg 60173
Tel: (312) 310-8031

MASSACHUSETTS

3 Carlisle Road
Westford 01886
Tel: (617) 692-3222

NEW YORK

300 Motor Parkway
Hauppauge 11788
Tel: (516) 231-3300

EUROPEAN SALES OFFICES

BELGIUM

Intel Corporation S.A.
Rue de Cottages 65
B-1180 Brussels
Tel: (02) 347-0666

DENMARK

Intel Denmark A/S*
Glentevej 61 - 3rd Floor
DK-2400 Copenhagen
Tel: (01) 19-80-33
TLX: 19567

FINLAND

Intel Finland OY
Rousilantie 2
00390 Helsinki
Tel: (8) 0544-644
TLX: 123332

FRANCE

Intel Paris
1 Rue Edison, BP 303
78054 Saint-Quentin-en-Yvelines Cedex
Tel: (33) 1-30-57-7000
TLX: 69901677

Intel Corporation, S.A.R.L.
Immeuble BBC
4 Quai des Etroits
69005 Lyon
Tel: (7) 842-4089
TLX: 305153

WEST GERMANY

Intel Semiconductor GmbH*
Seidlstrasse 27
D-8000 Muenchen 2
Tel: (89) 53891
TLX: 05-23177 INTL D

Intel Semiconductor GmbH
Verkaufsbuero Wiesbaden
Abraham-Lincoln Str. 16-18
6200 Wiesbaden
Tel: (6121) 76050
TLX: 04186183 INTW D

Intel Semiconductor GmbH
Verkaufsbuero Hannover
Hohenzollernstrasse 5
3000 Hannover 1
Tel: (511) 34-40-81
TLX: 923625 INTH D

Intel Semiconductor GmbH
Verkaufsbuero Stuttgart
Bruckstrasse 61
7012 Fellbach
Tel: (711) 58-00-82
TLX: 7254826 INTS D

ISRAEL

Intel Semiconductor Ltd*
Attidim Industrial Park
Neve Sharet
Dvora Hanevia
Bldg. No. 13, 4th Floor
P.O. Box 43202
Tel Aviv 61430
Tel: (3) 491-099, 491-098
TLX: 371215

ITALY

Intel Corporation S.P.A.*
Milanofiori, Palazzo E/4
20090 Assago (Milano)
Tel: (02) 824-4071
TLX: 341286 INTMIL

NETHERLANDS

Intel Semiconductor (Nederland) B.V.*
Alexanderpoort Building
Marten Meesweg 93
3068 Rotterdam
Tel: (10) 21-23-77
TLX: 22283

NORWAY

Intel Norway A/S
P.O. Box 92
Hvamveien 4
N-2013, Skjetten
Tel: 06-842420
TLX: 78018

SPAIN

Intel Iberia
Calle Zurbaran 28-IZQDA
28010 Madrid
Tel: (1) 410-4004
TLX: 46880

SWEDEN

Intel Sweden A.B.*
Dalvagen 24
S-171 36 Solna
Tel: (8) 734-0100
TLX: 12261

SWITZERLAND

Intel Semiconductor A.G.*
Talackerstrasse 17
8152 Glattbrugg
CH-8065 Zurich
Tel: (01) 829-2977
TLX: 57989 ICH CH

UNITED KINGDOM

Intel Corporation (U.K.) Ltd.*
Pipers Way
Swindon, Wiltshire SN3 1RJ
Tel: (0793) 696000
TLX: 444447 INT SWN

EUROPEAN DISTRIBUTORS/REPRESENTATIVES

AUSTRIA

Bacher Elektronics Ges.m.b.H.
Rotenmuehlgasse 26
A-1120 Wien
Tel: (222) 835-6460
TLX: 131532

BELGIUM

Inelco Belgium S.A.
Ave. des Croix de Guerre, 94
Bruxelles 1120
Tel: (02) 216-01-60
TLX: 64475

BENELUX

Koning en Hartman Electrotechniek B.V.
Postbus 125
2600 AC Delft
Tel: (15) 609-906
TLX: 38250

DENMARK

ITT MultiKomponent
Naverland 29
DK-2600 Glostrup
Tel: (02) 456-66-45
TLX: 33355 ITTCG DK

FINLAND

Oy Fintronic AB
Melkonkatu 24A
SF-00210 Helsinki 21
Tel: (0) 692-60-22
TLX: 124224 FTRON SF

FRANCE

Generim
Zone d'Activite de Courtaboeuf
Avenue de la Baltique
91943 Les Ulis Cedex
Tel: (1) 69-07-78-78
TLX: 691700

Jermyn
73-79 Rue des Solets
Silic 585
94663 Rungis Cedex
Tel: (1) 45-60-04-00
TLX: 290967

Metrologie
Tour d'Asnieres
4, Avenue Laurent Cely
92606 Asnieres
Tel: (1) 47-90-62-40
TLX: 611448

FRANCE (Cont'd)

Tekelec Airtronic
Cite des Bruyeres
Rue Carle Vernet BP 2
92310 Sevres
Tel: (1) 45-34-75-35
TLX: 204552

WEST GERMANY

Electronic 2000 Vertriebs AG
Stahlgruberring 12
8000 Muenchen 82
Tel: (089) 42-00-10
TLX: 522561 ELEC D

Jermyn GmbH
Schulstrasse 84
6277 Bad Camberg
Tel: (064) 34-231
TLX: 415257-0 JERM D

Metrologie GmbH
Meglingerstr. 49
8000 Muenchen 71
Tel: (089) 570-940
TLX: 5213189

Metrologie GmbH
Rheinstr. 94-96
6100 Darmstadt
Tel: (06151) 33661
TLX: 176151820

Proelectron Vertriebs AG
Max-Planck-Strasse 1-3
6072 Dreieich
Tel: (06103) 3040
TLX: 417972

ITT-MultiKomponent
Bahnhofstrasse 44
7141 Moeglingen
Tel: (07141) 4879
TLX: 7264399 MUKO D

ISRAEL

Eastronics Ltd.
11 Rosanis Street
P.O. Box 39300
Tel Aviv 61392
Tel: (3) 47-51-51
TLX: 342610 DATIX IL or
33638 RONIX IL

ITALY

Eledra Componenti S.P.A.
Via Giacomo Watt, 37
20143 Milano
Tel: (02) 82821
TLX: 332332

ITALY (Cont'd)

Intesi
Milanofiori E5
20090 Assago
Tel: (02) 824701
TLX: 311351

Lasi Elettronica S.P.A.
Viale Fulvio Testi, 126
20092 Cinisello Balsamo
Tel: (02) 244-0012, 244-0212
TLX: 352040

NORWAY

Nordisk Electronik A/S
Postboks 130
N-1364 Hvalstad
Tel: (2) 846-210
TLX: 77546 NENAS N

PORTUGAL

Ditram
Avenida Marques de Tomar, 46A
Lisboa P-1000
Tel: (1) 545-313
TWX: (0404) 14182

SPAIN

A.T.D. Electronica S.A.
Pl. Ciudad de Viena 6
28040 Madrid
Tel: (1) 234-40-00
TWX: 42477

ITT SESA
21-3 Miguel Angel
Madrid 28010
Tel: (1) 419-54-00
TWX: 27461

SWEDEN

Nordisk Elektronik AB
Box 1409
S-171 27 Solna
Tel: (8) 734-97-70
TLX: 10547

SWITZERLAND

Industrade AG
Hertistrasse 31
CH-8304 Wallisellen
Tel: (01) 830-5040
TLX: 56788

UNITED KINGDOM

Accent Electronic Components Ltd.
Jubilee House, Jubilee Way
Letchworth, Herts SG6 1QH
England
Tel: (0462) 686666
TLX: 626923

Bytech Ltd.
Unit 2 Western Centre
Western Industrial Estate
Bracknell, Berkshire RG12 1RW
England
Tel: (0344) 482211
TLX: 848215

Comway Microsystems Ltd.
John Scott House, Market St.
Bracknell, Berkshire RJ12 1QP
England
Tel: (0344) 55333
TLX: 847201

IBR Microcomputers Ltd.
Unit 2 Western Centre
Western Industrial Estate
Bracknell, Berkshire RG12 1RW
England
Tel: (0344) 486-555
TLX: 849381

Jermyn Industries
Vestry Estate, Otford Road
Sevenoaks, Kent TN14 5EU
England
Tel: (0732) 450144
TLX: 95142

Rapid Silicon
Rapid House, Denmark St.
High Wycombe, Bucks HP11 2ER
England
Tel: (0494) 442266
TLX: 837931

Rapid Systems
Rapid House, Denmark St.
High Wycombe, Bucks HP11 2ER
England
Tel: (0494) 450244
TLX: 837931

Micro Marketing
Glenageary Office Park
Glenageary, Co. Dublin
Ireland
Tel: (0001) 856288
TLX: 31584

YUGOSLAVIA

H.R. Microelectronics Corp.
2005 De La Cruz Blvd., Ste. 223
Santa Clara, CA 95050 U.S.A.
Tel: (408) 988-0286
TLX: 387452

*Field Application Location

INTERNATIONAL SALES OFFICES

AUSTRALIA

Intel Australia Pty. Ltd.*
Spectrum Building
200 Pacific Hwy., Level 6
Crows Nest, NSW, 2065
Tel: (2) 957-2744
TLX: 20097
FAX: (2) 923-2632

BRAZIL

Intel Semicondutores do Brasil LTDA
Av. Paulista, 1159-CJS 404/405
01311 - Sao Paulo - S.P.
Tel: 9-011-55-11-287-5899
TLX: 1153146

CHINA

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

HONG KONG

Intel Semiconductor Ltd.*
1701-3 Connaught Centre
1 Connaught Road
Tel: (5) 844-4555
TWX: 63869 ISLHK HX
FAX: (5) 294-589

JAPAN

Intel Japan K.K.
5-6 Tokodai Toyosato-machi
Tsukuba-gun, Ibaraki-ken 300-26
Tel: 029747-8511
TLX: 3656-160
FAX: 029747-8450

Intel Japan K.K.*
Daiichi Mitsugi Bldg.
1-8889 Fuchu-cho
Fuchu-shi, Tokyo 183
Tel: 0423-60-7871
FAX: 0423-60-0315

Intel Japan K.K.*
Flower-Hill Shin-machi Bldg.
1-23-9 Shinmachi
Setagaya-ku, Tokyo 154
Tel: 03-426-2231
FAX: 03-427-7620

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.*
Mitsui-Seimei Musashi-kosugi Bldg.
915 Shinmaruko, Nakahara-ku
Kawasaki-shi, Kanagawa 211
Tel: 044-733-7011
FAX: 044-733-7010

JAPAN (Cont'd)

Intel Japan K.K.
Nihon Seimei Atsugi Bldg.
1-2-1 Asahi-machi
Atsugi-shi, Kanagawa 243
Tel: 0462-29-3731
FAX: 0462-29-3781

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: 03-201-3621
FAX: 03-201-6850

Intel Japan K.K.
Tokai Bldg.
1-16-30 Meieki Minami
Nakamura-ku, Nagoya-shi
Aichi 450
Tel: 052-561-5181
FAX: 052-561-5317

KOREA

Intel Technology Asia Ltd.
Room 906, Singsong Bldg.
25-4, Yoido-Dong, Youngdeungpo-ku
Seoul 150
Tel: (2) 784-8186
TLX: 29312 INTELKO
FAX: (2) 784-8096

SINGAPORE

Intel Singapore Technology, Ltd.
101 Thomson Road #21-06
Goldhill Square
Singapore 1130
Tel: 250-7811
TLX: 39921 INTEL
FAX: 250-9256

TAIWAN

Intel Technology (Far East) Ltd.
Taiwan Branch
10/F., No. 205, Tun Hua N. Road
Taipei, R.O.C.
Tel: 886-2-716-9660
TLX: 13159 INTELTWN
FAX: 886-2-717-2455

INTERNATIONAL DISTRIBUTORS/REPRESENTATIVES

ARGENTINA

VLC S.R.L. Bartalome Mitre 1711
3 Piso
1037 Buenos Aires
Tel: 54-1-49-2092
TLX: 17575 EDARG-AR

AUSTRALIA

Total Electronics
Private Bag 250
9 Harker Street
Burwood, Victoria 3125
Tel: 61-3-288-4044
TLX: AA 31261

Total Electronics
P.O. Box 139
Artamon, N.S.W. 2064
Tel: 61-02-438-1855
TLX: 26297

BRAZIL

Elebra Microelectronica S/A
Geraldo Flausino Gomes, 78
9 Andar
04575 - Sao Paulo - S.P.
Tel: 55-11-534-9600
TLX: 3911125131 ELBR BR
FAX: 55-11-534-9424

CHILE

DIN Instruments
Suecia 2323
Casilla 6055, Correo 22
Santiago
Tel: 56-2-225-8139
TLX: 440422 RUDY CZ

CHINA

Novel Precision Machinery Co., Ltd.
Flat D, 20 Kingsford Ind. Bldg.
Phase 1, 26 Kwai Hei Street
N.T., Kowloon
Hong Kong
Tel: 852-0-223-222
TWX: 39114 JINMI HX
FAX: 852-0-261-602

CHINA (Cont'd)

Schmidt & Co. Ltd.
18/F Great Eagle Centre
23 Harbour Road
Wanchai, Hong Kong
Tel: 852-5-833-0222
TWX: 74766 SCHMC HX
FAX: 852-5-891-8754

INDIA

Micronic Devices
Arun Complex
No. 65 D.V.G. Road
Basavanagudi
Bangalore 560 004
Tel: 91-812-600-631
TLX: 0845-8332 MD BG IN

Micronic Devices
403, Gagan Deep
12, Rajendra Place
New Delhi 110 008
Tel: 91-58-97-71
TLX: 03163235 MDND IN

Micronic Devices
No. 516 5th Floor
Swastik Chambers
Sion, Chambray Road
Bombay 400 071
Tel: 91-52-39-63
TLX: 9531 171447 MDEV IN

JAPAN

Asahi Electronics Co. Ltd.
KMM Bldg. 2-14-1 Asano
Kokurakita-ku
Kitakyushu-shi 802
Tel: 093-511-6471
FAX: 093-551-7861

C. Itoh Techno-Science Co., Ltd.
C. Itoh Bldg., 2-5-1 Kita-Aoyama
Minato-ku, Tokyo 107
Tel: 03-497-4840
FAX: 03-497-4969

JAPAN (Cont'd)

Dia Semicon Systems, Inc.
Wacore 64, 1-37-8 Sangenjaya
Setagaya-ku, Tokyo 154
Tel: 03-487-0386
FAX: 03-487-8088

Okaya Koki
2-4-18 Sakae
Naka-ku, Nagoya-shi 460
Tel: 052-204-2911
FAX: 052-204-2901

Ryoyo Electro Corp.
Konwa Bldg.
1-12-22 Tsukiji
Chuo-ku, Tokyo 104
Tel: 03-546-5011
FAX: 03-546-5044

KOREA

J-Tek Corporation
6th Floor, Government Pension Bldg.
24-3 Yoido-Dong
Youngdeungpo-ku
Seoul 150
Tel: 82-2-782-8039
TLX: 25299 KODIGIT
FAX: 82-2-784-8391

Samsung Semiconductor &
Telecommunications Co., Ltd.
150, 2-KA, Tafpyung-ro, Chung-ku
Seoul 100
Tel: 82-2-751-3987
TLX: 27970 KORSST
FAX: 82-2-753-0967

MEXICO

Dicopel S.A.
Tochtli 368 Fracc. Ind. San Antonio
Azcapotzalco
C.P. 02760-Mexico, D.F.
Tel: 52-5-561-3211
TLX: 1773790 DICOME

NEW ZEALAND

Northrup Instruments & Systems Ltd.
459 Kyber Pass Road
P.O. Box 9464, Newmarket
Auckland 1
Tel: 64-9-501-219, 501-801
TLX: 21570 THERMAL

Northrup Instruments & Systems Ltd.
P.O. Box 2406
Wellington 856658
Tel: 64-4-856-658
TLX: NZ3380
FAX: 64-4-857276

SINGAPORE

Francotone Electronics Pte Ltd.
17 Harvey Road #04-01
Singapore 1336
Tel: 283-0888, 289-1618
TWX: 56541 FRELS
FAX: 2895327

SOUTH AFRICA

Electronic Building Elements, Pty. Ltd.
P.O. Box 4609
Pine Square, 18th Street
Hazelwood, Pretoria 0001
Tel: 27-12-469921
TLX: 3-227786 SA

TAIWAN

Mitac Corporation
No. 585, Ming Shen East Rd.
Taipei, R.O.C.
Tel: 886-2-501-8231
FAX: 886-2-501-4265

VENEZUELA

P. Benavides S/A
Avilanes a Rio
Residencias Kamarata
Locales 4 A17
La Candelaria, Caracas
Tel: 58-2-571-0396
TLX: 28450 PBVEN VC
FAX: 58-2-572-3321

---

*Field Application Location

UNITED STATES
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

JAPAN
Intel Japan K.K.
5-6 Tokodai Toyosato-machi
Tsukuba-gun, Ibaraki-ken 300-26

FRANCE
Intel Paris
1 Rue Edison, BP 303
78054 Saint-Quentin-en-Yvelines Cedex

UNITED KINGDOM
Intel Corporation (U.K.) Ltd.
Pipers Way
Swindon
Wiltshire, England SN3 1RJ

WEST GERMANY
Intel Semiconductor GmbH
Seidlstrasse 27
D-8000 Muenchen 2

HONG KONG
Intel Semiconductor Ltd.
1701-3 Connaught Centre
1 Connaught Road

CANADA
Intel Semiconductor of Canada, Ltd.
190 Attwell Drive, Suite 500
Rexdale, Ontario M9W 6H8

Printed in U.S.A./0787/C87-222/33K/RRD LD
Microprocessors

ISBN 1-55512-069-5