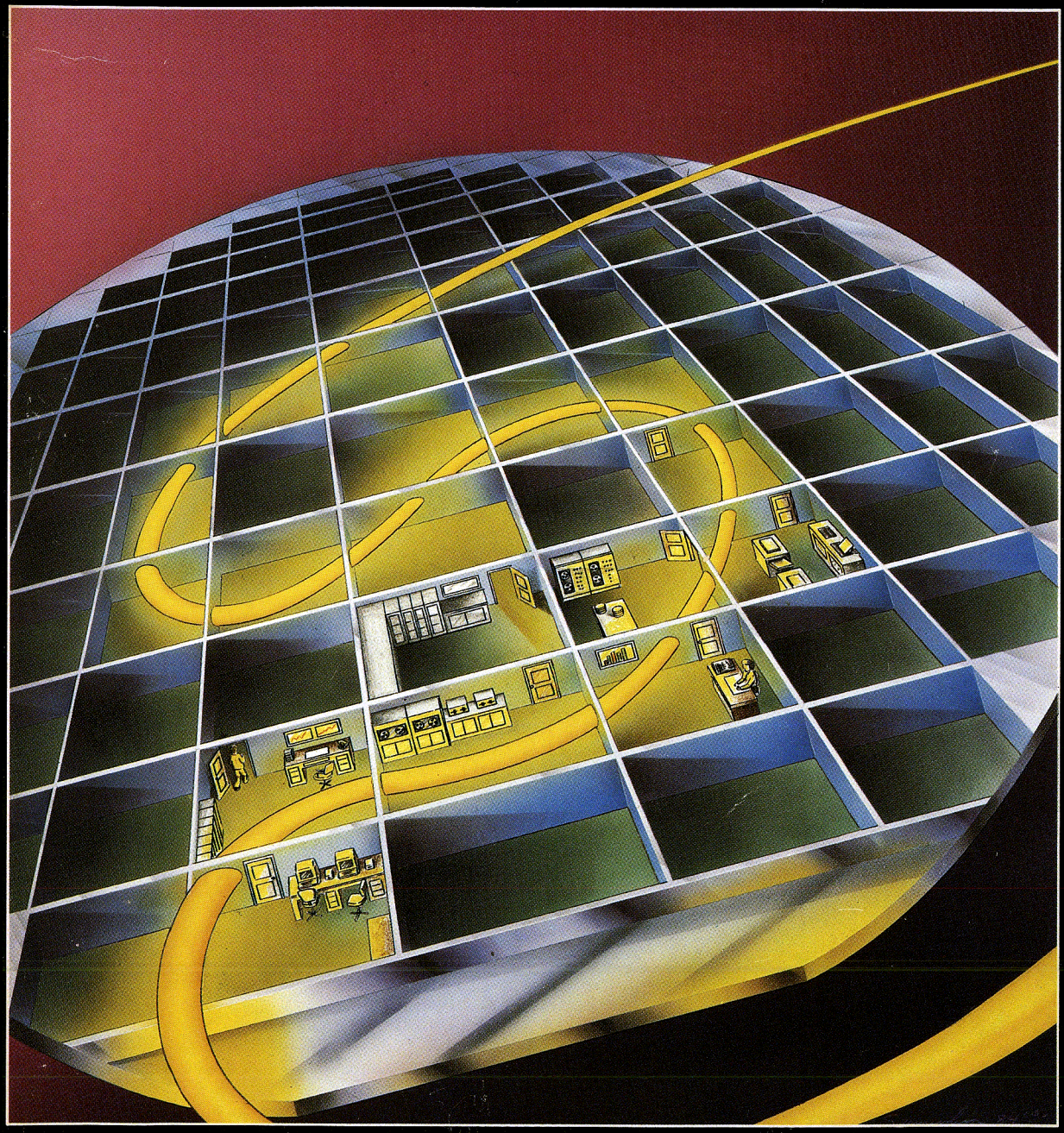
intel

Local Area Networking (LAN) Component User's Manual



Order Number: 230814-004

LITERATURE

To order Intel literature write or call:

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

Toll Free Number:
(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 customers outside the U.S. Prices are subject to change.

1988 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 8 HANDBOOKS Save $50.00 off the retail price of $175.00	**231003**	**$125.00**
AUTOMOTIVE HANDBOOK (Not included in handbook Set)	231792	**$20.00**
COMPONENTS QUALITY/RELIABILITY HANDBOOK (Available in July)	210997	**$20.00**
EMBEDDED CONTROLLER HANDBOOK (2 Volume Set)	210918	**$23.00**
MEMORY COMPONENTS HANDBOOK	210830	**$18.00**
MICROCOMMUNICATIONS HANDBOOK	231658	**$22.00**
MICROPROCESSOR AND PERIPHERAL HANDBOOK (2 Volume Set)	230843	**$25.00**
MILITARY HANDBOOK (Not included in handbook Set)	210461	**$18.00**
OEM BOARDS AND SYSTEMS HANDBOOK	280407	**$18.00**
PROGRAMMABLE LOGIC HANDBOOK	296083	**$18.00**
SYSTEMS QUALITY/RELIABILITY HANDBOOK	231762	**$20.00**
PRODUCT GUIDE Overview of Intel's complete product lines	210846	N/C
DEVELOPMENT TOOLS CATALOG	280199	N/C
INTEL PACKAGING OUTLINES AND DIMENSIONS Packaging types, number of leads, etc.	231369	N/C
LITERATURE PRICE LIST List of Intel Literature	210620	N/C

*Good in the U.S. and Canada

**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

LAN COMPONENTS USER'S MANUAL

1988

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, $\hat{i}$, ICE, iCEL, iCS, iDBP, iDIS, I²ICE, 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, MAP-NET, 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 1988

Table of Contents

SECTION ONE—LOCAL AREA NETWORKING INTRODUCTION

CHAPTER 1

Introduction 1-1

SECTION TWO—82586 LOCAL AREA NETWORK COPROCESSOR

CHAPTER 2

82586 Reference Manual

82586 LAN Coprocessor 2-1
Programming the 82586 2-65
82586 Application Examples 2-83

CHAPTER 3

82586 Application Notes

82586 Data Link Driver AP-235 3-1
Implementing Ethernet/Cheapernet with the Intel 82586 LAN Controller 3-81

SECTION THREE—82588 HIGH INTEGRATION LAN CONTROLLER

CHAPTER 4

82588 Reference Manual 4-1

CHAPTER 5

Implementing StarLAN with the Intel 82588 Controller AP-236 5-1

SECTION FOUR—ADVANCED CSMA/CD LAN CONTROLLERS

CHAPTER 6—82590/92 REFERENCE MANUAL

Chapter 1—Introduction 6-12
Chapter 2—Framing and Link Management 6-24
Chapter 3—Network Management and Diagnostic Functions 6-49
Chapter 4—System Interface 6-53
Chapter 5—Programming Model 6-58
Chapter 6—Initialization/Configuration 6-65
Chapter 7—Controlling the 82590 through Port 0 6-88
Chapter 8—Controlling the 82590 through Port 1 6-101
Chapter 9—Port 0 Operations and Status 6-109
Chapter 10—Port 1 Operations 6-141
Chapter 11—Performance Considerations 6-151
Chapter 12—Applications 6-156

SECTION FIVE—DATA SHEETS

CHAPTER 7

Local Area Network Data Sheets

82586 Local Area Network Coprocessor 7-1
82501 Ethernet Serial Interface 7-38
82502 Ethernet Transceiver Chip 7-51
82588 **High Integration LAN Controller** 7-64
82560 Host Interface and Memory Manager Architectural Overview 7-90
82590 Advanced CSMA/CD LAN Controller with 8-bit Data Path 7-100
82592 Advanced CSMA/CD LAN Controller with 16-bit Data Path 7-133

SECTION SIX—GLOSSARY

CHAPTER 8

Glossary 8-1

Introduction 1

CHAPTER 1
INTRODUCTION

OVERVIEW

Through the 1970's, with the evolution of the microprocessor, the concept of distributed processing became ever more attractive. The ability to integrate the intelligence of yesterday's boards onto todays chips drove the cost of computing to a point where decentralization of processing was not only feasible, but practical as well. In the 1980's, we have seen the advent of engineering workstations, instrumentation, personal computers, printers, and much more, all with the capability of performing their respective tasks independently of any supervisory machine. This decentralization of computing power has brought about the need to interconnect the different "nodes" in order to fulfill the following requirements. First, by interconnecting, we allow for the sharing of expensive peripheral resources like printers, plotters and file servers. Second, we allow for the sharing of information via common data bases. Finally, high level services, such as electronic mail, are supplied. Thus was the concept of networking born.

The definition of a network in its broadest sense is a system of processing units connected by communication lines. This definition can be broken down further based on the characteristics of a specific network. One special type of network is a Local Area Network or LAN. A LAN is defined as being a network supporting peer-to-peer communication over distances of tens of meters to several kilometers. Peer-to-peer implies that each station on the network is its own boss, i.e., there is no master-slave relationship. This chapter describes Local Area Networks vis-a-vis open system architecture. Also, a discussion of existing and emerging Local Area Network standards is presented, followed by an overview of Intel's Local Area Network components and related products and how they map into the existing and emerging standards.

LAN Requirements

Figure 1 depicts a Local Area Network model which is becoming representative of the office automation environment. Tier 1 consists of departmental clusters linking, for example, a group of PCs in the Finance department or several process control stations on the factory floor. Tier 2 provides the interface between various computers and departmental clusters, plus interconnection to system resources such as file servers, print servers, and communications servers.

Just as the different tiers have varied requirements, so, too, will the characteristics of each department's LAN vary depending on which organization they serve within the company. In general, the different departments within most companies can be divided into three distinct environments: factory, engineering, and office. On the factory floor, where a typical application is networking process control stations and robotics controllers, a network must be able to span a large distance while maintaining a high degree of noise immunity. Futhermore, the ability for each node to send a message must be deterministic, in that each must have a chance to transmit in a given interval of time. For engineering applications, in which CAD/CAE workstations are interconnected, the required network characteristics are high throughput over short to moderate distances. Finally, in the office the primary need is the interconnection of personal computers and servers. This application is highly cost sensitive but doesn't need the same high throughput or distance as in the factory. Because of the different applications and varied requirements of each, no single LAN can meet all needs. As such, many LANs have evolved over the last five to ten years.

There are five attributes that a Local Area Network must have in order to insure its own longevity. First, it must be backed by one or more major companies. Not only will the associated companies add credibility to the network, but they also provide the capital to cover development costs. Second, the existence of VLSI is essential. VLSI reduces cost and simplifies the design. Standardized software is the third essential ingredient; it allows for standard interfaces in addition to minimizing time-to-market. Fourth, the network must be easy to install and manage. Finally, and most importantly, the network itself must be an industry standard. The end user needs to be able to purchase a personal computer from one company, a file server from a second and a printer from a third, interconnect them by simply "plugging them in" and not have to worry about low level concerns such as protocols or media.

The International Standards Organization (ISO) and the IEEE have for several years been developing the models on which networking standards are based as well as the standards themselves. The next section overviews the activities of these two bodies in generating networking standards.

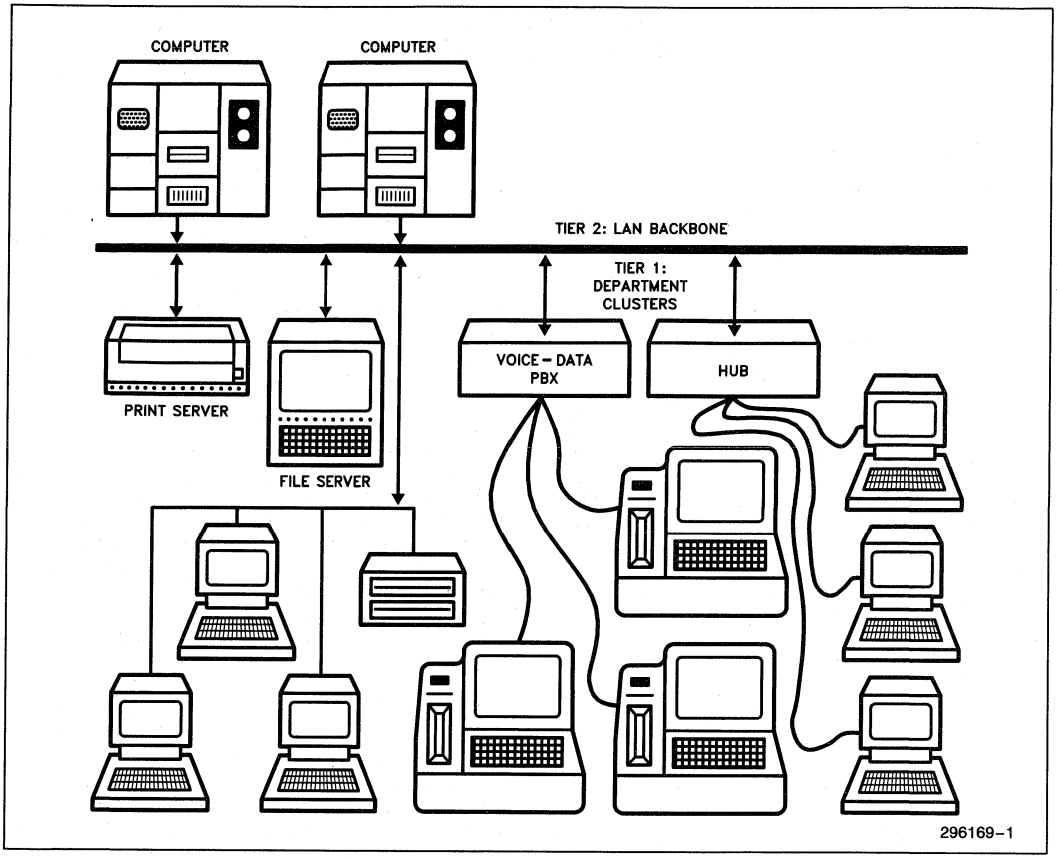


Figure 1. LAN Model

1.0 NETWORKING SOLUTIONS VIA STANDARDS

Networking is not a new concept, but until fairly recently, networks were of a proprietary nature. In the past, an end user had to buy all of its computing equipment from the same vendor in order to interconnect it. This problem was solved when the concept of "open systems" was developed. An open system is built using widely accepted standards. Open systems allow the end user to purchase equipment from several vendors in order to realize an optimal solution for any given application. Note that when a standard becomes widely accepted, it also becomes feasible to implement the required logic in VLSI, thereby lowering the connection cost.

1.1 The OSI Reference Model

The International Standard Organization (ISO), in an effort to encourage "open" networks, developed the Open Systems Interconnect (OSI) Reference Model. In simple terms, the model logically groups the functions and sets of rules, called protocols, necessary to establish and conduct communication between two or more parties. The model consists of seven sets of functions, often referred to as layers. The OSI model describes the functions of each later in broad terms, not specific implementations.

This layered model approach affords two key advantages. First, layers allow a clear division of the design task through modularity. Second, systems based on a layered architecture are flexible. Flexibility is achieved because each layer functions independently of the layer above or below it. Thus, specific layer implementations can be changed easily. For example, layers 1 and 2 of a network can be changed to be either CSMA/CD based (e.g., IEEE 802.3) or token ring based (e.g., IEEE 802.5), without affecting layers 3 through 7.

The layer functions of the OSI model are summarized in Figure 2.

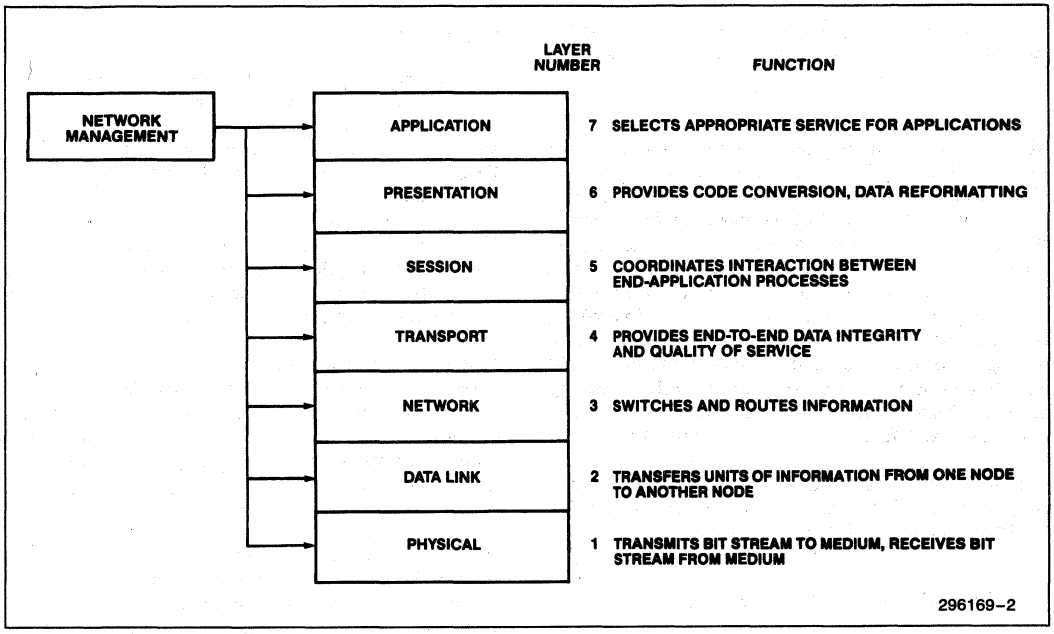


Figure 2. Layer Functions of the OSI Model

The **Physical Layer** describes the physical media over which the bit stream is to be transmitted. This Layer specifies type of cable (coax, twisted pair, etc.), connectors, signal levels, bit rate, data encoding method, modulation method, and method for detecting collisions in contention networks. In short, this Layer describes the actual physical media over which the bit stream is transmitted and the method of transmission, i.e., baseband or broadband.

The **Data Link** describes the rules for transmitting on the channel (made up of the encoder/decoder, transceiver cable, and transmission medium). Such items as the format of the information (frame) and procedures for gaining control of the channel (access method), transmitting the frame and releasing the physical media are specified by the Data Link Layer.

The **Network Layer** controls switching between links in a multihop network. The Network Layer is not necessary for a single LAN system because all stations connected onto a LAN share the same channel. This Layer is critical in gateway, communication server, and dial-up-communication applications.

The **Transport Layer** ensures end-to-end message integrity and provides for the required quality of service for exchanged information. For example, end-to-end acknowledgements and flow control are performed by the Transport Layer.

The **Session Layer** establishes and terminates logical connections between network entities. This Layer is also responsible for the mapping of logical names into network addresses.

The **Presentation Layer** provides for any necessary translation, format conversion, or code conversion to put the information into a recognizable form.

The **Application Layer** provides network based services to the end user. Examples of network services are distributed data bases and electronic mail. The Application Layer is not to be confused with the end user application itself.

Network Management is responsible for operation planning, which includes the gathering of operational statistics such as errors and traffic. It is also responsible for network initialization and maintenance (fault isolation). Network Management interfaces to each of the seven layers.

THE OSI MODEL AND NETWORK IMPLEMENTATIONS

The Physical and Data Link Layers of the OSI model ensure interconnectability. By implementing a particular physical and data link specification, equipment from multiple vendors can be physically and electrically con-

nected. The remaining five layers of the OSI model ensure interoperation among the interconnected stations in an open network.

For example, Intel's NDS-II Multi-User Networked Microcomputer Development System is a LAN based system utilizing IEEE 802.3 10Base5 (Ethernet) and IEEE 802.2 Logical Link Control (LLC) for Layers 1 and 2, and Intel OpenNet Architecture for Layers 3 through 7. Non-Intel equipment wishing to connect to the physical network need only adhere to the Ethernet and LLC specifications to ensure proper interconnection and gain access to the "data highway." In order to communicate with the system's Network Resource Manager (for interoperation), a non-Intel station would have to conform to the same protocols as the remaining layers of OpenNet, for example ISO 8043 for the Transport Layer and ISO 8743 for the Network Layer.

The ISO open system interconnect model has been adopted by the IEEE standards board for use in defining their standards for the various layers.

1.2 The IEEE 802 Committee

The Institute of Electrical and Electronics Engineers (IEEE), in response to a need for standardization in the field of Local Area Networks, formed the IEEE 802 standards body. The 802 standards specify the different protocols, access methods and their relationship with the ISO Open System Interconnection (OSI) Reference Model for Layer 1 and Layer 2 (see Figure 3). IEEE 802.1 explains how the different standards relate to each other and how they map into the OSI model. The Logical Link Control standard is specified by IEEE 802.2. IEEE 802.3, 802.4 and 802.5 specify the three different media access methods; CSMA/CD, Token Passing Bus and Token Ring, respectively.

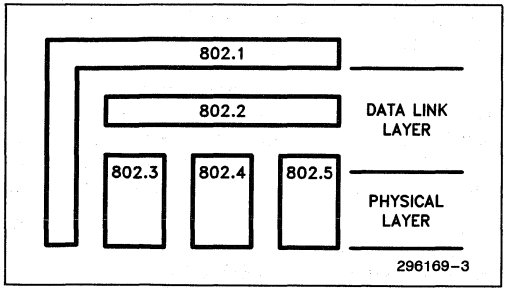


Figure 3. The IEEE 802 Standards Committees

CSMA/CD

Carrier Sense Multiple Access with Collision Detection (CSMA/CD) is a simple and efficient means of determining how a station transmits information over common medium that is shared with other stations. CSMA/CD is the access method defined by the IEEE 802.3 standard.

Carrier Sense (CS) means that any station wishing to transmit "listens" first. When the channel is busy (i.e., some other station is transmitting) the station waits (defers) until the channel is clear before transmitting.

Multiple Access (MA) means that any stations wishing to transmit can do so. No central controller is needed to decide who is able to transmit and in what order. The environment in which all stations on the network are peers with equal access is commonly referred to as distributed control.

Collision Detection (CD) means that when the channel is idle (no other station is transmitting) a station can start transmitting. It is possible for two stations to start transmitting almost simultaneously, causing a "collision". In the event of a collision, the transmitting stations will continue transmitting for a fixed time to ensure that all transmitting stations detect the collision. This is known as jamming. After the jam, the stations stop transmitting and wait a random period of time before retrying. The range of random wait times increases with the number of successive collisions such that collisions can be resolved even if a large number of stations are colliding.

There are three significant advantages to the CSMA/CD protocol. The first and foremost is that CSMA/CD is a proven technology. One CSMA/CD network, Ethernet, has been used by Xerox since 1975. Ethernet is so well understood and accepted that IEEE adopted it (with minor changes) as the IEEE 802.3 type 10BASE5 (10 Mbps, Baseband, 500 meters per segment) standard. Reliability is the second advantage to the 802.3 protocol. This media access method enables the network to operate without central control or switching. Thus, if a single station malfunctions, the rest of the network can continue operation. Finally, since CSMA/CD networks are of a passive, distributed nature, they allow for easy expansion. New nodes can be added at any time without reinitializing the entire network.

IEEE 802.3 Type 10BASE2 (about 200 meters per segment) is a variant which uses lower cost coaxial cable, and is more oriented to installation by the user. Popularly known as Cheapernet, 10BASE2 also eliminates a rather costly cable interconnecting the computer equipment to the transceiver which attaches to the coaxial cable in the case of 10BASE5.

TOKEN PASSING BUS

The IEEE 802.4 standard outlines the Token Bus media access method. A token is a group of coded bits that are passed from station to station on the network. The station that controls or "possesses" the token also controls the right of access to the link. So, as the token is passed, control of medium access is also passed. The link is a physical bus in that all the stations are attached to a common medium, but the token is passed from station to station forming a logical ring. For this reason, the token bus protocol has been identified as being a "logical ring on a physical bus".

During steady state operation, there are two general activities, data transfer and token passing. During the data transfer phase, a given station possesses the token and simply transmits data to one or more of the other stations of the link. Stations not possessing the token do not transmit, but listen to transmission for an address match. After a transmission is complete, the token is passed to the next station in the logical ring. This is the token passing phase.

Although the transmission of data and token passing are straightforward operations, network maintenance tasks make the token passing bus protocol relatively complex. The functions of network initialization, lost token recovery, adding new stations and general logical ring housekeeping require substantial logic. Additionally, all these functions must be replicated among all the token using stations on a given network in a prioritized fashion.

The primary advantage of token passing protocols is that they are deterministic as opposed to probabilistic. Every station has a definite opportunity to transmit within a given interval of time. With CSMA/CD, there is a statistical probability that each station will be able to transmit. In addition to the deterministic feature, token bus networks have the ability to span large distances, more than 10 kilometers.

TOKEN RING

Token ring is similar to token bus except that in addition to being a logical ring it is also a physical ring. The token ring procotol is governed by the IEEE 802.5 specification. It has similar performance characteristics as the token bus protocol.

1.3 Existing and Emerging Medium Access Standards

It has been established that no single LAN technology can satisfy the needs of every application. But, at the same time, the LAN marketplace cannot support every protocol that comes along. As a result, there are certain medium access control technologies that will be accepted as industry standards for given applications. Described here are the protocols that are already accepted, as well as those that are emerging, as industry-wide standards.

ETHERNET/CHEAPERNET

The IEEE 802.3 10BASE5 standard (Ethernet) has gained wide acceptance by both large and small corporations as a high speed (10 Mbps) Local Area Network. The Ethernet channel is a low noise, shielded 50Ω coaxial cable over which information is transmitted at 10 million bits per second. Each segment of cable can be up to 500 meters in length, can support up to 100 nodes, and can be connected into longer networks using repeaters. Repeaters are responsible for regenerating the signal from one cable segment on another. At each end of a cable segment a terminator is attached. This passive device provides the proper electrical termination to eliminate reflections.

The transceiver transmits and receives signals on the coaxial cable. In addition, it isolates the node from the channel so that a failure within the node will not affect the rest of the network. The transceiver is also responsible for detecting when two or more stations transmit simultaneously (collisions). Ethernet transceivers are connected to the network coaxial cable using a simple tap, and to the station it serves via the transceiver cable which can be up to 50 meters in length.

An Ethernet interface, which includes a serial interface and a data link controller, provides the connection to the user or server station. It also performs frame manipulation, addressing, detecting transmission errors, network link management and encoding and decoding of the data to and from the transceiver.

Due to its high speed and relatively high cost, Ethernet is targeted to be a LAN backbone for the engineering and office environments. It is also feasible to utilize Ethernet in engineering workstation and personal computer cluster applications where cost is not a sensitive issue.

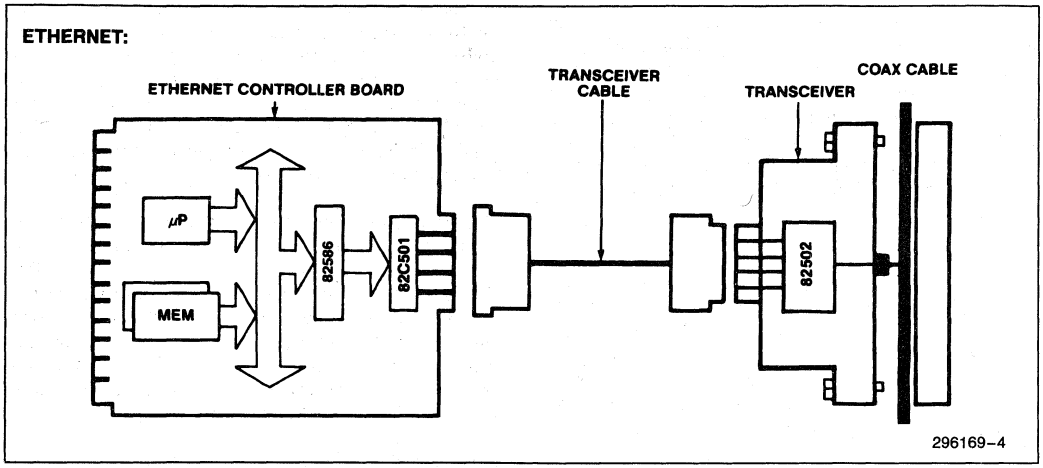
One of the major contributors to the high cost per connection of Ethernet is the cable. Ethernet cable must be highly immune to noise so that each cable segment can be 500 meters in length. Cheapernet, IEEE 802.3 10BASE2, is identical to Ethernet with the following exceptions. First, less expensive RG-58 CATV coaxial cable is used as the network link. Second, the transceiver function is integrated into the node which is connected to the link via a BNC T-connector. Use of the lower performance cable limits Cheapernet to 185 meter segments with 30 nodes per segment. Finally, the transceiver cable is eliminated.

Cheapernet preserves the high data transfer rate of Ethernet, but reduces the per node cost considerably. For this reason, Cheapernet is optimized for high performance workstation or personal computer network clusters.

IBM PC NETWORK

In August 1984, IBM announced a new lost cost IBM PC Network based on 2 Mbps CSMA/CD broadband technology. The IBM PC Network uses standard CATV coaxial cables and hardware connections.

The IBM PC Network, in a user installable configuration, can handle up to 72 nodes and span 1000 foot radius. A key feature of this network is the IBM PC Network Translator Unit. The Translator Unit provides broadband frequency translation from the return channel to the forward channel, for a passive IBM PC Network. For greater capability, the IBM PC Network can be used in a professionally installed broadband network allowing up to 1000 IBM Personal Computers within a 5 Km radius of the network translator unit.



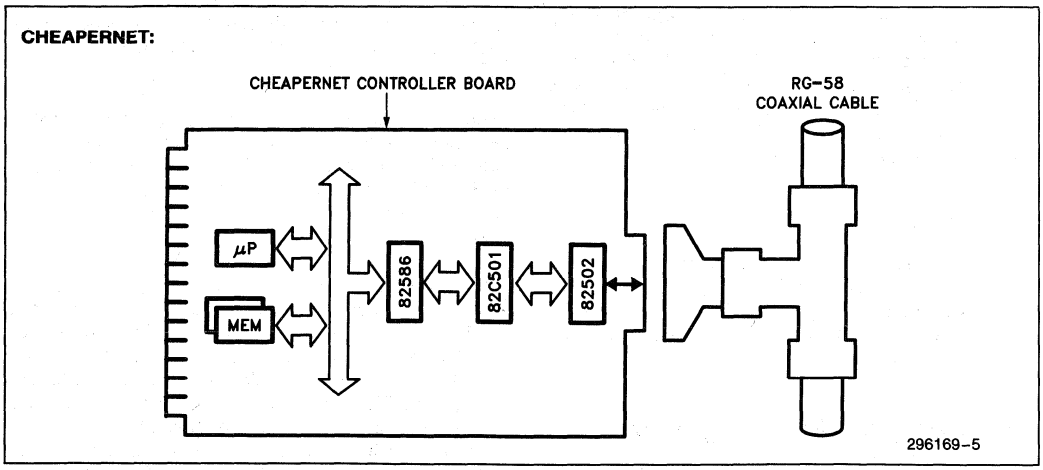


Figure 4. Ethernet/Cheapernet Configurations

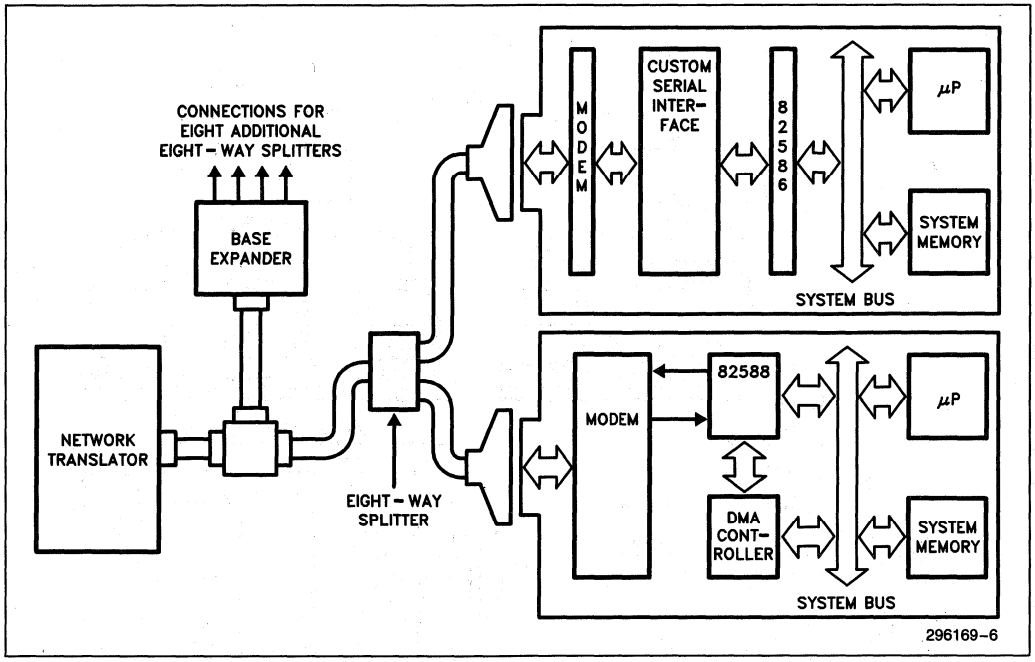


Figure 5. IBM PC Network Using Either the 82586 and the 82588 in Intelligent Adapters

StarLAN

IEEE 802.3 Type 1BASE5 (1 Mbps, Baseband, 500 meters diameter in a minimum configuration) is a CSMA/CD network oriented to personal computer interconnection in office environment. Each station is connected to a "hub" and each hub can be connected to another hub in point-to-point fashion. The result is cascaded star-shaped clusters. Each hub serves as the point of concentration, similar to a telephone wiring closet and performs two main functions: signal regeneration and retiming (for retransmission to other stations and hubs) and collision detection. When two stations transmit simultaneously and a collision occurs, the hub sends a collision presence signal to all receivers. The hubs can be cascaded up to five levels and each station-to-hub or hub-to-hub interconnection can be up to 250 meters in length.

StarLAN has two distinctive characteristics. Cost is the most sensitive issue in the personal computer field and the StarLAN hardware and cabling scheme are designed to be inexpensive. The cabling scheme uses standard twisted-pair telephone wiring and is laid out in a similar fashion where the StarLAN hub is analagous to a telephone wiring closet. Todays buildings are designed with this cabling scheme in mind which means installation, reconfiguring and servicing will be easy and low cost as well. Futhermore, most buildings use only about one half of the existing telephone cabling so the spares can be easily used for a StarLAN network.

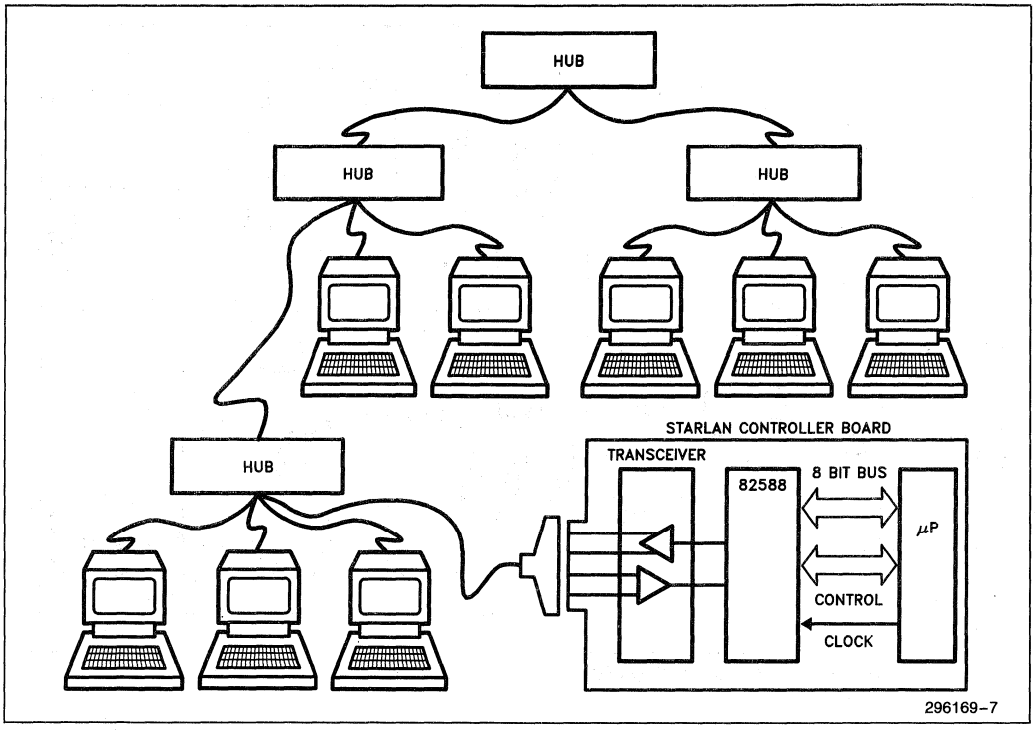


Figure 6. StarLAN Configuration

2.0 THE INTEL LAN SOLUTION

2.1 A Commitment To Standards

Intel has the most complete range of VLSI, software and boards to support the existing and emerging industry standards for Local Area Networks. The 82586 LAN Coprocessor, the 82C501 Ethernet Serial Interface and the 82502 Ethernet Transceiver Chip make up Intel's three chip set for IEEE 802.3 10 Mbps networks (10BASE5 and 10BASE2). For the mid-range LAN applications, i.e., StarLAN and the IBM PC Network, Intel offers the 82588 LAN Controller and the 82586 LAN Coprocessor. Finally, iNA960 software supports Intel hardware solutions with ISO conformant implementations. Intel also supplies several addition board and software solutions in the networking field. For information on these product lines, please contact your local Intel Field Sales representative.

2.2 Intel's Ethernet/Cheapernet Chip Set

THE 82586 LAN COPROCESSOR

The 82586 is an intelligent peripheral that completely manages the processes of transmitting and receiving frames over a CSMA/CD network. The 82586 offloads the host CPU of the tasks related to managing communication activities. The 82586's memory based architecture and integrated DMA controller enables it to function independently of the host CPU for time critical functions. Hence, the 82586 is truly a coprocessor.

In addition to the high performance, the 82586 also features a high degree of flexibility. The 82586's Network parameters are programmable so as to optimize the LAN design to specific applications. When powered up, the chip defaults to 10BASE5/10BASE2 network parameters. But when programmed accordingly, the 82586 can support StarLAN, the IBM PC Network, and a wide range of proprietary Local Area Networks. Because of its flexibility, the 82586 is also ideal for Serial Backplane applications.

The 82586 interfaces easily to available microprocessors. Systems requiring minimum component count can

take advantage of the 82586's direct interface (no "TTL glue") to Intel's 80188 (8-bit data bus) and 80186 (16-bit data bus) microprocessors.

The 82586 uses memory efficiently through data (or buffer) chaining. System memory is not wasted because short frames can be saved in buffers of minimal size, while long frames are saved by chaining buffers together. The 82586's memory based architecture, together with its data chaining capability result in a high degree of CPU independence, along with high system performance.

The 82586 provides a rich set of network management capabilities. Included are:

- Error tallies in system memory to monitor:
  - — Number of frames incorrectly received due to CRC errors
  - — Number of frames incorrectly received due to misaligned frames
  - — Number of collisions experienced in the process of transmitting frames
  - — Number of frames lost due to lack of receive buffers
  - — Number of frames lost due to DMA overrun while receiving frames
- Monitoring of Signal Quality Error (SQE) function as defined by the IEEE 802.3 specification (signaling that the transceiver's collision detection mechanism is not functioning correctly).

The 82586 provides diagnostic capability through internal and external loopbacks. Distance to cable breaks and shorts is provided by on-chip time domain reflectometry logic.

THE 82C501 ETHERNET SERIAL INTERFACE

The 82C501 is designed to work directly with the 82586 in 10 Mbps LAN applications. The primary function of the 82C501 is to perform Manchester encoding/decoding, provide the 10 MHz transmit to the 82586, recover clock from the received signal, generate carrier sense and collision presence signals appropriate to the 82586, and interface the transceiver cable. The 82C501 provides for fault isolation via an internal loopback. Continuous transmission (babbling) is prevented by an on-chip watchdog timer.

The 82C501 conforms to the IEEE 802.3 10BASE5 and 10BASE2 specifications. It is designed with Intel's CHMOS II technology.

THE 82502 ETHERNET TRANSCEIVER CHIP

The 82502 rounds out Intel's three chip set for IEEE 802.3 10 Mbps networks. The 82502's primary functions are transmission of data onto the network coaxial cable, reception of data from the coax, collision detection, and interface the transceiver cable. Additional features are:

- A jabber mechanism, namely a watchdog timer to prevent continuous transmission by the parent station.
- A defeatable Signal Quality Error (SQE) test which verifies functionality of the collision detection circuitry.
- On-chip precision voltage reference which allows for relaxed power supply tolerances.
- CHMOS technology which allows the 82502 to run at very low power consumption levels, thus enhancing reliability.

The 82502 supports 10BASE5, 10BASE2 and related repeater applications.

2.3 The 82588 LAN Controller

The 82588 is a highly integrated LAN controller targeted for use in cost sensitive CSMA/CD LAN applications, in particular, personal computer interconnection. The 82588 integrates CSMA/CD, Media Access Control (MAC) functions, data encoding/decoding, clock recovery from received signal, two different collision detection mechanisms and transmit clock generation. This high-integration allows the system designer to reduce component count, and thus reduce board space and development time. The functionality of the 82588 is optimized at up to 2 Mbps in either baseband or broadband networks.

Like the 82586, the network parameters of the 82588 are programmable, providing the flexibility to support numerous LAN applications. Among the programmable parameters are:

- — Slot Time
- — Minimum Frame Length
- — Framing (End-of-Transmission or HDLC)
- — Address Field Length
- — Interframe Spacing
- — NRZI, Manchester, or Differential Manchester encoding/decoding

Additionally, the 82588 supports various network topologies. The 82588 conforms to the IEEE 802.3 1BASE5 specification.

The 82588 implements two methods of logic based collision detection. The Code Violation method detects a collision when a data transition occurs outside of the specified regions. The Proprietary Bit Comparison method compares the "signature" of a transmitted frame to the receive frame "signature".

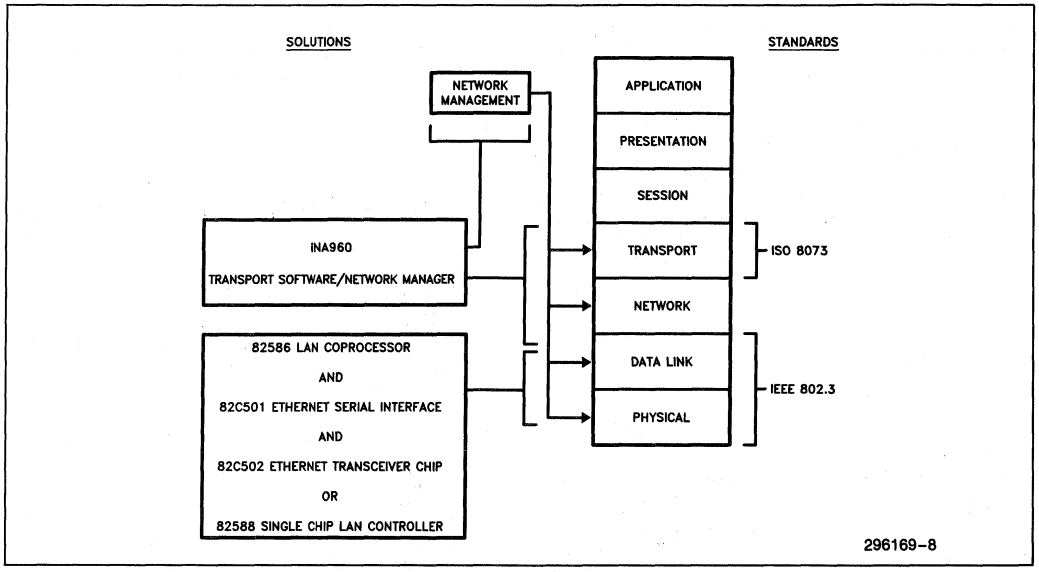


Figure 7. Intel's LAN Component Solutions

Another significant feature of the 82588 is its system interface. High level commands such as TRANSMIT and CONFIGURE are used. The 82588 supports receive buffer chaining, which provides for efficient memory usage. Finally, the 82588 has a complete set of network management, maintenance and diagnostic capabilities.

2.4 No Copy

2.5 The iNA 960 Transport Software

iNA 960 is a general purpose Local Area Network software package that provides the user with guaranteed end to end message delivery. iNA 960 conforms to the International Standards Organization's 8073 specification for Class 4 Transport Layer services as well as ISO 8473 Network Layer services. iNA 960 also provides network management functions, along with 82586 and 82588 device drivers.

TRANSPORT SERVICES

The iNA 960 transport layer implements two kinds of message delivery services: virtual circuits and datagram. Virtual circuits provide a reliable point-to-point message delivery service ensuring maximum data integrity and are fully compatible with the ISO 8073 Class 4 protocol. In addition to guaranteed message integrity, iNA 960:

- Provides flow control (data rate matching between sender and receiver).
- Supports multiple simultaneous connections (process multiplexing).
- Handles variable length messages (independently of physical frame size).
- Supports expedited delivery (to transmit urgent data).

The datagram option provides "best effort" delivery service for non-critical messages. The datagram service does not guarantee message integrity but requires less channel overhead than virtual circuits.

NETWORK MANAGEMENT SERVICES

The Network Management facility supports the users of the network in planning, operating, and maintaining the network by providing network usage statistics, by allowing the monitoring of network functions and by detecting, isolating, and correcting network faults.

The Network Management facility also supports upline dumping and down-line loading of data bases or to boot systems without local mass storage.

USER ENVIRONMENT

In the iRMX (Intel's real time, multitasking operating system) environment, both the user programs and iNA 960 run under iRMX 86. The communications software is implemented as an iRMX 86 job requiring the nucleus only for most operations. The only exception is the boot server option, which also needs the Basic I/O Sys-

tem. iNA 960 will run in any iRMX environment including configurations based on the 80130 software on silicon component.

In those systems where iRMX 86 is not the primary operating system, or where offloading the host of the communications tasks is necessary for performance reasons, the user may wish to dedicate a processor for communication purposes. iNA 960 can be configured to support such implementations by providing network services on an 8086, 8088, or 80186 microprocessor.

iNA 960 is also available for operation under MS DOS in personal computer environments.

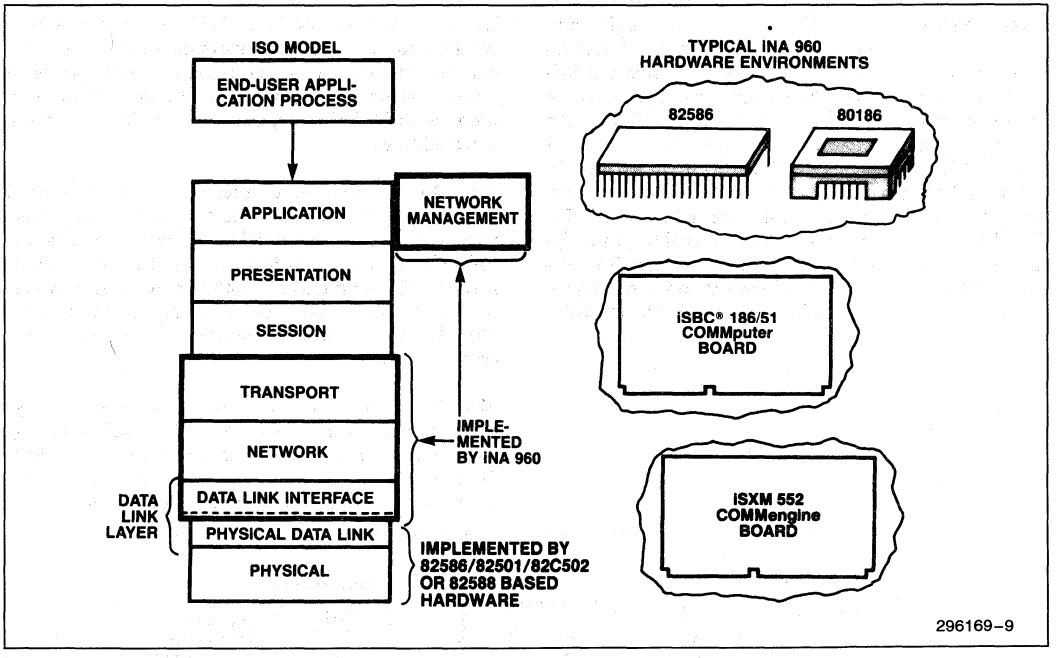


Figure 8. iNA 960 Software

2.6 MAP-NET Layer 5–7 Map Software

MAP-NET 2.1, iNA 960 Rel 2.0 and the iSBC 554 MAP Board and ready-to-use software building blocks for OEM suppliers of networked systems to implement ISO/OSI layers 1–7, as specified by MAP version 2.1. The Intel iSBC 554 board provides the data link and the IEEE 802.4 based physical layer for MULTIBUS® based systems. MAP-NET is designed to use the services and interface provided by Intel's iNA 960 Rel 2.0 Software package. iNA 960 Rel 2.0 provides the ISO 8473 network and ISO 8073 transport—layers 3 and 4 of MAP. MAP-NET 2.1 provides layers 5–7 of the MAP 2.1 specifications and is designed to run on top of iNA 960 Rel 2.0 on the iSBC 554 board. Together the board and software modules provide a complete, seven layer, configurable MAP solution for industrial OEM's. The MAP-NET software is also available preconfigured with iNA 960 Rel 2.0 to run on the Intel iSBC 554 board. This preconfigured software with the board provides a complete seven layer turnkey solution for MAP 2.1.

FUNCTIONAL OVERVIEW

The Intel MAP-NET 2.1 software provides the following services specified by MAP 2.1; the session service, network management, FTAM and CASE. These services fit into the upper 3 layers of the ISO/OSI 7 layer model.

Using the Services of MAP-NET, users can initiate communications with other users on a MAP LAN, access information regarding resources available on a LAN, transfer files across a LAN and address other users on the LAN by logical names rather than numbered addresses.

MAP-NET is designed to interface with iNA 960 Rel 2.0. iNA 960 Rel 2.0 provides the network and transport protocol that is required by the MAP specification. Please refer to the iNA 960 data sheet for more information. The configurable software packages—MAP-NET and iNA 960 Rel 2.0—are designed to run on the iSBC 554 for a complete, on-board, seven layer MAP commengine.

MAP 2.1 SXMSW is a preconfigured software package that incorporates the functions of MAP-NET and iNA 960 Rel 2.0. This package provides layers 3–7 and is designed to run on the iSBC 554 for a complete, on-board, seven layer, turnkey MAP commengine.

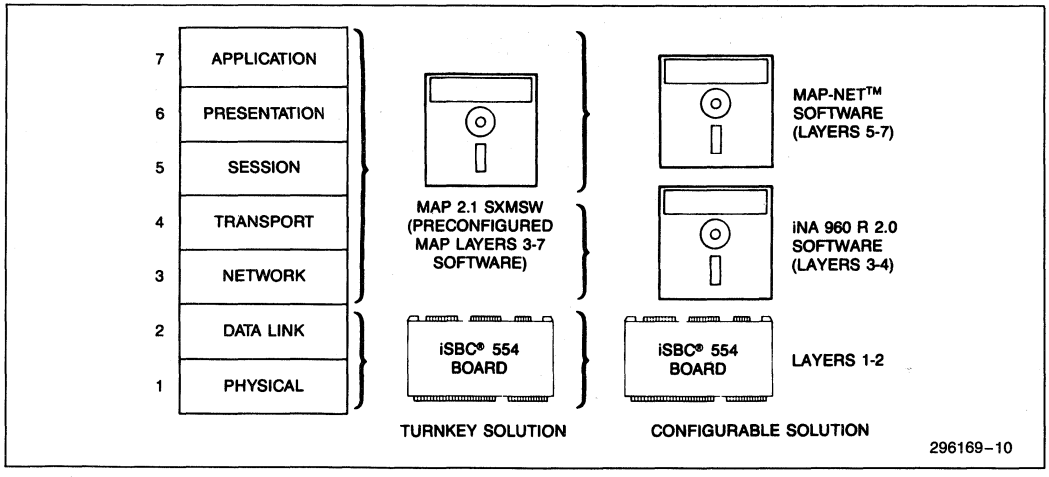


Figure 9. OSI Reference Model. Figure 9 Above Indicates How Intel's MAP Software and Hardware Products Fit in the OSI Reference Model for MAP.

82586 Reference Manual

2

CHAPTER 2
THE 82586 LAN COPROCESSOR

OVERVIEW

This chapter describes the features and operation of the 82586 LAN Coprocessor. It is assumed that the reader is familiar with the basic concepts of Data Communication, Local Area Networks (LAN), and the IEEE 802.3 Standard.

This Chapter is divided into three parts, each part covers several sections:

1) A general description of the 82586.
2) A description of the major functions performed by the 82586 for the user:
   - Transmit Functions
   - Receive Functions
   - Network Management and Diagnostic Functions
   - Interaction with the Host CPU
3) Detailed instructions on how to use the 82586:
   - Initializing the 82586
   - Controlling the 82586
   - Action Commands
   - Frame Reception
   - Bus Interface Hardware
   - Network Interface Hardware

Pin functions, electrical and timing characteristics are located in the 82586 Data Sheet, included as part of this User's Manual.

1.0 OVERVIEW OF THE 82586 LAN COPROCESSOR

The 82586 VLSI chip is an intelligent, high performance, CSMA/CD (Carrier Sense Multiple Access with Collision Detection) communications controller. The 82586 performs all functions associated with data transfer between user memory and the Network: framing, link management, address filtering, error detection, data encoding, network management, Direct Memory Access (DMA), buffer chaining, and interpretation of high level commands from the user. Called the LAN Coprocessor, the 82586 was designed to relieve the host CPU of most tasks associated with controlling access to a LAN.

The 82586 meets the performance requirements of the IEEE 802.3 Standard: 10 megabits per second bit rate and 9.6 microseconds Interframe Spacing. In addition to providing DMA transfers at 4 megabytes per second, it tolerates local bus latency of over 10 microseconds without losing data, and bus transfer rates as low as 2 megabytes per second. The high performance permits the 82586 to be used in distributed processing applications such as high speed resource sharing and interprocessor communications.

The 82586's programmable network parameters allows it to serve as controller for a wide range of CSMA/CD type LAN's. It is compatible with network specifications such as high service (broadband), high performance (short topologies) and low cost (1 Mbps) networks. Data rates less than 10 megabits per second are supported. Many parameters are configurable including all framing parameters (i.e. address length, End of Carrier or Bitstuffing frame boundary delineation, etc.), Slot Time and Interframe Spacing.

Network and station reliability is enhanced by built-in diagnostic aids, such as Time Domain Reflectometer (TDR), external and internal loopback, Transceiver integrity verification, internal register dump and a self test procedure.

The 82586 is contained in a 48-pin dual in-line package. It is also available in a 68-pin PLCC. Figure 1 shows the 48-pin DIP pin layout.

Signals in parentheses are available in Minimum mode.

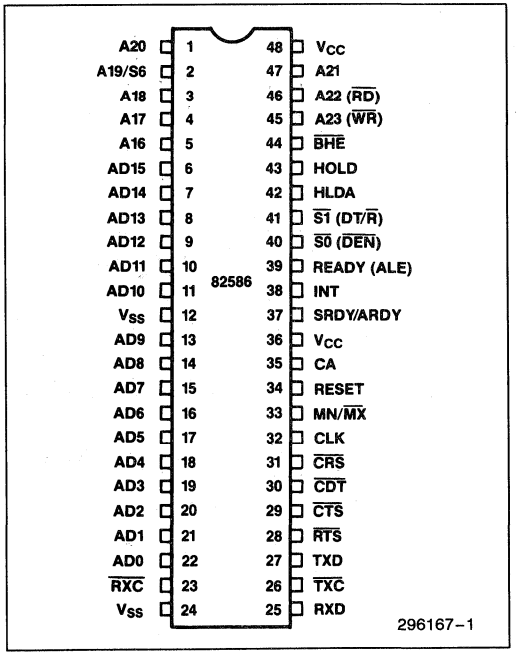


Figure 1. The 82586 Pin Configuration

The major application of the 82586 is to serve as the communication manager in a station connected to a LAN. Such a LAN station typically consists of a host CPU, shared memory, an 82586 Local Communication Controller, Serial Interface Unit, Transceiver, and LAN link. The 82586's task is to perform functions associated with transferring data between shared memory and the LAN link. As an example, Figure 2 shows the 82586 in a workstation connected to an IEEE 802.3 network.

The 82586 has two interfaces: Bus Interface to the local bus and the CPU; Network Interface to the Serial Interface Unit.

On the Bus side, the 82586 is a 'master' on the 8 or 16-bit local bus, and communicates directly with the CPU via Channel Attention (CA) and Interrupt (INT) signals. It is optimized for operation with the iAPX 186 bus but can be used with other general purpose processors.

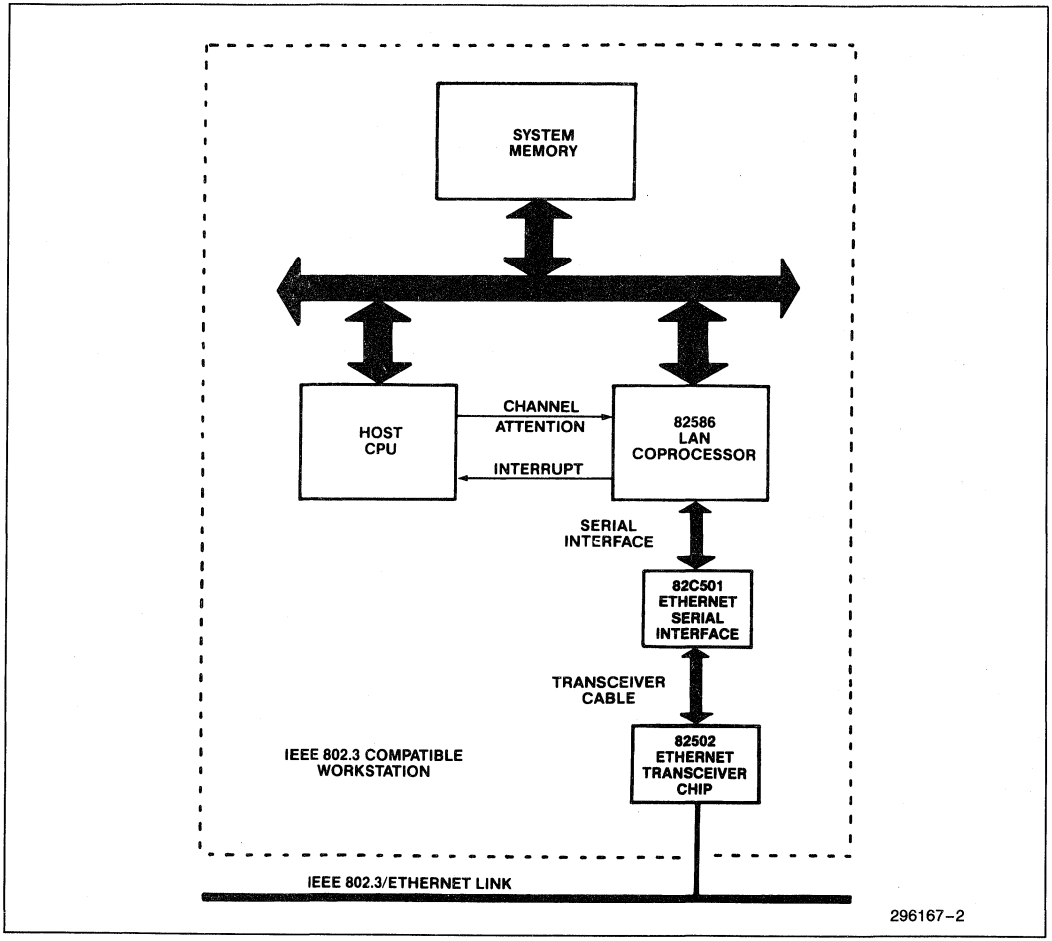


Figure 2. An IEEE 802.3 (10BASE5) Compatible Workstation

On the Network side, the 82586 is connected to an Ethernet Serial Interface that provides Transmit and Receive Clock and Data, Collision Detect, Carrier Sense, and Request to Send/Clear to Send signals to the 82586. The Ethernet Serial Interface is connected to the Transceiver, which is connected to the LAN link. In the particular case of the IEEE 802.3 (10BASE5) station, the Ethernet Serial Interface is Intel's 82C501 and the Ethernet Transceiver is Intel's 82502.

2.0 82586 TRANSMIT FUNCTIONS

The 82586 LAN Coprocessor performs two major tasks: transmitting data from host memory to the Network and receiving data from the Network and placing it in memory. This section describes the transmission process. Reception is described in section 3.

The data units handled by the 82586 are frames. A frame is a sequence of bits that travels on the link. A frame is divided into fields: address, data, frame check sequence, etc. The host CPU prepares a sequence of frames in shared memory and instructs the 82586 to start transmission. Frames are transmitted by the 82586 one at a time. The chip resolves access and contention on the link using the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Link Management mechanism.

This section presents the framing, link management, and priority mechanisms.

2.1 Framing

Framing has three primary functions: to determine the beginning and end of the frame (frame boundary delineation); to determine the frame's source and destination (addressing); and to perform error detection. Framing can be summarized in terms of the generalized frame format shown in Figure 3.

PREAMBLE
START FRAME DELIMITER FIELD
DESTINATION ADDRESS
SOURCE ADDRESS
LENGTH FIELD
DATA FIELD
FRAME CHECK SEQUENCE
END OF FRAME FLAG (OPTIONAL)
PADDING (OPTIONAL)

Figure 3. Frame Fields

Figure 3 shows the fields in a frame: the Preamble is used as a synchronizing sequence for bit decoding, followed by the Start Frame Delimiter Field, SFD. Next, the Destination Address is the frame target address. This field is followed by the Source Address (sender's address). The Length and Data Fields contain user supplied data. The Frame Check Sequence is a Cyclic Redundancy Check, CRC, used in detecting bit errors. Two optional fields may follow, End of Frame (EOF) flag and Padding. The latter extends the length of the frame to ensure minimum frame length.

The 82586 handles Frame Boundary Delineation completely transparent to the user. The fields involved in the Frame Boundary Delineation are: Preamble, SFD Field, EOF flag and Padding. The 82586 is configurable to one of two Frame delineation methods: End of Carrier or Bitstuffing.

In the End of Carrier method, the 82586 transmits (depending on the configuration) 8, 24, 56, or 120 Preamble bits of alternating ones and zeros followed by a SFD field (10101011). The end of frame is indicated by the carrier going inactive immediately after the Frame Check Sequence field. This frame boundary delineation method is compatible with IEEE 802.3.

The Bitstuffing method implements the HDLC zero bit insertion/deletion mechanism. The 82586 transmits (depending on the configuration) a Preamble of 8, 24, 56 or 120 alternating ones and zeros followed by an HDLC Flag (01111110). End of frame is indicated by an HDLC flag. The 82586 performs HDLC zero bit insertion (insert 0 after five consecutive 1's) on the fields between flags (exclusive). The chip can be configured to pad the frame with additional flags so that frame length becomes longer than Slot Time (see section 2.2).

Regardless of the Frame Boundary Delineation method, the two fields following the SFD Field are Destination Address and Source Address. The former is fetched from memory and the latter is usually inserted from the internal Individual Address register (unless configured not to). The address length can be configured to a value from 0 to 6 bytes. Addresses are sent with the least significant byte first. The Destination Address can be one of three types: Individual Address (least significant bit is zero), Multicast Address (least significant bit is one), or Broadcast (all ones). See section 3.2 for more details on addressing.

The Length Field and Data Field are fetched by the 82586 from memory and transmitted after the Source Address. The Length Field is 2 bytes long. The 82586 itself places no limit on the Data Field length.

The Frame Check Sequence field protects against bit errors in the frame. It is the result of a Cyclic Redundancy Check computed from the Destination Address, Source Address, Length Field, and Data Field. The chip can be configured to one of two CRC algorithms: the CCITT V.41 16-bit polynomial or the Autodin II (IEEE 802.3) 32-bit polynomial.

2.2 Link Management

The 82586 handles CSMA/CD link management algorithms according to the IEEE 802.3 standard. 82586 Link management algorithms are adaptable to a variety of network topologies via programmable configuration parameters. Station priorities are also programmable.

The 82586 constantly monitors link activity. Whenever it senses the carrier (data transitions) on the link, the 82586 defers to the passing frame by delaying any pending transmission. After carrier goes inactive, the 82586 continues to defer for Interframe Spacing time. Interframe Spacing is configurable from 32 to 255 TCLK (Transmit Clock) units. If, at the end of that time, it has a frame waiting to be transmitted, transmission is initiated independent of the sensed carrier.

After transmission has started, the 82586 attempts to transmit the entire frame. In the normal case, frame transmission is completed, and the host notified. Otherwise, one or more of the following events causes transmission to terminate prematurely: Clear-to-Send signal goes inactive during transmission, data transfer rate from memory to the chip did not keep up with transmission (DMA underrun), Carrier Sense goes inactive (in case the chip is configured to expect the return of Carrier Sense signal), a collision is detected via Collision Detect, or the collision retry counter exceeded the maximum specified value.

When the 82586 has finished deferring and has started transmission, it is still possible to experience link contention. This situation is called a collision and it is generally detected by the Transceiver. The 82586 enforces a collision by transmitting a Jam pattern of 32 ones. If a collision is detected during the Preamble, transmission of the Preamble is completed before jamming starts.

The dynamics of collision handling are largely determined by the Slot Time. Slot Time is the maximum end to end round trip delay time of the network plus jam time. Slot time is important because it is the worst case time to detect a collision. Long networks have longer slot times than short networks. Thus, the 82586 provides for slot time to be configurable from 1 to 2048 TCLK units.

After waiting the Backoff time, the 82586 attempts to retransmit the frame, unless the number of retransmissions has exceeded the maximum allowed. The 82586 calculates the Backoff algorithm according to the IEEE 802.3 standard: Backoff is an integral number of Slot Times. It is a random number, from 0 to maximum. The maximum number is (2**R)-1, where R is the minimum between 10 and the number of retransmission attempts. This range can be extended using Accelerated Contention Resolution mechanism, see section 2.3 and 2.4. The beginning of Backoff time is configurable to one of two methods: if configured to the IEEE 802.3 compatible method, it starts immediately after the end of jamming; if configured to the alternate Backoff method (designed for lower bit rates and/or shorter topologies), Backoff starts after the deferring period following collision.

The 82586 maintains a retry counter that is incremented after each retransmission attempt. If retransmission is successful, the user is notified. If the number of retries exceeds the maximum, an error is reported. The number of allowed retries is configurable from 0 to 15 attempts. The only difference between transmission and retransmission is that transmission clears the retry counter and retransmission increments it.

On completion of transmission or retransmission, the 82586 reports the number of collisions that occurred and whether it exceeded the maximum. It also indicates if the chip had to defer to passing traffic on its first transmission attempt.

The user may attempt to abort transmission. Upon receipt of the Abort command, the 82586 transmits a Jam pattern to cause a CRC mismatch. The chip reports to the host either that the abort succeeded or that the frame transmission completed before the abort was accepted.

2.3 Priority Mechanism

One of the goals for the IEEE 802.3 standard is to ensure fairness among stations trying to access the link. However, there are applications (e.g. transmitting voice) where station priority improves the performance. The 82586 implements two priority mechanisms: linear and accelerated contention resolution.

Linear priority determines the number of Slot Times the 82586 waits after deferring or the end of Backoff (whichever comes last) before transmitting. If the link becomes busy during the wait period, the process of deferring and waiting starts again. Linear priority is programmable from 0 to 7. Zero provides the highest priority and is IEEE 802.3 compatible.

Accelerated contention resolution extends the range from which the random number for Backoff is drawn. It is configurable from 0 to 7. Zero provides the highest priority and is IEEE 802.3 compatible.

2.4 Details of the Link Management Algorithms

The 82586 supports the IEEE 802.3 link management algorithms. In addition, the 82586 provides alternative station priority and backoff mechanisms.

The transmit link management includes two cooperating sequences: Frame Transmitter and Deference. The following variables provide the communication between these sequences: DEFERRING and WAITING from the Deference sequence and BACKOFF from the Frame Transmitter sequences.

Deference is an ever running sequence and performs the following algorithm compatible with IEEE 802.3:

1) Waits for Carrier Sense to become active.

2) Sets the DEFERRING variable.

3) Waits for Carrier Sense to become inactive.

4) Waits for a period of Interframe Spacing.

5) Clears the DEFERRING variable, waits for pending frame and returns to step 1.

The Frame Transmitter sequence is initiated when the 82586 is instructed to transmit a frame. It runs in parallel with Deferring and performs the following:

1) Assembles the frame and clears RETRIES.

2) Waits for DEFERRING and WAITING to clear.

3) Starts frame transmission.

4) If transmission is completed without detecting a collision, the sequence is completed with a successful transmission.

5) If a collision is detected, the 82586 does the following:

   a. Completes transmission of the Preamble (if applicable).

   b. Transmits Jam pattern.

   c. If RETRIES is equal to the maximum number of retries, the Transmit command is completed and status posted in the Transmit command status field.

   d. RETRIES is incremented if it was less than the maximum number of retries.

   e. The Backoff time is computed based on the following algorithm:

      Backoff time = (slot time) $\times$ (random number)

      Where:

      Random Number R is given by

      $0 \leq R < 2 \exp [\min (10, \text{retries})]$

   f. The 82586 waits for the period of Backoff time. If the Backoff time is zero (R = 0), then the Backoff period is equal to Interframe Spacing. For a random number greater than zero, Backoff is computed as shown in 'e' above.

   g. If Backoff time was equal to Interframe Spacing (R = 0), then the 82586 transmits after the Backoff time has expired.

   h. If random number is greater than 0, the 82586 waits for the Backoff time and returns to step 2.

   i. Clears the BACKOFF variable and returns to step 2.

LINEAR PRIORITY

In the case where the 82586's linear priority option has been selected, the link management algorithm differs in the deferring sequence:

1) Waits for Carrier Sense to become active.

2) Sets deferring variable.

3) Waits for Carrier Sense to become inactive.

4) Waits for a period of Interframe Spacing.

5) Calculates Wait Time as follows:

   Wait Time = P $\times$ Slot Time

   Where P is the programmed linear priority number

6) Sets the waiting variable.

7) Waits according to Wait Time calculating in step 5. If during the Wait Time, Carrier Sense goes active, the 82586 sets the deferring variable, clears the WAITING variable and returns to step 3.

ALTERNATE BACKOFF ALGORITHM

In the case of IEEE 802.3, the Backoff time after a collision is as discussed above. The random number R determines the Backoff time. The range of random number in turn depends on the number of retries. For example, on first transmission attempt, retry variable is zero, hence R could be either 0 or 1, resulting in a Backoff time of Interframe Spacing or one slot time. On the first retry, retry variable is equal to 1, and the range for R is 0, 1, 2, 3, resulting in Backoff times of Interframe Spacing, or one, two or three times the slot time. In the first case, the probability of two stations backing off for the same period of time is 50% (IFS or 1 slot time). In the second case, this probability will reduce to 25% and so on.

In some applications it may be desirable to resolve the contention for the channel faster. If the colliding stations were to pick the backoff random number from a larger range of numbers, the probability of the stations picking the same random number (and hence colliding again) will be reduced. The 82586 offers this capability by programming the alternate backoff method. If this method is selected, the backoff period is computed as follows:

Backoff Time = R × Slot Time where

$0 \leq R < 2 \exp [\min (10, \text{Retry} + K)]$

Where K is the alternate backoff number, programmable between 0 and 7.

Note that for IEEE 802.3, K = 0.

3.0 82586 RECEIVE FUNCTIONS

This section describes how the 82586 processes received frames. The 82586 checks all frames that appear on the link, decides which frames should be passed to host memory, and checks them for errors.

3.1 Frame Reception

The 82586 recognizes the boundary of an incoming frame by monitoring the link. When the 82586 detects a carrier (data transitions) on the link and it is not transmitting or executing Action Commands (see section 8), it starts accepting the incoming bits. The frame Preamble provides bit synchronization and the frame SFD Field is used to locate the first bit of the Destination Address field, see Figure 3. The Preamble and SFD Field are discarded. The 82586 compares the incoming frame's Destination Address to the receiving station's Individual Address or Multicast Address; the Broadcast Address is also checked.

If there is an address match (and the frame satisfies the minimum frame size), the 82586 passes the Destination Address, Source Address, Length Field, and Information Field to system memory, calculates the CRC and verifies its correctness during reception of the FCS field. If there is not an address match, the 82586 never requests the system bus, and the 82586 becomes ready to receive the next frame.

If a frame is received in error (CRC violation, alignment, No Resources, DMA overrun), the 82586 automatically reclaims the memory used to store that frame. The next received frame is stored in the reclaimed memory.

When configured to End of Carrier delineation (IEEE 802.3), end of frame is indicated by the carrier going inactive. The number of bits after the SFD Field must be a multiple of eight, residual ('dribble') bits are discarded, and not included in the Frame Check Sequence. When there are residual bits the frame is 'misaligned.'

When configured to Bitstuffing delincation, the 82586 performs zero bit deletion, discards the EOF flag, and all following bits until end of carrier. Residual bits are discarded in a manner similar to the End of Carrier method. An error is reported if the carrier goes inactive prior to recognition of an EOF flag.

The minimum frame length is configurable in the range of 0 to 255 bytes. Any frame containing less than the minimum (configured) number of bytes is presumed to be a fragment resulting from a collision. A collision fragment, longer than 6 bytes, is either stored in memory and marked as a short frame, or its area is reclaimed (depending on the Save Bad Frame configuration).

3.2 Addressing

Addressing allows frames to be directed to one or more specific hosts. The 82586 provides flexible addressing techniques allowing a frame to be received by a single host, a group of hosts (multicast), or all hosts (broadcast). The chip checks the incoming Destination Address according to the three methods simultaneously, the frame is accepted if there is a match. A frame whose address does not match has no effect on the 82586 nor on any other component of the station.

The 82586 is normally configured with a specific Individual Address using the IA-SETUP command (see section 8.3). The default configuration is an all ones address 6 bytes long. An Individual Address is indicated by a zero in the least significant bit, and its length is determined by the configured Address Length parameter. During reception, the 82586 compares the incoming Destination Address with its Individual Address. All bits must be equal for an Individual Address match to be determined.

A frame may be targeted to all hosts (Broadcast) by using the Broadcast Address, an all ones address. During reception, the 82586 checks if the Destination Address is a Broadcast Address. If it is, an address match is confirmed. The 82586 can be configured to disable reception of frames with Broadcast Destination Addresses (for stations with limited storage resources).

The user can target a frame to a group of hosts. Using a method not involving the 82586, hosts are placed into groups, each group is assigned a Multicast Address. A host may belong to a number of groups and a group may contain a number of hosts. A host may address a frame to all hosts that belong to a particular group by specifying that group's Multicast-Address in the Destination Address field. A one in the least significant bit of the Destination Address distinguishes Multicast-Addresses from Individual Addresses.

The 82586 maintains a 64-bit Hash table. The 82586 maps every Multicast-Address into a single bit in the table. Using the MC-SETUP command (see section 8.5), the user provides a set of Multicast-Addresses, and the 82586 maps and stores them in the Hash table. During reception of a frame whose Destination Address is a Multicast-Address, the 82586 maps the Address, and checks if it appears in the Hash table. If it does, an address match is determined and the frame is passed to the host.

It is possible for more than one Multicast Address to be mapped into a given Hash bit. Thus, the host may have to perform additional checking. If 64 or fewer Multicast-Addresses are used in a system, it is possible to select address values that map into unique bits in the Hash table. The Hashing function is the CRC polynomial used for bit error detection. The 6 most significant bits (2–7), are selected from the first byte of the CRC shift register to index the 64-bit Hash table.

4.0 82586 NETWORK MANAGEMENT AND DIAGNOSTIC FUNCTIONS

The behavior of data communication networks is typically very complex due to their distributed and asynchronous nature. Thus, it is particularly difficult to pinpoint a failure when it occurs. The 82586 was designed in anticipation of these problems and includes a set of features for improving reliability and testability.

The 82586 offers the four diagnostic services. First, monitoring the transmission and reception of frames. Second, statistics gathering and diagnostics of the Network as a whole. Third, diagnostic support for a particular station on the Network. Fourth, a means to test the proper operation of the chip itself.

4.1 Transmission/Reception Error Reporting

The 82586 stores, in system memory, status information after completing transmission or reception of every frame. If transmission or reception is successful, the OK status bit is set. If transmission is unsuccessful, the cause is given in the status. In case of unsuccessful reception, the cause is provided only if the 82586 is configured to Save Bad Frame, otherwise, only the statistics counters are updated.

The 82586 reports on the following events after each transmitted frame:

- Transmission unsuccessful; lost Carrier Sense.
- Transmission unsuccessful; lost Clear-to-Send.
- Transmission unsuccessful; DMA underrun occurred because the system bus did not keep up with the transmission.
- Transmission unsuccessful; the number of collisions exceeded the maximum allowed.

The 82586 checks each incoming frame and reports on the following errors, (if configured to 'Save Bad Frame'):

- CRC error: incorrect CRC in a well aligned frame. (CRC continues to be checked if the 82586 enters the NO RESOURCES state.)
- Alignment error: incorrect CRC in a misaligned frame. A misaligned frame with a correct CRC is not reported by the 82586.
- Frame too short: the frame is shorter than the configured value for minimum frame length.
- No EOF flag: valid only in Bitstuffing mode, carrier went inactive before EOF flag detection.
- Overrun: the frame was not completely placed in memory because the system bus did not keep up with incoming data.
- Out of buffers: no memory resources to store the frame, so part of the frame was discarded.

4.2 Network Planning and Maintenance

To perform proper planning, operation, and maintenance of a communication network, the network management entity must accumulate information on network behavior. The 82586 provides a rich set of network-wide diagnostics that can serve as the basis for a

network management entity. The features include: gathering network activity information, updating error counters, saving all frames that appear on the link, and locating opens or shorts on the link.

Network Activity information is provided in the status returned to the host after each frame transmitted. The activity indicators are:

1) Number of collisions: the number of collisions the 82586 experienced in attempting to transmit this frame.

2) Deferred transmission: indicates if the 82586 had to defer to traffic on the link during the first transmission attempt.

Statistics registers are updated after each received frame that passes address filtering, and is longer than the Minimum Frame Length configuration parameter. They reside in shared memory and are incremented by the 82586. The statistics registers provide the following information:

1) CRC errors: the number of frames that experienced a CRC error and were properly aligned.

2) Alignment errors: the number of frames that experienced a CRC error and were misaligned.

3) No-resources: the number of correct frames lost due to lack of memory resources.

4) Overrun errors: the number of frame sequences lost due to DMA overruns.

The 82586 can be configured to Promiscuous Mode. In this mode the 82586 captures all frames transmitted on the Network without checking the Destination Address. This mode is useful in implementing a monitoring station to capture all frames for network analysis.

Each 82586 is capable of determining if there is a short or open circuit anywhere in the network using the built in Time Domain Reflectometer (TDR) mechanism. When a TDR command (see section 8.7) is issued, the chip transmits a TDR frame and measures the reflection return time. If the network is properly terminated, there are no reflections so the timer runs out and the user is notified that there are no link problems. If a problem is detected, the distance to the reflection source and reason (short or open) are recorded.

4.3 Station Diagnostics

To support the testing of station hardware, the 82586 provides external loopback and Signal Quality Error test.

The 82586 can be configured to External Loopback. In this mode the 82586 operates full duplex at full speed. The maximum number of bytes in a frame to be looped back is 18 in the case of IEEE 802.3 (10 Mb/s) and 8 MHz system bus, but the actual maximum depends on bit rate and system bus speed. The transmitter to receiver interconnection can be placed anywhere between the 82586 and the link to locate faults, for example: the 82586 output pins, the Ethernet Serial Interface, the Transceiver cable, or in the Transceiver.

The IEEE 802.3 specification requires that the transceiver should verify the operation of the collision detect circuitry and pass the result to the controller. If the transceiver operates correctly, it sends a short 10 MHz pulse train to the controller on the Collision Detect pair. This mechanism is called the 'heart beat,' or Signal Quality Error Test, SQE TEST, signal. The SQE TEST signal is sent during the Interframe Spacing Time, just after completing transmission of a frame. The 82586 returns detection of the SQE TEST in status returned to the host at completion of transmission.

4.4 82586 Self Testing

The 82586 provides several features to check the chip operation.

The 82586 can be configured to Internal Loopback (section 11.5). In this mode, the 82586 disconnects itself from the Serial Interface Unit, and any frame transmitted is received immediately. The 82586 connects the Transmit Data to the Receive Data signal and the Transmit Clock to the Receive Clock. Internally, the Transmit Clock is divided by 4 to allow internal full duplex operation. Internal Loopback overrides External Loopback. Equality between the transmitted and received frames implies that a large portion of the chip operates correctly. In addition, internal loopback can be used in conjunction with inhibiting the source address insertion and/or CRC insertion by the chip. For example: in internal loopback, if a frame is transmitted with an erroneous CRC (using CRC inhibition), the CRC checking mechanism must detect a CRC error.

The Dump Command (see section 8.8) causes the 82586 to write over 100 bytes of its internal registers to memory. This is a very powerful capability that can serve as the basis for comprehensive diagnostics.

There are parts of the chip, in particular the logic that uses the exponential Backoff random number generator that cannot be checked from the outside. The Diagnose Command (see section 8.9) initiates a self test procedure that exercises the otherwise unaccessible registers and counters, and reports the result.

5.0 82586/HOST CPU INTERACTION

Communication between the 82586 and the host is carried out via shared memory. The 82586's direct access to memory capability allows autonomous transfer of data blocks (buffers, frames) and relieves the CPU of byte transfer overhead. The 82586 operates easily with general purpose processors, however, a minimum hardware configuration is realizable with the 80186/80188 processors. In discussing 82586/Host interaction, the logical interface and the hardware bus interface are referred to separately.

5.1 Logical Interface

The 82586 consists of two independent units: the Command Unit (CU) and the Receive Unit (RU). The CU executes commands from shared memory. The RU handles all activities related to frame reception. The CU and RU enable the 82586 to engage in the two activities simultaneously: the CU may be fetching and executing commands out of memory, and the RU may be storing received frames in memory. CPU intervention is only required after the CU executes a sequence of commands or the RU stores a sequence of frames.

The Shared Memory structure is composed of four parts: Initialization Root, System Control Block (SCB), Command List, and Receive Frame Area (RFA), see Figure 4.

The Initialization Root is at a predetermined location in the memory space, (0FFFFF6H), known to both the host CPU and the 82586. The root is accessed at initialization and points to the System Control Block. Section 6 provides details on the initialization root and the algorithms performed by the 82586 during initialization.

The System Control Block (SCB) serves as a bidirectional mailbox between the host CPU, CU and RU. It is the central element through which the CPU and the

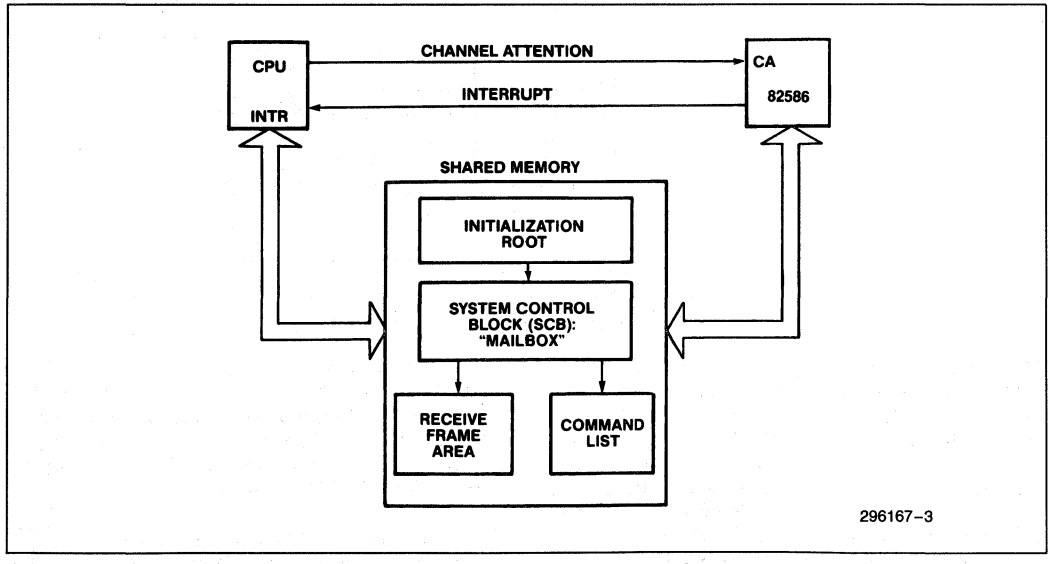


Figure 4. 82586/Host CPU Interaction

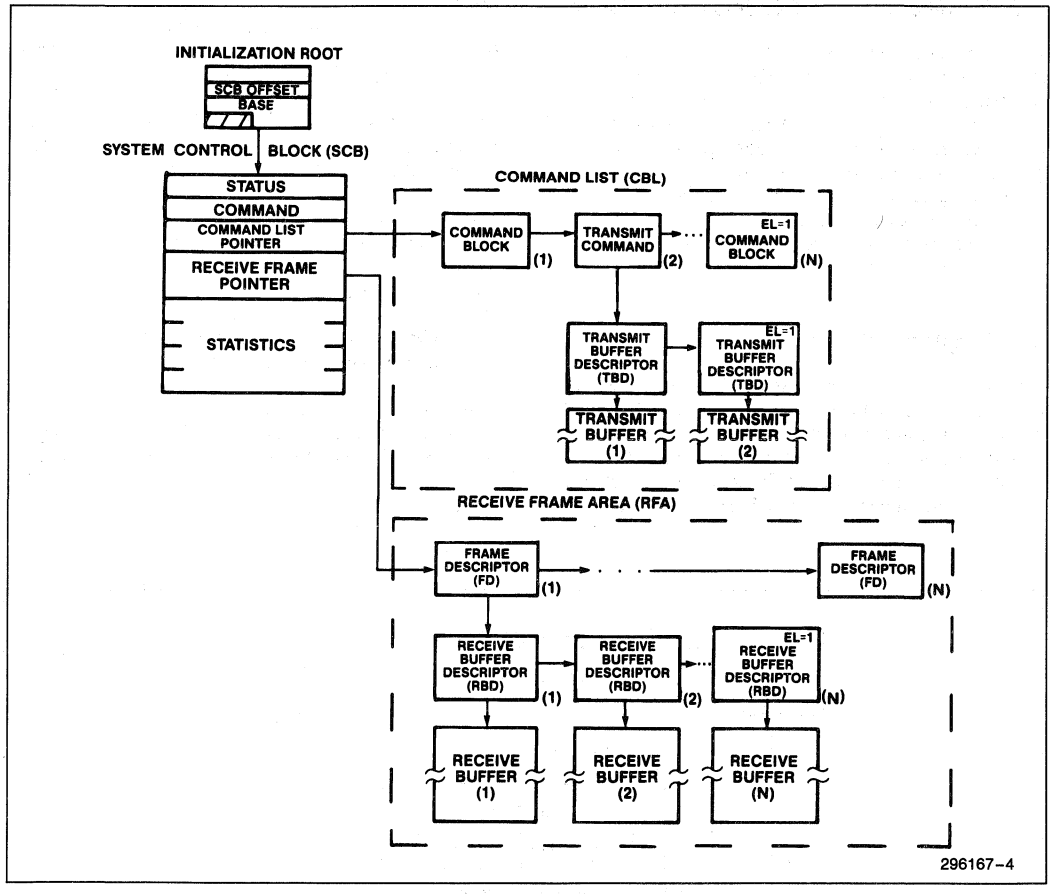


Figure 5. 82586 Shared Memory Structure

82586 exchange control and status information. The SCB is composed of two parts. First, instructions from the CPU to the 82586. These include: control of the CU and RU (START, ABORT, SUSPEND, RESUME), a pointer to the list of commands to execute a pointer to the Receive Frame Area, and a set of interrupt acknowledge bits. Second, information from the 82586 to the CPU that includes: state of the CU and RU (e.g. IDLE, ACTIVE/READY, SUSPENDED, NO RECEIVE RESOURCES), interrupt bits (command completed, frame received, CU gone not ready, RU gone not ready), and statistics. See Figure 5. Section 7 provides a detailed description of the SCB.

The only hardware signals that connect the CPU and the 82586, are the INTERRUPT and CHANNEL ATTENTION, see Figure 4. The Interrupt line is used by the 82586 to draw the CPU's attention to a change in the SCB. The Channel Attention line is used by the CPU to draw the 82586's attention.

The Command List serves as a program for the CU. Individual commands are placed in memory units called a Command Block, or CB. CBs contain command specific parameters and command specific statuses. Specifically, these commands are called Action Commands (e.g. Transmit, Configure) and are discussed in detail in section 8.

A specific command, Transmit, causes transmission of a frame by the 82586. The Command Block includes Destination Address, Length Field, and a pointer to a list of linked buffers that hold the frame to be constructed from several buffers scattered in memory. The Command Unit performs in parallel, without the CPU intervention, the DMA of each buffer and the prefetching of references to new buffers. The CPU is notified only after successful transmission or retransmission.

The Receive Frame Area is a list of Free Frame Descriptors (Descriptors not yet used) and list of buffers prepared by the user. It is conceptually distinct from the Command List. Because frames arrive without being solicited by the 82586, the 82586 must be prepared to receive them even if it is engaged in other activities and to store them in the Free Frame Area. The Receive Unit fills the buffers upon frame reception and reformats the Free Buffer List into received frame structures. The frame structure is virtually identical to the format of the frame to be transmitted. The first frame descriptor is referenced by SCB.

Receive buffer chaining (i.e. storing incoming frames in a linked list of buffers) improves memory utilization significantly. Without buffer chaining, the user must allocate consecutive blocks of the maximum frame size (1518 bytes in IEEE 802.3) for each frame. Maximum frame length buffers are inefficient because 75 percent of the frames on a network are control (request for status, message acknowledgement, etc.) frames, which are typically less than 100 bytes. With buffer chaining, the user can allocate small buffers and the 82586 uses only as many as needed.

In the past, the drawback of buffer chaining was CPU processing overhead and the time involved in buffer switching (especially at 10 Mbps). The 82586 overcomes this drawback by performing buffer management in hardware, completely transparent to the user.

Section 9 provides details on the format of the Received Frame Area and the algorithms associated with frame reception.

5.2 Hardware Bus Interface

The local bus interface has 24 address lines. The low order 16 lines are multiplexed with data, and address line number 19 is multiplexed with status. Like other Master peripherals, the 82586 provides all control signals required to handle Direct Memory Access. The bus interface operates at up to 10 MHz.

Similar to the 8086, the 82586 can be pin strapped into either Minimum mode or Maximum mode. Minimum mode is used in small systems that do not require external bus controller circuitry. The 82586 provides the minimum bus control information. Maximum mode is used in systems that use large memory, and have system peripherals.

The maximum Data Transfer Rate on the bus interface is 4 megabytes per second when the clock is run at 8 MHz. Note that this is significantly higher than required by the IEEE 802.3 (which is 1.25 megabytes per second). This leaves much bus bandwidth for 82586 buffer, status and control overhead and general application processing. Although the 82586 performs command chaining, frame chaining, and buffer chaining on the fly, it can operate with buses that transfer data as slow as 2 megabytes per second without DMA overruns or underruns.

The 82586 shares the system bus with the host CPU, and possibly other peripherals. Therefore, there is a delay between the time the 82586 requests the bus and receives it. This delay is referred to as bus latency. Latency time is typically 1 to 2 microseconds but may reach about ten microseconds in worst case situations. Note that 10 microseconds is equivalent to 100 data bits, or 12.5 bytes. DMA overruns or underruns due to bus latency is significantly reduced by the 82586's individual on-chip transmit and receive FIFOs. Associated with the FIFOs is a user programmable threshold mechanism that improves the bus access efficiency by making the traffic bursty.

Section 10 provides details on all aspects of the Bus Interface.

5.3 Memory Addressing

The 82586 accesses memory with 24-bit addresses (in Minimum mode the two most significant bits are used as $\overline{RD}$ and $\overline{WR}$ lines). The memory structure uses two types of address representation: Physical (Real) and Segmented.

A Physical Address is a single 24-bit entity that specifies the physical byte address in memory. The representation of this type of address is in three consecutive bytes in memory, starting at an even location (see Figure 6). It is used for referring to all the buffers as well as to elements of the Initialization Root. When the chip is configured to operate with a 16-bit bus, the Physical Address must be even (least significant bit = 0).

A Segmented Address consists of a base and offset. The base is a 24 bits long real address and the offset is

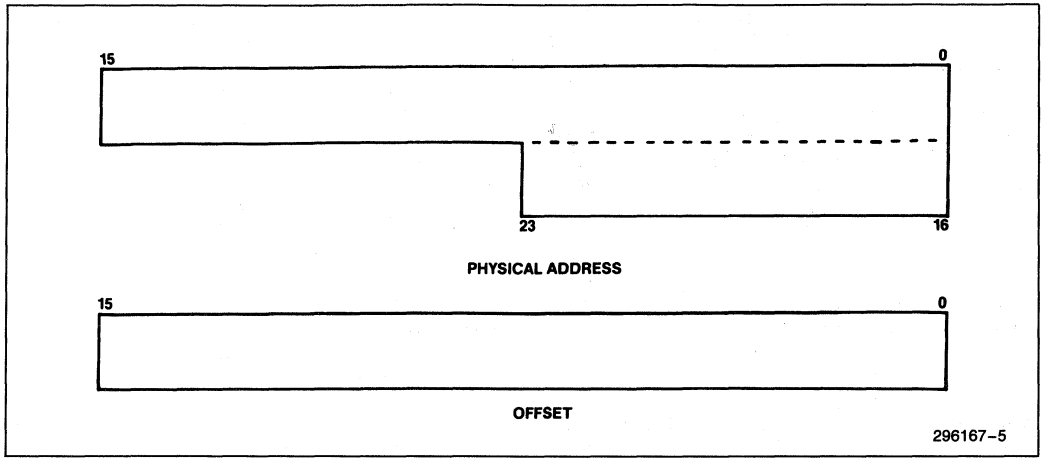


Figure 6. Memory Addressing

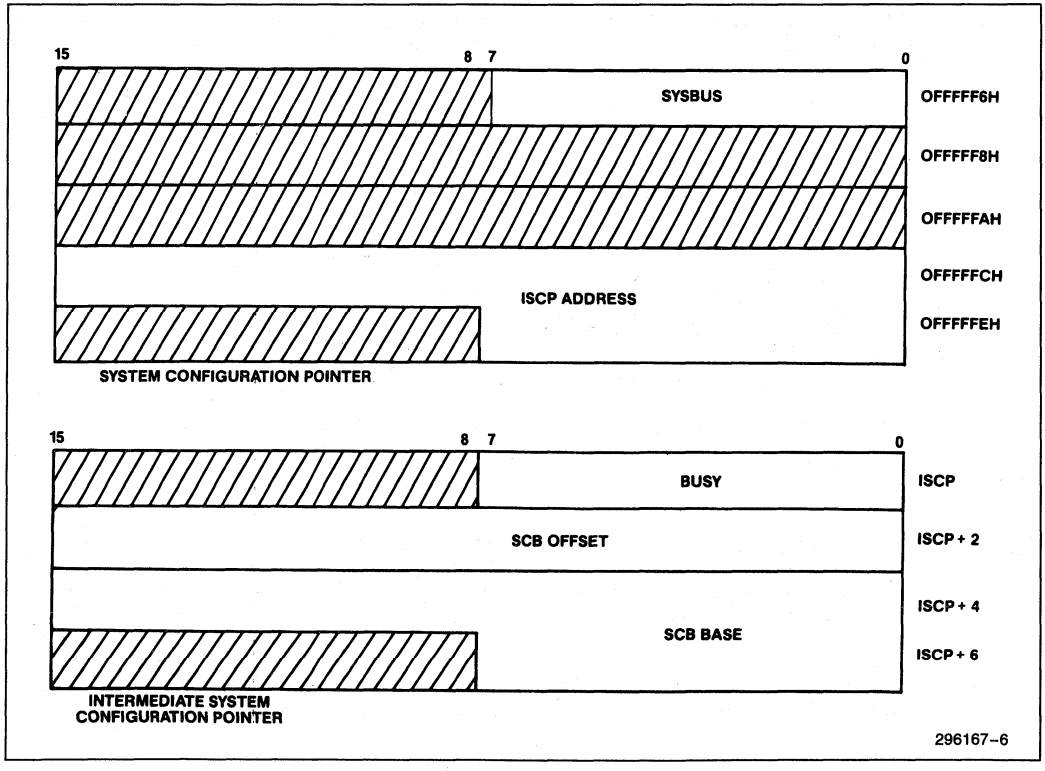


Figure 7. Initialization Root

16 bits long. The base is setup during the initialization process and remains the same until the chip is reset. The base must be even in all cases. The 82586 calculates the physical address by adding the base to the offset. The representation of the offset is a word starting on an even byte boundary in memory, (see Figure 6). Segmented Addresses are used for referring to most parts of the memory structure. It must be even in all cases.

6.0 INITIALIZING THE 82586

After the clocks (System, Transmit and Receive) applied to the 82586 have stabilized and a RESET is issued, the 82586 performs an initialization procedure that prepares it for normal operation. This section specifies the memory structure involved in the initialization, how the user must prepare it, and how the chip behaves during initialization.

6.1 Initialization Root Format

Initialization involves the System Configuration Pointer (SCP) and the Intermediate System Configuration Pointer (ISCP). Their formats are described in detail below, and they are shown in Figure 7.

SYSTEM CONFIGURATION POINTER (SCP)

The SCP is located in the ten bytes from 0FFFFF6H to 0FFFFFFH, the highest bytes in the address space. Note that this is a fixed address, the only one in the system.

Only two items of information are present in the SCP, the system bus width and a pointer to ISCP, the next structure. ISCP is an arbitrary location in the address space. The SCP includes the following fields:

0FFFFF6H:

BUS (Bit 0) - This bit specifies the width of the system bus. It takes the following values:

0 16-bit bus

1 8-bit bus

ISCP ADDRESS: This 24-bit value is the ISCP physical address.

The 82586 is indifferent to the remaining SCP contents. This does not mean that these areas have no function, they may be meaningful with respect to other peripherals which refer to SCP. Because the SCP and ISCP are sampled by the 82586 only during initialization, they may be used at any other time by other peripherals.

INTERMEDIATE SYSTEM CONFIGURATION POINTER (ISCP)

The ISCP is common to the 82586 and all other master peripherals referring to the same SCP. At the same time however, information contained in ISCP is specific to each master peripheral, and the structure contains a status element. Consequently, ISCP must be located in RAM, and not in ROM with SCP. This distinction is the reason for separating SCP and ISCP. The ISCP includes the following fields:

BUSY (8 bits): when set to 01H, indicates that the 82586 is in the initialization process. The 82586 clears this byte immediately after reading the information contained in ISCP.

SCB-OFFSET: the address offset of the SCB (within the segment defined by the SCB-BASE).

SCB-BASE: a 24-bit value giving the starting address of the 64-kilobyte segment containing SCB and all other control structures dealing with the 82586.

6.2 Initialization Process

The initialization process establishes communication between the CPU and the 82586. Without it, no interaction takes place.

Prior to starting the initialization process, the CPU must setup the following fields in SCP and ISCP:

- BUS - memory bus width specification.
- ISCP ADDRESS - physical address of ISCP.
- BUSY - this field in ISCP must be 01H to indicate that the CPU is ready for initialization. Subsequent clearing of the field by the 82586, indicates initialization completion.
- SCB BASE - base address of the entire memory structure.
- SCB OFFSET - offset (from SCB BASE) of SCB. After completing initialization, all 82586 control is via SCB.

The CPU must reset the 82586. After power-up, this must be done with a hardware RESET. Subsequently, a software RESET (specific to the 82586) may be used. RESET causes all major internal flags to be set to their inactive states. In particular, CU and RU are set to IDLE state and configuration parameters are set to their default values (see section 7.5).

Initialization is triggered by the Channel Attention (CA) signal from the CPU. Note that initialization can only occur by issuing the sequence: RESET-CA. After CA, the 82586 performs the following sequence:

1) Reads the first word from location 0FFFFF6H. The least significant bit determines bus width of all subsequent memory accesses. Before reading this word, the system bus width is assumed to be 8 bits.

2) Reads the ISCP ADDRESS from location 0FFFFFCH and the following word.

3) Reads SCB OFFSET from the word following the one determined by ISCP ADDRESS. SCB OFFSET is saved for subsequent references to SCB.

4) Reads SCB BASE field from the next two words.

5) Clears the BUSY byte. This is done by reading the word in location ISCP, clearing the least significant byte, and writing back to the word (with the cleared byte).

6) The SCB BASE (read in Step 4) is saved internally to serve as a base for all subsequent references to the memory structure (excluding data buffers).

7) Writes the STATUS word of SCB with CX and CNA bits set, all remaining fields are cleared (see section 7.1).

8) Raises the INTERRUPT Hardware Signal.

The chip is now ready to be controlled by the host CPU via SCB, as described in the next section.

7.0 CONTROLLING THE 82586

This section discusses how the CPU controls operation of the 82586's Command Unit (CU) and Receive Unit (RU). Operation of the CU and RU themselves, namely how the 82586 executes Action Commands and receives frames, is discussed in subsequent sections. This section provides complete explanations of the following functions:

- Starting and Completing Control Commands
- CU Control
- RU Control
- RESET
- Statistics Registers

7.1 System Control Block (SCB) Format

The System Control Block is the communication mailbox between the 82586 and the host CPU. The SCB format is shown in Figure 8.

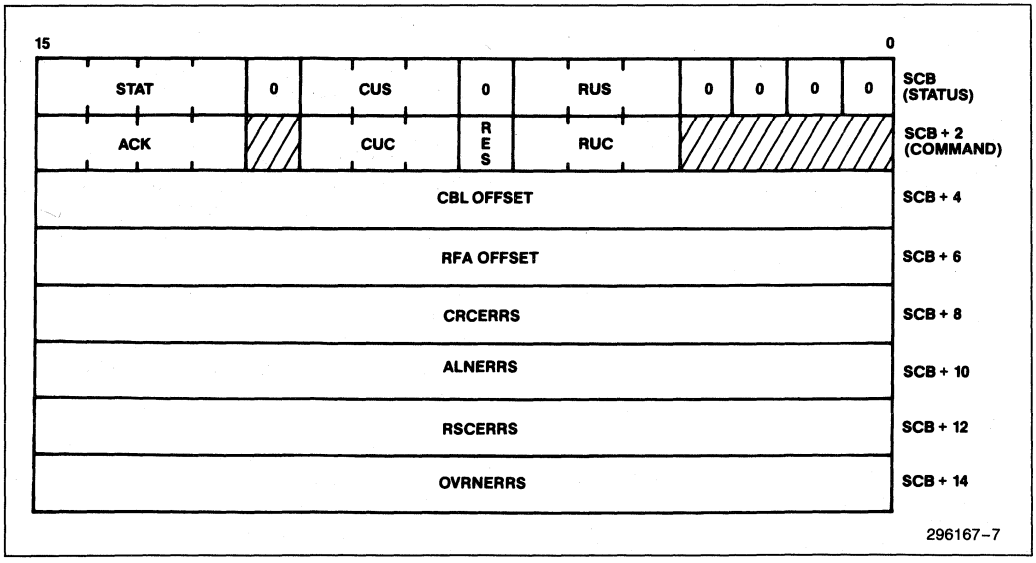


Figure 8. System Control Block (SCB) Format

The host CPU issues Control Commands to the 82586 via the SCB. These commands may appear at any time during routine operation, as determined by the host CPU. The CPU informs the 82586 that a control command is ready by issuing a CA signal.

The SCB is also used by the 82586 to return status information to the host CPU. After inserting the required status bits into SCB, the 82586 issues an Interrupt to the CPU.

The format is as follows:

STATUS WORD:

Indicates status of the 82586 to the CPU. This word is modified only by the 82586. Defined bits are:

CX (Bit 15) - The CU finished executing an Action Command with its 'I' (Interrupt) bit set.

FR (Bit 14) - The RU finished receiving a frame.

CNA (Bit 13) - The CU left the ACTIVE state.

RNR (Bit 12) - The RU left the READY state.

CUS (Bits 8–10) - (3 bits) this field contains the status of the Command Unit.

Valid values are:

0 - IDLE
1 - SUSPENDED
2 - ACTIVE
3–7 - Not used

RUS (Bits 4–6) - (3 bits) this field contains the status of the Receive Unit.

Valid values are:

0 - IDLE
1 - SUSPENDED
2 - NO RESOURCES
3 - Not used
4 - READY
5–7 - Not used

The remaining bits of the status word are always set to zero by the 82586.

COMMAND WORD:

Provides the communication mechanism from the CPU to the 82586. This word is set by the CPU and cleared by the 82586. Defined bits are:

ACK-CX (Bit 15) - Acknowledges that the CU completed an Action Command.

ACK-FR (Bit 14) - Acknowledges that the RU received a frame.

ACK-CNA (Bit 13) - Acknowledges that the Command Unit left the ACTIVE state.

ACK-RNR (Bit 12) - Acknowledges that the Receive Unit left the READY state.

CUC (Bits 8–10) - (3 bits) this field contains the command to the Command Unit. Valid values are:

0 - NOP (doesn't affect current state of the unit).
1 - Start execution of the **first** command on the Command List (CBL). If a command is in execution, then complete it before starting the new CBL. The beginning of the CBL is in CBL OFFSET.
2 - Resume the operation of the Command Unit by executing the next command. This operation assumes that the Command Unit has been previously suspended.
3 - Suspend execution of commands on the CBL after current command is complete.
4 - Abort current command immediately.
5–7 - Reserved, illegal for use.

RUC (Bits 4–6) - (3 bits) This field contains the command to the receive unit. Valid values are:

0 - NOP (does not alter current state of unit).
1 - Start reception of frames. If a frame is being received, then complete reception before starting. The beginning of the RFA is contained in the RFA OFFSET.
2 - Resume frame receiving (only when in the SUSPEND state.)
3 - Suspend frame receiving. If a frame is being received, then complete its reception before suspending.
4 - Abort receiver operation immediately.
5–7 - Reserved, illegal for use.

RESET (Bit 7) - Reset chip (logically the same as hardware RESET).

CBL-OFFSET: Gives the 16-bit offset address of the first command (Action Command) in the Command List to be executed following CU-START. Thus, the 82586 will read this word only if the CUC field contains a CU-START Control Command.

RFA-OFFSET: Gives the 16-bit offset address of the first Receive Frame Descriptor in the Receive Frame Area, to be accessed following a RU-START Control command. Thus, the 82586 will read this word only if the RUC field contains a RU-START Control Command.

CRCERRS: CRC Errors - contains the number of properly aligned frames received with a CRC error.

ALNERRS: Alignment Errors - contains the number of misaligned frames received with a CRC error.

RSCERRS: Resource Errors - records the number of correct incoming frames discarded due to lack of memory resources (buffer space or Receive Frame Descriptors).

OVRNERRS: Overrun Errors - counts the number of received frame sequences lost because the memory bus was not available to the 82586 in time to transfer them.

7.2 Starting and Completing Control Commands

The CPU issues Control Commands by writing in the SCB COMMAND field and issuing Channel Attention (CA). Acceptance of a Control Command is indicated by the 82586 clearing the SCB COMMAND field. Therefore, the CPU must wait for this word to be cleared before the next Control Command can be issued.

The 82586 does not necessarily accept the Control Command immediately after CA is issued; it may be engaged in higher priority tasks. For example, prefetching new buffers, handling buffer switches, or finishing frame reception. When the 82586 becomes available it performs the Starting of Control Command sequence:

1) Clears hardware Interrupt signal.

2) Reads the SCB COMMAND word, and analyzes its fields.

3) If the RESET bit is set, it performs the RESET sequence, as described in section 7.5.

4) Clears the internal INTERRUPT-IS ('In-Service') flag for each acknowledged interrupt bit.

5) If the RUC field is different from zero, an internal request to the RU to perform a RU command acceptance sequence is issued (see section 7.4).

6) If the CUC field is different from zero, the acceptance sequence for a CU command is performed immediately (see section 7.3).

After both CU and RU complete the acceptance sequence, the 82586 performs the Completion of Control Command sequence:

1) If there is any internal interrupt request then it sets the respective INTERRUPT-IS flag.

2) Update SCB STATUS word, according to internal CU status, RU status and Interrupt requests.

3) If any INTERRUPT-IS is set then the 82586 sets the hardware Interrupt signal.

4) Clears SCB COMMAND word.

7.3 Command Unit (CU) Control

The Command Unit is the logical unit that executes Action Commands from a list of commands very similar to a CPU program. A Command Block (CB) is associated with each Action Command.

This section describes how the CPU controls Action Command execution, namely, how it starts, stops, suspends or resumes the CU. Execution of the Action Commands themselves, is the subject of the next section.

The CU can be modeled as a logical machine that takes, at any given time, one of the following states:

- IDLE - the CU is not executing a command and is not associated with a CB on the list. This is the initial state.
- SUSPENDED - the CU is not executing a command but (different from IDLE) is associated with a CB on the list.
- ACTIVE - the CU is currently executing an Action Command, and points to its CB.

The CPU may affect the CU operation in two ways: issuing a CU control Command or setting bits in the COMMAND word of the Action Command. In general, the CPU can cause the CU to do the following:

- Start executing a list of Action Commands.
- Suspend execution after completing the current Action Command.
- Resume execution if the CU is in the SUSPENDED state,
- Abort execution and return to the IDLE state.
- Stop execution after completing a particular Action Command. This will be the last CB in the list.
- Suspend execution after completing a particular Action Command.
- Have the 82586 issue Interrupts after completing particular Action Commands.

There are three points in time relevant to the execution of Action Commands:

- Acceptance time (of a control command)—the time following a Channel Attention issued by the CPU, the CU reads the Control Command and takes initial action.
- Beginning of Execution—the time the CU starts executing an Action Command. This may be following a Control Command or in continuing the list of Action Commands. The details are presented in section 8.1.
- Completion of Execution—the time the CU completes executing an Action Command. This is the decision time for the CU on how to proceed. The details are presented in section 8.1.

The CU uses 3 internal flags to remember requests from acceptance time to be acted upon at completion of execution: CU-START-REQUEST, CU-SUSPEND-REQUEST and CU-ABORT-REQUEST.

At acceptance Time, after the 82586 has finished higher priority tasks and a CA is detected, it reads the SCB command word and analyzes its fields. The higher priority tasks are: receive end of frame processing, receive or transmit buffer prefetching, retransmission due to collisions, completion of Action Commands. The CUC field is one of the following: CU-START, CU-SUSPEND, CU-RESUME or CU-ABORT.

CU-START

Starts execution of the first Action Command in the list. The CU performs the following sequence:

1) Reads CBL OFFSET and saves it as a pointer to the next CB to be executed.

2) If the CU is not in the ACTIVE state, it goes to the ACTIVE state and requests the beginning of the next CB.

3) If the CU is in the ACTIVE state, then it does nothing at Acceptance time. Beginning of the next CB (i.e. the new one) starts after completion time of the current CB.

CU-SUSPEND

Suspends operation of the CU after completing the current CB. The 82586 performs the following sequence:

1) If the CU is in the ACTIVE state, it sets the internal CU-SUSPEND-REQUEST flag. This flag causes CU to enter the SUSPENDED state at command completion time.

2) If the CU is not in the ACTIVE state, it ignores the CU-SUSPEND command.

CU-RESUME

Resumes CU operation. The 82586 performs the following:

1) If the CU is in the SUSPEND state, it goes to the ACTIVE state and requests the beginning of the next CB.

2) If the CU is in the ACTIVE state, it clears internal CU-SUSPEND-REQUEST to prevent suspension at command completion time.

3) If the CU is in the IDLE state, it ignores the CU-RESUME command.

CU-ABORT

Causes the CU to stop all activity immediately and go to the IDLE State. The CU performs the following:

1) If the CU is not in the ACTIVE state, it goes to the IDLE state.

2) If the CU is in the ACTIVE state, it sets the internal CU-ABORT-REQUEST flag. This flag causes the CU to go to the IDLE state at command completion time.

The CU ABORT control command causes the immediate termination (at acceptance time) of the transmission process. The effect on specific Action Commands (see section 8) is as follows:

- NOP, TDR, DUMP and DIAGNOSE commands are not stopped. Following execution, the CU switches to its IDLE state.

- IA setup, MC setup, CONFIGURE commands are always stopped. A command aborted bit is set in the CB status field. The CPU **MUST** execute these commands again since the 82586 may not be properly programmed to continue its operation.
- There is an attempt to abort a TRANSMIT. The DMA process is stopped. In certain timings, the Abort attempt may not be successful. The 82586 internally tests for this situation and flags CMD ABORTED status only if the TRANSMIT is known to have failed.

NOTES ON ABORTED TRANSMISSION:

1) The CU attempts to stop as soon as possible by stopping DMA and forcing a FIFO underrun.

2) Aborted transmit frames may result in small frames on the Serial Link. The 82586 sends to the Serial Link, the partial frame and appends to it four bytes of 'All Ones' (jam pattern).

3) Apart from aborted transmit frames and collided frames, the 82586 appends the jam pattern, also in the following cases:
   - Underrun on transmit buffers.
   - Lost Carrier Sense, if programmed to stop transmission when Carrier Sense is lost.

See Figure 9.

SUSPENDING THE CU

The START, SUSPEND, RESUME and ABORT commands can be given through the SCB command fields as described above. The Command Block offset pointer in the SCB points to the first executable Action Command (to be discussed in section 8). It is sufficient to state here that these Action Commands have a field for EL and S Bits (see section 8.1). The EL, End of List, bit indicates that the current Action Command is the last on the Command List. The S Bit indicates that the user desires the CU to enter into Suspended state after executing the current command. Tables 1 and 2 illustrate the state transitions for various conditions.

Thus, there are two mechanisms to suspend the CU: one through the Suspend Command in SCB, and the other through the S bit in the Action Command Block. The Suspend Command in SCB will suspend the CU after completing the execution of the current command on the Command List, without knowing exactly which command the 82586 was executing before execution. Suspension of CU through Action Command Block will result in suspension after a specific command is executed. In both cases, the CU can be activated by issuing a Resume command.

Table 1. CU Control Commands: Actions at Acceptance Time

Present State	Start		Resume		Suspend		Abort	
	Next State	Action	Next State	Action	Next State	Action	Next State	Action
Idle	Active	Set Up CB	Idle	None	Idle	None	Idle	None
Suspended	Active	Set Up CB	Active	Set Up CB	Suspended	None	Idle	None
Active	Active	None	Active	None	Active	Request Suspend	Idle	Abort DMA/CNA Interrupt

Table 2. CU Activities Performed at End of Execution

EL Bit	S Bit	Request	Next State	Action
0	0	None	Active	Set Up CB
0	0	Suspend	Suspended	CNA Interrupt
0	1	None	Suspended	CNA Interrupt
0	1	Suspend	Suspended	CNA Interrupt
1	0	None	Idle	CNA, CX Interrupts
1	0	Suspend	Idle	CNA, CX Interrupts
1	1	None	Idle	CNA, CX Interrupts
1	1	Suspend	Idle	CNA, CX Interrupts

NOTES:
1) After a CB with 'I' bit set is completed, CX interrupt is generated.
2) Since the transition ACTIVE to ACTIVE STATE via the START Command is smoothly performed, no interrupt, related to state transition, is generated and no action is required at the end of Action Command Execution.

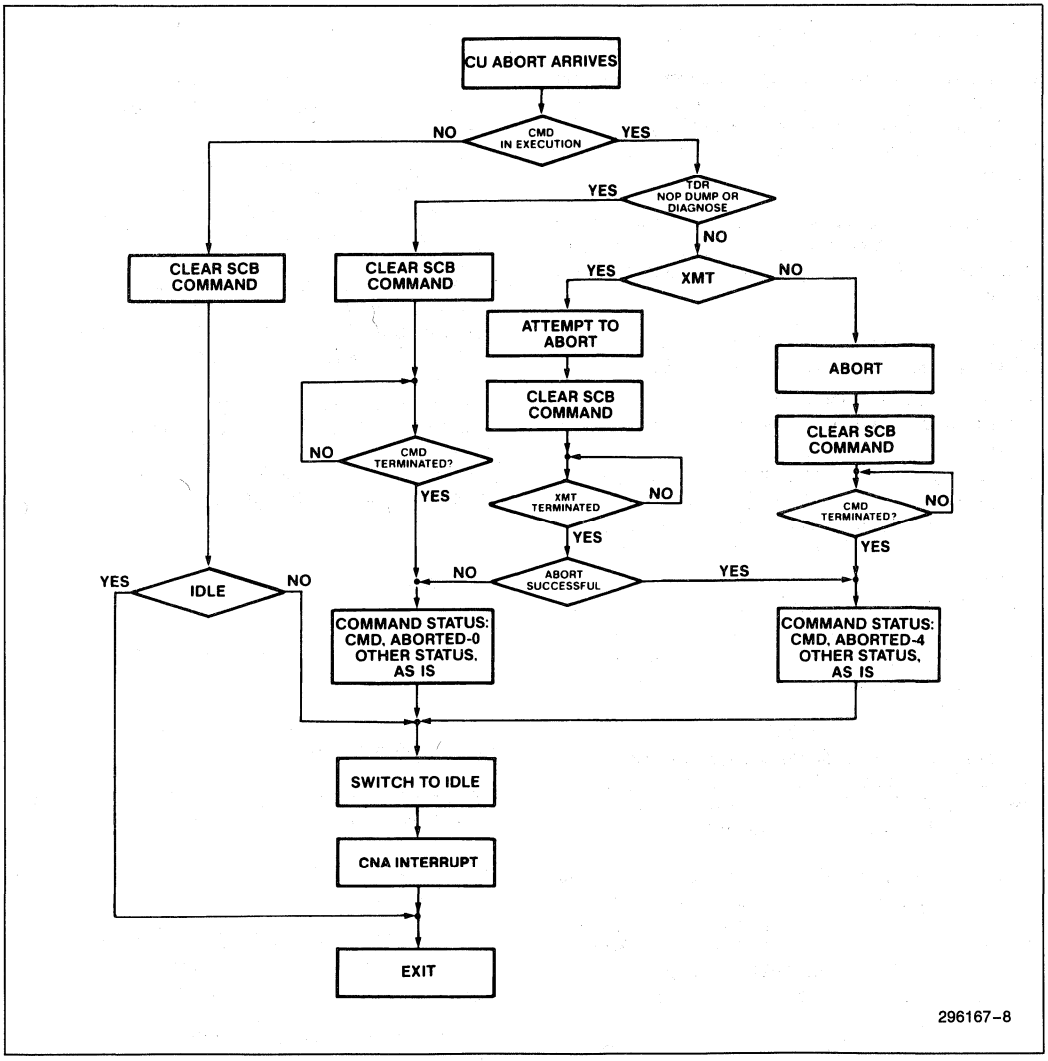


Figure 9. CU Abort Flowchart

7.4 Receive Unit (RU) Control

The Receive Unit is the logical unit that receives frames and stores them in memory. The RU uses free (i.e. unused) buffers and descriptors prepared by the CPU. The details on RU frame reception are presented in section 9.

This section describes how the CPU controls frame reception, namely, how it starts, stops, suspends, or resumes the RU. Reception may also stop due to No Resources.

The RU is modeled as a logical machine that takes, at any given time, one of the following states:

- IDLE - The RU has no memory resources and is discarding incoming frames. This is the initial RU state.
- NO-RESOURCES - The RU has no memory resources and is discarding incoming frames. This state differs from the IDLE state in that RU accumulates statistics on the number of frames it had to discard.
- SUSPENDED - The RU has free memory resources to store incoming frames but discards them anyway.
- READY - The RU has free memory resources and is ready to store incoming frames.

NOTE:
Frames arrive at the 82586 independent of the state of the RU. The situation that a frame is arriving is referred to as Actively Receiving, even when RU is Not Ready and the frame is being discarded.

The CPU may affect RU operation in three ways: issuing an RU Control Command, setting bits in the Frame Descriptor COMMAND word of the frame currently being received, or setting the EL (End of List) Bit of the Receive Buffer Descriptor, RBD, of the buffer currently being filled. In general, the CPU may cause the RU to do the following:

- Start frame reception.
- Suspend frame reception (start discarding) after current frame reception is complete.
- Resume reception if the RU is in SUSPENDED state.
- Abort reception at once and return to IDLE state.
- Stop reception after a particular FD is filled (frame received). This FD is referred to as the last FD on the list.
- Suspend reception after particular FD is filled (frame received).
- Stop reception after a particular RBD is filled. This is the last RBD in the list.

NOTE:
The RU issues an Interrupt after every received frame.

There are two points in time relevant to frame reception:

- Acceptance time - the time after a Channel Attention is issued by the CPU, the RU reads the Control Command and takes initial action.
- Completion of Reception - the time the RU ends receiving or discarding an incoming frame. This is a decision time for the RU on how to proceed and is discussed in detail in section 9.2.

The RU uses two internal flags to remember requests from acceptance time to be acted upon at completion of reception time: RU-START-REQUEST and RU-SUSPEND-REQUEST.

At acceptance time, after CU passes the Control Command to the RU and finishes accepting its own Control Command, the RU starts analyzing its command. The SCB's RUC field may take one of the following values: RU-START, RU-SUSPEND, RU-RESUME, or RU-ABORT.

RU-START

The RU is made ready to receive frames. At Acceptance time, the RU may or may not be actively receiving a frame.

First case - the RU performs the following sequence if it is actively receiving:

1) Clears the internal RU-START-REQUEST and RU-SUSPEND-REQUEST flag.

2) Reads the RFA OFFSET word from SCB and saves it internally as the pointer to the next FD.

3) Sets RU-START-REQUEST, thus when the current frame has been received or discarded, the RU goes to the READY state and sets up the next FD.

Second case - The RU performs the following sequence if it is not actively receiving:

1) Clears internal RU-START-REQUEST and RU-SUSPEND-REQUEST flags.

2) Reads RFA OFFSET word from SCB and saves it internally as the pointer to the next FD.

3) If the RU is not in the READY state, or the RU is in the READY state and assures that DMA did not transfer any data to the current FD, it does the following: it stops discarding, goes to the READY state, gives up buffers that have been prefetched (if any), and sets up a new FD. Setting up a new FD uses the pointer to the next FD to prepare reception of the next frame (see section 9).

4) If the RU is in the READY state and DMA started transferring data to memory before the RU had the opportunity to stop it, the following occurs: the RU goes to the NO-RESOURCES state (without issuing a RNR Interrupt) and sets the internal RU-START-REQUEST. Next, (in the particular case where RU does not manage to stop DMA in time) one frame is discarded and the user is notified by temporarily being in the NO-RESOURCES state.

The CPU must ensure that the 82586 is properly configured before starting the RU. This rule specifically applies to IA-SETUP, MC-SETUP, CONFIGURE Action Commands. Failure to perform CONFIGURE Action Command before stating the RU may result in unpredictable behavior of the receive process.

Failure to perform IA-SETUP before a RU START may result in a coincidental address match on a received frame.

RU-SUSPEND

Suspends RU operation after reception of current frame is complete, or reception of first frame to be received is complete. The RU performs the following:

1) If the RU is in the READY state, sets internal RU-SUSPEND request.

2) If the RU is not in the READY state, ignores the command.

RU-RESUME

Resumes frame reception. The RU performs the following sequence:

1) Clears RU-SUSPEND-REQUEST.

2) If the RU is in the SUSPENDED state, and not actively discarding a frame, it goes to the READY state and sets up a new FD (see section 9).

Table 3. RU Control Commands - Actions at Acceptable Time

		Start		Resume		Suspend		Abort	
	Present State	**Next State**	**Action**	**Next State**	**Action**	**Next State**	**Action**	**Next State**	**Action**
RU Not Actively Receiving	**Idle**	Ready	Set Up FD	Idle	None	Idle	None	Idle	None
	No Resources	Ready	Set Up FD	No Resources	None	No Resources	None	Idle	None
	Suspended	Ready	Set Up FD	Ready	Set Up FD	Suspended	None	Idle	None
		Ready	Set Up FD	Ready	None	Ready	Request Suspend	Idle	Start Discarding RNR Interrupt
RU Actively Receiving	**Idle**	Idle	Request Start	Idle	None	Idle	None	Idle	None
	No Resources	No Resources	Request Start	No Resources	None	No Resources	None	Idle	None
	Suspended	Suspended	Request Start	Suspended	Request Start	Suspended	None	Idle	None
	Ready	Ready	Request Start	Ready	None	Ready	Request Suspend	Idle	Abort DMA Start Discarding RNR Interrupt

Table 4. RU Activities Performed at End of Execution

EL Bit	S Bit	Request	Next State	Action
0	0	None	Ready	Set Up FD
0	0	Suspend	Suspended	RNR Interrupt
0	0	Start	Ready	Set Up FD
0	1	None	Suspended	RNR Interrupt
0	1	Suspend	Suspended	RNR Interrupt
0	1	Start	Suspended	RNR Interrupt
1	0	None	No Resources	RNR Interrupt
1	0	Suspend	No Resources	RNR Interrupt
1	0	Start	Ready	Set Up FD
1	1	None	No Resources	RNR Interrupt
1	1	Suspend	No Resources	RNR Interrupt
1	1	Start	Ready	Set Up FD

NOTE:
After a frame is received, FR interrupt is generated.

3) If the RU is in the SUSPENDED state and actively discarding a frame, it sets RU-START-REQUEST.

4) If the RU is not in the SUSPENDED state, it ignores the command.

RU-ABORT

RU stops reception immediately and goes to the IDLE state. RU performs the following:

1) If the RU is in the READY state, it requests an RNR Interrupt.

2) Stops all DMA activity and starts discarding incoming data.

3) Changes the RU to the IDLE state.

7.5 Reset

The 82586 has both a software and hardware RESET. After power up, a hardware RESET is required by the 82586. A Channel Attention following this reset will result in the 82586 accessing the System Control Pointer located at 0FFFFF6H. The first access is made on an 8-bit data boundary. Depending on the contents of location 0FFFFF6H, the subsequent access will be made on 8 or 16-bit data boundary.

A software RESET is available on the 82586 through bit 7 of the Control Command word in the System Control Block. It may be used after the 82586 has been initialized and it has the ISCP and SCP addresses. The software RESET is intended to operate selectively on a particular 82586.

The hardware RESET causes all major hardware control flip flops to be cleared. The CU and RU also clear their internal flags, they include:

END-OF-LAST-BUFFER = 0
ID-FULL = 0
RU-state = IDLE
CU-state = IDLE
CU-SUSPEND-REQUEST = 0
CU-ABORT-REQUEST = 0
CNA-REQUEST = 0
CX-REQUEST = 0
RU-SUSPEND-REQUEST = 0
RU-START-REQUEST = 0
RNR-REQUEST = 0
PR-REQUEST = 0
ADDRESS-LENGTH = 6

NOTE:
The System and Transmit clocks must be stable at their correct levels and timings before hardware RESET can be issued.

The CU performs the following upon recognition of a software RESET:

1) Terminates DMA activity.

2) Writes all zeros to SCB COMMAND word.

3) Triggers a hardware RESET.

NOTE:
A Channel Attention from the CPU must be delayed for at least 8 clocks after the SCB COMMAND word (with a RESET) is cleared. Several internal flip flops, including the internal Channel Attention flip flop, are cleared 8 clocks after the SCB COMMAND word is cleared therefore, a premature Channel Attention may not get recognized.

7.6 Error Statistics Registers

The last four words in the SCB are statistics registers that accumulate tallies on: CRC errors, Alignment errors, number of frames lost due to No Resources, number of frames lost due to DMA overruns.

The 82586 increments the registers at Completion of Reception time if the appropriate event happens and the register has not reached 0FFFFH. The registers are 'sticky,' i.e. they do not wrap around from 0FFFFH to zero. The registers can be cleared (or set to any value) only by the host CPU.

The CPU cannot update the register while the 82586 is incrementing it (restoring the old value plus one) because the 82586 performs the read counter/increment/write counter operation without relinquishing the bus. This is done to ensure that no logical contention exists between the 82586 and the CPU.

In a dual port memory configuration, the CPU should check the state of the counter immediately after updating it. This practice checks for the case where the 82586 placed into the counter the incremented 'old counter' value after the CPU has cleared the counter. Because the 82586 does not write to the counter when 0FFFFH is reached, the CPU may safely reset the counter without this check.

Incrementing the registers does not depend on the state of the Receive Unit. The CRC, Alignment and overrun error counters are incremented for each frame that experiences the specific error, was targeted for this specific station, and was longer than the Minimum Frame Length configuration parameter. The No Resources counter is incremented when a frame targeted to this station, has no error, and is lost or partially lost because there was no room in the memory structure.

The counters get incremented during External Loop Back operation, as well as in normal Transmit and Receive operation.

CRC ERROR COUNTER:

Contains the number of properly aligned frames received with a CRC error. It is incremented by the 82586 for every frame with a CRC error that was targeted for this specific station and was longer than the Minimum Frame Length.

ALIGNMENT ERROR COUNTER:

Counts the number of misaligned frames received with a CRC error. It is incremented by the 82586 for every misaligned frame with a CRC error that was targeted for this specific station and was longer than the Minimum Frame Length.

NO RESOURCES ERROR COUNTER:

Records the number of correctly received frames discarded due to the lack of memory resources (buffer space or received frame descriptors). It is incremented by the 82586 for every frame discarded due to lack of memory resources that was targeted for this specific station and did not experience any error.

OVERRUN ERROR COUNTER:

Counts the number of received frame sequences lost because the memory bus was not avaialble in time to transfer them. Note that more than one frame may get lost due to overrun, and in this case the counter is incremented only once.

7.7 SCB Status Update

The 82586 updates the SCB status word in the following events:

1) At the end of the initialization procedure, the 82586 writes SCB status denoting CU IDLE, RU IDLE, CX interrupt and CNA interrupt.

2) When a control command is accepted, the 82586 updates the SCB status and clears the SCB command word to denote 'acceptance completed.'

3) After execution of an Action Command. However, the CX bit setting is directly connected to the I bit.

4) After receiving a frame.

8.0 ACTION COMMANDS

The 82586 executes a 'program' that is made up of Action Commands in the Command List, CBL. As shown in Figure 10, each command contains the command field, status and control fields, link to the next Action Command in the CBL, and any command-specific parameters. This command format is called the Command Block.

The 82586 has a repertoire of 8 commands:

NOP	Transmit
Individual Address Setup	TDR
Configure	Diagnose
Multicast Address Setup	Dump

NOP:

This command results in no action by the 82586 other than the normal command processing, such as fetching the command and decoding the command field.

INDIVIDUAL ADDRESS SETUP:

This command is used to load the 82586's unique address. The unique address is contained in the parameter field of the command.

CONFIGURE:

The Configure command is used to load the 82586 with its operating parameters. Upon reset, the 82586 initializes to the IEEE 802.3 based parameters. If the user wishes to use any other values, the Configure command is used.

MULTICAST ADDRESS SETUP:

This command allows the programmer to setup one or more multicast addresses into the 82586. The multicast addresses to be setup are located in the parameter field of the command.

TRANSMIT:

One Transmit command is used to send a single frame. If more than one frame is to be sent, the host CPU can link multiple Transmit commands together. The destination address, Length field and pointer to buffers containing the data field are contained in the parameter field of the Transmit command (AL-LOC = 0).

TDR:

This command performs the Time Domain Reflectometry test on the coaxial cable. The TDR command is used to detect and locate cable faults caused by either short or open circuits on the coaxial cable.

DIAGNOSE:

The Diagnose command puts the 82586 through a self-test procedure and reports on the success or failure of the internal test.

DUMP:

This command causes the 82586 to dump its internal registers into memory. The registers included are those loaded by the Configure and Address Set-Up commands, plus status and other internal working registers.

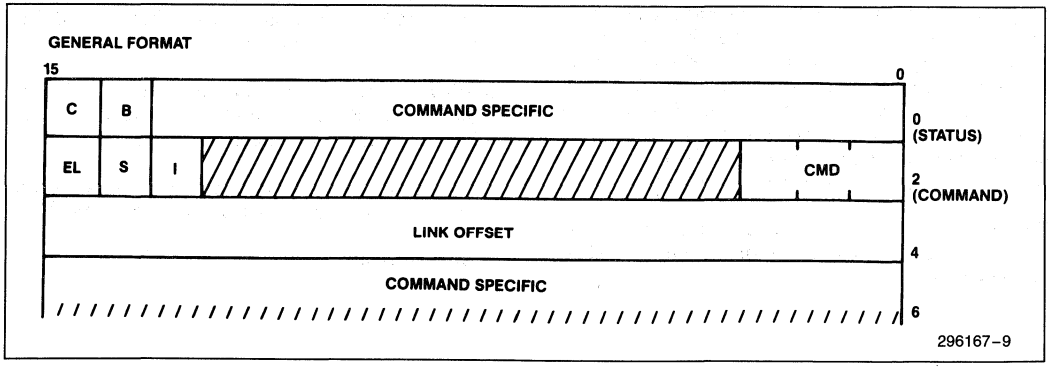


Figure 10. General Action Command

Table 5. 82586 Action Command Summary

- NOP, Code = 0
  Parameters : None
  Action : None
- IA-SETUP, Code = 1
  Parameters : Individual Address
  Action : Setup Individual Address
- CONFIGURE, Code = 2
  Parameters : Byte count and configuration values list
  Action : Configure the 82586
- MC-SETUP, Code = 3
  Parameters : Byte count and Multicast Address list
  Action : Setup Multicast HASH table
- TRANSMIT, Code = 4
  Parameters : Destination Address, Length Field, linked buffer list holding the Data Field
  Action : Transmit frame, and in case of collisions, retransmit
- TDR, Code = 5
  Parameters : None
  Action : Perform Time Domain Reflectometry test and return results
- DUMP, Code = 6
  Parameters : Address of memory location to write results
  Action : Dump set of internal registers to memory
- DIAGNOSE, Code = 7
  Parameters : None
  Action : Perform internal test procedure and returns results

8.1 General Action Command

The format common to all Action Commands and the algorithms for beginning and completing the execution (also common to all the commands) is described below.

NOTE:
System and Transmit Clocks must be applied to the 82586 before Action Commands can be executed.

GENERAL FORMAT

The general format of the Command Block (CB) includes the following fields:

STATUS word (written by the 82586):

C (Bit 15) - is always set at the completion of execution.

B (Bit 14) - is always set at the beginning of execution, and reset at completion of execution concurrently with C bit being set.

NOTE:
The 82586 does not read the B or C bits. If the SCB pointer points to a CB and C = 1, the 82586 will still execute the command.

COMMAND word:

EL (Bit 15) - Indicates that this is the last Command Block in the command list.

S (Bit 14) - Indicates that the CU should suspend after completion of execution of this command.

I (Bit 13) - Indicates that the 82586 should issue an interrupt after completion of executing this command.

CMD (Bits 0–2) - Indicates the specific command.

LINK OFFSET: Address of the next command block in the list.

BEGINNING OF EXECUTION

An Action Command may be started by either the CU-START or CU-RESUME Control Command, or may follow a previous Action Command. However, the actual command start may be delayed by RU activity.

The following sequence is performed by the CU at the beginning of execution of each Action Command:

1) Writes a word whose B bit is set and the remainder of the field cleared to the STATUS word of the next CB. The content of the STATUS word is specified for each Action Command in the description that follows.

2) The next CB becomes the current CB.

3) Reads the COMMAND word of the current CB and saves the following fields for later use: EL, I, S and CMD.

4) Reads the LINK OFFSET of the current CB and makes it the next CB.

5) Performs specific actions according to the Action Command specified in the CMD field.

COMPLETION OF EXECUTION

Command completion time is asynchronous to the beginning of the command. It is determined by the command type, RU activity, CU Control Commands, etc. The CU is always in the ACTIVE state at this time. The decision on how to proceed, depends on the following flags: CU-SUSPEND-REQUEST, CU-ABORT-REQUEST EL, I, S.

The following sequence is performed by the CU at completion of execution of an Action Command:

1) Writes command specific status to the STATUS word of the current CB.

2) If I bit is set, sets request for the CX Interrupt.

3) If EL or CU-ABORT-REQUEST is set, the CU becomes IDLE and generates request for CNA interrupt. If S or CU-SUSPEND-REQUEST is set, CU becomes SUSPENDED. Otherwise, requests beginning of next Action Command.

4) Sets appropriate INTERRUPT-IS flags and clears all Interrupt request flags.

5) Updates STATUS word in SCB.

6) Sets hardware Interrupt signal if any INTERRUPT-IS flag is set. The detailed description of commands is given below.

8.2 NOP

This command results in no action by the 82586, except as performed in normal command processing (section 8.1). It is present to aid in CBL manipulation.

NOP COMMAND FORMAT

NOP command includes the following fields:

STATUS word (**written** by the 82586):

C	(Bit 15)	- Command completed (see section 8.1)
B	(Bit 14)	- Busy executing command
OK	(Bit 13)	- Error free completion

COMMAND word:

EL	(Bit 15)	- End of command list
S	(Bit 14)	- Suspend after completion
I	(Bit 13)	- Interrupt after completion
CMD	(Bits 0–2)	- NOP = 0

LINK OFFSET: Address of next Command Block

DETAILED OPERATION OF NOP COMMAND

CU performs the following sequence:

1) Starts Action Command (see section 8.1).

2) Prepares STATUS word with C=1, B=0, OK=1.

3) Completes Action Command (see section 8.1).

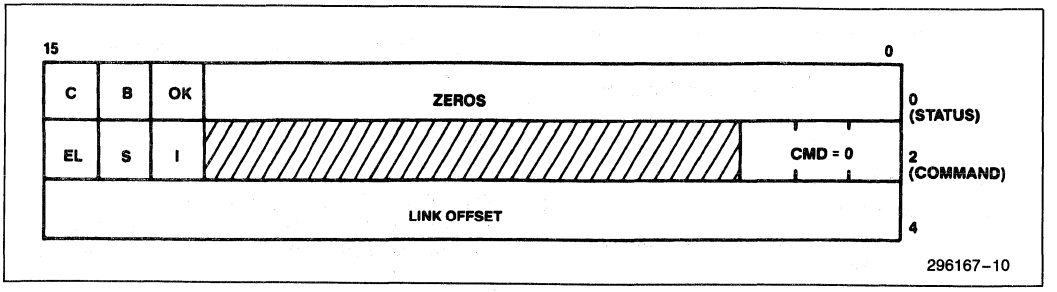


Figure 11. The NOP Command Block

8.3 IA-Setup

This command loads the 82586 with the Individual Address. This address is used by the 82586 for recognition of Destination Address during reception and insertion of Source Address during transmission.

IA-SETUP COMMAND FORMAT

The IA-SETUP command includes the following fields:

STATUS word (**written** by the 82586):

C	(Bit 15)	- Command completed (see section 8.1)
B	(Bit 14)	- Busy executing command
OK	(Bit 13)	- Error free completion
A	(Bit 12)	- Command aborted

COMMAND word:

EL	(Bit 15)	- End of command list
S	(Bit 14)	- Suspend after completion
I	(Bit 13)	- Interrupt after completion
CMD	(Bits 0–2)	- IA-SETUP = 1

LINK OFFSET: Address of next Command Block

INDIVIDUAL ADDRESS: Individual Address parameter

The least significant bit of the Individual Address parameter must be zero for IEEE 802.3. However, no enforcement of 0 is provided by the 82586. Thus, an Individual Address with least significant bit 1, is possible.

The 82586 will read only the number of bytes configured as address length.

DETAILED OPERATION OF THE IA-SETUP COMMAND

The Individual Address is transferred by TX-DMA via TX-FIFO to Transmit-Byte-Machine. See section 13. The CU performs the following sequence:

1) Starts Action Command.
2) Writes the IA-SETUP command byte to TX-FIFO.
3) Initiates TX-DMA with the address of the first byte of INDIVIDUAL ADDRESS and byte count according to configured Address Length.
4) Upon completion of DMA, writes the End of Command byte to TX-FIFO.
5) Waits for the Transmit-Byte-Machine to complete the updating of Individual Address register.
6) If the internal CU-ABORT-REQUEST flag is set, prepares STATUS word with C = 1, B = 0, OK = 0, A = 1; otherwise, prepares STATUS word with C = 1, B = 0, OK = 1, A = 0.
7) Completes the Action Command.

The Transmit-Byte-Machine maintains a 48-bit Individual Address register used for Source Address insertion during transmission and for Destination Address recognition during reception. RESET causes the Individual Address register to be set to all ones.

After the Transmit-Byte-Machine reads an IA-SETUP command from TX-FIFO, it reads the following bytes into the Individual Address register. The expected number of bytes is determined by the Address length configuration parameter. If Address length is less than 6 bytes then only part of the register is used. If fewer bytes than Address length are read from TX-FIFO (in case of abortion) only that number of bytes is updated.

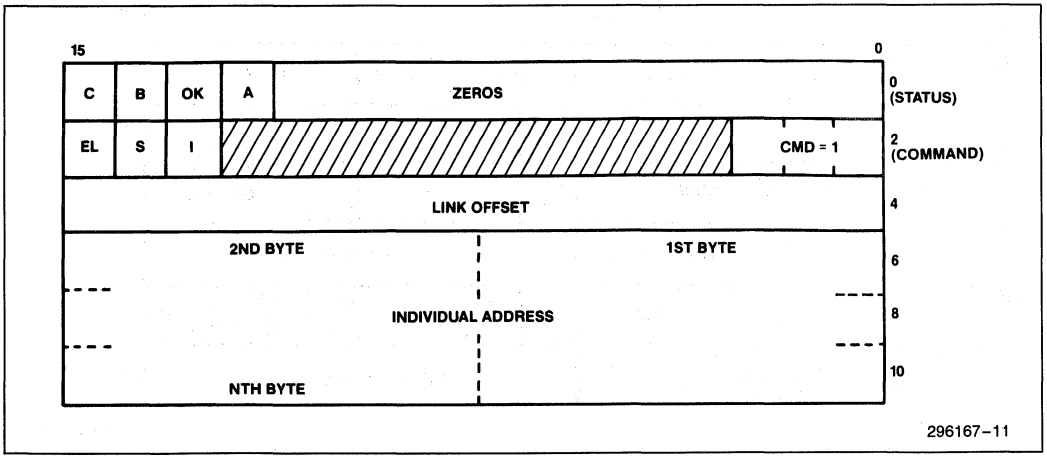


Figure 12. The IA-SETUP Command Block

8.4 Configure

The CONFIGURE command is used to update the 82586 operating parameters. It allows from 4 to 12 parameter bytes to be updated. Refer to section 12 for details on the configuration parameters.

CONFIGURE COMMAND FORMAT

The CONFIGURE command includes the following fields:

STATUS word (**written** by the 82586):

C (Bit 15) - Command completed (see section 8.1)

B (Bit 14) - Busy executing command

OK (Bit 13) - Error free completion

A (Bit 12) - Command aborted

COMMAND word:

EL (Bit 15) - End of command list

S (Bit 14) - Suspend after completion

I (Bit 13) - Interrupt after completion

CMD (Bits 0–2) - Configure = 2

LINK OFFSET: Address of next Command Block

Byte 6-7:

BYTE CNT (Bits 0–3) - Byte Count, Number of bytes including this one, holding the parameters to be configured. A number smaller than 4 is interpreted as 4. A number greater than 12 is interpreted as 12.

FIFO-LIM (Bits 8–11) - Value of TX-FIFO-Threshold

Byte 8-9:

SRDY/$\overline{\text{ARDY}}$ (Bit 6) - 0 - SRDY/ARDY pin operates as ARDY (internal synchronization).

1 - SRDY/ARDY pin operates as SRDY (external synchronization).

SAV-BF (Bit 7) - 0 - Received bad frames are not saved in memory.

1 - Received bad frames are saved in memory.

ADDR-LEN (Bits 8–10) - Number of address bytes. NOTE: 7 is interpreted as 0.

AL-LOC (Bit 11) - 0 - Address and Length Fields separated from data and associated with Transmit Command Block or Receive Frame Descriptor. For transmitted Frame, Source Address is inserted by the 82586.

1 - Address and Length Fields are part of the Transmit/Receive data buffers, including Source Address (which is not inserted by the 82586).

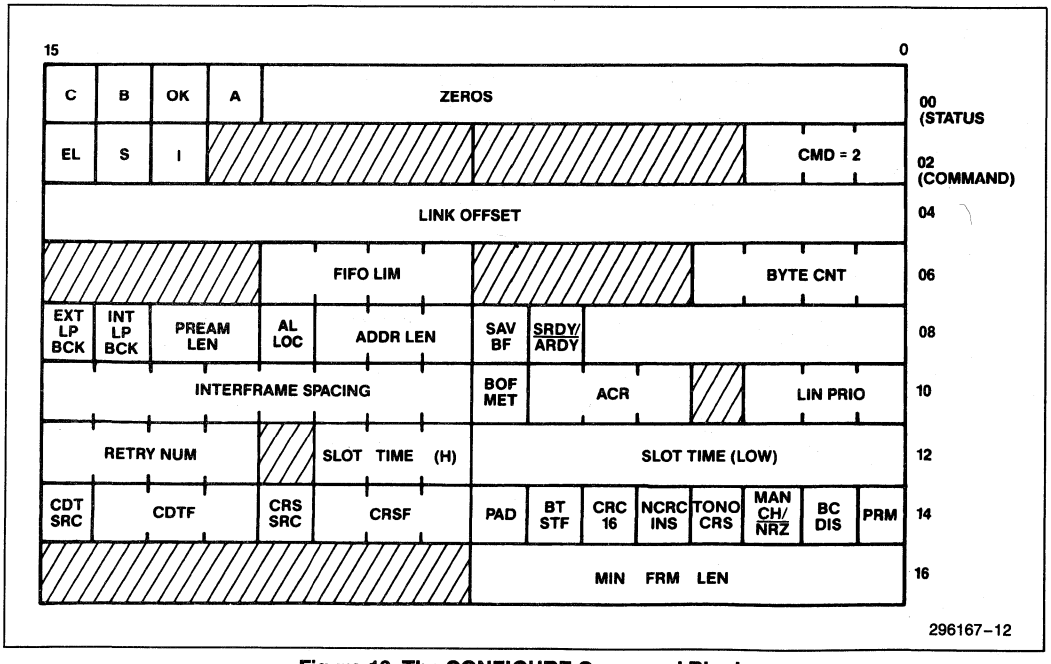


Figure 13. The CONFIGURE Command Block

PREAM-LEN (Bits 12–13) - Preamble Length including Start Frame Delimiter

00 - 2 bytes

01 - 4 bytes

10 - 8 bytes

11 - 16 bytes

INT-LPBCK (Bit 14) - Internal Loopback

EXT-LPBCK (Bit 15) - External Loopback. NOTE: Bits 14 and 15 configured to 1, cause Internal Loopback.

Byte 10–11:

LIN-PRIO (Bits 0–2) - Linear Priority

ACR (Bits 4–6) - Accelerated Contention Resolution

BOF-MET (Bit 7) - Exponential Backoff Method

0 - IEEE 802.3

1 - Alternate method

INTERFRAME-SPACING (Bits 8–15) - Number indicating the Interframe Spacing in TxC period units

Byte 12–13:

SLOT-TIME (L) (Bits 0–7) - Slot Time number, low byte

SLOT-TIME (H) (Bits 8–10) - Slot Time number, high bits

RETRY-NUM (Bits 12–15) - Maximum number of transmission retries on collisions

Byte 14–15:

PRM (Bit 0) - Promiscuous Mode

BC-DIS (Bit 1) - Broadcast Disable

MANCH/$\overline{\text{NRZ}}$ (Bit 2) - Manchester or NRZ encoding/decoding

0 - NRZ

1 - Manchester

TONO-CRS (Bit 3) - Transmit on No Carrier Sense

0 - Cease transmission if CRS goes inactive during frame transmission

1 - Continue transmission even if no Carrier Sense

NCRC-INS (Bit 4) - No CRC Insertion

CRC-16 (Bit 5) - CRC Type

0 - 32 bit Autodin II CRC polynomial

1 - 16 bit CCITT CRC polynomial

BT-STF (Bit 6) Bitstuffing:

0 - End of Carrier mode (IEEE 802.3)

1 - HDLC like Bitstuffing mode

PAD (Bit 7) - Padding

0 - No Padding

1 - Perform padding by transmitting flags for remainder of Slot Time

CRSF (Bits 8–9) - Carrier Sense Filter in bit times

CRS-SRC (Bit 11) Carrier Sense Source

0 - External

1 - Internal

CDTF (Bits 12–14) - Collision Detect Filter in bit times

CDT-SRC (Bit 15) - Collision Detect Source

0 - External

1 - Internal

Byte 16:

MIN-FRM-LEN (Bits 0–7) - Minimum number of bytes in a frame

CONFIGURATION DEFAULTS

The default values of the configuration parameters are compatible with the IEEE 802.3 Standard. RESET configures the 82586 according to the defaults shown in Table 6:

Table 6. 82586 Default Values

Parameter	Value	Units/Meaning
Preamble Length	10	8 Bytes
Address Length	110	6 Bytes
Broadcast Disable	0	Broadcast Reception Enabled
CRC-16/CRC-32	0	Use CRC-32
No CRC Insertion	0	CRC Appended to Frame
Bitstuffing/EOC	0	End of Carrier Framing
Padding	0	No Automatic Padding
Min-Frame-Length	01000000	64 Bytes
Interframe Spacing	01100000	96 Bits
Slot Time	1000000000	512 Bits
Number of Retries	1111	15 Retries
Linear Priority	0	802.3 Algorithm
Accelerated Contention	0	802.3 Algorithm
Exponential Backoff Method	0	802.3 Algorithm
Manchester/NRZ	0	NRZ Encoding/Decoding
Internal CRS	0	Use External CRS
CRS Filter	0	0 Bits
Internal CDT	0	Use External CDT
CDT Filter	0	0 Bits
Transmit on No CRS	0	Disabled
FIFO Threshold	8	Bytes
$\overline{\text{SRDY}}$/ARBY	0	Pin Operates as ARDY
Save Bad Frames	0	Discard Bad Frames
AL-LOC	0	Address/Length Located in FD
Internal Loopbace	0	Internal Loopback Disabled
External Loopback	0	External Loopback Disabled
Promiscuous Mode	0	Disabled

DETAILED OPERATION OF THE CONFIGURE COMMAND

Some parameters are kept by the CU, the remainder are transferred by TX-DMA to the Transmit-Byte-Machine via TX-FIFO. The CU performs the following sequence.

1) Starts the Action Command.
2) Reads bytes 6, 7, 8, and 9, and saves the following: BYTE-CNT, FIFO-LIM, SRDY/$\overline{\text{ARDY}}$, SAV-BF, ADDR-LEN, AL-LOC.
3) Writes the Configure byte to TX-FIFO.
4) Initiates TX-DMA to address of byte 6 and byte count specified by BYTE-CNT.
5) Upon completion of DMA writes the End of Command byte to TX-FIFO.
6) Waits for the Transmit-Byte-Machine to complete updating configuration registers.
7) If the internal CU-ABORT-REQUEST flag is set, prepares STATUS word with C = 1, B = 0, OK = 0, A = 1; otherwise, prepares STATUS word with C = 1, B = 0, OK = 1, A = 0.
8) Completes the Action Command.

The Transmit-Byte-Machine maintains 10 bytes of Configuration registers for determining the 82586's 'personality.' RESET causes Configuration registers to be set to values specified in Table 6 (compatible to IEEE 802.3).

After the Transmit-Byte-Machine reads the CONFIGURE command from TX-FIFO it reads and ignores 2 bytes, reads and updates Configuration registers with the next 10 bytes, and reads and ignores the remaining bytes. If less than 12 bytes are read when End of Command is encountered, only part of the Configuration registers are updated. The Transmit-Byte-Machine notifies CU after completion.

8.5 MC-Setup

This command sets up the 82586 with a set of Multicast Addresses. Subsequently, incoming frames with Destination Addresses from this set are accepted.

MC-SETUP COMMAND FORMAT

The MC-SETUP command includes the following fields:

STATUS word (**written** by the 82586):

C	(Bit 15)	- Command completed (see section 8.1)
B.	(Bit 14)	- Busy executing command
OK	(Bit 13)	- Error free completion
A	(Bit 12)	- Command aborted

COMMAND word:

EL	(Bit 15)	- End of command list
S	(Bit 14)	- Suspend after completion
I	(Bit 13)	- Interrupt after completion
CMD	(Bits 0–2)	- MC-SETUP = 3

LINK OFFSET: Address of next Command Block

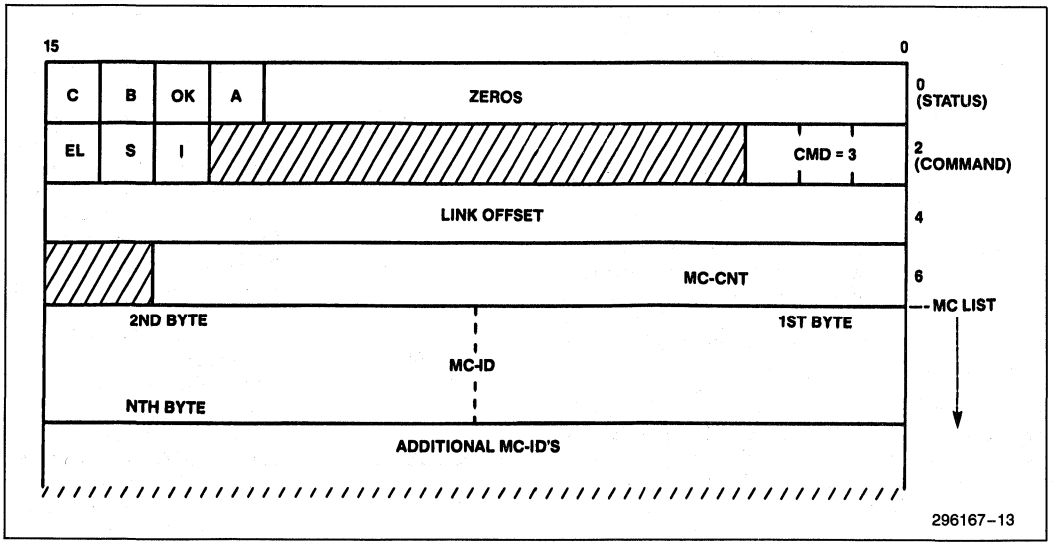


Figure 14. The MC-SETUP Command Block

MC-CNT: A 14-bit field indicating the number of bytes in the MC-LIST field. MC-CNT is truncated to the nearest multiple of Address Length (in bytes). Issuing a MC-SETUP command with MC-CNT = 0 disables reception of any incoming frame with a Multicast Address. The 14 bit field for MC addresses accommodates a maximum of 2730 Multicast Addresses, assuming a 6 byte address length.

MC-LIST: A list of Multicast Addresses to be accepted by the 82586. Note that the most significant byte of an address is followed immediately by the least significant byte of the next address. Note also that the least significant bit of each Multicast Address in the set must be a one.

DETAILED OPERATION OF MC-SETUP COMMAND

TX-DMA transfers the list of Multicast Addresses from memory to Transmit-Byte-Machine via TX-FIFO. See section 13. CU performs the following sequence:

1) Starts Action Command.
2) Reads MC-CNT and saves it internally.
3) Writes a MC-SETUP command byte to TX-FIFO.
4) Initiates TX-DMA with the MC-LIST address and byte count according to MC-CNT.
5) Upon completion of DMA, writes End of Command bit to TX-FIFO.
6) Waits for Transmit-Byte-Machine to complete HASH table update.
7) If the internal CU-ABORT-REQUEST flag is set, prepares STATUS word with C = 1, B = 0, OK = 0, A = 1; otherwise, prepares STATUS word with C = 1, B = 0, OK = 1, A = 0.
8) Completes the Action Command.

The Transmit-Byte-Machine maintains a 64-bit HASH table used for checking Multicast Addresses during reception.

An incoming frame is accepted if it has a Destination Address whose least significant bit is a one, and after hashing points to a bit in the HASH table whose value is one. The hash function is selecting bits 2 to 7 of the Transmit CRC register. RESET causes the HASH table to become all zeros.

After the Transmit-Byte-Machine reads a MC-SETUP command from TX-FIFO, it clears the HASH table and reads the bytes in groups whose length is determined by the ADDRESS length. Each group is hashed using CRC logic and the bit in the HASH table to which bits 2–7 of the CRC register point is set to one. A group that is not complete has no effect on the HASH table. Transmit-Byte-Machine notifies CU after completion.

8.6 Transmit

The TRANSMIT command causes transmission and if necessary retransmission of a frame.

TRANSMIT COMMAND BLOCK FORMAT

TRANSMIT CB includes the following fields:

STATUS word (**written** by the 82586):

C (Bit 15) - Command completed (see section 8.1)

B (Bit 14) - Busy executing command

OK (Bit 13) - Error free completion

A (Bit 12) - Command aborted

S10 (Bit 10) - No Carrier Sense signal during transmission (between beginning of Destination Address and end of Frame Check Sequence). On the B-step this bit will always be zero when the 82586 is configured with TONO-CRS = 1. All subsequent steppings S10 will be set if the Carrier Sense signal is lost, regardless of the TONO-CRS Configure bit.

S9 (Bit 9) - Transmission unsuccessful (stopped) due to loss of Clear-to-Send signal.

S8 (Bit 8) - Transmission unsuccessful (stopped) due to DMA underrun, (i.e. data not supplied from the system for transmission).

S7 (Bit 7) - Transmission had to Defer to traffic on the link.

S6 (Bit 6) - SQE TEST. Indicates that after the previous transmission and during the Interframe Spacing period, the SQE TEST signal was detected on the Collision Detect pin. This bit is meaningful if S5 and MAX-COLL fields are both zero, i.e. no collisions occurred during transmission.

S5 (Bit 5) - Transmission attempt stopped due to number of collisions exceeding the maximum number of retries.

MAX-COLL (Bits 3-0) - Number of Collisions experienced by this frame. S5 = 1 and MAX-COLL = 0 indicates that there were 16 collisions.

COMMAND word:

EL	(Bit 15)	- End of command list
S	(Bit 14)	- Suspend after completion
I	(Bit 13)	- Interrupt after completion
CMD	(Bits 0–2)	- TRANSMIT = 4

LINK OFFSET: Address of next Command Block

TBD OFFSET: Address of the first Transmit Buffer Descriptor. TBD-OFFSET = 0FFFFH indicates that there is no Data field.

DESTINATION ADDRESS: Destination Address of the frame.

LENGTH FIELD: Length Field of the frame.

TRANSMIT BUFFER DESCRIPTOR FORMAT

STATUS word:

EOF - Indicates that this is the Buffer Descriptor of the last buffer of this frame's Data Field.

ACT-COUNT (Bits 0–13) - Actual number of data bytes in the buffer (can be even or odd).

NEXT BD OFFSET: Points to next Buffer Descriptor on the list. When the EOF bit is set, this field is meaningless.

BUFFER ADDRESS: 24-bit absolute address of buffer.

DETAILED OPERATION OF TRANSMIT COMMAND

Execution of TRANSMIT Command causes transmission of a frame. If the frame experiences collisions, the CU retransmits the frame up to a configurable maximum number of times.

To transmit a frame, the user must setup the following:

- Command code.
- Destination Address and Length Field in TRANSMIT CB.
- TBD OFFSET should point to the first Buffer Descriptor of the list of buffers that holds the Data Field. If the frame does not include Data, TBD OFFSET should be set to 0FFFFH.
- For each buffer, BUFFER ADDRESS, ACTUAL COUNT and NEXT BD OFFSET should be setup. The EOF flag indicates the last buffer in the list representing the Data Field.

Unlike other Action Commands having parameters in one block in memory, the TRANSMIT Command may have parts of the parameters scattered in a linked list of buffers. The CU prefetches the buffers in the list on the fly. If the AL-Location configuration is zero, the Destination Address and Length Field are in the TRANSMIT CB and are treated as the first buffer. The Information Field is in the list of buffers. If the AL-Location Bit is one, the whole frame is in the buffers. See Figure 17.

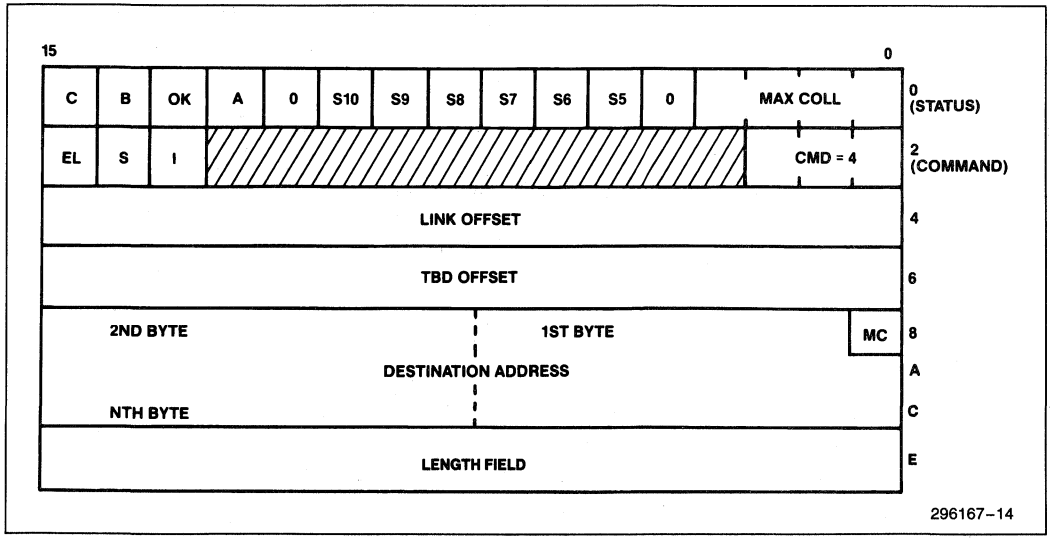


Figure 15. The Transmit Command Block

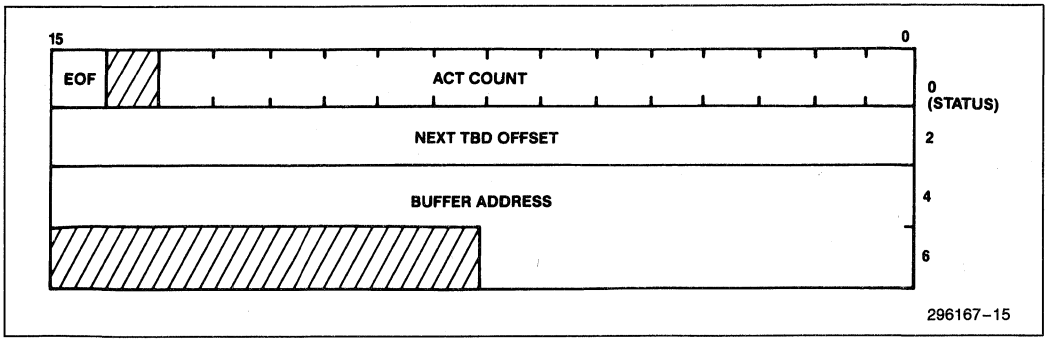


Figure 16. The Transmit Buffer Descriptor

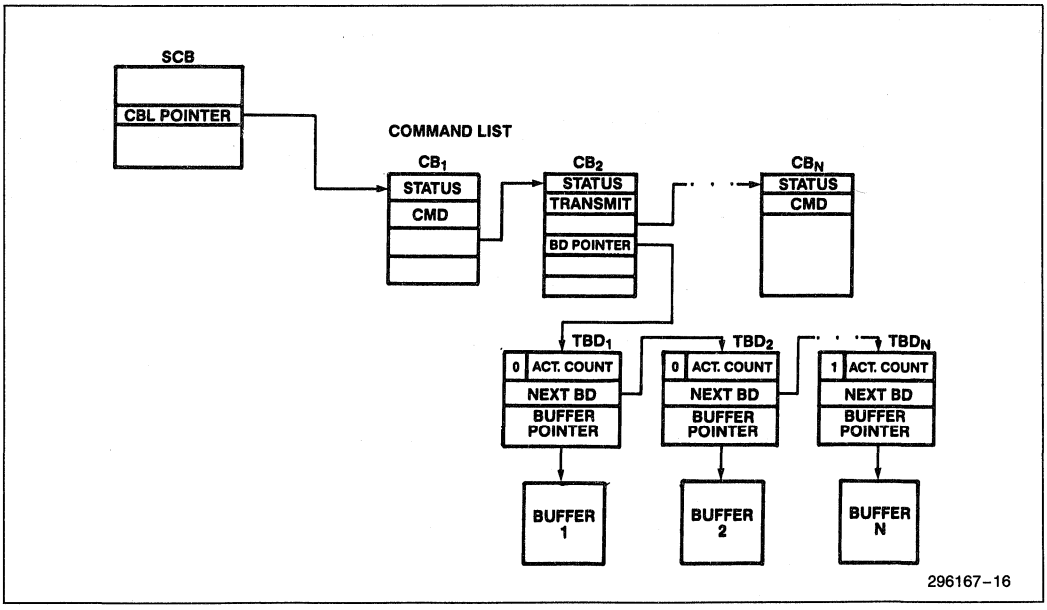


Figure 17. The Transmit Command Data Structure

While the CU is prefetching the address and byte count of one buffer, TX-DMA is transferring the previous buffer to the Transmit-Byte Machine via TX-FIFO. Completion of buffer transfer by TX-DMA, triggers the CU to initiate TX-DMA for the next buffer (if already prefetched) and to start the prefetch of the next buffer. The buffer prefetch/transfer cycle is terminated when a buffer, whose EOF bit is set, is transferred to TX-FIFO.

CU performs the following sequence during transmission:

1) Starts the Action Command.
2) Writes the TRANSMIT Command bit to TX-FIFO.
3) Reads BD OFFSET and saves it as the Look Ahead BD Address.
4) If the ADDRESS/LENGTH LOCATION configuration bit is zero, the 82586 does the following:

   If the Look Ahead BD Address is not equal to 0FFFFH, then goes to the buffer prefetch/transfer cycle. Initiates TX-DMA to the address of the first byte of the Destination Address field in the CB and to the byte count of the Address length bit plus 2. If the Look Ahead BD Address is equal to 0FFFFH, then after completing DMA of the Destination Address and Length Field writes End of Command byte to TX-FIFO.
5) If the ADDRESS/LENGTH LOCATION configuration bit is one, then goes to the buffer prefetch/transfer cycle and forces one dummy DMA completion.
6) Waits for completion of the TRANSMIT Command. This includes transfer of the whole frame to the Channel Interface Module and transmission of the frame by the latter. The command can be completed either with a collision (and not exceeding maximum collision) or without a collision.
7) If execution completed without a collision and without an error (regardless of CU-ABORT-REQUEST), then prepares STATUS word with C = 1, B = 0, OK = 1, A = 0, S6, S7 and NUM-COLL updated, and remaining bits zero.
8) If execution completed with CU-ABORT-REQUEST (and a collision or error) prepare STATUS word with C = 1, B = 0, OK = 0, A = 1 and NUM-COLL = 1.
9) If execution completed with a Collision (and not exceeding maximum collision), regardless of errors and without CU-ABORT-REQUEST, writes Retransmit command byte to TX-FIFO and returns to Step 3 of this sequence. This causes retransmission of the frame.
10) If the execution completed with an error (no collision or CU-ABORT-REQUEST), prepares STATUS word with C = 1, B = 0, OK = 0, A = 0 and the remaining bits updated.
11) Completes the Action Command.

Note that transmission is terminated by the CU performing one of the following steps; 7, 8, 9, or 10.

The Chart below summarizes which sequence step listed above the CU performs:

Collision	Error	Abort-Request	Step	Remarks
no	no	don't care	7	normal
no	yes	no	10	error
no	yes	yes	8	abort
yes	don't care	no	9	retransmit
yes	don't care	yes	8	abort

BUFFER PREFETCH/TRANSFER CYCLE

This cycle is called by the sequence-performing TRANSMIT command execution. Collision detection (not exceeding the maximum collision) anytime during this sequence, causes CU to Read BD OFFSET and save it as the Look Ahead BD Address, write retransmit command byte to TX-FIFO and go to Step 4 of the TRANSMIT command sequence.

The buffer prefetch/transfer cycle is as follows:

1) Store Look Ahead BD address in Next-BD-Address.
2) Read and save 14-bit ACT COUNT field and EOF flag.
3) Read NEXT BD OFFSET and save it in Look Ahead BD Address.
4) Read and save the 24-bit BUFFER ADDRESS field.
5) Wait for completion of TX-DMA (first time initiated in step 4 of transmit sequence).
6) Initiate TX-DMA with address according to BUFFER ADDRESS and byte count according to ACT COUNT.
7) If EOF bit is not set, repeat the cycle.
8) After DMA completion, write the End of Command byte to TX-FIFO.

FRAMING OPERATION

The Transmit-Byte-Machine maintains the following registers for construction of frames: Preamble pattern, SFD Field (pattern determined by End-of-Carrier/Bit-stuffing configuration parameters), Source Address, CRC generator, Jam patterns (all ones).

After the Transmit-Byte-Machine reads the TRANSMIT command from TX-FIFO, a frame is constructed and transferred to the Transmit-Bit-Machine (see section 13) for bit transmission. The Transmit-Byte-Machine performs the following sequence:

1) Transfer Preamble bytes according to Preamble length configuration parameter minus one.
2) Transfer the SFD Field.
3) Start CRC calculation.
4) Read and transfer the Destination Address bytes from TX-FIFO (number determined by Address length).
5) If AL-Location configuration parameter is zero, transfer Individual Address as Source Address. Otherwise, read and transfer Source Address from TX-FIFO. If AL-Location = 1 and there are less than Address-Length bytes in TX-FIFO, a DMA underrun is forced.
6) Read and transfer all remaining bytes from TX-FIFO. These are the Length and Information fields.
7) Transfer CRC (calculated according to CRC 16/32 configuration parameter).
8) If the 82586 is configured to Bitstuffing, transfer one more flag.
9) If the 82586 is configured to Bitstuffing and to padding, transfer flag bytes to cause the number of transferred bytes (excluding Preamble and SFD Field) to exceed the configurable Slot Time divided by 8.

If a collision, underrun, or lost Carrier Sense occurred during transmission, the Transmit-Byte-Machine completes the transfer of the Preamble and transfers 4 bytes of the Jam pattern. If there is a collision, the retry counter will be incremented.

Jamming will not start before completing Preamble transmission.

If a collision is detected during transmission of the last 11 bits in the frame it will not result in jamming. If the collision is detected during transmission of the last bit or later, the collision will not be reported and retransmission does not take place. This may happen for an **invalid frame** which is shorter in length than the slot time.

Note that a DMA underrun cannot logically occur during the preamble because the serial subsystem generates its own preamble. Also, the 82586 is insensitive to carrier sense during the preamble.

After completing transmission, or collision, the Transmit-Byte-Machine passes bits 0–10 of the STATUS word to the CU.

8.7 TDR (Time Domain Reflectometer)

This command performs a Time Domain Reflectometer test on the serial link. By performing the command, the user is able to identify short or opens and their location. Along with transmission of 'All Ones,' the 82586 triggers an internal timer. The timer measures the time elapsed from transmission start until 'echo' is obtained. 'Echo' is indicated by Collision Detect going active or Carrier Sense signal drop.

TDR COMMAND FORMAT

TDR command includes the following fields:

STATUS word (**written** by the 82586):

C	(Bit 15)	- Command completed (see section 8.1)
B	(Bit 14)	- Busy executing command
OK	(Bit 13)	- Error free completion

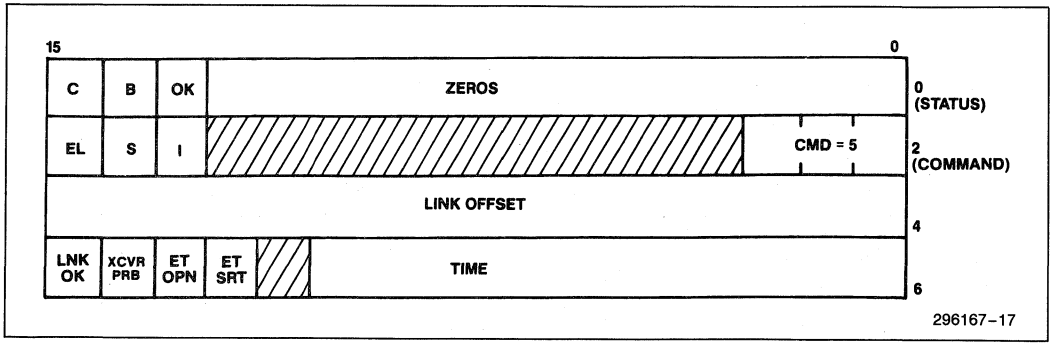


Figure 18. The TDR Command Block

COMMAND word:

EL	(Bit 15)	- End of command list
S	(Bit 14)	- Suspend after completion
I	(Bit 13)	- Interrupt after completion
CMD	(Bits 0–2)	- TDR = 5

LINK OFFSET: Address of next Command Block

RESULT word:

LNK-OK	(Bit 15)	- No link problem identified
XCVR-PRB	(Bit 14)	- Transceiver Cable Problem identified (valid only in the case of a Transceiver that should return Carrier Sense during transmission).
ET-OPN	(Bit 13)	- Open on the link identified.
ET-SRT	(Bit 12)	- Short on the link identified (valid only in the case of a Transceiver that should return Carrier Sense during transmission).
TIME	(Bits 0–10)	- Specifying the distance to a problem on the link (if one exists) in transmit clock cycles.

DETAILED OPERATION OF TDR COMMAND

TDR command triggers the Time Domain Reflectometer test to be performed by the Transmit-Byte-Machine, see section 13.

The CU performs the following sequence:

1) Starts the Action Command.
2) Writes the TDR Command byte to TX-FIFO.
3) Waits for the Transmit-Byte-Machine to complete TDR test.
4) Writes results to RESULT word.
5) Prepares STATUS word with C = 1, B = 0, OK = 1.
6) Completes the Action Command.

After the Transmit-Byte-Machine reads a TDR command from TX-FIFO, it performs a Time Doman Reflectometer test by transmitting a TDR frame (2048 ones) and monitoring Carrier Sense and Collision Detect signals. When a TDR frame is transmitted, an internal 12-bit counter starts counting.

There are four possible results:

1) The Carrier Sense signal does not go active before the counter expires. For a Transceiver that should return Carrier Sense during transmission, this means that there is a problem on the cable between the 82856 and the Transceiver. For a Transceiver that should not return Carrier Sense during transmission, this is normal.
2) The Carrier Sense signal goes active and then inactive before the counter expires. For a Transceiver that should return Carrier Sense during transmission, this means that there is a short on the link.
3) The Collision Detect signal goes active before the counter expires. This means that the link is not properly terminated (an open).
4) The Carrier Sense signal goes active but does not go inactive and Collision Detect does not go active before the counter expires. This is the normal case and indicates that there is no problem on the link.

The distance to the cable failure can be calculated as follows:

Distance = TIME × (Vs/(2×Fs))
where:

Vs - wave propagation speed on the link (M/s)

Fs - serial clock frequency (Hz)

Accuracy is plus/minus Vs/(2×Fs)

Upon completion, the Transmit-Byte-Machine passes the RESULT word to the CU.

Note that the TDR frame does not contain the SFD field.

8.8 Dump

This command causes the contents of over a hundred bytes of internal registers to be placed in memory. It is supplied as a self diagnostic tool, as well as to supply registers of interest to the user.

DUMP command includes the following fields:

STATUS word (**written** by the 82586):

C	(Bit 15)	- Command completed (see section 8.1)
B	(Bit 14)	- Busy executing command
OK	(Bit 13)	- Error free completion

COMMAND word:

EL	(Bit 15)	- End of command list
S	(Bit 14)	- Suspend after completion
I	(Bit 13)	- Interrupt after completion
CMD	(Bits 0–2)	- DUMP = 6

LINK OFFSET: Address of the next Command Block

BUFFER OFFSET: This word specifies the offset portion of the memory address which points to the top of the buffer allocated for the dumped registers contents. The length of the buffer is 170 bytes.

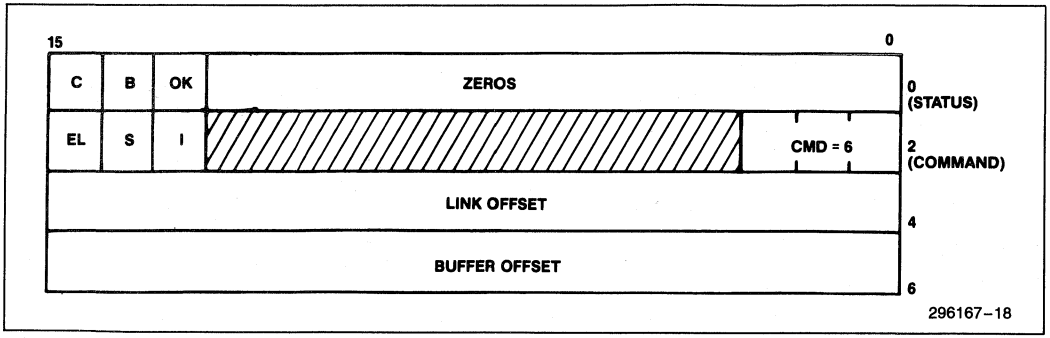


Figure 19. The DUMP Command Block

DUMP AREA FORMAT

Figure 20 shows the format of the DUMP area. The fields are as follows:

Bytes 00H to 0AH: These bytes correspond to the 82586 CONFIGURE command field.

Bytes 0CH to 11H: The Individual Address Register content. IAR0 is the Individual Address least significant byte.

Bytes 12H to 13H: Status word of last command block (only bits 0–13).

Bytes 14H to 17H: Content of the Transmit CRC generator. TXCRCR0 is the least significant byte. The contents are dependent on the activity before the DUMP command:

After RESET - 'All Ones.'

After successful transmission - 'All Zeros.'

After MC-SETUP command - Generated CRC value of the last MC address, on MC-LIST.

After unsuccessful transmission, depends on where it stopped.

NOTE:
For 16-bit CRC only TXCRCR0 and TXCRCR1 are valid.

Bytes 18H to 1BH: Contents of Receive CRC Checker. TXCRCR0 is the least significant byte.

The contents are dependent on the activity performed before the DUMP command:

After RESET - 'All Ones.'

After good frame reception -

1) For CRC-CCITT - 0D1F0H.

2) For CRC-Autodin-II - C704DD7B.

After Bad Frame reception - corresponds to the received information.

After reception attempt, i.e. unsuccessful check for address match, corresponds to the CRC performed on the frame address.

NOTE:
Any frame on the serial link modifies this register contents.

Bytes 1CH to 21H: Temporary Registers.

Bytes 22H to 23H: Receive Status Register. Bits 6, 7, 8, 10, 11 and 13 assume the same meaning as corresponding bits in the Receive Frame Descriptor Status field.

Bytes 24H to 2BH: HASH TABLE.

Bytes 2CH to 2DH: Status bits of the last time TDR command that was performed.

NXT-RB-SIZE: Let N be the last buffer of the last received frame, then NXT-RB-SIZE is the number of available bytes in the N + 1 buffer.

EL - The EL bit of the Receive Buffer Descriptor.

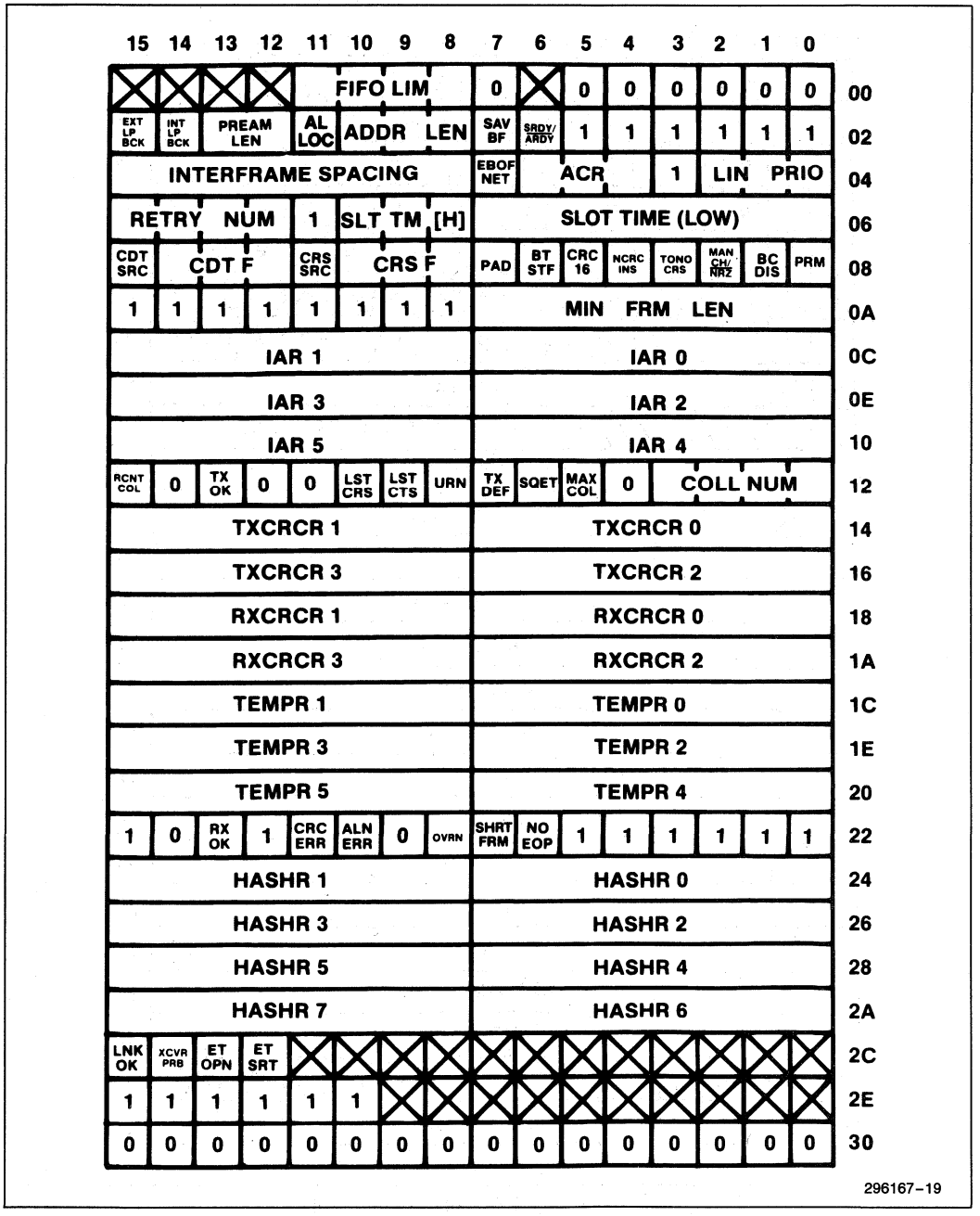
NXT-RB-ADR: Let N be the last Receive Buffer used, then NXT-RB-ADR is the BUFFER-ADDRESS field in the N + 1 Receive-Buffer Descriptor, i.e. the pointer to the N + 1 Receive Buffer.

CUR-RB-SIZE: The number of bytes in the last buffer of the last received frame.

EL - The EL bit of the last buffer in the last received frame.

LA-RBD-ADR: Look Ahead Buffer Descriptor, i.e. the pointer to N + 2 Receive Buffer Descriptor.

NXT-RBD-ADR: Next Receive Buffer Descriptor Address. Similar to LA-RBD-ADR but points to N + 1 Receive Buffer Descriptor.



296167-19

Figure 20. The DUMP Area

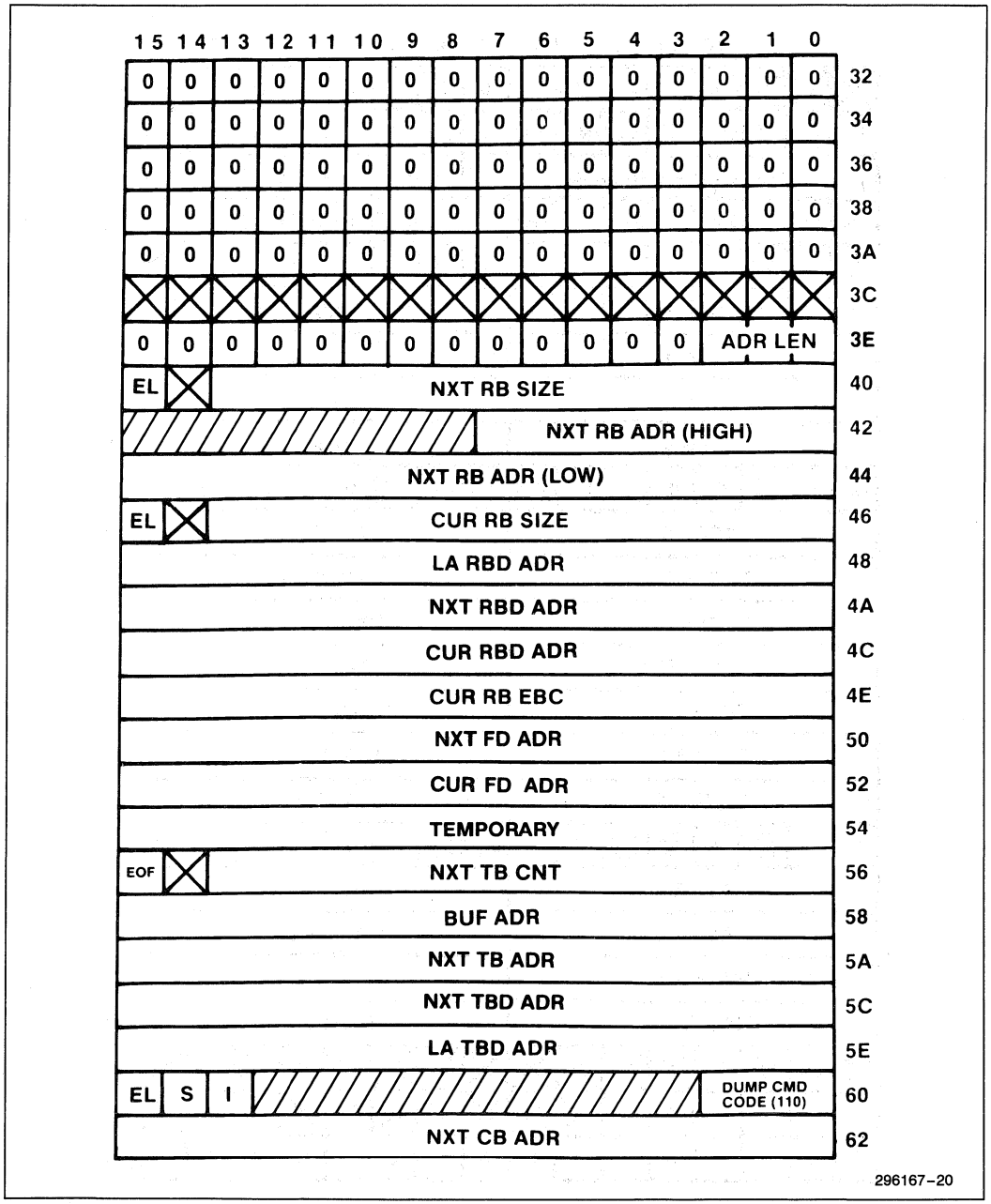


Figure 20. The DUMP Area (Continued)

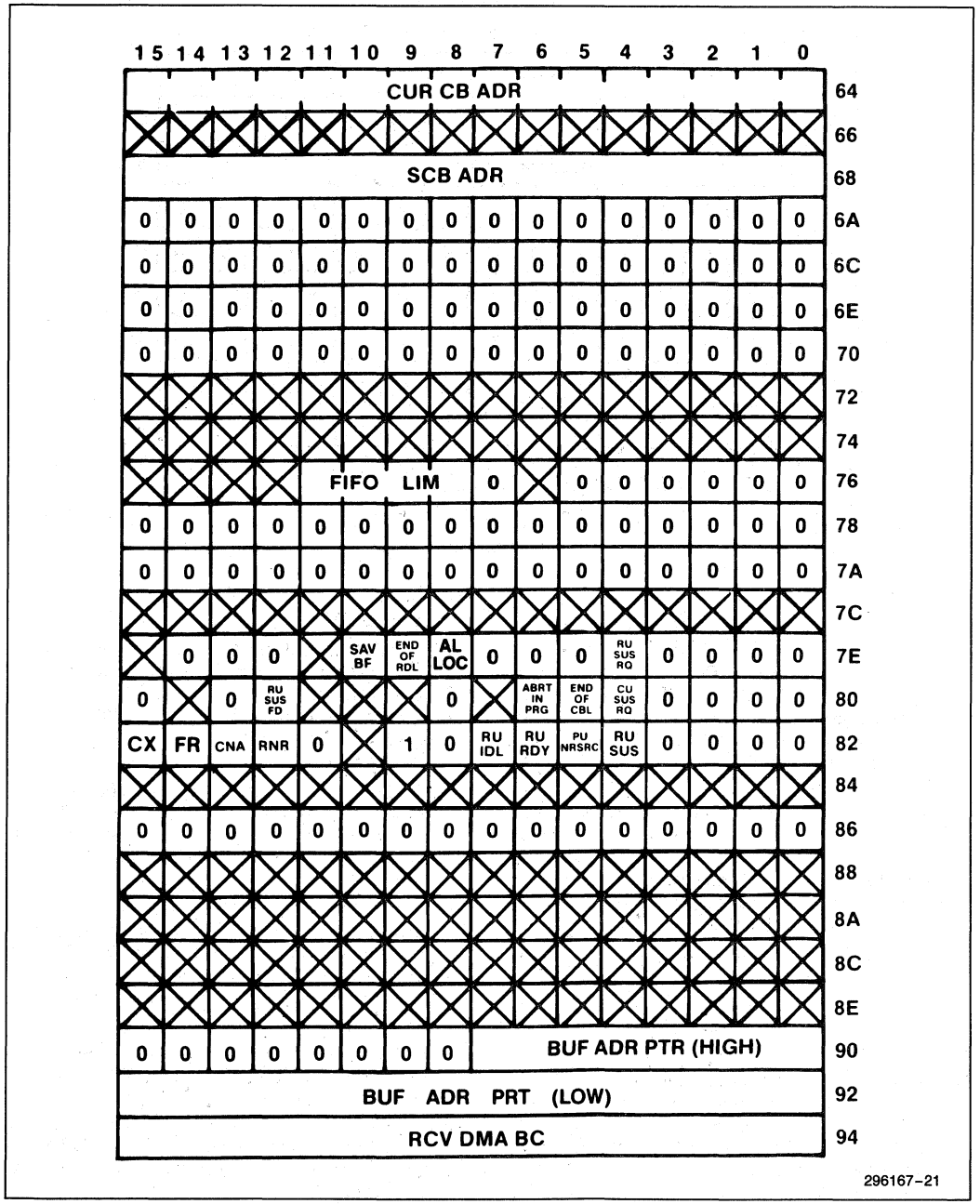


Figure 20. The DUMP Area (Continued)

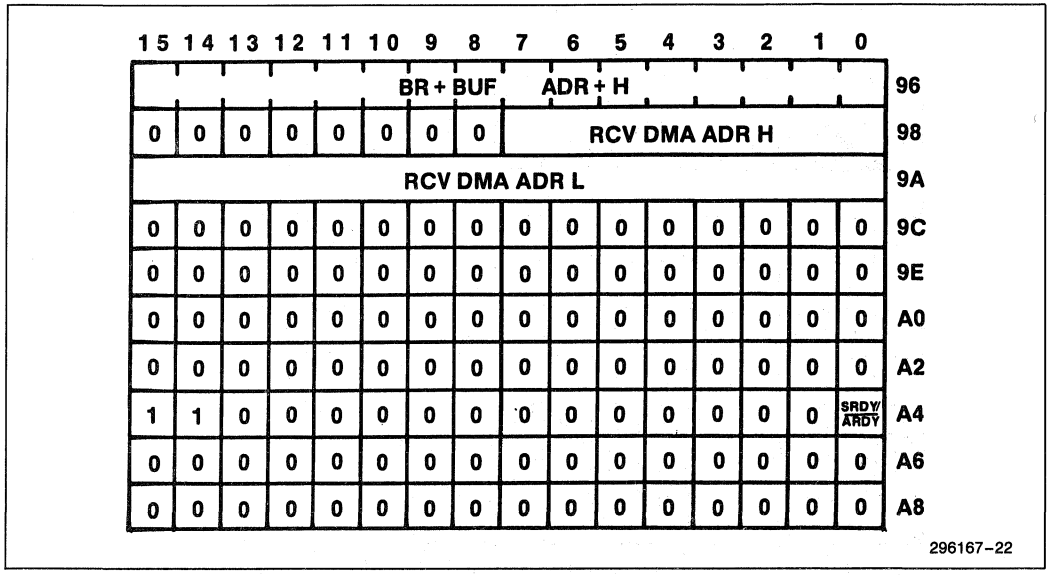


Figure 20. The DUMP Area (Continued)

CUR-RBD-ADR: Current Receive Buffer Descriptor Address. Similar to LA-RBD-ADR, but points to Nth Receive Buffer Descriptor.

CUR-RB-EBC: Current Receive Buffer Empty Byte Count. Let N be the currently used Receive Buffer. Then CUR-RB-EBC indicates the Empty part of the buffer, i.e. the ACT-COUNT of buffer N is given by the difference between its SIZE and the CUR-RB-EBC.

NXT-FD-ADR: Next Frame Descriptor Address. Define N as the last Receive Frame Descriptor with bits C = 1 and B = 0, then NXT-FD-ADR is the address of N+2 Receive Frame Descriptor (with B = C = 0) and is equal to the LINK-ADDRESS field in N+1 Receive Frame Descriptor.

CUR-FD-ADR: Current Frame Descriptor Address. Similar to next NXT-FD-ADR but refers to N1 Receive Frame Descriptor (with B = 1, C = 0).

Bytes 54H to 55H: Temporary register.

NXT-TB-CNT: Next Transmit Buffer Count. Let N be the last transmitted buffer of the TRANSMIT command executed recently, the NXT-TB-CNT is the ACT-COUNT field in the Nth Transmit Buffer Descriptor.

EOF - Corresponds to the EOF bit of the Nth Transmit Buffer Descriptor. EOF = 1 indicates that the last buffer accessed by the 82586 during Transmit was the last Transmit Buffer in the data buffer chain associated with the Transmit Command.

BUF-ADR: Buffer Address. The BUF-PTR field in the DUMP-STATUS Command Block.

NXT-TB-AD-L: Next Transmit Buffer Address Low. Let N be the last Transmit Buffer in the transmit buffer chain of the TRANSMIT Command performed recently, then NXT-TB-AD-L are the two least significant bytes of the Nth buffer address.

NXT-TBD-ADR: Next Transmit Buffer Descriptor Address. Similar in function to LA-TBD-ADR but related to Transmit Buffer Descriptor N-1. Actually, it is the address of Transmit Buffer Descriptor N.

LA-TBD-ADR: Look Ahead Transmit Buffer Descriptor Address. Let N be the last Transmit Buffer in the transmit buffer chain of the TRANSMIT Command performed recently, then LA-TBD-ADR is the NEXT-BD-ADDRESS field of the Nth Buffer Descriptor.

Bytes 60H, 61H: This is a copy of the 2nd word in the DUMP-STATUS command presently executing.

NXT-CB-ADR: Next Command Block Address. The LINK-ADDRESS field in the DUMP Command Block presently executing. Points to the next command.

CUR-CB-ADR: Current Command Block Address. The address of the DUMP Command Block currently executing.

SCB-ADR: Offset of the System Control Block (SCB).

Bytes 7EH, 7FH:

RU-SUS-RQ (Bit 4) - Receive Unit Suspend Request.

Bytes 80H, 81H:

CU-SUS-RQ (Bit 4) - Command Unit Suspend Request

END-OF-CBL (Bit 5) - End of Command Block List. If '1' indicates that DUMP-STATUS is the last command in the command chain.

ABRT-IN-PROG (Bit 6) - Command Unit Abort Request.

RU-SUS-FD (Bit 12) - Receive Unit Suspend Frame Descriptor Bit. Assume N is the Receive Frame Descriptor used recently, then RU-SUS-FD is equivalent to the S bit of N + 1 Receive Frame Descriptor.

Bytes 82H, 83H:

RU-SUS (Bit 4) - Receive Unit in SUSPENDED state.

RU-NRSRC (Bit 5) - Receive Unit in NO RESOURCES state.

RU-RDY (Bit 6) - Receive Unit in READY state.

RU-IDL (Bit 7) - Receive Unit in IDLE state.

RNR (Bit 12) - RNR Interrupt In Service bit.

CNA (Bit 13) - CNA Interrupt In Service bit.

FR (Bit 14) - FR Interrupt In Service bit.

CX (Bit 15) - CX Interrupt In Service bit.

Bytes 90H to 93H:

BUF-ADR-PTR - Buffer pointer is the absolute address of the bytes following the DUMP Command block.

Bytes 94H to 95H:

RCV-DMA-BC - Receive DMA Byte Count. This field contains number of bytes to be transferred during the next Receive DMA operation. The value depends on AL-LOCation configuration bit.

1) If AL-LOCation = 0 then RCV-DMA-BC = (2 times ADDR-LEN plus 2) if the next Receive Frame Descriptor has already been fetched.

2) If AL-LOCation = 1 then it contains the size of the next Receive Buffer.

BR + BUF-PTR + 96H - Sum of Base Address plus BUF-PTR field and 96H.

RCV-DMA-ADR - Receive DMA absolute Address. This is the next RCV-DMA start address. The value depends on AL-LOCation configuration bit.

1) If AL-LOCation = 0, then RCV-DMA-ADR is the Destination Address field located in the next Receive Frame Descriptor.

2) If AL-LOCation = 1, then RCV-DMA-ADR is the next Receive Data Buffer Address.

The following notes apply to all the information of the dump area:

1) The listed pointers are meaningful only if the 82586 has completed the prefetch.

2) All Transmit Pointers refer to the last executed TRANSMIT Command.

3) The Receive Pointers are meaningful only if they exist, namely a frame was received into the existing memory structure.

4) The following nomenclature has been used in the DUMP table:
0 - The 82586 writes zero in this location.
1 - The 82586 writes one in this location.
X - The 82586 writes zero or one in this location.
///- The 82586 copies this location from the corresponding position in the memory structure.

DETAILED OPERATION OF DUMP COMMAND

Configuration parameters and contents of other registers are transferred from the Channel Interfaced Module via RCV-FIFO by Receive Unit to memory.

The Command Unit performs the following sequence:

1) Starts Action Command.

2) Writes DUMP command byte to TX-FIFO.

3) Waits for completion of DUMP.

4) Prepares STATUS word with C= 1, B = 0, OK = 1.

5) Completes Action Command.

The Receive Unit performs the following sequence.

1) Writes the word with the FIFO-Threshold configuration field to memory.

2) For all data written by the Channel Interface Module to RCV-FIFO: Reads 2 bytes from FIFO, assembles them into a word and writes it to the next location in memory buffer.

3) Writes 64 CU and RU registers to the remaining space in memory buffer.

4) Notifies CU that DUMP completed.

The Channel Interface Module passes 46 bytes of internal registers to RU and notifies CU on completion.

NOTE:
This is the only Action Command involving the RU.

8.9 Diagnose

The DIAGNOSE Command triggers an internal self test procedure of backoff related register and counters.

DIAGNOSE COMMAND FORMAT

The DIAGNOSE command includes the following:

STATUS word (written by 82586):

C	(Bit 15)	- Command completed (see section 8.1)
B	(Bit 14)	- Busy executing command
OK	(Bit 13)	- Error free completion
FAIL	(Bit 11)	- Indicates that the self test procedure failed

COMMAND word:

EL	(Bit 15)	- End of command list
S	(Bit 14)	- Suspend after completion
I	(Bit 13)	- Interrupt after completion
CMD	(Bits 0–2)	- DIAGNOSE = 7

LINK OFFSET: Address of next Command Block

OPERATION DETAILS OF DIAGNOSE COMMAND

CU triggers the self test procedure of the Channel Interface Module. It performs the following sequence:

1) Starts Action Command.
2) Writes DIAGNOSE command byte to TX-FIFO.
3) Waits for command completion.
4) If diagnose succeeded, then prepares STATUS word with C = 1, B = 0, OK = 1, FAIL = 0; otherwise, prepares STATUS word with C = 1, B = 0, OK = 0, FAIL = 1.
5) Completes Action Command.

The Channel Interface Module performs the self test procedure in two phases: Phase 1 tests the counters and Phase 2 tests the trigger logic.

During Phase 1, Free Run, Exponential Backoff Time-out, Slot Time and Collision Counters are checked.

The test is performed in the following steps:

1) All counters are RESET simultaneously.
2) Start counting.
3) Stop counting when the Free Run Counter (10 bits), and Exponential Backoff Counter (10 bits), wrap from 'All Ones' to 'All Zeros.' Simultaneously, the Slot Time counter switches from 01111111111 to 10000000000, and the collision counter (4 bits) wraps from 'All Ones' to 'All Zeros.'
4) Phase 1 is successful if the 10 least significant bits (when applicable), of all four counters are 'all zeros.'

During Phase 2, the test is performed in the following steps:

1) Reset Exponential Backoff Shift Register, and all counters.
2) Temporarily configure internally, Exponential Backoff logic according to the following:

   SLOT-TIME = 0BH
   LIN-PRIO = 02H
   EXP-PRIO = 01H
   BOF-MET = 00H
3) Emulate internally, transmission and collision.
4) If the most significant bit of Exponential Backoff Shift Register is 0, then go to step 3.
5) Check Mask Logic output for being 'All Ones' (Free Run Counter is 'All Ones' at this point and Exponential Backoff Shift Register is also 'All Ones').

If Step 5 is successful, then a 'Passed' status is returned otherwise, 'Failed' status is returned.

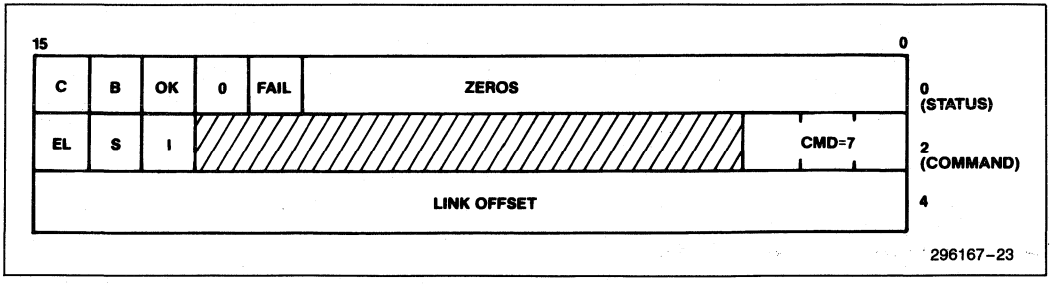


Figure 21. The DIAGNOSE Command Block

9.0 FRAME RECEPTION

Section 7 described how the user can control frame reception. This section presents the details of receiving and storing frames in memory.

9.1 Receive Frame Area (RFA)

The Receive Frame Area, RFA, is prepared by the host CPU. The 82586 places data into the RFA as frames are received. The RFA consists of a list of Receive Frame Descriptors (FD), each of which is associated with a frame. The RFA-OFFSET field of the SCB points to the first FD of the chain; the last FD is identified by the End of List flag (EL). See Figure 22.

In general, the incoming frame length is unknown beforehand. Rather than allocating buffers with a length greater than the largest expected frame, the 82586 makes it possible to store frames in a sequence of small buffers, which are chained into complete frames. Buffers are chained via Receive Buffer Descriptors (RBD), which point to a single buffer.

9.2 Frame Descriptor (FD) Format

The FD (see Figure 23) includes the following fields:

STATUS word (**written** by the 82586):

C	(Bit 15)	- Completed storing a frame.
B	(Bit 14)	- The 82586 knows about this FD and is ready to use it.
OK	(Bit 13)	- Frame received successfully. If this bit is set, then all others will be reset; if it is reset, then the other bits will indicate the nature of the error.
S11	(Bit 11)	- CRC Error: The received frame experienced a CRC error and was *well aligned.*

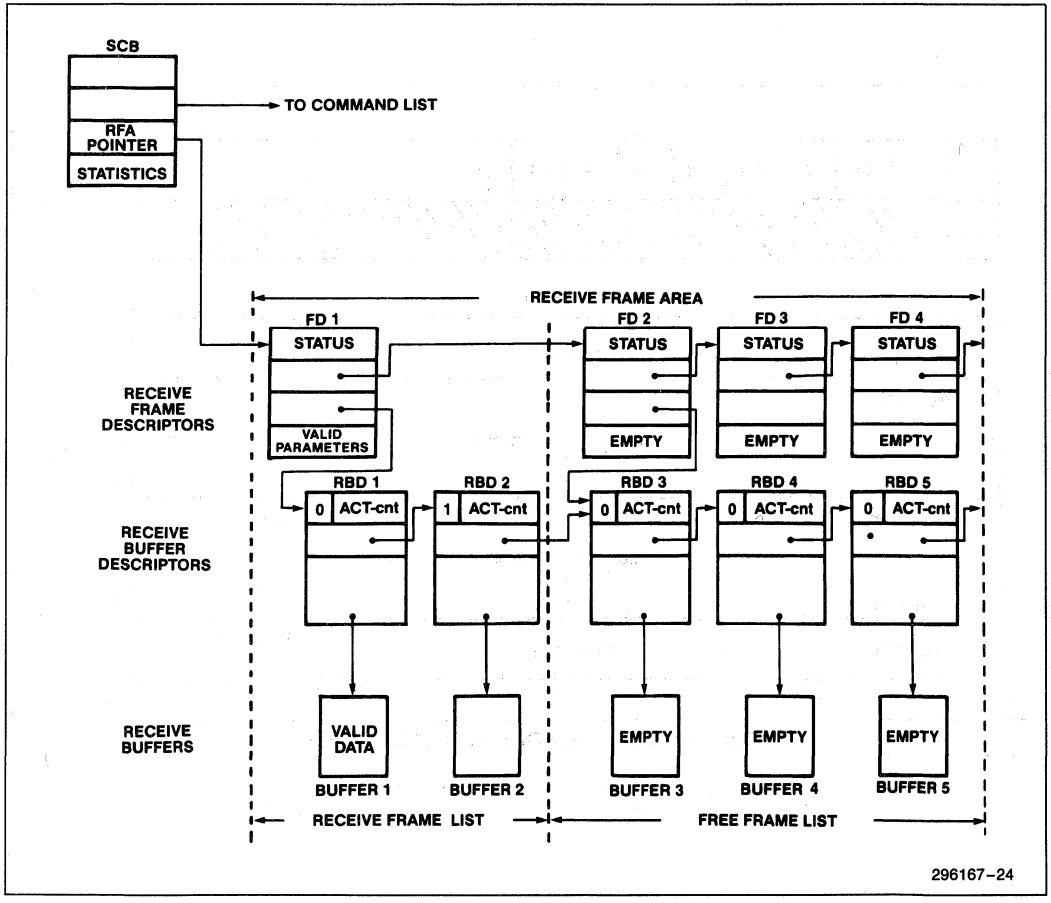


Figure 22. The Receive Frame Area

S10 (Bit 10) - Alignment Error: The received frame experienced a CRC error and was *misaligned.* The 82586 does not report misaligned frames with a correct CRC.

S9 (Bit 9) - The RU ran out of resources during reception of this frame.

S8 (Bit 8) - RCV-DMA overrun.

S7 (Bit 7) - The received frame had fewer bytes than configured Minimum Frame Length.

S6 (Bit 6) - No EOF flag detected (only when configured to Bitstuffing).

COMMAND word:

EL (Bit 15) - The last FD on the list.

S (Bit 14) - The RU should be suspended after receiving this frame.

LINK OFFSET: Address of the next FD on the list.

RBD-OFFSET (initially prepared by the CPU and later may be updated by the 82586): Address of the first RBD that represents the Information Field. RBD-OFFSET = 0FFFFH means that there are no RBDs associated with this FD.

DESTINATION ADDRESS (**written** by the 82586): Contains Destination Address of received frame. The length in bytes is determined by the Address Length configuration parameter.

SOURCE ADDRESS (**written** by the 82586): Contains Source Address of received frame. Its length is the same as DESTINATION ADDRESS.

LENGTH FIELD (**written** by the 82586): Contains the 2 byte Length Field of the received frame.

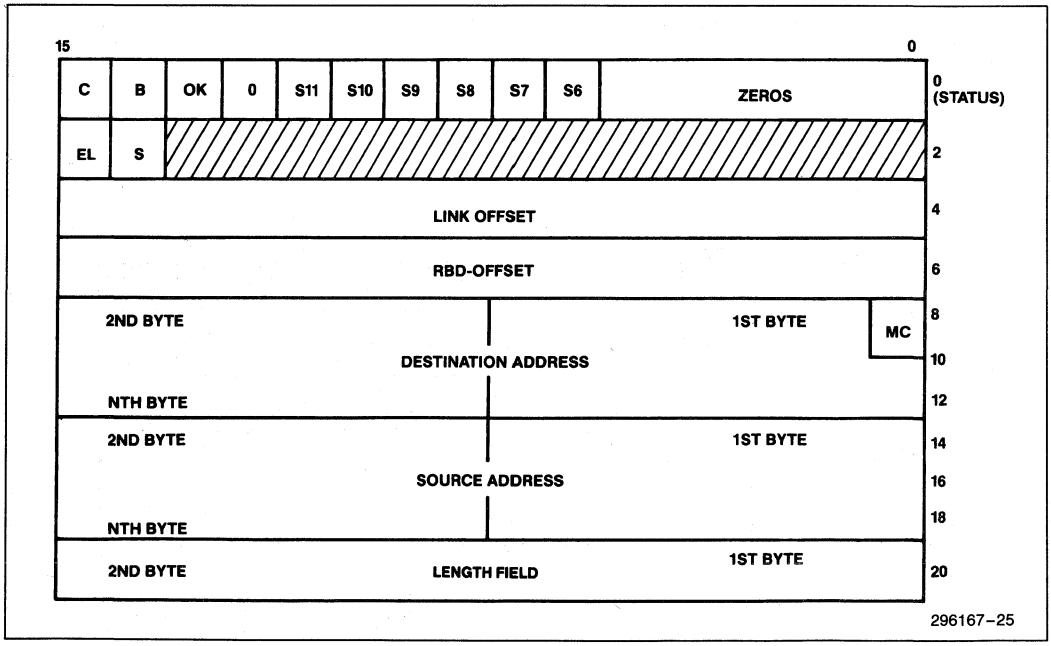


Figure 23. The Frame Descriptor (FD) Format

9.3 Receive Buffer Descriptor Format

The Receive Buffer Descriptor (RBD) holds information about a buffer; size and location, and the means for chaining RBDs (forward pointer and end-of-frame indication).

The Receive Buffer Descriptor contains the following fields:

STATUS word (**written** by the 82586):

EOF (Bit 15) - Last buffer holding the received frame.

F (Bit 14) - ACT COUNT field is valid.

ACT COUNT (Bits 0–13) - Number of bytes in the buffer that are actually occupied.

NEXT RBD OFFSET (**written** by the CPU): Address of the next BD in the list of BDs.

BUFFER ADDRESS (**written** by the CPU): 24-bit absolute address of buffer.

EL/SIZE (**written** by the CPU):

EL (Bit 15) - Last RBD in the list.

SIZE (Bits 0–13) - number of bytes the buffer is capable of holding.

9.4 Initial Structure of the Receive Frame Area

To enable the 82586 to receive frames, the host CPU must setup the following structure:

- The RFA OFFSET word in SCB should point to the first FD on the list.
- The LINK OFFSET of each FD in the list should point to the next FD.
- The EL bit in the last FD should be set.
- The RBD OFFSET of the first FD in the list should point to the first RBD in the RBD list. The RBD OFFSET in the remaining FDs must be 0FFFFH; it will be later loaded by the 82586 with the address offset of the first RBD assigned to the frame (unless the No Resources state is reached).
- The NEXT RBD OFFSET of each RBD in the list should point to the next one.
- The EL bit in the last RBD should be set.
- The BUFFER ADDRESS and SIZE fields in each BD should indicate the location and available space in a buffer.

9.5 Detailed Operation of Receiving a Frame

The Channel Interface Module selects the frames destined for the 82586 according to the Destination Address of the frames passing on the link. A frame is selected if it is at least 6 bytes long and its address matches the Individual Address or Multicast Address or Broadcast Address. It transfers the selected frames, with their status, to the RCV-FIFO. RCV-DMA transfers the frames from the RCV-FIFO to memory under control of the Receive Unit.

For every frame, the RU sets up a FD and RBDs in memory. The loading of each buffer is done by RCV-DMA in parallel with prefetching the next buffer by RU. After completing frame reception, the RU closes the last RBD and the FD, and sets up the structure for receiving the next frame.

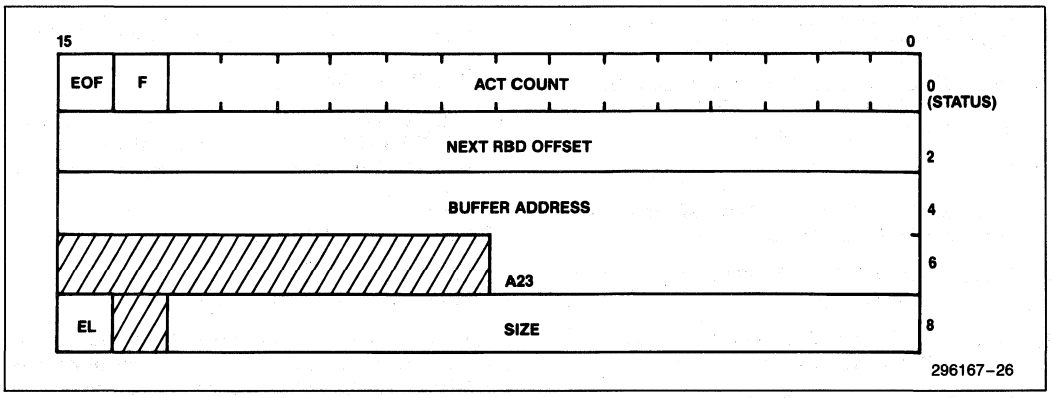


Figure 24. The Receive Buffer Descriptor (RBD) Format

SETUP NEXT FD

The RU performs the following sequence to setup a FD:

1) Writes a word with B = 1, and the remaining bits set to zero, to STATUS word of the next FD.

2) If AL-Location configuration parameter is zero, initiates RCV-DMA with the address of the first byte of the DESTINATION ADDRESS and a byte count of twice the Address Length configuration plus two.

3) Saves the value of the NXT FD ADR in the CUR FD ADR. This operation is done internally and is transparent to the BUS.

4) Reads COMMAND word of the current FD and saves EL and S bits internally.

5) Reads LINK OFFSET word of FD and saves it in the address of NEXT FD.

6) If a buffer has been prefetched, writes the offset part of its address to the RBD-OFFSET field of the current FD.

7) If a buffer is not available, reads the RBD OFFSET word from the current FD and saves it in the Look Ahead RBD address. If it is all ones, sets internal END-OF-LAST-BUFFER flag.

8) If AL-Location is one, forces a RCV-DMA completion.

9) Goes to the buffer prefetch/transfer cycle.

BUFFER PREFETCH/TRANSFER CYCLE

The RU prefetches a buffer according to the Look Ahead BD address. It also maintains two internal flags for beginning and end of a cycle.

1) If a buffer is already full, writes F = 1 and current size to STATUS word of current BD.

2) Saves address of next RBD in current RBD and address of the Look Ahead RBD in next RBD. Saves Next Size in current size.

3) If END-OF-LAST-BUFFER is set, go to 7.

4) Reads RBD-OFFSET from Next BD and saves it in Look Ahead BD address.

5) Reads 24-bit BUFFER ADDRESS from Next BD and saves it internally.

6) Reads EL SIZE word from Next BD and saves it in internal EL bit and Next size.

7) Waits for completion of RCV-DMA.

8) IF END OF LAST BUFFER is set, starts discarding incoming frame and quits.

9) Initiates RCV-DMA with Next Buffer Address and Next Size as byte count.

10) If the EL bit is set, sets END-OF-LAST-BUFFER.

11) Repeats the cycle.

CLOSE FRAME

When the RU reads an End of Frame from RCV-FIFO (indicating the end of a received or discarded frame), it performs the following sequence to close the frame:

1) Reads the status (2 bytes) from RCV-FIFO and saves it internally.

2) If the RU is not in the READY state, it goes straight to Completion of Reception sequence.

3) If the frame status indicates that there is an error (including short frame) and the Save Bad Frame configuration parameter is zero, or ADDRESS and LENGTH fields in FD are not full, it reuses the FD; Backs up Current FD to Next FD, discards pointer to the current RBD, and goes straight to step 9.

4) If the frame included only ADDRESS and LENGTH fields (i.e. no buffers used) writes 'all ones' to BD-OFFSET of Current FD.

5) If the frame filled at least part of a buffer, writes to STATUS word of the current RBD: EOF = 1, F = 1, and actual byte count.

6) If the RU ran out of resources during reception of this frame, writes to STATUS word of Current FD: C = 1, B = 0, OK = 0, S9 = 1, and the remaining bits from the Receive-Byte-Machine.

7) If the RU did not run out of resources, write to STATUS word of current FD: C = 1, B = 0, and the remaining bits from the Receive-Byte-Machine.

8) Requests FR interrupt.

9) If there was no RU-START-REQUEST or RU-SUSPEND-REQUEST or S bit and the RU did not run out of resources, it sets up a new FD (see section 10.2).

10) Goes to complete reception of frame.

COMPLETION OF RECEPTION

Reception completion occurs when the RU encounters an end of frame (after closing it) regardless of its state (i.e. also when discarding a frame). The decision on how to proceed is determined by the following: EL bit, S bit, RU-START-REQUEST, RU-SUSPEND-REQUEST and whether the RU ran out of buffer descriptors.

The following sequence is performed by the RU at Reception completion:

1) If the RU ran out of buffers or Frame Descriptors during current frame reception, then: change state to NO-RESOURCES, request an RNR interrupt, clear RU-SUSPEND-REQUEST and (internal) S bit, and start discarding.
2) If the received frame was longer than the Minimum Frame Length, update all four statistics registers (see details in section 8.6). Note that the RSCERRS statistics register is updated if the RU is in the NO-RESOURCES state or if it ran out of buffers during this frame.
3) If the S bit of current frame or RU-SUSPEND-REQUEST is set and RU-START-REQUEST is not, then: if the RU is in the READY state, request an RNR interrupt, and change state to SUSPENDED.
4) If RU-START-REQUEST is set and the S bit not, then: change the RU state to READY and perform Setup new FD.
5) If the RU-START-REQUEST is set and also S bit is set, perform Setup new FD and change state to SUSPENDED.
6) If the RU state is different from READY, start discarding.
7) If the RU is in the READY state or just exited it, then perform the following:
   a) Set respective INTERRUPT-IS flags.
   b) Deactivate hardware Interrupt signal.
   c) Clear interrupt request flags.
   d) Update SCB STATUS word according to the new state and INTERRUPT-IS flags.
   e) Activate hardware Interrupt signal.

If a command is being executed by the serial machine (e.g. Configure, IA Set-up, Dump, etc.) it will ignore incoming frames during command execution. Thus, frames can be lost during this time. Conversely, if a frame is being received, the serial machine will wait until the entire frame has been received before the command is executed. Section 10.5 describes simultaneous Transmit and Receive operation.

OPERATION OF RECEIVE-BYTE MACHINE DURING RECEPTION

For every frame that arrives from the Receive-Bit-Machine (after it strips the Preamble and detects the SFD field) the Receive-Byte-Machine decides, according to the Destination Address, whether to accept the frame (see section 12). If the frame is accepted, the Receive-Byte-Machine transfers the Destination Address, Source Address, Length Field and Data field to RCV-FIFO. If it is not accepted, nothing enters RCV-FIFO. If RCV-FIFO overruns, the Receive-Byte-Machine keeps overwriting the input buffer of RCV-FIFO (without affecting the bytes that are already in RCV-FIFO). As soon as space is available, new bytes enter RCV-FIFO. The Receive-Bit-Machine strips the CRC and Padding (if applicable) and prepares the STATUS word with bits indicating: occurrence of any error, CRC or alignment error, DMA overrun, frame too short, or absence of EL flag (in Bitstuffing). The Receive-Byte-Machine enters the STATUS word in two consecutive bytes to the RCV-FIFO.

10.0 BUS INTERFACE

The 82586 can operate on the host CPU local bus as a bus master with shared status lines. Since status lines are in common, all local bus resources (Clock Generator, 8288 Bus Controller, Address Latches and Data Transceivers) can be shared by the CPU and the 82586. To facilitate minimum component count designs, the 82586 bus interface has been optimized for use with an 80186 16-bit microprocessor. The 82586 also interfaces to the 8086/88 and other processors. Dual port memory configurations can be used to minimize the impact of 82586 bus bandwidth requirements on system performance.

When operating in Maximum Mode (MN/$\overline{MX}$ pin strapped to logic zero) pins $\overline{S0}$ and $\overline{S1}$ are connected to the 8288 bus controller, which in turn generates all control signals for the system interface. In this case, the full address range of 16 megabytes is available.

In Minimum Mode (MN/$\overline{MX}$ pin strapped to logic one), the 82586 generates ALE, $\overline{DEN}$, DT/$\overline{R}$, $\overline{RD}$ and $\overline{WR}$ signals. An address space of 4 megabytes is available.

The system bus is acquired and released using HOLD/HLDA protocol directly interfaced to the 80186. External hardware can be used to convert HOLD/HLDA to $\overline{RQ}$/$\overline{GT}$ protocol suitable for interfacing to the 8086 CPU.

When interfacing to a 16-bit bus, data is multiplexed with the 16 lower address pins. Examples of this mode are systems with 8086, 80186, 80286 or other 16-bit microprocessors. On an 8-bit data bus, data is multiplexed with the 8 lower address pins. Examples of this mode are systems with 8080/8085, 8088 or other 8-bit microprocessors.

All bus timing and loading specifications are consistent with those of the 80186 system. Bus timing and loading specifications are given in the 82586 Data Sheet.

The 82586 CLK signal input must receive MOS level inputs. The 82586 can operate with an 80186 or a 8086-1 using an 8 MHz 50% Duty Cycle clock (full performance), or with a 5 MHz 8086 using a 5 MHz 30% Duty Cycle clock (reduced performance).

10.1 Memory Addressing and Organization

The 82586 addresses memory as a linear sequence of 16 megabytes. It establishes transfers as 16-bit words when it is used with an 8086/80186/80286 or as 8-bit bytes using an 8088 or other 8-bit processors.

As a Master Peripheral with the 8088, the 82586 treats memory as a single bank (D7–D0) of 1M 8-bit bytes addressed by address lines A19–A0. Address lines A23–A20 are not used, $\overline{BHE}$ is strapped HIGH.

When operating with the 8086/80186, the 82586 treats memory as a high bank (D15–D8) and a low bank (D7–D0) of 512K 8-bit words addressed in parallel by A19–A1. The 82586 performs Reading/Writing from/to full word locations only. The $\overline{BHE}$ line is always LOW. Word transfers are restricted to even addressing, although the 82586 does not enforce pin A0 to LOW. When attempting to write to an odd location (i.e. when an odd buffer address is set), the odd address will be output to the address lines, but a full word will be output to the data lines, in which the low byte will be undefined. The manner in which line A0 being HIGH is treated is system dependent. The data words flow through pins D15–D0.

The Chart below summarizes the operation of $\overline{BHE}$ and A0 signals:

$\overline{BHE}$	A0
0	0 whole word
0	1 system dependent
1	0 lower byte to/from even address
1	1 upper byte to/from odd address

When operating with the 80286, the full 16 megabyte address space is available. It has the same characteristics and restrictions as the 8086/80186.

The 82586 accesses all control information from a segmented memory structure. The 16-bit pointers residing in the various structures are used as an offset to a 24-bit base location. Hence, all FD, RBD, CB, TBD, SCB structures must reside in 64K byte boundaries. Data buffers may reside anywhere in address space, and are directly addressed as a 24-bit address.

10.2 Bus Operation

Each memory transfer cycle consists of at least four CLK cycles. These are referred to as t1, t2, t3 and t4. The address is generated by the 82586 beginning at t1, and data transfer occurs on the bus during t2 through t3. In the event that a READY indication is not received from the addressed memory, WAIT states (tW) are inserted between t3 and t4. Each inserted WAIT state has the same duration as a CLK cycle.

The 82586 outputs status ($\overline{S1}$ and $\overline{S0}$) to provide type-of-memory-cycle information to the 8288 Bus Controller in Maximum Mode or in Multibus Configuration. The 8288 generates memory read and write commands, and issues control signals to the address buffers and data transceivers. The 82586 provides $\overline{RD}$, $\overline{WR}$ and ALE signals for memory transfers in Minimum Mode. A multi-master system bus can be constructed using the 8289 Bus Arbiter. The key arbiter inputs are the same as for the 8288: local status lines $\overline{S1}$ and $\overline{S0}$.

Operation of the 82586 is structured so that all command, status and data flow is via memory. Therefore, the S2 status line of the 8086 and 80186 families is not required and is not provided by the 82586. With the exception of SCP, which is always read from location 0FFFFF6H, there are no transfers to/from a fixed address system resources (I/O memory).

READ

The read cycle begins at t1 by generating the address. The memory read command signal (MRDC from the 8288) is asserted at t2. This command causes the addressed device to enable its data bus drivers onto the system bus. Some time later, valid data will drive the READY line HIGH. The data will be sampled by the 82586 at t4. If the READY line is still LOW in t3, then tW cycles are inserted between t3 and t4 until the READY line goes HIGH. When the MRDC signal returns to the HIGH level, the addressed device will again tri-state its data bus drivers. If a transceiver (8286/8287) is required to buffer the local bus, the direction ($DT/\overline{R}$) and enable ($\overline{DEN}$) controls are provided by the 8288 Bus Controller (or from the 82586).

WRITE

A write cycle begins by generating the address during t1. At t2 the processor generates data to be written. This data remains valid until the middle of t4. During t2, t3 and tW, the advanced memory write command (AMWTC) from the 8288 is asserted, while the normal memory write command (MWTC) is asserted during t3 and tW only. MWTC is used by older memories requiring valid data prior to the write command.

SYNCHRONIZATION WITH LOWER SPEED MEMORIES

The 82586 uses the standard READY mechanism to work with memories having long access times.

As bus state t3 is reached, the READY signal is tested. While the READY signal is LOW, wait states (tW), are added and all control signals are stretched accordingly. While in wait states, (tW), the READY signal is tested at every clock cycle. The first time that READY is found to be HIGH, state t4 is entered, finishing the bus cycle.

In a synchronous environment, the user is able to guarantee setup and hold times for the READY signal related to the system clock. Synchronous READY input is chosen and the number of wait states is predictable.

For asynchronous environments, the user is not required to provide precise setup and hold times for the READY signal. A high speed resolution circuit synchronizes the asynchronous READY input internally to the 82586. Setup and hold times for the asynchronous READY input are not required for correct operation but for recognition at a certain clock. The number of wait states is therefore not always predictable.

Synchronous READY input is sampled by the 82586 at the beginning of t3. Setup and hold time requirements refer to the falling edge of the CLK when entering t3 state.

Asynchronous READY input is synchronized by the 82586 in the middle of t2. Setup and hold times refer to the rising edge of the CLK while in state t2.

In Minimum Mode, the 82586 has one input READY pin, SRDY/$\overline{\text{ARDY}}$. This pin is software programmable to work as either synchronous or asynchronous. Upon RESET, the 82586 automatically defaults to asynchronous mode. The CONFIGURE command is used to program the circuitry to synchronous or asynchronous READY.

In Maximum Mode the SRDY/$\overline{\text{ARDY}}$ pin behaves exactly as in Minimum Mode. In addition, the READY pin provides an alternate synchronous READY connection. Internally, the 82586 performs a logic OR function, between the SRDY/$\overline{\text{ARDY}}$ pin and the READY pin.

10.3 Bus Acquisition

The system bus is acquired and released by means of the HOLD/HLDA protocol.

When the 82586 needs the bus, it activates the Hold signal. Upon receiving the HLDA, it initiates bus transfers. At the end of bus transfers, it relinquishes the bus by deactivating the HOLD and, subsequently, the host takes away the HLDA.

Figure 25A shows the number of clocks the 82586 requires after getting an HLDA before it starts a bus transfer.

The 82586 does not hold the bus unless it is doing memory transfers, with one exception. At the end of a received frame, the 82586 starts post-frame processing, which includes posting Status for the recently received frame and getting ready for the next frame to be received. All these processes have to be completed such that Interframe Spacing requirements (9.6 μs for IEEE 802.3) are met. To be able to do that, the 82586 may hold the bus for 173 clock periods (Typical value for AL-LOC = 0, SAV-BF = 0). The chip, however, does not really use the bus for entire 173 clock periods for post-frame processing. About 66 of the 173 clock periods are not used (no Read or Write operations). The chip simply holds the bus for 173 clock periods in anticipation of the next frame which may come in after the IFS time (9.6 μs) in the worst case (see Figure 25). If the RU finds that the next frame is not coming in immediately, then the 82586 will release the bus after 173 clock periods. If the next frame is coming in, the chip will hold the bus even longer and transfer the first part of the next frame into memory. In this case, receive data DMA will be interleaved with receive control DMA. The system design must not interpret post-frame processing as a fault condition.

The CPU can force the 82586 off the bus by removing HLDA. The 82586 will complete the current bus transfer and will relinquish the bus within a maximum of four clock cycles in Word mode or eight clock cycles in Byte mode, see Figures 26A and 26B. However, if the 82586 still has some pending transactions, the Hold will be activated again after 1 clock cycle (minimum hold drop time is 1 clock cycle).

READY TIMEOUT

The 82586 Ready input is sampled during DMA operations. If a system error (e.g. RAM error) causes the Ready to not be returned to the 82586, the 82586 will indefinitely hold on to the bus. A removal of HLDA will not make the 82586 release the bus under these conditions (typically, the 82586 gives up the bus within 3 clock cycles of HLDA going inactive). The system designer must incorporate some external logic to detect the 'Ready Absence' condition and reset the 82586 in such an event. In addition, the CPU must initiate some recovery procedures (e.g. RAM test) to isolate the problem and follow up with remedial procedures.

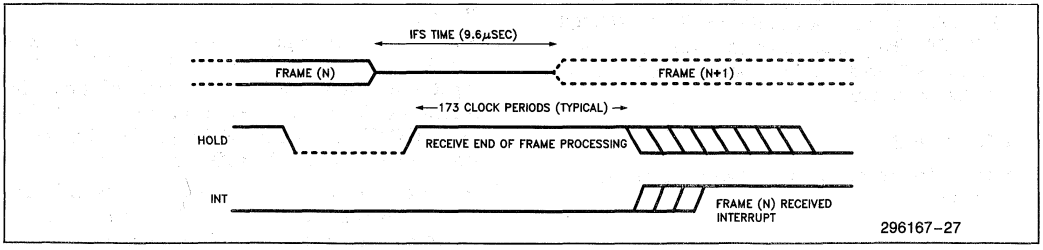


Figure 25. Receive End of Frame Processing

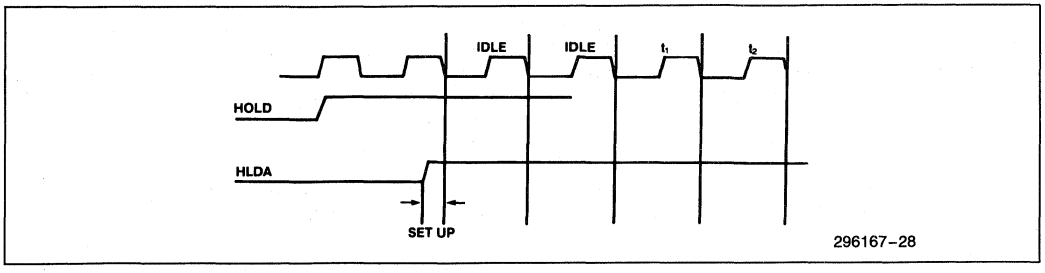


Figure 25A. HOLD/HLDA: Bus Transfer Timing

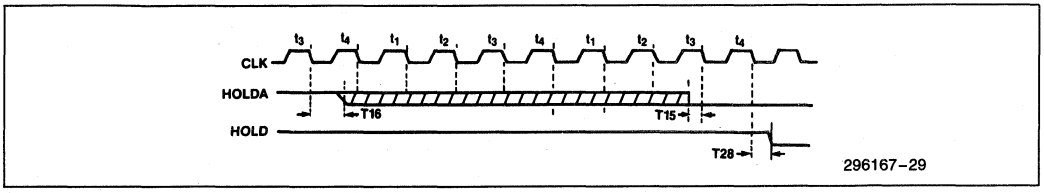


Figure 26A. 82586 Byte Mode Bus Relinquish Timing

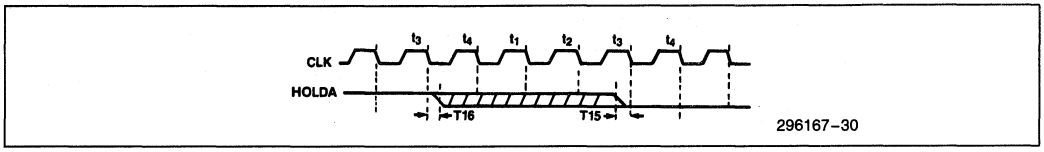


Figure 26B. 82586 Word Mode Bus Relinquish Timing

DETAILS OF THE HOLD/HLDA PROTOCOL

The state diagram of Figure 27 defines the states and excitations of the bus acquisition protocol.

STATES:

NO BUS REQUEST: The 82586 is not requesting the bus via HOLD, because Internal Bus Request (IBRQ) is not pending, or when the IBRQ arrived, HLDA level was HIGH.

BUS REQUESTED: The 82586 has issued HOLD (bus request). HLDA is LOW, i.e. bus was not yet granted.

BUS MASTER/TRANSFERS PERFORMED: The 82586 owns the bus and performs transfers.

BUS MASTER/NO TRANSFERS PERFORMED: The 82586 owns the bus, but does not use it for transfers. This is used by the 82586 to lock the bus for a few system clock cycles for READ/MODIFY/WRITE operations, or when the 82586 knows that it will need the bus soon (e.g. when buffer switching occurs).

EXCITATIONS:

RESET: Software or Hardware Reset initializes the machine.

IBRQ: Internal Bus Request. Any internal demand to perform bus transfers on the bus: Receive or Transmit DMA, CU or RU.

STD: Special state different from t1, t2, t3, tW and t4, for which the 82586 masters the bus, but no transfers are performed (ALE, $\overline{RD}$, $\overline{WR}$ are inactive in Minimum Mode or $\overline{S0}$, $\overline{S1}$ are inactive in Maximum Mode).

SPT4: State prior to t4. May be t3 for zero wait states or the last tW for any number of wait states.

BM: Configured to 8-bit bus width.

MA0: Memorized A0. A0 was true during the last t1 (Odd-Address bus cycle).

The following algorithm is followed regarding bus acquisition.

1) RESET sets the 82586 to the NO BUS REQUEST state.
2) The internal bus request raises HOLD output and switches the state to BUS REQUESTED. HLDA input, being LOW, enables this switch of state. The reason for this condition is to differentiate HLDA being HIGH as a result of the previous bus cycle from newly generated HOLD acknowledge.
3) Appearance of HLDA sets the state to BUS MASTER/TRANSFERS PERFORMED. The bus is acquired and read/write operations are started.

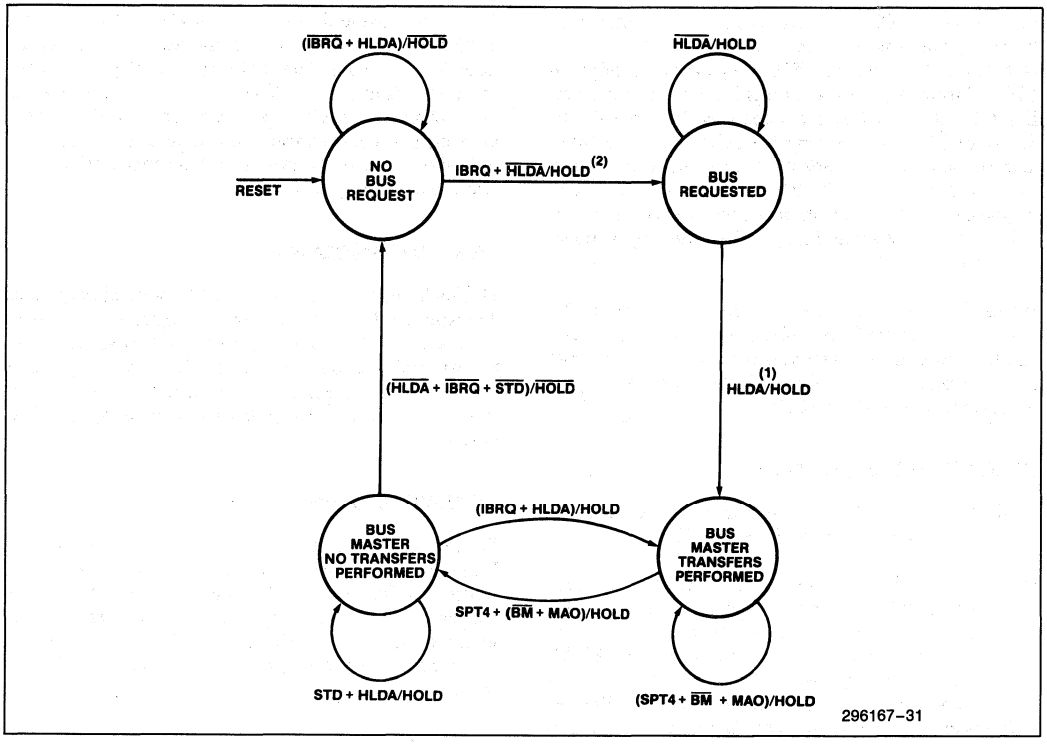


Figure 27. The HOLD/HLDA Handshake

4) Prior to entering bus state t4, the 82586 evaluates the next action: If HLDA input is active and there are pending internal bus requests, the state remains BUS MASTER/TRANSFERS PERFORMED. Back to back bus cycles result. If HLDA is deactivated or no more internal bus requests are present, the 82586 switches to NO BUS REQUEST state.
5) State BUS MASTER/NO TRANSFERS PERFORMED is entered for all cases that no internal bus request exists but performance considerations require holding the bus until the request shows up.

10.4 FIFO-Threshold Mechanism

The on-chip Receive and Transmit FIFOs provide buffering between the serial channel and system bus. On the system side, data is written into the Transmit FIFO or read from the Receive FIFO, using the 82586 DMA data channels. The Transmit DMA bursts fill the Transmit FIFO at the bus data rate. The 82586 Channel Interface Module pulls bytes of data from the Transmit FIFO and transmits to the link at the serial bit rate. The Receive FIFO operates analogously, filled by the Channel Interface Module and emptied by bursts for Receive DMA channel.

The transactions involved in bus acquisition and release imply overhead resulting in losing system clocks due to signal propagation delays and sample time requirements. For many short DMA bursts, up to 5 clocks are lost due to this switching. On the other hand, long DMA bursts imply that bus masters, other than 82586, may experience long delays in bus acquisition. Thus, an optimal FIFO-Threshold setting exists.

The 82586 provides a means for choosing the optimal setting required by the user's application, using the FIFO-Threshold configuration parameter.

TRANSMIT FIFO OPERATION

Starting from an empty Transmit FIFO, the transmit DMA starts reading memory locations, and pushing bytes into Transmit FIFO. The Channel Interface Module starts transmitting on the link, pulling bytes from the Transmit FIFO, at the serial bit rate. Because the data system (byte) rate is faster than the serial's, the Transmit FIFO will eventually become full. At this point, the DMA burst stops.

Meanwhile, the Channel Interface Module continues to transmit, thus emptying the FIFO. When the number of bytes in the Transmit FIFO equals the configured FIFO-Threshold, the system bus is requested for a new DMA burst. If the bus is not acquired fast enough, the Transmit FIFO will be emptied by the Channel Interface Module and a Transmit FIFO underrun occurs. The optimal value for the FIFO-Threshold is such that it guarantees bus acquisition before underrunning the FIFO, and at the same time provides the longest possible burst.

When the bus is acquired, the new Transmit DMA burst continues until the FIFO is full. This mechanism can be viewed as an hysteresis process, with the upper bound Transmit FIFO full and the lower bound the programmed FIFO-Threshold.

RECEIVE FIFO OPERATION

At the beginning of frame reception, the Channel Interface Module writes into the Receive FIFO.

When the number of bytes in FIFO equals 15 minus FIFO-Threshold, the system bus is requested. This mechanism allows for a delay in bus acquisition equal to the number of empty bytes left in the Receive FIFO. If the bus is not acquired in time, a receive overrun occurs. The system bus is released when the Receive FIFO is emptied by the Receive DMA. Further acquisition of the bus starts when the Receive FIFO contains a number of bytes equal to 15 minus FIFO-Threshold. The mechanism is analogous to the Transmit FIFO.

NOTE:
In the case of a Receive FIFO overrun, the Channel Interface Module keeps overwriting the input buffer to the FIFO, and the bytes get lost. A special status will indicate this condition.

10.5 Bus Cycle Interleaving

The 82586 has four independent on-chip DMA channels:

- Receive DMA channel. Used for writing received frames to memory.
- Transmit DMA channel. Used for reading transmit frames from memory.
- CU input/output channel, used for CU read and write operations.
- RU input/output channel, used for RU read and write operations.

Receive and Transmit DMA channels operate in bursts controlled by the FIFO-Threshold mechanism. The CU and RU channels initiate single read or write operations as dictated by CU and RU activities. CU related activities are: CU Initialization, SCB processing including CA acceptance and SCB status update, CB execution, TBD prefetch. RU related activities are: CA acceptance, SCB status update, FD setup, RBD prefetch, end of receive frame processing. In the case that two or more channels require the system bus simultaneously, an arbitration mechanism located in the bus interface unit allocates the channels and determines priority between various requests.

TRANSMIT PROCESS

During Transmit operation, the CU prefetches the next Transmit Buffer Descriptor while the current data buffer is being read by the Transmit DMA channel. In order to minimize disturbances to the data rate, and to avoid a Transmit FIFO underrun, any two CU operations are separated by at least two data reads (if requested).

RECEIVE PROCESS

During reception, the RU prefetches the next Receive Buffer Descriptor while the current data buffer is being written by the Receive DMA channel. As in the case of transmit, any two RU operations are separated by at least two data writes. This rule minimizes disturbances to the data rate and helps to avoid Receive FIFO overruns.

SIMULTANEOUS RECEIVE AND TRANSMIT OPERATION

There are two cases where Transmit and Receive DMA channels work simultaneously.

In Internal/External Loopback configuration, both channels operate simultaneously, on an equal priority basis. Furthermore, CU or RU memory cycles are also interleaved with the same priority as Transmit and Receive. Effectively, if three sources compete for the bus, they operate on an interleaved basis.

The second case of simultaneous Transmit and Receive occurs when the receive frame arrives while the CU is setting up a transmission. The Transmit process continues simultaneously with Receive until the Transmit FIFO fills. As in Loopback, Transmit and Receive DMA priorities are equal. As soon as the receive frame ends, Transmit resumes operation.

The 82586 will interleave command related operations with receive related operations according to the rules described below.

- Control transfers (interaction with the System Control Block, Command Blocks, Receive Frame Descriptors and Buffer Descriptors) and data transfers (interaction with Data Buffers) are interleaved re-

gardless whether the transfers pertain to command or received operations. Two data transfers are executed for each control transfer.

- Data transfers to command and receive operations occur concurrently, on an alternating basis (one transfer in each direction).
- Control transfers related to receive operations have priority over command operations. In other words, command control transfers are suspended or delayed for as long as necessary to complete pending receive control transfers.
- Receive data transfers, command data transfers, and control transfers can occur concurrently. There will be one data transfer in each direction of each control transfer.

Consider the following example. Assume the 82586 is in the process of receiving a frame. The CPU wants to have a frame transmitted and activates Channel Attention.

While writing the received frame into system memory, the 82586 will concurrently read the appropriate command control information (SCB, Command Block, and TBD). If the 82586 needs to execute receive control transfers (beginning-of-frame processing, buffer look-ahead, end-of-frame processing), it delays command control transfers. Two receive data transfers are executed for each (receive or command) control transfer.

When all control transfers related to the transmit command are completed, the 82586 will begin executing transmit data transfers. One transmit data transfer will be executed for each receive data transfer, a transmit data transfer and a receive control transfer (not necessarily in that sequence). Transmit data transfers will stop when the Transmit FIFO is filled up.

HOLDING THE BUS

In special cases, the 82586 does not release the bus, even though no specific reads or writes are executed. The purpose of this operation is to avoid bus acquisition and release dead times if several input/output operations are required. Another reason is to guarantee 82586 performance in critical timing cases.

The 82586 holds the bus in the following cases:

- RBD Prefetch. The bus is not released between two consecutive bus accesses to minimize prefetch time. This is important since minimum buffer length is strongly dependent on the time it takes the 82586 to prefetch a RBD.
- TBD Prefetch. Analogous to RBD Prefetch.
- Receive End of Frame Processing. The bus is not released throughout this process to ensure only one bus request.
- Transmit or Receive Buffer Switching. To minimize the danger of Transmit FIFO underruns or Receive FIFO overruns.

10.6 CPU/82586 (CA/INT) Handshake

The INT pin is used to notify the CPU about one or more of the following events:

- A Command Block with its 'I' bit set was executed (CX interrupt).
- A frame was received (FR interrupt).
- The CU became Not Active (CNA interrupt).
- The RU became Not Ready (RNR interrupt).

82586 INTERRUPT REQUEST SEQUENCE

Once an event requiring an interrupt has occurred, the following sequence is performed by the 82586:

1) INT pin is set to its low level (inactive).
2) The status word in SCB is written, denoting the source of the interrupt (CX, FR, CNA or RNR interrupt), together with the states of the CU and RU.
3) INT pin is raised (set to active).

82586 RESPONSE TO CA

The CU is responsible for control command acceptance, following the trailing edge on CA input. The CU will first finish all its higher priority activities and only then accept the control commands.

Higher priority CU activities that delay CA acceptance are:

a. Transmit Buffer Descriptor prefetch

b. Transmit buffer switching

c. Current Command Block completion

The 82586 will accept a CA prior to the setup of the next CB in the CBL.

The CU recognizes an RU control command and notifies the RU. The RU will first finish all its higher priority activities, and only then accept the control command.

Higher priority RU activities that delay CA acceptance are:

a. Receive Buffer Descriptor prefetch

b. Receive buffer switching

c. Receive end of frame processing

Only after the CU and RU have accepted the control command, the SCB command word is cleared. At that time, the CPU may issue the next CA to the 82586.

Internally to the 82586, the CA trailing edge is detected and latched. Prior to reading the SCB control command, the 82586 clears the latch. A new CA, given to the 82586 before the SCB command word is cleared, may be lost due to its being cleared before serviced. The user must refrain from such violations.

Upon detecting a falling edge on its CA input, the 82586 performs the CA acceptance sequence, as follows:

1) Determine which interrupt requests were acknowledged by the CPU. For each of them, clear the corresponding interrupt request bit in SCB status word.
2) Perform the control command acceptance procedure.
3) The INT pin is set LOW.
4) Write the SCB status word indicating the unacknowledged interrupt requests, and newly generated interrupt requests, together with CU and RU states.
5) If any interrupt request bit is active, set the INT pin to HIGH.

The 82586 does not wait for reception or transmission to end in order to process a CA. The SCB related operations will be carried out on an interleaved basis with the transmission or reception process.

11.0 NETWORK INTERFACE HARDWARE

The Network interface supports bit encoding, carrier sense, collision detection, link acquisition, and loopback.

NOTE:
The 82586 Receive-Byte-Machine and Receive-Bit-Machine are clocked by the Transmit Clock. Thus the Transmit Clock must be constantly applied to the 82586.

11.1 Encoding/Decoding

The 82586 receives an externally generated Transmit clock at the transmission bit rate. Data is transmitted in either NRZ (binary) or Manchester encoded form, depending on the chip configuration.

Due to the semi-static nature of the 82586's internal circuits, the Transmit clock HIGH time should not be longer than 1000 nanoseconds. Manchester encoded data requires 50% clock duty cycle. Therefore, when the 82586 is configured to perform Manchester encoding, the Transmit Clock frequency must be between 0.5 MHz and 10 MHz.

In the case of NRZ encoded data, Transmit Clock frequency can be from 100 KHz to 10 MHz, with the same limitation of the clock HIGH time as the Manchester encoded data. When Transmit data is idling, the TXD pin is held HIGH (logic '1').

The 82586 accepts an external Receive Clock, $\overline{RXC}$, that strobes the incoming Receive Data signal, RXD.

The Manchester/$\overline{NRZ}$ configuration parameter does not apply to the Receive data. The 82586 requires NRZ Receive data. Manchester data decoding is accomplished externally (the 82C501 chip for IEEE 802.3). The Receive Clock can be presented to the 82586 in two ways:

- all the time
- only during the Receive frame

In the case of Receive Clock only during Receive frame, the $\overline{RXC}$ pin should be held inactive when no Receive frame exists.

Receive Clock frequencies can be from 100 KHz to 10 MHz with HIGH time not longer than 1000 nanoseconds.

11.2 Carrier Sense

Carrier Sense is an indication of activity on the link, i.e. a signal from a transmitting station has reached this station. The 82586 can be configured to either accept it externally, or generate Carrier Sense internally (for Serial Interface Devices that generate Receive clock only during actual reception). The internally generated Carrier Sense occurs when the Receive Clock is present.

When transmitting, 82586 behavior related to Carrier Sense is as follows:

- When ready to transmit, and if Carrier Sense is active, the 82586 defers.
- When $\overline{CRS}$ goes inactive ($\overline{CRS}$ is synchronized to the transmit clock), the 82586 sets its Interframe Spacing (IFS) timer.
- When the IFS timer expires, the 82586 initiates its transmission regardless of the state of $\overline{CRS}$.
- The 82586 will abort transmission if $\overline{CRS}$ goes inactive anytime after it transmits the preamble. An override option is available.

When receiving, the 82586's behavior related to $\overline{CRS}$ is as follows:

- The 82586 starts the IFS timer when $\overline{CRS}$ goes inactive. The receiver is insensitive to external signals during the time-out.
- Carrier Sense being active any time other than during the IFS time causes the 82586 to sample its Receive Data input at a rate determined by the Receive Clock.

- When receiving, the 82586 samples the data on the falling edge of the Receive Clock after $\overline{CRS}$ becomes active.

Carrier Sense signal going active edge can be asynchronous to both Receive and Transmit Clocks.

The 82586 requires five bit times from the instant the $\overline{CRS}$ pin goes active to achieve internal synchronization with the received bit stream. The sixth bit is the first bit sampled as data. In the End of Carrier frame delimiting method, the 82586 hunts for the SFD field (10101011). Conversely, if two consecutive '0' bits are detected before the SFD field, the frame is aborted.

To have a clean frame closure, $\overline{CRS}$ signal going inactive edge should be synchronized to Receive Clock.

Carrier Sense can be configured to undergo filtering to ignore negative going glitches. The filter is programmable to a number of bit times (0–7) that must be exceeded before the signal is considered valid. To be classified as a glitch, the glitch must meet set-up and hold times. $\overline{CRS}$ is synchronized to the Transmit Clock.

Figure 28A shows a negative going glitch on $\overline{CRS}$. If the glitch lasts shorter than the filter setting, it will be filtered, and the IFS timer will not start on the rising edge of $\overline{CRS}$.

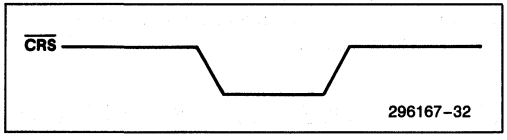


Figure 28A. The CRS Filter Stops Negative Going Glitches: the IFS Timer Does Not Become Active

Figure 28B shows a positive going pulse, which will not be filtered. At point 1, the $\overline{CRS}$ inactive will initiate the IFS timer and the 82586 will not respond to external signals for the IFS period. See section 12.3.

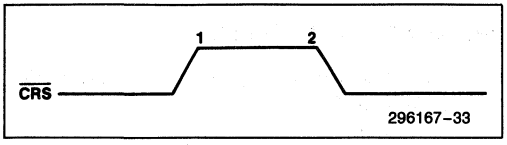


Figure 28B. The CRS Filter Passes Positive Going Glitches: the IFS Timer Becomes Active at Point 1

Positive going $\overline{CRS}$ glitches cause the 82586 back off timer to start; the 82586 will not respond to any external signal for the IFS period.

11.3 Collision Detection

Collision detection is usually done external to the 82586, typically by the Transceiver. An internal filtering mechanism prevents acceptance of a Collision Detection signal shorter than a configurable number of TCLK units (from 0 to 7).

The 82586 only looks for collision detect while actively transmitting, i.e. the first bit of the preamble has been transmitted. The collision is synchronized and recognized internally within a maximum of 4 Transmit Clock times after collision detection pin goes active.

The 82586 can be configured to cease transmission upon active Carrier Sense, instead of Collision Detect. This capability is called Internal Collision Detect. Internal Collision Detect is useful in two cases:

1) In conjunction with a class of transceiver that generates a Receive Data signal equal to the difference between the signal carried by the channel and the transmitted signal.
2) For point-to-point interconnection without using collision sensing transceivers. For example, collision detection could be realized by ANDing the $\overline{RTS}$ signal of each 82586, the result then tied to $\overline{CRS}$ of each 82586.

SQE TEST

Some transceivers (IEEE 802.3 compatible) generate the SignalQualityError Test, SQE TEST, (or 'heart beat') signal after the completion of each transmission to indicate proper operation of the collision detection circuitry. The SQE TEST signal is issued on the Collision Detect line, and is a 10 MHz signal, 10 ± 5 bit times in length. The 82586 checks the Collision Detect line for the SQE TEST for the duration of an Interframe Spacing after completion of transmission. The status of the next transmitted frame reports on the presence or absence of the SQE TEST. The SQE TEST mechanism is only meaningful if the 82586 is configured to external collision detect.

See section 12.3.

11.4 Serial Link Acquisition

The handshake between the 82586 and the Ethernet Serial Interface during transmission is accomplished with 'Request-to-Send' ($\overline{RTS}$) and 'Clear-to-Send' ($\overline{CTS}$) signals. $\overline{RTS}$ provides a means for turning on the Ethernet Serial Interface prior to actually sending bits. $\overline{CTS}$ is the means by which the Ethernet Serial Interface confirms that it is ready. It is also an external means for implementing a watchdog timer.

When the 82586 is ready to place bits on the serial link and $\overline{CRS}$ is inactive, it asserts $\overline{RTS}$ and awaits $\overline{CTS}$. Actual transmission starts within 1 transmit clock time after $\overline{CTS}$ rises. If it loses $\overline{CTS}$ prior to end of frame, transmission is aborted and error status is raised. An Ethernet Serial Interface that does not require this handshake may ground $\overline{CTS}$ (Intel 82C501). The 82586 deactivates $\overline{RTS}$ after transmission is completed.

The 82586 can operate with Serial Interface Devices that either do or do not return Carrier Sense during transmission. The 82586 must be configured to one of the two cases. If configured to expect Carrier Sense, then transmission will stop if Carrier Sense goes inactive (and after the preamble). If configured to transmit on no $\overline{CRS}$, then the 82586 is indifferent to Carrier Sense going inactive.

11.5 Loopback

The Loopback Modes are called by setting the respective configuration bits.

If both bits are set, Internal Loopback Mode is valid, and overrides External Loopback. In Internal Loopback Mode, the 82586 is logically disconnected from the Serial Interface Unit, therefore, the 82586 does not monitor link activity. The 82586 connects the Transmit Data to the Receive Data Signal, and Transmit Clock to the Receive Clock. To avoid FIFO overruns and underruns, bit processing is performed at one quarter $\overline{TXC}$ frequency. The 82586 divides the Transmit Clock internally. The Receive Bit machine is clocked by $\overline{TXC}$. During Internal Loopback, NRZ data Encoding Mode is used regardless of the data encoding configuration bit. There is no limit to the number of data bytes looped back.

Ethernet Serial Interface and Transceiver diagnostics may be performed by configuring the 82586 to External Loopback. External Loopback Mode is performed at full link speed. Therefore, to avoid receive overruns, the frame length should be limited to a value that is a function of several parameters such as link speed, Preamble length and transceiver cable length. A frame of 18 bytes, including address type and CRC bytes, is guaranteed by chip design, not to cause overrun. See section 12.5.

In external loopback the 82586 can receive a frame from another station provided the 82586 detects carrier before it starts executing the Transmit command associated with loopback. The 82586 behaves as in the normal case of a pending transmit while a frame is being received.

The 82586 will only be able to receive a frame transmitted to itself (i.e. destination address = source address, or set to Promiscuous mode) in the loopback mode. Address Checking and Minimum Frame Size Checking are always performed by the 82586, even in loopback mode.

The Intel 82501, Ethernet Serial Interface, has a Loopback ($\overline{LPBCK}$) pin which if activated disconnects the 82C501 from the Transceiver Link and transmitted data is fed back to the 82586. This mode enables diagnosing the 82C501 without the Transceiver.

11.6 Interframe Spacing Timer

At the end of a transmission the Interframe Spacing Timer is triggered by the **later** of two events:

a. The last bit has been transmitted, or

b. Carrier has dropped.

This rule applies regardless of whether the entire frame was transmitted or the transmission was aborted due to a collision and the jam sent. In the latter case, the 82586 is sensitive to other stations not having completed their transmissions even after the 82586 has completed its jam; the 82586 defers until the channel goes 'not busy' and then sets the IFS timer.

This rule applies whether or not the 82586 is configured to expect carrier sense while transmitting; also, whether or not it is configured for internal collision detection as discussed in section 11.3.

12.0 CONFIGURATION PARAMETERS

The 82586 provides a high degree of flexibility via programmable parameters. This section summarizes the configuration parameters that may be modified (using the CONFIGURE command). Refer to section 8.4 for a summary of default settings.

12.1 Framing Parameters

PREAMBLE LENGTH (2 BITS)

Determines the length, in bytes, of the Preamble (including the SFD field).

00	2 bytes
01	4 bytes
10	8 bytes
11	16 bytes

For IEEE 802.3, these bits should be programmed to 10B (8 bytes). Preamble lengths other than 64 bits may

be tailored to particular networks. The length of the preamble is based on several factors, but is primarily determined by the maximum number of transceivers in the data path and the number of bits lost at each transceiver.

ADDRESS LENGTH (3 BITS)

Determines the length, in bytes, of the address referred to by the 82586. This parameter applies to Individual, Source, Destination, Multicast, or Broadcast Addresses. An address length of 7 is treated as zero.

NOTE:
The Individual Address is set using the IA-SETUP command. Multicast addresses are set using the MC-SETUP command.

BROADCAST DISABLE (1 BIT)

Disables reception of frames with a Broadcast destination address or multicast addresses of 'all ones.' The Promiscuous Mode setting overrides the Broadcast disable.

0 Broadcast enabled

1 Broadcast disabled

CRC-16/CRC-32 (1 BIT)

Specifies which CRC polynomial is used for CRC generation and checking.

0 32-bit Autodin - II CRC

1 16-bit CCITT CRC

The 32-bit Autodin-II polynomial is $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$.

The 16-bit CCITT polynomial is $X^{16} + X^{12} + X^5 + 1$.

NO CRC INSERTION (1 BIT)

Specifies whether or not the CRC is inserted after the information field during transmission. This capability allows CRC generation external to the 82586, enabling the user to verify the CRC checking mechanism during reception. The 82586 always checks for a valid CRC (CRC check cannot be disabled).

0 CRC is inserted

1 CRC is not inserted

BITSTUFFING/EOC (1 BIT)

Specifies the frame delineation method.

0 End of Carrier Framing (IEEE 802.3 compatible)

1 Bitstuffing Framing (HDLC like)

In the bitstuffing mode, the 82586 checks for the entire HDLC flag pattern (01111110), 7EH, before accepting it as a valid flag. Only one flag is allowed following the preamble bits. Two consecutive flags will be interpreted as a beginning and a closing flag to the 82586, i.e. a frame with no data.

The Bitstuffing/Flag technique offers high reliability, especially for networks susceptible to line ringing.

PADDING (1 BIT)

If padding is set, those frames shorter than the Slot Time will be padded with HDLC flags to the shortest frame that is longer than Slot Time (during transmission). Valid for Bitstuffing only. In End of Carrier framing, padding must be provided by user software.

0 Frame not padded

1 Frame padded (only in Bitstuffing)

MIN-FRAME-LENGTH (8 BITS)

Specifies the minimum frame size, in bytes. No frame that is shorter than the minimum will be accepted by the 82586. The 82586 does not accumulate statistics on short frames, and discards them if not configured to Save Bad Frames. NOTE: apart from this mechanism, there are other limitations on the minimum frame length:

First, frames that are shorter than 6 bytes (even in Save Bad Frame Mode, Promiscuous Mode, Address Length of Zero) are discarded. No status is reported on such received frames.

Second, for AL-LOC = 0 (when Address/Length Location implies data separated from control), also frames shorter than 2 × ADDR-LEN + 2 (not including the Frame Check Sequence) are discarded.

For the IEEE 802.3 specification, this parameter should be set to 64. Transmission time of the 64 bytes ensure collision detection.

In slower or short topology networks, a shorter minimum frame size may be desirable to reduce channel overhead.

12.2 Link Management Parameters

INTERFRAME SPACING (8 BITS)

Specifies the time period (in TCLK units) that the 82586 must defer after detecting that Carrier Sense is inactive. The minimum value is 32, any value less than that defaults to 32.

SLOT TIME (11 BITS)

Specifies Slot Time for the Network (in TCLK units). Zero is interpreted as 2048. This value is used in calculating Backoff and Linear Priority delays. It must be longer than the maximum round trip time of a frame in the network plus jam time.

For IEEE 802.3, Slot Time should be set at 200H corresponding to 51.2 μs. However, it may be programmed to any number between 1 and 211.

The user may change the Slot Time to optimize the network to specific application environments. For many applications, like serial backplanes, this number will be significantly smaller than for IEEE 802.3. Setting the number to 7FFH allows the 82586 to operate over distances 8 times longer than specified in the IEEE 802.3 specification.

NUMBER OF RETRIES (4 BITS)

Specifies the maximum number of transmission retries (after a collision) that the 82586 performs before transmission is aborted by the 82586.

LINEAR PRIORITY (3 BITS)

Specifies the number of Slot Times periods the 82586 waits after Interframe Spacing or Backoff, before enabling transmission. A high number indicates low priority. Stations configured to zero, the highest priority, are equivalent to IEEE 802.3.

ACCELERATED CONTENTION RESOLUTION, ACR (3 BITS)

This parameter is added to the exponential number from which the random number is drawn for Backoff. In essence, ACR increases the range of random numbers to quickly resolve the case of stations contending for access to the network.

Specifically, let:

ACR - be the ACR priority number

N - be the number of collisions

r - be the random number multiplier of the slot time. r is a random number between 0 and 2 exp (min. [N + ACR, 10]).

ACR = 0, is equivalent to the IEEE 802.3 exponential backoff delay.

EXPONENTIAL BACKOFF METHOD (1 BIT)

This parameter determines when to start the back off time out:

0 Immediately after jamming or concurrent with Interframe Spacing. According to the IEEE 802.3 specification.

1 After deferring period expires (short topology).

This capability prevents inefficiencies and throughput loss in short topology or low data rate networks where the Interframe Spacing time may be longer than the slot time.

If the IEEE 802.3 backoff algorithm were applied to short topology networks where slot time is much smaller than Interframe Spacing, the 82586 would retry over and over again until the sum of the slot times exceeded the Interframe Spacing time, these retrys would waste channel bandwidth. See Figures 29A and 29B.

Linear Priority and Accelerated Contention Resolution can be combined in the short topology network environment. See Figure 30 (Alternate Backoff Method).

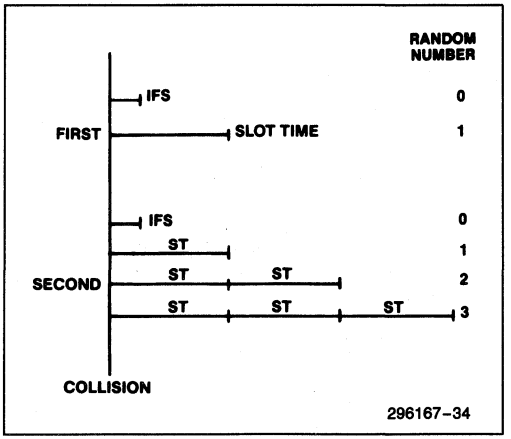


Figure 29A. IEEE 802.3 Retry Algorithm, Slot Time (ST) Greater Than IFS

12.3 Serial Interface Parameters

MANCHESTER/$\overline{\text{NRZ}}$ (1 BIT)

Specifies the bit encoding scheme used during transmission.

0 NRZ encoding (binary)

1 Manchester encoding

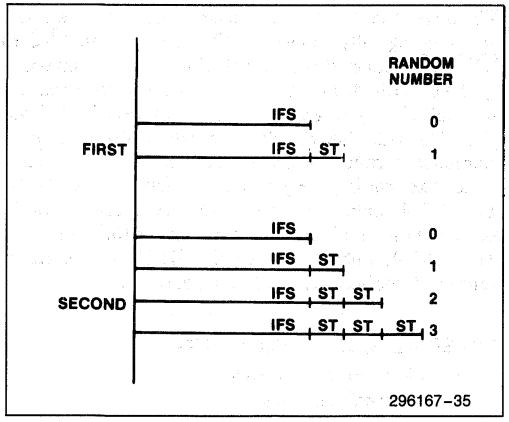


Figure 29B. Exponential Back Off Method = 1, Slot Time (ST) Less Than IFS

In the Manchester mode the 82586 requires external Receive clock recovery logic from the Receive data.

INTERNAL CRS (1 BIT)

Specifies whether Carrier Sense is generated internally or externally. External Carrier Sense is fed through the $\overline{CRS}$ pin. If set, presence of the Receive clock is interpreted as Carrier Sense active. Absence of the Receive clock means Carrier Sense inactive. This capability is useful in point-to-point transceiverless networks. See section 11.2.

0 External

1 Internal

CRS-FILTER (3 BITS)

Specifies the required width of the Carrier Sense signal (in TCLK units), before it is recognized as active. Carrier Sense deactivation is recognized immediately, i.e. only negative going glitches are filtered. This capability is useful in noisy cable environments. See section 11.2.

INTERNAL CDT (1 BIT)

Determines whether Collision Detect is generated internally or externally. External Collision Detect is fed through the CDT pin. Internal Collision Detect interprets presence of Carrier Sense during transmission as a collision. If internal Carrier Sense is used with internal Collision Detect, presence of the Receive clock during transmission will be interpreted as a collision.

0 External Collision Detect

1 Internal Collision Detect

See section 11.3.

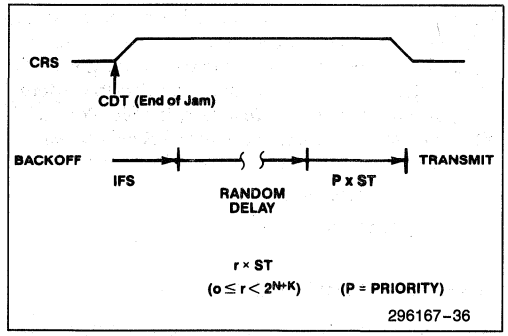


Figure 30. Combined Linear Priority and Accelerated Contention Resolution Using Alternate Backoff Method

CDT-FILTER (3 BITS)

Specifies the required width of the external Collision Detect signal (in TCLK units), before the 82586 recognizes that a collision occurred. Only negative going glitches are filtered.

TRANSMIT ON NO CRS (1 BIT)

Determines whether transmission, even if no Carrier Sense is returned from the Transceiver, is allowed. This option is used with Transceivers that do not return Carrier Sense during transmission.

0 Stop transmission if Carrier Sense drops

1 Transmit on no Carrier Sense

12.4 Host Interface Parameters

FIFO-THRESHOLD (4 BITS)

Specifies the point in the FIFO at which the 82586 requests the bus to transfer data to/from its internal FIFO from/to memory.

The FIFO limit is dependent upon the worst-case time from bus request to bus grant and serial channel and system clock rates. See section 10.4.

SRDY/$\overline{ARDY}$ (1 BIT)

Selects between Synchronous and Asynchronous Ready function of the SRDY/ARDY pin (active HI). If Asynchronous Ready is selected (SRDY/$\overline{ARDY}$ bit = 0), the SRDY/ARDY signal is synchronized internally by the 82586. If Synchronous Ready is selected (SRDY/$\overline{ARDY}$ bit = 1), the SRDY/ARDY signal is assumed to have been synchronized externally.

When using internal synchronization of the ready signal, one additional wait state may be added to each cycle, thus reducing the performance at most by 20%. Full performance systems must implement external ready synchronization. In that case, the SRDY/$\overline{\text{ARDY}}$ bit will be set to one and the external Ready signal will be connected to the SRDY/ARDY input pin.

0 Asynchronous Ready

1 Synchronous Ready

SAVE BAD FRAME (1 BIT)

Determines whether erroneous frames (CRC error, Alignment error, etc.) are to be discarded or saved. Erroneous frames are those where the OK bit in the Frame Descriptor status field is equal to zero. All frames are saved regardless of length.

In Save Bad Frame mode, the Frame Descriptor, as well as the Receive Buffer Descriptors and Receive Buffers are NOT reused for the next frame. If Save Bad Frames is not enabled, all the descriptors and buffers used for bad frames will be reused, thus, not leaving any information about the lost frame except for updating the SCB statistical tallies.

0 Discard erroneous frames

1 Save erroneous frames

ADDRESS/LENGTH FIELD LOCATION (1 BIT)

Specifies the location of the Address and Length fields in the memory structures, and whether the Source Address is inserted during transmission.

0 Address and Length Fields are located in consecutive bytes in the Frame Descriptor. Source address is inserted by the 82586 during transmission.

1 The whole frame is located in the data buffers. Source Address is not inserted by the 82586.

This capability is useful in address/control schemes that the 82586 cannot manipulate. It is also helpful for diagnostics.

12.5 Network Management Parameters

INTERNAL LOOPBACK (1 BIT)

Specifies whether frames are internally looped back or not.

0 No internal loopback

1 Internal loopback

When set, the 82586 disconnects itself from the serial wire and logically connects TXD to RXD and $\overline{\text{TXC}}$ to $\overline{\text{RXC}}$. $\overline{\text{TXC}}$ must still be supplied by the user. Internally, $\overline{\text{TXC}}$ is divided by 4. This slows down the serial bit rate sufficiently to enable 82586 operation in full duplex. This will alter the effective values of all configure command parameters that are defined in terms of $\overline{\text{TxC}}$. Note that this is purely Internal Loopback capability. The INT-Loopback bit overrides the EXT-Loopback, i.e. having an INT-Loopback bit set, at the same time with EXT-Loopback, causes the 82586 to operate in Internal Loopback Mode. See section 11.5.

EXTERNAL LOOPBACK (1 BIT)

0 No external loopback

1 External loopback

The 82586 will receive and transmit simultaneously, at full rate, a frame limited to 18 bytes (including the Frame Check Sequence). This allows checking of external hardware as well as the serial link to the transceiver. For IEEE 802.3 transceivers, since the transmitted data is fed back via the receive pair, practically nothing has to be done to perform External Loopback. For other transceiver types, the user is responsible for external transmit-receive interconnection. See section 11.5.

NOTE:
Internal Loopback bit overrides External Loopback bit.

PROMISCUOUS MODE (1 BIT)

Determines whether or not the 82586 accepts all frames regardless of their Destination Address.

0 Normal address filtering

1 Promiscuous mode

When Promiscuous mode is set, the optional broadcast disable is overridden.

13.0 INTERNAL ARCHITECTURE

The 82586 is divided into three functional modules: Host Interface Module, Channel Interface Module, and FIFO Module. The Host Interface Module communicates with the host CPU and shared memory via the Bus Interface. It performs direct memory access, buffer chaining, and interpretation of high level commands.

The Channel Interface Module communicates with the Network via the Network Interface. It performs network related activities: framing, link management, address filtering, data encoding, and network management.

The two units inter-communicate via the FIFO Module that consists of two 16-byte FIFOs.

A block diagram of the 82586 is shown in Figure 31. The three modules are separated by dashed lines.

13.1 The Host Interface Module

The Host Interface Module appears on the right side of Figure 31. It consists of separate units for data, address, and control interfacing to the local bus, all under control of two micro-sequencers (for executing commands and receiving frames, respectively), a micro-instruction ROM, an Arithmetic Logic Unit, and a register file.

Communication between units of the Host Interface Module is by means of an internal 16-bit bus (P-bus), controlled by the two micro-sequencers: the Command and Receive Units. Both sequencers operate on micro-instructions from a common ROM; each is dedicated to a separate class of tasks, and has its own program counter and stack. Only one of them can run at any given moment.

The Command Unit fetches commands from shared memory space, and controls other units of the Host Interface Module as necessary for executing the commands. It also receives status signals, processes them, and updates shared memory as required. The Command Unit controls the DMA machine, loads starting pointers and byte counts for DMA transfers, triggers the start of transfers, and aborts them if necessary. It uses the Arithmetic Logic Unit (ALU) as needed. Its commands are sent to the Channel Interface Module via the Transmit FIFO. The Command Unit responds to the Channel Attention (CA) signal from the host CPU, and manages the initialization process.

The Receive Unit fetches, from shared memory, information defining buffer availability, size, and location, and controls data transfer to the buffers. In executing these tasks, it performs similar functions as described above for the Command Unit, with the exception of initialization and response to Channel Attention.

The Micro-Instruction ROM supplies micro-program to the Command and Receive Units. It contains a thousand words, 20-bit long. The Arithmetic Logic Unit is used by the micro sequencer to perform simple arithmetic and logical operations. The Register File consists of twenty four 16-bit registers, plus an additional 48 flags. The Register File makes up an internal data storage facility for the Micro-Machine.

The DMA Unit is a memory address generator which operates on starting addresses and byte counts supplied by the Micro-sequencers to transfer information between the 82586 and shared memory. The unit is composed of four channels. Two channels are designed for block transfers, one each for transmission and reception; each of these includes a 14-bit byte counter and a

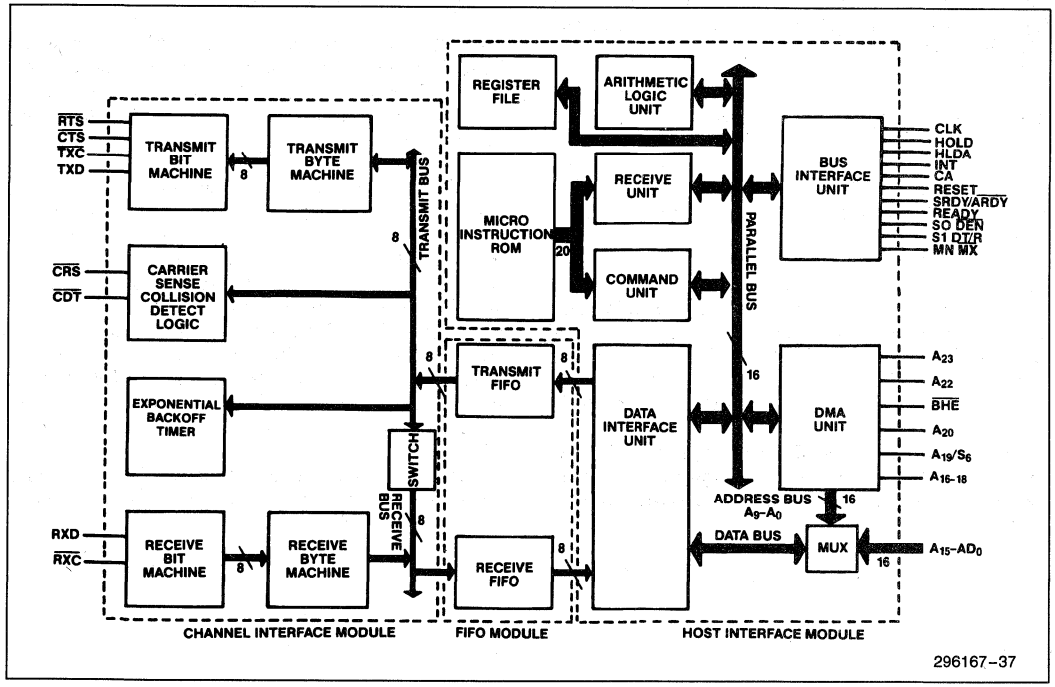


Figure 31. Block Diagram of the 82586

24-bit absolute address register. The remaining two channels are for single word transfers, one for each of the micro sequencers; each includes a 16-bit address offset register. The two channels share a common 24-bit base address register.

The sixteen lower order lines of the generated addresses are multiplexed with the data lines in Word mode.

All control signals between the 82586 and the local bus are generated or received by the Bus Interface Unit (BIU). The BIU also handles the CA line, used by the host CPU for the 82586 attention calls, and the INT line, for the 82586 to request the attention of the host. CLK is a clock supplied by the host, and is used for all Host Interface Module and FIFOs timing. Signals that control the local bus are also defined in terms of CLK. These signals are optimized for the iAPX 186, 8086, and 8088 bus.

The Data Interface Unit is a switching matrix which routes data and status signals to the appropriate destinations by interconnecting input and output as required. Command data are routed from the P-bus to the Transmit FIFO; transmit and receive status signals are sent from the Receive FIFO to the external bus; and data to be transmitted are routed from the external bus to the Transmit FIFO. Note that the two FIFOs are 8 bits wide, while the P-bus width is 16 bits. Data packing and unpacking are performed in the Data Interface Unit as required.

13.2 The Channel Interface Module

This is the second portion of the 82586, shown on the left side in Figure 31. It contains separate units that perform transmission and reception over the Network Interface.

The Transmission logic is composed of three units: Transmit Byte Machine, Transmit Bit Machine and Carrier Sense/Collision Detect Logic. The Transmit Byte Machine interprets commands from the Transmit FIFO, executes them, and generates appropriate status signals, which are returned through the Receive FIFO.

During transmission, the Transmit-Byte-Machine assembles data frames, calculates and appends the Frame Check Sequence, and passes the frames to the Transmit Bit Machine. The latter converts the frame into a pulse train and transmits it to the Network Interface. The Transmit Bit Machine performs Bitstuffing (if configured to do so).

The Exponential Backoff timer implements Backoff, Defer, Wait and Priority algorithms and other 82586 timers.

The Receive logic consists of the Receive Byte Machine and Receive Bit Machine. When a frame is received from the Network, the Receive Bit Machine strips the Preamble and SFD Field from the frame, and determines the end of the received frame by the occurrence of either EOF flag or End of Carrier. If Bitstuffing is active, inserted zeros are deleted, and computes and verifies the CRC (Cyclic Redundancy Check).

The Receive-Byte-Machine checks the Destination Address of the received frame to determine if the Individual Address matches the Receiver's; for Multicast transmissions, it performs the hash filter function to determine whether the frame is directed to this Station.

Received data and status signals are relayed through the Receive FIFO to the Host Interface Module.

13.3 The FIFO Module

The 82586 contains two 16-byte FIFO storage arrays located between the Host Interface Module and the Channel Interface Module. One FIFO transfers data in the transmit direction and the other in the receive direction.

The FIFOs improve local bus utilization by virtue of temporary data storage on the way to or from the Network. For continuous transmission in the absence of the transmit FIFO, the local bus would have to be dedicated to frame transfer during the entire transmission and the host CPU would be unable to use the bus for other tasks. The FIFO allows the 82586 to relinquish the local bus for intermittent intervals during transmission. During these intervals data in the FIFO empties toward the Network, maintaining transmission until the bus returns to the 82586 control.

Similarly, the receive FIFO accumulates incoming bytes while the bus is otherwise occcupied, reception continues without data loss.

PROGRAMMING THE 82586

INTRODUCTION

The 82586 LAN Coprocessor handles most of the functions of the data link and physical link layers of the ISO Open Systems Interconnect model. It does this with a minimum amount of supervision by the host CPU and usually executes concurrently with the host CPU. The host CPU and the 82586 communicate through a set of shared memory control structures. The 82586 is a DMA master which allows it to operate on these control structures as needed to handle commands and manage its own buffers.

The asynchronous nature of data communications and the autonomous operation of the 82586 requires special attention to the CPU/82586 software interface. In order to simplify communication between the CPU and the 82586, a protocol has been defined that must be used by both units. This chapter discusses the algorithms that the CPU must use in communicating with 82586. Since the 82586 is expecting the CPU to follow these algorithms, the user must be sure that the CPU does so. Failure to follow the algorithms may result in system failure.

This chapter will discuss two basic issues. The first is how the 82586 can be placed into a variety of operating system environments. This topic is important because the 82586 is an intelligent peripheral component: a coprocessor. The second issue is the algorithms by which the host CPU controls the 82586, that is the details of device control. To properly use the 82586, the user must address both sets of issues.

1.0 FITTING THE 82586 INTO A SYSTEM

The first problem to be solved in using the 82586 is to define how the 82586 fits into the software environment of a host system. The 82586 is more powerful than most peripheral components and may present some unique problems fitting it into a system. The 82586 is more a coprocessor than it is a peripheral component and must be treated as such. The 82586 is a coprocessor primarily due to its buffer management capabilities as well as its ability to chain commands.

In order to understand how the 82586 might fit into a system it is helpful to consider a general model of how distributed systems software might be implemented, see Figure 1.

Figure 1 shows four distinct entities:

1) The 82586 itself

2) The 82586 handler

3) Upper Layer Communications Software (ULCS)

4) User application software.

The 82586 is simply the controller itself. Inputs from the host CPU include Commands, Command Blocks, FDs, and Receive buffers as well as Channel Attention signals. Outputs from the 82586 to the host include completed commands, received frames and interrupts.

The 82586 handler consists of software routines that actually control the 82586. These routines perform:

- Generating Channel Attentions to the 82586
- Receiving interrupts from the 82586
- Reading and writing the 82586 control structures
- Giving commands to the 82586 and determining when they are completed
- providing free (unused) buffers to the 82586 and determining when they have been filled.

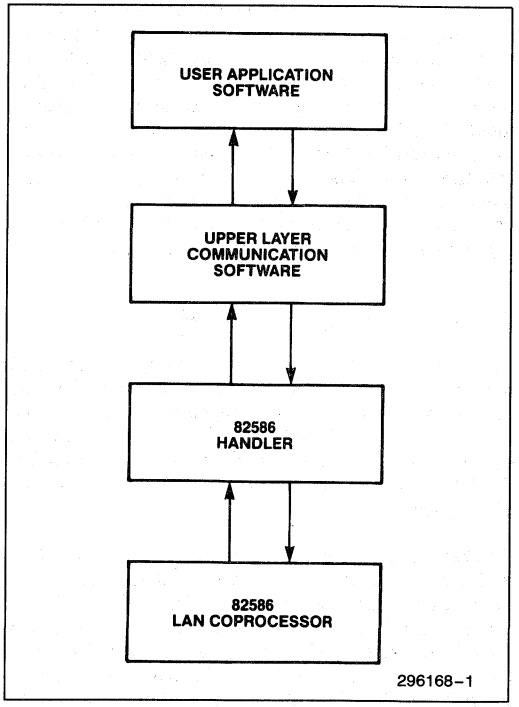


Figure 1. Distributed System Software Model

The ULCS consists of software that provides network capability for user application software. The ULCS usually consists of Transport Layer (Layer Four of the OSI model) software that provides reliable, process to process delivery service. Such software is required in one form or another because the CSMA/CD protocols supported by the 82586 (such as Ethernet and IEEE 802.3) provide only datagram or 'best effort' box to box delivery service. The Transport layer provides enhanced communication services compared to datagram based data links. These services include process-to-process message delivery. This capability is useful since there are usually several sources, or sinks, of data at a single node. Also provided is guaranteed end-to-end message service. Transport ensures that data is not lost, duplicated, or delivered out of order. The ULCS interfaces to the 82586 handler when it wishes to transmit a frame or when the handler pases a received frame to it.

The User Application Software consists of software that uses the communications channel to perform tasks in a distributed manner. This software uses the ULCS to provide reliable communication path to other processes on the network. This software is usually not aware of the presence of the 82586.

The following sections address the problem of how the 82586 handler fits into a system. This problem involves two major areas of discussion. The first concerns the nature of the interface between the 82586 handler, the host operating system and the ULCS. That is, how does the 82586 handler fit into the operating system environment. Before the handler for the 82586 can be written, there should be clear model in mind for how it will fit into the system. The second major area concerns buffer management. That is, how buffers are allocated, utilized and moved between the two entities. Understanding these issues is the topic of the following sections.

2.0 THE 82586 HANDLER

There are basically two approaches for fitting the 82586 handler into an operating system. The first has the 82586 handler appear as a standard I/O driver. The second makes the 82586 a special type of device that bypasses normal I/O conventions. In the latter model, the 82586 handler would likely be integrated into the ULCS, see Figure 2.

The basic issue in the two approaches is to what extent will the 82586 be allowed to control the rest of the system, especially the communications software, in order to allow full use of its capabilities. If the 82586 must be fitted into existing structures, it is unlikely that its full power can be utilized. Alternatively, to extract the full benefits of the 82586, the designer will have to stretch the operating system to accommodate the 82586.

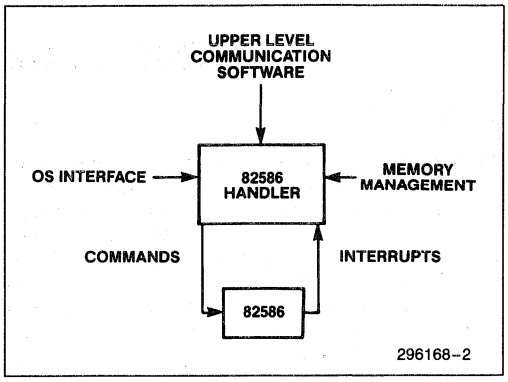


Figure 2. The 82586 Integrated Into ULCS

2.1 The 82586 Handler as a Standard Device Driver

Most modern operating systems have an I/O system that can be broken down into three parts (Figure 3). The first part consists of a standard interface used by all application programs. This interface is used by all types of devices ranging from disks to line printers. Its purpose is to make all devices look alike so that application programs can operate in a device independent manner. The second part consists of I/O service routines that handle particular types of devices. Usually two major types of devices are considered, structured devices such as disk or tape files and nonstructured devices such as TTY drivers and line printers. The third part are the device drivers which are routines that actually manipulate the hardware such as the disk controller or USART.

Device driver interfaces are usually standardized within an operating system to allow varieties of devices to be used without rewriting any of the I/O service routines. The interface is usually fairly simple because its goal is to allow as many types of devices to be used as possible.

In this type of environment the 82586 handler would fit into the system as a device driver for an unstructured device. The user would issue commands to the 82586 via the standard I/O interface which would be relayed to its device driver (Figure 4). In such an environment the ULCS would exist as an application software routine using a standard interface to the 82586 handler.

There are a number of advantages using this approach. First, it may be the only way to fit the handler into the system. If the system has memory protection or virtual memory, being part of the I/O system may be the only

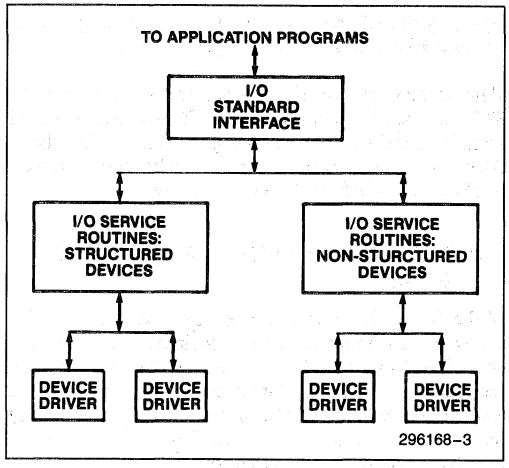


Figure 3. Common I/O System Structure

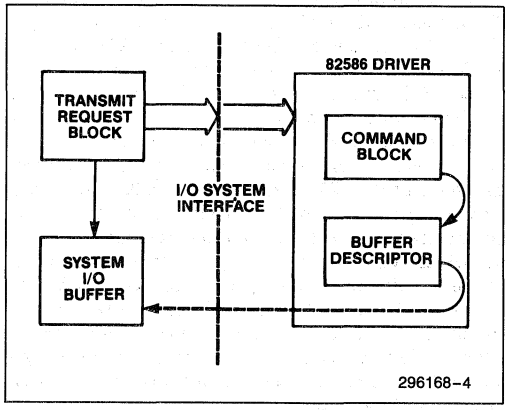


Figure 4. The 82586 Handler as a Standard Device Driver

way the handler can get access to user memory or actual physical memory (rather than virtual memory). Second, there would be a reasonable degree of device independence. The ULCS could be written independent of the 82586 and so other data links could be inserted with minimum changes to the ULCS. Third, the 82586 handler would not have to reinvent the mechanisms already used by all device drivers, such as interrupt handling, buffer management or fault handling.

However, this approach is likely to prevent many of the high level function capabilities of the 82586 from being used. The standard I/O interface and the device driver interfaces are likely to prevent use of buffer and command block chaining. As noted above, these standard interfaces are usually fairly general, and especially at the device driver level, assume rather simple devices (i.e. devices that perform one task at a time, do not manage their own memory and require constant attention from the CPU). In addition, the buffers that are passed across such interfaces are usually assumed to be one large contiguous block of memory. It could be fairly difficult trying to fit the 82586's linked list mechanisms into such a model. In some operating systems, only one command at a time can be issued across the interface. Often such an interface is not very good for asynchronous events (increased probability of losing frames). In addition, the performance across such an interface may be slow. Despite these problems, undoubtedly there are many systems in which this is the best approach. In such situations there are a number of mechanisms that allow many of the features of the 82586 to be used.

Looking first at command block handling, a common model of operation would be one in which an operating system command block with something like a WRITE command contained inside it would be passed to the device driver. There would also be a pointer in this command block to an area of memory containing data to be 'written' (a System I/O Buffer). The 82586 device driver could interpret this WRITE command as being a TRANSMIT command with the data being the message to be sent. Inside the device driver would be an 82586 Command Block (CB) and a Transmit Buffer Descriptor (TBD). When the driver receives the operating system request block it will set up its internal CB to the desired function and its TBD to point to the System I/O Buffer. The driver then gives the CB and TBD to the 82586 for execution. When the 82586 has completed processing, the Driver returns completed operating system command block back to the operating system. The internal CB and TBD are then freed.

There are a variety of variations on this technique but all are based upon an internal CB/TBD within the device driver being used to pass the command to the 82586. For example, one might consider passing the 82586 command block and message as part of the data to 'written'. In this case the contents of the provided 82586 Command Block would be copied to the internal CB and the TBD set up to the message part of the data. By having an internal CB and TBD the need for them to be in the same 64K byte segment as FDs, RBDs, and the SCB can be easily met.

Turning to receive handling there are a variety of options. The most simple approach is to reserve a FD, RBD, and single buffer (long enough to contain the biggest message that will be received) within the device driver. When the 82586 receives a frame, it fills the buffer and passes it back to the driver. The operating system would recover the frame by issuing a READ with a System I/O Buffer. The received frame would be copied into the System I/O Buffer and the operating system READ command returned. While very simple

this approach suffers from possible poor performance due to lost messages. Lost messages are caused by the single receive buffer within the device driver being freed only when the operating system posts a receive buffer. There is also the cost of doing the copy into the System I/O Buffer.

A variation on this technique that addresses the copy problem is to move the data directly into the System I/O Buffer by using the buffers supplied by the READ command. As in the command block case, internal FDs and RBDs would be used as control structures. With either approach a frame can get lost if the System I/O Buffer is not supplied with new buffers frequently.

An alternative approach is to reserve a circular list of FDs, RBDs within the device driver, see Figure 5. The 82586 places received frames into the buffers. When the operating system issues a READ command, the 82586 driver copies this information into the supplied System I/O Buffer. The device driver reclaims the old FD, RBD, and buffers.

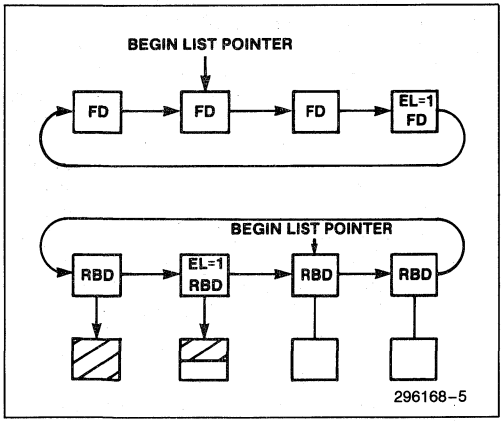


Figure 5. 82586 Device Driver Using Circular Lists

The circular list is managed using a FIFO philosophy. Frames are always processed in the order they were received, and processed FDs and RBDs are replaced in order. The links among the FDs on the RDL and the RBDs on the FBL are never altered. The EL bit is used to mark the logical end of the list. The 82586 itself manages the linking FDs to RBDs as they are needed. If the 82586 runs out of FDs or RBDs, it enters the No Resources state. In order to know the logical beginning of the lists, the devide driver will maintain a pointer to the 'first' free FD and RBD. As frames are received The driver will always know which FD is to be used next. As FDs and RBDs move from the empty to filled state, the pointers should be moved to the next FD and RBD. Recovering FDs and RBDs simply involves changing the EL bit from one to zero on the previous FD and RBDs to the one being recovered.

This circular list approach takes better advantage of the 82586's capabilities in that the 82586 can receive frames asynchronously to the operating system reading them. Also, the 82586's buffer chaining capability can be used to obtain more efficient memory usage.

2.2 The 82586 Handler as a Special Driver

If the 82586 handler is treated as a standard device driver, device independence is maintained, but at the cost of performance. Performance is degraded for two reasons. First, copies of data must be made between the System I/O Buffer and the driver. Second, frames may be lost due to a lack of buffers. This suggests that if possible, another approach be found.

This other approach consists of moving the 82586 handler outside of the I/O system and be treated as a special kind of device. In this model, the 82586 handler would exist as a separate entity, most likely made part of the ULCS routines, see Figure 6. The only support required from the operating system would be interrupt handling. The ULCS would likely operate on linked buffer structures and perhaps be aware of the control structures of the 82586. The 82586 handler would still manipulate the control blocks given it by the ULCS and interface to the 82586, but operate much closer to the ULCS. In this environement more attention must be paid to the buffer management model used by the 82586 and the ULCS. The 82586 can operate with a variety of buffer models in this type of environment although there is one that can be considered its 'design model.' This model and several others will be discussed in the following section.

The 'Design' memory management model for the 82586 is shown in Figure 7. This model assumes an explicit Transport (and perhaps Network) Layer. Two pools of memory, one for Transmit (called the CB pool), and the other for Receive (called the RFA pool) contain various control blocks needed for 82586 operation. The CB pool contains CBs, TBDs, and transmit buffers. The RFA pool contains FDs, RBDs, and receiver buffers. Blocks do not move between pools. These data structures are 'owned' by the 82586 handler. Usually these pools are independent of the operating system's Free Space Manager (it is possible for them to be part of a system free space manager, but this will significantly increase overhead). Both pools are 'owned' by the 82586 handler or by the 82586 itself. The CB pool belongs to the handler while the RFA pool belongs to the 82586 itself. The RFA pool is basically the memory pool managed by the 82586 Receive Unit.

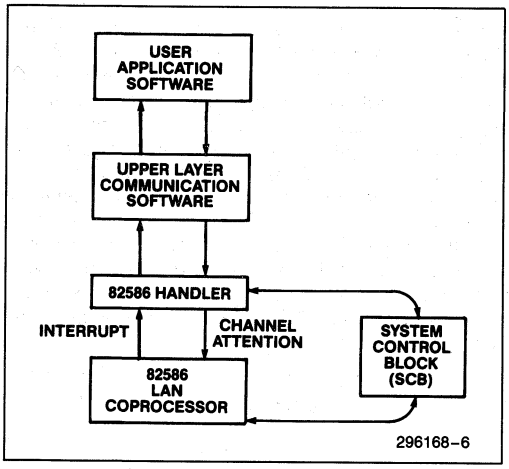


Figure 6. The 82586 Handler as Part of the Communications Software

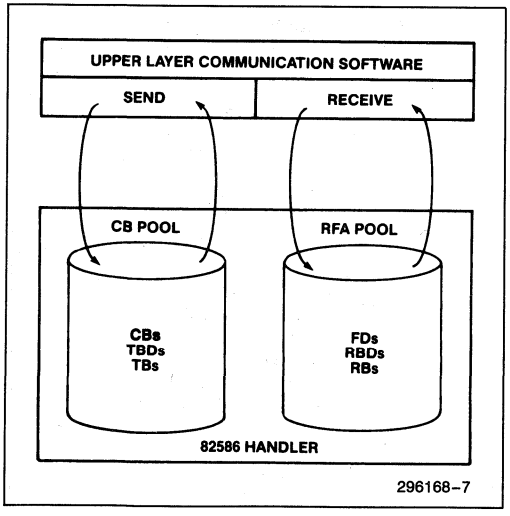


Figure 7. 82586 Handler Memory Management Model

To transmit a Frame, the Transport Layer requests a buffer from the 82586 handler. This might be done by making a subroutine call to the handler. The handler returns a command block and enough buffers (with TBDs) from the CB pool to the Transport Layer. The Transport Layer will fill the buffer with some portion of the user message and initialize the CB to whatever values are required for the operation desired. The CB (and buffer) are then returned to the handler where it is given to the 82586. When the command is executed by the 82586, the handler will receive an interrupt and will reclaim the CB and TBD and places them back into the pool.

On the receive side, the RFA pool is managed by the 82586 ifself. When a frame has been received, the 82586 interrupts the handler. The handler passes the FD, RBD(s), and receive buffer(s) to the receive side of the Transport Layer. The Transport Layer extracts what it needs, and returns the FD, RBD(s), and receive buffers(s) to the handler. The handler reclaims these structures and places them at the end of the Receive Frame Area managed by the 82586.

As noted above there are a variety of memory management models that the 82586 can operate with besides the 'design' model. Another model of interest eliminates the copy of data from user buffers into transmit data link buffers and from receive data link buffers to user buffers. One approach to eliminate copies of data is to eliminate the transmit buffers from the CB pool and only provide CBs and TBDs. The Transport Layer software then sets up pointers to header information and user data using the TBDs. On receive, the Transport Layer passes on the data link buffers to the user and returns only the FDs and RBDs (plus new receive buffers to replace those passed to the user).

This technique eliminates copies of data, it does have some problems. First, both transmit and receive buffers must be in 'fast' memory (that is, memory that can be accessed directly by the 82586 at a minimum of 1.25 Mbytes/sec.). Second, the Transport header may not be long enough to meet the minimum buffer size required for the 82586. On receive this problem manifests itself as to how to handle the transport header information; it cannot be passed up to the user. Third, care must be taken to ensure that the memory used in data link buffers is accessible by the user program. If user programs are restricted to only some memory areas, then the situation may arise in which the user program cannot access a received buffer. In this case a copy would be required.

As noted above the 82586 was designed to work with one buffer management model. The 82586 does operate with other models provided the algorithms in the following sections are followed. **It is important to recognize that there must be a buffer management model present in the system. It should be a model that is chosen upfront in the system design, not after.**

3.0 INITIALIZATION OF THE 82586

Initialization of the 82586 occurs in these three phases. The first phase involves use of the SCP and the ISCP to locate the SCB and define the bus width. The second phase involves issuing Configure, Individual Address Setup and Multicast Address Setup commands. The third phase involves starting the RU by supplying it with an initial allocation of FDs and RBDs. This section addresses only the first two phases; section 6.0 addresses starting the RU. It should be noted that it is

important that three phases be done in the order indicated. Starting the RU prior to setting up the individual address is generally undesireable.

The SCP and ISCP are designed for maximum flexibility in locating the SCB. Usually in iAPX 86 family systems, the SCP is located in ROM together with the processor bootstrap routines. The model of operation is that the SCP will point to the ISCP which is located in RAM. The address of the SCB can be written into this RAM location so that it can vary with different system configurations. The SCP and ISCP will exist only for initialization purposes. Once the SCB is located, the SCP and ISCP locations can be reclaimed. The SCB should be initialized prior to beginning the SCP/ISCP sequence.

The second phase involves issuing commands that configure the 82586. If the 82586 is used in IEEE 802.3 configuration, it is not necessary to issue a configure command since the 82586 initializes itself as an IEEE 802.3 controller. Otherwise the user must issue the Configure command. After configuration, the user should issue an Individual Address Setup command to load the 82586 with the host address. This procedure may be followed by the MC Setup command as required. These commands may be chained together or issued one at a time.

After address setup, the RU can be started and supplied with FDs and RBDs, see section 6.4.

4.0 SIMPLE COMMAND PROCESSING

With an understanding of how the 82586 handler fits into a system, the operations to be performed by the handler can be discussed. This section discusses giving commands to the 82586 and servicing 82586 interrupts. Simple command processing in this context means that the 82586 handler accepts only one command at a time for the 82586. ULCS is not allowed to give a second command to the handler before the previous command has been executed. More complex processing will be discussed in section 5.0.

4.1 Adding CBs to the CBL

Simple command processing means that commands are issued one at a time to the 82856. The handler is given a single CB by the ULCS, writes the address of the CB into the CBL_OFFSET field in the SCB, places a Start CU command into the SCB, and issues a CA. The 82586 will accept the Start CU command and clear the SCB field. It will then execute the CB, update status in the SCB and generate an interrupt to the handler. The handler will process the interrupt and then reclaim the CB for future use. This procedure can be repeated as required.

Figure 8 illustrates this basic procedure of giving a new CB to the 82586. **Note that the CPU must mask off interrupts from the 82586 during various parts of the algorithm.** Masking interrupts will prevent the interrupt service routine from issuing commands to the 82586 which might overwrite the command just given to the 82586 by the command processing routine. Overwriting might occur if an interrupt occurred between the recognition that the SCB command field was clear and the issuing of the CA. Either the command processing routine or the interrupt service routine could find their command to the 82586 being overwritten by the other. The result of this race condition would likely be a system hang up.

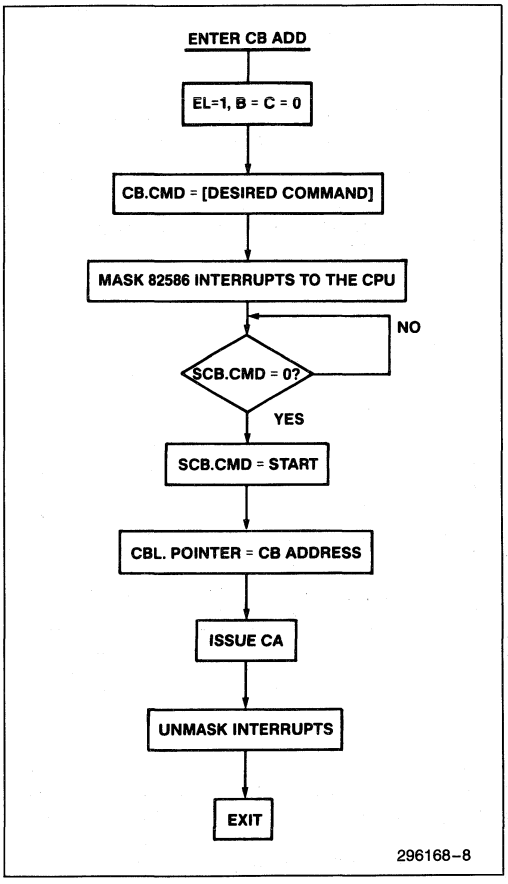


Figure 8. Simple Procedure to Add a New CB to the CBL

4.2 Basic Interrupt Service Routine

When a command has been completed by the 82586, an interrupt will be issued to the host CPU (if the I bit was

set in the (CB). Figure 9 shows the basic interrupt service routine. This routine's interrupt acknowledgement sequence will be used over again in the command and receive interrupt service routines that will be described later. The routine starts by saving CPU context (this may have already been done by the operating system). The handler then waits for the SCB command word to clear, which signifies that the 82586 has no pending control commands. The handler determines the nature of the interrupt, sets the appropriate ACK bits in the SCB Command Word, and issues a Channel Attention (CA) to mark acknowledgement of the interrupt. The handler then waits for the 82586 to accept confirmation of the interrupt acknowledgement by clearing the SCB command field and removing the interrupt signal (provided no new events that may generate an interrupt have occurred). The CB is then examined by checking the C bit (command completed bit). If completed, the CB is returned to the handler for re-use. If the command was not completed then some error is likely to have occurred. The routine is exited by issuing an End of Interrupt, EOI, to the interrupt controller and restoring interrupt context.

The basic algorithm is referred to by all the following sections. The only part to change will be the parts that refer to CB processing.

5.0 ADVANCED COMMAND PROCESSING

This section primarily is concerned with how commands may be chained together. In practice, it is difficult to chain two or more commands together for two reasons. First, the 82586 can execute commands much quicker than most operating systems can prepare the next command. For example, it takes less than 1 ms to transmit 1000 bytes at 10 Mbps for IEEE 802.3. Most operating systems would be hard pressed to prepare another command before the first was completed. Second, chained commands must be given to the 82586 in groups. If a single command is given to the 82586 and the CU is started, the 82586 will not execute the added CBs. The new CBs are ignored because the 82586 reads the first CB's EL bit set to one and concluded that there were no more Command Blocks to be executed. This behavior will occur even if the new Command Block is chained to the first and the EL bit on the first is reset; once the 82586 has started working on the first CB it does not reexamine the EL bit. Thus if one wishes to chain commands it will be necessary to ensure that the 82586 has at least one unfetched Command Block present. **This rule requires that the 82586 be started with at least two Command Blocks present.**

Despite these problems there are situations in which some form of Command Block chaining might be useful. The first case is where the ULCS may have the capability to occasionally give the 82586 handler multi-

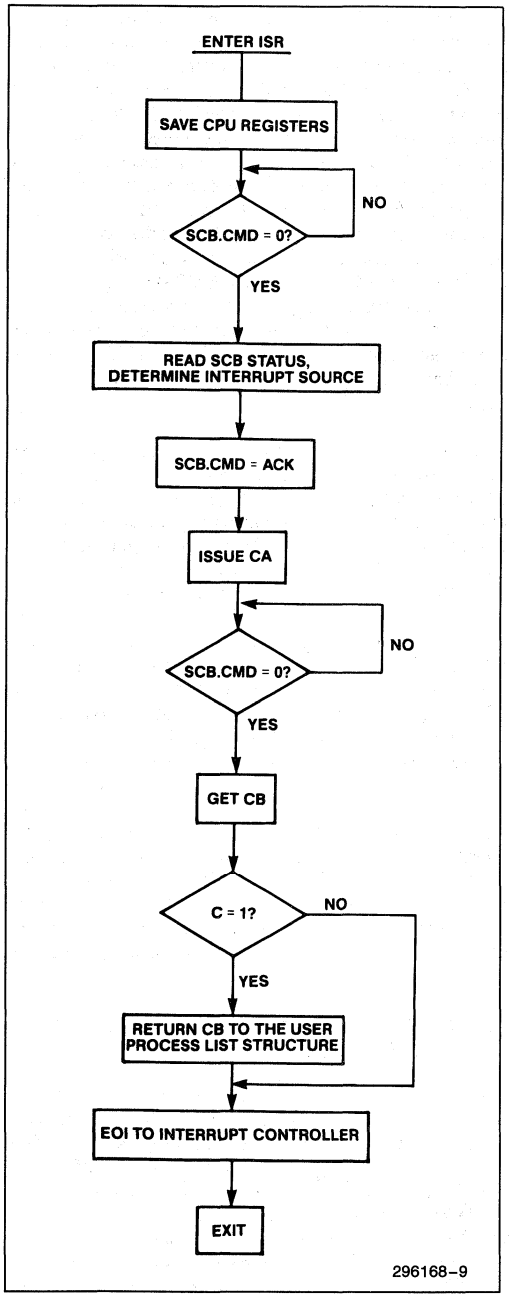


Figure 9. Basic Interrupt Service Routine Sequence

ple Command Blocks (not at the same time) and some structure is desired to handle it. In this case the commands are most likely to be executed one at a time by the 82586 simply because they are given to the handler one at a time. This case is called **static lists**. In this approach Command Blocks are chained together on a list but only one Command Block is actually given to the 82586 at a time. Thus the 82586 will execute one Command Block, halt (the CU goes to the IDLE state), and interrupt the host CPU. The handler would then process the completed Command Block and, if another one is on the list, give it to the 82586 for execution. If there are no commands, the CU is not restarted. Given how fast most CPUs can actually prepare request blocks and the manner in which the 82586 operates upon lists, static lists are probably the best overall approach.

The second approach is similar to the first except that the 82586 can be given multiple commands to execute. This case, called **dynamic lists**, occurs only if there are two or more unexecuted commands on the Command List and the CU is in the IDLE state (requiring it to be STARTed). If there is only one command on the list (or the 82586 is on the last Command Block of a list of commands) then trying to add additional commands yields nothing. Dynamic lists are an attempt to take advantage of the fact that the 82586 executes one command at a time and does not look beyond the current CB being executed (except for the implications of setting the EL bit). Most of the time, dynamic lists will function as static lists, but occasionally an interrupt may be saved.

This section discusses both static and dynamic lists. They share a very similar method of adding Command Blocks to the end of the CBL. The major difference is in the interrupt service routine. As a result, the algorithm for chaining commands will cover both cases, and interrupt processing will be discussed separately.

5.1 Adding Command Blocks to Static and Dynamic Lists

To add a CB to the list, the ULCS will most likely call a procedure that does it. A procedure for CB chaining for static and dynamic lists is shown in Figure 10. The handler defines two pointers: Begin.CBL and End.CBL. Begin.CBL locates the first CB on the list while End.CBL locates the last CB. If there are no CBs on the list then Begin.CBL is set to 0FFFFH. The pointers bound the list of CBs as defined by the CPU. The 82586 will operate on all or some part of the CBs on the list.

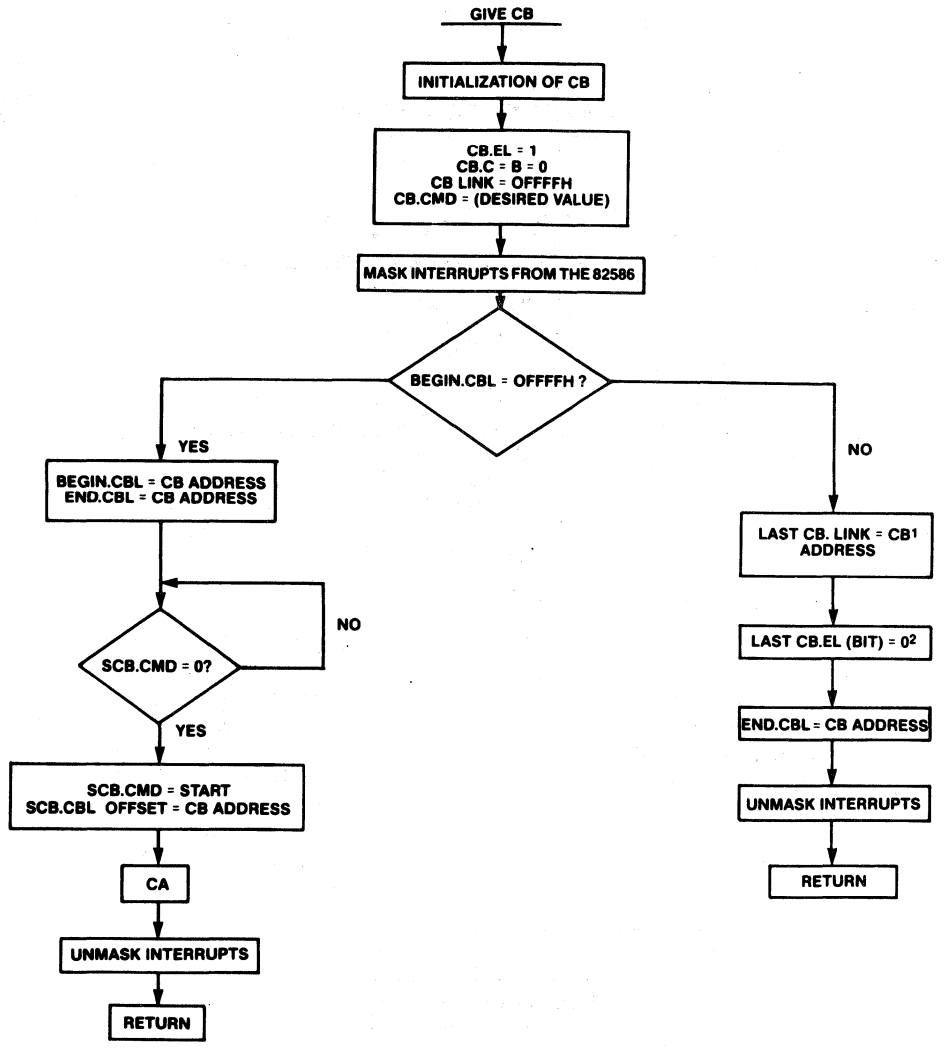
The procedure shown in Figure 10 first clears the CB's C and B bits, sets the I and EL bits to one, and sets the Link field to 0FFFFH. It will then examine Begin.CBL. If Begin.CBL = 0FFFFH, there are no CBs on the list and the process proceeds as in the simple case (Figure 8). Begin.CBL and End.CBL are both set to the address of the CB. If Begin.CBL is not 0FFFFH, then there are CBs on the list to which the new CB is to be added. First, interrupts from the 82586 are masked off to prevent race conditions between interrupt service routine and CB chaining procedure. The new CB is added by inserting its address in the current last CB's Link field, and the END.CBL pointer is updated. For dynamic lists, the EL bit of the formerly last CB on the list is set to zero. **It is important that the EL bit is not set to zero prior to the Link field being valid.** Otherwise, the 82586 may go off to an invalid address and never return. For static lists, the EL bit is set for all CBs.

5.2 Static List Interrupts

When a single command has been completed, the 82586 will interrupt the host CPU. The interrupt service routine, ISR, for static lists, shown in Figure 11, builds upon the basic interrupt acknowledgement routine described in Figure 9. After the basic routine is completed, two sanity checks are performed. First, that there is a Command Block present (Begin.CBL is not equal to 0FFFFH), and second, that the Command Block was executed (the C bit is one). If either is not true, then a serious error has occurred. If there is a completed CB present, then it is returned to the user. A search is made for the next CB by checking the Link field of the current CB. If it is 0FFFFH, then there are no more CBs. Begin.CBL should be set to 0FFFFH in this case and the routine exited. Otherwise SCB.CBL_OFFSET and Begin.CBL are set equal to the address of the next CB, START is issued, and the routine is exited. As in the simple CB add case, the last action before leaving the ISR is to issue an EOI to the programmable interrupt controller, PIC.

5.3 Dynamic List Interrupts

For dynamic lists, the 82586 receives an interrupt after processing a CB with EL = 1. Figure 12 illustrates the interrupt service routine. First, the basic interrupt acknowledgement routine is used from Figure 9. A pointer called CB.pointer is defined to track the CB of immediate interest. CB.pointer is initially set equal to Begin.CBL. It is then tested for the empty condition (0FFFFH), and if empty the routine is exited. If not, the CB itself (pointed to by the CB.pointer) is obtained and the C bit of the CB is examined for completion of execution. If set, the CB.pointer is updated with the location of the next CB (i.e. the CB Link field of the used CB) and the used CB is returned to the handler. The process repeats until the end of the CBL is found. If the C bit was not set, the CU is checked for the IDLE state, and is restarted if required. In either case, EOI is executed, and the procedure is exited.



NOTES:
1. Placement of this statement is critical. For example, if EL is set to zero before CB Link is valid, the 82586 could go to an invalid CB address.
2. For static lists, EL = 1, and this line is not required.

Figure 10. Procedure to Chain CBs: Process One CB at a Time (Static List), or Multiple CBs (Dynamic List)

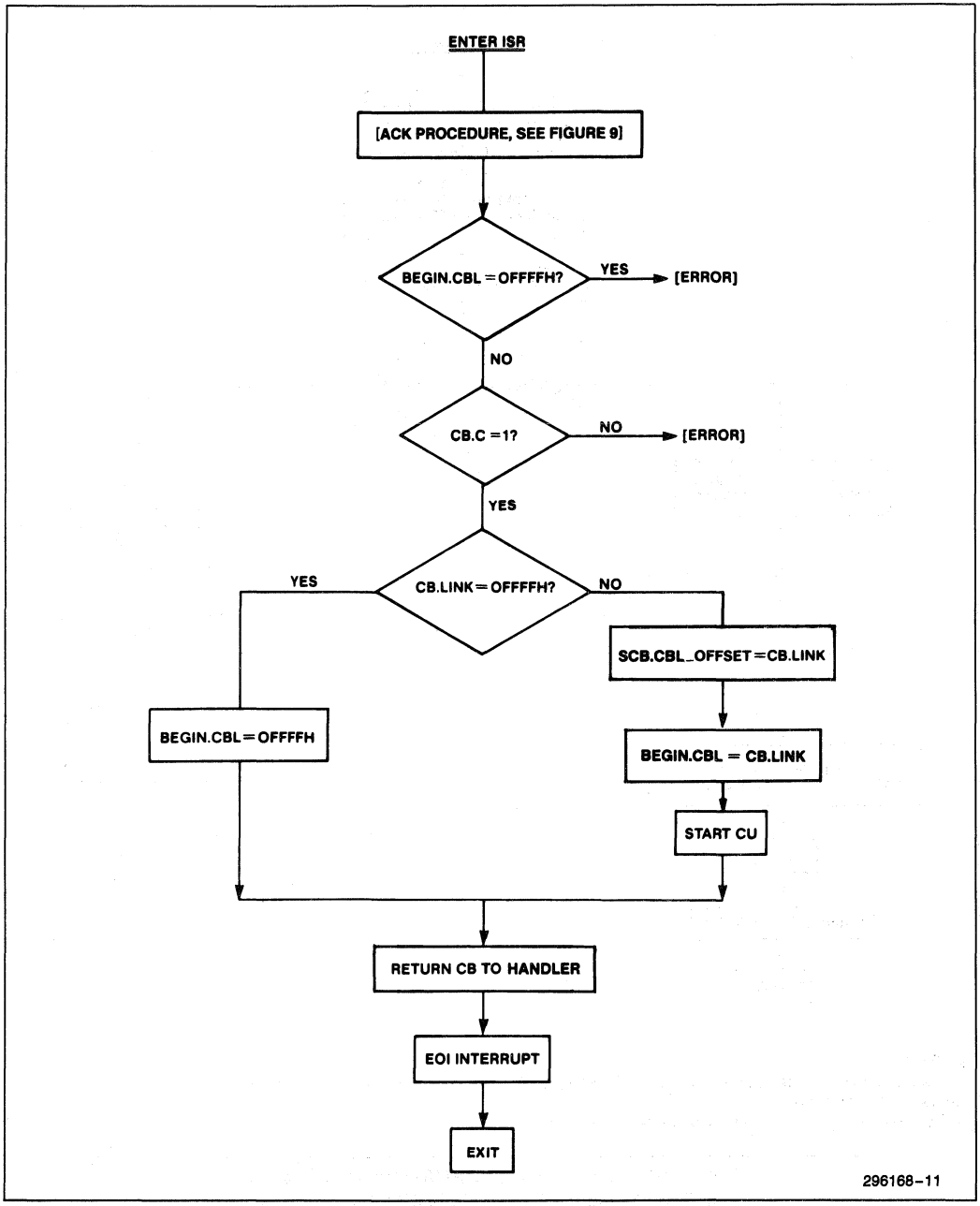


Figure 11. Interrupt Service Procedure for Static Lists (EL Bit Set in all CBs)

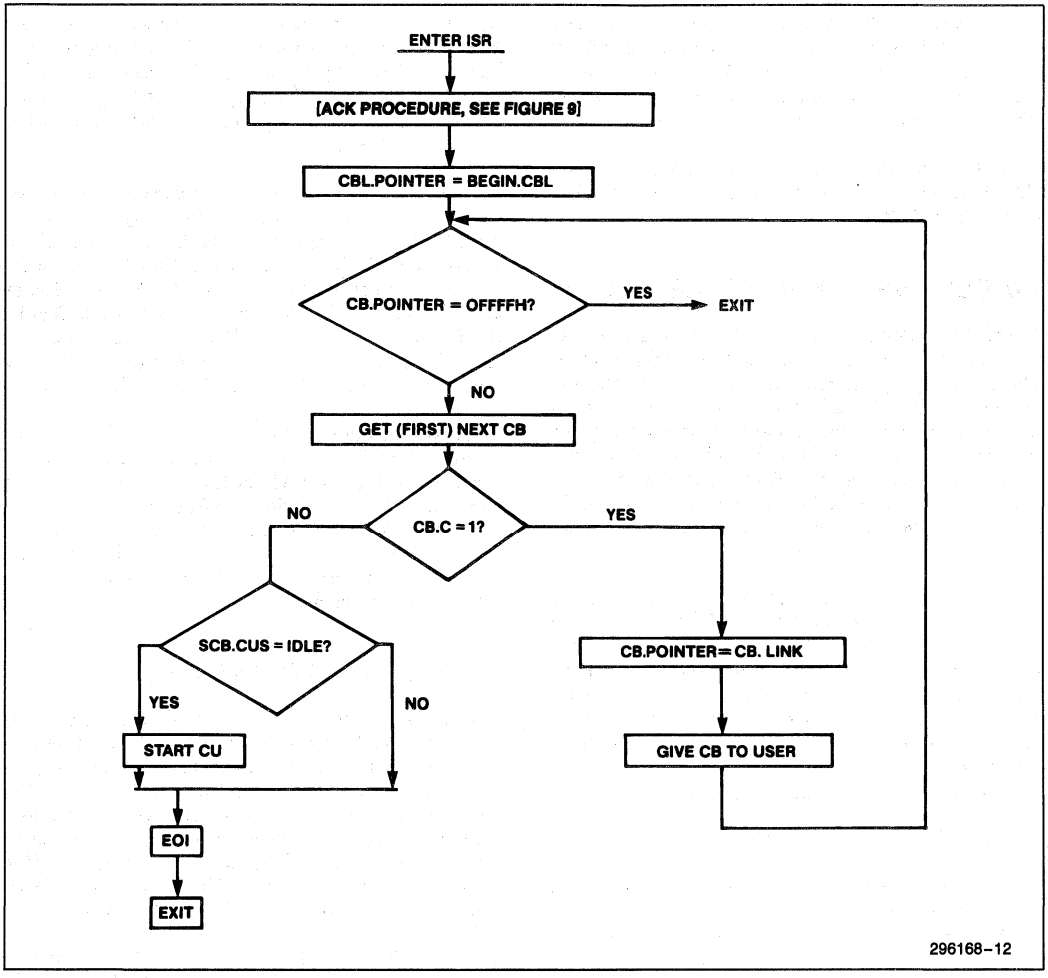


Figure 12. Dynamic List Interrupt Service Routine

The major difference between static and dynamic list interrupt processing is that in the static list case there will always be exactly one completed Command Block whenever SCB.CX is one. With dynamic lists there may be zero, one, or serveral completed commands and the user must always determine which case applies. In most systems, it is not worth the additional overhead trying to handle the processing of dynamic lists when it is unlikely that multiple Command Blocks will ever be found on the Command List anyway.

5.4 CU Command Simplification

From the discussion above, it was shown that CBs could be added to the CBL without using the SUSPEND and RESUME commands.

By adhering to the approach set forth above, the programmer can eliminate the complexity of managing 2 additional control commands. No performance penalty is paid because from the software execution viewpoint, all CU commands are equivalent.

The approach works because when the CU recognizes an EL = 1, it goes directly to the IDLE state without reading the Link field. This rule allows the handler to specify location 0FFFFH as the empty condition, which is helpful in managing CBs.

It will be shown in the next section that RESUME and SUSPEND are not required to append FDs and RBDs to the Receive Data and Free Buffer lists.

6.0 RECEIVE FRAME PROCESSING

The 82586's automatic receive buffer chaining management capability is one of its most powerful features. This capability affords efficient memory usage which in turn can result in fewer lost frames due to lack of buffers. In order to gain the greatest advantage from this capability, the software must be able to dynamically handle interrupts reclaim and add FDs/RBDs to the Receive Frame Area, RFA. 'Dynamically' here means that the CPU need not halt or suspend the RU in order to add either FDs or RBDs.

The RFA can be divided into two lists: the Received Data List, RDL, and the Free Buffer List, FBL (see Figure 13). The RDL is the list of free FDs while the FBL is the list of free RBDs. The basic technique to make additions to the RDL and FBL is similar to adding CBs as described for dynamic lists in section 5.0. As before, use of the RU commands SUSPEND and RESUME is not required.

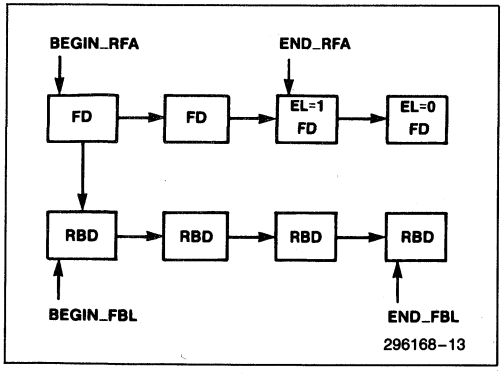


Figure 13. The Receive Frame Area

In order to operate on the RDL and FBL, a pair of pointers is needed for each list. The pointers BEGIN_RFA and END_RFA are used to indicate the beginning and end of the RFA. The pointers BEGIN_FBL and END_FBL are used to indicate the beginning and end of the FBL. **It should be noted that 'beginning' and 'end' here refer to how the CPU views the lists, not the 82586.** As in the case of the chained Command Blocks, the lists that the CPU operates on may contain blocks that the 82586 either has already processed or does not know are present (in the sense of being past the 'last block' as viewed by the 82586). The BEGIN pointers are used by interrupt routines for searching purposes. The END pointers are used to designate a true end of the RDL and FBL.

An interesting problem occurs in receive frame processing that does not occur in CB processing, namely 'orphan Buffer Descriptors.' Consider the case of 20 RBDs, each 128 bytes long, and 3 FDs. If three short frames are received, there will be 17 'RBDs' without any FDs present (they have become 'orphans'). That is, the RBDs do not have a FD to point to them. The following algorithms will also deal with this problem by use of the BEGIN_FBL and END_FBL pointers.

Before presenting the actual algorithms, it is helpful to recall how the RU itself operates on the RBL and FBL. First, the RU never modifies links between individual FDs and between individual RBDs. Second, the RU updates the RBD_OFFSET field in the FD to point to the next free RBD. Third, the RU only reads the RFA_OFFSET field in the SCB at START; the 82586 never modifies this field.

6.1 Supplying FDs to the RDL

The first task of concern is how to place free FDs onto the RDL. Free FDs may be placed on the RDL in two instances. First, at startup time when the 82586 is provided with its initial pool of FDs. Second, when the ULCS is done with a received frame and is returning the FDs and RBDs to the handler. A likely method to give FDs to the handler is to call a subroutine. This routine can be called 'Supply FD.' Supply_FD places new FDs to the end of the RDL, see Figure 14.

Supply_FD first initializes the new FD by setting EL = 1 and LINK_OFFSET and RBD_OFFSET = 0FFFFH. Setting LINK_OFFSET to 0FFFFH indicates that the FD is the last block on the RFA insofar as the CPU is concerned (the 82586 uses EL). **Initializing the RBD OFFSET is critical to avoid system hangups.** If no prefetched RBDs exist when the 82586 is setting up the new FD, the 82586 will read the RBD-OFFSET in the FD expecting to find a pointer to a new RBD. If the RBD-OFFSET = 0FFFFh, then the 82586 assumes that no RBD exists for this FD. If the RBD-OFFSET ≠ 0FFFFh, then the 82586 assumes that a valid RBD exists at the address specified by the RBD-OFFSET.

A check is then made to determine if the RFA is empty (i.e. BEGIN.RFA = 0FFFFH). If it is, the BEGIN and END RFA pointers are set to the address of the FD. The RU is not started because there is only one FD available to the 82586 (see section 6.4). The routine returns back to the handler.

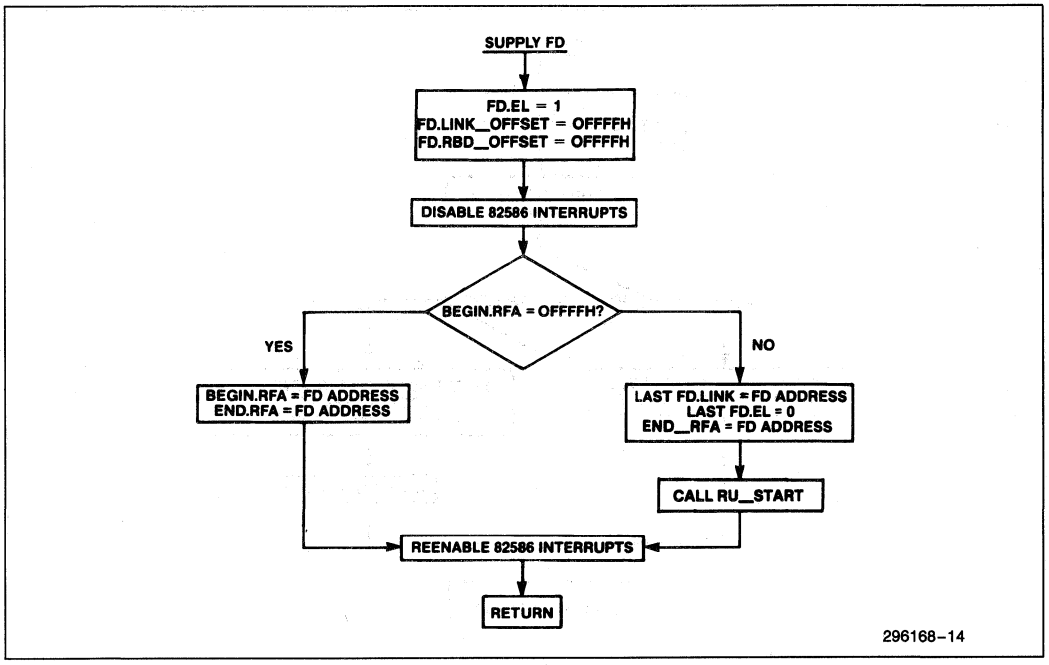


Figure 14. Procedure to Add FDs to the RDL

If the RFA is not empty, the FD is appended to the end of the list. First, interrupts from the 82586 are masked off to prevent an interrupt service routine from modifying the RDL. The last FD.LINK OFFSET (i.e. the FD that was the last one on the list prior to the subroutine call) is updated to the address of the FD to be added to the list. The EL bit of this last FD is set to 0, and the End RFA pointer is updated to point to the 'new' last FD. To prevent system failure, the link to the new FD must be established before the EL bit in the 'old' last FD is set to zero. Finally, interrupts from the 82586 are unmasked, and the RU may be restarted as described in section 6.4.

6.2 Supplying RBDs to the FBL

The process of adding RBDs to the FBL is very similar to that for adding FDs to the RDL. As with the FDs there is likely to be a routine called 'Supply_RBD'. Supply_RBD places new RBDs at the end of the FBL (see Figure 15).

Supply_RBD first initializes the new RBD by setting EL = 1 and LINK OFFSET. Setting LINK_OFFSET to 0FFFFH indicates that the RBD is the last block on the FBL insofar as the CPU is concerned (the 82586 uses EL). A check is then made to determine if the FBL is empty (i.e. Begin FBL = 0FFFFH). If it is, the BEGIN and END FBL pointers are set to the address of the RBD. The RU is not started because there is only one RBD available to the 82586, (see section 6.4). The routine returns back to the handler.

If the FBL is not empty, the RBD is appended to the end of the list. First, interrupts from the 82586 are masked off to prevent an interrupt service routine from modifying the RDL. The last RBD.LINK OFFSET (i.e. the RBD that was the last one on the list prior to the subroutine call) is updated to the address of the RBD to be added to the list. The EL bit of this last RBD is set to 0, and the End RDL pointer is updated to point to the 'new' last RBD. To prevent system failure, the link to the new RBD must be established before the EL bit in the 'old' last RBD is set to zero. Finally, interrupts from the 82586 are unmasked, and the RU may be restarted as described in section 6.4.

The two routines Supply_FD and Supply_RBD were constructed to illustrate the algorithms. In most real systems, it is likely that the actual routines would differ slightly from the ones discussed. The Supply_FD routine would likely accept FDs with a list of attached RBDs. Returning linked FDs and RBDs is useful since most of the time a previously received frame consisting of one RD and at least one RBD will be returned to the handler. Likewise the Supply_RBD routine could also accept a list of RBDs instead of just one. The processing is slightly more complex but takes advantage of the already existing lists instead of first taking the lists apart and then putting them back together again.

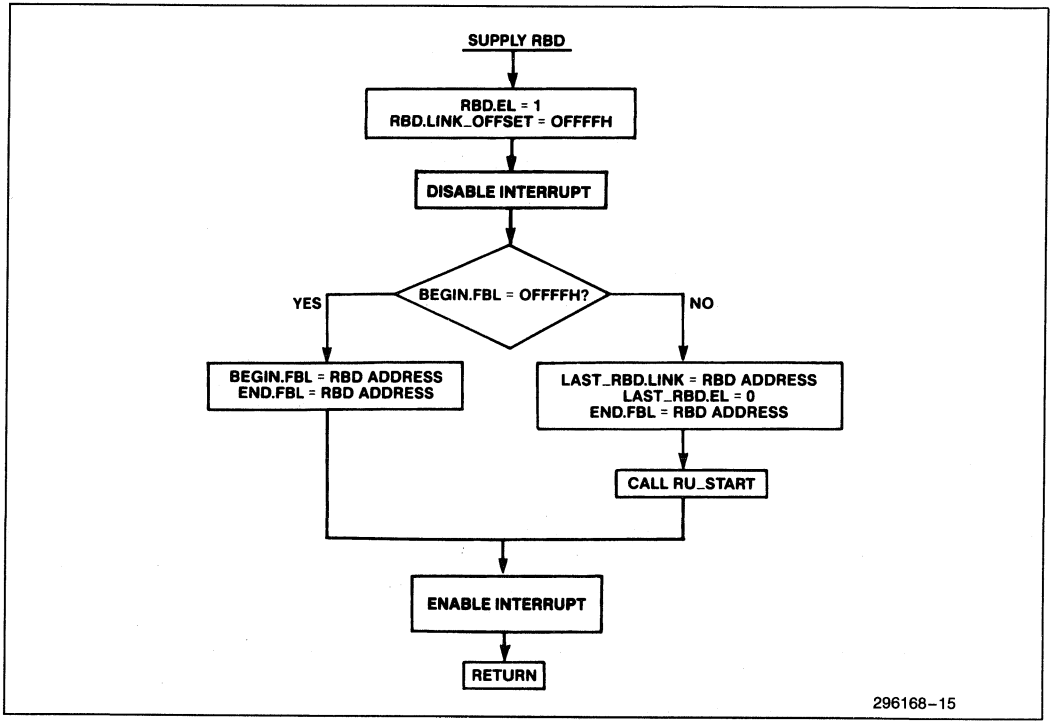


Figure 15. Procedure to Add RBDs to the FBL

6.3 Receive Interrupt Processing

The procedure for handling receive frame interrupts is shown in Figure 16. There are basically three phases to receive interrupt processing. The first phase is to acknowledge receipt of the interrupt from the 82586 (as was done in Figure 9). The second phase is to remove any received frames from the RFA and update the pointers to the RDL and FBL. The third phase is to restart the RU if it is not in the READY state. The first phase is described in section 4.2.

The second phase begins by examining the FR bit in the SCB status field. If it is set, then there may be received frames present. (It should be noted that the word 'may' is used. It is possible that the received frames had been already removed during previous interrupt service processing). To locate received frames the following algorithm is used. A Frame Descriptor pointer, FD.pointer, is defined to access FDs. FD.pointer is initially set to the location of the first known free FD (contained in Begin_RFA). This location is first tested for the empty case (0FFFFH) and if empty, processing is terminated on the FDL. If a FD is present, then it is tested to see if reception of the frame was completed (the C bit is checked). If not, the second phase is completed and processing moves on to the third phase. If the FD's C bit is set, then the location of the next FD is taken from the FD,LINK and stored in Begin_RFA. The new beginning of the FBL is located by examining the RBD pointed at by the RBD_OFFSET field of the completed FD. A search is performed by examining each RBD until the one with EOF set is found. This RBD is the last RBD used by the completed FD. The next RBD (provided that RBD does not also have EL set) is the new first RBD on the FBL. Its address should be stored in Begin_FBL. When this procedure is done, the FD and associated RBDs are forwarded to the user and FD.pointer is set to the next FD. This process continues until either the end of the RDL is found or a FD with the C bit not set is located.

The third phase is concerned with restarting the RU if required. The RU may be in either the IDLE or NO-RESOURCES state and may need to be restarted. The rules for doing this are presented in the next section, 6.4.

6.4 Rules for Starting the RU

Care must be taken when restarting the RU after adding FDs and RDBs. The following rules are helpful to ensure smooth processing:

1) It is unnecessary to restart the RU if it is already running (i.e. in the READY state).

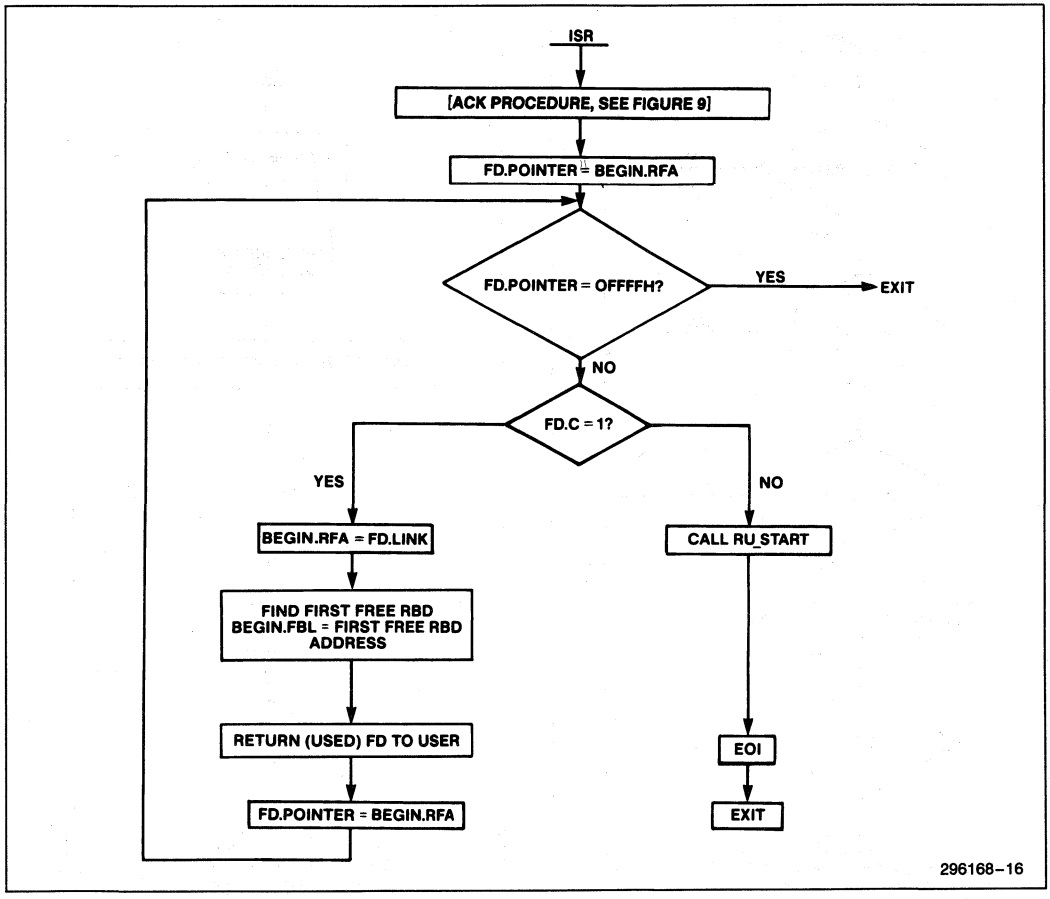


Figure 16. Receive Frame Interrupt Processing Routine

2) There should be at least 2 available FDs on the RDL. For example, if the first FD has EL = 1, then the RU will ignore all remaining FDs. In this case, the RU will go into the NO RESOURCES state after a frame is received. If two or more FDs are available, the RU will stay active because it hasn't seen EL = 1.
3) There should be at least 2 RBDs available to the RU before starting.

A procedure for starting the RU is shown in Figure 17. The procedure begins by checking the rules set forth above (it is assumed that SUSPEND is not used). The RBD_OFFSET of the first FD is set to the beginning of the FBL (using the Begin.FBL pointer). The procedure waits for the SCB command field to be zero; this check ensures that there are no pending commands to the 82586. The RFA offset in the SCB is set to location of the first FD in the list. The RU is then started. Channel Attention is given, and the routine is exited.

6.5 Considerations in Using Receive Buffers

In using the linked buffer structures present with the 82586 there are several important considerations related to how many receive buffers are required and how large should they be. Although the exact numbers will depend upon the performance and available memory in

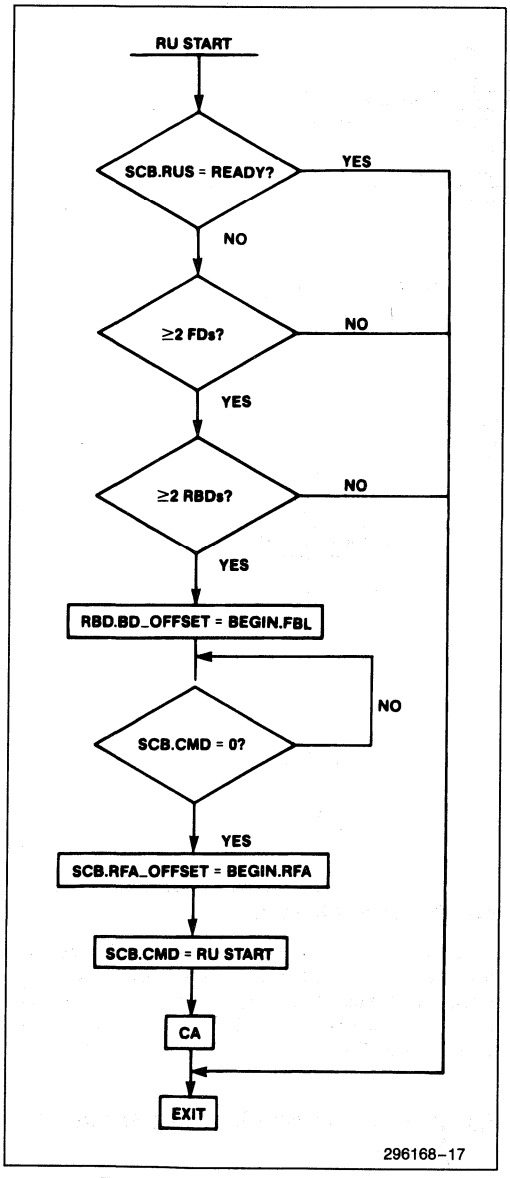


Figure 17. RU Start Procedure

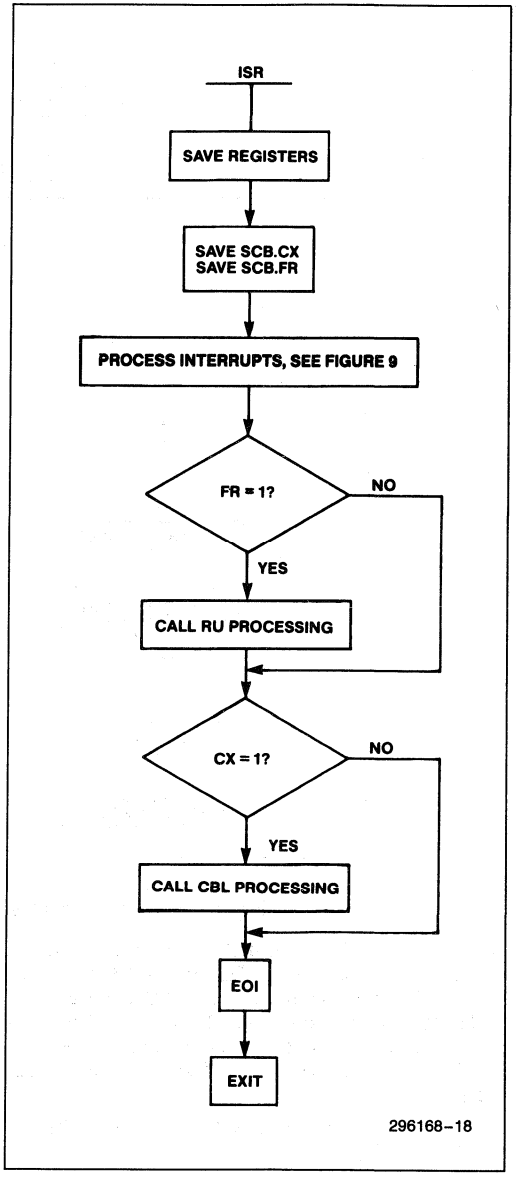


Figure 18. Combined Command and Receive Interrupt Service Routine

a particular system, there are some general guidelines that can be established. These guidelines are:

1) The size of the receive buffer should completely contain the usual 'small' message of the network. In most networks up to 75% of the frames will be 'small'. This number should be determined and the size of receive buffers set to be slightly greater than this value. In many networks, a value of 100 to 128 bytes is adequate. The worst case system bus bandwidth requirements is when every frame received takes two or more receive buffers.
2) The number of receive buffers depends on the acceptable loss rate of frames due to lack of buffers. Since the CSMA/CD protocols supported by the 82586 do not offer guaranteed delivery of frames some loss is inevitable, but if too many are lost there will be significant network performance degradation. The minimum number is enough to contain the largest possible frame. The designer must consider the following questions:
   a. How much traffic is going to be received? Obviously a file server will require many more buffers than a terminal or simple workstation.
   b. How long does it take the processor to process and return a received frame? For a given level of performance, the longer the processing time, the more buffers will be required.
   c. What is the maximum number of back to back frames that can be expected in normal operation and how big will they be? There should be enough buffers to contain them.

In trying to determine the optimum number of receive buffers, it is helpful to use the Resource Error tally in the SCB. This tally will record the number of frames lost to insufficient buffer resources. In most systems a value of five to ten percent of the total received frames is acceptable.

7.0 COMBINING RECEIVE AND COMMAND PROCESSING

The discussions in section 4.0 and 5.0 assumed that the Receive and Command interrupt processing were performed in isolation. In practice the two procedures are combined as shown in Figure 18. In the previously described routines, redundant servicing routines (Save Registers, Process Interrupts, EOI and Exists) would be performed only once in the combined case.

82586 APPLICATIONS

OVERVIEW

This chapter is a collection of brief notes related to system considerations when using the 82586 LAN Coprocessor. The chapter is based on work performed by Intel's Data Communications Application Engineering staff. This work includes computer simulations and actual debugged hardware/software designs.

1.0 MINIMUM 82586 SYSTEM BUS SPEED

82586 Bus bandwidth requirements are an important concern. Parameters that dictate 82586 system bus useage are: buffer size, FIFO-Threshold, interframe spacing, serial data rate, wait states, bus latency, and 8 or 16 bit bus width. The relationship between these parameters is complex.

A worst case analysis can be done to determine the minimum bus frequency that the 82586 must have to receive two or more back to back frames. There are **two** variables of concern. First, the ratio of the parallel bus frequency, Fp, to the serial clock frequency, Fs. The Fp/Fs ratio affects bus latency (i.e. the time it takes for the 82586 to acquire control of the bus), number of wait states (i.e. the additional CPU cycle times required between memory address to valid data received), and handling 82586 buffer/status overhead that can be tolerated before the on-chip FIFOs exceed their capacity.

Second, interframe spacing gives additional time for the 82586 to complete placing received data into memory and handle 82586 buffer/status overhead. Figure 1 shows the timing relationship and activities performed by the serial side and parallel side of the 82586.

Simulations were performed to provide **guidelines** to designers to establish minimum Fs/Fp ratios given various bus latency, wait state, and interframe spacing time conditions. The simulations assumed a system using word mode, and worst case conditions consisting of: last buffer exactly filled up, next buffer remaining empty, and prefetch of next buffer descriptor. The most recently received frame was assumed OK.

The results of these simulations are shown in Figure 2. Note that various wait state (N_W) and bus latency (N_L) environments are represented by sloped lines, according to the empirical formula $16\ N_W + N_L = X$ clock cycles.

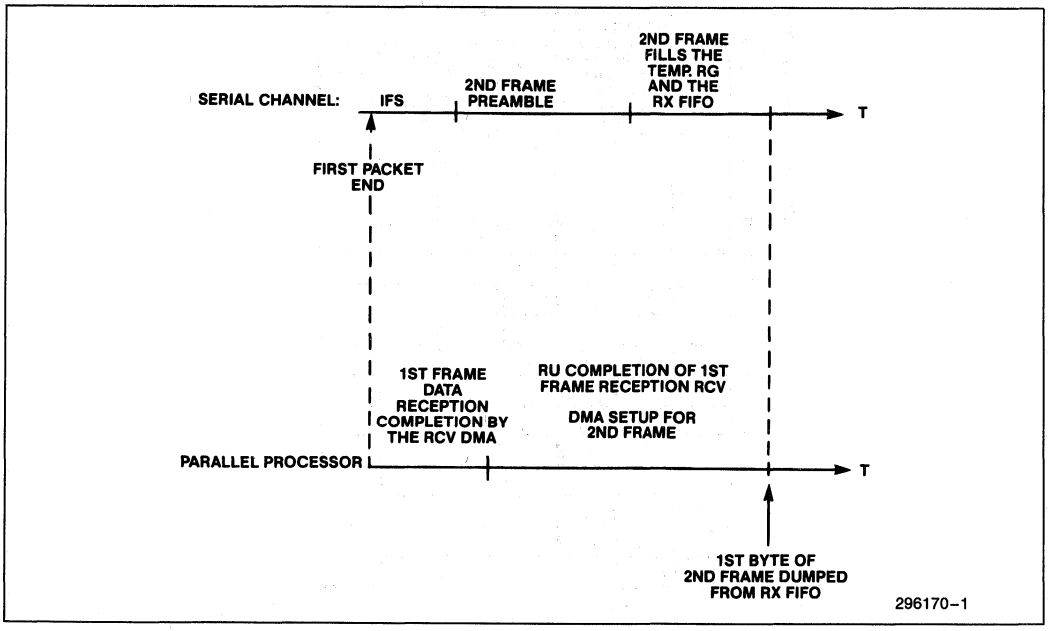


Figure 1. Receive to Receive Interframe Spacing Time Relationships

The following example illustrates use of Figure 2:
Wait states, N_W = 0 clock cycles
Bus latency, N_L = 0 clock cycles
IFS = 9.6 μs

From Table 1, the minimum Fp/Fs is 0.58. In other words, for Fs = 10 MHz, i.e. IEEE 802.3, then Fp can be as slow as 5.8 MHz.

One can verify that for IEEE 802.3, where Fp/Fs = 0.8 (i.e. Fp = 8 MHz), that the 82586 can handle the 9.6 μs interframe spacing (IFS) even when the sum of 16 times the number of wait states (N_W) plus bus latency (N_L) is equal to 80. For example, a 82586 system of 0 wait states, and bus latency of 80 clock cycles can handle back to back frames, separated by the IFS time.

Table 1 summarizes Fp for various bus latency/wait state conditions for IEEE 802.3 from Figure 2.

Table 1. Minimum Bus Frequency for IFS = 9.6μs

N_W (Clock)	N_L (Clock)	Fp (Min) MHz
0	0	5.8
0	5	5.9
1	0	6.2
1	5	6.3
2	0	6.5
2	5	6.8

2.0 SETTING THE 82586 FIFO-THRESHOLD

The 82586 features a programmable FIFO-Threshold. When the threshold value is reached, the 82586 attempts to acquire the system bus via the HOLD/HLDA protocol.

If the FIFO-Threshold is set too low, the 82586 will be constantly accessing the system bus, creating system inefficiencies because the CPU and other system peripherals must perform additional overhead duties related to relinquishing the bus. If the FIFO-Threshold is set too high, the 82586 may not acquire the bus before the FIFO becomes full (or empty), causing received (or transmit) data to be lost. Thus, an optimal threshold setting can be found based on the number of times the 82586 accesses the bus, and ensuring that received data does not get lost.

The variables of concern to setting the FIFO-Threshold are parallel bus frequency (Fp), serial clock frequency (Fs), and the bus latency time (N_L). The ratio of Fp/Fs is important because it quantifies the relationship between how quickly the FIFOs are filled, and how fast the FIFOs can be emptied. The latency time quantifies the amount of time the 82586 must wait before it can begin to empty the FIFOs.

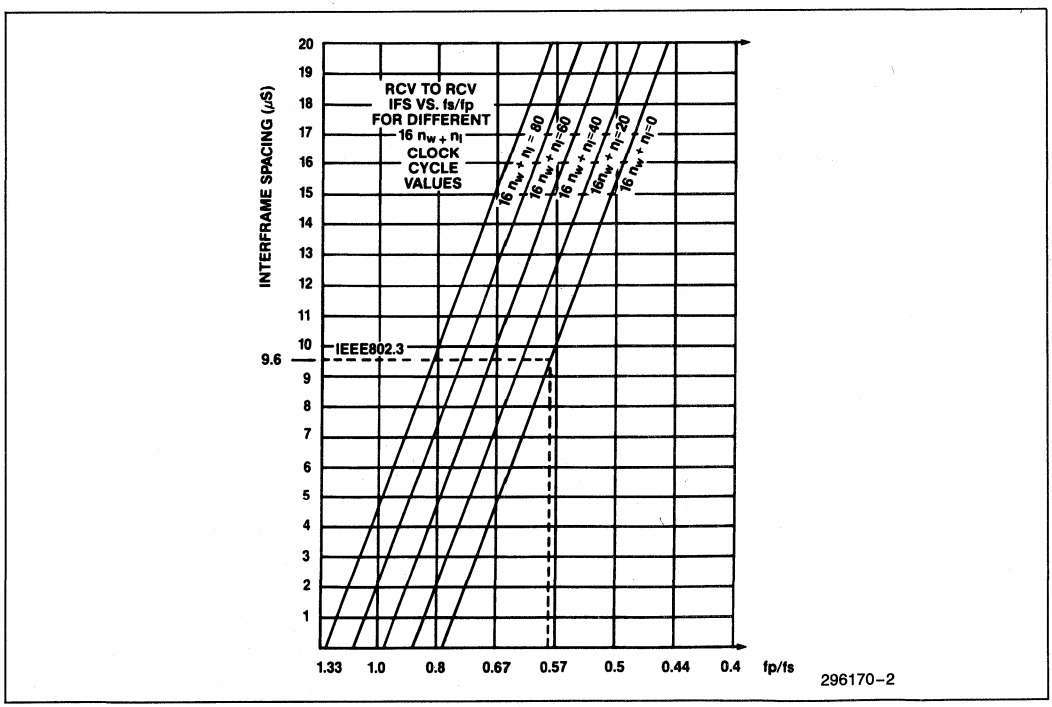


Figure 2. CPU/82586 Parallel Bus Frequency Requirements for given Wait States (N_W) and Bus Latencies (N_L)

82586 system simulations were performed to provide a starting point with which to set an optimal threshold setting. For an actual system, **the FIFO-Threshold setting should be further optimized to accommodate particular application, system and network environments.** The results of the simulations are summarized in Tables 2 and 3. Table 2, the minimum safe FIFO limit presents guidelines for threshold setting, for a given bus latency. Table 3 takes into account time required for the 82586 to perform buffer switching. The results from both tables must be added together to yield the recommended threshold limit.

Consider the following example:
Parallel Bus Frequency, Fp = 8 MHz
Serial Clock Frequency, Fs = 10 MHz
Maximum bus latency, N_L = 4 clock cycles

From Table 2 (Fp/Fs = 0.8, N_L = 4), the minimum safe FIFO limit = 1.

From Table 3 (using N_L = 5), the FIFO limit offset = 4. Taking the sum of the results from Tables 2 and 3, the FIFO-Threshold setting for preliminary evaluation should be 5.

Table 2. Minimum Safe FIFO Limit

Fp/Fs	NL	Minimum FIFO Limit
0.5	0	1
0.5	1	2
0.5	2	2
0.5	3	2
0.5	4	2
0.5	5	3
0.5	8	3
0.5	10	4
0.5	15	5
0.5	20	6
0.5	25	8
0.5	30	9
0.5	40	11
0.8	0	0
0.8	1	0
0.8	2	1
0.8	3	1
0.8	4	1
0.8	5	1
0.8	8	1
0.8	10	2
0.8	15	3
0.8	20	3
0.8	25	4
0.8	30	5
0.8	40	6

NOTES:
Fp = Parallel Bus Frequency
Fs = Serial Clock Frequency
NL = Bus Latency (in cycles)

Table 3. FIFO Limit Offset

Fp/Fs	NL	FIFO Limit Offset
0.4	0	6
0.5	0	5
0.8	0	3
0.4	5	8
0.5	5	6
0.8	5	4
0.4	10	10
0.5	10	8
0.8	10	5
0.4	20	13
0.5	20	10
0.8	20	7
0.4	40	-
0.5	40	15
0.8	40	10

3.0 THE MINIMUM BUFFER SIZE

The 82586 employs memory buffer chaining to ensure efficient use of memory.

From a network point of view, the minimum buffer size should be no less than the largest 'short' (control) frame plus a few bytes. This practice will eliminate unnecessary buffer linking. A minimum of 64 to 128 bytes is usually sufficient.

From the 82586 point of view, the upper bound of receive buffer size is a function of the network environment and system memory constraints; **the lower bound is a function of the 82586's ability to prefetch receive buffer descriptors and buffer fill time.** In particular, while the 82586 is filling up a buffer, the 82586 will prefetch the next receive buffer descriptor. Thus, the prefetch must be completed before the current buffer becomes full.

The variables affected receive buffer fill time, and consequently minimum buffer size, are the parallel bus frequency, Fp, and serial clock frequency, Fs, bus width, and wait states, N_W. Computer simulations were performed on a general system to provide designers with the minimum receive buffer size. The results are shown in Table 4.

The following example illustrates use of Table 4.

Given:

- Parallel bus frequency, Fp = 8 MHz
- Serial clock frequency, Fs, = 10 MHz
- Wait states, N_W = 1
- Bus width = 16 bits (word mode)

From Table 4, the minimum receive buffer size is 20 bytes.

Note that from Table 4 that the largest minimum is 55 bytes. Thus, it can be concluded that in general receive buffers in excess of 64 bytes are sufficient for the 82586 to handle linked lists in hardware.

Table 4. Minimum Buffer Size

	$N_W = 0$		$N_W = 1$		$N_W = 2$	
Fp/Fs	Word Mode	Byte Mode	Word Mode	Byte Mode	Word Mode	Byte Mode
0.4	51	*	55	*	42	*
0.5	47	26	51	22	42	20
0.8	36	24	34	22	30	20
1	36	24	34	22	30	20

NOTES:
*Will always underrun or overrun
Fp = Parallel Bus Frequency
Fs = Serial Clock Frequency

The minimum transmit buffer size is determined by the same factors as receive buffers plus end-of-frame-processing (updating status, pointers, BD count, and statistics; setting up the RU for the next reception; etc.). The 82586 can underrun if the values of Table 4 are used for the first transmit buffer of a frame to be transmitted. Underruns will occur when the 82586 is issued a Transmit command just prior to or during a receive operation. In this case, during reception the 82586 will fetch the first TBD and fill the Transmit FIFO. After the completed reception, the 82586 will start transmission and simultaneously complete end-of-frame-processing. End-of-frame-processing will delay the prefetch of the second TBD, and cause the underrun. Thus the minimum length of the first Transmit Buffer must be long enough so that the next TBD prefetch will occur before the first buffer is transmitted. For systems with Fp/Fs = 0.6 a first transmit buffer in excess of 75 bytes is usually sufficient to avoid underruns; for systems with Fp/Fs = 0.8, the first transmit buffer should be in excess of 54 bytes. The succeeding transmit buffers for a single frame can follow the recommendations of Table 4.

The 82586 supports simultaneous transmit and end-of-frame-processing operations to ensure that the 82586 will be able to transmit soon after a receive operation. Thereby avoiding the case where a receiving station continually gets beaten onto the network by other stations. This case is important in communication intensive applications like file servers.

4.0 SYSTEM CONFIGURATIONS

4.1 80186 Elementary Maximum Mode System

SYSTEM INTERFACE

The 82586 does not require any 'TTL glue' to interface to an 80186 microprocessor bus. Thus, it is highly recommended for minimum component count communication systems (see Figure 3). The 82586 is configured to Maximum Mode by strapping the MN/$\overline{MX}$ pin to ground; in this mode the 82586 generates status signals that will be used for bus control signal generation.

The 82586 is clocked by the CLKOUT signal generated by 80186. Thus, the existing address latches, data transceivers and bus controller can be shared by the CPU and the 82586.

The $\overline{S0}$, $\overline{S1}$ signals of the 80186 and 82586 are wired together, driving the 8288 bus controller. The 80186 and 82586 have internal pullups so no external resistors are required on the $\overline{S0}$, $\overline{S1}$, $\overline{S2}$ lines. The 8288's $\overline{S2}$ input is driven only from the 80186; this status is not generated by the 82586 because it accesses memory only (no I/O). The $\overline{S0}$, $\overline{S1}$ and $\overline{S2}$ lines are used by the 8288 to generate a full set of standard bus control signals.

The shared data bus of the 80186 and the 82586 is 16 bits wide. The system memory can be accessed either by the 80186 or by the 82586. Bus arbitration is resolved by the HOLD/HLDA protocol. The CPU grants the bus to the 82586 by issuing the HLDA high as a response to bus request from the 82586 (HOLD high). The CPU is able to withdraw its bus grant by dropping HLDA low. In this case the 82586 will release the bus within a maximum four clock cycles in word mode.

Care must be taken so that the CPU does not take away HLDA during transmission or reception because there will be a high chance of frame abortion by underrun or overrun respectively.

Communication on the task level between the CPU and the 82586 is accomplished by a shared system memory mailbox, the System Control Block, and the CA/INT handshake.

It is possible to create a multimaster system by using the 8289 bus arbiter. It is suggested that in these systems the 82586 communication node is assigned the highest priority. A high priority will ensure a minimum bus latency for the 82586 whenever it needs the bus, thus avoiding waste of bus bandwidth by underrun or overrun frames.

To realize the IEEE 802.3 standard (i.e. 10 Mbps serial data rate and 9.6 μs Interframe Spacing), it is sufficient to operate at 8 MHz system clock and zero wait state bus cycles. Also, lower rates and slower memories (that introduce wait state) are possible in IEEE 802.3 systems, see section 1.

SERIAL INTERFACE

Figure 3 displays an 80186 based Elementary Maximum Mode System, containing an Intel 82501 Ethernet Serial Interface.

4.2 Stand Alone Multibus System

SYSTEM INTERFACE

It is possible for an 82586 CPU system to share a memory interface via the Multibus as shown in Figure 4. The Multibus is the Intel's standard bus structure which allows Intel's board products to communicate. Standard address latches, data transceivers and a bus controller are needed in order to interface to a demultiplexed Multibus.

The 8289 Bus Arbiter is needed to resolve Multibus arbitration. It is recommended to assign the 82586 system the highest priority in order to efficiently handle communication tasks. The highest priority interrupt request on the Multibus, INTO, is used to inform the remote CPU that a communications task was completed.

4.3 Dual Port RAM Systems

SYSTEM INTERFACE

The 82586 operating at 10 Mbps serial bit rate, may utilize a significant percentage of bus bandwidth, thus leaving insufficient bus bandwidth for other purposes. When the 82586 bus bandwidth utilization is intolerable, a dual port memory system with one port dedicated to the 82586 is recommended. Figure 7 displays a typical dual port memory based system.

The basic building blocks for the dual port RAM system are standard dynamic RAM's and the 8207 DRAM Controller. The 8207 provides not only dynamic RAM refresh, but also arbitrates between each of the process requests and directs data to or from the appropriate port. The 8207 has a LOCK capability that enables the 82586 to lock out CPU accesses while the 82586 finishes descriptor and error tally processing.

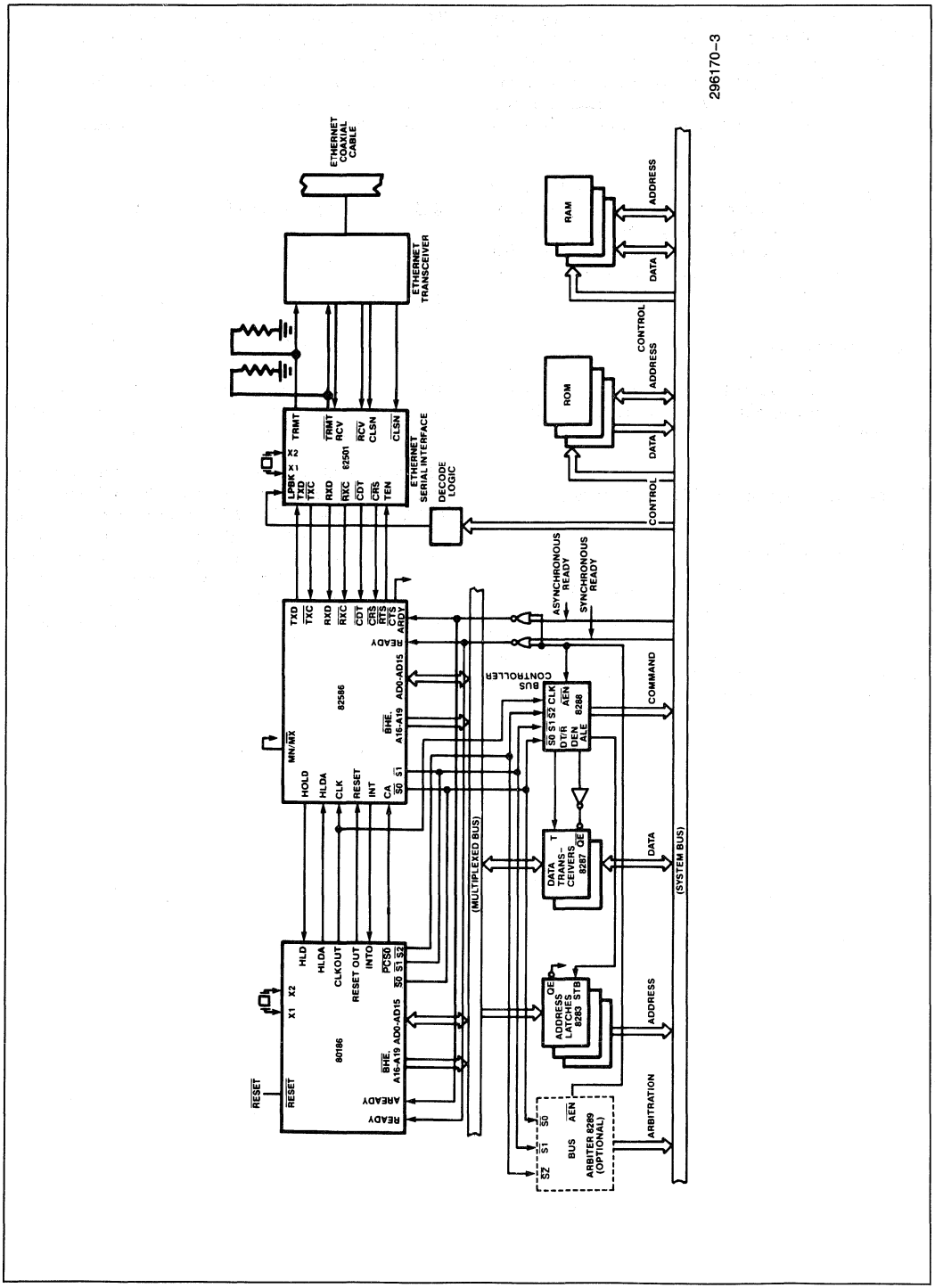


Figure 3. 80186 Elementary Maximum Mode System

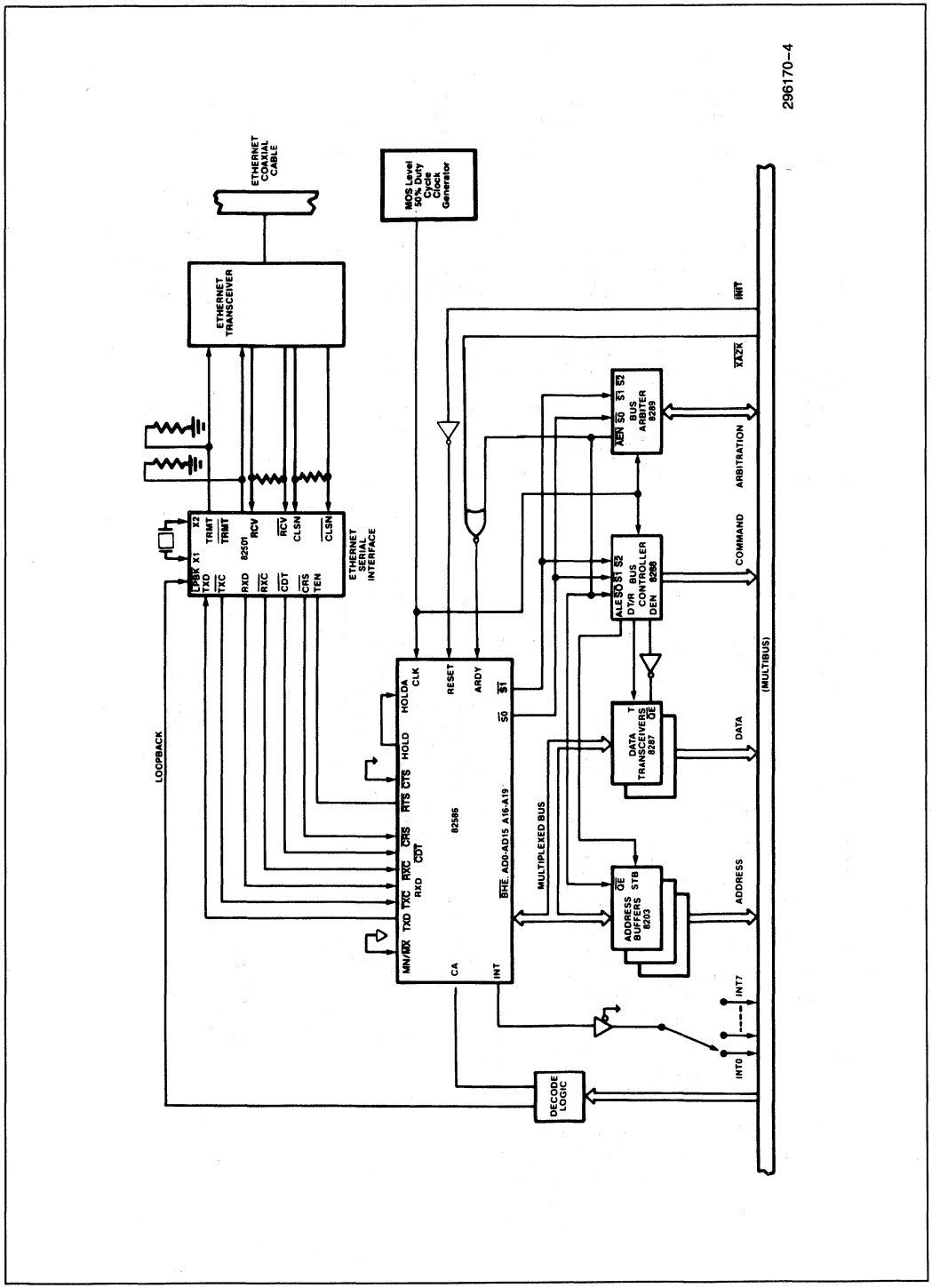


Figure 4. 82586 Stand Alone Multibus Interface

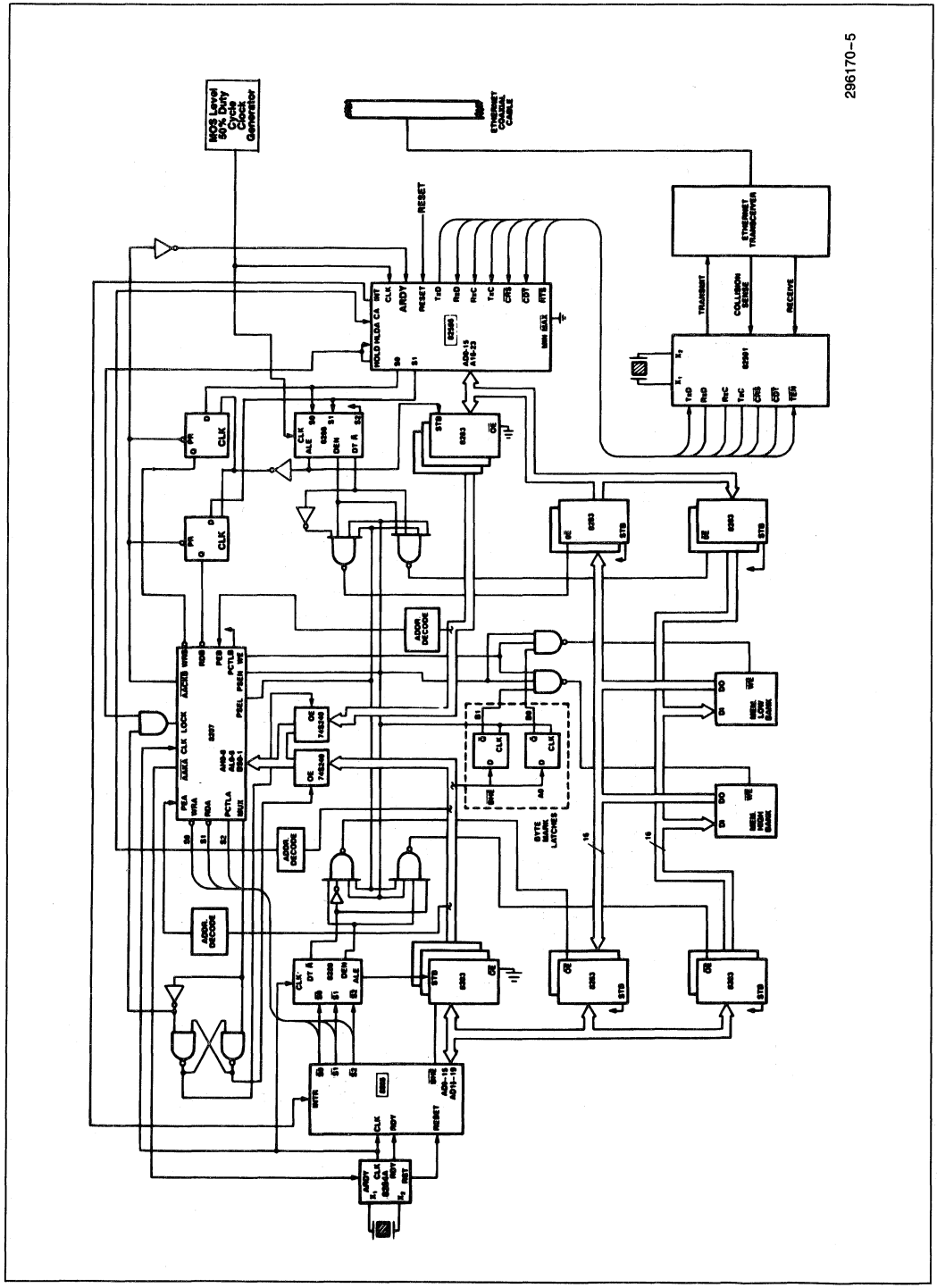


Figure 5. 82586 Used in a Dual Port Configuration

The dual port RAM consists of two memory banks (LOW and HIGH), eight 8283 latches, their steering logic (four NAND gates plus two inverters), 8207, a MUX, and a memory read/write control logic (BYTE MARK LATCHes plus two NAND gates). The port is selected to read/write from/to the memory via PSEL, PSEN 8207 generated signals and DEN signals generated by both ports systems. The LOW or HIGH bank is selected by the BYTE MARK LATCH (driven by A0, $\overline{BHE}$ and PSEN). The transfer direction (read or write) is selected via the WE signal from the 8207 and DT/$\overline{R}$ signals generated by both port systems. The port arbitration is provided by the 8207, that is strobing-in both port addresses via a MUX and outputs the presently selected port address.

The strobed-in address is chosen by the 8207 generated MUX signal. Once the processor commanded transfer is executed, an $\overline{XACK}$ signal is issued by the 8207 to acknowledge transfer completion. As shown in Figure 5 configuration, the 82586 has absolute control of the dual ported memory, immediately after issuing the HOLD signal. This is accomplished by the 8207 LOCK signal driven by the 82586 HOLD.

The 82586 port system shown in Figure 5 is similar to the MULTIBUS stand-alone configuration. Thus a dedicated MOS level Clock Generator is needed on the 82586 port.

NOTE:
If a 22 bit address space is sufficient, the 82586 can be configured to Minimum Mode (strapping the MN/$\overline{MX}$ pin to VCC). In this mode, the 82586 generates directly the DEN and DT/$\overline{R}$ signals, thus saving the 8288 Bus Controller.

A low cost dual port RAM design is discussed in section 6.

4.4 Multiple Bus Master Systems

The 82586 is commonly used in systems with other bus masters, for example, Intel's Text Coprocessor, the 82730, or multiple 82586s. To ensure that these bus masters are initialized separately, the following procedure is recommended.

1) Reset the system through a hardware RESET.
2) Set up the SCP, ISCP for Master Number 1, with appropriate pointers.
3) Give Channel Attention to Master Number 1.
4) Change the ISCP SCB OFFSET address in ISCP for Master Number 2.
5) Give Channel Attention to Master Number 2.

Steps 4 and 5 may be repeated for the third or more master on the same bus.

5.0 CALCULATING UNIQUE MULTICAST ADDRESSES

The 82586 performs multicast address filtering through a hashing algorithm. The CRC polynomial is used to map the received multicast address into a hash table consisting of 64 bits. Because the CRC polynomial is used, it is possible for two or more multicast addresses to be mapped into the same bit in the hash table.

It is possible to select up to 64 unique multicast addresses. The Pascal program Hash Calculation, shown in Table 5, selects unique addresses. The user may specify either the 16 or 32-bit CRC function, and variable address length (1 to 6 bytes). A warning will be issued for non-multicast addresses (i.e. starting with a 0 bit). The function HASH computes the Hash value which is the bit address in the 82586 Hash table.

Table 5. PROGRAM Hash Calculation (INPUT, OUTPUT)

```
TYPE MCtype                 = ARRAY [0..47] OF BOOLEAN; (* bit pattern
                                  that represents the Multicast Address.
                                  0 is the least significant bit.
                                  Addresses shorter than 6 bytes are
                                  justified to the least significant bit. *)
     HTtype                 = 0..63;        (* entry of hash table *)

VAR AddressLength           : 1..6;         (* Address Length *)
     CRCLength              : INTEGER;      (* if 16 then CRC 16 else CRC 32*)
     MCV                    : MCtype;       (* Multicast Address Vector *)
     k, l, m                   : INTEGER;      (* temp *)
     digit                  : CHAR;         (* hexadecimal digit *)
     numb                   : INTEGER;      (* numerical equivalence *)

FUNCTION HASH (AL                           (* Address Length in bytes *)
               CL:INTEGER;                  (* CRC length: 16 or 32 bits *)
               X:MCtype)                    (* bit array that represents Multicast
                                               Address *)
               : HTtype;                    (* result: HASH value between 0 and 63 *)
VAR
     A          : ARRAY [0..31] OF BOOLEAN; (* the CRC generator *)
     FB         : BOOLEAN;                  (* Feedback *)
     i          : INTEGER;                  (* temp *)
     Y          : HTtype;                   (* Y = Hash(X) *)

     BEGIN      (* this function returns the hashed value of X.
                   according to the Address Length and the CRC Length *)

     FOR i:=0 TO 31 DO A[i] := true; (* initialize to all ones *)

     FOR i:=0 TO AL*8-1 DO  (* repeat according to the address length *)
          BEGIN
          IF CL=16 THEN    (* separate CRC computation for CRC-16 *)
               BEGIN
               FB := A[15] <> X[i];         (* compute feedback *)
                                            (* shift the CRC register *)
               A[15] := A[14];
               A[14] := A[13];
               A[13] := A[12];
               A[12] := A[11] <> FB;        (* exclusive or *)
               A[11] := A[10];
               A[10] := A[9];
               A[9] := A[8];
               A[8] := A[7];
               A[7] := A[6];
               A[6] := A[5];
               A[5] := A[4] <> FB;
               A[4] := A[3];
               A[3] := A[2];
               A[2] := A[1];
               A[1] := A[0];
               A[0] := FB;
               END
```

Table 5. PROGRAM Hash Calculation (INPUT, OUTPUT) (Continued)

```
ELSE
    BEGIN       (* separate computation for CRC-32 *)
    FB := A[31] <> X[i];     (* compute feedback *)
                             (* shift the CRC register *)
    A[31] := A[30];
    A[30] := A[29];
    A[29] := A[28];
    A[28] := A[27];
    A[27] := A[26];
    A[26] := A[25] <> FB;
    A[25] := A[24];
    A[24] := A[23];
    A[23] := A[22] <> FB;
    A[22] := A[21] <> FB;
    A[21] := A[20];
    A[20] := A[19];
    A[19] := A[18];
    A[18] := A[17];
    A[17] := A[16];
    A[16] := A[15] <> FB;
    A[15] := A[14];
    A[14] := A[13];
    A[13] := A[12];
    A[12] := A[11] <> FB;
    A[11] := A[10] <> FB;
    A[10] := A[9] <> FB;
    A[9] := A[8];
    A[8] := A[7] <> FB;
    A[7] := A[6] <> FB;
    A[6] := A[5];
    A[5] := A[4] <> FB;
    A[4] := A[3] <> FB;
    A[3] := A[2];
    A[2] := A[1] <> FB;
    A[1] := A[0] <> FB;
    A[0] := FB;
    END;

END;       (* of CRC calculation *)

           (* select the 6 bits out of the CRC register *)
IF A[5] THEN Y := 1 ELSE Y := 0;
IF A[6] THEN Y := Y + 2;
IF A[7] THEN Y := Y + 4;
IF A[2] THEN Y := Y + 8;
IF A[3] THEN Y := Y + 16;
IF A[4] THEN Y := Y + 32;
HASH := Y;   (* return the value *)
END; (*of hash function *)
```

Table 5. PROGRAM Hash Calculation (INPUT, OUTPUT) (Continued)

```
BEGIN       (* main program that interacts with the user *)
WRITELN ('This program calculates the hash function of Multicast addresses');
WRITELN ('Type CTRL/Y for exiting the program')'
WRITELN(' ');
WRITE('Enter CRC length (16 for CRC-16) >> ');
READLN(CRCLength);
WRITE ('Enter Address length (in bytes) >> ');
READLN(AddressLength);
IF AddressLength>6 THEN WRITELN(' Out of range !')

ELSE BEGIN

REPEAT      (* for each Multicast Address in the set *)

FOR k:=0 TO 47 DO MCV[k] := FALSE;      (* initialize the multicast vector *)

WRITELN('Enter Multicast Address (exactly ', AddressLength*2:3,' hexadecimal
  digits');
WRITE('starting with MOST significant) >> ');

FOR k:= AddressLength*2-1 DOWN TO 0 DO        (* for all digits *)
     BEGIN                                    (* read a hexadecimal digit *)
     READ(digit);                             (* and convert it to binary *)
     IF (48<=ORD(digit)) AND (ORD(digit)<58) THEN numb := (ORD(digit)-48)
ELSE
     IF (65<=ORD(digit)) AND (ORD(digit)<71) THEN numb := (ORD(digit)-55)
ELSE
     IF (97<=ORD(digit)) AND (ORD(digit)<103) THEN numb := (ORD(digit)-87);

     FOR l:=0 TO 3 DO
          BEGIN                               (* convert the digit to binary and
                                              insert it to the vector *)
          IF ODD(numb) THEN MCV[4*k+l] := TRUE ELSE MCV[4*k+l] := FALSE;
          numb := numb DIV 2;
          END;
     END;
     READLN;                                  (* flush the line *)
     IF NOT MCV[0] THEN WRITELN ('not a multicast address!');

                                              (* call the hash function *)
     WRITELN('HASH = ',HASH(AddressLength, CRCLength, MCV));

UNTIL FALSE;    (* for ever *)
END;            (* of loop for specific Multicast Address *)
END.
```

6.0 A LOW COST DUAL PORT MEMORY DESIGN

The 82586 is a bus master designed to share the CPU system bus. The shared bus configuration affords systems that use the fewest components and lower cost. However, in some applications the bus bandwidth required by the 82586 may seriously degrade system performance. The solution to this problem is a dual port memory between the host CPU and the 82586. Figure 6 illustrates a shared bus system and Figure 7 shows a dual port 82586/CPU system configuration.

In 82586 applications the dual port approach is useful in the following cases:

1) To minimize bus contention between the CPU and the 82586. The 82586 bus interface design is free of bus latency concerns. This issue is important in systems where the peripherals attached to the system bus cause latency that cannot be tolerated by the 82586, or vice versa.
2) To interface to a limited bandwidth bus. The 82586 may consume as much as 2M bytes/s bandwidth while transmitting or receiving data at 10 Mbps. The instantaneous bandwidth required to meet the IEEE 802.3 specification is even greater. In a dual port system the host CPU is able to access memory when the 82586 is accessing memory, thus high system performance is maintained. This issue is particularly important in 8-bit bus systems.
3) To simplify the interface between the 82586 and non-Intel CPUs.

This section describes an example of dual port design using the 82586 with an 80186.

6.1 Hardware Design

The major characteristics of the hardware design are as follows:

Dual Port Memory Size	-	8K bytes
Memory Type	-	Static RAM's
CPU	-	80186
Bus Arbitration Logic	-	TTL SSI/MSI chips
82586 Bus Speed	-	8 MHz
82586 Serial Data Rate	-	10 Mbps

The objective of this hardware design is to keep the design simple and inexpensive. Figure 8 shows a block diagram of the hardware. The CPU interface is general enough to simplify the interface to CPU's with multiplexed or demultiplexed data/address buses. The size of the system RAM is a function of the following factors:

1) Network traffic size.

2) Net traffic pattern (size of frames).

3) Other CPU tasks.

The memory size is 8K bytes to satisfy typical needs. However, the hardware accommodates expansion of the dual port RAM. Static RAMs are used to keep the design simple and low cost.

The 82586 is set to Minimum Mode to keep the chip count low. The CPU for the example is the 80186. Figure 9 shows a general purpose CPU interface.

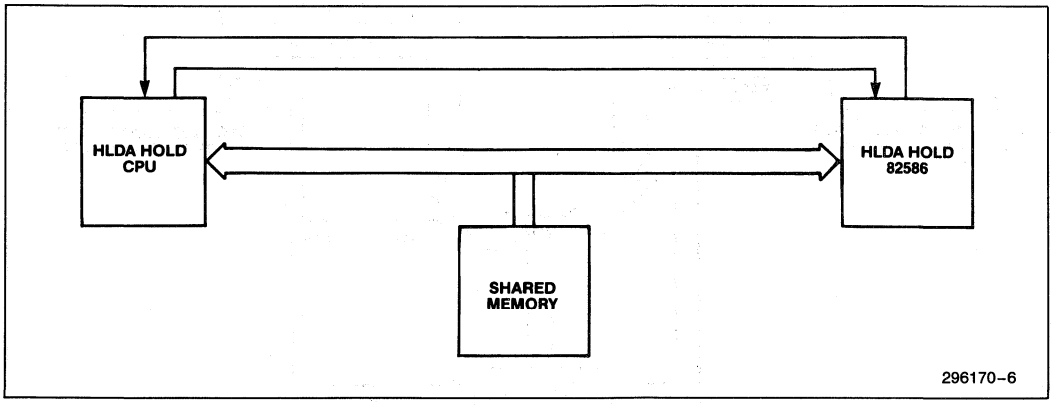


Figure 6. A Typical Shared Memory Design Using the 82586

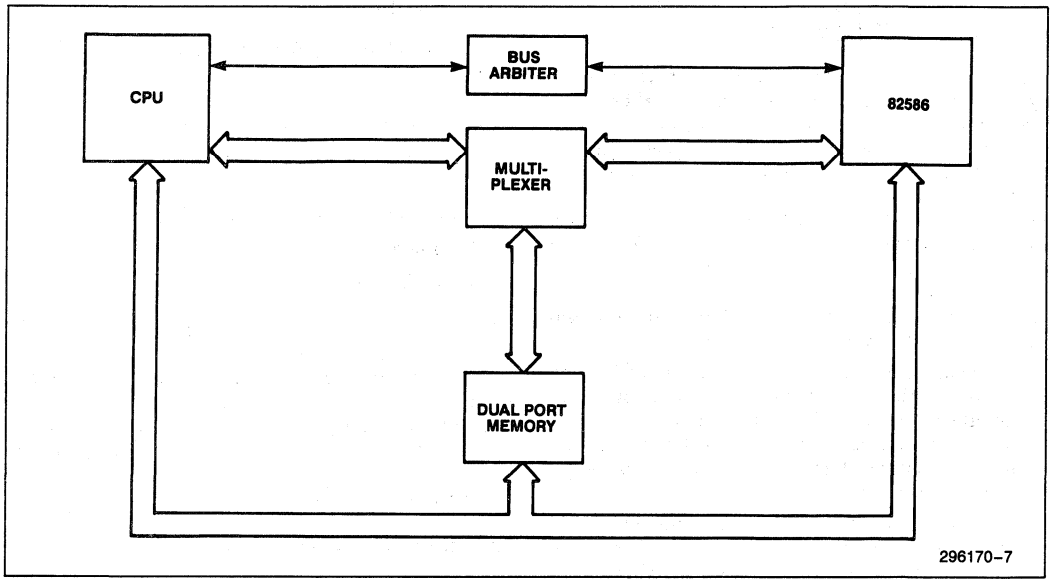


Figure 7. A Typical Dual Port Memory Design Using the 82586

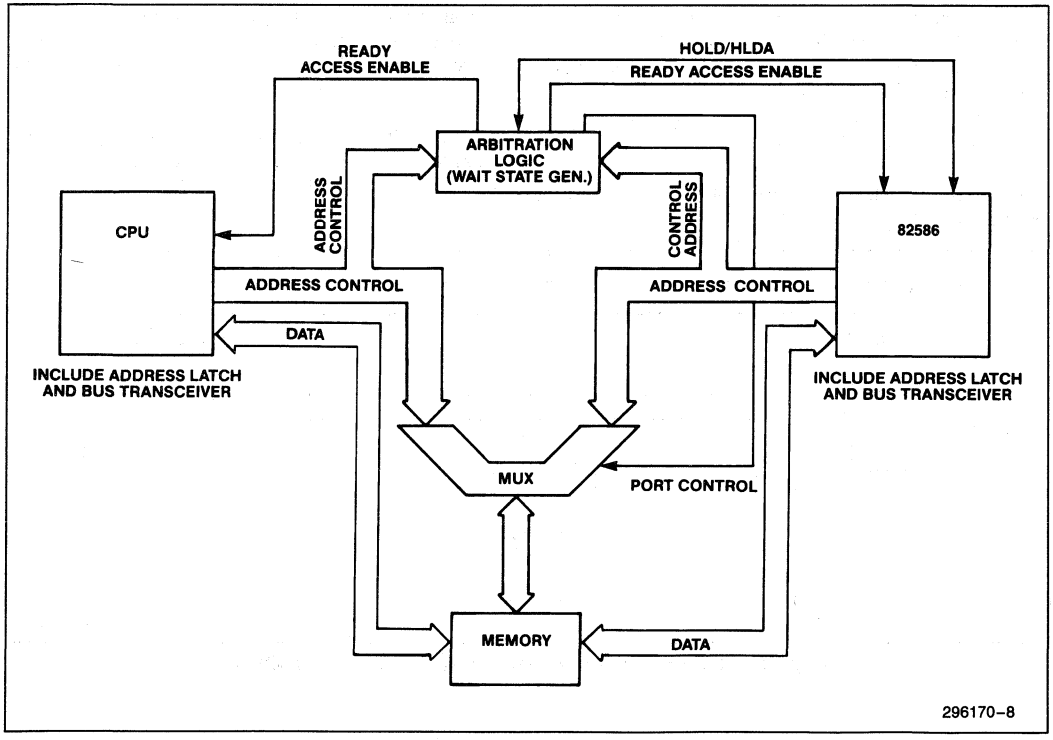


Figure 8. Hardware Block Diagram

BUS ARBITRATION LOGIC

Bus contention between the 82586 and the 80186 is resolved on the basis of the following principles:

1) When the 82586 is accessing the Dual Port Memory RAM, the CPU is prevented from accessing the memory by inhibiting the Ready signal.

2) The CPU can access memory whenever the 82586 is not accessing it.

3) The 82586 gains control of the bus within four to five clocks after a request, and the CPU is put into the wait mode.

The sequence of events when the 82586 requests the bus is:

1) The 82586 requests the bus by activating the HOLD signal.

2) The HLDA is given to the 82586 after five clocks max.

3) The Ready signal to the CPU is disabled within one clock of the 82586 HOLD request.

4) The CPU is isolated from the dual port RAM by disabling the data buffers within 2 clocks of the 82586 HOLD request.

5) The multiplexer is switched to the 82586 bus.

6) If the CPU is accessing the dual port RAM at the time of the 82586 bus request, the CPU is either placed in a wait mode, or it is given enough time to complete its bus cycle.

7) If the CPU has been placed in a wait mode, it remains there until the 82586 is done with the Dual Port RAM. Then the CPU's data and address buffers are enabled, the Ready signal is enabled and the multiplexers are switched back to the CPU, thus allowing the CPU enough time to complete its cycle.

The Channel Attention for the 82586 is generated using a peripheral chip select line of the 80186. In this application, the interrupts from the 82586 are ignored. The diagnostic software, discussed in section 6.2, works under an interactive polled environment.

READY GENERATION

The design adopts a simple scheme to generate the Ready signal for the 82586 and the 80186. The Ready is generated when either the Read or Write signal goes active. The 82586 is given the Ready signal if it is holding the bus (HOLD active) and either Read or Write signal is active. During this period the Ready signal going to the 80186 Ready generation logic is disabled.

The Dual Port RAM generates its own Ready signal for the 80186. This Ready signal is only generated when the 80186 accesses the Dual Port RAM and the 82586 HOLD is inactive. Other memory blocks and peripherals should generate their own Ready signal. Thus all the Ready signals going to the 80186 can be ORed together provided that they are normally low.

Asynchronous Ready is used for both 82586 and 80186 for ease of implementation. This design requires zero wait states for both 82586 and 80186 access. However, for slower memories, a wait state generator may be added to the Ready generation logic.

SETTING THE FIFO LIMIT

When the 82586 needs the bus, it activates its HOLD signal and waits for the HOLD Acknowledge signal (HLDA) before starting any memory access cycles. In the case when the CPU and the 82586 are on the same bus, the HOLD signal from 82586 will go to the HOLD input of the CPU. The CPU will eventually grant the HOLD Acknowledge (HLDA) to the 82586. The time to grant HLDA is dependent on the CPU and the task it is executing at the time of the HOLD request. When HLDA is granted, the 82586 starts its memory access operations.

The 82586 Receive and Transmit FIFO-Threshold limits are set as a function of the HOLD-HLDA delay. The following Receive case illustrates how the FIFO-Thresholds are set. Assume that the Receive FIFO trigger is set at 6. This setting will ensure that the 82586 will make a bus request when 6 bytes have been placed into the Receive FIFO. Since the FIFO is 16 bytes deep, the 82586 must start emptying out the FIFO before the FIFO fills up. In this example, the 82586 must acquire the bus within 10 bytes time (16 − 6 bytes) to avoid an overrun condition. At 10 Mbps, 10 bytes equates to 8 microseconds or 64 CPU clocks (assuming the CPU is running at 8 MHz clock rate). Depending upon the application environment, granting the bus to the 82586 in 64 clock times may or may not be easy to accomplish.

The dual port design avoids the problems associated with bus latency. The HOLD Acknowledge is given to the 82586 within 5 clock cycles of the HOLD request, thus allowing for very small HOLD-HLDA delays.

The 82586 operates in a burst mode when it is reading or writing to the memory. The length of the burst is a function of the number of bytes in the Receive or Transmit FIFO at the time the 82586 gets the HOLD Acknowledge signal. When the 82586 obtains control of the bus, it empties the entire contents of the FIFOs.

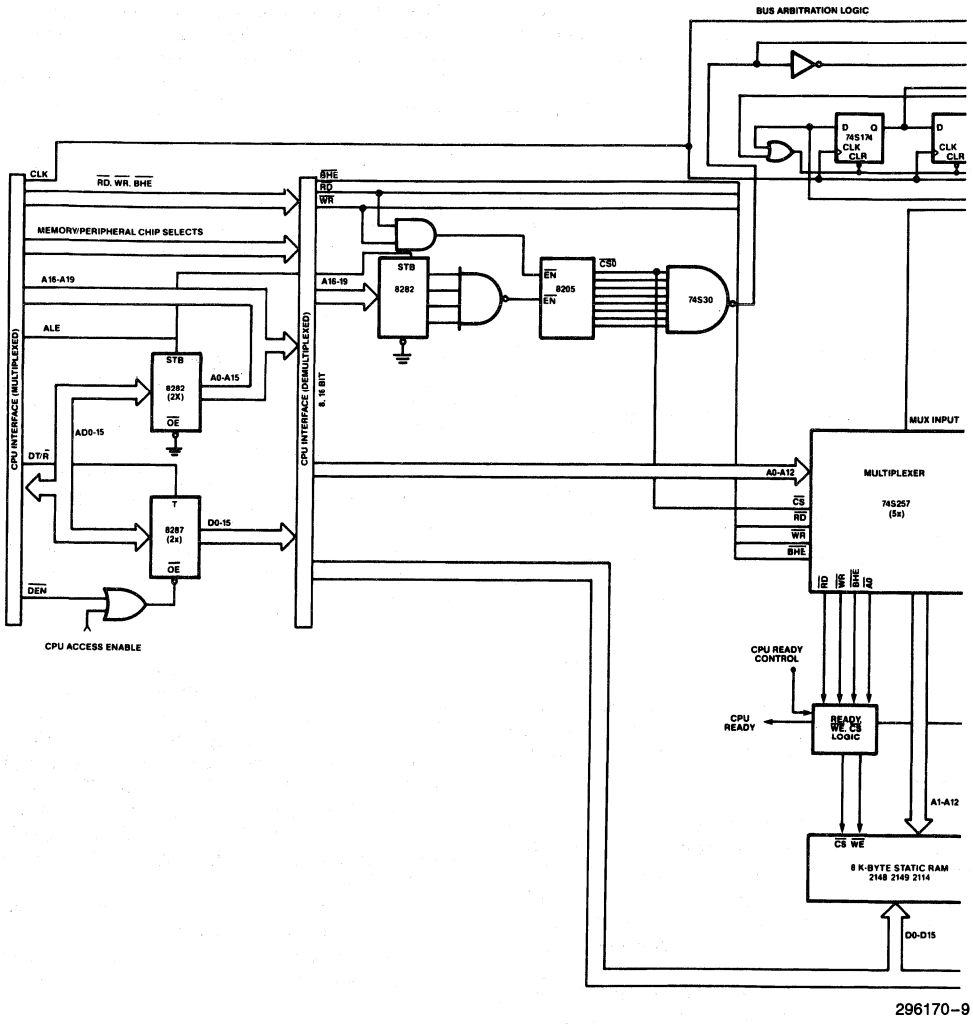


Figure 9. General Purpose CPU Interface

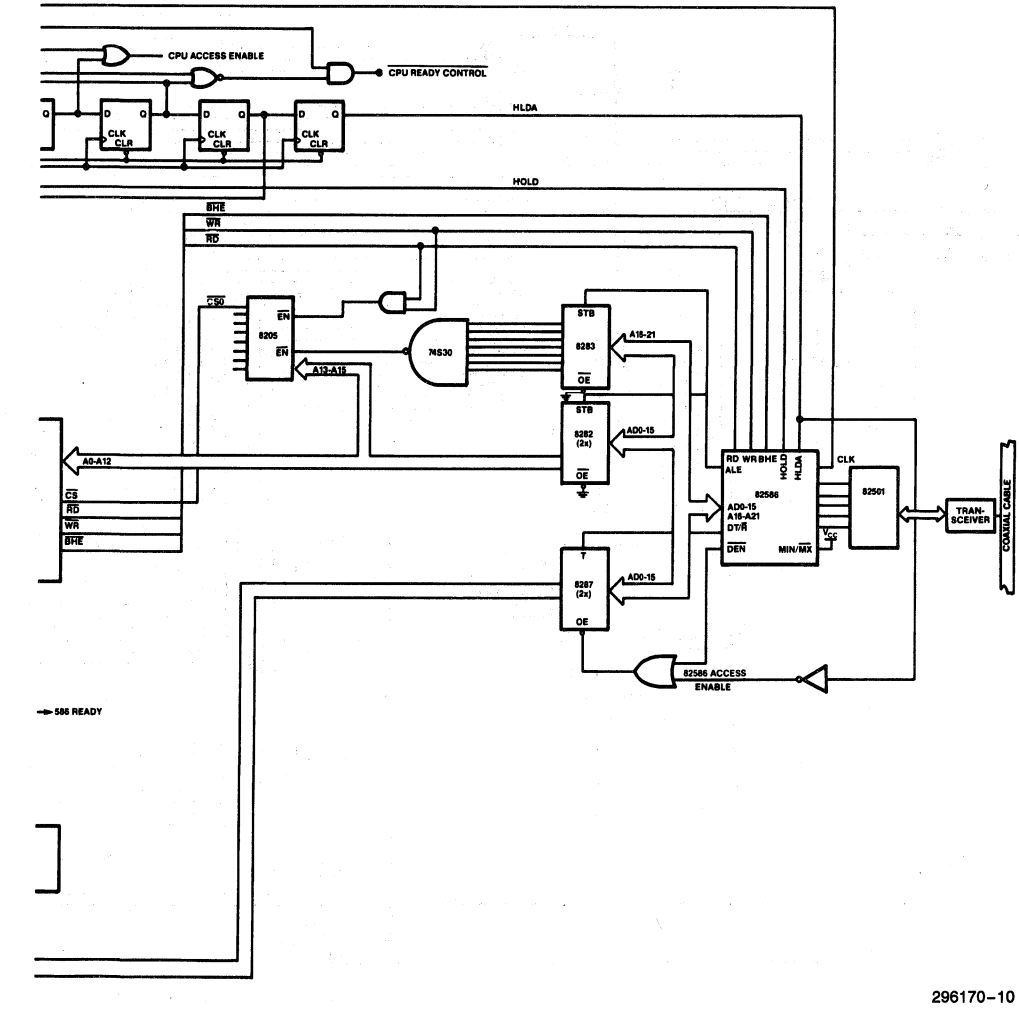


Figure 9. General Purpose CPU Interface (Continued)

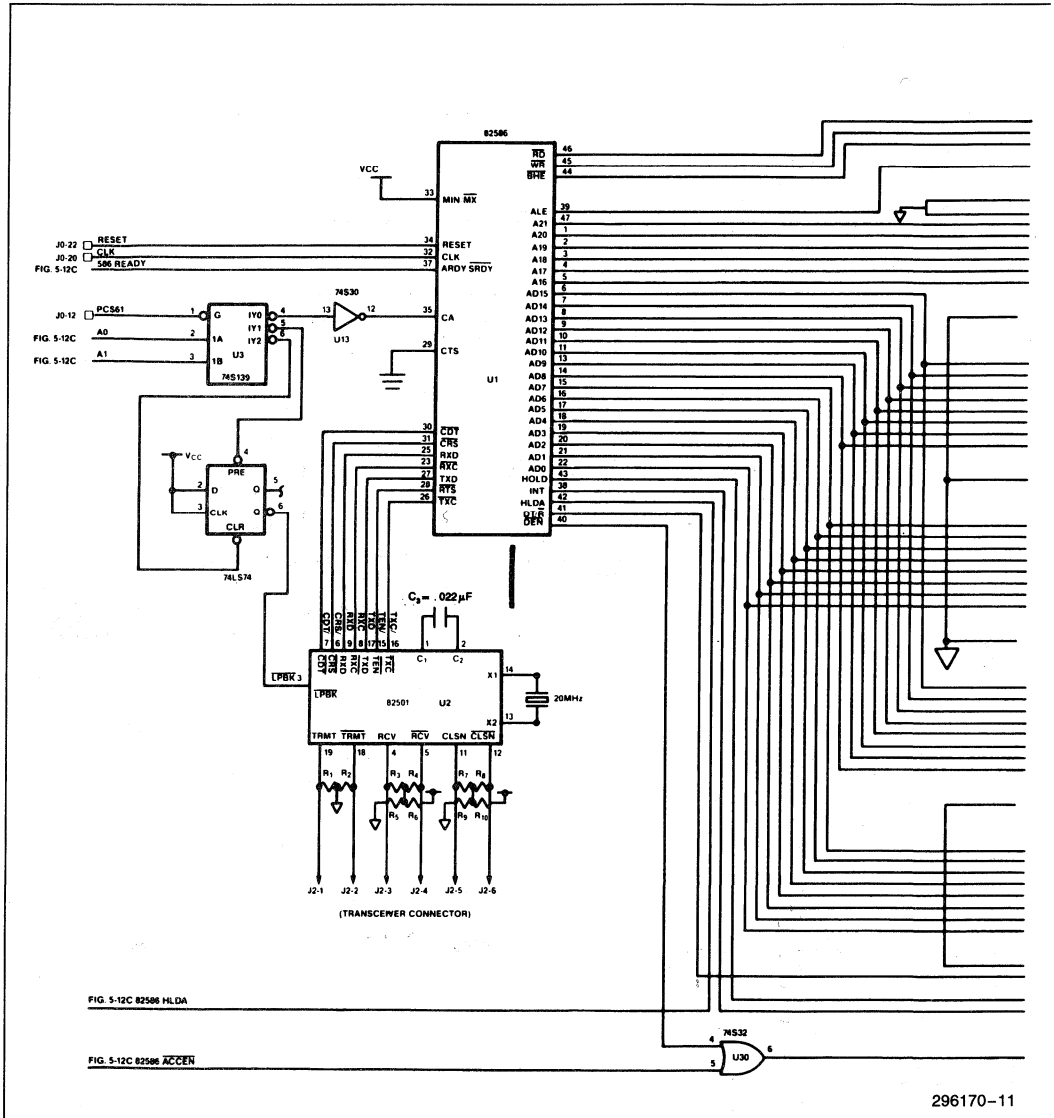


Figure 10A. 82586 Dual Port Design Schematics

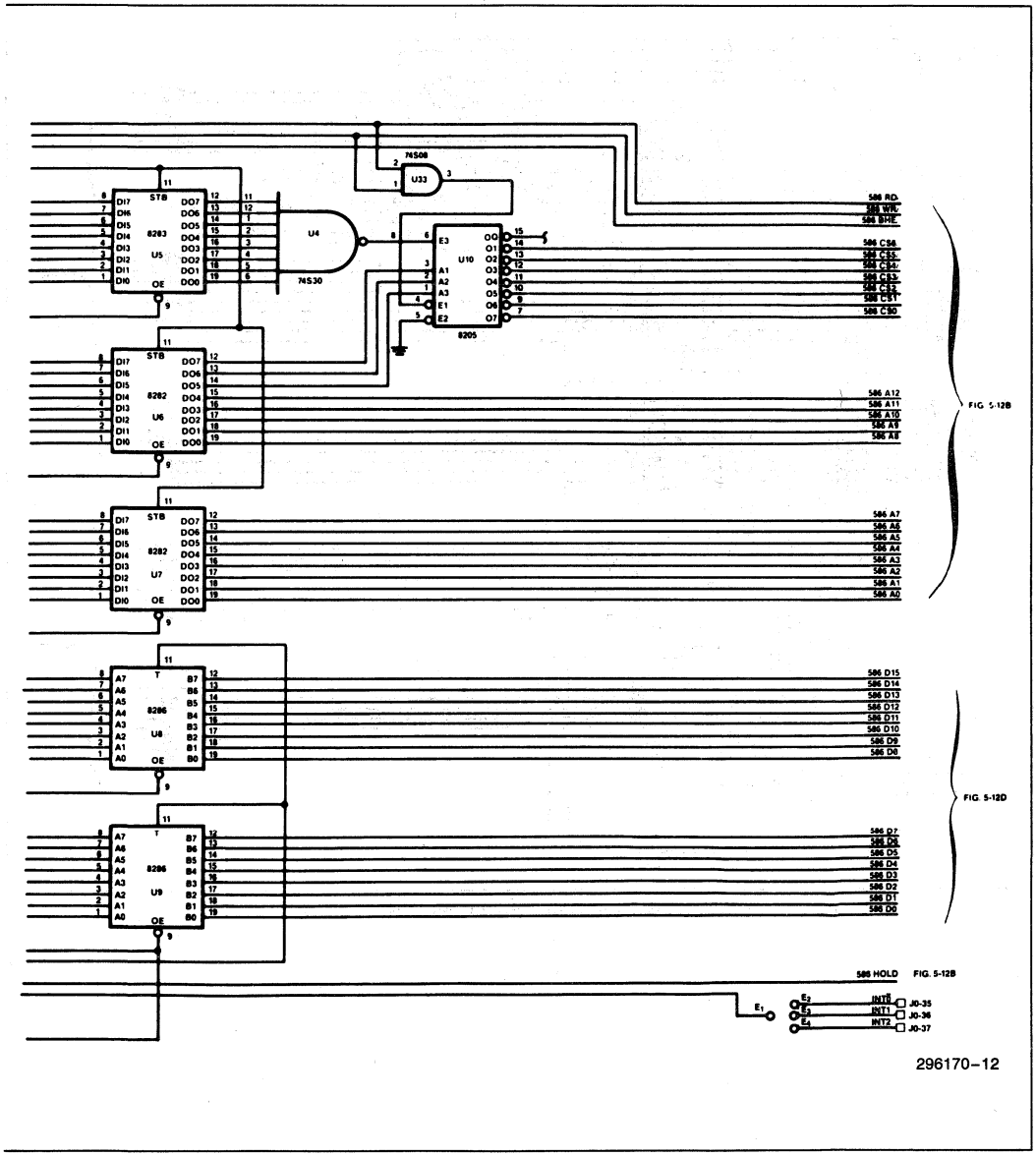


Figure 10A. 82586 Dual Port Design Schematics (Continued)

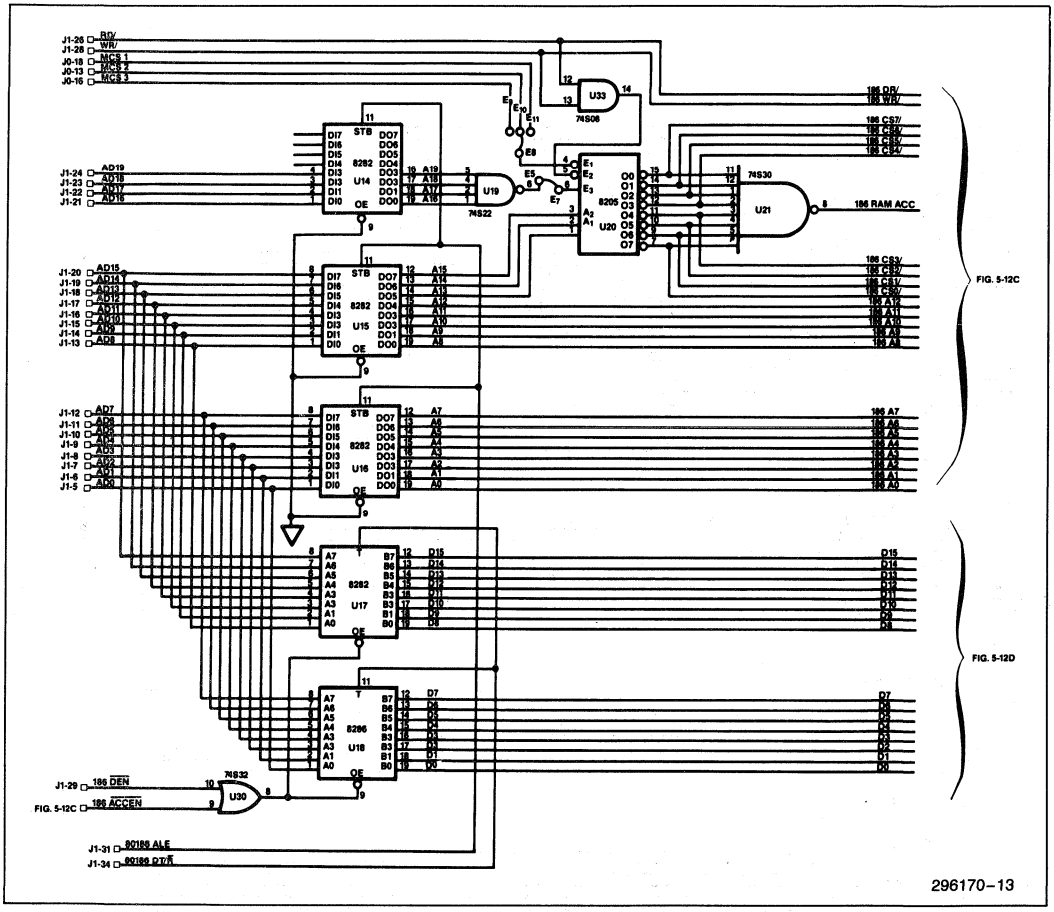


Figure 10B. 82586 Dual Port Design Schematics

Thus maximum bus efficiency is achieved when the 82586 acquires the bus with a full FIFO. A very small HOLD-HLDA delay will allow a high setting of the FIFO-Threshold. The HOLD-HLDA handshake between the 82586 and the CPU always results in a finite amount of wasted bus clock cycles due to HOLD and HOLD Acknowledge synchronization by the 82586 and the CPU.

In summary, the HOLD-HLDA handshake results in wasted bus clocks. In order to realize an efficient system, the 82586 must transfer the maximum number of bytes before giving up control of the bus. In the case of the 82586, these burst sizes should be 16 bytes (i.e. the FIFO size). The objective of the design should be to have 16 bytes in the FIFO when the 82586 gets the bus, so that it may empty out the FIFO in a single burst. A short HOLD-HLDA delay reduces the concern for DMA overruns; thus a high FIFO-Threshold setting is possible.

Another advantage of this design is that HLDA is always given to the 82586 between 4 and 5 clock cycles after a HOLD request. This provides more efficient bus utilization than with a direct microprocessor interface where HOLD and HLDA latency will vary much more than one clock cycle.

The discussion above centered on reception, however the same arguments are true for Transmission. When programming the FIFO-Threshold using the CONFIGURE command, the parameter programmed is the threshold point for the Transmit FIFO. The Receive FIFO-Threshold is sixteen's compliment (15-trigger parameter) of the value programmed.

Hardware schematics are shown in Figure 10A through 10D.

6.2 Application Software

The 82586 dual port hardware design was tested using an 80186 based board, running at 8 MHz. Two ribbon cables access the 80186 address, data, and control signals. These signals are brought to the 82586 board via connectors J0 and J1 (see Figure 10E). The 80186 board operated with an Intel iSBC 957B package monitor program. The 957B monitor was used to debug 82586 hardware and software. For more information on the 957B monitor, refer to the "iSBC 957B™ iAPX 86/88 User's Guide" (Intel order number 143979-002).

The 82586 diagnostic software allows interactive usage. The software was developed using an Intel MDS Series III and downloaded to the 82586 board through the serial link on the 80186 board.

The diagnostic software has the following capabilities.

- Initialize the 82586 by setting up the System Control Pointer, Intermediate System Control Pointer and System Control Block.
- Create a linked list of 16 Frame Descriptors for receive frames. These Frame Descriptors were linked to a linked list of 16 Buffer Descriptors, which in turn pointed to 16 buffers, each of 128 bytes. Figure 11 illustrates the memory structure set up.
- The interactive software package is capable of setting up and executing one or all (through linked lists) the 82586 Action Commands, namely NOP, IA SETUP, CONFIGURE, MC SETUP, TRANSMIT, DUMP, TDR, and DIAGNOSE. Additionally, the contents of these commands can be modified from the console and more than one command may be executed by linking them onto a linked list.
- The System Control Block commands for the Command Unit, as well as the Receive Unit, can also be given from the console.
- The Transmit and Receive Frame Descriptors and buffers can be modified interactively from the console.
- Several commands are available for interactive observation. In particular, the following fields in the 82586 memory structures are available for observation:

1) System Control Block:
   - Status field
   - Acknowledge bits
   - CUC, RUC fields
   - Error tallies (alignment, CRC, overrun and resource errors)

2) Command Blocks:
   - Status field
   - Link field
   - Command related parameters

3) Transmit and Receive Buffers:
   - Transmit and Receive Buffer Descriptors and buffer areas
   - Receive Frame Descriptors

Table 6 contains the software listings for this application. Figure 12A through 12K illustrates the flow for the software modules.

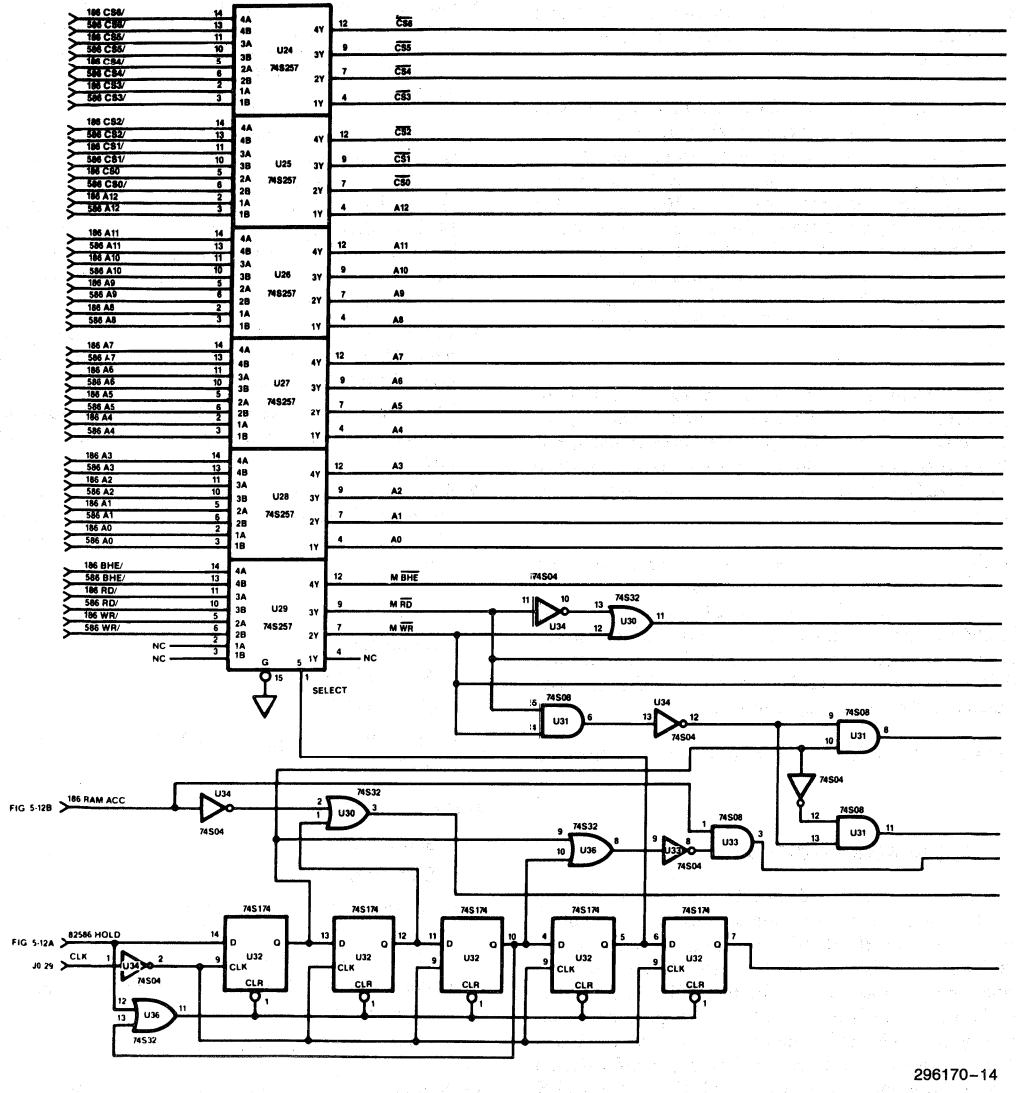


Figure 10C. 82586 Dual Port Design Schematics

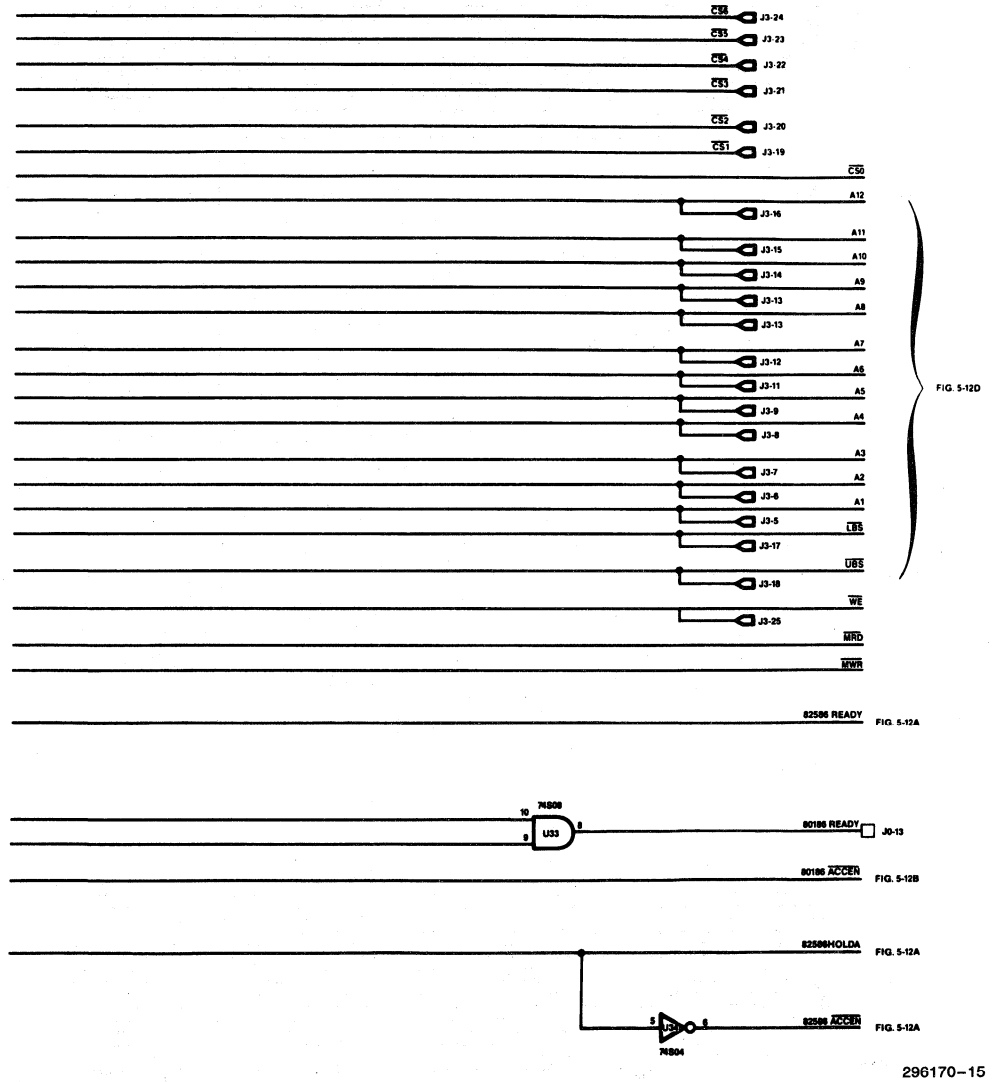


Figure 10C. 82586 Dual Port Design Schematics (Continued)

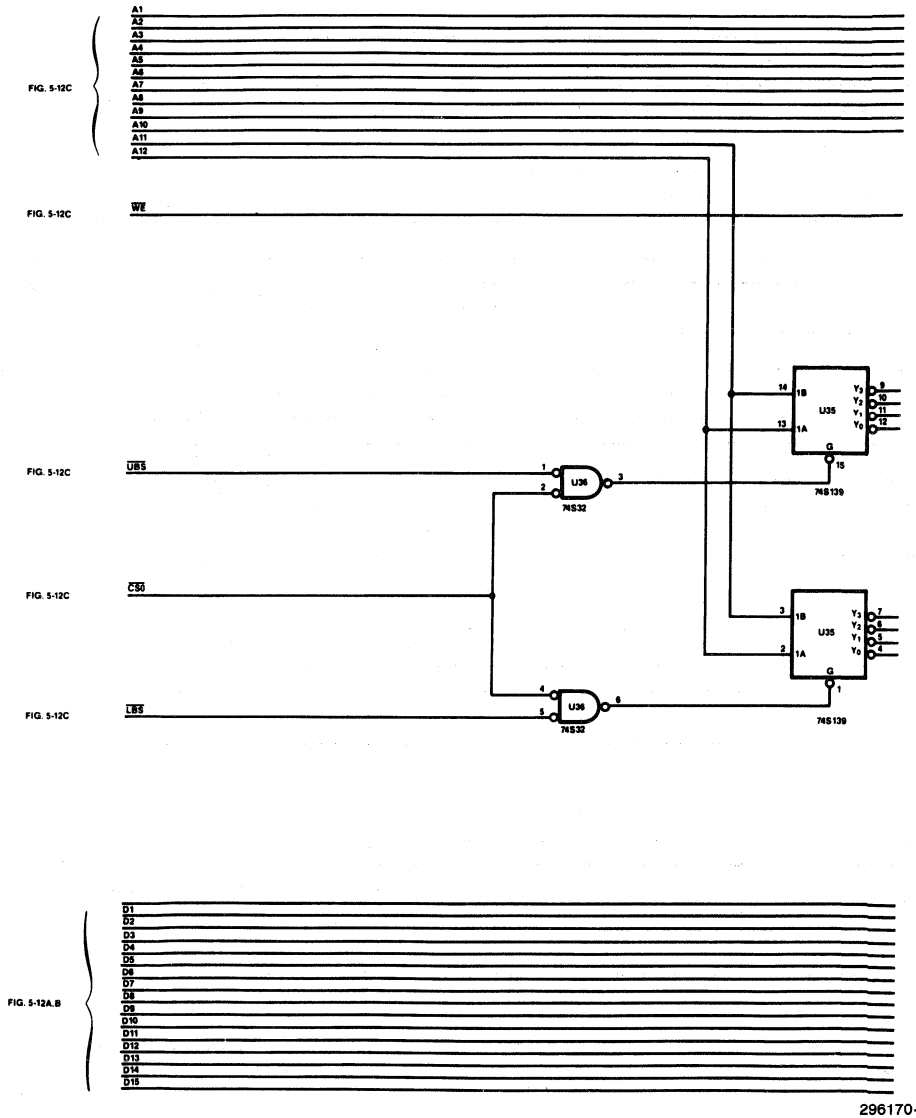


Figure 10D. 82586 Dual Port Design Schematics

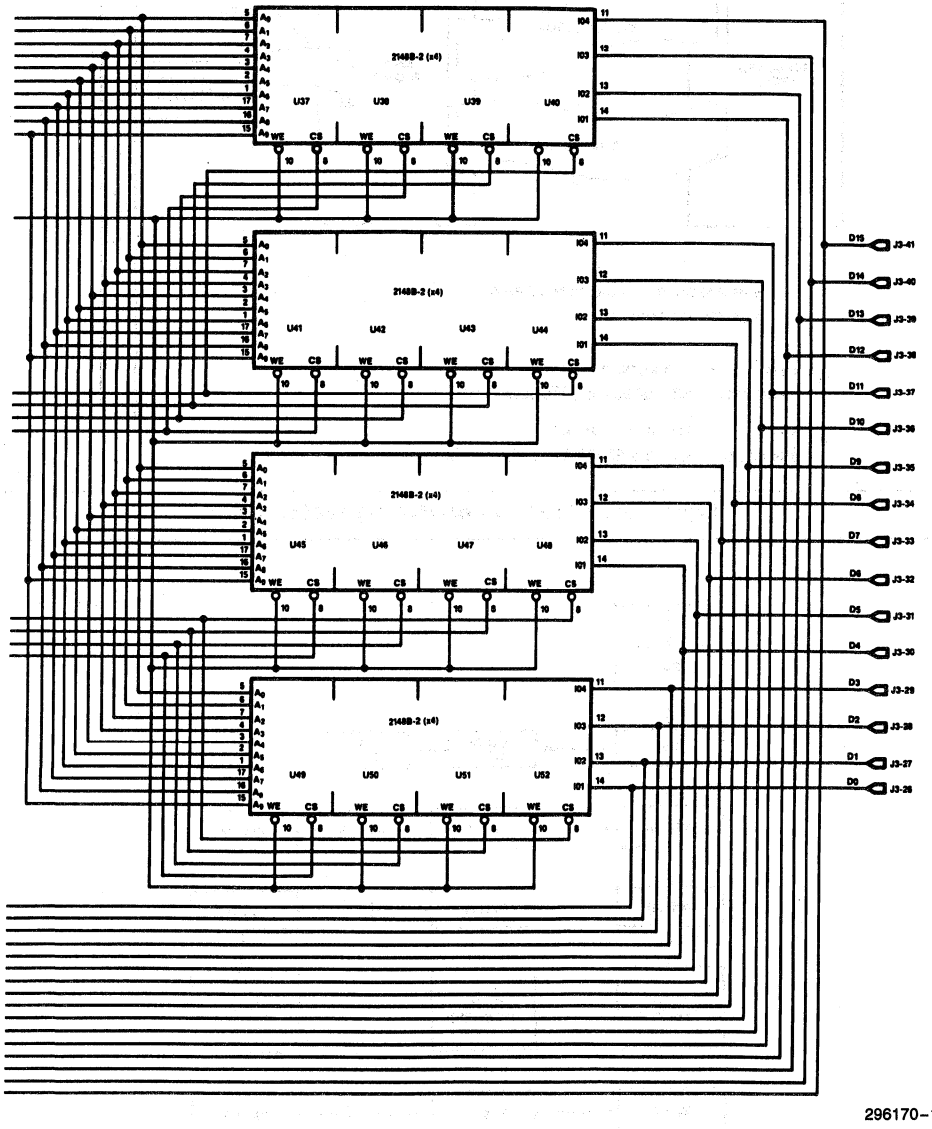


Figure 10D. 82586 Dual Port Design Schematics (Continued)

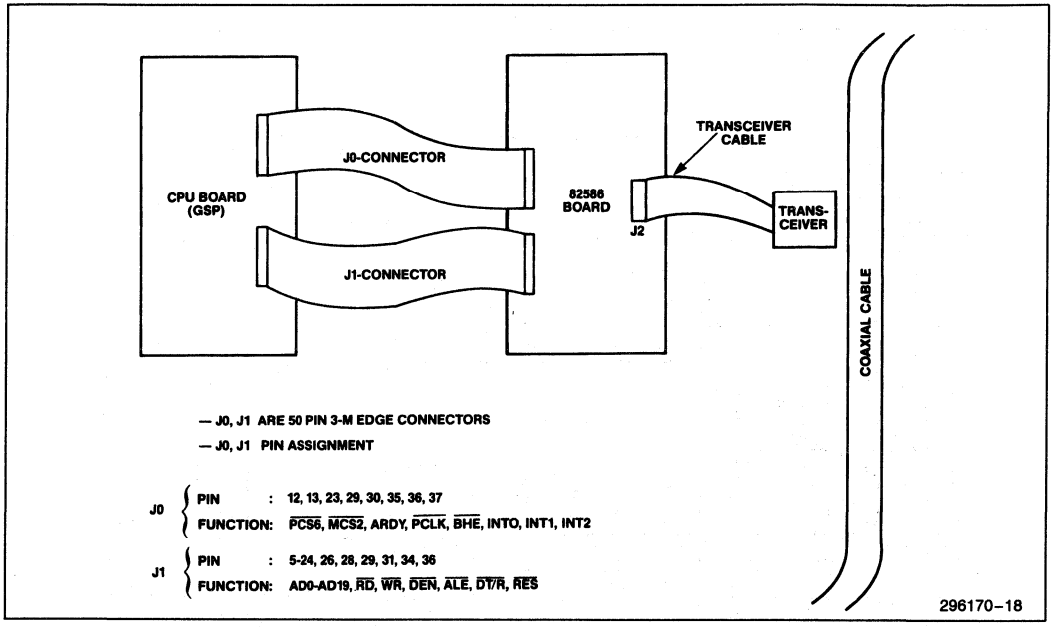


Figure 10E. 82586 Dual Port Memory Test Hardware

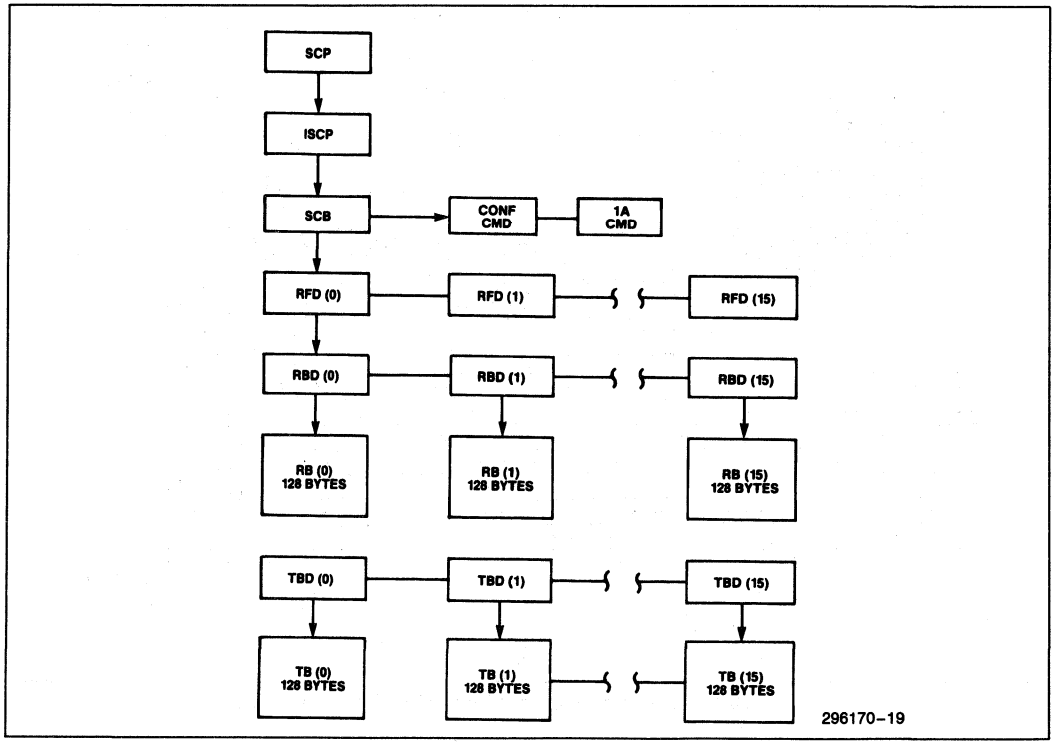


Figure 11. Memory Map for 82586

Table 6. Software Listings

```
/*****************************************************************
*                                                                *
*                                                                *
*                    82586 DUAL PORT MEMORY DESIGN               *
*                                                                *
*                    INTERACTIVE DIAGNOSTIC SOFTWARE             *
*                                                                *
*                    JAN. '84    SN/MH                           *
*                                                                *
*                                                                *
*****************************************************************/




1       DIAGNOSTICS$FOR$82586: DO;

2   1   declare MAIN label public;


        /* Word Declare */

3   1       declare FOREVER          literally 'WHILE 1';

        /* Constant Declare */

4   1       declare CA$PTR               literally '700H',
                LPBK$ON$PTR              literally '701H',
                LPBK$OFF$PTR             literally '702H',
                CMD$START                literally '0100H',
                RCV$START                literally '0010H',
                CMD$ABORT                literally '0400H',
                RCV$ABORT                literally '0040H',
                CMD$SUSPEND              literally '0300H',
                RCV$SUSPEND              literally '0030H',
                CMD$RESUME               literally '0200H',
                RCV$RESUME               literally '0020H',
                C$RES                    literally '0080H',
                CR                       literally '0DH',
                LF                       literally '0AH',
                BS                       literally '08H',
                SP                       literally '20H',
                DEL                      literally '07FH',
                BEL                      literally '07H',
                TRUE                     literally '0FFH',
                FALSE                    literally '0H',
                TRUEW                    literally '0FFFFH';


        /* Address Declare */

5   1       declare SCP$OFFSET           word data (0FFF6H);
6   1       declare ISCP$OFFSET          word data (0FFE0H);
7   1       declare SCB$OFFSET           word data (0FFD0H);
8   1       declare RX$OFFSET            word data (0E000H);
9   1       declare TX$OFFSET            word data (0E800H);
10  1       declare NOP$OFFSET           word data (0FF70H);
11  1       declare IA$SETUP$OFFSET      word data (0FF76H);
12  1       declare CONF$OFFSET          word data (0FF82H);
13  1       declare MC$SETUP$OFFSET      word data (0FF94H);
14  1       declare TRANSMIT$OFFSET      word data (0FFA2H);
15  1       declare TDR$OFFSET           word data (0FFB2H);
16  1       declare DUMP$STAT$OFFSET     word data (0FFBAH);
17  1       declare DIAG$OFFSET          word data (0FFC2H);
18  1       declare RFD$OFFSET           word data (0F200H);
19  1       declare RBD$OFFSET           word data (0F000H);
20  1       declare TBD$OFFSET           word data (0F100H);
21  1       declare DS$BUFF$OFFSET       word data (0F400H);
22  1       declare EX$TRANSMIT$OFFSET   word data (0F500H);


23  1       declare MSG$PTR                          pointer,
                MSG$BUF  based MSG$PTR  (5)          byte,
                INT$PTR                              pointer,
```

296170-20

Table 6. Software Listings (Continued)

```
                        C$BUF     (160)                 byte,
                        ASCII$BUF   (2)                 byte,
                        INT$BUF                         byte,
                        NEXT$CMD$OFFSET                 word,
                        CURRENT$CMD$OFFSET              word,
                        CMD$PTR                         pointer,
                        DAT based CMD$PTR               byte,
                        MESSAGE$WORD                    word,
                        MESSAGE$BYTE                    byte,
                        MESSAGE$STATUS                  byte,
                        CMD$TOP$STATUS                  byte,
                        CMD$TOP$OFFSET                  word;

 24   1          declare COMMON$BASE          selector    data (6000H);
 25   1          declare SCB$BASE             word        data (6);


              /* System Configuration Pointer */

 26   1          declare SCP$PTR              pointer;
 27   1          declare SCP                  structure
                                              (SYSBUS                word,
                                               DUMMY                 dword,
                                               ISCP$offset           word,
                                               iscp$base             word) at(6FFF6h);


              /* Intermediate System Control Pointer */
 28   1          declare ISCP$PTR             pointer;
 29   1          declare ISCP                 structure
                                              (BUSY                  word,
                                               SCB$OFFSET            word,
                                               SCB$BASE1             word,
                                               SCB$BASE2             word) at(6FFE0H);


              /* System Control Block */

 30   1          declare SCB$PTR              pointer;
 31   1          declare SCB                  structure
                                              (STAT                  word,
                                               CMD                   word,
                                               CBL$OFFSET            word,
                                               RFD$OFFSET            word,
                                               CRC$ERRS              word,
                                               ALN$ERRS              word,
                                               RSC$ERRS              word,
                                               OVRN$ERRS             word) at(6FFD0H);



              /* Receive Frame Descriptor */

 32   1          declare RFD$PTR              pointer;
 33   1          declare RFD  (10h)           structure
                                              (STAT                  word,
                                               EL$S                  word,
                                               LINK$ADDRESS          word,
                                               BD$PTR                word,
                                               DEST$ADDRESS (6)      byte,
                                               SOURCE$ADDRESS (6)    byte,
                                               TYPE$FIELD            word) at(6F200H);


              /* Receive Buffer Descriptor */

 34   1          declare RBD$PTR              pointer;
 35   1          declare RBD  (10H)           structure
                                              (ACT$COUNT             word,
                                               NEXT$BD$ADD           word,
                                               BUFF$OFFSET           word,
                                               BUFF$BASE             word,
                                               SIZE                  word) at(6F000H);


              /* Transmit Buffer Descriptor */

 36   1          declare TBD$PTR              pointer,
                         TBD   (10H)          structure
                                              (ACT$COUNT             word,
                                               NEXT$BD$ADD           word,
                                               BUFF$OFFSET           word,
                                               BUFF$BASE             word) at(6F100H);
```

296170-21

Table 6. Software Listings (Continued)

```
                /* NOP Command */

37    1         declare NOP$PTR             pointer;
38    1         declare NOP                 structure
                                            (STAT                word,
                                             CMD                 word,
                                             LINK$ADDRESS        word) at(6FF70H);


                /* Individual Address Setup */

39    1         declare IA$SETUP$PTR        pointer,
                        IA$SETUP            structure
                                            (STAT                word,
                                             CMD                 word,
                                             LINK$ADDRESS        word,
                                             IA$ADDRESS (6)      byte) at(6FF76H);


                /* Configuration Command */

40    1         declare CONF$PTR            pointer,
                        CONF                structure
                                            (STAT                word,
                                             CMD                 word,
                                             LINK$ADDRESS        word,
                                             BYTE$CNT            byte,
                                             FIFO$LIM            byte,
                                             SAV$BP$RDY          byte,
                                             LPBK$PRELEN$ADDRLEN byte,
                                             PRIORITY            byte,
                                             IF$SPACE            byte,
                                             SLOT$TIME           byte,
                                             RETRY$NUM$SLT$TMH   byte,
                                             MOD$CNTL            byte,
                                             FILTER              byte,
                                             MIN$FRAME$LEN       word) at(6FF82H);


                /* Multicust-ID Setup */

41    1         declare MC$SETUP$PTR        pointer,
                        MC$SETUP            structure
                                            (STAT                word,
                                             CMD                 word,
                                             LINK$ADDRESS        word,
                                             MC$CNT              word,
                                             MC$ID (6)           byte) at(6FF94H);


                /* Transmit Command */

42    1         declare TRANSMIT$PTR        pointer,
                        TRANSMIT            structure
                                            (STAT                word,
                                             CMD                 word,
                                             LINK$ADDRESS        word,
                                             BUF$DESC$PTR        word,
                                             DEST$ADDRESS (6)    byte,
                                             TYPE$FIELD          word) at(6FFA2H);


                /* Time Domain Reflectmeter Test */

43    1         declare TDR$PTR             pointer,
                        TDR                 structure
                                            (STAT                word,
                                             CMD                 word,
                                             LINK$ADDRESS        word,
                                             TIME                word) at(6FFB2H);


                /* Dump Status */

44    1         declare DUMP$STAT$PTR       pointer,
                        DUMP$STAT           structure
                                             (STAT               word,
                                             CMD                 word,
                                              LINK$ADDRESS       word,
                                              BUF$PTR            word) at(6FFBAH);
```

296170-22

Table 6. Software Listings (Continued)

```
                /* Diag */

 45   1             declare DIAG$PTR              pointer,
                      DIAG                      structure
                                                 (STAT                word,
                                                  CMD                 word,
                                                  LINK$ADDRESS        word) at(6FFC2H);



                /* Receive Buffer */

 46   1             declare RXb$PTR pointer,
                      RX  (10H)                 structure
                                                (BUF(80H) byte) at(6E000H);


                /* Transmit Buffer */

 47   1             declare TXB$PTR               pointer,
                      TX  (10H)                 structure
                                                (BUF(80H) byte) at(6E800H);

                /* Extra Transmit Command */

 48   1             declare EX$TRANSMIT$PTR       pointer,
 49   1             declare EX$TRANSMIT(10)       structure
                                                (STAT                word,
                                                 CMD                 word,
                                                 LINK$ADDRESS        word,
                                                 BD$PTR              word,
                                                 DEST$ADDRESS(6)     byte,
                                                 TYPE                word) at(6F500H);

                /* Channel Attention */

 50   1             CA: procedure;
 51   2                 call time(1);
 52   2                 output(CA$PTR) = 1;
 53   2             end CA;


                /* Acknowledge routine */

 54   1             ACK: procedure;

 55   2                 SCB.CMD = SCB.STAT and 0F000H;
 56   2                 call CA;

 57   2             end ACK;


                /* Command Block Initialization Procedure */


                /* NOP COMMAND */

 58   1             NOP$CMD: procedure;


 59   2                 NOP.STAT = 0;
 60   2                 NOP.CMD = 0A000H;
 61   2                 NOP.LINK$ADDRESS = 0FFFFH;
 62   2                 SCB.CMD = CMD$START;

 63   2             end NOP$CMD;


                /* INDIVIDUAL ADDRESS SET UP COMMAND */

 64   1             IA$SETUP$CMD: procedure;


 65   2                 IA$SETUP.STAT = 0;
 66   2                 IA$SETUP.CMD  = 0A001H;
 67   2                 IA$SETUP.LINK$ADDRESS = 0FFFFH;
 68   2                 IA$SETUP.IA$ADDRESS (0) = 0;        /* INDIVIDUAL ADDRESS  6BYTE */
 69   2                 IA$SETUP.IA$ADDRESS (1) = 0AAH;
 70   2                 IA$SETUP.IA$ADDRESS (2) = 0;
 71   2                 IA$SETUP.IA$ADDRESS (3) = 12H;
 72   2                 IA$SETUP.IA$ADDRESS (4) = 34H;
 73   2                 IA$SETUP.IA$ADDRESS (5) = 56H;
 74   2                 SCB.CMD = CMD$START;
```

296170-23

Table 6. Software Listings (Continued)

```
 75   2          end IA$SETUP$CMD;

                /* CONFIGURATION BLOCK INITIALIZATION */

 76   1          CONF$CMD: procedure;

 77   2              CONF.STAT = 0;                      /* Configuration command status,
                                                           it should be set to 0. */
 78   2              CONF.CMD = 0A002H;                  /* Configuration command word */
 79   2              CONF.LINK$ADDRESS = OFFFFH;         /* Next command link address */
 80   2              CONF.BYTE$CNT = 12;                 /* Configuration parameter byte
                                                           count */
 81   2              CONF.FIFO$LIM = 8;                  /* 8-bytes FIFO limit */
 82   2              CONF.SAV$BP$RDY = 0;
 83   2              CONF.LPBK$PRELEN$ADDRLEN = 26H;     /* 6-bytes address length,
                                                           8-bytes preamble length */
 84   2              CONF.PRIORITY = 0;
 85   2              CONF.IF$SPACE = 96;                 /* Interframe spacing 9.6 micro sec */
 86   2              CONF.SLOT$TIME = 0;
 87   2              CONF.RETRY$NUM$SLT$TMH = 0F2H;      /* Retry number =15, slot time 51.2
                                                           micro sec */
 88   2              CONF.MOD$CNTL = 1;                  /* Promiscuous mode */
 89   2              CONF.FILTER = 0;                    /* CRS and CDT filters not used  */
 90   2              CONF.MIN$FRAME$LEN = 64;            /* Minimum frame length 64-bytes */
 91   2              SCB.CMD = CMD$START;

 92   2          end CONF$CMD;

                /* MULTICAST-ID SETUP COMMAND */

 93   1          MC$SETUP$CMD: procedure;

 94   2              MC$SETUP.STAT = 0;                   /* it should be 0. */
 95   2              MC$SETUP.CMD = 0A003H;               /* Mulitcast command */
 96   2              MC$SETUP.LINK$ADDRESS = OFFFFH;
 97   2              MC$SETUP.MC$CNT = 6H;
 98   2              MC$SETUP.MC$ID (0) = 0FFH;           /* Mulitcast IDs */
 99   2              MC$SETUP.MC$ID (1) = 0FFH;
100   2              MC$SETUP.MC$ID (2) = 12H;
101   2              MC$SETUP.MC$ID (3) = 34H;
102   2              MC$SETUP.MC$ID (4) = 56H;
103   2              MC$SETUP.MC$ID (5) = 78H;
104   2              SCB.CMD = CMD$START;
105   2          end MC$SETUP$CMD;

                /* TRANSMIT COMMAND */

106   1          TX$CMD: procedure;

107   2              TRANSMIT.STAT = 0,
108   2              TRANSMIT.CMD  = 0A004H,
109   2              TRANSMIT.LINK$ADDRESS =0FFFFH,
110   2              TRANSMIT.BUF$DESC$PTR = TBD$OFFSET;
111   2              TRANSMIT.DEST$ADDRESS (0) = 00,   /* TRANSMIT DESTINATION ADDRESS */
112   2              TRANSMIT.DEST$ADDRESS (1) = 0AAH; /* 6BYTES                       */
113   2              TRANSMIT.DEST$ADDRESS (2) = 00,
114   2              TRANSMIT.DEST$ADDRESS (3) = 00;
115   2              TRANSMIT.DEST$ADDRESS (4) = 0AH;
116   2              TRANSMIT.DEST$ADDRESS (5) = 8EH;
117   2              TRANSMIT.TYPE$FIELD = 0;
118   2              SCB.CMD = CMD$START;

119   2          end TX$CMD;

                /* EXTRA TRANSMIT COMMAND */
120   1          EX$TX$CMD: procedure;
121   2              declare I                    byte,
122   2              do I = 0 to 9;
123   3                  EX$TRANSMIT(I).STAT = 0;
124   3                  EX$TRANSMIT(I).CMD  = 0A004H;
125   3                  EX$TRANSMIT(I).LINK$ADDRESS = OFFFFH;
126   3                  EX$TRANSMIT(I).BD$PTR = TBD$OFFSET;
127   3                  EX$TRANSMIT(I).DEST$ADDRESS(0) = 00;
128   3                  EX$TRANSMIT(I).DEST$ADDRESS(1) = 0AAH;
129   3                  EX$TRANSMIT(I).DEST$ADDRESS(2) = 00;
130   3                  EX$TRANSMIT(I).DEST$ADDRESS(3) = 00;
131   3                  EX$TRANSMIT(I).DEST$ADDRESS(4) = 0AH;
132   3                  EX$TRANSMIT(I).DEST$ADDRESS(5) = 8EH;
133   3                  EX$TRANSMIT(I).TYPE = I;
134   3              end;
135   2              SCB.CMD = CMD$START;
136   2          end EX$TX$CMD;                                              296170-24
```

Table 6. Software Listings (Continued)

```
                /* TIME DOMAIN REFLECTOMETER TEST COMMAND */

137   1             TDR$CMD: procedure;

138   2                 TDR.STAT = 0;
139   2                 TDR.CMD = 0A005H;
140   2                 TDR.LINK$ADDRESS = OFFFFH;
141   2                 TDR.TIME = 0;
142   2                 SCB.CMD = CMD$START;

143   2             end TDR$CMD;


                /* DUMP STATUS COMMAND */

144   1             DUMP$STAT$CMD: procedure;

145   2                 DUMP$STAT.STAT = 0;
146   2                 DUMP$STAT.CMD = 0A006H;
147   2                 DUMP$STAT.LINK$ADDRESS = OFFFFH;
148   2                 DUMP$STAT.BUF$PTR = DS$BUFF$OFFSET;     /* DUMP STATUS RESULT OFFSET */
149   2                 SCB.CMD = CMD$START;
150   2             end DUMP$STAT$CMD;


                /* Diagnose Command */

151   1             DIAG$CMD: procedure;

152   2                 DIAG.STAT = 0;
153   2                 DIAG.CMD  = 0A007H;
154   2                 DIAG.LINK$ADDRESS = OFFFFH;
155   2                 SCB.CMD = CMD$START;

156   2             end DIAG$CMD;




                /* Receive Frame Descriptor set up */

157   1             INIT$RFD: procedure;

158   2                 declare I                   word;
159   2                 call setb(0,@RFD(0).STAT,352);
160   2                 RFD (0).STAT = 0;
161   2                 RFD (0).LINK$ADDRESS = RFD$OFFSET + offset$of(@RFD (1));
162   2                 RFD (0).BD$PTR = RBD$OFFSET + offset$of(@RBD(0));

163   2                 do I = 1 to 14;
164   3                     RFD (I).STAT = 0;
165   3                     RFD (I).LINK$ADDRESS = RFD$OFFSET + offset$of(@RFD(I + 1));
166   3                     RFD (I).BD$PTR = OFFFFH;
167   3                 end;
168   2                 RFD (15).STAT = 0;
169   2                 RFD (15).LINK$ADDRESS = OFFFFH;
170   2                 RFD (15).BD$PTR = OFFFFH;
171   2                 RFD (15).EL$S = 8000H;

172   2             end INIT$RFD;


                /* Receive Buffer Descriptor set up

                   ------------     ------------                    --------------
              --->  | RBD(0)   | ---> | RBD(1)   | ---> ..... ---> | RBD(15)    |
                   ------------     ------------                    --------------
                   | EL = 0   |     | EL = 0   |                    | EL = 1     |
                   ------------     ------------                    --------------
                   | NBD=RB(0)|     | NBD=RB(1)|                    | NBD= FFFFH |
                   ------------     ------------                    --------------


                                                                                  */

173   1             INIT$RBD: procedure;


174   2                 declare I         word;

175   2             do I = 0 TO 14;
176   3                     RBD(I).ACT$COUNT = 0;
177   3                     RBD(I).NEXT$BD$ADD = RBD$OFFSET + offset$of(@RBD(I+1));
```

296170-25

Table 6. Software Listings (Continued)

```
178   3                 RBD(I).SIZE = 80H;
179   3           end;
180   2                 RBD(15).ACT$COUNT = 0;
181   2                 RBD(15).NEXT$BD$ADD = OFFFFH;
182   2                 RBD(15).SIZE = 8080H;
183   2           end INIT$RBD;

              /* Receive Buffer set up */

184   1           INIT$RB: procedure;

185   2               declare I                 word;
186   2               declare VALUE             byte;
187   2               declare COUNT             word;

188   2               VALUE = 0;
189   2               COUNT = 800H;
190   2               call setb(VALUE,@RX,COUNT); /* Full up receive buffer with 00h */

191   2               do I = 0 TO 15;
192   3                   RBD(I).BUFF$BASE = SCB$BASE;
193   3                   RBD(I).BUFF$OFFSET = RX$OFFSET + offset$of(@RX(I));
194   3               end;

195   2           end INIT$RB;



              /* Transmit Buffer Descriptor setup

                                 1) Setup actual count for Transmit Buffer Descriptor,
                                       and one frame is transmitted by one Transmit
                                       command.

                                 2) Setup next buffer descriptor address for each TBD
                                       in order to link with next TBD.

                    ------------      ------------                    ------------
              --->  | TBD (0)  | ---> | TBD (1)  | ---> ..... ---> | TBD (15)   |
                    ------------      ------------                    ------------
                    | EL = 0   |      | EL = 0   |                    | EL = 1     |
                    ------------      ------------                    ------------
                    | NBD=TB(0)|      | NBD=TB(1)|                    | NBD= FFFFH |
                    ------------      ------------                    --------------
                                                                                   */

196   1           INIT$TBD: procedure;

197   2               declare I             word;

198   2               do I = 0 TO 14;
199   3                   TBD(I).ACT$COUNT = 80H;
200   3                   TBD(I).NEXT$BD$ADD = TBD$OFFSET + offset$of(@tbd(i + 1));
201   3               end;
202   2               TBD(15).ACT$COUNT = 8080H;
203   2               TBD(15).NEXT$BD$ADD = OFFFFH;
204   2           end INIT$TBD;

              /* Transmit buffer set up

                                 1. Set up TBD
                                       -  each TBD has own transmit buffer
                                 2. Set up byte count of each buffer
                                       - 80h bytes
                                 3. Set up buffer data

                 */

205   1           INIT$TB: procedure;

206   2               declare I                 byte,
                              J                 byte,
                              NEW$VALUE         byte,
                              DEST$ADDR         pointer,
                              BYT$CNT           byte,
                              TXB$PTR           pointer;

207   2               do I = 0 TO 15;
208   3                   TBD(I).BUFF$BASE = SCB$BASE;
209   3                   TBD(I).BUFF$OFFSET = TX$OFFSET + offset$of(@TX(I));
210   3               end;
```

296170-26

Table 6. Software Listings (Continued)

```
211   2             J = 0;
212   2             do I = 0 TO 15;
213   3             byt$cnt = 80h;
214   3             NEW$VALUE = J;
215   3             DEST$ADDR = @TX(I);
216   3             call setb(NEW$VALUE,DEST$ADDR,BYT$CNT); /* Set up buffer data for
                                                              transmission          */


217   3             J = J + 01H;
218   3             end;

219   2          end INIT$TB;




                /* System I/O routine */

                /* Console input routine --- returns to the AL register an ASCII character
                                             received from the console device.  The AX, CX,
                                             and DX registers and the CPU condition codes
                                             are affected by this operation     */


220   1          CI: procedure byte external;
221   2              end CI;


                /* Console output routine --- transfers a character from the low-order byte
                                              of the word on the top of the stack to the
                                              console device.  The AX, CX, and DX registers
                                              and the CPU condition codes are affected by
                                              this operation.                          */

222   1          CO: procedure (CHARACTER) external;
223   2              declare CHARACTER       byte;
224   2          end CO;


                /* Exit routine --- causes a jump to the iAPX 86, 88 monitor command level.
                                    EXIT does not close any files.                    */


225   1          EXIT: procedure external;
226   2          end EXIT;

                /* Carriage  Return ---- Parameter  : none */

227   1          CR$LF:  procedure;
228   2                  call CO(0DH);
229   2                  call CO(0AH);
230   2          end  CR$LF;



                /* Out string routine --- transfers string to console device.  Null code
                                          (0h) found in the string, terminates the string
                                          transfer.                                    */

231   1          OUTS: procedure;
232   2              declare CHAR            byte;
233   2              declare I               byte;

234   2              I = 0;
235   2              CHAR = 0FFH;
236   2              do while CHAR <> 0;
237   3                       CHAR = MSG$BUF(I);
238   3                       call CO (CHAR);
239   3                       I = I + 1;
240   3              end;
241   2          end OUTS;

                /* Input string routine --- transfers string from console device.  The input
                                            code of carriage return (CR, 0DH) or line feed (LF,
                                            0AH) causes termination of this routine.
                                            This routine supports rubout and back space also. */
```

296170-27

Table 6. Software Listings (Continued)

```
 242   1          INS: procedure;
 243   2              declare CHAR                    byte;
 244   2              declare I                       byte;

 245   2              CHAR = 1OH;
 246   2              do I = 0 to 159;
 247   3                  C$BUF(I) = CHAR;
 248   3              end;

 249   2              I = 0;
 250   2              CHAR = 1OH;
 251   2              do while ((CHAR <> CR) and (CHAR <> LF) and (I <> 158));
 252   3                  CHAR = CI and 7FH;
 253   3                  if (char = del) or (CHAR = BS)
 254   3                  then do;
 255   4                      if I > 0
 256   4                      then do;
 257   5                          I = I - 1;
 258   5                          call CO(BS);
 259   5                          call CO(SP);
 260   5                          call CO(BS);
 261   5                      end;
 262   4                      else call CO(BEL);
 263   4                  end;
 264   3                  else if CHAR >= SP
 265   3                  then  do;
 266   4                      call CO(CHAR);
 267   4                      if ((CHAR >= 'a') and (CHAR <= 'z'))
 268   4                      then C$BUF(I) = CHAR - 20H;
 269   4                      else C$BUF(I) = CHAR;
 270   4                      I = I + 1;
 271   4                  end;
 272   3                  else if ((CHAR = CR) or (CHAR = LF))
 273   3                  then do;
 274   4                      C$BUF(I) = CR;
 275   4                      C$BUF(I + 1) = LF;
 276   4                      I = I + 2;
 277   4                  end;
 278   3                  else  call CO(BEL);
 279   3              end;
 280   2          end INS;


 281   1          IN$YES: procedure byte;

 282   2              declare CHAR            byte;
 283   2              CHAR = 0;
 284   2              do while ((CHAR <> 'Y' or CHAR <> 'y')
                           and (CHAR <> 'N' or CHAR <> 'n'));

 285   3                  CHAR = CI and 7FH;
 286   3                  call CO(CHAR);
 287   3                  if (CHAR = 'Y' or CHAR = 'y') then
 288   3                      return TRUE;
 289   3                  else if (CHAR = 'N' or CHAR = 'n') then
 290   3                      return FALSE;
 291   3                  else do;
 292   4                      MSG$PTR = @(ODH, OAH, '** KEY IN Y or N', ODH, OAH, 2AH, 0);
 293   4                      call OUTS;
 294   4                  end;
 295   3              end;
 296   2          end IN$YES;


              /* Software chip reset

                                      Name:       CHIP$RESET
                                      Input:      None
                                      Output:     None
                                      Function:   This routine performs 82586 chip reset
                                                  by software.  It is same as Hardware
                                                  RESET.                                  */

 297   1          CHIP$RESET: procedure;
 298   2              SCB.CMD = 80H;
 299   2              call CA;
 300   2          end CHIP$RESET;
```

296170-28

Table 6. Software Listings (Continued)

```
                /* System Initialize */

 301    1           INIT$SYS: procedure;


 302    2           call setb(0,@rx,2000h);
 303    2           CMD$TOP$STATUS = FALSE;

                /* Pointer Declare */

 304    2                SCP$PTR = build$ptr(COMMON$BASE, SCP$OFFSET);
 305    2                ISCP$PTR = build$ptr(COMMON$BASE, ISCP$OFFSET);
 306    2                SCB$PTR = build$ptr(COMMON$BASE, SCB$OFFSET);
 307    2                NOP$PTR = build$ptr(COMMON$BASE, NOP$OFFSET);
 308    2                IA$SETUP$PTR = build$ptr(COMMON$BASE, IA$SETUP$OFFSET);
 309    2                CONF$PTR = build$ptr(COMMON$BASE, CONF$OFFSET);
 310    2                MC$SETUP$PTR = build$ptr(COMMON$BASE, MC$SETUP$OFFSET);
 311    2                TRANSMIT$PTR = build$ptr(COMMON$BASE, TRANSMIT$OFFSET);
 312    2                TDR$PTR = build$ptr(COMMON$BASE, TDR$OFFSET);
 313    2                DUMP$STAT$PTR = build$ptr(COMMON$BASE, DUMP$STAT$OFFSET);
 314    2                DIAG$PTR = build$ptr(COMMON$BASE, DIAG$OFFSET);
 315    2                RFD$PTR = build$ptr(COMMON$BASE, RFD$OFFSET);
 316    2                RBD$PTR = build$ptr(COMMON$BASE, RBD$OFFSET);
 317    2                RXB$PTR = build$ptr(COMMON$BASE, RX$OFFSET);
 318    2                TBD$PTR = build$ptr(COMMON$BASE, TBD$OFFSET);
 319    2                TXB$PTR = build$ptr(COMMON$BASE, TX$OFFSET);
 320    2                EX$TRANSMIT$PTR = build$ptr(COMMON$BASE, EX$TRANSMIT$OFFSET);

                /* SCP initialization */

 321    2                SCP.SYSBUS = 0H;            /* System bus width - 16 bits. */
 322    2                SCP.ISCP$BASE = SCB$BASE;   /* ISCP start address 24 - bits */
 323    2                SCP.ISCP$OFFSET = ISCP$OFFSET;

                /* ISCP initialization */

 324    2                ISCP.BUSY = 01H;            /* This words set by CPU to 01H,
                                                        82586 clears it. */
 325    2                ISCP.SCB$OFFSET = offset$of(SCB$PTR);   /* SCB start address offset */
 326    2                ISCP.SCB$BASE1 = 0H;        /* Lower word of SCB base address. */
 327    2                ISCP.SCB$BASE2 = 06H;       /* MS-byte of SCB base address. */

                /* SCB initalization */


 328    2                SCB.STAT = 0H;              /* SCB STATUS word, it should be set to 0. */
 329    2                SCB.CMD  = 0H;              /* SCB Command word */
 330    2                SCB.CBL$OFFSET = CONF$OFFSET;
                                                    /* Command block list offset, the address
                                                       of command block list is the summation
                                                       of SCB BASE  and OFFSET value. */

 331    2                SCB.RFD$OFFSET = offset$of(RFD$PTR); /* Receive frame area offset */
 332    2                SCB.CRC$ERRS = 0;           /* CRC Error counter */
 333    2                SCB.ALN$ERRS = 0;           /* Alignment error counter */
 334    2                SCB.RSC$ERRS = 0;           /* No resource error counter (# of frames
                                                       were lost by no resource errors) */
 335    2                SCB.OVRN$ERRS = 0;          /* DMA OVER-RUN error counter */
 336    2                call CA;                    /* NOP command execution */
 337    2                call CHIP$RESET;            /* Chip reset */
 338    2                call ACK;                   /* Acknowlidge for Chip reset */
 339    2                call CONF$CMD;              /* Set configuration command parameter */
 340    2                CONF.CMD = 0002H;           /* EL = 0, S = 0, I = 0, CMD = CONF */
 341    2                CONF.LINK$ADDRESS = IA$SETUP$OFFSET; /* IA setup cmd for next cmd block */
 342    2                call IA$SETUP$CMD;          /* IA setup cmd parameter set */
 343    2                IA$SETUP.CMD = 0A001H;      /* IA command set */
 344    2                NEXT$CMD$OFFSET = OFFFFH;   /* Next command offset for last cmd block */
 345    2                call CA;                    /* Execute them */
 346    2           end INIT$SYS;

                /* Message for HELP command */

 347    1           COMMAND$MSG: procedure;


 348    2                call CR$LF;
 349    2                MSG$PTR = @('NOP -------------------- NOP
                                     INDIVIDUAL ADDR SETUP - IA ', 0DH, 0AH,
                                     'CONFIGURATION -------- CONF
                                     MULTICAST-ID ---------- MC ', 0DH, 0AH, 0);
 350    2                call OUTS;
 351    2                MSG$PTR = @('TRANSMI" -------------- TX
                                     TDF -------------------- ;DR', 0DH, 0AH,
                                     'DUMP STATUS ----------- DS
                                     DIAGNOSE -------------- DIAG', 0DH, 0AH, 0);
```

296170-29

Table 6. Software Listings (Continued)

```
352   2             call OUTS;

353   2         end COMMAND$MSG;



354   1         BD$MSG: procedure;
355   2             MSG$PTR = @('RX FRAME DESCRIPTOR --- RFD',ODH, OAH,
                              'RX BUFFER DESCRIPTOR -- RBD
                             RX BUFFER -------------- RB',ODH,OAH,0);
356   2             call OUTS;
357   2             MSG$PTR = @('TX BUFFER DESCRIPTOR --- TBD
                             TX BUFFER -------------- TB',ODH,OAH,0);
358   2         end BD$MSG;

              /* Message for illigal input */

359   1         MSG$ILL$CMD: procedure;
360   2                 MSG$PTR = @('     <---- ILLEGAL INPUT', ODH, OAH, 0);
361   2                 call OUTS;
362   2         end MSG$ILL$CMD;




              /* Integer to ASCII code conversion

                                 Name:        INT$TO$ASCII
                                 Input:       INT$BUF     Integer buffer (1 byte);
                                 Output:      ASCII$BUF   ASCII buffer (2 bytes);
                                 Function:    INTEGER TO ASCII conversion */



363   1         INT$TO$ASCII: procedure;

364   2             declare TEMP$CHAR      byte;

365   2                 TEMP$CHAR = INT$BUF;
366   2                 TEMP$CHAR = TEMP$CHAR and OFH;
367   2                 if TEMP$CHAR < 10
368   2                     then ASCII$BUF(0) = TEMP$CHAR + 30H;
369   2                 else if (10 <= TEMP$CHAR) and (TEMP$CHAR <= OFH)
370   2                     then ASCII$BUF(0) = TEMP$CHAR + 37H;

371   2                     TEMP$CHAR = shr((INT$BUF and OFOH), 4);
372   2                 if TEMP$CHAR < 10
373   2                     then ASCII$BUF(1) = TEMP$CHAR + 30H;
374   2                     else if (10 <= TEMP$CHAR) and (TEMP$CHAR <= OFH)
375   2                     then ASCII$BUF(1) = TEMP$CHAR + 37H;

376   2         end INT$TO$ASCII;

              /* ASCII code to integer value conversion

                                 Name:        ASCII$TO$INT
                                 Input:       ASCII$BUF    (2 bytes)
                                 Output:      INT$BUF      (1 byte)
                                 Function:    ASCII to INTEGER conversion */

377   1         ASCII$TO$INT: procedure;
378   2             declare TEMP$CHAR (2)           byte;

379   2                     TEMP$CHAR (0) = ASCII$BUF(0);
380   2                     if ('0' <= TEMP$CHAR(0)) and (TEMP$CHAR(0) <= '9')
381   2                         then  TEMP$CHAR(0) = (TEMP$CHAR(0) - 30H);
382   2                     else if ('A' <= TEMP$CHAR(0)) and (TEMP$CHAR(0) <= 'F')
383   2                         then  TEMP$CHAR(0) = TEMP$CHAR(0) - 37H;
384   2                     TEMP$CHAR (1) = ASCII$BUF(1);
385   2                     if ('0' <= TEMP$CHAR(1)) and (TEMP$CHAR(1) <= '9')
386   2                         then  TEMP$CHAR(1) = TEMP$CHAR(1) - 30H;
387   2                     else if ('A' <= TEMP$CHAR(1)) and (TEMP$CHAR(1) <= 'F')
388   2                         then  TEMP$CHAR(1) = TEMP$CHAR(1) - 37H;
389   2                 INT$BUF = (TEMP$CHAR(0) and OFH) or shl((TEMP$CHAR(1) and OFH), 4);

390   2         end ASCII$TO$INT;


391   1         DEC$TO$HEX: procedure(NUMBER) word;

392   2             declare NUMBER      word;
393   2             declare TEMP$WORD0  word;
394   2             declare TEMP$WORD1  word;
```

296170-30

Table 6. Software Listings (Continued)

```
395   2                 TEMP$WORD0 = shr((NUMBER and 0F000H),12);
396   2                 TEMP$WORD1 = TEMP$WORD0 * 1000;
397   2                 TEMP$WORD0 = shr((NUMBER and 0F00H),8);
398   2                 TEMP$WORD1 = TEMP$WORD1 + (TEMP$WORD0 * 100);
399   2                 TEMP$WORD0 = shr((NUMBER and 0F0H),4);
400   2                 TEMP$WORD1 = TEMP$WORD1 + (TEMP$WORD0 * 10);
401   2                 TEMP$WORD0 = NUMBER and 0FH;
402   2                 TEMP$WORD1 = TEMP$WORD1 + TEMP$WORD0;
403   2                 return TEMP$WORD1;
404   2           end DEC$TO$HEX;


405   1           INPUT$WORD: procedure word;

406   2               declare IN$WORD     word;
407   2               declare I           byte;

408   2               call INS;
409   2               I, IN$WORD = 0;
410   2               do while (I < 4 and C$BUF(I) <> CR and C$BUF(I) <> LF);
411   3                   ASCII$BUF(0) = C$BUF(I);
412   3                   ASCII$BUF(1) = 0;
413   3                   call ASCII$TO$INT;
414   3                   IN$WORD = shl(IN$WORD,4) or double(INT$BUF);
415   3                   I = I + 1;
416   3               end;
417   2               return IN$WORD;
418   2           end INPUT$WORD;


            /* Read 1 byte data from Memory

                                Name:       MEM$BYTE$READ
                                Input:      CMD$OFFSET
                                Output:     MSG$BUF (console out)
                                Function:   Output one byte data stored in memory to
                                            console                                  */


419   1           MEM$BYTE$READ: procedure (CMD$OFFSET);
420   2               declare
                              CMD$DAT based CMD$PTR   byte,
                              COUNT                   byte,
                              CMD$OFFSET              word;

421   2                   CMD$PTR = build$ptr(COMMON$BASE,CMD$OFFSET);
422   2                   DAT = CMD$DAT;
423   2                   INT$BUF = CMD$DAT;
424   2                   call INT$TO$ASCII;
425   2                   call CO(ASCII$BUF(1));
426   2                   call CO(ASCII$BUF(0));

427   2           end MEM$BYTE$READ;


            /* Receive buffer data display

                                Name:       BUFFER$DISP
                                Input:      C$BUF (console in)
                                Output:     MSG$BUF (console out)
                                Function:   Display Receive buffer data to console */


428   1           BUFFER$DISP: procedure;
429   2               declare BYTE$COUNT          byte;
430   2               declare TEMP$CMD$OFFSET     word;
431   2               declare I                   byte;
432   2               declare J                   byte;

433   2       LOOP7:  MSG$PTR = @(0DH,0AH,'RXB> KEY IN THE BUFFER DESCRIPTOR NUMBER (0 - F: 128 byte unit
                        )',0DH, 0AH,2AH,0);
434   2                   call OUTS;
435   2       LOOP8:      call INS;
436   2                   call CR$LF;
437   2                   ASCII$BUF(0) = C$BUF(0);
438   2                   ASCII$BUF(1) = 0;
439   2                   call ASCII$TO$INT;
440   2                   if INT$BUF <= 15 then do;
442   3                       TEMP$CMD$OFFSET = RX$OFFSET + (INT$BUF * 80h);
443   3                       do I = 0 to 7;
444   4                           INT$BUF = I;
445   4                           call INT$TO$ASCII;
446   4                           call CO(C$BUF(0));
447   4                           call CO(':');
```

296170-31

Table 6. Software Listings (Continued)

```
448   4                        call CO(ASCII$BUF(0));
449   4                        call CO('0');
450   4                        call CO(SP);
451   4                        do J = 0 to 15;
452   5                            call MEM$BYTE$READ(TEMP$CMD$OFFSET);
453   5                            call CO(SP);
454   5                            TEMP$CMD$OFFSET = TEMP$CMD$OFFSET + 1;
455   5                        end;
456   4                        call CR$LF;
457   4                    end;
458   3                end;
459   2                else do;
460   3                call MSG$ILL$CMD;
461   3                MSG$PTR = @(ODH, OAH, 'RXB> KEY IN 0 --- F(H) !!!', ODH, OAH, 2AH, 0);
462   3                goto LOOP8;
463   3            end;
464   2            MSG$PTR = @(ODH, OAH, 'RXB> MORE RXB ? (Y or N)', ODH, OAH, 2AH, 0);
465   2            call OUTS;
466   2            if IN$YES then goto LOOP7;
468   2        end BUFFER$DISP;



            /* DUMP STATUS  status display

                                Name:       DS$BUFF$DISP
                                Input:      C$BUF (console in)
                                Output:     MSG$BUF (console out)
                                Function:   Display DUMP STATUS contents to Console */

469   1         DS$BUFF$DISP: procedure;
470   2             declare TEMP$CMD$OFFSET         word,
                          I                         byte,
                          J                         byte;

471   2             call CR$LF;
472   2             TEMP$CMD$OFFSET = DS$BUFF$OFFSET;
473   2             do I = 0 to 10;
474   3                 INT$BUF = I;
475   3                 call INT$TO$ASCII;
476   3                 call CO(ASCII$BUF(1));
477   3                 call CO(ASCII$BUF(0));
478   3                 call CO('0');
479   3                 call CO(SP);
480   3                 do J = 0 to 15;
481   4                     call MEM$BYTE$READ(TEMP$CMD$OFFSET);
482   4                     call CO(SP);
483   4                     TEMP$CMD$OFFSET = TEMP$CMD$OFFSET + 1;
484   4                 end;
485   3                 call CR$LF;
486   3             end;
487   2         end DS$BUFF$DISP;

            /* Transmit data change

                                Name:       TRANSMIT$DATA$CHANGE
                                Input:      C$BUF (console in)
                                Output:     Transmit buffer
                                Function:   Determine the transmit buffer size
                                            (128 bytes unit) and data */

488   1         TRANSMIT$DATA$CHANGE: procedure;
489   2             declare I                   byte;
490   2             declare BYTE$DATA           byte;
491   2             declare BUFFER$NUMBER       byte;
492   2             declare BUFFER$POINTER      pointer;
493   2             declare C$COUNT             word;
494   2             declare TEMP                word;

495   2         LOOP:   MSG$PTR = @(ODH, OAH, 'TXB> DO YOU WANT TO CHANGE THE TRANSMIT DATA ?
                            (Y or N)', ODH, OAH, 2AH, 0);
496   2                 call OUTS;
497   2                 if IN$YES then
498   2                 do;
499   3                     MSG$PTR = @(ODH, OAH, 'TXB> DATA CHANGE: KEY IN THE BUFFER NUMBER
                                (0 - F: 128 bytes unit)', ODH, OAH, 2AH, 0);
500   3                     call OUTS;
501   3                     INT$BUF = LOW(INPUT$WORD);
502   3                     BUFFER$NUMBER = INT$BUF and OFH;
503   3                     BUFFER$POINTER = @TX(BUFFER$NUMBER);
504   3                     MSG$PTR = @(ODH, OAH, 'TXB> KEY IN THE VALUE (00 - FF)', ODH, OAH, 2AH, 0)
505   3                     call OUTS;
506   3                     INT$BUF = low(INPUT$WORD);
507   3                     call setb(INT$BUF, BUFFER$POINTER, 80H);
508   3                     MSG$PTR = @(ODH, OAH, 'TXB> DATA WAS WRITTEN', ODH, OAH, 0);
509   3                     call OUTS;
```

296170-32

Table 6. Software Listings (Continued)

```
510   3                    call CR$LF;
511   3                    goto LOOP;
512   3                end;
513   2                else do;
514   3                    MSG$PTR = @(0DH, 0AH, 'TXB> HOW MANY BYTE(S) DO YOU NEED ?',
                               0);
515   3                    call OUTS;
516   3                    MSG$PTR = @(0DH, 0AH, '      KEY IN THE DECIMAL NUMBER'
                                   ,0DH, 0AH, 2AH, 0);
517   3                    call OUTS;
518   3                    TEMP = INPUT$WORD;
519   3                    C$COUNT = DEC$TO$HEX(TEMP);
520   3                    I = 0;
521   3                    do while (C$COUNT > 80H and C$COUNT <= 2048);
522   4                        C$COUNT = C$COUNT - 80H;
523   4                        TBD(I).ACT$COUNT = 80H;
524   4                        I = I + 1;
525   4                    end;
526   3                    if C$COUNT <= 80H
527   3                        then TBD(I).ACT$COUNT = 8000H + C$COUNT;
528   3                    else if C$COUNT > 2048 then
529   3                    do;
530   4                        call CR$LF;
531   4                        MSG$PTR = @('TXB> BYTE COUNT IS LIMITED UP TO 2048',0DH,
                                       0AH, 0);
532   4                        call OUTS;
533   4                    end;
534   3                end;

535   2            end TRANSMIT$DATA$CHANGE;

              /* Display command word contents

                                   Name:        DISP$CONTENTS
                                   Input:       C$BUF (console in)
                                   Output:      MSG$BUF (console out)
                                   Function:    Display the command word contents to console */

536   1            DISP$CONTENTS: procedure;

537   2                declare DISP$OFFSET          word;
538   2                declare BYTE$COUNT           byte;
539   2                declare J                    byte;

540   2                MSG$PTR = @(0DH, 0AH, 'CSET> DO YOU WANT TO SEE THE CONTENTS ? (Y or N)',0DH,
                           0AH, 2AH, 0);
541   2                call OUTS;
542   2                DISP$OFFSET = MESSAGE$WORD;

543   2                if IN$YES then
544   2                do;
545   3                    J = (MESSAGE$BYTE - 2)/2;
546   3                    do BYTE$COUNT = 0 to J;
547   4                        call CR$LF;
548   4                        if MESSAGE$STATUS = TRUE then
549   4                        do;
550   5                            MESSAGE$STATUS = FALSE;
551   5                            MSG$PTR = @('WORD ',0);
552   5                            call OUTS;
553   5                            INT$BUF = BYTE$COUNT;
554   5                            call INT$TO$ASCII;
555   5                            call CO(ASCII$BUF(0));
556   5                            call CO(SP);
557   5                            call CO('=');
558   5                            call CO(SP);
559   5                            DISP$OFFSET = MESSAGE$WORD + 1;
560   5                            call MEM$BYTE$READ(DISP$OFFSET);
561   5                            DISP$OFFSET = DISP$OFFSET - 1;
562   5                            call MEM$BYTE$READ(DISP$OFFSET);
563   5                            call CO('H');
564   5                        end;
565   4                        else do;
566   5                            MSG$PTR = @('WORD ',0);
567   5                            call OUTS;
568   5                            INT$BUF = BYTE$COUNT;
569   5                            call INT$TO$ASCII;
570   5                            call CO(ASCII$BUF(0));
571   5                            call CO(SP);
572   5                            call CO('=');
573   5                            call CO(SP);
574   5                            DISP$OFFSET = DISP$OFFSET + 3;
575   5                            call MEM$BYTE$READ(DISP$OFFSET);
576   5                            DISP$OFFSET = DISP$OFFSET - 1;
577   5                            call MEM$BYTE$READ(DISP$OFFSET);
578   5                            call CO('H');
579   5                        end;
```

296170-33

Table 6. Software Listings (Continued)

```
590   4                 end;
591   3             end;
582   2         end DISP$CONTENTS;


          /* SCB STATUS, ACK field display

                                Name:       SCB$BITMAP$DISPLAY
                                Input:      SCB.STAT,SCB.CMD
                                Output:     MSG$BUF (console out)
                                Function:   Display the SCB STATUS, ACK field
                                            to console as binaly value */


583   1         SCB$BITMAP$DISPLAY: procedure;
584   2             declare I                       byte;

585   2             do I = 0 to 3;
586   3                 if rol(MESSAGE$WORD,I + 1)  then call CO(31H);
588   3                 else call CO(30H);
589   3             end;

590   2         end SCB$BITMAP$DISPLAY;


          /* SCB COMMAND/STATUS field display

                                Name:       SCB$UNIT$DISPLAY
                                Input:      SCB.STAT,SCB.CMD
                                Output:     MSG$BUF (console out)
                                Function:   Display the CUS,RUS,CUC and RUC field
                                            to console as BCD value */

591   1         SCB$UNIT$DISPLAY: procedure;
592   2             declare SHIFT$COUNT             byte;

593   2             if MESSAGE$STATUS = 0 then SHIFT$COUNT = 8;
595   2             else SHIFT$COUNT = 4;
596   2             INT$BUF = low(shr(MESSAGE$WORD,SHIFT$COUNT) and 7);
597   2             call INT$TO$ASCII;
598   2             call CO(ASCII$BUF(1));
599   2             call CO(ASCII$BUF(0));

600   2         end SCB$UNIT$DISPLAY;


          /* Word contents display

                                Name:       WORD$DISP
                                Input:      MESSAGE$WORD
                                Output:     MSG$BUF (console out)
                                Function:   display the word value */


601   1         WORD$DISP:  procedure;
602   2             declare I               byte;

603   2                 INT$BUF = high(MESSAGE$WORD);
604   2                 call INT$TO$ASCII;
605   2                 call CO(ASCII$BUF(1));
606   2                 call CO(ASCII$BUF(0));
607   2                 INT$BUF = low(MESSAGE$WORD);
608   2                 call INT$TO$ASCII;
609   2                 call CO(ASCII$BUF(1));
610   2                 call CO(ASCII$BUF(0));

611   2         end WORD$DISP;


          /*change the link command condition

                                Name:       CHANGE
                                Input:      C$BUF (console in)
                                Output:     Command blocks
                                Function:   Change the command linking and command block
                                            parameter change                    */

612   1         CHANGE: procedure;

613   2             declare CHANGE$DATA$BYTE        word;
614   2             declare CHANGE$PTR              pointer;
615   2             declare CHANGE$OFFSET           word;
616   2             declare BYTE$COUNT              byte;
617   2             declare J                       byte;
618   2             declare TMP$WORD                word;
```

296170-34

Table 6. Software Listings (Continued)

```
619   2          TMP$WORD = MESSAGE$WORD;
620   2          call CR$LF;
621   2          MSG$PTR = @('CSET> DO YOU NEED CHANGE THIS BLOCK ? (Y or N)',ODH,OAH,2AH,0);
622   2          call OUTS;
623   2          if IN$YES then
624   2              do;
625   3                  CHANGE$OFFSET = MESSAGE$WORD;
626   3                  MESSAGE$STATUS = FALSE;
627   3                  do BYTE$COUNT = 0 to (MESSAGE$BYTE - 2) by 2;
628   4                      CHANGE$PTR = build$ptr(COMMON$BASE,CHANGE$OFFSET);
629   4                      call CR$LF;
630   4                      MSG$PTR = @('WORD ',0);
631   4                      call OUTS;
632   4                      INT$BUF = BYTE$COUNT / 2;
633   4                      call INT$TO$ASCII;
634   4                      call CO(ASCII$BUF(0));
635   4                      call CO(SP);
636   4                      call CO('=');
637   4                      call CO(SP);
638   4                      call movw(CHANGE$PTR,@MESSAGE$WORD,1);
639   4                      call WORD$DISP;
640   4                      MSG$PTR = @(20H,'<--- DO YOU NEED CHANGE THIS WORD?
                                  (Y or N)',ODH,OAH,2AH,0);
641   4                      call OUTS;
642   4                      if IN$YES then
643   4                      do;
644   5                          MSG$PTR = @(ODH,OAH,'CSET> WHAT IS THE VALUE ?  (FOUR DIGIT)',
                                    ODH,OAH,2AH,0);
645   5                          call OUTS;
646   5                          CHANGE$DATA$BYTE = INPUT$WORD;
647   5                          call setw(CHANGE$DATA$BYTE,CHANGE$PTR,1);
648   5                      end;
649   4                      CHANGE$OFFSET = CHANGE$OFFSET + 2;
650   4                  end;
651   3              end;
652   2              MESSAGE$WORD = TMP$WORD;
653   2          end CHANGE;


              /* Command set

                                  Name:       LINK$ADDRESS$SET
                                  Input:      CURRENT$CMD$OFFSET, NEXT$CMD$OFFSET
                                  Output:     CURRENT$CMD$OFFSET, NEXT$CMD$OFFSET
                                  Function:   Set the command link (Max. 2 commands)
                                              this routine sets the following parameter:
                                                  a: link address set
                                                  b: EL bit of previous command is reset
                                                  c: EL bit of next command is set */

654   1          LINK$ADDRESS$SET: procedure;

655   2              declare TEMP$PTR                    pointer;
656   2              declare TEMP$WORD based TEMP$PTR    word;
657   2              declare TEMP$OFFSET                 word;

658   2              if CMD$TOP$STATUS = FALSE then do;
660   3                  CMD$TOP$OFFSET = NEXT$CMD$OFFSET;
661   3                  CMD$TOP$STATUS = TRUE;
662   3              end;
663   2              else do;

                      /* link address set */

664   3              TEMP$OFFSET = CURRENT$CMD$OFFSET + 4;
665   3              TEMP$PTR = build$ptr(COMMON$BASE, TEMP$OFFSET);
666   3              TEMP$WORD = NEXT$CMD$OFFSET;


                      /* current$command EL-bit clear */

667   3              TEMP$OFFSET = CURRENT$CMD$OFFSET +2;
668   3              TEMP$PTR = build$ptr(COMMON$BASE, TEMP$OFFSET);
669   3              TEMP$WORD = TEMP$WORD and 7fffh;

                      /* next command EL-bit set */

670   3              TEMP$OFFSET = MESSAGE$WORD  + 2;
671   3              TEMP$PTR = build$ptr(COMMON$BASE, TEMP$OFFSET);
672   3              TEMP$WORD = TEMP$WORD or 8000h;
673   3              end;
674   2          end LINK$ADDRESS$SET;
```

296170-35

Table 6. Software Listings (Continued)

```
                /* Transmit Command Link */

675   1             TRANSMIT$COMMAND$LINK: procedure;

676   2                 declare I                               word;

677   2                 CURRENT$CMD$OFFSET = NEXT$CMD$OFFSET;
678   2                 NEXT$CMD$OFFSET = MESSAGE$WORD;
679   2                 call LINK$ADDRESS$SET;
680   2                 call CHANGE;
681   2                 MESSAGE$STATUS = TRUE;
682   2                 call DISP$CONTENTS;
683   2                 I = 0;
684   2           LOOP6: if I <= 9 then do;
686   3                     MSG$PTR  = @(0DH, 0AH, 'CSET> MORE TRANSMIT COMMAND ? (Y or N)',
                                  0DH, 0AH, 2AH, 0);
687   3                     call OUTS;
688   3                     if IN$YES then do;
690   4                         MESSAGE$WORD = 0F500H + 16 * I;
691   4                         CURRENT$CMD$OFFSET = NEXT$CMD$OFFSET;
692   4                         NEXT$CMD$OFFSET = MESSAGE$WORD;
693   4                         call LINK$ADDRESS$SET;
694   4                         call CHANGE;
695   4                         MESSAGE$STATUS = TRUE;
696   4                         call DISP$CONTENTS;
697   4                         I = I + 1;
698   4                         goto LOOP6;
699   4                     end;
700   3                     else do;
701   4                         call CR$LF;
702   4                         INT$BUF = I + 1;
703   4                         call INT$TO$ASCII;
704   4                         call CO(ASCII$BUF(0));
705   4                         MSG$PTR = @(' TRANSMIT COMMAND WERE SET', 0DH, 0AH, 0);
706   4                         call OUTS;
707   4                     end;
708   3                 end;
709   2                 else do;
710   3                     EX$TRANSMIT(9).CMD = 0A004H;
711   3                     EX$TRANSMIT(9).LINK$ADDRESS = 0FFFFH;
712   3                     EX$TRANSMIT(9).BD$PTR = 0F100H;
713   3                     MSG$PTR = @(0DH, 0AH
                                  , 'TRANSMIT COMMAND LINKS ARE LIMITED UP TO 10 !!!'
                                  , 0DH, 0AH, 0);
714   3                     call OUTS;
715   3                 end;
716   2             end TRANSMIT$COMMAND$LINK;


                /* Command link check

                                Name:       COMMAND$LINK$CHECK
                                Input:      MESSAGE$WORD
                                Output:     LINK$ADDR
                                Function:   Check the command link and pass the command
                                            link condition to command set routine */

717   1             COMMAND$LINK$CHECK: procedure;
718   2                 declare TEMP$CMD$OFFSET     word,
                              TEMP$CMD$WORD       word;

719   2                 TEMP$CMD$WORD = MESSAGE$WORD;

720   2                 if NEXT$CMD$OFFSET = MESSAGE$WORD then
721   2                 do;
722   3                     MSG$PTR = @(0DH, 0AH, 'CSET> DO YOU WANT TO EXECUTE THE SAME COMMAND ? (Y or N)',
                                  0DH, 0AH, 2AH, 0);
723   3                     call OUTS;
724   3                     if IN$YES then
725   3                     do;
726   4                         call LINK$ADDRESS$SET;
727   4                         CURRENT$CMD$OFFSET = NEXT$CMD$OFFSET;
728   4                         NEXT$CMD$OFFSET = MESSAGE$WORD;
729   4                     end;
730   3                     else do;
731   4                         MSG$PTR = @(0DH, 0AH, 'CSET> COMMAND LINK INPUT WAS CANCELLED',
                                      0DH, 0AH, 0);
732   4                         call OUTS;
733   4                     end;
734   3                 end;
735   2                 else do;
736   3                     CURRENT$CMD$OFFSET = NEXT$CMD$OFFSET;
737   3                     NEXT$CMD$OFFSET = MESSAGE$WORD;
738   3                     call LINK$ADDRESS$SET;
739   3                 end;
740   2                 MESSAGE$WORD = TEMP$CMD$WORD;                                    296170-36
```

Table 6. Software Listings (Continued)

```
741   2          end COMMAND$LINK$CHECK;


742   1          MORE$COMMAND: procedure;

743   2              MSG$PTR = @(ODH, OAH, 'CSET> DO YOU WANT MORE COMMAND ?
                          (Y or N)', ODH, OAH, 2AH, 0);
744   2              call OUTS;
745   2          end MORE$COMMAND;


              /* Command set

                                   Name:        COMMAND$SET
                                   Input:       C$BUF (console in)
                                   Output:      MSG$WORD, MSG$BYTE and MSG$STATUS
                                   Function:    Interpret the Command set input from
                                                console and pass to command parameter set
                                                routine                                  */

746   1          COMMAND$SET: procedure;

747   2              declare BYTE$COUNT      byte;
748   2              declare J               byte;
749   2          LOOP1:   MSG$PTR = @(ODH, OAH, 'CSET> WHAT IS THE COMMAND?', ODH, OAH, 'KEY IN H FOR HELP',
                       ODH, OAH, 2AH, 0);
750   2              call OUTS;
751   2              call INS;
752   2              if (cmpb (@C$BUF(0), @('NOP', ODH), 4) = TRUEW)
753   2                  then do;
754   3                      MESSAGE$STATUS = TRUE;
755   3                      MESSAGE$WORD = NOP$OFFSET;
756   3                      MESSAGE$BYTE = 6;
757   3                      call NOP$CMD;
758   3                      call COMMAND$LINK$CHECK;
759   3                      call DISP$CONTENTS;
760   3                      call MORE$COMMAND;
761   3                      if IN$YES then goto LOOP1;
763   3                  end;


764   2              else do;
765   3                  if (cmpb (@C$BUF(0), @('IA', ODH), 3) = TRUEW)
766   3                  then do;
767   4                      MESSAGE$STATUS = TRUE;
768   4                      MESSAGE$WORD = IA$SETUP$OFFSET;
769   4                      MESSAGE$BYTE = 12;
770   4                      call IA$SETUP$CMD;
771   4                      call COMMAND$LINK$CHECK;
772   4                      call CHANGE;
773   4                      MESSAGE$STATUS = TRUE;
774   4                      call DISP$CONTENTS;
775   4                      call MORE$COMMAND;
776   4                      if IN$YES then goto LOOP1;
778   4                  end;

779   3              else do;
780   4                  if (cmpb (@C$BUF(0), @('CONF', ODH), 5) = TRUEW)
781   4                  then do;
782   5                      MESSAGE$STATUS = TRUE;
783   5                      MESSAGE$WORD = CONF$OFFSET;
784   5                      MESSAGE$BYTE = 18;
785   5                      call CONF$CMD;
786   5                      call COMMAND$LINK$CHECK;
787   5                      call CHANGE;
788   5                      MESSAGE$STATUS = TRUE;
789   5                      call DISP$CONTENTS;
790   5                      call MORE$COMMAND;
791   5                      if IN$YES then goto LOOP1;
793   5                  end;
794   4              else do;
795   5                  if (cmpb (@C$BUF(0), @('MC', ODH), 3) = TRUEW)
796   5                  then do;
797   6                      MESSAGE$STATUS = TRUE;
798   6                      MESSAGE$WORD = MC$SETUP$OFFSET;
799   6                      MESSAGE$BYTE = 14;
800   6                      call MC$SETUP$CMD;
801   6                      call COMMAND$LINK$CHECK;
802   6                      call CHANGE;
803   6                      MESSAGE$STATUS = TRUE;
804   6                      call DISP$CONTENTS;
805   6                      call MORE$COMMAND;
806   6                      if IN$YES then goto LOOP1;
808   6                  end;
```

296170-37

Table 6. Software Listings (Continued)

```
809   5              else do;
810   6                  if (cmpb (@C$BUF(0), @('TX',ODH), 3) = TRUEW)
811   6                  then do;
812   7                      MESSAGE$STATUS = TRUE;
813   7                      MESSAGE$WORD = TRANSMIT$OFFSET;
814   7                      MESSAGE$BYTE = 16;
815   7                      call TX$CMD;
816   7                      call EX$TX$CMD;
817   7                      call TRANSMIT$COMMAND$LINK;
818   7                      call MORE$COMMAND;
819   7                      if IN$YES then goto LOOP1;
821   7                  end;

822   6              else do;
823   7                  if (cmpb (@C$BUF(0), @('TDR',ODH), 4) = TRUEW)
824   7                  then do;
825   8                      MESSAGE$STATUS = TRUE;
826   8                      MESSAGE$WORD = TDR$OFFSET;
827   8                      MESSAGE$BYTE = 8;
828   8                      call TDR$CMD;
829   8                      call COMMAND$LINK$CHECK;
830   8                      call CHANGE;
831   8                      MESSAGE$STATUS = TRUE;
832   8                      call DISP$CONTENTS;
833   8                      call MORE$COMMAND;
834   8                      if IN$YES then goto LOOP1;
836   8                  end;

837   7              else do;
838   8                  if (cmpb (@C$BUF(0), @('DS',ODH), 3) = TRUEW)
839   8                  then do;
840   9                      MESSAGE$STATUS = TRUE;
841   9                      MESSAGE$WORD = DUMP$STAT$OFFSET;
842   9                      MESSAGE$BYTE = 8;
843   9                      call DUMP$STAT$CMD;
844   9                      call COMMAND$LINK$CHECK;
845   9                      call CHANGE;
846   9                      MESSAGE$STATUS = TRUE;
847   9                      call DISP$CONTENTS;
848   9                      call MORE$COMMAND;
849   9                      if IN$YES then goto LOOP1;
851   9                  end;

852   8              else do;
853   9                  if (cmpb (@C$BUF(0), @('DIAG',ODH), 5) = TRUEW)
854   9                  then do;
855  10                      MESSAGE$STATUS = TRUE;
856  10                      MESSAGE$WORD = DIAG$OFFSET;
857  10                      MESSAGE$BYTE = 6;
858  10                      call DIAG$CMD;
859  10                      call COMMAND$LINK$CHECK;
860  10                      call CHANGE;
861  10                      MESSAGE$STATUS = TRUE;
862  10                      call DISP$CONTENTS;
863  10                      call MORE$COMMAND;
864  10                      if IN$YES then goto LOOP1;
866  10                  end;

867   9              else do;
868  10                  if (cmpb(@C$BUF(0),@('H',ODH), 2) = TRUEW)
869  10                      then do;
870  11                          call COMMAND$MSG;
871  11                          call CR$LF;
872  11                          goto LOOP1;
873  11                      end;
874  10              else do;
875  11                  if (cmpb(@C$BUF(0),@('EXIT',ODH),5) = TRUEW)
876  11                  then call EXIT;
877  11                  else    call MSG$ILL$CMD;
878  11                                                  end;
879  10                                              end;
880   9                                          end;
881   8                                      end;
882   7                                  end;
883   6                              end;
884   5                          end;
885   4                      end;
886   3                  end;
887   2          end COMMAND$SET;                                        296170-38
```

Table 6. Software Listings (Continued)

```
                /* SCP, ISCP contents display */

  888   1           CP$DISPLAY: procedure;
  889   2               MSG$PTR = @('   *** SCP ***', ODH, OAH, 0);
  890   2               call OUTS;
                  /* SCP */
  891   2               MESSAGE$WORD = SCP.SYSBUS;
  892   2               call WORD$DISP;
  893   2               MSG$PTR = @('   SCP SYSBUS', ODH, OAH, 0);
  894   2               call OUTS;
  895   2               MESSAGE$WORD = SCP.ISCP$OFFSET;
  896   2               call WORD$DISP;
  897   2               MSG$PTR = @('   ISCP OFFSET', ODH, OAH, 0);
  898   2               call OUTS;
  899   2               MESSAGE$WORD = SCP.ISCP$BASE;
  900   2               call WORD$DISP;
  901   2               MSG$PTR = @('   ISCP BASE', ODH, OAH, 0);
  902   2               call OUTS;
                  /* ISCP */
  903   2               MSG$PTR = @('   *** ISCP ***', ODH, OAH, 0);
  904   2               call OUTS;
  905   2               MESSAGE$WORD = ISCP.BUSY;
  906   2               call WORD$DISP;
  907   2               MSG$PTR = @('   ISCP BUSY', ODH, OAH, 0);
  908   2               call OUTS;
  909   2               MESSAGE$WORD = ISCP.SCB$OFFSET;
  910   2               call WORD$DISP;
  911   2               MSG$PTR = @('   SCB OFFSET', ODH, OAH, 0);
  912   2               call OUTS;
  913   2               MESSAGE$WORD = ISCP.SCB$BASE1;
  914   2               call WORD$DISP;
  915   2               MSG$PTR = @('   SCB BASE L', ODH, OAH, 0);
  916   2               call OUTS;
  917   2               MESSAGE$WORD = ISCP.SCB$BASE2;
  918   2               call WORD$DISP;
  919   2               MSG$PTR = @('   SCB BASE H', ODH, OAH, 0);
  920   2               call OUTS;

  921   2           end CP$DISPLAY;


                /* All of SCB information display

                                Name:       SCB$STATUS$DISPLAY
                                Input:      SCB
                                Output:     MSG$BUF (console out)
                                Function:   Pass contents of SCB information to
                                            SCB contents display routine              */

  922   1           SCB$STATUS$DISPLAY: procedure;

  923   2               MSG$PTR = @('   *** SCB ***', ODH, OAH, 0);
  924   2               call OUTS;
  925   2               MESSAGE$WORD = SCB.STAT;
  926   2               call SCB$BITMAP$DISPLAY;
  927   2               MSG$PTR = @('B  STATUS', ODH, OAH, 0);
  928   2               call OUTS;
  929   2               MESSAGE$WORD = SCB.CMD;
  930   2               call SCB$BITMAP$DISPLAY;
  931   2               MSG$PTR = @('B  ACK', ODH, OAH, 0);
  932   2               call OUTS;
  933   2               MESSAGE$WORD = SCB.STAT;
  934   2               MESSAGE$STATUS = 0;
  935   2               call SCB$UNIT$DISPLAY;
  936   2               MSG$PTR = @('H    CUS', ODH, OAH, 0);
  937   2               call OUTS;
  938   2               MESSAGE$WORD = SCB.STAT;
  939   2               MESSAGE$STATUS = 1;
  940   2               call SCB$UNIT$DISPLAY;
  941   2               MSG$PTR = @('H    RUS', ODH, OAH, 0);
  942   2               call OUTS;
  943   2               MESSAGE$WORD = SCB.CMD;
  944   2               MESSAGE$STATUS = 0;
  945   2               call SCB$UNIT$DISPLAY;
  946   2               MSG$PTR = @('H    CUC', ODH, OAH, 0);
  947   2               call OUTS;
  948   2               MESSAGE$WORD = SCB.CMD;
  949   2               MESSAGE$STATUS = 1;
  950   2               call SCB$UNIT$DISPLAY;
  951   2               MSG$PTR = @('H    RUC', ODH, OAH, 0);
  952   2               call OUTS;
  953   2               MESSAGE$WORD = SCB.CBL$OFFSET;
  954   2               call WORD$DISP;
  955   2               MSG$PTR = @('H  CBL OFFSET', ODH, OAH, 0);
  956   2               call OUTS;
```

296170-39

Table 6. Software Listings (Continued)

```
  957   2             MESSAGE$WORD = SCB.RFD$OFFSET;
  958   2             call WORD$DISP;
  959   2             MSG$PTR = @('H  RFD OFFSET', ODH, OAH, 0);
  960   2             call OUTS;
  961   2             MESSAGE$WORD = SCB.CRC$ERRS;
  962   2             call WORD$DISP;
  963   2             MSG$PTR = @('H  CRC ERRORS', ODH, OAH, 0);
  964   2             call OUTS;
  965   2             MESSAGE$WORD = SCB.ALN$ERRS;
  966   2             call WORD$DISP;
  967   2             MSG$PTR = @('H  ALIGNMENT ERRORS', ODH, OAH, 0);
  968   2             call OUTS;
  969   2             MESSAGE$WORD = SCB.RSC$ERRS;
  970   2             call WORD$DISP;
  971   2             MSG$PTR = @('H  NO RESORCE ERRORS', ODH, OAH, 0);
  972   2             call OUTS;
  973   2             MESSAGE$WORD = SCB.OVRN$ERRS;
  974   2             call WORD$DISP;
  975   2             MSG$PTR = @('H  OVERRUN ERRORS', 0);
  976   2             call OUTS;
  977   2         end SCB$STATUS$DISPLAY;


            /* Action commands status display

                                Name:       ACTION$CMD$STATUS$DISPLAY
                                Input:      Action commands status word
                                Output:     MESSAGE$WORD
                                Function:   Pass the action command status display
                                            parameter to WORD$DISPLAY routine.
                                            This command performs all the action command
                                            status display at once.                    */

  978   1         ACTION$CMD$STATUS$DISPLAY: procedure;

  979   2             declare     I   byte;

  980   2             call CR$LF;
  981   2             MSG$PTR = @('NOP ---------- ', 0);
  982   2             call OUTS;
  983   2             MESSAGE$WORD = NOP.STAT;
  984   2             call WORD$DISP;
  985   2             call CO('H');
  986   2             call CR$LF;
  987   2             MSG$PTR = @('IA SETUP ------- ', 0);
  988   2             call OUTS;
  989   2             MESSAGE$WORD = IA$SETUP.STAT;
  990   2             call WORD$DISP;
  991   2             call CO('H');
  992   2             call CR$LF;
  993   2             MSG$PTR = @('CONFIGURATION -- ', 0);
  994   2             call OUTS;
  995   2             MESSAGE$WORD = CONF.STAT;
  996   2             call WORD$DISP;
  997   2             call CO('H');
  998   2             call CR$LF;
  999   2             MSG$PTR = @('MC SETUP ------- ', 0);
 1000   2             call OUTS;
 1001   2             MESSAGE$WORD = MC$SETUP.STAT;
 1002   2             call WORD$DISP;
 1003   2             call CO('H');
 1004   2             call CR$LF;
 1005   2             MSG$PTR = @('TRANSMIT ------- ', 0);
 1006   2             call OUTS;
 1007   2             MESSAGE$WORD = TRANSMIT.STAT;
 1008   2             call WORD$DISP;
 1009   2             call CO('H');
 1010   2             call CR$LF;
 1011   2             MSG$PTR = @('TDR ---------- ', 0);
 1012   2             call OUTS;
 1013   2             MESSAGE$WORD = TDR.STAT;
 1014   2             call WORD$DISP;
 1015   2             call CO('H');
 1016   2             call CR$LF;
 1017   2             MSG$PTR = @('DUMP STATUS ---- ', 0);
 1018   2             call OUTS;
 1019   2             MESSAGE$WORD = DUMP$STAT.STAT;
 1020   2             call WORD$DISP;
 1021   2             call CO('H');
 1022   2             call CR$LF;
 1023   2             MSG$PTR = @('DIAGNOSE ------- ', 0);
 1024   2             call OUTS;
 1025   2             MESSAGE$WORD = DIAG.STAT;
 1026   2             call WORD$DISP;
 1027   2             call CO('H');
 1028   2             call CR$LF;
```

296170-40

Table 6. Software Listings (Continued)

```
1029  2             MSG$PTR = @('EXTRA TRANSMiT STATUS', ODH, OAH, ODH, OAH, 0);
1030  2             call OUTS;
1031  2             do I = 0 to 9;
1032  3                 MESSAGE$WORD = EX$TRANSMIT(I).STAT;
1033  3                 call WORD$DISP;
1034  3                 call CO('H');
1035  3                 call CO(SP);
1036  3             end;
1037  2             call CR$LF;
1038  2         end ACTION$CMD$STATUS$DISPLAY;


              /* Time Domain Refrect-meter result display

                                 Name:        TDR$RESULT
                                 Input:       TDR.TIME
                                 Output:      MESSAGE$WORD
                                 Function:    Pass the TDR command execution result to
                                              WORD$DISP routine.
                                              This routine performs TDR result display */
1039  1          TDR$RESULT: procedure;

1040  2             MESSAGE$WORD = TDR.TIME;
1041  2             MSG$PTR = @(ODH, OAH, 'TDR RESULT = ', 0);
1042  2             call OUTS;
1043  2             call WORD$DISP;
1044  2             call CR$LF;
1045  2         end TDR$RESULT;


              /* Buffer status display

                                 Name:        BD$STATUS$DISPLAY
                                 Input:       All of buffer descriptors status and
                                              actual count field.
                                 Output:      MESSAGE$WORD
                                 Function:    Pass the parameters of all of buffer descriptors
                                              status and actual count to WORD$DISP routine. */


1046  1          BD$STATUS$DISPLAY: procedure;


1047  2             declare I                 byte;

1048  2             MSG$PTR = @(ODH, OAH, '** RFD STATUS', ODH, OAH, 0);
1049  2             call OUTS;
1050  2             do I = 0 to 15;
1051  3                 MESSAGE$WORD = RFD(I).STAT;
1052  3                 call WORD$DISP;
1053  3                 call CO(SP);
1054  3             end;

1055  2             MSG$PTR = @('** RFD EL AND S', ODH, OAH, 0);
1056  2             call OUTS;
1057  2             do I = 0 to 15;
1058  3                 MESSAGE$WORD = RFD(I).EL$S;
1059  3                 call WORD$DISP;
1060  3                 call CO(SP);
1061  3             end;

1062  2             MSG$PTR = @('** RFD LINK ADDRESS', ODH, OAH, 0);
1063  2             call OUTS;
1064  2             do I = 0 to 15;
1065  3                 MESSAGE$WORD = RFD(I).LINK$ADDRESS;
1066  3                 call WORD$DISP;
1067  3                 call CO(SP);
1068  3             end;

1069  2             MSG$PTR = @('** RFD BUFFER DESCRIPTOR POINTER', ODH, OAH, 0);
1070  2             call OUTS;
1071  2             do I = 0 to 15;
1072  3                 MESSAGE$WORD = RFD(I).BD$PTR;
1073  3                 call WORD$DISP;
1074  3                 call CO(SP);
1075  3             end;

1076  2             MSG$PTR = @('** RFD DESTINATION ADDRESS', ODH, OAH, 0);
1077  2             call OUTS;
1078  2             do I = 0 to 15;
1079  3                 MESSAGE$WORD = (shl(double (RFD(I).DEST$ADDRESS(1)),8) +
                                      double(RFD(I).DEST$ADDRESS(0)));
1080  3                 call WORD$DISP;
1081  3                 call CO(SP);
1082  3             end;
```

296170-41

Table 6. Software Listings (Continued)

```
1083  2          do I = 0 to 15;
1084  3              MESSAGE$WORD = (shl(double (RFD(I).DEST$ADDRESS(3)),8) +
                                     double(RFD(I).DEST$ADDRESS(2)));
1085  3              call WORD$DISP;
1036  3              call CO(SP);
1087  3          end;
1088  2          do I = 0 TO 15;
1089  3              MESSAGE$WORD = (shl(double (RFD(I).DEST$ADDRESS(5)),8) +
                                     double(RFD(I).DEST$ADDRESS(4)));
1090  3              call WORD$DISP;
1091  3              call CO(SP);
1092  3          end;

1093  2          MSG$PTR = @('** RFD SOURCE ADDRESS', 0DH, 0AH, 0);
1094  2          call OUTS;
1095  2          do I = 0 to 15;
1096  3              MESSAGE$WORD = (shl(double (RFD(I).SOURCE$ADDRESS(1)),8) +
                                     double(RFD(I).SOURCE$ADDRESS(0)));
1097  3              call WORD$DISP;
1098  3              call CO(SP);
1099  3          end;
1100  2          do I = 0 to 15;
1101  3              MESSAGE$WORD = (shl(double (RFD(I).SOURCE$ADDRESS(3)),8) +
                                     double(RFD(I).SOURCE$ADDRESS(2)));
1102  3              call WORD$DISP;
1103  3              call CO(SP);
1104  3          end;
1105  2          do I = 0 to 15;
1106  3              MESSAGE$WORD = (shl(double (RFD(I).SOURCE$ADDRESS(5)),8) +
                                     double(RFD(I).SOURCE$ADDRESS(4)));
1107  3              call WORD$DISP;
1108  3              call CO(SP);
1109  3          end;

1110  2          MSG$PTR = @('** RFD TYPE FIELD', 0DH, 0AH, 0);
1111  2          call OUTS;
1112  2          do I = 0 to 15;
1113  3              MESSAGE$WORD = RFD(I).TYPE$FIELD;
1114  3              call WORD$DISP;
1115  3              call CO(SP);
1116  3          end;

1117  2          MSG$PTR = @(0DH, 0AH, ' TYPE <CR> TO CONTINUE', 0);
1118  2          call OUTS;
1119  2          call INS;
1120  2          MSG$PTR = @(0DH, 0AH, '** RBD ACTUAL COUNT', 0DH, 0AH, 0);
1121  2          call OUTS;
1122  2          do I = 0 to 15;
1123  3              MESSAGE$WORD = RBD(I).ACT$COUNT;
1124  3              call WORD$DISP;
1125  3              call CO(SP);
1126  3          end;

1127  2          MSG$PTR = @('** RBD NEXT BD ADDRESS', 0DH, 0AH, 0);
1128  2          call OUTS;
1129  2          do I = 0 to 15;
1130  3              MESSAGE$WORD = RBD(I).NEXT$BD$ADD;
1131  3              call WORD$DISP;
1132  3              call CO(SP);
1133  3          end;

1134  2          MSG$PTR = @('** RBD BUFFER OFFSET, BASE', 0DH, 0AH, 0);
1135  2          call OUTS;
1136  2          do I = 0 to 15;
1137  3              MESSAGE$WORD = RBD(I).BUFF$OFFSET;
1138  3              call WORD$DISP;
1139  3              call CO(SP);
1140  3          end;

1141  2          do I = 0 to 15;
1142  3              MESSAGE$WORD = RBD(I).BUFF$BASE;
1143  3              call WORD$DISP;
1144  3              call CO(SP);
1145  3          end;

1146  2          MSG$PTR = @('** RBD BUFFER SIZE', 0DH, 0AH, 0);
1147  2          call OUTS;
1148  2          do I = 0 to 15;
1149  3              MESSAGE$WORD = RBD(I).SIZE;
1150  3              call WORD$DISP;
1151  3              call CO(SP);
1152  3          end;

1153  2          MSG$PTR = @('** TBD ACTUAL COUNT', 0DH, 0AH, 0);
1154  2          call OUTS;
1155  2          do I = 0 to 15;                                          296170-42
```

Table 6. Software Listings (Continued)

```
1156   3                MESSAGE$WORD = TBD(I).ACT$COUNT;
1157   3                call WORD$DISP;
1158   3                call CO(SP);
1159   3             end;

1160   2             MSG$PTR = @('** TBD NEXT BD ADDRESS', ODH, OAH, 0);
1161   2             call OUTS;
1162   2             do I = 0 to 15;
1163   3                MESSAGE$WORD = TBD(I).NEXT$BD$ADD;
1164   3                call WORD$DISP;
1165   3                call CO(SP);
1166   3             end;

1167   2             MSG$PTR = @('** TBD BUFFER OFFSET, BASE', ODH, OAH, 0);
1168   2             call OUTS;
1169   2             do I = 0 to 15;
1170   3                MESSAGE$WORD = TBD(I).BUFF$OFFSET;
1171   3                call WORD$DISP;
1172   3                call CO(SP);
1173   3             end;
1174   2             do I = 0 to 15;
1175   3                MESSAGE$WORD = TBD(I).BUFF$BASE;
1176   3                call WORD$DISP;
1177   3                call CO(SP);
1178   3             end;

1179   2          end BD$STATUS$DISPLAY;


1180   1          IN$STAT: procedure byte;
1181   2             declare CHAR            byte;
1182   2             do while ((CHAR <> 'A' or CHAR <> 'a') or
                            (CHAR <> 'B' or CHAR <> 'b') or
                            (CHAR <> 'S' or CHAR <> 's'));
1183   3             CHAR = CI and 7FH;
1184   3             call CO(CHAR);
1185   3             if CHAR = 'A' or CHAR = 'a'
1186   3                then return 'A';
1187   3             else if CHAR = 'B' or CHAR = 'b'
1188   3                then return 'B';
1189   3             else if CHAR = 'S' or CHAR = 's'
1190   3                then return 'S';
1191   3             else do;
1192   4                MSG$PTR = @(ODH, OAH, 'STAT> KEY IN S, A, or B', ODH, OAH, 2AH, 0);
1193   4                call OUTS;
1194   4                end;
1195   3             end;
1196   2          end IN$STAT;


             /* Status display

                                Name:       STATUS$DISPLAY
                                Input:      C$BUF (console in)
                                Output:     None
                                Function:   Interpret the key input command what status
                                            should be displayed                          */


1197   1          STATUS$DISPLAY: procedure;
1198   2             declare I          byte;


1199   2             MSG$PTR = @(ODH, OAH, 'STAT> KEY IN COMMAND THAT YOU WANT TO SEE IN THE STATUS',
                         ODH, OAH, 0);
1200   2             call OUTS;
1201   2             MSG$PTR = @('S FOR SCB, A FOR ACTION COMMANDS, B FOR BUFFER DESCRIPTORS',
                         ODH, OAH, 2AH, 0);
1202   2             call OUTS;
1203   2             MESSAGE$BYTE = IN$STAT;
1204   2             if MESSAGE$BYTE = 'A' then call ACTION$CMD$STATUS$DISPLAY;
1206   2             else if MESSAGE$BYTE = 'B' then call BD$STATUS$DISPLAY;
1208   2             else if MESSAGE$BYTE = 'S' then do;
1210   3                                             call CP$DISPLAY;
1211   3                                             call SCB$STATUS$DISPLAY;
1212   3             end;

1213   2          end STATUS$DISPLAY;

             /* Initialize individual block

                                Name:       INIT$COMMAND
                                Input:      C$BUF (console in)
                                Output:     None
```

296170-43

Table 6. Software Listings (Continued)

```
                                Function:       Interpret the key in command to determine
                                                what should be initialized individually.
                                                This routine is able to initialize
                                                the command block, buffer descriptors and
                                                buffers.                                  */

1214   1        INIT$COMMAND: procedure;

1215   2        LOOP3:  MSG$PTR = @(ODH,OAH,'INIT> KEY IN YOUR COMMAND (H FOR HELP)',
                        ODH,OAH,2AH,0);
1216   2            call OUTS;
1217   2            call INS;
1218   2            if (cmpb(@C$BUF(0), @('NOP',ODH), 4) = TRUEW) then call NOP$CMD;
1220   2            else if (cmpb(@C$BUF(0),@('IA',ODH), 3) = TRUEW) then call IA$SETUP$CMD;
1222   2            else if (cmpb(@C$BUF(0),@('CONF',ODH), 5) = TRUEW) then call CONF$CMD;
1224   2            else if (cmpb(@C$BUF(0),@('MC',ODH), 3) = TRUEW) then call MC$SETUP$CMD;
1226   2            else if (cmpb(@C$BUF(0),@('TX',ODH), 3) = TRUEW) then call TX$CMD;
1228   2            else if (cmpb(@C$BUF(0),@('TDR',ODH), 4) = TRUEW) then call TDR$CMD;
1230   2            else if (cmpb(@C$BUF(0),@('DS',ODH), 3) = TRUEW) then   call DUMP$STAT$CMD;
1232   2            else if (cmpb(@C$BUF(0),@('DIAG',ODH), 5) = TRUEW) then call DIAG$CMD;
1234   2            else if (cmpb(@C$BUF(0),@('RFD',ODH), 4) = TRUEW) then call INIT$RFD;
1236   2            else if (cmpb(@C$BUF(0),@('RBD',ODH), 4) = TRUEW) then call INIT$RBD;
1238   2            else if (cmpb(@C$BUF(0),@('RB',ODH), 3) = TRUEW)  then call INIT$RB;
1240   2            else if (cmpb(@C$BUF(0),@('TBD',ODH), 4) = TRUEW) then call INIT$TBD;
1242   2            else if (cmpb(@C$BUF(0),@('H',ODH), 1) = TRUEW) then
1243   3                do; call COMMAND$MSG;
1245   3                    call BD$MSG;
1246   3                    call CR$LF;
1247   3                    goto LOOP3;
1248   3                end;
1249   2            else    call MSG$ILL$CMD;

1250   2        end INIT$COMMAND;


1251   1        IN$CR: procedure byte;

1252   2            declare CHAR                    byte;
1253   2            CHAR = 0;
1254   2            do while ((CHAR <> 'C') or (CHAR <> 'c') or
                            (CHAR <> 'R') or (CHAR <> 'r') or
                            (CHAR <> 'B') or (CHAR <> 'b'));
1255   3                CHAR = CI and 7FH;
1256   3                call CO(CHAR);
1257   3                if CHAR = 'C' or CHAR = 'C'
1258   3                    then return 'C';
1259   3                else if CHAR = 'R' or CHAR = 'r'
1260   3                    then return 'R';
1261   3                else if CHAR = 'B' or CHAR = 'b'
1262   3                    then return 'B';
1263   3                else do;
1264   4                    MSG$PTR = @(ODH,OAH,'** KEY IN C , R or B',ODH,OAH,2AH,0);
1265   4                    call OUTS;
1266   4                end;
1267   3            end;
1268   2        end IN$CR;

            /* Unit control

                                Name:       UNIT$CONTROL
                                Input:      C$BUF (Console)
                                Output:     None
                                Function:   Controls the CU or RU execution.
                                            SCB information will be displayed.
                                            This command is used to control CU or RU    */

1269   1        UNIT$CONTROL: procedure(CONTROL);
1270   2            declare     CONTROL     byte;

1271   2            MSG$PTR = @(ODH,OAH,'KEY IN  C  FOR COMMAND,  R  FOR RECEIVE',
                        ODH,OAH,2AH,0);
1272   2            call OUTS;
1273   2            MESSAGE$BYTE = IN$CR;
1274   2            if MESSAGE$BYTE = 'C' then
1275   2            do case CONTROL;
1276   3                SCB.CMD = CMD$START;
1277   3                SCB.CMD = CMD$ABORT;
1278   3                SCB.CMD = CMD$SUSPEND;
1279   3                SCB.CMD = CMD$RESUME;
1280   3            end;
1281   2            else if MESSAGE$BYTE = 'R' then
1282   2            do case CONTROL;
1283   3                SCB.CMD = RCV$START;
1284   3                SCB.CMD = RCV$ABORT;
1285   3                SCB.CMD = RCV$SUSPEND;
```

296170-44

Table 6. Software Listings (Continued)

```
1286   3                SCB.CMD = RCV$RESUME;
1287   3            end;
1288   2            if MESSAGE$BYTE = 'B' then
1289   2            do case CONTROL;
1290   3                SCB.CMD = CMD$START or RCV$START;
1291   3                SCB.CMD = CMD$ABORT or RCV$ABORT;
1292   3                SCB.CMD = CMD$SUSPEND or RCV$SUSPEND;
1293   3                SCB.CMD = CMD$RESUME or RCV$RESUME;
1294   3            end;
1295   2                call CR$LF;
1296   2                SCB.CBL$OFFSET = CMD$TOP$OFFSET;
1297   2                CMD$TOP$STATUS = FALSE;
1298   2                call SCB$STATUS$DISPLAY;
1299   2                call CA;
1300   2        end UNIT$CONTROL;



            /* Master help display

                                    Name:       MASTER$HELP
                                    Input:      None
                                    Output:     MSG$BUF (Console out)
                                    Function:   This routine displays help for
                                                 master command interpret routine. */

1301   1        MASTER$HELP: procedure;
1302   2            call CR$LF;
1303   2            MSG$PTR = @('START THE UNIT(RU,CU) ---------- START
                                 ABORT THE UNIT(RU,CU) ---------- ABORT',ODH,OAH,
                                 'SUSPEND THE UNIT(RU,CU) ------ SUSPEND
                                 RESUME THE UNIT(RU,CU) -------- RESUME',ODH,OAH,0);
1304   2            call OUTS;
1305   2            MSG$PTR = @('COMMAND BLOCK SET UP ------------ CSET
                                 STATUS DISPLAY ------------------ STAT',ODH,OAH,
                                 'INITIALIZE ---------------------- INIT
                                 RECEIVE BUFFER DISPLAY ---------- RXB',ODH,OAH,0);
1306   2            call OUTS;
1307   2            MSG$PTR = @('TRANSMIT BUFFER DATA CHANGE ------ TXB',ODH,OAH,
                                 'EXIT FROM THIS MONITOR ---------- EXIT
                                 TDR RESULT ---------------------- TRES',ODH,OAH,
                                 'DS RESULT BUFFER DISPLAY ------- DSRES  ',0);
1308   2            call OUTS;
1309   2            MSG$PTR = @('CHIP RESET ---------------------- CHRES',ODH,OAH,
                                 'ACKNOWRIGE TO INTERRUPT ---------- ACK  ',0);
1310   2            call OUTS;
1311   2        end MASTER$HELP;

            /* Master command interpreter

                                    Name:       MASTER$CMD
                                    Input:      C$BUF (console in)
                                    Output:     None
                                    Function:   Interpret the First Level commands. */

1312   1        MASTER$CMD: procedure;

1313   2        LOOP4:  MSG$PTR = @(ODH,OAH,'MAIN> KEY IN YOUR COMMAND (H FOR HELP)',  ODH,OAH,2AH,0);
1314   2            call OUTS;
1315   2            call INS;

1316   2            if (cmpb (@C$BUF(0),@('START',ODH), 6) = TRUEW)
1317   2                then call UNIT$CONTROL(0);
1318   2            else if (cmpb(@C$BUF(0),@('ABORT',ODH), 6) = TRUEW)
1319   2                then call UNIT$CONTROL(1);
1320   2            else if (cmpb(@C$BUF(0),@('SUSPEND',ODH), 8) = TRUEW)
1321   2                then call UNIT$CONTROL(2);
1322   2            else if (cmpb(@C$BUF(0),@('RESUME',ODH), 7) = TRUEW)
1323   2                then call UNIT$CONTROL(3);
1324   2            else if (cmpb(@C$BUF(0),@('CSET',ODH), 5) = TRUEW)
1325   2                then call COMMAND$SET;
1326   2            else if (cmpb(@C$BUF(0),@('STAT',ODH), 5 ) = TRUEW)
1327   2                then call STATUS$DISPLAY;
1328   2            else if (cmpb(@C$BUF(0),@('INIT',ODH), 5) = TRUEW)
1329   2                then call INIT$COMMAND;
1330   2            else if (cmpb(@C$BUF(0),@('RXB',ODH), 4) = TRUEW)
1331   2                then call BUFFER$DISP;
1332   2            else if (cmpb(@C$BUF(0),@('TXB',ODH), 4) = TRUEW)
1333   2                then call TRANSMIT$DATA$CHANGE;
1334   2            else if (cmpb(@C$BUF(0),@('H',ODH), 2) = TRUEW)
1335   2                then do;
1336   3                    call MASTER$HELP;
1337   3                    goto LOOP4;
```

296170-45

Table 6. Software Listings (Continued)

```
1338   3                end;
1339   2             else if (cmpb(@C$BUF(0),@('EXIT',ODH), 5) = TRUEW)
1340   2                then call EXIT;
1341   2             else if (cmpb(@C$BUF(0),@('TRES',ODH), 5) = TRUEW)
1342   2                then call TDR$RESULT;
1343   2             else if (cmpb(@C$BUF(0),@('DSRES',ODH), 5) = TRUEW)
1344   2                then call DS$BUFF$DISP;
1345   2             else if (cmpb(@C$BUF(0),@('CHRES',ODH), 6) = TRUEW)
1346   2                then call CHIP$RESET;
1347   2             else if (cmpb(@C$BUF(0),@('ACK',ODH), 4) = TRUEW)
1348   2                then call ACK;

1349   2             else call MSG$ILL$CMD;
1350   2          end MASTER$CMD;



               /* Main Routine */


1351   1          MAIN: do;

1352   2             call INIT$SYS;
1353   2             call INIT$RFD;
1354   2             call INIT$RBD;
1355   2             call INIT$RB;
1356   2             call INIT$TBD;
1357   2             call INIT$TB;
1358   2             CMD$TOP$STATUS = FALSE;
1359   2             NEXT$CMD$OFFSET = TRUEW;

1360   2             MSG$PTR = @('82586 DUAL PORT MEMORY TEST PROGRAM X206',ODH,OAH,0);
1361   2             call OUTS;
1362   2             do FOREVER;
1363   3                call MASTER$CMD;
1364   3             end;
1365   2          end main;
1366   1       end;



MODULE INFORMATION:

     CODE AREA SIZE     = 29C0H  10688D
     CONSTANT AREA SIZE = 0DF5H   3573D
     VARIABLE AREA SIZE = 0160H    352D
     MAXIMUM STACK SIZE = 0020H     32D
     2259 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

DICTIONARY SUMMARY:
     159KB MEMORY
```

296170-46

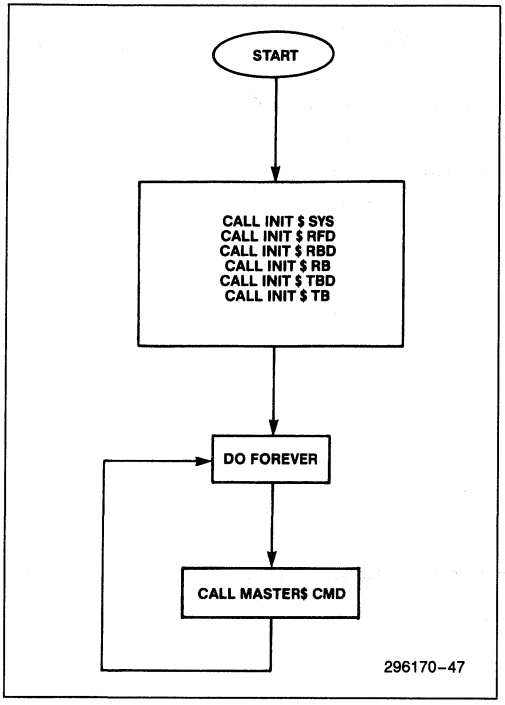


Figure 12A. Main Program Flow

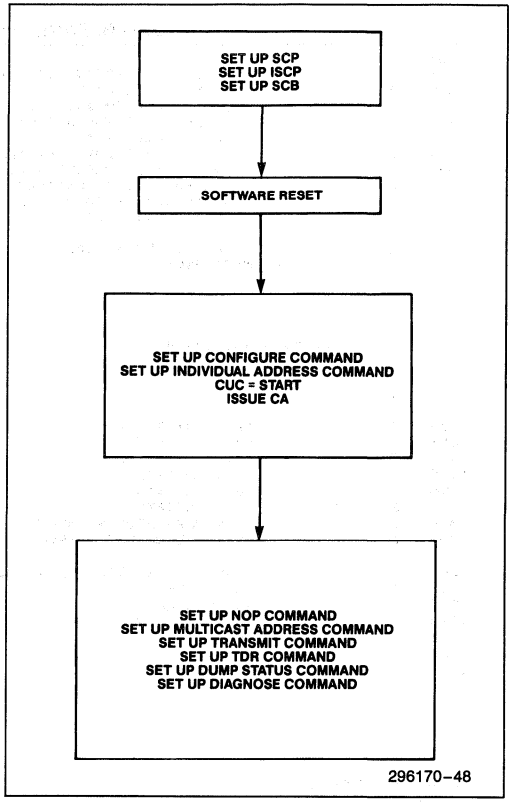


Figure 12B. System Initialization Routine

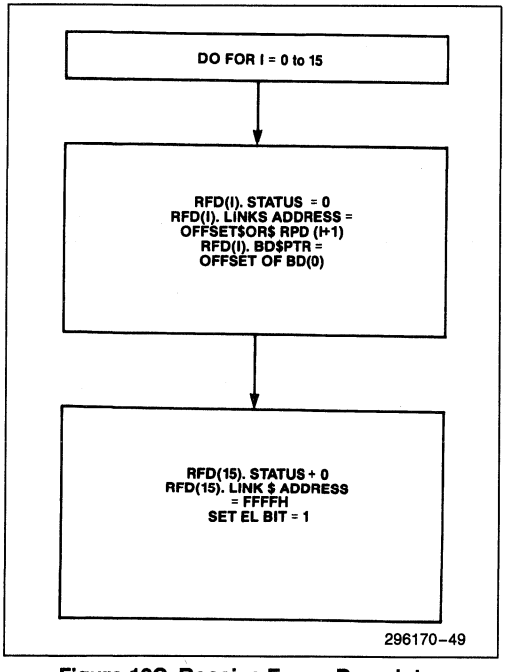


Figure 12C. Receive Frame Descriptor Initialization

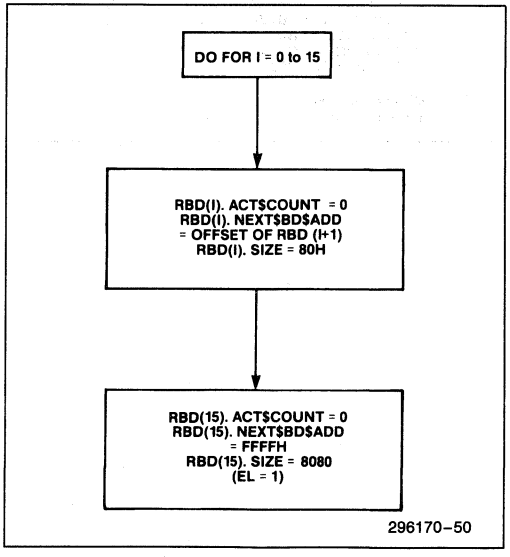


Figure 12D. Receive Buffer Descriptor Initialization

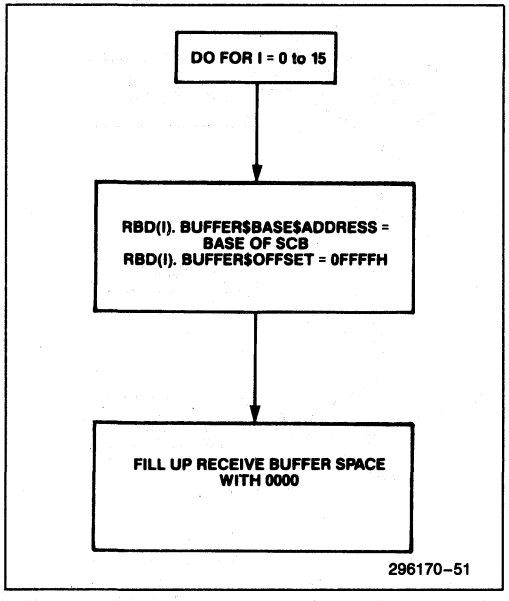


Figure 12E. Receive Buffer Space Initialization

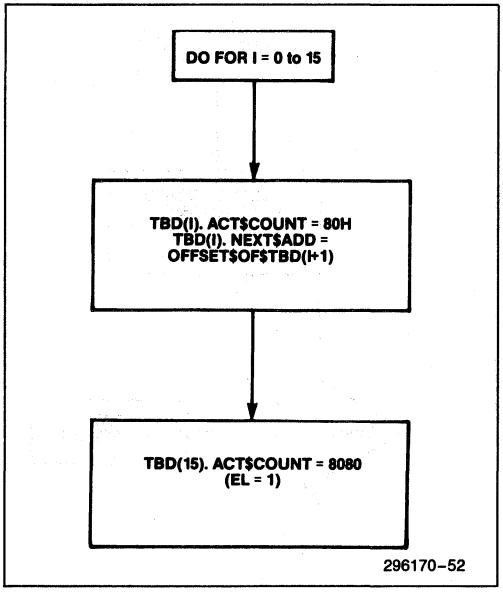


Figure 12F. Initialization of Transmit Buffer Descriptors

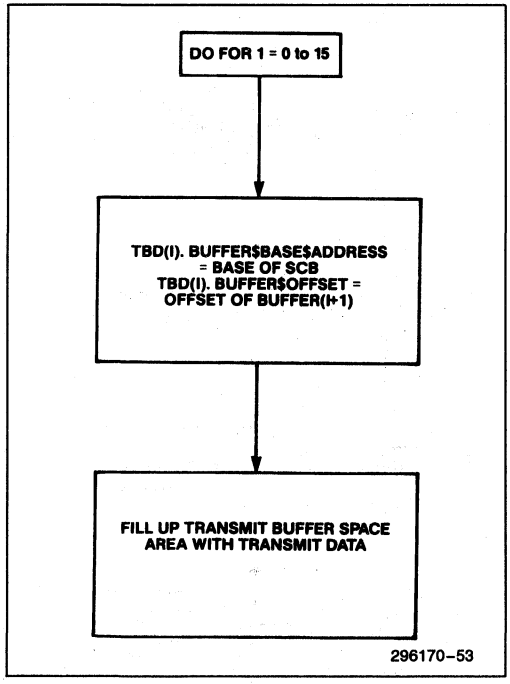


Figure 12G. Transmit Buffer Data Initialization

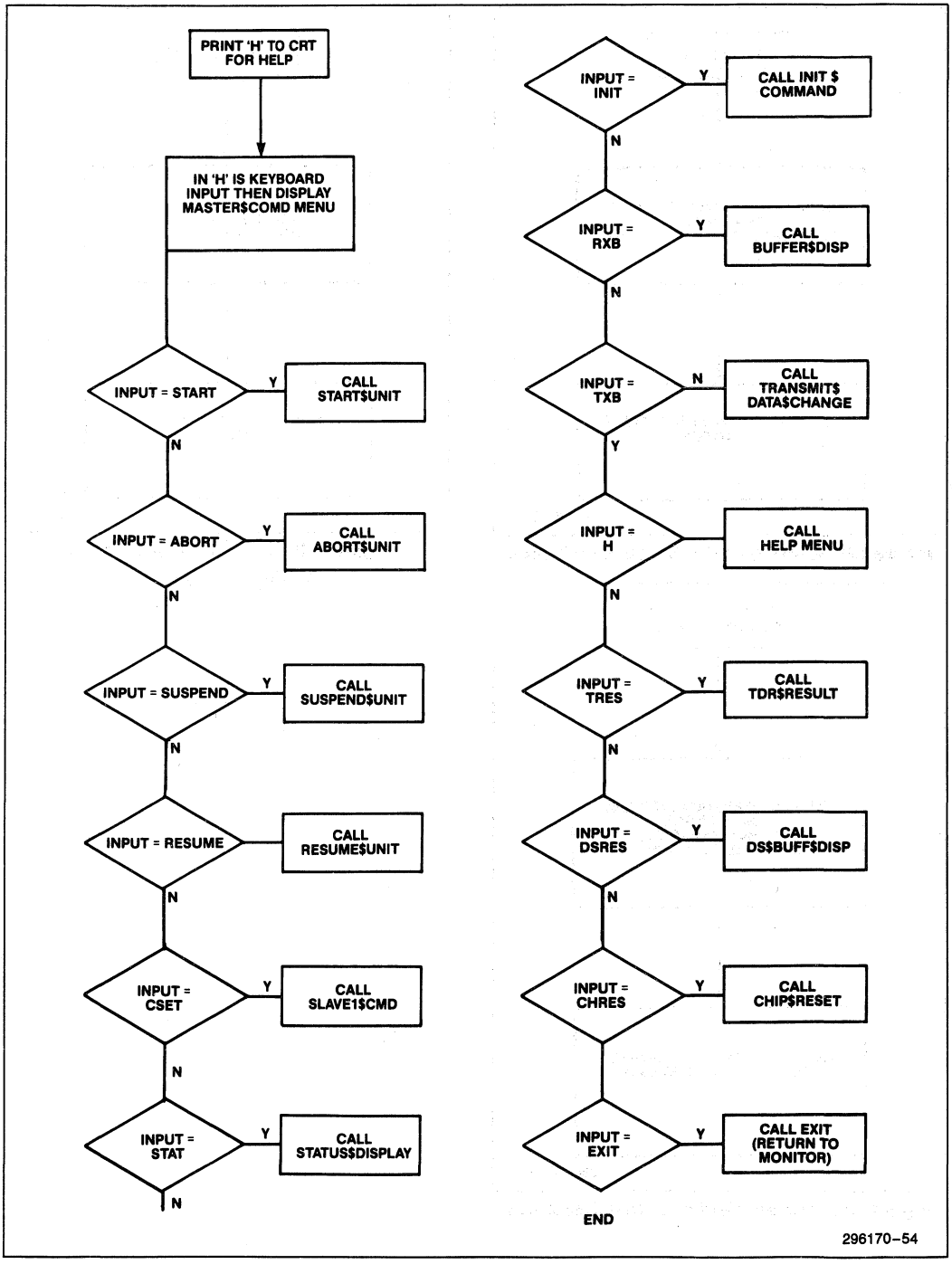


Figure 12H. Master Command Flow Diagram

```
'START'    : START COMMAND UNIT ON RECEIVE UNIT
'ABORT'    : ABORT CU OR RU
'SUSPEND'  : SUSPEND CU OR RU
'RESUME'   : RESUME CU OR RU
'CSET'     : COMMAND SET MENU
'STAT'     : STATUS MENU
'INIT'     : INITIALIZE COMMAND BLOCKS,
             : RECEIVE OR TRANSMIT BUFFERS
'RXB'      : DISPLAY RECEIVE BUFFERS
'TXB'      : DISPLAY TRANSMIT BUFFERS
'H'        : HELP
'EXIT'     : EXIT THE MONITOR
'TRES'     : DISPLAY TDR RESULT
'DSRES'    : DISPLAY DUMP STATUS BUFFER
'CHRES'    : SOFTWARE CHIP RESET
                                    296170-55
```

Figure 12I. Master Command Menu

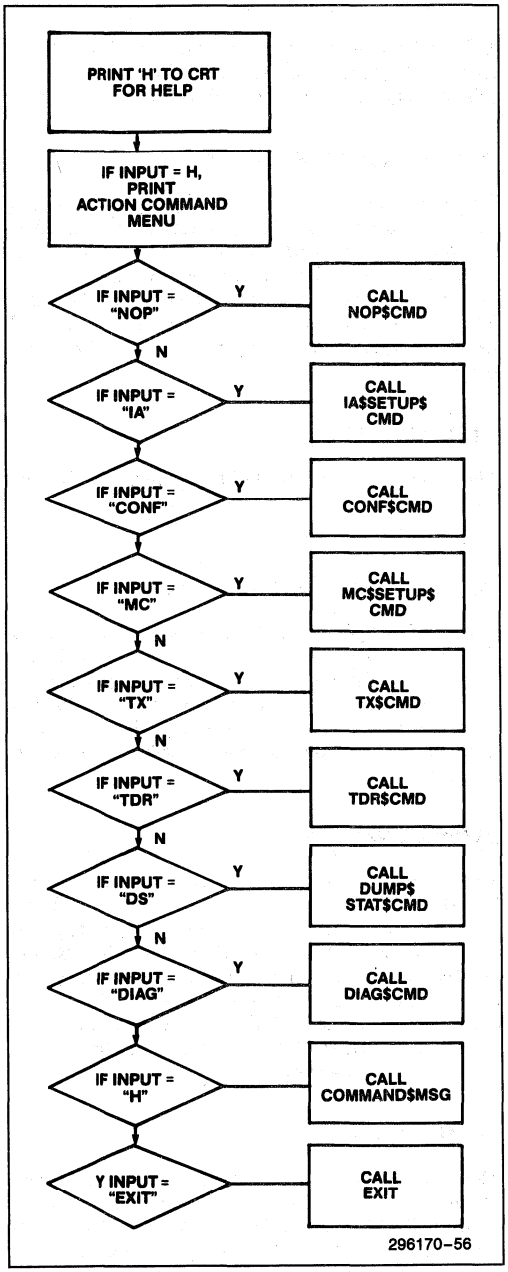


Figure 12J. Action Commands Flow Diagram

```
NOP   :  SET UP NOP COMMAND
IA    :  SET UP INDIVIDUAL ADDRESS COMMAND
CONF  :  SET UP CONFIGURE COMMAND
MC    :  SET UP MULTICAST COMMAND
TX    :  SET UP TRANSMIT COMMAND
TDR   :  SET UP TDR COMMAND
DS    :  SET UP DUMP STATUS COMMAND
DIAG  :  SET UP DIAGNOSE COMMAND
                                    296170-57
```

Figure 12K. Action Commands Menu

6.3 Special Considerations

This application example is based on a design using the Intel 80186. This section discusses the ease with which this memory design can be applied to other CPU's.

82586 CLOCK GENERATION

The 82586 clock requirement is 8 MHz (maximum) 50% duty cycle with MOS voltage levels. The 80186 generates this clock and it is used in this example. However, for non-80186 based designs, this clock may be obtained from the Intel clock generator chip, the 82285. This device produces a 50% duty cycle clock with MOS levels and is fully compatible with 82586 requirements.

8-BIT DATA BUS

The same design may easily be modified for 8-bit data buses. Figure 9 illustrates an approach for an 8088 based design.

DEMULTIPLEXED ADDRESS/DATA BIT

The 82586 design with CPU's having demultiplexed address and data buses can easily be implemented using the approach illustrated in this section. Figure 9 shows how the same design can be used for demultiplexed address and data buses with relatively minor changes to this design. The same approach is valid for both 8 or 16-bit data buses.

DYNAMIC RAM MEMORIES

To keep the design simple and avoid the extra cost of furnishing refresh for dynamic RAMs, static RAMs were selected for this design. However, the design approach is independent of whether DRAMSs, SRAMs, or iRAMs are used. System designs requiring large dual port memories may choose to use DRAMs. In such cases it is recommended that Intel's Dynamic RAM Controller, 8207, for DRAM refresh and bus arbitration be used. See section 4.3.

STATIC-RAM INTERFACE

The static RAM used in this design is the 2148. However, the 2114 or 2149 may also be used. The chip selects for these RAMs is selected via the A0 and BHE signals. These memories do not have a separate output enable inputs, making it necessary to use Read and Write signals for chip select generation, thus avoiding bus contention between chip select and Write active.

SERIAL INTERFACE

The 82586 Serial Interface is a typical 10 Mbps Ethernet interface. The 82501 uses a 20 MHz antiresonant crystal as its clock source, and provides the Transmit and Receive clocks for the 82586.

6.4 Conclusions

The concept of dual port memory design is not new. However, the cost of such an approach has always been an issue. This design example illustrates a simple low cost design which still retains all of the advantages of the dual port design. This design may be easily modified for 8-bit CPU architectures or for systems using CPU's with demultiplexed address and data buses. This approach is suitable for system designs where the CPU bus bandwidth requirements and bus latency may retard system performance, especially when receiving serial data at 10 Mbps. This approach, while preserving the CPU bus bandwidth for other tasks, enables the user to optimize the 82586 design for very efficient bus utilization. Another advantage of this approach is the relative ease with which the 82586 can be interfaced to CPU's that are considerably different than the iAPX 186, or iAPX 86 family.

The total number of devices used for bus arbitration in this design consisted of 10–12 SSI/MSI chips. The entire circuit used up about 36 square inches of board space and the estimated cost of the board, components and connectors was less than $75, less the cost of the 82586/82501.

7.0 iNA 960 TRANSPORT ENGINE

7.1 Introduction

There are two alternatives to provide a system with communications capability. One configuration is to run the communications software as a job under the host operating system. In this implementation, the communication software shares host processor resources with other user tasks, see Figure 13. The second configuration is to run the communications software on a dedicated processor. In this implementation, the communication software runs separately from the host and other user programs. This latter implementation is frequently called a Communications Front End Processor, or Comm-Engine. When the Comm-Engine is implementing the Transport Layer function, it is called a Transport Engine, see Figure 14.

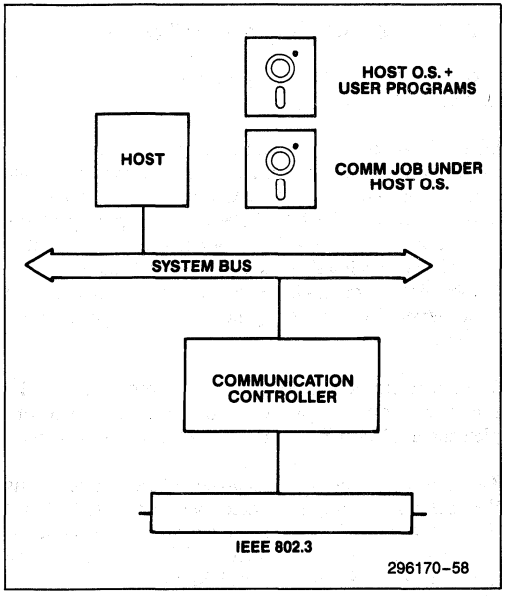


Figure 13. Communications Software Running on a Host

The Comm-Engine approach provides system designers with three important benefits: performance, flexibility and easy system upgrades. First, the Comm-Engine affords high performance because it offloads the host from supporting communication tasks. Thus, high communication throughput can be realized because the CPU is not supporting other user programs. Second, the Comm-Engine can be used to contribute to systems because it fits into a modular system architecture. Modular systems are attractive because they facilitate future system expansion and upgrades. Modularity also simplifies service and maintenance of the system. Third, the Comm-Engine will enable existing systems to be upgraded to embody networking capability, without degrading system performance.

The Intel 82586 LAN Coprocessor, a dedicated iAPX 86/88/186 processor and iNA 960 Network Software are the building blocks required to realize a Transport Engine. This Transport Engine provides the user with ISO standard compatible Transport capabilities in any hardware and software system environment.

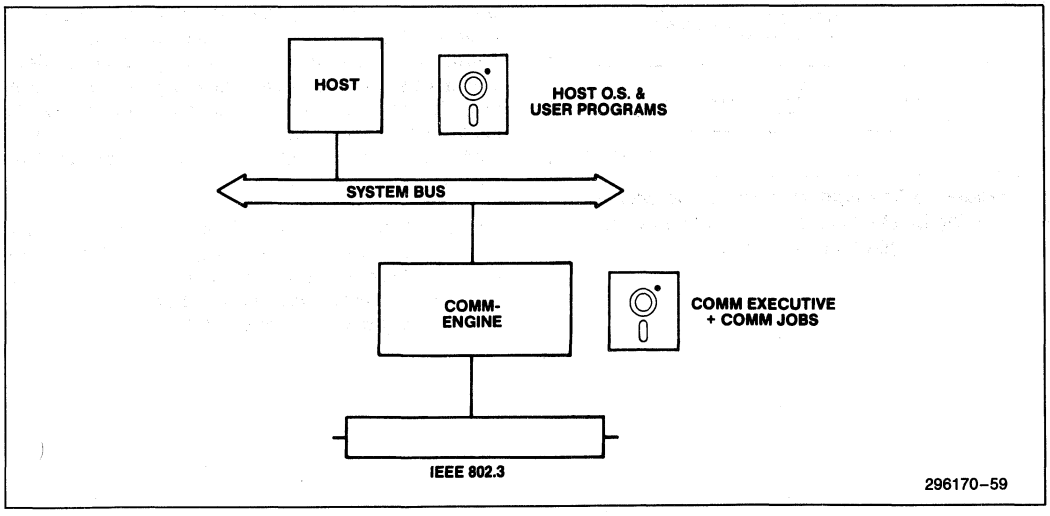


Figure 14. Communications Front End Processor, Comm-Engine

7.2 Transport Engine Hardware

Figure 15 shows the block diagram of a generic Transport Engine based on the 82586. The 80186 processor is used to eliminate the need for 'TTL glue,' and thus be the lower cost implementation.

In addition to the processor, the Transport Engine requires local RAM for data buffering and ROM to store communication software. A typical configuration requires 30 to 50K bytes of ROM for iNA 960, and 16 to 64K bytes of RAM for data buffering and supporting the software.

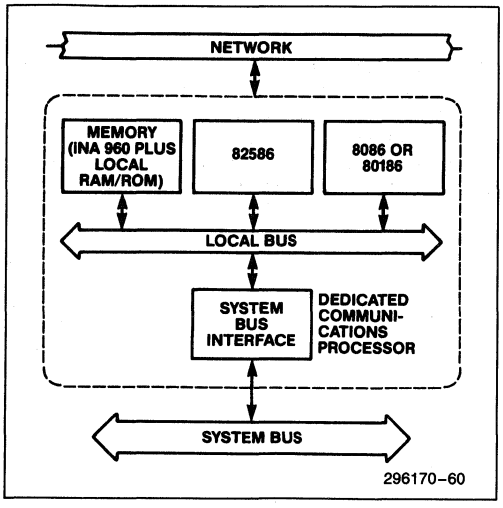


Figure 15. Transport Engine Implemented Using iNA 960 Running on a Dedicated 8086, 8088, or 80186

7.3 Transport Engine Software

Figure 16 shows the Transport Engine software interface to the host CPU.

iNA 960 consists of three parts: a communications subsystem, a hardware dependent module, and a message delivery mechanism. The communications subsystem implements the ISO 8073 Class 4 Transport protocol for virtual circuits and a network management facility. In addition, optional functionalities include a connectionless datagram service, external data link access, and boot server for supporting nodes without local mass storage.

The hardware dependent module is configured by the user to support the specific hardware environment implemented. iNA 960 provides 82586 software drivers.

The message delivery mechanism, MDM, implements the exchange of request blocks between iNA 960 and the host. Request blocks are memory segments containing command or status information and remain the same in all environments and implementations. iNA 960 provides the user with three choices of message delivery mechanisms. First, the Base Control Block interface mechanism to interface iNA 960 to single host environments. Second, the Multibus Interprocessor Protocol, MIP, for multiple processor environments. Third, iNA 960 supports a user supplied mechanism. The first two alternatives are fully implemented in iNA 960, while the third provides the 'hooks' for the user to implement his own version. Regardless of the option used, a compatible mechanism has to be provided by the user to run on the host side.

The iNA 960/82586 LAN Coprocessor based Transport Engine provides an industry standard high performance Transport solution for most hardware/software environments. This solution dramatically reduces the effort required to develop a LAN capability.

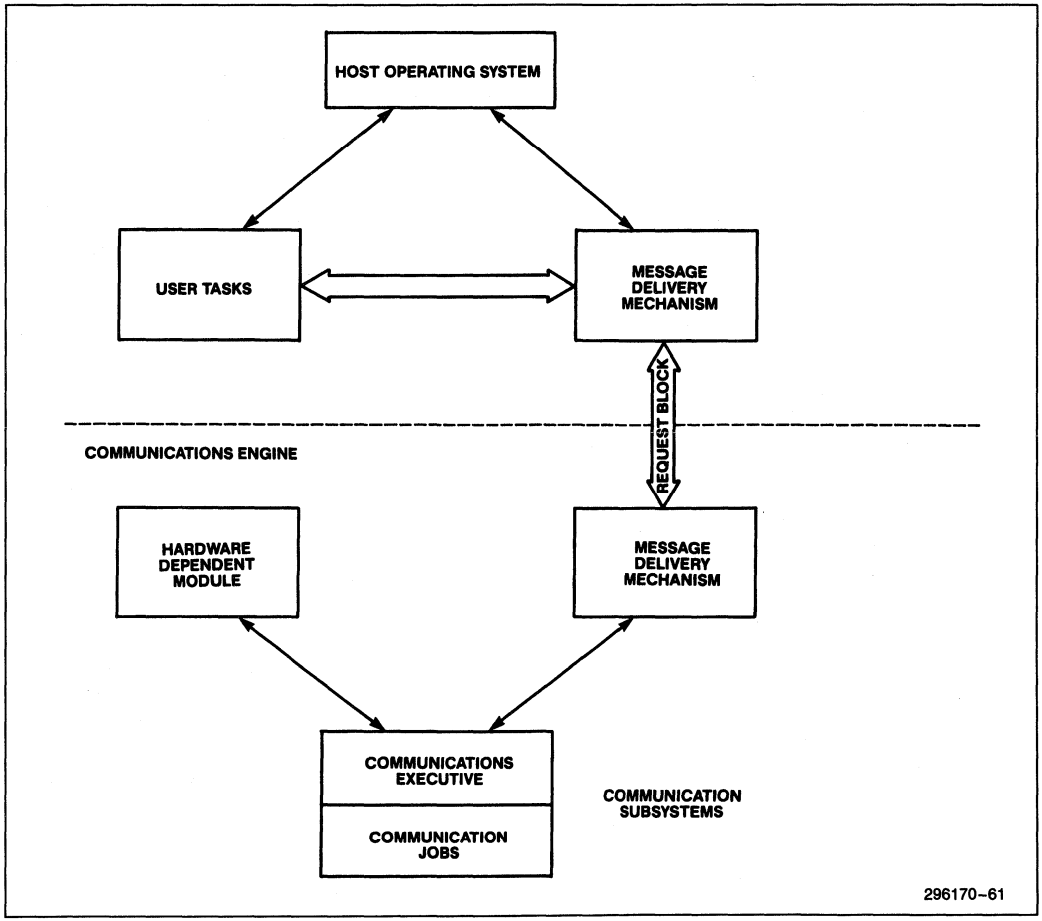


Figure 16. The Comm-Engine/Host CPU Software Interface

82586 Application Notes

3

intel

APPLICATION NOTE

AP-235

November 1986

An 82586 Data Link Driver

CHARLES YAGER

Order Number: 231421-002

INTRODUCTION

This application note describes a design example of an IEEE 802.2/802.3 compatible Data Link Driver using the 82586 LAN Coprocessor. The design example is based on the "Design Model" illustrated in "Programming the 82586". It is recommended that before reading this application note, the reader clearly understands the 82586 data structures and the Design Model given in "Programming the 82586".

"Programming the 82586" discusses two basic issues in the design of the 82586 data link driver. The first is how the 82586 handler fits into the operating system. One approach is that the 82586 handler is treated as a "special kind of interface" rather than a standard I/O interface. The special interface means a special driver that has the advantage of utilizing the 82586 features to enhance performance. However the performance enhancement is at the expense of device dependent upper layer software which precludes the use of a standard I/O interface.

The second issue "Programming the 82586" discusses is which algorithms to choose for the CPU to control the 82586. The algorithms used in this data link design are taken directly from "Programming the 82586". Command processing uses a linear static list, while receive processing uses a linear dynamic list.

The application example is written in C and uses the Intel C compiler. The target hardware for the Data Link Driver is the iSBC 186/51 COMMputer, however a version of the software is also available to run on the LANHIB Demo board.

1.0 FITTING THE SOFTWARE INTO THE OSI MODEL

The application example consists of four software modules:

- Data Link Driver (DLD): drives the 82586, also known as the 82586 Handler.
- Logical Link Control (LLC): implements the IEEE 802.2 standard.
- User Application (UAP): exercises the other software modules and runs a specific application.
- C hardware support: written in assembly language, supports the Intel C compiler for I/O, interrupts, and run time initialization for target hardware.

Figure 1 illustrates how these software modules combined with the 82586, 82501 and 82502 complete the first two layers of the OSI model. The 82502 implements an IEEE 802.3 compatible transceiver, while the 82501 completes the Physical layer by performing the serial interface encode/decode function.

The Data Link Layer, as defined in the IEEE 802 standard documents, is divided into two sublayers: the Logical Link Control (LLC) and the Medium Access Control (MAC) sublayers. The Medium Access Control sublayer is further divided into the 82586 Coprocessor plus the 82586 Handler. On top of the MAC is the LLC software module which provides IEEE 802.2 compatibility. The LLC software module implements the Station Component responses, dynamic addition and deletion of Service Access Points (SAPs), and a class 1 level of service. (For more information on the LLC sublayer, refer to IEEE 802.2 Logical Link Control Draft Standard.) The class 1 level of service provides a connectionless datagram interface as opposed to the class 2 level of service which provides a connection oriented level of service similar to HDLC Asynchronous Balanced Mode.

On top of the Data Link Layer is the Upper Layer Communications Software (ULCS). This contains the Network, Transport, Session, and Presentation Layers. These layers are not included in the design example, therefore the application layer of this ap note interfaces directly to the Data Link layer.

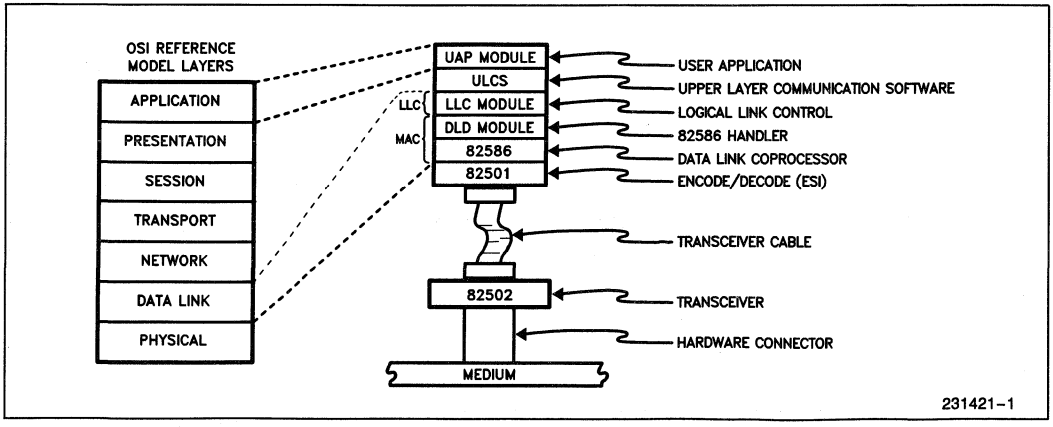


Figure 1. Data Link Driver's Relationship to OSI Reference Mode 1

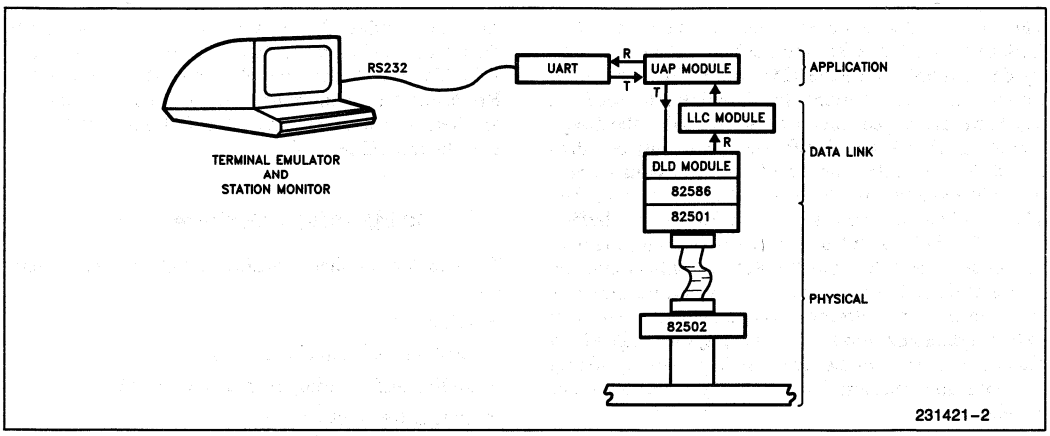


Figure 2. Block Diagram of the Hardware and Software

The application layer is implemented in the User Application (UAP) software module. The UAP module operates in one of three modes: Terminal Mode, Monitor Mode, and High Speed Transmit Mode. The software initially enters a menu driven interface which allows the program to modify several network parameters or enter one of the three modes.

The Terminal Mode implements a virtual terminal with datagram capability (connectionless "class 1" service). This mode can also be thought of as an async to IEEE 802.3/802.2 protocol converter.

The Monitor Mode provides a dynamic update on the terminal of 6 station related parameters. While in the monitor mode, any size frame can be repeatedly transmitted to the cable in a software loop.

High Speed Transmit Mode transmits frames to the cable as fast as the software possibly can. This mode demonstrates the throughput performance of the Data Link Driver.

The UAP gathers network statistics in all three modes as well as when it is in the menu. In addition, the UAP module provides the capability to alter MAC and LLC addresses and re-initialize the data link. (Figure 2 shows a combined software and hardware block diagram.)

2.0 LARGE MODEL COMPILATION

All the modules in this design example are compiled under the Large Model option. This has the advantages of using the entire 1 Mbyte address space, and allowing the string constants to be stored in ROM. In the Large Model it is important to consider that the 82586's data structures, SCB, CB, TBD, FD, and RBD, must reside within the same data segment. This data segment is determined at locate time.

The C_Assy_Support module has a run time start off function which loads the DLD data segment into a global variable SEGMT_. This data segment is used by the 82586 Handler for address translation purposes. The 82586 uses a flat address while the 80186 uses a segmented address. Any time a conversion between 82586 and 80186 addresses are needed the SEGMT_ variable is used.

Pointers for the 80186 in the large model are 32 bits, segment and offset. All the 82586 link pointers are 16 bit offsets. Therefore when trading pointers between the 82586 and the 80186, two functions are called: Offset (ptr), and Build_Ptr (offset). Offset (ptr) takes a 32 bit 80186 pointer and returns just the offset portion for the 82586 link pointer. While Build_Ptr (offset) takes an 82586 link pointer and returns a 32 bit 80186 pointer, with the segment part being the SEGMT_ variable. Offset ( ) and Build_Ptr( ) are simple functions written in assembly language included in the C_Assy_Support module.

In the small model, Offset ( ) and Build_Ptr( ) are not needed, but the variable SEGMT_ is still needed for determining the SCB pointer in the ISCP, and in the Transmit and Receive Buffer Descriptors.

3.0 THE 82586 HANDLER

3.1 The Buffer Model

The buffer model chosen for the 82586 Handler is the "Design Model" as described in "Programming the 82586". This is based on the 82586 driver as a special driver rather than as a standard driver. Using this approach the ULCS directly accesses the 82586's Transmit and Receive Buffers, Buffer Descriptors and Frame Descriptors. This eliminates buffer copying. Transmit and receiver buffer passing is done entirely through pointers.

The only hardware dependencies between the Data Link and ULCS interface are the buffer structures. The ULCS does not handle the 82586's CBs, SCB or initialization structures. To isolate the data link interface from any hardware dependencies while still using the design model, another level of buffer copying must be introduced. For example, when the ULCS transmits a frame it would have to pass its own buffers to the data link. The data link then copies the data from ULCS buffers into 82586 buffers. When a frame is received, the data link copies the data from the 82586's buffers into the ULCS buffers. The more copying that is done the slower the throughput. However, this may be the only way to fit the data link into the operating system. The 82586 Handler can be made hardware independent by adding a receive and transmit function to perform the buffer copying.

The 82586 Handler allocates buffers from two pools of memory: the Transmit pool, and the Receive pool as illustrated in Figure 3. The Transmit pool contains Transmit Buffer Descriptors (TBDs) and Transmit Buffers (TBs). The Receive pool contains Frame Descriptors (FDs), Receive Buffer Descriptors (RBDs), and Receive Buffers (RBs).

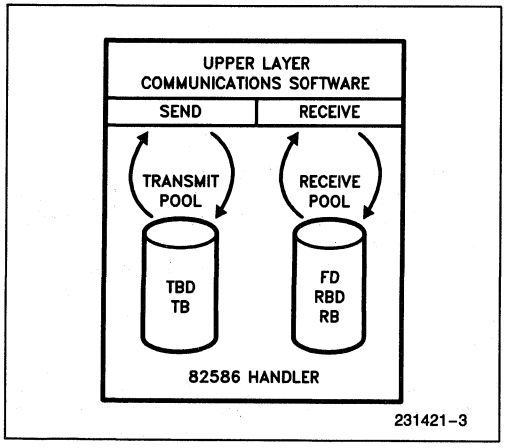


Figure 3. 82586 Handler Memory Management Model

When the ULCS wants to transmit, it requests a TBD from the handler. The handler returns a pointer to a free TBD. Each TBD has a TB attached to it. The ULCS fills the buffer, sets the appropriate fields in the TBD, and passes the TBD pointer back to the handler for transmission. After the frame is transmitted, the handler places the TBD back into the free TBD pool. If the ULCS needs more than one buffer per frame, it simply requests another TBD from the handler and performs the necessary linkage to the previous TBD.

On the receive side, the RFA pool is managed by the 82586 itself. When a frame is received, the 82586 interrupts the handler. The handler passes a FD pointer to the ULCS. Linked to the FD is one or more RBDs and RBs. The ULCS extracts what it needs from the FD, RBDs and RBs, and returns the FD pointer back to the handler. The handler places the FD and RBDs back into the free RFA pool.

3.2 The Handler Interface

The handler interface provides the following basic functions:

- initialization
- sending and receiving frames
- adding and deleting multicast addresses
- getting transmit buffers
- returning receive buffers

Figure 4 lists the Handler Interface functions.

On power up, the initialization function is called. This function initializes the 82586, and performs diagnostics. After initialization, the handler is ready to transmit and receive frames, and add and delete multicast addresses.

To send a frame, the ULCS gets one or more transmit buffers from the handler, fills them with data, and calls the send function. When a frame is received, the handler calls a receive function in the ULCS. The ULCS receive function removes the information it needs and returns the receive buffers to the handler. The addition and deletion of multicast addresses can be done "on the fly" any time after initialization. The **receiver doesn't have to be disabled** when this is done.

The command interface to the handler is totally asynchronous—the ULCS can issue transmit commands or multicast address commands whenever it wants. The commands are queued by the handler for the 82586 to execute. If the command queue is full, the send frame procedure returns a false status rather than true. The size of the command queue can be set at compile time by setting the CB—CNT constant. Typically the command queue never has more than a few commands on it because the 82586 can execute commands faster than the ULCS can issue them. This is not the case in a heavily loaded network when deferrals, collisions, and retries occur.

The command interface to the 82586 handler is hardware independent; the only hardware dependence is the buffering. A hardware independent command interface doesn't have any performance penalty, but some 82586 programmability is lost. This shouldn't be of concern since most data links do not change configuration parameters during operation. One can simply modify a few constants and recompile to change frame and network parameters to support other data links.

Handler Interface Functions	Description
Init_586( )	Initialize the Handler
Send_Frame (ptbd, padd)	Sends a frame to the cable. ptbd—Transmit Buffer Descriptor pointer padd—Destination Address pointer
Recv_Frame (pfd)	Handler calls this function which resides in the ULCS. pfd_Frame Descriptor pointer
Add_Multicast_Address (pma)	Adds one multicast address pma—Multicast Address pointer
Delete_Multicast_Address (pma)	Deletes one multicast address
Get_Tbd( )	Get a Transmit Buffer Descriptor pointer
Put_Free_Rfa (pfd)	Returns a Frame Descriptor and Receive Buffer Descriptors to the 82586.

Figure 4. List of Handler Interface Functions

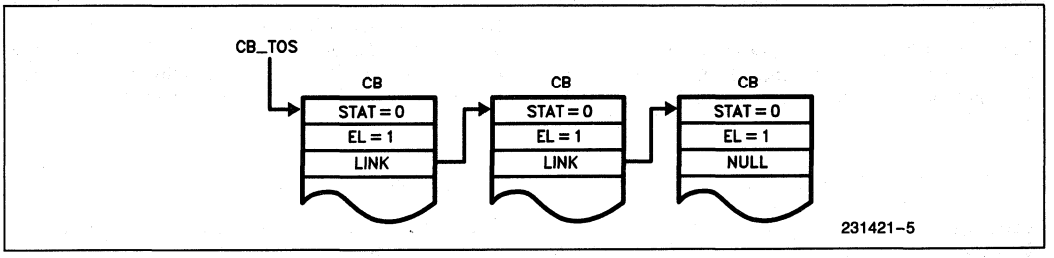


Figure 5. Free CB Pool

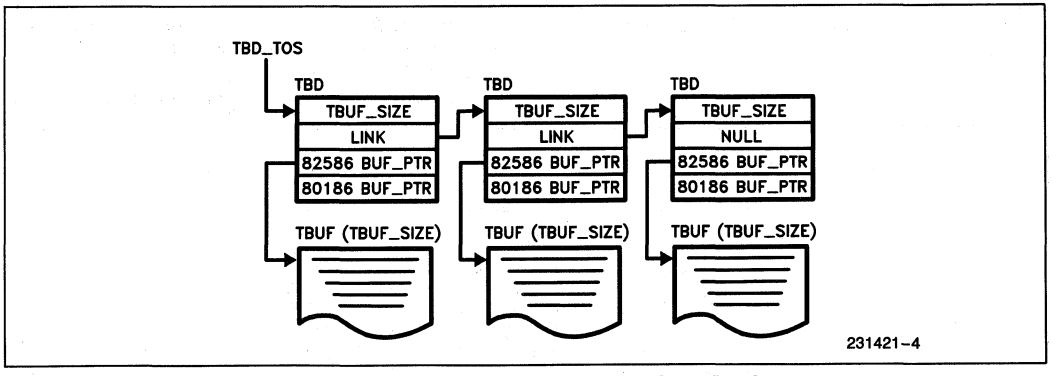


Figure 6. Free Transmit Buffer Descriptor Pool

3.3 Initialization

The function which initializes the 82586 handler, Init_586( ), is called by the ULCS on power up or reinitialization. **Before this function is called, an 82586 hardware or software reset should occur.** The Initialization occurs in three phases. The first phase is to initialize the memory. This includes flags, vectors, counters, and data structures. The second phase is to initialize the 82586. The third phase is to perform self test diagnostics. Init_586( ) returns a status byte indicating the results of the diagnostics.

Init_586( ) begins by toggling the 82501 loopback pin. If the 82501 is powered up in loopback, the $\overline{CRS}$ and $\overline{CDT}$ pin may be active. To reset this condition, the loopback pin is toggled. The 82501 should remain in loopback for the first part of the initialization function.

Phase 1 executes initialization of all the handlers flags, interrupt vectors, counters, and 82586 data structures. There are two separate functions which initialize the CB and RFA pools: Build_CB( ) and Build_Rfa( ).

3.3.1 BUILDING THE CB AND RFA POOLS

Build_CB( ) builds a stack of free linked Command Blocks, and another stack of free linked Transmit Buffer Descriptors. (See Figures 5 and 6.) Each stack has a Top of Stack pointer, which points to the next free structure. The last structure on the list has a NULL link pointer.

The CBs within the list are initialized with 0 status, EL bit set, and a link to the next CB. The TBD structures are initialized with the buffer size, which is set at compile time with the TBUF_SIZE constant, a link to the next TBD, and an 82586 pointer to the transmit buffer. This pointer is a 24 bit flat/physical address. The address is built by taking the transmit buffer's data segment address, shifting it to the left by 4 and adding it to the transmit buffer offset. An 80186 pointer to the transmit buffer is added to the TBD structure so that the 80186 does not have to translate the address each time it accesses the transmit buffer.

Build_Rfa( ) builds a linear linked Frame Descriptor list and a Receive Buffer Descriptor list as shown in Figure 7. The status and EL bits for all the free FDs are 0. The last FD's EL bit is 1 and link pointer is NULL. The first FD on the FD list points to the first RBD on the RBD list. The RBDs are initialized with both 82586 and 80186 buffer pointers. The 80186 buffer pointer is added to the end of the RBD structure. Begin and end pointers are used to mark the boundaries of the free lists.

3.3.2 82586 INITIALIZATION

The 82586 initialization data structure SCP is already set since it resides in ROM, however, the ISCP must be loaded with information. Within the SCP ROM is the pointer to the ISCP; the ISCP is the only absolute address needed in the software. Once the ISCP address is determined, the ISCP can be loaded. The SCB base is obtained from the C_Assy_Support module. The global variable SEGMT_contains the address of the

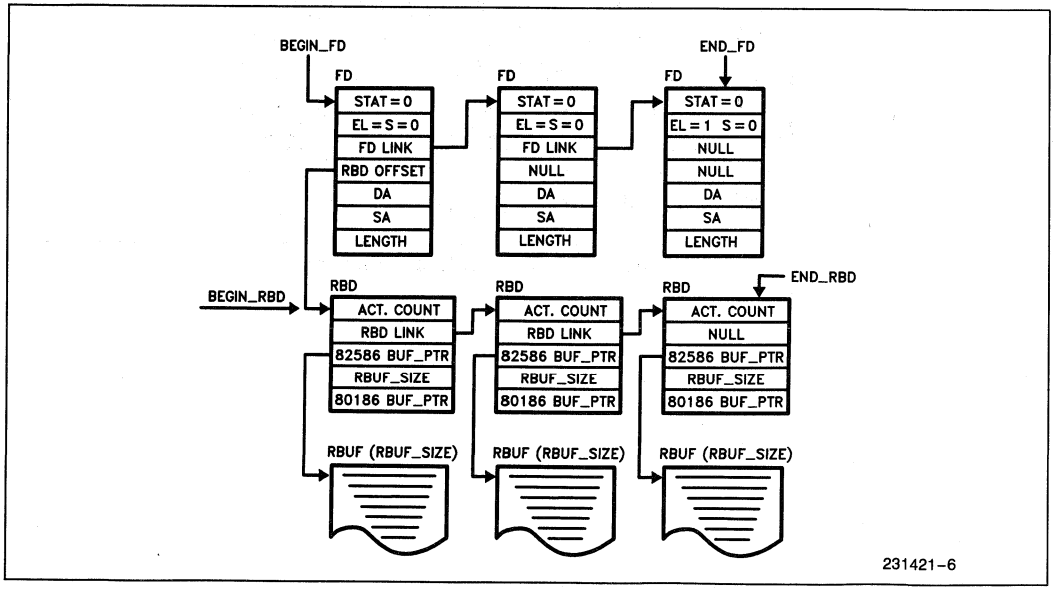


Figure 7. Free RFA

data segment of the handler. The 80186 shifts this value to the left by 4 and loads it into the SCB base. The SCB offset is now determined by taking the 32 bit SCB pointer and passing it to the Offset( ) function.

The 82586 interrupt is disabled during initialization because the interrupt function is not designed to handle 82586 reset interrupts. To determime when the 82586 is finished with its reset/initialization, the SCB status is polled for both the CX and CNA bits to be set. After the 82586 is initialized, both the CX and CNA interrupts are acknowledged.

The 82586 is now ready to execute commands. The Configuration is executed first to place the 82586 in internal loopback mode, followed by the IA command. The address for the IA command is read off of a prom on the PC board.

3.3.3 SELF TEST DIAGNOSTICS

The final phase of the handler initialization is to run the self test diagnostics. Four tests are executed: Diagnose command, Internal loopback, External loopback through the 82501, and External loopback through the transceiver. If these four tests pass, the data link is ready to go on line.

The function that executes these diagnostics is called Test_Link( ). If any of the tests fail, Test_Link( ) returns immediately with the Self_Test global variable set to the type of failure. This Self_Test global variable is then returned to the function which originally called Init_586( ). Therefore Init_586( ) can return one of five results: FAILED_DIAGNOSE, FAILED_LPBK_INTERNAL, FAILED_LPBK_EXTERNAL, FAILED_LPBK_TRANSCEIVER or PASSED.

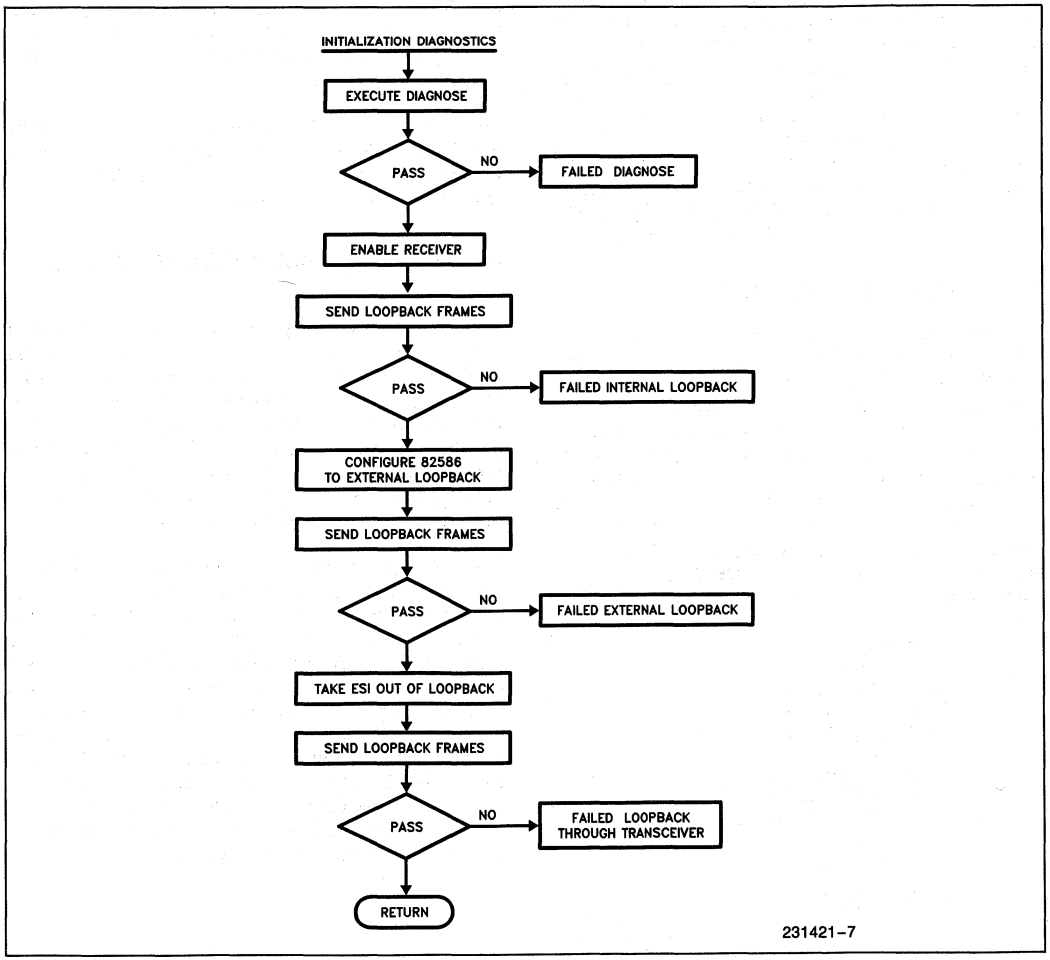


Figure 8. Initialization Diagnostics: Test_Link ()

The Diagnose( ) function, called by Test_Link( ), does not return until the diagnose command is completed. If the interrupt service routine detects that a Diagnose command was completed then it sets a flag to allow the Diagnose( ) function to return, and it also sets the Self_Test variable to FAIL if the Diagnose command failed. If the Diagnose command completed successfully, the loopback tests are performed.

Before any loopback tests are executed, the Receive Unit is enabled by calling Ru_Start( ). Loopback tests begin by calling Send_Lpbk_Frame( ), which sends 8 frames with known loopback data and its own destination address. More than one loopback frame is sent in case one or more of them are lost. Also several of the frames will have been received by the time flags.lpbk_ test is checked.

Two flag bits are used for the loopback tests: flags.lpbk_mode, and flags.lpbk_test. flags.lpbk_ mode is used to indicate to the receive section that the frames received are potentially loopback frames. The receive section will pass receive frames to the Loopback Check( ) function if the flags.lpbk_mode bit is set. The Loopback_Check( ) function first compares the source address of the frame with its station address. If this matches then the data is checked with the known loopback data. If the data matches, then the flags.lpbk_test bit is set, indicating a successful loopback. The flow of the Test_Link( ) function is displayed in Figure 8.

3.4 Command Processing

Command blocks are queued up on a static list for the 82586 to execute. The flow of a command block is given in Figure 9. When the handler executes a command it first has to get a free command block. It does this by calling Get_CB( ) which returns a pointer to a free command block. The CB structure is a generic one in which all commands except the MC-Setup can fit in. The handler then loads into the CB structure the type of command and associated parameters. To issue the command to the 82586 the Issue_CU_Cmd( ) function is called with the pointer to the CB passed to this function. Issue_CU_Cmd( ) places the command on the 82586's static command block list. After the 82586 executes the command, it generates an interrupt. The interrupt routine, Isr_586( ), processes the command and returns the Command Block to the free command block list by calling Put_Cb( ).

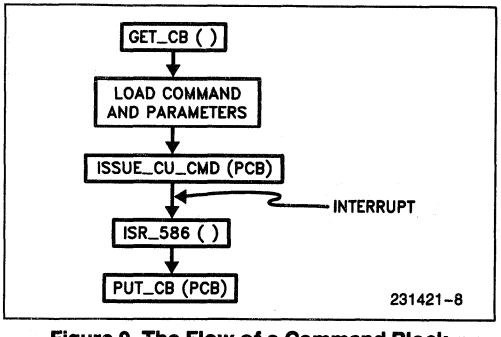


Figure 9. The Flow of a Command Block

3.4.1 ACCESSING COMMAND BLOCKS–GET_ CB() and PUT_CB()

Get_Cb( ) returns a pointer to a free command block. The free command blocks are in a linear linked list structure which is treated as a stack. The pointer cb_ tos points to the next available CB. Each time a CB is requested, Get_Cb( ) pops a CB off the stack. It does this by returning the pointer of cb_tos. cb_tos is then updated with the CB's link pointer. When the CB list is empty, Get_Cb( ) returns NULL.

There are two types of nulls, the 82586 'NULL' is a 16 bit offset, OFFFFH, in the 82586 data structures. The 80186 null pointer, 'pNULL', is a 32 bit pointer; with OFFFFH offset and the 82586 handler's data segment, SEGMT_, as the base.

Put_Cb( ) pushes a free command block back on the list. It does this by placing the cb_tos variable in the returned CB's link pointer field, then updates cb_tos with the pointer to the returned CB.

3.4.2 ISSUING CU COMMANDS–ISSUE_CU_ CMD()

This function queues up a command for the 82586 to execute. Since static lists are used, each command has its EL bit set. There is a begin_cbl pointer and an end_cbl pointer to delineate the 82586's static list. If there are no CBs on the list, then begin_cbl is set to pNULL. (Figure 10 illustrates the static list.) Each time a command is issued, a deadman timer is set. When the 82586 interrupts the CPU with a command completed, the deadman timer is reset.

Issue_Cu_Cmd( ) begins by disabling the 82586's interrupt. It then determines whether the list is empty or not. If the list is empty, begin and end pointers are loaded with the CB's address. The CU must then be started. Before a CU_START can be issued, the SCB's cbl_offset field must be loaded with the address of the command, the Wait_Scb( ) function must be called to insure that the SCB is ready to accept a command, and the deadman timer must be initialized. If the list is not empty, then the command block is queued at the end of the list, and the interrupt service routine Isr_586( ), will continue generating CAs for each command linked on the CB list until the list is empty.

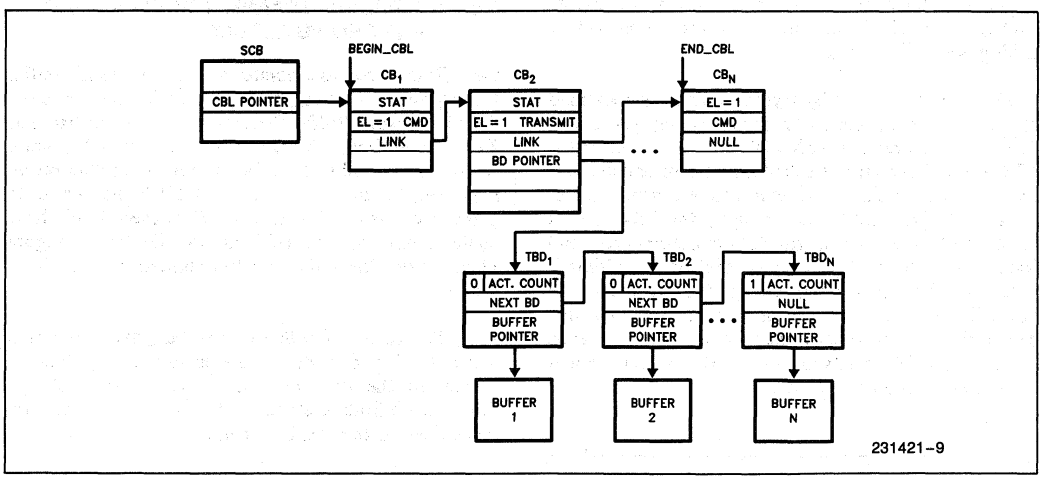


Figure 10. The Static Command Block List

3.4.3 INTERRUPT SERVICE ROUTINE-ISR__586()

Isr__586( ) starts off by saving the interrupts that were generated by the 82586 and acknowledging them. Acknowledgment must be done immediately because if a second interrupt were generated before the acknowledgment, the second interrupt would be missed. The interrupt status is then checked for a receive interrupt and if one occurred the Recv__Int__Processing( ) function is called. After receive processing is check the CPU checks whether a command interrupt occurred. If one did, then the deadman timer is reset and the results of the command are checked. There are only two particular commands which the interrupt results are checked for: Transmit and Diagnose. The Diagnose command needs to be tested to see if it passed, plus the diagnose status flag needs to be set so that the initialization process can continue.

The transmit command status provides network management and station diagnostic information which is useful for the "Network Management" function of the ISO model. The following statistics are gathered in the interrupt routine: good__transmit__cnt, sqe__err__cnt, defer__cnt, no__crs__cnt, underrun__cnt, max__col__cnt. To speed up transmit interrupt processing a flag is tested to determine whether these statistics are desired, if not this section of code is skipped.

The sqe error requires special considerations when used for statistic gathering or diagnostics. The sqe status bit indicates whether the transceiver passed its self test or not. The transceiver executes a self test after each transmission. If the transceiver's self test passed, it will activate the collision signal during the IFS time.

The sqe status bit will be set if the transceiver's self test passed. However if the sqe status bit is not set, the transceiver may still have passed its self test. Several events can prevent the sqe bit from being set. For example, the first transmit command status after power up will not have the sqe bit set because the sqe is always from the previous command. Also if any collisions occur, the sqe bit might not be set. This has to do with the timing of when the sqe signal comes from the transceiver. It is possible that a JAM signal from a remote station can overlap the sqe signal in which case the 82586 will not set the sqe status bit. Therefore the sqe error count should only be recorded when no collisions occur.

One other situation can occur which will prevent the SQE status bit from being set. If transmit command reaches the maximum retry count, the next transmit command's SQE bit will not be set.

The final phase of interrupt command processing determines if another command is linked, and returns the CB to the free command block list. Another command being linked is indicated by the CB link field not being NULL. In this case the deadman timer and the 82586's CU are re-started. If the CB link is NULL, there are no further commands to execute, and begin__cbl is set to pNULL.

3.4.4 SENDING FRAMES-SEND__FRAME (PTBD, PADD)

Send__Frame( ) receives two parameters, a pointer to the first Transmit Buffer Descriptor, and a pointer to the destination address. There may be one or more TBDs attached. The last TBD is indicated by its link

field being NULL and the EOF bit set. It is the responsibility of the ULCS to make sure this is done before calling Send_Frame( ).

Send_Frame( ) begins by trying to obtain a command block. If the free command block list is empty, the send frame function returns with a false result. It is up to the ULCS to either continue attempting transmission or attempt at a later time. The send frame function calculates the length field by summing up the TBDs actual count field. After the length field is determined, send frame checks to see if padding is required. If padding is necessary, Send Frame will change the act count field in the TBD to meet the minimum frame requirements. This technique transmits what ever was in the buffer as padding data. If security is an issue, the padding data in the buffer should be changed.

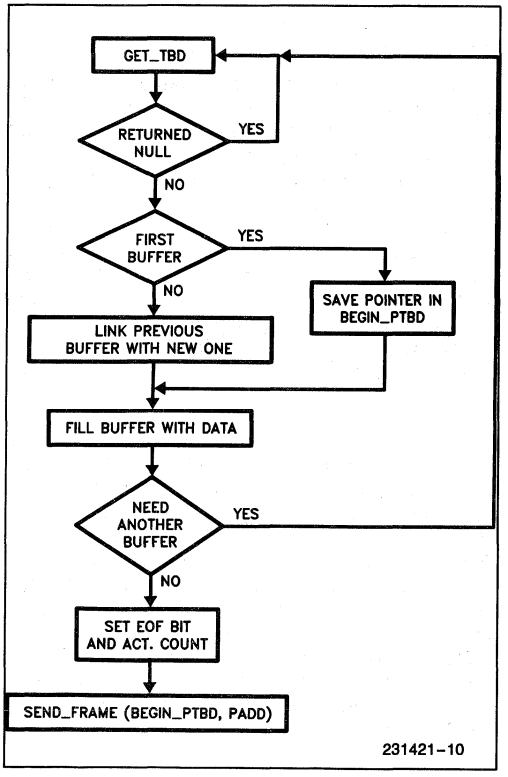


Figure 11. Flow Chart for Sending a Frame

3.4.5 ACCESSING TRANSMIT BUFFERS–GET_TBD() AND PUT_TBD()

Get_Tbd( ) returns a pointer to a free Transmit Buffer Descriptor, and Put_Tbd( ) returns one or more linked Transmit Buffer Descriptors to the free list. The TBD which Get_Tbd( ) allocates has its link pointer set to NULL, and its EOF bit cleared. If another buffer is needed, the link field in the old TBD must be set to point to the new TBD. The last TBD used should have its link pointer set to NULL and its EOF bit set. Figure 11 shows the flow chart of getting buffers and sending a frame.

Put_Tbd (ptbd) is called by the Isr_586( ) function when the 82586 is done transmitting the buffers. A pointer to the first TBD is passed to Put_Tbd( ). Put_Tbd( ) finds the end of the list of TBDs and returns them to the free buffer list.

3.4.6 MULTICAST ADDRESSES

The 82586 handler maintains a table of multicast addresses. Initially this table is empty. To enable a multicast address the Add_Multicast_Address(pma) function is called; to disable a multicast address, Delete_Multicast_Address(pma) function is called. Both functions accept a parameter which points to the multicast address. Add and Delete functions perform linear searches through the Multicast Address Table (MAT).

Add scans the entire MAT once to check if the address being added is a duplicate of one already loaded. Add will not enter a duplicate muilticast address. If there are no duplicates Add goes to the beginning of the MAT and looks for a free location. If it finds one, it loads the new address into the free location and sets the location status to INUSE. If no free locations are available, Add returns a false result.

Delete looks for a used location in the MAT. When it finds one, it compares the address in the table with the address passed to it. If they match, the location status is set to FREE and a TRUE result is returned. If no match occurs, the result returned is FALSE.

If Add or Delete change the MAT, they update the 82586 by calling Set_Multicast_Address( ). This function executes an 82586 MC Setup command. Set_Mulitcast_Address( ) uses the addresses in the MAT to build the MC Setup command. The MC Setup command is too big to be built from the free CBs. Free CB

command blocks are 18 bytes long, while the MC Setup command can be up to 16,392 bytes. Therefore a separate Multicast Address Command Block (ma_cb) must be allocated and used. The size of the ma_cb and MAT are determined at compile time based on the MULTI_ADDR_CNT constant. The design example allows up to 16 multicast addresses.

Since there is only one ma_cb, and it is not compatible with the other CBs, it must be treated differently. Only one ma_cb can be on the 82586 command list. The ma_cb command word is used as a semaphore. If it is zero, the command is available. If not, Set_Multicast_Address( ) must wait until the ma_cb is free. Also the interrupt routine can't return the ma_cb to the free CB list. It just clears the cmd field, to indicate that ma_cb is available.

The 82586's receiver does not have to be disabled to execute the MC Setup command. If the 82586 is receiving while this command is accessed, the 82586 will finish reception before executing the MC Setup comand. If the MC Setup command is executing, the 82586 automatically ignores incoming frames until the MC Setup is completed. Therefore multicast addresses can be added and deleted on the fly.

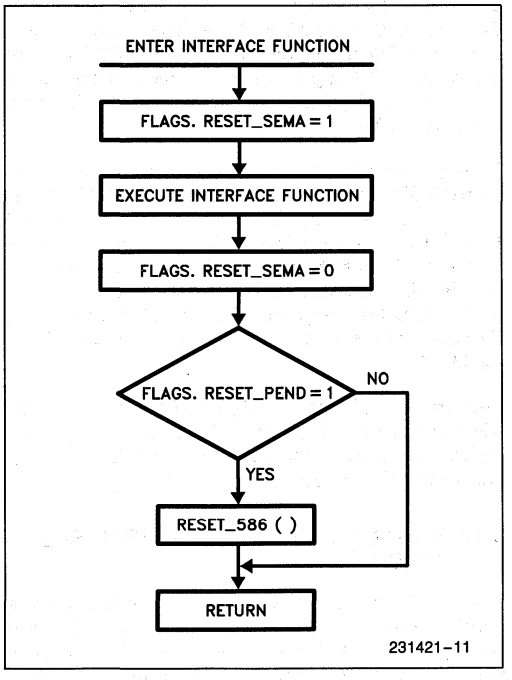


Figure 12. Reset Semaphore

3.4.7 RESETTING THE 82586-RESET_586()

The 82586 rarely if ever locks up in a well behaved network; (i.e. one that obeys IEEE 802.3 specifications). The lock-ups identified were artificially created and would normally not occur. This data link driver has been tested in an 8 station network under various loading conditions. No lock-ups occurred under any of the data link drivers test conditions. However the reset software has been tested by simulating a lockup. This can be done by having the 82586 transmit, and disabling the CTS pin for a time longer than the deadman timer.

An 82586 deadlock is not a fatal error. The handler is designed to recover from this problem. As mentioned before, each time the 82586 is given a CA to begin executing a command, a deadman timer is set. The deadman timer is reset when a CNR interrupt is generated. If the CNR interrupt is not generated before the deadman timer expires, the 82586 must be reset.

Resetting of the 82586 should not be done while the handler software is executing. This could create a software deadlock by interrupting a critical section of code in the handler. To insure that the Reset_586( ) function is not executed while the handler is executing, all of the entry points to the handler (i.e. interface functions) set a semaphore flag bit called flags.reset_sema. This flag is cleared when the interface functions are exited.

If the Deadman timer interrupt occurs while flags.reset_sema is set, another flag is set (flag.reset_pend) indicating that the Reset_586( ) function should be called when the interface functions are exited. However if the deadman timer interrupt occurs when flags.reset_sema is clear, Reset_586( ) is called immediately. Figure 12 shows the logic for entering and exiting interface functions.

Reset_586( ) begins by disabling the 82586 interrupt, placing the ESI in loopback, and resetting the 82586. The reset can be a software or a hardware reset. However, there are certain lockups in the 82586 where only a hardware reset will suffice. (The 82586 errata sheet explicitly indicates which deadlocks require a hardware reset.) After the reset, Reset_586( ) executes a Configure, IA-Setup, and a MC-Setup command; the MC_Setup command is built from the multicast address table (MAT). The 82586 Command Queues and Receive Frame Queues are left untouched so that the 82586 can continue executing where it left off before the deadlock. This way no frames or commands are lost. This requires that a separate reset CB and reset Multicast CB is used, because other CBs already in use cannot be disturbed.

3.5 Receive Frame Processing

The following functions are used for Receive Frame Processing:

Recv_Int_Processing( )	Called by Isr_586( ) to remove FDs and RBDs from the 82586's RFA
Recv_Frame (pfd)	Called by Recv_Int_Processing( ). This function resides in the ULCS
Check_Multicast (pfd)	Used for perfect Multicast filtering
Put_Free_Rfa (pfd)	Returns FDs and RBDs to the 82586's RFA
Ru_Start( )	Restarts the RU when in the IDLE or No Resources state.

3.5.1 RECEIVE INTERRUPT PROCESSING-RECV_INT_PROCESSING()

The Recv_Int_Processing( ) function is called by Isr_586( ) when the FR bit in the SCB is set. The Recv_Int_Processing( ) function checks whether any FDs and RBDs on the free list have been used by the 82586. If they have, Recv_Int_Processing( ) removes the used FDs and RBDs from the free list, and passes them to the ULCS.

The Recv_Int_Processing( ) function is a loop where each pass removes a frame from the 82586's RFA. When there are no more used FDs and RBDs on the RFA, the function calls RU_Start( ), then returns to Isr_586( ). The first part of the loop checks to see if the C bit in the first FD of the free FD list is set. If the C bit is set, the function determines if one or more RBDs are attached. If there are RBDs attached, the end of the RBD list is found. The last RBD's link field is used to update begin_rbd pointer, and then it's set to NULL.

After the receive frame has been delineated from the RFA, some information about the frame is needed to determine which function to pass it to. Since the save bad frame configure bit is not set, the only bad frame on the list could be an out of resource frame. An out of resource frame is returned to the RFA by calling Put_Free_RFA (pfd). If the flags.lpbk_mode bit is set, the frame is given to the loopback check function. If the destination address of the frame indicates a multicast, the check multicast function is called. If the frame has passed all of the above tests and still has not been returned, it is passed to the Recv_Frame( ) function which resides in the ULCS.

Check_Multicast (pfd) determines whether the multicast address received is in the multicast address table. This is necessary because the 82586 does not have perfect multicast address filtering. Check_Multicast does a byte by byte comparison of the destination address with the addresses in the multicast address table. If no match occurs, it returns false, and Recv_Int_Processing calls Put_Free_RFA( ) to return the frame to the RFA. If there is a match, Check_Multicast( ) returns TRUE and Recv_Int_Processing( ) calls Recv_Frame( ), passing the pointer to the FD of the frame received.

3.5.2 RETURNING FDs AND RBDs-PUT_FREE_RFA (pfd)

Put_Free_RFA combines Supply_FD and Supply_RBD algorithms described in "Programming the 82586" into one function. The begin and end pointers delineate what the CPU believes is the beginning and end of the free list. The decision of whether to restart the RU is made when examining both the free FD list and the free RBD list. This is why two ru_start_flags are used, one for the FD list and one for the RBD list. Both flags are initialized to FALSE.

The function starts off by initializing the FD so that the EL bit is set, the status is 0, and the FD link field is NULL. The rbd pointer is saved before the rbd pointer field in the FD is set to NULL. The free FD list is examined and if it's empty, begin—fd and end—fd are loaded with the address of the FD being returned. In this case the RU should not be restarted, because there is only one FD on the free list. If the free FD list is not empty, the FD being returned is placed on the end of the list, the end pointer is updated, and the RU start flag is set TRUE.

To begin the RBD list processing the end of the returned RBD list is determined, and this last RBD's EL bit is set. If the free RBD list is empty, the returned RBD list becomes the free RBD list. If there is more than one RBD on the returned list, the ru start flag is set TRUE. If the free RBD list is not empty, the returned RBD list is appended on the end of the free list, the end—rbd pointer is updated, and the ru start flag is set TRUE.

The last part of Put_Free_RFA( ) is to determine whether to call RU_Start( ). Both ru start flags are ANDed together, and if the result is TRUE, the Ru_Start( ) function is called.

3.5.3 RESTARTING THE RECEIVE UNIT-RU_START()

The Ru_Start( ) function checks two things before it decides to restart the RU. The first thing it checks is whether the RU is already READY. If it is, there is no reason to restart it. If the RU is IDLE or in NO_RESOURCES, then the second thing to check is whether the first free FD on the free FD list has its C bit set. If it does, then the RU should not be restarted. The reason is that the free FD list should only contain free FDs

when the RU is started. If the C bit is set in the FD, then not all the used FD have been removed yet. If the RU is started when used FDs are still in the RFA, the 82586 will write over the used FDs and frames will be lost. Therefore Ru_Start( ) is exited if the first FD in the RFA has its C bit set. If the RU is not READY, and begin_fd doesn't point to a used FD, then the RU is restarted.

Note that in "Programming the 82586" there are two more conditions to be met before the RU is started: two or more FD on the RFA, and two or more RBD on the RFA. These conditions are checked in Put_Free_RFA( ), and Ru_Start( ) isn't called unless they are met.

4.0 LOGICAL LINK CONTROL

The IEEE 802.2 LLC function completes the Data Link Layer of the OSI model. The LLC module in this design example implements a class 1 level of service which provides a connectionless datagram interface. Several data link users or processes can run on top of the data link layer. Each user is identified by a link service access point (LSAP). Communication between data link users is via LSAPs. An LSAP is an address that identifies a specific user process or another layer (see Figure 13). The LSAP addresses are defined as follows:

Data Link Layer (Station Component)	00H
Transport Layer	FEH
Network Management Layer	08H
User Processes	multiples of 4 in the range OCH < LSAP ≤ FCH

Each receiving process is identified by a destination LSAP (DSAP) and each sending process is identified by a source LSAP (SSAP). Before a destination process can receive a packet, its DSAP must be included in a list of active DSAPs for the data link.

Figure 14 illustrates the relationship between the Station Component and the SAP components. (The SAP components are user processes.) The Station Component receives all of the good frames from the Handler and checks the DSAP address. If the DSAP address is 0, then the frame is addressed to the Station Component and a Station Component Response is generated. If the DSAP address is on the active DSAP list, then the Station Component passes the frame to the addressed SAP. If the DSAP address is unknown, the frame is returned to the handler.

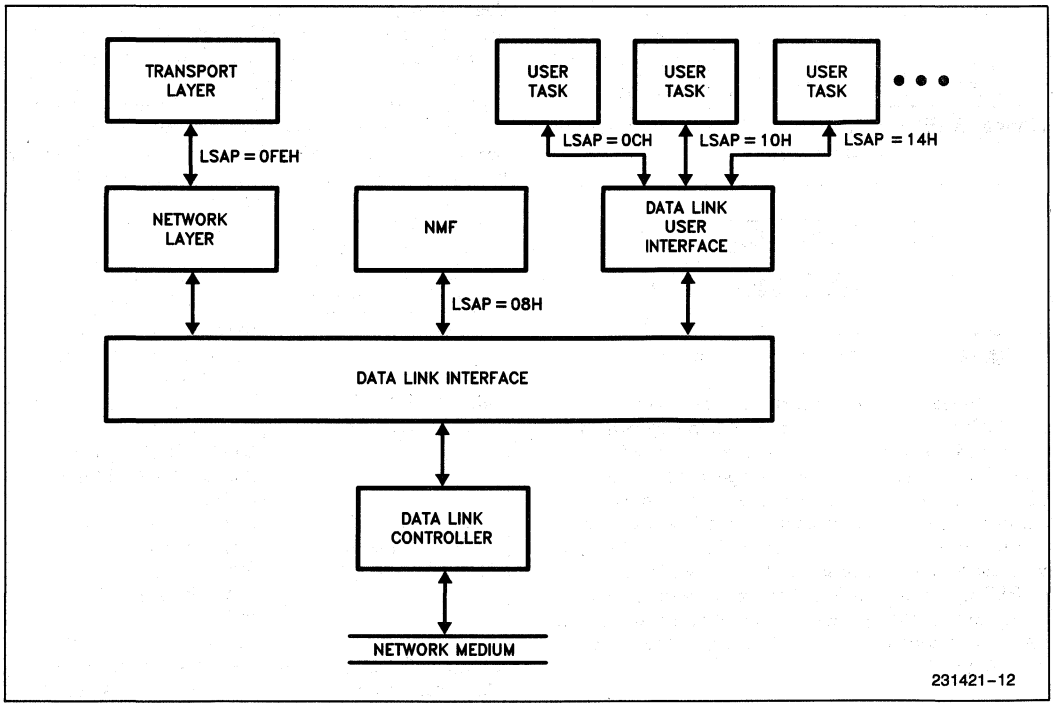


Figure 13. Data Link Interface

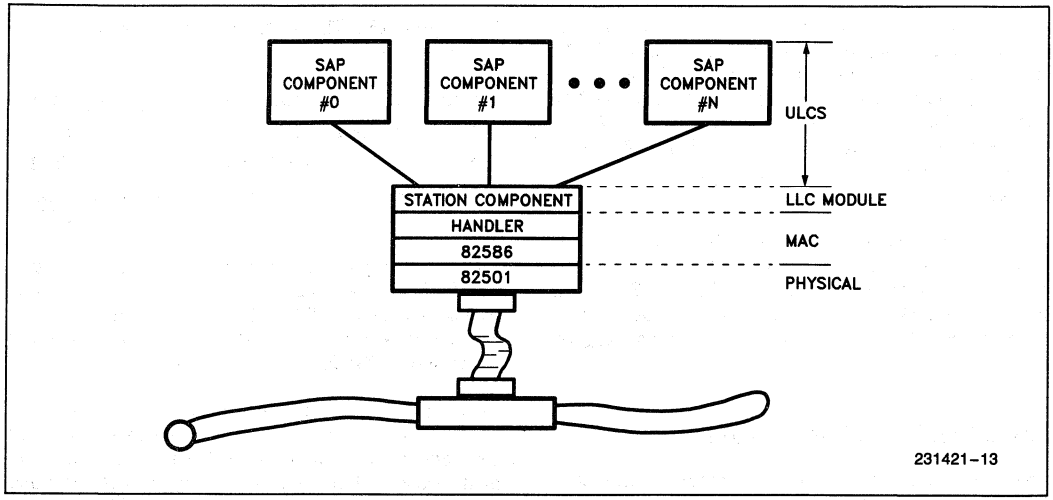


Figure 14. Station Component Relationship

There are 3 commands and 2 responses which the class 1 LLC layer must implement. Figure 15 shows IEEE 802.2 Class 1 commands and responses and Figure 16 shows the IEEE 802.2 Class 1 frame format.

Commands	Responses	Description
UI		Unnumbered Information
XID	XID	Exchange ID
TEST	TEST	Remote Loopback

Figure 15. IEEE 802.2 Class 1, Type 1 Commands and Responses

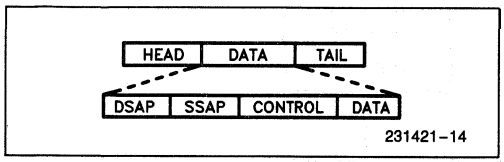


Figure 16. IEEE 802.2 Class 1 Frame Format

From Figure 15 it can be seen that there are no LLC class 1 UI responses because information frames are not acknowledged at the data link level. The only command frames that may require responses are XID and TEST. If a command frame is addressed to the Station Component, it checks the control field to see what type of frame it is. If it's an XID frame, the Station Component responds with a class 1 XID response frame. If it's a TEST frame, the Station Component responds with a TEST frame, echoing back the data it received. In both cases, the response frame is addressed to the source of the command frame.

Any frames addressed to active SAPs are passed directly to them. The Station Component will not respond to SAP addressed frames. Therefore it is the responsibility of the SAPs to recognize and respond to frames addressed to them. When a SAP transmits a frame, it builds the IEEE 802.2 frame itself and calls the Handler's Send_Frame( ) function directly. The LLC module is not used for SAP frame transmission. The only functions which the LLC module implement are the dynamic addition and deletion of DSAPs, multiplexing the frames to user SAPs, and the Station Component command recognition and responses. This is one implementation of the IEEE 802.2 standard. Other implementations may have the LLC module do more functions, such as SAP command recognitions and responses. A list of the functions included in the LLC module is as follows:

LLC Functions	Description
Init_Llc( )	Initializes the DSAP address table and calls Init_586( )
Add_Dsap_ Address (dsap, pfunc)	Add a DSAP address to the active list dsap - DSAP address pfunc - pointer to the SAP function
Delete—Dsap— Address (dsap)	Delete a DSAP address dsap - DSAP address
Recv—Frame (pfd)	Receives a frame from the 82586 Handler pfd - Frame Descriptor Pointer
Station—Component— Response (pfd)	Generates a response to a frame addressed to the Station Component pfd - Frame Descriptor Pointer

4.1 Adding and Deleting LSAPs

When a user process wants to add a LSAP to the active list, the process calls Add_Dsap_Address(dsap, pfunc). The dsap parameter is the actual DSAP address, and the pfunc parameter is the address of the function to be called when a frame with the associated DSAP address is received.

The LLC module maintains a table of active dsaps which consists of an array of structures. Each structure contains two members: stat - indicates whether the address is free or inuse, and (*p_sap_func)( ) contains the address of the function to call. The index into the array of structures is the DSAP address. This speeds up processing by eliminating a linear search. Delete_Dsap_Address (dsap) simply uses the DSAP index to mark the stat field FREE.

5.0 APPLICATION LAYER

For most networks the application layer resides on top of several other layers referred to here as ULCS. These other layers in the OSI model run from the network layer through the presentation layer. The implementation of the ULCS layers is beyond the scope of this application note, however Intel provides these layers as well as the data link layer with the OpenNET product line. For the purpose of this application note the application layer resides on top of the data link layer and its use is to demonstrate, exercise and test the data link layer design example.

There can be several processes sitting on top of the data link layer. Each process appears as a SAP to the data link. The UAP module, which implements the application layer, is the only SAP residing on top of the data link layer in this application example. Other SAPs could certainly be added such as additional "connectionless" terminals, a networking gateway, or a transport layer, however in the interest of time this was not done.

5.1 Application Layer Human Interface

The UAP provides a menu driven human interface via an async terminal connected to port B on the iSBC 186/51 board. The menu of the commands is listed in Figure 17 along with a description that follows:

Terminal Mode - implements a virtual terminal with datagram capability (connectionless "class 1" service). This mode can also be thought of as an async to IEEE 802.2/802.3 protocol converter.

Monitor Mode - allows the station to repeatedly transmit any size frame to the cable. While in the Monitor Mode, the terminal provides a dynamic update of 6 station related parameters.

High Speed Transmit Mode - sends frames to the cable as fast as the software possibly can. This mode demonstrates the throughput performance of the Data Link Driver.

Change Transmit Statistics - When Transmit Statistics is on several transmit statistics are gathered during transmission. If Transmit Statistics is off, statistics are not gathered and the program jumps over the section of code in the interrupt routine which gathers these statistics. The transmission rate is slightly increase when Transmit Statistics is off.

Print All Counters - Provides current information on the following counters.

Good frames transmitted:
Good frames received:

CRC errors received:
Alignment errors received:

Out of Resource frames:
Receiver overrun frames:

Each time a frame has been successfully transmitted the Good frames transmitted count is incremented. The same holds true for reception. CRC, Alignment, Out of Resources, and Overrun Errors are all obtained from the SCB. Underrun, lost CRS, SQE error, Max retry, and Frames that deferred are all transmit statistics that are obtained from the Transmit command status word. 82586 Reset is a count which is incremented each time the 82586 locks up. This count has never normally been incremented.

T - Terminal Mode
X - High Speed Transmit Mode
P - Print All Counters
A - Add a Multicast Address
S - Change the SSAP Address
N - Change Destination Node Address
R - Re-Initialize the Data Link
M - Monitor Mode
V - Change Transmit Statistics
C - Clear All Counters
Z - Delete a Multicast Address
D - Change the DSAP Address
L - Print All Addresses
B - Change the Number Base

Figure 17. Menu of Data Link Driver Commands

Clear All Counters - Resets all of the counters.

Add/Delete Multicast Address - Adds and Deletes Multicast Addresses.

Change SSAP Address - Deletes the previous SSAP and adds a new one to the active list. The SSAP in this case is this stations LSAP. When a frame is received, the DSAP address in the frame received is compared with any active LSAPs on the list. The SSAP is also used in the SSAP field of all transmitted frames.

Change DSAP Address - Delete the old DSAP and add a new one. The DSAP is the address of the LSAP which all transmit frames are sent to.

Change Destination Node Address - Address a new node.

Print All Addresses - Display on the terminal the station address, destination address, SSAP, DSAP, and all multicast addresses.

Re-initialize Data Link - This causes the Data Link to completely reinitialize itself. The 82586 is reset and reinitialized, and the selftest diagnostic and loopback tests are executed. The results of the diagnostics are printed on the terminal. The possible output messages from the 82586 selftest diagnostics are:

Passed Diagnostic Self Tests

Failed: Self Test Diagnose Command

Failed: Internal Loopback Self Test

Failed: External Loopback Self Test

Failed: External Loopback Through Transceiver Self Test

Change Base - Allows all numbers to be displayed in Hex or Decimal.

5.2 A Sample Session

The following text was taken directly from running the Data Link software on a 186/51 board. It begins with the iSDM monitor signing on and continues into executing the Data Link Driver software.

```
iSDM 86 Monitor, V1.0

Copyright 1983 Intel Corporation

.G D000:6

            ****************************************************************
            *                                                              *
            * 82586 IEEE 802.2/802.3 Compatible Data Link Driver           *
            *                                                              *
            ****************************************************************

Passed Diagnostic Self Tests

Enter the Address of the Destination Node in Hex -> 00AA0000179E

Enter this Station's LSAP in Hex -> 20

Enter the Destination Node's LSAP in Hex -> 20

Do you want to Load any Multicast Addresses? (Y or N) -> Y

Enter the Multicast Address in Hex -> 00AA00111111

Would you like to add another Multicast Address? (Y or N) -> N

This Station's Host Address is: 00AA00001868

The Address of the Destination Node is: 00AA0000179E

This Station's LSAP Address is: 20

The Address of the Destination LSAP is: 20

The following Multicast Addresses are enabled: 00AA00111111
```

```
Commands are:
T - Terminal Mode                          M - Monitor Mode
X - High Speed Transmit Mode               V - Change Transmit Statistics
P - Print All Counters                     C - Clear All Counters
A - Add a Multicast Address                Z - Delete a Multicast Address
S - Change the SSAP Address                D - Change the DSAP Address
N - Change Destination Node Address        L - Print All Addresses
R - Re-Initialize the Data Link            B - Change the number Base
Enter a command, type H for Help -> P
Good frames transmitted:     24      Good frames received:       1
CRC errors received:          0      Alignment errors received:  0
Out of Resource frames:       0      Receiver overrun frames:    0
82586 Reset:                  0      Transmit underrun frames:   0
Lost CRS:                     0      SQE errors:                 9
Maximum retry:                0      Frames that deferred:       4
Enter a command, type H for Help --> T
Would you like the local echo on? (Y or N) --> Y
This program will now enter the terminal mode.
Press ^C then CR to return back to the menu
Hello this is a test.
/*^C CR */
Enter a command, type H for Help --> M
Do you want this station to transmit? (Y or N) --> Y
Enter the number of data bytes in the frame --> 1500
Hit any key to exit Monitor Mode.
 # of Good      # of Good       CRC       Alignment       No       Receive
   Frames        Frames         Errors     Errors        Resource Overrun
Transmitted     Received                                  Errors   Errors
        32            0        00000      00000          00000    00000
/* CR */
Enter a command, type H for Help --> X
Hit any key to exit High Speed Transmit Mode.
/* CR */
Enter a command, type H for Help --> R
Passed Diagnostic Self Tests
```

5.3 Terminal Mode

The Terminal mode buffers characters received from the terminal and sends them in a frame to the cable. When a frame is received from the cable, data is extracted and sent to the terminal. One of three events initiate the UAP to send a frame providing there is data to send: buffering more than 1500 bytes, receiving a Carriage Return from the terminal, or receiving an interrupt from the virtual terminal timer.

The virtual terminal timer employs timer 1 in the 80130 to cause an interrupt every .125 seconds. Each time the interrupt occurs the software checks to see if it received one or more characters from the terminal. If it did, then it sends the characters in a frame.

The interface to the async terminal is a 256 byte software FIFO. Since the terminal communication is full duplex, there are two half duplex FIFOs: a Transmit FIFO and a Receive FIFO. Each FIFO uses two functions for I/O: Fifo_In( ) and Fifo_Out( ). A block diagram is displayed in Figure 18.

The serial I/O for the async terminal interface is always polled except in the Terminal mode where it is interrupt driven. The Terminal mode begins by enabling the 8274 receive interrupt but leaves the 8274 transmit interrupt disabled. This way any characters received from the terminal will cause an interrupt. These characters are then placed in the Transmit FIFO. The only time the 8274 transmit interrupt is enabled is when the Receive FIFO has data in it. The receive FIFO is filled from frames being received from the cable. Each time a transmit interrupt occurs a byte is removed from the Receive FIFO and written to the 8274. When the Receive FIFO empties, the 8274 transmit interrupt is disabled.

The flow control implemented for the terminal interface is via RTS and CTS. When the Transmit FIFO is full, RTS goes inactive preventing further reception of characters (see Table 1). If the Receive FIFO is full, receive frames are lost because there is no way for the data link using class 1 service to communicate to the remote station that the buffers are full. Lost receive frames are accounted for by the Out of Resources Frame counter.

The Async Terminal bit rate sets the throughput capability of the station in the terminal mode because the bottle neck for this network is the RS232 interface. Using this fact a simple test was conducted to verify the data link driver's capability of switching between the receiver's No Resource state and the Ready State. For example if station B is sending frames in the High Speed Transmit mode to station A which is in the Terminal mode, frames will be lost in station A. Under these circumstances station A's receiver will be switching from Ready state to Out of Resources state. The sum of Good frames received plus Out of Resource frames from station A should equal Good frames transmitted from station B; unless there were any underruns or overruns.

Table 1. FIFO State Table

Function	Present State	Next State	Action
FIFO_T_IN( )	EMPTY	IN USE	Start Filling Transmit Buffer
	IN USE	FULL	Shut Off RTS
FIFO_T_OUT( )	FULL	IN USE	Enable RTS
	IN USE	EMPTY	Stop Filling Transmit Buffer
FIFO_R_IN( )	EMPTY	IN USE	Turn on TxInt
	IN USE	FULL	Stop Filling FIFO from Receive Buffer
FIFO_R_OUT( )	FULL	IN USE	Start Filling FIFO from Receive Buffer
	IN USE	EMPTY	Turn Off TxInt

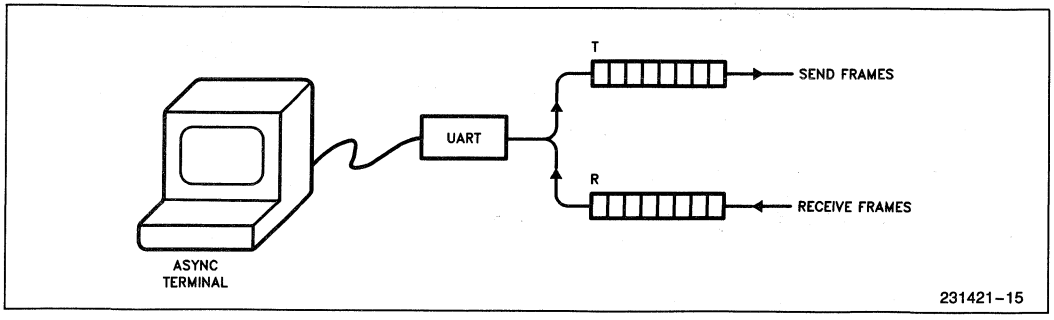


Figure 18

5.3.1 SENDING FRAMES

The Terminal Mode is entered when the Terminal_Mode( ) function is called from the Menu interface. The Terminal_Mode( ) function is one big loop, where each pass sends a frame. Receiving frames in the Terminal Mode is handled on an interrupt driven basis which will be discussed next.

The loop begins by getting a TBD from the 82586 handler. The first three bytes of the first buffer are loaded with the IEEE 802.2 header information. The loop then waits for the Transmit FIFO to become not EMPTY, at which point a byte is removed from the Transmit FIFO and placed in the TBD. After each byte is removed from the Transmit FIFO several conditions are tested to determine whether the frame needs to be transmitted, or whether a new buffer must be obtained. A frame needs to be transmitted if: a Carriage Return is received, the maximum frame length is reached, or the send_frame flag is set by the virtual terminal timer. A new buffer must be obtained if none of the above is true and the max buffer size is reached.

If a frame needs to be sent the last TBD's EOP bit is set and its buffer count is updated. The 82586 Handler's Send_Frame( ) function is called to transmit the frame, and continues to be called until the function returns TRUE.

The loop is repeated until a ^C followed by a Carriage Return is recieved.

5.3.2 RECEIVING FRAMES

Upon initialization the UAP module calls the Add_Dsap_Address(dsap, pfunc) function in the LLC module. This function adds the UAP's LSAP to the active list. The pfunc parameter is the address of the function to call when a frame has been received with the UAP's LSAP address. This function is Recv_Data_1( ). Recv_Data_1( ) looks at the control field of the frame received and determines the action required.

The commands and responses handled by Recv_Data_1( ) are the same as the Station Component's commands and responses given in Figure 15. One difference is that Recv_Data_1( ) will process a UI command while the Station Component will ignore a UI command addressed to it.

Recv_Data_1( ) will discard any UI frames received unless it is in the Terminal Mode. When in the Terminal Mode, Recv_Data_1( ) skips over the IEEE 802.2 header information and uses the length field to determine the number of bytes to place in the Receive FIFO. Before a byte is placed in the FIFO, the FIFO status is checked to make sure it is not full. Recv_Data_1( ) will move all of the data from the frame into the Receive FIFO before returning.

When a frame is received by the 82586 handler an interrupt is generated. While in the 82586 interrupt routine the receive frame is passed to the LLC layer and then to the UAP layer where the data is placed in the Receive FIFO by Recv_Octal_Data_1( ). Since Recv_Data_1( ) will not return until all of the data from the frame has been moved into the Receive FIFO, the 8274 transmit interrupt must be nested at a higher priority than the 82586 interrupt to prevent a software lock. For example if a frame is received which has more than 256 bytes of data, the Receive FIFO will fill up. The only way it can empty is if the 8274 interrupt can nest the 82586 interrupt service routine. If the 8274 could not interrupt the 82586 ISR then the software would be stuck in Recv_Data_1( ) waiting for the FIFO to empty.

5.4 Monitor Mode

The Monitor Mode dynamically updates 6 station related parameters on the terminal as shown below.

The Monitor_Mode( ) function consists of one loop. During each pass through the loop the counters are updated, and a frame is sent. Any size frame can be transmitted up to a size of the maximum number of transmit buffers available. Frame sizes less than the minimum frame length are automatically padded by the 82586 Handler.

The data in the frames transmitted in the Monitor Mode are loaded with all the printable ASCII characters. This way when one station is in the Monitor Mode transmitting to another station in the Terminal Mode, the Terminal Mode station will display a marching pattern of ASCII characters.

# of Good Frames Transmitted	# of Good Frames Received	CRC Errors	Alignment Errors	No Resource Errors	Receive Overrun Errors
32	0	00000	00000	00000	00000

5.5 High Speed Transmit Mode

The High Speed Transmit Mode demonstrates the throughput performance of the 82586 Handler. The Hs_Xmit_Mode( ) function operates in a tight loop which gets a TBD, sets the EOF bit, and calls Send_Frame( ). The flow chart for this loop is shown in Figure 19.

The loop is exited when a character is received from the terminal. Rather than polling the 8274 for a receive buffer full status, the 8274's receive interrupt is used. When the Hs_Xmit_Mode( ) function is entered, the hs_stat flag is set true. If the 8274 receive interrupt occurs, the hs_stat flag is set false. This way the loop only has to test the hs_stat flag rather than calling inb( ) function each pass through the loop to determine whether a character has been received.

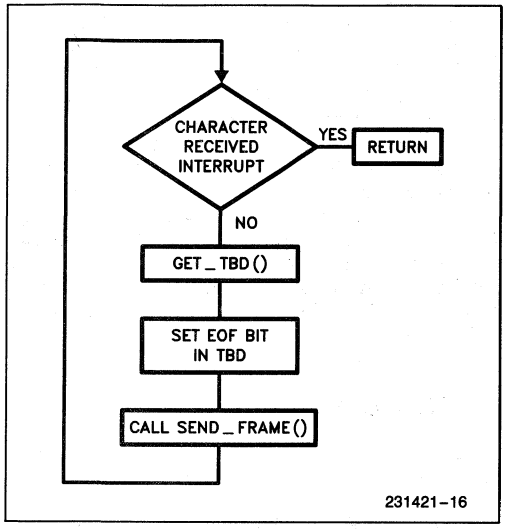


Figure 19. High Speed Transmit Mode Flow Chart

The performance measured on an 8 MHz 186/51 board is 593 frames per second. The bottle neck in the throughput is the software and not the 82586. The size of the buffer is not relevant to the transmit frame rate. Whether the buffer size is 128 bytes or 1500 bytes, linked or not, the frame rate is still the same. Therefore assuming a 1500 byte buffer at 593 frames per second, the effective data rate is 889,500 bytes per second.

This can easily be demonstrated by using two 186/51 boards running the Data Link software. The receiving stations counters should be cleared then placed in the Monitor mode. When placing it in the monitor mode, transmission should not be enabled. When the other station is placed in the High Speed Transmit Mode a timer should be started. One can use a stop watch to determine the time interval for transmission. The frame rate is determined by dividing the number of frames received in the Monitor station by the time interval of transmission.

APPENDIX A
COMPILING, LINKING, LOCATING, AND RUNNING THE SOFTWARE ON THE 186/51 BOARD

********* Instructions for using the 186/51 board *********

Use 27128A for no wait state operation. 27128s can be used but wait states will have to be added.

Copy HI.BYT and LO.BYT files into EPROMs
PROMs go into U34 - HI.BYT and U39 - LO.BYT on the 186/51 board

JUMPERS REQUIRED

Jumper the 186/51 board for 16K byte PROMs in U34 and U39 Table 2-5 in 186/51 HARDWARE REFERENCE MANUAL (Rev-001)

186/51(ES)	186/51 (S)/186/51
E151-E152 OUT	E199-E203 OUT
E152-E150 IN	E203-E191 IN
E94-E95 IN	E120-E119 IN
E100-E106 IN	E116-E112 IN
E107-E113 IN	E111-E107 IN
E133-E134 IN	E94-E93 IN

also change interrupt priority jumpers - switch 8274 and 82586 interrupt priorities

E36-E44 OUT	E43-E47 OUT
E39-E47 OUT	E46-ES0 OUT
E37-E45 OUT	E44-E48 OUT

WIRE WRAP

E36-E47 IN	E43-E50 IN
E39-E44 IN	E46-E47 IN
E79-E45 IN	E90-E48 IN

USE SDM MONITOR

The SDM Monitor should have the 82586's SCP burned into ROM. The ISCP is located at OFFFOH. Therefore for the SCP the value in the SDM ROM should be:

ADDRESS	DATA
FFFF6H	XXOOH
FFFF8H	XXXXH
FFFFAH	XXXXH
FFFFCH	FFFOH
FFFFEH	XXOOH

To run the program begin execution at 0D000:6H

```
I.E. G D000:6

GOOD LUCK!

**********    submit file for compiling one module:    **********

run

cc86.86  :F6:%0 LARGE ROM DEBUG DEFINE(DEBUG) include(:F6:)

exit

**********    submit file for linking and locating:    **********

run

link86     :F6:assy.obj, :F6:dld.obj, :F6:llc.obj, &

:F6:uap.obj, lclib.lib to :F6:dld.lnk segsize(stack(4000h)) notype

loc86 :F6:dld.lnk to :F6:dld.loc&

initcode (0D0000H) start(begin) order(classes(data, stack, code)) &
addresses(classes(data(3000H), stack(0CB00H), code(0D0020H)))

oh86 :F6:dld.loc to :F6:dld.rom

exit

**********    submit file for burning EPROMs using IPPS:    **********

ipps

i 86

f :F6:dld.rom (0d0000h)

3

2

1

0 to :F6:lo.byt

y

1 to :F6:hi.byt

y


t 27128

9

c :F6:lo.byt t p

n

C :F6:hi.byt t p

n

exit
```

/PCD/USR/CHUCK/CSRC/DLD.H

```
/*************************************************************************
*                                                                        *
*                        82586 Structures and Constants                  *
*                                                                        *
*************************************************************************/

/* general purpose constants */

#define INUSE    0
#define EMPTY    1
#define FULL     2
#define FREE     1
#define TRUE     1
#define FALSE    0
#define NULL     0xFFFF

/* Define Data Structures */

#define RBUF_SIZE        128 /* receive buffer size */
#define TBUF_SIZE        128 /* transmit buffer size */
#define ADD_LEN          6
#define MULTI_ADDR_CNT   16

typedef unsigned short int u_short;

/* results from Test_Link(): loaded into Self_Test char */

#define PASSED                  0
#define FAILED_DIAGNOSE         1
#define FAILED_LPBK_INTERNAL    2
#define FAILED_LPBK_EXTERNAL    3
#define FAILED_LPBK_TRANSCEIVER 4

/* Frame Commands */
#define     UI       0x03    /* Unnumbered Information Frame */
#define     XID      0xAF    /* Exchange Identification */
#define     TEST     0xE3    /* Remote Loopback Test */
#define     P_F_BIT 0x10     /* Poll/Final Bit Position */
#define     C_R_BIT 0x01     /* Command/Response bit in SSAP */


#define     DSAP_CNT     8   /* Number of allowable DSAPs; must be a multiple
                                of 2**N, and DSAP addresses assigned must be
                                divisible by 2**(8-N).
                                (i.e. the N LSBs must be 0) */

#define     DSAP_SHIFT   5   /* DSAP_SHIFTS must equal 8-N */


#define     XID_LENGTH   6   /* Number of Info bytes for XID Response frame */


/* System Configuration Pointer SCP */

   struct SCP  {
               u_short sysbus; /* 82586 bus width, 0 - 16 bits
                                  1 - 8 bits */
```

231421-17

```
/PCO/USR/CHUCK/CSRC/DLD.H

                u_short junk[2];
                u_short iscpl;  /* lower 16 bits of iscp address */
                u_short iscph;  /* upper 8 bits of iscp address */
               };

/* Intermediate System Configuration Pointer ISCP */

   struct ISCP {
                u_short busy ;  /*set to 1 by cpu before its first CA,
                                 cleared by 82586 after reading */
                u_short offset  ;   /* offset of system control block */
                u_short base1 ; /* base of system control block */
                u_short base2 ;
               } ;

/* System Control Block SCB */

   struct SCB  {
                u_short stat;       /* Status word */
                u_short cmd;        /* Command word */
                u_short cbl_offset; /* Offset of first command block in CBL */
                u_short rfa_offset; /* Offset of first frame descriptor in RFA */
                u_short crc_errs;   /* CRC errors accumulated */
                u_short aln_errs;   /* Alignment errors */
                u_short rsc_errs;   /* Frames lost because of no Resources */
                u_short ovr_errs;   /* Overrun errors */
               };

/*  Command Block */

   struct  CB  {
                u_short stat;   /* Status of Command */
                u_short cmd;    /* Command */
                u_short link;   /* link field */
                u_short parm1;  /* Parameters */
                u_short parm2;
                u_short parm3;
                u_short parm4;
                u_short parm5;
                u_short parm6;
          };


/*  Multicast Address Command Block MA_CB */

   struct MA_CB{
                u_short stat;   /* Status of Command */
                u_short cmd;    /* Command */
                u_short link;   /* Link field */
                u_short mc_cnt; /* Number of MC addresses */
                char mc_addr[ADD_LEN*MULTI_ADDR_CNT]; /* MC address area */
               };


/* Transmit Buffer Descriptor TBD */

   struct TBD  {
```

231421-18

/PCO/USR/CHUCK/CSRC/DLD.H

```
                u_short act_cnt;     /* Number of bytes in buffer */
                u_short link;        /* offset to next TBD */
                u_short buff_l;      /* lower 16 bits of buffer address */
                u_short buff_h;      /* upper 8 bits of buffer address */
                struct TB *buff_ptr;     /* not used by the 586: used by the
                                         software to save address translation
                                         routine.  */
               };

/* Transmit Buffers */
    struct TB   {
                 char data [TBUF_SIZE];
                };

/* Frame Descriptor FD */

   struct  FD  {
                u_short stat;                /* Status Word of FD */
                u_short el_s;                /* EL and S bits */
                u_short link;                /* link to next FD */
                u_short rbd_offset;          /* Receive buffer descriptor offset */
                char dest_addr[ADD_LEN];/*Destination address */
                char src_addr[ADD_LEN]; /* Source address */
                u_short length;              /* Length field */
               };

/* Receive Buffer Descriptor RBD */

   struct RBD {
                u_short act_cnt;         /* Actual number of bytes received */
                u_short link;            /* Offset to next RBD */
                u_short buff_l;          /* Lower 16 bits of buffer address */
                u_short buff_h;          /* upper 8 bits of buffer address */
                u_short size;            /* size of buffer */
                struct RB *buff_ptr;     /* not used by the 586: used by the
                                         software to save address translation
                                         routine. */
               };


/* Receive Buffers */

   struct RB   {
                char data[RBUF_SIZE];
               };


struct  FRAME_STRUCT
        {
            unsigned char   dsap;    /* Destination Service Access Point */
            unsigned char   ssap;    /* Source Service Access Point */
            unsigned char   cmd;     /* ISO Data Link Command */
        };

    /* LSAP Address Table */
struct  LAT {
                char    stat;            /* INUSE or FREE */
```

231421-19

```
/PCD/USR/CHUCK/CSRC/DLD.H

                int     (*p_sap_func)();/* Pointer to LSAP function; associated
                                           with dsap address */
            };


struct MAT {                        /* Multicast Address Table */
        char    stat;               /* INUSE or FREE */
        char    addr[ADD_LEN];      /* actual mc address */
        };

/* general purpose flags */

struct FLAGS {
    unsigned diag_done : 1 ;        /* diagnose command complete */
    unsigned stat_on : 1 ;          /* network diagnostic statistics on/off */
    unsigned reset_sema: 1 ;        /* don't reset when this bit is set */
    unsigned reset_pend: 1 ;        /* reset when this bit is set */
    unsigned lpbk_test: 1 ;         /* loopback test flag */
    unsigned lpbk_mode: 1 ;         /* loopback mode on/off */
       } ;


/* General purpose bits */

#define ELBIT   0x8000
#define EOFBIT  0x8000
#define SBIT    0x4000
#define IBIT    0x2000
#define CBIT    0x8000
#define BBIT    0x4000
#define OKBIT   0x2000

/* SCB patterns */

#define CX          0x8000
#define FR          0x4000
#define CNA         0x2000
#define RNR         0x1000
#define RESET       0x0080
#define CU_START    0x0100
#define RU_START    0x0010
#define RU_ABORT    0x0040
#define CU_MASK     0x0700
#define RU_MASK     0x0070
#define RU_READY    0x0040

/* 82586 Commands */

#define NOP         0x0000
#define IA          0x0001
#define CONFIGURE   0x0002
#define MC_SETUP    0x0003
#define TRANSMIT    0x0004
#define TDR         0x0005
#define DUMP        0x0006
#define DIAGNOSE    0x0007
```

231421-20

```
/PCO/USR/CHUCK/CSRC/DLD.H

/* 82586 Command and Status Masks */

#define CMD_MASK          0x0007
#define NOERRBIT          0x2000
#define COLLMASK          0x000F
#define DEFERMASK         0x0080
#define NOCRSMASK         0x0400
#define UNDERRUNMASK      0x0100
#define SQEMASK           0x0040
#define MAXCOLMASK        0x0020
#define OUT_OF_RESOURCES 0x0200

/* Configure Parameters */

#define FIFO_LIM     0x0800       /* use FIFO lim of 8 */
#define BYTE_CNT     0x000B
#define SRDY         0x0040
#define SAV_BF       0x0080
#define ADDR_LEN     0x0600       /* address length of 6 bytes */
#define AC_LOC       0x0800
#define PREAM_LEN    0x2000       /* preamble length of 8 bytes */
#define INT_LPBCK    0x4000
#define EXT_LPBCK    0x8000
#define LIN_PRIO     0x0000       /* no priority */
#define ACR          0x0000
#define BOF_MET      0x0080
#define IFS          0x6000       /* IFS time 9.6 usec */
#define SLOT_TIME    0x0200       /* slot time 51.2 usec */
#define RETRY_NUM    0xF000       /* retry number 15 */
#define PRM          0x0001
#define BC_DIS       0x0002
#define MANCHESTER   0x0004
#define TONO_CRS     0x0008
#define NCRC_INS     0x0010
#define CRC_16       0x0020
#define BT_STUFF     0x0040
#define PAD          0x0080
#define CRSF         0x0000       /* no carrier sense filter */
#define CRS_SRC      0x0800
#define CDTF         0x0000       /* no collision detect filter */
#define CDT_SRC      0x8000
#define MIN_FRM_LEN 0x0040       /* 64 bytes */
#define MIN_DATA_LEN MIN_FRM_LEN - 18   /* assumes Ethernet/IEEE 802.3
                                           frames with 6 bytes of address */
#define MAX_FRAME_SIZE  1500 - 3
```

231421-21

```
/PCO/USR/CHUCK/CSRC/DLD.C

/*************************************************************************
*                                                                        *
*                          82586 Handler                                 *
*                                                                        *
*************************************************************************/

/* Define constants for storage area */

#define CB_CNT      8   /* Number of available Command Blocks */
#define FD_CNT      16  /* Number of available Frame Descriptors */
#define RBD_CNT     64  /* Number of available Receive Buffer descriptors */
#define TBD_CNT     16  /* Number of available Transmit Buffer descriptors */

/* loopback parameters passed to Configure() */

#define INTERNAL_LOOPBACK   0x4000
#define EXTERNAL_LOOPBACK   0x8000
#define NO_LOOPBACK         0x0000

#include "dld.h"            /* 586 Data Structures */


/* 186 Timer Addresses */

#define TIMER1_CTL  0xFF5E
#define TIMER1_CNT  0xFF58
#define TIMER2_CTL  0xFF66
#define TIMER2_CNT  0xFF60


/* external functions */

/* I/O */
int     inw();      /* input word : inw(address) */
void    outw();     /* output word: outw(address, value) */
void    init_intv();/* initialize the interrupt vector table */
void    enable();   /* enable 80186 interrupts */
void    disable();  /* disable 80186 interrupts */

extern  char    *Build_Ptr();

u_short     SEGMT;      /* Data segment value */
char        *pNULL;     /* NULL pointer */

/* Macro 'type' of definitions */

#define CA  outw(0xCB,0)    /* the command to issue a Channel Attention */

#define ESI_LOOPBACK outw(0xCB,0) /* put the ESI in Loopback */
#define NO_ESI_LOOPBACK outw(0xCB,8) /* take the ESI out of Loopback */

#define EOI_80130           outb(0xE0,0x63)   /* End Of Interrupt */
#define TIMER1_EOI_80186 outw(0xFF22,0x04) /* EOI for Timer 1 on the 186 */
#define TIMER1_EOI_80130 outb(0xE0,0x64) /*EOI for 186's Timer1 on the 130 */
```

231421-22

```
/PCO/USR/CHUCK/CSRC/DLD.C


/******** memory  allocation ****************/

int Self_Test;       /* used for diagnostic purposes */
u_short temp;        /* temporary storage */

#define LPBK_FRAME_SIZE     4                /* loopback frame storage */
char    lpbk_frame[LPBK_FRAME_SIZE] = {
0x55, 0xAA, 0x55, 0xAA};

#define whoami_io_add   0x00F0  /* I/O address of Host Address Prom */
char    whoami[ADD_LEN];        /* Ram array where host address is stored */


/* transmission statistic variables */

unsigned long   good_xmit_cnt;
u_short         underrun_cnt;
u_short         no_crs_cnt;
unsigned long   defer_cnt;
u_short         sqe_err_cnt;
u_short         max_col_cnt;
unsigned long   recv_frame_cnt;
u_short         reset_cnt;

    /* Allocate storage for structures and buffers */


struct FLAGS flags;


/* 586 structures */

/* System Configuration Pointer: Rom Initialization */
/* struct SCP scp = {0x0000,0x0000,0x0000,0x1FF6,0x0000};   */

/* struct ISCP iscp;     Intermediate System Configuration Pointer */

struct SCB scb;      /* System Control Block */

struct CB cb[CB_CNT],    /* Command Blocks */
*cb_tos, *begin_cbl, *end_cbl;
                            /* pointer to the beginning of the free
                               command block list (cb_tos) and the
                               beginning and end of the 82586 cbl */

struct TBD tbd[TBD_CNT],    /* Transmit Buffer Descriptor */
*tbd_tos;                 /* pointer to the free Transmit buffer
                               descriptors */

struct TB tbuf[TBD_CNT];     /* Transmit Buffers */


struct FD fd[FD_CNT],        /* Frame Descriptors */
*begin_fd, *end_fd;       /* pointers to the beginning and end of
                                the free FD list */

struct RBD rbd[RBD_CNT],     /* Receive Buffer Descriptors */
```

231421-23

```
/PCO/USR/CHUCK/CSRC/DLD.C

*begin_rbd, *end_rbd;          /* pointers to the beginning and the
                                        end of the rbd list */

struct RB rbuf[RBD_CNT];     /* Receive Buffers */

struct MAT mat[MULTI_ADDR_CNT];     /* Multicast Address Table */
struct MA_CB ma_cb;                 /* Multicast Address Command Block */


/* The following structures are used only in Reset_586() function */
struct      CB      res_cb;     /* Temporary CB for reinitializing the 586 */
struct      MA_CB   res_ma_cb;  /* Temporary MA_CB for reloading Multicast */

/*  Hardware Support Functions */

Enable_586_Int()
{
    int     c;

    c = inb(0xE2);          /* read the 80130 interrupt mask register */
    outb(0xE2, 0x00F7 & c); /* write to the 80130 interrupt mask register */
}

Disable_586_Int()
{
    int     c;

    c = inb(0xE2);
    outb(0xE2, 0x0008 | c);
}

Set_Timeout()
{
    outw(TIMER1_CNT, 0);    /* Write a 0 to Timer1 count register */
    outw(0xFF5E, 0xE009);   /* Set ENable bit in Timer1 Mode/Control register */
}

Reset_Timeout()
{
    outw(0xFF5E, 0x6009); /* Reset ENable bit in Timer1 Mode/Control register */
}

Init_Timer() /* 186's Timer 2 is a prescaler for Timer 1.  It clocks Timer 1
                every 32.7 msec.  The deadman timeout is set for 1.25 sec */
{
    outw(0xFF38, 0x000C);    /* Set Timer1 Interrupt Control register */
    outw(0xFF62, 0xFFFF);    /* set max count register for timer2 to OFFFFH */
    outw(0xFF5A,     3B);    /* set max count register A for timer 1 */
    outw(0xFF66, 0xC001);    /* Set Timer2 Mode/Control register */
    outw(0xFF5E, 0x6009);    /* Set Timer1 Mode/Control register */
    outw(0xFF28, (inw(0xFF28) & 0xFFEF)); /* Enable 186 Timer1 interrupt */
    outb(0xE2,(inb(0xE2) & 0x00EF));  /* enable 80130 interrupt from 80186 */
}

/* end hardware support functions */

Clear_Cnt()
```

231421-24

```
/PCO/USR/CHUCK/CSRC/DLD.C


{
    scb.crc_errs = 0;        /* clear 586 error statistic counters */
    scb.aln_errs = 0;
    scb.rsc_errs = 0;
    scb.ovr_errs = 0;

    good_xmit_cnt = 0;       /* init data link statistics */
    underrun_cnt = 0;
    no_crs_cnt = 0;
    defer_cnt = 0;
    sqe_err_cnt = 0;
    max_col_cnt = 0;
    recv_frame_cnt = 0;
    reset_cnt = 0;
}

Init_586()
{
    struct ISCP *piscp;
    u_short i;
    struct  MAT *pmat;

    NO_ESI_LOOPBACK;  /* Done for 82501. Inactivates CRS if powered up
                         in loopback */
    ESI_LOOPBACK;

    init_intv();     /* Initialization DLDs interrupt vectors */

    Init_Timer();

    flags.reset_sema = 0;    /* Initialize Reset Flags */
    flags.reset_pend = 0;
    flags.stat_on = 1;

    Disable_586_Int();


    piscp =  0x0000FFF0 ; /* Initialize the ISCP pointer*/
    piscp->busy = 1;
    piscp->offset =  Offset(&scb);
    piscp->base1 = SEGMT << 4;
    piscp->base2 = (SEGMT >> 12) & 0x000F ;

    pNULL = Build_Ptr(NULL);     /* build a NULL pointer - 8086 type: 32 bits */
    Build_Rfa();     /* init Receive Frame Area */
    Build_Cb(); /* init Command Block list */
    ma_cb.cmd = 0;       /* multicast address semaphore init */

    Clear_Cnt();

    scb.stat = 0;

    CA;      /* wait for the 586 to complete initialization */


    for ( i = 0; i <= 0xFF00; i++)
```

231421-25

/PCO/USR/CHUCK/CSRC/DLD.C

```
        if (scb.stat == (CX | CNA))
            break;


    if (i >0xFF00)
    Fatal("DLD:init - Did not get an interrupt after Reset/CA\n");

    /* Ack the reset Interrupt */
    scb.cmd = (CX | CNA);
    CA;
    Wait_Scb();
    Enable_586_Int();

    scb.cbl_offset = Offset(&cb[0]);     /* link scb to cb and fd lists */
    scb.rfa_offset = Offset(&fd[0]);

     /* move the prom bytes into whoami array */

    for (i = 0; i < ADD_LEN; i++)
        whoami[(ADD_LEN - 1) - i] = inb(whoami_io_add + i*2);

        /* Initialization the Multicast Address Table */

    for (pmat = &mat[0]; pmat <= &mat[MULTI_ADDR_CNT - 1]; pmat++)
        pmat->stat = FREE;

    Configure(INTERNAL_LOOPBACK);   /* Put 586 in internal loopback */

    SetAddress();   /* Set up the station address */

    /* run diagnostics */

    Test_Link();

    if (Self_Test != PASSED)
        return(Self_Test);

    Configure(NO_LOOPBACK); /* Configure the 82586 */

    return(Self_Test);
}

Build_Rfa()
{
    struct   FD     *pfd;
    struct   RBD    *prbd;
    struct   RB     *pbuf;
    unsigned long   badd;

    /* Build a linear linked frame descriptor list */

    for (pfd = &fd[0]; pfd <= &fd[FD_CNT - 1]; pfd++) {

        pfd->stat = pfd->el_s = 0;
        pfd->link = Offset(pfd+1);
        pfd->rbd_offset = NULL;
    }
```

231421-26

/PCO/USR/CHUCK/CSRC/DLD.C

```
    end_fd = --pfd;                 /* point to &fd[FD_CNT - 1] */
    pfd->link = NULL;               /* last fd link is NULL */
    pfd->el_s = ELBIT;              /* last fd has EL bit set */
    begin_fd = pfd = &fd[0];        /* point to first fd */
    pfd->rbd_offset = Offset(&rbd[0]);  /* link first fd to first rbd */


    /* Build a linear linked receive buffer descriptor list */

    for (prbd = &rbd[0], pbuf = &rbuf[0]; prbd <= &rbd[RBD_CNT - 1];
                                                        prbd++, pbuf++) {
        badd = SEGMT << 4;
        badd += Offset(pbuf);
        prbd->buff_l = badd;
        prbd->buff_h = badd >> 16;
        prbd->buff_ptr = pbuf;

        prbd->act_cnt =  0;
        prbd->link = Offset(prbd + 1);
        prbd->size = RBUF_SIZE;
    }

    end_rbd = --prbd;
    prbd->link = NULL;          /* last rbd points to NULL */
    prbd->size |= ELBIT;        /* last rbd has el bit set */

    begin_rbd = &rbd[0];

}

Build_Cb()  /* Build a stack of free command blocks */
{
    struct  CB  *pcb;
    struct  TBD *ptbd;
    struct  TB  *pbuf;
    unsigned    long    badd;

    for (pcb = &cb[0]; pcb <= &cb[CB_CNT - 1]; pcb++) {
        pcb->stat = 0;
        pcb->cmd = ELBIT;
        pcb->link = Offset(pcb + 1);
    }
    --pcb;
    begin_cbl = end_cbl = pNULL;
    pcb->link = NULL;
    cb_tos = &cb[0];

    /* Build a stack of transmit buffer descriptors */

    for (ptbd = &tbd[0], pbuf = &tbuf[0]; ptbd <= &tbd[TBD_CNT - 1];
                                                        ptbd++, pbuf++) {

        ptbd->act_cnt  = TBUF_SIZE;
        ptbd->link = Offset(ptbd + 1);

        badd = SEGMT << 4;
```

231421-27

/PCO/USR/CHUCK/CSRC/DLD.C

```
        badd += Offset(pbuf);
        ptbd->buff_l = badd;
        ptbd->buff_h = badd >> 16;
        ptbd->buff_ptr = pbuf;
    }

    --ptbd;
    ptbd->link = NULL;   /* last tbd link is NULL */
    tbd_tos = &tbd[0];   /* Set the Top Of the Stack */

}

    /* Get a Command Block from the free list */

struct  CB   *Get_Cb() /* return a pointer to a free command block */
{
    struct  CB *pcb;

    if (Offset(pcb = cb_tos) == NULL)
        return(pNULL);
    cb_tos = (struct CB *) Build_Ptr(pcb->link);
    pcb->link = NULL;
    return(pcb);
}

/* Put a Command Block back onto the free list */

Put_Cb(pcb)

    struct  CB *pcb;

{
    pcb->stat = 0;
    pcb->link = Offset(cb_tos);
    cb_tos = pcb;
}

struct TBD  *Get_Tbd()  /* return a pointer to a free transmit buffer
                           descriptor */
{
    struct   TBD    *ptbd;

    flags.reset_sema = 1;
    Disable_586_Int();
    if ((ptbd = tbd_tos) != pNULL) {
        tbd_tos = (struct TBD *) Build_Ptr(ptbd->link);
        ptbd->link = NULL;
    }
    Enable_586_Int();
    flags.reset_sema = 0;
    if (flags.reset_pend == 1)
        Reset_586();
    return(ptbd);
}


Put_Tbd(ptbd)
```

231421-28

```
/PCO/USR/CHUCK/CSRC/DLD.C


struct      TBD     *ptbd;
{
    struct      TBD     *p ;

    /* find the end of the tbd list returned, ptbd is the beginning */

    for (p = ptbd; p->link != NULL; p = (struct TBD *) Build_Ptr(p->link)) ;

    p->act_cnt = TBUF_SIZE;     /* clear EOFBIT and update size on last tbd */
    p->link = Offset(tbd_tos);
    tbd_tos = ptbd;
}


SetAddress()
{
    struct  CB  *pcb;

#ifdef DEBUG

    if ((pcb = Get_Cb()) == pNULL)
    Fatal("dld.c - SetAddress - couldn't get a CB\n");

#else

    pcb = Get_Cb();

#endif  /* DEBUG */

    bcopy((char *)&pcb->parm1, &whoami[0], ADD_LEN); /* move the prom
                                                        address to IA cmd */
    pcb->cmd = IA | ELBIT;

    Issue_CU_Cmd(pcb);
}


Wait_Scb()  /* wait for the scb command word to be clear */
{
    u_short     i, stat;

    for (stat = FALSE; stat == FALSE; ) {

        for (i=0; i<=0xFF00; i++)
            if (scb.cmd == 0)
                break;

        if (i > 0xFF00) {
            Bug("DLD: Scb command not clear\n");
            CA;
        }
        else
            stat = TRUE;
    }
```

231421-29

```
/PCO/USR/CHUCK/CSRC/DLD.C

}


Issue_CU_Cmd(pcb) /* Queue up a command and issue a start CU command if no
                     other commands are queued */
    struct  CB  *pcb;
{
    Disable_586_Int();
    if (begin_cbl == pNULL) {   /* if the list is inactive start CU */
        begin_cbl = end_cbl = pcb;
        scb.cbl_offset = Offset(pcb);
        Wait_Scb();
        scb.cmd = CU_START;
        Set_Timeout();              /* set deadman timer for CU */
        CA;
    }
    else {
        end_cbl->link = Offset(pcb);
        end_cbl = pcb;
    }
    Enable_586_Int();
}

Isr7()
{
    outb(0xE0, 0x67);    /* EOI 80130 */
}

Isr6()
{
    Write("\nInterrupt 6\n");
    outb(0xE0, 0x66);    /* EOI 80130 */
}

Isr5()
{
    Write("\nInterrupt 5\n");
    outb(0xE0, 0x65);    /* EOI 80130 */
}

    /* Deadman Timer Interrupt Service Routine */

Isr_Timeout()   /* Interrupt 4 */
{
    Reset_Timeout();
    if (flags.reset_sema == 1)
        flags.reset_pend = 1;
    else
        Reset_586();

    TIMER1_EOI_80186;
    TIMER1_EOI_80130;
}

    /* Interrupt 0 is Uart in UAP Module */
    /* Interrupt 2 is Timer in UAP Module */
```

231421-30

```
/PCO/USR/CHUCK/CSRC/DLD.C


Isr1()
{
    Write("\nInterrupt 1\n");
    outb(0xE0, 0x61);    /* EOI 80130 */
}

    /* 586 Interrupt service routine: Interrupt 3 */

Isr_586()
{

    u_short     stat_scb;
    struct  CB      *pcb;

    enable();    /* nesting only the uart interrupt */

    Wait_Scb();
    scb.cmd = (stat_scb = scb.stat) & (CX | CNA | FR | RNR);
    CA;

    if (stat_scb & (FR | RNR))
        Recv_Int_Processing();

    if (stat_scb & CNA) {  /* end of cb processing */

        Reset_Timeout();     /* clear deadman timer  */
        pcb = Build_Ptr(scb.cbl_offset);

#ifdef DEBUG

        if (begin_cbl == pNULL){
            Bug("DLD: begin_cbl == NULL in interrupt routine\n");
            return;
        }

        if ((pcb->stat & 0xC000) != 0x8000)
            Fatal("DLD: C bit not set or B bit set in interrupt routine\n");

#endif /* DEBUG */

        switch (pcb->cmd & CMD_MASK) {

        case TRANSMIT:

            if (flags.stat_on == 1) {/* if Transmit Statistics are collected do */

                /* if sqe bit = 0 and there were no collisions -> sqe error
                   this condition will occur on the first transmission if
                   there were no collisions, or if the previous transmit
                   command reached the max collision count, and the current
                   transmission had no collisions */

                if ((pcb->stat & (SQEMASK | MAXCOLMASK | COLLMASK)) == 0)
                    ++sqe_err_cnt;

                if (pcb->stat & DEFERMASK)
                    ++defer_cnt;
```

231421-31

/PCD/USR/CHUCK/CSRC/DLD.C

```
                if (pcb->stat & NOERRBIT)
                    ++good_xmit_cnt;
                else {

                    if (pcb->stat & NOCRSMASK)
                        ++no_crs_cnt;
                    if (pcb->stat & UNDERRUNMASK)
                        ++underrun_cnt;
                    if (pcb->stat & MAXCOLMASK)
                        ++max_col_cnt;
                }
            }
            if (pcb->parm1 != NULL)
                Put_Tbd(Build_Ptr(pcb->parm1));
            break;

        case DIAGNOSE:

            flags.diag_done = 1;
            if ((pcb->stat & NOERRBIT) == 0)
                Self_Test = FAILED_DIAGNOSE;
            break;


        default:
                ;
        }


        /* check to see if another command is queued */

        if (pcb->link == NULL)
            begin_cbl = pNULL;

        else { /* restart the CU and execute the next command on the cbl */

            begin_cbl = Build_Ptr(pcb->link);
            scb.cbl_offset = pcb->link;
            Wait_Scb();
            scb.cmd = CU_START;
            CA;
            Wait_Scb();
            Set_Timeout();      /* START deadman timer */
        }
        if ((pcb->cmd & CMD_MASK) == MC_SETUP)
            pcb->cmd = 0; /* clear MC_SETUP cmd word, this will implement a
                             lock semephore so that it won't be reused until
                             it is completed */
        else
            Put_Cb(pcb); /* Don't return MC_SETUP cmd block.  It's not a
                            general purpose command block from free CB list */
    }
    disable();  /* disable cpu int so that the 586 isr will not nest */
    EOI_80130;
}
```

231421-32

/PCD/USR/CHUCK/CSRC/DLD.C

```
Recv_Int_Processing()
{
    struct      FD      *pfd;   /* points to the Frame Descriptor */
    struct      RBD     *q,     /* points to the last rbd for the frame */
                        *prbd;  /* points to the first rbd for the frame */

    for (pfd = begin_fd; pfd != pNULL; pfd = begin_fd)
        if (pfd->stat & CBIT) {
            begin_fd = (struct FD *) Build_Ptr(pfd->link);
            prbd = (struct RBD *) Build_Ptr(pfd->rbd_offset);
            if (prbd != pNULL) { /* check to see if a buffer is attached */

#ifdef DEBUG
                if (prbd != begin_rbd)
                    Fatal("DLD: prbd != begin_rbd in Recv_Int_Processing\n");
#endif /* DEBUG */
                for (q = prbd; (q->act_cnt & EOFBIT) != EOFBIT;
                                        q = (struct RBD *) Build_Ptr(q->link));

                begin_rbd = (struct RBD *) Build_Ptr(q->link);
                q->link = NULL;
            }
            if (pfd->stat & OUT_OF_RESOURCES)
                Put_Free_RFA(pfd);
            else {
                /* if the DLD is in a loopback test, check the frame recv */
                if (flags.lpbk_mode == 1)
                    Loopback_Check(pfd);
                else

            /* if it's a multicast address check to see if it's
                 in the multicast address table, if not discard the frame */

              if ( ((pfd->dest_addr[0] & 01) == 01) && (!Check_Multicast(pfd)))
                    Put_Free_RFA(pfd);
              else
                {   Recv_Frame(pfd);
                    ++recv_frame_cnt;
                }
            }
        }
        else {
            Ru_Start(); /* If RU has gone into no resources, restart it */
            break;
        }
}


Loopback_Check(pfd)          /* Called by Recv_Int_Processing; checks address
                                    and data of potential loopback frame */
    struct  FD  *pfd;
{
    struct RBD  *prbd;
    struct RB   *pbuf;
```

231421-33

```
/PCO/USR/CHUCK/CSRC/DLD.C

    if ( bcmp((char *) &pfd->src_addr[0], &whoami[0], ADD_LEN) != 0 ) {
        Put_Free_RFA(pfd);
        return;
    }
    prbd = (struct RBD *) Build_Ptr(pfd->rbd_offset); /* point to receive
                                                         buffer descriptor */
    pbuf = (struct RB *) prbd->buff_ptr; /* point to receive buffer */

    if ( bcmp((char *) pbuf, &lpbk_frame[0], LPBK_FRAME_SIZE) != 0) {
        Put_Free_RFA(pfd);
        return;
    }

    flags.lpbk_test = 1;  /* passed loopback test */
    Put_Free_RFA(pfd);
}

Check_Multicast(pfd)    /* returns true if multicast address is in MAT */
    struct      FD  *pfd;
{
    struct      MAT *pmat;

    for (pmat = &mat[0]; pmat <= &mat[MULTI_ADDR_CNT - 1]; pmat++)
        if ( pmat->stat == INUSE &&
        (bcmp((char *) &pfd->dest_addr[0], &pmat->addr[0], ADD_LEN) == 0))
        break;

    if (pmat > &mat[MULTI_ADDR_CNT - 1])
        return(FALSE);
    return(TRUE);
}


    /* Test the Link function: executes Diagnose and Loopback tests */

Test_Link()
{
    Self_Test = PASSED;
    Diagnose();
    if (Self_Test == FAILED_DIAGNOSE)
        return;
    Ru_Start();      /* start up the RU for loopback tests */
    flags.lpbk_mode = 1; /* go into loopback mode */

    flags.lpbk_test = 0; /* set looback test to false */
    Send_Lpbk_Frame();   /* internal loopback test */
    if (flags.lpbk_test == 0) {
        Self_Test = FAILED_LPBK_INTERNAL;
        flags.lpbk_mode = 0;
        return;
    }

    flags.lpbk_test = 0;
    Configure(EXTERNAL_LOOPBACK);    /* external loopback test w/ ESI in lpbk */
    Send_Lpbk_Frame();
    if (flags.lpbk_test == 0) {
        Self_Test = FAILED_LPBK_EXTERNAL;
```

231421-34

```
/PCO/USR/CHUCK/CSRC/DLD.C

        flags.lpbk_mode = 0;
        return;
    }

    flags.lpbk_test = 0;     /* external loopback test through transceiver */
    NO_ESI_LOOPBACK;
    Send_Lpbk_Frame();
    if (flags.lpbk_test == 0)
        Self_Test = FAILED_LPBK_TRANSCEIVER;

    flags.lpbk_mode = 0; /* leave loopback mode */
}

Send_Lpbk_Frame()
{
    struct   TBD    *ptbd;
    int             i;

    for (i = 0; i < 8; i++) {   /* send lpbk frame 8 times, since it's
                                   best effort delivery */


#ifdef DEBUG
        if ((ptbd = Get_Tbd()) == pNULL)
            Fatal("dld - Send_Lpbk_Frame - couldn't get a TBD\n");
#else
        ptbd = Get_Tbd();

#endif /* DEBUG */

        ptbd->act_cnt = EOFBIT | LPBK_FRAME_SIZE;
        bcopy((char *) ptbd->buff_ptr, &lpbk_frame[0], LPBK_FRAME_SIZE);

        while(!Send_Frame(ptbd, &whoami[0]));
    }
}


Diagnose()
{
    struct    CB    *pcb;

#ifdef DEBUG
        if ((pcb = Get_Cb()) == pNULL)
        Fatal("dld - Diagnose - couldn't get a CB\n");
#else
        pcb = Get_Cb();

#endif /* DEBUG */

        flags.diag_done = 0;
        Self_Test = FALSE;
        pcb->cmd = DIAGNOSE | ELBIT;

        Issue_CU_Cmd(pcb);

        while (flags.diag_done == 0) ;  /* wait for Diag cmd to finish */
```

231421-35

```
/PCO/USR/CHUCK/CSRC/DLD.C

}

Configure(loopflag)

    u_short loopflag;
{
    struct  CB    *pcb;

#ifdef DEBUG
    if ((pcb = Get_Cb()) == pNULL)
        Fatal("dld - Configure - couldn't get a CB\n");
#else
    pcb = Get_Cb();

#endif /* DEBUG */


    /* Ethernet default parameters */

    pcb->parm1 = 0x080C;
    pcb->parm2 = 0x2600 | loopflag;
    pcb->parm3 = 0x6000;
    pcb->parm4 = 0xF200;
    pcb->parm5 = 0x0000;
    if (loopflag == NO_LOOPBACK)
        pcb->parm6 = 0x0040;
    else
        pcb->parm6 = 0x0006;     /* loopback frame is less bytes than
                                    the minimum frame length */
    pcb->cmd    = CONFIGURE | ELBIT;


    Issue_CU_Cmd(pcb);
}

/* Send a frame to the cable, pass a pointer to the destination address
   and a pointer to the first transmit buffer descriptor. */

Send_Frame(ptbd, padd)  /* returns false if it can't get a Command block */
struct      TBD     *ptbd;
char                *padd;
{
    struct      CB  *pcb;

    u_short         length;

    flags.reset_sema = 1;

    if ((pcb = Get_Cb()) == pNULL) {
        flags.reset_sema = 0;
        if (flags.reset_pend == 1)
            Reset_586();
        return(FALSE);
    }

    pcb->parm1 = Offset(ptbd);
```

231421-36

```
/PCO/USR/CHUCK/CSRC/DLD.C


    /* move destination address to command block */
    bcopy((char *)&pcb->parm2, (char *)padd, ADD_LEN);


    /* calculate the length field by summing up all the buffers */

    for (length = 0; ptbd->link != NULL; ptbd = Build_Ptr(ptbd->link))
        length += ptbd->act_cnt;

    length += (ptbd->act_cnt & 0x3FFF);  /* add the last buffer */

    /* check to see if padding is required, do not do padding on loopback */

    /* this will not work if MIN_DATA_LEN > TBUF_SIZE */

    if ((length < MIN_DATA_LEN) &&  /* assumes a 4 byte CRC */
        (bcmp(&whoami[0], (char *)padd, ADD_LEN) != 0))

            ptbd->act_cnt = MIN_DATA_LEN | EOFBIT;

    pcb->parm5 = length;          /* length field */


    pcb->cmd = TRANSMIT | ELBIT;


    Issue_CU_Cmd(pcb);
    flags.reset_sema = 0;
    if (flags.reset_pend == 1)
        Reset_586();
    return(TRUE);
}


Add_Multicast_Address(pma)  /* pma - pointer to multicast address */
char    *pma;               /* returning false means the Multicast address
                               table is full */
{
    struct      MAT     *pmat;

    flags.reset_sema = 1;

/* if the multicast address is a duplicate of one already in the MAT,
   then return */

    for (pmat = mat; pmat <= &mat[MULTI_ADDR_CNT - 1]; pmat++)
        if ( pmat->stat == INUSE &&
            (bcmp( &pmat->addr[0], (char *) pma, ADD_LEN) == 0)) {
            return(TRUE);
        }


    for (pmat = mat; pmat <= &mat[MULTI_ADDR_CNT - 1]; pmat++)
        if (pmat->stat == FREE) {
            pmat->stat = INUSE;
            bcopy( &pmat->addr[0], (char *) pma, ADD_LEN);
            break;
```

231421-37

/PCO/USR/CHUCK/CSRC/DLD.C

```
        }

    if (pmat > &mat[MULTI_ADDR_CNT - 1]) {
        flags.reset_sema = 0;
        if (flags.reset_pend == 1)
            Reset_586();
        return(FALSE);
    }

    Set_Multicast_Address();
    flags.reset_sema = 0;
    if (flags.reset_pend == 1)
        Reset_586();
    return(TRUE);
}


Delete_Multicast_Address(pma)  /* returning false means the multicast address
                                  was not found */
char    *pma;
{
    struct      MAT     *pmat;

    flags.reset_sema = 1;

    for (pmat = mat; pmat <= &mat[MULTI_ADDR_CNT - 1]; pmat++)
        if ( pmat->stat == INUSE &&
            (bcmp( &pmat->addr[0], (char *) pma, ADD_LEN) == 0)) {
            pmat->stat = FREE;
            break;
        }

    if (pmat > &mat[MULTI_ADDR_CNT - 1]) {
        flags.reset_sema = 0;
        if (flags.reset_pend == 1)
            Reset_586();
        return(FALSE);
    }

    Set_Multicast_Address();
    flags.reset_sema = 0;
    if (flags.reset_pend == 1)
        Reset_586();
    return(TRUE);
}

Set_Multicast_Address()
{
    struct      MAT     *pmat;
    struct      MA_CB   *pma_cb;
    u_short     i;

    i = 0;
    pma_cb = &ma_cb;
    while (pma_cb->cmd != 0) ; /* if the MA_CB is inuse, wait until it's free */
    pma_cb->link = NULL;
```

231421-38

/PCD/USR/CHUCK/CSRC/DLD.C

```
    for (pmat = mat; pmat <= &mat[MULTI_ADDR_CNT - 1]; pmat++)
        if ( pmat->stat == INUSE) {
            bcopy( &pma_cb->mc_addr[i], &pmat->addr[0], ADD_LEN);
            i += ADD_LEN;
        }

    pma_cb->mc_cnt = i;
    pma_cb->cmd = MC_SETUP | ELBIT;

    Issue_CU_Cmd(pma_cb);

}


Put_Free_RFA(pfd)  /* Return Frame Descriptor and Receive Buffer
                      Descriptors to the Free Receive Frame Area */

    struct      FD      *pfd;
{
    struct      RBD     *prbd,  /* points to beginning of returned RBD list */
                        *q;     /* points to end of returned RBD list */
    char        ru_start_flag_fd,  /* indicates whether to restart RU */
                ru_start_flag_rbd;

    flags.reset_sema = 1;
    ru_start_flag_fd = ru_start_flag_rbd = FALSE;
    pfd->el_s = ELBIT;
    pfd->stat = 0;
    prbd = (struct RBD *) Build_Ptr(pfd->rbd_offset);/* pick up the link to the rbd */
    pfd->link = pfd->rbd_offset = NULL;

    /*  Disable_586_Int(); this command is only necessary in a multitasking
    program. However in this single task environment this routine is originally
    called from isr_586(), therefore interrupts are already disabled */


    if (begin_fd == pNULL)
        begin_fd = end_fd = pfd;
    else {
        end_fd->link = Offset(pfd);
        end_fd->el_s = 0;
        end_fd = pfd;
        ru_start_flag_fd = TRUE;
    }

    if (prbd != pNULL) {        /* if there is a rbd attached to the fd then
                                find the beginning and end of the rbd list */

        for (q = prbd; q->link != NULL; q = Build_Ptr(q->link))
            q->act_cnt = 0;

            /* now prbd points to the beginning of the rbd list and
               q points to the end of the list */

        q->size = RBUF_SIZE | ELBIT;
        q->act_cnt = 0;
```

231421-39

/PCO/USR/CHUCK/CSRC/DLD.C

```
        if (begin_rbd == pNULL) { /* if there is nothing on the list
                                     create a new list */

            begin_rbd = prbd;
            end_rbd = q;
            if (prbd != q)
                ru_start_flag_rbd = TRUE; /* if there is more than one rbd
                                             returned start the RU */
        }
        else {
                                  /* if the rbd list already exists add on
                                     the new returned rbds */
            end_rbd->link = Offset(prbd);
            end_rbd->size = RBUF_SIZE;
            end_rbd = q;
            ru_start_flag_rbd = TRUE;
        }
    }
    if (ru_start_flag_fd && ru_start_flag_rbd)
        Ru_Start();

    /*  Enable_586_Int(); if Disable_586_Int() is used above */

    flags.reset_sema = 0;
    if (flags.reset_pend == 1)
        Reset_586();
}

Ru_Start()
{
    if ((scb.stat & RU_MASK) == RU_READY) /* if the RU is already 'ready'
                                             then return */
        return;

    if ((begin_fd->stat & CBIT) == CBIT)
        return;

    begin_fd->rbd_offset = Offset(begin_rbd); /* link the beginning of the rbd
                                                 list to the first fd */
    scb.rfa_offset = Offset(begin_fd);
    Wait_Scb();
    scb.cmd = RU_START;
    CA;
}

Software_Reset()
{
    scb.cmd = RESET;
    CA;
    Wait_Scb();
}

Issue_Reset_Cmds()
{
    Wait_Scb();
    scb.cmd = CU_START;
    CA;
```

231421-40

```
/PCO/USR/CHUCK/CSRC/DLD.C


    Wait_Scb();

    outw(0xFF5E, 0) ;        /* shut off timer 1 interrupt */
    outw(TIMER1_CNT, 0);
    outw(0xFF5E, 0xC009);    /* use timer 1 without interrupt as a deadman */

    while ((inw(0xFF5E) & 0x0020) == 0)  /* if Max Cnt bit is set before CNA
                                            is set, 586 Cmd deadlocked */
        if ((scb.stat & CNA) == CNA)
            break;

    if (scb.stat & CNA != CNA)
        Fatal("DLD:Issue_Reset_Cmds - Command deadlock during reset procedure\n");

    Reset_Timeout();

    scb.cmd = CNA;  /* Acknowledge CNA interrupt */
    CA;
    Wait_Scb();
}


/* Execute a reset, Configure, SetAddress, and MC_Setup, then restart the
    Receive Unit and the Command Unit */
Reset_586()
{
    struct      MAT     *pmat;
    u_short     i;

    ++reset_cnt;
    Disable_586_Int();
    ESI_LOOPBACK;
    Software_Reset();

    scb.stat = 0;

    CA;     /* wait for the 586 to complete initialization */


    for ( i = 0; i <= 0xFF00; i++)
        if (scb.stat == (CX | CNA))
            break;


    if (i >0xFF00)
        Fatal("DLD:init - Did not get an interrupt after Software Reset\n");

    /* Ack the reset Interrupt */
    Wait_Scb();
    scb.cmd = (CX | CNA);
    CA;
    Wait_Scb();

#ifdef  DEBUG
    if ( begin_cbl == pNULL)
        Fatal("DLD: begin_cbl = NULL in Reset_586");
#endif  /* DEBUG */
```

231421-41

/PCO/USR/CHUCK/CSRC/DLD.C

```
    /* Configure the 586 */
    /* Ethernet default parameters; Configure is not necessary when using
       default parameters  */

    res_cb.link = NULL;

    res_cb.parm1 = 0x080C;
    res_cb.parm2 = 0x2600;
    res_cb.parm3 = 0x6000;
    res_cb.parm4 = 0xF200;
    res_cb.parm5 = 0x0000;
    res_cb.parm6 = 0x0040;
    res_cb.cmd   = CONFIGURE | ELBIT;

    scb.cbl_offset = Offset(&res_cb.stat);

    Issue_Reset_Cmds();

    /* Set the Individual Address */

    bcopy((char *) &res_cb.parm1, &whoami[0], ADD_LEN); /* move the prom
                                                           address to IA cmd */
    res_cb.cmd = IA | ELBIT;

    Issue_Reset_Cmds();

    /* reload the multicast addresses */

    i = res_ma_cb.stat = 0;
    res_ma_cb.link = NULL;

    for (pmat = &mat[0]; pmat <= &mat[MULTI_ADDR_CNT - 1]; pmat++)
        if ( pmat->stat == INUSE ) {
            bcopy( &res_ma_cb.mc_addr[i], &pmat->addr[0], ADD_LEN);
            i += ADD_LEN;
        }

    res_ma_cb.mc_cnt = i;
    res_ma_cb.cmd = MC_SETUP | ELBIT;
    scb.cbl_offset = Offset(&res_ma_cb.stat);


    Issue_Reset_Cmds();

    /* Restart the Command Unit and the Receive Unit */

    flags.reset_sema = 0;
    flags.reset_pend = 0;

    NO_ESI_LOOPBACK;

    Recv_Int_Processing();

    scb.cbl_offset = begin_cbl;
    Wait_Scb();
```

231421-42

/PCO/USR/CHUCK/CSRC/DLD.C

```
    scb.cmd = CU_START;
    Set_Timeout();      /* Set Deadman Timer */
    CA;
    Enable_586_Int();
}


/* bcopy -- byte copy routine */
bcopy(dst, src, nbytes)
char    *dst, *src;
int     nbytes;
{
    while (nbytes--) *dst++ = *src++;
}

/* bcmp -- byte compare */
bcmp(s1, s2, nbytes)
char    *s1, *s2;
int     nbytes;
{
    while (nbytes-- && *s1++ == *s2++);
    return(*--s1 - *--s2);
}
```

231421-43

/PCO/USR/CHUCK/CSRC/LLC.C

```
/*****************************************************************************
*                                                                            *
*               IEEE 802.2 Logical Link Control Layer                        *
*                        (Station Component)                                 *
*****************************************************************************/

#include "dld.h"

extern  char    *pNULL;

extern  struct  TBD *Get_Tbd();
extern  char        *Build_Ptr();

readonly char   xid_frame[XID_LENGTH] = { 0, 0, XID, 0x81, 0x01, 0};
/* DSAP, SSAP, XID, xid class 1 response */

struct LAT lat[DSAP_CNT];

Init_Llc()
{
    struct  LAT     *plat;

    for (plat = &lat[0]; plat <= &lat[DSAP_CNT - 1]; plat++)
        plat->stat = FREE;
    return(Init_586());
}

/* Function for adding a new DSAP */

Add_Dsap_Address(dsap, pfunc) /* DSAP must be divisible by 2**(8-N), where
                                 2**N = DSAP_CNT. (i.e. N LSBs must be 0).
                                 The function will return FALSE if does not
                                 meet the above requirements, or the Lsap
                                 Address Table is full, or the address has
                                 already been used. NULL DSAP address is
                                 reserved for the Station Component */
int dsap, (*pfunc) ();
{
    struct  LAT     *plat;

    if ((dsap << (8-DSAP_SHIFT) & 0x00FF) != 0 || dsap == 0)
        return (FALSE);

    /* Check for duplicate dsaps. */
    if ( (plat = &lat[dsap >> DSAP_SHIFT])->stat == FREE) {
            plat->stat = INUSE;
            plat->p_sap_func = pfunc;
            return (TRUE);
    }
    else
        return(FALSE);
}

/* Function for deleting DSAPs */

Delete_Dsap_Address(dsap) /* If the specified connection exists, it is severed.
                             If the connection does not exist, the command is ignored. */
```

231421-44

/PCD/USR/CHUCK/CSRC/LLC.C

```
int dsap;
{
    lat[dsap >> DSAP_SHIFT].stat = FREE;
}

Recv_Frame(pfd)
    struct  FD              *pfd;
{
    struct  RBD             *prbd;
    struct  FRAME_STRUCT    *pfs;
    struct  LAT             *plat;

    prbd = (struct RBD *) Build_Ptr(pfd->rbd_offset);
    pfs  = (struct FRAME_STRUCT *) prbd->buff_ptr;

    if (pfd->rbd_offset != NULL) { /* There has to be a rbd attached
                                      to the fd, or else the frame is
                                      too short. */
        if (pfs->dsap == 0) {   /* if the frame is addressed to the Station
                                   Component, then a response may be required */

            if ( !(pfs->ssap & C_R_BIT) ) {/* if the frame received is a response,
                                              instead of a command, then reject it.
                                              Because this software does not implement
                                              DUPLICATE_ADDRESS_CHECK. -> no response
                                              frames should be recv'd */
                Station_Component_Response(pfd);
            }
        }
        /* not addressed to Station Component, */
        /* check to see if the dsap addressed is active */
        else if ((pfs->dsap << (8-DSAP_SHIFT) & 0x00FF) == 0 &&
                 (plat = &lat[(pfs->dsap) >> DSAP_SHIFT])->stat == INUSE ) {
            (*plat->p_sap_func)(pfd);    /* call the function associated
                                            with the dsap received */
            return;
        }
    }
    Put_Free_RFA(pfd); /* return the pfd if not given to the user saps */
}


Station_Component_Response(pfd)

    struct  FD      *pfd;
{
    struct FRAME_STRUCT     *prfs, *ptfs;
    struct TBD              *ptbd, *begin_ptbd, *q;
    struct RBD              *prbd;

    prbd = (struct RBD *) Build_Ptr(pfd->rbd_offset);
    prfs  = (struct FRAME_STRUCT *) prbd->buff_ptr;

    switch (prfs->cmd & ~P_F_BIT)
        {
        case    XID:
```

231421-45

/PCO/USR/CHUCK/CSRC/LLC.C

```
            while ((ptbd = Get_Tbd()) == pNULL);
            ptbd->act_cnt = EOFBIT | XID_LENGTH;
            bcopy ((char *) ptbd->buff_ptr, &xid_frame[0], XID_LENGTH);
            ptfs = (struct FRAME_STRUCT *) ptbd->buff_ptr;
            ptfs->cmd = prfs->cmd;

            ptfs->dsap = prfs->ssap | C_R_BIT; /* return the frame
                                                  to the sender */
            ptfs->ssap = 0;
            while(!Send_Frame(ptbd, Build_Ptr(pfd->src_addr)));
            break;

        case    TEST:

            for (prbd = (struct RBD *) Build_Ptr(pfd->rbd_offset),
                        q = begin_ptbd = pNULL; prbd != pNULL;
                                prbd = Build_Ptr(prbd->link)) {

                while ((ptbd = Get_Tbd()) == pNULL);
                if (q != pNULL)
                    q->link = Offset(ptbd);
                else
                    begin_ptbd = ptbd;
                ptbd->act_cnt = prbd->act_cnt;
                bcopy((char *) ptbd->buff_ptr, (char *) prbd->buff_ptr,
                                                    ptbd->act_cnt & 0x3FFF);
                q = ptbd;
            }

            ptfs = (struct FRAME_STRUCT *) begin_ptbd->buff_ptr;
            ptfs->cmd = prfs->cmd;

            ptfs->dsap = prfs->ssap | C_R_BIT;  /* return the frame to
                                                   the sender */
            ptfs->ssap = 0;
            while(!Send_Frame(begin_ptbd, Build_Ptr(pfd->src_addr)));
            break;

        }
}
```

231421-46

/PCO/USR/CHUCK/CSRC/UAP.C

```
/*****************************************************************************
*                                                                            *
*                        User Application Program                            *
*                 Async to IEEE 802.2/802.3 Protocol Converter               *
*                                                                            *
*****************************************************************************/

#include "dld.h"

/* ASCII Characters */
#define ESC      0x1B
#define LF       0x0A
#define CR       0x0D
#define BS       0x08
#define BEL      0x07
#define SP       0x20
#define DEL      0x7F
#define CTL_C        0x03

/* Hardware */
#define CH_B_CTL     0x00DE
#define CH_A_CTL     0x00DC
#define CH_B_DAT     0x00DA
#define CH_A_DAT     0x00D8
#define UART_STAT_MSK    0x70

/* Interrupt cases for 8274 */
#define UART_TX_B        0
#define UART_RECV_B      0x08
#define UART_RECV_ERR_B 0x0C
#define  EXT_STAT_INT_B 0x04
#define  EXT_STAT_INT_A 0x14

char     fifo_t[256];
char     fifo_r[256];
char     wra[5], wrb[5];
unsigned     char    in_fifo_t, out_fifo_t, in_fifo_r, out_fifo_r, actual;
u_short      t_buf_stat, r_buf_stat;

char     cbuf[80];    /* Command line buffer */
char     line[81];    /* Monitor Mode display line */

unsigned     char    dsap, ssap, send_flag, local_echo;
char     Dest_Addr[ADD_LEN];
char     Multi_Addr[ADD_LEN];

int tmstat; /* terminal mode status: for leaving terminal mode */
int dhex, monitor_flag, hs_stat;        /* flags */


extern   struct TBD      *Get_Tbd();
extern   char            *Build_Ptr();

extern   struct FLAGS    flags;

extern   char     xid_frame[];
extern   char     whoami[];
```

231421-47

```
/PCD/USR/CHUCK/CSRC/UAP.C

extern  struct  MAT     mat[];
extern  struct  LAT     lat[];
extern  char    *pNULL;

extern  unsigned long   good_xmit_cnt;
extern  u_short         underrun_cnt;
extern  u_short         no_crs_cnt;
extern  unsigned long   defer_cnt;
extern  u_short         sqe_err_cnt;
extern  u_short         max_col_cnt;
extern  unsigned long   recv_frame_cnt;
extern  u_short         reset_cnt;

extern  struct  SCB     scb;

/* Macro 'type' of definitions */

#define RTS_ONB outb(CH_B_CTL,0x05);outb(CH_B_CTL,wrb[5]=wrb[5]|0x02)
#define RTS_OFFB outb(CH_B_CTL,0x05);outb(CH_B_CTL,wrb[5]=wrb[5]&0xFD)
#define RTS_ONA outb(CH_A_CTL,0x05);outb(CH_A_CTL,wra[5]=wra[5]|0x02)
#define RTS_OFFA outb(CH_A_CTL,0x05);outb(CH_A_CTL,wra[5]=wra[5]&0xFD)
#define UART_TX_DI_B outb(CH_B_CTL,0x01);outb(CH_B_CTL,wrb[1]=wrb[1]&0xFD)
#define UART_TX_EI_B outb(CH_B_CTL,0x01);outb(CH_B_CTL,wrb[1]=wrb[1]|0x02)
#define UART_RX_DI_B outb(CH_B_CTL,0x01);outb(CH_B_CTL,wrb[1]=wrb[1]&0xE7)
#define UART_RX_EI_B outb(CH_B_CTL,0x01);outb(CH_B_CTL,wrb[1]=wrb[1]|0x10)
#define RESET_TX_INT outb(CH_B_CTL,0x28)
#define EOI_8274 outb(CH_A_CTL,0x38) /* 8274 int is IR3 on 80130 */
#define EOI_80130_8274 outb(0xE0,0x60)
#define EOI_80130_TIMER outb(0xE0,0x62)


Enable_Uart_Int()
{
    int     c;

    c = inb(0xE2);  /* read the 80130 interrupt mask register */
    outb(0xE2, 0x00FE & c); /* write to the 80130 interrupt mask register */
}

Disable_Uart_Int()
{
    int     c;

    c = inb(0xE2);
    outb(0xE2, 0x0001 | c);
}

Enable_Timer_Int()
{
    int     c;

    outb(0xEA, 125);
    outb(0xEA, 0x00); /* Timer 1 interrupts every .125 sec */
    send_flag = FALSE;
    c = inb(0xE2);  /* read the 80130 interrupt mask register */
    outb(0xE2, 0x00FB & c); /* write to the 80130 interrupt mask register */
}
```

231421-48

/PCO/USR/CHUCK/CSRC/UAP.C

```
Disable_Timer_Int()
{
    int     c;

    c = inb(0xE2);
    outb(0xE2, 0x0004 | c);
}


Co(c)
    char    c;
{
    while ( (inb(CH_B_CTL) & 4) == 0 );
    outb(CH_B_DAT, c);
}

Ci()
{
    while ( (inb(CH_B_CTL) & 1) == 0 );
    return(inb(CH_B_DAT) & 0x7F);
}

Read(pmsg, cnt, pact)
    char    *pmsg;
    unsigned char   cnt, *pact;
{
    unsigned char   i;
    char    c,  buf[200];

    for (i = c = 0; (c != CR) && (c != LF) && (i < 198); ) {
        c = Ci() & 0x7F;
        if (c == BS || c == DEL) {
            if (i > 0) {
                --i;
                Co(BS); Co(SP); Co(BS);
            }
        }
        else
            if (c >= SP) {
                Co(c);
                buf[i++] = c;
            }
        else
            if ((c == CR) || (c == LF)) {
                buf[i++] = CR;
                buf[i++] = LF;
            }
        else Co(BEL);
    }
    Co(CR); Co(LF);
    if (i > cnt)
        *pact = cnt;
    else
        *pact = i;
    for (i = 0; i < *pact ; i++)
        *pmsg++ = buf[i];
```

231421-49

/PCO/USR/CHUCK/CSRC/UAP.C

```
}

Read_Char()
{
    unsigned char   i;

    Read(&cbuf[0], 80, &actual);
    i = Skip(&cbuf[0]);
    return(cbuf[i]);
}

Write(pmsg)
    char    *pmsg;
{
    while (*pmsg != '\0') {
        if (*pmsg == '\n')
            Co(CR);
        Co(*pmsg++);
    }
}

Fatal(pmsg) /* write a message to the screen then stop */
    char    *pmsg;
{
    Write("Fatal: ");
    Write(pmsg);
    for(;;);
}

Bug(pmsg)   /* write a message to the screen then continue */
    char    *pmsg;
{
    Write("Bug: ");
    Write(pmsg);
}

Ascii_To_Char(c) /* convert ASCII-Hex to Char */
    char    c;
{
    if (('0' <= c) && (c <= '9'))
        return(c - '0');
    if (('A' <= c) && (c <= 'F'))
        return(c - 0x37);
    if (('a' <= c) && (c <= 'f'))
        return(c - 0x57);
    return(0xFF);
}

Lower_Case(c)
char    c;
{
    if (('a' <= c) && (c <= 'z'))
        return(c);
    if (('A' <= c) && (c <= 'Z'))
        return(c + 0x20);
    return(0);
}
```

231421-50

/PCO/USR/CHUCK/CSRC/UAP.C

```
Char_To_Ascii(c, ch) /* convert char to ASCII-Hex */
    unsigned char   c, ch[];
{
    unsigned char   i;

    i = (c & 0xF0) >> 4;
    if (i < 10)
        ch[0] = i + 0x30;
    else
        ch[0] = i + 0x37;
    i = (c & 0x0F) ;
    if (i < 10)
        ch[1] = i + 0x30;
    else
        ch[1] = i + 0x37;
    ch[2] = '\0';
}

Skip(pmsg)  /* skip blanks */
    char    *pmsg;
{
    int     i;

    for (i = 0; *pmsg == ' '; i++, pmsg++);
    return(i);
}

Read_Int()  /* Read a 16 bit Integer */
{
    u_short     wd, wh, wd1, wh1, j;
    char        i, done, hex, dover, hover;

    for (done = FALSE; done == FALSE; ) {
        Read(&cbuf[0], 80, &actual);
        i = Skip(&cbuf[0]);

        for (hex = dover = hover = FALSE, wd = wh = wd1 = wh1 = 0;
                            (j = Ascii_To_Char(cbuf[i])) <= 15; i++) {
            if (j > 9)
                hex = TRUE;
            wd = wd*10 + j;
            wh = wh*16 + j;
            if (wd < wd1)
                dover = TRUE;
            if (wh < wh1)
                hover = TRUE;
            wd1 = wd; wh1 = wh;
        }
        if (cbuf[i] == 'H' || cbuf[i] == 'h' || cbuf[i] == CR ||
                                        cbuf[i] == LF || cbuf[i] == ' ') {
            if (cbuf[i] == 'H' || cbuf[i] == 'h')
                hex = TRUE;
            if (hex == TRUE && hover == FALSE)
                done = TRUE;
            if (hex == FALSE && dover == FALSE)
                done = TRUE;
```

231421-51

/PCO/USR/CHUCK/CSRC/UAP.C

```
            if (!done) {
                Write("\n This number is too big.\n It has to be less than 65536.\n");
                Write("\n Enter number --> ");
            }
        }
        else
            Write(" Illegal Character\n Enter a number -->");
    }
    if (hex)
        return(wh);
    return(wd);
}

Int_To_Ascii(value, base, ld, ch, width) /* convert an integer to an ASCII string */
    unsigned long   value;
    u_short     base, width;
    char    ch[], ld;
{
    u_short i, j;

    for (i = 0; i < width; i++) {
        j = value % base;
        if (j < 10)   ch[i] = j + 0x30;
        else ch[i] = j + 0x37;
        value = value / base;
    }
    for (i = width - 1; ch[i] == '0' && i > 0; i--)
        ch[i] = ld;
    ch[width] = '\0';
}

Write_Long_Int(dw, i)
    unsigned long   dw;
    u_short     i;
{
    u_short     j;
    char    ch[11];

    if (dhex)
        Int_To_Ascii(dw, 16, ' ', &ch[0], 8);
    else
        Int_To_Ascii(dw, 10, ' ', &ch[0], 10);
    for (j = 0; ch[j] != '\0'; i--, j++)
        line[i] = ch[j];
}

Write_Short_Int(w, i)
    u_short w, i;
{
    u_short j;
    char    ch[6];
    unsigned long   dw;

    dw = w;
    if (dhex)
        Int_To_Ascii(dw, 16, '0', &ch[0], 4);
    else
```

231421-52

/PCD/UBR/CHUCK/CBRC/UAP.C

```
        Int_To_Ascii(dw, 10, '0', &ch[0], 5);
    for (j = 0; ch[j] != '\0'; i--, j++)
        line[i] = ch[j];
}


Yes()
{
    char    b;

    for ( ; ; ) {
        b = Read_Char();
        if ((b == 'Y') || (b == 'y'))
            return(TRUE);
        if ((b == 'N') || (b == 'n'))
            return(FALSE);
        Write(" Enter a Y or N --> ");
    }
}

Read_Addr(pmsg, add, cnt) /* pmsg - pointer to the output message */
                          /* add - pointer to the address */
                          /* cnt - number of bytes in the address */
    char    *pmsg, add[], cnt;
{
    char    i, j;

    for ( ; ; ) {
        Write(pmsg);
        Read(&cbuf[0], 80, &actual);
        for (j = skip(&cbuf[0]), i = 0; i < 2*cnt ; i++, j++) {
            if (('0' <= cbuf[j]) && (cbuf[j] <= '9'))
                cbuf[i] = cbuf[j] - '0';
            else
                if (('A' <= cbuf[j]) && (cbuf[j] <= 'F'))
                    cbuf[i] = cbuf[j] - 0x37;
            else
                if (('a' <= cbuf[j]) && (cbuf[j] <= 'f'))
                    cbuf[i] = cbuf[j] - 0x57;
            else {
                Write(" Illegal Character\n");
                break;
            }
        }
        if (i >= 2*cnt - 1)
            break;
    }
    for (i = 0; i <= cnt - 1; i++)
        add[(cnt - 1) - i] = cbuf[2*i] << 4 | cbuf[2*i + 1];
}

Write_Addr(padd, cnt)
    char    padd[], cnt;
{
    unsigned char   i, c[3];

    for ( ; cnt >0 ; cnt--) {
```

231421-53

```
/PCD/USR/CHUCK/CSRC/UAP.C


        i = padd[cnt-1];
        Char_To_Ascii(i, &c[0]);
        Write(&c[0]);
    }
    c[0] = '\n';
    c[1] = '\0';
    Write(&c[0]);
}

Recv_Data_1(pfd)    /* Receives the frame from the 802.2 module */

    struct  FD              *pfd;
{
    struct FRAME_STRUCT     *prfs, *ptfs;
    struct TBD              *ptbd, *begin_ptbd, *q;
    struct RBD              *prbd;
    char                    *prbuf;
    int                     cnt;


    prbd = (struct RBD *) Build_Ptr(pfd->rbd_offset);
    prfs  = (struct FRAME_STRUCT *) Build_Ptr(prbd->buff_ptr);

    switch (prfs->cmd & ~P_F_BIT)   {
        case    UI:
            if (monitor_flag)
                break;  /* Don't put data in fifo unless in terminal mode */
            prbuf = (char *) prfs;
            prbuf += 3; /* skip over the header info and point to the data */
            cnt = 3;
            pfd->length -= 3;
            for (; prbd != pNULL; cnt = 0, prbuf = (char *) prbd->buff_ptr){
                for ( ; cnt < (prbd->act_cnt & 0x03FFF) && pfd->length > 0;
                                        cnt++, prbuf++, pfd->length--) {
                    while(r_buf_stat == FULL);
                    Fifo_R_In(*prbuf);
                }
                prbd = Build_Ptr(prbd->link);
#ifdef DEBUG
                if (pfd->length == 0 && prbd != pNULL)
                    Fatal("Uap: Recv_Data_1(pfd) ");
#endif /* DEBUG */
            }
            break;

        case    XID:

            while ((ptbd = Get_Tbd()) == pNULL);
            ptbd->act_cnt = EOFBIT | XID_LENGTH;
            bcopy ((char *) ptbd->buff_ptr, &xid_frame[0], XID_LENGTH);
            ptfs = (struct FRAME_STRUCT *) ptbd->buff_ptr;
            ptfs->cmd = prfs->cmd;

            ptfs->dsap = prfs->ssap | C_R_BIT; /* return the frame
                                                  to the sender */
            ptfs->ssap = ssap;
            while(!Send_Frame(ptbd, Build_Ptr(pfd->src_addr)));
```

231421-54

```
/PCO/USR/CHUCK/CSRC/UAP.C

            break;

        case    TEST:

            for (prbd = (struct RBD *) Build_Ptr(pfd->rbd_offset),
                        q = begin_ptbd = pNULL; prbd != pNULL;
                                prbd = Build_Ptr(prbd->link)) {
                while ((ptbd = Get_Tbd()) == pNULL);
                if (q != pNULL)
                    q->link = Offset(ptbd);
                else
                    begin_ptbd = ptbd;
                ptbd->act_cnt = prbd->act_cnt;
                bcopy((char *) ptbd->buff_ptr, (char *) prbd->buff_ptr,
                                                ptbd->act_cnt & 0x3FFF);
                q = ptbd;
            }

            ptfs = (struct FRAME_STRUCT *) begin_ptbd->buff_ptr;
            ptfs->cmd = prfs->cmd;

            ptfs->dsap = prfs->ssap | C_R_BIT;  /* return the frame to
                                                   the sender */
            ptfs->ssap = ssap;
            while(!Send_Frame(begin_ptbd, Build_Ptr(pfd->src_addr)));
            break;

    }
    Put_Free_RFA(pfd);  /* return the frame */
}


Fifo_T_Out()    /* called by main program */
{
    char    c;

    c = fifo_t[out_fifo_t++];

    Disable_Uart_Int();
    if (out_fifo_t == in_fifo_t)    /* if the fifo is empty */
        t_buf_stat = EMPTY; /* stop filling Transmit Buffer Descriptors */
    else        /* if the fifo was full and is now draining */
        if (t_buf_stat == FULL && out_fifo_t - 80 == in_fifo_t) { /* turn on
                                                                    the spigot */
            RTS_ONB;
            t_buf_stat = INUSE;
        }
    Enable_Uart_Int();
    return(c);
}

Fifo_T_In(c)    /* called by Uart receive interrupt */
    char    c;
{

    fifo_t[in_fifo_t++] = c;
    if (t_buf_stat == EMPTY)
```

231421-55

```
/PCO/USR/CHUCK/CSRC/UAP.C

        t_buf_stat = INUSE; /* start filling Transmit Buffer Descriptor */
    else        /* if there are only 20 locations left, turn off the spigot */
        if (t_buf_stat == INUSE && in_fifo_t + 20 == out_fifo_t) {
            RTS_OFFB;
            t_buf_stat = FULL;
        }
}

Fifo_R_Out()    /* called by transmit interrupt */
{
    char    c;

    c = fifo_r[out_fifo_r++];

    if (out_fifo_r == in_fifo_r)    /* if the fifo is empty */
        r_buf_stat = EMPTY;
    else        /* if the fifo was full and is now draining */
        if (r_buf_stat == FULL && out_fifo_r - 81 == in_fifo_r)
            r_buf_stat = INUSE;
    return(c);
}

Fifo_R_In(c)    /* called by Recv_Data_1() */
    char    c;
{
    fifo_r[in_fifo_r++] = c;
    Disable_Uart_Int();
    if (r_buf_stat == EMPTY) {
        UART_TX_EI_B;
        Co(0);                  /* prime the interrupt */
        r_buf_stat = INUSE;
    }
    else        /* if the buffer is full, indicate it */
        if (r_buf_stat == INUSE && in_fifo_r == out_fifo_r)
            r_buf_stat = FULL;
    Enable_Uart_Int();
}

Isr_Uart()
{
    int     stat;
    char    c;

    outb(CH_B_CTL, 2);  /* point to RR2 in 8274 */

    switch(inb(CH_B_CTL) & 0x1C){ /* read 8274 interrupt vector and service it */

    case UART_TX_B:

        if (r_buf_stat == EMPTY) {
            UART_TX_DI_B;           /* if fifo is empty disable transmitter */
            RESET_TX_INT;
        }
        else
            outb (CH_B_DAT, Fifo_R_Out());
        break;
```

231421-56

```
/PCO/USR/CHUCK/CSRC/UAP.C

    case UART_RECV_ERR_B:

        outb(CH_B_CTL, 1); /* point to RR1 in 8274 */
        stat = inb(CH_B_CTL);
        outb(CH_B_CTL, 0x30);
        if (stat & 0x0010)
            Write("\nParity Error Detected\n");
        if (stat & 0x0020)
            Write("\nOverrun Error Detected\n");
        if (stat & 0x0040)
            Write("\nFraming Error Detected\n");
        break;

    case UART_RECV_B:

        c = inb(CH_B_DAT);

        if (hs_stat == TRUE) {
            hs_stat = FALSE;  /* Flag to terminate High Speed Transmit mode */
            break;
        }

        if (local_echo)
            Co(c);            /* echo the char back to the terminal; could cause
                                 a transmit overrun if Tx interrupt is enabled */
        if (c == CTL_C)
            tmstat = FALSE;
        else
            Fifo_T_In(c);
        break;

    case EXT_STAT_INT_B:

        outb(CH_B_CTL, 0x10);   /* reset external status interrupts */
        break;

    case EXT_STAT_INT_A:

        outb(CH_A_CTL, 0x10);
        break;


    default:
        ;
    }
    EOI_80130_8274;
    EOI_8274;
}

Isr2()
{
    send_flag = TRUE;
    outb(0xEA, 125);
    outb(0xEA, 0x00); /* Timer 1 interrupts every .125 sec */
    outb(0xE0, 0x62);   /* EOI 80130 */
}
```

231421-57

/PCO/USR/CHUCK/CSRC/UAP.C

```
Load_Lsap()
{
    int     Recv_Data_1();

    for(;;) {
        Read_Addr("\n\nEnter this Station's LSAP in Hex --> ", &ssap, 1);
        if (!Add_Dsap_Address(ssap, Recv_Data_1)) {
            Write("\n\nError: LSAP Address must be one of the following:\n");
            Write("\n       20H, 40H, 60H, 80H, A0H, C0H, E0H \n");
        }
        else break;
    }
}

Load_Multicast()
{

    for ( ; ; ) {
        Read_Addr("\nEnter the Multicast Address in Hex -->",
                                            &Multi_Addr[0], ADD_LEN);
        if ((Multi_Addr[0] & 0x01) == 0)
            Write("\nSorry, the LSB of the Multicast Address must be 1\n");
        else { if (!Add_Multicast_Address(&Multi_Addr[0])) {
                    Write("\n\nSorry, Multicast Address Table is full!\n");
                    break;
                }
                else {
                    Write("\n\nWould you like to add another Multicast Address?");
                    Write(" (Y or N) --> ");
                    if (!Yes())
                        break;
                }
        }
    }
}

Remove_Multicast()
{

    for ( ; ; ) {
        Read_Addr("\nEnter the Multicast Address that you want to delete in Hex -->",
                                            &Multi_Addr[0], ADD_LEN);
        if ((Multi_Addr[0] & 0x01) == 0)
            Write("\nSorry, the LSB of the Multicast Address must be 1\n");
        else { if (!Delete_Multicast_Address(&Multi_Addr[0])) {
                    Write("\n\nSorry, that Multicast Address doesn't exist!\n");
                    break;
                }
                else {
                    Write("\n\nWould you like to delete another Multicast Address?");
                    Write(" (Y or N) --> ");
                    if (!Yes())
                        break;
                }
        }
    }
}
```

231421-58

```
/PCO/USR/CHUCK/CSRC/UAP.C


Print_Addresses()
{
    struct  MAT *pmat;
    int         stat;

    Write("\n This Stations Host Address is: ");
    Write_Addr(&whoami[0], ADD_LEN);
    Write("\n The Address of the Destination Node is: ");
    Write_Addr(&Dest_Addr[0], ADD_LEN);
    Write("\n This Stations LSAP Address is: ");
    Write_Addr(&ssap, 1);
    Write("\n The Address of the Destination LSAP is: ");
    Write_Addr(&dsap, 1);
    stat = FALSE;
    for (pmat = &mat[0]; pmat <= &mat[MULTI_ADDR_CNT - 1]; pmat++)
        if (pmat->stat == INUSE) {
            stat = TRUE;
            break;
        }
    if (stat)   {
        Write("\n The following Multicast Addresses are enabled: ");
        for (pmat = &mat[0]; pmat <= &mat[MULTI_ADDR_CNT - 1]; pmat++)
            if (pmat->stat == INUSE) {
                Write_Addr(&pmat->addr[0], ADD_LEN);
                Write("                                                    ");
            }
    }
    else
        Write("\n There are no Multicast Addresses enabled.\n");
}

Init_DataLink()
{
    int     stat;

    if ((stat = Init_Llc()) == PASSED)
        Write("\n\nPassed Diagnostic Self Tests\n\n\n");
    else
        if(stat == FAILED_DIAGNOSE)
            Write("\n\nFailed: Self Test Diagnose Command\n");
    else
        if(stat == FAILED_LPBK_INTERNAL)
            Write("\n\nFailed: Internal Loopback Self Test\n");
    else
        if(stat == FAILED_LPBK_EXTERNAL)
            Write("\n\nFailed: External Loopback Self Test\n");
    else
        if(stat == FAILED_LPBK_TRANSCEIVER)
            Write("\n\nFailed: External Loopback Through Transceiver Self Test\n");
}

Init_Uap()
{

    outb(0xE0, 0x31);   /*initalize 80130 pic - ICW1 */
    outb(0xE2, 0x20);   /* ICW2 */
```

231421-59

```
/PCO/USR/CHUCK/CSRC/UAP.C

    outb(0xE2, 0x10);    /* ICW3 */
    outb(0xE2, 0x0D);    /* ICW4 */
    outb(0xE2, 0x10);    /* ICW6 */
    outb(0xE2, 0xFF);    /* mask all interrupts */

    outw(0xFF20, 0x0020);    /* set 80186 vector base */

    /* Initialize the 80130 timers for Terminal Mode */

    outb(0xEE, 0x34);
    outb(0xE8, 0xB8);
    outb(0xE8, 0x08); /* SYSTICK set for 1 msec */
    outb(0xEE, 0x70);
    outb(0xEA, 125);
    outb(0xEA, 0x00); /* Timer 1 interrupts every .125 sec */

    /* Initialize the 8274 */
    outb(CH_B_CTL, 0x10); outb(CH_B_CTL, 0x28); outb(CH_B_CTL, 0x30);
    outb(CH_A_CTL, 0x38);
    outb(CH_B_CTL, 2); outb(CH_B_CTL, wrb[2] = 0x14);
    outb(CH_B_CTL, 1); outb(CH_B_CTL, wrb[1] = 0x15);
    outb(CH_B_CTL, 5); outb(CH_B_CTL, wrb[5] = 0xEA);

    Write("\n\n\n\n\n\n\n\n\n\n\n\n\n");
    Write("           ********************************************************\n");
    Write("           * 82586 IEEE 802.2/802.3 Compatible Data Link Driver *\n");
    Write("           ********************************************************\n");
    Write("\n\n\n\n\n\n\n\n");

    Init_DataLink();

    dhex = FALSE;
    monitor_flag = TRUE;

    Read_Addr("\n\nEnter the Address of the Destination Node in Hex --> ",
                                                    &Dest_Addr[0], ADD_LEN);
    Load_Lsap();

    Read_Addr("\n\nEnter the Destination Node's LSAP in Hex --> ", &dsap, 1);

    Write("\n\nDo you want to Load any Multicast Addresses? (Y or N) -->");

    if (Yes())
        Load_Multicast();

    Print_Addresses();
}

Terminal_Mode()
{
    int     frame_cnt, buf_cnt;
    struct  TBD     *ptbd, *q, *begin_ptbd;
    char            *pbuf, c;

    Write("\n Would you like the local echo on? (Y or N)-->");

    if(Yes())
```

231421-60

/PCO/USR/CHUCK/CSRC/UAP.C

```
        local_echo = TRUE;
    else
        local_echo = FALSE;

    Write("\n This program will now enter the terminal mode.\n\n");
    Write("\n Press ^C then CR to return back to the menu\n\n");

    /* Initialize Fifo variables */

    out_fifo_t = in_fifo_t = out_fifo_r = in_fifo_r = 0;
    t_buf_stat = EMPTY;  r_buf_stat = EMPTY;

    EOI_80130_8274;
    Enable_Uart_Int();
    Enable_Timer_Int();
    monitor_flag = FALSE;
    tmstat = TRUE;
    while (tmstat)  {

        for (frame_cnt = 0; frame_cnt < MAX_FRAME_SIZE;  q = ptbd) {

            while ((ptbd = Get_Tbd()) == pNULL); /* get a xmit buffer from the
                                                    data link */
            pbuf = (char *) ptbd->buff_ptr;  /* point to the buffer */
            buf_cnt = 0;

            if (frame_cnt == 0) { /* if this is the first buffer, add on IEEE 802.2
                                     header information */
                begin_ptbd = ptbd;
                *pbuf++ = dsap;
                *pbuf++ = ssap;
                *pbuf++ = UI;
                buf_cnt = 3;
            }
            else  q->link = Offset(ptbd); /* if this isn't the first buffer
                                     link the previous buffer with the new one */
            /* fill up a data link xmit buffer from async transmit fifo */
            for ( ; buf_cnt < TBUF_SIZE && frame_cnt < MAX_FRAME_SIZE;
                                        buf_cnt++, pbuf++, frame_cnt++) {
                if (frame_cnt != 0 && send_flag)
                    break;

                while (t_buf_stat == EMPTY);    /* wait until fifo has data */
                if ((c = *pbuf = Fifo_T_Out()) == CR) {
                    ++buf_cnt ; ++pbuf; ++frame_cnt;
                    break;
                }
            }
            if (c == CR || buf_cnt < TBUF_SIZE || send_flag) { /* last buffer in list */
                ptbd->act_cnt = buf_cnt | EOFBIT;
                send_flag = FALSE;
                break;
            }
        }
        while(!Send_Frame(begin_ptbd, &Dest_Addr[0])); /* keep trying until
                                                          successful */
    }
```

231421-61

/PCO/USR/CHUCK/CSRC/UAP.C

```
    Disable_Uart_Int();
    Disable_Timer_Int();
    monitor_flag = TRUE;
}


struct  TBD     *Build_Frame(cnt)
    u_short     cnt;
{
    u_short         buf_cnt, frame_cnt, i;
    struct  TBD     *ptbd, *q, *begin_ptbd;
    char            *pbuf;


    i = 0x20; frame_cnt = 0;

    for ( ; ; q = ptbd) {

        while ((ptbd = Get_Tbd()) == pNULL); /* get a xmit buffer from the
                                                data link */

        pbuf = (char *) ptbd->buff_ptr;  /* point to the buffer */
        buf_cnt = 0;

        if (frame_cnt == 0) { /* if this is the first buffer, add on IEEE 802.2
                                 header information */
            begin_ptbd = ptbd;
            *pbuf++ = dsap;
            *pbuf++ = ssap;
            *pbuf++ = UI;
            buf_cnt = 3;
        }
        else  q->link = Offset(ptbd); /* if this isn't the first buffer
                                 link the previous buffer with the new one */
        /* fill up a data link xmit buffer with ASCII characters */
        for (; buf_cnt < TBUF_SIZE && cnt > 0;
                                 i++, buf_cnt++, pbuf++, cnt--, frame_cnt++) {
            *pbuf = i;
            if (i > 0x7E)
                i = 0x1F;
        }
        if (cnt == 0) { /* last buffer in list */
            ptbd->act_cnt = buf_cnt | EOFBIT;
            break;
        }
    }
    return(begin_ptbd);
}


Monitor_Mode()
{
    u_short     xmit, cnt, i;
    struct  TBD     *Build_Frame(), *ptbd;

    Write(" Do you want this station to transmit? (Y or N) --> ");
    if (Yes())
```

231421-62

/PCO/USR/CHUCK/CSRC/UAP.C

```
        for (xmit = FALSE; xmit == FALSE; ) {
        Write("\n Enter the number of data bytes in the frame --> ");
        cnt = Read_Int();
        if (cnt > 2045)
            Write ("\n Sorry, the number has to be less than 2046!\n");
        else
            xmit = TRUE;
        }

    else xmit = FALSE;

    Write("\n Hit any key to exit Monitor Mode.\n\n");

Write("  # of Good      # of Good      CRC       Alignment      No            Receive\n");
Write("   Frames          Frames      Errors      Errors        Resource      Overrun\n");
Write(" Transmitted      Received                               Errors        Errors\n");

   /* "01234567890123456789012345678901234567890123456789012345678901234567890123456789012345678
         xxxxxxxxxx     xxxxxxxxxx     xxxx         xxxx           xxxx            xxxx
           xxxxxxxx       xxxxxxxx
                11             25       33           44             57              71 */

    for (i = 0; i < 79; i++)
        line[i] = 0x20;
    line[79] = CR;
    line[80] = '\0';

    while ((inb(CH_B_CTL) & 1) == 0) {
        for (i = 0; i < 72; i++)
            line[i] = 0x20;
        Write_Long_Int(good_xmit_cnt, 11);
        Write_Long_Int(recv_frame_cnt, 25);
        Write_Short_Int(scb.crc_errs, 33);
        Write_Short_Int(scb.aln_errs, 44);
        Write_Short_Int(scb.rsc_errs, 57);
        Write_Short_Int(scb.ovr_errs, 71);
        Write(&line[0]);
        if (xmit) {
            ptbd = Build_Frame(cnt);
            while(!Send_Frame(ptbd, &Dest_Addr[0]));
        }
    }
    i = Ci();
}

Hs_Xmit_Mode()
{
    struct  TBD  *ptbd;

    Write("\n Hit any key to exit High Speed Transmit Mode.\n\n");

    hs_stat = TRUE;
    EOI_80130_8274;
    Enable_Uart_Int();

    /* Execute this loop until a recv char interrupt happends at Uart */
```

231421-63

/PCO/USR/CHUCK/CBRC/UAP.C

```
    while (hs_stat) {
        while ((ptbd = Get_Tbd()) == pNULL); /* get a xmit buffer from
                                                the data link */
        ptbd->act_cnt |= EOFBIT;          /* set the End Of Frame bit */
        while(!Send_Frame(ptbd, &Dest_Addr[0])); /* Send Frame */
    }

    Disable_Uart_Int();
}

Print_Cnt()
{
    char    ch[11], base, dwidth, width, i;
    unsigned    long    temp;

    if (dhex) {
        dwidth = 8;
        width = 4;
        base = 16;
    }
    else {
        base = 10;
        dwidth = 10;
        width = 5;
    }

    Write("\n\n Good frames transmitted: ");
    for (i = 1; i <= 11 - dwidth; i++)
        Co(SP);
    Int_To_Ascii(good_xmit_cnt, base, ' ', &ch[0], dwidth);
    for (i = dwidth - 1; i >= 0; i--)
        Co(ch[i]);
    Write("  Good frames received: ");
    for (i = 1; i <= 15 - dwidth; i++)
        Co(SP);
    Int_To_Ascii(recv_frame_cnt, base, ' ', &ch[0], dwidth);
    for (i = dwidth - 1; i >= 0; i--)
        Co(ch[i]);
    Write("\n\n CRC errors received: ");
    for (i = 1; i <= 15 - width; i++)
        Co(SP);
    temp = scb.crc_errs;
    Int_To_Ascii(temp, base, ' ', &ch[0], width);
    for (i = width - 1; i >= 0; i--)
        Co(ch[i]);
    Write("  Alignment errors received: ");
    for (i = 1; i <= 10 - width; i++)
        Co(SP);
    temp = scb.aln_errs;
    Int_To_Ascii(temp, base, ' ', &ch[0], width);
    for (i = width - 1; i >= 0; i--)
        Co(ch[i]);
    Write("\n\n Out of Resource frames: ");
    for (i = 1; i <= 12 - width; i++)
        Co(SP);
    temp = scb.rsc_errs;
    Int_To_Ascii(temp, base, ' ', &ch[0], width);
```

231421-64

```
/PCO/USR/CHUCK/CSRC/UAP.C

    for (i = width - 1; i >= 0; i--)
        Co(ch[i]);
    Write("  Receiver overrun frames: ");
    for (i = 1; i <= 12 - width; i++)
        Co(SP);
    temp = scb.ovr_errs;
    Int_To_Ascii(temp, base, ' ', &ch[0], width);
    for (i = width - 1; i >= 0; i--)
        Co(ch[i]);
    Write("\n\n 82586 Reset: ");
    for (i = 1; i <= 23 - width; i++)
        Co(SP);
    temp = reset_cnt;
    Int_To_Ascii(temp, base, ' ', &ch[0], width);
    for (i = width - 1; i >= 0; i--)
        Co(ch[i]);
    Write("  Transmit underrun frames: ");
    for (i = 1; i <= 11 - width; i++)
        Co(SP);
    temp = underrun_cnt;
    Int_To_Ascii(temp, base, ' ', &ch[0], width);
    for (i = width - 1; i >= 0; i--)
        Co(ch[i]);
    Write("\n\n Lost CRS: ");
    for (i = 1; i <= 26 - width; i++)
        Co(SP);
    temp = no_crs_cnt;
    Int_To_Ascii(temp, base, ' ', &ch[0], width);
    for (i = width - 1; i >= 0; i--)
        Co(ch[i]);
    Write("  SQE errors: ");
    for (i = 1; i <= 25 - width; i++)
        Co(SP);
    temp = sqe_err_cnt;
    Int_To_Ascii(temp, base, ' ', &ch[0], width);
    for (i = width - 1; i >= 0; i--)
        Co(ch[i]);
    write("\n\n Maximum retry: ");
    for (i = 1; i <= 21 - width; i++)
        Co(SP);
    temp = max_col_cnt;
    Int_To_Ascii(temp, base, ' ', &ch[0], width);
    for (i = width - 1; i >= 0; i--)
        Co(ch[i]);
    Write("  Frames that deferred: ");
    for (i = 1; i <= 15 - dwidth; i++)
        Co(SP);
    Int_To_Ascii(defer_cnt, base, ' ', &ch[0], dwidth);
    for (i = dwidth - 1; i >= 0; i--)
        Co(ch[i]);
}


Print_Help()
{
    Write ("\n\n Commands are:\n\n");
    Write (" T - Terminal Mode                    M - Monitor Mode\n");
```

231421-65

/PCO/USR/CHUCK/CSRC/UAP.C

```
    Write (" X - High Speed Transmit Mode          V - Change Transmit Statistics\n");
    Write (" P - Print All Counters                C - Clear All Counters\n");
    Write (" A - Add a Multicast Address           Z - Delete a Multicast Address\n");
    Write (" S - Change the SSAP Address           D - Change the DSAP Address\n");
    Write (" N - Change Destination Node Address   L - Print All Addresses\n");
    Write (" R - Re-Initialize the Data Link       B - Change the number Base\n");
}

Main()
{
    int     c;

    Init_Uap();
    Print_Help();

    for (;;) {
        Write ("\n\n Enter a command, type H for Help --> ");
        c = Read_Char();
        switch (Lower_Case(c)) {

        case 'h':
            Print_Help();
            break;
        case 'm':
            Monitor_Mode();
            break;
        case 't':
            Terminal_Mode();
            break;
        case 'x':
            Hs_Xmit_Mode();
            break;
        case 'v':
            Write("\n Transmit Statistics are now ");
            if (flags.stat_on == 1)
                Write("on.\n Would you like to change it ? (Y or N) --> ");
            else
                Write("off.\n Would you like to change it ? (Y or N) --> ");
            if (Yes()) {
                if (flags.stat_on == 1)
                    flags.stat_on = 0;
                else flags.stat_on = 1;
            }
            break;
        case 'p':
            Print_Cnt();
            break;
        case 'c':
            Clear_Cnt();
            break;
        case 'a':
            Load_Multicast();
            break;
        case 'z':
            Remove_Multicast();
            break;
        case 's':
```

231421-66

/PCO/USR/CHUCK/CBRC/UAP.C

```
            Delete_Dsap_Address(ssap);
            Load_Lsap();
            break;
        case 'd':
            Read_Addr("\n\nEnter the Destination Node's LSAP in Hex --> ", &dsap, 1);
            break;
        case 'n':
            Read_Addr("\n\nEnter the Address of the Destination Node in Hex --> ",
                                                    &Dest_Addr[0], ADD_LEN);
            break;
        case 'l':
            Print_Addresses();
            break;
        case 'r':
            Software_Reset();
            Init_DataLink();
            Add_Dsap_Address(ssap, Recv_Data_1);
            break;
        case 'b':
            Write("\n The current base is ");
            if (dhex == TRUE)
                Write("Hex.\n Would you like to change it ? (Y or N) --> ");
            else
                Write("Decimal.\n Would you like to change it ? (Y or N) --> ");
            if (Yes()) {
                if (dhex == TRUE)
                    dhex = FALSE;
                else dhex = TRUE;
            }
            break;

        default:
            Write ("\n Unknown command\n");
            break;
        }
    }
}
```

231421-67

/PCO/USR/CHUCK/CSRC/ASSY.ASM

```
name    c_assy_support

stack   segment stack   'stack'
stktop  label   word
stack   ends

DLD_DATA                segment public  'DATA'
extrn   SEGMT_:word                    ; data segment address
DLD_DATA                ends

UAP_DATA        segment public   'DATA'
UAP_DATA        ends

DLD_CODE        segment public  'CODE'
        extrn    Isr_Timeout_:far, Isr_586_:far, Isr7_:far
        extrn    Isr6_:far, Isr5_:far, Isr1_:far
DLD_CODE        ends

UAP_CODE        segment public  'CODE'
        extrn   Isr_Uart_:far, Isr2_:far, Main_:far
UAP_CODE        ends


DG_CODE         segment public  'CODE'

        public   inw_, outw_, init_intv_, enable_, disable_, Build_Ptr_
        public   Offset_, begin, inb_, outb_

arg1    equ     [BP + 6]
arg2    equ     [BP + 8]

        assume  CS:DG_CODE
        assume  DS:DLD_DATA

;+
;       initialization program for the 82586 data link driver
;-

begin:

        sti
        mov     ax, DLD_DATA ;get base of dgroup and
        mov     SEGMT_, ax      ;pass the segment value to the c program
        mov     ds, ax
        call Main_              ;go to the c program
        hlt



inb_    proc    far
        push    BP
        mov     BP, SP
        push    DX
        mov     DX, arg1
        in      AL, DX
        pop     DX
        mov     SP, BP
```

231421-68

/PCO/USR/CHUCK/CSRC/ASSY.ASM

```
        pop     BP
        ret
inb_    endp

outb_   proc    far
        push    BP
        mov     BP, SP
        push    DX
        push    AX
        mov     DX, arg1
        mov     AX, arg2
        out     DX, AL
        pop     AX
        pop     DX
        mov     SP, BP
        pop     BP
        ret
outb_   endp

inw_    proc    far
        push    BP
        mov     BP, SP
        push    DX
        mov     DX, arg1
        in      AX, DX
        pop     DX
        mov     SP, BP
        pop     BP
        ret
inw_    endp

outw_   proc    far
        push    BP
        mov     BP, SP
        push    DX
        push    AX
        mov     DX, arg1
        mov     AX, arg2
        out     DX, AX
        pop     AX
        pop     DX
        mov     SP, BP
        pop     BP
        ret
outw_   endp

Build_Ptr_      proc    far
        push    BP
        mov     BP, SP
        mov     DX, DLD_DATA
        mov     AX, arg1
        mov     SP, BP
        pop     BP
        ret
Build_Ptr_      endp

Offset_ proc    far
```

231421-69

```
/PCO/USR/CHUCK/CSRC/ASSY.ASM

        push    BP
        mov     BP, SP
        mov     AX, arg1
        mov     SP, BP
        pop     BP
        ret
Offset_ endp

serve_int_isr   proc    far
        push    AX
        push    BX
        push    CX
        push    DX
        push    SI
        push    DI
        push    DS
        push    ES

        mov     AX, DLD_DATA
        mov     DS, AX
        mov     ES, AX

        call    Isr_386_

        pop     ES
        pop     DS
        pop     DI
        pop     SI
        pop     DX
        pop     CX
        pop     BX
        pop     AX
        iret
serve_int_isr   endp

serve_int_8274  proc    far
        push    AX
        push    BX
        push    CX
        push    DX
        push    SI
        push    DI
        push    DS
        push    ES

        mov     AX, UAP_DATA
        mov     DS, AX
        mov     ES, AX

        call    Isr_Uart_

        pop     ES
        pop     DS
        pop     DI
        pop     SI
        pop     DX
```

231421-70

```
/PCO/USR/CHUCK/CSRC/ASSY.ASM

        pop     CX
        pop     BX
        pop     AX
        iret
serve_int_8274  endp

serve_int_timeout       proc    far
        push    AX
        push    BX
        push    CX
        push    DX
        push    SI
        push    DI
        push    DS
        push    ES

        mov     AX, DLD_DATA
        mov     DS, AX
        mov     ES, AX

        call    Isr_Timeout_

        pop     ES
        pop     DS
        pop     DI
        pop     SI
        pop     DX
        pop     CX
        pop     BX
        pop     AX
        iret
serve_int_timeout       endp

serve_int7_isr  proc    far
        push    AX
        push    BX
        push    CX
        push    DX
        push    SI
        push    DI
        push    DS
        push    ES

        mov     AX, DLD_DATA
        mov     DS, AX
        mov     ES, AX

        call    Isr7_

        pop     ES
        pop     DS
        pop     DI
        pop     SI
        pop     DX
        pop     CX
        pop     BX
        pop     AX
```

231421-71

```
/PCO/USR/CHUCK/CSRC/ASSY.ASM

        iret
serve_int7_isr  endp

serve_int6_isr  proc    far
        push    AX
        push    BX
        push    CX
        push    DX
        push    SI
        push    DI
        push    DS
        push    ES

        mov     AX, DLD_DATA
        mov     DS, AX
        mov     ES, AX

        call    Isr6_

        pop     ES
        pop     DS
        pop     DI
        pop     SI
        pop     DX
        pop     CX
        pop     BX
        pop     AX
        iret
serve_int6_isr  endp

serve_int5_isr  proc    far
        push    AX
        push    BX
        push    CX
        push    DX
        push    SI
        push    DI
        push    DS
        push    ES

        mov     AX, DLD_DATA
        mov     DS, AX
        mov     ES, AX

        call    Isr5_

        pop     ES
        pop     DS
        pop     DI
        pop     SI
        pop     DX
        pop     CX
        pop     BX
        pop     AX
        iret
serve_int5_isr  endp
```

231421-72

```
/PCO/USR/CHUCK/CSRC/ASSY.ASM

serve_int2_isr  proc    far
        push    AX
        push    BX
        push    CX
        push    DX
        push    SI
        push    DI
        push    DS
        push    ES

        mov     AX, UAP_DATA
        mov     DS, AX
        mov     ES, AX

        call    Isr2_

        pop     ES
        pop     DS
        pop     DI
        pop     SI
        pop     DX
        pop     CX
        pop     BX
        pop     AX
        iret
serve_int2_isr  endp

serve_int1_isr  proc    far
        push    AX
        push    BX
        push    CX
        push    DX
        push    SI
        push    DI
        push    DS
        push    ES

        mov     AX, DLD_DATA
        mov     DS, AX
        mov     ES, AX

        call    Isr1_

        pop     ES
        pop     DS
        pop     DI
        pop     SI
        pop     DX
        pop     CX
        pop     BX
        pop     AX
        iret
serve_int1_isr  endp


enable_ proc    far
        sti
```

231421-73

```
/PCO/USR/CHUCK/CSRC/ASSY.ASM


        ret
enable_ endp

disable_        proc    far
        cli
        ret
disable_        endp

init_intv_      proc    far
        push    DS
        push    AX

        xor     AX, AX
        mov     DS, AX

        ; Interrupt types for the 186/51 COMMputer

        mov     DS:word ptr  80h, offset serve_int_8274         ; int 0
        mov     DS:word ptr  82h, DG_CODE
        mov     DS:word ptr  84h, offset serve_int1_isr         ; int 1
        mov     DS:word ptr  86h, DG_CODE
        mov     DS:word ptr  88h, offset serve_int2_isr         ; int 2
        mov     DS:word ptr  8Ah, DG_CODE
        mov     DS:word ptr  8Ch, offset serve_int_isr          ; int 3
        mov     DS:word ptr  8Eh, DG_CODE
        mov     DS:word ptr  90h, offset serve_int_timeout      ; int 4
        mov     DS:word ptr  92h, DG_CODE
        mov     DS:word ptr  94h, offset serve_int5_isr         ; int 5
        mov     DS:word ptr  96h, DG_CODE
        mov     DS:word ptr  98h, offset serve_int6_isr         ; int 6
        mov     DS:word ptr  9Ah, DG_CODE
        mov     DS:word ptr  9Ch, offset serve_int7_isr         ; int 7
        mov     DS:word ptr  9Eh, DG_CODE


        pop     AX
        pop     DS
        ret

init_intv_      endp

DG_CODE ends
                end             begin, ds:dld_data, ss:stack:stktop
```

231421-74

APPLICATION NOTE

AP-274

November 1986

Implementing Ethernet/Cheapernet with the Intel 82586

KIYOSHI NISHIDE
APPLICATIONS ENGINEER

Order Number: 292010-002

PREFACE

Intel's three VLSI chip set, the 82586, 82501, and 82502, is a complete solution for IEEE 802.3 10M bps LAN standards—10BASE5 (Ethernet) and 10BASE2 (Cheapernet). The 82586 is an intelligent peripheral which completely manages the processes of transmitting and receiving frames over a network under the CSMS/CD protocol. The 82586 with its on-chip four DMA channels offloads the host CPU of the tasks related to managing communication activities. The chip, for example, does not depend on the host CPU for time critical functions, such as transmissions/retransmissions and receptions of frames. The 82501 is a 10 MHz serial interface chip specially designed for the 82586. The primary function of the 82501 is to perform Manchester encoding/decoding, provide 10 MHz transmit and receive clocks to the 82586, and drive the transceiver (AUI) cable in Ethernet applications. In addition, the 82501 provides a loopback function and on-chip watchdog timer. The 82502 is a CMOS transceiver chip. The 82502 is the chip which actually drives the coaxial cable used for Ethernet or Cheapernet.

This Ap Note presents a design example of a simple but general Ethernet/Cheapernet board based on the three chip set. The board is called LANHIB (LAN High Integration Board) and uses an 80186 microprocessor as the host CPU. The LANHIB is an independent single board computer and requires only a power supply and ASCII terminal. Demo software, called TSMS (Traffic Simulator and Monitor Station) is also included in this Ap Note. The TSMS program is a network debugger and exercise tool used to exercise the 82586. In addition, flowcharts for troubleshooting are provided in order to minimize debugging time of the LANHIB board.

1.0 INTRODUCTION

A brief overview of the CSMA/CD protocol is described in Section 2. Ethernet and Cheapernet are also compared in this section.

Section 3 discusses hardware of the LANHIB in detail. This section should be helpful not only to understand the LANHIB, but also to learn in general how a system based on the three chip set can be put together. Since the 82502 involves analog circuitry, an explanation on proper layout is provided.

Demo software is presented in Section 4.0. It covers EPROM programming procedures and three sample sessions. Step by step operations at a terminal are illustrated in the figures.

Section 5 describes LANHIB troubleshooting procedures. Flowcharts are used to guide troubleshooting.

Complete LANHIB schematics and parts list are found in Appendix A. If a LANHIB is to be built, the schematics and Section 5 can be submitted to an available wire wrap facility. In parallel to board construction, Sections 3 and 4 can be studied. A factory wire wrap board for the LANHIB is offered at a discount price by Augat Corporation. Please return the enclosed card for more information.

Listing of the TSMS program and LANHIB Initialization Routine are in Appendix B. The source codes and related files are available on a diskette by returning the card enclosed in this design kit or through Insite (Intel's Software Index and Technology Exchange Library).

2.0 ETHERNET/CHEAPERNET OVERVIEW

2.1 CSMA/CD

Carrier Sense Multiple Access with Collision Detection (CSMA/CD) is a simple and efficient means of determining how a station transmits information over common medium that is shared with other stations. CSMA/CD is the access method defined by the IEEE 802.3 standard.

Carrier Sense (CS) means that any station wishing to transmit "listens" first. When the channel is busy (i.e., some other station is transmitting) the station waits (defers) until the channel is clear before transmitting ("listen before talk").

Multiple Access (MA) means that any stations wishing to transmit can do so. No central controller is needed to decide who is able to transmit and in what order.

Collision Detection (CD) means that when the channel is idle (no other station is transmitting) a station can start transmitting. It is, however, possible for two or more stations to start transmitting simultaneously causing a "collision". In the event of a collision, the transmitting stations will continue transmitting for a fixed time to ensure that all transmitting stations detect the collision. This is known as jamming. After the jam, the stations stop transmitting and wait a random period of time before retrying. The range of random wait times increases with the number of successive collisions such that collisions can be resolved even if a large number of stations are colliding.

There are three significant advantages to the CSMA/CD protocol. The first and foremost is that CSMA/CD is a proven technology. One CSMA/CD network, Ethernet, has been used by Xerox since 1975. Ethernet is so well understood and accepted that IEEE adopted

it (with minor changes) as the IEEE 802.3 10Base5 (10 Mbps, Baseband, 500 meters per segment) standard. Reliability is the second advantage to the 802.3 protocol. This media access method enables the network to operate without central control or switching. Thus, if a single station malfunctions, the rest of the network can continue operation. Finally, since CSMA/CD networks are passive and distributed in nature, they allow for easy expansion. New nodes can be added at any time without reinitializing the entire network.

2.2 Ethernet and Cheapernet

The IEEE 802.3 Type 10BASE5 standard (Ethernet) has gained wide acceptance by both large and small corporations as a high speed (10 Mbps) Local Area Network. The Ethernet channel is a low noise, shielded 50Ω coaxial cable over which information is transmitted at 10 million bits per second. Each segment of cable can be up to 500 meters in length and can be connected to longer network lengths using repeaters. Repeaters regenerate the signal from one cable segment onto another. At each end of a cable segment a terminator is attached. This passive device provides the proper electrical termination to eliminate reflections. The transceiver transmits and receives signals on the coaxial cable. In addition, it isolates the node from the channel so that a failure within the node will not affect the rest of the network. The transceiver is also responsible for detecting collisions—simultaneous transmissions by two or more stations. Ethernet transceivers are connected to the network coaxial cable using a simple tap, and to the station it serves via the transceiver cable which can be

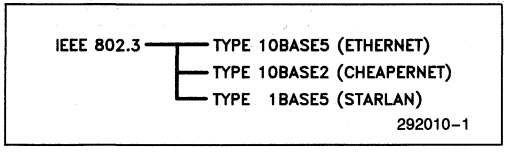


Figure 1. Different Implementations of IEEE 802.3 (Note: "10BASE5", for example, implies 10 Mbps, Baseband, and 500 meters span.)

up to 50 meters in length. The transceiver cable is made of four individually shielded twisted pairs of wires. An Ethernet interface at a computer (DTE), which includes a serial interface and data link controller, provides the connection to the user or server station. It also performs frame manipulation, addressing, detecting transmission errors, network link management, and encoding and decoding of the data to and from the transceiver.

The IEEE 802.3 Type 10BASE2 (Cheapernet) has the same functional and electrical specifications as Type 10BASE5 (Ethernet) with only two exceptions in physical (or rather mechanical) characteristics. Cheapernet is as shown in Figure 1 just a different implementation of the IEEE standard. Ethernet and Cheapernet are both 10 million bits/second CSMA/CD LANs and use the identical network parameters, such as slot time = 51.2 μs. Ethernet and Cheapernet can, therefore, be built by the same VLSI components with the same software (Figure 2).

The two physical differences attribute to the cost reduction purpose of Cheapernet—cheaper implementation of Ethernet. First, the cable used in Cheapernet may be a lower cost 50Ω coaxial cable than the one for Ethernet. The most common coaxial cable for Cheapernet is RG58 which cost about $0.15/ft. A typical Ethernet cable costs about $0.83/ft.

Second, the transceiver is integrated into the DTE in Cheapernet. The coaxial cable physically comes to the DTE, connects to the transceiver within the DTE, and goes to the next DTE (see Figure 3). The kind of connector used at the DTE is an off-the-shelf BNC "T" connector. Topology is, therefore, a simple daisy chaining. This cabling scheme contributes to further cost reduction due to omission of the Transceiver (AUI) Cable, cheaper connectors, and easier installation. The Ethernet transceiver cable costs about $1.49/ft. More flexible thin coaxial cables and familiar BNC "T" connectors are making Cheapernet a user installable Ethernet compatible network.

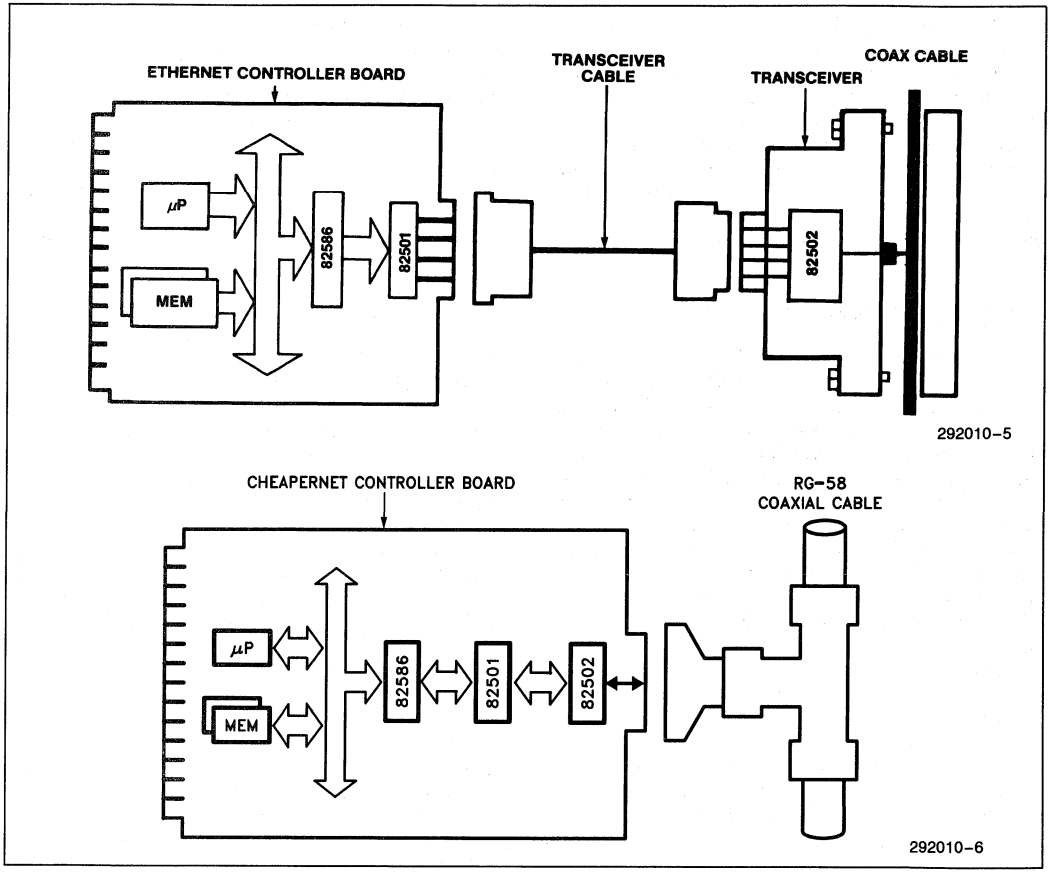


Figure 2. 82586/82501/82502 in Ethernet and Cheapernet

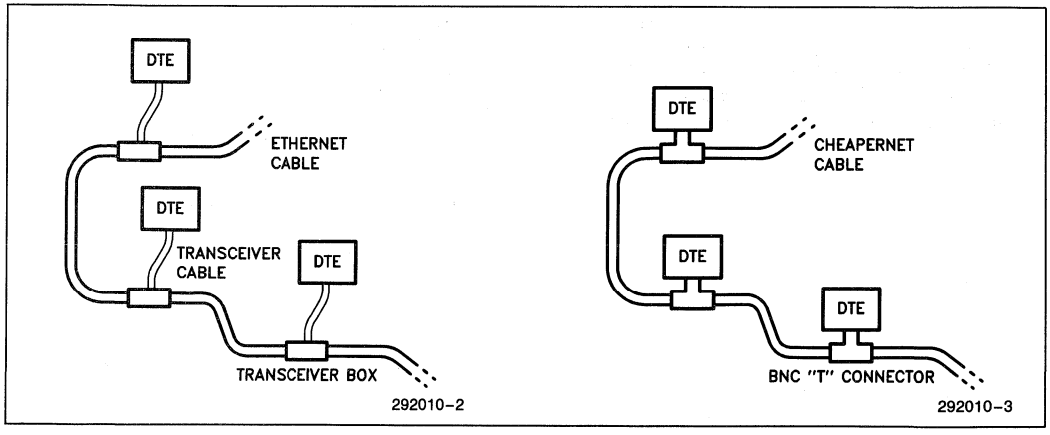


Figure 3. Ethernet Cabling vs Cheapernet Cabling

Table 1. Differences between Ethernet and Cheapernet

	Ethernet (10BASE5)	Cheapernet (10BASE2)
Data Rate	10 M bits/sec.	10 M bits/sec.
Baseband or Broadband	Baseband (Manchester)	Baseband (Manchester)
Cable Length per Segment	500m	185m
Nodes per Segment	100	30
Node Spacing	2.5m	0.5m
Cable Type	0.4 in diameter 50Ω Double Shielded example: Ethernet Coax.	0.2 in diameter 50Ω Single or Double Shielded example: RG 58 A/U or RG 58 C/U
Transceiver Cable	Yes, up to 50m	No, not needed
Capacitance per node	4 pF	8 pF
Typical Connector	Clamp-on Tap Connector or Type N Plug Connector	BNC Female Connector

Because of the lower quality cables and connectors used in Cheapernet, there are some drawbacks. The maximum distance for one Cheapernet cable segment is only 185m (600 feet), as compared to 500m (1640 feet) for Ethernet. The maximum number of nodes allowed for one Cheapernet cable segment is 30. Ethernet on the other hand allows a maximum of 100 nodes per segment. A BNC "T" connector used in Cheapernet introduces more electrical discontinuity on the transmission line than the clamp-on tap connector widely used for Ethernet. The maximum capacitance load allowed at a Cheapernet connection is 8 pF, while it is 4 pF for Ethernet. These differences are summarized in Table 1.0.

Since Ethernet and Cheapernet share the same functional and electrical characteristics, both may be mixed in a network as shown in Figure 4. In this hybrid Ethernet/Cheapernet network, it is important to keep the network propagation delay within 46.4 μs. The network may be expanded as required within this round trip propagation delay limit. Ethernet, for example, may serve as a backbone for Cheapernet in a hybrid Ethernet/Cheapernet network.

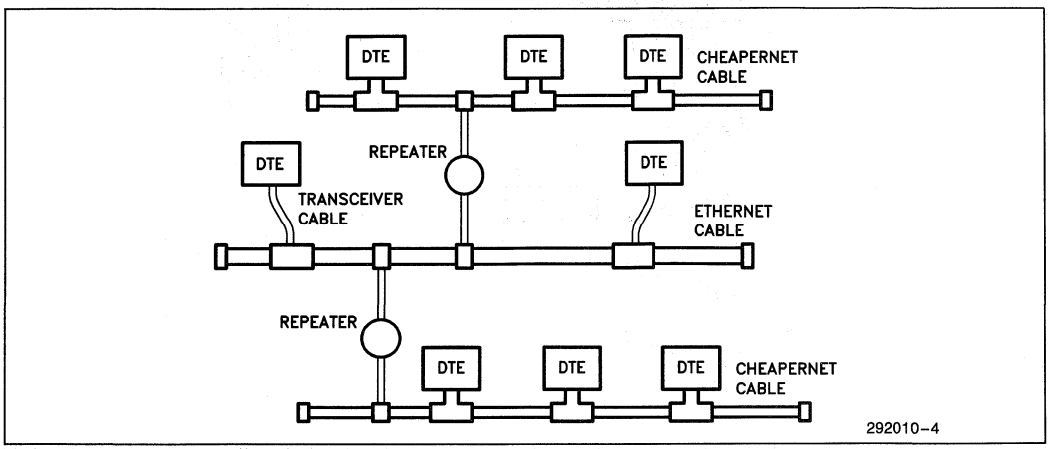


Figure 4. Ethernet/Cheapernet Hybrid Network

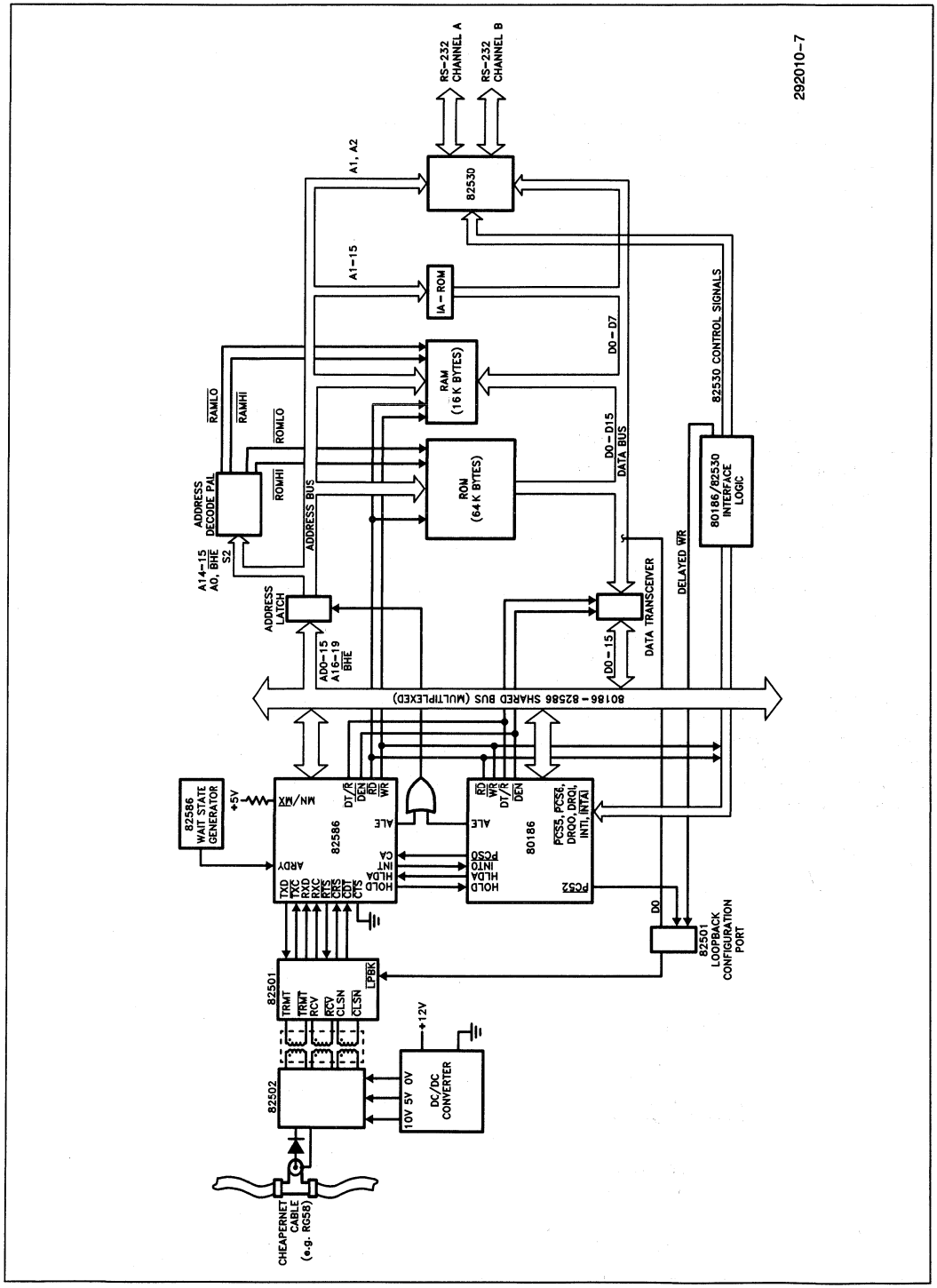


Figure 5. LANHIB Block Diagram

3.0 ETHERNET/CHEAPERNET NODE DESIGN

Details on LAN High Integration Board (LANHIB) design are presented in this section. The LANHIB is an 82586/80186 shared bus board and can be configured to Ethernet or Cheapernet. The 82586 is used in minimum mode to reduce chip count.

The reader is advised to refer to the 80186, 82586, 82501, and 82502 data sheets. Basic understanding of the 80186 microprocessor is assumed. Figure 5 shows the block diagram of the LANHIB. Schematics are in Appendix A.

3.1 82586 (Min Mode) Interface to the 80186

The 82586 can be placed in minimum mode by strapping the MN/$\overline{MX}$ pin to V_{CC}. In the minimum mode, the chip directly provides all bus control signals—ALE, $\overline{RD}$, $\overline{WR}$, DT/$\overline{R}$, and $\overline{DEN}$, saving the 8288 Bus Controller. The 80186, which is the only other bus master on the shared bus, also generates these bus control signals directly. The HOLDs and HLDAs of these two chips are connected together so that only one of the two bus masters can exclusively drive the bus at a time under the HOLD/HLDA protocol. Except for the ALE, all bus signals including address and data lines float when the chip does not have control of the bus. In this design example, $\overline{RD}$s, $\overline{WR}$s, DT/$\overline{R}$ and $\overline{DEN}$ from the two chips are connected together respectively. ALEs from the two chips are connected to an OR-gate to generate a system ALE. Multiplexed address data lines AD0–AD15 and address lines A15–A19 of the two chips are also connected line by line correspondingly.

3.2 82586 Address Latch Interface

Figure 6 shows the timing of the address signals with respect to the ALE signal. The ALE of the 82586 is OR-ed with the ALE of the 80186 and the result is connected to the latch enable inputs of Octal Transceiver Latches. The latches transfer the input data to the output as long as the latch enable is high, and captures the input data into the latch when the latch enable goes low. In this timing diagram, the setup and hold times of the input data (82586 address) required by the address latch can be verified. Estimating 7 ns of propagation delay in the 74S32, the setup time is T38 + 7, which is 32 ns at 8 MHz. The hold time for A19 is shorter than the other address lines because it is valid only during T1. The hold time for the A19 is T4 − T36 − 7, which is 3 ns. The hold time for the other address lines is T39 − 7, which is 38 ns. In this design, a 74F373 was chosen to latch address lines A16–A19 and two 74LS373s were used to latch address lines AD0–AD15. Required setup and hold times of the 74F and 74LS 373s are summerized in Table 2.

Note that address lines A16–A18 and $\overline{BHE}$ of the 82586 are not really needed to be latched. These lines stay valid for an entire memory cycle.

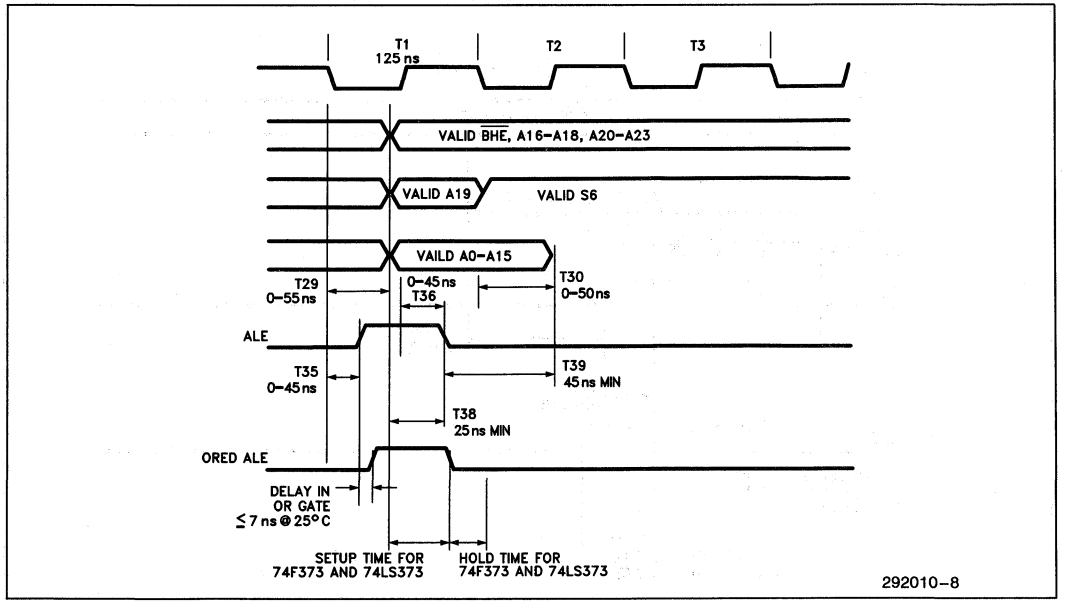


Figure 6. 82586 Address Timing

Table 2. 74F and 74LS Data Setup and Hold Time Specifications at 25°C

	74F373			74LS373			Unit
	Min	Nom	Max	Min	Nom	Max	
Data Setup Time	2↓			5↓			ns
Data Hold Time	3↓			20↓			ns

3.3 80186 Address Latch Interface

The address latch used by the 82586 is shared by the 80186. Figure 7 shows the 80186 address line timing with respect to the ALE. Again estimating 7 ns delay in the 74S32, the setup time for the latch is TAVAL + 7 and the hold time is TLLAX − 7. These are 37 ns and 23 ns respectively at 8 MHz. Comparing to the required values shown in Table 2, it is quite obvious that the setup and hold times of the latch are met by wide margins. Note that the 80186's address lines A16–A18 and $\overline{BHE}$ are not valid for an entire memory cycle; therefore, they have to be latched.

3.4 82586 Memory Interface

The 74LS373 has a delay of 18 ns for input data to reach the output assuming the latch enable is high. A demultiplexed valid address (output of the address latch), therefore, becomes available after T29 + 18 measuring from the beginning of T1 (Figure 8). The demultiplexed address remains valid until the ALE of the next memory access becomes active. Upper address lines, A14 through A20, are connected to a 16L8 PAL, which provides address decode logic for all memory devices. The PAL truth table is in Appendix A. The PAL has a maximum of 35 ns propagation delay, so chip selects will become active after 55 + 18 + 35 ns (max.) from the beginning of T1 as indicated in Figure 8. Since address decode logic is implemented by a PAL, any memory expansion would only require a reprogramming of this PAL.

Two 74LS245 bus transceiver chips are controlled by the DT/$\overline{R}$ and $\overline{DEN}$. Output enable and disable times of the 74LS245 are 40 and 25 ns respectively. The maximum propagation delay when the output enable is active is 12 ns.

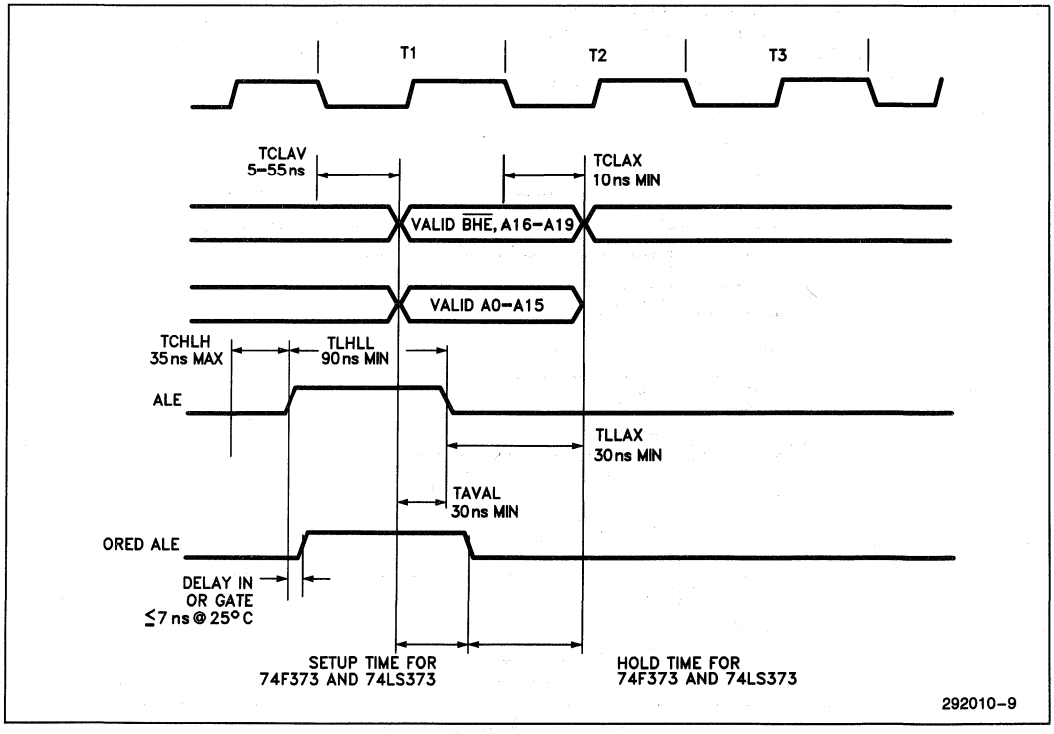


Figure 7. 80186 Address Timing

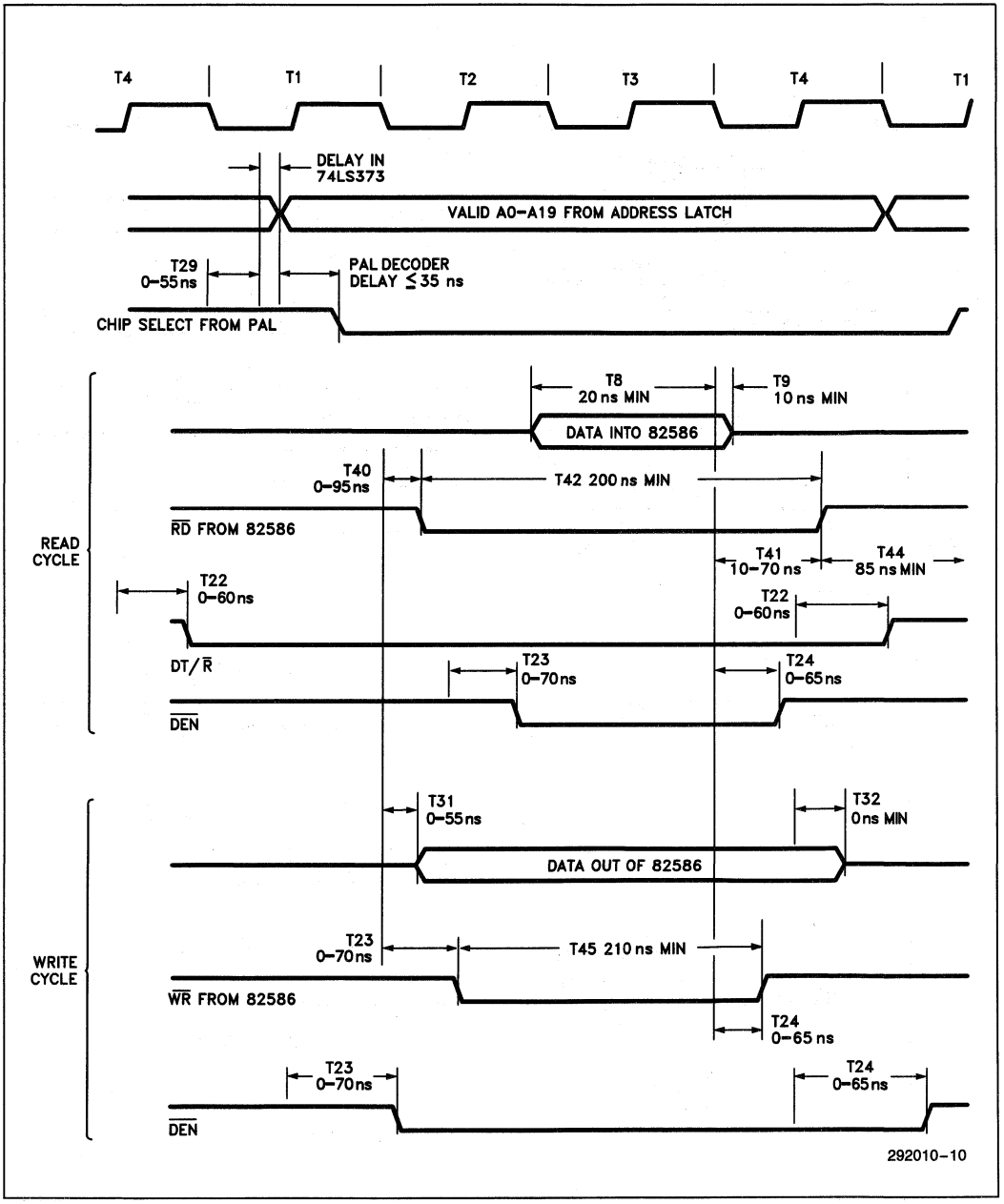


Figure 8. 82586 Memory Interface Timing

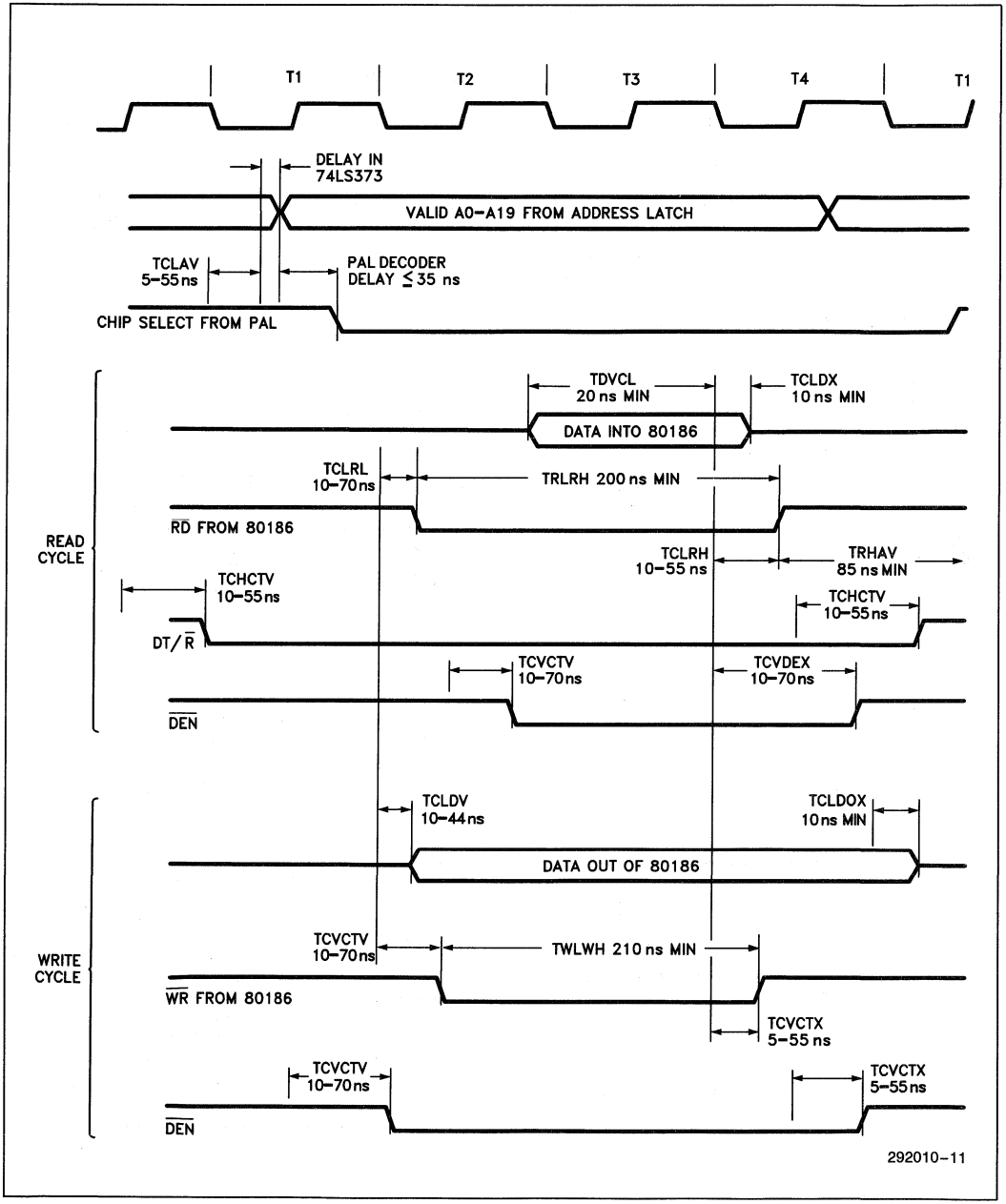


Figure 9. 80186 Memory Interface Timing

Address access time is 3 × T1 − T29 − 18 − T8 − 12 + n × T1, where n is the number of wait states. For 0 wait states operation at 8 MHz, it is 270 ns minimum. Chip select access time is 3 × T1 − T29 − 18 − T8 − 12 + n × T1 − 35, which is 235 ns for 0 wait state operation. Command access time for a read cycle is 2 × T1 − T40 − T8 − 12 + n × T1, which is 123 ns. Address setup time for a write cycle is T1 − T29 − 18 + T23, which is 52 ns minimum.

To meet these timing requirements, 2764-20s must be used for ROM. Static RAM chips, HM6264P-15, offer very wide timing margins and were selected for this design.

3.5 80186 Memory Interface

Figure 9 shows the timing of the 80186 memory interface. By comparing this figure to Figure 7, it is easy to notice that the 80186 offers a little faster bus interface. For example, TCLRL which is equivalent to T40 (0 to 95 ns) of the 82586 is specified as 10 to 70 ns. Since the memory choice satisfies the 82586 memory timing parameters, it also satisfies the 80186 memory timing parameters.

3.6 Memory Map

With 2764-20 EPROMs and 6264P-15 SRAMs, this board has 32 K bytes of ROM space and 16 K bytes of RAM space. Memory map is given in Figure 10. If 27128-20 EPROMs are used, the ROM space becomes 64 K bytes.

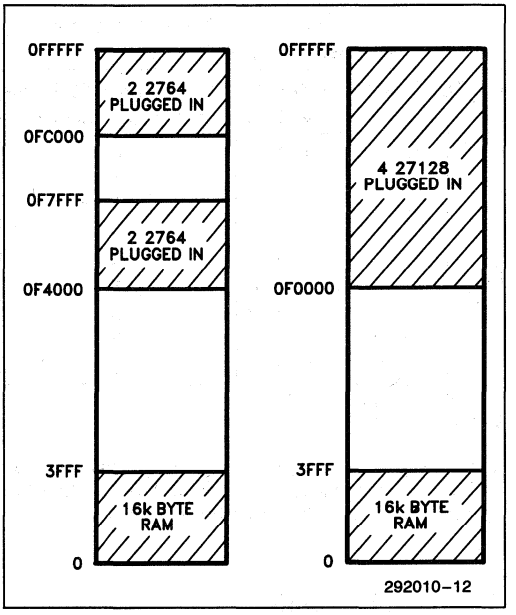


Figure 10. LANHIB Memory Map

3.7 80186 I/O Interface

3.7.1 82586 CHANNEL ATTENTION GENERATION

The active low Peripheral Chip Select 0 ($\overline{PCS0}$) was used to generate a channel attention (CA) signal to the 82586. This way of CA generation satisfies the requirement that the width of a CA which must be wider than a clock period of the system clock.

3.7.2 82586 HARDWARE RESET PORT

$\overline{PCS1}$ of the 80186 will reset the 82586 if any I/O command is executed using this I/O chip select.

3.7.3 82530 INTERFACE

82530 interface to the 80186 was derived from the design example presented in the 82530 SCC-80186 Interface Ap Brief. This document is attached to this Ap Note as Appendix C.

3.7.4 82501 LOOPBACK CONFIGURATION PORT

A 74LS74 D-type flip flop was used for this port. On power up, it configures the 82501 to Non-Loopback mode by providing a high level to pin 3 ($\overline{LOOPBACK}$). The chip select is generated from the 80186's $\overline{PCS2}$ and the sychronized $\overline{WR}$ command of the 82530 interface. The least significant bit of I/O output data becomes the state of the 82501's pin 3.

3.7.5 ON-BOARD INDIVIDUAL ADDRESS PORT

To provide the 82586 a hardware configured host address, a 32x8 ROM is connected to the bus. The chip select for this ROM is generated from the 80186's $\overline{PCS3}$, so that the address for the ROM is mapped into the I/O space. Six or two (IEEE 802.3 specified address lengths) consecutive I/O reads starting from the lowest address of ROM will transfer the board address stored in the ROM to an IA-Setup command block of the 82586.

3.8 82586 Ready Signal Generation

82586 asynchronous ready (ARDY) signal is generated from a shift register. The shift register provides the 82586 a "normally ready" signal. When a wait state is needed, the ready signal is dropped to the low state. As shown in Table 3, the 82586 can be programmed to have 0 to 8 wait states by setting the DIP switch properly. Even though the on-board memory devices are

Table 3. DIP Switch Settings for Various Numbers of 82586 Wait States

Dip Switch Setting								Number of Wait States the 82586 Inserts
7	6	5	4	3	2	1	0	
1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	0	0	2
1	1	1	1	1	0	0	0	3
1	1	1	1	0	0	0	0	4
1	1	1	0	0	0	0	0	5
1	1	0	0	0	0	0	0	6
1	0	0	0	0	0	0	0	7
0	0	0	0	0	0	0	0	8

1 = Switch Open
0 = Switch Closed

fast enough for 0 wait states operation, this programmable wait state capability was added so that the effect of wait states on the 82586 performance could be evaluated.

3.9 82501 Circuits

Since the 82501 is designed to work with the 82586, no interfacing circuits are required.

The transceiver cable side of the 82501 requires some passive components. The receive and collision differential inputs must be terminated by 78Ω ±5% resistors. Common mode voltages on these differential inputs are established internally. 240Ω ±5% pull down resistors must be connected on the TRMT and $\overline{\text{TRMT}}$ output pins.

A 0.022 μF ±10% capacitor connected between pin 1 and 2 of the 82501 is for the analog phase-locked loop.

Connected between the X1 and X2 pins is a 20 MHz parallel resonant quartz crystal (antiresonant with 20 pF load fundamental mode). An internal divide-by-two counter generates the 10 MHz clock. Since both Ethernet and Cheapernet tolerate an error of only ±0.01% in bit rate, a high quality crystal is recommended. The accuracy of a crystal should be equal to or better than ±0.002% @ 25°C and ±0.005% for 0–70°C. A 30–35 pF capacitor is connected from each crystal pin (X1 and X2) to ground in order to adjust effective capacitance load for the crystal, which should be about 20 pF including stray capacitance.

3.10 82502 Circuits

3.10.1 ISOLATION AND POWER REQUIREMENTS

The IEEE 802.3 standard requires an electrical isolation within the transceiver (MAU). Cheapernet (10BASE2) requires the isolation means to withstand 500V ac, rms for one minute. Ethernet (10BASE5) requires 250 Vrms. This electrical isolation is normally accomplished by transformer coupling of each signal pair. The kind of transformers recommended for the 82502 are the pulse transformers which have a 1:1 turn ratio and at least 50 microhenry inductance. PE64102 and PE64107 manufactured by Pulse Engineering are found to be good selections for this purpose. The PE 64102 offers 500 Vrms isolation. The PE64107 offers 2000 Vrms isolation. Both products provide three transformers in one package. Even though the current Type 10BASE5 specification requires only 250 Vrms, it is very common to have a higher isolation, at least 500 Vrms, in transceivers.

The standard specifies the voltage input level and maximum current allowed on the power pair of the transceiver cable. The voltage level may be between +11.28V dc and +15.75V dc. The maximum current is limited to 500 mA. Since the 82502 requires +10V ±10% and +5V ±10% as power, there has to be a DC/DC converter. In addition the DC/DC converter must be isolated due to the requirement described above. The DC/DC converter should be able to supply about 100 mA on the +10V line and 60 mA on the 5V line. The efficiency required in the converter is, therefore, ((11V × 100 mA + 5.5V × 60 mA) / ((11.28V − 0.5A × 4Ω) × 500 mA)) × 100 = 31% worst case. 4Ω is the maximum round trip resistance the power pair may have. 82502's CMOS process is the major contributor to this low DC/DC efficiency requirement.

Since the DC/DC converter has an isolation transformer inside, the output voltages are all floating voltages. The 0V output of the converter, for example, has no voltage relationship with the DTE's ground. The V_{SS} and AV_{SS} pins of the 82502 should be connected to the 0V output of the DC/DC converter which is the 82502's ground (reference voltage).

Both Pulse Engineering and Reliability Incorporated produce DC/DC converters that meet the 82502's requirements. The Pulse Engineering's part number is PE64369 (enclosed in this design kit). The device measures about 1.5" x 1.5" x 0.5" and provides 2000 Vrms breakdown. The Reliability's part number is 2E12R10-5. Preliminary data sheets are available from Reliability.

3.10.2 OTHER PASSIVE AND ACTIVE DEVICES FOR THE 82502

A 78Ω ±5% resistor is required to terminate the transmit pair of the Transceiver cable. The chip has an internal circuit that establishes a common mode voltage, thus no voltage divider is required. The receive and collision pair drivers need pull up resistors. A 43.2 ±1% resistor must be connected from each output pin to +5V.

A 243Ω ±0.5% precision resistor is required on the REXT pin to the ground. The accuracy of this resistor is very important since this resistor is a part of current and voltage reference circuits in the analog sections of the 82502.

Grounding the HBD (Heartbeat Disable) pin will allow the chip to perform Signal Quality Error check (Heartbeat) as required by the IEEE 802.3. The chip will transmit the collision presence signal after each transmission during Interframe Spacing (IFS) time. In a repeater application, this feature is disabled (HBD = +5V).

Diodes connected on the CXTD pin are to reduce the capacitive loading onto the coaxial cable. One diode is sufficient, but two will provide a protection in case one burns out (Short Circuit). The diode should have about 2 pF shunt capacitance at Vd = 0V and be able to handle at least 100 mA when biased in forward direction. A few candidates are 1N5282, 1N3600, and 1N4150.

A 100Ω fusible resistor connected on the CXRD pin is purely for protection. It is there as a fuse, not as a resistor. The 82502 works without this resistor. The IEEE 802.3, however, states that "component failures within the MAU (Media Attachment Unit or Transceiver) electronics should not prevent communication among other MAUs on the coaxial cable." It is recommending a transceiver design that minimizes the probability of total network failure. The fusible resistor will provide an open circuit in an event of excess current. A short circuit from the CXRD pin to ground will not bring down the network due to the blown fuse.

A 1 MΩ resistor connected between the coaxial cable shield and the Transceiver cable shield will provide a static discharge path. The Ethernet coaxial cable should also have an effective earth ground at one point in a network as required by the standard. A 0.01 μF in parallel to the 1 MΩ resistor provides ground for RF signals.

3.10.3 LAYOUT CONSIDERATION FOR THE 82502 CIRCUITS

It is strongly recommended that the board have a special ground plane for the 82502 (see Figure 11). The 0V (reference) output of the isolated DC/DC converter should be connected to the ground plane. The V_{SS} and AV_{SS} pins of the 82502 should be connected to the ground plane with minimum lead wires.

There should be a 0.22 μF capacitor connected between the coaxial cable shield and ground. The signal path from the coax. shield through the 0.22 μF capacitor to

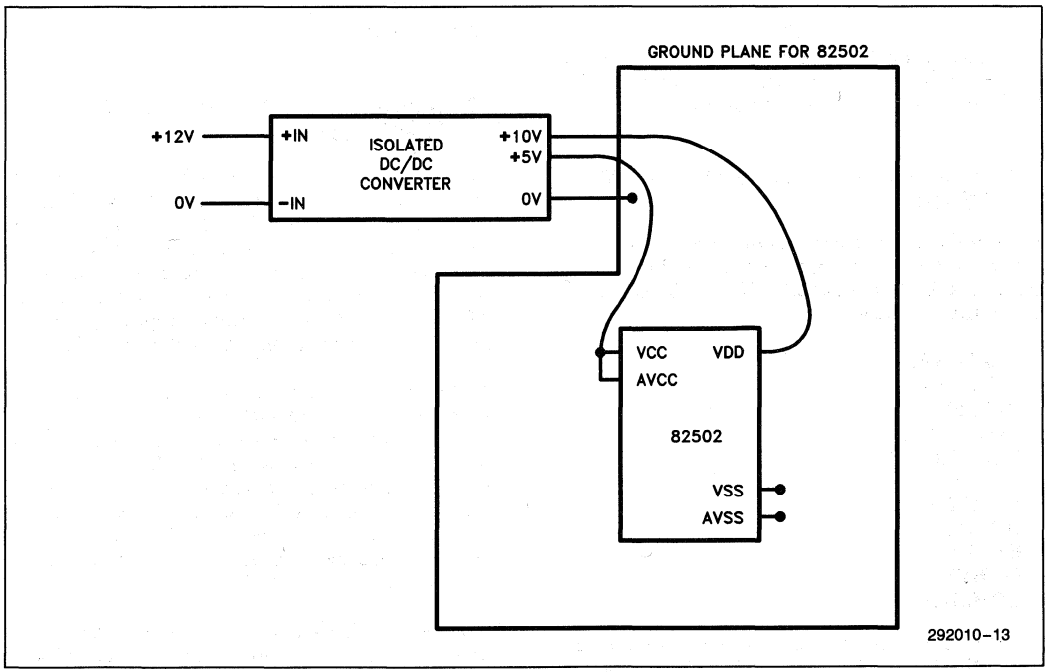


Figure 11. Ground Plane for the 82502

the ground should be kept as short as possible—leads of the 0.22 μF capacitor should be as short as possible.

The path length from the CXTD pin through two diodes to the center conductor of the coax should also be minimized.

These are recommendations which will produce a more reliable circuit if followed carefully. Remember that the 82502 has analog circuits in it.

4.0 DEMONSTRATION SOFTWARE

The demonstration software included in this Ap Note is called "Traffic Simulator and Monitor Station" (TSMS) program. The TSMS program is written in PL/M and has the following features:

1. Programmable network load generation
2. Network statistical monitoring capabilities
3. Interactive command execution of all 82586 commands
4. Interactive buffer monitoring

The environment created with the TSMS program was found to be very useful for network debugging and other individual station's hardware and software debugging. The TSMS software listing is found in Appendix B.

NOTE:
The 82586 Date Link Driver presented in AP Note 235 also runs on the LANHIB. Please refer to the Ap Note for detailed operations of the software.

4.1 Programming PROMs to Run the TSMS Program

By returning the card enclosed in this kit or by contacting Insite, the TSMS program and related batch files can be obtained on a diskette. TSMS related files that are on the diskette are:

READ.ME
TSMS.PLM
IO.PLM
INI186.PLM
LANHIB.BAT
SBC.BAT
IUPHIB.BAT
IUPSBC.BAT
HI.BYT
LO.BYT
ROM.BAT

The READ.ME file contains instructions for programming PROMs. HI.BYT and LO.BYT are the files which can be downloaded to PROMs directly. These files are already configured for the LANHIB. The batch file ROM.BAT invokes the Intel PROM Programming Software (iPPS) under the DOS operation system and programs two 2764 EPROMs. The Intel Universal Programmer must be placed in ON-LINE mode.

Other files contained in the diskette are for compiling and locating the original TSMS program. Using these files, the original TSMS program can be changed or can be compiled for an iSBC 186/51. 'TSMS.PLM' is the original TSMS source program. 'IO.PLM' contains the IO driver needed when the TSMS program is run on the iSBC 186/51. INI186.PLM is the LANHIB initialization routine. LANHIB.BAT is the batch file that compiles, links, and locates the TSMS program and the LANHIB initialization routine. SBC.BAT compiles, links, and locates the TSMS program and the IO driver for the iSBC 186/51. IUPHIB.BAT programs two 2764s for the LANHIB. IUPSBC.BAT programs two 2764s for the iSBC 186/51.

Therefore, if the TSMS program is to be run on the LANHIB (Demo board), steps required are:

1. C:>LANHIB
2. C:>IUPHIB

If the TSMS program is to be run on the iSBC 186/51, steps required are:

1. C:>SBC
2. C:>IUPSBC

4.2 Capabilities and Limits of the TSMS Program

The TSMS program initializes the LANHIB Ethernet/Cheapernet station by executing 82586's Diagnose, Configure, IA-Setup, and MC-Setup commands. The program asks a series of questions in order to set up a linked list of these 82586 commands. After initialization is completed, the program automatically starts the 82586's Receive Unit (monitoring capability). Transmissions are optional (traffic simulation capability).

The TSMS program has two modes of operation: Continuous mode and Interactive Command Execution mode. The program automatically gets into the Continuous mode after initialization. The Interactive Command Execution mode can be entered from the Continuous mode. Once entered in the Continuous mode, the software uses the format shown in Figure 12 to display information. Detailed description of each of these fields is as follows:

Host Address: host (station) address used in the most recently prepared IA-Setup command. The software simply writes the address stored in the IA-Setup command block with its least significant bit being in the most right position. Note that if the IA-Setup com-

```
**************************** Station Configuration ************************

 Host Address: 00  AA  00  00  18  6D
 Multicast Address(es): No Multicast Addresses Defined
 Destination Address: FF  FF  FF  FF  FF  FF
 Frame Length: 118 bytes
 Time Interval between Transmit Frames:  159.4 microseconds
 Network Percent Load generated by this station: 35.7 %
 Transmit Frame Terminal Count: Not Defined
 82586 Configuration Block: 08  00  26  00  60  00  F2  00  00  40

**************************** Station Activities ***************************

 # of Good     # of Good     CRC          Alignment    No           Receive
 Frames        Frames        Errors       Errors       Resource     Overrun
 Transmitted   Received                                 Errors       Errors
 10130         0             0            0            0            0
                                                                    292010-14
```

Figure 12. Continuous Mode Display

mand was just set up and not executed, the address displayed in this field may not be the address stored in the 82586.

Multicast Address(es): multicast addresses used in the most recently prepared MC-Setup command. As in the case of host address, the software simply writes the addresses stored in the MC-Setup command block. Note that if the MC-Setup command was just set up and not executed, the addresses displayed in this field may not be the addresses stored in the 82586.

Destination Address: destination address stored in the transmit command block if AL-LOC=0. If AL-LOC=1, destination address is picked up from the transmit buffer. The least significant bit is in the most right position.

Frame Length: transmit frame byte count including destination address, source address, length, data, and CRC field.

Time Interval Between Transmit Frames: approximate time interval obtainable between transmit frames (Figure 13). The number is correct if there are no other stations transmitting on the network.

Network Percent Load Generated by This Station: approximate network percent load that is generated by this station (Figure 13). The number is correct if there are no other stations transmitting on the network.

Transmit Frame Terminal Count: number of frames this station will transmit before it stops network traffic load generation. If this station is transmitting indefinitely, this field will be 'Not Defined'.

82586 Configuration Block: configuration parameters used in the most recently prepared Configure command. As in the case of IA-Setup command, the software simply writes the parameters from the Configure command block. The least significant byte (FIFO Limit) of the configuration parameters is printed in the most left position.

# of Good Frames Transmitted: number of good frames transmitted. This is a snap shot of the 32-bit transmit frame counter. It is incremented only when both C and OK bits of the transmit command status are set after an execution. The counter is 32-bit wide.

# of Good Frames Received: number of good frames received. This is a snap shot of the 32-bit receive frame counter. It is incremented only when both C and OK bits of a receive frame descriptor status are set after a reception. The counter is 32-bit wide.

CRC Errors: number of frames that had a CRC error. This is a snap shot of the 16-bit CRC counter maintained by the 82586 in the SCB.

Alignment Errors: number of frames that had an alignment error. This is a snap shot of the 16-bit alignment counter maintained by the 82586 in the SCB.

No Resource Errors: number of frames that had a no resource error. This is a snap shot of the 16-bit no resource counter maintained by the 82586 in the SCB.

Receive Overrun Errors: number of frames that had a receive overrun error. This is a snap shot of the 16-bit receive overrun error counter maintained by the 82586 in the SCB.

If the station is actively transmitting, # of good frames transmitted should be incrementing. If the station is actively receiving frames, # of good frames received should be incrementing. In this continuous mode, a user can see the activities of the network.

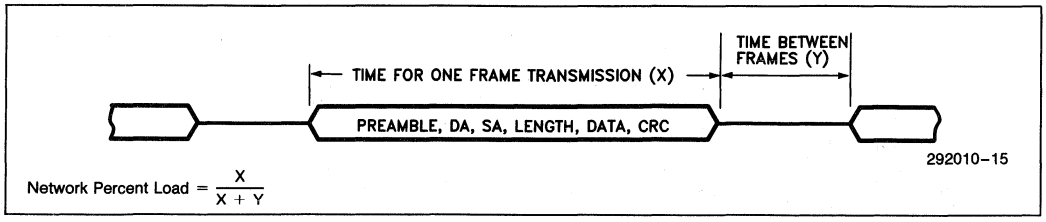


Figure 13. Network Percent Load

Hitting any key on the keyboard while the program is running in the Continuous mode will exit the mode. The program will respond with a message 'Enter Command (H for Help) → '. In this Interactive Command Execution mode, a user can set up any one of the 82586 action commands and/or execute any one of the 82586 SCB control commands. Setting up a Dump command and executing a SCB Command Unit Start command will, for example, execute the Dump command. Display commands are also available to see the contents of the 82586's data structure blocks. A display command will enable a user to see the contents of the 82586's dump (see Section 6.3).

Typing 'E' after 'Enter command (H for help) → ', executing a SCB Command Unit Start command with a transmit command, or executing a SCB Receive Unit Start command will exit the Interactive Command Execution mode. The program will be back in the Continuous mode. Using this Interactive Command Execution mode, one can, for example, reconfigure the station and come back to the Continuous mode. Section 6 lists actual example executions of the TSMS program.

The TSMS program should be run in an 8 MHz system. The software running at 8 MHz with a maximum of 2 wait states has been tested and verified to be able to receive back-to-back frames separated by 9.6 microseconds and still keep track of the correct number of frames received. This capability, for example, can be used to find out exactly how many frames a new station in the network had transmitted.

The software does not perform extensive loopback tests and hardware diagnostics during the initialization. A loopback operation can be performed interactively in the Interactive Command Execution mode.

The software allows a user to set up only 8 multicast addresses maximum. It is not possible with this program to set up more than 8 multicast addresses.

The command chaining feature of the 82586 is not used in the Interactive Command Execution mode. Each command setup performed by a 'S' command after 'Enter command (H for help) → ' sets up a command with its EL bit set, I bit reset, and S bit reset. Diagnose, Configure, IA-Setup, and MC-Setup commands are chained together during the initialization routine and executed at once with only one CA.

The software sets up 5 Receive Frame Descriptors linked in a circular list. Therefore, a user can see only the last 5 frames the station has received. It also sets up 5 receive buffers, each being 1514 bytes long, linked in circle. Therefore, the 82586 never goes into the NO RESOURCES state.

4.3 Example Executions of the TSMS Program

This section presents three example executions of the TSMS program. When the TSMS program needs a command to be typed, it asks a question with ' → '. Anything after ' → ' is what a user needs to type in on the keyboard. To switch from the continuous mode to the interactive command execution mode, type any key on the keyboard.

4.3.1 EXAMPLE 1: EXTERNAL LOOPBACK EXECUTION

In this example, 500 external loopback transmissions and receptions are executed (Figure 14). In order for the software to process each loopback properly, a large delay was given between transmissions.

4.3.2 EXAMPLE 2: FRAME RECEPTION IN PROMISCUOUS MODE

The 82586 is configured to receive any frame that exists in the network (Figure 15). In this example, the station received 100 frames.

4.3.3 EXAMPLE 3: 35.7% NETWORK TRAFFIC LOAD GENERATION

The station is programmed to transmit 118 byte long frames with a time interval of 159.4 microseconds in between (Figure 16). The network load is about 35.7 percent if no other stations are transmitting in the network.

A key was hit to enter the Interactive Command Execution mode. In that mode, a Dump command was executed and the result was displayed. After the Dump execution, a transmit command was set up again and the station was put in the Continuous mode.

Traffic Simulator and Monitor Station Program

```
Initialization begun


Configure command is set up for default values.
Do you want to change any bytes? (Y or N) ==> Y
Enter byte number (1 - 11) ==> 4
Enter byte 4 (4H) ==> A6H
Any more bytes? (Y or N) ==> Y
Enter byte number (1 - 11) ==> 11
Enter byte 11 (BH) ==> 6
Any more bytes? (Y or N) ==> N
Configure the 586 with the prewired board address ==> N
Enter this station's address in Hex ==> 000000002200
You can enter up to 8 Multicast Addresses.
Would you like to enter a Multicast Address? (Y or N) ==> N
You entered 0 Multicast Address(es).

Would you like to transmit?
Enter a Y or N ==> Y
Enter a destination address in Hex ==> 000000002200

Enter TYPE ==> 0
How many bytes of transmit data?
Enter a number ==> 2
Transmit Data is continuous numbers (0, 1, 2, 3, ... )
Change any data bytes? (Y or N) ==> N
Enter a delay count ==> 10000000000
The number is too big.
It has to be less than or equal to 65535 (FFFFH).
Enter a number ==> 60000

Setup a transmit terminal count? (Y or N) ==> Y
Enter a transmit terminal count ==> 500

Destination Address: 00  00  00  00  22  00
Frame Length: 20 bytes
Time Interval between Transmit Frames: 30.18 miliseconds
Network Percent Load generated by this station:   .0 %
Transmit Frame Terminal Count: 500

Good enough? (Y or N) ==> Y

Receive Unit is active.
```

292010-16

Figure 14. External Loopback Execution

```
---Transmit Command Block---
0000 at 033E
8004
FFFF
034E
2200
0000
0000
0000

Hit <CR> to countinue

transmission started!




**************************** Station Configuration ************************

 Host Address: 00  00  00  00  22  00
 Multicast Address(es): No Multicast Addresses Defined
 Destination Address: 00  00  00  00  22  00
 Frame Length: 20 bytes
 Time Interval between Transmit Frames: 30.18 miliseconds
 Network Percent Load generated by this station:    .0 %
 Transmit Frame Terminal Count: 500
 82586 Configuration Block: 08  00  A6  00  60  00  F2  00  00  06

**************************** Station Activities ***************************

 # of Good     # of Good     CRC          Alignment     No           Receive
 Frames       Frames        Errors       Errors        Resource     Overrun
 Transmitted  Received                                  Errors       Errors
 500          500           0            0             0            0
```

292010-17

Figure 14. External Loopback Execution (Continued)

Traffic Simulator and Monitor Station Program

```
Initialization begun


Configure command is set up for default values.
Do you want to change any bytes? (Y or N) ==> Y
Enter byte number (1 - 11) ==> 9
Enter byte 9 (9H) ==> 1
Any more bytes? (Y or N) ==> N
Configure the 586 with the prewired board address ==> Y
You can enter up to 8 Multicast Addresses.
Would you like to enter a Multicast Address? (Y or N) ==> N
You entered 0 Multicast Address(es).

Would you like to transmit?
Enter a Y or N ==> N

Receive Unit is active.




**************************** Station Configuration ************************

Host Address: 00  AA  00  00  18  6D
Multicast Address(es): No Multicast Addresses Defined
82586 Configuration Block: 08  00  26  00  60  00  F2  01  00  40

***************************** Station Activities ***************************

# of Good     # of Good     CRC          Alignment    No           Receive
Frames        Frames        Errors       Errors       Resource     Overrun
Transmitted   Received                                 Errors       Errors
0             100           0            0            0            0
Enter command (H for help) ==> D

Command Block or Receive Area? (R or C) ==> R
Frame Descriptors:
4000 at 036C  A000 at 0382  A000 at 0398  A000 at 03AE  A000 at 03C4
0000          0000          0000          0000          0000
0382          0398          03AE          03C4          036C
03DA          03E4          03EE          03F8          0402
2200          2200          2200          2200          2200
2200          2200          2200          2200          2200
0000          0000          0000          0000          0000
```

292010-18

Figure 15. Frame Reception in Promiscuous Mode

```
0000          0000          0000          0000          0000
0000          0000          0000          0000          0000
0000          0000          0000          0000          0000
0000          0000          0000          0000          0000

 Receive Buffer Descriptors:
C064 at 03DA  C064 at 03E4  C064 at 03EE  C064 at 03F8  C064 at 0402
03E4          03EE          03F8          0402          03DA
040C          09F6          0FE0          15CA          1BB4
0000          0000          0000          0000          0000
05DC          05DC          05DC          05DC          05DC

 Display the receive buffers? (Y or N) ==> Y
 Receive Buffers:

 Receive Buffer 0 :
002C:014C 00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F
002C:015C 10  11  12  13  14  15  16  17  18  19  1A  1B  1C  1D  1E  1F
002C:016C 20  21  22  23  24  25  26  27  28  29  2A  2B  2C  2D  2E  2F
002C:017C 30  31  32  33  34  35  36  37  38  39  3A  3B  3C  3D  3E  3F
002C:018C 40  41  42  43  44  45  46  47  48  49  4A  4B  4C  4D  4E  4F
002C:019C 50  51  52  53  54  55  56  57  58  59  5A  5B  5C  5D  5E  5F
002C:01AC 60  61  62  63

Hit <CR> to countinue

 Receive Buffer 1 :
002C:0736 00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F
002C:0746 10  11  12  13  14  15  16  17  18  19  1A  1B  1C  1D  1E  1F
002C:0756 20  21  22  23  24  25  26  27  28  29  2A  2B  2C  2D  2E  2F
002C:0766 30  31  32  33  34  35  36  37  38  39  3A  3B  3C  3D  3E  3F
002C:0776 40  41  42  43  44  45  46  47  48  49  4A  4B  4C  4D  4E  4F
002C:0786 50  51  52  53  54  55  56  57  58  59  5A  5B  5C  5D  5E  5F
002C:0796 60  61  62  63

Hit <CR> to countinue

 Receive Buffer 2 :
002C:0D20 00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F
002C:0D30 10  11  12  13  14  15  16  17  18  19  1A  1B  1C  1D  1E  1F
002C:0D40 20  21  22  23  24  25  26  27  28  29  2A  2B  2C  2D  2E  2F
002C:0D50 30  31  32  33  34  35  36  37  38  39  3A  3B  3C  3D  3E  3F
002C:0D60 40  41  42  43  44  45  46  47  48  49  4A  4B  4C  4D  4E  4F
002C:0D70 50  51  52  53  54  55  56  57  58  59  5A  5B  5C  5D  5E  5F
002C:0D80 60  61  62  63

Hit <CR> to countinue

 Receive Buffer 3 :
002C:130A 00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F
002C:131A 10  11  12  13  14  15  16  17  18  19  1A  1B  1C  1D  1E  1F
002C:132A 20  21  22  23  24  25  26  27  28  29  2A  2B  2C  2D  2E  2F
002C:133A 30  31  32  33  34  35  36  37  38  39  3A  3B  3C  3D  3E  3F
002C:134A 40  41  42  43  44  45  46  47  48  49  4A  4B  4C  4D  4E  4F
002C:135A 50  51  52  53  54  55  56  57  58  59  5A  5B  5C  5D  5E  5F
002C:136A 60  61  62  63

Hit <CR> to countinue
```

292010-19

Figure 15. Frame Reception in Promiscuous Mode (Continued)

```
 Receive Buffer 4 :
002C:18F4 00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F
002C:1904 10  11  12  13  14  15  16  17  18  19  1A  1B  1C  1D  1E  1F
002C:1914 20  21  22  23  24  25  26  27  28  29  2A  2B  2C  2D  2E  2F
002C:1924 30  31  32  33  34  35  36  37  38  39  3A  3B  3C  3D  3E  3F
002C:1934 40  41  42  43  44  45  46  47  48  49  4A  4B  4C  4D  4E  4F
002C:1944 50  51  52  53  54  55  56  57  58  59  5A  5B  5C  5D  5E  5F
002C:1954 60  61  62  63

Hit <CR> to countinue

 Enter command (H for help) ==> E




***************************** Station Cofiguration *************************

 Host Address: 00  AA  00  00  18  6D
 Multicast Address(es): No Multicast Addresses Defined
 82586 Configuration Block: 08  00  26  00  60  00  F2  01  00  40

***************************** Station Activities ***************************

 # of Good     # of Good     CRC          Alignment    No           Receive
 Frames        Frames        Errors       Errors       Resource     Overrun
 Transmitted   Received                                 Errors       Errors
 0             100           0            0            0            0
```

292010-20

Figure 15. Frame Reception in Promiscuous Mode (Continued)

Traffic Simulator and Monitor Station Program

```
 Initialization begun


 Configure command is set up for default values.
 Do you want to change any bytes? (Y or N) ==> N
 Configure the 586 with the prewired board address ==> Y
 You can enter up to 8 Multicast Addresses.
 Would you like to enter a Multicast Address? (Y or N) ==> N
 You entered 0 Multicast Address(es).

 Would you like to transmit?
 Enter a Y or N ==> Y
 Enter a destination address in Hex ==> FFFFFFFFFFFF

 Enter TYPE ==> 0
 How many bytes of transmit data?
 Enter a number ==> 100
 Transmit Data is continuous numbers (0, 1, 2, 3, ... )
 Change any data bytes? (Y or N) ==> N

 Enter a delay count ==> 0

 Setup a transmit terminal count? (Y or N) ==> N

 Destination Address: FF  FF  FF  FF  FF  FF
 Frame Length: 118 bytes
 Time Interval between Transmit Frames:  159.4 microseconds
 Network Percent Load generated by this station: 35.7 %
 Transmit Frame Terminal Count: Not Defined

 Good enough? (Y or N) ==> Y

 Receive Unit is active.

---Transmit Command Block---
0000 at 033E
8004
FFFF
034E
FFFF
FFFF
FFFF
0000


Hit <CR> to countinue
```

292010-21

Figure 16. 35.7% Network Load Generation

```
transmission started!


**************************** Station Configuration ************************

 Host Address: 00  AA  00  00  18  6D
 Multicast Address(es): No Multicast Addresses Defined
 Destination Address: FF  FF  FF  FF  FF  FF
 Frame Length: 118 bytes
 Time Interval between Transmit Frames:  159.4 microseconds
 Network Percent Load generated by this station: 35.7 %
 Transmit Frame Terminal Count: Not Defined
 82586 Configuration Block: 08  00  26  00  60  00  F2  00  00  40

**************************** Station Activities ***************************

 # of Good     # of Good     CRC          Alignment    No           Receive
 Frames       Frames        Errors       Errors       Resource     Overrun
 Transmitted  Received                                 Errors       Errors
 10459        0             0            0            0            0
 Enter command (H for help) ==> H

 Commands are:
 S - Setup CB              D - Display RFD/CB
 P - Print SCB             C - SCB Control CMD
 L - ESI Loopback On       N - ESI Loopback Off
 A - Toggle Number Base
 Z - Clear Tx Frame Counter
 Y - Clear Rx Frame Counter
 E - Exit to Continuous Mode


 Enter command (H for help) ==> S

 Enter command block type (H for help) ==> H
Command block type:
 N - Nop                   I - IA Setup
 C - Configure             M - MA Setup
 T - Transmit              R - TDR
 D - Diagnose              S - Dump Status
 H - Print this message

 Enter command block type (H for help) ==> S

 Enter command (H for help) ==> C

 Do you want to enter any SCB commands? (Y or N) ==> Y
 Enter CUC ==> 1
 Enter RES bit ==> 0
 Enter RUC ==> 0
 Issued Channel Attention



 Enter command (H for help) ==> D
```

292010-22

Figure 16. 35.7% Network Load Generation (Continued)

```
 Command Block or Receive Area? (R or C) ==> C
---Dump Status Command Block---
A000 at 0364
8006
FFFF
27D6

 Dump Status Results
 at 27D6
00  E8  3F  26  08  60  00  FA  00  00  40  FF  6D  18  00  00
AA  00  40  20  00  00  00  00  FF  FF  FF  FF  B5  9E  EE  CF
62  63  3F  B0  00  00  00  00  00  00  00  00  FF  85  08  FC
00  00  00  00  00  00  00  00  00  00  00  00  70  03  06  00
DC  05  00  00  0C  04  DC  05  E4  03  DA  03  DA  03  78  05
82  03  6C  03  F8  03  64  80  D6  27  E8  21  FF  FF  4E  03
06  80  FF  FF  64  03  00  00  D2  02  00  00  00  00  00  00
00  00  D6  27  00  01  00  28  00  00  00  00  30  26  00  00
20  00  40  06  30  01  00  00  90  00  10  01  00  00  6C  03
00  00  6A  03  0E  00  6C  28  00  00  74  03  00  00  00  00
00  00  00  00  00  C0  00  00  00  00

 Enter command (H for help) ==> S

 Enter command block type (H for help) ==> T
 Enter a destination address in Hex ==> FFFFFFFFFFFF

 Enter TYPE ==> 0
 How many bytes of transmit data?
 Enter a number ==> 100
 Transmit Data is continuous numbers (0, 1, 2, 3, ... )
 Change any data bytes? (Y or N) ==> N

 Enter a delay count ==> 0

 Setup a transmit terminal count? (Y or N) ==> N

 Destination Address: FF  FF  FF  FF  FF  FF
 Frame Length: 118 bytes
 Time Interval between Transmit Frames:  159.4 microseconds
 Network Percent Load generated by this station: 35.7 %
 Transmit Frame Terminal Count: Not Defined

 Good enough? (Y or N) ==> Y

 Enter command (H for help) ==> C

 Do you want to enter any SCB commands? (Y or N) ==> Y
 Enter CUC ==> 1
 Enter RES bit ==> 0
 Enter RUC ==> 0
 Issued Channel Attention
```

292010-23

Figure 16. 35.7% Network Load Generation (Continued)

```
***************************** Station Configuration ************************

 Host Address: 00  AA  00  00  18  6D
 Multicast Address(es): No Multicast Addresses Defined
 Destination Address: FF  FF  FF  FF  FF  FF
 Frame Length: 118 bytes
 Time Interval between Transmit Frames:  159.4 microseconds
 Network Percent Load generated by this station: 35.7 %
 Transmit Frame Terminal Count: Not Defined
 82586 Configuration Block: 08  00  26  00  60  00  F2  00  00  40

***************************** Station Activities ***************************

 # of Good     # of Good     CRC          Alignment    No           Receive
 Frames        Frames        Errors       Errors       Resource     Overrun
 Transmitted   Received                                 Errors       Errors
 106020        0             0            0            0            0
```

292010-24

Figure 16. 35.7% Network Load Generation (Continued)

5.0 IN CASE OF DIFFICULTY

This section presents methods of troubleshooting ("debugging") a LANHIB board. When a LANHIB board is powered up with the TSMS program stored in EPROMs, it should display "TRAFFIC SIMULATOR AND MONITOR STATION PROGRAM" message on a terminal screen. If the message is not displayed, the board has to be debugged. Section 5.1 describes basic 80186/82586 system troubleshooting procedures. Section 5.2 is for troubleshooting 82501 and 82502 circuits. After the 80186/82586 system is debugged, the 82501/82502 circuits have to be tested.

5.1 Troubleshooting 80186/82586 System

Shown in Figure 17 is a flow chart for troubleshooting 80186/82586 system. The procedure requires an oscilloscope. A logic analyzer is needed if problems appear to be serious. The procedures will debug the board to the point where the 82530 is initialized properly. If the 82530 can be initialized properly, ROM and RAM interfaces must be functioning. Board initialization routines (INI186.PLM) linked to the TSMS program requires ROM and RAM accesses. Since the 82586 shares most of the system with the 80186, no special debugging is required for the 82586. Wiring of all 82586 parallel signal pins should, however, be checked.

The flow chart branches to two major paths after the first decision box. One path debugs the RS-232 channel and the other debugs the 80186/82586 system. The waveform of the $\overline{TRXCB}$ output of the 82530 determines which path to be taken. If the 82530 is getting programmed properly, there should be 153.6 KHz ($1/f = 6.51\ \mu s$) clock on this output pin. If there is a clock, the problem is probably in the RS-232 interface. If there is no clock, then the system has to be debugged using a logic analyzer.

5.2 Troubleshooting 82501/82502 Circuits

If the TSMS program runs on the LANHIB but the 82586 is not able to transmit or receive, there must be a problem in 82501/82502 circuits. The flow chart in Figure 19 will guide troubleshooting in these circuits. An oscilloscope is required.

The board should be configured to Cheapernet and disconnected from the network. Two terminators will be required to terminate a "T" BNC connector providing an effective load resistance of 25Ω to the 82502.

The 82586 must have the system and transmit clocks running upon reset. Since the transmit clock is generated by the 82501, the 82501 transmit clock output pin (pin 16) should be checked. The TSMS program executes 82586's Diagnose, Configure, IA-Setup, and MC-Setup commands during initialization. If the 82586 has active $\overline{CRS}$ (Carrier Sense) signal, it cannot complete execution of these commands. The 82501 should, therefore, be checked if it is generating inactive $\overline{CRS}$ signal to the 82586 after power up. The LANHIB powers up the 82501 in non-loopback mode.

After making sure that the 82501 is generating proper signals to the 82586, the TSMS program is restarted with an initialization shown in Figure 20. The 82586 is configured to EXT-LPBK = 1, TONO-CRS = 1, and MIN-FRM-LEN = 6. The chip is also loaded with a destination address identical to the source address. If there are no problems in the 82501/82502 circuits, the station will be receiving its own transmitted frames. If problems exist, the station will only be transmitting. Since the 82586 is configured to TONO-CRS (Transmission On NO Carrier Sense), the chip will keep transmitting regardless of the state of carrier sense. The 82501/82502 circuits can then be probed with an oscilloscope at the locations indicated in Figure 21. Probing will catch problems like wiring mistakes, missing load resistors, etc.

Once the station is debugged, it can be connected to the network. If there is a problem in the network, the 82586's TDR command can be used to find the location and nature of the problem.

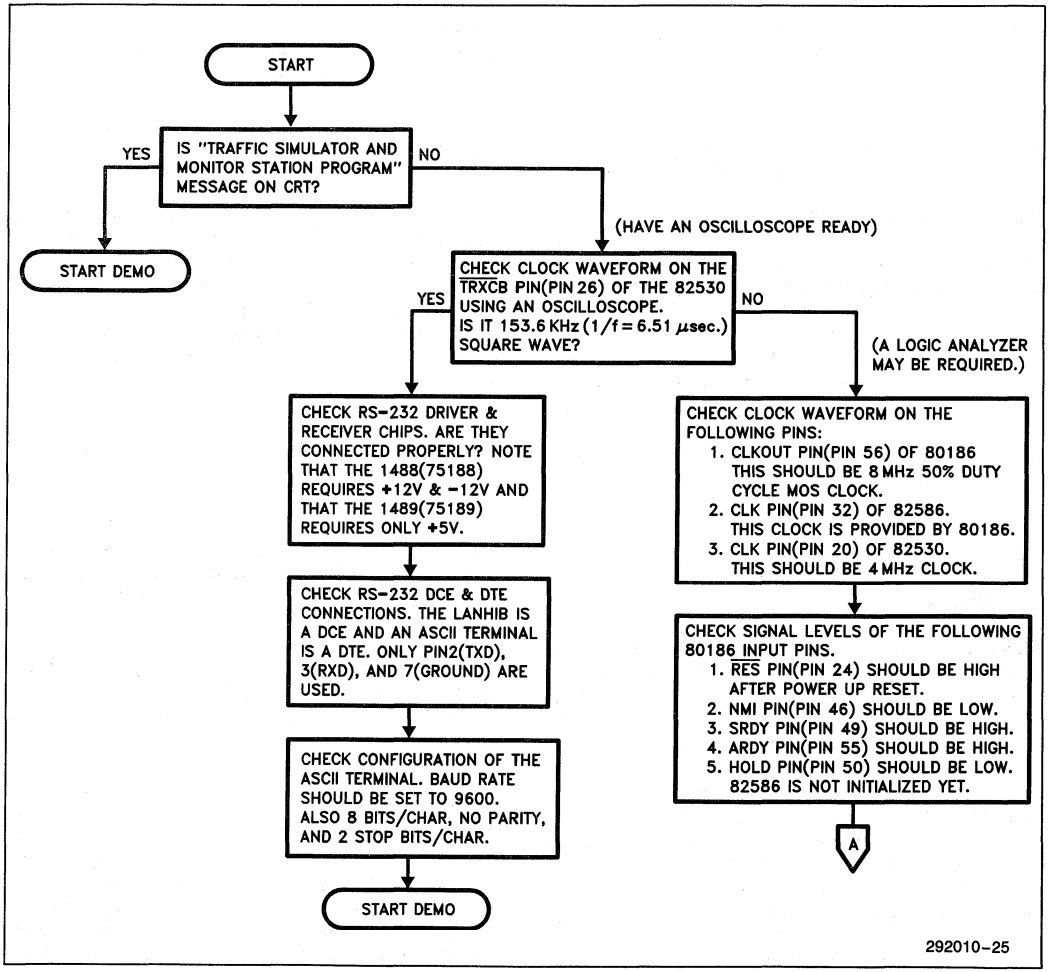


Figure 17. Flowchart for 80186/82586 System Troubleshooting

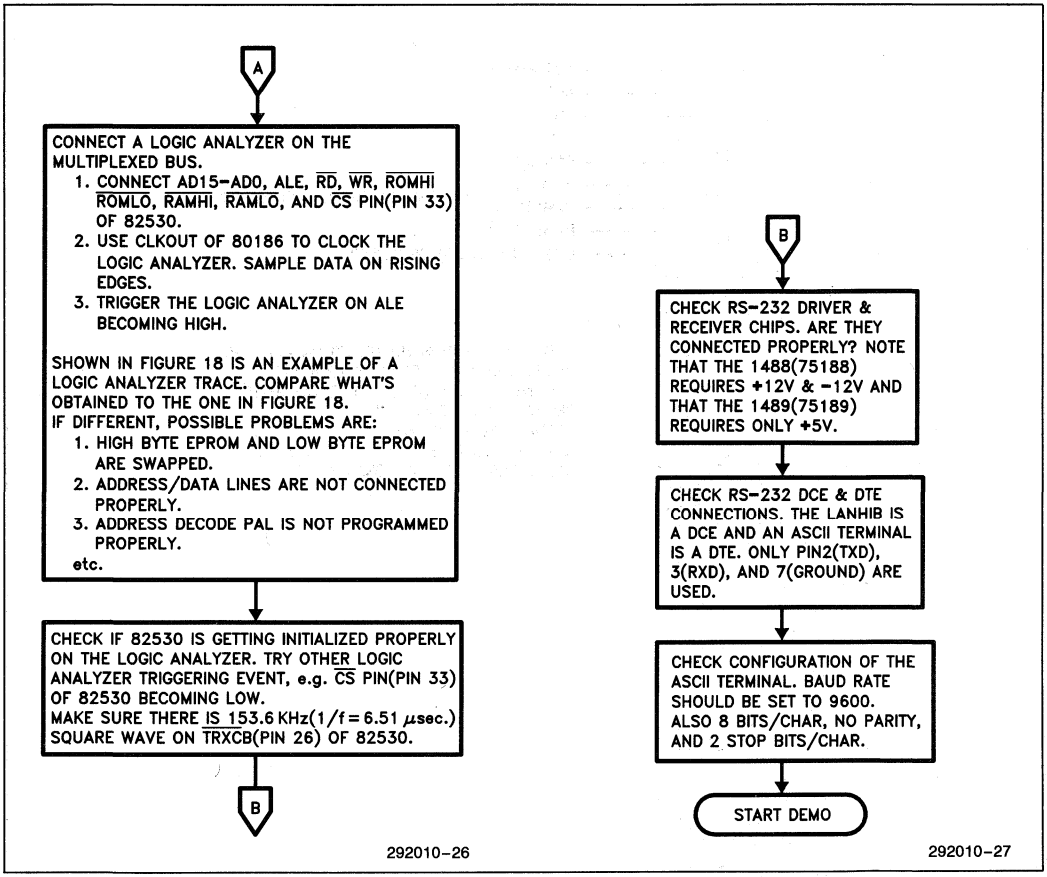


Figure 17. Flowchart for 80186/82586 System Troubleshooting (Continued)

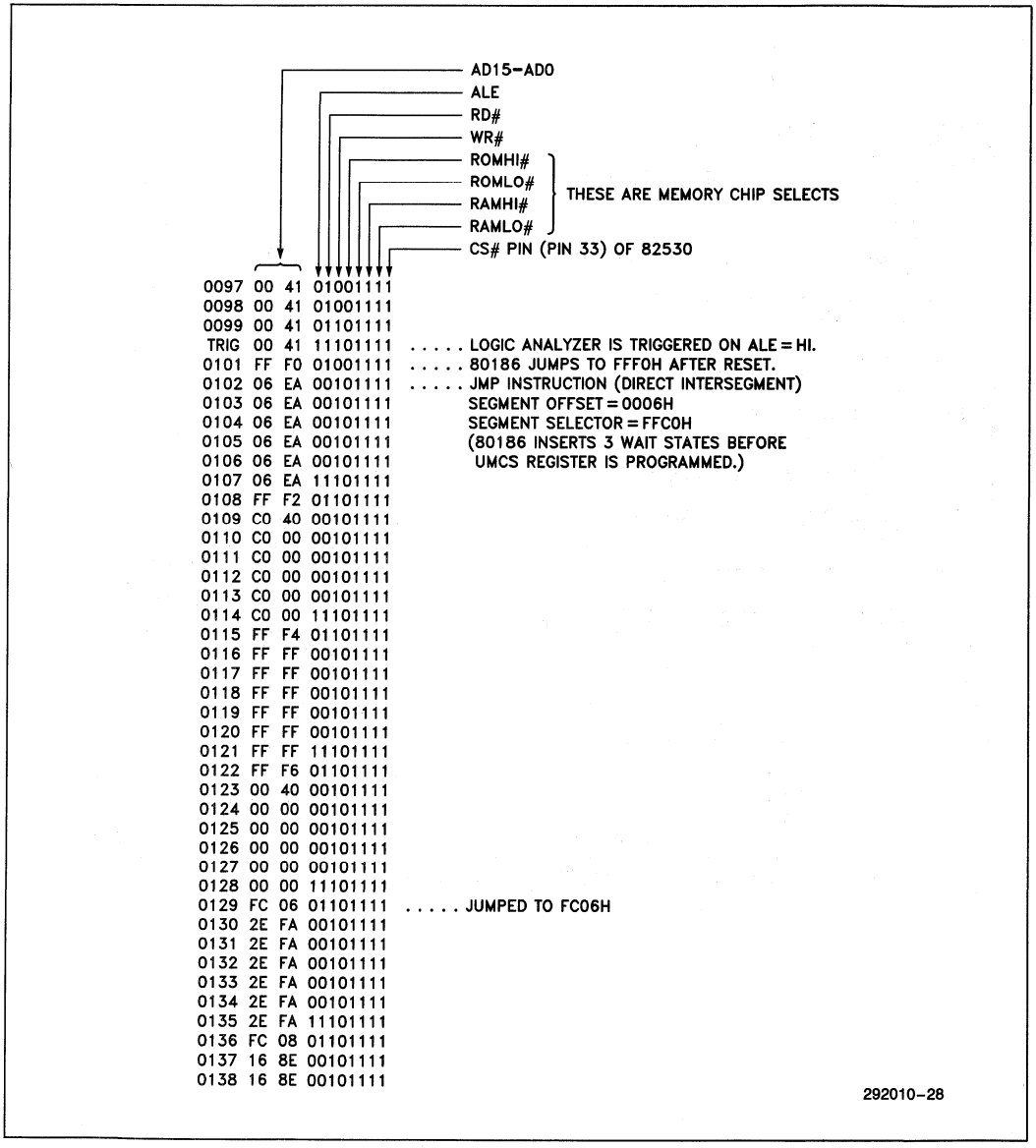


Figure 18. Example of Logic Analyzer Trace

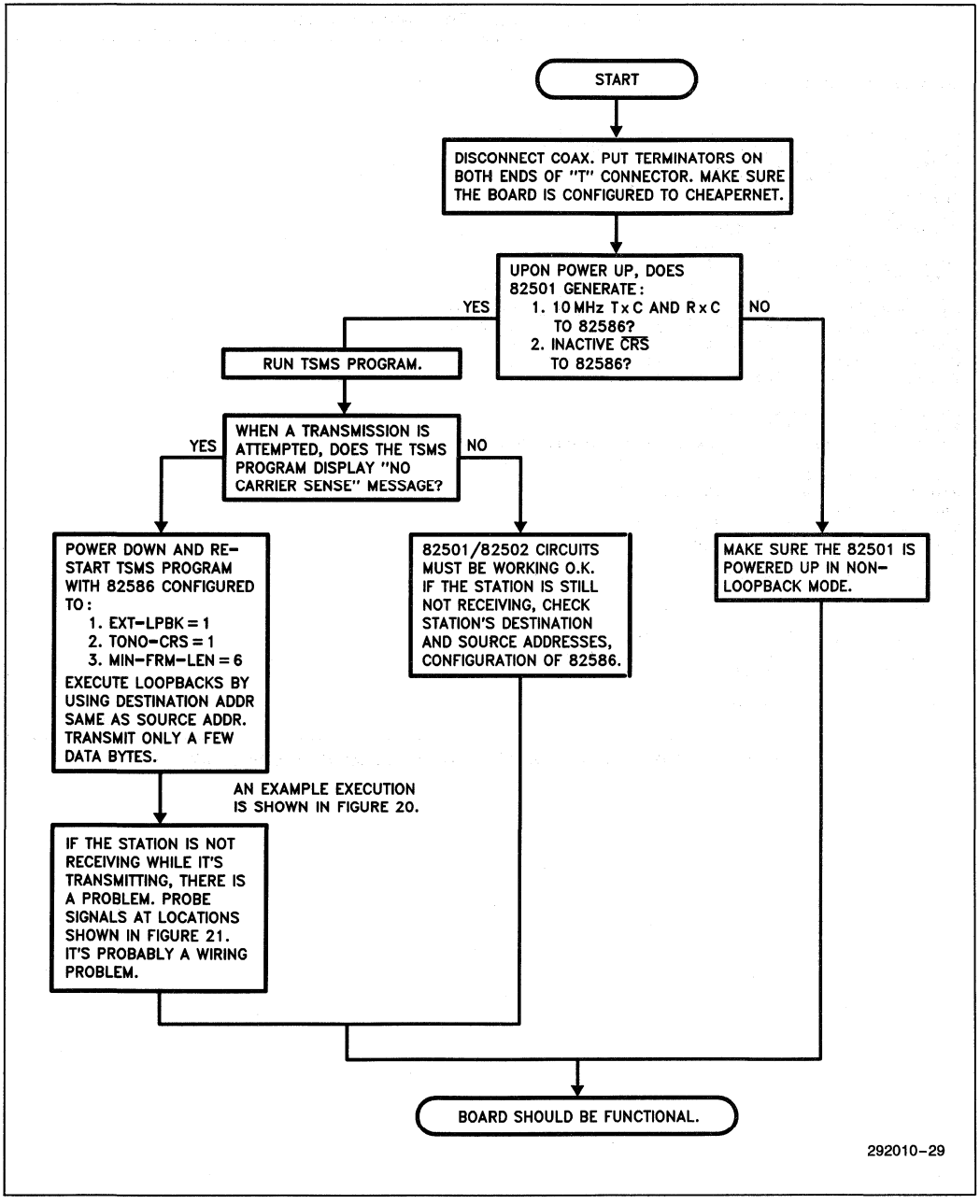


Figure 19. Flowchart for 82501/82502 Circuits Troubleshooting

Traffic Simulator and Monitor Station Program

```
Initialization begun


Configure command is set up for default values.
Do you want to change any bytes? (Y or N) ==> Y
Enter byte number (1 - 11) ==> 4
Enter byte 4 (4H) ==> A6H
Any more bytes? (Y or N) ==> Y
Enter byte number (1 - 11) ==> 9
Enter byte 9 (9H) ==> 08H
Any more bytes? (Y or N) ==> Y
Enter byte number (1 - 11) ==> 11
Enter byte 11 (BH) ==> 6
Any more bytes? (Y or N) ==> N
Configure the 586 with the prewired board address ==> N
Enter this station's address in Hex ==> 000000002200
You can enter up to 8 Multicast Addresses.
Would you like to enter a Multicast Address? (Y or N) ==> N
You entered 0 Multicast Address(es).

Would you like to transmit?
Enter a Y or N ==> Y
Enter a destination address in Hex ==> 000000002200

Enter TYPE ==> 0
How many bytes of transmit data?
Enter a number ==> 2
Transmit Data is continuous numbers (0, 1, 2, 3, ... )
Change any data bytes? (Y or N) ==> N
Enter a delay count ==> 0
Setup a transmit terminal count? (Y or N) ==> N

Destination Address: 00  00  00  00  22  00
Frame Length: 20 bytes
Time Interval between Transmit Frames: 159.4 seconds
Network Percent Load generated by this station: 11.0 %
Transmit Frame Terminal Count: Not Defined

Good enough? (Y or N) ==> Y
```

292010-77

Figure 20. TSMS Initialization for 82501/82502 Circuits Troubleshooting

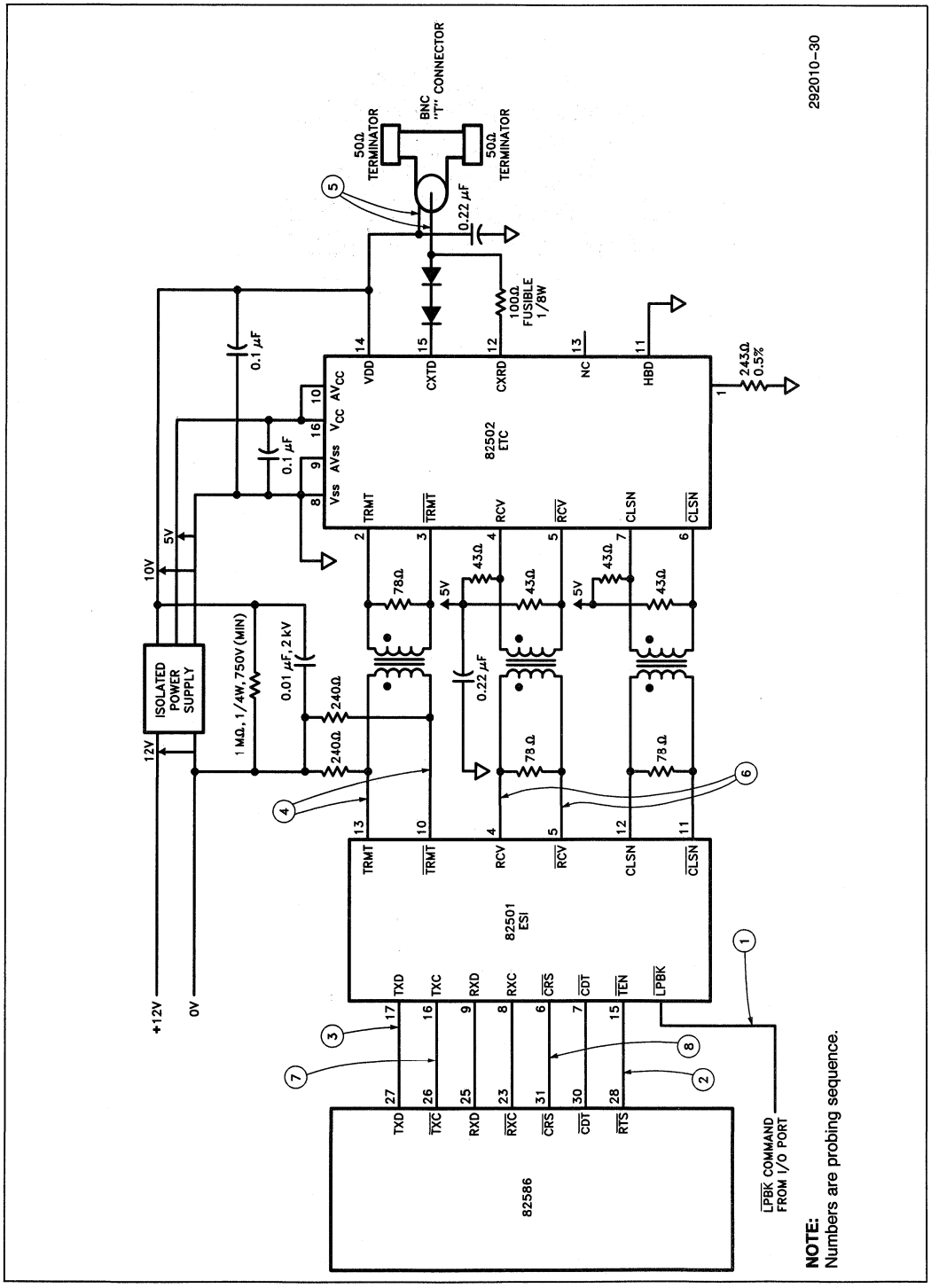
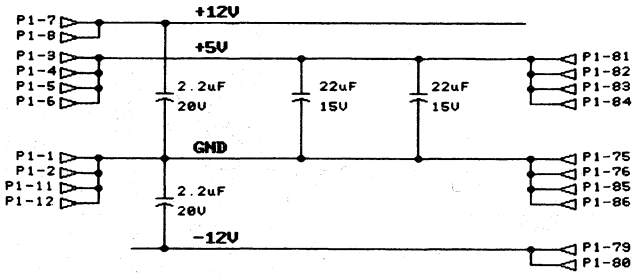


Figure 21. Probing 82501/82502 Circuits

APPENDIX A
LANHIB SCHEMATICS
PARTS LIST
PAL EQUATIONS
DIP SWITCH SETTINGS
WIRE WRAP SERVICES

PARTS LIST

REFERENCES	DESCRIPTION	MFR. PART NO.	MFR. CODE	QTY.
U1	IC	74S32	OBD	1
U2	IC	74LS04	OBD	1
U3, U4	IC	74LS245	OBD	2
U5	IC	74F373	OBD	1
U6,U7	IC	74LS373	OBD	2
U8	IC	80186	INT	1
U9	IC	82586	INT	1
U10	IC	16L8	OBD	1
U11	IC	74LS02	OBD	1
U12, U27	IC	74LS74	OBD	2
U28	IC	74AS74	OBD	1
U13	IC	74LS165	OBD	1
U14	IC	82501	INT	1
U15	Pulse Transformer Pack	PE64102	PE	1
U16	IC	82502	INT	1
U17	DC/DC Converter	PE64369	PE	1
U18, U20	IC, 64K-Bit EPROM	2764-20	INT	2
U26	IC	74AS08	OBD	1
U22, U23	IC, SRAM	HM6264-15	HIT	2
U24	IC, 256-Bit PROM	TBP18S030	TI	1
U25	IC	74AS04	OBD	1
U29	IC	82530	INT	1
U30	IC	1489	OBD	1
U31	IC	1488	OBD	1
U32	IC, 1M-Bit EPROM (Optional)	27210	INT	1
R1-R3, R6 R19, R20	Resistor, 10K ohm, 1/4W, 5%	COML	OBD	6
R4	Resistor, 100K ohm, 1/4W, 5%	COML	OBD	1
R5	Resistor, 2.2K ohm, 1/4W, 5%	COML	OBD	1
R7, R8, R12	Resistor, 78.7 ohm, 1/8W, 1%	COML	OBD	3
R9, R10	Resistor, 240 ohm, 1/4W, 5%	COML	OBD	2
R11	Resistor, 1M ohm, 1/4W, 750Vdc (min), 5%	COML	OBD	1
R13-R16	Resistor, 43.2ohm 1/8W, 1%	COML	OBD	4
R17	Resistor, 100 ohm, Fusible 1/8W, 5%	COML	RCD	1
R18	Resistor, 243 ohm, 1/8W, 0.5%	COML	OBD	1
R21, R22	Resistor, 5K ohm, 1/4W, 5%	COML	OBD	2
RP1	Resistor Pack, 1K ohm, 16 pin	COML	OBD	1
R23-R26	Resistor, 1K ohm, 1/4W, 5%	COML	OBD	4
C1, C2	Capacitor, 20pF, 100V, 5%	COML	OBD	2
C3	Capacitor, 10uF, 20V	COML	OBD	1
C4, C5	Capacitor, 30pF, 100V, 5%	COML	OBD	2
C6	Capacitor, 0.022uF, 50V	COML	OBD	1
C7	Capacitor, 1.0uF, 50V	COML	OBD	1
C11,C12	Capacitor, 0.01uF, 50V	COML	OBD	2
C10	Capacitor, 0.01uF, 2KV	COML	OBD	1
C8, C9	Capacitor, 0.22uF, 50V	COML	OBD	2
CR1	Diode	1N914	OBD	1
CR2, CR3	Diode	1N5282	OBD	2
Y1	Parallel Resonant Crystal, 16M Hz	COML	OBD	1
Y2	Parallel Resonant Crystal, 20M Hz	COML	OBD	1



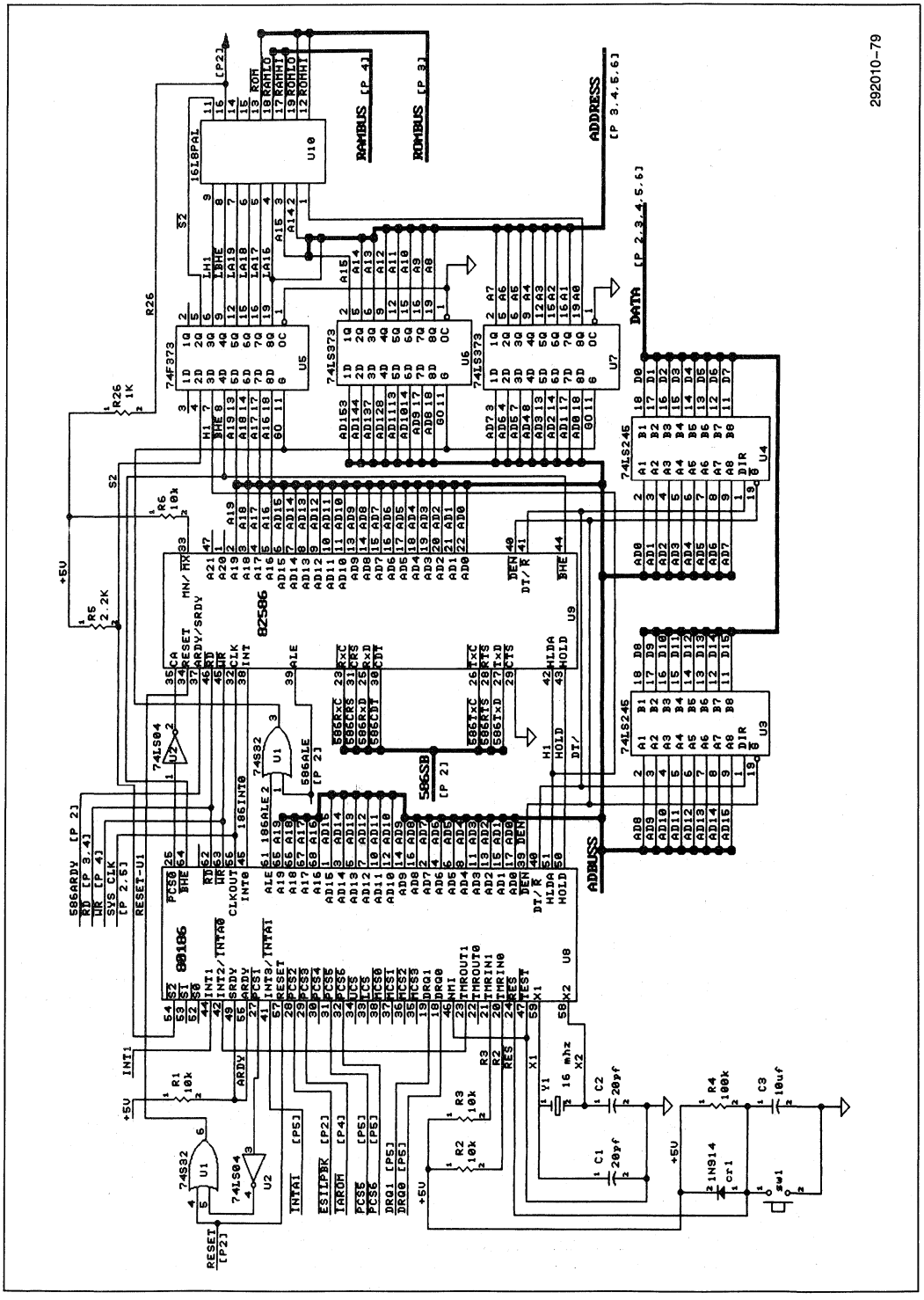
POWER SUPPLY CONNECTIONS

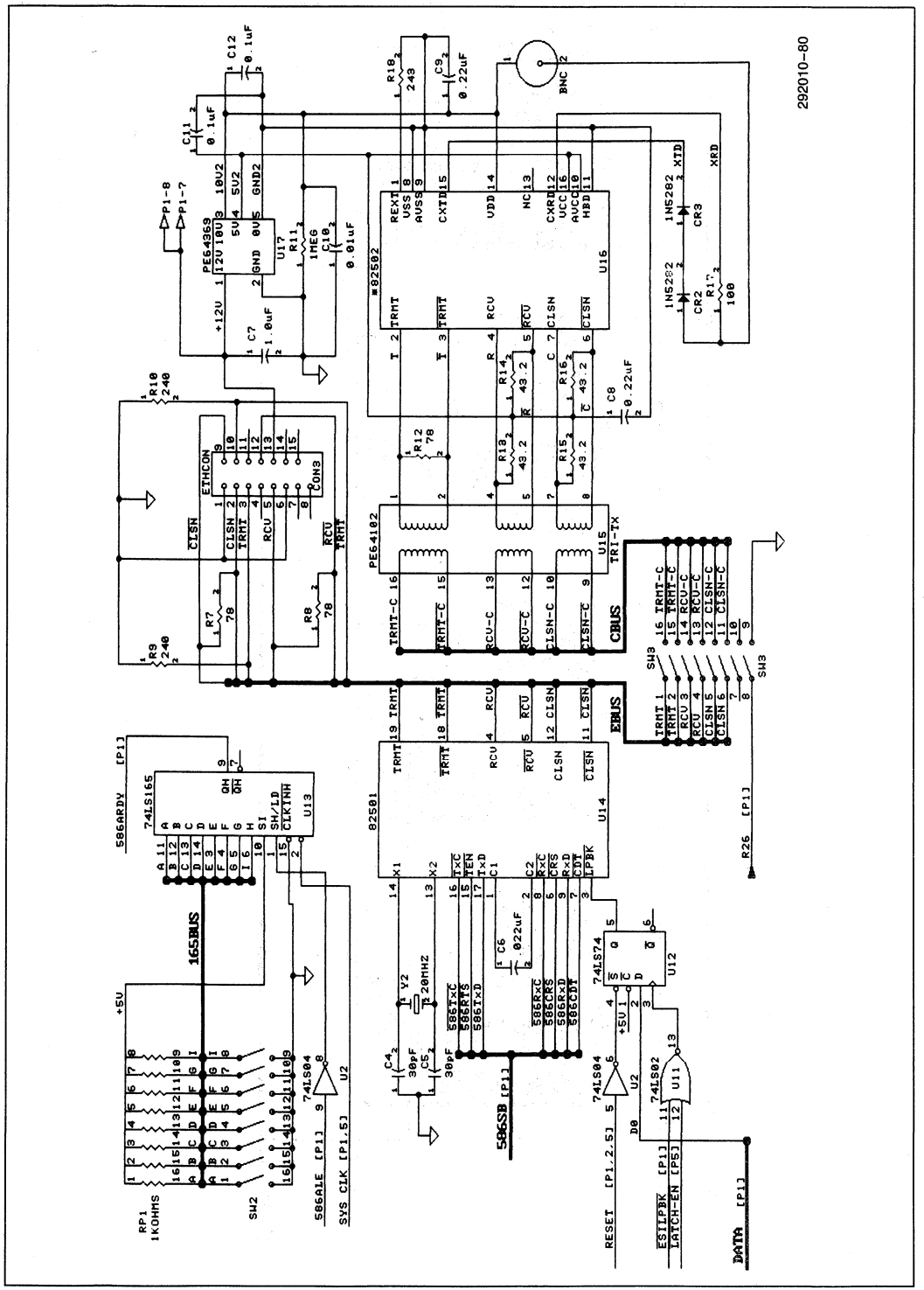
NOTES:

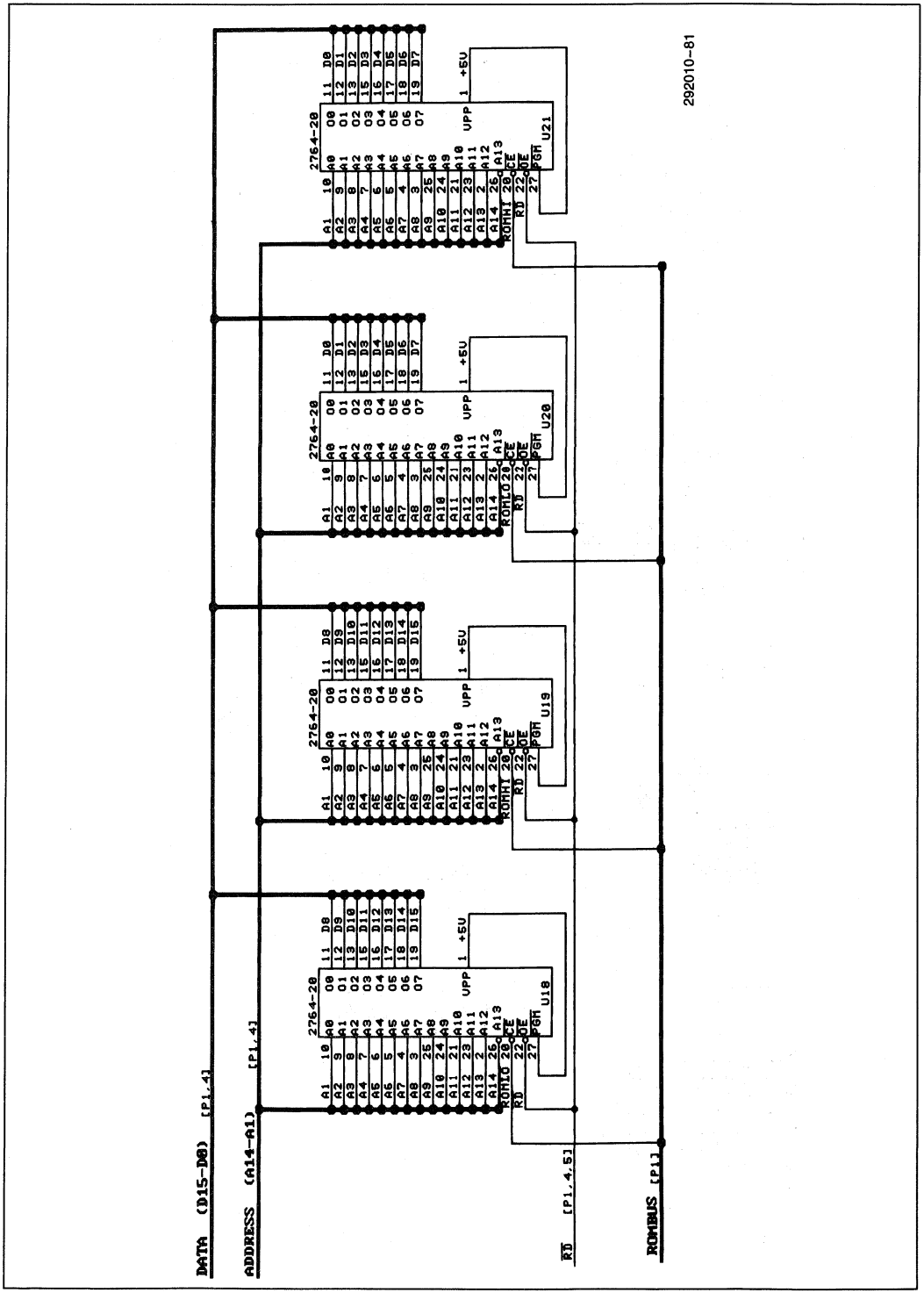
1. THE BOARD REQUIRES +5V, +12V, AND -12V. MULTIBUS POWER PINS FOR THESE VOLTAGES AND GROUND ARE SHOWN ABOVE.
2. EACH IC SHOULD HAVE A 0.1uF CAPACITOR BETWEEN POWER PIN AND GROUND PIN. PARTS LIST DOES NOT INCLUDE DECOUPLING CAPACITORS.
3.

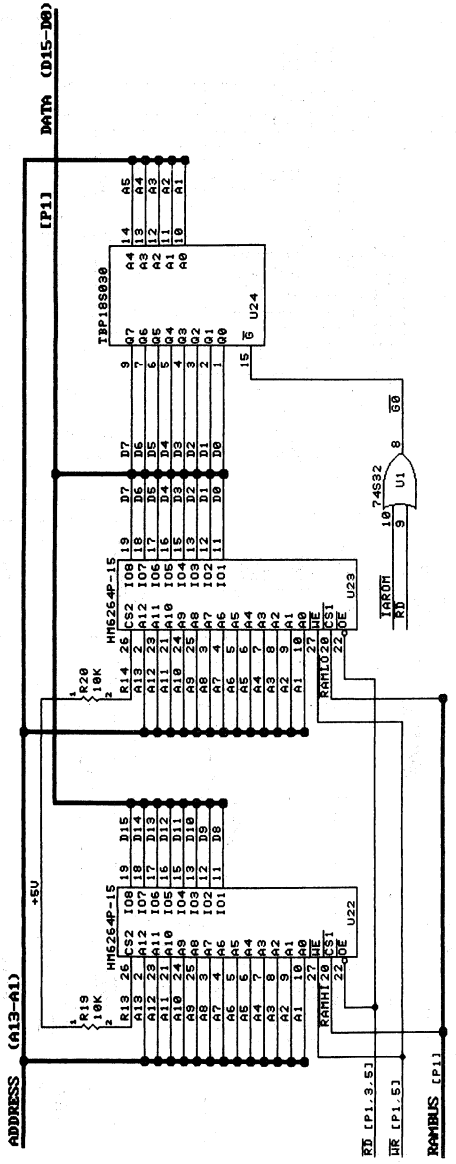
MFR. CODE	MANUFACTURE	LOCATION
INT	INTEL CORPORATION	SANTA CLARA, CA
HIT	HITACH AMERICA LTD.	SAN JOSE, CA
OBD	ORDER BY DESCRIPTION (ANY COMMERCIAL (COML) SOURCE)	
PE	PULSE ENGINEERING	SAN DIEGO, CA
RCD	RCD COMPONENTS INC.	MANCHESTER, NH
TI	TEXAS INSTRUMENTS	DALLAS, TX

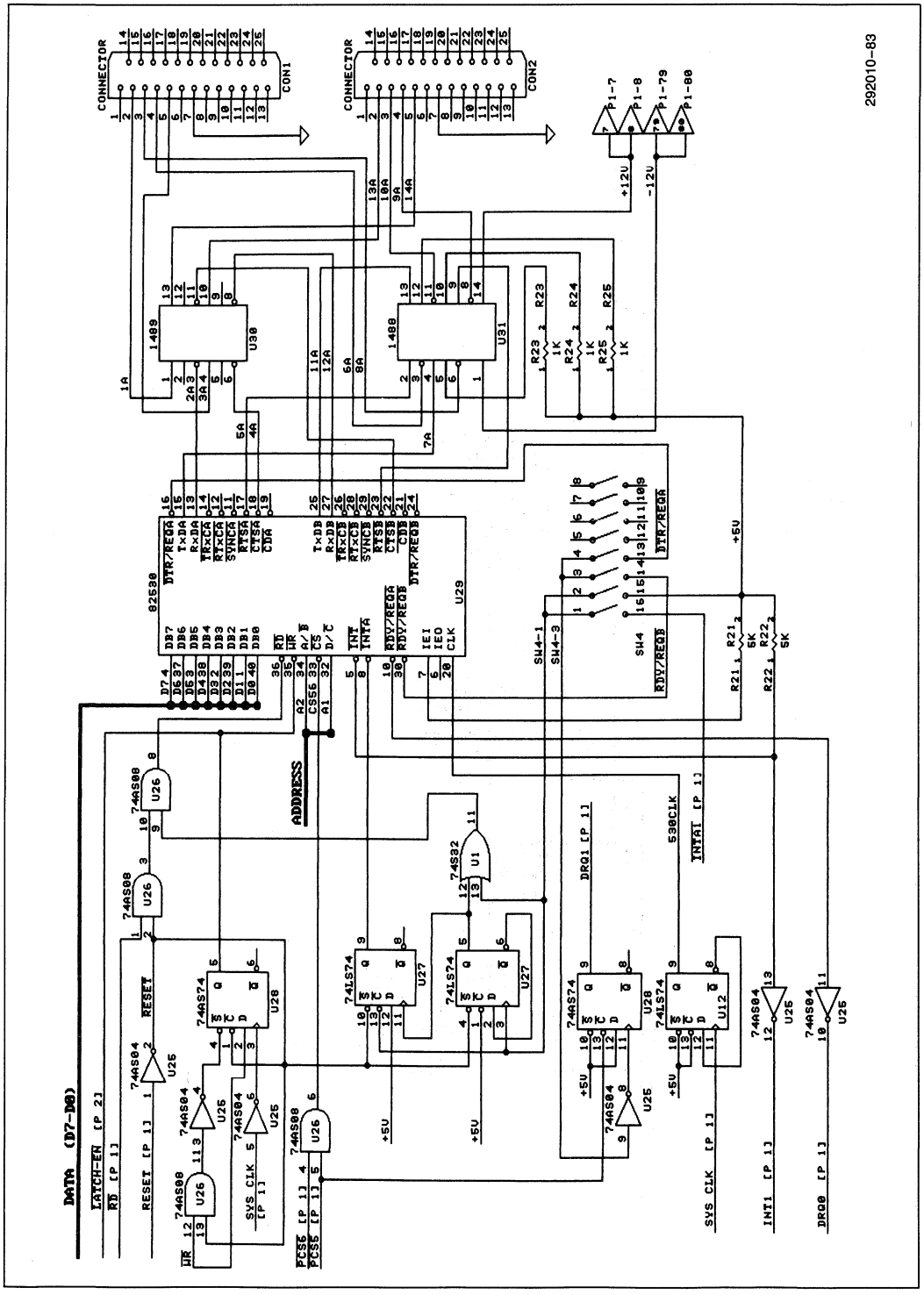
292010-78

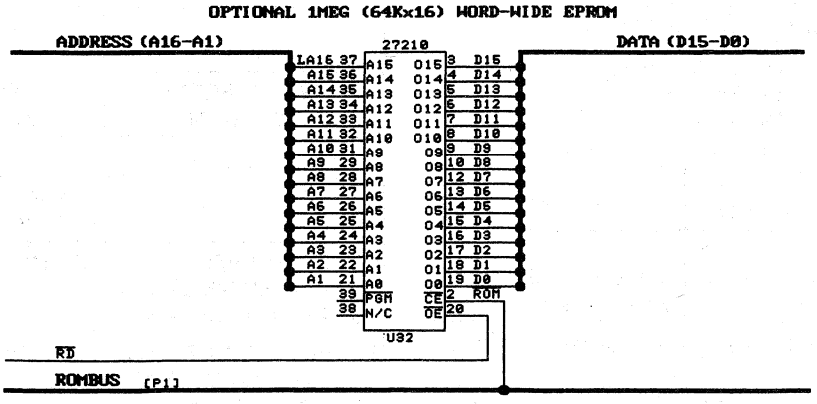

292010-79
ADDRESS
[P 3.4.5.6]
DATA
[P 2.3.4.5.6]
RAMBUS [P 4]
ROMBUS [P 3]
ADDBUSS
RESET-U1
SYS CLK
[P 2.5]
RD [P 3.4]
WR [P 4]
586ARDY [P 2]
RESET
[P2]
INT1
[P2]
INTA1 [P5]
ESILPBK [P2]
TAROM [P4]
PCS5 [P5]
PCS6 [P5]
DRQ1 [P5]
DRQ0 [P5]
82586
80186
586SB
74F873
74LS373
74LS245
74S32
74LS04
16I8PAI
X1
16 mhz
X2
1N914
sw1


292010-80
82501
82502
PE64102
PE64369
74LS165
74LS74
74LS04
74LS02
ETHCON
CBUS
EBUS
TRI-TX
165BUS
586BUS
586ARDY
586ALE [P1]
SYS CLK [P1.5]
RESET [P1.2.5]
ESILPBK [P1]
LATCH-EN [P5]
DATA [P1]
RP1
1KOHMS
SW2
SW3
BNC
XTD
XRD
+5V
+12V
CISN
RCU
TRMT


DATA (D15-D8) [P1.4]
ADDRESS (A14-A1) [P1.4]
RD [P1.4,5]
ROMBUS [P1]
2764-28
U18
U19
U20
U21
+5V
VPP
PGM
CE
OE
292010-81


292010-82
DATA (D15-D0)
[P1]
ADDRESS (A13-A1)
+5V
HM6264P-15
R19
10K
R20
10K
IBP18S030
U24
74S32
U1
U22
U23
RAMH1
RAML0
CS1
OE
WE
IOR [P1.5]
RD [P1.3.5]
RAMBUS [P1]


292010-83
CONNECTOR
CON1
CON2
U30
U31
1489
1488
R23
R24
R25
1K
+12V
-12V
P1-7
P1-8
P1-79
P1-80
82530
U29
SW4
SW4-1
SW4-3
R21
R22
5K
+5V
ADDRESS
DATA (D7-D0)
LATCH-EN [P 2]
RD [P 1]
RESET [P 1]
WR
SYS CLK [P 1]
PCS5 [P 1]
PCS6 [P 1]
74AS08
74AS04
74AS74
74LS74
74S32
U26
U25
U28
U27
U1
U12
DRQ1 [P 11]
INTA1 [P 11]
SYS CLK [P 11]
INT1 [P 11]
DRQ0 [P 11]



292010-84

```
Module Addr_dec
Title 'LANHIB Address Decode Logic
       Kiyoshi Nishide    Intel Corp.    March, 1986'

"Declarations

    PAL1                            device              'P16L8';

    A0, A14, A15        pin                             1, 2, 3;
    A16, A17, A18       pin                             4, 5, 6;
    A19, BHE                        pin                 7, 8;
    HLDA, S2                        pin                 9, 11;
    RAML0, RAMHI        pin                             18, 17;
    ROML0, ROMHI        pin                             19, 12;
    ROM                 pin         13;
    R104                pin         16;

Equations

    !ROMHI = A15 & A16 & A17 & A18 & A19 & (HLDA # S2) & R104;
    !ROML0 = !A15 & A16 & A17 & A18 & A19 & (HLDA # S2) & R104;
    !ROM = A17 & A18 & A19 & (HLDA # S2) & !R104;
    !RAMHI = !A14 & !A15 & !A16 & !A17 & !A18 & !A19 & !BHE & (HLDA # S2);
    !RAML0 = !A0 & !A14 & !A15 & !A16 & !A17 & !A18 & !A19 & (HLDA # S2);

End Addr_dec
```

PAL Equations

DIP SWITCH SETTINGS FOR VARIOUS OPERATIONS

"1" indicates ON (Switch is closed).
"0" indicates OFF (Switch is open).
"X" indicates Don't Care.

1. To configure the board to Ethernet or Cheapernet:

	SW3 87654321	Comment
Ethernet	XX000000	
Cheapernet	XX111111	Transceiver Cable should not be connected.

2. To run the TSMS program or the Data Link Driver program:

	SW4 87654321	Comment
TSMS Program or Data Link Driver Program	XXXX0001	TSMS program uses the 82530 in Asynchronous Polling mode. Data Link Driver program uses the 825830 in Asynchronous Polling and Vectored Interrupt modes.

3. To select the 2764-20 EPROMs or 27210 EPROM:

	SW3 87654321
2764-20 EPROMs	0XXXXXXX
27210 EPROM	1XXXXXXX

4. Dip Switch Setting Examples:

	SW3 87654321	SW4 87654321
1) To run the TSMS Program from the 2764-20 EPROMs in Cheapernet Configuration	0X111111	XXXX0010
2) To run the TSMS Program from the 2764-20 EPROMs in Ethernet Configuration	0X000000	XXXX0010
3) To run the TSMS Program or the Data Link Driver program from the 27210 EPROM in Cheapernet Configuration	1X111111	XXXX0001
4) To run the TSMS Program or the Data Link Driver program from the 27210 EPROM in Ethernet Configuration	1X000000	XXXX0001

5. Dip Switch SW2 programs the number of wait states for the 82586 (see Table 3).

COMPANIES OFFERING WIRE WRAP SERVICES

AUGAT
Interconnection Systems Division

40 Perry Avenue
P.O. Box 1037
Attleboro, MA 02703
(617) 222-2202

100935 South Wilcrest Drive
Houston, TX 77099
(713) 495-3100

Automation Delectronics Corporation

1650 Locust Avenue
Bohemia, NY 11716
(516) 567-7007

dataCon, Inc.

Eastern Division
60 Blanchard Road
Burlington, MA 01803
(617) 273-5800

Mid-Western Division
502 Morse Avenue
Schaumburg, IL 60193
(312) 529-7690

Western Division
20150 Sunburst Street
Chatsworth, CA 91311-6280
(818) 700-0600

South-Western Division
1829 Monetary Lane
Carrollton, TX 75006
(214) 245-6161

European Division
In der Klinge 5
D-7100 Heilbronn, West Germany
(01731) 217 12

DATAWRAP

37 Water Street
Wakefield, MA 01880
(617) 938-8911

Elma/EMS
A Division of Sandberg Industries

Berkshire Industrial Park
Bethel, CT 06801
(203) 797-9711

1851 Reynolds Avenue
Irvine, CA 92714
(714) 261-9473

3042 Scott Boulevard
Santa Clara, CA 95054
(408) 970-8874

WRAPEX Corporation

96 Mill Street
Woonsocket, RI 02895
(401) 769-3805

APPENDIX B
SOFTWARE LISTINGS—TSMS PROGRAM AND LANHIB INITIALIZATION ROUTINE

```
/*************************************************************************/
/*                                                                       */
/*                  Traffic Simulator/Monitor Station Program            */
/*                  for 186/586 High Integration Board and               */
/*                  iSBC 186/51                                          */
/*                                                                       */
/*                  Ver. 1.0              December 17, 1984              */
/*                                                                       */
/*                  Kiyoshi Nishide       Intel Corporation              */
/*                                                                       */
/*************************************************************************/


        /* This software can be conditionally compiled to work on the iSBC 186/51 or
           on the LANHIB.  If 'set(SBC18651)' is added to the compiler call statement,
           this source program will be compiled for the iSBC18651.  */

1       tsms:

        do;

2   1   declare main label public;

        /* literals */

        $IF SBC18651

        declare lit             literally 'literally',
                true                    lit '1',
                false                   lit '0',
                forever                 lit 'while 1',
                ISCP$LOC$LO             lit '0FFF0H',
                ISCP$LOC$HI             lit '0',
                SCB$BASE$LO             lit '0',
                SCB$BASE$HI             lit '0',
                CA$PORT                 lit '0C8H',
                BOARD$ADDRESS$BASE      lit '0F0H',
                INT$TYPE$586            lit '20H',
                INT$TYPE$TIMER0         lit '30H',
                INT$CTL$TIMER0          lit '0FF32H',
                INT$7                   lit '27H',
                PIC$MASK$130            lit '0E2H',
                PIC$MASK$186            lit '0FF28H',
                ENABLE$586              lit '0FEH',
                ENABLE$586$186          lit '0EEH',
                PIC$EOI$130             lit '0E0H',
                EOI$CMD0$130            lit '60H',
                EOI$CMD4$130            lit '64H',
                PIC$EOI$186             lit '0FF22H',
                EOI$CMD0$186            lit '0',
                PIC$VTR$186             lit '0FF20H',
```

292010-31

Traffic Simulator/Monitor Station Program

```
                TIMER0$CTL              lit '0FF56H',
                TIMER0$COUNT            lit '0FF50H',
                MAX$COUNT$A             lit '0FF52H',
                CA                      lit '0',
                ESI$PORT                lit '0C8H',
                NO$LOOPBACK             lit '8',
                LOOPBACK                lit '0';

        $ELSE

3   1   declare lit             literally 'literally',
                true                    lit '1',
                false                   lit '0',
                forever                 lit 'while 1',
                ISCP$LOC$LO             lit '03FF8H',
                ISCP$LOC$HI             lit '0',
                SCB$BASE$LO             lit '0',
                SCB$BASE$HI             lit '0',
                CA$PORT                 lit '8000H',
                BOARD$ADDRESS$BASE      lit '8180H',
                INT$TYPE$586            lit '12',
                INT$TYPE$TIMER0         lit '8',
                INT$CTL$TIMER0          lit '0FF32H',
                PIC$MASK$186            lit '0FF28H',
                ENABLE$586              lit '0EFH',
                ENABLE$586$186          lit '0EEH',
                PIC$EOI$186             lit '0FF22H',
                EOI$CMD0$186            lit '12',
                EOI$CMD4$186            lit '8',
                TIMER0$CTL              lit '0FF56H',
                TIMER0$COUNT            lit '0FF50H',
                MAX$COUNT$A             lit '0FF52H',
                CA                      lit '0',
                ESI$PORT                lit '8100H',
                NO$LOOPBACK             lit '1',
                LOOPBACK                lit '0';

        $ENDIF

        $IF NOT SBC18651

        /* System Configuration Pointer */

4   1   declare scp structure
                (
                sysbus byte,
                unused (5) byte,
                iscp$addr$lo word,
                iscp$addr$hi word
                )
                at (0FFFF6H) data (0, 0, 0, 0, 0, 0, ISCP$LOC$LO, ISCP$LOC$HI);

        $ENDIF


        /* Intermediate System Configuration Pointer */
```

292010-32

Traffic Simulator/Monitor Station Program (Continued)

```
 5   1      declare iscp$ptr pointer,
                    iscp based iscp$ptr structure
                    (
                    busy byte,      /* set to 1 by CPU before its first CA to 586,
                                       cleared by 586 after reading info from it */
                    unused byte,    /* unused */
                    scb$o word,     /* offset of system control block */
                    scb$b (2) word /* base of system control block */
                    );

              /* System Control Block */

 6   1      declare scb structure
                    (
                    status word,      /* cause(s) of interrupt, CU state, RU state */
                    cmd word,         /* int acks, CU cmd, RESET bit, RU cmd */
                    cbl$offset word, /* offset of first command block in CBL */
                    rpa$offset word, /* offset of first packet descriptor in RPA */
                    crc$errs word,    /* crc error encounterd so far */
                    aln$errs word,    /* alignment errors */
                    rsc$errs word,    /* no resources */
                    ovrn$errs word    /* overrun errors */
                    );

              /* 82586 Action Commands */

              /* NOP */

 7   1      declare nop structure
                    (
                    status word,
                    cmd word,
                    link$offset word
                    );


              /* Individual Address Setup */

 8   1      declare ia$setup structure
                    (
                    status word,
                    cmd word,
                    link$offset word,
                    ia$address (6) byte
                    );

              /* Configure */

 9   1      declare configure structure
                    (
                    status word,
                    cmd word,
                    link$offset word,
                    byte$cnt byte,
                    info (11) byte
                    );
```

292010-33

Traffic Simulator/Monitor Station Program (Continued)

```
                /* Multicast Address Setup */

10    1         declare mc$setup structure
                        (
                        status word,
                        cmd word,
                        link$offset word,
                        mc$byte$count word,
                        mc$address (48) byte  /* only 8 MC addresses are allowed */
                        );


                /* Transmit */

                /* This transmit command is made of one transmit buffer descriptor and one
                   1518 bytes long buffer. */

11    1         declare transmit structure
                        (
                        status word,
                        cmd word,
                        link$offset word,
                        bd$offset word,
                        dest$adr (6) byte,
                        type word
                        );

                /* Transmit Buffer Descriptor */

12    1         declare tbd structure
                        (
                        act$count word,
                        link$offset word,
                        ad0 word,
                        ad1 word
                        );

                /* Transmit Buffer */

13    1         declare tx$buffer (1518) byte;


                /* TDR */

14    1         declare tdr structure
                        (
                        status word,
                        cmd word,
                        link$offset word,
                        result word
                        );

                /* Diagnose */

15    1         declare diagnose structure
                        (
```

292010-34

Traffic Simulator/Monitor Station Program (Continued)

```
                        status word,
                        cmd word,
                        link$offset word
                        );


                /* Dump Status */

16    1         declare dump structure
                        (
                        status word,
                        cmd word,
                        link$offset word,
                        buff$ptr word
                        );

                /* Dump Area */

17    1         declare dump$area (170) byte;


                /* Frame Descriptor */

                /* Receive frame area is made of 5 RFDs, 5 RBDs, and 5 1514 bytes long
                   buffers. */

18    1         declare rfd (5) structure
                        (
                        status word,
                        el$s word,
                        link$offset word,
                        bd$offset word,
                        dest$adr (3) word,
                        src$adr (3) word,
                        type word
                        );

                /* Receive Buffer Descriptor */

19    1         declare rbd (5) structure
                        (
                        act$count word,
                        next$bd$link word,
                        ad0 word,
                        ad1 word,
                        size word
                        );

                /* Receive Buffer */

20    1         declare rbuf (5) structure
                        (buffer (1514) byte);


                /* global variables */

21    1         declare status word,              /* UART status */
```

292010-35

Traffic Simulator/Monitor Station Program (Continued)

```
                    actual word,                /* actual number of chars UART transferred */
                    c$buf (80) byte,            /* buffer for a line of chars */
                    dhex byte,                  /* number base switch */
                    ch byte at (@c$buf),
                    char$count byte,
                    receive$count dword,        /* counter for received frames */
                    count dword,                /* counter for transmitted frames */
                    preamble word,              /* preamble length in word */
                    address$length byte,        /* address length in byte */
                    ad$loc byte,                /* address location control of 82586 */
                    crc byte,                   /* crc length */
                    goback byte,                /* if set, go back to Continuous Mode */
                    reset byte,                 /* reset flag */
                    delay word,                 /* delay conunt for tranmission delay */
                    cur$cb$offset word,         /* offset of current command block */
                    current$frame byte,         /* offset of frame descriptor just used */
                    no$transmission byte,
                    stop$count dword,           /* transmit terminal frame count */
                    stop byte,
                    mc$count byte,
                    z byte,
                    y byte;

            /* external procedures */

22   1      read: procedure (a, b, c, d, e) external;
23   2      declare (a, c) word,
                    (b, d, e) pointer;
24   2      end read;


25   1      write: procedure (a, b, c, d) external;
26   2      declare (a, c) word,
                    (b, d) pointer;
27   2      end write;


28   1      csts: procedure byte external;
29   2      end csts;


            /* utility procedures */

30   1      offset: procedure (ptr) word;

            /* This procedure takes a pointer variable (selector:offset), caluculates an
               absolute address, subtracts the 82586 SCB offset from the absolute address,
               and then returns the result as an offset value for the 82586. */

31   2      declare (ptr, ptr$loc) pointer,
                    base586 dword,
                    w based ptr$loc (2) word;

32   2          ptr$loc = @ptr;

               /* 82586 SCB Base Address (20-bit wide in this 186 based system) */
```

292010-36

Traffic Simulator/Monitor Station Program (Continued)

```
33   2          base586 = (shl(double (iscp.scb$b(1)), 16) and 000F0000H) + iscp.scb$b(0);
34   2          return low((shl(double (w(1)), 4) + w(0)) - base586);

35   2      end offset;


36   1      writeln: procedure (a, b, c, d);

            /* This procedure writes a line and put a CR/LF at the end. */

37   2      declare (a, c) word,
                     (b, d) pointer;

38   2          call write(a, b, c, d);
39   2          call write(0, @(0DH, 0AH), 2, @status);

40   2      end writeln;


41   1      cr$lf: procedure;

            /* This procedure writes a CR/LF. */

42   2          call write (0, @(0DH, 0AH), 2, @status);

43   2      end cr$lf;


44   1      pause: procedure;

            /* This procedure breaks a program flow, and waits for a char to be typed. */

45   2          call write(0, @(0DH, 0AH, 'Hit <CR> to countinue'), 23, @status);
46   2          call read(1, @c$buf, 80, @actual, @status);
47   2          call cr$lf;

48   2      end pause;


49   1      skip: procedure byte;

            /* This procedure skips all leading blank characters and returns the first
               non-blank character.  */

50   2      declare i byte;

51   2          i = 0;
52   2          do while (c$buf(i) = ' ');
53   3              i = i + 1;
54   3          end;
55   2          return i;

56   2      end skip;


57   1      read$char: procedure byte;
```

292010-37

Traffic Simulator/Monitor Station Program (Continued)

```
                /* This procedure reads a line and returns ther first non-blank character.  */

58    2         declare i word;

59    2             call read(1, @c$buf, 80, @actual, @status);
60    2             i = skip;
61    2             return(c$buf(i));

62    2         end read$char;


63    1         read$bit: procedure byte;

                /* This procedure reads a bit and returns the value.  */

64    2         declare b byte;

65    2             do forever;
66    3                 b = read$char;
67    3                     if b = '1' then return 1;
69    3                 else
70    3                     if b = '0' then return 0;
71    3                 else
                          call write(0, @(' Enter a 0 or 1 ==> '), 20, @status);
72    3             end;

73    2         end read$bit;


74    1         yes: procedure byte;

                /* This procedure reads a character and determines if it is a Y(y) or N(n).   */

75    2         declare b byte;

76    2             do forever;
77    3                 b = read$char;
78    3                 if (b = 'Y') or (b = 'y') then return true;
80    3                 else
81    3                     if (b = 'N') or (b = 'n') then return false;
82    3                 else
                          call write(0, @(0DH, 0AH, ' Enter a Y or N ==> '), 22, @status);
83    3             end;

84    2         end yes;


85    1         char$to$int: procedure (c) byte;

                /* This procedure converts a byte of ASCII integer to an integer.   */

86    2         declare c byte;

87    2             if ('0' <= c) and (c <= '9') then return (c - 30H);
89    2             else
90    2                 if('A' <= c) and (c <= 'F') then return (c - 37H);
91    2                 else
```

292010-38

Traffic Simulator/Monitor Station Program (Continued)

```
 92   2                if ('a' <= c) and (c <= 'f') then return (c - 57H);
 93   2                else return OFFH;

 94   2      end char$to$int;


 95   1      int$to$asci: procedure (value, base, ld, bufadr, width);

              /* This procedure converts an interger < OFFFFFFFFH to an array of ASCII
                 codes.
                 Input variables are:  value = integer to be converted,
                                       base = number base to be used for conversion,
                                       ld = leading character to be filled in,
                                       bufadr = buffer address of the array,
                                       width = size of array. */

 96   2      declare value dword,
                      bufadr pointer,
                      (i, j, base, ld, width) byte,
                      chars based bufadr (1) byte;

 97   2          do i = 1 to width;
 98   3              j = value mod base;
 99   3              if j < 10 then chars (width - i) = j + 30H;
101   3              else chars (width - i) = j + 37H;
102   3              value = value / base;
103   3          end;
104   2          i = 0;
105   2          do while chars (i) = '0' and i < width - 1;
106   3              chars (i) = ld;
107   3              i = i + 1;
108   3          end;
109   2          char$count = width - i;

110   2      end int$to$asci;


111   1      out$word: procedure (w$ptr, distance);

              /* An integer at (selector of w$ptr):(offset of w$ptr + distance) is printed
                 as a 4 digit hexadecimal number.  */

112   2      declare chars(4) byte,
                      w$ptr pointer,
                      distance byte,
                      w based w$ptr (1) word;

113   2          call int$to$asci(w(distance), 16, '0', @chars(0), 4);
114   2          call write(0, @chars(0), 4, @status);

115   2      end out$word;


116   1      write$int: procedure(dw, t);

              /* An integer (dw) is printed in hexadecimal (t = 1) or in decimal (t = 0). */
```

292010-39

Traffic Simulator/Monitor Station Program (Continued)

```
117   2       declare dw dword,
                      chars (10) byte,
                      t byte;

118   2           if t then
119   2           do;
120   3               call int$to$asci(dw, 16, 0, @chars(0), 8);
121   3               call write(0, @chars(8-char$count), char$count, @status);
122   3           end;
123   2           else
              do;
124   3               call int$to$asci(dw, 10, 0, @chars(0), 10);
125   3               call write(0, @chars(10-char$count), char$count, @status);
126   3           end;

127   2       end write$int;


128   1       out$dec$hex: procedure(dw);

              /* This procedure prints an integer in decimal and hexadecimal. */

129   2       declare dw dword;

130   2           call write$int(dw, 0);
131   2           call write(0, @(' ('), 2, @status);
132   2           call write$int(dw, 1);
133   2           call write(0, @('H)'), 2, @status);

134   2       end out$dec$hex;


135   1       write$offset: procedure(w$ptr);

              /* This procedure takes a pointer variable, converts it to a 82586 type offset,
                 and prints it in hexadecimal.  */

136   2       declare w$ptr pointer,
                      w word;

137   2           call write(0, @(' at '), 4, @status);
138   2           w = offset(w$ptr);
139   2           call out$word(@w, 0);
140   2           call write(0, @('  '), 2, @status);

141   2       end write$offset;


142   1       write$address: procedure (ptr);

              /* This procedure takes a pointer variable and prints it in the
                 'selector:offset' format.  */

143   2       declare (ptr, ptr$loc)  pointer,
                      w based ptr$loc (2) word;

144   2           ptr$loc = @ptr;
```

292010-40

Traffic Simulator/Monitor Station Program (Continued)

```
145   2          call out$word(@w(1), 0);
146   2          call write(0, @(': '), 1, @status);
147   2          call out$word(@w(0), 0);
148   2          call write(0, @(' '), 1, @status);

149   2      end write$address;


150   1      print$wds: procedure(w$ptr, no$words);

             /* This procedure prints no$words number of words starting at w$ptr.  */

151   2      declare w$ptr pointer,
                     (i, no$words) byte;

152   2          if no$words <> 0 then
153   2          do;
154   3              call cr$lf;
155   3              do i = 0 to no$words - 1;
156   4                  call out$word(w$ptr, i);
157   4                  if i = 0 then
158   4                      call write$offset(w$ptr);
159   4                  call cr$lf;
160   4              end;
161   3          end;

162   2      end print$wds;


163   1      print$str: procedure (str$ptr, len);

             /* This procedure prints len number of bytes starting at str$ptr. */

164   2      declare (len, i) byte,
                     chars (2) byte,
                     str$ptr pointer,
                     str based str$ptr (1) byte;

165   2          if len <> 0 then
166   2          do i = 0 to (len - 1);
167   3              call int$to$asci(str(i), 16, '0', @chars(0), 2);
168   3              call write(0, @chars(0), 2, @status);
169   3              call write(0, @('  '), 2, @status);
170   3          end;
171   2          call cr$lf;

172   2      end print$str;


173   1      print$buff: procedure (ptr, cnt);

             /* This procedure prints cnt number of buffer contents starting at ptr. */

174   2      declare ptr pointer,
                     bt based ptr (1) byte,
                     (i, j) byte,
                     cnt word;
```

292010-41

Traffic Simulator/Monitor Station Program (Continued)

```
175   2          if cnt > 16 then
176   2          do;
177   3              i = shr(cnt, 4) - 1;
178   3              do j = 0 to i;
179   4                  call write$address(@bt(16*j));
180   4                  call print$str(@bt(16*j), 16);
181   4                  if (j = 20) or (j = 40) or (j = 60) or (j = 80) then
182   4                  call pause;
183   4              end;
184   3              i = i + 1;
185   3              if cnt-16*i <> 0 then call write$address(@bt(16*i));
187   3              call print$str(@bt(16*i), cnt-16*i);
188   3          end;
189   2          else
              do;
190   3              call write$address(@bt(0));
191   3              call print$str(@bt(0), cnt);
192   3          end;

193   2      end print$buff;



194   1      read$int: procedure (limit) dword;

              /* This procedure reads integer characters and forms an integer.  If the
                 integer is bigger than 'limit' or an overflow error is encounterred, then
                 an error message is printed.  */

195   2      declare (wd, wh, limit) dword,
                      (i, j, k, done, hex, dover, hover) byte;

196   2          do forever;
197   3              call read(1, @c$buf, 80, @actual, @status);
198   3              i, k = skip;
199   3              hex, done, dover, hover = false;
200   3              wd, wh = 0;
201   3              j = char$to$int(c$buf(i));
202   3              do while j <= 15;
203   4                  if j > 9 then hex = true;
205   4                  if not dover then
206   4                      if wd > 429496729 then dover = true;
208   4                      else if (wd = 429496729) and (j > 5) then dover = true;
210   4                  wd = wd*10 + j;
211   4                  if not hover then if wh > OFFFFFFFH then hover = true;
214   4                  wh = wh*16 + j;
215   4                  i = i + 1;
216   4                  j = char$to$int(c$buf(i));
217   4              end;
218   3              if ((c$buf(i) <> 'H') and (c$buf(i) <> 'h') and (c$buf(i) <> ODH) and
                           (c$buf(i) <> OAH) and (c$buf(i) <> ' ')) or (i = k) then
219   3              call writeln(0, @(ODH, OAH, ' Illegal character'), 20, @status);
220   3              else
                      do;
221   4                  if (c$buf(i) = 'H') or (c$buf(i) = 'h') then hex = true;
223   4                  if hex then
```

292010-42

Traffic Simulator/Monitor Station Program (Continued)

```
224   4                 do;
225   5                     if not hover and (wh <= limit) then return wh;
227   5                 end;
228   4                 else
229   4                     if not dover and (wd <= limit) then return wd;
230   4                 call writeln(0, @(0DH, 0AH, ' The number is too big. '), 25,
                                                                          @status);
231   4                 call write(0, @(' It has to be less than or equal to '), 36,
                                                                          @status);
232   4                 call out$dec$hex(limit);
233   4                 call writeln(0, @('. '), 1, @status);
234   4             end;
235   3             call write(0, @(' Enter a number ==> '), 20, @status);
236   3         end;

237   2     end read$int;


238   1     put$address: procedure(where);

            /* This procedure puts an address typed in hexadecimal to the specified
               location 'where'. */

239   2     declare where pointer,
                    (i, j, m, err) byte,
                    addr based where (1) byte;

240   2         do forever;
241   3             err = false;
242   3             call read(1, @c$buf, 80, @actual, @status);
243   3             i = skip;
244   3             m = address$length;
245   3             do while (m <> 0) and not err;
246   4                 j = char$to$int(c$buf(i));
247   4                 if j = 0FFH then err = true;
249   4                 else
                          do;
250   5                     addr(m-1) = shl(j, 4);
251   5                     j = char$to$int(c$buf(i+1));
252   5                     if j = 0FFH then err = true;
254   5                     else addr(m-1) = addr(m-1) or j;
255   5                 end;
256   4                 i = i + 2;
257   4                 m = m - 1;
258   4             end;
259   3             if not err then
260   3             do;
261   4                 m = c$buf(i);
262   4                 if (m = 0DH) or (m = 0AH) or (m = 'h') or (m = 'H') or (m = ' ')
263   4                     then return;
264   4             end;
265   3             call writeln(0, @(0DH, 0AH, ' Illegal character'), 20, @status);
266   3             call write(0, @(' Enter an address in Hex ==> '), 29, @status);
267   3         end;

268   2     end put$address;
```

292010-43

Traffic Simulator/Monitor Station Program (Continued)

```
269   1     percent: procedure;

            /* This procedure calculates and prints a network percent load generated
               by this station.  The equation used in this procedure was obtained
               from actual measurements.  */

270   2     declare i word,
                    (j, k) dword,
                    pcent (3) byte;

271   2         j = (tbd.act$count and 3FFFH)*8;
272   2         if not ad$loc then k = (2*address$length + 2 + crc + preamble)*8;
274   2         else k = (crc + preamble)*8;
275   2         if delay <> 0 then

            $IF NOT SBC18651

276   2             i = low((1000*(j + k))/(1805 + k + 5*double(delay) + j));

            $ELSE

                  i = low((1000*(j + k))/(2021 + k + 5*double(delay) + j));

            $ENDIF

277   2             else

            $IF NOT SBC18651

                  i = low((1000*(j + k))/(1810 + k + j));

            $ELSE

                  i = low((1000*(j + k))/(2026 + k + j));

            $ENDIF

278   2             call int$to$asci(i, 10, 0, @pcent(0), 3);
279   2             call write(0, @pcent(0), 2, @status);
280   2             call write(0, @('. '), 1, @status);
281   2             call write(0, @pcent(2), 1, @status);
282   2             call writeln(0, @(' %'), 2, @status);

283   2     end percent;


284   1     print$network$addr: procedure (ptr);

            /* This station's address is printed with its least significant bit
               in the most right position.  */

285   2     declare ptr pointer,
                    addr based ptr (1) byte,
                    char (6) byte,
                    i byte;
```

292010-44

Traffic Simulator/Monitor Station Program (Continued)

```
286   2          do i = 1 to address$length;
287   3              char(i-1) = addr(address$length-i);
288   3          end;
289   2          call print$str(@char(0), address$length);

290   2      end print$network$addr;



291   1      print$parameters: procedure;

             /* This procedure prints transmission parameters. */

292   2      declare w dword,
                     stgs (6) byte;

293   2          call write(0, @(' Destination Address: '), 22, @status);
294   2          if not ad$loc then
295   2              call print$network$addr(@transmit.dest$adr(0));
296   2          else
                     call print$network$addr(@tx$buffer(0));
297   2          if not ad$loc then
298   2          w =  (tbd.act$count and 3FFFH) + address$length * 2 + 2 + crc;
299   2          else w = (tbd.act$count and 3FFFH) + crc;
300   2          call write(0, @(' Frame Length: '), 15, @status);
301   2          call write$int(w, 0);
302   2          call writeln(0, @(' bytes'), 6, @status);
303   2          call write(0, @(' Time Interval between Transmit Frames: '), 40, @status);
304   2          if delay <> 0 then
305   2          do;
             $IF NOT SBC18651

306   3              w = 1810 + (double(delay) - 1) * 5;

             $ELSE

                     w = 2026 + (double(delay) - 1) * 5;

             $ENDIF

307   3              call int$to$asci(w, 10, 0, @stgs, 6);
308   3              if w >= 10000 then
309   3              do;
310   4                  call write(0, @stgs(0), 2, @status);
311   4                  call write(0, @('.'), 1, @status);
312   4                  call write(0, @stgs(2), 2, @status);
313   4                  call writeln(0, @(' miliseconds'), 12, @status);
314   4              end;
315   3              else
                     do;
316   4                  call write(0, @stgs(0), 5, @status);
317   4                  call write(0, @('.'), 1, @status);
318   4                  call write(0, @stgs(5), 1, @status);
319   4                  call writeln(0, @(' microseconds'), 13, @status);
320   4              end;
321   3          end;
322   2          else
```

292010-45

Traffic Simulator/Monitor Station Program (Continued)

```
            $IF NOT SBC18651

                call writeln(0, @(' 159.4 microseconds'), 19, @status);

            $ELSE

                call writeln(0, @(' 172.8 microseconds'), 19, @status);

            $ENDIF

323   2         call write(0, @(' Network Percent Load generated by this station: '), 49,
                                                                          @status);
324   2         call percent;
325   2         call write(0, @(' Transmit Frame Terminal Count: '), 32, @status);
326   2         if stop then call write$int(stop$count, dhex);
328   2             else call write(0, @('Not Defined'), 11, @status);
329   2         call cr$lf;

330   2     end print$parameters;


331   1     print$scb: procedure;

            /* prints the SCB */

332   2         call writeln(0, @(0DH, 0AH, '*** System Control Block ***'), 30, @status);
333   2         call print$wds(@scb.status, 8);

334   2     end print$scb;



335   1     wait$scb: procedure;

            /* This procedure provids a wait loop for the SCB command word to
               become cleared.  */

336   2     declare i word;

337   2         i = 0;
338   2         do while (scb.cmd <> 0) and (i < 8000H);
339   3             i = i + 1;
340   3         end;
341   2         if scb.cmd <> 0 then
342   2             do;
343   3                 call write(0, @(0DH, 0AH, ' Wait Time = '), 15, @status);
344   3                 call write$int(i, 0);
345   3                 call cr$lf;
346   3             end;

347   2     end wait$scb;


348   1     start$timer0: procedure;

            /* 80186 timer0 is started. */
```

292010-46

Traffic Simulator/Monitor Station Program (Continued)

```
349   2           output(TIMER0$CTL) = 0E000H;

350   2       end start$timer0;


351   1       isr: procedure interrupt INT$TYPE$586 reentrant;

                /* interrupt service routine for 82586 interrupt */

352   2       declare i byte;

                    /* Enable 82586 Interrupt */

                $IF SBC18651

                    output (PIC$EOI$130) = EOI$CMD0$130;
                    enable;

                $ELSE

353   2             output (PIC$EOI$186) = EOI$CMD0$186;
354   2             enable;

                $ENDIF

                    /* Frame Received Interrupt has the highest priority */

355   2             if (scb.status and 4000H) = 4000H then
356   2             do;
357   3                 disable;
358   3                 scb.cmd = 4000H;
359   3                 output (CA$PORT) = CA;
360   3                 call wait$scb;
361   3                 if rfd(current$frame).status = 0A000H then
362   3                 do;
363   4                     receive$count = receive$count + 1;
364   4                     current$frame = current$frame + 1;
365   4                     if current$frame = 5 then current$frame = 0;
367   4                 end;
368   3                 return;
369   3             end;

370   2             if (scb.status and 2000H) = 2000H then
371   2             do;
372   3                 disable;
373   3                 scb.cmd = 2000H;
374   3                 output(CA$PORT) = CA;
375   3                 call wait$scb;
376   3                 enable;
377   3                 if (transmit.status and 0A000H) = 0A000H then
378   3                 do;
379   4                     count = count + 1;
380   4                     if (stop and (count = stop$count)) then return;
382   4                     else
                            do;
```

292010-47

Traffic Simulator/Monitor Station Program (Continued)

```
383   5                     transmit.status = 0;
384   5                     if delay = 0 then
385   5                     do;
386   6                         disable;
387   6                         scb.cmd = 0100H;
388   6                         output(CA$PORT) = CA;
389   6                         call wait$scb;
390   6                         return;
391   6                     end;
392   5                     else
                            do;
393   6                         call start$timer0;
394   6                         return;
395   6                     end;
396   5                 end;
397   4             end;
398   3             if (transmit.status and 0020H) = 0020H then
399   3             do;
400   4                 transmit.status = 0;
401   4                 disable;
402   4                 scb.cmd = 0100H;
403   4                 output (CA$PORT) = CA;
404   4                 call wait$scb;
405   4                 return;
406   4             end;
407   3             if (transmit.status and 0400H) = 0400H then
408   3             do;
409   4                 call write(0, @(0DH, ' No Carrier Sense!', 0DH), 20, @status);
410   4                 transmit.status = 0;
411   4                 disable;
412   4                 scb.cmd = 0100H;
413   4                 output (CA$PORT) = CA;
414   4                 call wait$scb;
415   4                 return;
416   4             end;
417   3             if (transmit.status and 0200H) = 0200H then
418   3             do;
419   4                 call write(0, @(0DH, ' Lost Clear to Send!', 0DH), 22, @status);
420   4                 transmit.status = 0;
421   4                 disable;
422   4                 scb.cmd = 0100H;
423   4                 output (CA$PORT) = CA;
424   4                 call wait$scb;
425   4                 return;
426   4             end;
427   3             if (transmit.status and 0100H) = 0100H then
428   3             do;
429   4                 call write(0, @(0DH, ' DMA Underrun!', 0DH), 16, @status);
430   4                 transmit.status = 0;
431   4                 disable;
432   4                 scb.cmd = 0100H;
433   4                 output (CA$PORT) = CA;
434   4                 call wait$scb;
435   4                 return;
436   4             end;
437   3         end;
438   2         if (scb.status and 8000H) = 8000H then
```

292010-48

Traffic Simulator/Monitor Station Program (Continued)

```
439   2           do;
440   3               disable;
441   3               scb.cmd = 8000H;
442   3               output (CA$PORT) = CA;
443   3               call wait$scb;
444   3           end;
445   2           if (scb.status and 1000H) = 1000H then
446   2           do;
447   3               disable;
448   3               scb.cmd = 1000H;
449   3               output (CA$PORT) = CA;
450   3               call wait$scb;
451   3               call write(0, @(ODH, ' Receive Unit became not ready.', ODH), 33,
                                                                                @status);
452   3           end;

453   2           if reset then
454   2           do;
455   3               if iscp.busy then
456   3               do;
457   4                   call writeln(0, @(ODH, OAH, ' Reset failed.'), 16, @status);
458   4                   disable;
459   4                   scb.cmd = 0080H;
460   4                   output (CA$PORT) = CA;
461   4                   call wait$scb;
462   4                   output (CA$PORT) = CA;
463   4                   call writeln(0, @(' Software Reset Executed!'), 25, @status);
464   4               end;
465   3               else reset = false;
466   3           end;

467   2       end isr;



468   1       tx$isr: procedure interrupt INT$TYPE$TIMER0;

                /* interrupt survice routine for 80186 timer interrupt*/

469   2           scb.cmd = 0100H;
470   2           output(CA$PORT) = CA;
471   2           call wait$scb;

                $IF SBC18651

                    output(PIC$EOI$130) = EOI$CMD4$130;
                    enable;
                    output(PIC$EOI$186) = EOI$CMD0$186;

                $ELSE

472   2             output(PIC$EOI$186) = EOI$CMD4$186;

                $ENDIF

473   2       end tx$isr;
```

292010-49

Traffic Simulator/Monitor Station Program (Continued)

```
            $IF SBC18651

            isr$7: procedure interrupt INT$7;

            /* The 80130 generates an interrupt 7 if the original interrupt is not
               active any more when the first interrupt acknowledge is received. */

                call write(0, @(0DH, 'Interrupt 7', 0DH), 13, @status);

            end isr$7;

            $ENDIF


474   1     read$byte: procedure (k) byte;
475   2     declare k word;

476   2         call write(0, @(0DH, 0AH, ' Enter byte '), 14, @status);
477   2         call out$dec$hex(k);
478   2         call write(0, @(' ==> '), 5, @status);
479   2         return read$int(0FFH);

480   2     end read$byte;


481   1     init$186$timer0: procedure;

            /* This procedure initializes the 80186 timer 0.  */

482   2     declare i byte;

            $IF SBC18651

                output(INT$CTL$TIMER0) = 8;
                call write(0, @(0DH, 0AH, ' Enter a delay count ==> '), 27, @status);
                delay = read$int(0FFFFH);
                if (delay < 100) and (delay <> 0) then
                do;
                    call cr$lf;
                    call cr$lf;
                    call loop$char(35, '*');
                    call write(0, @(' WARNING '), 9, @status);
                    call loop$char(35, '*');
                    call writeln(0, @(0DH, 0AH, 'A delay count between 0 and 100 may be very ',
                            'dangerous when this station starts'), 80, @status);
                    call writeln(0, @('to receive many frames separated only by the ',
                            'IFS period (9.6 microseconds). '), 75, @status);
                    call writeln(0, @('If this station never receives a frame, then ',
                                'ignore this warning. '), 65, @status);
                    call loop$char(79, '*');
                end;
                output(MAX$COUNT$A) = delay;
                call cr$lf;
                output(PIC$MASK$186) = 3EH;
```

292010-50

Traffic Simulator/Monitor Station Program (Continued)

```
            $ELSE

 483   2          output(INT$CTL$TIMER0) = 0CH;
 484   2          call write(0, @(0DH, 0AH, ' Enter a delay count ==> '), 27, @status);
 485   2          delay = read$int(0FFFFH);
 486   2          output(MAX$COUNT$A) = delay;
 487   2          call cr$lf;
 488   2          output(PIC$MASK$186) = ENABLE$586$186;

            $ENDIF

 489   2      end init$186$timer0;


 490   1      setup$ia$parameters: procedure;
 491   2      declare i byte;

 492   2          call write(0, @(0DH, 0AH, ' Configure the 586 with the prewired'
                                          ,' board address ==> '), 57, @status);
 493   2          if yes then
 494   2          do i = 0 to address$length - 1;
 495   3              ia$setup.ia$address(i) = input(BOARD$ADDRESS$BASE + 10 - 2 * i);
 496   3          end;
 497   2          else
             ;    do;
 498   3              call write(0, @(0DH, 0AH, ' Enter this station''s address',
                                                ' in Hex ==> '),43, @status);
 499   3              call put$address(@ia$setup.ia$address(0));
 500   3          end;

 501   2      end setup$ia$parameters;


 502   1      setup$mc$parameters: procedure;
 503   2      declare (j, k, done) byte;

 504   2          j = 0;
 505   2          call writeln(0, @(0DH, 0AH, ' You can enter up to 8 Multicast Addresses. '),
                                                                        45, @status);
 506   2          done = false;
 507   2          call write(0, @(' Would you like to enter a Multicast Address?',
                                                    ' (Y or N) ==> '), 59, @status);
 508   2          do while not done;
 509   3              if yes then
 510   3              do;
 511   4                  k = j * address$length;
 512   4                  j = j + 1;
 513   4                  call cr$lf;
 514   4                  if j = 9 then
 515   4                  do;
 516   5                      call write(0, @(' You already entered 8 Multicast addresses. '),
                                                              43, @status);
 517   5                      done = true;
 518   5                  end;
 519   4                  else
                            do;
 520   5                      call write(0, @(' Enter a Multicast Address ==> '), 31, @status);
```

292010-51

Traffic Simulator/Monitor Station Program (Continued)

```
521   5                    call put$address(@mc$setup.mc$address(k));
522   5                    call write(0, @(0DH, 0AH, ' More Multicast Addresses?',
                                                    ' (Y or N) ==> '), 42, @status);
523   5                end;
524   4            end;
525   3            else done = true;
526   3        end;
527   2        if j = 9 then j = j - 1;
529   2        mc$count = address$length * j;
530   2        mc$setup.mc$byte$count = mc$count;
531   2        call write(0, @(0DH, 0AH, ' You entered '), 15, @status);
532   2        call write$int(j, 0);
533   2        call writeln(0, @(' Multicast Address(es). '), 23, @status);

534   2    end setup$mc$parameters;


535   1    setup$configure$parameters: procedure;
536   2    declare (k, j) byte;

537   2        configure.byte$cnt = 11;
538   2        configure.info(0) = 8;
539   2        configure.info(1) = 0;
540   2        configure.info(2) = 26H;
541   2        configure.info(3) = 0;
542   2        configure.info(4) = 96;
543   2        configure.info(5) = 0;
544   2        configure.info(6) = 0F2H;
545   2        configure.info(7) = 0;
546   2        configure.info(8) = 0;
547   2        configure.info(9) = 64;
548   2        j = 0;
549   2        call write(0, @(0DH, 0AH, ' Configure command is set up for default',
                 ' values. ', 0DH, 0AH, ' Do you want to change any bytes?',
                                                    ' (Y or N) ==> '), 99, @status);
550   2        do while yes;
551   3            do while j = 0;
552   4                call write(0, @(0DH, 0AH, ' Enter byte number (1 - 11) ==> '), 34,
                                                                        @status);
553   4                j = read$int(11);
554   4                if j = 0 then
555   4                call write(0, @(0DH, 0AH, ' Illegal byte number'), 22, @status);
556   4            end;
557   3            if j = 1 then configure.byte$cnt = read$byte(j);
559   3                else configure.info(j - 2) = read$byte(j);
560   3            j = 0;
561   3            call write(0, @(0DH, 0AH, ' Any more bytes? (Y or N) ==> '), 32,
                                                                        @status);
562   3        end;
563   2        preamble = shl(1, shr((configure.info (2) and 30H), 4)+1);
564   2        address$length = configure.info(2) and 07H;
565   2        if address$length = 7 then address$length = 0;
567   2        ad$loc = shr((configure.info(2) and 08H), 3);
568   2        if shr((configure.info(7) and 20H), 5) then crc = 2; else crc = 4;
571   2        if shr((configure.info(7) and 10H), 4) then crc = 0;

573   2    end setup$configure$parameters;
```

292010-52

Traffic Simulator/Monitor Station Program (Continued)

```
574   1       setup$tx$parameters: procedure;
575   2       declare (size, i) word;

576   2           do forever;
577   3               no$transmission = false;
578   3               transmit.bd$offset = offset (@tbd.act$count);
579   3               if not ad$loc then
580   3               do;
581   4                   call write(0, @(0DH, 0AH,
                            ' Enter a destination address in Hex ==> '), 42, @status);
582   4                   call put$address(@transmit.dest$adr(0));
583   4               end;
584   3               else call writeln(0, @(' 82586 is configured to pick up DA, IA, ',
                                    ' and TYPE from TX buffer.'), 64, @status);
585   3               call cr$lf;
586   3               if not ad$loc then
587   3               do;
588   4                   call write(0, @(0DH, 0AH, ' Enter TYPE ==> '), 18, @status);
589   4                   transmit.type = read$int(0FFFFH);
590   4               end;
591   3               call writeln(0, @(0DH, 0AH,' How many bytes of transmit data?'), 35,
                                                                        @status);
592   3               call write(0, @(' Enter a number ==> '), 20, @status);
593   3               size = read$int(1518);
594   3               tbd.act$count = size or 8000H;
595   3               if size <> 0 then
596   3               do;
597   4                   tbd.link$offset = 0FFFFH;
598   4                   tbd.ad0 = offset (@tx$buffer(0));
599   4                   tbd.ad1 = 0;
600   4                   do i = 0 to 1517;
601   5                       tx$buffer(i) = i;
602   5                   end;
603   4                   call writeln(0,
                            @(0DH, 0AH, ' Transmit Data is continuous numbers (0, 1, 2, 3, ',
                                                                    ' ... )'), 57, @status);
604   4                   call write(0, @(' Change any data bytes? (Y or N) ==> '), 37,
                                                                        @status);
605   4                   do while yes;
606   5                       call write(0, @(0DH, 0AH, ' Enter a byte number ==> '),
                                                                    27, @status);
607   5                       i = read$int(size);
608   5                       call write(0, @(0DH, 0AH, ' Byte '), 8, @status);
609   5                       call out$dec$hex(i);
610   5                       call write(0, @(' currently contains '), 20, @status);
611   5                       call out$dec$hex(tx$buffer(i));
612   5                       call write(0, @('. '), 1, @status);
613   5                       tx$buffer(i) = read$byte(i);
614   5                       call write(0, @(0DH, 0AH, ' Any more bytes? (Y or N) ==> '),
                                                                    32, @status);
615   5                   end;
616   4               end;
617   3               else
                        transmit.bd$offset = 0FFFFH;
618   3               call cr$lf;
```

292010-53

Traffic Simulator/Monitor Station Program (Continued)

```
619   3              call init$186$timer0;
620   3              call write(0, @(0DH, 0AH, ' Setup a transmit terminal count?',
                                                    ' (Y or N) ==> '), 49, @status);
621   3              if yes then
622   3              do;
623   4                  stop = true;
624   4                  call write (0, @(0DH, 0AH, ' Enter a transmit',
                                                   ' terminal count ==> '), 39, @status);
625   4                  stop$count = read$int(0FFFFFFFFH);
626   4              end;
627   3              else stop = false;
628   3              call cr$lf;
629   3              call cr$lf;
630   3              call print$parameters;
631   3              call write(0, @(0DH, 0AH, ' Good enough? (Y or N) ==> '), 29,
                                                                          @status);
632   3              if yes then return;
634   3          end;

635   2      end setup$tx$parameters;


636   1      loop$char: procedure (i, j);
637   2      declare (i, j, k) byte;

638   2          do k = 1 to i;
639   3              call write(0, @j, 1, @status);
640   3          end;

641   2      end loop$char;


642   1      init: procedure;

643   2      declare  i byte;

644   2          call cr$lf;
645   2          call loop$char(13, 0AH);
646   2          call loop$char(15, ' ');
647   2          call writeln(0, @('TRAFFIC SIMULATOR AND MONITOR',
                                                ' STATION PROGRAM '), 46, @status);
648   2          call loop$char(7, 0AH);
649   2          call writeln(0, @(0DH, 0AH, ' Initialization begun'), 23, @status);
650   2          call cr$lf;
651   2          reset = true;
652   2          cur$cb$offset = 0FFFFH;
653   2          output(ESI$PORT) = NO$LOOPBACK;
654   2          output(ESI$PORT) = LOOPBACK;
655   2          dhex = false;

                /* set up interrupt logic */

656   2          call set$interrupt(INT$TYPE$586, isr);
657   2          call set$interrupt(INT$TYPE$TIMER0, tx$isr);

            $IF SBC18651
```

292010-54

Traffic Simulator/Monitor Station Program (Continued)

```
                    call set$interrupt(INT$7, isr7);
                    output (PIC$MASK$130) = ENABLE$586$186;
                    output (PIC$EOI$130) = EOI$CMDO$130;
                    output (PIC$EOI$130) = EOI$CMD4$130;
                    output (PIC$EOI$186) = EOI$CMDO$186;
                    output (PIC$VTR$186) = 30H;

                $ELSE

658   2             output (PIC$EOI$186) = EOI$CMDO$186;
659   2             output (PIC$EOI$186) = EOI$CMD4$186;
660   2             output (PIC$MASK$186) = ENABLE$586;

                $ENDIF

                    /* locate iscp */

661   2             iscp$ptr = ISCP$LOC$LO;

                    /* set up fields in ISCP */

662   2             iscp.busy = 1;
663   2             iscp.scb$b(0) = SCB$BASE$LO;
664   2             iscp.scb$b(1) = SCB$BASE$HI;
665   2             iscp.scb$o = offset (@scb.status);

                    /* set up SCB */

666   2             scb.status = 0;
667   2             scb.cbl$offset = offset (@diagnose.status);
668   2             scb.rpa$offset = offset (@rfd(0).status);
669   2             scb.crc$errs = 0;
670   2             scb.aln$errs = 0;
671   2             scb.rsc$errs = 0;
672   2             scb.ovrn$errs = 0;

                    /* set up Diagnose command */

673   2             diagnose.status = 0;
674   2             diagnose.cmd = 7;
675   2             diagnose.link$offset = offset (@configure.status);

                    /* set up CONFIGURE command */

676   2             configure.status = 0;
677   2             configure.cmd = 2;
678   2             configure.link$offset = offset (@ia$setup.status);;
680   2             call setup$configure$parameters;


                    /* set up IA command */

681   2             ia$setup.status = 0;
682   2             ia$setup.cmd = 1;
683   2             ia$setup.link$offset = offset (@mc$setup.status);
684   2             call setup$ia$parameters;
```

292010-55

Traffic Simulator/Monitor Station Program (Continued)

```
                    /* set up MC command */

685   2             mc$setup.status = 0;
686   2             mc$setup.cmd = 8003H;
687   2             mc$setup.link$offset = OFFFFH;
688   2             call setup$mc$parameters;



                    /* set up one transmit cb linked to itself */

689   2             transmit.status = 0;
690   2             call writeln(0, @(0DH, 0AH, ' Would you like to transmit?'), 30, @status);
691   2             call write(0, @(' Enter a Y or N ==> '), 20, @status);
692   2             if yes then
693   2             do;
694   3                 transmit.cmd = 8004H;
695   3                 transmit.link$offset = OFFFFH;
696   3                 transmit.bd$offset = offset (@tbd.act$count);
697   3                 call setup$tx$parameters;
698   3             end;
699   2             else no$transmission = true;


                    /* initialize receive packet area */

700   2             do i = 0 to 3;
701   3                 rfd(i).status = 0;
702   3                 rfd(i).el$s = 0;
703   3                 rfd(i).link$offset = offset (@rfd(i+1).status);
704   3                 rfd(i).bd$offset = OFFFFH;
705   3                 rbd(i).act$count = 0;
706   3                 rbd(i).next$bd$link = offset (@rbd(i+1).act$count);
707   3                 rbd(i).ad0 = offset (@rbuf(i).buffer(0));
708   3                 rbd(i).ad1 = 0;
709   3                 rbd(i).size = 1500;
710   3             end;
711   2             rfd(0).bd$offset = offset (@rbd(0).act$count);
712   2             rfd(4).status = 0;
713   2             rfd(4).el$s = 0;
714   2             rfd(4).link$offset = offset (@rfd(0).status);
715   2             rfd(4).bd$offset = OFFFFH;
716   2             rbd(4).act$count = 0;
717   2             rbd(4).next$bd$link = offset (@rbd(0).act$count);
718   2             rbd(4).ad0 = offset (@rbuf(4).buffer(0));
719   2             rbd(4).ad1 = 0;
720   2             rbd(4).size = 1500;



                    /* initialize counters */

721   2             count = 0;
722   2             receive$count = 0;
723   2             current$frame = 0;

                    /* issue the first CA */
```

292010-56

Traffic Simulator/Monitor Station Program (Continued)

```
724   2          output(CA$PORT) = CA;

725   2      end init;


726   1      print$help: procedure;

727   2          call writeln(0, @(0DH, 0AH, ' Commands are:'), 16, @status);
728   2          call writeln(0, @(0DH, 0AH, ' S - Setup CB              D - Display RFD/CB'),
                                                                        45, @status);
729   2          call writeln(0, @(' P - Print SCB           C - SCB Control CMD'), 44,
                                                                        @status);
730   2          call writeln(0, @(' L - ESI Loopback On     N - ESI Loopback Off'), 45,
                                                                        @status);
731   2          call writeln(0, @(' A - Toggle Number Base'), 23,
                                                                        @status);
732   2          call writeln(0, @(' Z - Clear Tx Frame Counter'), 27, @status);
733   2          call writeln(0, @(' Y - Clear Rx Frame Counter'), 27, @status);
734   2          call writeln(0, @(' E - Exit to Continuous Mode'), 28, @status);

735   2      end print$help;



736   1      enter$scb$cmd: procedure;
737   2      declare i byte;

              /* enter a command into the SCB */

738   2          call cr$lf;
739   2          if scb.cmd <> 0 then
740   2          do;
741   3              call writeln(0, @(' SCB command word is not cleared'), 32, @status);
742   3              call write(0, @(' Try a Channel Attention? (Y or N) ==> '),
                                                                        39, @status);
743   3              if yes then
744   3              do;
745   4                  output(CA$PORT) = CA;
746   4                  call writeln(0, @(' Issued channel attention'), 25, @status);
747   4                  call cr$lf;
748   4                  return;
749   4              end;
750   3          end;
751   2          call write(0, @(' Do you want to enter any SCB commands? (Y or N) ==> '),
                                                                        53, @status);
752   2          if not yes then return;
754   2          call write(0, @(0DH, 0AH, ' Enter CUC ==> '), 17, @status);
755   2          i = read$int(4);
756   2          scb.cmd = scb.cmd or shl(double(i), 8);
757   2          if i = 1 then scb.cbl$offset = cur$cb$offset;
759   2          call write(0, @(0DH, 0AH, ' Enter RES bit ==> '), 21, @status);
760   2          i = read$bit;
761   2          scb.cmd = scb.cmd or shl(i, 7);
762   2          call write(0, @(0DH, 0AH, ' Enter RUC ==> '), 17, @status);
763   2          i = read$int(4);
764   2          scb.cmd = scb.cmd or shl(i, 4);
```

292010-57

Traffic Simulator/Monitor Station Program (Continued)

```
765   2          if (((scb.cbl$offset = offset (@transmit.status))
                     and ((scb.cmd and 0100H) = 0100H)) or ((scb.cmd and 0010H) = 0010H))
                     and not ((scb.cmd and 0080H) = 0080H)
766   2               then goback = 1;
767   2          call writeln(0, @(0DH, 0AH, ' Issued Channel Attention'), 27, @status);
768   2          call cr$lf;
769   2          output(CA$PORT) = CA;

770   2      end enter$scb$cmd;


771   1      print$type$help: procedure;

772   2          call writeln(0, @(0DH, 0AH, 0AH, 'Command block type:'), 22, @status);
773   2          call writeln(0, @(' N - Nop                I - IA Setup'), 35, @status);
774   2          call writeln(0, @(' C - Configure          M - MA Setup'), 35, @status);
775   2          call writeln(0, @(' T - Transmit           R - TDR'), 30, @status);
776   2          call writeln(0, @(' D - Diagnose           S - Dump Status'), 38, @status);
777   2          call writeln(0, @(' H - Print this message'), 23, @status);

778   2      end print$type$help;


779   1      setup$cb: procedure;
780   2      declare (t, valid) byte;

781   2          valid = false;
782   2          do while not valid;
783   3              call write(0, @(0DH, 0AH, ' Enter command block type (H for',
                                                ' help) ==> '), 45, @status);
784   3              t = read$char;
785   3              if (t <> 'H') and (t <> 'h') and (t <> 'T') and (t <> 't') and
                       (t <> 'N') and (t <> 'n') and (t <> 'R') and (t <> 'r') and
                       (t <> 'D') and (t <> 'd') and (t <> 'C') and (t <> 'c') and
                       (t <> 'I') and (t <> 'i') and (t <> 'M') and (t <> 'm') and
                       (t <> 'S') and (t <> 's') then
786   3              call write(0, @(0DH, 0AH, ' Illegal command block type'), 29,
                                                                        @status);
787   3              else
788   3              if (t = 'H') or (t = 'h') then call print$type$help;
789   3              else valid = true;
790   3          end;
791   2          if (t = 'N') or (t = 'n') then
792   2          do;
793   3              cur$cb$offset = offset (@nop.status);
794   3              nop.status = 0;
795   3              nop.cmd = 8000H;
796   3              nop.link$offset = OFFFFH;
797   3          end;
798   2          if (t = 'I') or (t = 'i') then
799   2          do;
800   3              cur$cb$offset = offset (@ia$setup.status);
801   3              ia$setup.status = 0;
802   3              ia$setup.cmd = 8001H;
803   3              ia$setup.link$offset = OFFFFH;
804   3              call setup$ia$parameters;
805   3          end;
```

292010-58

Traffic Simulator/Monitor Station Program (Continued)

```
806   2           if (t = 'C') or (t = 'c') then
807   2           do;
808   3               cur$cb$offset = offset (@configure.status);
809   3               configure.status = 0;
810   3               configure.cmd = 8002H;
811   3               configure.link$offset = 0FFFFH;
812   3               call setup$configure$parameters;
813   3           end;
814   2           if (t = 'M') or (t = 'm') then
815   2           do;
816   3               cur$cb$offset = offset (@mc$setup.status);
817   3               mc$setup.status = 0;
818   3               mc$setup.cmd = 8003H;
819   3               mc$setup.link$offset = 0FFFFH;
820   3               call setup$mc$parameters;
821   3           end;
822   2           if (t = 'T') or (t = 't') then
823   2           do;
824   3               cur$cb$offset = offset (@transmit.status);
825   3               transmit.status = 0;
826   3               transmit.cmd = 8004H;
827   3               transmit.link$offset = 0FFFFH;
828   3               call setup$tx$parameters;
829   3           end;
830   2           if (t = 'R') or (t = 'r') then
831   2           do;
832   3               cur$cb$offset = offset (@tdr.status);
833   3               tdr.status = 0;
834   3               tdr.cmd = 8005H;
835   3               tdr.link$offset = 0FFFFH;
836   3               tdr.result = 0;
837   3           end;
838   2           if (t = 'S') or (t = 's') then
839   2           do;
840   3               cur$cb$offset = offset (@dump.status);
841   3               dump.status = 0;
842   3               dump.cmd = 8006H;
843   3               dump.link$offset = 0FFFFH;
844   3               dump.buff$ptr = offset (@dump$area(0));
845   3           end;
846   2           if (t = 'D') or (t = 'd') then
847   2           do;
848   3               cur$cb$offset = offset (@diagnose.status);
849   3               diagnose.status = 0;
850   3               diagnose.cmd = 8007H;
851   3               diagnose.link$offset = 0FFFFH;
852   3           end;

853   2       end setup$cb;



854   1       display$command$block: procedure;
855   2       declare (i, j) byte,
                     wh pointer,
                     sel selector,
                     w word;
```

292010-59

Traffic Simulator/Monitor Station Program (Continued)

```
856   2          call cr$lf;
857   2          if cur$cb$offset = OFFFFH then
858   2              call write(0, @(' No Command Block to display'), 28, @status);
859   2          if cur$cb$offset = offset (@nop.status) then
860   2          do;
861   3              call write(0, @('---NOP Command Block---'), 23, @status);
862   3              call print$wds(@nop.status, 3);
863   3          end;
864   2          if cur$cb$offset = offset (@tdr.status) then
865   2          do;
866   3              call write(0, @('---TDR Command Block---'), 23, @status);
867   3              call print$wds(@tdr.status, 4);
868   3          end;
869   2          if cur$cb$offset = offset (@diagnose.status) then
870   2          do;
871   3              call write(0, @('---Diagnose Command Block---'), 28, @status);
872   3              call print$wds(@diagnose.status, 3);
873   3          end;
874   2          if cur$cb$offset = offset (@transmit.status) then
875   2          do;
876   3              call write(0, @('---Transmit Command Block---'), 28, @status);
877   3              if not address$length then i = address$length;
879   3              else i = address$length + 1;
880   3              if ad$loc then call print$wds(@transmit.status, 4);
882   3              else call print$wds(@transmit.status, i/2+1);
883   3              call cr$lf;
884   3              call cr$lf;
885   3              if transmit.bd$offset <> OFFFFH then
886   3              do;
887   4                  call write(0, @('---Transmit Buffer Descriptor---'), 33, @status);
888   4                  call print$wds(@tbd.act$count, 4);
889   4                  call write(0, @(ODH, OAH, OAH,
                            ' Display the transmit buffer? (Y or N) ==> '), 46, @status);
890   4                  if yes then
891   4                  do;
892   5                      call cr$lf;
893   5                      call writeln(0, @(' Transmit Buffer: '), 17, @status);
894   5                      w = tbd.act$count and 3FFFH;
895   5                      call print$buff(@tx$buffer(0), w);
896   5                  end;
897   4              end;
898   3          end;
899   2          if cur$cb$offset = offset (@ia$setup.status) then
900   2          do;
901   3              call write(0, @('---IA Setup Command Block---'), 28, @status);
902   3              call print$wds(@ia$setup.status, 6);
903   3          end;
904   2          if cur$cb$offset = offset (@configure.status) then
905   2          do;
906   3              call write(0, @('---Configure Command Block---'), 29, @status);
907   3              call print$wds(@configure.status, 9);
908   3          end;
909   2          if cur$cb$offset = offset (@mc$setup.status) then
910   2          do;
911   3              call write(0, @('---MC Setup Command Block---'), 28, @status);
912   3              i = 4 + mc$count/2;
```

292010-60

Traffic Simulator/Monitor Station Program (Continued)

```
913   3              if mc$count > 24 then
914   3                  do;
915   4                      call print$wds(@mc$setup.status, 16);
916   4                      call pause;
917   4                      i = i - 16;
918   4                      call print$wds(@mc$setup.mc$address(8), i);
919   4                  end;
920   3              else call print$wds(@mc$setup.status, i);
921   3          end;
922   2          if cur$cb$offset = offset (@dump.status) then
923   2          do;
924   3              call write(0, @('---Dump Status Command Block---'), 31, @status);
925   3              call print$wds(@dump.status, 4);
926   3              if dump.status = 0A000H then
927   3              do;
928   4                  call writeln(0, @(0DH, 0AH, ' Dump Status Results'), 22, @status)
929   4                  call write$offset(@dump$area(0));
930   4                  call cr$lf;
931   4                  do i = 0 to 9;
932   5                      call print$str(@dump$area(16*i), 16);
933   5                  end;
934   4                  call print$str(@dump$area(160), 10);
935   4                  call cr$lf;
936   4              end;
937   3          end;

938   2      end display$command$block;


939   1      display$receive$area: procedure;
940   2      declare (i, k, j, l) byte,
                  chars(4) byte;

941   2          call writeln(0, @(0DH, 0AH, ' Frame Descriptors:'), 21, @status);
942   2          if ad$loc then
943   2          do;
944   3              call writeln(0, @(0DH, 0AH, ' DA, SA, and TYPE are in buffer.', 0DH,
                                                                        0AH), 36, @status);
945   3              j = 3;
946   3          end;
947   2          else j = address$length + 4;
948   2          do k = 0 to j;
949   3              do i = 0 to 4;
950   4                  call out$word(@rfd(i).status, k);
951   4                  if k = 0 then call write$offset(@rfd(i).status);
953   4                      else call loop$char(10, ' ');
954   4              end;
955   3              call cr$lf;
956   3          end;
957   2          call writeln(0, @(0DH, 0AH, 0AH, ' Receive Buffer Descriptors:'), 31,
                                                                            @status);
958   2          do k = 0 to 4;
959   3              do i = 0 to 4;
960   4                  call out$word(@rbd(i).act$count, k);
961   4                  if k = 0 then call write$offset(@rbd(i).act$count);
963   4                      else call loop$char(10, ' ');
964   4              end;
```

292010-61

Traffic Simulator/Monitor Station Program (Continued)

```
 965   3              call cr$lf;
 966   3          end;
 967   2          call write(0, @(ODH, OAH, OAH, ' Display the receive',
                                          ' buffers? (Y or N) ==> '), 46, @status);
 968   2          if not yes then return;
 970   2          call writeln(0, @(ODH, OAH, ' Receive Buffers.'), 19, @status);
 971   2          do i = 0 to 4;
 972   3              call write(0, @(ODH, OAH, ' Receive Buffer '), 18, @status);
 973   3              call write$int(i, 0);
 974   3              call writeln(0, @(' : '), 2, @status);
 975   3              k = rbd(i).act$count and 3FFFH;
 976   3              call print$buff(@rbuf(i).buffer(0), k);
 977   3              call pause;
 978   3          end;

 979   2      end display$receive$area;


 980   1      display$cb$rpa: procedure;
 981   2      declare i byte;

 982   2          call write(0, @(ODH, OAH, ' Command Block or Receive Area? (R or C) ==> '),
                                                                        47, @status);
 983   2          i = read$char;
 984   2          do while (i <> 'R') and (i <> 'r') and (i <> 'C') and (i <> 'c');
 985   3              call writeln(0, @(ODH, OAH, ' Illegal command'), 18, @status);
 986   3              call write(0, @(' Enter R or C ==> '), 18, @status);
 987   3              i = read$char;
 988   3          end;
 989   2          if (i = 'R') or (i = 'r') then call display$receive$area;
 991   2          else call display$command$block;

 992   2      end display$cb$rpa;


 993   1      process$cmd: procedure;
 994   2      declare (b, i) byte;

 995   2          goback = 0;
 996   2          b = read$char;
 997   2          call cr$lf;
 998   2          if (b <> 'H') and (b <> 'h') and (b <> 'S') and (b <> 's') and
                     (b <> 'D') and (b <> 'd') and (b <> 'P') and (b <> 'p') and
                     (b <> 'C') and (b <> 'c') and (b <> 'E') and (b <> 'e') and
                     (b <> 'L') and (b <> 'l') and (b <> 'N') and (b <> 'n') and
                     (b <> 'Z') and (b <> 'z') and (b <> 'Y') and (b <> 'y') and
                     (b <> 'A') and (b <> 'a') then
 999   2          call write(0, @(' Illegal command'), 16, @status);
1000   2          if (b = 'H') or (b = 'h') then call print$help;
1002   2          if (b = 'A') or (b = 'a') then
1003   2              if dhex then
1004   2              do;
1005   3                  dhex = false;
1006   3                  call write(0, @(' Counters are displayed in decimal.'), 35,
                                                                        @status);
1007   3              end;
1008   2              else
```

292010-62

Traffic Simulator/Monitor Station Program (Continued)

```
                    do;
1009    3                   dhex = true;
1010    3                   call write(0, @(' Counters are displayed in hexadecimal. '), 39,
                                                                                    @status);
1011    3               end;
1012    2           if (b = 'L') or (b = '1') then
1013    2           do;
1014    3               output(ESI$PORT) = LOOPBACK;
1015    3               call write(0, @(' ESI is in Loopback Mode. '), 25, @status);
1016    3           end;
1017    2           if (b = 'N') or (b = 'n') then
1018    2           do;
1019    3               output(ESI$PORT) = NO$LOOPBACK;
1020    3               call write(0, @(' ESI is NOT in Loopback Mode. '), 29, @status);
1021    3           end;
1022    2           if (b = 'Z') or (b = 'z') then
1023    2           do;
1024    3               count = 0;
1025    3               call write(0, @(' Transmit Frame Counter is cleared. '), 35, @status);
1026    3           end;
1027    2           if (b = 'Y') or (b = 'y') then
1028    2           do;
1029    3               receive$count = 0;
1030    3               scb.crc$errs, scb.aln$errs, scb.rsc$errs, scb.ovrn$errs = 0;
1031    3               call write(0, @(' Receive Frame Counter is cleared. '), 34, @status);
1032    3           end;
1033    2           if (b = 'C') or (b = 'c') then call enter$scb$cmd;
1035    2           if (b = 'S') or (b = 's') then call setup$cb;
1037    2           if (b = 'P') or (b = 'p') then call print$scb;
1039    2           if (b = 'D') or (b = 'd') then call display$cb$rpa;
1041    2           if (b = 'E') or (b = 'e') then goback = 1;
1043    2           call cr$lf;

1044    2       end process$cmd;


1045    1       getout: procedure;
1046    2       declare b byte;

1047    2           b = read$char;
1048    2           goback = 0;
1049    2           call write(0, @(0DH, 0AH, ' Enter command (H for help) ==> '), 34,
                                                                                    @status);
1050    2           do forever;
1051    3               if csts then
1052    3               do;
1053    4                   disable;
1054    4                   call process$cmd;
1055    4                   enable;
1056    4                   if goback then return;
1058    4                   call write(0, @(0DH, 0AH, ' Enter command (H for help) ==> '), 34,
                                                                                    @status);
1059    4               end;
1060    3           end;

1061    2       end getout;
```

292010-63

Traffic Simulator/Monitor Station Program (Continued)

```
1062   1       update: procedure;
1063   2       declare i byte;

1064   2           call cr$lf;
1065   2           call loop$char(10, 0AH);
1066   2           call loop$char(28, '*');
1067   2           call write(0, @(' Station Configuration '), 23, @status);
1068   2           call loop$char(27, '*');
1069   2           call cr$lf;
1070   2           call cr$lf;
1071   2           call write(0, @(' Host Address: '), 15, @status);
1072   2           call print$network$addr(@ia$setup.ia$address(0));
1073   2           i = 0;
1074   2           call write(0, @(' Multicast Address(es): '), 24, @status);
1075   2           if mc$setup.mc$byte$count = 0
1076   2           then call writeln(0, @('No Multicast Addresses Defined'), 30, @status);
1077   2           else
                        do while i < mc$setup.mc$byte$count;
1078   3                       call print$network$addr(@mc$setup.mc$address(i));
1079   3                       call loop$char(24, ' ');
1080   3                       i = i + 6;
1081   3                   end;
1082   2           call write(0, @(0DH), 1, @status);
1083   2           if not no$transmission then call print$parameters;
1085   2           call write(0, @(' 82586 Configuration Block: '), 28, @status);
1086   2           call print$str(@configure.info(0), 10);
1087   2           call cr$lf;
1088   2           call loop$char(29, '*');
1089   2           call write(0, @(' Station Activities '), 20, @status);
1090   2           call loop$char(29, '*');
1091   2           call cr$lf;
1092   2           call cr$lf;
1093   2           call writeln(0,
                  @(' # of Good    # of Good    CRC          Alignment    No           Receive'),
                                                                                    73, @status);
1094   2           call writeln(0,
                  @(' Frames       Frames       Errors       Errors       Resource     Overrun'),
                                                                                    73, @status);
1095   2           call writeln(0,
                  @(' Transmitted  Received                                 Errors       Errors'),
                                                                                    72, @status);

1096   2       end update;


1097   1       main:

                    call init;
1098   1           enable;
1099   1           do while reset;
1100   2           end;
1101   1           disable;
1102   1           scb.cmd = 0100H;
1103   1           output(CA$PORT) = CA;
1104   1           call wait$scb;
1105   1           enable;
```

292010-64

Traffic Simulator/Monitor Station Program (Continued)

```
1106   1          do while (diagnose.status and 8000H) <> 8000H;
1107   2          end;
1108   1          call cr$lf;
1109   1          if diagnose.status <> 0A000H
1110   1          then call writeln(0, @(' Diagnose failed!'), 17 , @status);
1111   1          if configure.status <> 0A000H
1112   1          then call writeln(0, @(' Configure failed!'), 18 , @status);
1113   1          if ia$setup.status <> 0A000H
1114   1          then call writeln(0, @(' IA Setup failed!'), 17 , @status);
1115   1          if mc$setup.status <> 0A000H
1116   1          then call writeln(0, @(' MC Setup failed!'), 17 , @status);
1117   1          scb.cbl$offset = offset (@transmit.status);
1118   1          call writeln(0, @(0DH, 0AH, ' Receive Unit is active.'), 26, @status);
1119   1          disable;
1120   1          scb.cmd = 0010H;
1121   1          output(CA$PORT) = CA;
1122   1          call wait$scb;
1123   1          enable;
1124   1          output(ESI$PORT) = NO$LOOPBACK;
1125   1          call cr$lf;
1126   1          if not no$transmission then
1127   1          do;
1128   2              call write(0, @('---Transmit Command Block---'), 28, @status);
1129   2              call print$wds(@transmit.status, 8);
1130   2              call cr$lf;
1131   2              cur$cb$offset = offset (@transmit.status);
1132   2              call pause;
1133   2              do z = 1 to 60;
1134   3                  call time(250);
1135   3              end;
1136   2              call writeln(0, @(0DH, 0AH, 'transmission started!'), 23, @status);
1137   2              call cr$lf;
1138   2              disable;
1139   2              scb.cmd = 0100H;
1140   2              output (CA$PORT) = CA;
1141   2              call wait$scb;
1142   2              enable;
1143   2          end;
1144   1          call update;
1145   1          do forever;
1146   2              call write(0, @(0DH, ' '), 2, @status);
1147   2              do y = 0 to 5;
1148   3                  do case y;
1149   4                      call write$int(count, dhex);
1150   4                      call write$int(receive$count, dhex);
1151   4                      call write$int(scb.crc$errs, dhex);
1152   4                      call write$int(scb.aln$errs, dhex);
1153   4                      call write$int(scb.rsc$errs, dhex);
1154   4                      call write$int(scb.ovrn$errs, dhex);
1155   4                  end;
1156   3                  char$count = 13 - char$count;
1157   3                  call loop$char(char$count, ' ');
1158   3              end;
1159   2              if csts then
1160   2              do;
1161   3                  disable;
1162   3                  call getout;                                    292010-65
1163   3                  call update;
1164   3              end;
1165   2          end;

1166   1      end tsms;


MODULE INFORMATION:

     CODE AREA SIZE     = 23C3H   9155D
     CONSTANT AREA SIZE = 0F85H   3973D
     VARIABLE AREA SIZE = 265EH   9822D
     MAXIMUM STACK SIZE = 0092H    146D
     1994 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

DICTIONARY SUMMARY:

     159KB MEMORY AVAILABLE
     23KB MEMORY USED   (14%)
     0KB DISK SPACE USED

END OF PL/M-86 COMPILATION                                           292010-66
```

Traffic Simulator/Monitor Station Program (Continued)

```
/*****************************************************************************
/*                                                                           */
/*          186/586 High Integration Board Initialization Routine            */
/*          (This driver is configured for Ethernet/Cheapernet Design        */
/*           Kit Demo Board)                                                 */
/*                                                                           */
/*          Ver. 2.0                          March 14, 1986                 */
/*                                                                           */
/*          Kiyoshi Nishide                   Intel Corporation              */
/*                                                                           */
/*****************************************************************************/

/* The conditional compilation parameter 'EPROM27128' determines board ROM
   size.  If it is true, the 80186's wait state generator is programmed to
   0 wait state for upper 64K-byte memory locations.  If it is false, the
   wait state generator is programmed to 0 wait state for upper 128K-byte
   memory locations   */


1       ini186:

        do;

2   1   declare hib_ir label public;
3   1   declare main label external;
4   1   declare menu label external;

        /* literals */

5   1   declare lit       literally 'literally',
                UMCS_reg        lit '0FFA0H',
                LMCS_reg        lit '0FFA2H',
                PACS_reg        lit '0FFA4H',
                MPCS_reg        lit '0FFA8H',
                INT_MASK_reg    lit '0FF28H',
                ISCP$LOC$LO     lit '03FF8H',
                ISCP$LOC$HI     lit '0',
                SCC_CH_B_CMD    lit '8300H',
                SCC_CH_B_DATA   lit '8302H',
                SCC_CH_A_CMD    lit '8304H',
                SCC_CH_A_DATA   lit '8306H',
                NUL             lit '0',
                CR              lit '0DH',
                LF              lit '0AH',
                BS              lit '08H',
                SP              lit '20H',
                QM              lit '3FH',
                DEL             lit '07FH',
                BEL             lit '07H';
```

292010-67

186/586 High Integration Board Initialization Routine

```
                /* System Configuration Pointer */

 6   1          declare scp structure
                        (
                        sysbus byte,
                        unused (5) byte,
                        iscp$addr$lo word,
                        iscp$addr$hi word
                        )
                        at (OFFFF6H) data (0, 0, 0, 0, 0, 0, ISCP$LOC$LO, ISCP$LOC$HI);


 7   1          init$int$clt: procedure;

 8   2                  output(INT_mask_reg) = OFFH; /* mask all interrupts */

 9   2          end init$int$clt;


10   1          rra: procedure (reg_no) byte;
11   2          declare reg_no byte;

12   2              if (reg_no and OFH) <> 0 then output(SCC_CH_A_CMD) = reg_no and OFH;
14   2              return input(SCC_CH_A_CMD);

15   2          end rra;


16   1          rrb: procedure (reg_no) byte;
17   2          declare reg_no byte;

18   2              if (reg_no and OFH) <>0 then output (SCC_CH_B_CMD) = reg_no and OFH;
20   2              return input(SCC_CH_B_CMD);

21   2          end rrb;


22   1          wra: procedure (reg_no, value);
23   2          declare (reg_no, value) byte;

24   2              if (reg_no and OFH) <> 0 then output (SCC_CH_A_CMD) = reg_no and OFH;
26   2              output (SCC_CH_A_CMD) = value;

27   2          end wra;

28   1          wrb: procedure (reg_no, value);
29   2          declare (reg_no, value) byte;

30   2              if (reg_no and OFH) <> 0 then output (SCC_CH_B_CMD) = reg_no and OFH;
32   2              output (SCC_CH_B_CMD) = value;

33   2          end wrb;

34   1          init$SCC$B: procedure;

35   2              call wrb(09, 01000000b);  /* channel B reset */
```

292010-68

186/586 High Integration Board Initialization Routine (Continued)

```
 36   2          call wrb(04, 01001110b);  /* 2 stop, no parity, brf = 16x */
 37   2          call wrb(03, 11000000b);  /* rx 8 bits/char, no auto-enable */
 38   2          call wrb(05, 01100000b);  /* tx 8 bits/char */
 39   2          call wrb(10, 00000000b);
 40   2          call wrb(11, 01010110b);  /* rxc = txc = BRG, trxc = BRG out */
 41   2          call wrb(12, 00001011b);  /* baud rate = 9600 */
 42   2          call wrb(13, 00000000b);
 43   2          call wrb(14, 00000011b);  /* BRG source = SYS CLK, enable BRG */
 44   2          call wrb(15, 00000000b);  /* all ext status interrupts off */

 45   2          call wrb(03, 11000001b);  /* scc-b receive enable */
 46   2          call wrb(05, 11101010b);  /* scc-b transmit enable, dtr on, rts on */

 47   2      end init$SCC$B;


 48   1      c$in: procedure byte public;

 49   3          do while (input(SCC_CH_B_CMD) and 1) = 0; end;
 51   2          return (input(SCC_CH_B_DATA));

 52   2      end c$in;


 53   1      c$out: procedure (char) public;
 54   2      declare char byte;

 55   3          do while (input(SCC_CH_B_CMD) and 4) = 0; end;
 57   2          output(SCC_CH_B_DATA) = char;

 58   2      end c$out;



 59   1      read: procedure (file$id, msg$ptr, count, actual$ptr, status$ptr) public;
 60   2      declare file$id word,
                  msg$ptr pointer,
                  count word,
                  actual$ptr pointer,
                  status$ptr pointer,
                  msg based msg$ptr (1) byte,
                  buf (200) byte,
                  actual based actual$ptr word,
                  status based status$ptr word,
                  i word,
                  ch byte;

              /* This procedure implements the ISIS read procedure. All control characters */
              /* except LF, BS, and DEL are ignored. If BS or DEL is encountered, a        */
              /* backspace is done.                                                        */

 61   2          status = 0;
 62   2          i, ch = 0;
 63   2          do while (ch <> CR) and (ch <> LF) and (i < 198);
 64   3              ch = c$in and 07FH;
 65   3              if (ch = BS) or (ch = DEL) then
 66   3                  do;
```

292010-69

186/586 High Integration Board Initialization Routine (Continued)

```
 67   4                  if i > 0 then
 68   4                  do;
 69   5                      i = i - 1;
 70   5                      call c$out(DEL);
 71   5                      call c$out(BS);
 72   5                      call c$out(SP);
 73   5                      call c$out(DEL);
 74   5                      call c$out(BS);
 75   5                  end;
 76   4                  else
                             call c$out(BEL);
 77   4              end;
 78   3              else
                         if ch >= SP then
 79   3                      do;
 80   4                          call c$out(ch);
 81   4                          buf(i) = ch;
 82   4                          i = i + 1;
 83   4                      end;
 84   3              else
                         if (ch = CR) or (ch = LF) then
 85   3                      do;
 86   4                          buf(i) = CR;
 87   4                          buf(i + 1) = LF;
 88   4                          i = i + 2;
 89   4                      end;
 90   3              else
                         call c$out(BEL);
 91   3          end;
 92   2          call c$out(CR);
 93   2          if i > count then i = count;
 95   2          actual = i;
 96   2          do i = 0 to actual - 1;
 97   3              msg(i) = buf(i);
 98   3          end;

 99   2      end read;


100   1      csts: procedure byte public;

101   2          return ((input(SCC_CH_B_CMD) and 1) <> 0);

102   2      end csts;


103   1      write: procedure (file$id, msg$ptr, count, status$ptr) public;
104   2      declare (file$id, count) word,
                     (msg$ptr, status$ptr) pointer,
                     msg based msg$ptr (1) byte,
                     status based status$ptr word,
                     ch byte,
                     i word;

              /* This procedure implements the ISIS write */

105   2          status = 0;
```

292010-70

186/586 High Integration Board Initialization Routine (Continued)

```
106   2          i = 0;
107   2          do while i < count;
108   3              ch = msg(i);
109   3              if ((ch >= SP) and (ch < DEL)) or (ch = CR) or (ch = LF) or (ch = NUL)
                      then
110   3                  call c$out(ch);
111   3              else
                          call c$out(QM);
112   3              i = i + 1;
113   3          end;

114   2      end write;


115   1      hib_ir:

             $IF EPROM27128

                     output(UMCS_reg) = 0F038H;    /* Starting Address = 0F0000H,
                                                      no wait state */

             $ELSE

                     output(UMCS_reg) = 0E038H;    /* Starting Address = 0E0000H,
                                                      no wait state */

             $ENDIF

116   1                 output(LMCS_reg) = 03FCH;     /* 16K, no wait state */
117   1                 output(PACS_reg) = 083CH;     /* PBA = 8000H, no wait state for
                                                         PSC0-3 */
118   1                 output(MPCS_reg) = 0BFH;      /* Peripherals in I/O space, no A1 & A2
                                                         provided, 3 wait states for PSC4-6 */

119   1                 call init$int$clt;
120   1                 call init$SCC$B;
121   1                 go to main;

122   1      end ini186;
```

292010-71

186/586 High Integration Board Initialization Routine (Continued)

APPENDIX C
THE 82530 SCC - 80186 INTERFACE AP BRIEF

INTRODUCTION

The object of this document is to give the 82530 system designer an in-depth worst case design analysis of the typical interface to a 80186 based system. This document has been revised to include the new specifications for the 6 MHz 82530. The new specifications yield better margins and a 1 wait state interface to the CPU (2 wait states are required for DMA cycles). These new specifications will appear in the 1987 data sheet and advanced specification information can be obtained from your local Intel sales office. The following analysis includes a discussion of how the interface TTL is utilized to meet the timing requirements of the 80186 and the 82530. In addition, several optional interface configurations are also considered.

INTERFACE OVERVIEW

The 82530 - 80186 interface requires the TTL circuitry illustrated in Figure 1. Using five 14 pin TTL packages, 74LS74, 74AS74, 74AS08, 74AS04, and 74LS32, the following operational modes are supported:

- Polled
- Interrupt in vectored mode
- Interrupt in non-vectored mode
- Half-duplex DMA on both channels
- Full-duplex DMA on channel A

A brief description of the interface functional requirements during the five possible BUS operations follows below.

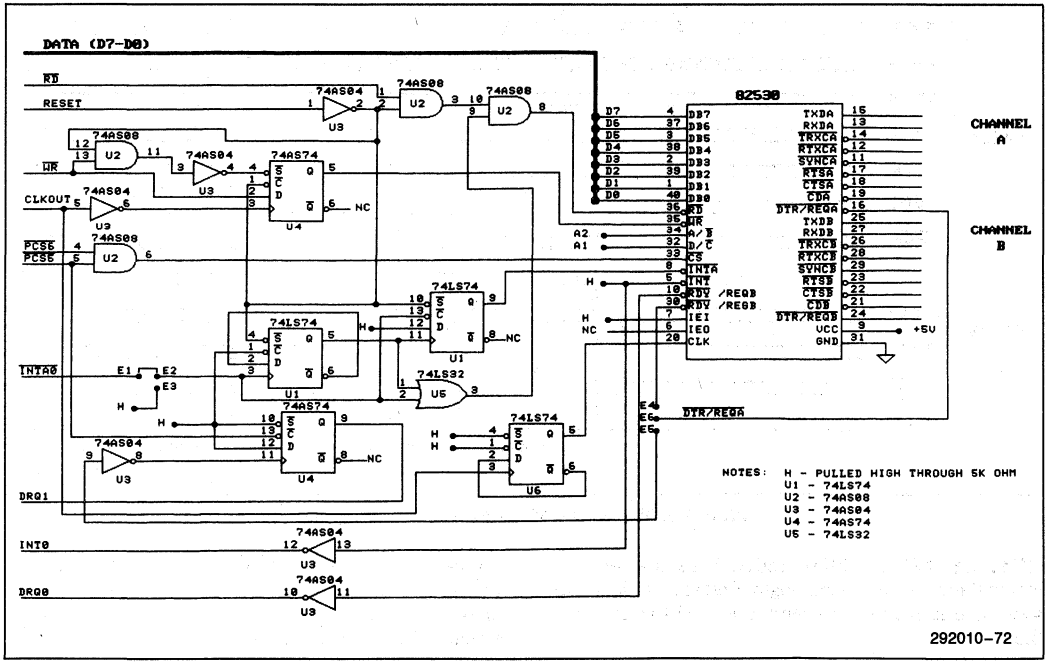


Figure 1. 82530–80186 Interface

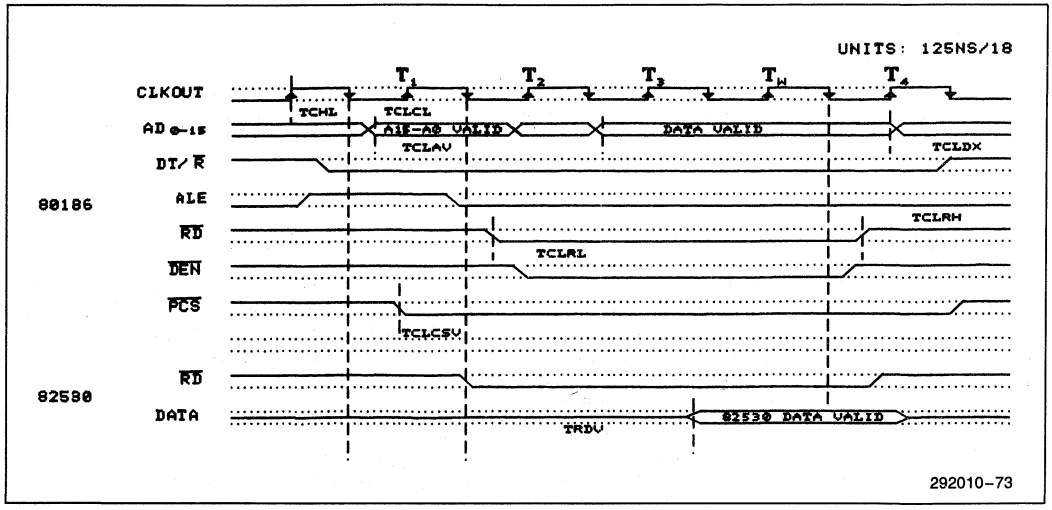


Figure 2. 80186–82530 Interface Read Cycle

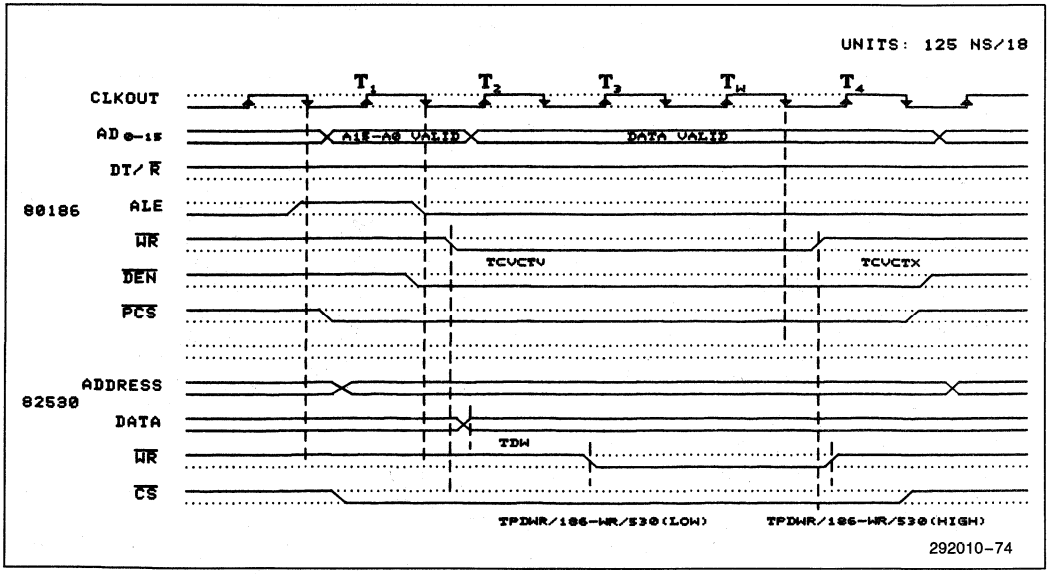


Figure 3. 80186–82530 Interface Write Cycle

READ CYCLE: The 80186 read cycle requirements are met without any additional logic, Figure 2. At least one wait state is required to meet the 82530 tAD access time.

WRITE CYCLE: The 82530 requires that data must be valid while the $\overline{WR}$ pulse is low, Figure 3. A D Flip-Flop delays the leading edge of $\overline{WR}$ until the falling edge of CLOCKOUT when data is guaranteed valid and $\overline{WR}$ is guaranteed active. The CLOCKOUT signal is inverted to assure that $\overline{WR}$ is active low before the D Flip-Flop is clocked. No wait states are necessary to meet the 82530's $\overline{WR}$ cycle requirements, but one is assumed from the $\overline{RD}$ cycle.

INTA CYCLE: During an interrupt acknowledge cycle, the 80186 provides two $\overline{INTA}$ pulses, one per bus cycle, separated by two idle states. The 82530 expects only one long $\overline{INTA}$ pulse with a $\overline{RD}$ pulse occurring only after the 82530 $\overline{IEI}/\overline{IEO}$ daisy chain settles. As

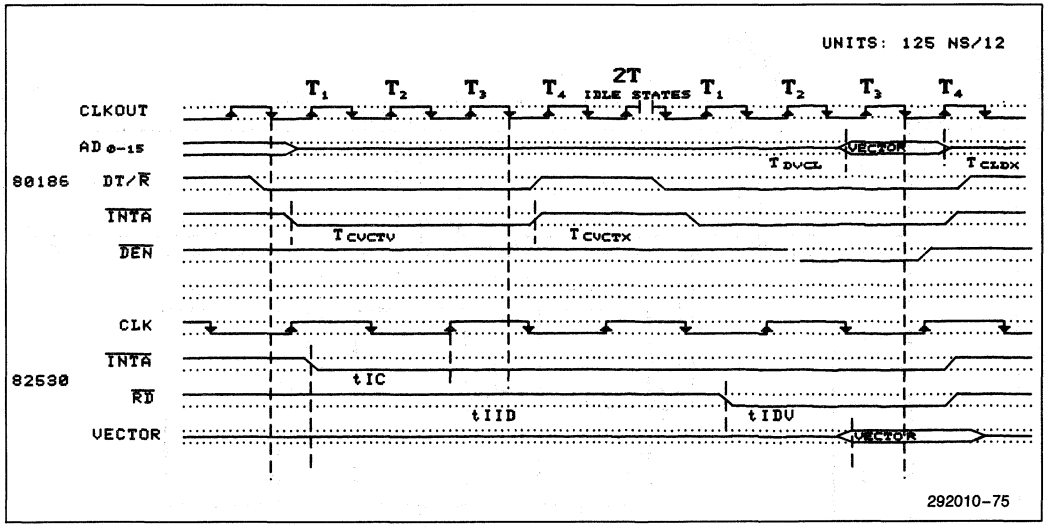


Figure 4. 82530–80186 INTA Cycle

illustrated in Figure 4, the INTA signal is sampled on the rising edge of CLK (82530). Two D Flip-Flops and two TTL gates, U2 and U5, are implemented to generate the proper INTA and RD pulses. Also, the INT signal is passively pulled high, through a 1 k resistor, and inverted through U3 to meet the 80186's active high requirement.

DMA CYCLE: Conveniently, the 80186 DMA cycle timings are the same as generic read and write operations. Therefore, with two wait states, only two modifications to the DMA request signals are necessary. First, the RDYREQA signal is inverted through U3 similar to the INT signal, and second the DTR/REQA signal is conditioned through a D Flip-Flop to prevent inadvertent back to back DMA cycles. Because the 82530 DTR/REQA signal remains active low for over five CLK (82530)'s, an additional DMA cycle could occur. This uncertain condition is corrected when U4 resets the DTR/REQ signal inactive high. Full Duplex on both DMA channels can easily be supported with one extra D Flip-Flop and an inverter.

RESET: The 82530 does not have a dedicated RESET input. Instead, the simultaneous assertion of both RD and WR causes a hardware reset. This hardware reset is implemented through U2, U3, and U4.

ALTERNATIVE INTERFACE CONFIGURATIONS

Due to its wide range of applications, the 82530 interface can have many varying configurations. In most of these applications the supported modes of operation need not be as extensive as the typical interface used in this analysis. Two alternative configurations are discussed below.

8288 BUS CONTROLLER: An 80186 based system implementing an 8288 bus controller will not require the preconditioning of the WR signal through the D Flip-Flop U4. When utilizing an 8288, the control signal IOWC does not go active until data is valid, therefore, meeting the timing requirements of the 82530. In such a configuration, it will be necessary to logically OR the IOWC with reset to accommodate a hardware reset operation.

NON-VECTORED INTERRUPTS: If the 82530 is to be operated in the non-vectored interrupt mode (B step only), the interface will not require U1 or U5. Instead, INTA on the 82530 should be pulled high, and pin 3 of U2 (RD AND RESET) should be fed directly into the RD input of the SCC.

Obviously, the amount of required interface logic is application dependent and in many cases can be considerably less than required by the typical configuration, supporting all modes of SCC operation.

DESIGN ANALYSIS

This design analysis is for a typical microprocessor system, pictured in Figure 5. The Timing analysis assumes an 8 MHz 80186 and a 4 MHz 82530. Also, included in the analysis are bus loading, and TTL-MOS compatibility considerations.

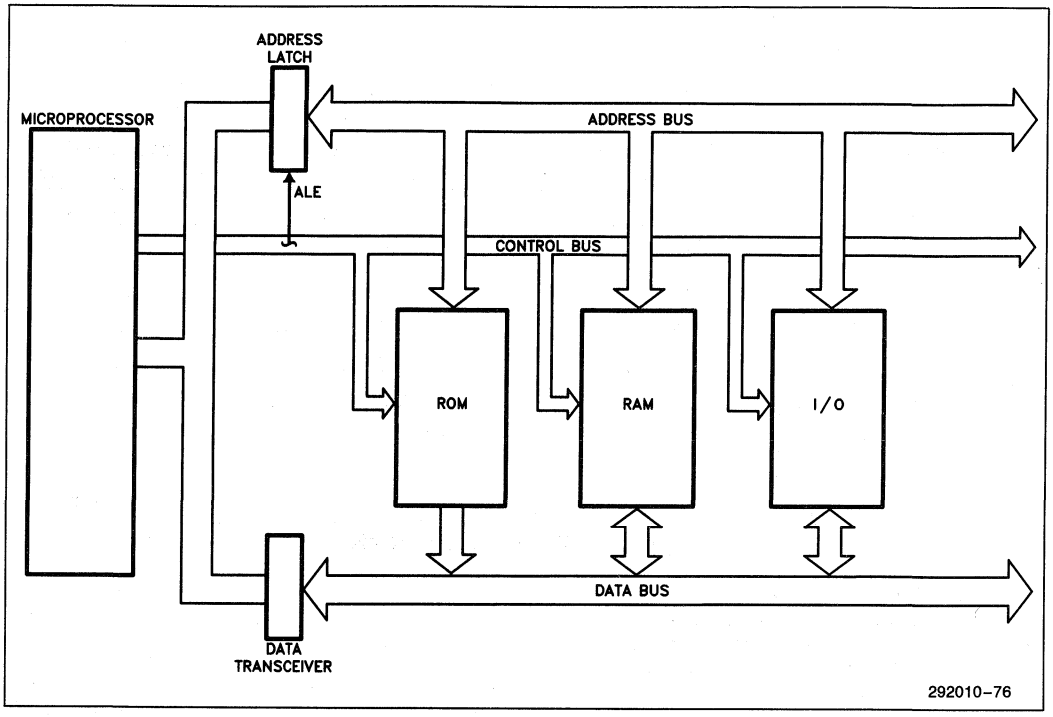


Figure 5. Typical Microprocessor System

Bus Loading and Voltage Level Compatabilities

The data and address lines do not exceed the drive capability of either 80186 or the 82530. There are several control lines that drive more than one TTL equivalent input. The drive capability of these lines are detailed below.

$\overline{WR}$: The $\overline{WR}$ signal drives U3 and U4.

* Iol (2.0 mA) > Iil (−0.4 mA + −0.5 mA)
Ioh (−400 μA) > Iih (20 μA + 20 μA)

$\overline{PCS5}$: The $\overline{PCS5}$ signal drives U2 and U4.

* Iol (2.0 mA) > Iil (−0.5 mA + −0.5 mA)
Ioh (−400 μA) > Iih (20 μA + 20 μA)

$\overline{INTA}$: The $\overline{INTA}$ signal drives 2(U1) and U5.

* Iol (2.0 mA) > Iil (-0.4 mA + −0.8 mA + −0.4 mA)
Ioh (−400 μA) > Iih (20 μA + 40 μA + 20 μA)

All the 82530 I/O pins are TTL voltage level compatible.

TIMING ANALYSIS

Certain symbolic conventions are adhered to throughout the analysis below and are introduced for clarity.

1. All timing variables with a lower case first letter are 82530 timing requirements or responses (i.e., tRR).
2. All timing variables with Upper case first letters are 80186 timing responses or requirements unless preceded by another device's alpha-numeric code (i.e., Tclcl or '373 Tpd).
3. In the write cycle analysis, the timing variable Tpd$\overline{WR}$186-$\overline{WR}$530 represents the propagation delay between the leading or trailing edge of the $\overline{WR}$ signal leaving the 80186 and the $\overline{WR}$ edge arrival at the 82530 $\overline{WR}$ input.

Read Cycle

1. **tAR:** Address valid to $\overline{RD}$ active set up time for the 82530. Since the propagation delay is the worst case path in the assumed typical system, the margin is calculated only for a propagation delay constrained and not an ALE limited path. The spec value is 0 ns minimum.

* 1 Tclcl − Tclav(max) − '245 Tpd(max) + Tclrl(min) + 2(U2) Tpd(min) − tAR(min)

= 125 − 55 − 20.8 + 10 + 2(2) − 0 = 63.2 ns margin

2. **tRA:** Address to $\overline{RD}$ inactive hold time. The ALE delay is the worst case path and the 82530 requires 0 ns minimum.

* 1 Tclcl − Tclrh (max) + Tchlh(min) + '373 LE Tpd(min) − 2(U2) Tpd(max)

= 55 − 55 + 5 + 8 − 2(5.5) = 2 ns margin

3. **tCLR:** $\overline{CS}$ active low to $\overline{RD}$ active low set up time. The 82530 spec value is 0 ns minimum.

* 1 Tclcl − Tclcsv(max) − Tclrl(min) − U2 skew($\overline{RD}$ − $\overline{CS}$) + U2 Tpd(min)

= 125 − 66 − 10 − 1 + 2 = 50 ns margin

4. **tRCS:** $\overline{RD}$ inactive to $\overline{CS}$ inactive hold time. The 82530 spec calls for 0 ns minimum.

* Tcscsx(min) − U2 skew($\overline{RD}$ − $\overline{CS}$) − U2 Tpd(max)

= 35 − 1 − 5.5 = 28.5 ns margin

5. **tCHR:** $\overline{CS}$ inactive to $\overline{RD}$ active set up time. The 82530 requires 5 ns minimum.

* 1 Tclcl + 1 Tchcl − Tchcsx(max) + Tclrl(min) − U2 skew ($\overline{RD}$ − $\overline{CS}$) + U2 Tpd(min) − tCHR

= 125 + 55 − 35 − 10 − 1 + 2 − 5 = 131 ns margin

6. **tRR:** $\overline{RD}$ pulse active low time. One 80186 wait state is included to meet the 150 ns minimum timing requirements of the 82530.

* Trlrh(min) + 1($\overline{Tclcl}$wait state) − 2(U2 skew) − tRR

= (250−50) + 1(125) − 2(1) − 150 = 173 ns margin

7. tRDV: $\overline{RD}$ active low to data valid maximum delay for 80186 read data set up time (Tdvcl = 20 ns). The margin is calculated on the Propagation delay path (worst case).

* 2 Tclcl + 1($\overline{Tclcl}$wait state) − Tclrl(max) − Tdvcl(min) − '245 Tpd(max) − 82530 tRDV(max) − 2(U2) Tpd(max)

= 2(125) + 1(125) − 70 − 20 − 14.2 − 105 − 2(5.5) = 154 ns margin

8. **tDF:** $\overline{RD}$ inactive to data output float delay. The margin is calculated to $\overline{DEN}$ active low of next cycle.

* 2 Tclcl + Tclch(min) − Tclrh(max) + Tchctv(min) − 2(U2) Tpd(max) − 82530 tDF(max)

= 250 + 55 −55 + 10 − 11 − 70 = 179 ns margin

9. **tAD:** Address required valid to read data valid maximum delay. The 82530 spec value is 325 ns maximum.

* 3 Tclcl + 1($\overline{Tclcl}$wait state) − Tclav(max) − '373 Tpd(max) − '245 Tpd − Tdvcl(min) − tAD

= 375 + 125 − 55 − 20.8 −14.2 − 20 −325 = 65 ns margin

Write Cycle

1. **tAW:** Address required valid to $\overline{WR}$ active low set up time. The 82530 spec is 0 ns minimum.

* Tclcl − Tclav(max) − Tcvctv(min) − '373 Tpd(max) + Tpd$\overline{WR}$186 − $\overline{WR}$530(LOW) [Tclcl − Tcvctv(min) + U3 Tpd(min) + U4 Tpd(min)] − tAW

= 125 − 55 − 5 − 20.8 + [125 − 5 + 1 + 4.4] − 0 = 170.6 ns margin

2. **tWA:** $\overline{WR}$ inactive to address invalid hold time. The 82530 spec is 0 ns.

* Tclch(min) − Tcvctx(max) + Tchlh(min) + '373 LE Tpd(min) − Tpd$\overline{WR}$186=$\overline{WR}$530(HIGH) [U2 Tpd(max) + U3 Tpd(max) + U4 Tpd(max)]

= 55 − 55 + 5 + 8 − [5.5 + 3 + 7.1] = −2.6 ns margin

3. **tCLW:** Chip select active low to $\overline{WR}$ active low hold time. The 82530 spec is 0 ns.

* 1 Tclcl − Tclcsv(max) + Tcvctv(min) − U2 Tpd(max) + Tpd$\overline{WR}$186=$\overline{WR}$530(LOW) [Tclcl − Tcvctv(min) + U3 Tpd(min) + U4 Tpd(min)]

= 125 − 66 + 5 − 5.5 + [125 − 5 + 1 + 4.4] = 183.9 ns margin

4. **tWCS:** $\overline{WR}$ invalid to Chip Select invalid hold time. 82530 spec is 0 ns.

* Tcxcsx(min) − U2 Tpd(max) − Tpd$\overline{WR}$186=$\overline{WR}$530(HIGH) [U2 Tpd(max) + U3 Tpd(max) + U4 Tpd(max)]

= 35 + 1.5 − [5.5 + 3 + 7.1] = 20.9 ns margin

5. **tCHW:** Chip Select inactive high to $\overline{WR}$ active low set up time. The 82530 spec is 5 ns.

* 1 Tclcl + Tchcl(min) + Tcvctv(min) − Tchcsx(max) − U2 Tpd(max) + Tpd$\overline{WR}$186=$\overline{WR}$530(LOW) [Tclcl − Tcvctv(min) + U3 Tpd(min) + U4 Tpd(min)] − tCHW

= 125 + 55 + 5 − 35 − 5.5 + [125 −5 + 1 + 4.4] − 5 = 264 ns margin

6. **tWW:** $\overline{WR}$ active low pulse. 82530 requires a minimum of 60 ns from the falling to the rising edge of $\overline{WR}$. This includes one wait state.

* Twlwh [2Tclcl − 40] + 1 (Tclclwait state) − TpdWR/186−WR530(LOW) [Tclcl − Tcvctv(min) + U3 Tpd(max) + U4 Tpd(max)] + TpdWR/186=WR/530(HIGH) [U2 Tpd(min) U3 Tpd(min) + U4 Tpd(min)] − tWW

= 210 + 1(125) − [125 − 5 + 4.5 + 9.2] − [1.5 + 1 + 3.2] − 60 = 135.6 ns margin

7. **tDW:** Data valid to $\overline{WR}$ active low setup time. The 82530 spec requires 0 ns.

* Tcvctv(min) − Tcldv(max) − '245 Tpd(max) + TpdWR186−WR530(LOW) [Tclcl − Tcvctv(min) + U3 Tpd(min) + U4 Tpd(min)]

= 5 − 44 − 14.2 + 125 − 5 + 1.0 + 4.4 = 72.2 ns margin

8. **tWD:** Data valid to $\overline{WR}$ inactive high hold time. The 82530 requires a hold time of 0 ns.

* Tclch − skew {Tcvctx(max) + Tcvctx(min)} + '245 OE Tpd(min) − TpdWR186−WR530(HIGH) [U2 Tpd(max) + U3 Tpd(max) + U4 Tpd(max)]

= 55 − 5 + 11.25 − [5.5 + 3.0 + 7.1] = −50.6 ns margin

INTA Cycle:

1. **tIC:** This 82530 spec implies that the $\overline{INTA}$ signal is latched internally on the rising edge of CLK (82530). Therefore the maximum delay between the 80186 asserting $\overline{INTA}$ active low or inactive high and the 82530 internally recognizing the new state of $\overline{INTA}$ is the propagation delay through U1 plus the 82530 CLK period.

* U1 Tpd(max) + 82530 CLK period

= 45 + 250 = 295 ns

2. **tCI:** rising edge of CLK to $\overline{INTA}$ hold time. This spec requires that the state of $\overline{INTA}$ remains constant for 100 ns after the rising edge of CLK. If this spec is violated any change in the state of $\overline{INTA}$ may not be internally latched in the 82530. tCI becomes critical at the end of an $\overline{INTA}$ cycle when $\overline{INTA}$ goes inactive. When calculating margins with tCI, an extra 82530 CLK period must be added to the $\overline{INTA}$ inactive delay.

3. **tIW:** $\overline{INTA}$ inactive high to $\overline{WR}$ active low minimum setup time. The spec pertains only to 82530 $\overline{WR}$ cycle and has a value of 55 ns. The margin is calculated assuming an 82530 $\overline{WR}$ cycle occurs immediately after an $\overline{INTA}$ cycle. Since the CPU cycles following an 82530 $\overline{INTA}$ cycle are devoted to locating and executing the proper interrupt service routine, this condition should never exist. 82530 drivers should insure that at least one CPU cycle separates $\overline{INTA}$ and $\overline{WR}$ or $\overline{RD}$ cycles.

4. **tWI:** $\overline{WR}$ inactive high to $\overline{INTA}$ active low minimum hold time. The spec is 0 ns and the margin assumes CLK coincident with $\overline{INTA}$.

* Tclcl − Tcvctx(max) − TpdWR186 − WR530(HIGH) [U3 Tpd(max) + U4 Tpd(max)] + Tcvctv(min) + U1 Tpd(min)

= 125 − 55 − [5.5 + 3 + 7.1] + 5 + 10 = 69.4 ns margin

5. **tIR:** $\overline{INTA}$ inactive high to $\overline{RD}$ active low minimum setup time. This spec pertains only to 82530 $\overline{RD}$ cycles and has a value of 55 ns. The margin is calculated in the same manner as tIW.

6. **tRI:** $\overline{RD}$ inactive high to $\overline{INTA}$ active low minimum hold time. The spec is 0 ns and the margin assumes CLK coincident with $\overline{INTA}$.

* Tclcl − Tclrh(max) − 2 U2 Tpd(max) + Tcvctv(min) + U1 Tpd(min)

= 125 − 55 − 2(5.5) + 5 + 10 = 74 ns margin

7. **tIID:** $\overline{INTA}$ active low to $\overline{RD}$ active low minimum setup time. This parameter is system dependent. For any SCC in the daisy chain, tIID must be greater than the sum of tCEQ for the highest priority device in the daisy chain, tEI for this particular SCC, and tEIEO for each device separating them in the daisy chain. The typical system with only 1 SCC requires tIID to be greater than tCEQ. Since tEI occurs coincidently with tCEQ and it is smaller it can be neglected. Additionally, tEIEO does not have any relevance to a system with only one SCC. Therefore tIID > tCEQ = 250 ns.

* 4 Tclcl + 2 Tidle states − Tcvctv(max) − tIC [U1 Tpd(max) + 82530 CLK period] + Tcvctv(min) + U5 Tpd(min) + U2 Tpd(min) − tIID

= 500 + 250 − 70 − [45 + 250] + 5 + 6 + 2 − 250 = 148 ns margin

8. **tIDV:** $\overline{RD}$ active low to interrupt vector valid delay. The 80186 expects the interrupt vector to be valid on the data bus a minimum of 20 ns before T4 of the second acknowledge cycle (Tdvcl). tIDV spec is 100 ns maximum.

* 3 Tclcl − Tcvctv(max) − U5 Tpd(max) − U2 Tpd(max) − tIDV(max) − '245 Tpd(max) − Tdvcl(min)

= 375 − 70 − 25 − 5.5 − 100 − 14.2 − 20 = 140.3 ns margin

9. **tII:** $\overline{RD}$ pulse low time. The 82530 requires a minimum of 125 ns.

* 3 TclcI − Tcvctv(max) − U5 Tpd(max) − U2 Tpd(max) + Tcvctx(min) + U5 Tpd(min) + U2 Tpd(min) − tII(min)

= 375 − 70 − 25 − 5.5 + 5 + 6 + 1.5 − 125 = 162 ns margin

DMA Cycle

Fortunately, the 80186 DMA controller emulates CPU read and write cycle operation during DMA transfers. The DMA transfer timings are satisfied using the above analysis. Because of the 80186 DMA request input requirements, two wait states are necessary to prevent inadvertent DMA cycles. There are also CPUDMA intracycle timing considerations that need to be addressed.

1. **tDRD:** $\overline{RD}$ inactive high to $\overline{DTRREQ}$ (REQUEST) inactive high delay. Unlike the $\overline{READYREQ}$ signal, $\overline{DTRREQ}$ does not immediately go inactive after the requested DMA transfer begins. Instead, the $\overline{DTRREQ}$ remains active for a maximum of 5 tCY + 300 ns. This delayed request pulse could trigger a second DMA transfer. To avoid this undesirable condition, a D Flip Flop is implemented to reset the $\overline{DTRREQ}$ signal inactive low following the initiation of the requested DMA transfer. To determine if back to back DMA transfers are required in a source synchronized configuration, the 80186 DMA controller samples the service request line 25 ns before T1 of the deposit cycle, the second cycle of the transfer.

* 4 TclcI − Tclcsv(max) − U4Tpd(max) − Tdrqcl(min)

= 500 − 66 − 10.5 − 25 = 398.5 ns margin

2. tRRI: 82530 $\overline{RD}$ active low to $\overline{REQ}$ inactive high delay. Assuming source synchronized DMA transfer, the 80186 requires only one wait state to meet the tRRI spec of 200 ns. Two are included for consistency with tWRI.

* 2 TclcI + 2($\overline{TclcI}$wait state) − TclrI(max) − 2(U2) Tpd(max) − Tdrqcl − tRRI

=2(125) + 2(125) − 70 − 2(5.5) − 200 = 219 ns margin

3. tWRI: 82530 $\overline{WR}$ active low to $\overline{REQ}$ inactive high delay. Assuming destination synchronized DMA transfers, the 80186 needs two wait states to meet the tWRI spec. This is because the 80186 DMA controller samples requests two clocks before the end of the deposit cycle. This leaves only 1 TclcI + n(wait states) minus $\overline{WR}$ active delay for the 82530 to inactivate its $\overline{REQ}$ signal.

* TclcI + 2($\overline{TclcI}$wait state) − Tcvctv(min) − Tpd$\overline{WR}$186−$\overline{WR}$530(LOW) [TclcI − Tcvctv(min) + U3 Tpd(max) + U4 Tpd(max)] − Tdrqcl − tWRI

=375 − 5 − [125 − 5 + 4.5 + 9.2] − 25 − 200 = 11.3 ns margin

NOTE:
If one wait state DMA interface is required, external logic, like that used on the $\overline{DTRREQ}$ signal, can be used to force the 82530 $\overline{REQ}$ signal inactive.

4. **tREC:** CLK recovery time. Due to the internal data path, a recovery period is required between SCC bus transactions to resolve metastable conditions internal to the SCC. The DMA request lines are marked from requesting service until after the tREC has elapsed. In addition, the CPU should not be allowed to violate this recovery period when interleaving DMA transfers and CPU bus cycles. Software drivers or external logic should orchestrate the CPU and DMA controller operation to prevent tREC violation.

Reset Operation

During hardware reset, the system RESET signal is asserted high for a minimum of four 80186 clock cycles (1000 ns). The 82530 requires $\overline{WR}$ and $\overline{RD}$ to be simultaneously asserted low for a minimum of 250 ns.

* 4 TclcI − U3 Tpd(max) − 2(U2) Tpd(max) + U4 Tpd(min) − tREC

= 1000 − 17.5 − 2(5.5) + 3.5 − 250 ns = 725 ns margin

82588 Reference Manual

4

November 1986

82588 Reference Manual

Order Number: 231368-003

1.0 INTRODUCTION

The 82588 is a highly integrated Local Area Network (LAN) Controller that lends itself easily to cost sensitive LAN applications such as Personal Computers and Intelligent Terminals. Because of its high integration, the 82588 reduces component count which in turn reduces board space and development time. Additionally, due to the flexibility of the 82588, the system design engineer can program a variety of device parameters to a configuration that allows it to be used in the emerging IEEE 802.3 standards for PCs. The combination of low cost and high flexibility makes the 82588 an attractive solution for the LAN system designer.

The 82588 provides most of the functions of the ISO Physical and Data Link layers of LAN architecture. This includes a CSMA/CD controller, two different collision detect mechanisms, and a data encoder/decoder that supports data transfer rates of up to 2 Mbps.

As mentioned, the 82588 is programmable which allows it to operate in a variety of LAN environments including the emerging IEEE 802.3 standard of 1 Mbps baseband (called STARLAN) and the IBM PC Network 2 Mbps broadband. Some of the programmable parameters are:

— Framing (End of Carrier or SDLC)
— Address Field Length
— Station Priority
— Interframe Spacing
— Slot Time
— CRC-32 or CRC-16
— NRZI or Manchester encoding/decoding

In addition, the 82588 allows for a variety of frame formats, network topologies, data encoding and decoding schemes and data transfer rates.

A major innovation of the 82588 is its on-chip logic based collision detection. So that a high probability of collision detection can be maintained, two mechanisms are provided. First, the Code Violation method defines a collision when a transition edge occurs outside of the region specified for NRZI or Manchester encoding. Second, the Bit Comparison method compares the signature of a transmitted frame to the receive frame signature while "listening to itself."

The 82588 goes beyond the basic provisions required for LAN physical interfacing. It also has an extremely friendly system interface that makes it easy to design with. The 82588 has a high level command interface (the CPU sends commands such as CONFIGURE or TRANSMIT) so the designer does not have to bother with low level software development. This saves on CPU overhead as well. The 82588 makes efficient use of the system memory available by employing Multiple Buffer Reception in which receive frames are saved in buffers that are chained together in system memory. This is an important feature for those applications with limited memory such as personal computers. The 82588 has two independent 16 byte FIFOs (one for reception and one for transmission) that allows the device to tolerate bus latency. Finally, the 82588 provides an 8-bit data path that supports up to 4 Mbytes/second using external DMA. The DMA interface has been optimized for use with the 80186/80188 microprocessors.

The 82588 provides a rich set of diagnostic and network management functions that allow the designer to minimize debug time and maintain top network efficiency. Specifically, the 82588 provides internal and external loopback capability and network channel activity indicators, it can capture all frames regardless of address (Promiscuous Mode) and it supports Time Domain Reflectometry for locating shorts and opens in the transmission cable. There is also a Register Dump command which allows the user to examine the 82588 internal status registers.

The following chapters describe the features and capabilities of the 82588 in greater detail.

2.0 THE 82588 INTERNAL ARCHITECTURE

There are two major functional blocks of the 82588: a parallel system interface to the host CPU and a serial interface to the local area network. Linking these functional blocks are two on-chip, 16-byte FIFOs, one each for transmitting and receiving (see Figure 1).

2.1 Parallel Section

The parallel system interface consists of a Bus Interface Unit (BIU) and several registers. The BIU has an 8-bit data bus, and is responsible for all interfacing to the system bus. It handles all transfer of data to and from memory — at speeds of up to 4 Mbytes/second — as well as issuing CPU commands and interrupts. There are two DMA channels allowing for simultaneous transmission and reception. The register section consists of three register sets. The first stores configuration information, the second is for the posting of commands and the third contains status information.

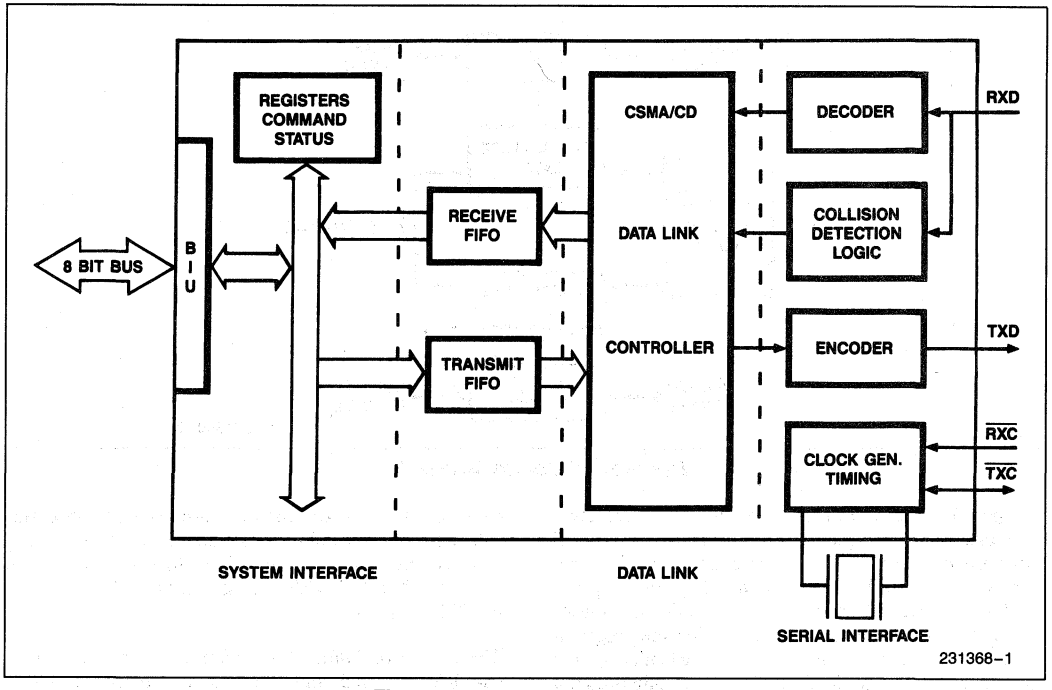


Figure 1. 82588 Block Diagram

2.2 Serial Section

The serial machine of the 82588 employs the IEEE 802.3 CSMA/CD protocol. The serial section is responsible for the following tasks:

— converting data from parallel to serial form and vice versa

— formatting the frame per the programmed configuration

— computing the CRC "signature" and monitoring for code violations in order to detect collisions

— performing carrier sense and deference if the network is busy

— calculating random delay time before retransmission in the event of deference or collision

The efficiency of CSMA/CD was demonstrated in a 1979 study of an Ethernet LAN of about 120 host computers and network server devices. The study showed that a network using collision detection has a throughput rate of 98 percent. This can be compared to the efficiency rate of 37 percent in networks with collision avoidance and 18 percent in networks with no collision regulating provisions. Until recently, the relative sophistication of collision detection schemes such as CSMA/CD prevented their inclusion in cost effective LAN components.

The data encoder/decoder of the 82588 can transmit and receive data in any one of three formats: Manchester, differential Manchester and NRZI encoding. Data rates with the internal encoder/decoder, can be up to 2 Mbs. The serial machine is driven by a clock generator using an up to 16 MHz crystal. This clock generator is for the serial side only; the parallel side receives its timing from the system clock.

COLLISION DETECTION

A major breakthrough introduced in the 82588 is the on-chip logic based collision detection capability. There are two programmable forms of collision detection. The "code violation" detection method checks the incoming bits to see if they violate Manchester or NRZI standards. If a violation does occur, the 82588 assumes a collision has occurred. The second method is called "bit comparison." For this method the 82588 listens to itself while calculating a signature for both the transmit and receive data. If the signatures do not match, the 82588 will back-off and retry immediately, without having to wait for transmission of the frame to be completed. In this way, total data throughput is increased significantly.

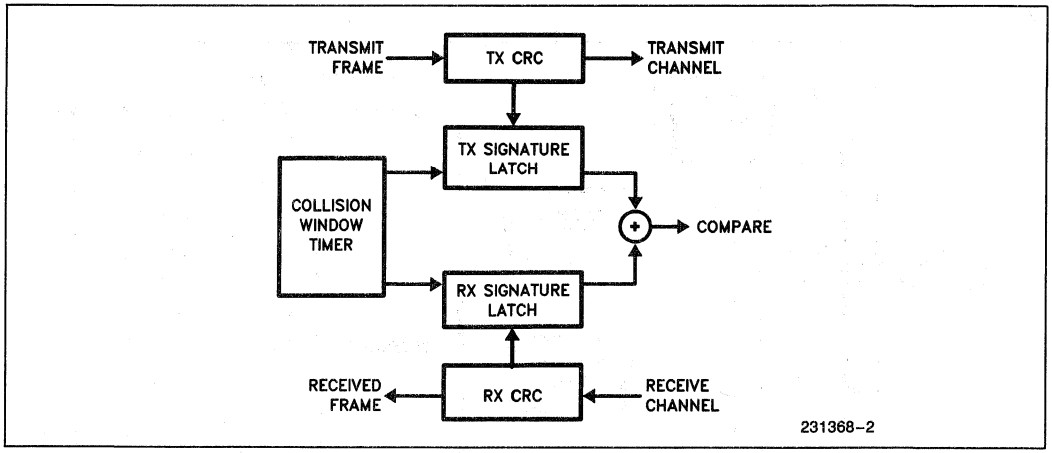


Figure 2. Bit Comparison Detect

The parallel and serial section of the 82588 are asynchronous. The interface between them is made up of two unique FIFOs, one for transmission and one for reception. The FIFOs are 16 bytes deep and improve the data transfer rate in two ways. First, DMA requests to the CPU are minimized by fine tuning the programmable FIFO threshold to match the particular system bus latencies. Second, the CPU side of the chip does not have to wait for the network side which transfers data at a slower rate.

3.0 WORKING WITH THE 82588

This section describes how the 82588 interacts with system CPU, and how the 82588 transmits and receives frames. The emphasis here is not on any particular commands, but rather on how the 82588 works. The details on Framing, Network Management, Initializing the 82588, Configuring the 82588, Controlling the 82588, System Interface, and Link Interface are contained in the following chapters.

3.1 82588/Host CPU Interaction

The CPU communicates with the 82588 through the System's memory and 82588's on-chip registers. The CPU creates a data structure in the memory, programs the external DMA controller with the start address and byte count of the block, and issues the command to the 82588.

The 82588 is optimized for operating with the iAPX 186/188, but due to the very conventional nature of hardware signals between the 82588 and the CPU, the 82588 can operate easily with other processors. The data bus is 8 bits wide and there is no address bus.

Chip select and Interrupt lines are used to communicate between the 82588 and the host as shown in Figure 3. Interrupt is used by the 82588 to draw the CPU's attention. The Chip Select is used by the CPU to draw the 82588's attention.

There are two kinds of transfers over the bus: Command/Status and data transfers. Command/Status transfers are always performed by the CPU. Data Transfers are requested by the 82588, and are typically performed by a DMA controller.

The CPU writes to 82588 using $\overline{CS}$ and $\overline{WR}$ signals. The CPU reads the 82588 status register using $\overline{CS}$ and $\overline{RD}$ signals.

To initiate a command like Transmit or Configure, a Write operation to 82588 is issued by the CPU. A Read operation from CPU gives the status of the 82588. Although there are four status registers, they are read using the same port in a round robin fashion. Section Eight discusses details on these commands, and the status registers.

Any parameters or data associated with the command are transferred between the memory and 82588 using DMA. The 82588 has two data channels, each having Request and Acknowledge line. Typically one channel is issued to receive data and the other to transmit data and to do all the other initialization and maintenance operations like Configure, Address Set-Up, Dump, Diagnose, etc. The two channels are identical and can be used interchangeably.

When 82588 requires access to the memory for parameter or data transfer, it activates the DMA request line and uses the DMA controller to achieve the data transfer. Upon the completion of an operation, the 82588 interrupts the CPU. The CPU then reads results of the operation and the status of the 82588.

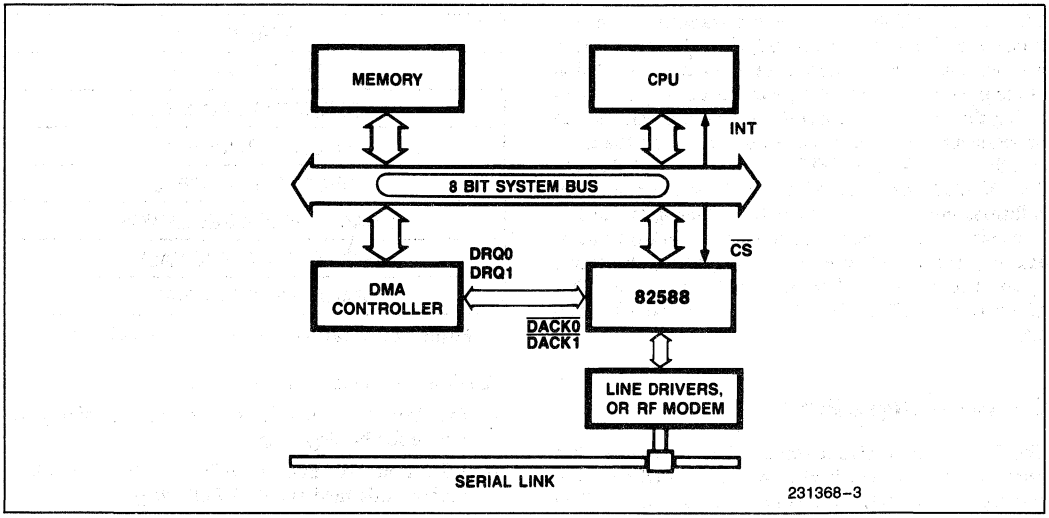


Figure 3. 82588/Host CPU Interaction

3.2 Transmitting a Frame

To transmit a frame, the CPU prepares a Transmit Data Block in memory as shown in Figure 4. Its first two bytes specify the length of the rest of the block. The next few bytes (up to 6 bytes long) contain the destination address of the node it is being sent to. The rest of the block is the data field. The CPU programs the DMA controller with the start address of the block, length of the block and other control information and then issues the Transmit command to the 82588. (Section 4 discusses details on the framing).

Upon receiving the command, the 82588 fetches the first two bytes of the block using DMA to determine the length of the block. If the link is free, the 82588 begins transmitting the preamble and concurrently fetches the bytes from the Transmit Data Block and loads them into a 16 byte FIFO to keep them ready for transmitting. The FIFO is a buffer between the serial and parallel part of the 82588. If the link is busy, the 82588 fills up the transmit FIFO and defers. The on-chip FIFOs help the 82588 to tolerate system bus latency as well as provide efficient usage of system bus bandwidth.

The destination address is sent out after the preamble. This is followed by the source or the station individual address, which is stored earlier on the 82588 using the IA-SETUP command. After that, the entire information field is transmitted followed by a CRC field calcu-

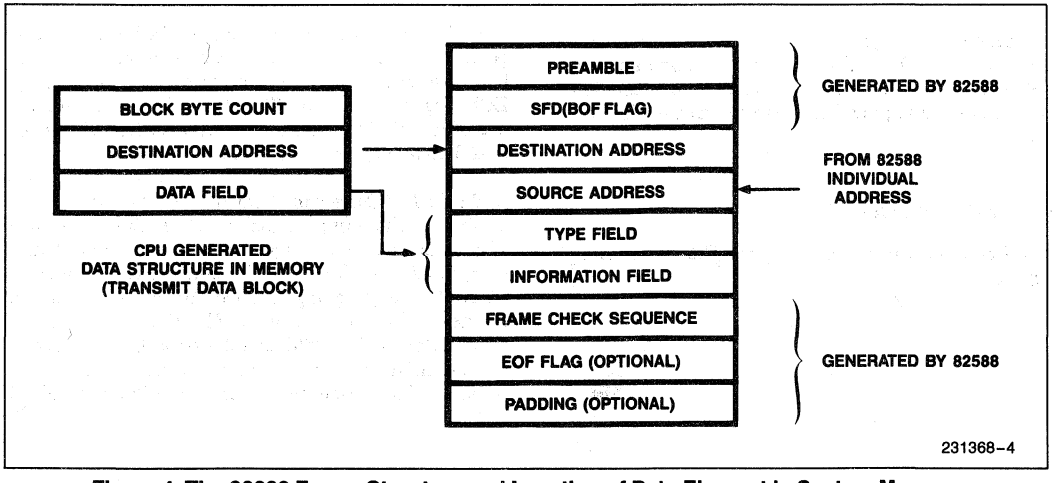


Figure 4. The 82288 Frame Structure and Location of Data Element in System Memory

lated by the 82588. If during the transmission of the frame, a collision is encountered, then the transmission is aborted and a jam pattern is sent out after completion of the preamble. The 82588 generates an Interrupt indicating the experience of a collision and the frame has to be retransmitted. Retransmission is done by the CPU exactly as the TRANSMIT command except the RETRANSMIT command keeps track of the number of collisions encountered. When the 82588 gets the Retransmit command and the exponential back-off time is expired, the 82588 attempts transmitting the frame again. The transmitted frame can be coded to either Manchester, Differential Manchester or NRZI methods.

3.3 Receiving a Frame

The 82588 can receive a frame when its receiver has been enabled. The received frame is decoded by either on-chip Manchester, Differential Manchester or NRZI decoders in High Integration Mode or by external decoders in the High Speed Mode. The 82588 checks for an address match for an Individual address, a Multicast address or a Broadcast address. In the Promiscuous mode the 82588 receives all frames. Only when the address match is successful does the 82588 transfer the frame to the memory using the DMA controller. Before enabling the receiver, the CPU makes a memory buffer area available to the Receive Unit, and programs the starting address of the buffer in the DMA controller. The received frame is transferred to the memory buffer.

4.0 FRAMING AND LINK MANAGEMENT

The data structures handled by the 82588 are called frames. A frame is a sequence of bits that travels on the link.

Framing has three primary functions: to determine the beginning and the end of the frame (frame boundary delineation); to determine the frame's source and destination (addressing); and to perform transmission error detection.

4.1 Frame Format

The Frame format supported by the 82588 is shown in Figure 5.

PREAMBLE
SFD (BOF FLAG)
DESTINATION ADDRESS
SOURCE ADDRESS
INFORMATION FIELD
FRAME CHECK SEQUENCE
EOF FLAG (OPTIONAL)
PADDING (OPTIONAL)

Figure 5. 82588 Generalized Frame Format

The different fields of the frame are:

— The Preamble which is used as a synchronizing sequence for bit decoding.

— SFD, Start of Frame Delimiter (for bitstuffing method, this field is called BOF flag).

— The Destination Address (frame target address).

— The Source Address (sender's address).

— The Information field containing user supplied data.*

— The Frame Check Sequence (FCS): Cyclic Redundancy Check (CRC) used in detecting bit errors.

— The End of Frame (EOF) flag - optional. (For bitstuffing mode.)

— Padding: optional field which extends the length of the frame to ensure minimum frame length. Optional (for bitstuffing mode).

* The 802.3 Length Field is included in the Information field.

4.2 Frame Boundary Delineation

The 82588 handles Frame Boundary Delineation transparently to the user. The fields involved in Frame boundary delineation are: Preamble, SFD (BOF Flag), EOF flag and Padding. The 82588 is configurable to one of two Frame delineation methods: End of Carrier (802.3 compatible) or Bitstuffing.

In the End of Carrier method, the 82588 transmits (depending on the configuration) 1, 3, 7, 15 preamble bytes of alternating ones and zeros followed by a SFD (BOF Flag) of pattern 10101011. The end of frame is indicated by the carrier going inactive immediately after the Frame Check Sequence field. The Carrier is a signal which informs the 82588 of activity on the link. It can be internally or externally generated. This frame boundary delineation method is compatible with IEEE 802.3.

The Bitstuffing method implements the HDLC zero bit insertion/deletion mechanism. The 82588 transmits (depending on the configuration) a Preamble of 1, 3, 7 or 15 bytes of alternating ones and zeroes followed by an HDLC flag (01111110). End of frame is indicated by a HDLC flag. The 82588 can perform HDLC zero bit insertion (insert 0 after five consecutive 1's) on the fields between the BOF and EOF flags. The chip can be configured to pad the frame with additional flags so that the frame length becomes equal to or longer than the Slot Time.

4.3 Addressing

Regardless of the Frame Boundary Delineation method, the two fields following the SFD (BOF Flag) are Destination Address and Source Address.

Addressing allows frames to be directed to one or more specific nodes. The 82588 provides flexible addressing techniques allowing a frame to be received by a single node, a group of nodes (multicast addressing), or all nodes (broadcast addressing).

After initialization, the 82588 is configured with an Individual Address using the IA-SETUP command. The address length is determined by the Address Length parameter (0 to 6 bytes). The default configuration is an all ones address (Broadcast Address). 6 bytes long.

During transmission the 82588 usually inserts its Individual Address from the internal Individual Address register into the Source Address field. The source address insertion can be overridden when configuring the chip. During reception, the 82588 compares the incoming Destination Address with its Individual Address, the programmed multicast set of address and the broadcast address. If an address match is made (all bits must be equal) the frame is accepted. A frame whose address does not match is simply ignored by the 82588 and has no effect on the 82588 nor on any other component of the station.

INDIVIDUAL ADDRESS

The 82588 is normally configured with a specific Individual Address. An Individual Address must have a zero in the least significant bit.

BROADCAST ADDRESSING

An address with all 1's is the Broadcast Address. Every station on the link "hears" a Broadcast. A frame may be targetted to all nodes by using the broadcast address. The 82588 can be configured to disable reception of frames with Broadcast Destination Address, e.g., for stations with limited storage resources.

MULTICAST ADDRESSING

Frames can be directed to a specific group of nodes. This group can have a particular group address called Multicast Address. A node may belong to several different groups. A one in the least significant bit of the destination address distinguishes Multicast Address from Individual Address. MC-SETUP command is used to program the set of Multicast addresses for a station by setting up a 64 bit "Hash Table." The 82588 maps and stores every Multicast Address into a single bit of the Hash table. During reception of a frame whose Destination Address is a Multicast Address, the 82588 checks whether this address is mapped in the Hash table. If an address match is determined, the 82588 begins the reception process.

It is possible for more than one Multicast Address to be mapped into a given Hash bit. Thus, the host may have to perform additional checking. If 64 or fewer Multicast addresses are used in a system, it is possible to select address values that map into unique bits in the Hash table. The Hashing function is the CRC polynomial used for bit error detection. The 6 most significant bits (2–7) are selected from the first byte of the CRC shift register to index the 64-bit Hash table.

INFORMATION FIELD

The information field follows the source address field. It contains the actual data being transferred in the frame. Its maximum length is $(2^{16}) - 2 \times$ (address length) $- 2$ which includes destination and source addresses.

4.4 Error Detection

The Frame Check Sequence (FCS) Field protects against bit errors in the frame. It is the result of a Cyclic Redundancy Check computed on the Destination Address, Source Address, and Information Fields. The chip can be configured to one of two CRC algorithms: the CCITT V-41 (HDLC) 16-bit polynomial or the Autodin II (IEEE 802.3) 32-bit polynomial.

The CRC mechanism is transparent to the user. During transmission, the 82588 calculates and inserts the Frame Check Sequence. During reception the chip verifies the correctness of the incoming Frame but does not pass the FCS to memory. The CRC is applied to an integral number of 8-bit words excluding the potential residue. CRC insertion can be disabled for diagnostic purposes. The user should then provide the FCS field after the data field.

4.5 Frame Transmission

The 82588 transfers data from host memory to the network when a CPU issues a TRANSMIT command. Before instructing the 82588 to start a transmission, the host CPU prepares the frame in memory. Parts of the frame such as Preamble Source Address, CRC and Flags are inserted by the 82588 Data Link Controller. The 82588 resolves access and contention on the link using the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) link Management protocol. When the transmission is completed, the 82588 updates a set of status registers and raises its Interrupt signal to inform the CPU.

4.6 Link Management

The 82588 handles CSMA/CD link management algorithms according to the IEEE 802.3 standard. 82588 link management algorithms are adaptable to a variety of network topologies via programmable configuration parameters. Station priorities are also programmable.

The 82588 constantly monitors link activity. Whenever it senses activity on the link the 82588 defers to the passing frame by delaying any pending transmission. When there is no more activity on the link, the 82588 continues to defer for Inter Frame Spacing (IFS) time (minimum time between two consecutive frame transmissions). This parameter is configurable from 12 to 255 bit times. If at the end of that time, the 82588 has a frame waiting to be transmitted, transmission is initiated independently of the link activity.

Once the 82588 has finished deferring and has started transmission, it is possible to experience link contention. This situation is called a collision and is generally detected and signalled by the transceiver. The 82588 can be programmed to detect collisions internally or externally via the CDT pin. When the 82588 experiences a collision, it enforces the collision by transmitting a jam pattern of 32 to 48 bits.

If a collision is detected during the Preamble, transmission of the preamble is completed before jamming starts. The collision enforcement mechanism ensures detection of the collision by all the stations.

The jam pattern is all ones in all configurations except for NRZI encoding, where it is a sequence of all zeros. (Transition every bit time).

The dynamics of collision handling are largely determined by the Slot Time. Slot Time is the maximum end to end round trip delay time of the network plus jam time. Slot Time is important because it is the worst case time to detect a collision. The Slot Time is programmable from 1 to 2048 bit times.

The 82588 notifies the user of a collision and gives an opportunity to retransmit at the end of the Backoff time out.

The 82588 implements the IEEE 802.3 scheduling of the retransmission. The controlled randomization process is called "truncated binary exponential backoff." A retransmission is delayed an integral multiple of slot times. The number of slot times to delay before the nth retransmission attempt is chosen as a uniformly distributed random integer in the range $0 \leq r < 2^k$, where $K = \min [n, 10]$. The number of retransmission attempts is programmable in the range 0 to 15. The beginning of the Backoff time is configurable to one of two methods: If configured to the IEEE 802.3 compatible method, Backoff starts immediately after the end of the Jam pattern; if configured to the alternate Backoff method (designed for lower bit rates and/or shorter topologies), Backoff starts after the deferring period following collision.

After the backoff time has expired and the CPU has issued a RETRANSMIT command, the 82588 attempts to retransmit the frame (after deferring to any traffic on the link) unless the number of retransmissions has exceeded the maximum allowed.

The 82588 maintains a retry counter that is incremented after each collision. If retransmission is successful, the user is notified. If the number of retries equals the programmed maximum, an error is reported. The number of allowed retries is configurable from 0 to 15 attempts. The only difference between transmission and retransmission is that transmission clears the retry counter and retransmission increments it.

On completion of transmission and retransmission, the 82588 reports the total number of collisions that have occurred and whether it equalled or exceeded the maximum. It also indicates if the chip had to defer to passing traffic on its first transmission attempt.

After transmission has started the 82588 attempts to transmit the entire frame. In the normal case, frame transmission is completed and the host is notified through interrupt and status register. Otherwise, one or more of the following events causes transmission to terminate prematurely: Clear-to-Send signal goes inactive during transmission, data transfer rate from memory to the chip did not keep up with transmission (DMA underrun), Carrier Sense goes inactive during transmission (if the chip is in the mode where carrier sense is expected during transmission), a collision is detected via Collision Detect Signal. These events are also reported to the CPU via the status registers.

The user may attempt to abort transmissions. Upon receipt of the ABORT command, the 82588 transmits a Jam pattern to cause a CRC mismatch. The chip reports to the host either that the ABORT succeeded or that the frame transmission was completed before the ABORT was accepted.

4.7 Priority Mechanism

The 82588 implements two different priority mechanisms: linear and accelerated contention resolution. Either may be used to distribute relative priority among stations.

Linear priority determines the number of Slot Times that the 82588 waits after deferring or after the end of the Backoff Time (whichever occurs last) before transmitting. If the link becomes busy during the wait period, the process of deferring and waiting starts again. The Linear priority is programmable from 0 to 7. Zero provides the highest priority and is IEEE 802.3 compatible.

Accelerated contention resolution extends the range from which the random number for backoff is drawn. The random number for the first Backoff delay is in the range $0 \leq r < 2^P$, where P is configurable from 0 to 7 with 0 providing the highest priority.

4.8 Frame Reception

The 82588 receives and passes to memory every frame whose address matches its individual, multicast or broadcast address. By configuring the chip to the promiscuous mode, it will receive and pass to memory all frames regardless of address.

The 82588 constantly monitors the serial link activity. When the link becomes active, the 82588 starts accepting the incoming bits.

The Preamble and SFD (BOF flag) are discarded. The 82588 compares the incoming frame's Destination Address to its own Individual Address. If there is an address match and the frame length equals or exceeds 6 bytes, the 82588 passes the Destination Address, Source Address, and Information fields to system memory via the DMA controller. The 82588 verifies correctness during reception of the FCS field. If there is no address match, the 82588 does not request DMA controller's attention and becomes ready to receive the next frame.

When the reception is completed, the 82588 updates a set of status registers and raises its Interrupt signal to inform the CPU. If the received frame has errors (CRC violation, alignment error, DMA overrun), the CPU can reclaim the memory used to store that frame. The next received frame can then be stored in the reclaimed memory.

When configured to IEEE 802.3 End of Carrier delineation, end of frame is indicated by the carrier going inactive. The number of bits after the BOF flag must be a multiple of eight. Residual "dribble" bits are discarded, and not included in the Frame Check Sequence. An alignment error is reported when the frame is "misaligned" (has got residual bits) and a CRC error is detected.

When configured to Bitstuffing delineation, the 82588 performs zero bit deletion, discards the EOF flag, and all following bits until end of carrier. Residual bits are discarded in a manner similar to the End of Carrier method. An error is reported if the carrier goes inactive prior to recognition of an EOF flag.

The minimum frame length is configurable in the range from 6 to 255 bytes. Any frame containing less than the minimum (configured) number of bytes is presumed to be a fragment resulting from a collision or an aborted transmission. Any frame shorter than 6 bytes is discarded without notifying the user.

SINGLE BUFFER MODE

The received frame is transferred to the memory buffer in the format shown in Figure 6. This method of reception is called "Single Buffer" reception. The entire frame is contained in one continuous buffer. Upon completion of reception the total number of bytes written into the memory buffer is loaded into status registers 1 and 2, and the status of the reception itself is appended to the received frame. An interrupt to the CPU follows.

MULTIPLE BUFFER MODE

If the frame size is unknown, memory usage is optimized by using "Multiple Buffer" reception. The frame, as it comes in, is deposited into a series of fixed size buffers. This way the user does not have to allocate large memory space for the short frames. Instead, the 82588 can dynamically allocate memory space as it receives frames. This method requires both data channels alternately to receive the frame. As the frame reception starts, the 82588 interrupts the CPU and automatically requests assignment of the next sequential buffer. The CPU does this and loads the second DMA channel with the next buffer information so that the 82588 can immediately switch to the other channel as soon as the current buffer is full. When the 82588 switches from the first to the second buffer it again interrupts the CPU requesting it to allocate another buffer on the other (previous) channel in advance. This process continues until the entire frame is received. The received frame is spread over multiple memory buffers. The link between the buffers is easily maintained by the CPU using a buffer chain descriptor structure in memory (see Figure 7).

This dynamic (pre) allocation of memory buffers results in efficient use of available storage when handling frames of widely varying sizes. Since the buffers are pre-allocated one block in advance, the system is not time critical.

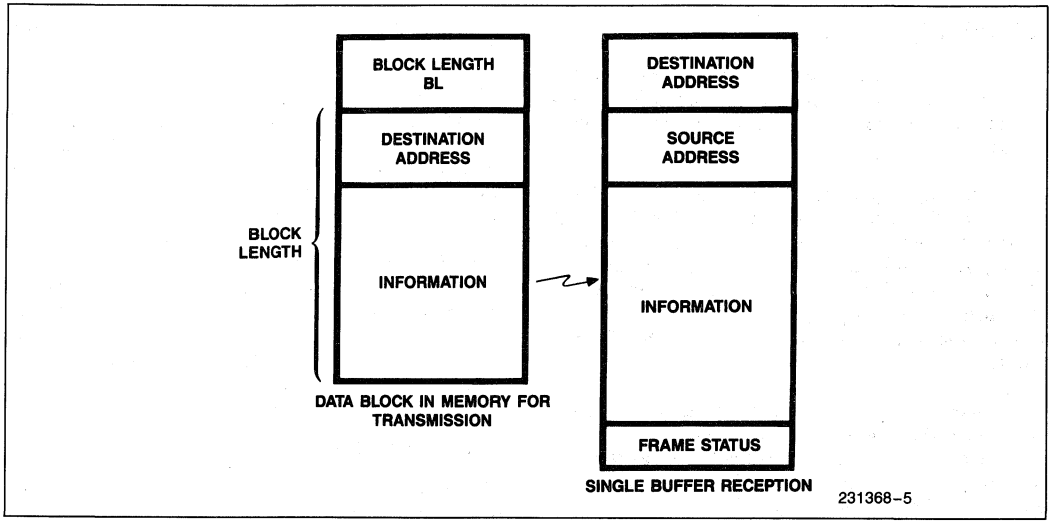


Figure 6. Single Buffer Reception

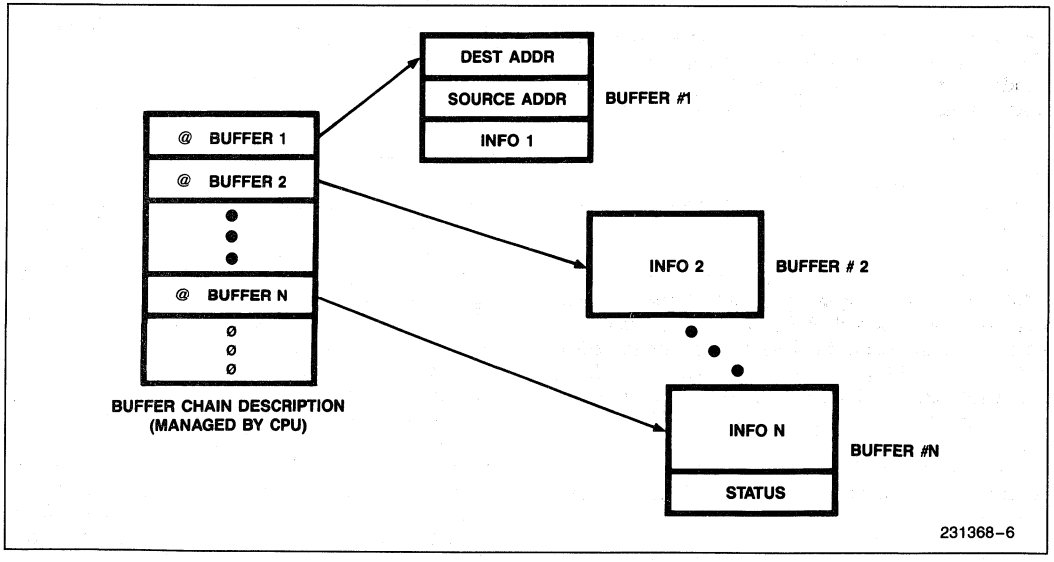


Figure 7. Multiple Buffer Reception

4.9 Physical Link Interface

The Physical Link Interface to a CSMA/CD network includes the following functions: clock generation, data encoding, data decoding, carrier sensing, collision detection and transceiver handshake. The 82588 can be programmed to support either of two interfaces. In Mode 0 it supports a highly integrated interface that provides most of these functions on chip and requires minimum external support logic. In Mode 1 it supports a highly flexible interface, that is identical to the physical inteface of the 82586 and requires external logic in order to connect to the link. The Physical Link layer functions are presented below in three groups: clock generation, data encoding and recovery; detecting carrier and collisions; handshake with the transceiver.

4.9.1 CLOCK GENERATION/ENCODING/ DECODING

In Mode 0, the user can provide the 82588 with an external sampling clock or provide it with a crystal and let the 82588 generate the sampling clock. The bit clock is generated internally from an external clock signal or from an on chip crystal oscillator that runs at 8× or 16× the data bit rate. The frequency range of the crystal or the external clock can vary between DC and 16 Mbps. The clock must also be programmed to either 8× or 16× of the bit rates. In Mode 1, the 82588 receives externally generated Transmit Clock ($\overline{TXC}$) and Receive Clock ($\overline{RXC}$) at the exact bit rate. This frequency can vary from D.C. to 5 MHz.

ENCODING/DECODING

In Mode 0, the waveforms sent out by the 82588 on the TXD pin and received on the RXD pin, are programmable to be either Manchester, Differential Manchester or NRZI encoded. The idle state of the signal is high.

In Mode 1, the 82588 transmits and receives NRZ encoded data, that is synchronized to the respective clocks. Encoding/decoding to/from different schemes is done externally. The 82588 can be programmed to generate Manchester encoded data in this mode (in the frequency range of 1–5 Mbps).

Figure 8 depicts various kinds of data encoding methods implemented in the 82588.

Figure 9 illustrates the encoding rules for these methods.

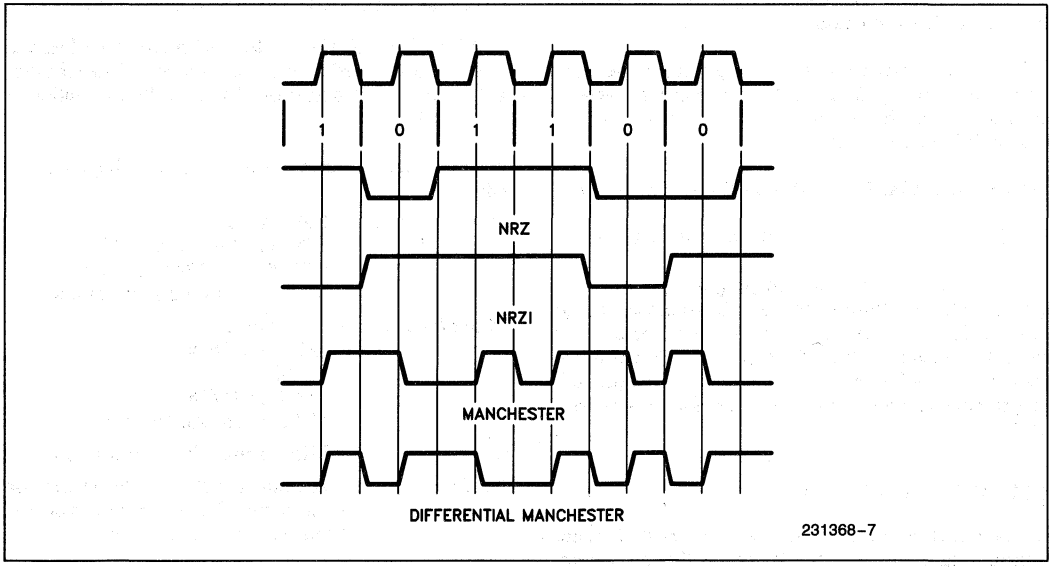


Figure 8. 82588 Data Encoding Methods

Encoding Method	Mid Bit Cell Transitions	Bit Cell Boundary Transitions
NRZ	do not exist	identical to original data
NRZI	do not exist	exist only if original data bit equals 0. Dependent on present encoded signal level: to 0 if 1 to 1 if 0
MANCHESTER	exist for every bit of the original data: from 0 to 1 for 1 from 1 to 0 for 0	exist for consequent equal bits of original data: from 1 to 0 for 1 1 from 0 to 1 for 0 0
DIFFERENTIAL MANCHESTER	exist for every bit of the original data. Dependent on present Encoded signal level: to 0 if 1 to 1 if 0	exist only if original data bit equals 0. Dependent on present Encoded signal level: to 0 if 1 to 1 if 0

Figure 9. 82588 Data Encoding Rules

4.9.2 CARRIER SENSE/COLLISION DETECT

CARRIER SENSE

Carrier Sense indicates that there is activity on the link, i.e., the signal from a transmitting station has reached the station that is checking.

In Mode 0, Carrier Sense signal is generated internally from the incoming data. In this mode carrier sense is consider active after detection of 3 consecutive edges. Carrier is considered inactive after:

DIFF/MANCHESTER -1.5 bit times of high level.

NRZI -8 bit times of high level.

In Mode 1, the 82588 can be programmed to either generate Carrier Sense internally (for transceivers that generate RXC only during actual reception) or accept it from the outside, via the CRS# pin. The Carrier Sense may undergo noise filtering, i.e., it must be present for a programmable number of bit times before it is considered valid.

COLLISION DETECTION

Collision Detect indicates that two or more stations are transmitting simultaneously. When a collision is detected during transmission, the 82588 jams the link, stops transmitting and goes into backoff. Jamming will not start unless preamble transmission is first completed. Collisions are detected during receptions, causing a receive frame abortion. (Only when not in external loopback mode). The 82588 can be programmed to detect collisions internally or to accept it from the outside, via the CDT# pin. The Collision Detect signal may undergo filtering, similar to Carrier Sense.

COLLISION DETECTION BY CODE VIOLATION

In Mode 0, a collision is detected internally when the receive data experiences code violations (Manchester, Differential Manchester or NRZI) while the station is transmitting.

A code violation occurs if any of the following is detected:

×8 Manchester - Pulse width:
1/8 to 2/8 or 11/8 to 12/8
"0" level for 13/8 or more
Missing mid-bit-cell transition

×16 Manchester - Pulse width:
1/16 to 5/16 or
11/16 or
21/16 to 24/16
"0" level for 25/16 or more
Missing mid-bit-cell transition

NRZI - A transition that is 1/4 bit time or more out of phase. More than two transitions in half bit cell

COLLISION DETECTION BY BIT COMPARISON

An additional mechanism of bit comparison is provided. This mechanism compares the "signature" of the transmitted data to the signature of the received data for the duration of the collision window (one slot time). (Wherever 588 configured to internal collision detection.)

A collision is reported if, after the transmission of the Opening Flag, any of the following conditions becomes true:

1. Half a slot-time has elapsed and Carrier Sense did not go active.
2. Half a slot-time + 16 bit times have elapsed and an opening flag was not yet recognized (16 bit times are a margin factor).
3. Carrier Sense went inactive after an opening flag was received - Transmit still active.
4. Collision window has elapsed and Transmit signature differs from Receive signature.

In mode 1, internal collision detect works only with transceivers that do not return the transmitted signal. In that case, detection of a carrier during transmission indicates a collision.

NOTE:
For broadband applications the slot time is usually twice the round trip delay.

5.0 82588 NETWORK MANAGEMENT AND DIAGNOSTIC FUNCTIONS

The 82588 includes a set of features for improving reliability and testability.

The 82588 offers four diagnostic services: first, monitoring the transmission and reception of frames; second, gathering statistics and diagnostics about the network; third, diagnostic support for a particular station on the network; fourth, a means to test the proper operation of the chip itself.

5.1 Transmission/Reception Error Reporting

The 82588 stores status information after completing transmission or reception of every frame. If transmission or reception is successful, the OK status bit is set. If transmission is unsuccessful, the cause is given in the status registers.

The 82588 reports on the following events after each transmitted frame:

— Lost Carrier Sense: Carrier sense signal did not become active or was lost during transmission.

— Lost Clear-to-Send: Clear to send signal was removed during transmission.

— DMA underrun occurred because the system bus did not keep up with the transmission.

— The number of collisions equals the maximum allowed.

Any of these events causes the report of an unsuccessful transmission.

The 82588 checks each incoming frame and reports on the following errors:

— CRC error: incorrect CRC in a well aligned frame.

— Alignment error: incorrect CRC in a misaligned frame. A misaligned frame with a correct CRC is not reported by the 82588. (Considered a good frame.)

— Frame too short: the frame is shorter than the configured value for minimum frame length.

— No EOF flag: valid only in Bitstuffing mode, carrier went inactive before EOF flag detection.

— Overrun: the frame was not completely placed in memory because the system bus did not keep up with coming data.

The occurrence of any of these events causes the report of an unsuccessful reception.

5.2 Network Planning and Maintenance

To perform proper planning, operation, and maintenance of a communication network, the network management entity must accumulate information on network behavior. The 82588 provides a rich set of network diagnostics that can serve as the basis for a network management entity. The features include: gathering network activity information, saving all frames that appear on the link (optional), and locating opens or shorts on the link.

Network activity information is provided in the status available to the host after each frame is transmitted. The activity indicators are:

1. Number of collisions: the number of collisions the 82588 experienced in attempting to transmit this frame.
2. Deferred transmission: indicates if the 82588 had to defer to traffic on the link during the first transmission attempt.

The 82588 can be configured in the Promiscuous Mode. In this mode the 82588 captures all frames transmitted on the network regardless of the Destination Address. This mode is useful in implementing a monitoring station to capture all frames for network analysis.

The 82588 is capable of determining if there is a short or open circuit anywhere in the network using the Time Domain Reflectometry (TDR) command. When a TDR command (see section "Controlling the 82588") is issued, the chip transmits a pulse train and measures the reflection return time. If the network is properly terminated, there are no reflections so the timer runs out and the user is notified that there are no link problems. If a problem is detected, the distance to the reflection source and reason (short or open) are reported.

5.3 Station Diagnostics

To support Station Diagnostics, the 82588 can be configured to External Loopback.

In this mode the 82588 operates full duplex at full speed. The actual maximum throughput depends on bit rate and system bus speed. The transmitter to receiver interconnection can be placed anywhere between the 82588 and the link.

5.4 82588 Self Testing

The 82588 provides several features for self testing.

The 82588 can be configured to Internal Loopback. In this mode, the 82588 disconnects itself from the internal or external Serial Interface Units, and any frame transmitted is received immediately. The 82588 connects the Transmit Data to the Receive Data signal and the Transmit Clock to the Receive Clock. The 82588 can be configured in the External loopback mode. In this mode, the Transmitted signal at full data rate comes out at the TxD pin and can be returned to the RxD pin through any external circuitry. This works full duplex at the full data rate. It tests the complete transmitter, receiver and the external path. Equality between the transmitted and received frames implies that a large portion of the chip operates correctly. In addition, internal loopback can be used in conjunction with inhibiting the source address insertion and/or CRC insertion by the chip. For example, in internal loopback, if a frame is transmitted with an erroneous CRC (using CRC inhibition), the CRC checking mechanism must detect a CRC error. In internal loopback the transmission and reception occurs at one fourth the programmed data rate. The Dump Command causes the 82588 to write 63 bytes of its internal registers to memory. This is a very powerful capability that can serve as the basis for comprehensive diagnostics.

There are parts of the chip, in particular the logic, that uses the Backoff random number generator that cannot be checked from the outside. The Diagnose Command initiates a self test procedure that exercises these otherwise unaccessible registers and counters, and reports the result.

6.0 INITIALIZING/CONFIGURING THE 82588

6.1 Initializing the 82588

A hardware or software reset brings the 82588 to its initial state where the 82588 is put into the idle state with the transmitter, receiver clock generator, interrupts, etc., inactive. To use the 82588 it has to be configured with a set of parameters.

6.2 Configuring the 82588

This is done using the CONFIGURE command. This command initiates the writing of the configuration parameters into the 82588 using DMA. The configuration parameters are shown in Figure 10.

	Configurable Parameters	Default Values on Reset
1.	Serial Interface	*
2.	Sampling Rate	*
3.	Oscillator Range	*
4.	FIFO Limit	*
5.	Chaining	*
6.	Buffer Length	0
7.	Preamble Length	8
8.	Address Length	6
9.	No Source Address Insertion	0
10.	Promiscuous Mode	0
11.	Broadcast Disable	0
12.	CRC-16/CRC-32	0
13.	No CRC Insertion	0
14.	Padding	0
15.	Bitstuffing/EOC	0
16.	Min Frame Size	64
17.	Interframe Spacing Time	96
18.	Slot Time	512
19.	Number of Retries	15
20.	Linear Priority	0
21.	Back-off Method	0
22.	Manchester/NRZI	0
23.	Diff. Manch/Manchester	0
24.	CRS Filter	0
25.	Internal CRS	0
26.	CDT Filter	0
27.	CDBBC	0
28.	Internal CDT	0
29.	INT Loopback	0
30.	EXT Loopback	0
31.	Transmit on No CRS	0
32.	Exponential Priority	0

*these parameters must be configured after RESET

Figure 10. Configuration Parameters

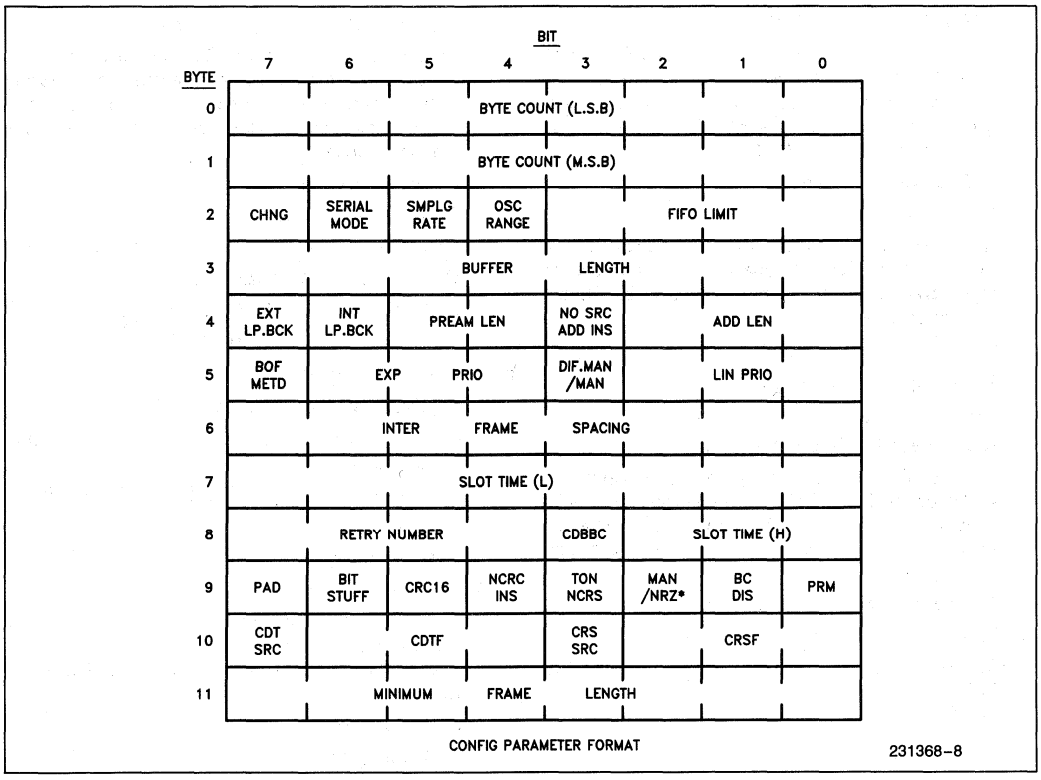


Figure 11. Configuration Block

All but 5 parameters have a default value after reset. Figure 11 shows the configuration block in memory on issuing the configure command. The first two bytes contain the number of bytes (byte count) which follow. Note that all the parameters not having a default value on reset are contained in the first bytes after the byte count field. The minimum byte count is 1 and maximum 10 - values greater than 16 are interpreted as module 16. (Meaning only the 4 lsb bits are used). After reset a configure command with byte count of minimum 1 is essential for the operation of the 82588.

6.3 Configuration Parameters

1. Serial Interface Mode - Byte 2, Bit 6

   This bit selects between high integration (mode 0) and high speed (mode 1) modes. It also selects the function of the TXC/X1, RXC/X2 and TCLK/CRS pins.

   In High Integration Mode, the 82588 generates its Internal clock from the x1, x2 input; encodes/decodes the data waveform (either Manchester, Differential Manchester or NRZI); generates the Carrier Sense and (optionally) the Collision detect from the incoming RXD signal.

   In High Speed Mode, the Clock Generation, Data Encoding/Decoding, Carrier Sense and Collision Detect, are generated externally.

   0 - mode 0, high integration

   1 - mode 1, high speed

   Default setting: None

2. Sampling Rate - Byte 2, Bit 5

   Valid only in High Integration Mode. It specifies the ratio between the frequency of the sampling (serial) clock and the data bit rate:

   0 - 8×

   1 - 16×

   Default setting: None

3. Oscillator Range - Byte 2, Bit 4

   Valid only in High Integration Mode. It specifies the frequency range of the sampling (serial) clock:

   0 - high range (1 – 16 MHz)

   1 - low range (DC – 1 MHz)

   Default setting: None

4. FIFO - Limit - Byte 2, Bits 0–3

   Specifies the byte count in the 16 byte FIFOs at which the 82588 is to resume requests for transfer of data to/from memory. This parameter is used for the Transmit FIFO - it's 2's complement is used for the Receive FIFO.

   Default setting: None

5. Chaining - Byte 2, Bit 7

   If this bit is set, the 82588 switches the DMA during reception channel whenever the number of bytes transferred to memory equals the Buffer Length parameter.

   0 - Single Buffer reception

   1 - Multiple Buffer reception

   Default setting: None

6. Buffer Length - Byte 3, Bits 0–7

   Specifies the length of the buffer, after which the DMA channel is switched. This parameter is only valid when the Chaining bit is set. It is specified in units of 4 bytes. Buffer length of 0 is interpreted as 1024 bytes.

   Default setting: 0 (= 1024 bytes)

7. Preamble Length - Byte 4, Bits 4–5

   Selects the length of the Preamble (including the SFD).

   00 - 2 bytes

   01 - 4 bytes

   10 - 8 bytes

   11 - 16 bytes

   Default setting: 10 (= 8 bytes)

8. Address Length - Byte 4, Bits 0–2

   Determines the length, in bytes, of the address that the 82588 refers to. This applies to source, destination, Multicast, and broadcast addresses. Address length of 7 is treated as zero.

   Default setting: 6 bytes

9. No Source Address Insertion - Byte 4, Bit 3

   0 - Source address is inserted by the 82588 during transmission

   1 - Source address insertion is disabled (source address is provided by the user in the transmit block)

   Default setting: 0

10. Promiscuous Mode - Byte 9, Bit 0

    When set, causes 82588 to receive all frames regardless of their destination address.

    0 - Promiscuous mode off

    1 - Promiscuous mode on

    Default setting: 0

11. Broadcast Disable - Byte 9, Bit 1

    Disables reception of any frame with a broadcast destination address. Protects against overloading stations with limited resources. Note that when promiscuous mode (when set) is on, Broadcast is always enabled.

    0 - Broadcast enabled

    1 - Broadcast disabled

    Default setting: 0

12. CRC-16/CRC-32 - Byte 9 Bit 5

    0 - 32 bit Autodin - 11 CRC (IEEE 802.3)

    1 - 16 bit CCITT CRC (HDLC)

    Default setting: 0

13. No CRC Insertion - Byte 9, Bit 4

    0 - CRC is inserted by the 82588 after the information field

    1 - No CRC insertion (CRC is provided by the user)

    Default setting: 0

14. Padding - Byte 9, Bit 7

    Valid for bitstuffing only. If padding is set, the frames shorter than 1 slot time will be padded with flags to the shortest frame that is longer than 1 slot time.

    0 - Padding off

    1 - Padding on

    Default setting: 0

15. Bitstuffing/EOC - Byte 9, Bit 6

    0 - End of Carrier Framing (i.e., IEEE 802.3)

    1 - Bitstuffing Framing (HDLC, with 01111110 flag and zero bit insertion/deletion)

    Default setting: 0

16. Min-Frame-Size - Byte 11, Bit 0–7

    The minimum frame size, (not including preamble) in bytes. The 82588 sets a status bit if a shorter frame is received. Note that frames shorter than 6 bytes are discarded with no status update.

    Default setting: 64 bytes

17. Interframe Spacing - Byte 6, Bits 0–7

    Specifies the time period, in bit times, that the 82588 must wait after detecting that the Carrier Sense dropped and before it can begin transmission or reception of a frame. The minimum value is 12 and any value less than that will default to 12.

    Default setting: 96

18. Slot Time - Byte 7, Bits 0–7; Byte 8, Bits 0–2

    The slot time for the network, in Bit Times. This value is used in calculating backoff and Linear Priority delays. It must be longer than the maximum roundtrip time of a frame in the network plus the jam time. Zero is interpreted as 2**11.

    Default setting: 512

19. Number of Retries - Byte 8, Bit 4–7

    The maximum number of retries, due to collisions, that the 82588 performs. This information gives the CPU the choice of aborting transmission after a programmed number of collisions.

    Default setting: 15

20. Linear Priority - Byte 5, Bit 0–2

    The number of slot times that the 82588 waits after Inter Frame Spacing or after Backoff, before enabling transmission. A higher number reduces the priority.

    Default setting: 0

21. Backoff Method - Byte 5, Bit 7

    Determines the method for starting the backoff timer:

    0 - According to the IEEE 802.3 standard; immediately
    after the end of transmission

    1 - After the interframe spacing period expires

    Default setting: 0

22. Manchester/NRZ(I) - Byte 9, Bit 2

    This bit specifies the data encoding/decoding method - it is mode dependent:

	High Integration	High Speed
0 -	NRZI (*)	NRZ
1 -	Manchester	Manchester (**)

    * Bitstuffing should also be applied

    ** Manchester encoding is performed on transmitted data (TXD)

    NRZ data should, however, be provided for reception (External Manchester decoding)

23. Differential Manchester/Manchester - Byte 5, Bit 3

    Valid only in High Integration Mode provided that Manchester is set in preceding parameter. This bit specifies the data encoding/decoding method:

    0 - Manchester

    1 - Differential Manchester

    Default setting: 0

24. CRS - Filter - Byte 10, Bits 0–2

    Specifies the required width of CRS in bit times, before it is recognized as being active. Carrier sense deactivation is recognized immediately.

    Default setting: 0

25. CRS Source - Byte 10, Bit 3

    Valid only in High Speed Mode. Specifies whether Carrier Sense is to be performed internally or externally (via CRS pin).

    0 - External

    1 - Internal

    Default setting: 0

26. CDT-Filter - Byte 10, Bit 4–6

    Specifies the required width of CDT, in bit times, before it is recognized that a collision has occurred.

    Default setting: 0

27. CDT Source - Byte 10, Bit 7

    Specifies for both High Integration and High Speed Modes whether Collision Detect is to be performed internally or externally (via CDT pin):

    0 - External

    1 - Internal

    Default setting: 0

28. CDBBC - Byte 8, Bit 3

    Valid only in High Integration Mode. Specifies whether Collision Detect by Bit Comparison is to be performed.

    0 - CDBBC off

    1 - CDBBC on

    Default setting: 0

29. INT - Internal Loopback - Byte 4, Bit 6

    When set, the 82588 disconnects itself from the serial link and logically connects TXD to RXD and TXC to RXC.

    Note: If set, it overrides EXT-loopback

    0 - Internal Loopback off

    1 - Internal Loopback on

    Default setting: 0

30. EXT - External Loopback - Byte 4, Bit 7

    When set, the 82588 will transmit and receive simultaneously, at full rate. This allows checking of external hardware as well as the serial link to the transceiver interface. The user is responsible for external transmit-receive interconnection. It is overridden by INT-Loopback.

    0 - External Loopback off

    1 - External Loopback on

    Default setting: 0

31. Transmit on No CRS - Byte 9, Bit 3

    When set, allows transmission even if there is no CRS back from the transceiver. Required for transceivers that do not return carrier during transmission.

    0 - Transmit only when CRS present

    1 - Transmit only when no CRS present

    Default setting: 0

32. Exponential Priority - Byte 5, Bit 4-0

    Extends the range from which the random number for back-off is selected (i.e., first collision is biased to a number different than 0, 1)

    Default setting: 0

7.0 CONTROLLING THE 82588

This chapter specifies how to control the 82588, from the user's point of view. It starts with the definition of the command and status registers. It then specifies the behavior of the 82588 during execution of operations and reception of frames. The next chapter provides the detailed format and meaning of all the operations and their status.

The 82588 performs OPERATIONS, such as transmitting a frame, enabling reception, configuring the chip, performing diagnostics, or aborting an operation. An operation is initiated by the CPU, by writing a COMMAND into the command register in the 82588. There are three groups of operations: EXECUTION, RECEPTION, and STATUS POINTERS CONTROL (in addition to a Software Reset). The first two groups correspond to two logical units of the 82588: the EXECUTION UNIT and the RECEIVE UNIT. The third group provides control for the status registers.

There is a set of EVENTS that may occur asynchronously in response to commands (typically as a result of the behavior of the link). These events include: end of incoming frame, frame aborted, execution of a command is completed, execution aborted. The 82588 reports on these events by updating a set of status registers and raising the INTERRUPT signal. There are four status registers (STATUS 0 → STATUS 3), which are all read by the CPU, from the same address. An internal two bit POINTER is used to determine which register is being read. The pointer is advanced to the next status register automatically when status is read. It can also be set explicitly by a field in the command. Status pointer progress may be disabled by a command that fixes it to a particular status register and allows reading of the same register. Information read from any of the first 3 status registers is valid provided Interrupt signal is high or Bit 7 of Status 0 is set. Information of status register 3 is continuously updated, provided the register pointer is not fixed at three.

Status is updated and interrupt raised following an event only when the interrupt signal is low. When the interrupt is high, the CPU can read the status. The interrupt is cleared by a command from the CPU where the INTERRUPT ACKNOWLEDGE bit is set. Note that it is the responsibility of the CPU to clear interrupt in order to prevent a dead lock.

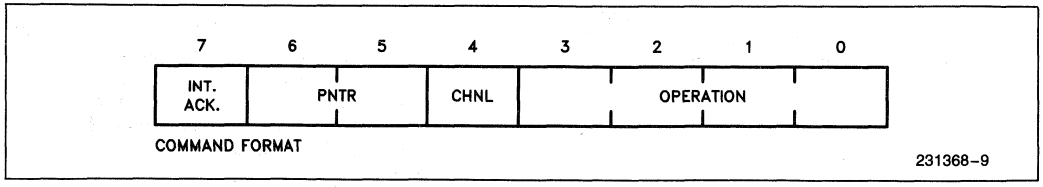
The 82588 interacts with the DMA controller via two REQUEST/ACKNOWLEDGE lines (referred to as two CHANNELS). The 82588 requests the transfer of bytes, and the DMA performs and acknowledges the transfer. Frames and parameters are transferred via the two channels from/to memory. Commands allocate a specific channel for each operation.

The 82588 can be configured to support receive buffer CHAINING. In this mode, two channels must be allocated to the reception of frames and the 82588 switches channels when a buffer is full. It issues a "Request Alternate Buffer" interrupt, in order to allow the CPU to set up the alternate buffer on the other channel.

7.1 The Command Register

This section specifies the format of the 82588 command set. A command is given to the 82588 by writing into the command register. The command can be issued at any time, but in case it is not accepted, the operation is treated like a NOP and will be ignored (although the INT and the PTR will be updated).

Format:



7.1.1 OPERATIONS (BITS 0–3)

The OPERATION field initiates a specific operation. There are three groups of operations that can be initiated independently: EXECUTION, RECEPTION, or STATUS POINTER CONTROL.

The 82588 implements the following EXECUTION operations (shown with their codes):

*	NOP	—	0
*	IA-SETUP	—	1
*	CONFIGURE	—	2
*	MC-SETUP	—	3
*	TRANSMIT	—	4
*	TDR	—	5
*	DUMP	—	6
*	DIAGNOSE	—	7
*	RETRANSMIT	—	12
*	ABORT	—	13

Execution operations except ABORT are accepted only when the Execution unit is IDLE. ABORT is only accepted if the Execution unit is active.

Following are the reception operations (and their codes):

*	RCV-ENABLE	-	8
*	ASSIGN ALT BUF (chaining only)	-	9
*	RCV-DISABLE	-	10
*	STOP-RCV	—	11

RCV-ENABLE is accepted only when the Receive Unit is IDLE. The other receive operations are only accepted if the Receive Unit is NOT IDLE.

Status Pointer Control Operation:

*	FIX PTR	—	15 (1)
*	RLS PTR	—	15 (0)

The bit in brackets is Bit #4 of the command (channel bit).

The last operation is software reset:

*	RESET	—	14

RESET can always be issued. It has the same effect as a hardware reset.

Operations that are not accepted are simply ignored, (although INT and PTR are updated).

7.1.2 CHANNEL ALLOCATION (BIT 4)

The CH field selects the channel for the operation that is requested. Note this field applies only for IA-SETUP, CONFIGURE, MC-SETUP, (RE) TRANSMIT, DUMP and RCV-ENABLE. If the selected channel is already allocated the operation is ignored (acts like NOP).

7.1.3 SETTING THE POINTER

The PTR field specifies the new value of the internal pointer. The pointer selects the next status register to be read. It is advanced (module 4) automatically when status is read. Automatic advancement may be disabled by issuing the FIX PTR command.

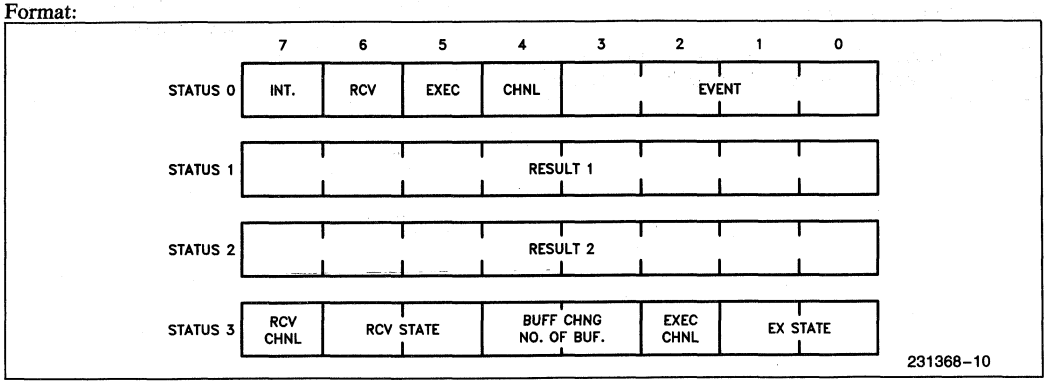
7.1.4 ACKNOWLEDGING INTERRUPT (BIT 7)

The INT-ACK bit, if set, causes the interrupt hardware signal and the interrupt bit to be cleared. This is the only way to clear the interrupt bit and reset the interrupt signal other than by a reset.

7.2 The Status Registers

There are four Status Registers that may be read by the user. They are numbered 0–3. The register that the internal pointer points to, is read by doing a "read" from the 82588. STATUS 1 and STATUS 2 are also called "result Registers."

Format:



The 82588 provides the information about the last operation that was executed, or the last frame received, via the first three status registers. The fourth status register provides the state of the chip itself.

STATUS 0, STATUS 1, and STATUS 2 are only updated when INT = 0. STATUS 3 is updated whenever an update is needed. Status register 3 is **not** updated if the pointer is pointing to it.

7.2.1 STATUS 0

The first status register (STATUS 0) includes the interrupt bit and the cause of the interrupt. Information about the relevant channel is also provided.

7.2.1.1 Event (Bits 0–3)

The event field specifies the reason why the 82588 needs attention of the CPU.

The following events may occur (shown with their codes):

* IA-SET-DONE - 1
* CONFIGURE-DONE - 2
* MC-SETUP-DONE - 3
* TRANSMIT-DONE - 4
* TDR-DONE - 5
* DUMP-DONE - 6
* DIAGNOSE-PASSED - 7
* END OF FRAME - 8
* REQUEST ALT BUFFER - 9 (chaining only)
* RECEPTION ABORTED - 10
* RETRANSMIT-DONE - 12
* EXECUTION-ABORTED - 13
* DIAGNOSE-FAILED - 15

7.2.1.2 Channel (Bit 4)

The CH bit indicates which channel was allocated for the operation that caused the event. Valid for all events except for Request Alternate Buffer.

7.2.1.3 Execution (Bit 5)

The EX bit indicates the completion of an execution operation. Event number 13 indicates the abortion of the operation.

7.2.1.4 Receive (Bit 6)

The RCV bit indicates that the reception of a frame was completed, or a new buffer is required to allow reception to proceed. Event numbers 8–10 provide the details.

7.2.1.5 Interrupt (Bit 7)

The INT bit is set together with the hardware interrupt signal. Setting the INT bit indicates the occurrence of an event. This bit is cleared by any command whose INT-ACK bit is set.

7.2.2 STATUS 1/STATUS 2 (RESULT REGISTERS)

These registers hold the result of completing an operation. STATUS 1 is the least significant byte and STATUS 2 is the most significant. The Result Registers are only updated when any of the following occur:

— TRANSMIT-DONE
— TDR-DONE
— RETRANSMIT-DONE
— END OF FRAME

7.2.3 STATUS 3

The last status register holds the state of the 82588: three lower bits for the Execution unit, five higher bits for the Receiver unit.

7.2.3.1 Execution State (Bits 0–1)

The EX-CH field specifies the state of the Execution Unit. It can take one of the following values:

0 - IDLE: The Exec Unit is disabled

2 - ACTIVE: The Exec Unit is actively executing an operation

3 - ABORT IN PROG: The execution of an operation was aborted, but the completion event was not yet issued.

7.2.3.2 Exec Channel (Bit 2)

The EX-CH bit indicates which channel is allocated for transferring parameters from/to memory. It is only valid when the Execution Unit is active, performing a parametric execution.

7.2.3.3 Buffer Chaining - Number of Buffers (Bit 3-4)

Specifies the number of available buffers for reception, when configured to "Buffer Chaining." It can take one of the following values:

0 - no buffers are available
1 - one buffer is available (automatic on RCV-enable)
2 - two buffers are available

7.2.3.4 Receive State (Bits 5-6)

The RCV-STATE field specifies the state of the Receive Unit. It can take one of the following values:

0 — IDLE	The Receive Unit is disabled
1 — READY	The Receive Unit is set up for an incoming frame
2 — ACTIVE	The Receive Unit is actively transferring bytes to memory (provided buffer is available)
3 — STOP IN PROGRESS	The Receive Unit is active and will be disabled following the reception of current frame

7.2.3.5 Receive Channel (Bit 7)

The RCV-CH bit indicates which channel is allocated for transferring incoming data to system memory. It is valid only when the Receive Unit is **NOT** IDLE. When configured for Buffer Chaining, at least one buffer should be available. 82588 assumes that one buffer is available whenever the receiver is enabled. The DMA controller must be initialized and ready to transfer data prior to enabling the receiver.

7.3 Performing Execution Operations

The 82588 has a repertoire of 11 execution operations. These operations perform transmission, configuration, diagnostics and abortion. All the operations are initiated by writing into the Command Register. This causes the Execution unit to become Active. Some of the operations read parameters from memory.

The parameters (for IA-SETUP, CONFIGURE, MC-SETUP, TRANSMIT and RETRANSMIT) are organized in a block that starts with a 16 bit byte-count. The byte count specifies the length of the rest of the block. Before beginning the operation, the DMA pointer of the selected channel must point to the first byte of the byte count. There is no restriction on the memory structure of the frame as long as for every DMA request from the 82588 it receives the next byte of the frame. Transferring the bytes is the job of the DMA controller or the CPU. If the DMA does not keep up with transmission (i.e., causes an underrun), the transmission is terminated and an underrun is indicated.

The 82588 requests the 2 bytes of byte-count on the allocated channel and determines the length of the parameter block. It then requests the parameters and starts the execution of the operation.

Upon completion of the operation, (when interrupt is LOW) the Exec unit does the following: it updates STATUS 0; updates STATUS 1 and STATUS 2 (for TRANSMIT, TDR or RETRANSMIT); raises the interrupt signal; and goes IDLE.

The NOP operation is ignored, exactly like any operation that is initiated during the execution of another one.

The ABORT operation causes a request for abortion of the operation that is in progress. In some cases the operation cannot be aborted anymore (e.g., for a Transmit operation when the whole frame was already transmitted). In these cases the Abort operation is ignored. Non-parametric executions like: TDR and DIAGNOSE cannot be aborted.

The DUMP operation causes a set of internal registers to be written to memory over the allocated channel. It is completed by updating STATUS 0 and raising the interrupt.

7.4 Reception of Frames

The 82588 receives and passes to memory all frames whose address matches the individual, multicast or broadcast address. Reception of frames is pipelined, i.e., the reception of a frame can start before the end of frame processing of the previous frame is done. 82588 is configured to either "chaining" or "consecutive" mode. In consecutive mode the whole incoming frame is passed to a single buffer in memory via the allocated channel. In chaining mode, the frame is received and deposited in a series of fixed size buffers using the two available DMA channels alternately. The chained buffer size if configurable in steps of 4 bytes to a maximum of 1K bytes.

7.4.1 CONSECUTIVE MODE (SINGLE BUFFER MODE)

Before the reception starts, the DMA pointer of the respective channel points to the first byte of the available structure of the frame. Transferring the bytes is the job of DMA controller or the CPU.

If the receiver is ready and a frame arrives, the 82588 requests the transfer of bytes to memory, using DMA. After transferring all the bytes (addresses, control and information) the 82588 transfers two "frame status" bytes to memory. Upon completion of the reception, and when interrupt is LOW, the Receive Unit does the following: it updates STATUS 0 with an "end of Frame" event; loads the frame byte count, which includes the frame status bytes, into STATUS 1 and STATUS 2; raises the interrupt signal; and goes to READY state. (The DMA Controller points to the byte following the frame status.)

If the receiver is not ready when the first byte of the frame arrives, then the frame is not received. No status is updated. Disabling reception after the first byte has passed to memory, causes the rest of the frame to be ignored. An INTERRUPT with "Receive Aborted" event is issued.

Reception can begin even when the interrupt that indicates the completion of execution or reception is still pending. This capability allows pipelining of the transfer to memory of a frame by the 82588 and the DMA controller, with the CPU handling the completion of the previous execution or reception (including the interrupt latency). The entire frame gets deposited in a single memory buffer of 64K bytes maximum size.

7.4.2 CHAINING MODE (MULTIPLE BUFFER MODE)

Before the reception starts, the DMA pointer of the first DMA channel points to the first byte of the available buffer. If receiver is ready (one buffer allocated) and a frame arrives, the 82588 generates an interrupt with a "Request Alternate Buffer" event that notifies the CPU to allocate an additional buffer. Simultaneously, the 82588 requests the transfer of bytes to memory. Note that if an additional buffer is available when the first byte arrives, then the "Request Alternate Buffer" event is not issued. This may happen when more than one frame is received sequentially.

When the currently used buffer is full and an additional buffer is available the 82588 makes it the current one (i.e., switches buffers) and continues transferring bytes to memory using the other DMA channel. If an additional buffer is not available, the receiver deasserts the DMA Request signal and suspends further data transfers to memory. The CPU should then provide another buffer or disable receive. If a new buffer is provided, byte transfers to memory continues. If a Disable Receive command is issued then the rest of the frame is ignored.

This process of switching from channel to channel continues until the end of the frame arrives. At this point, two "frame status" bytes are transferred to memory; buffer switch occurs; STATUS 0 is updated; the byte count of the frame is loaded into STATUS 1 and STATUS 2; if a new buffer is available, then the 82588 is ready to receive the next frame. Otherwise, reception is suspended. In both cases, and End of Frame event is issued.

If the receiver is not ready, when the first byte of the frame arrives, then the whole frame is ignored - status is not updated. Disabling reception after the first byte was passed to memory causes the rest of the frame to be ignored, and an interrupt with "Receive Aborted" event to be issued.

If reception is suspended (Ready, but with no buffers assigned) when the first byte of the frame arrives, the receiver interrupts the CPU for a buffer. The user may provide a buffer or disable Reception.

8.0 OPERATIONS AND STATUS

This section specifies all the operations that the 82588 can execute and the events it can generate. The semantics, as well as the format of the parameters, status, and result, are specified. The 4 bit op-code is shown in parentheses.

The value of byte count (where applicable) does not include the 2 bytes of the field itself. All shaded areas mean they are "reserved."

8.1 Operations

8.1.1 NOP (00)

This operations does not affect the 82588. It has no parameters and no result.

8.1.2 IA-SETUP (01)

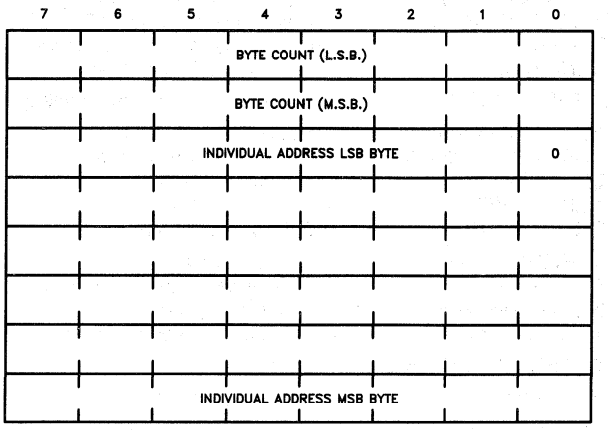
This operation sets up the individual address of the 82588. This address is inserted as the source address during transmission and used for address filtering during reception. (Both can be disabled via configuration parameters.) The Byte Count should match the "Address Length" configuration parameter. The length of the address (in bytes) that is set up is determined by the minimum of:

"Address Length" configuration parameter or the value of the Byte Count field.

Only the 4 least significant bits of the byte count are used.

NOTE:
The first bit of the first byte must be a zero.

The format of the parameter is as follows:



231368-11

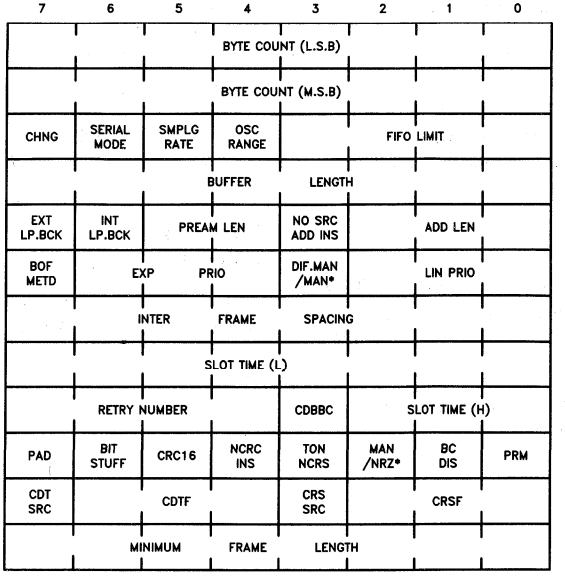
An interrupt with the event IA-SETUP-DONE is issued when this operation is done (unless ABORT was executed in the meantime).

8.1.3 CONFIGURE (02)

This operation configures the 82588. The length of the configurations block that is modified is determined by the byte count. If the byte count is less than 10, then only part of the parameters are updated. Only the least significant 4 bits of byte count are used.

Note that 3 bits (Mode, SMPLG RATE, and OSC RANGE) have an effect only during the first Configure command after RESET.

The format of the parameters is as follows:



231368-12

The interpretation of the fields is as follows:

Byte 0	FIFO Limit (Bits 0–3)	FIFO limit
	OSC RANGE (Bit 4)	High or low frequency range of oscillator
	SMPLG RATE (Bit 5)	Sampling rate. 8× or 16× external oscillator
	MODE (Bit 6)	High integration or high speed mode
	CHAINING (Bit 7)	Receive buffer chaining
Byte 1	BUFFER LENGTH	Buffer length used for chaining
Byte 2	ADR LEN (Bits 0–2)	Address length
	NO SRC ADDR INS (Bit 3)	Address control location
	PREAM LEN (Bits 4–5)	Preamble length
	INT LP BCK (Bit 6)	Internal Loopback
	EXT LP BCK (Bit)	External Loopback
Byte 3	LIN PRIO (Bits 0–2)	Linear priority
	DIF MAN/MAN (Bit 3)	Differential/Manchester Encoding
	EXP PRIO (Bits 4–6)	Exponential Priority
	BOF METD (Bit 7)	Exponential backoff method
Byte 4	INTER FRAME SPACING	Inter frame spacing
Byte 5	SLOT TM (L)	Slot time, low byte
Byte 6	SLOT TM (H) (Bits 0–2)	Slot time, high part
	CDBBC (Bit 3)	Collision Detection by bit comparison
	RETRY NUM (Bits 4–7)	Number of transmission retries on collision
Byte 7	PRM (Bit 0)	Promiscuous mode
	BC DIS (Bit 1)	Broadcast disable
	MANCH/NRZ (Bit 2)	Manchester/NRZ or Manchester/NRZI encoding
	TON NCRS (Bit 3)	Transmit on NO CRS
	NCRC INS (Bit 4)	No CRC insertion
	CRC-16/CRC-32 (Bit 5)	CRC type
	BIT STF (Bit 6)	Bit stuffing
	PAD (Bit 7)	Padding
Byte 8	CRSF (Bit 0–2)	Carrier sense filter bits
	CRS SRC (Bit 3)	Carrier sense source
	CDTF (Bits 4–6)	Collision detect filter bits
	CDT SRC (Bit 7)	Collision detect source
Byte 9	MIN FRAME LEN	Minimum frame length

A Configure-Done interrupt is issued when the operation is done (unless ABORT was executed in the meantime).

8.1.4 MC-SETUP (03)

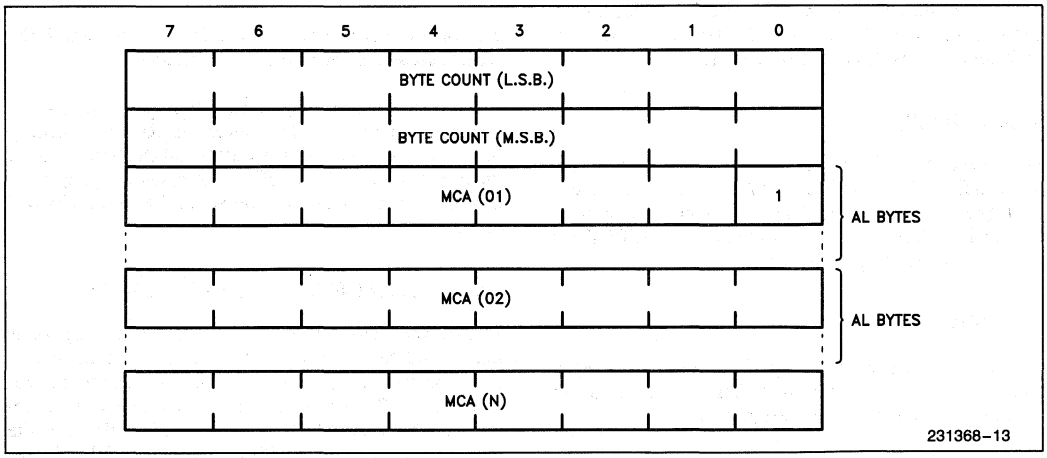
This operation clears and sets up a new 64 bit Multicast Address table. The parameter block may include a number of MC-addresses. Each contributes one bit to the Hash Table.

The first byte of the n-th MC-address in the parameter block begins at byte number n × AL + 1, where AL is the "Address Length" configuration parameter and n is a integer counting from zero.

NOTE:
The least significant bit in the first byte of EACH MC address must be a ONE.

The number of MC addresses that are set up during the operation is equal to Byte Count divided by AL, rounded down, i.e., if the last MC address is incomplete, it is ignored.

The format:



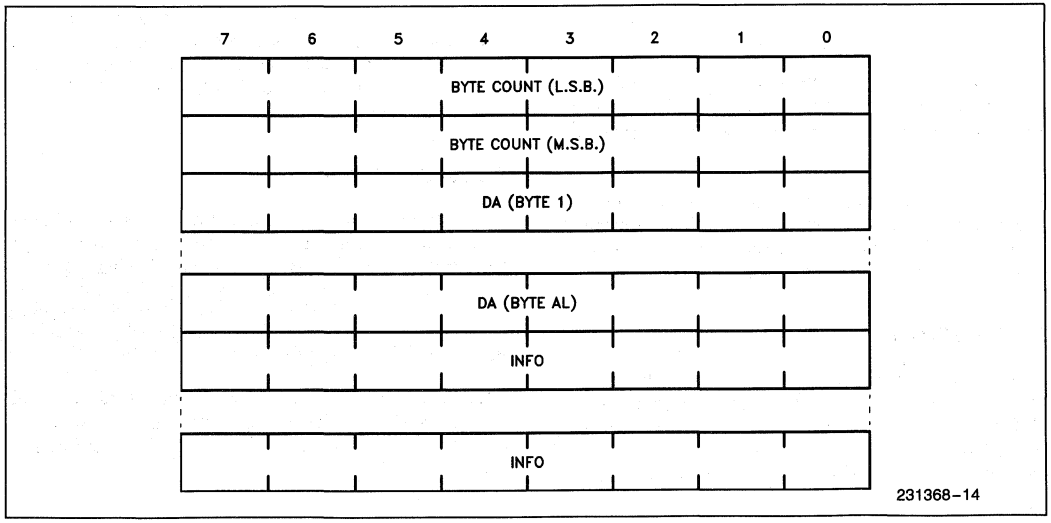
The MC-Setup-Done interrupt is issued after completing this operation (unless ABORT was executed in the meantime).

8.1.5 TRANSMIT (04)

This operation transmits one frame. The parameter block includes the following parameters:

- Destination Address— length determined by the "Address Length" configuration parameter.
- Source Address— **ONLY IF** the no source address insertion configuration parameter is one. The length is determined by "Address Length".
- Information— The length is determined by the Byte Count minus Destination Address Minus Source Address (if any) minus CRC (if any).
- CRC— **ONLY IF** the "No-CRC-insertion" configuration parameter is one. The length is determined by the "CRC-16/32" parameter.

To summarize, the typical transmit operation parameter block includes the Destination Address, and Information fields:



The transmit operation will either complete the execution or be aborted by a specific ABORT operation. A Transmit-Done or "Execution Aborted" interrupt is issued upon completion of this operation.

8.1.6 TDR (05)

This operation activates the Time Domain Reflectometer, which is a mechanism to detect open or short circuits (and their distance from the diagnosing station) on the link. There are no parameters. A TDR-Done interrupt is issued upon completion.

8.1.7 DUMP (06)

This operation causes dumping of a set of internal registers. There are no parameters. The 63 registers are dumped to memory on the assigned channel.

The DUMP operation will either complete the execution or be aborted by a specific ABORT operation. A DUMP- DONE or "Execution Aborted" interrupt is issued upon completion of this operation.

8.1.8 DIAGNOSE (07)

The Diagnose Command checks the 82588 timers hardware which includes:

* Exponential Backoff Random Number Generator.
* Exponential Backoff Timeout Counter.
* Slot Time Period Counter.
* Collision Number Counter.
* Exponential Backoff Shift Register.
* Exponential Backoff Mask Logic.
* Time Trigger Logic.

8.1.9 RETRANSMIT (12)

This operation is almost identical to the TRANSMIT operation. The only difference is the RETRANSMIT does not reset the retry counter and the exponential back-off mechanism.

The Retransmit operation will either complete the Execution or be aborted by a specific ABORT operation. A RETRANSMIT-DONE or "Execution Aborted" is issued upon completion of this operation.

8.1.10 ABORT (13)

This operation causes the attempt to abort the completion of an operation under execution. It is valid for: IA-SETUP, CONFIGURE, MC-SETUP, (RE)TRANSMIT and DUMP. It is ignored for any of the above, if the transfer of parameters was completed.

8.1.11 RCV-ENABLE (8)

This operation enables the reception. It is ignored if the receive state is already Ready or Active.

The 82588 receives only frames that pass the address filtering. The frame and its status are placed in memory (by the DMA Controller or CPU) and the byte count of the frame is placed in the Result Registers. The completion of frame reception causes an "End of Frame" event.

8.1.12 ASSIGN ALTERNATE BUFFER (9)

This operation assigns an alternate buffer if reception is ready (and the chip is configured to chaining mode). This operation may be issued when an execution is active and the Ex channel is used for transfer of parameters. In this case ASSIGN ALT BUF will cause the execution to be aborted (the same as ABORT operation). If the on-going execution is thus aborted, an "execution-aborted" interrupt occurs after the reception is complete. An executing command so aborted can then be restarted.

This operation has no parameters and no results. It is usually issued by the CPU following a "Request Alternate Buffer" event.

8.1.13 RCV-DISABLE (10)

This operation causes the reception to be disabled. If transfer of a frame to memory has already begun, then it causes a Reception Aborted event. It has no parameters and no results.

8.1.14 STOP-RCV (11)

This operation requests the disabling of reception. If reception is actually in progress, then it will continue until the end of the frame. If the reception is not in progress, then this command acts like the RCV-DISABLE command.

8.1.15 RESET (14)

This command resets the chip. It has the same effect as hardware reset. Note that the reset command does not reset bits of status registers 0, 1 and 2 other than the Int. field.

8.1.16 FIX PTR (1, 15)

This operation stops the cyclic progress of the read pointer of the STATUS Registers. The pointer is assigned to the register specified in the Pointer Field of the Command.

The channel bit must be set to "1". After this command every read from the status port accesses only the selected (PTR) register. Status 3 register updates stop when it is selected.

8.1.17 RLS PRT (0, 15)

This operation releases the assignment of the read pointer to a specific status register and restores the cyclic progress mode (any read advances the pointer to the next register).

The channel bit must be set to "0". This command with pointer = 0 is executed automatically on reset.

8.2 Illegal Commands

8.2.1 ILLEGAL EXECUTIONS

8.2.1.1 Non-parametric Executions (TDR, DIAGNOSE, ABORT)

TDR, Diagnose will be rejected if the Execution Machine is not idle.

Abort is rejected if issued when Execution Machine is not active, or a non-parametric execution is performed, or transfer of parameters has already been accomplished.

8.2.1.2 Parametric Executions

Will be rejected if one of the following exists:

1. Execution Machine is not idle.
2. The Exec Channel equals the RCV Channel and the RCV Machine is not idle, and it is either configured for Non-chaining Mode or it has at least one channel.
3. The RCV Machine is configured for chaining and it is either active or has been assigned both channels.

8.2.2 ILLEGAL RCV COMMANDS

8.2.2.1 Receive Enable

Will be rejected if one of the following exists:

1. Receive Machine is not idle.
2. The RCV Channel equals the Exec Channel and the Exec Machine already performs a parametric execution.

8.2.2.2 Assign Alternate Buffer

Will be rejected if one of the following exists:

1. The RCV Machine is not configured for Buffer Chaining.
2. The RCV Machine is in IDLE state.
3. An alternate buffer is already assigned.

8.2.2.3 Receive Disable & Stop Receive

Will be rejected if the RCV Machine is in IDLE state.

8.3 Event Statuses

This section specifies all statuses that indicate the occurrence of events This status is always accompanied by an interrupt bit.

8.3.1 IA - SETUP-DONE (01)

This event indicates the completion of an IA-SETUP operation.

8.3.2 CONFIGURE-DONE (02)

This event indicates the completion of a CONFIGURE operation.

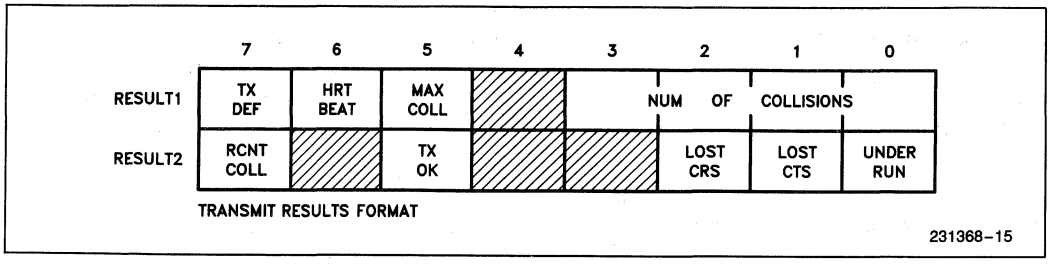
8.3.3 MC-SETUP-DONE (03)

This event indicates the completion of an MC-SETUP operation.

8.3.4 TRANSMIT-DONE (04)

This event indicates the completion of the TRANSMIT operation.

The two Result Registers provide information about errors and other diagnostic information. The format of the Result Registers is as follows:



TRANSMIT RESULTS FORMAT

231368-15

Where:

NUMBER OF COLL — Number of collisions the frame has experienced (modulo 16).

MAX COLL — Transmission failed due to a collision. The configured number of retries is exhausted.

HRT BEAT — Collision detect test passed after the previous frame.

TX DEF — Transmit deferred due to link activity.

UNDERRUN (*) — Indicates that the DMA did not keep up with transmission data rate.

LOST CTS (*) — Clear-to-Send lost during transmission.

LOST CRS (*) — Carrier Sense lost during transmission, or not set until end of preamble.

TX OK — Transmission executed okay.

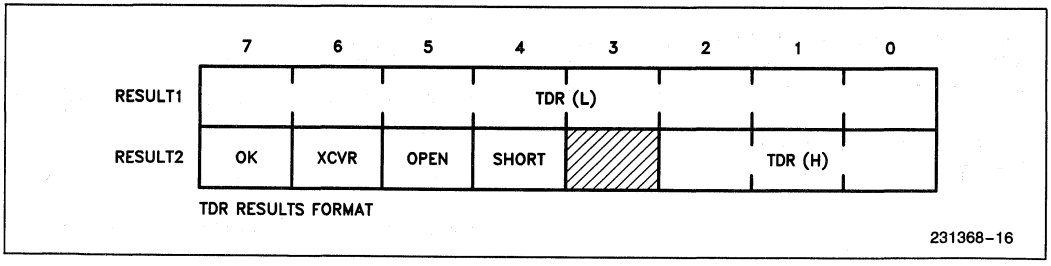
COLL (*) — The transmission of the last frame experienced a collision.

"Number of Collisions" provides redundant information. The setting of one of the bits marked (*) will cause TX-OK bit to be reset.

8.3.5 TDR-DONE (05)

This event indicates the completion of a TDR operation.

The Result Registers indicate whether there is a problem, which type and where.



TDR RESULTS FORMAT

231368-16

Where:

TDR : Twice the radial distance between the 82588 and the location of the problem (in bit times).

SHORT : Indicates there is a short circuit on the transmission line. (Carrier Sense Signal dropped)

OPEN : Indicates the transmission line is not properly terminated. (Collision Detect went active)

XCVR : Indicates a transceiver problem. (Carrier Sense was inactive for 2048 bit time period)

OK : Indicates that no problem was found.

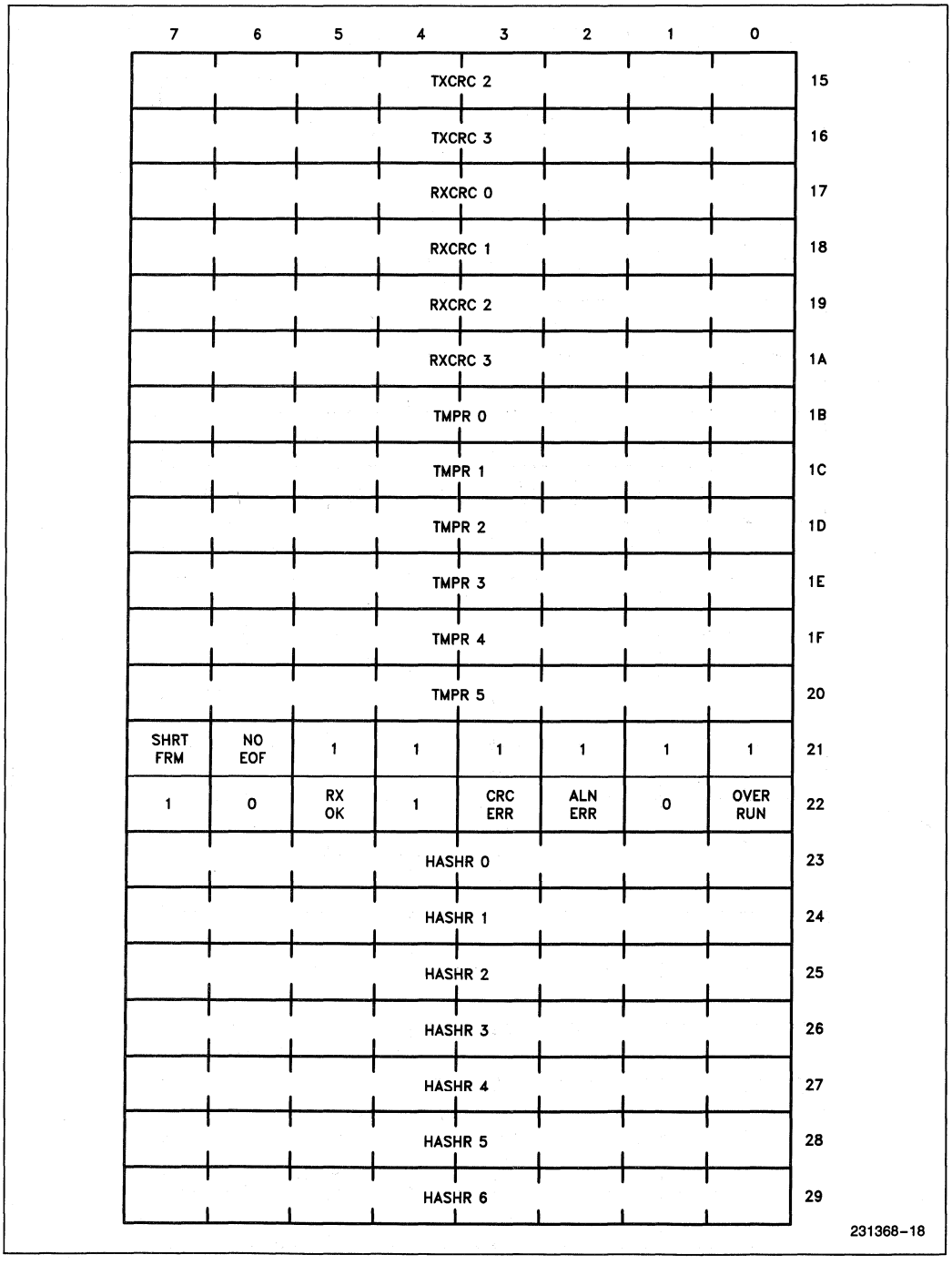
8.3.6 DUMP-DONE (06)

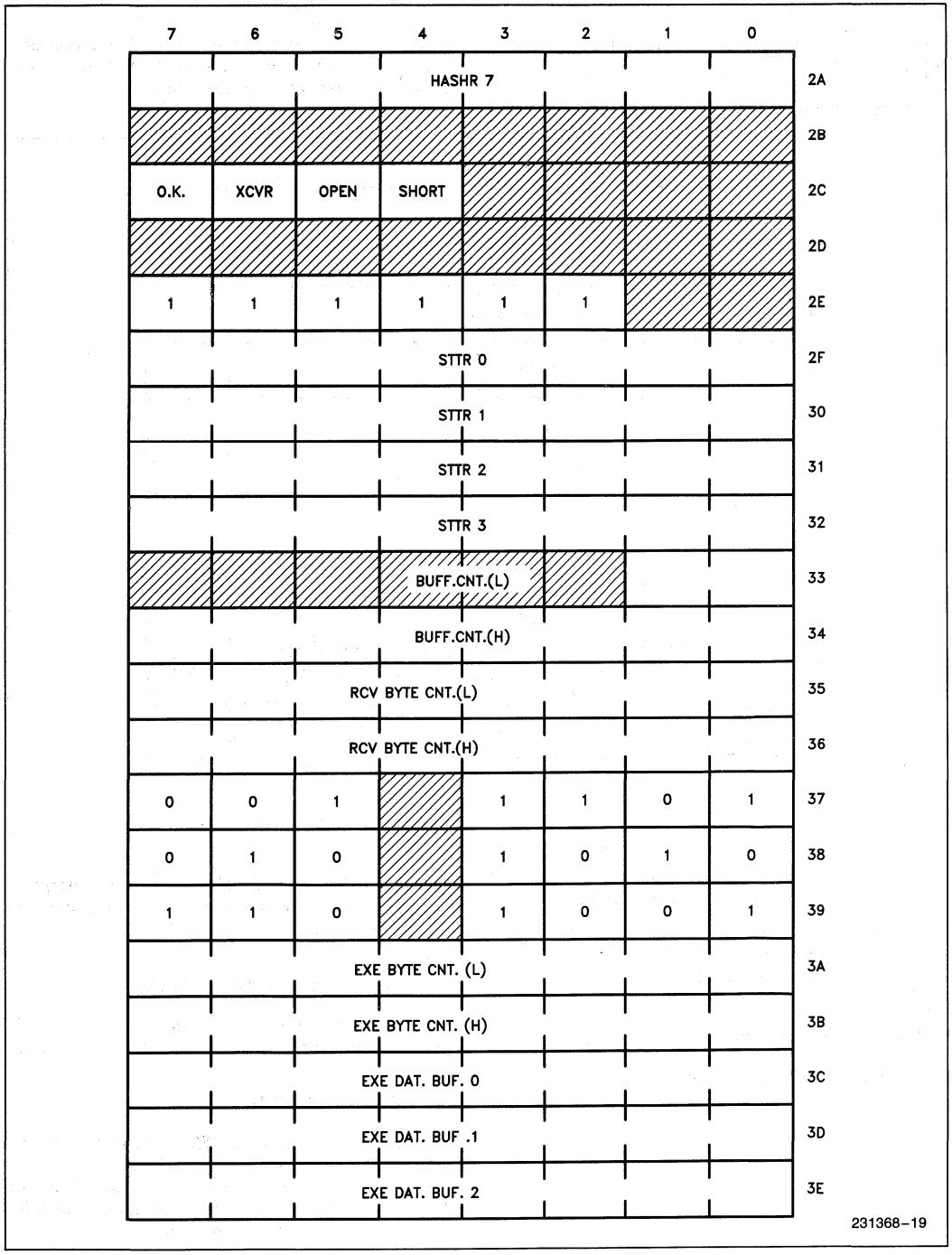
This event indicates that the DUMP operation is completed. The result registers are not updated. Contents of 63 internal registers are transferred to memory on the allocated channel.

The format of the dumped registers is as follows:

7	6	5	4	3	2	1	0	
CHNG	SERIAL MODE	SMPLG RATE	OSC RANGE	FIFO LIMIT				00
BUFFER LENGTH								01
EXT LP.BCK	INT LP.BCK	PREAM LEN		NO SRC ADD INS	ADD LEN			02
BOF METD	EXP PRIO			DIF.MAN /MAN	LIN PRIO			03
INTER FRAME SPACING								04
SLOT TIME (L)								05
	RETRY NUMBER			CDBBC	SLOT TIME (H)			06
PAD	BIT STUFF	CRC16	NCRC INS	TON NCRS	MAN /NRZ*	BC DIS	PRM	07
CDT SRC	CDTF			CRS SRC	CRSF			08
MINIMUM FRAME LENGTH								09
1	1	1	1	1	1	1	1	0A
IAR0								0B
IAR1								0C
IAR2								0D
IAR3								0E
IAR4								0F
IAR5								10
TX DEF	HRT BEAT	MAX COLL	0	NUM OF COLLISIONS				11
RCNT COLL	0	TX OK	0	0	LOST CRS	LOST CTS	UNDER RUN	12
TXCRC0								13
TXCRC1								14

231368-17


7 6 5 4 3 2 1 0
TXCRC 2 15
TXCRC 3 16
RXCRC 0 17
RXCRC 1 18
RXCRC 2 19
RXCRC 3 1A
TMPR 0 1B
TMPR 1 1C
TMPR 2 1D
TMPR 3 1E
TMPR 4 1F
TMPR 5 20
SHRT FRM | NO EOF | 1 | 1 | 1 | 1 | 1 | 1 21
1 | 0 | RX OK | 1 | CRC ERR | ALN ERR | 0 | OVER RUN 22
HASHR 0 23
HASHR 1 24
HASHR 2 25
HASHR 3 26
HASHR 4 27
HASHR 5 28
HASHR 6 29
231368-18


7 6 5 4 3 2 1 0
HASHR 7
2A
2B
O.K. XCVR OPEN SHORT
2C
2D
1 1 1 1 1 1
2E
STTR 0
2F
STTR 1
30
STTR 2
31
STTR 3
32
BUFF.CNT.(L)
33
BUFF.CNT.(H)
34
RCV BYTE CNT.(L)
35
RCV BYTE CNT.(H)
36
0 0 1 1 1 0 1
37
0 1 0 1 0 1 0
38
1 1 0 1 0 0 1
39
EXE BYTE CNT. (L)
3A
EXE BYTE CNT. (H)
3B
EXE DAT. BUF. 0
3C
EXE DAT. BUF .1
3D
EXE DAT. BUF. 2
3E
231368-19

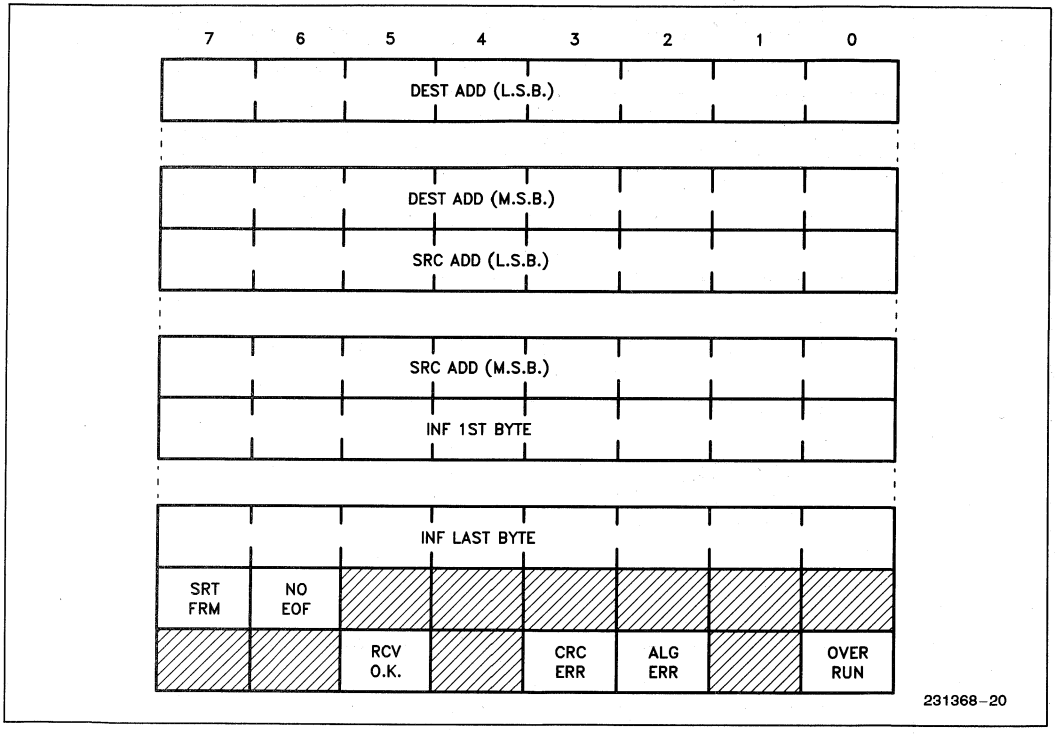
8.3.7 DIAGNOSE-PASSED (07)

This event indicates that the self check procedure was completed successfully.

8.3.8 END-OF-FRAME (8)

This event indicates the completion of reception of a frame, and depositing of it in memory. The result registers hold the byte count of the frame.

The format of the received frame is as follows:



Where:

NO EOF	No EOF flag (in bitstuffing only).
SRT FRM	Received frame is shorter than Minimum Frame Length parameter.
OVER RUN	The DMA did not keep up with incoming bits.
ALG ERR	Alignment error.
CRC ERR	CRC error.
RCV OK	Frame received OK.

RCV OK is reset if any of the others is set.

8.3.9 REQUEST ALTERNATE BUFFER (9)

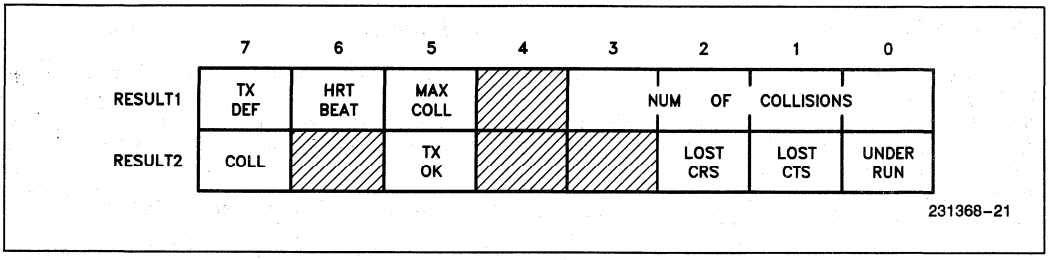
This event is issued in Chaining mode, when reception is enabled, a frame arrives and an additional alternate buffer is not yet assigned.

8.3.10 RECEPTION ABORTED (10)

This event is issued as a result of a RCV-DISABLE operation that causes part of a frame to be discarded.

8.3.11 RETRANSMIT-DONE (12)

This event indicates that the RETRANSMIT operation was completed. It DOES update the result registers. The format of the result registers is identical to the TRANSMIT-DONE, except that the "number of collision" provides essential information.



Where:

NUMBER OF COLL : Number of collisions the frame has experienced (modulo 16)

MAX COLL : Transmission failed due to a collision. The configured number of retries is exhausted.

HRT BEAT : Collision detect test passed after the previous frame.

TX DEF : Transmit deferred due to line activity.

UNDERRUN (*) : Indicates that the DMA did not keep up with bit transmission.

LOST CTS (*) : Clear-to-send lost during transmission, or not set until end of preamble.

LOST CRS (*) : Carrier sense lost during transmission, or not set until end of preamble.

TX OK : Transmission executed OK.

COLL : The transmission of the last frame experienced a collision.

The setting of the bits marked with (*) will cause the TX-OK bit to be reset.

8.3.12 EXECUTION-ABORTED (13)

This event indicates that the execution of the last operation was aborted. The reason for the abortion could be either an ABORT or an ASSIGN-ALT-BUF command.

8.3.13 DIAGNOSE-FAILED (15)

This event indicates that the self check procedure was completed unsuccessfully.

9.0 SYSTEM INTERFACE

The data bus is 8 bits wide and there is no address bus. The 82588 can interface to wider buses, but only 8 bits will be used. The 82588 is a slave on the bus, i.e., it does not perform memory transfers by itself. The transfers are performed by the CPU or the DMA controller. The 82588 has a very general and simple interface allowing it to operate with a variety of processors and DMA controllers.

There are two kinds of transfers over the bus: Command/Status and Data transfers. The command/status transfers are always performed by the CPU (with its own initiative). The data transfers are requested by the 82588 (using the DMA request lines). They are typically performed by a DMA controller. In relatively slow systems the CPU might also perform the data transfers. In that case, the requests from the 82588 will serve as interrupts to the CPU. This mode of operation depends on the serial data rate and on the dedication of the CPU.

The system interface performs command/status transfers, data transfers, and interrupts.

9.1 Command/Status Transfers

The CPU controls the 82588 by writing into the Command Register and reading from the Status Register. The CPU writes a command by activating the CS input of the 82588, putting the command onto the bus, and activating the WR input of the 82588. In order to read status from the 82588, the CPU activates the CS line and then activates the RD line. The 82588 responds by putting the status onto the bus (during the time that RD is active). There is no address line by which the CPU can select a specific register, within the 82588. The register is selected by the internal 2 bit pointer.

9.2 Data Transfer

Data transfers are controlled by two pairs of REQUEST/ACKNOWLEDGE lines (referred to as two CHANNELS): DMA Request lines (DRQ0 and DRQ1) and DMA Acknowledge lines (DACK0 AND DACK1). Frames and parameters are transferred via the two channels (0 and 1) to or from memory. Commands allocate a specific channel for each operation.

In order to request a transfer from memory, the 82588 activates a DRQ signal on a channel that has been set up for transfer from memory. In response, the DMA controller activates the respective DACK and performs the data transfer. Data is transferred on the bus and written into the 82588 on the rising edge of the WR signal, which is activated by the DMA controller.

In order to request a transfer to memory, the 82588 activates a DRQ signal on a channel that has been previously set up for transfer to memory. In response, the DMA controller activates the respective DACK and performs the data transfer. The 82588 outputs data onto the bus during the time that the RD line is activated by the DMA controller.

9.3 Interrupt

The 82588 reports on completion of an event (see section "Controlling the 82588") by updating a set of status registers and raising the INTERRUPT signal (assuming this signal is initially low). The interrupt is cleared by the command from the CPU where the INTERRUPT ACKNOWLEDGE bit is set. Note that it is the responsibility of the CPU to clear interrupts in order to prevent deadlocks.

9.4 Performance Considerations

This section elaborates on data transfer limitations, bus utilization and interframe spacing considerations. Note, the analysis is only for "steady state," and therefore, should only be regarded as a rough guideline.

9.4.1 TRANSFER RATE LIMITATIONS

The system interface of the 82588 must be able to keep up with the serial link. Assuming f_s is the serial data bit rate, then $f_s/8$ is the serial data rate in bytes/sec. The data rate of the system interface depends on the system clock (f_p) and the number of clocks required to transfer one byte (n).

In order for the interface to keep up with the serial link, the following must hold:

$$\frac{f_p \times 8}{f_s \times n} \text{ is greater than } 1$$

The 82588 uses two on chip 16 byte FIFOs to increase burstiness. A configurable FIFO limit is associated with the FIFOs. The receive FIFO (that is filled by the Receiver Machine) requests the bus only when it is filled above the limit's 2's complement. It transfers bytes in a burst till the FIFO is empty. The transmit FIFO (that is emptied by the Transmit Machine) requests the bus only when it is emptied till the limit. It then transfers a burst of bytes from the system bus till it is full.

Pin Name			Function
$\overline{CS}$*	$\overline{RD}$	$\overline{WR}$	
1	x	x	No transfer to/from Command/Status
0	1	1	
0	0	0	Illegal
0	0	1	Read from status register
0	1	0	Write to Command register
$\overline{DACK0}$[$\overline{DACK1}$]*	$\overline{RD}$	$\overline{WR}$	
1	X	X	No DMA transfer
0	1	1	
0	0	0	Illegal
0	0	1	Data Read from DMA channel 0 [or 1]
0	1	0	Data Write to DMA channel 0 [or 1]

*Only one of $\overline{CS}$, $\overline{DACK0}$ and $\overline{DACK1}$ may be active at any time.

The 82588 must take into account that it takes a number of clocks to acquire the bus (in the range between N_{Amin} to NA_{max}). In order to prevent underruns (or overruns) the maximum acquisition time must be less than the time to empty the FIFO from the limit.

$$\frac{N_{Amax}}{f_p} \text{ is less than } \frac{LIMIT}{f_s/8}E$$

i.e.,

$$\text{LIMIT is greater than } \frac{f_s \times N_{Amax}}{f_p \times 8}$$

In order to assure an internal of N_I clocks that the CPU can have the bus between bursts, the following must hold: the time to empty the FIFO till the limit, plus the minimum acquisition time, must be greater than the time between bursts:

$$\frac{16\text{-LIMIT}}{f_s/8} + \frac{N_{Amin}}{f_p} \text{ is greater than } \frac{N_I}{f_p}$$

i.e.,

$$\text{LIMIT is less than } 16 - \frac{f_s \times (N_I - N_A\text{min})}{f_p \times 8}$$

9.4.2 BUS UTILIZATION

We define bus utilization as the fraction of time that the bus is not busy servicing the 82588 (including bus acquisition). The following analysis, again, is based on "steady state" and therefore, should be regarded as a guideline only.

The complement of the CPU utilization is the time that the CPU is busy filling the FIFO (t_F), divided by the cycle time from FIFO full till its is full again:

$$\text{CPUUTIL} = 1 - \frac{t_F}{\frac{16 - \text{LIMIT}}{f_s/8} + t_F + t_A}$$

t_A = acquisition time

To calculate t_F we use the fact that, during one cycle the number of bytes entered into the FIFO equals the number removed:

$$16 - \text{LIMIT} + t_A \times f_s/8 + t_F \times f_s/8 = t_F \times f_p/n$$

$$\text{i.e., } t_F = \frac{16 - \text{LIMIT} + t_A \times f_s/8}{f_p/n - f_s/8}$$

Therefore, by substituting t_F into CPUUTIL:

$$\text{CPUUTIL} = 1 - \frac{f_s \times n}{f_p \times 8}$$

This same expression could be derived directly, as it represents the complement of the ratio of the time to transfer a byte to/from memory, over the time to transfer a byte from/to the link. Note, this expression applies only to the period of transferring, no taking into account the effects between buffers or frames.

9.4.3 BUFFER SIZE CONSIDERATIONS

The 82588 provides support for chaining buffers in memory during reception of frames. The 82588 compares the number of bytes transferred to memory to a programmable buffer limit register. When the count equals the limit and the alternate channel is setup, then the 82588 switches to the alternate DMA channel, resets the counter and issues a "Request Alternate Buffer" interrupt. The CPU interrupt service routine takes care of updating the DMA Controller parameters, issues an "Alternate Buffer Assigned" and acknowledges the interrupt. The minimum buffer size depends on the maximum CPU SERVICE TIME which is composed of the following: the time from issuing the interrupt till the CPU services it, the time the CPU needs to sort out the reason for the interrupt, and the time it takes the CPU to respond. This time depends on the system (not on the 82588), but note, that the data transfer may contend for the bus. Note, activity on the EX channel changes the buffer size considerations.

There are several constraints on the size of the buffer (not taking into account the effects of the case of a partially filled last buffer). The advantage of using short buffers is that, assuming frames of random length, half a buffer will be fragmented (per frame). The smaller the buffer the less memory is wasted. However, the buffer has a minimum size that is determined by the fact that, in order to prevent overruns, the CPU service time must be shorter than the time to fill a buffer.

The advantages of longer buffers is less CPU overhead for buffer switching and memory overhead for storing buffer descriptors. However, the maximum buffer length is 1K.

9.4.4 INTER FRAME SPACING

Inter frame spacing (IFS) is a configuration parameter that is enforced by the link management. The minimum IFS is intended to assure recovery time between completion of transmit or receive to receive, and between transmit to retransmit.

The 82588 supports pipelining between received frames. This feature relaxes the constraints on the IFS. In consecutive mode (i.e., not chaining) the CPU service time, following a frame, must be shorter than the SUM of the IFS and the length of the frame.

Therefore, if the CPU service time is short enough, and the minimum frame long enough, the IFS may be as short as 12 bit times (which is the minimum configurable).

In chaining mode the constraint is more severe, and the IFS time must be longer than the CPU service time. This is due to worst case where the last byte of the frame is placed in the first byte of the buffer.

The 82588 does not perform the retransmission by itself. It only notifies the CPU about a collision, stops transmission and disables transmission for backoff delay, which is in the minimum case equal to IFS. It is desirable (but not mandatory) that the station be able to retransmit within the deferring time (IFS). In order to assure being ready for retransmission, within the deferring time, the IFS must be longer than the CPU service time.

10.0 80188 BASED SYSTEM

Figure 12 shows a high performance, high integration configuration of the 82588 with the 80188 in a typical iAPX188-based microcomputer. The 80188 controls the 82588, as well as providing DMA control services for data transfer, using its "on chip" two channel DMA controller.

10.1 Link Interface

The Serial Interface Mode configuration parameter selects either a highly integrated Direct Link Interface (High Integration Mode) or a highly flexible Transceiver Interface (High Speed Mode).

10.2 Application

In the High Integration mode it is possible to connect the 82588 on a very short "Wired OR" link, on a longer twisted pair cable, or a broadband connection.

TWISTED PAIR CONNECTION

The link consists of a twisted pair that interconnects the 82588. The transmit data pin is connected via a driver and the receive data pin is connected via a buffer. The twisted pair must be properly terminated to prevent reflections.

In the minimum configuration, TXD and RXD are connected to the twisted pair and CTS is grounded. The 82588 may control the driver with the RTS pin. It is also possible to use external circuitry for performing collision detection and feeding it to the 82588 through the CDT pin.

BROADBAND CONNECTION

The 82588 supports data communications over a broadband link in both its modes. Proper MODEM interface should be provided. Collision Detection by Bit Comparison, in High Integration Mode, fits transmission over broadband links.

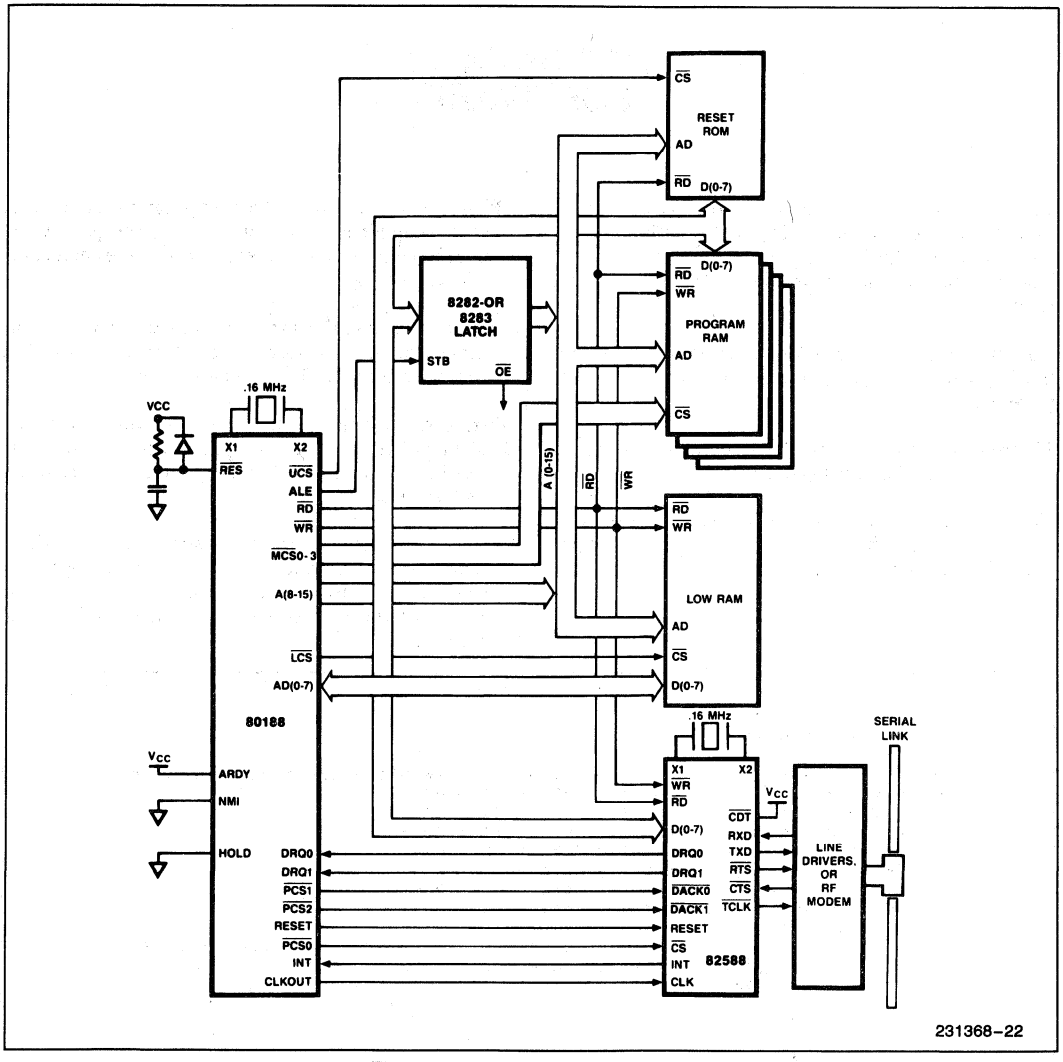


Figure 12. 80188 Based System

APPENDIX A: SOFTWARE

This section covers the basic software procedures to drive the 82588. It is written for an 80186-82588 system where both the DMA channels of the 80186 are used. The on-chip interrupt controller of the 80186 is also used.

This is not a complete software driver for the 82588, but it gives all the procedures needed to write a complete driver.

Figure A-1 shows the way of assigning the port definitions for an 80186 based system.

Figure A-2 shows how to start the various operations possible with the chip. This is not a procedure, but just examples of calls to operations.

Figure A-3 shows how the configuration parameters, individual address and multicast address parameters can be filled in the buffers before issuing the commands to program these.

Figure A-4 shows the different commands for transmit, receive, configure, etc. are issued. They also show what needs to be set up before issuing the commands.

Figure A-5 shows how the DMA controller is loaded and initialized for data and parameter transfer. This procedure is used by all other procedures requiring DMA service.

Figure A-6 shows an interrupt service routine which handles interrupts resulting due to various possible events. It also shows how to extract receive status from a received frame and retransmit sequence required after a collision.

```
/* port definitions */

  declare pba       literally '400h';     /* 80186 i/o base address */
  declare pcs0      literally 'pba + (0 * 80h)';
  declare pcs1      literally 'pba + (1 * 80h)';
  declare pcs2      literally 'pba + (2 * 80h)';
  declare pcs3      literally 'pba + (3 * 80h)';
  declare pcs4      literally 'pba + (4 * 80h)';
  declare pcs5      literally 'pba + (5 * 80h)';
  declare pcs6      literally 'pba + (6 * 80h)';

  declare cs_588    literally 'pcs1 + 0',  /* 82588 command/status */
          ch_a_588  literally 'pcs2 + 0',  /* 82588 DMA channel A */
          ch_b_588  literally 'pcs3 + 0';  /* 82588 DMA channel B */
```

231368-24

Figure A-1. Port Definition for 80186 Based System

```
/* typical calls to execute 82588 commands */

  call ia_set(1);                                  /* 1 */
  call config(1);                                  /* 2 */
  call multicast(1);                               /* 3 */
  call transmit(1,100,@tx_buff_588);               /* 4 */
  call tdr;                                        /* 5 */
  call dump_588(0);                                /* 6 */
  call diagnose;                                   /* 7 */
  call rcv_enable(0,@buffer_588(0) rx(0));         /* 8 */
  call rcv_stop;                                   /* B */
  call retransmit;                                 /* C */
  call abort(1);                                   /* D */
  call reset_588;                                  /* E */

/*-------------------------------------------------------------------------*/
```

231368-25

Figure A-2. An Example of Executing Commands

```
/* 82588 init */

init_588:  procedure;

  config_588(00) = 10;            /* to configure all parameters */
  config_588(01) = 00;
  config_588(02) = 00101000b;     /* mode 0, 16 MHz clock, 1 MB/s */
  config_588(03) = buff_len/4;    /* Receive Buffer length = 256 bytes */
  config_588(04) = 10100110b;     /* Ext. loopback, address len = 6, Preamble = 8 */
  config_588(05) = 00000000b;     /* Differential Manchester = off  */
  config_588(06) = 48;            /* IFS = 48 TCLK */
  config_588(07) = 70h;           /* Slot time = 70h TCLK */
  config_588(08) = 11111000b;     /* Col Det By Bit Comp. = on  */
  config_588(09) = 00000100b;     /* Manchester encoded, bit-stuffing = off */
  config_588(10) = 10001100b;     /* Internal CRS and CDT, CRSF = 4 */
  config_588(11) = 06;

  ia_set_buff_588(0) = 6;         /* 6 byte individual address */
  ia_set_buff_588(1) = 0;
  ia_set_buff_588(2) = 000h;
  ia_set_buff_588(3) = 011h;
  ia_set_buff_588(4) = 022h;
  ia_set_buff_588(5) = 044h;
  ia_set_buff_588(6) = 088h;
  ia_set_buff_588(7) = 0ffh;

  multicast_buff_588(00) = 12;    /* Two 6 byte multicast addresses */
  multicast_buff_588(01) = 00h;
  multicast_buff_588(02) = 11h;
  multicast_buff_588(03) = 12h;
  multicast_buff_588(04) = 13h;
  multicast_buff_588(05) = 14h;
  multicast_buff_588(06) = 15h;
  multicast_buff_588(07) = 16h;
  multicast_buff_588(08) = 21h;
  multicast_buff_588(09) = 22h;
  multicast_buff_588(10) = 23h;
  multicast_buff_588(11) = 24h;
  multicast_buff_588(12) = 25h;
  multicast_buff_588(13) = 26h;

  call reset_588;

end init_588;
```

231368-26

Figure A-3. Defining Configuration, IA and Multicast Address Parameters

```
/*-------------------------------------------------------------------------*/

nop_588: procedure;                    /* command - 00 */

  output (cs_588) = 00,   /* NOP */
  return;

end nop_588;

/*-------------------------------------------------------------------------*/

ia_set: procedure(channel);            /* command - 01 */

  declare channel byte;

  call dma_load(channel,1,8,@ia_set_buff_588);
  output (cs_588) = 1 or shl (channel,4);   /* ia_set */
  return;

end ia_set;

/*-------------------------------------------------------------------------*/

config: procedure(channel);            /* command - 02 */

  declare channel byte;

  call dma_load(channel,1,12,@config_588);
  output (cs_588) = 2 or shl (channel,4);   /* configure */
  return;

end config;

/*-------------------------------------------------------------------------*/

multicast: procedure(channel);         /* command - 03 */

  declare channel byte;

  call dma_load(channel,1,14,@multicast_buff_588);
  output (cs_588) = 3 or shl (channel,4);   /* multicast */
  return;

end multicast;


/*-------------------------------------------------------------------------*/

                                       /* command - 04 */

transmit: procedure(channel,buffer_len,buffer_pointer);

  declare channel byte;
  declare buffer_len word;
  declare buffer_pointer pointer;

  tx_buffer_588(00) = buffer_len mod 256,
  tx_buffer_588(01) = buffer_len / 256;

  tx_buff_ptr = buffer_pointer;   /* temporary storage */
  tx_channel = channel;
  tx_buffer_len = 2048;          /* max. frame size */

  call dma_load(tx_channel,1,tx_buffer_len,tx_buff_ptr);
  output (cs_588) = 4 or shl (tx_channel,4);   /* transmit */
  return;

end transmit;
```

231368-27

Figure A-4a. Setup and Execution of Commands

```
/*-------------------------------------------------------------------------*/

tdr: procedure;                         /* command - 05 */

  output (cs_588) = 5 ;   /* tdr */
  return;

end tdr;

/*-------------------------------------------------------------------------*/

dump_588: procedure(channel);           /* command - 06 */

  declare channel byte;

  call dma_load(channel,0,150,@dump_buff_588);
  output (cs_588) = 6 or shl (channel,4);   /* dump */
  return;

end dump_588;



/*-------------------------------------------------------------------------*/

diagnose: procedure;                    /* command - 07 */

  output (cs_588) = 7 ;   /* diagnose */
  return;

end diagnose;

/*-------------------------------------------------------------------------*/

                                        /* command - 08 */

rcv_enable: procedure(channel,buffer_ptr);

  declare channel byte;
  declare buffer_ptr pointer;

  call dma_load(channel,0,2048,buffer_ptr);
  output(cs_588)= 8 or shl (channel,4);
  return;

end rcv_enable;

/*-------------------------------------------------------------------------*/

allocate_buffer: procedure(buffer_ptr);   /* command - 09 */

  declare buffer_ptr pointer;
  declare new_channel byte;

  new_channel = not(rol(status_588(3),1)) and 00000001b;
  call dma_load(new_channel,0,buff_len,buffer_ptr);
  output(cs_588)= 9;
  return;

end allocate_buffer;
```

231368-28

Figure A-4b. Setup and Execution of Commands

```
rcv_disable: procedure;              /* command - 10 */

  output(cs_588)= 10;
  return;

end rcv_disable;

/*-------------------------------------------------------------------------*/

rcv_stop: procedure;                 /* command - 11 */

  output(cs_588)= 11;
  return;

end rcv_stop;

/*-------------------------------------------------------------------------*/

retransmit: procedure;               /* command - 12 */

  /* Parameters for this command are taken from the temporary
     storage used during the last Transmit command */

  call dma_load(tx_channel,1,tx_buffer_len,tx_buff_ptr);
  output (cs_588) = 12 or shl (tx_channel,4);   /* retransmit */
  return;

end retransmit;

/*-------------------------------------------------------------------------*/

abort: procedure(channel);           /* command - 13 */

  declare channel byte;

  output(cs_588)= 13 or shl (channel,4);
  return;

end abort;


/*-------------------------------------------------------------------------*/

reset_588: procedure;                /* command - 14 */

  output(cs_588) = 14;

  call config(1);              /* configure on reset */

  return;

end reset_588;

/*-------------------------------------------------------------------------*/


read_status_588: procedure;          /* command - 15 */

  output (cs_588) = 15;        /* release pointer, initial = 00 */

  status_588(0) = input (cs_588); /* refresh status register image */
  status_588(1) = input (cs_588);   /* in memory.
  status_588(2) = input (cs_588);
  status_588(3) = input (cs_588);
  return;

end read_status_588;
```

231368-29

Figure A-4c. Setup and Execution of Commands

```
dma_load: procedure(channel,direction,trans_len,buff_ptr) reentrant;

/* To load and start the  80186 DMA controller for the desired operation */

  declare dma_rx_mode   literally '1010001001000000b'; /* rx channel */
        /* src=IO, dest=M(inc), sync=src, TC, noint, priority, byte */

  declare dma_tx_mode   literally '0001011010000000b'; /* tx channel */
        /* src=M(inc), dest=IO, sync=dest, TC, noint, noprior, byte */

  declare channel byte;       /* channel # */
  declare direction byte;     /* 0 = rx, 588 -> mem; 1 = tx, mem -> 588 */
  declare trans_len word;     /* byte count */
  declare buff_ptr pointer;   /* buffer pointer in seg:offset form */

  declare  buff_ptr_20bit dword;
  declare ptr1 pointer;
  declare (wrd based ptr1)(2) word;

  ptr1 = @buff_ptr;           /* convert buff_ptr to 20bit buff_ptr */
  buff_ptr_20bit = shl((buff_ptr_20bit := wrd(1)),4) + wrd(0);

  do case channel and 00000001b;
    do case direction and 00000001b;
      do;                     /* channel 0 , 588 to memory */
      outword(dma_0_dpl) = low (buff_ptr_20bit);
      outword(dma_0_dph) = high(buff_ptr_20bit);
      outword(dma_0_spl) = ch_a_588;
      outword(dma_0_sph) = 0;
      outword(dma_0_tc)  = trans_len;
      outword(dma_0_cw)  = dma_rx_mode or 0006h;  /* start DMA channel 0 */
      end;

      do;                     /* channel 0 , memory to 588 */
      outword(dma_0_dpl) = ch_a_588;
      outword(dma_0_dph) = 0;
      outword(dma_0_spl) = low (buff_ptr_20bit);
      outword(dma_0_sph) = high(buff_ptr_20bit);
      outword(dma_0_tc)  = trans_len;
      outword(dma_0_cw)  = dma_tx_mode or 0006h;  /* start DMA channel 0 */
      end;
    end;

    do case direction and 00000001b;
      do;                     /* channel 1 , 588 to memory */
      outword(dma_1_dpl) = low (buff_ptr_20bit);
      outword(dma_1_dph) = high(buff_ptr_20bit);
      outword(dma_1_spl) = ch_b_588;
      outword(dma_1_sph) = 0;
      outword(dma_1_tc)  = trans_len;
      outword(dma_1_cw)  = dma_rx_mode or 0006h;  /* start DMA channel 1 */
      end;

      do;                     /* channel 1 , memory to 588 */
      outword(dma_1_dpl) = ch_b_588;
      outword(dma_1_dph) = 0;
      outword(dma_1_spl) = low (buff_ptr_20bit);
      outword(dma_1_sph) = high(buff_ptr_20bit);
      outword(dma_1_tc)  = trans_len;
      outword(dma_1_cw)  = dma_tx_mode or 0006h;  /* start DMA channel 1 */
      end;
    end;
  end;
  return;

end dma_load;
```

231368-30

Figure A-5. Loading and Starting the 80186 DMA Controller

```
intr_588: procedure interrupt 13;

  declare event byte;

  call read_status_588;
  event = status_588(0) and 00001111b);

  do case event;

    event_00:  ;
    event_01:  outword(dma_1_cw)  = (dma_tx_mode or 0004h);  /* stop DMA channel 1 */
    event_02:  outword(dma_1_cw)  = (dma_tx_mode or 0004h);  /* stop DMA channel 1 */
    event_03:  outword(dma_1_cw)  = (dma_tx_mode or 0004h);  /* stop DMA channel 1 */
    event_04:  do;                                 /* transmit done */
                 outword(dma_1_cw)  = (dma_tx_mode or 0004h); /* stop DMA channel 1 */
                 if (status_588(2) and 10000000b) <> 0    /* collision */
                 then if (status_588(1) and 00100000b) = 0  /* max collision */
                   then intr_588_flag = 'X';          /* retransmit */
               end;
    event_05:  ;
    event_06:  outword(dma_0_cw)  = (dma_rx_mode or 0004h);  /* stop DMA channel 0 */
    event_07:  outword(dma_0_cw)  = (dma_rx_mode or 0004h);  /* stop DMA channel 0 */
    event_08:  do;
                 call rcv_disable;

              /* extract the rx_status bytes 0,1 from the received frame */
              /* frame length is in status_588(0 & 1) */
              /* multiple buffer scheme is assumed */

                 rx_buff_off = shl(double(status_588(2)),8)
                             +  double(status_588(1)) - 2;
                 rx_buff_no = low(rx_buff_off / buff_len);
                 rx_buff_off = rx_buff_off mod buff_len;
                 /* status 0 */
                 rx_stat(0) = read_byte(@buffer_588(rx_buff_no).rx(rx_buff_off));
                 rx_buff_off = rx_buff_off + 1;
                 if rx_buff_off = buff_len  /* status across buff boundaries */
                 then do;
                   rx_buff_off = 0;
                   rx_buff_no = rx_buff_no + 1;
                     end;
                 /* status 1 */
                 rx_stat(1) = read_byte(@buffer_588(rx_buff_no).rx(rx_buff_off));

                 call rcv_enable(0,@buffer_588(0).rx(0));

               end;
    event_09:  call allocate_buffer(new_buffer);
               /* new_buffer is a procedure returning the pointer to new buffer */
    event_10:  outword(dma_0_cw)  = (dma_rx_mode or 0004h);  /* stop DMA channel 0 */
    event_11:  ;
    event_12:  do;  /* re-transmit done */
                 outword(dma_1_cw)  = (dma_tx_mode or 0004h);  /* stop DMA channel 1 */
                 if (status_588(2) and 10000000b) <> 0    /* collision */
                 then if (status_588(1) and 00100000b) = 0  /* no max collision */
                   then intr_588_flag = 'X';  /* retransmit */
               end;
    event_13:  do;  /* execution aborted */
                 outword(dma_1_cw)  = (dma_tx_mode or 0004h);  /* stop DMA channel 1 */
                 intr_588_flag = 'X';
               end;
    event_14:  ;
    event_15:  ;

  end;

  outword (eoir_186)  = 8000h;  /* non specific EOI to 80186 */

  output (cs_588) = 10000000b;  /* intack, */

  return;

end intr_588;
```

231368-31

Figure A-6. Interrupt Service Routine

Implementing StarLAN with the Intel 82588 Controller

5

intel®

APPLICATION NOTE

AP-236

November 1986

Implementing StarLAN with the Intel 82588

ADI GOLBERT
DATA COMMUNICATIONS OPERATION

SHARAD GANDHI
FIELD APPLICATIONS-EUROPE

Order Number: 231422-003

1.0 INTRODUCTION

Personal computers have become the most prolific workstation in the office, serving a wide range of needs such as word processing, spreadsheets, and data bases. The need to interconnect PCs in a local environment has clearly emerged, for purposes such as the sharing of file, print, and communication servers; downline loading of files and application programs; electronic mail; etc. Proliferation of the PC makes it the workstation of choice for accessing the corporate mainframe/s; this function can be performed much more efficiently and economically when clusters of PCs are already interconnected through Local Area Networks (LANs). According to market surveys, the installed base of PCs in business environments reached about 10 million units year-end '85, with only a small fraction connected via LANs. The installed base is expected to double by 1990. There is clearly a great need for locally interconnecting these machines; furthermore, end users expect interconnectability across vendors. Thus, there is an urgent need for industry standards to promote cost effective PC LANs.

A large number of proprietary PC LANs have become available for the office environment over the past several years. Many of these suffer from high installed cost, technical deficiencies, non-conformance to industry standards, and general lack of industry backing. StarLAN, in Intel's opinion, is one of the few networks which will emerge as a standard. It utilizes a proven network access method, it is implemented with proven VLSI components; it is cost effective, easily installable and reconfigurable; it is technically competent; and it enjoys the backing of a large cross section of the industry which is collaborating to develop a standard (IEEE 802.3, type 1BASE5).

1.1 StarLAN

StarLAN is a 1 Mb/s network based on the CSMA/CD access method (Carrier Sense, Multiple Access with Collision Detection). It works over standard, unshielded, twisted pair telephone wiring. Typically, the wiring connects each desk to a wiring closet in a star topology (from which the IEEE Task Force working on the standard derived the name StarLAN in 1984). In fact, telephone and StarLAN wiring can coexist in the same twisted pair bundle connecting a desk to the wiring closet. Abundant quantities of unused phone wiring exist in most office environments, particularly in the U.S. The StarLAN concept of wiring and networking concepts was originated by AT&T Information Systems.

1.2 The 82588

The 82588 is a single-chip LAN controller designed for CSMA/CD networks. It integrates in one chip all functions needed for such networks. Besides inplementing the standard CSMA/CD functions like framing, deferring, backing off and retrying on collisions, transmitting and receiving frames, it performs data encoding and decoding in Manshester or NRZI format, carrier sensing and collision detection, all up to a speed of 2 Mb/s (independent of the chosen encoding scheme). These functions make it an optimum controller for a StarLAN node. The 82588 has a very conventional microcomputer bus interface, easing the job of interfacing it to any processor.

1.3 Organization of the Application Note

This application note has two objectives. One is to describe StarLAN in practical terms to prospective implementers. The other is to illustrate designing with 82588, particularly as related to StarLAN which is expected to emerge as its largest application area.

Section 2 of this Application Note describes the StarLAN network, its basic components, collision detection, signal propagation and network parameters. Sections 3 and 4 describe the 82588 LAN controller and its role in the StarLAN network. Section 5 goes into the details of designing a StarLAN node for the IBM PC. Section 6 describes the design of the HUB. Both these designs have been implemented and operated in an actual StarLAN environment. Section 7 documents the software used to drive the 82588. It gives the actual procedures used to do operations like, configure, transmit and receive frames. It also shows how to use the DMA controller and interrupt controller in the IBM PC and goes into the details of doing I/O on the PC using DOS calls. Appendix A shows oscilloscope traces of the signals at various points in the network. Appendix B describes the multiple point extension (MPE) being considered by IEEE. Appendixes C and D talk about advanced usages of the 82588; working with only one DMA channel, and measuring network delays with the 82588.

1.4 References

For additional information on the 82588, see the Intel Microcommunications Handbook. StarLAN specification are currently available in draft standard form through the IEEE 802.3 Working Group.

2.0 StarLAN

StarLAN is a low cost 1 Mb/s networking solution aimed at office automation applications. It uses a star

topology with the nodes connected in a point-to-point fashion to a central HUB. HUBs can be connected in a hierarchical fashion. Up to 5 levels are supported. The maximum distance between a node and the adjacent HUB or between two adjacent HUBs is 800 ft. (about 250 meters) for 24 gauge wire and 600 ft. (about 200 meters) for 26 gauge wire. Maximum node-to-node distance with one HUB is 0.5 km, hence IEEE 802.3 designation of type 1BASE5. 1 stands for 1 Mb/s and BASE for baseband. (StarLAN doesn't preclude the use of more than 800 ft wiring provided 6.5 dB maximum attenuation is met, and cable propagation delay is no more than 4 bit times).

One of the most attractive features of StarLAN is that it uses telephone grade twisted pair wire for the transmission medium. In fact, existing installed telephone wiring can also be used for StarLAN. Telephone wiring is very economical to buy and install. Although use of telephone wiring is an obvious advantage, for small clusters of nodes, it is possible to work around the use of building wiring.

Factors contributing to low cost are:

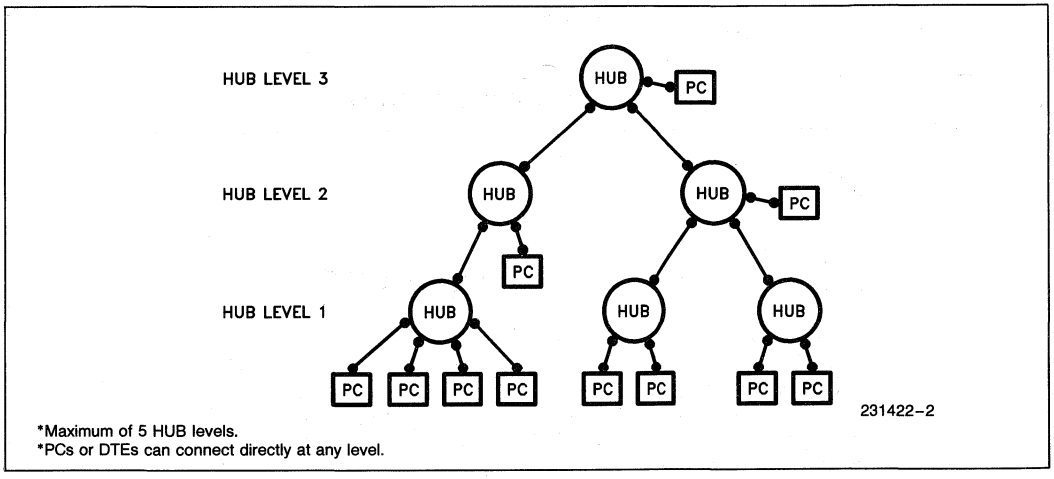
1) Use of telephone grade, unshielded, 24 or 26 gauge twisted pair wire transmission media.
2) Installed base of redundant telephone wiring in most buildings.
3) Buildings are designed for star topology wiring. They have conduits leading to a central location.
4) Availability of low cost VLSI LAN controllers like the 82588 for low cost applications and the 82586 for high performance applications.
5) Off-the-shelf, Low cost RS-422, RS-485 drivers/receivers compatible with the StarLAN analog interface requirements.

2.1 StarLAN Topology

StarLAN, as the name suggests, uses a star topology. The nodes are at the extremities of a star and the central point is called a HUB. There can be more than one HUB in a network. The HUBs are connected in a hierarchical fashion resembling an inverted tree, as shown in Figure 1, where nodes are shown as PCs. The HUB at the base (at level 3) of the tree is called the Header Hub (HHUB) and others are called Intermediate HUBs (IHUB). It will become apparent, later in this section, that topologically, this entire network of nodes and HUBs is equivalent to one where all the nodes are connected to a single HUB. Also StarLAN doesn't limit the number of nodes or HUBS at any given level.

2.1.1 TELEPHONE NETWORK

StarLAN is structured to run parallel to the telephone network in a building. The telephone network has, in fact, exactly the same star topology as StarLAN. Let us now examine how the telephone system is typically laid out in a building in the USA. Figure 2 shows how a typical building is wired for telephones. 24 gauge unshielded twisted pair wires emanate from a Wiring Closet. The wires are in bundles of 25 or 50 pairs. The bundle is called D inside wiring (DIW). The wires in these cables end up at modular telephone jacks in the wall. The telephone set is either connected directly to



*Maximum of 5 HUB levels.
*PCs or DTEs can connect directly at any level.

Figure 1. StarLAN Topology

the jack or through an extension cable. Each telephone generally needs one twisted pair for voice and another for auxilliary power. Thus, each modular jack has 2 twisted pairs (4 wires) connected to it. A 25 pair DIW cable can thus be used for up to 12 telephone connections. In most buildings, not all pairs in the bundle are used. Typically, a cable is used for only 4 to 8 telephone connections. This practice is followed by telephone companies because it is cheaper to install extra wires initially, rather than retrofitting to expand the existing number of connections. As a result, a lot of extra, unused wiring exists in a building. The stretch of cable between the wiring closet and the telephone jack is typically less than 800 ft. (250 meters). In the wiring closet the incoming wires from the telephones are routed to another wiring closet, a PABX or to the central office through an interconnect matrix. Thus, the wiring closet is a concentration point in the telephone network. There is also a redundancy of wires between the wiring closets.

2.1.2 StarLAN AND THE TELEPHONE NETWORK

StarLAN does not have to run on building wiring, but the fact that it can significantly adds to its attractiveness. Figure 3 shows how StarLAN piggybacks on telephone wiring. Each node needs two twisted pair wires to connect to the HUB. The unused wires in the 25 pair DIW cables provide an electrical path to the wiring closet, where the HUB is located. Note that the telephone and StarLAN are electrically isolated. They only use the wires in the same bundle cable to connect to the wiring closet. Within the wiring closet, StarLAN wires connect to a HUB and telephone wires are routed to a different path. Similar cable sharing can occur in connecting HUBs to one another. See Figure 4 for a typical office wired for StarLAN through telephone wiring.

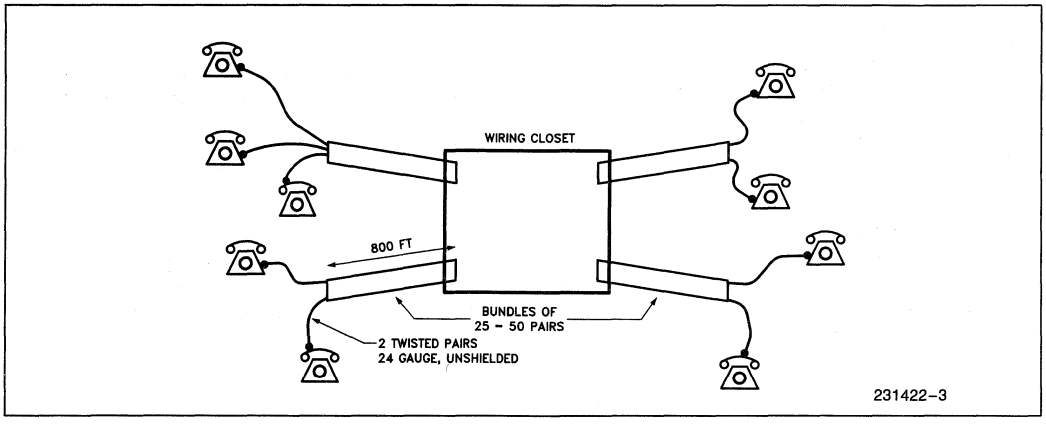


Figure 2. Telephone Wiring in a Building

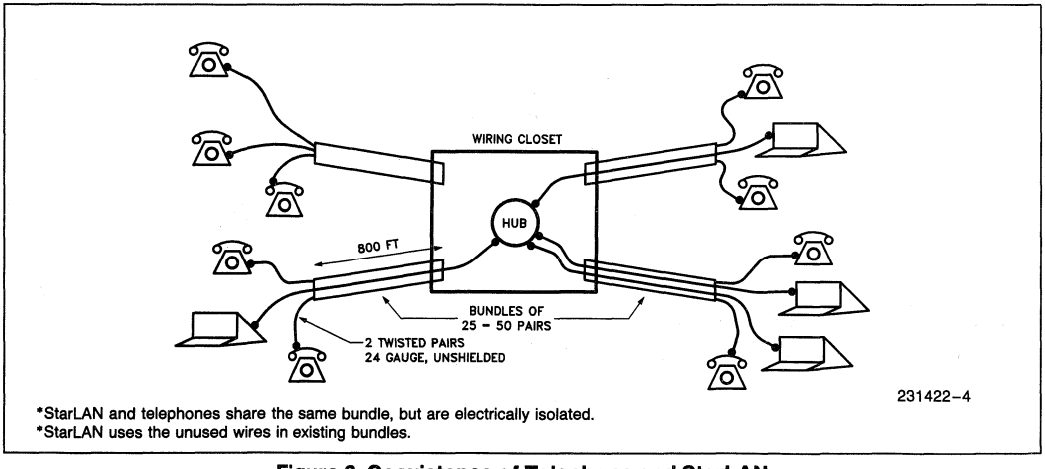


Figure 3. Coexistence of Telephone and StarLAN

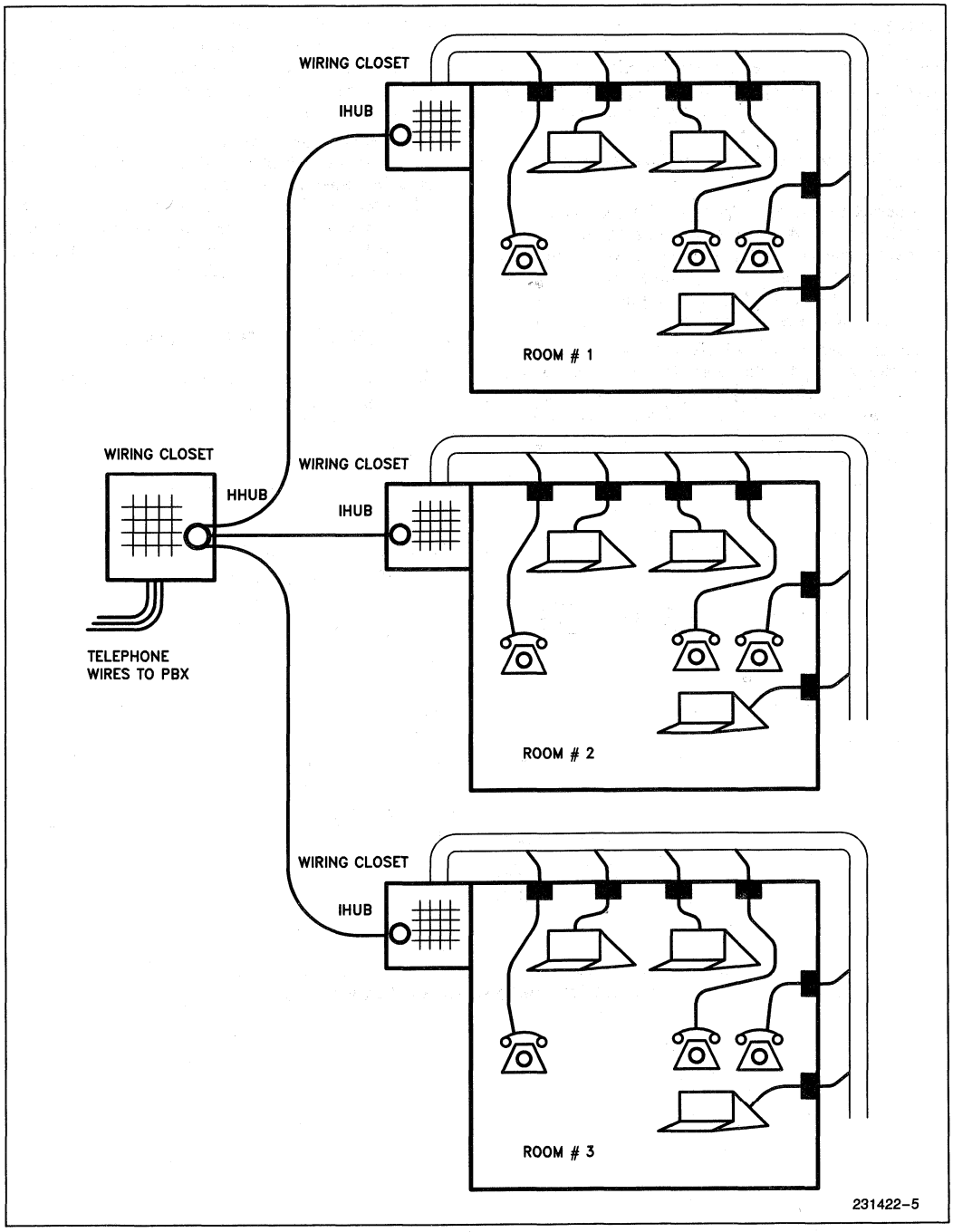


Figure 4. A Typical Office Using Telephone Wiring for StarLAN

2.1.3 StarLAN AND Ethernet

StarLAN and Ethernet are similar CSMA/CD networks. Since Ethernet has existed longer and is better understood, a comparison of Ethernet with StarLAN is worthwhile.

1. The data rate of Ethernet is 10Mb/s and that of StarLAN is 1 Mb/s.
2. Ethernet uses a bus topology with each node connected to a coaxial cable bus via a 50 meter transceiver cable containing four shielded twisted pair wires. StarLAN uses a star topology, with each node connected to a central HUB by a point to point link through two pairs of unshielded twisted pair wires.
3. Collision detection in Ethernet is done by the transceiver connected to the coaxial cable. Electrically, it is done by sensing the energy level on the coax cable. Collision detection in StarLAN is done in the HUB by sensing activity on more than one input line connected to the HUB.
4. In Ethernet, the presence of collision is signalled by the transceiver to the node by a special collision detect signal. In StarLAN, it is signalled by the HUB using a special collision presence signal on the receive data line to the node.
5. Ethernet cable segments are interconnected using repeaters in a non-hierarchical fashion so that the distance between any two nodes does not exceed 2.8 kilometers. In StarLAN, the maximum distance between any two nodes is 2.5 kilometers. This is achieved by wiring a maximum of five levels of HUBs in a hierarchical fashion.

2.2 Basic StarLAN Components

A StarLAN network has three basic components:

1. StarLAN node interface
2. StarLAN HUB
3. Cable

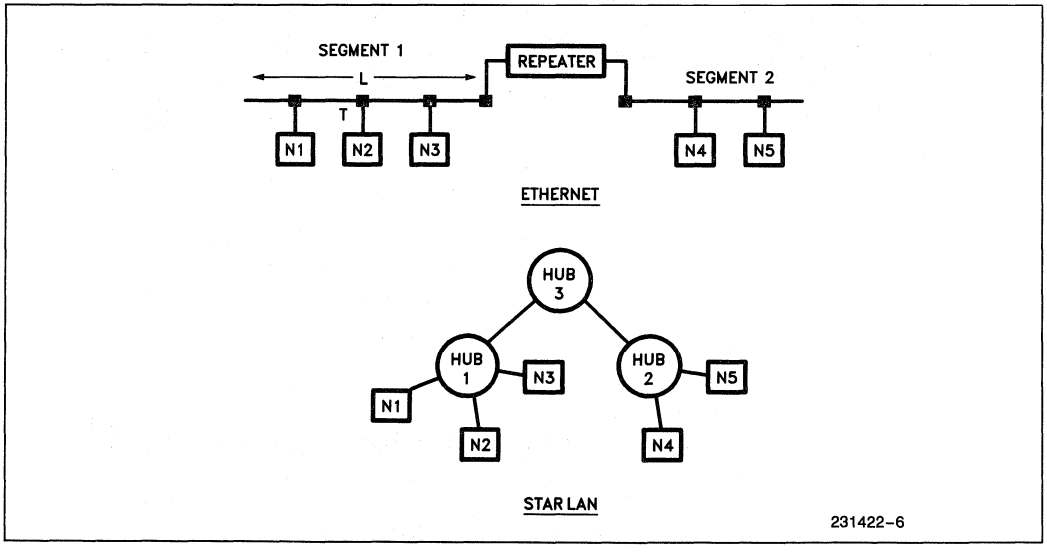


Figure 5. Ethernet and StarLAN Similarities

2.2.1 A StarLAN NODE INTERFACE

Figure 6 shows a typical StarLAN node interface. It interfaces to a processor on the system side. The processor runs the networking software. The heart of the node interface is the LAN controller which does the job of receiving and transmitting the frames in adherence to the IEEE 802.3 standard protocol. It maintains all the timings—like the slot time, interframe spacing etc.—required by the network. It performs the functions of framing, deferring, backing-off, collision detection which are necessary in a CSMA/CD network. It also does Manchester encoding of data to be transmitted and clock separation—or decoding—of the Manchester encoded data that is received. These signals before going to the unshielded twist pair wire, may undergo pulse shaping (optional) pulse shaping basically slows down the fall/rise times of the signal. The purpose of that is to diminish the effects of cross-talk and radiation on adjacent pairs sharing the same bundle (digital voice, T1 trunks, etc). The shaped signal is sent on to the twisted pair wire through a pulse transformer for DC isolation. The signals on the wire are thus differential, DC isolated from the node and almost sinusoidal (due to shaping and the capacitance of the wire).

NOTE:
Work done by the IEEE 802.3 committee has shown that no slew rate control on the drivers is required. Shaping by the transformer and the cable is sufficient to avoid excessive EMI radiation and crosstalk.

The squelch circuit prevents idle line noise from affecting the receiver circuits in the LAN controller. The squelch circuit has a 600 mv threshold for that purpose. Also as part of the squelch circuitry an envelope detector is implemented. Its purpose is to generate an envelope of the transitions of the RXD line. Its output serve as a carrier sense signal. The differential signal from the HUB is received using a zero-crossing RS-422 receiver. Output of the receiver, qualified by the squelch circuit, is fed to the RxD pin of the LAN controller. The RxD signal provides three kinds of information:

1) Normal received data, when receiving the frame.
2) Collision information in the form of the collision presence signal from the HUB.
3) Carrier sense information, indicating the beginning and the end of frame. This is useful during transmit and receive operations.

2.2.2 StarLAN HUB

HUB is the point of concentration in StarLAN. All the nodes transmit to the HUB and receive from the HUB. Figure 7 shows an abstract representation of the HUB. It has an upstream and a downstream signal processing unit. The upstream unit has N signal inputs and 1 signal output. And the downstream unit has 1 input and N output signals. The inputs to the upstream unit come from the nodes or from the intermediate HUBs (IHUBs) and its output goes to a higher level HUB. The downstream unit is connected the other way around; input from an upper level HUB and the outputs to nodes or lower level IHUBs. Physically each input and output consist of one twisted pair wire carrying a differential signal. The downstream unit essentially just re-times the signal received at the input, and sends it to all its outputs. The functions performed by the upstream unit are:

1. Collision detection
2. Collision Presence signal generation
3. Signal Retiming
4. Jabber Function
5. Start of Idle protection timer

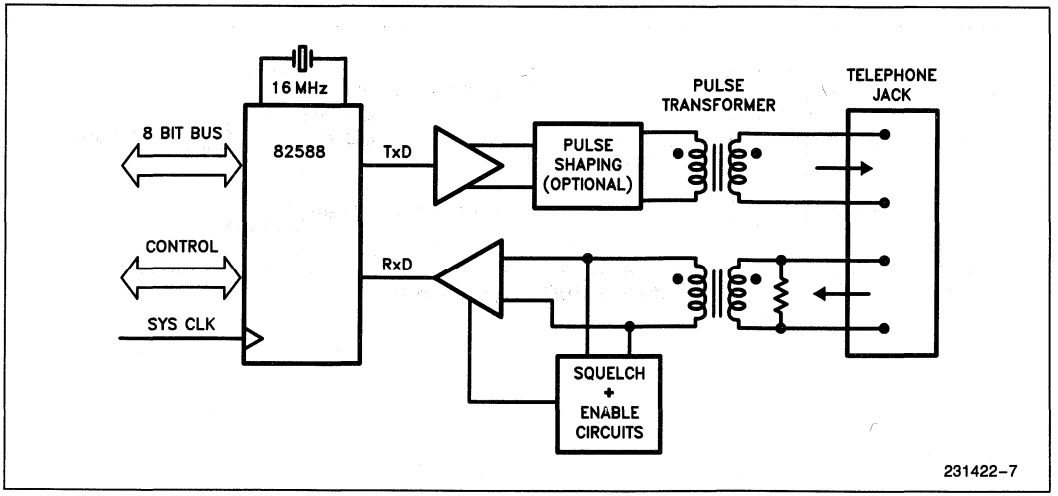


Figure 6. 82588 Based StarLAN Node

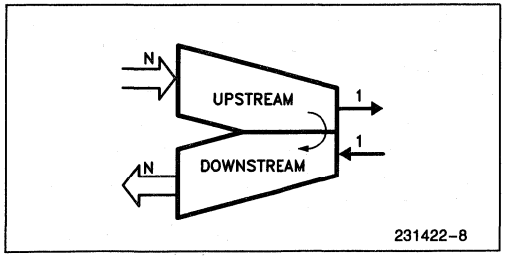


Figure 7. A StarLAN HUB

The collision detection in the HUB is done by sensing the activity on the inputs. If there is activity (or transitions) on more than one input, it is assumed that more than one node is transmitting. This is a collision. If a collision is detected, a special signal called the Collision Presence Signal is generated. This signal is generated and sent out as long as activity is sensed on any of the input lines. This signal is interpreted by every node as an occurrence of collision. If there is activity only on one input, that signal is re-timed—or cleaned up of any accumulated jitter—and sent out. Figure 8 shows the input to output relations of the HUB as a black box.

If a node transmits for too long the HUB exercises a Jabber function to disable the node from interfering with traffic from other nodes. There are two timers in the HUB associated with this function and their operation is described in section 6.

The last function implemented by the HUB is the start of Idle protection timer. During the end of reception, the HUB will see a long undershoot at its input port. This undershoot is a consequence of the transformer discharging accumulated charge during the 2 microseconds of high of the idle pattern. The HUB should implement a protection mechanism to avoid the undesirable effects of that undershoot.

Figure 9 shows a block diagram of the HUB. A switch position determines whether the HUB is an IHUB or a HHUB (Header HUB). If the HUB is an IHUB, the switch decouples the upstream and the downstream units. HHUB is the highest level HUB; it has no place to send its output signal, so it returns its output signal (through the switch) to the outputs of the downstream unit. There is one and only one HHUB in a StarLAN network and it is always at the base of the tree. The returned signal eventually reaches every node in the network through the intermediate nodes (if any). StarLAN specifications do not put any restrictions on the number of IHUBS at any level or on number of inputs to any HUB. The number of inputs per HUB are typically 6 to 12 and is dictated by the typical size of clusters in a given networking environment.

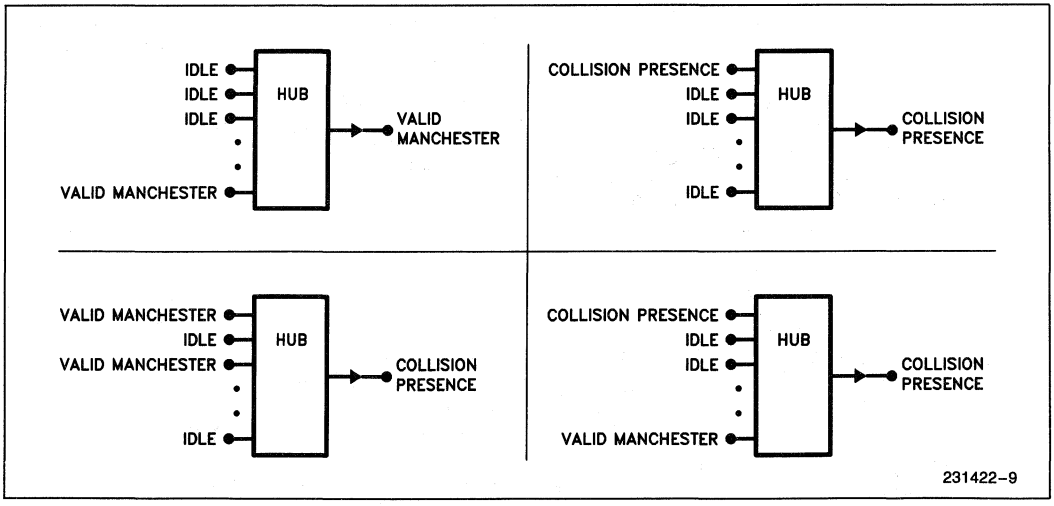


Figure 8. HUB as a Black Box

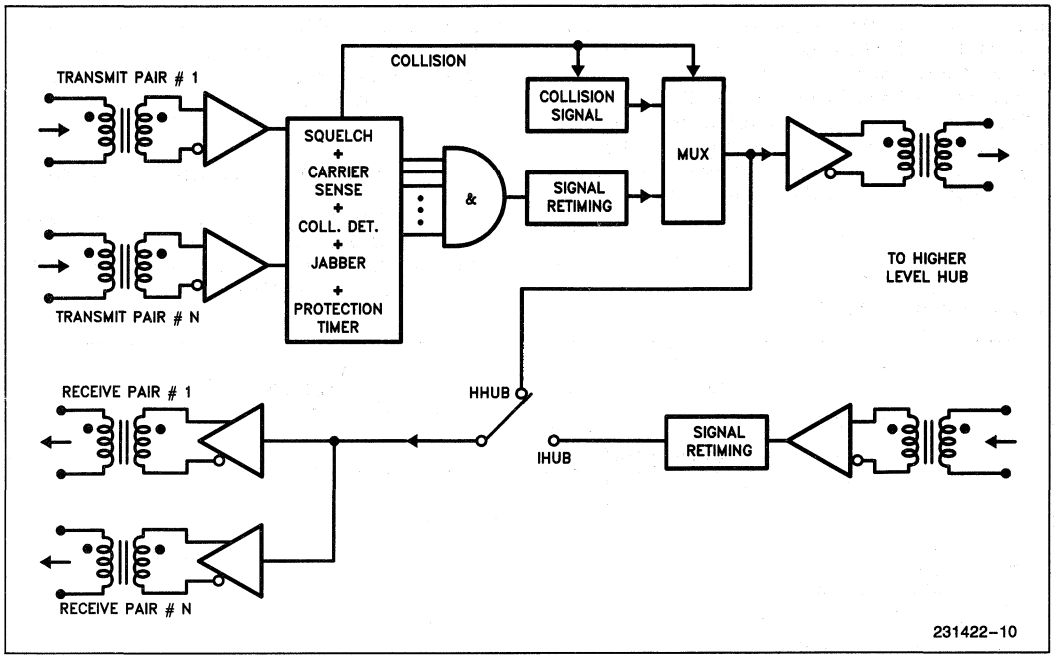


Figure 9. StarLAN HUB Block Diagram

2.2.3 StarLAN CABLE

Unshielded telephone grade twisted pair wires are used to connect a node to a HUB or to connect two HUBs. This is one of the cheapest types of wire and an important factor in bringing down the cost of StarLAN.

Although the 24 gauge wire is used for long stretches, the actual connection between the node and the telephone jack in the wall is done using extension cable, just like connecting a telephone to a jack. For very short StarLAN configurations, where all the nodes and the HUB are in the same room, the extension cable with plugs at both ends may itself be sufficient for all the wiring. (Extension cables must be of the twisted pair kind, no flat cables are allowed).

The telephone twisted pair wire of 24 gauge has the following characteristics:

Attenuation	: 42.55 db/mile @ 1 MHz
DC Resistance	: 823.69 Ω/mile
Inductance	: 0.84 mH/mile
Capacitance	: 0.1 μF/mile
Impedance	: 92.6Ω, −4 degrees @ 1 MHz

Experiments have shown that the sharing of the telephone cable with other voice and data services does not cause any mutual harm due to cross-talk and radiation, provided every service meets the FCC limits.

Although it is outside the scope of the IEEE 802.3 1BASE5 standard, there is considerable interest in using fiber optics and coaxial cable for node to HUB or HUB to HUB links especially in noisy and factory environments. Both these types of cables are particularly suited for point-to-point connections. Even mixing of different types of cables is possible (this kind of environments are not precluded).

NOTE:
StarLAN IEEE 802.3 1BASE5 draft calls for a maximum attenuation of 6.5 dB between the transmitter and the corresponding receiver at all frequencies between 500 KHz to 1 MHz. Also the maximum allowed cable propagation delay is 4 microseconds.

2.3 Framing

Figure 10 shows the format of a 802.3 frame. The beginning of the frame is marked by the carrier going active and the end marked by carrier going inactive. The preamble has a 56 bit sequence of 101010 . . . . ending in a 0. This is followed by 8 bits of start of frame delimiter (sfd) − 10101011. These bits are transmitted with the MSB (leftmost bit) transmitted first. Source and destination fields are 6 bytes long. The first byte is the least significant byte. These fields are transmitted with LSB first. The length field is 2 bytes long and gives the length of data in the Information field. The entire information field is a minimum of 46 bytes and a maximum of 1500 bytes. If the data content of the Informa-

tion field is less than 46, padding bytes are used to make the field 46 bytes long. The Length field indicates how much real data is in the Information field. The last 32 bits of the frame is the Frame Check Sequence (FCS) and contains the CRC for the frame. The CRC is calculated from the beginning of the destination address to the end of the Information field. The generating polynomial (Autodin II) used for CRC is:

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^{8} + X^{7} + X^{5} + X^{4} + X^{2} + X + 1$$

No need for Figure N.

The frames can be directed to a specific node (LSB of address must be 0), to a group of nodes (multicast or group—LSB of address must be 1) or all nodes (broadcast—all address bits must be 1).

2.4 Signal Propagation and Collision

Figure 11 will be used to illustrate three typical situations in a StarLAN with two IHUBs and one HHUB. Nodes A and B are connected to HUB1, nodes C and D to HUB2 and node E to HUB3.

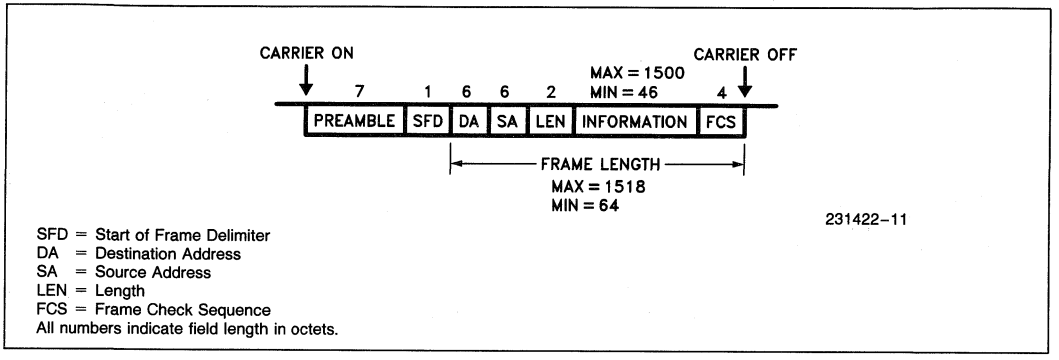
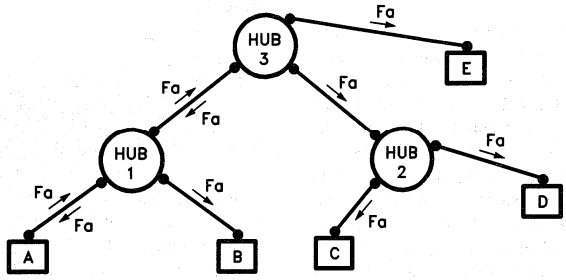
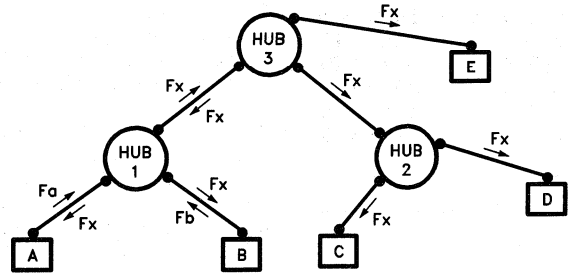


Figure 10. Framing



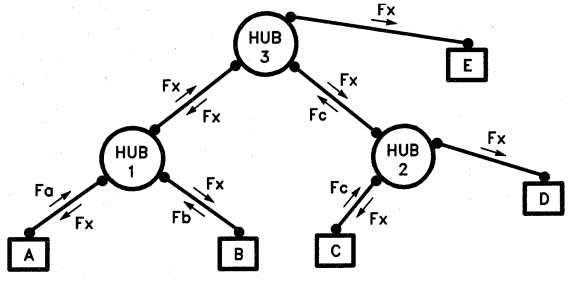
231422-12

Situation #1. A Transmitting



231422-13

Situation #2. A & B Transmitting



231422-14

Situation #3. A, B & C Transmitting

HUB1, HUB2 are IHUBs
HUB3 is the HHUB
Fa, Fb, Fc—Frames from nodes A, B & C
Fx—Collision Presence Signal

Figure 11. Signal Propagation and Collisions

2.4.1 Situation #1

Whenever node A transmits a frame Fa, it will reach HUB1. If node B is silent, there is no collision. HUB1 will send Fa to HUB3 after re-timing the signal. If nodes C, D and E are also silent, there is no collision at HUB2 or HUB3. Since HUB3 is the HHUB, it sends the frame Fa to HUB1, HUB2 and to node E after re-timing. HUB1 and HUB2 send the frame Fa to nodes A, B and C, D. Thus, Fa reaches all the nodes on the network including the originator node A. If the signal received by node A is a valid Manchester signal and not the Collision Presence Signal (CPS) for the entire duration of the slot time, then the node A assumes that it was a successful transmission.

2.4.2 Situation #2

If both nodes A and B were to transmit, HUB1 will detect it as a collision and will send signal Fx (the Collision Presence Signal) to the HUB3—Note that HUB1 does not send Fx to nodes A and B yet. HUB 3 receives a signal from HUB1 but nothing from node E or HUB2, thus it does not detect the situation as a collision and simply re-times the signal Fx and sends it to node E, HUB2 and HUB1. Fx ultimately reach all the nodes. Nodes A and B detect this signal as CPS and call it a collision.

2.4.3 Situation #3

In addition to nodes A and B, if node C were also to transmit, the situation at HUB1 will be the same as in situation #2. HUB2 will propagate Fc from C towards HUB3. HUB3 now sees two of its inputs active and hence generates its own Fx signal and sends it towards each node.

These situations should also illustrate the point made earlier in the chapter that, the StarLAN network, with nodes connected to multiple HUBs is, logically, equivalent to all the nodes connected to a single HUB (Yet there are some differences between stations connected at different HUB levels, those are due to different delays to the header hub HHUB).

2.5 StarLAN System and Network Parameters

Preamble length (incl. sfd) ..... 64 bits
Address length ..... 6 bytes
FCS length CRC (Autodin II) ..... 32 bits
Maximum frame length ..... 1518 bytes
Minimum frame length ..... 64 bytes
Slot time ..... 512 bit times
Interframe spacing ..... 96 bit times
Minimum jam timing ..... 32 bit times
Maximum number of collisions 16
Backoff limit 10
Backoff method ..... Truncated binary exponential
Encoding ..... Manchester

Clock tolerance ..... ±0.01% (100 ppm)
Maximum jitter per segment ..... ±62.5 ns

3.0 LAN CONTROLLER FOR StarLAN

One of the attractive features of StarLAN is the availability of the 82588, a VLSI LAN controller, designed to meet the needs of a StarLAN node. The main requirements of a StarLAN node controller are:

1. IEEE 802.3 compatible CSMA/CD controller.
2. Configurable to StarLAN network and system parameters.
3. Generation of all necessary clocks and timings.
4. Manchester data encoding and decoding.
5. Detection of the Collision Presence Signal.
6. Carrier Sensing.
7. Squelch or bad signal filtering.
8. Fast and easy interface to the processor.

82588 performs all these functions in silicon, providing a minimal hardware interface between the system processor and the StarLAN physical link. It also reduces the software needed to run the node, since a lot of functions, like deferring, back off, counting the number of collisions etc., are done in silicon.

3.1 IEEE 802.3 Compatibility

The CSMA/CD control unit on the 82588 performs the functions of deferring, maintaining the Interframe Space (IFS) timing, reacting to collision by generating a jam pattern, calculating the back-off time based on the number of collisions and a random number, decoding the address of the incoming frame, discarding a frame that is too short, etc. All these are performed by the 82588 in accordance to the IEEE 802.3 standards. For inter-operability of different nodes on the StarLAN network it is very important to have the controllers strictly adhere to the same standards.

3.2 Configurability of the 82588

Almost all the networking parameters are programmable over a wide range. This means that the StarLAN parameters form a subset of the total potential of the 82588. This is a major advantage for networks whose standards are being defined and are in a flux. It is also an advantage when carrying over the experience gained with the component in one network to other applications, with differing parameters (leveraging the design).

The 82588 is initialized or configured to its working environment by the CONFIGURE command. After the execution of this command, the 82588 knows its system and network parameters. A configure block in

memory is loaded into the 82588 by DMA. This block contains all the parameters to be programmed as shown in Figure 12. Following is a partial list of the parameters with the programmable range and the StarLAN value:

Parameter	Range	StarLAN Value
Preamble length	2, 4, 8, 16 bytes	8
Address length	0 to 6 bytes	6
CRC type	16, 32 bit	32
Minimum frame length	6 to 255 bytes	64
Interframe spacing	12 to 255 bit times	96
Slot time	1 to 2047 bit times	512
Number of retries	0 to 15	15
Data encoding	NRZI, Man., Diff. Man.	Manch.
Collision detection	Code viol., Bit comp.	Code Viol.

Beside these, there are many other options available, which may or may not apply to StarLAN:

Data sampling rate of 8 or 16
Operating in Promiscuous mode
Reception of Broadcast frames
Internal loopback operation
External loopback operation
Transmit without CRC
HDLC Framing
:
:

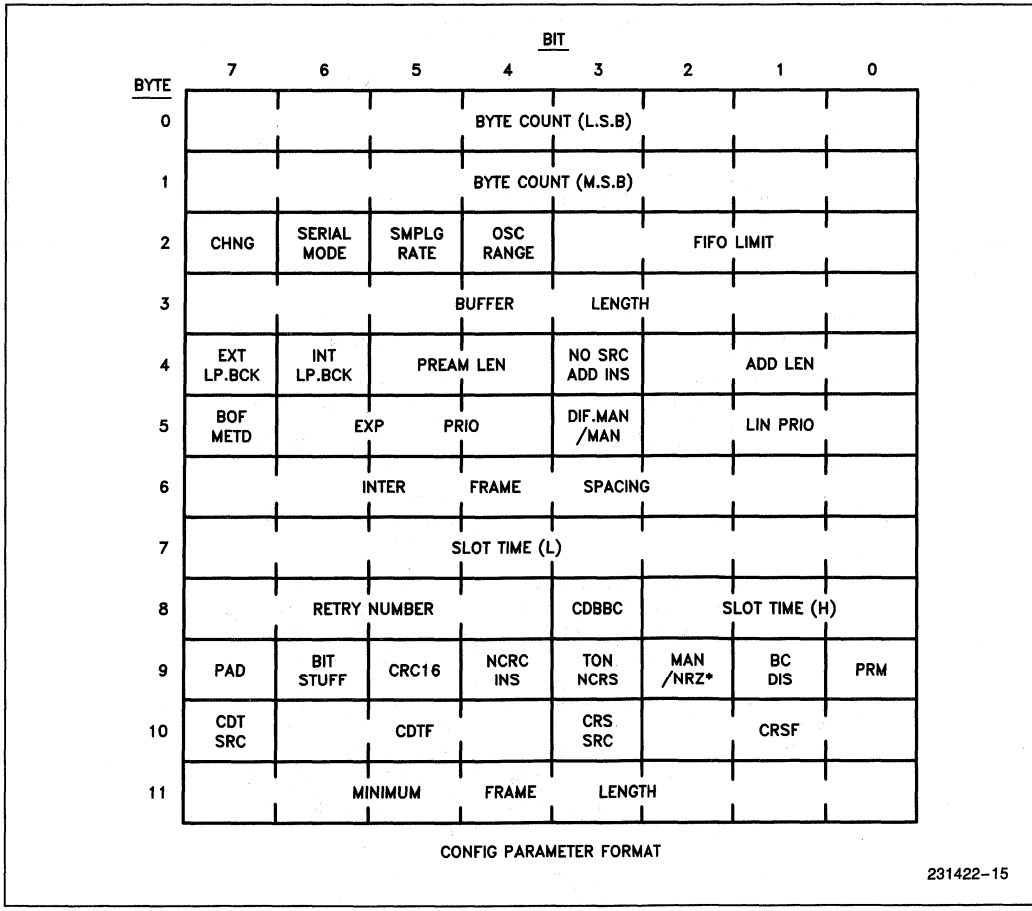


Figure 12. Configuration Block

3.3 Clocks and Timers

The 82588 requires two clocks; one for the operation of the system interface and another for the serial side. Both clocks are totally asynchronous to each other. This permits transmitting and receiving frames at data rates that are virtually independent of the speed at which the system interface operates.

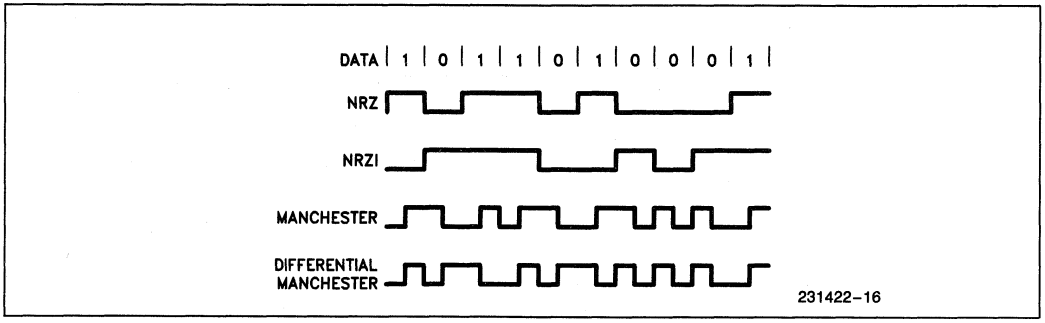
The serial clock can be generated on chip using just an external crystal of a value 8 or 16 times the desired bit rate. An external clock may also be used.

The 82588 has a set of timers to maintain various timings necessary to run the CSMA/CD control unit. These are timings for the Slot time, Interframe spacing time, Back off time, Number of collisions, Minimum frame length, etc. These timers are started and stopped automatically by the 82588.

3.4 Manchester Data Encoding and Decoding

In StarLAN the data transmitted by the node must be encoded in Manchester format. The node should also be able to decode Manchester encoded data when receiving a frame—a process also known as clock recovery. The 82588 does the encoding and decoding of data bits on chip for data rates up to 2 Mb/s.

Besides Manchester, the 82588 can also do encoding and decoding in NRZI and Differential Manchester formats. Figure 13 shows samples of encoding in



Encoding Method	Mid Bit Cell Transitions	Bit Cell Boundary Transitions
NRZ	Do not exist.	Identical to original data.
NRZI	Do not exist.	Exist only if original data bit equals 0. Dependent on present encoded signal level: to 0 if 1 to 1 if 0
Manchester	Exist for every bit of the original data: from 0 to 1 for 1 from 1 to 0 for 0	Exist for consequent equal bits of original data: from 1 to 0 for 1 1 from 0 to 1 for 0 0
Differential Manchester	Exist for every bit of the original data. Dependent on present Encoded signal level: to 0 if 1 to 1 if 0	Exist only if original data bit equals 0. Dependent on present Encoded signal level: to 0 if 1 to 1 if 0

Figure 13. 82588 Data Encoding Rules

these three formats. The main advantage of NRZI over the other two is that NRZI requires half the channel bandwidth, for any given data rate. On the other hand, since the NRZI signal does not have as many transitions as the other two, clock recovery from it is more difficult. The main advantage of Differential Manchester over straight Manchester is that for a signal that is differentially driven (as in RS 422), crossing of the two wires carrying the data does not change the data received at the receiver. In other words, NRZI and Differential Manchester encoding methods are polarity insensitive (Even though NRZI, Differential Manchester are polarity insensitive, the 82588 expects a high level in the RXD line to detect carrier inactive at the end of frames).

3.5 Detection of the Collision Presence Signal

In a StarLAN network, HUB informs the nodes that a collision has occurred by sending the Collision Presence Signal (CPS) to the nodes. The CPS signal is a special signal which contains violations in Manchester encoding. Figure 14 shows the CPS signal. It has a 5 ms period, looking very much like a valid Manchester signal except for missing transitions (or violations) at periodic intervals. When the 82588 decodes this signal, it fails to see mid-cell transitions repeatedly at intervals of 2.5 bit times and hence calls it a code violation. The edges of CPS are marked for illustration as a, b, c, d, . . . l. Let us see how the 82588 interprets the signal if it starts calling the edge 'a' as the mid-cell transition for '1'. Then edge at 'b' is '0'. Now the 82588 expects to see an edge at '*' but since there is none, it is a Manchester code violation. The edge that eventually does occur at 'd' is then used to re-synchronize and, since it is a falling edge, it is taken as a mid-cell transition for '0'. The edge at 'e' is for a '1' and then again there is no edge at '*'. This goes on, with the 82588 flagging code violation and re-synchronizing again every 2.5 bit times. When a transmitting node sees this CPS signal being returned by the HUB (instead of a valid Manchester signal it transmitted), it assumes that a collision occurred. The 82588 has two built-in mechanisms to detect collisions. These mechanisms are very general and can be used for a very broad class of applications to detect collisions in a CSMA/CD network. Using these mechanisms, the 82588 can detect collisions (two or more nodes transmitting simultaneously) by just receiving the collided signal during transmission, even if there was no HUB generating the CPS signal.

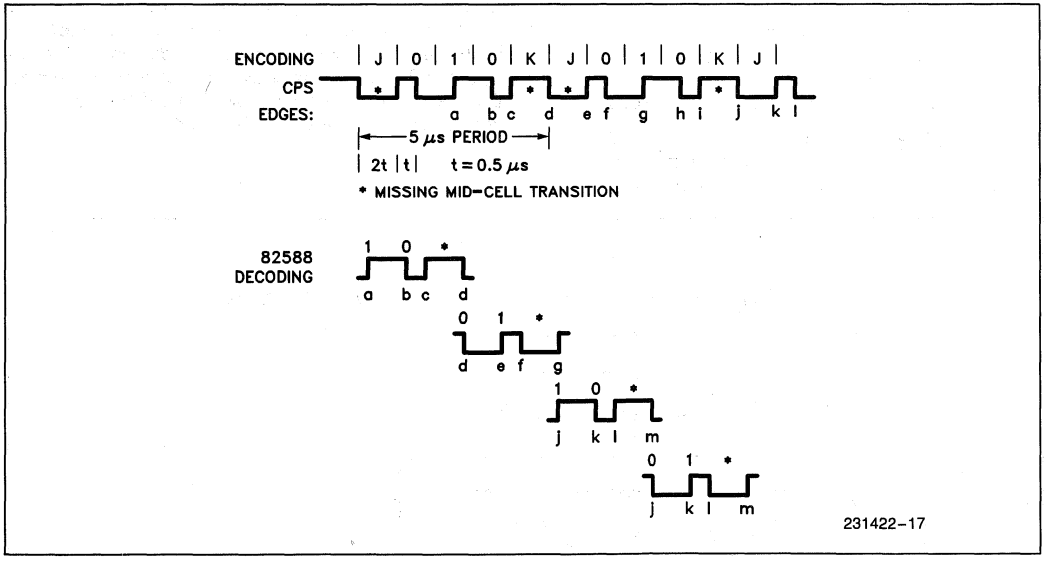


Figure 14. 82588 Decoding the Collision Presence Signal

3.5.1 COLLISION DETECTION BY CODE VIOLATION

If during transmission, the 82588 sees a violation in the encoding (Manchester, NRZI or Differential Manchester) used, then it calls it a collision by aborting the transmission and transmitting a 32 bit jam pattern. The algorithm used to detect collisions, and to do the data decoding, is based on finding the number of sampling clocks between an edge to the next one. Suppose an edge occurred at time 0, the sampling instant of the next edge determines whether it was a collision (C), a long pulse (L)—with a nominal width of 1 bit time—,or a short pulse (S)—nominal width of half a bit time. The following two charts show the decoding and collision detection algorithm for sampling rates of 8 and 16 when using Manchester encoding. The numbers at the bottom of the line indicate sampling instances after the occurrence of the last edge (at 0). The alphabets on the top show what would be inferred by the 82588 if the next edge were to be there.

Sampling rate = 8 (clock is 8x bit rate)

```
  C  C  S  S  S  L  L  L  L  L  C  C
  |  |  |  |  |  |  |  |  |  |  |  |  |
  0  1  2  3  4  5  6  7  8  9 10 11 12 13
```

Collision also if:
RxD stays low for 13 samples or more
A mid cell transition is missing

Sampling rate = 16 (clock is 16x bit rate)

```
  CCCCCSSSSSCLLLLLLLLLCCCC
  |  |  |  |  |  |  |  |  |  |  |  |  |
  0  2  4  6  8 10 12 14 16 18 20 22 24 26
```

Collision also if:
RxD stays low for 25 samples or more
A mid cell transition is missing

A single instance of code violation can qualify as collision. The 82588 has a parameter called collision detect filter (CDT Filter) that can be configured from 0 to 7. This parameter determines for how many bit times the violation must remain active to be flagged as a collision. For StarLAN CDT Filter must be configured to 0—that is disabled.

3.5.2 COLLISION DETECTION BY SIGNATURE (OR BIT) COMPARISON

This method of collision detection compares a signature of the transmitted data with that of the data received on the RxD pin while transmitting. Figure 15 shows a block diagram of the logic. As the frame is transmitted it flows through the CRC generation logic. A timer, called the Tx slot timer, is started at the same time that the CRC generation starts. When the count in the timer reaches the slot time value, the current value of the CRC generator is latched in as the transmit signature. As the frame is returned back (through the HUB) it flows through the CRC checker. Another timer—Rx slot timer—is started at the same time as the CRC checker starts checking. When this timer reaches the slot time value, the current value of the CRC checker is latched in as the receive signature. If the received signature matches the transmitted one, then it is assumed that there was no collision. Whereas, if the signatures do not match, a collision is assumed to have occurred.

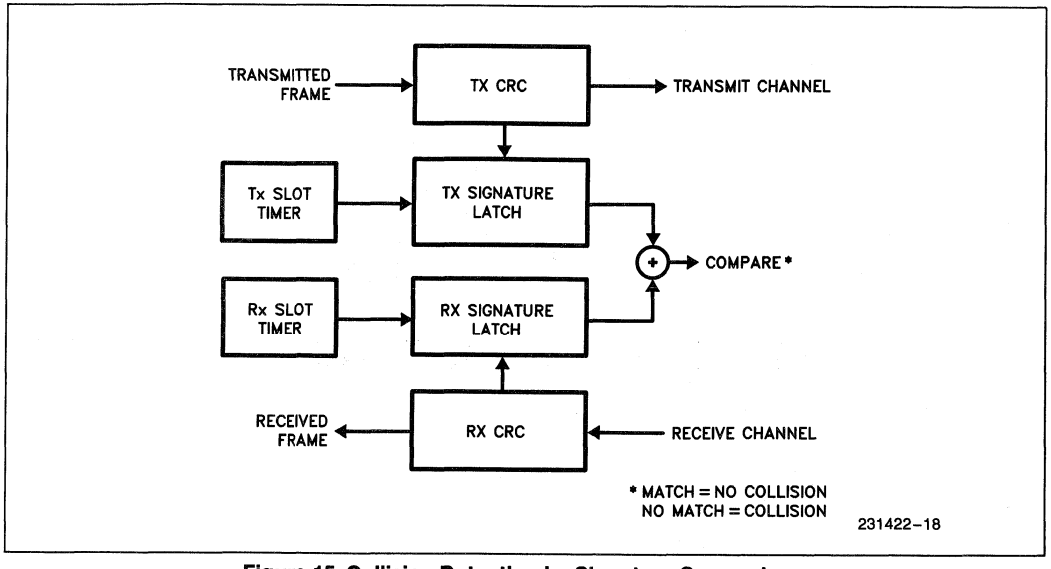


Figure 15. Collision Detection by Signature Comparison

Note that, even if the collision were to occur in the first few bits of the frame, a slot time must elapse before it is detected. In the code violation method, collision is detected within a few bit times. However, since the signature method compares the signatures, which are characteristic of the frame being transmitted, it is more robust. The code violation method can be fooled by returning a signal to the 82588 which is not the same as the transmitted signal but is a valid Manchester signal—like a 1 MHz signal. Both methods can be used simultaneously giving a combination of speed and robustness.

NOTE:
In order to reliably detect a collision using the collision by bit comparison mode, the transmitter must still be transmitting up to the point where the receiver has seen enough bits to complete its signature. Otherwise, the transmitter may be done before the RX signature is completed resulting in an undetected collision. A sufficient condition to avoid this situation is to transmit frames with a minimum length of 1.5 * slot-time (see Figure 16).

3.5.3 ADDITIONAL COLLISION DETECTION MECHANISM

In addition to the collision detection mechanisms described in the preceding sections, the 82588 also flags collision when after starting a transmission any of the following conditions become valid:

a) Half a slot time elapses and the carrier sense of 82588 is not active.

b) Half a slot time + 16 bit times elapse and the opening flag (sfd) is not detected.

c) Carrier sense goes inactive after an opening flag is received with transmitter still active.

These mechanisms add a further robustness to the collision detection mechanism of the 82588. It is also possible to OR an externally generated collision detect signal to the internally generated condition by bit comparison (see Figure 17).

3.6 Carrier Sensing

A StarLAN network is considered to be busy if there are transitions on the cable. Carrier is supposed to be active if there are transitions. Every node controller needs to know when the carrier is active and when not. This is done by the carrier sensing circuitry. On the 82588 this circuit is on chip. It looks at the RxD (receive data) pin and if there are transitions, it turns on an internal carrier sense signal. It turns off the carrier sense signal if RxD remains in idle (high) state for 13/8 bit times. This carrier sense information is used to mark the start of the interframe space time and the back off time. The 82588 also defers transmission when the carrier sense is active.

When operating in the NRZI encoded mode, carrier sense is turned off if RxD pin is in the idle state for 8 bit times or more (see Figure 18).

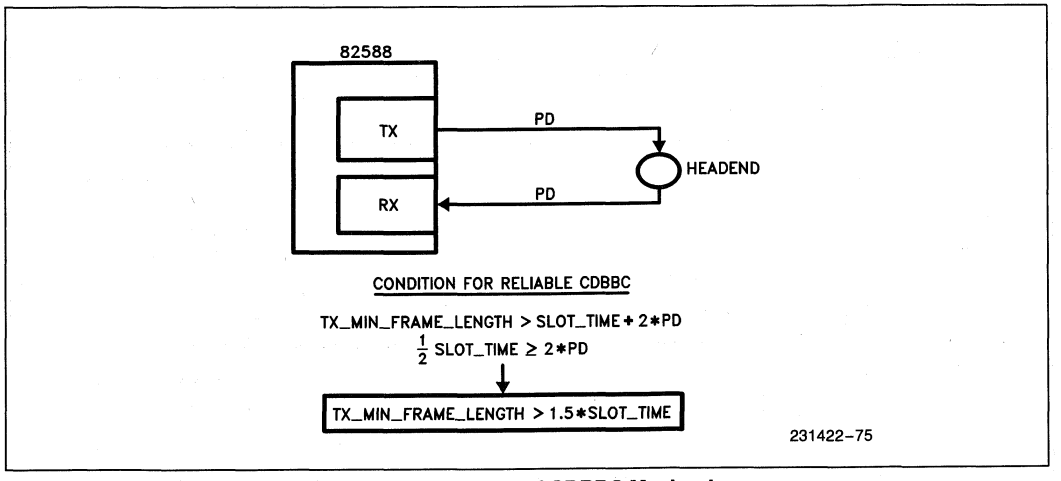


Figure 16. Limitation of CDBBC Mechanism

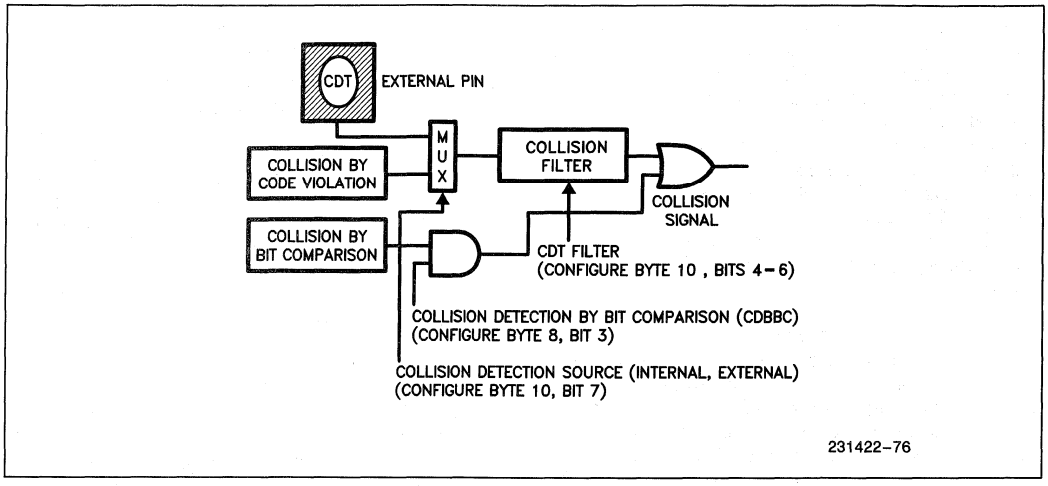


Figure 17. Mode 0, Collision Detection

3.7 Squelching the Input

Squelch circuit is used to filter idle noise on the receiver input. Basically two types of squelch may be used: Voltage and time. Voltage squelch is done to filter out signals whose strength is below a defined voltage threshold (0.6 volts for StarLAN). It prevents idle line noise from disturbing the receive circuits on the controller. The voltage squelch circuit is placed right after the receiving pulse transformer. It enables the input to the RxD pin of the 82588 only when the signal strength is above the threshold.

If the signal received has the proper level but not the proper timing, it should not bother the receiver. This is accomplished by the time squelch circuit on the 82588. Time squelching is essential to weed out spikes, glitches and bad signal especially at the beginning of a frame. The 82588 does not turn on its carrier sense (or receive enable) signal until it receives three consecutive edges, each separated by time periods greater than the fast time clock high time but less than 13/8 bit-times as shown in Figure 18.

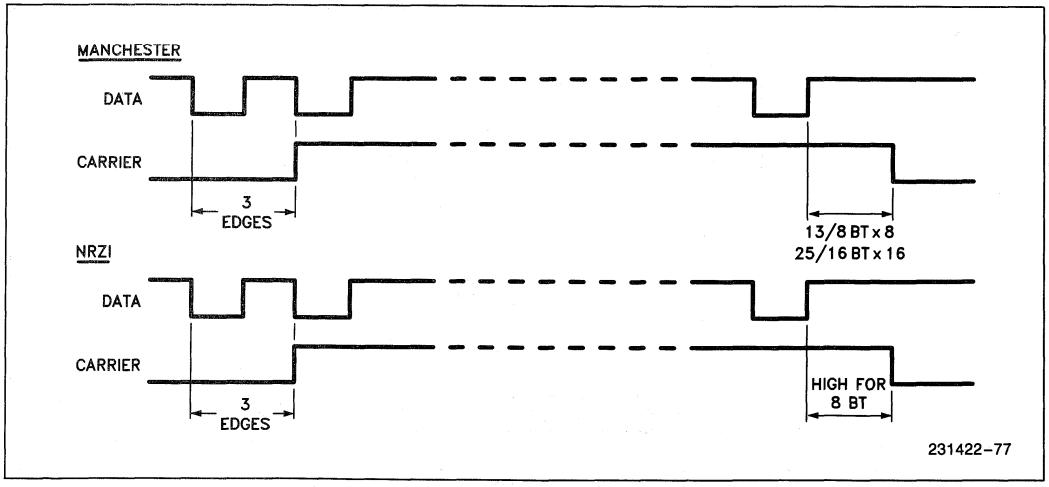


Figure 18. Carrier Sensing

The carrier sense activation can be programmed for a further delay by up to 7 bit times by a configuration parameter called carrier sense filter.

3.8 System Bus Interface

The 82588 has a conventional bus interface making it very easy to interface to any processor bus. Figure 19 shows that it has an 8 bit data bus, read, write, chip select, interrupt and reset pins going to the processor bus. It also needs an external DMA controller for data transfer. A system clock of up to 8 MHz is needed. The read and write access times of the 82588 are very short—95 ns—as shown by Figure 20. This further facilitates interfacing the controller to almost any processor.

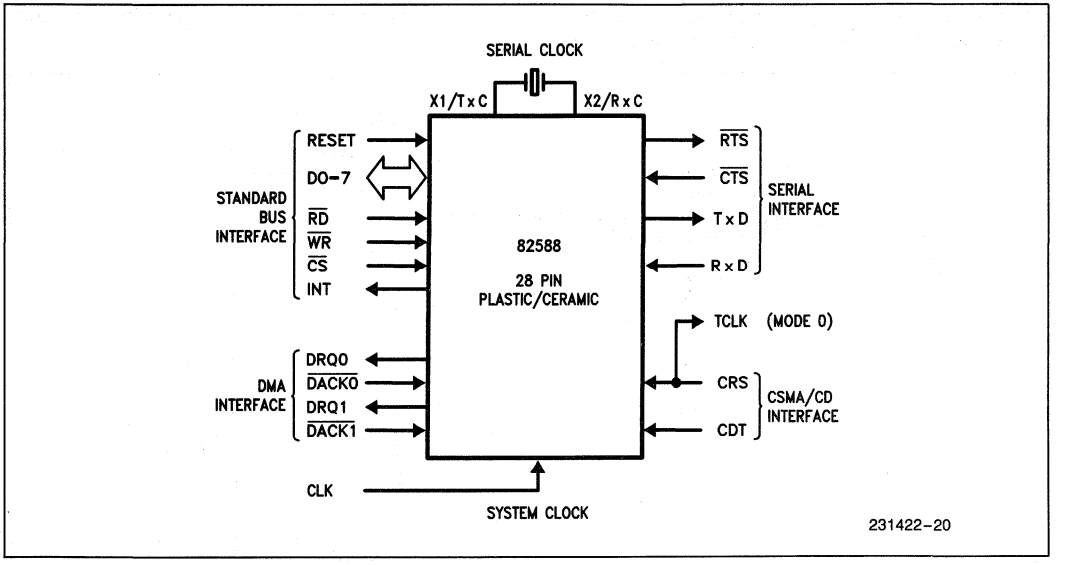


Figure 19. Chip Interface

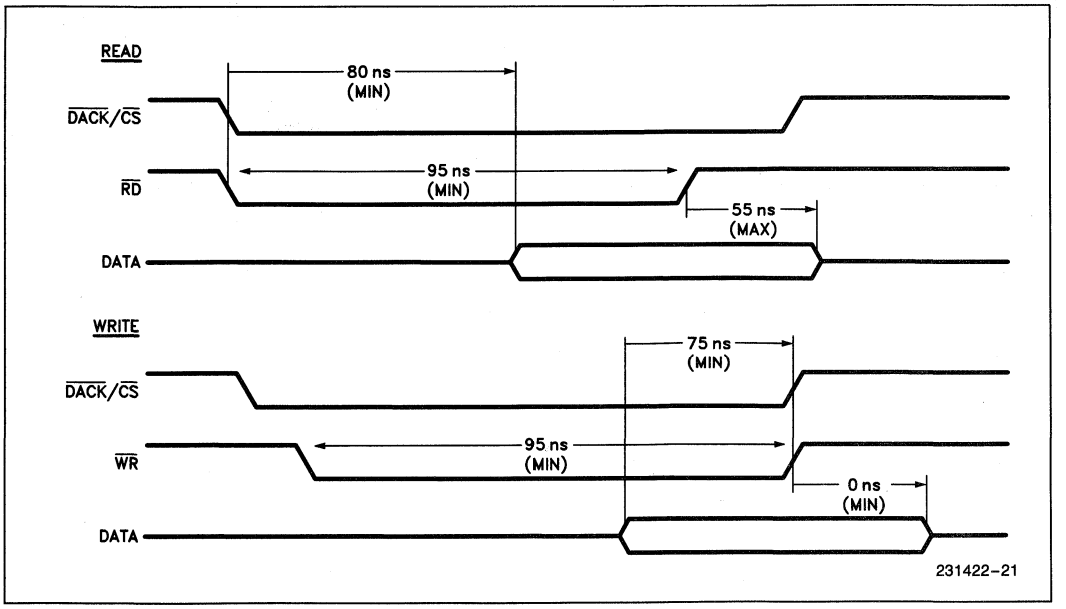


Figure 20. Access Times

The 82588 has over 50 bytes of registers, and most are accessed only indirectly. Figure 21 shows the register access mechanism of the 82588. It has one I/O port and 2 DMA channel ports. These are the windows into the 82588 for the CPU and the DMA controller. An external CPU can write into the Command register and read from the Status registers using I/O instructions and asserting chip select and write or read lines. Although there is just one I/O port and 4 status registers, they can be read out in a round robin fashion through the same port as shown in Figure 22. Other registers like the Configuration, Individual Address registers can be accessed only through DMA. All the internal registers can be dumped into memory by DMA using the Dump command. The execution of some of the commands is described in section 4. See the 82588 Reference Manual for details on these commands.

3.9 Debug and Diagnostic Aids

Besides the standard functions that can be used directly for StarLAN, the 82588 offers many debug and diag-

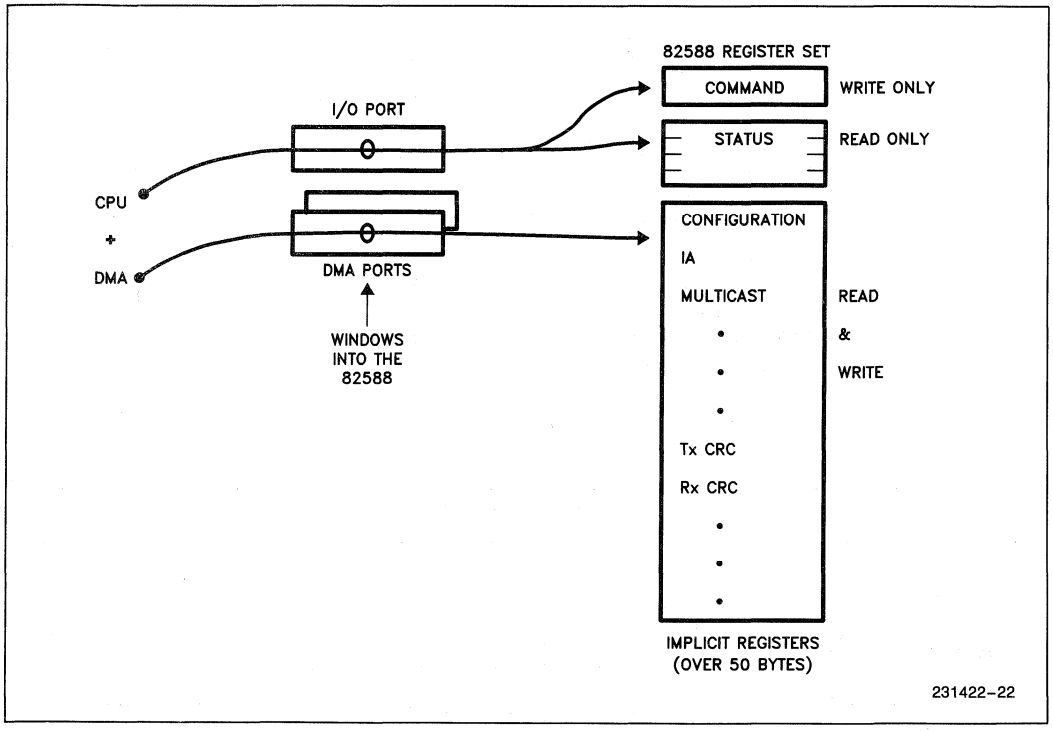
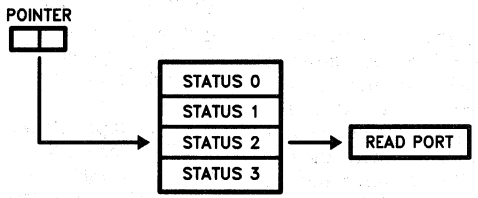


Figure 21. Register Access

4 Status registers are accessed through one read port



231422-23

The pointer can be changed using a command or can be automatically incremented.

```
READ_STATUS_588: PROCEDURE;                    /* COMMAND 15 */
     OUTPUT (CS_588) = 15;                  /* RELEASE POINTER, INITIAL = 00 */
     STATUS_588(0)=INPUT (CS_588);          /* REFRESH STATUS REGISTER IMAGE */
     STATUS_588(1)=INPUT (CS_588);          /* IN MEMORY.
     STATUS_588(2)=INPUT (CS_588);
     STATUS_588(3)=INPUT (CS_588);
     RETURN
END READ_STATUS_588;
```

READING 4 STATUS REGISTERS

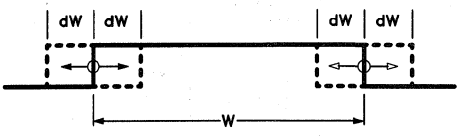
Figure 22. Reading the Status Register

nostics functions. The DIAGNOSE command of the 82588 does a self-test of most of the counters and timers in the 82588 serial unit. Using the DUMP command, all the internal registers of the 82588 can be dumped into the memory. The TDR command does Time Domain Reflectometery on the network. The 82588 has two loopback modes of operation. In the internal loopback mode, the TXD line is internally connected to the RXD one. No data appears outside the chip, and the 82588 is isolated from the link. This mode enables checking of the receive and transmit machines without link interference. In the external loopback mode, the 82588 becomes a full duplex device, being able to receive its own transmitted frames. In this mode data goes through the link and all CSMA/CD mechanisms are involved.

3.10 Jitter Performance

When the 82588 receives a frame from the HUB, the signal has jitter. Jitter is the shifting of the edges of the signal from their nominal position due to the transmission over a length of cable. Many factors like, intersymbol interference (pulses of different widths have different delays through the transmission media), rise and fall times of drivers and receivers, cross talk etc., contribute to the jitter. StarLAN specifies a maximum jitter of ±62.5 ns whenever the signal goes from a NODE/HUB or HUB/HUB. Figure 23 shows that the jitter tolerance of the 82588 is exactly the required ±62.5 ns at 1 Mbs for both 8X, 16X Manchester encoded data.

Jitter = ± variation of an edge from its nominal position.
Jitter can occur on every edge.



231422-78

	x8	x16
Manchester	±1/16	±1/16
NRZI (Code Violations Enabled)	±1/16 BT	±3/32 BT
NRZI (Code Violations Disabled)	±3/16 BT	±3/16 BT

Figure 23. 82588 Jitter Performance

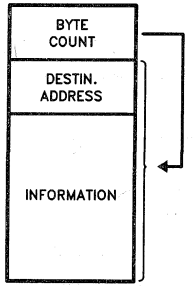
4.0 THE 82588

This chapter describes the basic 82588 operations. Please refer to the 82588 reference manual in Intel Microcommunications Handbook for a detailed description. Basic operations like transmitting a frame, receiving a frame, configuring the 82588 and dumping the register contents are discussed here to give a feel for how the 82588 works.

4.1 Transmit and Retransmit Operations

To transmit a frame, the CPU prepares a block in the memory called the transmit data block. As shown in Figure 24, this block starts with a byte count field, indicating how long the rest of the block is. The destination address field contains the node address of the destination. The rest of the block contains the information or the data field of the frame. The CPU also programs the DMA controller with the start address of the transmit data block. The DMA byte count must be equal to or greater than the block length. The 82588 is then issued a TRANSMIT command—an OUT instruction to the command port of the 82588. The 82588 starts generating DMA requests to read in the transmit data block by DMA. It also determines whether and how long it must defer on the link and after that, it starts transmitting the preamble. The 82588 constructs the frame on the fly. It takes the destination address from the memory, source address from its own individual address memory (previously programmed), data field from the memory and the CRC, is generated on chip, at the end of the frame.

1. Prepare Transmit Data—Block in Memory
2. Program DMA Controller
3. Issue Transmit Command on the Desired Channel



231422-25

Transmit Data Block

4. Interrupt is received on completion of command or if the command was aborted or there was a collision. The status bytes 1 and 2 indicate the result of the operation.

7	6	5	4	3	2	1	0	
TX DEF	HRT BEAT	MAX COLL		NUM. OF COLLISIONS				STATUS 1
COLL		TX OK			LOST CRS	LOST CTS	UNDER RUN	STATUS 2

231422-26

Transmit & Retransmit Results Format

Figure 24. Transmit Operation

At the conclusion of transmission the 82588 generates an interrupt to the CPU. The CPU can then read the status registers to find out if the transmission was successful. If a collision occurs during transmission, the 82588 aborts transmission and generates the jam sequence, as required by IEEE 802.3, and informs the CPU through interrupt and the status registers. It also starts the back-off algorithm.

To re-attempt transmission, the CPU must reinitialize the DMA controller 7to the start of the transmit data block and issue a RETRANSMIT command to the 82588. When the 82588 receives the retransmit command and the back-off timer has expired, it transmits again. Interrupt and the status register contents again indicate the success or failure of the (re)transmit attempt.

The main difference between transmit and retransmit commands is that retransmit does not clear the internal count for the number of collisions occurred, whereas transmit does. Moreoever, retransmit takes effect only when the back-off timer has expired.

4.2 Configuring the 82588

To initialize the 82588 and program its network and system parameters, a configure operation is performed. It is very similar to the transmit operation. Instead of a transmit data block as in transmit command, a configure data block—shown in Figure 12—is prepared by the CPU in the memory. The first two bytes of the block specify the length of the rest of the block, which specify the network and system parameters for the 82588. The DMA controller is then programmed by the CPU to the beginning of this block and a CONFIGURE command is issued to the 82588. The 82588 reads in the parameters by DMA and loads the parameters in the on-chip registers.

Similarly, for programming the INDIVIDUAL ADDRESS and MULTICAST ADDRESSes, the DMA controller is used to load the 82588 registers.

4.3 Frame Reception

Before enabling the 82588 for reception the CPU must make a buffer available for the frame to be received. The CPU must program the DMA controller with the starting address of the buffer and then issue the RX_ ENABLE command to the 82588. When a frame arrives at the RxD pin of the 82588, it starts being received. Only if the address in the destination address matches either the Individual address, Multicast address or if it is a broadcast address, is the frame deposited into memory by the 82588 using DMA. The format of storage in the memory is shown in Figure 25. At the end, a two byte field is attached which shows the status of the received frame. If CRC, alignment or overrun errors are encountered, they are reported. An inter-

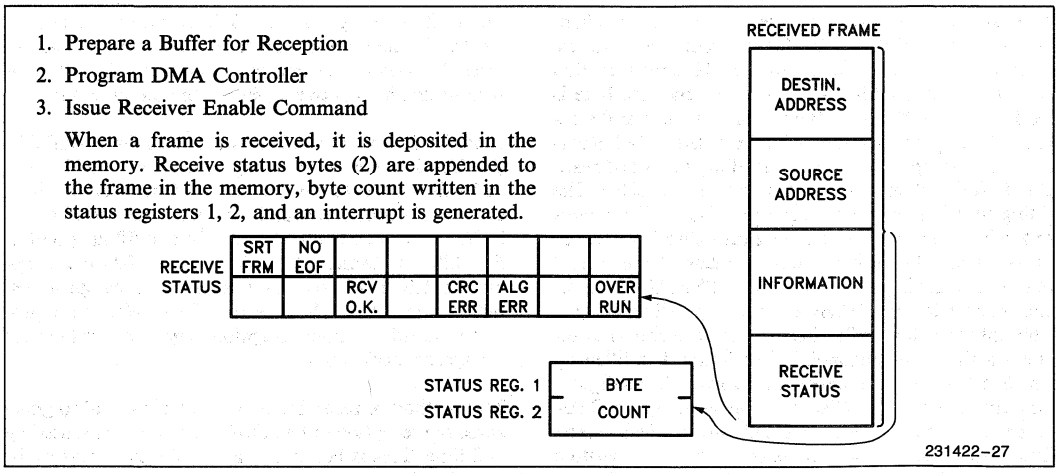


Figure 25. Receive Operation (Single Buffer)

rupt from 82588 occurs when all the bytes have been transferred to the memory. This informs the CPU that a new frame has been received.

If the received frame has errors, the CPU must recover (or re-use) the buffer. Note that the entire frame is deposited into one buffer. The 82588 when NOT configured for the external loopback mode, will detect collisions (code violations) during receptions. If a collision is detected, the reception is aborted and status updated. CPU is then informed by an interrupt (if the collided frame fragment is shorter than the address length, no reception will be started), and no interrupt will happen.

4.3.1 Multiple Buffer Frame Reception

It is also possible to receive a frame into a number of fixed size buffers. This is particularly economical if the received frames vary widely in size. If the single buffer scheme were used as described above, the buffer required would have to be bigger than the longest expected frame and would be very wasteful for very short (typically acknowledge or control) frames. The multiple buffer reception is illustrated in Figure 26. It uses two DMA channels for reception.

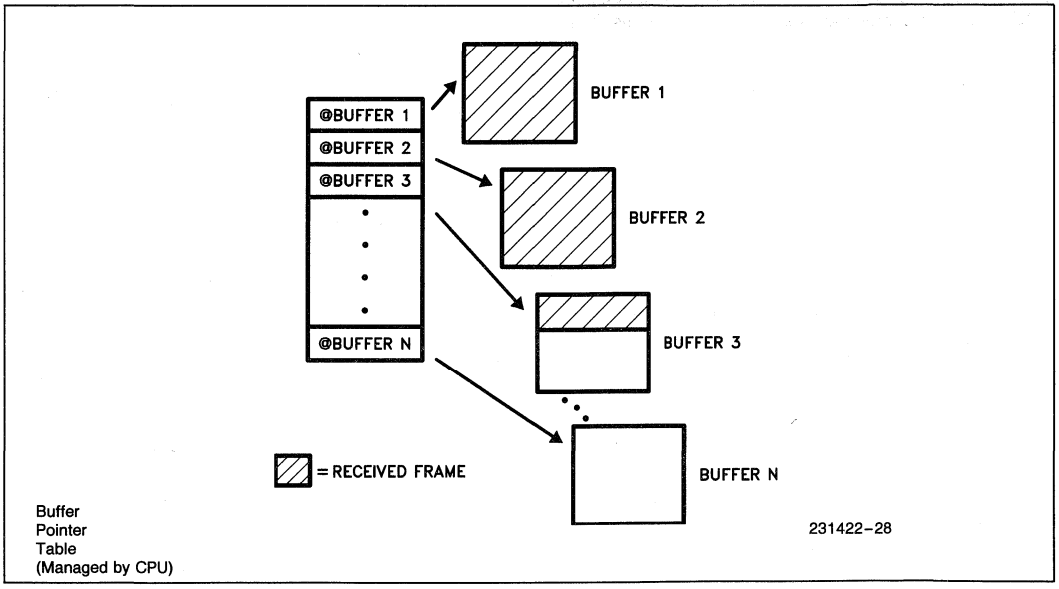


Figure 26. Multiple Buffer Reception

As in single buffer reception, the one channel, say channel 0, of the DMA controller is programmed to the start of buffer 1, and the 82588 is enabled for reception with the chaining bit set. As soon as the first byte is read out of the 82588 by the DMA controller and written into the first location of buffer 1, the 82588 generates an interrupt, saying that it is filling up its last available buffer and one more buffer must be allocated. The filling up of the buffer 1 continues. The CPU responds to the interrupt by programming the other DMA channel—channel 1—with the start address of the second buffer and issuing an ASSIGN ALTERNATE buffer command with an INTACK (interrupt acknowledge). This informs the 82588 that one more buffer is available on the other channel. When buffer 1 is filled up (the 82588 knows the size of buffers from the configuration command), the 82588 starts generating the DMA requests on the other channel. This automatically starts filling up buffer 2. As soon as the first byte is written into buffer 2, the 82588 interrupts the CPU again asking for one more buffer. The CPU programs the channel 0 of the DMA controller with the start address of buffer 3, issues an ASSIGN ALTERNATE buffer command with INTACK. This keeps the buffer 3 ready for the 82588. This switching of channels continues until the entire frame is received generating an end of frame interrupt. The CPU maintains the list of pointers to the buffers used.

Since a new buffer is allocated at the time of filling up of the last buffer, the 82588 automatically switches to the new buffer to receive the next frame as soon as the last frame is completely received. It can start receiving the new frame almost immediately, even before the end of frame interrupt is serviced and acknowledged by the CPU. If a new frame comes in, and the previous frame interrupt is not yet acknowledged, another interrupt needed for new buffer allocation is buffered (and not lost). As soon as the first one is acknowledged, the interrupt line goes active again for the buffered one.

If by the time a buffer fills up no new buffer is available, the 82588 keeps on receiving. An overrun will occur and will be reported in the received frame status. However, ample time is available for the allocation of a new buffer. It is roughly equal to the time to fill up a buffer. For 128 byte buffers it is $128 \times 8 = 1024$ ms or approximately 1 millisec. You get 1 ms to assign a new buffer after getting the interrupt for it. Hence the process of multiple buffer reception is not time critical for the system performance.

This method of reception is particularly useful to guarantee the reception of back-to-back frames separated by IFS time. This is because a new buffer is always available for the new frame after the current frame is received.

Although both the DMA channels get used up in receiving, only one channel is kept ready for reception and the other one can be used for other commands until the reception starts. If an execution command like transmit or dump command is being executed on a channel which must be allocated for reception, the command gets automatically aborted when the ASSIGN ALTERNATE BUFFER command is issued to the channel used for the execution command. The interrupt for command abortion occurs after the end of frame interrupt.

4.4 Memory Dump of Registers

All the 82588 internal registers can be dumped in the memory by the DUMP command. A DMA channel is used to transfer the register contents to the memory. It is very similar to reception of a frame; instead of data from the serial link, the data from the registers gets written into the memory. This provides a software debugging and diagnostic tool.

4.5 Other Operations

Other 82588 operations like DIAGNOSE, TDR, ABORT, etc. do not require any parameter or data transfer. They are executed by writing a command to the 82588 command register and knowing the results (if any) through the status registers.

5.0 StarLAN NODE FOR IBM PC

This chapter deals with the hardware—the StarLAN board—to interface the IBM PC to a StarLAN Network. This is a slave board which takes up one slot on the I/O channel of the IBM PC. Figure 27 shows an abstract block diagram of the board. It requires the IBM PC resources of the CPU, memory, DMA and interrupt controller on the system board to run it. Such a board has two interfaces. The IBM PC I/O Channel on the system or the parallel side and the telephone grade twisted pair wire on the serial side. Figures 28, 29 show the circuit diagram of the board.

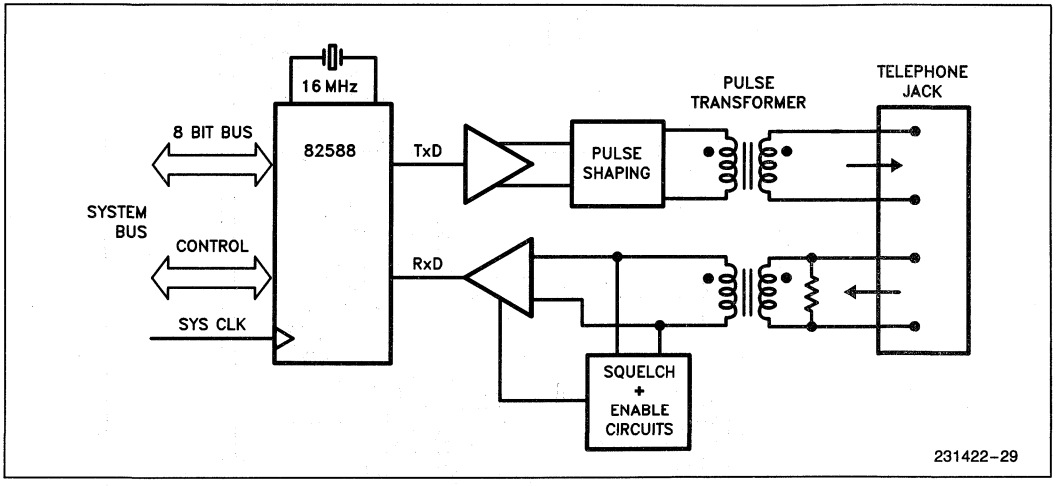


Figure 27. 82588 Based StarLAN Node

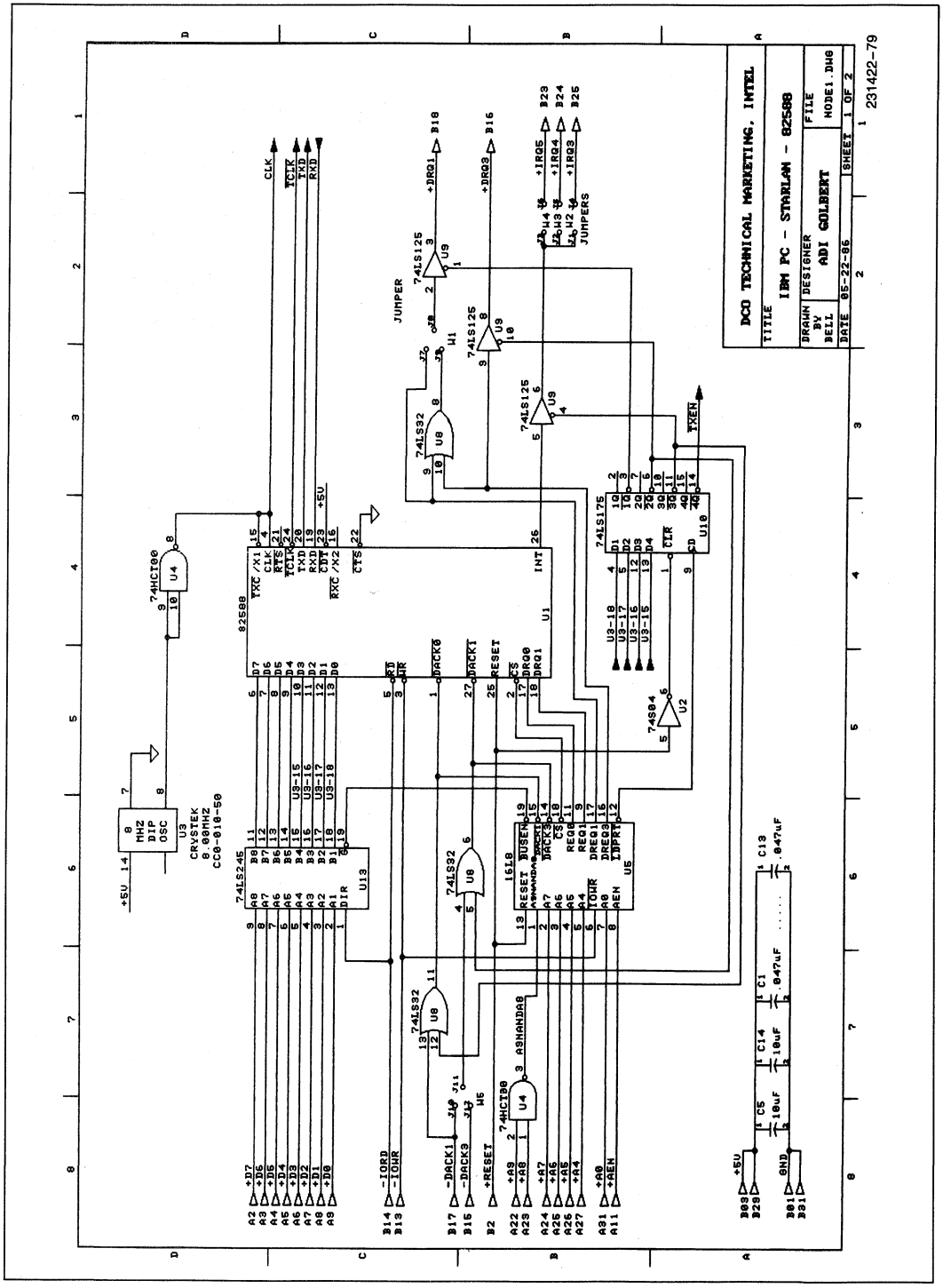

DCO TECHNICAL MARKETING, INTEL
TITLE
IBM PC - STARLAN - 82588
DRAWN BY BELL
DESIGNER ADI GOLBERT
FILE NODE1.DWG
DATE 05-22-86
SHEET 1 OF 2
231422-79
82588
74LS245
74LS32
74HCT00
74LS125
74LS175
74S04
16I8
CRYSTEK 8.00MHZ CC0-010-50
JUMPER
JUMPERS
CLK
TCLK
TXD
RXD
+5V
GND
-IORD
-IOWR
-DACK1
-DACK3
+RESET
ASHANDA8
+IRQ5
+IRQ4
+IRQ3
+DRQ1
+DRQ0
TXEN

Figure 28

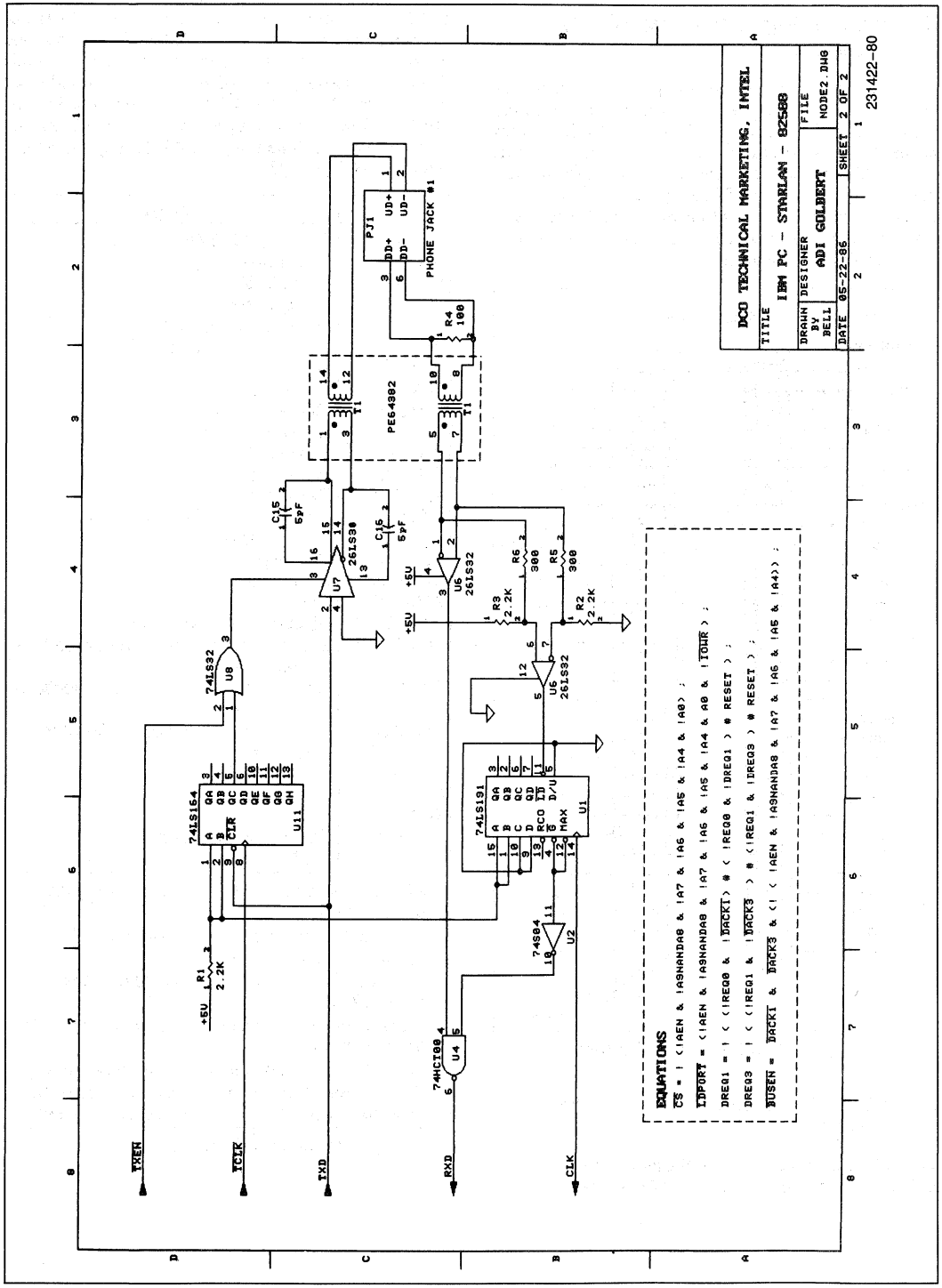

DCD TECHNICAL MARKETING, INTEL
TITLE
IBM PC - STARLAN - 82588
DRAWN BY BELL
DESIGNER ADI GOLDERT
FILE NODE2.DWG
DATE 05-22-86
SHEET 2 OF 2
231422-80
EQUATIONS
CS = ! (!AEN & !A9NAND8 & !A7 & !A6 & !A5 & !A4 & !A3) ;
IPORT = (!AEN & !A9NAND8 & !A7 & !A6 & !A5 & !A4 & A8 & !IOWR) ;
DREQ1 = ! ((!REQ0 & !DACK1) # (!REQ1 & !DREQ1) # RESET) ;
DREQ3 = ! ((!REQ1 & !DACK3) # (!REQ3 & !A7 & !A6 & !A5 & !A4)) ;
BUSEN = DACK1 & DACK3 & (! (!AEN & !A9NAND8 & !A7 & !A6 & !A5 & !A4)) ;
PJ1
PHONE JACK #1
T1
PE64902
U7 26LS30
U6 26LS32
U5 26LS32
U8 74LS32
U11 74LS164
U1 74LS191
U2 74S04
U4 74HCT00
R1 2.2K
R2 2.2K
R3 2.2K
R4 100
R5 300
R6 300
C15 5pF
C16 5pF
+5V
TXEN
TCLK
TXD
RXD
CLK

Figure 29

5.1 Interfacing to the IBM PC I/O Channel

IBM PC has 8 slots on the system board to allow expansion of the basic system. All of them are electrically identical and the I/O channel is the bus that links them all to the 8088 system bus. The I/O channel contains an 8 bit bidirectional data bus, 20 address lines, 6 levels of interrupt, 3 channels of DMA control lines and other control lines to do I/O and memory read/write operations. Figure 30 shows the signals and the pin assignment for the I/O Channel.

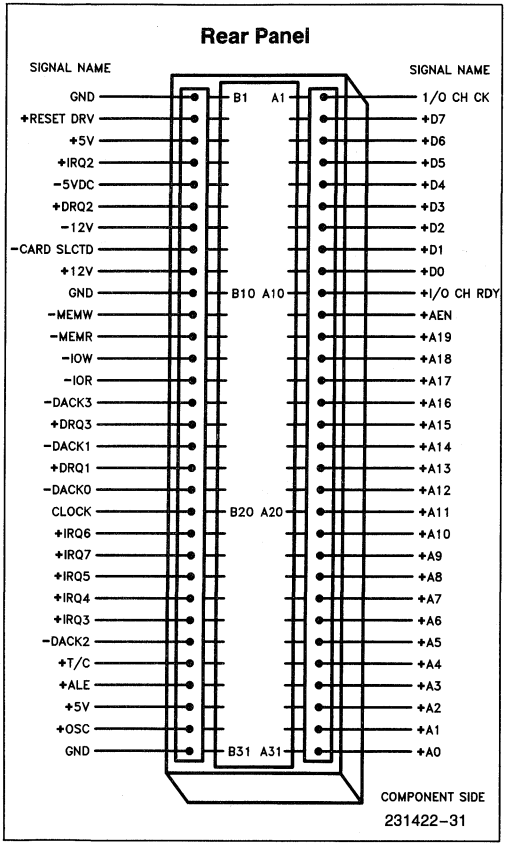


Figure 30. I/O Channel Diagram

5.1.1 REGISTER ACCESS AND DATA BUS INTERFACE

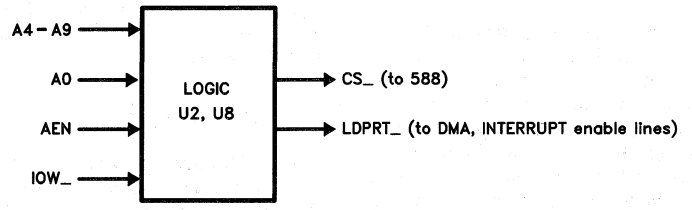
The CPU accesses the StarLAN adapter card through 2 I/O address windows. Address 300H is used to access to 82588 for commands and status, address 301H accesses an on board control port that enables the various interrupt and DMA lines. Even though only two addresses are needed, the card uses all the 16 addresses spaces from 300H to 30FH. This was done to keep simplicity and minimum component count. Registers address decoding is done using a PAL (16L8) and an external NAND gate (U8).

Hex Range	Usage
000-00F	DMA Chip 8237A-5
020-021	Interrupt 8259A
040-043	Timer 8253-5
060-063	PPI 8255A-5
080-083	DMA Page Registers
0AX*	NMI Mask Register
0CX	Reserved
0EX	Reserved
200-20F	Game Control
210-217	Expansion Unit
220-24F	Reserved
278-27F	Reserved
2F0-2F7	Reserved
2F8-2FF	Asynchronous Communications (Secondary)
300-31F	Prototype Card
320-32F	Fixed Disk
378-37F	Printer
380-38C**	SDLC Communications
380-389**	Binary Synchronous Communications (Secondary)
3A0-3A9	Binary Synchronous Communications (Primary)
3B0-3BF	IBM Monochrome Display/Printer
3C0-3CF	Reserved
3D0-3DF	Color/Graphics
3E0-3E7	Reserved
3F0-3F7	Diskette
3F8-3FF	Asynchronous Communications (Primary)

* At power-on time, the Non Mask Interrupt into the 8088 is masked off.
This mask bit can be set and reset through system software as follows:
Set mask: Write hex 80 to I/O Address hex A0 (enable NMI)
Clear mask: Write hex 00 to I/O Address hex A0 (disable NMI)

** SDLC Communications and Secondary Binary Synchronous Communications cannot be used together because their hex addresses overlap.

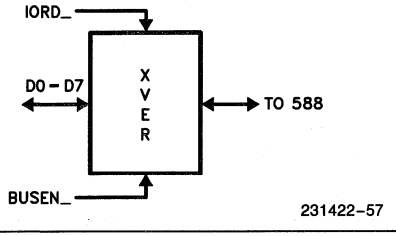
Figure 31. I/O Address Map



Register Access

```
Format of Following Equations Will Be According To
The Following Specifications:
    !  INVERT
    _  SIGNAL ACTIVE LOW
    &  LOGIC AND
    #  LOGIC OR
A9NANDA8 = ! (A9 & A8)
CS_ = ! ( !AEN & !A9NANDA8 & !A7 & !A6 & !A5 & !A4 & !A0 )
LDPORT_ = ! ( !AEN & !A9NANDA8 & !A7 & !A6 & !A5 & !A4 & A0 & !IOWR_ )
BUSEN_ = DACK1_ & DACK2_ & (! ( !AEN & !A9NANDA8 & !A7 & !A6 & !A5 & !A4));
```

The signal CS__ decodes address 300H, it is only active when AEN is inactive meaning CPU and not DMA cycles. LDPORT__ has exactly the same logic for address 301H, but it is only active during I/O write cycles. The I/O port sitting on address 301H is write only. The data BUS lines D0 to D7 are buffered from the 82588 to the PC bus using an 74LS245 transceiver chip.



Data Bus Interface

The Bus transceiver is enabled if: A DMA access is taking place, or I/O ports 300H to 30FH are being accessed.

5.1.2 Control Port

As mentioned the StarLAN adapter port has a 4-bit write only control port. The purpose of this port is to selectively enable the DMA and INTERRUPT request lines. Also it can completely disable the transmitter.

Control Port Definition

ENDRQ1	ENDRQ3	ENINTER	TXEN

ENDRQ1, ENDRQ2 : "1" Enable DMA requests.
ENINTER : "1" Enable INTERRUPT request.
TXEN : "1" Enable the transmitter.

On power up all bits default to "0".

5.1.3 CLOCK GENERATION

The 82588 requires two clocks for operation. The system clock and the serial clock. The serial clock can be generated on chip by putting a crystal across X1 and X2 pins. Alternatively, an externally generated clock can be fed in at pin X1 (with X2 left open). In both cases, the frequency must be either 8 or 16 times (sampling factor) the desired bit rate. For StarLAN, 8 or 16 MHz are the correct values to generate 1 Mb/s data rate. A configuration parameter is used to tell the 82588 what the sampling factor is. An externally supplied clock must have MOS levels (0.6V–3.9V). Specifications for the crystal and the circuit are shown in Figure 32.

The system clock has to be supplied externally. It can be up to 8 MHz. This clock runs the parallel side of the 82588. Its frequency does not have any impact on the read and write access times but on the rate at which data can be transferred to and from the 82588 (Maximum DMA data rate is one byte every two system clocks). This clock doesn't require MOS levels.

The I/O channel of the IBM PC supplies a 4.77 MHz signal of 33% duty cycle. This signal could be used as a system clock. It was decided, however, to generate a separate clock on the StarLAN board to be independent of the I/O channel clock so that this board can also be used in other IBM PCs and also in some other compatibles. The 8 MHz system clock is generated using a DIP OSCILLATOR which have the required 50 ppm tolerance to meet StarLAN. This clock is converted to MOS levels by 74HCT00 and fed into both the system and serial clock inputs.

5.1.4 DMA INTERFACE

The 82588 requires either one or two DMA channels for full operation. In this application, one channel is dedicated for reception and the other is used for transmissions and the other commands. Use of only one DMA channel is possible but may require more complex software, also some RX frames may be lost during switches of the DMA channel from the receiver to the transmitter (Those frames will be recovered by higher layers of the protocol). Also using only one DMA channel will limit the 82588 loopback functionality. So the recommendation is to operate with two DMA channels if available. Appendix C describes a method of operating with only one DMA channel without loosing RX frames.

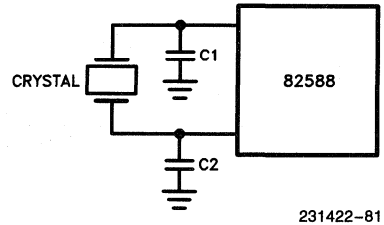
The IBM PC system board has one 8237A DMA controller. Channel 0 is used for doing the refresh of DRAMs. Channels 1, 2 and 3 are available for add-on boards on the I/O Channel. The floppy disk controller board uses the DMA channel 2 leaving exactly two channels (1 and 3) for the 82588. The situation is worse if the IBM PC/XT is used, since it uses channel 3 for the Winchester hard disk leaving just the channel 1 for

Series Resonance

—Frequency Will Drift by About 400 PPM from Nominal
—No Capacitors Needed
—Doesn't Meet StarLAN Requirements

Meeting StarLAN 100 PPM Requirements

—Use Parallel Resonance Crystal
—Recommended For Precise Frequencies
—82588 X-TAL Oscillator Stability ±35 PPM (0–70°C)



Crystal: Load Capacitance = 20 pF
Shunt Capacitance = 7 pF Maximum
Series Resistance = 30Ω Maximum
Frequency Tolerance = 50 PPM (0–70°C)

C1, C2 → 27 pF or 39 pF, 5%

Figure 32. Crystal Specifications

the 82588. On the other hand, the IBM PC/AT has 5 free DMA channels. We will assume that 8237A DMA channels 1 and 3 are available for the 82588 as in the case of the IBM PC.

Since the channel 0 of 8237A is used to do refresh of DRAMs all the channels should be operated in single byte transfer mode. In this mode, after every transfer for any channel the bus is granted to the current highest priority channel. In this way, no channel can hog the bus bandwidth and, more important, the refresh of DRAMs is assured every 15 microseconds since the refresh channel (number 0) has the highest priority. This mode of operation is very slow since the HOLD is dropped by the 8237A and then asserted again after every transfer. Demand mode of operation is a lot more suitable to 82588 but it cannot be used because of the refresh requirements.

Whenever the 82588 interfaces to the 8237A in the single transfer mode, there is a potential 8237A lock-up problem. The 82588 may deactivate its DMA request line (DREQ) before receiving an acknowledge from the DMA controller. This situation may happen during command abortions, or aborted receptions. The 8237A under those circumstances may lock-up. In order to solve this potential problem, an external logic must be used to insure that DREQ to the DMA controller is never deactivated before the acknowledge is received. Figure 33 shows the logic to implement this function. This logic is implemented in the 16L8 PAL.

The 82588 DREQ lines are connected to the IBM/PC bus through tri-state buffers which are enabled by writing to I/O port 301H. This function enables the use of either one or two DMA channels and also the sharing of DMA channels with other adapter boards.

5.1.5 INTERRUPT CONTROLLER

The 82588 interrupts the CPU after the execution of a command or on reception of a frame. It uses the 8259A interrupt controller on the system board to interrupt the CPU. There are 6 interrupt request lines, IRQ2 to IRQ7, on the I/O channel. Figure 34 shows the assignment of the lines. In fact, none of the lines are completely free for use. To add any new peripheral which uses a system board interrupt, this interrupt needs to have the capability to share the specific line, by driving the line with a tri-state driver. The 82588 StarLAN adapter board can optionally drive interrupt lines IRQ3, IRQ4 or IRQ5 (An 74LS125 driver is used).

Number	Usage
NMI	Parity
0	Timer
1	Keyboard
2	Reserved
3	Asynchronous Communications (Secondary) SDLC Communications BSC (Secondary)
4	Asynchronous Communications (Primary) SDLC Communications BSC (Primary)
5	Fixed Disk
6	Diskette
7	Printer

Figure 34. IBM PC Hardware Interrupt Listing

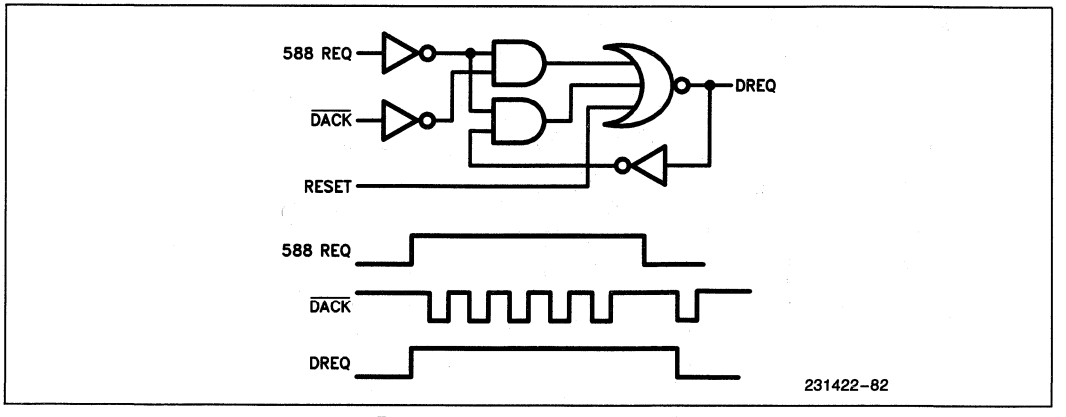


Figure 33. DMA Request Logic

5.2 Serial Link Interface

A typical StarLAN adapter board is connected to the twisted pair wiring using an extension cable (typically up to 8 meters—25 ft.). See Figure 35. One end of the cable plugs into the telephone modular jack on the StarLAN board and the other end into a modular jack in the wall. The twisted pair wiring starts at the modular jack in the wall and goes to the wiring closet. In the wiring closet, another telephone extension cable is used to connect to a StarLAN HUB. The transmitted signal from the 82588 reach the on-board telephone jack through a RS-422 driver with pulse shaping and a pulse transformer. The received signals from the telephone jack to the 82588 come through a pulse transformer, squelch circuit and a receive enable circuit.

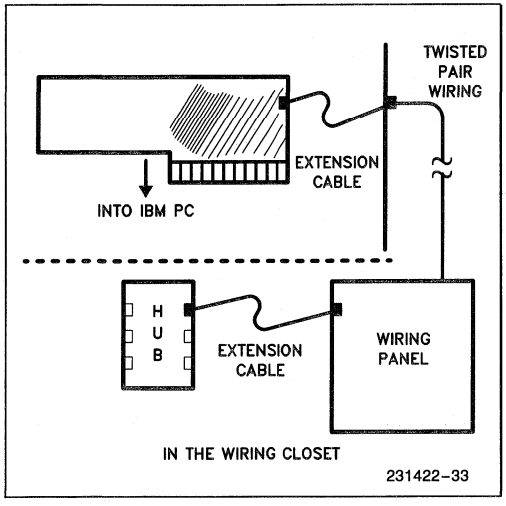


Figure 35. Path from StarLAN Board to HUB

5.2.1 TRANSMIT PATH

The single ended transmit signal on the TxD pin is converted to a differential signal and the rise and fall times are increased to 150 to 200 ns before feeding it to the pulse transformer (this pulse shaping is not a requirement, but proves to give good results). Am26LS30 is a RS-422 driver which converts the TxD signal to a differential signal. It also has slew rate control pins to increase to rise and fall times. A large rise and fall time reduces the possibility of crosstalk, interference and radiation. By the other hand a slower edge rate increases the jitter. In the StarLAN adapter card, the first approach was used. The 26LS30 converts a square pulse to a trapezoidal one—see Figure 36. The filtering effect of the cable further adds to reduce the higher frequency components from the waveform so that on the cable the signal is almost sinusoidal. The pulse transformer is for DC isolation. The pulse transformers from Pulse Engineering—type PE 64382—was used in this design. This is a dual transformer package which introduces an additional rise and fall time of about 70–100 ns on the signal, helping the former discussed waveshaping.

5.2.2 IDLE PATTERN GENERATION

StarLAN requires transmitters to generate an IDLE pattern after the last transmitted data bit. The IDLE pattern is defined to be a constant high level for 2–3 microseconds. The purpose of this pattern is to insure that receivers will decode properly the last transmitted data bits before signal decay. Currently the 82588 needs one external component to generate the IDLE. The operation principle is to have an external shift register (74LS164) that will kind of act as an envelope detector of the TXD line. Whenever the TXD line goes low

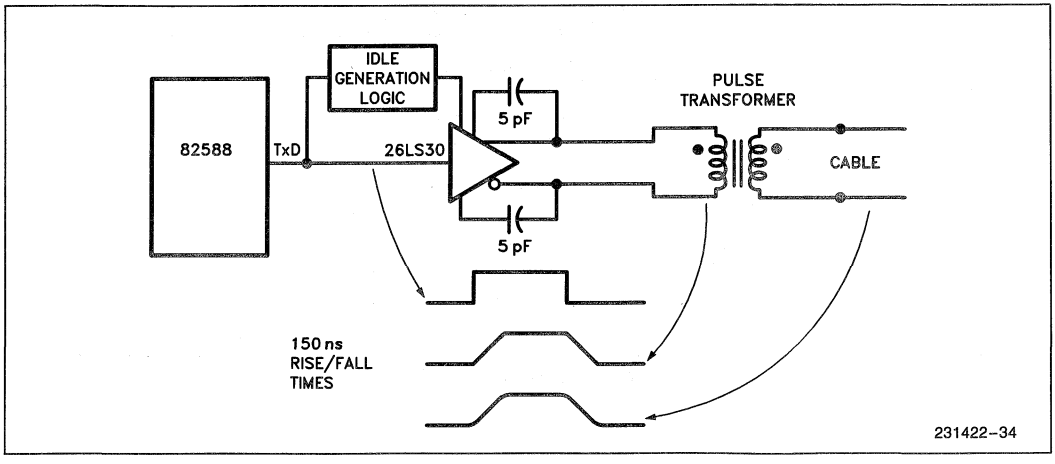


Figure 36. Wave Shaping

(first preamble bit), the output of the shift register (third cell) will immediately go low, enabling the RS-422 driver, the shift register being clocked by TCLK— will time the duration of the TXD high times. If the high time is more than 2 microseconds, meaning that the 82588 has gone idle, the transmitter will be disabled (See Figure 37). Another piece of this logic is the OR-ing of the output of the shift register with TXEN—signal which comes from the board control port. This signal completely disables the transmitter. The other purpose of this enable signal, is to make sure that after power-up, before the 82588 is configured, the RS-422 drivers won't be enabled (TCLK__ is not active before the configure command). See Figures 28, 29 for the complete circuit.

5.3 RECEIVE PATH

The signal coming from the HUB over the twisted pair wire is received on the StarLAN board through a 100Ω line termination resistor and a pulse transformer. The pulse transformer is of the same type as for the transmit side and its function is dc isolation. The received signal which is differential and almost sinusoidal is fed to the Am26LS32 RS-422 receiver. As seen from Figure 38 the pulse transformer feeds two RS-422 receivers. The one on the bottom is for squelch filtering and the one above is the real receiver which does real zero crossing detection on the signal and regenerates a square digital waveform from the sinusoidal signal that is received. Proper zero crossing detection is very essential; if the edges of the regenerated signal are not at zero crossings, the resulting signal may not be a proper Manchester encoded signal (self introduced jitter) even if the original signal is valid Manchester. The resistors in the lower receiver keep its differential inputs at a voltage difference of 600 mV. These bias resistors ensure that the output remains high as long as the input signal is more than −600 mV. It is very important that the RxD pin remains HIGH (not LOW or floating) whenever the receive line is idle. A violation of this may cause the 82588 to lock-up on transmitting. Remember, that based on the signal on the RxD pin, the 82588 extracts information on the data being received, Carrier Sense and Collision Detect. This squelch of 600 mV keeps the idle line noise from getting to the 82588. Figure 39 shows that when the differential input of the receiver crosses zero, a transition occurs at the output. It also shows that if the signal strength is higher than −600 mV, the output does not change. (This kind of squelching is called negative squelching, and it is done due to the fact that the preamble pattern starts with a going low transition). Note that the differential voltage at the upper receiver input is zero when the line is idle. The output of the squelch goes to a pulse stretcher which generates an envelope of the received frame. The envelope is a receive enable signal and is used to AND the signal from the real zero crossing receiver before feeding it to the RxD pin of the 82588.

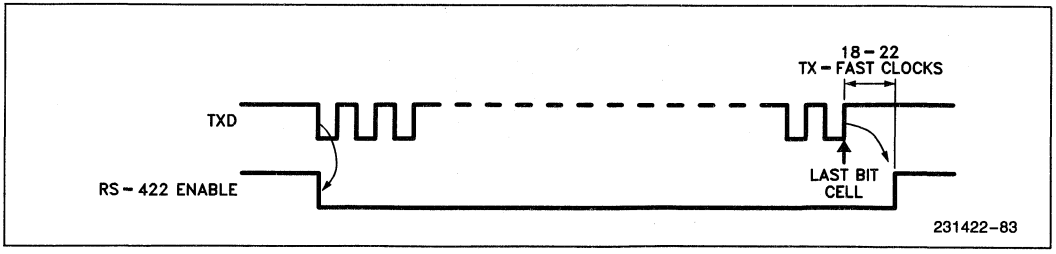


Figure 37. Idle Generation

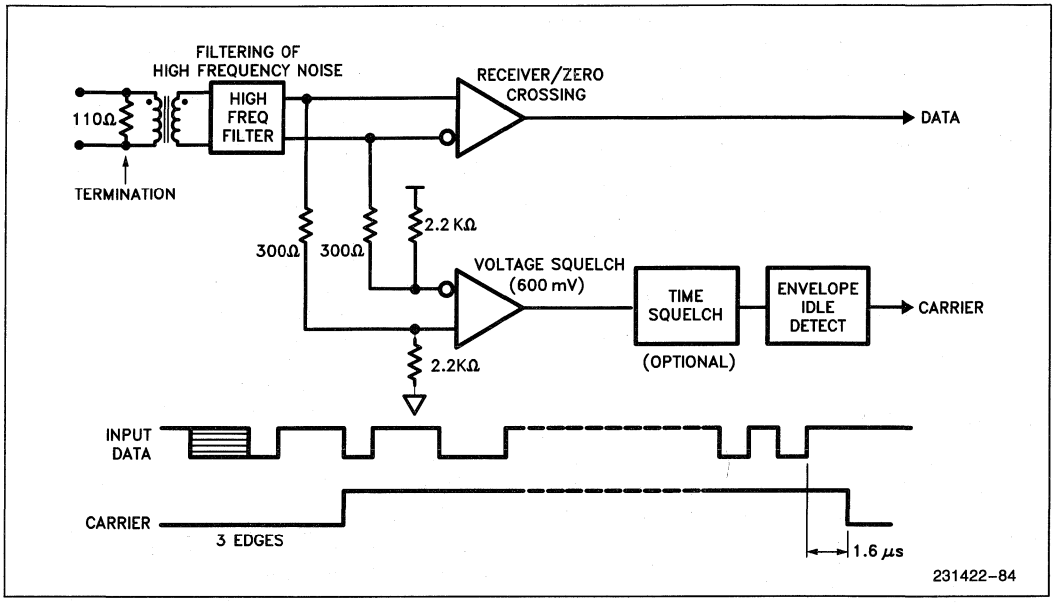


Figure 38. Input Ports

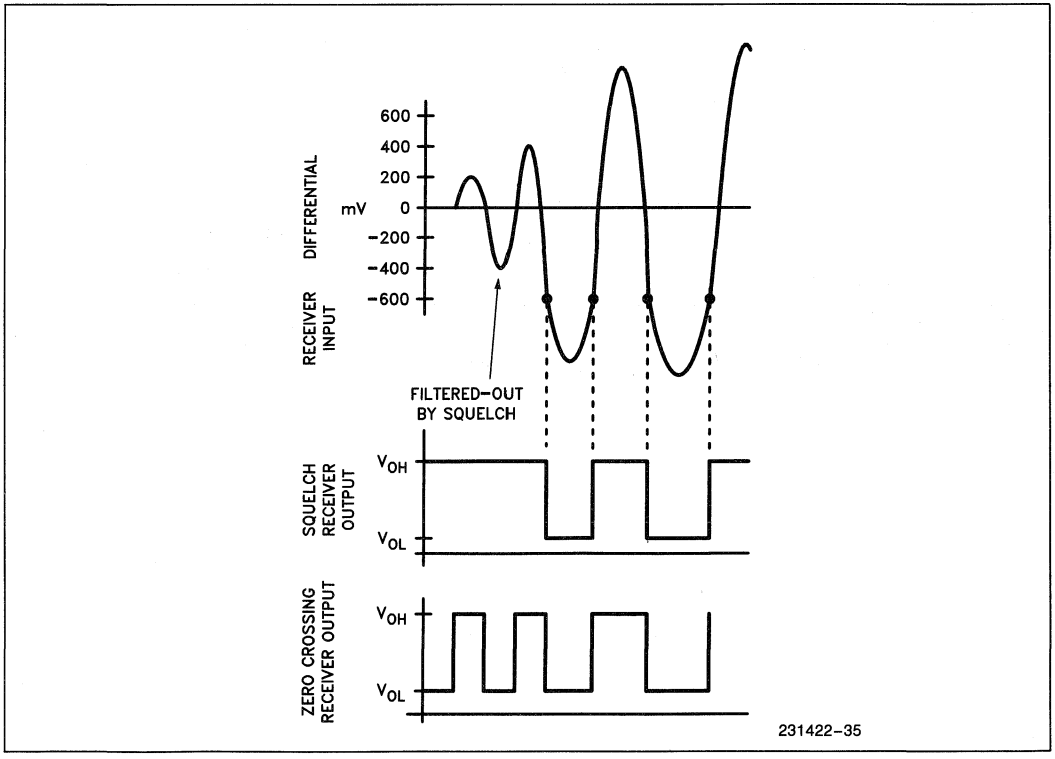


Figure 39. Squelch Circuit Output

5.4 80188 Interface to 82588

Although the 82588 interfaces easily to almost any processor, no processor offers as much of the needed functionality as the 80186 or its 8 bit cousin, the 80188. The 80188 is 8088 object code compatible processor with DMA, timers, interrupt controller, chip select logic, wait state generator, ready logic and clock generator functions on chip. Figure 40 shows how the 82588, in a StarLAN environment interfaces to the 80188. It uses the clock, chip select logic, DMA channels, interrupt controller directly from the 80188. The interface components between the CPU and the 82588 are totally eliminated.

5.5 iSBX Interface to StarLAN

Figure 41 shows how to interface the 82588 in a StarLAN environment to the iSBX bus. It uses 2 DMA channels—tapping the second DMA channel from a neighboring iSBX connector. Such a board can be used to make a StarLAN to an Ethernet or a SNA or DECNET gateway when it is placed on an appropriate SBC board. It may also be used to give a StarLAN access to any SBC board (with an iSBX connector) independent of the type of processor on the board.

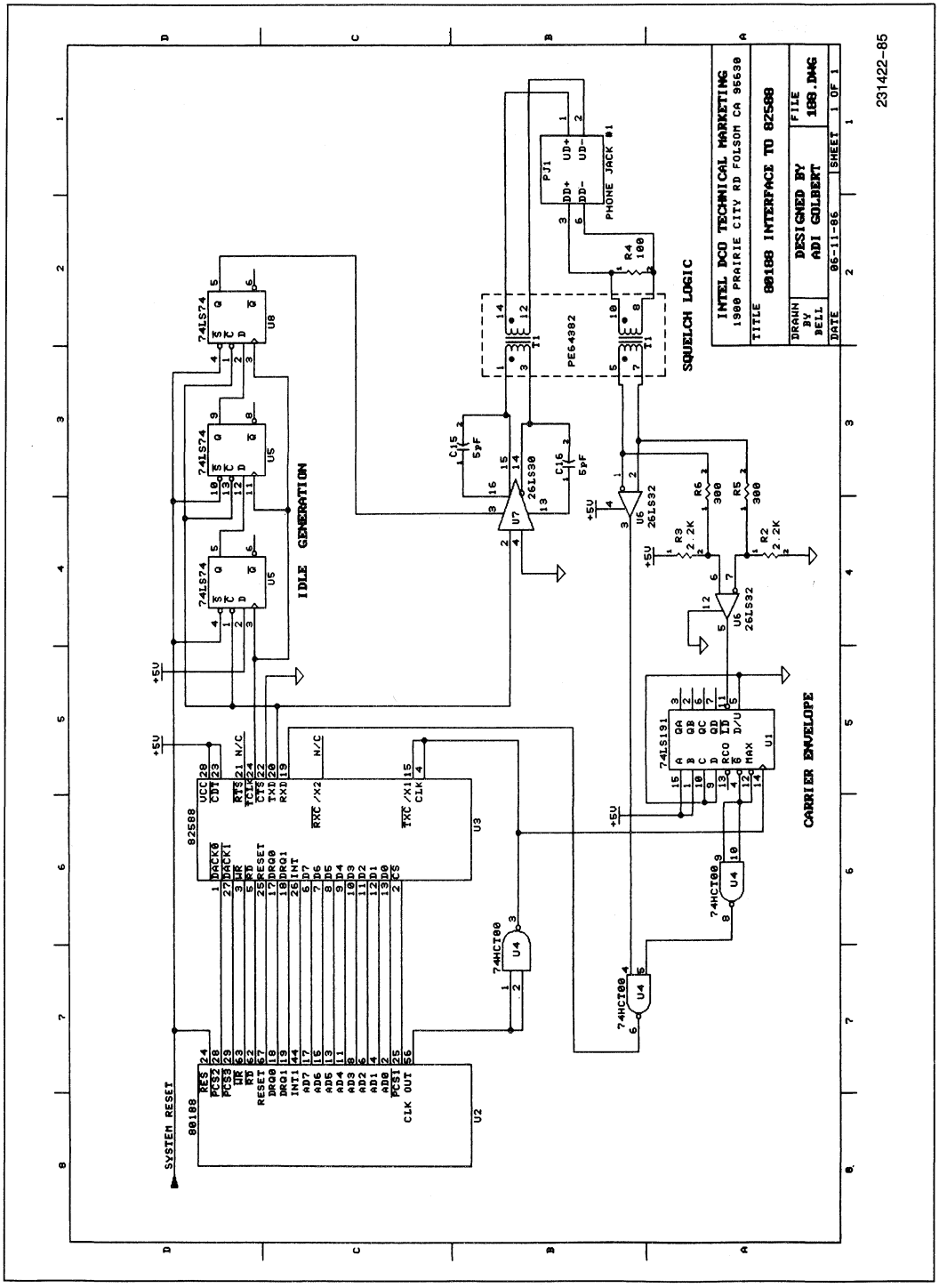


Figure 40. 80188 Interface to 82588

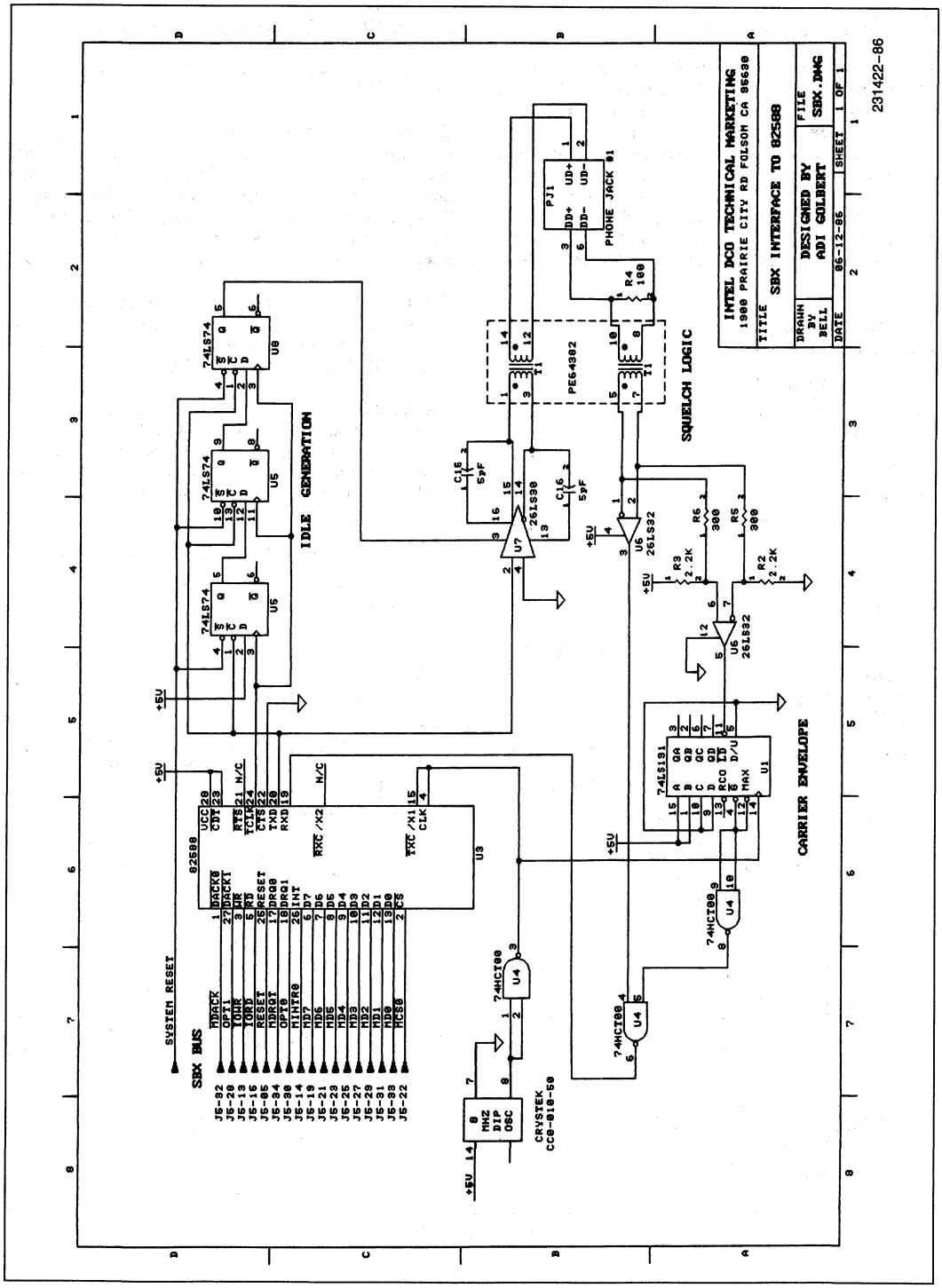


Figure 41. iSBX Interface to 82588

6.0 THE StarLAN HUB

The function of a StarLAN HUB is described in section 2.0. Figure 42 shows a block diagram of a HUB. It receives signals from the nodes (or lower level HUBs) detects if there is a collision, generates the collision presence signal, re-times the signal and sends it out to the higher level HUB. It also receives signals from the higher level HUB, re-times it and sends it to all the nodes and lower level HUBs connected to it. If there is no higher level HUB, a switch on the HUB routes the upstream received signal down to all the lower nodes. The functions performed by a HUB are:

*Receiving signals, squelch
*Carrier Sensing
*Collision Detection
*Collision Presence Signal Generation
*Signal Retiming
*Driving signals on to the cable
*Jabber Function
*Receive protection Timer

6.1 A StarLAN Hub for the IBM/PC

Figure 43 shows the implemention of a 5/6 port HUB for the IBM/PC.

The idea of the following design is to show a HUB that plugs into the IBM/PC backplane. This HUB not only gets its power from the backplane, but also enables the host PC to be one NODE into the StarLAN network. This embedded node scheme enables further savings due to the fact that all the analog interface for this port is saved (receiver, transmitter, transformer, etc).

This kind of board would suit very much a small cluster topology (very typical in departments and small offices) where the HUB board would be plugged into the FILE SERVER PC (PC/XT, PC/AT).

The HUB design doesn't implement the Jabber and the protection timers as called by the 1BASE5 draft standard. Those functions are optional and were not closed during the writing of this AP-NOTE. This HUB does implement the RETIMING circuit which is an essential requirement of StarLAN.

Figures 44 to 49 show a complete set of schematics for the HUB design.

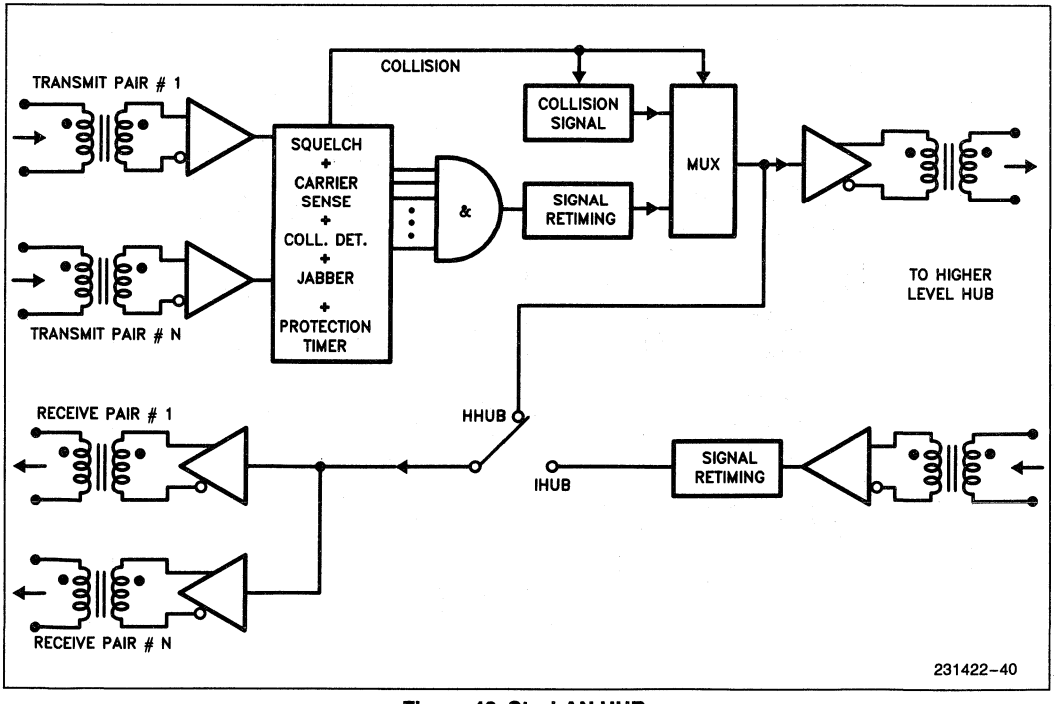
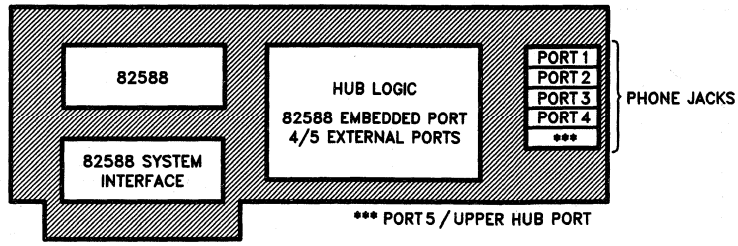


Figure 42. StarLAN HUB



231422-87

- Low Cost HUB, Uses IBM/PC Power Supply
- 82588, Embedded Port Savings
  - Transformers
  - 422 Drivers
- Functional StarLAN Cluster, For Low Cost/Small Topologies

Figure 43. IBM/PC Resident HUB

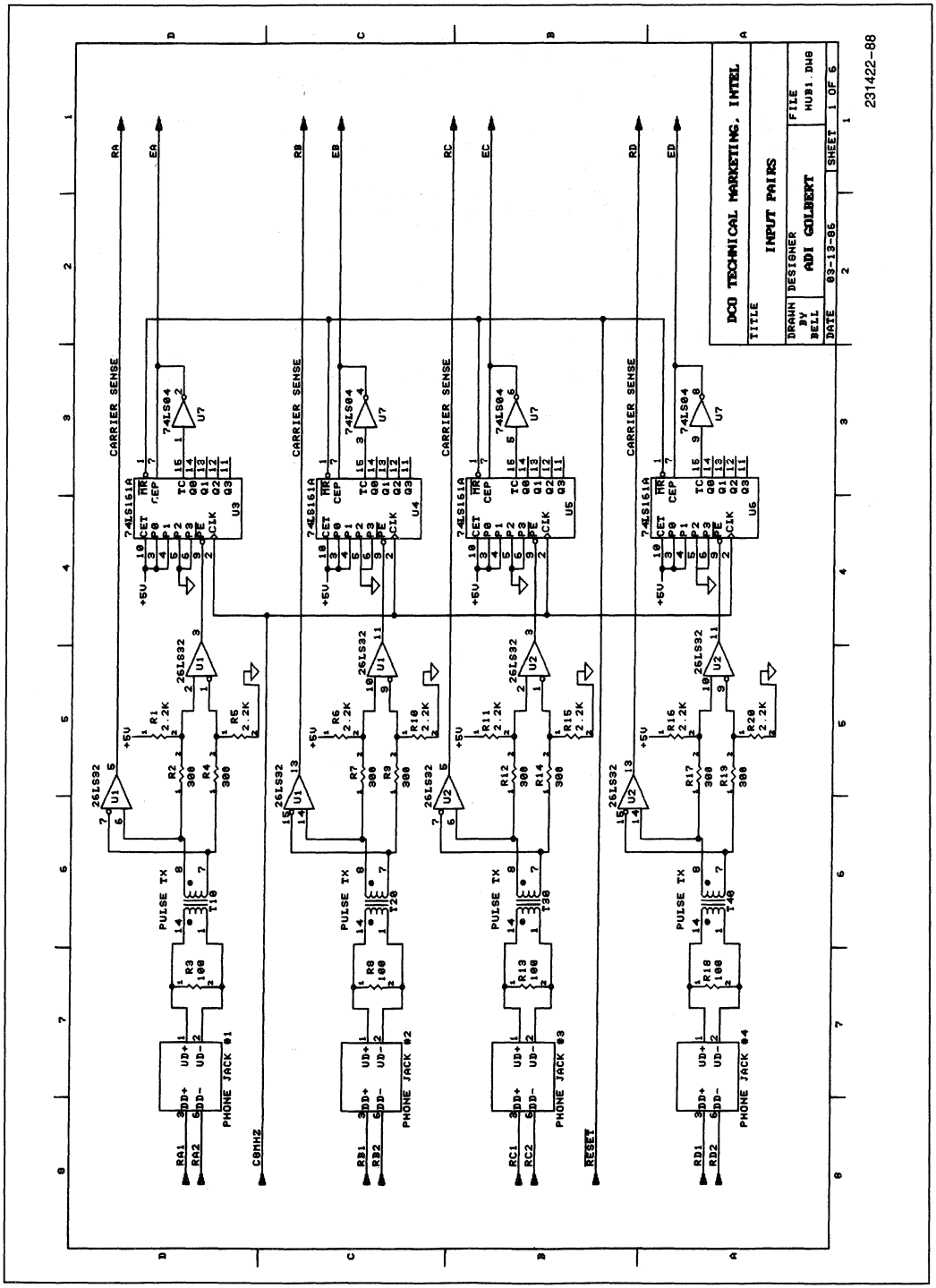

DCD TECHNICAL MARKETING, INTEL
INPUT PAIRS
DRAWN BY BELL
DESIGNER ADI GOLBERT
FILE HUB1.DWG
DATE 03-13-86
SHEET 1 OF 6
231422-88
CARRIER SENSE
PULSE TX
PHONE JACK #1
PHONE JACK #2
PHONE JACK #3
PHONE JACK #4
CONHZ
RESET

Figure 44

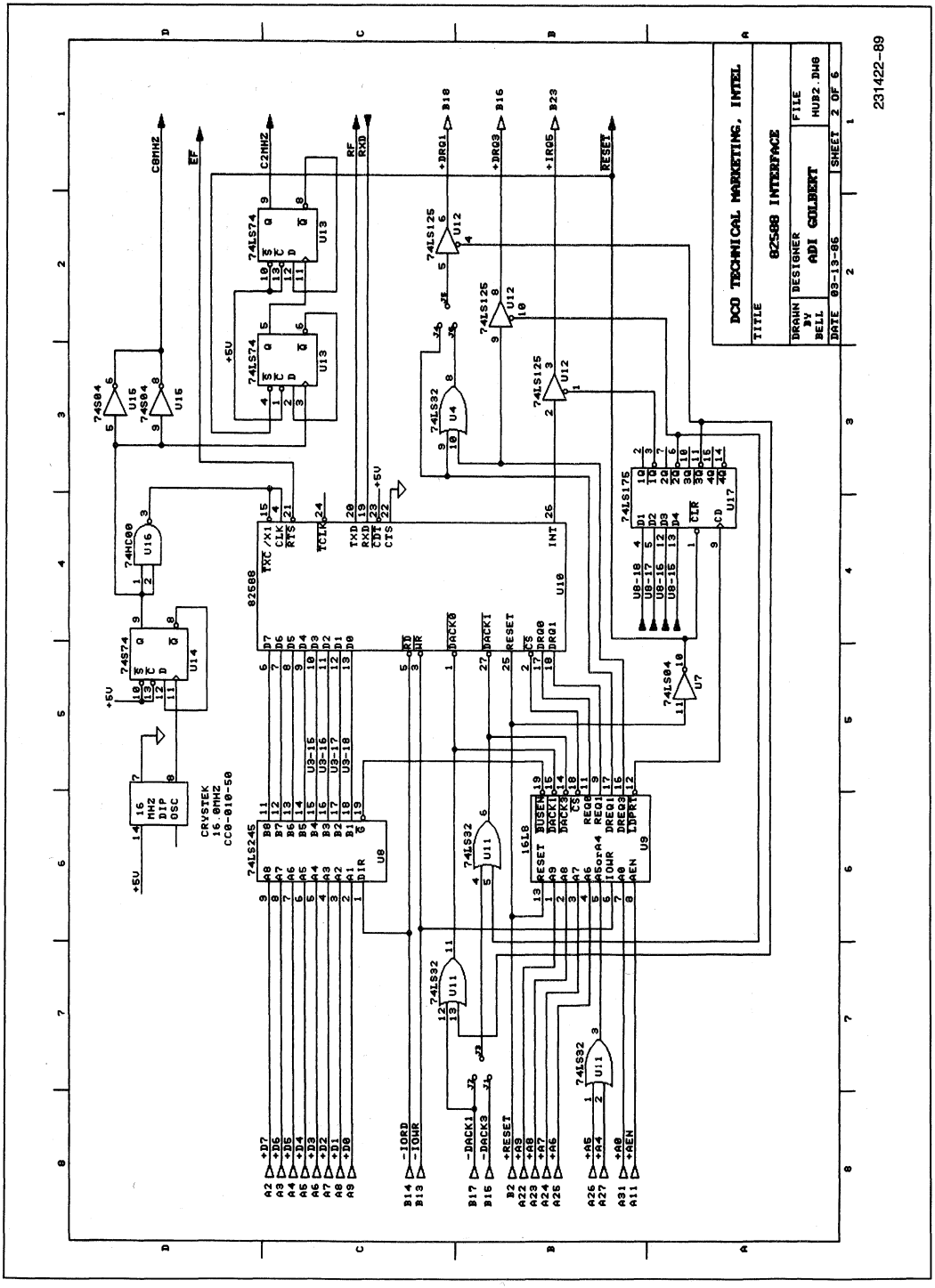

DCD TECHNICAL MARKETING, INTEL
82588 INTERFACE
DRAWN BY BELL
DESIGNER ADI GOLDSTEIN
FILE HUB2.DWG
DATE 03-13-86
SHEET 2 OF 6
231422-89
82588
74LS32
74LS245
74LS125
74LS175
74LS74
74S04
74HC00
16 MHZ DIP OSC
CRYSTEK 16.0MHZ CC8-010-50
C8MHZ
C2MHZ
EF
RF
RXD
TXC
TCLK
TXD
RXD
CTS
RTS
INT
RESET
+DRQ1
+DRQ3
+IRQ5
-IORD
-IOWR
-DACK1
-DACK3
+RESET
+D7
+D6
+D5
+D4
+D3
+D2
+D1
+D0
+5V

Figure 45

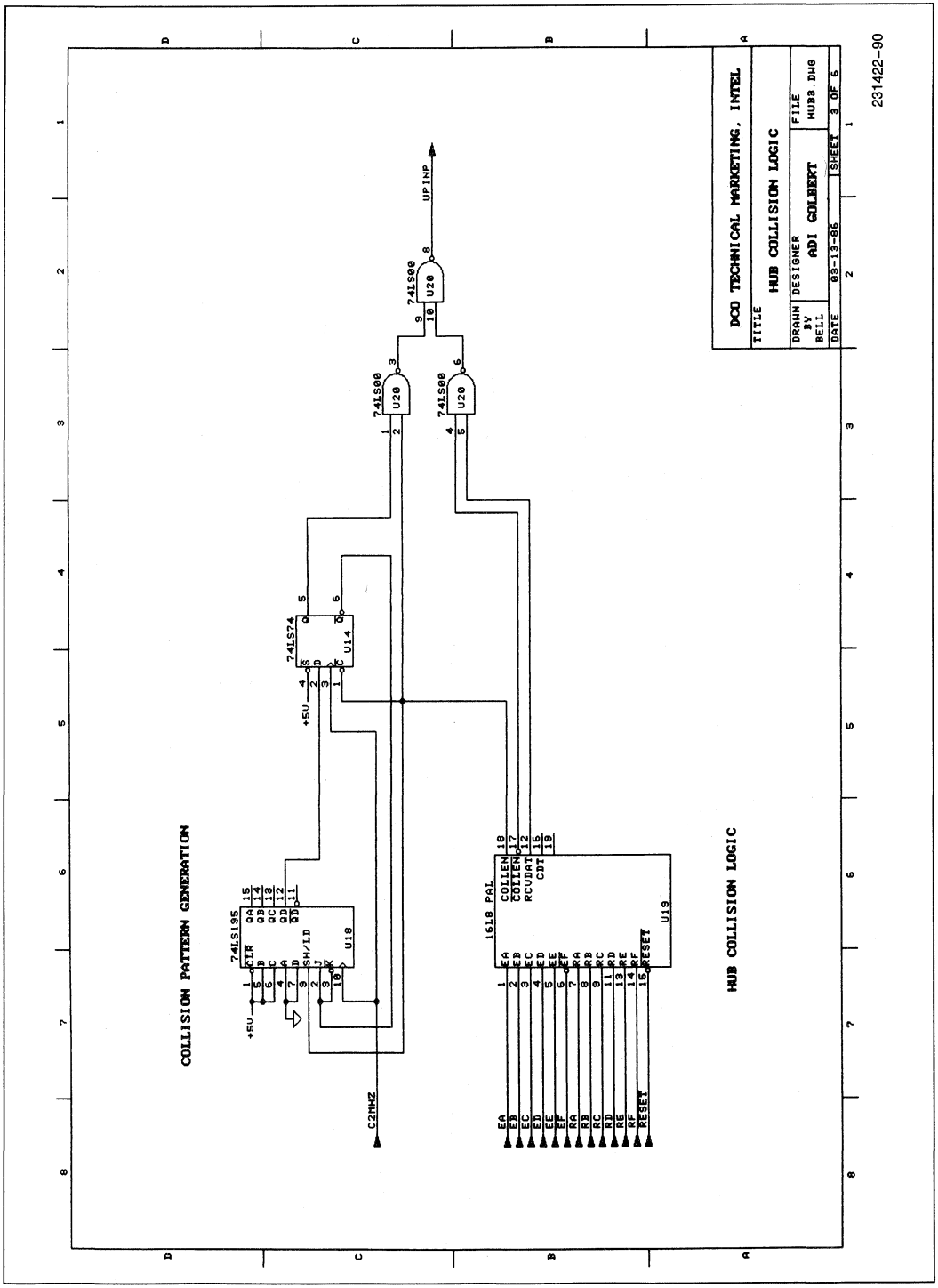

DCD TECHNICAL MARKETING, INTEL
TITLE
HUB COLLISION LOGIC
DRAWN BY BELL
DESIGNER ADI GOLBERT
FILE HUB3.DWG
DATE 03-13-86
SHEET 3 OF 6
231422-90
COLLISION PATTERN GENERATION
HUB COLLISION LOGIC
74LS195
74LS74
74LS00
16L8 PAL
U18
U14
U20
U19
+5V
C2MHZ
UPINP
RESET
COLEN
RCVDAT
CDT

Figure 46

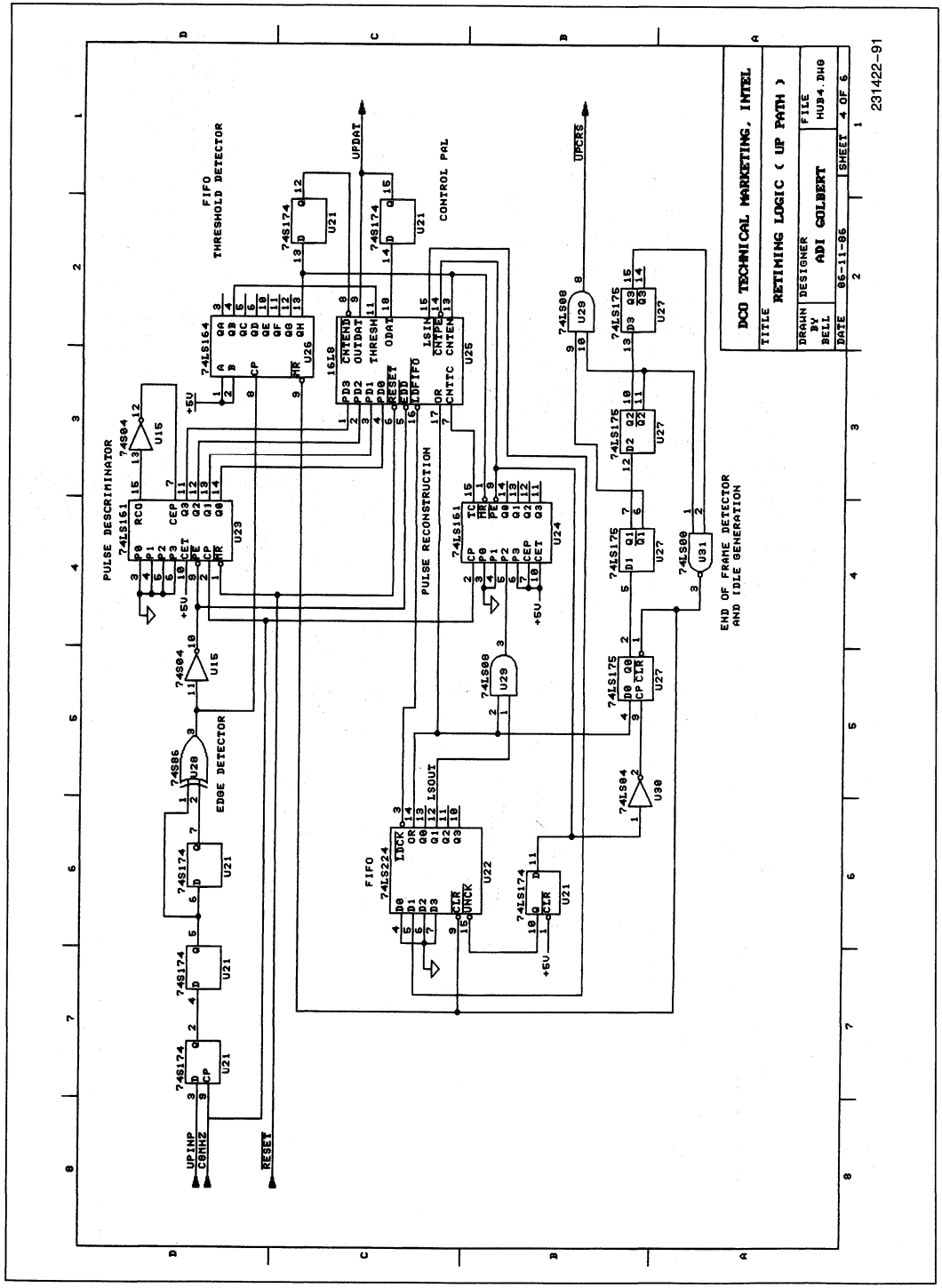

DCD TECHNICAL MARKETING, INTEL
TITLE
RETIMING LOGIC (UP PATH)
DRAWN BY
DESIGNER
ADI GOLBERT
FILE
HUB4.DWG
DATE
06-11-86
SHEET
4 OF 6
231422-91
PULSE DESCRIMINATOR
FIFO THRESHOLD DETECTOR
CONTROL PAL
PULSE RECONSTRUCTION
EDGE DETECTOR
END OF FRAME DETECTOR AND IDLE GENERATION
FIFO
UPDAT
UPCRS
UPINP
CRNHZ
RESET

Figure 47

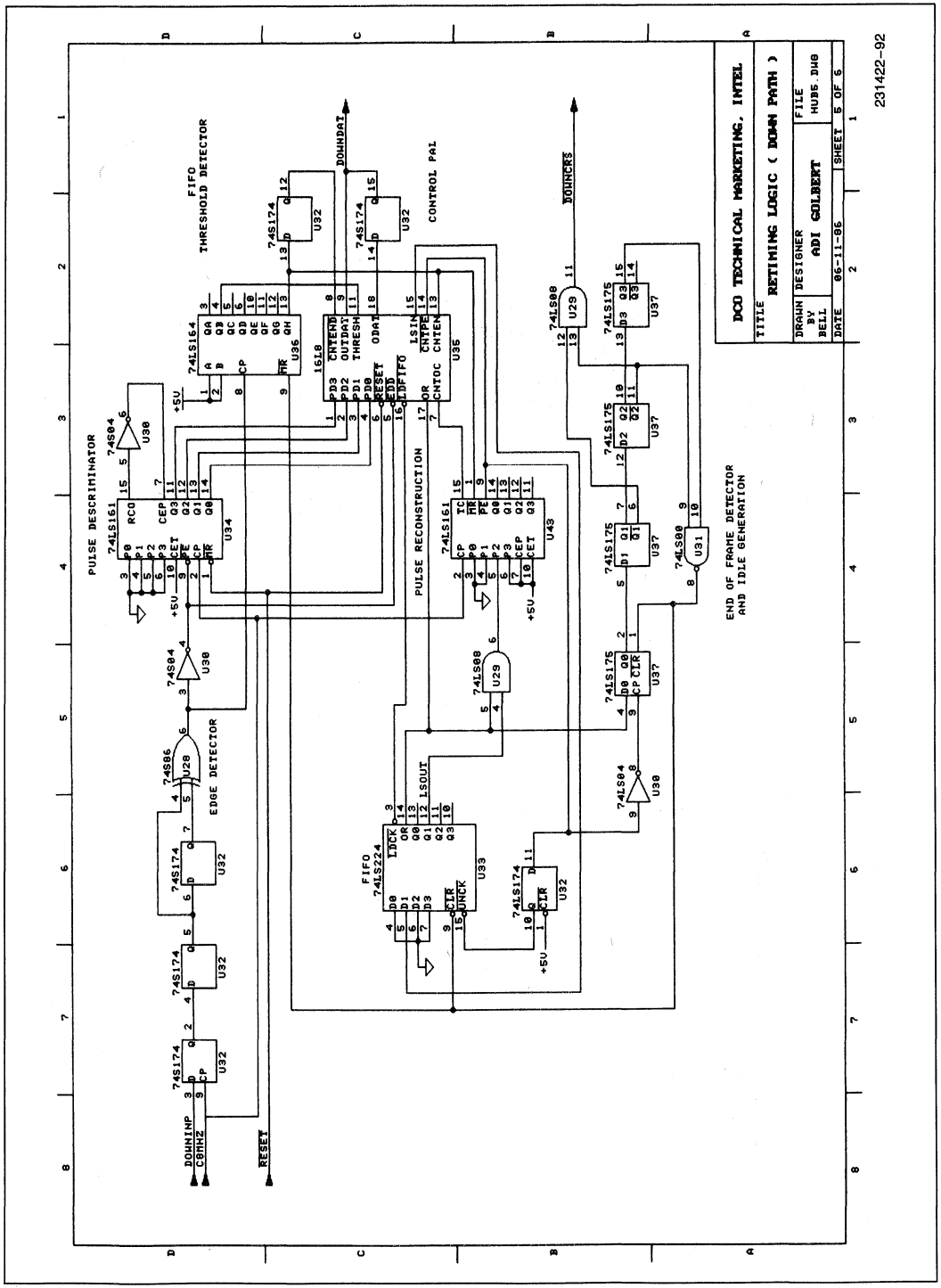

DCD TECHNICAL MARKETING, INTEL
TITLE
RETIMING LOGIC (DOWN PATH)
DRAWN BY
BELL
DESIGNER
ADI GOLDENT
FILE
HUB6.DWG
DATE
06-11-86
SHEET 5 OF 6
231422-92
PULSE DESCRIMINATOR
FIFO THRESHOLD DETECTOR
CONTROL PAL
PULSE RECONSTRUCTION
EDGE DETECTOR
END OF FRAME DETECTOR AND IDLE GENERATION
DOWNDAT
DOWNCKS
DOWNINP
C8MHZ
RESET

Figure 48

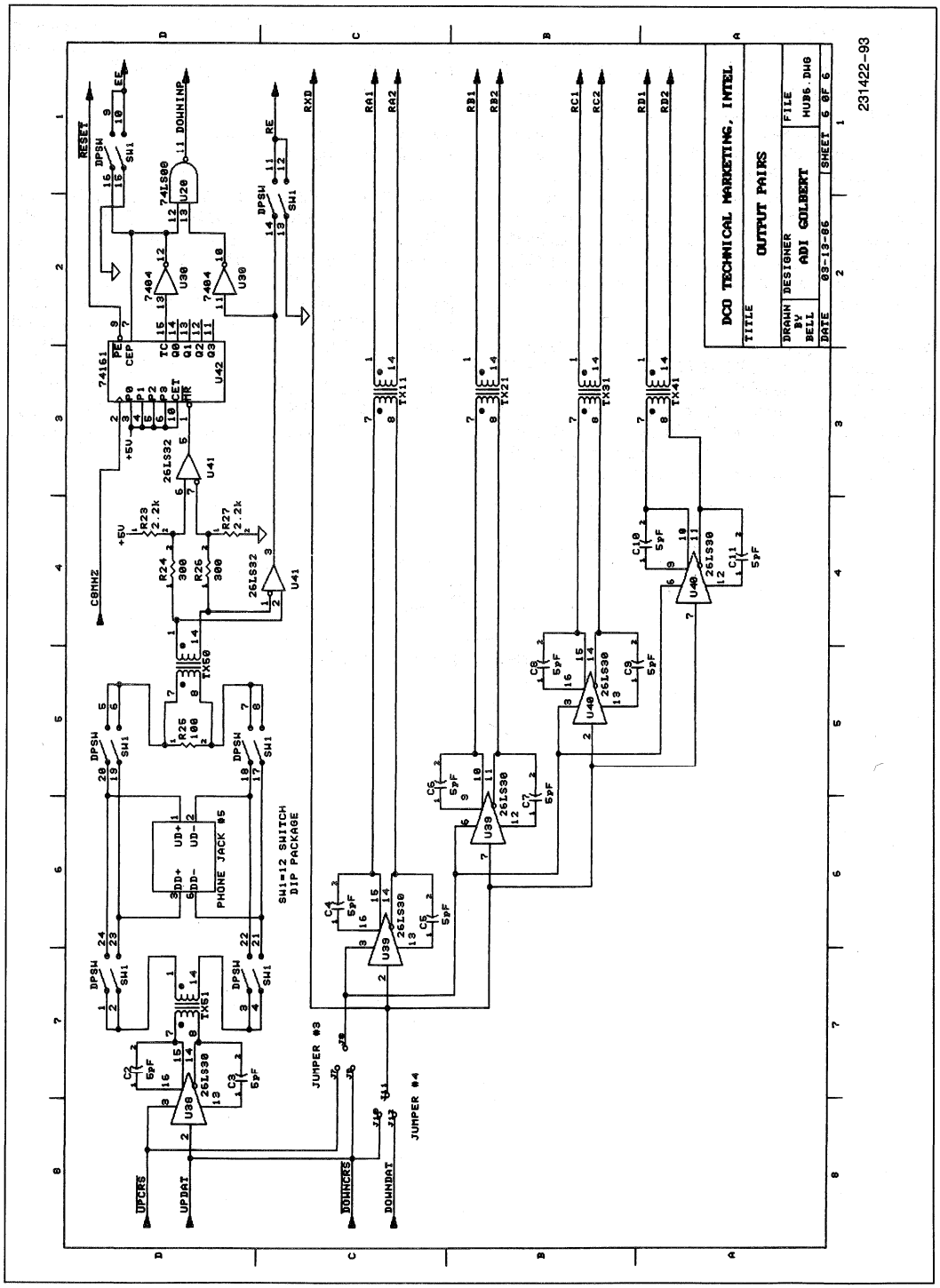

DCD TECHNICAL MARKETING, INTEL
TITLE
OUTPUT PAIRS
DRAWN BY BELL
DESIGNER ADI GOLBERT
FILE HUB6.DWG
DATE 03-13-86
SHEET 6 OF 6
231422-93
SW1=12 SWITCH DIP PACKAGE
PHONE JACK #5
JUMPER #3
JUMPER #4
UPCKRS
UPDAT
DOWNCKRS
DOWNDAT
RXD
RA1
RA2
RB1
RB2
RC1
RC2
RD1
RD2
RESET
DOWNINP
RE
EE
C8MHZ
+5V

Figure 49

6.1.1 HUB INPUT PORTS

Figure 38 shows a block diagram of an input port. Differently than the implementation in Figure 29 the HUB input port is potentially more complex than the NODE input port. The reason being that the HUB is a central resource and much more sensitive to noise. For example, if the NODE input port would falsely interpret noise on an IDLE line as valid signal, the worst case situation would be that this noise would be filtered out by the 82588 time squelch circuitry, on the HUB by the other hand, this false carrier sense could trigger a COLLISION and a good frame (on another input) potentially discarded.

As shown in Figure 38 immediately after the termination resistor, there is a HIGH FREQUENCY FILTER circuit. The purpose of this circuit is to eliminate high frequency noise components keeping noise jitter into the allocated budget (about ±30 ns). A 4 MHz two pole butterworth filter is being recommended by the IEEE 802.3 1BASE5 task force (see Figure 50).

The time squelch for the NODE board is implemented by the 82588 (see section 3.7) this circuit makes sure that pulses that are shorter than a specified duration will be filtered out.

The other components of the block diagram were explained in section 3.0.

The HUB design doesn't implement the HIGH FREQUENCY FILTER and TIME SQUELCH. In the HUB design as an output of each input port, two signals are available: Rn, En, (RA, RB . . . , EA, EB . . .). The Rn signals are the receive data after the zero crossing receivers. The En lines are CARRIER SENSE signals. The HUB design supports either 5 or 6 input ports, dependent upon if it is configured as IHUB or HHUB. Port RE, EE (Figure 49) is bidirectional, configurable for either input or output. Port RF, EF__ is the embedded 82588 port, and doesn't require the analog circuitry (EF is inverted, being generated from the RTS__ signal).

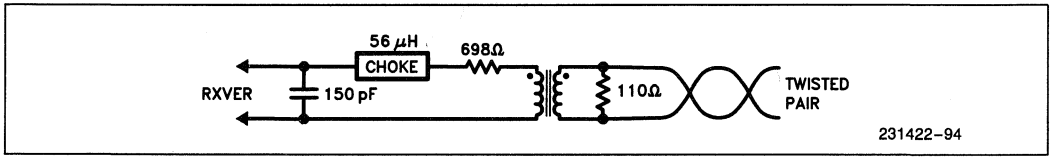


Figure 50. Receiver High Frequency Filter

6.1.2 COLLISION DETECTION

Rn and En signals from each channel are fed to a 16L8 PAL, where the collision detection function is performed.

Collision Detection in the StarLAN HUB is performed by detecting the presence of activity on more than one input channels. This means if the signal En is active for more than one channel, a collision is said to occur. This translates to the PAL equations:

COLLISION DETECTION:

```
CDT = ! (EA & !EB & !EC & !ED & !EE & EF__ #          (only EA active)
         ! EA & EB & !EC & !ED & !EE & EF__ #         (only EB active)
         ! EA & !EB & EC & !ED & !EE & EF__ #         (only EC active)
         ! EA & !EB & !EC & ED & !EE & EF__ #         (only ED active)
         ! EA & !EB & !EC & !ED & EE & EF__ #         (only EE active)
         ! EA & !EB & !EC & !ED & !EE & !EF__ #       (only EF active)
         ! EA & !EB & !EC & !ED & !EE & EF__ );   (none of the inputs active)
```

COLLISION DETECTION SR-FF:

```
COLLEN__ = ! (CDT # COLLEN );                          (set with collision)

COLLEN__ = ! ( RESET__ # COLLEN__ #
             ( !CDT & !EA & !EB & !EC & !ED & !EE & EF__);

                                          ( reset when all inputs inactive )
```

RECEIVE DATA OUTPUT:

```
RCVDAT = (  ( RA # !EA ) & ( RB # !EB ) & ( RC # !EC ) &

            ( RD # !ED ) & (RE # !EE ) & (RF # EF__ ) );

                                          ( output is high if no active input)
```

The COLLEN signal once triggered will stay active until all inputs go quiet. This signal is used externally to either enable passing RCVDAT or the collision presence signal (CPS) to the retiming logic. An external multiplexer using 3 nand gates is used for this function. Note that in this specific implementation the CPS/RCVDAT multiplexer is before the retiming logic, which is different from Figure 42 diagram. StarLAN provides enough BIT-BUDGET delay to allow the CPS signal to be generated through the retiming FIFO. In this HUB implementation it was decided to use this option to make sure that the CPS startup is synchronized with the previously transmitted bit as required by the 1BASE5 draft.

6.1.3 THE LOCAL 82588

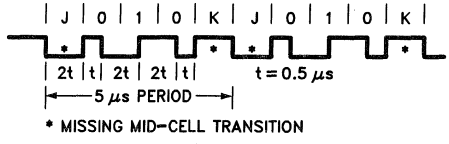
As described before, the purpose of the local 82588 is to enable the Host IBM/PC to also be a node into the StarLAN network. The interface of this 82588 is exactly similar to the one explained in section 5. The RTS__ signal serves as the carrier EF__ signal, and TXD as RF signal. This local node interfaces to the HUB without any analog interface which is a significant saving.

6.1.4 THE COLLISION PRESENCE SIGNAL

The Collision Presence Signal (CPS) is generated by the HUB whenever the HUB detects a collision. It then propagates the CPS to the higher level HUB. The CPS signal pattern is shown in Figure 51. Whenever a StarLAN node receives this signal, it should be able to detect within a very few bit times that a collision occurred. Since the nodes detect the occurrence of a collision by detecting violations in Manchester encoding, the CPS must obviously be a signal which violates Manchester encoding. Section 3.5 shows that the CPS has missing mid-cell transitions occurring every two and a half bit cells. These are detected as Manchester code violations. Thus, the StarLAN node is presented with collision detection indications every two and a half ms. This results in fast and reliable detection of collisions. CPS has a period of 5 ms.

One may wonder why such a strange looking signal was selected for CPS. The rationale is that this CPS looks very much like a valid Manchester signal—edges are 0.5 or 1.0 microsec. apart—resulting in identical radiation, cross-talk and jitter characteristics as a true Manchester. This also makes the re-timing logic for the signals simpler—it need not distinguish between valid Manchester and CPS. Moreover, this signal is easy to generate.

A few important requirements for CPS signal are: a) it should be generated starting synchronized with the last transmitted bit cell. CPS is allowed to start either low or high, but no bit cell of more than 1 microsecond is allowed (Avoid false idles, very long "low" bits). b) once it starts, it should continue until all the input lines to the HUB die out. Typically, when the collision occurs, the multiplexor in the HUB switches from RCV signal to the CPS. This switch is completely asynchronous to the currently being transmitted data, and by such may violate the requirement of not having bit cells longer than 1 μs. In order to avoid those long pulses, the output of the CPS/RCVDAT multiplexer is passed through the retiming circuitry which will correct those long pulses to their nominal value. The reason for restriction b) is to ensure that the CPS is seen by all nodes on the network since it is generated until every node has finished generating the Jam pattern.



231422-42

- Collision Presence Signal (CPS) is generated by the HUB when it detects more than one input line active.
- CPS violates Manchester encoding rules—due to missing mid-cell transitions—hence is detected as a collision by the DTE (82588).

Choice of Collision Presence Signal

- It is a Manchester look-alike signal—edges are 0.5 or 1.0 μs apart.
  - — Identical radiation, crosstalk and jitter characteristics
  - — Eases retiming of the signal in the HUB
- It is easy to generate—1.5 TTL pack, or in a PAL

Figure 51. Collision Presence Signal

CPS is generated using a 4-bit shift register and a flip-flop as shown in Figure 52. It works off a 2 MHz clock. A closer look at the CPS waveform shows that it is inverse symmetric within the 5 μs period. The circuit is a 5-bit shift register with a complementary feedback from the last to the first bit. The bits remain in defined states (01100) till collision occurs. On collision the bits start rotating around generating the pattern of 0011011001, 0011011001, 00110 ... with each state lasting for 0.5 μs.

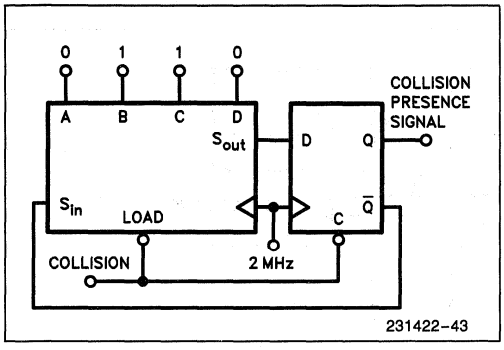


Figure 52. Collision Presence Signal Generation

6.1.5 SIGNAL RETIMING

Whenever the signal goes over a cable it suffers jitter. This means that the edges are no longer separated by the same 0.5 or 1.0 μs as at the point of origin. There are various causes of jitter. Drivers, receivers introduce some shifting of edges because of differing rise and fall times and thresholds. A random sequence of bits also produces a jitter which is called intersymbol interference, which is a consequence of different propagation delays for different frequency harmonics in the cable. Meaning short pulses have a longer delay than long ones. A maximum of 62.5 ns of jitter can accumulate in a StarLAN network from a node to a HUB or from a HUB to another HUB. The following values show what are the jitter components:

Transmitter skew	±10 ns
Cable Intersymbol interference	±9 ns
Cable Reflections	±8 ns
Reflections due to receiver termination mismatch	±5 ns
HUB fan-in, fan-out	±5 ns
Noise	±25.5
Total	±62.5 ns

It is important for the signal to be cleaned up of this jitter before it is sent on the next stretch of cable because if too much jitter accumulates, the signal is no longer meaningful. A valid Manchester signal would, as a result of jitter, may no longer be decodable. The process of either re-aligning the edges or reconstructing the signal or even re-generating the signal so that it once again "looks new" is called re-timing. StarLAN requires for the signal to be re-timed after it has travelled on a segment of cable. In a typical HUB two re-timing circuits are necessary; one for the signals going upstream towards the higher level HUB and the other for signals going downstream towards the nodes.

6.1.6 RETIMING CIRCUIT, THEORY OF OPERATION

This section will discuss the principles of designing a re-timing circuit. Figure 53 shows the block diagram of a re-timing circuit. The data coming in is synchronized using an 8 MHz sampling clock. Edges in the waveform are detected doing an XOR of two consecutive samples. A counter counts the number of 8 MHz clocks between two edges. This gives an indication of long (6 to 10 clocks) or short (3 to 5 clocks) pulses in the received waveform. Pulses shorter than 3 clocks are filtered out. Every time an edge occurs, the length—(S)hort or (L)ong—of the pulse is fed into the FIFO. Retiming of the waveform is done by actually generating a new waveform based on the information being pumped into the FIFO. The signal regeneration unit reads the FIFO and generates the output waveform out of 8 MHz clock pulses based on what it reads, either short or longs. In summary every time a bit is read from the fifo, it indicates that a transition needs to occur, and when to fetch the next bit. When idle the output of the retiming logic starts with a "high" level.

FIFO	Output
empty	1111
S	0000
S	1111
L	00000000
L	11111111

It can be seen that the output always has edges separated by 4 or 8 clock pulses—0.5 or 1.0 μs.

The FIFO is primarily needed to account for a difference of clock frequencies at the source and regeneration end. Due to this difference, data can come in faster or slower than the regeneration circuit expects. A 16 deep FIFO can handle frequency deviations of up to 200 ppm for frame lengths up to 1600 bytes. The FIFO also overcomes short term variations in edge separation. It is essential that the FIFO fills in up to about half before the process of regeneration is started. Thus, if the regeneration is done at a clock slightly faster than the source clock, there is always data in the FIFO to work from. That is why the FIFO threshold detect logic is necessary, which counts 8 edges and then enables the signal regeneration logic.

Example:

Input Waveform ...1111000111100000001111111111000111100...

Input into the FIFO <S> <S> <L> <L> <S> <S>

Regenerated Output:

Output: ...1111100001111000000001111111110000111...

FIFO: <S> <S> <L> <L> <S> <S>

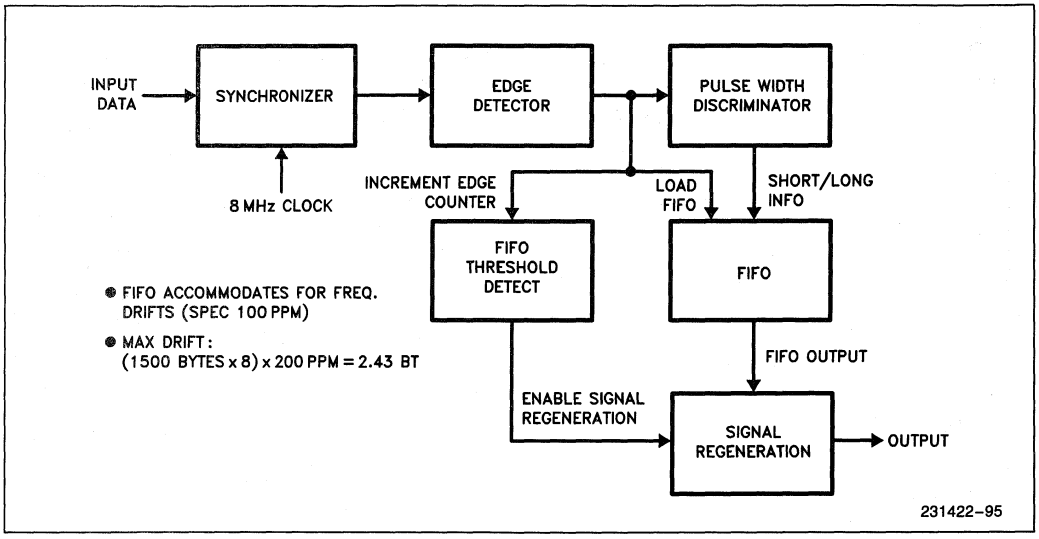


Figure 53. Retiming Block Diagram

6.1.7 RETIMING CIRCUIT IMPLEMENTATION

The retiming circuit implementation can be seen in Figures 47, 48. Both figures implement exactly the same function, one for the upstream, and the other for the downstream. The retiming circuit was implemented using about 8 SSI, MSI TTL components, one fifo chip and one PAL. The purpose of implementing this function with discrete components was to show the implementation details. The discussion of the implementation will refer to Figure 47 for unit numbers.

The signal UPIMP which is an output of the HUB multiplexing logic, is asynchronous to the local clock. This signal is synchronized by two flip-flops and fed into an edge generation logic (basically an XOR gate that compares the present sample with the previous one). On every input transition a 125 ns pulse will be generated at the output of the edge detector (U28). This pulse will reset the 74LS161 counter that is responsible for measuring pulse widths (in X8 clock increments). The output of the pulse discriminator will reflect the previous pulse width every time a new edge is detected. The following events will take place on every detected edge:

1. U26 which is the threshold detector will shift one "1" in. The outputs of U26 will be used by the control PAL to start the reconstruction process.
2. The output of U23 which specifies the last pulse width will be input into the control PAL for determining if it was a long or short pulse. The result of this evaluation will be the LSIN signal which will be loaded into the fifo (U22).

U22 is the retiming FIFO, it is 16x4 fifo, but only one bit is necessary to store the SHORT/LONG information.

CONTROL LOGIC PAL functions (U25):

Signals definition:

INPUTS:

PD0..PD3: Outputs of the pulse descriminator, indicate the width of the last measured pulse.

EDD__: Output of the edge detector, pulse of 125 ns width, indicates the occurrence of an edge in the input data.

THRESH: Output of the threshold logic, indicates at least one bit was already received.

CNTEN: Output of the Threshold logic, indicates 7 bits have been loaded into the FIFO, and that signal reconstruction can begin.

CNTEND: The same signal as before delayed by one clock.

OUTDAT: Output of the retiming logic, is feedback into the PAL to implement a clocked T-FF.

RESET__: Resets the retiming logic.

CNTTC: Terminal count of the reconstruction counter, indicating that reconstruction of a new bit will get started.

OR: Output of the FIFO indicating, that the FIFO is empty and that IDLE generation can get started.

OUTPUTS:

LDFIFO__: Loads SHORT/LONG indications into the FIFO.

LSIN: Indicates SHORT/LONG

CNTPE__: Loads FIFO SHORT/LONG output into the reconstruction counter.

ODAT: Together with the external U21 flip-flop and OUTDAT implement a clocked T-FF.

Loading the FIFO will be done every time there is an edge, we have passed the one bit filter threshold level, and the pulse width is longer than two 8X clocks. This one bit threshold level serves as a time domain filter discarding the first received preamble bit.

```
LDFIFO_ = ! ( PD1 # PD2 # PD3 ) & !EDD_ & THRESH ) ;

Whenever there is an edge, we are above the first received bit threshold
and the pulse width is longer than "1" the fifo is loaded.

LSIN    = ! (PD3 # (PD2 & PD0) # (PD2 & PD1)) ;

Every pulse longer than 6 is considered to be a long pulse.

CNTPE_  = ! ( (CNTEN & !CNTEND) # CNTTC ) ;

The reconstruction counter is loaded in two conditions:

Whenever CNTEN comes active, meaning the FIFO threshold of seven was exceeded.
Whenever the terminal count of U24 is active meaning a new pulse is going to be reconstructed.

ODAT    = !RESET_ # ( !CNTPE_ & !OUTDAT)    (A)
                  # ( CNTPE_ &  OUTDAT)    (B)
                  # ( !CNTPE_ & !OR)        (C)

Minterm (A) and (B) implement a T-FF, whenever CNTPE. is "low"
ODAT will toggle. The external U21 is part of this flip-flop.
Minterm (C) insures the output of the flip-flop will go inactive
"high" when the FIFO is empty. RESET. causes the output to go
"high" on initialization.
```

U24 as mentioned is the reconstruction counter. This counter is loaded by the control logic with either 8 or 12, it counts up and is reloaded on terminal count. Essentially generating at the output nominal length longs and shorts.

U22 is the retiming FIFO, and its function as mentioned is to accommodate frequency skews between the incoming and outgoing signal.

U27 is the IDLE generation logic. The purpose of this logic is to detect when the FIFO is empty, meaning that no more data needs to be transmitted. On detection of this event this component will generate 2 ms of IDLE time. On the end of IDLE the whole retiming logic will be reset.

6.1.8 DRIVER CIRCUITS

The signal coming out of the RETIMING LOGIC is fed into 26LS30s and pulse transformers to drive the twisted pair lines (See section 5.0 for details).

6.1.9 HEADER/INTERMEDIATE HUB SWITCH

As seen on Figure 43 this hub can be configured as either an intermediate hub, or a Header one. One of the phone jacks, more specifically JACK #5 is either an input port or an output one. In order to implement this function, an 8 position DIP SWITCH (SW1) is used. The phone jacks are marked with UD, DD notation, meaning upstream data, and downstream data respectively. As specified in the StarLAN 1BASE5 draft NODES transmit data on UD pair, and HUBS on the DD pair. Switch SW1 has the function to invert UD, DD in PHONE JACK #5 to enable it to be either input or output port.

6.1.10 JABBER FUNCTION

This design does not implement the jabber unit but it is described here for completeness. IEEE 802.3 does not mandate this feature, but it is "Strongly Recommended". The jabber function in the HUB protects the network from abnormally long transmissions by any node.

Two timers T1, T2 are used by the JABBER function. They may be implemented either as local timers (one for each HUB port) or as global timers shared by all ports. After detecting an input active, timers T1, T2 will be started, and T1 will time out after 25 to 50 ms. T2 will time-out after 51 to 100 ms. During T2 time, after T1 expired, the HUB will send the CP-PATTERN informing any jamming stations to quit their transmissions. If on T2 time-out there are still jamming ports, their input is going to be disabled. A disabled port, will be reenabled whenever its input becomes again active and the downward side is idle.

The following is an explanation of the requirement that the downward side be idle to reenable an input port. Consider the case of Figure 54. The figure shows a two port HUB. Port A has two wires A_u, A_d for the up and down paths. Port B has B_u, B_d respectively. Port C is the output port, that broadcasts to the other HUBs higher in the hierarchy. Consider the case as shown, where B_u and B_d are shorted together. Suppose the case that port A_u is active. Its signal will propagate up in the hierarchy through C_u and come down from C_d to A_d, and B_d. Due to the short between B_d and B_u the signal will start a loop, that will first cause a collision and jam the network forever. This kind of fault is taken care of by the jabber circuitry. T1 and T2 will expire, causing the jabber logic to disable B_u input. Upon this disabling B_u is going to go Idle and be a candidate for future enabling. Suppose now that A_u is once again active. If the reenable condition would not require C_d to be IDLE, B_u would be reenabled causing the same loop to happen once again. Note that in this case C_d will be active before B_u causing this port to continue to be disabled and avoiding the jamming situation (Figure 55) gives a formal specification of the jabber function).

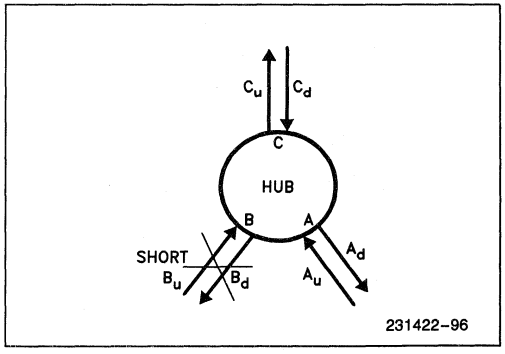


Figure 54. Jabber Function

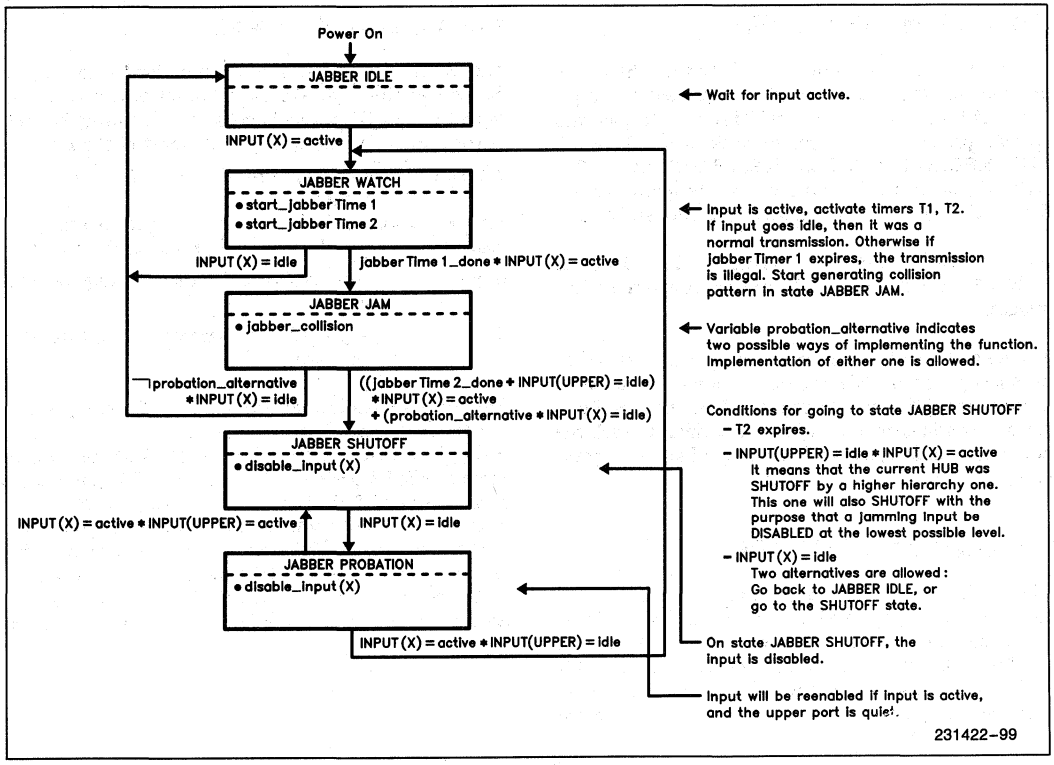


Figure 55. Jabber State Diagram

6.1.11 HUB RECEIVER PROTECTION TIMER

On the end of a transmission, during the transition from IDLE to high impedance state, the transmitter will exhibit an undershoot and/or ringing, as a consequence of transformer discharge. This undershoot/ringing will be transmitted to the receiver which needs to protect itself from false carriers due to this effect. One way of implementing this protection mechanism is to implement a blind timer, which upon IDLE detection will "blind" the receiver for a few microseconds.

Causes of the transmitter undershoot/ringing:

1. Difference in the magnitudes of the differential output voltage between the high and the low output stages.
2. Waveform assymmetry due to transmitter jitter.
3. Transmitter and receiver inductance (transformer L).
4. Two to three microseconds of IDLE pattern.

All the described elements will contribute to energy storage into the transformer inductor, which will discharge during the transition of the driver to high impedance.

The blinding timer is currently defined to be from 20 to 30 microseconds for the HUBs, being from 0 to 30 microseconds for the nodes (optional). The 82588 has built-in this function. It won't receive any frames for an inter-frame-spacing (IFS) from the idle detection.

6.1.12 HUB RELIABILITY

Since the StarLAN HUBs form focal points in the network, it is important for them to be very reliable, since they are single points of failure which can affect a number of nodes or can even bring down the whole network. StarLAN 1BASE5 draft requires HUBs to have a mean time between failures (MTBF) of at least 5 years of continuous operation.

7.0 SOFTWARE DRIVER

The software needed to drive the 82588 in a StarLAN environment is not different from that needed in a generic CSMA/CD environment. This section goes into specific procedures used for operations like TRANSMIT, RECEIVE, CONFIGURE, DUMP, ADDRESS SET-UP, etc. A special treatment will be given to interfacing with the IBM PC—DMA, interrupt and I/O.

Since all the routines were written and tried out in PLM-86 and ASM-86, all illustrations are in these languages.

The following software examples are pieces of an 82588 exerciser program. This program's main purpose was to exercise the 82588 functionality and provide the functions of traffic generation and monitoring. By such the emphasis was on speed and accuracy of statistics gathering.

7.1 Interfacing to IBM PC

The StarLAN board interfaces to the CPU, DMA controller and the interrupt controller on the IBM PC system board. The software to operate the 82588 runs on the system board CPU. The illustrated routines in this section show exactly how the software interface works between the system resources on the IBM PC and the StarLAN board.

7.1.1 DOING I/O ON IBM PC

The safest way to use the PC monitor as an output device and the keyboard as the input device is to use them through DOS system calls. The following is a set of routines which are handy to do most of the I/O:

key$stat	—to find out if a new key has been pressed
keyin$noecho	—to read a key from the keyboard
char$out	—to display a character on the screen
msg$out	—to display a character string on the screen
line$in	—to read in a character string from the keyboard

The exact semantics and the protocol for doing these functions through DOS system calls is shown in the listing in Figure 56. Refer to the DOS Manual for a more detailed description. To make a DOS system call, register AH of 8088 is loaded with the call Function Number and then, a software interrupt (or trap) 21 hex is executed. Other 8088 registers are used to transfer any parameters between DOS and the calling program. The code is written in Assembly language for register access. Let us see an example of the 'msg$out' routine:

```
lds dx,STRING_POINTER    ; load pointer to string in reg. ds:dx
mov ah,09h               ; 9 = function number for string o/p
int 21h                  ; DOS System Call
```

These procedures are called from another module, written in a higher level language like PLM-86. The parameters are transferred to the ASM-86 routines on the stack.

Examples of using the I/O routines:

```
KEY_STATUS = key$stat;                          /* INQUIRE KEYBOARD STATUS      */
NEW_KEY = keyin$noecho;                         /* INPUT NEW KEY                */
call line$in(@LINE_BUFFER);                     /* STRING INPUT                 */
call char$out(CHAR_OUT);                        /* TO OUTPUT CHAR_OUT ON SCREEN*/
call msg$out(@('THIS IS A MESSAGE.$'));         /* OUTPUT STRING                */
                                                /* NOTE $ TERMINATOR            */
```

```
/*-----------------------------------------------------------------------*/
/*     Declarations for external IBM PC I/O routines                     */
/*-----------------------------------------------------------------------*/

key$stat: procedure byte external;        /* key status routine */
end key$stat;

key$in$noecho: procedure byte external;       /* console input routine */
end key$in$noecho;

char$out: procedure(char) external;   /* console output routine */
declare char byte;
end char$out;

msg$out: procedure(msg$ptr) external;         /* console string output routine
declare msg$ptr pointer;
end msg$out;

line$in: procedure(line$ptr) external;        /* console string input routine
declare line$ptr pointer;
end line$in;


Assembly Language implementation of the routines

$TITLE(IBM/PC  DOS CALLS PROCEDURES)

            NAME    DOSPROCS
;
DGROUP              GROUP   DATA
CGROUP              GROUP   CODE
;
DATA                SEGMENT WORD PUBLIC 'DATA'
DATA                ENDS
;
DOS         EQU     21H
;
CODE                SEGMENT WORD PUBLIC 'CODE'
                    ASSUME  CS:CGROUP,DS:DGROUP
```

231422-58

```
;
; CHAR$OUT: PROCEDURE(CHAR) EXTERNAL;
; DECLARE CHAR BYTE;
; END CHAR$OUT;
; Outputs character to the screen.
; DOS system call 2
;
CHAR                EQU     [BP+4]        STACK
;
CHAROUT             PROC    NEAR          +------+
            PUBLIC  CHAROUT               ! CHAR ! x
            PUSH    BP                    +------+
            MOV     BP,SP                 !IP lo ! x-1
            MOV     DL,CHAR               +------+
            MOV     AH,2                  !IP hi ! x-2
            INT     DOS                   +------+
            POP     BP                    !BP lo ! x-3
            RET     2                     +------+
CHAROUT             ENDP                  !BP hi ! x-4  <--SP
;                                         +------+
; KEYIN$NOECHO: PROCEDURE BYTE EXTERNAL;
; END KEYIN$NOECHO;
; Reads character without echoing to display
;
KEYINNOECHO PROC    NEAR
            PUBLIC  KEYINNOECHO
            MOV     AH,8                  (DOS call 8)
            INT     DOS
            RET
KEYINNOECHO ENDP
```

Figure 7-56. I/O Routines for IBM/PC (continued)

231422-59

Figure 56. I/O Routines for IBM/PC

```
;
;
; MSG$OUT: PROCEDURE(MSG$PTR) EXTERNAL;
; DECLARE MSG$PTR POINTER;
; END MSG$OUT;
; /* NOTE: MESSAGE IS TERMINATED WITH A DOLLAR SIGN */
; MSG$PTR is double word pointer SEG:OFFSET

MSG_L           EQU     [BP+4]
MSG_H           EQU     [BP+6]
;
MSGOUT          PROC    NEAR
        PUBLIC  MSGOUT
;
        PUSH    BP
        MOV     BP,SP
        MOV     DX,MSG_L
        PUSH    DS
        MOV     AX,MSG_H
        MOV     DS,AX
        MOV     AH,9                    (DOS call 9)
        INT     DOS
        POP     DS
        POP     BP
        RET     4
MSGOUT          ENDP

;
; LINE$IN: PROCEDURE(LINE$PTR) EXTERNAL;
; DECLARE LINE$PTR POINTER;
; END LINE$IN
;

LINE_L          EQU     [BP+4]
LINE_H          EQU     [BP+6]
;
LINEIN          PROC    NEAR
        PUBLIC  LINEIN
        PUSH    BP
        MOV     BP,SP
        PUSH    DS
        MOV     AX,LINE_H
        MOV     DS,AX
        MOV     DX,LINE_L
        MOV     AH,10                   (DOS call 10)
        INT     DOS
        POP     DS
        POP     BP
        RET     4
LINEIN          ENDP                                    231422-60


;
; KEY$STAT: PROCEDURE BYTE EXTERNAL;
; END KEY$STAT;
; Indicates whether any keyboard key was pressed.

KEYSTAT         PROC    NEAR
        PUBLIC  KEYSTAT
        MOV     AH,11                   (DOS call 11)
        INT     DOS
        RET
KEYSTAT         ENDP
;
;
CODE            ENDS
        END
                                                        231422-61
```

Figure 56. I/O Routines for IBM/PC (Continued)

7.2 Initialization and Declarations

Figure 57 shows some declarations describing what addresses the devices have and also some literals to help understand the other routines in this section.

Figure 58 shows the initialization routines for the IBM PC and for the 82588. It also shows some of the typical values taken by the memory buffers for Configure, IA_Set, Multicast and transmit buffers.

Following are some literal declarations that are used in the procedure examples

```
Following are some literal declarations that are used in the
procedure examples
  declare

    os_588          literally '0300h' , /* 82588 COMMAND/STATUS        */
    brd_port            literally '0301h' , /* DMA/INTERUPT ENABLE PORT   */
    pic_mask            literally '021h'  , /* 8259A MASK REGISTER        */
    pic_ocw2            literally '020h'  , /* 8259A COMMAND WORD 2       */
    dma_mask            literally '0ah'   , /* 8237A MASK REGISTER        */
    dma_mode            literally '0bh'   , /* 8237A MODE REGISTER        */
    dma_f1ff            literally '0ch'   , /* 8237A 1ST/2ND BYTE FLOP    */
    dma_addr_1          literally '02h'   , /* 8237A CHANNEL 1 ADDR. REG. */
    dma_bc_1            literally '03h'   , /* 8237A CHANNEL 1 BYTE COUNT */
    dma_addrh_1         literally '083h'  , /* CHANNEL 1 PAGE REGISTER    */
    dma_addr_3          literally '06h'   , /* 8237A CHANNEL 3 ADDR. REG. */
    dma_bc_3            literally '07h'   , /* 8237A CHANNEL 3 BYTE COUNT */
    dma_addrh_3         literally '082h'  , /* CHANNEL 3 PAGE REGISTER    */
    dma_on_1            literally '01h'   , /* START CHANNEL 1            */
    dma_on_3            literally '03h'   , /* START CHANNEL 3            */
    dma_off_1           literally '05h'   , /* STOP CHANNEL  1            */
    dma_off_3           literally '07h'   , /* STOP CHANNEL  3            */
    enable_588          literally '0dfh'  , /* UNMASK INTERRUPT LEVEL 5   */
    seoi_pico           literally '065h'  , /* SPECIFIC EOI LEVEL 5       */
    tx_dir              literally   '1'   , /* MEMORY TO 82588            */
    rx_dir              literally   '0'   , /* 82588 TO MEMORY            */
    dma_rx_mode_1       literally '045h'  , /* RX ON CHANNEL # 1          */
    dma_rx_mode_3       literally '047h'  , /* RX ON CHANNEL # 3          */
    dma_tx_mode_1       literally '049h'  , /* TX ON CHANNEL # 1          */
    dma_tx_mode_3       literally '04bh'  , /* TX ON CHANNEL # 3          */
                                                                  231422-62
```

Figure 57. Literal Declarations

Initialization Routines

```
Initialization routines

/* SYSTEM INITIALIZE */

sys_init: procedure;

   call set$interrupt (13,intr_588);                  /* BASE 8, LEVEL 5    */
   output(pic_mask) = input(pic_mask) and enable_588; /* ENABLE 588 INTERR. */
   output(pic_ocw2) =    seoi_pico;                   /* ACKS PENDING INTERR*/

   wr_ptr,rd_ptr,fifoont=0;                           /* RESET STATUS FIFO  */

   /*************************************************/
   /* CONVERT SEG:OFFSET FORMAT TO 20 BIT ADDRESSES  */
   /* FOR ALL THE BUFFERS                            */
   /*************************************************/

   iaset_dma_addr = convert_20bit_addr(@ia_set_buff_588(0));
   onf_dma_addr   = convert_20bit_addr(@config_588(0));
   dmp_dma_addr   = convert_20bit_addr(@dump_buff_588(0));
   mc_dma_addr    = convert_20bit_addr(@multicast_buff_588(0));
   tx_dma_addr    = convert_20bit_addr(@tx_buffer_588(0));
   do i=0 to 7 ;
     rx_dma_addr(i)=convert_20bit_addr(@rx_buffer(i).buff(0));
   end;

   output(brd_port)=0ffh;  /* ENABLE DMA AND INTERRUPT DRIVERS */

   end sys_init;

82588 initialization

init_588: procedure;

  config_588(00) = 10;          /* TO CONFIGURE ALL 10 PARAMETERS          */
  config_588(01) = 00;
  config_588(02  = 00001000b;   /* MODE 0, 8  MHZ CLOCK, 1 MB/S            */
  config_588(03) = buff_len/4;  /* RECEIVE BUFFER LENGTH                   */
  config_588(04) = 00100110b;   /* NO LOOPBACK, ADDR LEN = 6, PREAMBLE = 8 */
  config_588(05) = 00000000b;   /* DIFFERENTIAL MANCHESTER = OFF           */
  config_588(06) = 96;          /* IFS = 96 TCLK                           */
  config_588(07) = 0;           /* SLOT TIME = 512 TCLK                    */
  config_588(08) = 11110010b;   /* MAX. NO. RETRIES = 15                   */
  config_588(09) = 00000100b;   /* MANCHESTER ENCODING                     */
  config_588(10) = 10001000b;   /* INTERNAL CRS AND CDT, CRSF = 0          */
  config_588(11) = 64;          /* MIN FRAME LENGTH = 64 BYTES = 512 BITS  */
                                                                  231422-63
```

Figure 58. Initialization Routines

```
ia_set_buff_588(0) = 6;
ia_set_buff_588(1) = 0;
ia_set_buff_588(2) = 000h;
ia_set_buff_588(3) = 041h;
ia_set_buff_588(4) = 000h;
ia_set_buff_588(5) = 000h;
ia_set_uff_588(6)  = 000h;
ia_set_buff_588(7) = 000h;

multicast_buff_588(00) = 12;
multicast_buff_588(01) = 00h;
multicast_buff_588(02) = 11h;
multicast_buff_588(03) = 12h;
multicast_buff_588(04) = 13h;
multicast_buff_588(05) = 14h;
multicast_buff_588(06) = 15h;
multicast_buff_588(07) = 16h;
multicast_buff_588(08) = 21h;
multicast_buff_588(09) = 22h;
multicast_buff_588(10) = 23h;
multicast_buff_588(11) =24h;
multicast_buff_588(12) = 25h;
multicast_buff_588(13) = 26h;

tx_buffer_588(00) = tx_frame_len mod 256;
tx_buffer_588(01) = tx_frame_len / 256;
tx_buffer_588(02) = 011h;   /* INITIAL DESTINATION ADDRESS - MC(1) */
tx_buffer_588(03) = 012h;
tx_buffer_588(04) = 013h;
tx_buffer_588(05) = 014h;
tx_buffer_588(06) = 015h;
tx_buffer_588(07) = 016h;

end init_588;
```

231422-64

Figure 58. Initialization Routines (Continued)

7.3 General Commands

Operations like Transmit, Receive, Configure, etc. are done by a simple sequence of loading the DMA controller with the necessary parameters and then writing the command to the 82588.

Example: Configure Command

To configure the operating environment of the 82588. This command must be the first one to be executed after a RESET.

```
call
DMA_LOAD(1,1,12,@CONFIG_588_ADDR);
output (CS_588) = 12h;
```

The first statement is the prologue to the configure command to the 82588 which calls a routine to load and initialize the DMA controller for the desired operation. This routine is described in section 7.4. The parameters for DMA__LOAD are:

```
first parameter  = 82588 channel
                   number ( = 1)
second parameter = direction ( = 1,
                   memory >> 82588)
third parameter  = length of DMA
                   transfer ( = 12)
fourth parameter = pointer to a 20 bit
                   address of the
                   memory buffer
                   (=@CONFIG_588_ADDR)
```

The second statement writes 12h to the command register of the 82588 to execute a Configure command on channel 1.

When the command execution is complete (successfully or not), 82588 interrupts the 8088 CPU through the 8259A, on the system board. This executes the interrupt service routine, described in section 7.5, which takes the epilogue action for the command.

Most operations are very similar in structure to Configure. The 82588 Reference Manual describes them in detail. Figure 59 shows a listing of the most commonly used operations like:

```
CONFIGURE     INDIVIDUAL-ADDRESS (IA)
              SET-UP
TRANSMIT      MULTICAST-ADDRESS (MC)
              SET-UP
DIAGNOSE      RECEIVE (RCV)-ENABLE
DUMP          RECEIVE (RCV)-DISABLE
TDR           RECEIVE (RCV)-STOP
RETRANSMIT    READ-STATUS
```

```
ia_set: procedure public;                    /* COMMAND - 01 */

    call dma_load(cmd_channel,tx_dir,8,@iaset_dma_addr);

    /* SET DMA CHANNEL 0 OR 1 TO TRANSFER FROM MEMORY
       TO THE 82588. iaset_dma_addr VARIABLE STORES THE
       20 BIT POINTER TO THE INDIVIDUAL ADDRESS BUFFER  */

    if cmd_channel then output (cs_588) = 11h;
    else output(cs_588) = 01h;

    /* EVERY COMMAND CAN BE EXECUTED IN EITHER DMA CHANNEL 0 OR 1.
       THE VARIABLE cmd_channel INDICATES THE REQUIRED CHANNEL  */

end ia_set;

/*-------------------------------------------------------------------------*/

config: procedure public;                    /* COMMAND - 02 */

    call dma_load(cmd_channel,tx_dir,12,@cnf_dma_addr);
    if cmd_channel then output (cs_588) = 12h;
    else output(cs_588) = 02h;

end config;

/*-------------------------------------------------------------------------*/

multicast: procedure public;                 /* COMMAND - 03 */

    call dma_load(cmd_channel,tx_dir,14,@mc_dma_addr);
    if cmd_channel then output (cs_588) = 13h;
    else output(cs_588) = 03h;

end multicast;

/*-------------------------------------------------------------------------*/

transmit: procedure(buffer_len) public;      /* COMMAND - 04 */

     declare buffer_len word;

    tx_buffer_588(00) = low(buffer_len);
    tx_buffer_588(01) = high(buffer_len);

    call dma_load(cmd_channel,tx_dir,1536,@tx_dma_addr);

    if cmd_channel then output (cs_588) = 14h;
    else output(cs_588) = 04h;

end transmit;
```

231422-65

Figure 59. General Commands

```
tdr: procedure public;                          /* COMMAND - 05 */

    if cmd_channel then output (cs_588) = 15h;
    else output(cs_588) = 05h;

end tdr;

/*-------------------------------------------------------------------------*/

dump_588: procedure public;                     /* COMMAND - 06 */

    call dma_load(cmd_channel,rx_dir,64,@dmp_dma_addr);
    if cmd_channel then output (cs_588) = 16h;
    else output(cs_588) = 06h;

end dump_588;

/*-------------------------------------------------------------------------*/

diagnose: procedure public;                     /* COMMAND - 07 */

    if cmd_channel then output (cs_588) = 17h;
    else output(cs_588) = 07h;

end diagnose;

/*-------------------------------------------------------------------------*/

rcv_enable: procedure(channel,buffer_no,len) public;    /* COMMAND - 08 */

    declare channel byte;
    declare len  word;
    declare buffer_no byte;
    call dma_load(channel,rx_dir,len,@rx_dma_addr(buffer_no));
    if rx_channel then output (cs_588) = 18h;
    else output(cs_588) = 08h;

end rcv_enable;

/*-------------------------------------------------------------------------*/

rcv_disable: procedure public;                  /* COMMAND - 10 */

    enable_rcv=0;
    output(cs_588)=0ah;

end rcv_disable;                                                    231422-66

/*-------------------------------------------------------------------------*/

rcv_stop: procedure public;                     /* COMMAND - 11 */

    enable_rcv=0;
    output(cs_588)= 0bh;

end rcv_stop;

/*-------------------------------------------------------------------------*/

retransmit: procedure public;                   /* COMMAND - 12 */

    call dma_load(cmd_channel,tx_dir,1536,@tx_dma_addr);
    if cmd_channel then output (cs_588) = 1ch;
    else output(cs_588) = 0ch;

end retransmit;

/*-------------------------------------------------------------------------*/

abort: procedure public;                        /* COMMAND - 13 */

    output(cs_588)= 1dh;
    call new_status(1);

end abort;

 *-------------------------------------------------------------------------*/

reset_588: procedure public;                    /* COMMAND - 14 */

    enable_rcv=0;
    output(cs_588) = 1eh;
    call config;

end reset_588;
                                                                    231422-67
```

Figure 59. General Commands (Continued)

7.4 DMA Routines

DMA_LOAD procedure is used to program the 8237A DMA controller for all the operations requiring DMA service. It also starts or enables the programmed DMA channel after programming it. Figure 60 shows the listing of this procedure. It accepts 4 parameters from the calling routine to decide the programming configuration for the 8237A. The parameters for DMA_LOAD are: Channel, direction, buff_len, and buff_addr.

```
Converting a pointer SEG:OFFSET to a 20 bit address
convert_20bit_addr:procedure(ptr) dword public;

   declare   ptr       pointer,
             ptr_addr  pointer,
             ptr_20bit dword   ,
             (wrd  based ptr_addr)(2) word;

   ptr_addr=@ptr;
   ptr_20bit=shl((ptr_20bit:=wrd(1)),4)+wrd(0);
   return(ptr_20bit);

end convert_20bit_addr;


IBM/PC DMA loading procedure

dma_load: procedure(channel,direction,buff_len,buff_addr) reentrant public;

    declare channel byte;            /* CHANNEL #, 0 or 1                  */
    declare direction byte;          /* 0=RX, 588 -> MEM; 1=TX, MEM -> 588 */
    declare buff_len word;           /* BYTE COUNT                         */
    declare buff_addr pointer;       /* BUFFER ADDR IN 20 BITS FORM        */
    declare (wrd based buff_addr)(2) word;

    channel=channel and 1;           /* GET LEAST SIGNIFICANT BIT          */

    if channel=0 then                /* EXECUTE COMMAND ON CHANNEL 1       */
      do;
      output(dma_flff)  = 0;         /* CLEAR FIRST/LAST FLIP-FLOP         */
      if direction=0
          then output(dma_mode)=dma_rx_mode_1;  /* DIRECTION BIT, TELLS    */
      else output(dma_mode)=dma_tx_mode_1;      /* TRANSMIT OR RECEIVE     */
      output(dma_addr_1)  = low (wrd(0));       /* LOAD LSB ADDRESS BYTE   */
      output(dma_addr_1)  = high(wrd(0));       /* LOAD MSB ADDRESS BYTE   */
      output(dma_addrh_1) = low (wrd(1));       /* LOAD PAGE REGISTER      */
      output(dma_bc_1)    = low (buff_len);     /* LOAD LSB BYTE COUNT     */
      output(dma_bc_1)    = high(buff_len);     /* LOAD MSB BYTE COUNT     */
      output(dma_mask) = dma_on_1;              /* START CHANNEL 1         */
      end;
    else do;                         /* SAME AS BEFORE FOR CHANNEL 3       */
      output(dma_flff)  = 0;
      if direction=0
          then output(dma_mode)=dma_rx_mode_3;
      else output(dma_mode)=dma_tx_mode_3;
      output(dma_addr_3)  = low (wrd(0));
      output(dma_addr_3)  = high(wrd(0));
      output(dma_addrh_3) = low (wrd(1));
      output(dma_bc_3)    = low (buff_len);
      output(dma_bc_3)    = high(buff_len);
      output(dma_mask) = dma_on_3;
      end;

end dma_load;
```

231422-68

Figure 60. DMA Routine

One peculiarity about this procedure is that in order to speed up the DMA step-up, this procedure doesn't get a pointer to the buffer, but a pointer to a 20 bit address in the 8237 format. The 8088/8086 architecture define pointers as 32 bits seg:offset entities, where seg and offset are 16 bit operands. By the other hand the IBM/PC uses an 8237A and a page register, requiring a memory address to be a 20 bit entity. The process of converting a seg:offset pointer to a 20 bit address is time consuming and could negatively affect the performance of the 82588 driver software. The decision was to make the pointer/address conversions during initialization, considering that the buffers are static in memory (essentially removing this calculation from the real time response loops).

Figure 61 is a listing of the DMA_LOAD procedure for the 80188 or 80188 on-chip DMA controller. It has the same caller interface as the 8237A based one.

```
dma_load: procedure(channel,direction,trans_len,buff_addr) reentrant;

/* To load and start the  80186 DMA controller for the desired operation */

  declare dma_rx_mode   literally '10100010010000000b'; /* rx channel */
        /* src=IO, dest=M(inc), sync=src, TC, noint, priority, byte */

  declare dma_tx_mode   literally '0000110100000000b'; /* tx channel */
        /* src=M(inc), dest=IO, sync=dest, TC, noint, noprior, byte */


  declare channel byte;       /* channel #                              */
  declare direction byte;     /* 0 = rx, 588 -> mem; 1 = tx, mem -> 588 */
  declare trans_len word;     /* byte count                             */
  declare buff_addr pointer; /* buffer pointer in 20 bit addr. form     */

  declare (wrd based buff_addr)(2) word;

  do case channel and 00000001b;
    do case direction and 00000001b;
      do;                /* channel 0, 588 to memory  */
      output(dma_0_dpl)  = wrd(0);
      output(dma_0_dph)  = wrd(1);
      output(dma_0_spl)  = ch_a_588;
      output(dma_0_sph)  = 0;
      output(dma_0_tc)   = trans_len;
      output(dma_0_cw)   = dma_rx_mode or 0006h; /* Start DMA ch1 0 */
      end;

      do;              /* channel 0, memory to 588 */
      output(dma_0_dpl)  = ch_a_588;
      output(dma_0_dph)  = 0;
      output(dma_0_spl)  = wrd(0);
      output(dma_0_sph)  = wrd(1);
      output(dma_0_tc)   = trans_len;
      output(dma_0_cw)   = dma_tx_mode or 0006h; /* Start DMA ch1 0 */
      end;
    end;
```

231422-69

Figure 61. 80186 DMA Routines

```
    do case direction and 00000001b;
        do;                 /* channel 1, 588 to memory  */
        output(dma_1_dpl)   = wrd(0);
        output(dma_1_dph)   = wrd(1);
        output(dma_1_spl)   = ch_b_588;
        output(dma_1_sph)   = 0;
        output(dma_1_tc)    = trans_len;
        output(dma_1_cw)    = dma_rx_mode or 0006h; /* Start DMA chl 1 */
        end;

        do;                 /* channel 1, memory to 588 */
        output(dma_1_dpl)   = ch_b_588;
        output(dma_1_dph)   = 0;
        output(dma_1_spl)   = wrd(0);
        output(dma_1_sph)   = wrd(1);
        output(dma_1_tc)    = trans_len;
        output(dma_1_cw)    = dma_tx_mode or 0006h; /* Start DMA chl 1 */
        end;
      end;

    end;

end dma_load;
```

231422-70

Figure 61. 80186 DMA Routines (Continued)

7.5 Interrupt Routine

The interrupt service routine, 'intr_588', shown in Figure 62, is invoked whenever the 82588 interrupts. The main difficulty in designing this interrupt routine was to speed its performance. Fast status processing was a basic requirement to be able to handle back to back frames.

The interrupt handler will read 82588 status, and put them into a 64 byte long EVENT_FIFO. Those statuses are going to be used in the main loop for updating screen counters. All the statistics are updated as fast as possible in the interrupt handler to fulfill the back-to-back frame processing requirement.

The interrupt handler is not reentrant, interrupts are disabled at the beginning and reenabled on exit.

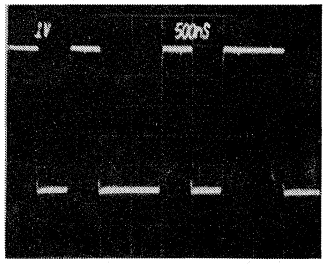
```
Interrupt service routine

intr_588:procedure interrupt 13;

    declare stat                    byte,
            event                   byte,
            i                       byte,
            (st0,st1,st2,st3)       byte,
            rx_st0                  byte,
            rx_st1                  byte;

   /* FOLLOWING LITERALS HAVE THE PURPOSE OF ENABLE ACTING
      ON EITHER CHANNEL 1 OR 3 SELECTIVELY                    */

   declare

     stop_cmd_dma  literally 'if cmd_channel
                                then output(dma_mask)=dma_off_3;
                              else output(dma_mask)=dma_off_1',
     stop_rx_dma   literally 'if rx_channel
                                then output(dma_mask)=dma_off_3;
                              else output(dma_mask)=dma_off_1',

     issue_rtx_cmd literally 'if cmd_channel
                                then output(os_588)=1Ch;
                              else output(os_588)=0ch',
     issue_tx_cmd  literally 'if cmd_channel
                                then output(os_588)=14h;
                              else output(os_588)=04h';

   disable;                                   /* DISABLE INTERRUPTS   */
                                              /* NO INTERR. NESTING   */
   output(os_588)  =0fh;                      /* RLS 588 PTR, START 0 */

   event_fifo(wr_ptr).st0,st0=input(os_588); /* READ 82588 STATUS    */
   event_fifo(wr_ptr).st1,st1=input(os_588); /* REGISTERS, PASSING   */
   event_fifo(wr_ptr).st2,st2=input(os_588); /* THEM TO THE MAIN     */
   event_fifo(wr_ptr).st3,st3=input(os_588); /* PROGRAM ON THE FIFO  */

   wr_ptr=(wr_ptr+1) and 0fh;                 /* INCREMENT FIFO       */
   fifoont=(fifoont+1) and 0fh;               /* COUNTERS             */

   event=st0 and 0fh;                         /* GET EVENT FIELD      */

   output(os_588)=80h;                        /* ACKNOWLEDGE 82588    */
                                              /* INTERRUPT            */   231422-71


  do case event;

  ev_00: ;                          /* NOP COMMAND          */
  ev_01: stop_cmd_dma;              /* IA_SETUP, STOP DMA   */
  ev_02: stop_cmd_dma;              /* CONFIGURE, STOP DMA  */
  ev_03: stop_cmd_dma;              /* MULTICAST, STOP DMA  */
  ev_04: do;                        /* TRANSMIT DONE        */
         stop_cmd_dma;

         /* CHECK IF THERE WAS A COLLISION AND IS NOT THE
            MAX COLLISION                                   */

         stat=(st2 and 10000000b) or (st1 and 00100000b);
         if (stat=80h)
            then do;                /* RETRANSMIT           */
                 call dma_load(cmd_channel,tx_dir,1536,@tx_dma_addr);
                 issue_rtx_cmd;
                 /* UPDATE STATISTICS                      */
                 total_tx_ocount=total_tx_ocount+1;
                 coll_cnt(17) = coll_cnt(17) + 1; /*TOTAL COLL*/
                 bad_tx_ocount = bad_tx_ocount + 1;
                 end;
         else do;
              if in_loop    /* EXECUTING TRANSMISSIONS IN LOOP  */
               then do;     /* RE ISSUE TRANSMIT COMMAND        */
                    call dma_load(cmd_channel,tx_dir,1536,@tx_dma_addr
                    issue_tx_cmd;
                    total_tx_ocount=total_tx_ocount+1;
                    end;
              if (st2 and 00100000b) = 0         /* BAD TRANSMIT*/
                then do;
                     bad_tx_ocount = bad_tx_ocount + 1;
                     /* INCREMENT UNDERRUN COUNTER          */
                     tmp=scr(tmp:=st2,1);
                     tx_under=tx_under plus 0;
                     /* INCREMENT LOST CTS COUNTER          */
                     tmp=scr(tmp,1);
                     lost_cts=lost_cts plus 0;
                     /* INCREMENT LOST CRS COUNTER          */
                     tmp=scr(tmp,1);
                     lost_crs=lost_crs plus 0;
                     if (stat=0A0h)  /* INC COLLISIONS COUNTER  */
                        then coll_cnt(17) = coll_cnt(17) + 1;
                     end;
              end;

              /* INCREMENT DEFER  COUNTER                  */
              tmp=scl((tmp:=st1),1);
              tx_defer=tx_defer plus 0;
        end;                                                    231422-72
```

Figure 62. Interrupt Routine

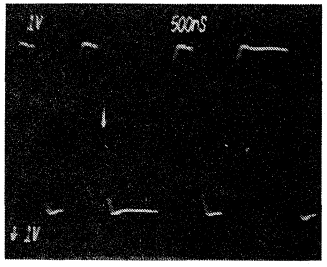
```
        ev_05: stop_cmd_dma;              /* TDR COMMAND, STOP DMA   */
        ev_06: stop_cmd_dma;              /* DUMP COMMAND, STOP DMA  */
        ev_07: stop_cmd_dma;              /* DIAGNOSE CMD, STOP DMA  */
        ev_08:                            /* RECEIVED FRAME          */
            do;
            stop_rx_dma;
            i=(current_buff+1) and 00000111b; /* INC BUFFER NO. MOD 8*/
            if enable_rov<>0              /* IF RECEIVER IS ON       */
                then do;                  /* PREPARE NEXT BUFFER */
                     call dma_load(rx_channel,rx_dir,1532,@rx_dma_addr(i));
                     if rx_channel then output(os_588)= 18h;
                     else output(os_588)=08h;
                     rx_buffer(i).chain_cnt=0;
                     end;
            else call rov_disable;        /* DISABLE RECEIVER    */

            /* FIND ADDRESS OF END OF CURRENTLY RECEIVED BUFFER      */
            /* BY CALCULATING IT WITH THE 82588 BYTE COUNT REGS.     */
            rx_buff_off=(shl(double(st2),8) or double(st1));
            /* READ STATUS BYTES FROM MEMORY                         */
            rx_st0=rx_buffer(current_buff).buff(rx_buff_off-2);
            rx_st1=rx_buffer(current_buff).buff(rx_buff_off-1);
            /* UPDATE ACTUAL BUFFER SIZE                             */
            rx_buffer(current_buff).actual_size=rx_buff_off;
            rx_buffer(current_buff).st0=rx_st0;
            rx_buffer(current_buff).st1=rx_st1;
            current_buff=i;
            /* UPDATE TOTAL RECEIVED BUFFERS                         */
            total_rov_count=total_rov_count+1;
            /* UPDATE STATISTICS                                     */
            if (rx_st1 and 00100000b)=0
                then do;
                     bad_rov_count=bad_rov_count+1;
                     /* INCREMENT  NO END OF FRAME COUNTER           */
                     tmp=sor(tmp:=rx_st0,7);
                     no_eof=no_eof plus 0;
                     /* INCREMENT  SHORT FRAME COUNTER               */
                     tmp=sor(tmp,1);
                     srt_frm=srt_frm plus 0;
                     /* INCREMENT  RX OVERRUN COUNTER                */
                     tmp=sor(tmp:=rx_st1,1);
                     rx_over=rx_over plus 0;
                     /* INCREMENT  ALIGNMENT ERROR COUNTER           */
                     tmp=sor(tmp,2);
                     alg_err=alg_err plus 0;
                     /* INCREMENT  CRC ERROR COUNTER                 */
                     tmp=sor(tmp,1);
                     crc_err=crc_err plus 0;
                     end;
            end;                                                  231422-73


             /*  EV_09 REQUESTS ASSIGNMENT OF A NEW BUFFER        */
            ev_09:  call allocate_new_buffer(not(rol(st3,1)) and 00000001b);
            ev_10:  stop_rx_dma;      /* RECEIVE DISABLE          */
            ev_11:  stop_rx_dma;      /* STOP RECEIVE             */
            ev_12:  do;               /* RE-TRANSMIT DONE         */
                    stat=(st2 and 10000000b) or (st1 and 00100000b);
                    if (stat=80h)
                    then do;                  /* RETRANSMIT       */
                         call dma_load(1,tx_dir,1536,@tx_dma_addr);
                         issue_rtx_cmd;
                         coll_cnt(17) = coll_cnt(17) + 1;
                         total_tx_count=total_tx_count+1;
                         bad_tx_count=bad_tx_count +1;
                         end;
                    else do;
                         if in_loop
                          then do;   /* LOOP RETRANSMISSIONS     */
                               call dma_load(cmd_channel,tx_dir,1536,@tx_dma_a
                               issue_tx_cmd;
                               total_tx_count=total_tx_count+1;
                               end;
                         if (stat=0A0h) /* MAX COLLISION          */
                            then do;
                                 coll_cnt(16) = coll_cnt(16)+1;
                                 coll_cnt(17) = coll_cnt(17)+1;
                                 bad_tx_count=bad_tx_count +1;
                                 end;
                         /* UPDATE SPECIFIC COLLISION COUNTER     */
                         else  coll_cnt(st1 and 0fh)
                                    = coll_cnt(st1 and 0fh) + 1;
                         end;
                    end;
            ev_13:  stop_cmd_dma;     /* EXECUTION ABORTED        */
            ev_14:  ;
            ev_15: stop_cmd_dma;      /* DIAGNOSE FAILED          */
            end;

        /*  ACKNOWLEDGE 8259A  INTERRUPT                          */
        output(pic_ocw2)= seoi_pico;  /* SPECIFIC EOI FOR 8259    */

    end intr_588;                                                 231422-74
```

Figure 62. Interrupt Routine (Continued)

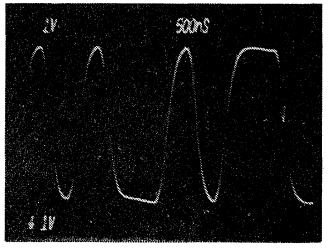
APPENDIX A
STARLAN SIGNALS



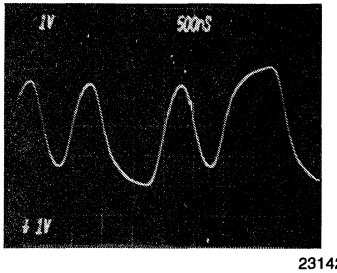
231422-51



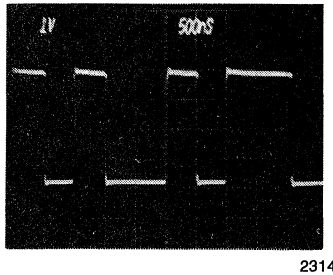
231422-52



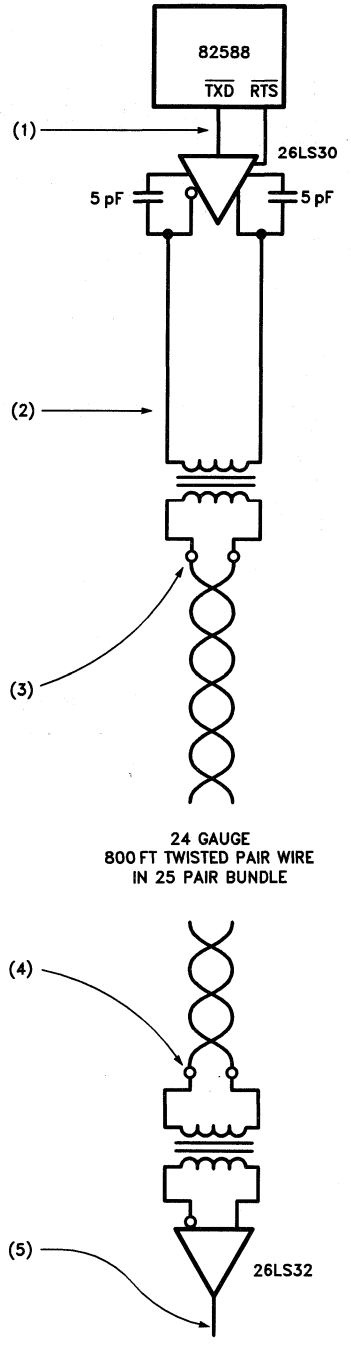
231422-53



231422-54



231422-55



231422-47

Figure 63. StarLAN Signals

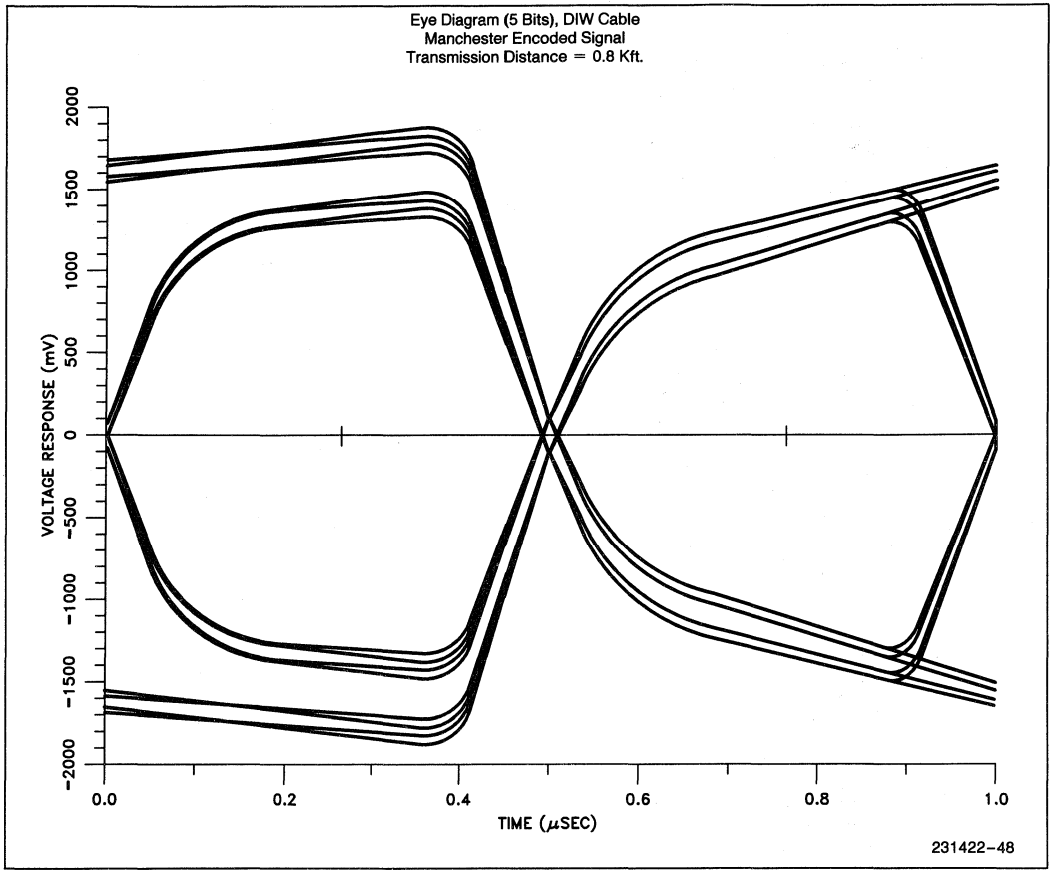


Figure 64. Received Signal Eye Diagram

APPENDIX B
802.3 1BASE5 MULTI-POINT EXTENSION (MPE)

As previously stated, one of the most important advantages of StarLAN is being able to work on already installed phone wires. This advantage is considerably diminished in Europe where numerous constraints exist to the using of those wires:

1. Wire belongs to local PTTs.
2. Not enough spare wires.

This same issue is raised when talking about small businesses where in a lot of cases no wiring closets and/or spare wires are available.

In summary, in a lot of cases rewiring will be necessary, in which case the STAR topology may not be the most economical one.

Recently the StarLAN 802.3 1BASE5 task force has been considering the extension of the StarLAN base topology. This extension called MULTI POINT EXTENSION (MPE) is going to be developed to address the previously described marketing requirements.

Currently no agreement has been reached by the StarLAN task force on the MPE exact topology and implementation. Multiple approaches have been presented, but no consensus met. It was decided though that the MPE is going to be an addendum to the STAR topology, and that its final specification will happen after the approval of the current 1BASE5 STAR topology (July 1986).

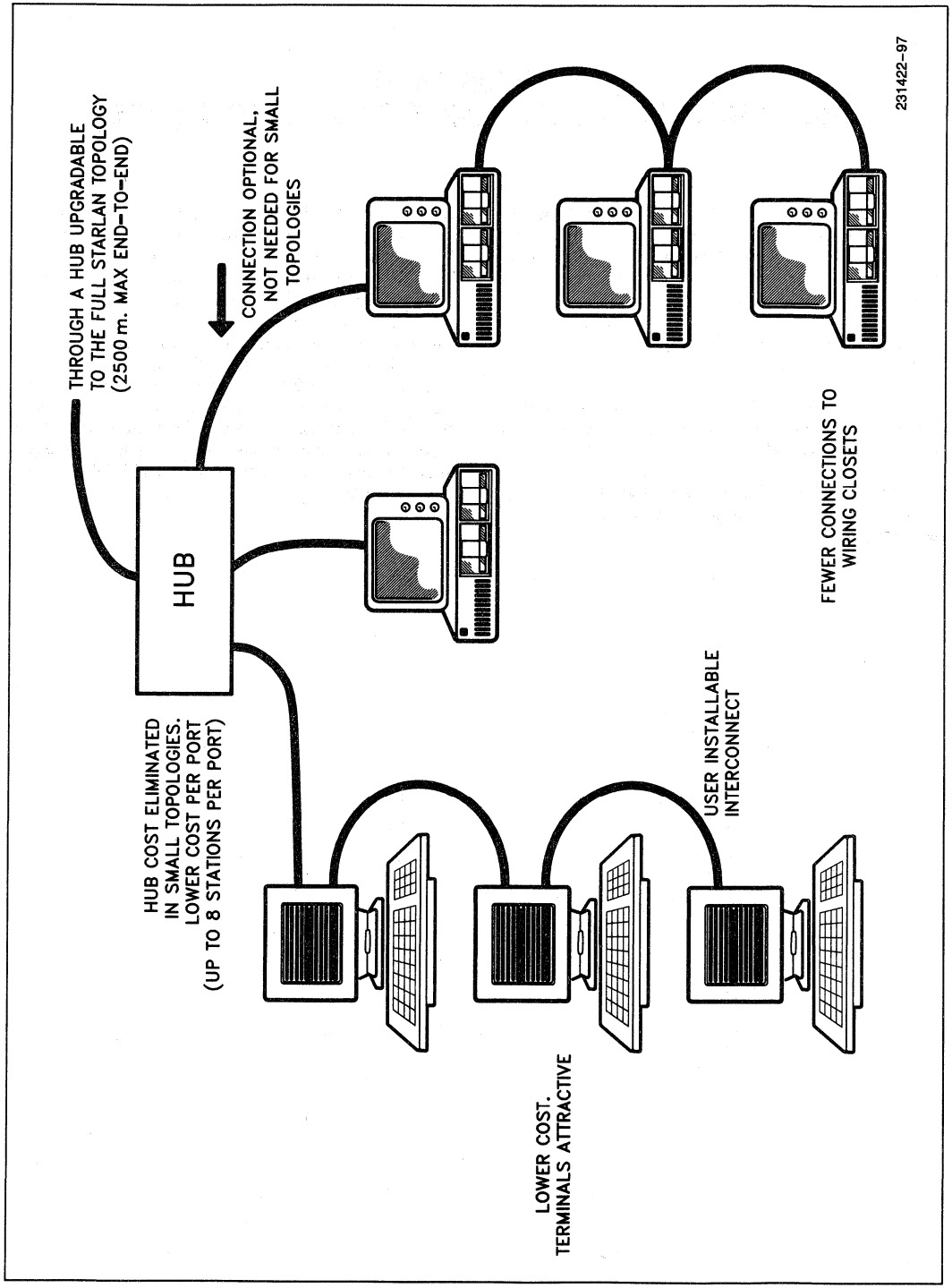


Figure 65. Multipoint Extension

APPENDIX C
SINGLE DMA CHANNEL INTERFACE

In a typical system, the 82588 needs 2 DMA channels to operate in a manner that no received frames are lost as discussed in section 5.1.3. If an existing system has only one DMA channel available, it is still possible to operate the 82588 in a way that no frames are lost. This method is recommended only in situations where a second DMA channel is impossible to get.

Figure 66 shows how the 82588 DMA logic is interfaced to one channel of a DMA controller. Two DRQ lines are ORed and go to the DMA controller DRQ line and the DACK line from the DMA controller is connected to DACK0 and DACK1 of the 82588. The 82588 is configured for multiple buffer reception (chaining), although the entire frame is received in a single buffer. Let us assume that channel CH-0 is used as the first channel for reception. After the ENAble RECeive command, CH-0 is dedicated to reception. As long as no frame is received, the other channel, CH-1, can be used for executing any commands like transmit, multicast address, dump, etc., by programming the DMA channel for the execution command. The status register should be checked for any ongoing reception, to avoid issuing an execution command when reception is active.

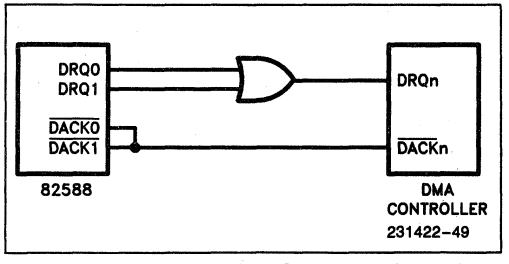


Figure 66. 82588 Using One DMA Channel

If a frame is received, an interrupt for additional buffer occurs immediately after an address match is established, as shown in Figure 67. After this, the received bytes start filling up the on-chip FIFO. The 82588 activates the DRQ line after 15—FIFO LIMIT + 3 bytes are ready for transfer in the FIFO (about 80 microseconds after the interrupt). The CPU should react to the interrupt within 80 μs and disable the DMA controller. It should also issue an ASSIGN ALTERNATE BUFFER command with INTACK to abort any execution command that may be active. The FIFO fills up in about 160 μs after interrupt. To prevent an underrun, the CPU must reprogram the DMA controller for frame reception and re-enable the DMA controller within 160 μs after the interrupt (time to receive about 21 bytes). No buffer switching actually takes place, although the 82588 generates request for alternate buffer every time it has no additional buffer. The CPU must respond to these interrupts with an ASSIGN ALTERNATE BUFFER command with INTACK. To keep the CPU overhead to a minimum, the buffer size must be configured to the maximum value of 1 kbyte.

If a frame transmission starts deferring due to the reception occurring just prior to an issued transmit command, the transmission can start once the link is free after reception. A maximum of 19 bytes are transmitted (stored in the FIFO and internal registers) followed by a jam pattern and then an execution aborted interrupt occurs. The aborted frame can be transmitted again.

If the transmit command is issued and the 82588 starts transmitting just prior to receiving a frame then transmit wins over receive—but this will obviously lead to a collision.

Note that the interrupt for additional buffer is used to abort an ongoing execution command and to program the DMA channel for reception just when a frame is received. This scheme imposes real time interrupt handling requirements on the CPU and is recommended only when a second DMA channel is not available.

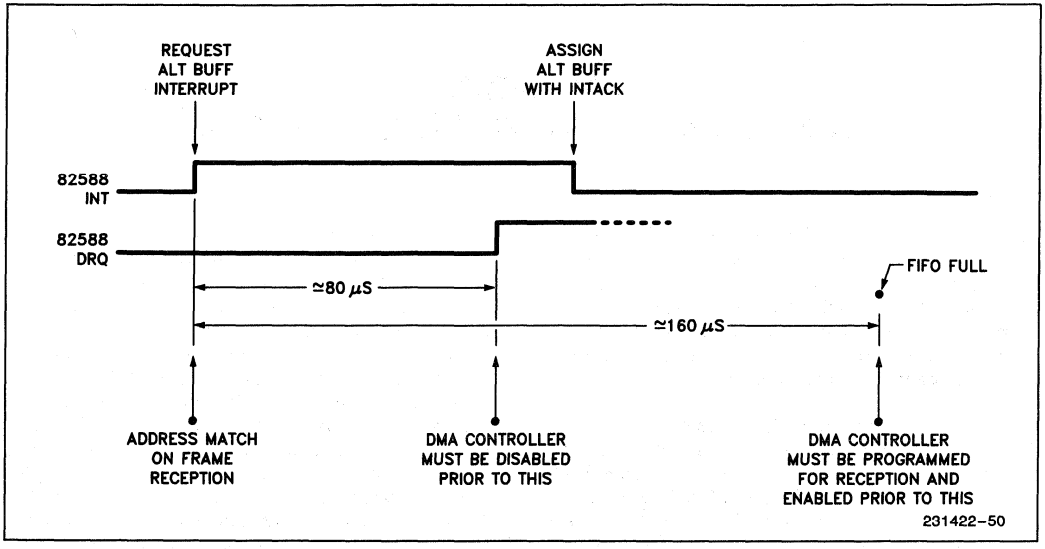


Figure 67. Timing at the Beginning of Frame Reception for Single DMA Channel Operation

APPENDIX D
MEASURING NETWORK DELAYS WITH THE 82588

Knowing networks round-trip delays in local area networks is an important capability. The round-trip delay very much defines the slot time parameter which by itself has a direct relationship to network efficiency and throughput. Very often the slot-time parameter is not flexible, due to standards requirements. Whenever it is flexible, optimization of this number may lead to significant improvement in network performance.

Another possible usage of the network delay knowledge is in balancing the inter-frame -spacing (IFS) on broadband networks. On those networks, stations nearer to the HEAD-END hear themselves faster than farther ones. Effectively having a shorter IFS than stations far from the HEAD-END. This difference causes an inbalance in network access time for different stations at different distances from the HEAD-END. Knowing the STATION/HEAD-END delay allows the user to reprogram the 82588 IFS accordingly, and by that balance the effective IFS for all the stations.

The 82588 has an internal mechanism that allows the user to measure this delay in BIT-TIME units. The method is based on the fact that the 82588 when configured for internal collision detection, requires that the carrier sense be active within half a slot-time after transmission has started. If this requirement is not fulfilled the 82588 notifies that a collision has occurred. Thus it is possible to configure the 82588 to different slot time values, then transmit a long frame (of at least half a slot-time). If the transmission succeeds, the network round-trip delay is less than half the programmed slot-time. If a collision is reported, the delay is longer. The value of the round-trip delay can be found by repeating this experiment process while scanning the slot-time configuration parameter value and searching the threshold. A binary search algorithm is used for that purpose. First the slot-time is configured for the maximum (2048 bits) and according if there was a collision or not, the number changed for the next try. (See Figure 68)

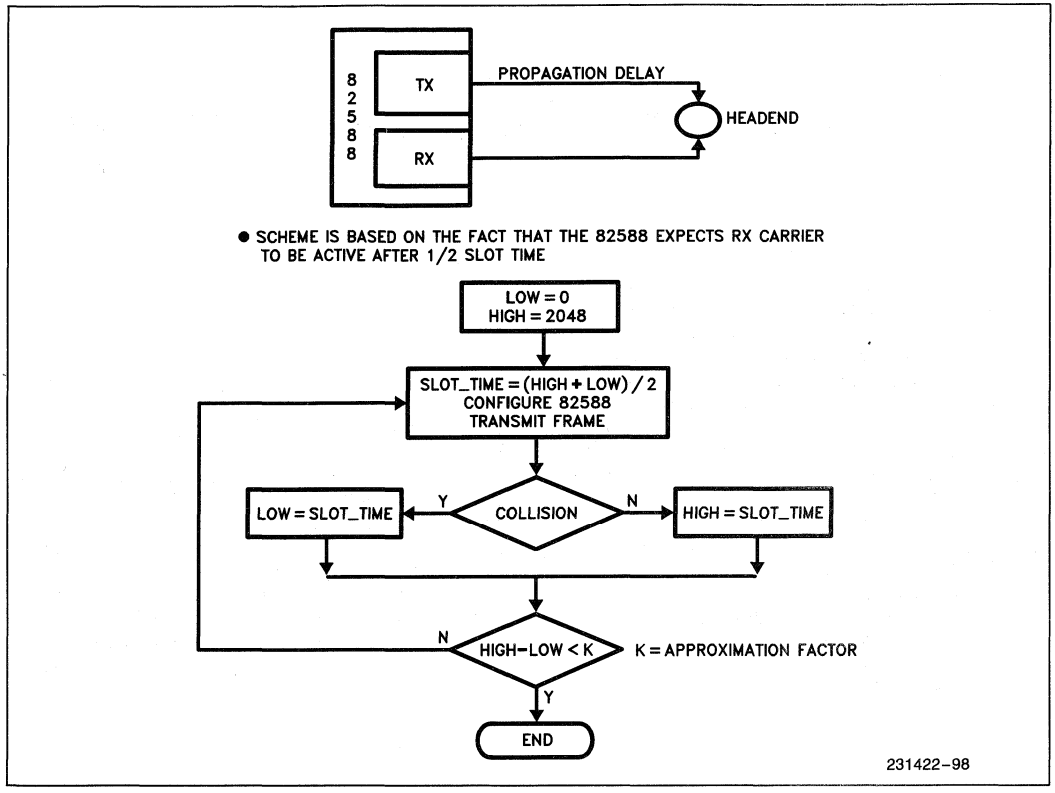


Figure 68. Network Delay Measurement using the 82588

82590/592 Reference Manual 6

82590 & 82592 ADVANCED CSMA/CD LAN CONTROLLER USER'S MANUAL

1988

PREFACE

Since the original Ethernet was introduced more than ten years ago, the CSMA/CD (Carrier Sense Multiple Access with Collision Detection) protocol has been widely used to implement Local Area Networks (LANs). In 1980 the IEEE accepted the Ethernet as the IEEE 802.3 CSMA/CD standard. Since then the IEEE 802.3 standard has been continually evaluated and revised as the LAN market develops. In order to satisfy different LAN environments, the standard now includes several implementation options—TYPE 1BASE5 (StarLAN), TYPE 10BASE5 (Ethernet), TYPE 10BASE2 (Cheapernet or Thin Ethernet), and TYPE 10BROAD36 (Broadband Ethernet). In addition to the IEEE 802.3 CSMA/CD LANs, there are other industry standard CSMA/CD LANs. One example is the IBM PC Network (Broadband and Baseband).

Intel has been involved in the CSMA/CD standard activities since the Ethernet standard was introduced. One of Intel's objectives in participating in these standard activities has been to provide optimal VLSI integration of LAN functions with state of the art technology. The Intel LAN VLSI products during the last several years include the 82586 LAN coprocessor, the 82C501 Ethernet serial interface chip, the 82502 Ethernet transceiver chip, and the 82588 high-integration LAN controller.

The 82590 controller family consists of the 82590 and the 82592. The 82590 is a second-generation, 8-bit, CSMA/CD LAN controller, both pin and software compatible with the 82588. It offers numerous performance and functional enhancements over the previous generation chip, the 82588. The 82592 is a 16-bit version of the 82590, offering the same functionality as the 82590 with a 16-bit data path. The 82590 and the 82592 are the latest additions to the 82588 LAN controller product line. This manual explains these two most advanced CSMA/CD LAN controllers.

Chapter 1 is an introductory chapter. Section 1.2 is provided for those users who are already familiar with the 82588. By reading this section first, those users will be able to save time in understanding the improved functionality of the 82590 and the 82592. Users who are not familiar with the 82588 are advised to skip Section 1.2. Chapters 2, 3, 4, and 5 describe the 82590 and the 82592 in terms of their capabilities. These chapters explain the 82590 and 82592 functions, but may not always explain how the functions are invoked. Chapters 6, 7, 8, 9, and 10 describe the 82590 and 82592 functions in terms of the on-chip registers and other control bits. Therefore, more precise definitions of the 82590 and 82592 functions are found in these chapters. Chapter 11 deals with the performance issues in using the 82590 or the 82592. Chapter 12 covers applications of the 82590 and the 82592.

This manual does not contain specifications for the 82590, the 82592, or the 82560. Refer to the following Data Sheets for the specifications.

82590 Advanced CSMA/CD LAN Controller With 8-Bit Data Path, Advance Information. Intel Corporation (Order No. 290147-001), 1987.

82592 Advanced CSMA/CD LAN Controller With 16-Bit Data Path, Advance Information. Intel Corporation (Order No. 290146-001), 1987.

82560 Host Interface and Memory Manager, Advance Information. Intel Corporation (Order No. 290159-001), 1987.

NOTE

Because the 82590 and the 82592 differ only in the data bus width, most of explanations given for the 82590 also apply to the 82592. In this manual, the 82590 refers to the 82590 and the 82592 unless otherwise stated.

TABLE OF CONTENTS

CHAPTER 1 **Page**

INTRODUCTION

1.1 82590 Product Family 6-13
1.2 82590 Compared with 82588 6-13
1.2.1 New Features 6-14
1.2.2 Differences Between the 82588 and the 82590 6-16
1.3 82590 Internal Architecture 6-16
1.3.1 Parallel Subsystem 6-16
1.3.2 Serial Subsystem 6-18
1.3.3 FIFO Subsystem 6-19
1.4 Programming the 82590 6-19
1.4.1 82590 and Host Interaction 6-19
1.4.2 Frame Transmission 6-20
1.4.3 Frame Reception 6-21

CHAPTER 2

FRAMING AND LINK MANAGEMENT

2.1 Frame Format 6-25
2.1.1 Frame Boundary Delineation 6-26
2.1.2 Addressing 6-27
2.1.2.1 Individual Address 6-27
2.1.2.2 Broadcast Address 6-27
2.1.2.3 Multicast Address 6-28
2.1.2.4 Recognizing Multiple Individual Addresses 6-28
2.1.3 Length Field 6-28
2.1.4 LLC Data and Pad Fields 6-28
2.1.5 Frame Check Sequence 6-31
2.2 Frame Transmission 6-32
2.2.1 Link Management 6-32
2.2.1.1 CSMA/CD 6-32
2.2.1.2 Deterministic Collision Resolution (DCR) 6-34
2.2.1.3 DCR Example 1 6-34
2.2.1.4 DCR Example 2 6-35
2.2.1.5 Full Duplex 6-36
2.2.2 Priority Mechanism 6-36
2.3 Frame Reception 6-36
2.3.1 Single Buffer Mode 6-37
2.3.2 Buffer Switching Mode 6-37
2.3.3 Frame Switching Mode 6-37
2.4 Physical Link Interface 6-38
2.4.1 Clock Generation/Encoding/Decoding 6-38
2.4.1.1 Clock Generation 6-38
2.4.1.2 Encoding/Decoding 6-39

Page

2.4.2 Carrier Sense/Collision Detect 6-39
2.4.2.1 Carrier Sense 6-39
2.4.2.2 Collision Detection 6-40
2.4.2.3 Collision Detection by Code Violation (CDBCV) 6-41
2.4.2.4 Collision Detection by Bit Comparison (CDBBC) 6-42
2.4.2.5 Collision Detection by Source Address Comparison (CDBSAC) 6-44
2.4.2.6 IEEE 802.3 Type 1BASE5 (StarLAN) CPS Recognition 6-44
2.4.3 Jitter Tolerance 6-44

CHAPTER 3
NETWORK MANAGEMENT AND DIAGNOSTIC FUNCTIONS
3.1 Transmission/Reception Error Reporting 6-50
3.2 Network Planning and Management 6-51
3.3 Station Diagnostics 6-52
3.4 82590 Self Testing 6-52

CHAPTER 4
SYSTEM INTERFACE
4.1 Command/Status Transfers 6-55
4.2 Data Transfers 6-55
4.3 Interrupt 6-57

CHAPTER 5
PROGRAMMING MODEL 6-59

CHAPTER 6
INITIALIZATION/CONFIGURATION
6.1 Initialization 6-66
6.2 Configuration 6-67
6.3 Configuration Parameters 6-68

CHAPTER 7
CONTROLLING THE 82590 THROUGH PORT 0
7.1 Command Register 6-89
7.1.1 Operations (Bits 0–3) 6-90
7.1.2 Channel Allocation 6-91
7.1.3 Setting the Pointer 6-91
7.1.4 Acknowledging Interrupt (Bit 7) 6-91
7.2 Status Registers 6-91
7.2.1 STATUS 0 6-93
7.2.1.1 Event (Bits 0–3) 6-93
7.2.1.2 Channel (Bit 4) 6-94
7.2.1.3 Execution (Bit 5) 6-94

Page

7.2.1.4 Receive (Bit 6) 6-94
7.2.1.5 Interrupt (Bit 7) 6-94
7.2.2 STATUS 1 (1–0) Through STATUS 2 (2–1) 6-94
7.2.3 STATUS 3 6-94
7.2.3.1 Execution State (Bits 0–1) 6-95
7.2.3.2 Execution Channel (Bit 2) 6-95
7.2.3.3 Buffer Chaining—Number of Buffers (Bits 3–4) 6-95
7.2.3.4 Receive State (Bits 5–6) 6-95
7.2.3.5 Receive Channel (Bit 7) 6-95
7.3 Performing Execution Operations 6-96
7.4 Frame Reception 6-96
7.4.1 Consecutive Mode (Single Buffer Mode) 6-97
7.4.2 Chaining Mode (Multiple Buffer Mode) 6-97
7.4.3 Frame Switching Mode 6-98
7.5 Tightly Coupled Interface 6-98
7.5.1 DMA Interface 6-98
7.5.2 CPU Interface 6-99
7.5.2.1 Interrupt Due to Command Execution 6-99
7.5.2.2 Interrupt Due to Frame Reception 6-99

CHAPTER 8
CONTROLLING THE 82590 THROUGH PORT 1

8.1 Command Register 6-102
8.1.1 TC/$\overline{GP}$ Select (Bit 4) 6-102
8.1.2 Operations (Bits 0–3) 6-102
8.1.3 Setting the Pointer 6-103
8.1.4 Acknowledging Interrupt (Bit 7) 6-104
8.2 Status and Timer/Counter Registers 6-104
8.2.1 STATUS 0 6-105
8.2.1.1 Interrupt (Bit 7) 6-105
8.2.1.2 GP (Bit 5) 6-105
8.2.1.3 TC (Bit 4) 6-105
8.2.1.4 Event (Bits 3–0) 6-105
8.2.2 STATUS 1 6-106
8.2.3 STATUS 2 and STATUS 3 Bits 0–3 6-107
8.2.4 STATUS 3 Bits 4–0 6-107
8.2.4.1 TC State (Bits 7–6) 6-107
8.2.4.2 GP State (Bits 5–4) 6-107
8.2.5 Timer/Counter Count Registers 6-107
8.2.6 Timer/Counter Maximum Count Registers 6-108

CHAPTER 9 **Page**

PORT 0 OPERATIONS AND STATUS

9.1 Operations 6-110
9.1.1 NOP (0,00) 6-110
9.1.2 SWT-TO-PORT-1 (1,00) 6-110
9.1.3 IA-SETUP (01) 6-110
9.1.4 CONFIGURE (02) 6-110
9.1.5 MC-SETUP (03) 6-115
9.1.6 TRANSMIT (04) 6-117
9.1.7 TDR (05) 6-118
9.1.8 DUMP (06) 6-118
9.1.9 DIAGNOSE (07) 6-121
9.1.10 RETRANSMIT (12) 6-121
9.1.11 ABORT (13) 6-121
9.1.12 RCV-ENABLE (8) 6-121
9.1.13 ASSIGN ALTERNATE BUFFER (9) 6-121
9.1.14 RCV-DISABLE (10) 6-122
9.1.15 STOP-RCV (11) 6-122
9.1.16 RESET (14) 6-122
9.1.17 FIX PTR (1,15) 6-122
9.1.18 RLS PRT (0,15) 6-122
9.2 Illegal Commands 6-123
9.2.1 Illegal Executions 6-123
9.2.1.1 Nonparametric Executions (TDR, Diagnose, Abort) 6-123
9.2.1.2 Parametric Executions 6-123
9.2.2 Illegal RCV Commands 6-123
9.2.2.1 Receive Enable 6-123
9.2.2.2 Assign Alternate Buffer 6-123
9.2.2.3 Receive Disable and Stop Receive 6-124
9.3 Event Status 6-124
9.3.1 IA-SETUP-DONE (01) 6-124
9.3.2 CONFIGURE-DONE (02) 6-124
9.3.3 MC-SETUP-DONE (03) 6-124
9.3.4 TRANSMIT-DONE (04) 6-125
9.3.5 TDR-DONE (05) 6-126
9.3.6 DUMP-DONE (06) 6-126
9.3.7 DIAGNOSE-PASSED (07) 6-134
9.3.8 End of Frame 6-134
9.3.9 Request Alternate Buffer (9) 6-139
9.3.10 Reception Aborted (10) 6-139
9.3.11 RETRANSMIT-DONE (11) 6-139
9.3.12 Execution-Aborted (13) 6-140
9.3.13 Diagnose-Failed (15) 6-140

CHAPTER 10 **Page**
PORT 1 OPERATIONS
10.1 General Purpose Operations 6-142
10.1.1 NOP (00) 6-142
10.1.2 SWT-TO-PORT-0 (01) 6-142
10.1.3 SET-TS (05) 6-142
10.1.4 RST-TS (07) 6-142
10.1.5 LCL-PWR-DWN (08) 6-142
10.1.6 RMT-PWR-DWN (09) 6-143
10.1.7 FIX-PTR (12) 6-143
10.1.8 RLS-PTR (13) 6-144
10.1.9 RESET (14) 6-144
10.1.10 SEL-RST (15) 6-144
10.2 Timer/Counter Operations 6-144
10.2.1 NOP (00) 6-144
10.2.2 START (01) 6-144
10.2.3 STOP (02) 6-144
10.2.4 RESUME (03) 6-144
10.2.5 LD&START (05) 6-145
10.2.6 ACK-INT (07) 6-145
10.2.7 COUNT (08) 6-145
10.2.8 START-ALL-COUNTERS (09) 6-145
10.2.9 SET-VAL (10) 6-145
10.2.10 SET-CONF (11) 6-145
10.2.10.1 WRP(TM-C3)—Parameter 1, Bits 7–4 6-146
10.2.10.2 IE(TM-C3)—Parameter 1, Bits 3–0 6-146
10.2.10.3 EXTTM—Parameter 2, Bit 0 6-147
10.2.10.4 EXTCN—Parameter 2, Bit 1 6-147
10.2.10.5 CLRTI—Parameter 2, Bit 2 6-147
10.2.10.6 TMCLK—Parameter 2, Bit 3 6-147
10.2.11 RD-MAX-COUNT-VAL (12) 6-148
10.2.12 RD-COUNT-VAL (13) 6-148
10.2.13 RESET (14) 6-148
10.2.14 Timer/Event Counter Diagnostics 6-148
10.3 Illegal Commands 6-148

CHAPTER 11
PERFORMANCE CONSIDERATIONS
11.1 Transfer Rate Limitations 6-152
11.2 Bus Utilization 6-153
11.3 Buffer Size 6-154
11.4 Interframe Spacing 6-154

CHAPTER 12
APPLICATIONS 6-157

Figures

Figure	Title	Page
1-1	82590/592 Block Diagram	6-17
1-2	82590 CPU Interaction	6-20
1-3	Frame Structure and Location of Data Element in Memory	6-21
1-4	Single Buffer Reception	6-22
1-5	Multiple Buffer Reception	6-23
2-1	Frame Format for End of Carrier Framing	6-25
2-2	Frame Format for HDLC (Bitstuffing) Framing	6-26
2-3	Hash Bit Calculation Program	6-29
2-4	DCR for a Two-Station Collision	6-35
2-5	Station C Transmits a Frame in its Slot	6-35
2-6	Data Encoding Schemes	6-39
2-7	8X Sampling/Decoding	6-42
2-8	Collision Detection by Bit Comparison (CDBBC)	6-43
2-9	Frame Length Requirement for CDBBC	6-44
2-10	StarLAN Collision Presence Signal (CPS) Waveforms	6-45
2-11	Jitter	6-45
2-12	Block Diagram of 82590 DPLL	6-46
2-13	Jitter Tolerance—NRZI (Integrator = 7)	6-48
2-14	Jitter Tolerance—NRZI (Integrator = 4)	6-48
4-1	82590 Interface Signals	6-54
4-2	Demultiplexing $\overline{DACK}/\overline{CS1}/\overline{EOP}$ Pin	6-55
4-3	Command/Status Transfers	6-56
4-4	Data Transfer	6-57
5-1	Programming Model—Directly Accessible Registers (Accessible Through 8-Bit I/O Port(s))	6-60
5-2	Port 0 Commands	6-61
5-3	Port 1 Commands	6-62
5-4	Parameter Format for the 82590 or the 82592 in 8-Bit Mode	6-63
5-5	Parameter Format for the 82592 in 16-Bit Mode (n = Even Number)	6-63
5-6	Parameter Format for the 82592 in 16-Bit Mode (n = Odd Number)	6-64
5-7	Parameter Format for the 82592 Configured to Byte Swap in 16-Bit Mode	6-64
6-1	82592 Bus Width Initialization Flowchart	6-66
6-2	Memory Structure for the 82592 when Configured for 16-Bit Mode	6-67
6-3	Configure Block	6-69
7-1	Port 0 Command Register	6-90
7-2	Port 0 Status Registers—4 Bytes	6-92
7-3	Port 0 Status Registers—6 Bytes	6-92
7-4	STATUS 2-0 Format in TCI Mode	6-93
8-1	Port 1 Command Register	6-102
8-2	Port 1 Status and Timer/Counter Register Bank	6-105

Figure	Title	Page
8-3	Port 1 Status Registers	6-106
9-1	IA Parameter Format for the 82590 or the 82592 in 8-Bit Mode	6-111
9-2	IA Parameter Format for the 82592 in 16-Bit Mode	6-111
9-3	IA Parameter Format for the 82592 Configured to Byte Swap in 16-Bit Mode	6-111
9-4	Configure Parameter Format for the 82590 or the 82592 in 8-Bit Mode	6-112
9-5	Configure Parameter Format for the 82592 in 16-Bit Mode	6-113
9-6	MC-SETUP Parameter Format for the 82590 or the 82592 in 8-Bit Mode	6-116
9-7	MC-SETUP Parameter Format for the 82592 in 16-Bit Mode	6-116
9-8	Transmit Parameter Format for the 82590 or the 82592 in 8-Bit Mode	6-117
9-9	Transmit Chaining Data Structure for the 82590 or the 82592 in 8-Bit Mode	6-119
9-10	Transmit Chaining Data Structure for the 82592 in 16-Bit Mode	6-120
9-11	Configure Result Format	6-124
9-12	Transmit Result	6-125
9-13	TDR Result	6-126
9-14	Dump Result for the 82590 or the 82592 in 8-Bit Mode	6-127
9-15	Dump Result for the 82592 in 16-Bit Mode	6-130
9-16	Receive Format for the 82590 or the 82592 in 8-Bit Mode (Tightly Coupled Interface Not Enabled)	6-135
9-17	Receive Format for the 82590 or the 82592 in 8-Bit Mode (Tightly Coupled Interface Enabled)	6-136
9-18	Receive Format for the 82592 in 16-Bit Mode (Tightly Coupled Interface Enabled, Byte Swap Not Enabled)	6-137
9-19	Receive Status (4-Byte Status Configuration)	6-138
9-20	Receive Status (6-Byte Status Configuration)	6-138
9-21	Receive Status in Tightly Coupled Interface Mode	6-139
9-22	Retransmit Result	6-140
10-1	Timer/Counter Configuration Parameters	6-146
10-2	Timer Diagnostics	6-149
10-3	Event Counter Diagnostics	6-150
12-1	82590/82560 StarLAN Adapter Board	6-158
12-2	82592/82560 Ethernet/Cheapernet Adapter Board	6-159

Tables

Table	Title	Page
2-1	Data Encoding Rules	6-40
2-2	Jitter Tolerance—Manchester	6-46
2-3	Jitter Tolerance—NRZI	6-47
6-1	Default Configuration	6-67
6-2	Receive Mode and EOP Signaling	6-71
6-3	Padding	6-74
6-4	Manchester/NRZ(I) Configuration Bit	6-76
6-5	CDBCV Configuration Bit	6-77
6-6	Sampling Rate	6-79
6-7	FIFO Banking	6-85
6-8	Port 1 Enable	6-85
7-1	Transmit/Receive Status Encoding on DRQ and $\overline{EOP}$	6-99

Introduction 1

CHAPTER 1
INTRODUCTION

1.1 82590 PRODUCT FAMILY

The 82590 product family consists of a new generation of CMOS, CSMA/CD LAN controllers (82590 and 82592) and a host interface chip (82560). Like the 82588, the 82590 is a complete CSMA/CD Medium Access Control (MAC) controller, supporting the IEEE 802.3 specification. The 82590 supports a wide variety of industry standard CSMA/CD LANs, including IEEE 802.3 TYPE 1BASE5 (StarLAN), TYPE 10BASE5 (Ethernet), TYPE 10BASE2 (Cheapernet or Thin Ethernet), Type 10BROAD36 (Broadband Ethernet), and IBM PC Network—Baseband and Broadband. When the 82590 is used for Ethernet or Cheapernet applications, it is interfaced to the 82C501 Ethernet serial interface chip and the 82502 Ethernet transceiver chip. Programmable parameters of the 82590 also enable it to operate in a broad range of proprietary network applications. The 82590 controller is pin and software compatible with the 82588, offering a smooth migration path and facilitating implementation of its enhanced functionality.

The 82560 is a companion chip for the 82590, the 82592, and the 82588. It interfaces these controllers to the IBM PC/XT/AT I/O Channels and provides on-chip DMA support for data transmission and reception. The 82560 integrates all the functions required to implement a nonintelligent, locally buffered LAN adapter for the PC/XT/AT environment.

The 82590, in conjunction with the 82560, provides a high integration, high performance interface for all the popular, industry standard office networks currently being implemented. The advanced features of the 82590 also make it an ideal solution for bridges, routers, and other specialized servers based on the PC architecture.

1.2 82590 COMPARED WITH 82588

Compared to the 82588 the 82590 has a number of functional and performance enhancements. This section is provided for those users who are already familiar with the 82588.

In order to maintain 82588 compatibility, most of the 82590's enhanced features are enabled through software. The 82590 has one additional command/status port and an expanded configure command block to facilitate smooth migration from the 82588. Existing 82588 software, which would use the 82590 within the bounds of the 82588 functionality, can be easily upgraded to 82590 software, thereby taking full advantage of the 82590 functionality.

While the 82588 has only one command/status port selected by an active $\overline{CS}$ (Chip Select) signal, the 82590 has two command/status ports, Port 0 and Port 1. Port 0 is the 82588 compatible port and Port 1 is the new command/status port which allows an extended set of 82590 commands and status to be accessed. The 82590 provides an option of having one or two chip select inputs. If only one hardware chip select line is available, software port switch commands can be used to access both Port 0 and Port 1.

1.2.1 New Features

This section lists the functional and performance enhancements of the 82590.

Overall device improvements are listed below:

- **CMOS**—The 82590 is fabricated in CMOS. When it is used in an 82588 socket its power dissipation is approximately one tenth of the 82588's.
- **Deeper FIFOs**—The 82590 has deeper FIFOs. A total of 64 bytes can be divided into 16 and 48 bytes, 32 and 32 bytes, 48 and 16 bytes, or 16 and 16 bytes, for the transmit and receive FIFOs respectively.
- **On-Chip Event Counters and a General Purpose Timer**—There are three 16-bit event counters and a 24-bit general purpose timer. These three event counters can be used to collect network statistics.
- **Full Duplex Mode**—The 82590 can operate in Full Duplex mode. The CSMA/CD protocol is disabled in this mode.
- **Power Down Modes**—Two power down modes, Local Power Down and Remote Power Down, are available. When the 82590 is in Remote Power Down mode, it can be powered up remotely by sending a special frame to it.
- **On-Chip System Clock Generation**—The clock signal used in the serial subsystem can be internally routed to the parallel subsystem. If this option is selected there is no need to provide the system clock to the 82590.
- **CRC Transfer to Memory**—The 82590 can be programmed to transfer the CRC field of a received frame into memory.
- **Receive Status in Registers**—The 82590 can be configured to store the status of a received frame in the status registers in addition to a byte count of the received frame.
- **Monitor Mode**—The monitor mode allows the 82590 to evaluate incoming frames without transferring their contents to memory.
- **Improved TDR Mechanism**—The 82590 TDR frame consists of preamble, source address, a series of 01101110 bit pattern, and a jam pattern. If a collision is detected during a TDR operation a jam pattern is transmitted.
- **Late Collision Detection**—The 82590 can report if a collision has occurred within or outside of the collision window.

Improvements made in the parallel subsystem are listed below:

- **16-Bit Mode**—The 82592 can be configured to have a 16-bit data bus interface for DMA transfers.
- **Tightly Coupled Interface**—The 82590 provides the Tightly Coupled Interface to, with a DMA Controller, implement transmit chaining, automatic retransmission, buffer reclamation, and receive buffer management.

- **Device and Stepping IDs**—The 82590's device and stepping identification codes can be found in registers.
- **Byte Swap**—When using the 82592 in 16-bit mode, the high and low order bytes can be swapped.

Improvements made in the serial subsystem are listed below:

- **Higher Bit Rate**—The 82590 can support a serial bit rate up to 20 Mb/s in High Speed mode and 4 Mb/s in High Integration mode.
- **Multi-IA**—The 82590 implements multiple-individual-address (Multi-IA) filtering.
- **Automatic Padding and IEEE 802.3 Length Field**—If a frame to be transmitted is shorter than the configured Slot Time, the 82590 automatically appends pad bytes up to the shortest frame greater than the Slot Time. If the data field of a received frame is longer than the byte count indicated in the Length field, the extra bytes are removed automatically according to the value in the Length field. Erroneous conditions are detected and reported by the 82590. An example of such a condition is reception of a frame which is shorter than the byte count indicated in the Length field.
- **Improved Jitter Tolerance**—The jitter tolerance of the 82590 is greater than that of the 82588.
- **Deterministic Collision Resolution**—The 82590 implements a Deterministic Collision Resolution method.
- **CDT Pin**—The collision detect (CDT) input pin is available for external collision detection even when the Collision Detection by Code Violation is enabled in High Integration mode.
- **Automatic Retransmission on Collision During Preamble**—The 82590 can be programmed to retransmit automatically if it detects a collision during transmission of the preamble.
- **Collsion Detection By Source Address Comparison**—The source address of a transmitted frame is compared to the source address of a received (returned) frame to detect a collision.
- **On-Chip Jabber Inhibit Function**—The 82590 can be programmed to provide an on-chip jabber inhibit function.
- **Loopback Signal to the 82C501**—The 82590 can be programmed to provide an active High loopback signal to the 82C501.
- **StarLAN**—The 82590 can be configured to recognize the IEEE 802.3 1BASE5 Collision Presence Signal (CPS). In this mode it also delays deactivation of the $\overline{RTS}$ signal at the end of a frame transmission in order to insert an end-of-frame marker for compliance with the standard.
- **Preamble Until Carrier**—The 82590 can be programmed to generate Preamble until an active Carrier is returned to the device.
- **Jam Pattern**—The 82590 has an improved jam pattern.

1.2.2 Differences Between the 82588 and the 82590

When none of the enhanced features of the 82590 are enabled the 82590 can operate as a CMOS version of the 82588, running the 82588 software in a socket originally designed for the 82588. However, some minor differences exist between the 82590 and the 82588 even when the 82590 is being used within the bounds of the 82588 functionality. Before the 82590 is placed in a 82588 socket the following items must be checked.

- **DUMP Area**—The 82590 dumps 69 bytes of internal information whereas the 82588 dumps only 63 bytes. In order to use the 82590 Dump command, at least 69 bytes must be allocated for the dump area.
- **Loopback Configuration**—If both bits 6 and 7 of the configuration byte 4 are set, the 82590 is configured to External Loopback with the loopback output signal active (see Section 6.3, Parameter 29). The same configuration bits place the 82588 in Internal Loopback mode.
- **Channel Bit for NOP**—If a NOP command is issued to Port 0 of the 82590 with the Channel bit set to 1, the active command/status port is switched to Port 1. If the command is issued with the Channel bit set to 0, the command is treated as a NOP. In case of the 82588, a NOP command is always treated as a NOP regardless of the Channel bit.
- **Preamble in NRZI and Differential Manchester**—The preamble pattern generated by the 82590 is a sequence of all zeros for NRZI encoding and all ones for Differential Manchester encoding. The 82588 uses alternating ones and zeros in both cases.
- **Padding Length for Bitstuffing**—The 82590 in Bitstuffing mode can perform padding up to the Slot Time measured from the SFD field. The 82588 in the same mode measures the padding length from the beginning of Preamble.
- **CDT Pin**—Even when the 82590 is configured for Collision Detection By Code Violation, the CDT (Collision Detect) pin is capable of detecting an external collision signal. If only the CDBCV is to be used, the CDT pin should be connected to Vcc. With the 82588, whenever the chip is configured for CDBCV, the CDT pin is not sensitive to the external signal connected to it.

1.3 82590 INTERNAL ARCHITECTURE

The 82590 consists of a parallel subsystem, a serial subsystem, and a FIFO subsystem (see Figure 1-1).

1.3.1 Parallel Subsystem

The parallel subsystem consists of a bus interface unit (BIU), command and status registers, a 24-bit general purpose timer, and three 16-bit event counters.

The BIU provides an 8- and/or 16-bit data bus interface to the external system bus. It handles all data transfers to and from memory (at speeds up to 16 Mbytes/sec.), accepts commands from the CPU, and provides status to the CPU. There are two separate I/O ports, Port 0 and Port 1; and two separate DMA channels, Channel 0 and Channel 1. The I/O ports are 8 bits wide and are interfaced to the CPU via the data lines D0-D7. The DMA

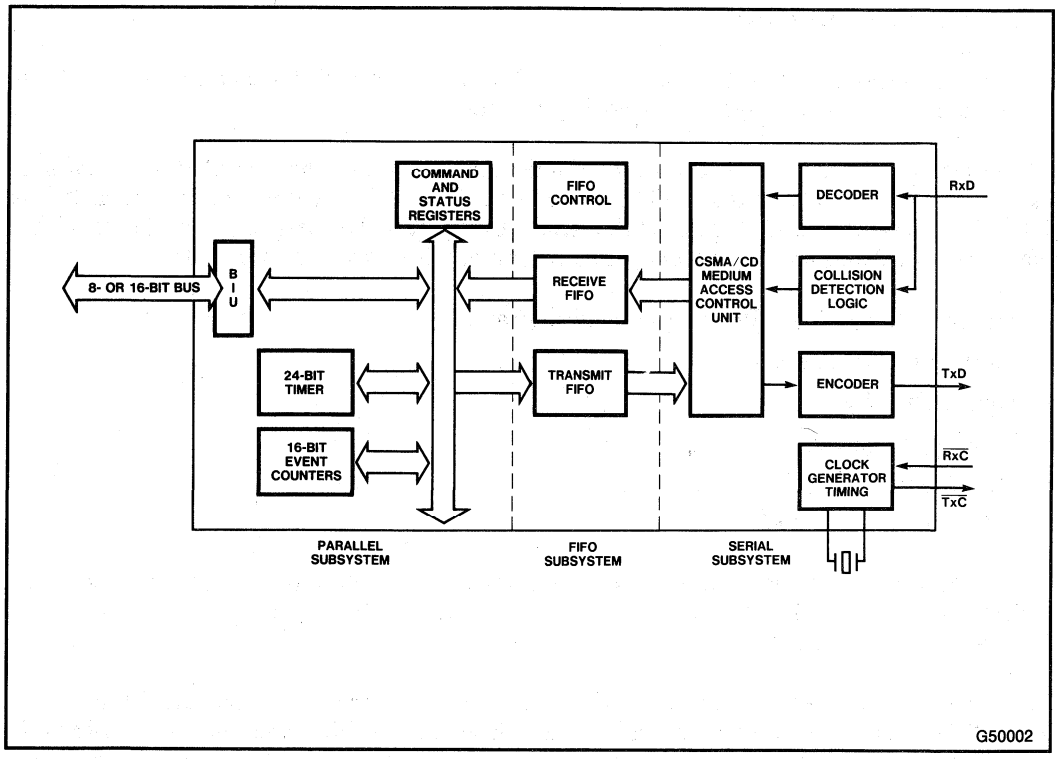


Figure 1-1. 82590/592 Block Diagram

channels are 8 bits wide for the 82590, and 8 or 16 bits wide for the 82592. When the 82592 is reset by hardware or software, the DMA channels are initialized for an 8-bit data path. The CPU can then configure the 82592 for a 16-bit data path if desired. Once the DMA channels are configured for a 16-bit data path all subsequent DMA transfers are performed on the data lines D0-D15. The two DMA channels are independent of each other and can be used for high-performance operations such as simultaneous transmission and reception.

The 24-bit timer consists of a 24-bit maximum count register, a 24-bit count register, and associated control bits in the command registers. Its clock source can be the transmit clock or the parallel subsystem clock. The timer can be programmed to halt or continue on a terminal count with or without causing an interrupt.

The three 16-bit event counters can be programmed to count valid frames, collided frames, and errored (CRC or Alignment) frames. When these event counters are used in Monitor mode, the 82590 is capable of maintaining the network statistics by itself; i.e., without requesting DMA services or causing interrupts to the CPU.

1.3.2 Serial Subsystem

The serial subsystem consists of a CSMA/CD unit, a data encoder and decoder, collision detect and carrier sense logic, and a clock generator.

The 82590's CSMA/CD unit is highly flexible in implementing the CSMA/CD protocol. It can operate in a variety of IEEE 802.3 and other CSMA/CD LAN environments, including 1BASE5 (StarLAN), 10BASE5 (Ethernet), 10BASE2 (Cheapernet), and the IBM PC Network (Baseband and Broadband). The programmable parameters include:

- Framing (IEEE 802.3 Framing or HDLC Framing)
- Address Field Length
- Station Priority
- Interframe Spacing
- Slot Time
- CRC-132 or CRC-116

The CSMA/CD unit also has a mode which implements deterministic collision resolution (DCR). The DCR algorithm is fully compatible with the MULTIBUS® II Serial System Bus (SSB) specifications.

The encoder and decoder in the serial subsystem is capable of NRZI, Manchester, and Differential Manchester encoding and decoding at bit rates up to 4 Mb/s in High-Integration Mode, and Manchester encoding at bit rates up to 20 Mb/s in High-Speed Mode. A digital phase-lock loop is used in High-Integration Mode to decode the receive data and to generate the synchronous receive clock.

The collision detect and carrier sense logic generate the internal collision detect and carrier sense signals for the CSMA/CD unit.

The 82590 implements several different internal, logic-based collision detect mechanisms. Two of these, Code Violation and Bit Comparison, are also implemented in the predecessor of the 82590, the 82588, and have been used in a variety of applications. The Code Violation method defines a collision where a transition edge occurs outside the area of normal transitions (as specified by the data encoding method). For example, if there are no mid-bit cell transitions in the Manchester encoded data, this method interprets that condition as a collision. The Bit Comparison method compares the signature of the transmitted frame to the signature of the received frame. If the signatures are different a collision is assumed to have occurred. Two other internal collision detect methods implemented in the 82590 are Source Address Comparison and StarLAN CPS (Collision Presence Signal) Recognition. The Source Address Comparison compares the source address field of the transmitted frame to the source address field of the received frame. If the source addresses are different, it assumes that a collision has occurred resulting in data corruption in the source address field. The StarLAN CPS recognition method looks for the specific collision presence signal defined by the IEEE 802.3 1BASE5 standard. Other abnormal circumstances, such as no carrier for more than one-half slot time in the receive channel during transmission, are interpreted as collisions by the 82590.

In addition to these internal, logic-based collision detection methods, an external means of collision detection can be used in parallel by using the $\overline{CDT}$ input pin.

The clock generator in the serial subsystem is available only in High-Integration Mode and provides timing for the serial subsystem. The clock signal can also be routed to the parallel subsystem, if desired. The oscillator circuit is designed for use with an external, parallel resonant, fundamental-mode crystal. The crystal frequency should be selected at 8X, 10X, 16X, or 18X the required serial bit rate.

1.3.3 FIFO Subsystem

The FIFO subsystem is located between the parallel subsystem and the serial subsystem. It consists of a transmit FIFO, a receive FIFO, and FIFO control logic. The transmit and receive FIFOs are independent, and individually provide optimal interfaces between the two subsystems which may have different speeds. There is a total of 64 bytes that can be used for the two separate FIFOs. During configuration these 64 bytes can be divided into one of three possible combinations: 16 and 48 bytes, 32 and 32 bytes, or 48 and 16 bytes for the transmit and receive FIFOs respectively. The FIFO threshold is programmed during configuration.

1.4 PROGRAMMING THE 82590

1.4.1 82590 and Host Interaction

The CPU interacts with the 82590 through the system's memory and the 82590's on-chip registers. The CPU creates a data structure in memory, programs the external DMA controller with the start address and byte count of the memory block, and issues a command to the 82590.

The chip select and interrupt lines are used to communicate between the 82590 and the CPU as shown in Figure 1-2. The interrupt signal is used by the 82590 to attract the CPU's attention. The chip select signal is used by the CPU to attract the 82590's attention. Note that the 82590 does not have any address lines.

There are two kinds of transfers over the bus: command/status and data transfers. The command/status transfers are always performed by the CPU. The data transfers are requested by the 82590, and are usually performed by a DMA controller. The CPU writes commands to the 82590 using the $\overline{CS0}$ (or $\overline{CS1}$) and $\overline{WR}$ signals, and reads status using the $\overline{CS0}$ (or $\overline{CS1}$) and $\overline{RD}$ signals. When data transfers are performed, $\overline{DACK0}$ or $\overline{DACK1}$ must be asserted by the DMA controller instead of the Chip Select. The 82590 has two I/O ports, Port 0 and Port 1, for command/status transfers, and two DMA channels, Channel 0 and Channel 1, for data transfers.

To initiate an operation such as Transmit or Configure, the command from the CPU must first be written to the 82590. Any parameters or data associated with the command are transferred from memory to the 82590 using DMA. Upon completion of the operation the 82590 updates the appropriate status registers and sends an interrupt to the CPU.

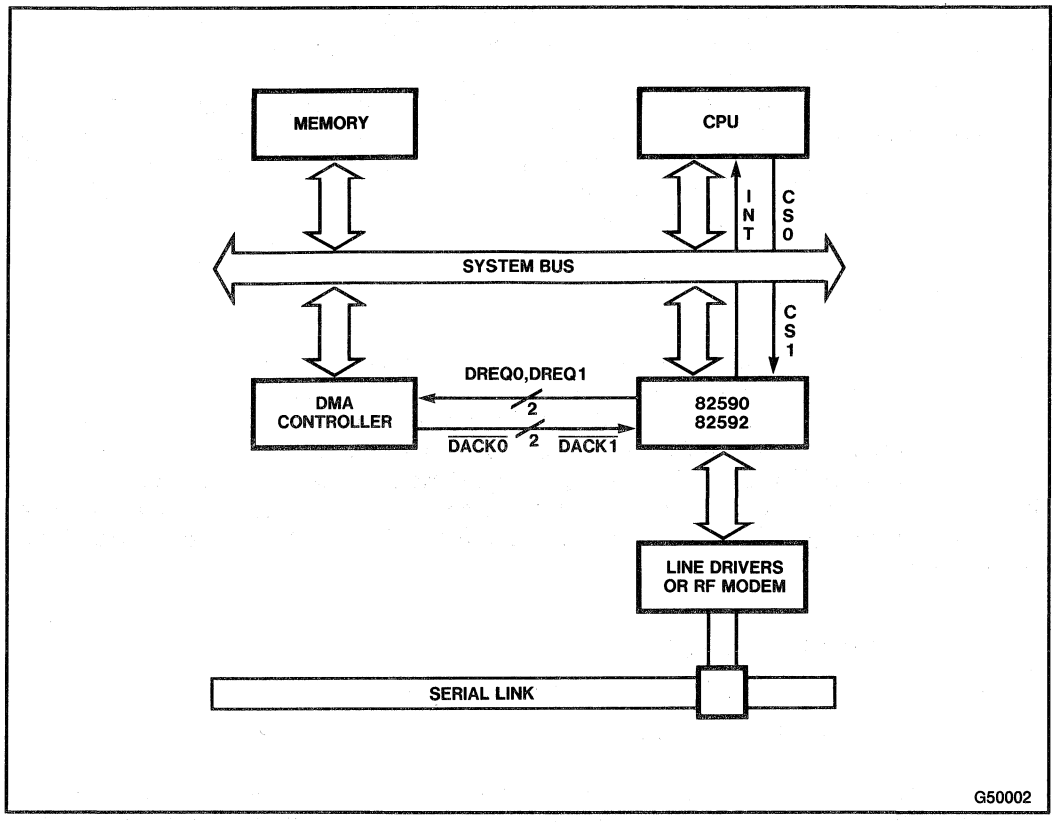


Figure 1-2. 82590 CPU Interaction

1.4.2 Frame Transmission

To transmit a frame, the CPU prepares a Transmit Data Block in memory as shown in Figure 1-3. Its first two bytes specify the length of the rest of the block. The next few bytes (up to six) contain the destination address of the station to which the frame is being sent. The rest of the block is the data field. The CPU programs the DMA controller with the start address of the block, length of the block, and other control information and then issues a Transmit command to the 82590. Upon receiving this command the 82590 fetches the first two bytes of the block to determine its length. If the link is free and the first data byte was fetched, the 82590 begins transmitting the preamble and concurrently fetches more bytes from the Transmit Data Block and loads them into the transmit FIFO to keep them ready for transmission.

The destination address is transmitted after the preamble. This is followed by the source or the station individual address, which was previously stored in the 82590 by the IA-Setup command. After this, the entire information field is transmitted, followed by a CRC field calculated by the 82590. If a collision is encountered during transmission of the frame, then the transmission is aborted after a jam pattern is sent. If the collision is detected during preamble, or SFD (Start Frame Delimiter) transmission, the 82590 transmits the jam pattern

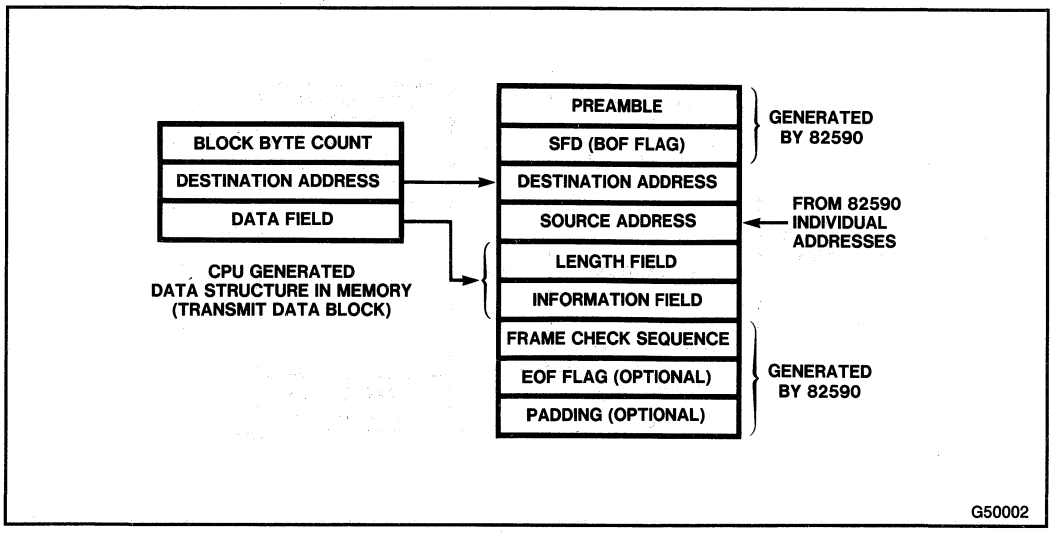


Figure 1-3. Frame Structure and Location of Data Element in Memory

after the SFD is transmitted. An interrupt is then generated to inform the CPU of the unsuccessful transmission due to a collision. The CPU reinitializes the DMA controller and issues a Retransmit command to the 82590. Retransmission is done by the CPU exactly as the Transmit command is done, except the Retransmit command keeps track of the number of collisions encountered. When the 82590 gets the Retransmit command and the backoff timer is expired, it transmits the frame again. Retransmission is repeated until the attempt is successful, or until the preprogrammed maximum number of retries is reached.

If the 82590 is programmed to generate the $\overline{EOP}$ signal to the 8237 or 82380 DMA controller, or if it is used with a DMA controller which implements the Tightly Coupled Interface, retransmission is performed without CPU intervention.

1.4.3 Frame Reception

The 82590 can receive frames when its receiver has been enabled. The 82590 checks for an address match for an Individual address, a Multicast address, or a Broadcast address. In the Promiscuous mode the 82590 receives all frames. When the address match is successful, the 82590 transfers the frame to memory using the DMA controller. Before enabling the receiver, it is the CPU's responsibility to make a memory buffer area available to the receiver and to properly program the starting address of the DMA controller. The received frame is transferred to the memory buffer in the format shown in Figure 1-4. This method of reception is called Single Buffer reception; the entire frame is contained in one continuous buffer. Upon completion of reception, the status of the reception is appended to the end of the received frame in the memory buffer, and the total number of bytes transferred to the memory buffer is loaded into the internal status registers 1 and 2. An interrupt is then generated to inform the CPU of the frame reception.

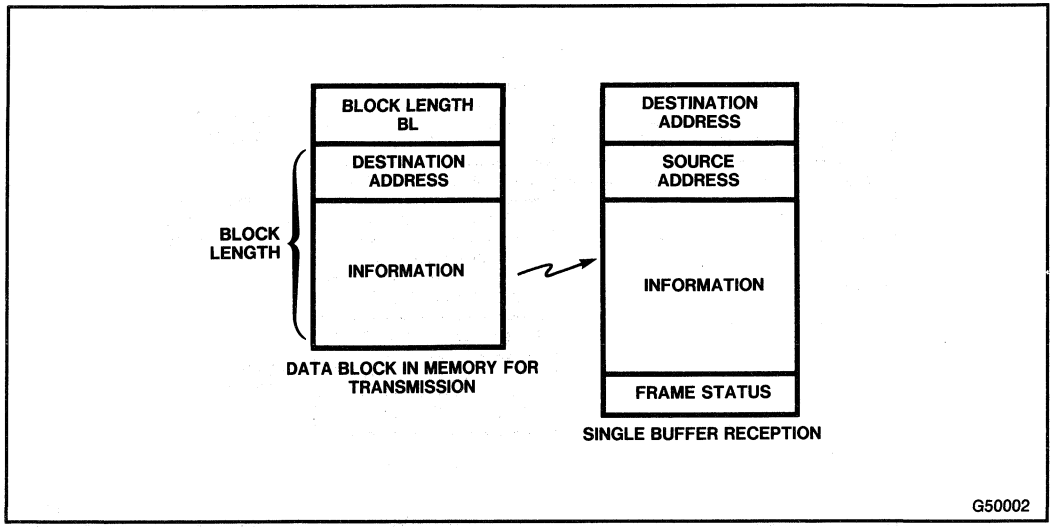


Figure 1-4. Single Buffer Reception

If the frame size is unknown, memory usage can be optimized by using Multiple Buffer reception. In this mode of operation the CPU and DMA controller can dynamically allocate memory space as they receive frames. This method requires both DMA channels to receive the frame alternately. As frame reception begins, the 82590 interrupts the CPU and automatically requests assignment of the next available buffer. The CPU does this and loads the second DMA channel with the next buffers information so the 82590 can immediately switch to the other channel when the current buffer is full. When the 82590 switches from the first to the second buffer it again interrupts the CPU and requests another buffer to be allocated on the previous channel. This process continues until the entire frame is received. The received frame is spread over multiple memory buffers. The link between the buffers is easily maintained by the CPU, using a buffer chain descriptor structure in memory as shown in Figure 1-5. This dynamic allocation of memory buffers results in efficient use of available storage when handling frames of widely differing sizes.

If the 82590 is programmed to generate the EOP signal to the 8237 or 82380 DMA controller, or if it is used with a DMA controller which implements the Tightly Coupled Interface, buffer reclamation and more advanced data structures for the buffer area can significantly improve system performance.

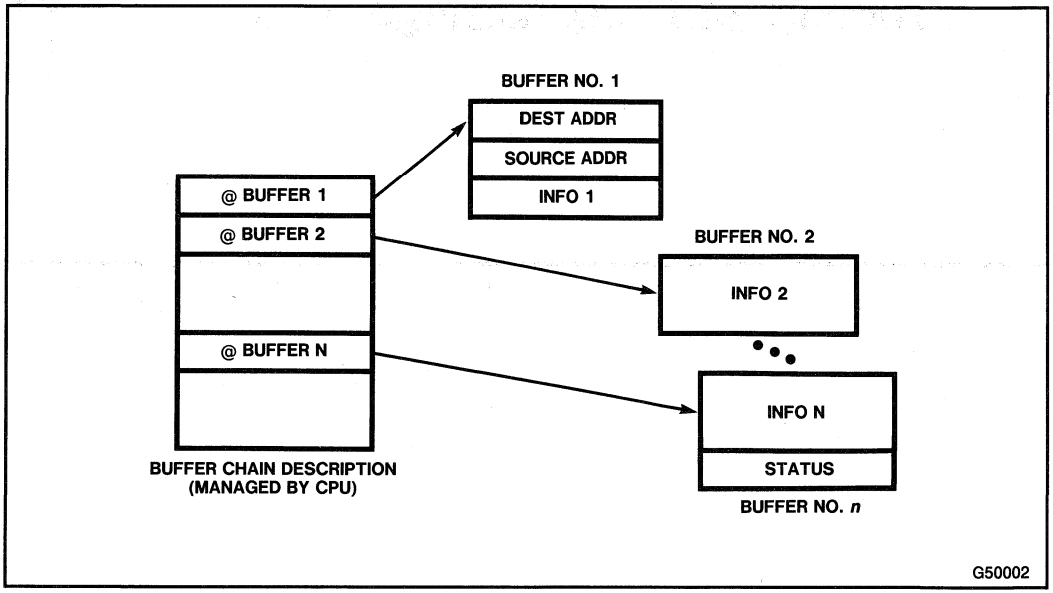


Figure 1-5. Multiple Buffer Reception

Framing and Link Management 2

CHAPTER 2
FRAMING AND LINK MANAGEMENT

The data structures handled by the 82590 on the link are called frames. A frame is a sequence of bits that travels on the link.

Framing has three primary functions: determining the beginning and end of the frame (frame boundary delineation), determining the frame's source and destination (addressing), and performing transmission error detection.

2.1 FRAME FORMAT

The Frame formats supported by the 82590 are shown in Figures 2-1 and 2-2.

The different fields of the frame are:

- Preamble—Used by the clock recovery and data decoding circuitry to reach steady state synchronization with the received frame timing. The preamble length is programmable with the 82590. The IEEE 802.3 specifications requires seven bytes of the preamble. When the 82590 is configured for NRZ or Manchester encoding it uses alternating ones and zeros as the preamble sequence. When configured for NRZI it uses repeating zeros. When configured for Differential Manchester encoding it uses repeating ones.
- Start of Frame Delimiter (SFD) Field or Beginning of Frame (BOF) flag—The 82590 can be programmed to insert either the SFD field or the BOF flag following the preamble. The SFD field is the sequence 10101011. The BOF flag is an HDLC flag (01111110) and is used only in Bitstuffing Mode.

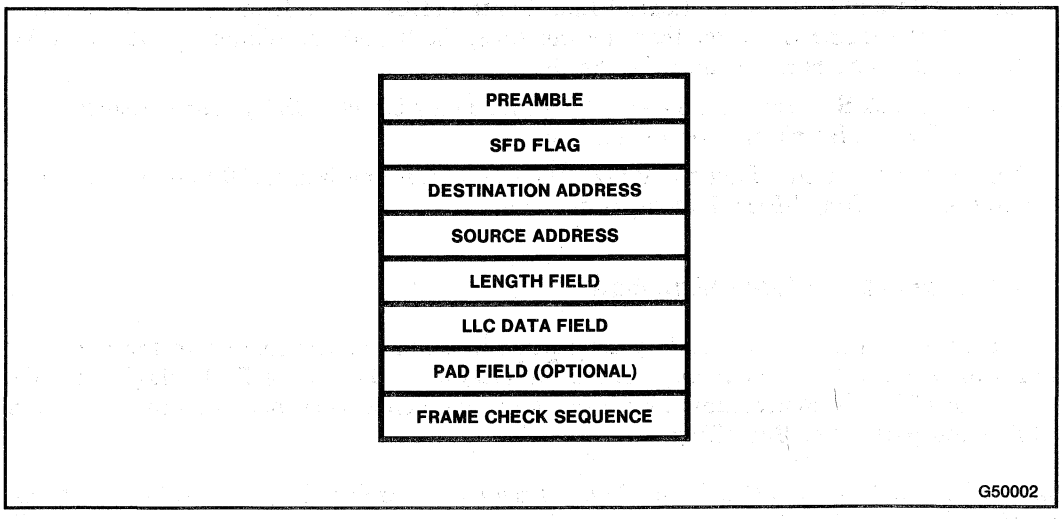


Figure 2-1. Frame Format for End of Carrier Framing

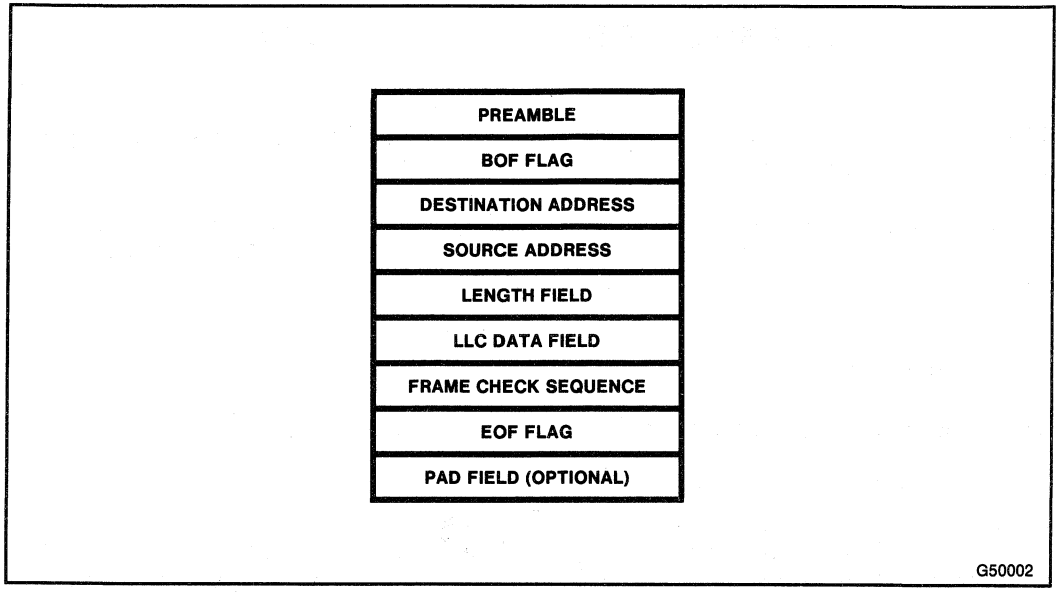


Figure 2-2. Frame Format for HDLC (Bitstuffing) Framing

- Destination Address—Specifies the station(s) for which the frame is intended (i.e., the frame target address). The address length is programmable with the 82590. The IEEE 802.3 specifications requires the address length to be two or six bytes.
- Source Address—Specifies the station sending the frame (i.e., the sender's address).
- Length field—Specifies the number of bytes contained in the LLC Data field. The length field is two bytes.
- LLC Data and PAD field—Logical Link Control (LLC) Data field carries the information. If the frame is shorter than the slot time, the 82590 can optionally add the PAD field so it becomes longer than the slot time.
- Frame Check Sequence (FCS)—Cyclic Redundancy Check (CRC) used in detecting bit errors. The FCS field is 2- or 4-bytes long.
- End of Frame (EOF) Flag and PAD field—The EOF flag is an HDLC flag and is used only in Bitstuffing Mode. Padding is optional.

2.1.1 Frame Boundary Delineation

The 82590 handles frame boundary delineation in a manner transparent to the user. The fields involved in frame boundary delineation are the SFD field (or BOF Flag) and EOF flag. The 82590 is configurable to one of two frame delineation methods: End of Carrier (802.3 compatible) or Bitstuffing.

In the End of Carrier method the 82590 transmits the SFD field (10101011) after transmitting the preamble. The end of frame is indicated by the Carrier going inactive immediately after the Frame Check Sequence field. The Carrier is a signal which informs the 82590

of activity on the link. It can be generated internally or externally. This frame boundary delineation method is compatible with the IEEE 802.3 specs.

In the Bitstuffing method the 82590 transmits an HDLC flag as the BOF flag immediately after transmitting the preamble. The end of frame is indicated by another HDLC flag. The 82590 performs HDLC zero bit insertion (insert 0 after five consecutive 1's) in the fields between the BOF and EOF flags to prevent the bit sequence 01111110 from appearing in any of these fields. The chip can be configured to pad the frame with additional flags so the frame length becomes equal to or longer than the Slot Time.

2.1.2 Addressing

Regardless of the Frame Boundary Delineation method, the two fields following the SFD field (or BOF flag) are Destination Address and Source Address.

Addressing allows frames to be directed to one or more specific stations on the link. The 82590 provides flexible addressing techniques which allow a frame to be received by a single station, a group of stations (multicast addressing), or all stations (broadcast addressing). The address length is determined during configuration by the Address Length parameter, and can be zero to six bytes long. The default address length is six bytes.

During transmission the 82590 usually inserts its station address from the internal Individual Address register into the Source Address field. The source address insertion can be overridden during configuration, if so desired. During reception, the 82590 compares the incoming Destination Address with its Individual Address, the programmed set of multicast addresses, and the broadcast address. If an address match is made (all bits must be equal) the frame is accepted. A frame whose address does not match is discarded by the 82590, unless the monitor function is enabled.

2.1.2.1 INDIVIDUAL ADDRESS

After initialization the 82590 is configured with an Individual Address using the IA-SETUP command. An Individual Address is indicated by a zero in the least significant bit position. During reception the 82590 performs a bit-wise comparison of the incoming Destination Address with its configured Individual Address. The frame is transferred to memory only when the two addresses match. The 82590 reports the address match by setting a status bit. The default for the Individual Address is all 1's.

The 82590 can be configured to recognize more than one Individual Address.

2.1.2.2 BROADCAST ADDRESS

Frames may be targetted to all stations by using the Broadcast Address. The Broadcast Address is an address with all ones. The 82590 receives frames whose Destination Address is the Broadcast Address (broadcast frames). The broadcast frames are received regardless of the 82590's Individual Address. Reception of Broadcast frames can only be disabled by

properly configuring the 82590. It may be advantageous to disable reception of Broadcast frames for stations with limited storage resources.

2.1.2.3 MULTICAST ADDRESS

Frames can be targetted to a specific group of stations. Each group shares a particular group address called the Multicast Address. The Multicast Address has one in its least significant bit position. A station may belong to several different groups. The 82590 has the MC-SETUP command to register a set of Multicast addresses in the 64-bit Hash table internal to the device. Each Multicast Address included in the MC-SETUP command is mapped into a single bit in the Hash table. During reception of a frame whose Destination Address is a Multicast Address, the 82590 maps the address in the Hash table to check if it is one of the registered Multicast Addresses. If it is, the 82590 continues the receive process.

It is possible for more than one Multicast Address to be mapped into a given Hash bit. Thus, the host CPU may have to perform additional checking to ensure perfect Multicast Address filtering. However, if 64 or fewer Multicast addresses are used, it is possible to choose addresses so that they all map into unique bits in the Hash table. The 82590 uses the CRC function, which is also used for bit error detection, as the hashing function. Bits 2 through 7 of the 16- or 32-bit CRC sequence are used as an index to the 64-bit Hash table. The Pascal program shown in Figure 2-3 implements the hash function.

The 82590 can be configured to recognize any frames which have Multicast Addresses.

2.1.2.4 RECOGNIZING MULTIPLE INDIVIDUAL ADDRESSES

The 82590 can recognize Multiple Individual Addresses (Multi-IA) during reception. When this feature is enabled it can receive multiple, individually addressed frames. The 82590 can register a set of Individual Addresses using the MC-Setup command. In order to use the feature a CONFIGURE command must be executed. The Multi-IA and Multicast address filterings share the same 64-bit Hash table and use the same hashing function.

2.1.3 Length Field

The Length field is a 2-byte field whose value indicates the number of LLC data bytes in the frame. The 82590 can be configured to extend short frames with pad bytes during transmission and to remove the pad bytes during reception according to the Length field. The operation of padding and stripping is compatible with the IEEE 802.3 standard. If this feature is not enabled, the Length field is treated in the same way the 82590 treats the LLC Data field.

2.1.4 LLC Data and Pad Fields

The Logical Link Control (LLC) Data and Pad fields follow the Length field. They contain the actual LLC data being transferred in the frame. If the total length of a frame is shorter than the slot time the frame may be padded with pad bytes.

```
program hashtable (input, output);

type mctype = array [0..47] of boolean; {Bit pattern that rep-
                                         resents a Multicast Address.
                                         0 is the least significant
                                         bit. Addresses shorter
                                         than 6 bytes are justified
                                         to the least significant bit.}
     httype = 0..63;  {Hash table}

var addresslength: 1..6;
    crclength: integer;  {16 or 32}
    mcv: mctype;  {Multicast Address Vector}
    k, l, m: integer;
    digit: char;
    numb: integer;

function hash (al, cl: integer; x: mctype): httype;
{This function returns a hash value of X, according to the address
 length (al) and the CRC length (cl).}
var
    a: array [0..31] of boolean;  {32-bit shift register}
    fb: boolean;  {Feedback path}
    i: integer;
    y: httype;  {y = Hash (x) }

begin
    for i := 0 to 31 do a [i] := true;  {Initialize to all ones}
    for i := 0 to al*8-1 do
    begin
        if cl = 16 then
        begin  {CRC-16}
            fb := a[15] <> x[i];  {Compute feedback}
            a[15] := a[14];  {Shift the CRC register}
            a[14] := a[13];
            a[13] := a[12];
            a[12] := a[11] <> fb;  {Exclusive-OR}
            a[11] := a[10];
            a[10] := a[9];
            a[9] := a[8];
            a[8] := a[7];
            a[7] := a[6];
            a[6] := a[5];
            a[5] := a[4] <> fb;
            a[4] := a[3];
            a[3] := a[2];
            a[2] := a[1];
            a[1] := a[0];
            a[0] := fb;
        end
        else
        begin  {CRC-32}
            fb := a[31] <> x[i];  {Compute feedback}
            a[31] := a[30];  {Shift the CRC register}
            a[30] := a[29];
            a[29] := a[28];
            a[28] := a[27];
```

G50002

Figure 2-3. Hash Bit Calculation Program

```
            a[27] := a[26];
            a[26] := a[25] <> fb;  {Exclusive-OR}
            a[25] := a[24];
            a[24] := a[23];
            a[23] := a[22] <> fb;
            a[22] := a[21] <> fb;
            a[21] := a[20];
            a[20] := a[19];
            a[19] := a[18];
            a[18] := a[17];
            a[17] := a[16];
            a[16] := a[15] <> fb;
            a[15] := a[14];
            a[14] := a[13];
            a[13] := a[12];
            a[12] := a[11] <> fb;
            a[11] := a[10] <> fb;
            a[10] := a[9] <> fb;
            a[9] := a[8];
            a[8] := a[7] <> fb;
            a[7] := a[6] <> fb;
            a[6] := a[5];
            a[5] := a[4] <> fb;
            a[4] := a[3] <> fb;
            a[3] := a[2];
            a[2] := a[1] <> fb;
            a[1] := a[0] <> fb;
            a[0] := fb;
        end;
    end;
    if a[5] then y := 1 else y := 0;
    if a[6] then y := y + 2;
    if a[7] then y := y + 4;
    if a[2] then y := y + 8;
    if a[3] then y := y + 16;
    if a[4] then y := y + 32;
    hash := y;
end;

begin {main}
    writeln (
      'This prog calculates the hash funct used for MC filtering');
    writeln ('Type CTRL/C for exiting the program');
    writeln;
    write ('Enter CRC length => ');
    readln (crclength);
    if (crclength <> 16) and (crclength <> 32) then writeln (' Error! ')
    else
    begin
        write ('Enter address length (in bytes) => ');
        readln (addresslength);
        if addresslength > 6 then writeln (' Out of Range! ')
        else
        begin
            repeat
                writeln;
                for k := 0 to 47 do mcv[k] := false;
                writeln ('Enter Multicast Address (exactly ',
                         addresslength*2:3, ' hexadecimal digits');
                write ('starting with most significant) => ');
                for k := addresslength*2-1 downto 0 do
```

G50002

Figure 2-3. Hash Bit Calculation Program (Cont'd.)

```
                begin
                    read (digit);
                    case digit of
                        '0': numb := 0;
                        '1': numb := 1;
                        '2': numb := 2;
                        '3': numb := 3;
                        '4': numb := 4;
                        '5': numb := 5;
                        '6': numb := 6;
                        '7': numb := 7;
                        '8': numb := 8;
                        '9': numb := 9;
                        'a': numb := 10;
                        'b': numb := 11;
                        'c': numb := 12;
                        'd': numb := 13;
                        'e': numb := 14;
                        'f': numb := 15;
                        'A': numb := 10;
                        'B': numb := 11;
                        'C': numb := 12;
                        'D': numb := 13;
                        'E': numb := 14;
                        'F': numb := 15;
                    end;

                    for 1 := 0 to 3 do
                    begin
                        if odd(numb) then mcv[4*k+1] := true
                        else mcv[4*k+1] := false;
                        numb := numb div 2;  {Shift right}
                    end;
                end;
                writeln;
                writeln;
                if not mcv[0] then writeln ('not a multicast address!');
                writeln ('HASH = ', hash (addresslength, crclength, mcv));
            until false;
        end;
    end;
end.
```

G50002

Figure 2-3. Hash Bit Calculation Program (Cont'd.)

2.1.5 Frame Check Sequence

The Frame Check Sequence (FCS) Field protects against bit errors in the frame. It is the result of a Cyclic Redundancy Check computed from the SFD field up to the FCS field. The chip can be configured to one of two CRC algorithms: the CCITT V-41 (HDLC) 16-bit polynomial or the Autodin II (IEEE 802.3) 32-bit polynomial.

The CRC mechanism is transparent to the user. During transmission the 82590 calculates a CRC sequence and inserts it in the Frame Check Sequence field. During reception the 82590 verifies correctness of the incoming frame by calculating a CRC sequence for an integral

number of bytes starting from the beginning of the Destination field to the end of the frame. Potential residual bits at the end of frame, beyond the last byte boundary, are ignored for the calculation. If the calculated result is different from the expected value for the CRC algorithm being used, an error is reported.

2.2 FRAME TRANSMISSION

The 82590 transfers data from host memory to the network when the host CPU issues a TRANSMIT command. The CPU prepares the frame in memory before instructing the 82590 to start a transmission. Parts of the frame such as the Preamble, Source Address, CRC, and End of Frame flag in the HDLC framing are automatically inserted by the 82590. The 82590 resolves access and possible contention on the link. When the transmission is completed the 82590 updates a set of status registers and raises its Interrupt signal to inform the CPU.

2.2.1 Link Management

The 82590 implements the following three link management algorithms:

1. Carrier Sense Multiple Access with Collision Detection (CSMA/CD) according to the IEEE 802.3 standard.
2. CSMA with Deterministic Collision Resolution (CSMA/DCR). This protocol is compatible with the Intel MULTIBUS II Serial System Bus (SSB) specifications.
3. Full Duplex Communication where the previous two protocols are completely disabled and the 82590 is controlled only by the Request-to-Send and Clear-to-Send signals.

In addition, the 82590 has numerous programmable configuration parameters, allowing any of the above algorithms to be optimized for a particular network topology or application. Station priorities are also programmable with the 82590.

2.2.1.1 CSMA/CD

The 82590 constantly monitors link activity. Whenever it senses activity on the link the 82590 defers to the passing frame by delaying any pending transmission. When there is no more activity on the link the 82590 continues to defer for Inter Frame Spacing (IFS) time (minimum time between two consecutive frame transmissions). This parameter is configurable from 12 to 255 bit times. If at the end of that time the 82590 has a frame waiting to be transmitted, transmission is initiated regardless of the link activity.

Once the 82590 has finished deferring and has started transmission, it is possible to experience link contention. This situation is called a collision and is generally detected and signaled by the transceiver. The 82590 can be programmed to detect collisions internally or externally via the $\overline{CDT}$ pin. When the 82590 experiences a collision it enforces the collision by transmitting 32 to 48 bits of jam pattern.

If a collision is detected during the Preamble, transmission of the preamble and SFD is completed before jamming starts. The collision enforcement mechanism ensures detection of the collision by all the stations.

In all encoding schemes except NRZI, the jam pattern is "one's complement" of the CRC sequence being calculated up to the time the collision is detected. The complement of the CRC is transmitted as the last two or four bytes of the jam sequence. In NRZI encoding it is a sequence of all zeros.

The dynamics of collision handling are largely determined by the Slot Time. Slot Time is the maximum end-to-end, round trip delay time of the network plus jam time in a baseband, bus topology network. It is twice the maximum end-to-end, round trip delay plus jam time in networks that have certain concentration points. Examples of the concentration points are the head-end remodulators in broadband networks and the hubs in StarLAN networks. Slot Time is very important because it is the worst case time to detect a collision (often referred to as the collision window). The Slot Time for the 82590 is programmable from 1 to 2048 bit times.

The 82590 notifies the CPU of a collision and provides an opportunity to retransmit at the end of the Backoff timeout. Alternatively, the 82590 can be programmed to notify the DMA controller of the collision by asserting the End-of-Process (EOP) signal. The 82590 and the DMA controller, which is configured for Auto-Initialized mode, automatically retransmit at the end of the Backoff timeout. When the 82590 is configured for this operation, retransmission is done without CPU intervention.

The 82590 implements the IEEE 802.3 scheduling of retransmission. The controlled randomization process is called "truncated binary exponential backoff." A retransmission attempt is delayed for an integral multiple of slot times. The number of slot times to delay before the nth retransmission attempt is chosen as a uniformly distributed random integer in the range of $0 \leq r < 2^k$, where $k = \min [n, 10]$. The number of retransmission attempts is programmable in the range 1 to 15. The beginning of the backoff time is configurable to one of two methods. If configured to the IEEE 802.3 compatible method, the backoff time starts immediately after jam transmission. If configured to the Alternate Backoff method (designed for lower bit rates and/or shorter topologies), the backoff time starts after the deferring period following jam transmission.

After the backoff time has expired and the CPU has issued a RETRANSMIT command, the 82590 attempts to retransmit the frame unless the number of retransmission attempts has exceeded the maximum allowed.

The 82590 maintains a retry counter that is incremented after each collision. If the number of retries equals the programmed maximum an error is reported and automatic retransmission is disabled. The number of allowed retries is configurable from 0 to 15. The only difference between transmission and retransmission is that transmission clears the retry counter and retransmission increments it.

On completion of transmission and successful retransmission the 82590 reports the total number of collisions that have occurred and whether they equaled or exceeded the maximum. It also indicates if the first transmission attempt had to defer to passing traffic.

After transmission has started, the 82590 attempts to transmit the entire frame. In the normal case, frame transmission is completed and the host CPU is notified through interrupt and status register. Otherwise, one or more of the following events causes transmission to terminate prematurely: Clear-to-Send signal goes inactive during transmission, data transfer rate from memory to the chip does not keep up with transmission (DMA underrun), Carrier Sense goes inactive during transmission (if the chip is in the mode where carrier sense is expected during transmission), or a collision is detected via Collision Detect Signal. These events are also reported to the CPU via the status registers.

The user may attempt to abort transmissions. Upon receipt of the ABORT command, the 82590 transmits a Jam pattern to cause a CRC error. It reports to the host CPU either that the ABORT succeeded or that the frame transmission was completed before the ABORT was accepted by the 82590.

2.2.1.2 DETERMINISTIC COLLISION RESOLUTION (DCR)

During non-collision mode the protocol operates as in normal CSMA/CD mode. However, when a collision occurs, all stations enter the deterministic collision resolution (DCR) mode. In order for the stations to enter the DCR mode upon a collision, a receive mode collision detection must be implemented. The collision resolution period is broken into a programmable number of slots, and each station is assigned a unique slot. The first slot begins one interframe space time after the link becomes inactive.

A station may begin transmitting only during its specific slot. When a station detects a transmission its slot counting is inhibited until the end of the transmission and one IFS expires. The slot counting then continues at the next slot number. During the DCR process, not only may stations involved in the collision transmit, but any other stations which have frames pending may also transmit in their assigned slots. After the last slot expires, normal CSMA/CD mode resumes.

If a collision is detected during resolution mode, which would be an exception condition, the resolution process is restarted.

2.2.1.3 DCR EXAMPLE 1

This example illustrates resolution of a two-station collision.

Let us suppose that a station which owns slot No. 2 is called *Station A*, and another station which owns slot No. 4 is *Station B*. Station A and Station B collide and enforce the collision by jamming (see Figure 2-4). When Station A detects the beginning of slot No. 2 at T3, it initiates transmission in its assigned slot. During Station A's frame transmission, slot counting is inhibited. After transmission ends at T4, one IFS time is measured and slot counting resumes at T5. When Station B detects the beginning of slot No. 4 at T6, it starts transmission. All stations suspend slot counting until T7, and resume counting at T8 after one IFS time. All stations count slots No. 5 and No. 6. The network operates in the normal CSMA/CD mode after the last slot—in this example slot No. 6—is counted.

2.2.1.4 DCR EXAMPLE 2

This example illustrates how a third station may transmit during resolution of a two-station collision.

A station not involved in a collision may transmit a frame during a DCR process. Suppose a third station, *Station C*, is assigned slot No. 3. If a transmission is pending at Station C before T5 in Figure 2-4, the station may transmit when slot No. 3 is reached. This is shown in Figure 2-5.

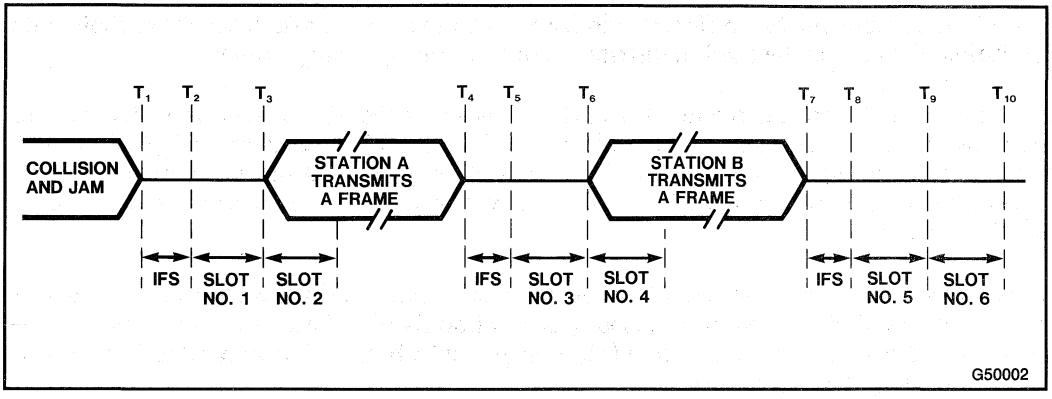


Figure 2-4. DCR for a Two-Station Collision

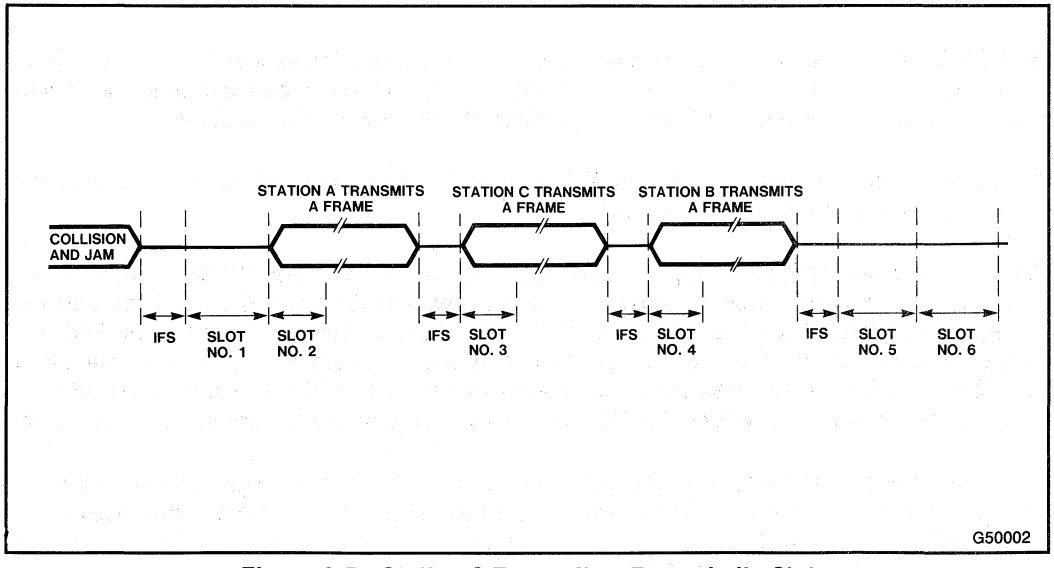


Figure 2-5. Station C Transmits a Frame in its Slot

If Station C decides to transmit after T6 (see Figure 2-4), it must wait until the DCR process is completed.

2.2.1.5 FULL DUPLEX

When the 82590 is configured for Full Duplex mode, it does not use the CSMA/CD or DCR protocol. The 82590 can start transmitting regardless of its receiver state. The IFS timer is never started after reception or transmission in this mode. Collisions are not allowed in this mode.

2.2.2 Priority Mechanism

The 82590 implements two different priority mechanisms: linear and accelerated contention resolution. Either may be used to distribute relative priority among stations.

Linear priority determines the number of Slot Times that the 82590 waits after deferring or after the end of the Backoff Time (whichever occurs last) before transmitting. If the link becomes busy during the wait period, the process of deferring and waiting starts again. The Linear priority is programmable from 0 to 7. Zero provides the highest priority and is IEEE 802.3 compatible.

With accelerated contention resolution, the number of slot times the 82590 will wait before the nth retransmission attempt, is choosen as a uniformly distributed random integer in the range of $0 \leq r < 2^k$; where $k = \min(10, n + p)$, and where p is configurable from 0 to 7, with 0 providing the highest priority.

2.3 FRAME RECEPTION

The 82590 receives and passes to memory every frame whose address matches its individual, multicast, or broadcast address. By configuring the chip to the promiscuous mode, it will receive and pass to memory all frames, regardless of their destination address.

The 82590 constantly monitors the serial link activity. When the link becomes active, the 82590 starts accepting the incoming bits.

The Preamble and the SFD (or the BOF flag) are discarded. The 82590 compares the incoming frame's Destination Address to its own Individual Address. If there is an address match and the frame length is at least 6 bytes, the 82590 passes the Destination Address, Source Address, Length field, and Data field to system memory via the DMA controller. The 82590 verifies correctness during reception of the FCS field. If there is no address match the 82590 does not request the DMA services, and prepares to receive the next frame.

When the reception is completed the 82590 updates a set of status registers and raises its Interrupt signal to inform the CPU. If the received frame has errors (CRC error, alignment error, DMA overrun, etc.), the CPU can reclaim the memory used to store that frame. The next received frame can then be stored in the reclaimed memory.

When configured to IEEE 802.3 End of Carrier delineation, End of Frame is indicated by the carrier going inactive. The number of bits after the SFD must be a multiple of eight. Residual 'dribble' bits are discarded, and not included for CRC checking. An alignment error is reported when the frame has a CRC error and dribble bits.

When configured to HDLC Bit Stuffing mode, the 82590 performs 0-bit deletion, discards the EOF flag, and discards all following bits until the carrier becomes inactive. Residual bits are discarded in a manner similar to the End of Carrier method. An error is reported if the carrier becomes inactive prior to recognition of an EOF flag.

The minimum frame length is configurable from 6 to 255 bytes. Any frame containing less than the minimum (configured) number of bytes is presumed to be a fragment resulting from a collision or an aborted transmission. Any frame shorter than 6 bytes is discarded without notifying the CPU.

2.3.1 Single Buffer Mode

The entire frame is received in one continuous buffer. This method of reception is called "Single Buffer" reception. Upon completion of reception the total number of bytes written into the memory buffer is loaded into the internal status registers, and the status of the reception itself is appended to the received frame. An interrupt to the CPU is then generated.

2.3.2 Buffer Switching Mode

If the frame size is unknown, memory usage is optimized by using "Buffer Switching" reception. The frame, as it comes in, is deposited into a series of fixed size buffers. This way the user does not have to allocate large memory spaces for short frames. Instead, the 82590 can dynamically allocate memory space as it receives frames. This method requires both data channels to alternately receive the frame. As the frame reception starts, the 82590 interrupts the CPU and automatically requests assignment of the next sequential buffer. The CPU does this and loads the second DMA channel with the next buffer information so the 82590 can immediately switch to the other channel as soon as the current buffer is full. When the 82590 switches from the first to the second buffer it again interrupts the CPU requesting it to allocate, in advance, another buffer on the other (previous) channel. This process continues until the entire frame is received. The received frame is spread over multiple memory buffers.

This dynamic (pre)allocation of memory buffers results in efficient use of available storage when handling frames of widely varying sizes. Since the buffers are preallocated one block in advance, the system is not time critical.

2.3.3 Frame Switching Mode

Instead of switching buffers during one frame reception, the 82590 can be configured to switch buffers between frame receptions. This mode of reception is called Frame Switching. Each frame is received into one continuous buffer, much like Single Buffer reception, but

buffer switching is automatically performed at the end of each frame reception. This method can be implemented with one or two DMA channels. When implemented with only one DMA channel, it uses the End-of-Process ($\overline{EOP}$) signal to request the DMA controller to reload its address register with a new address at the end of a frame reception. The 82380 DMA controller supports these functions and can be used for this application. When two DMA channels are available, the operation is very similar to Buffer Switching. As one frame reception starts, using one DMA channel, the 82590 interrupts the CPU and requests assignment of another buffer in order to be ready for the next frame reception. The CPU allocates a new buffer and loads the second DMA channel with the new buffer information. When the 82590 finishes receiving the current frame, it immediately switches to the alternate DMA channel. When the next frame arrives it is received in the new buffer using the alternate DMA channel. This process is repeated on every frame reception.

When Frame Switching is implemented with two DMA channels, the 82590 can be programmed to assert the $\overline{EOP}$ signal when it detects a receive error. The DMA controller, which is programmed to Auto-Initialize mode, will perform automatic buffer reclamation for the otherwise wasted buffer in this configuration, and the 82590 will perform buffer switching only at the end of a good frame reception.

2.4 PHYSICAL LINK INTERFACE

The Physical Link Interface includes the following functions: clock generation, data encoding, data decoding, carrier sensing, collision detection, and transceiver handshake. The 82590 can be programmed to support either of two interfaces. In High Integration Mode it supports a highly integrated interface that provides most of these functions on chip and requires minimum external support logic. In High-Speed Mode it supports a highly flexible interface that is identical to the physical interface of the 82586 and requires external logic in order to connect to the link. The Physical Link layer functions are presented below in three groups: clock generation, data encoding and recovery; detecting carrier and collisions; and handshake with the transceiver.

2.4.1 Clock Generation/Encoding/Decoding

2.4.1.1 CLOCK GENERATION

In High-Integration Mode, the user can provide the 82590 with an external sampling clock or let the 82590 generate the sampling clock using the on-chip crystal oscillator. The bit clock is generated internally from the sampling clock that runs at 8X, 10X, 16X, or 18X the data bit rate. The frequency range of the crystal or the external clock can vary between DC and 32 MHz. The maximum data bit rate in this mode is 4 Mb/s with the combination of 8X sampling and 2 32-MHz clock frequency. In High-Speed Mode, the 82590 receives the externally generated Transmit Clock ($\overline{TxC}$) and Receive Clock ($\overline{RxC}$) at the exact bit rate. This frequency can vary from DC to 20 MHz.

2.4.1.2 ENCODING/DECODING

In High-Integration Mode, the waveforms sent out by the 82590 on the TxD pin and received on the RxD pin, are programmable to be either Manchester, Differential Manchester, or NRZI encoded. The idle state of the signal is High.

In High-Speed Mode, the 82590 transmits and receives NRZ encoded data that is synchronized to the respective clocks. Encoding/decoding to and/or from different schemes is normally done externally. However, the 82590 can be programmed to generate Manchester encoded data in this mode (in the frequency range of 1 to 20 Mb/s).

Figure 2-6 depicts various kinds of data encoding methods implemented in the 82590. Table 2-1 illustrates the encoding rules for these methods.

2.4.2 Carrier Sense/Collision Detect

2.4.2.1 CARRIER SENSE

Indicates that there is activity on the link.

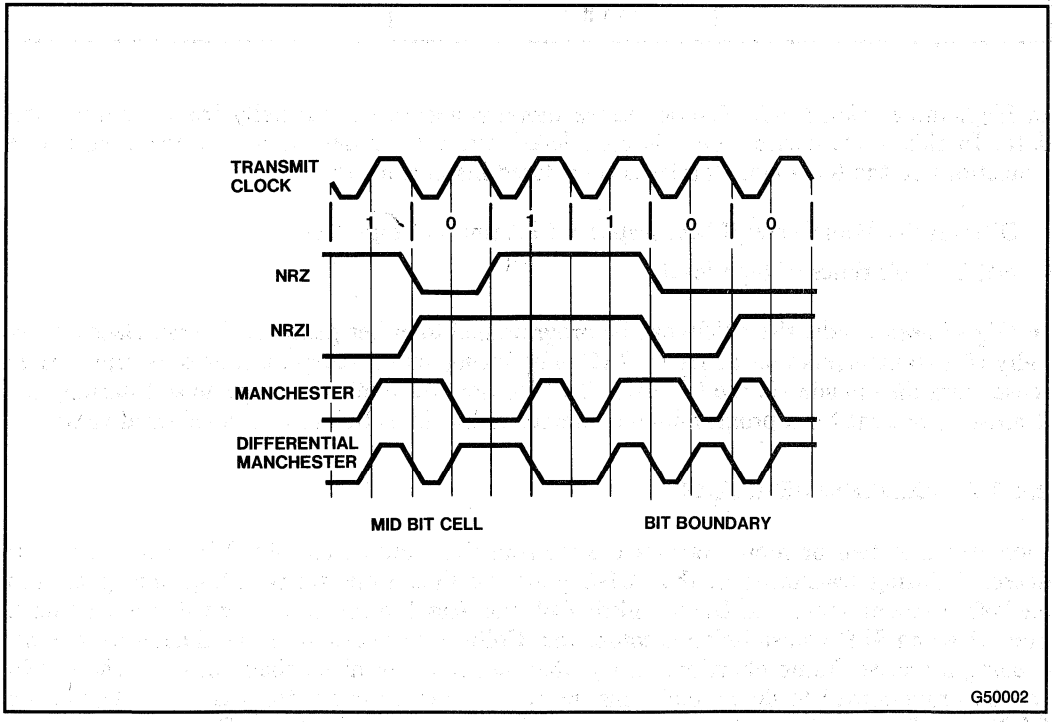


Figure 2-6. Data Encoding Schemes

Table 2-1. Data Encoding Rules

Encoding Method	Mid Bit Cell Transitions	Bit Cell Boundary Transitions
NRZ	Do Not Exist	Identical to Original Data
NRZI	Do Not Exist	Exist Only if Original Data Bit Equals 0. Dependent on Present Encoded Signal Level: to 0 if 1 to 1 if 0
MANCHESTER	Exist for Every Bit of the Original Data: from 0 to 1 for 1 from 1 to 0 for 0	Exist for Consequent Equal Bits of Original Data: from 1 to 0 for 1 1 from 0 to 1 for 0 0
DIFFERENTIAL MANCHESTER	Exist for Every Bit of the Original Data. Dependent on Present Encoded Signal Level: to 0 if 1 to 1 if 0	Exist Only if Original Data Bit Equals 0. Dependent on Present Encoded Signal Level: to 0 if 1 to 1 if 0

In High-Integration Mode, Carrier Sense signal is generated internally from the incoming data. In this mode carrier sense is considered active after detection of three consecutive transitions on the RxD pin. Carrier is considered inactive after:

- Differential Manchester/Manchester: 1.5 bit times of high level
- NRZI: 8 bit times of high level

In High-Speed Mode, the 82590 can be programmed to either generate Carrier Sense internally (for transceivers that generate RxC only during actual reception), or to accept carrier sense from the outside via the $\overline{\text{CRS}}$ pin. The Carrier Sense may undergo noise filtering, i.e., it must be present for a programmable number of bit times before it is considered valid.

2.4.2.2 COLLISION DETECTION

Indicates that two or more stations are transmitting simultaneously. When a collision is detected during transmission, the 82590 jams the link, stops transmitting, and goes into backoff (random backoff or Deterministic Collision Resolution). Jamming will not start until preamble and SFD transmission is completed. Collisions are also detected during receptions, causing a receive frame abortion. (Only when not in external loopback mode.) The 82590 can be programmed to detect collisions internally or to accept it from the outside, via the $\overline{\text{CDT}}$ pin. The Collision Detect signal may undergo filtering, similar to Carrier Sense.

2.4.2.3 COLLISION DETECTION BY CODE VIOLATION (CDBCV)

This scheme is available only in High-Integration Mode. A collision is detected when the receive data experiences code violations (Manchester, Differential Manchester, or NRZI). If the 82590 is transmitting at the time the collision is detected, it will perform jamming and start the backoff period. If it is receiving a frame, the frame will be truncated and an error will be reported.

Manchester code violations are detected during sampling and decoding of incoming Manchester-encoded data. The 82590's internal sampling circuits continuously count the number of sampling clock periods from one transition to the next. If a data transition should occur at time 0 (reference transition), then the sampling instant of the next transition determines whether the pulse was an invalid pulse (a code violation), a long pulse, or a short pulse. The Manchester encoding rules require the long pulse to have a nominal width of one bit time and the short pulse to have a nominal width of one-half bit time. Figure 2-7 illustrates this sampling/decoding mechanism for 8X sampling configuration. A bit is sampled eight times in this configuration. If the pulse width is found to be invalid, namely 1/8 to 2/8, 6/8, or 11/8 to 13/8, a collision due to a code violation is reported. A collision is also reported if the data line remains low for 13/8, or more, from the reference transition. Conditions for code violations are summarized below:

- 8X Manchester: Pulse width as sampled by the 82590:
  - —1/8 to 2/8, 6/8, or 11/8 to 13/8
  - —"0" level for 13/8 or more
  - —Missing mid-bit-cell transition
- 10X Manchester: Pulse Width as sampled by the 82590:
  - —1/10 to 2/10 or 13/10 to 15/10
  - —"0" level for 15/10 or more
  - —Missing mid-bit-cell transition
- 16X Manchester: Pulse width as sampled by the 82590:
  - —1/16 to 4/16, 12/16, or
  - —20/16 to 24/16
  - —"0" level for 24/16 or more
  - —Missing mid-bit-cell transition
- 18X Manchester: Pulse width as sampled by the 82590:
  - —1/18 to 4/18 or 23/18 to 27/18
  - —"0" level for 27/18 or more
  - —Missing mid-bit-cell transition

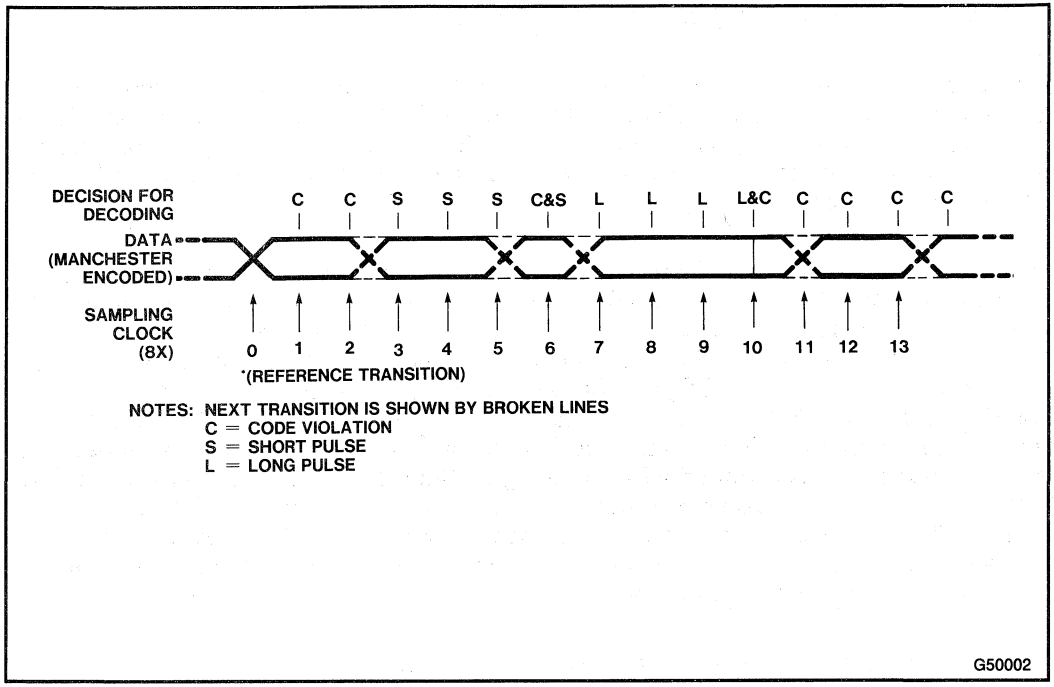


Figure 2-7. 8X Sampling/Decoding

Code violations in NRZI are detected by the on-chip Digital Phase-Lock Loop (DPLL). The DPLL detects the following conditions as code violations.

- NRZI: Depends on the DPLL Integrator configuration.
  - —For DPLL Integrator = 0: A transition that is 1/4 bit time or more out of phase. More than two transitions in one-half bit cell
  - —For DPLL Integrator = 1 to 7: Two transitions in one-half bit cell

Note that if the DPLL Integrator is configured to 0, the conditions for a code violation in NRZI encoding are identical to that of the 82588.

2.4.2.4 COLLISION DETECTION BY BIT COMPARISON (CDBBC)

This scheme is available both in High-Integration and High-Speed modes. In this scheme, the 82590 compares the signature of the transmitted data to the signature of the received data for the duration of the collision window (one slot time). Figure 2-8 shows the block diagram of the logic. As a frame is transmitted it flows through the CRC (CRC-16) generation logic. The transmit slot time timer is started at the same time the CRC generation starts. When the timer measures one slot time, the value of the CRC generator is latched in as the transmit signature. As the transmitted frame is returned in the receive channel it flows through the CRC checker. The receive slot time is started at the same time the CRC checker starts checking. When the receive slot time timer expires the value of the CRC

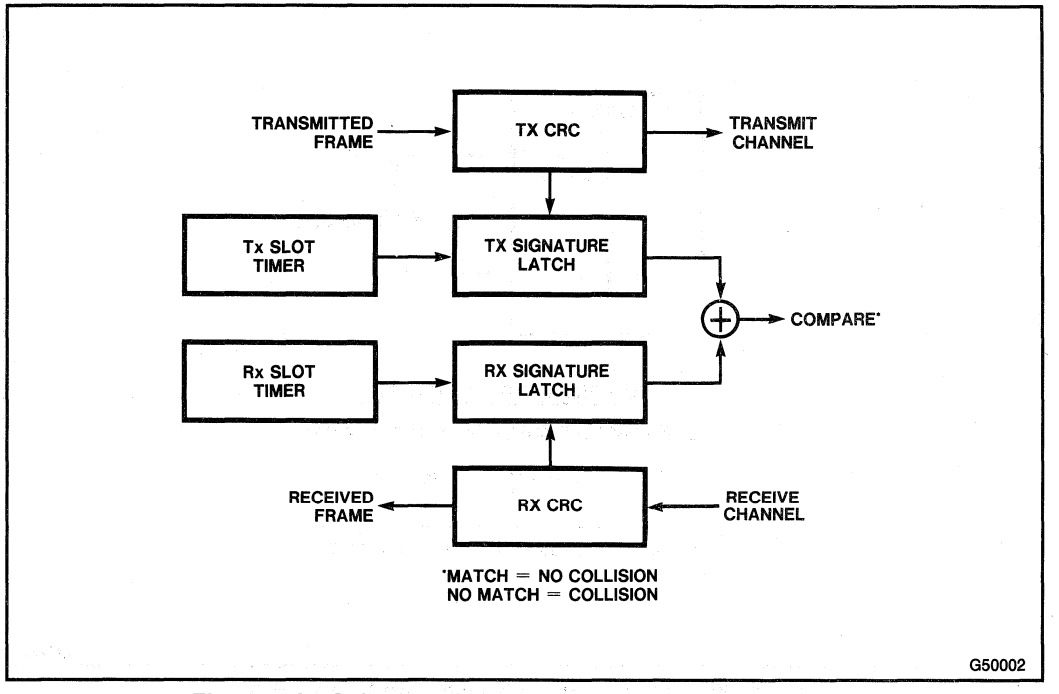


Figure 2-8. Collision Detection by Bit Comparison (CDBBC)

checker is latched in as the receive signature. A collision is reported if the transmit signature differs from the receive signature.

In order to detect a collision using the CDBBC method, the transmitter must be actively transmitting up to the point where the receiver has seen enough bits (one slot time after SFD) to complete its signature computation. Otherwise, the transmitter may be finished before the receive signature is computed, leaving the possibility of an undetected collision. In order to avoid this situation the 82590 supports automatic padding of transmit frames, so the transmission time of short frames can be extended. Short frames are padded up to 1.5 times the slot time. This padding scheme ensures reliable CDBBC operation even for networks which do not locally return transmitted frames to the receive channel. One such network is a broadband network. A diagram of a broadband network with a head-end remodulator is shown in Figure 2-9.

In Addition, whenever CDBCV or CDBBC is enabled, timeout mechanisms are automatically enabled to detect abnormal conditions. A collision is reported if, after the transmission of the SFD or BOF, any of the following conditions become true:

1. Half a slot time has elapsed and Carrier Sense did not go active.
2. Half a slot time plus 16 bit times have elapsed and the SFD was not yet recognized.
3. Carrier Sense went inactive after the SFD was received, while transmission is still active.

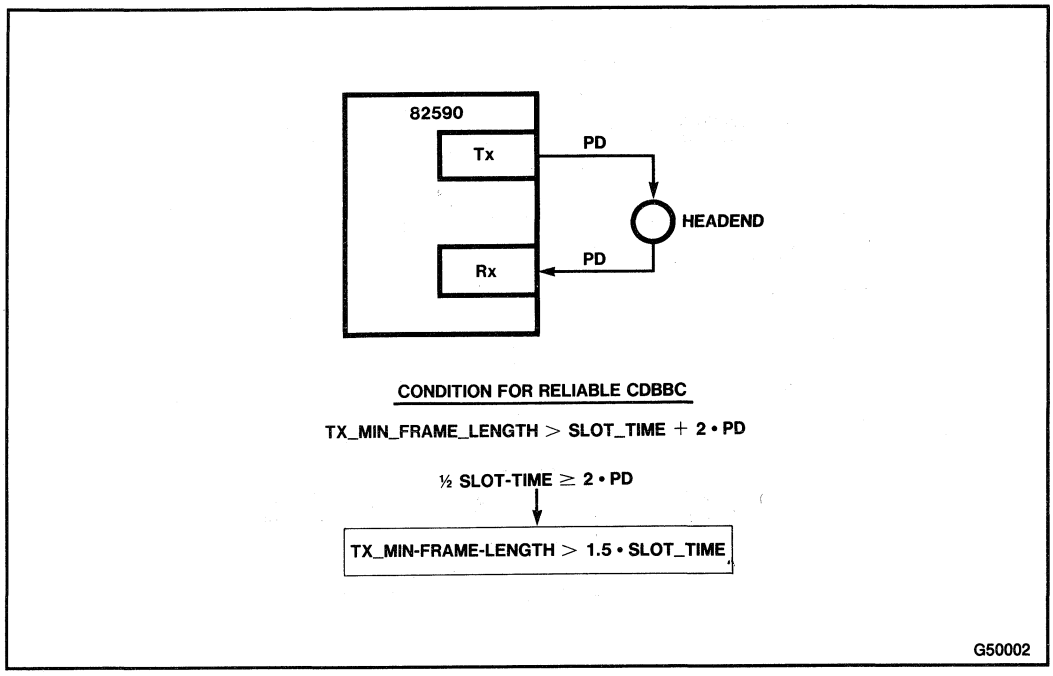


Figure 2-9. Frame Length Requirement for CDBBC

2.4.2.5 COLLISION DETECTION BY SOURCE ADDRESS COMPARISON (CDBSAC)

This scheme is available both in High-Integration and High-Speed modes. The 82590 compares the source address field of the transmitted frame to the source address field of the received frame. If the source addresses are different, a collision is reported.

2.4.2.6 IEEE 802.3 TYPE 1BASE5 (StarLAN) CPS RECOGNITION

This scheme is available only in High-Integration mode. In this scheme the 82590 can recognize the StarLAN specific Collision Presence Signal (see Figure 2-10) and interpret it as a collision.

In High-Speed Mode, internal collision detect works only with transceivers that do not return the transmitted signal. In that case, detection of a carrier during transmission indicates a collision.

2.4.3 Jitter Tolerance

As signals propagate through a transmission line, they experience various impairments (noise). Phase variations of the digital signals are often referred to as jitter because of their jittery edge movements. Some of the causes of jitter are envelope delay distortion, bias distortion,

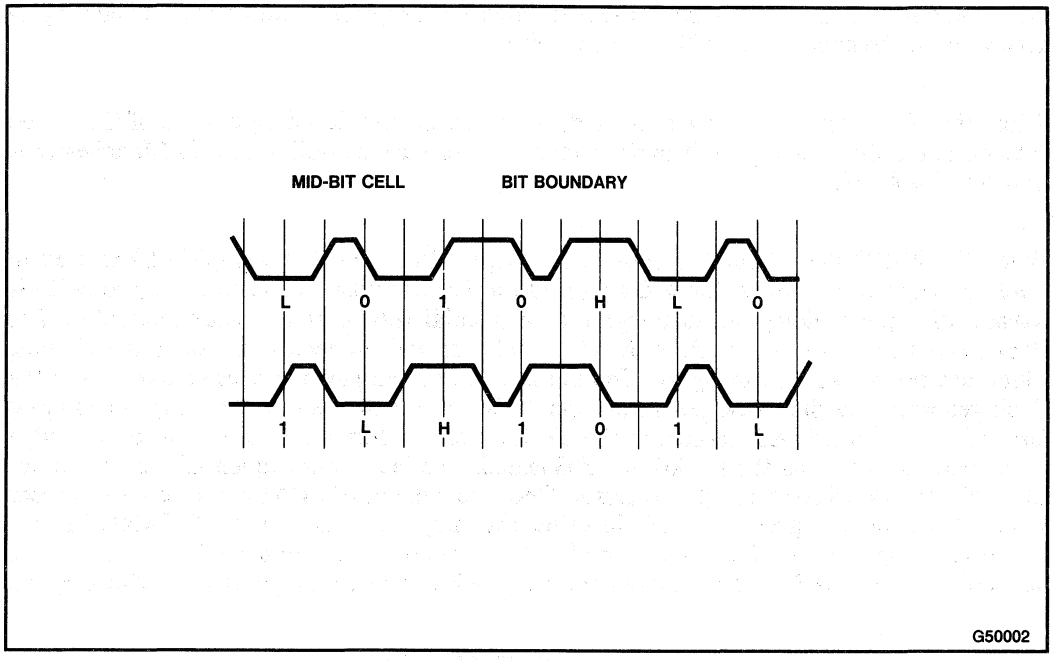


Figure 2-10. StarLAN Collision Presence Signal (CPS) Waveforms

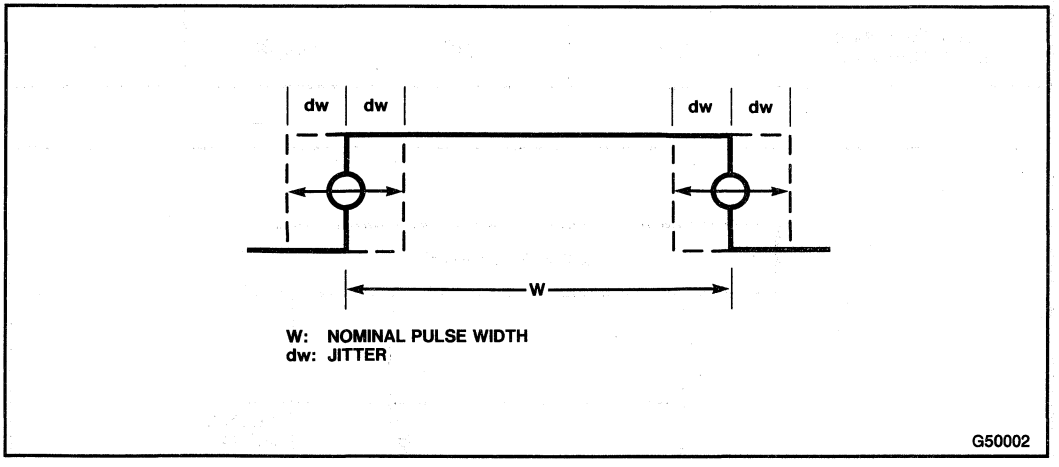


Figure 2-11. Jitter

inter-symbol interference, echo, difference in rise and fall times of line drivers and receivers, and crosstalk. Figure 2-11 shows a digital pulse which contains jitter at both rising and falling edges.

Jitter tolerance is the maximum phase (or edge) distortion the 82590 receiver can tolerate in receive data. Once the 82590 detects an edge which contains a distortion larger than the

jitter tolerance, it may misinterpret the associated bit, or generate an active collision signal according to the configured CDBCV mechanism.

When the Manchester data format is used, the sampling and decoding circuits of the 82590 determine the jitter tolerance. The jitter tolerance for each sampling rate in Manchester is shown in Table 2-2.

When the NRZI data format is chosen, the Digital Phase-Lock Loop (DPLL) is used to track the edge variations. A block diagram of the 82590 DPLL is shown in Figure 2-12. It consists of a phase detector, an integrator (sequential filter), and a clock generator. The clock generator provides a clock to the phase detector and the receive bit sampling circuits, which are not shown in the figure. The phase detector samples the receive data using the clock generated by the clock generator, and determines if the receive data edge has moved forward or backward with respect to the clock edge. It informs the integrator of any edge movements by asserting the LEAD or LAG signal. The integrator sequentially accumulates LEADS and LAGS counts in its integrator. Once the number of LEADS exceeds the number of LAGS by the intergrator's threshold value, the integrator asserts the ADVANCE signal to the clock generator. If the number of LAGS exceeds the number of LEADS by the threshold, the RETARD signal is asserted. The clock generator then adjusts its clock phases

Table 2-2. Jitter Tolerance - Manchester

Sampling Rate	x8	x10	x16	x18
Jitter Tolerance in Bit Times (dw)	±1/16	±1/10	±3/32	±1/9

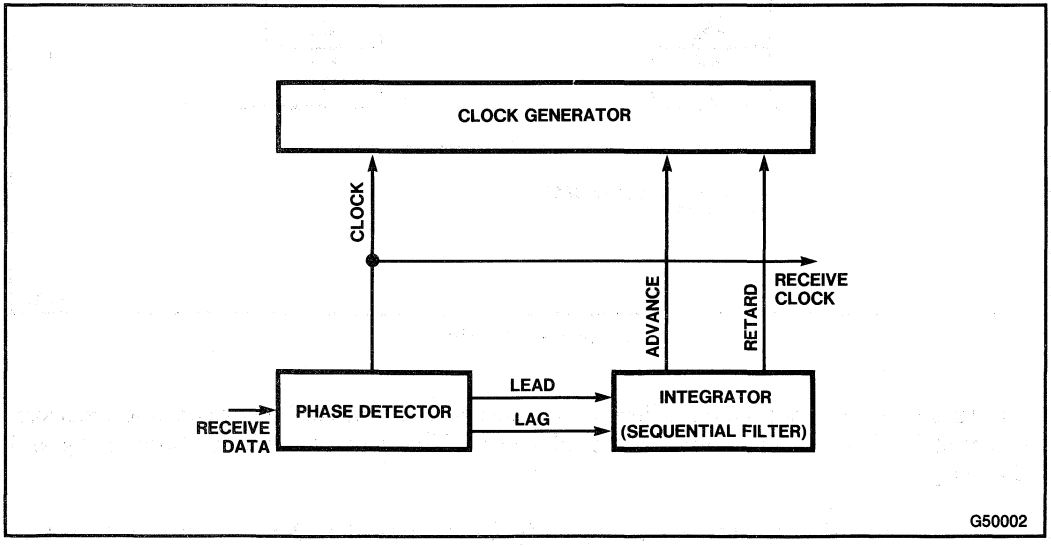


Figure 2-12. Block Diagram of 82590 DPLL

according to the ADVANCE and RETARD signals generated by the integrator. The integrator is reset once a clock adjustment is made in the clock generator.

The lower the integrator's threshold is, the more responsive the integrator is for the phase fluctuations detected by the phase detector. The higer the integrator's threshold is, the less responsive the integrator is for the phase fluctuations. The threshold for the 82590 integrator is calculated from a programmable value. The threshold is one plus the integrator length parameter (see Section 6, Parameter 48).

The jitter tolerance of the 82590 in NRZI mode has been experimentally characterized using artificially generated random jitter. Table 2-3 shows the experiment results. Since the acquisition and tracking behavior of the DPLL also depends on the preamble length (the longer the preamble is, the more edges the DPLL has to acquire the lock), it is important to experiment with the preamble of the desired length. Table 2-3 shows the jitter tolerance in terms of statistical error rates rather than the absolute jitter tolerance. Figures 2-13 and 2-14 graphically show the same results.

Table 2-3. Jitter Tolerance - NRZI

Jitter Tolerance in Bit Times	±3/16	±4/16	±5/16	±6/16
% Errored Frames Integrator = 7 Preamble = 4 Bytes Long	0	0.26	7.75	88.0
% Errorred Frames Integrator = 7 Preamble = 8 Bytes Long	0	0.00055	6.32	91.0
% Errored Frames Integrator = 7 Preamble = 16 Bytes Long	0	0.0004	1.17	69.2
% Errored Frames Integrator = 4 Preamble = 4 Bytes Long	0	0.76	25.9	87.4
% Errored Frames Integrator = 4 Preamble = 8 Bytes Long	0	0.14	25.9	87.1

NOTE: The sampling rate is 8x for all combinations.

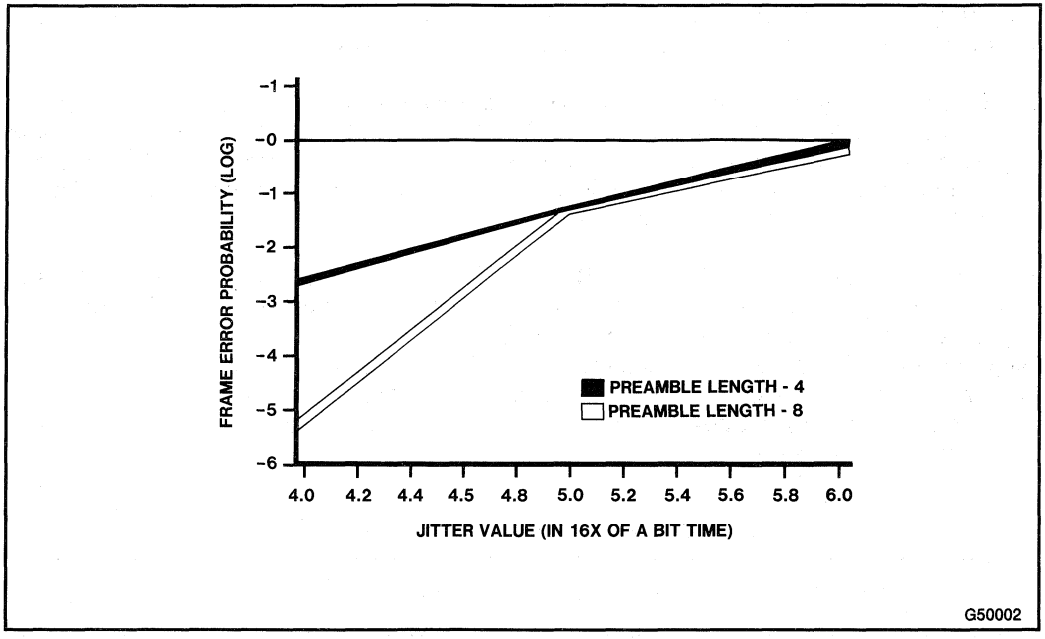


Figure 2-13. Jitter Tolerance—NRZI (Integrator = 7)

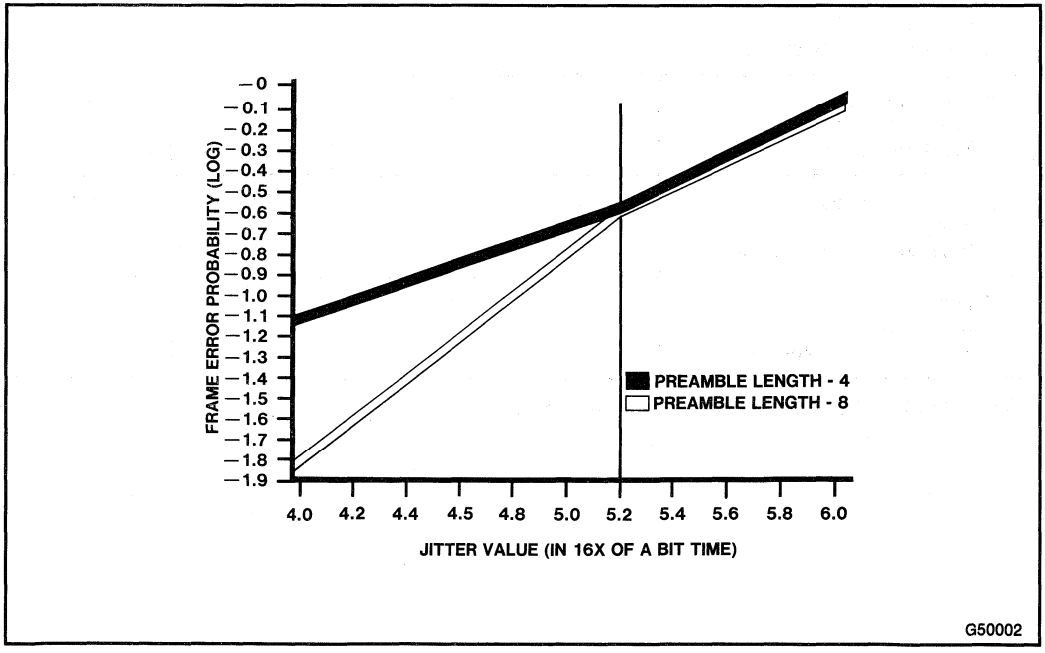


Figure 2-14. Jitter Tolerance—NRZI (Integrator = 4)

Network Management and Diagnostic Functions

3

CHAPTER 3
NETWORK MANAGEMENT AND DIAGNOSTIC FUNCTIONS

The 82590 includes a set of features for improving reliability and testability.

The 82590 offers four diagnostic services: first, monitoring the transmission and reception of frames; second, gathering statistics and diagnostics about the network; third, diagnostic support for a particular station on the network; fourth, a means to test the proper operation of the chip itself.

3.1 TRANSMISSION/RECEPTION ERROR REPORTING

The 82590 stores status information after completing transmission or reception of every frame. If transmission or reception is successful, the OK status bit is set. If transmission is not successful, the cause is given in the status registers. Unsuccessful transmission is caused by one of the following events:

- Lost Carrier Sense: Carrier sense signal did not become active or was lost during transmission.
- Lost Clear-to-Send: Clear-to-Send signal was removed during transmission.
- DMA Underrun: DMA underrun occurred because the system bus did not keep up with the serial transmission.
- Collision: A collision was detected during the last transmission attempt.
- Maximum Number of Collisions: The number of collisions exceeded the maximum allowed.
- Frame Too Long (Optional): The frame transmission was too long and caused the internal jabber inhibit mechanism to stop the transmission.
- Late Collision: A late collision (a collision after the Slot Time is elapsed) is detected.

The 82590 checks each incoming frame and reports on the following receive errors:

- CRC Error: Incorrect CRC in a well aligned frame.
- Alignment Error: Incorrect CRC in a misaligned frame. A misaligned frame with a correct CRC is considered as a good frame and not reported by the 82590.
- Frame Too Short: The frame is shorter than the configured value for minimum frame length.
- Frame Too Long (Optional): The frame is longer than 24 times the Slot Time and was truncated.
- No EOF Flag: Valid only in Bitstuffing mode, Carrier became inactive before EOF flag was detected.
- Overrun: The frame was not completely placed in memory because the system bus did not keep up with incoming serial data.
- Length Error: The number of bytes contained in the LLC data field (after removing pad bytes, if necessary) did not match the byte count indicated in the Length field.

3.2 NETWORK PLANNING AND MANAGEMENT

To perform proper planning, operation, and maintenance of a communication network, the network management entity must accumulate information on network behavior. The 82590 provides an extensive set of network management and diagnostic features that can serve as the basis for a network management entity. These features include: gathering network activity information in the on-chip event counters, reporting additional network activity information in the status bits, optionally storing all frames that appear on the link, and locating opens or shorts on the link. The 82590 also has a mode of operation called monitor mode where it listens to the network and evaluates incoming frames without transferring contents of the frames to memory.

There are three 16-bit event counters on chip. These event counters can be used in one of the following two modes:

1. CRC and Alignment Error Counting mode: Event Counter 1 is incremented by good frames, Event Counter 2 by CRC errors, and Event Counter 3 by Alignment Errors.
2. Fragment and Receive-Based Collision Counting mode: Event Counter 1 is incremented by good frames, Event Counter 2 by collided frames, and Event Counter 3 by other bad frames.

When the event counters are configured for the second mode, all possible network events are recorded in the event counters. Together with the monitor mode, the event counters enable the 82590-based station to be a simple network monitor without any extra hardware. When the 82590 is programmed to collect network statistics without generating the interrupt to the CPU, the monitoring function is done without any CPU intervention.

The following network activity information is provided in the status registers after each frame transmission:

1. Number of collisions: the number of collisions the 82590 experienced in attempting to transmit this frame.
2. Deferred transmission: indicates if the 82590 had to defer to traffic on the link during the first transmission attempt.

The 82590 can be configured for Promiscuous Mode. In this mode the 82590 captures all frames transmitted on the network regardless of the Destination Address. This mode is useful in implementing a monitoring station to capture all frames for network analysis. The Promiscuous Mode can be used with the Monitor Mode if the frame contents are not needed in memory.

The 82590 is capable of determining if there is a short or open circuit anywhere in the network using the Time Domain Reflectometry (TDR) command. When a TDR command (see Section 7.0) is issued, the chip transmits a pulse train and measures the reflection return time. If the network is properly terminated there is no reflection. Consequently the timer runs out and the user is notified that there are no link problems. If a problem is detected, the distance to the reflection source and reason (short or open) are reported.

3.3 STATION DIAGNOSTICS

The 82590 can be configured to External Loopback to support Station Diagnostics. In this mode the 82590 operates in full duplex at full speed. The actual maximum throughput depends on bit rate and system bus speed. The transmitter to receiver interconnection can be placed anywhere between the 82590 and the link.

3.4 82590 SELF TESTING

The 82590 provides several self-test features. The 82590 can be configured to Internal Loopback. In this mode the 82590 disconnects itself from the internal or external Serial Interface Units, and any frame transmitted is received immediately. The 82590 connects the Transmit Data to the Receive Data signal and the Transmit Clock to the Receive Clock. The 82590 can be configured in the External loopback mode. In this mode the Transmitted signal at full data rate comes out at the TxD pin and can be returned to the RxD pin through any external circuitry. This works in full duplex at the full data rate. It tests the complete transmitter, receiver, and the external path. Equality between the transmitted and received frames implies that a large portion of the chip operates correctly. In addition, internal loopback can be used in conjunction with inhibiting the source address insertion and/or CRC insertion by the chip. For example, in internal loopback if a frame is transmitted with an erroneous CRC (using CRC inhibition) the CRC checking mechanism must detect a CRC error. In internal loopback the transmission and reception occurs at one-fourth the programmed data rate. The Dump Command causes the 82590 to write 69 bytes of its internal registers to memory. This is a very powerful capability that can serve as the basis for comprehensive diagnostics.

There are parts of the chip that cannot be checked from the outside. One example is the internal Backoff random number generator. The Diagnose command initiates a self test procedure that exercises these otherwise unaccessible registers and counters, and reports the result.

System Interface 4

CHAPTER 4
SYSTEM INTERFACE

The 82590 has an 8-bit-wide data bus interface and the 82592 has a 16-bit data bus interface. The 82590 can be interfaced to wider buses, but only 8 or 16 bits will be used. The 82590 is a slave on the bus, i.e., it does not perform memory transfers by itself. The transfers are performed by the CPU or a DMA controller. The 82590 has a very general and simple interface, allowing it to operate with a variety of processors and DMA controllers. Figure 4-1 shows the 82590 interface signals. When the $\overline{DACK1}/\overline{CS1}/\overline{EOP}$ pin is configured to function as $\overline{CS1}/\overline{EOP}$, the $\overline{DACK0}$, $\overline{DACK1}$, $\overline{CS1}$, and $\overline{EOP}$ pins must be generated using the external logic shown in Figure 4-2.

There are two kinds of transfers over the bus: Command/Status and Data transfers. The command/status transfers are always 8 bits wide and are performed by the CPU using the D_0-D_7 lines. The data transfers are requested by the 82590 using the DMA request lines. They are typically performed by a DMA controller. Data transfers are 8 bits wide for the 82590, and are 8 or 16 bits wide for the 82592. In relatively slow systems the CPU might also perform the data transfers. In that case, the requests from the 82590 will serve as interrupts to the CPU. This mode of operation depends on the serial data rate and on the dedication of the CPU.

The system interface performs command/status transfers, data transfers, and interrupts.

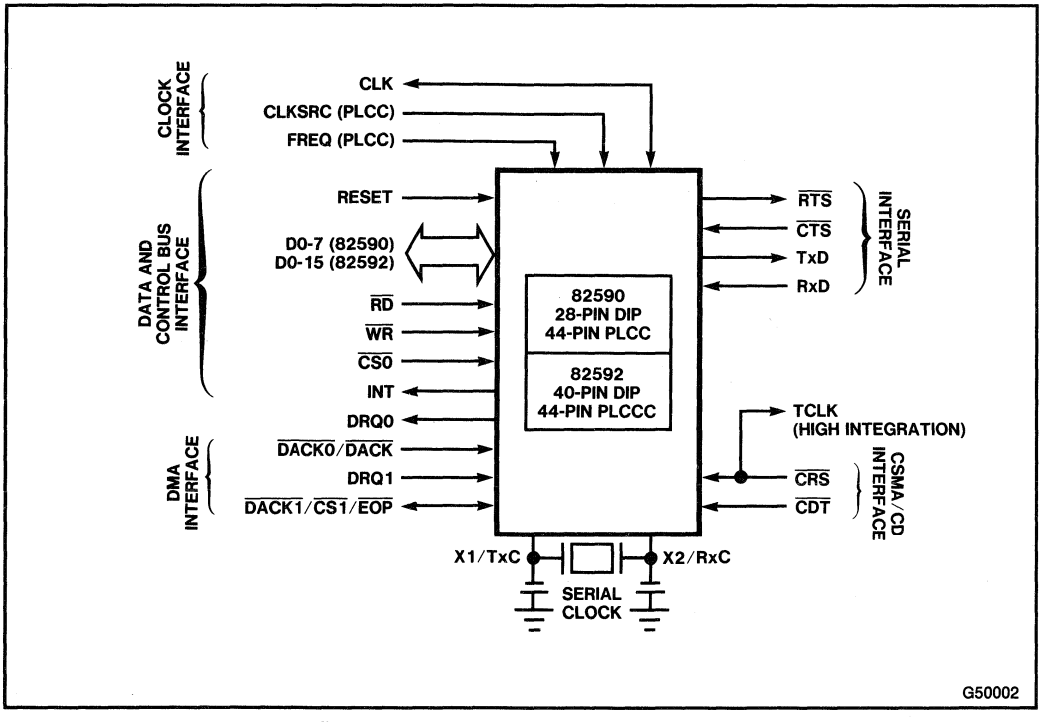


Figure 4-1. 82590 Interface Signals

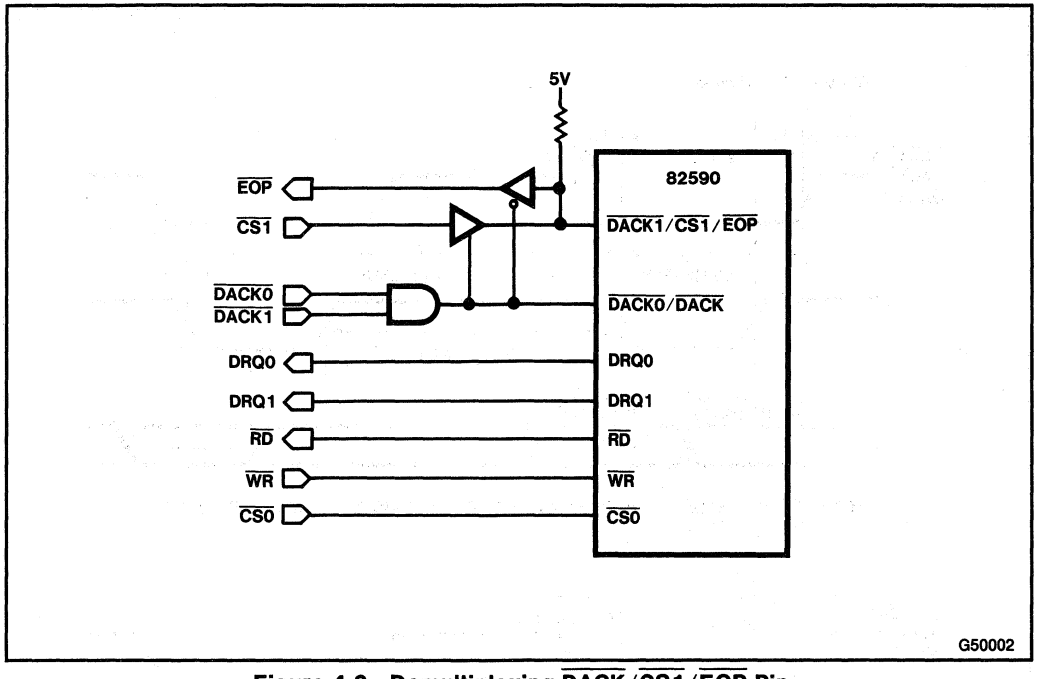


Figure 4-2. Demultiplexing $\overline{DACK}/\overline{CS1}/\overline{EOP}$ Pin

4.1 COMMAND/STATUS TRANSFERS

The CPU controls the 82590 by writing a command into a command register and reading status from a status register at Port 0 or Port 1. The CPU writes a command by activating the $\overline{CS}$ (Chip Select) signal of the desired port, putting the command onto the bus, and activating the $\overline{WR}$ input of the 82590 (see Figure 4-3). In order to read the status from the 82590 the CPU activates the $\overline{CS}$ signal of the desired port and then activates the $\overline{RD}$ line. The 82590 responds by putting the status onto the bus (during the time that $\overline{RD}$ is active). There is no address line with which the CPU can select a specific register within the 82590. The register is selected by the internal 2-bit pointer.

4.2 DATA TRANSFERS

Data transfers are controlled by two pairs of DMA request and acknowledge signals (referred to as two channels): DMA Request signals (DRQ0 and DRQ1) and DMA Acknowledge signals ($\overline{DACK0}$ and $\overline{DACK1}$). Frames and parameters are transferred via the two channels, 0 and 1, to or from memory. The Channel field of a command allocates a specific channel for each operation. A timing diagram for data transfers is shown in Figure 4-4.

In order to request a transfer from memory the 82590 activates a DRQ signal of the specified channel to the DMA controller. In response, the DMA controller activates the corresponding $\overline{DACK}$ signal and starts a data transfer. Data is transferred on the bus and written into the 82590 on the rising edge of the $\overline{WR}$ signal, which is activated by the DMA controller.

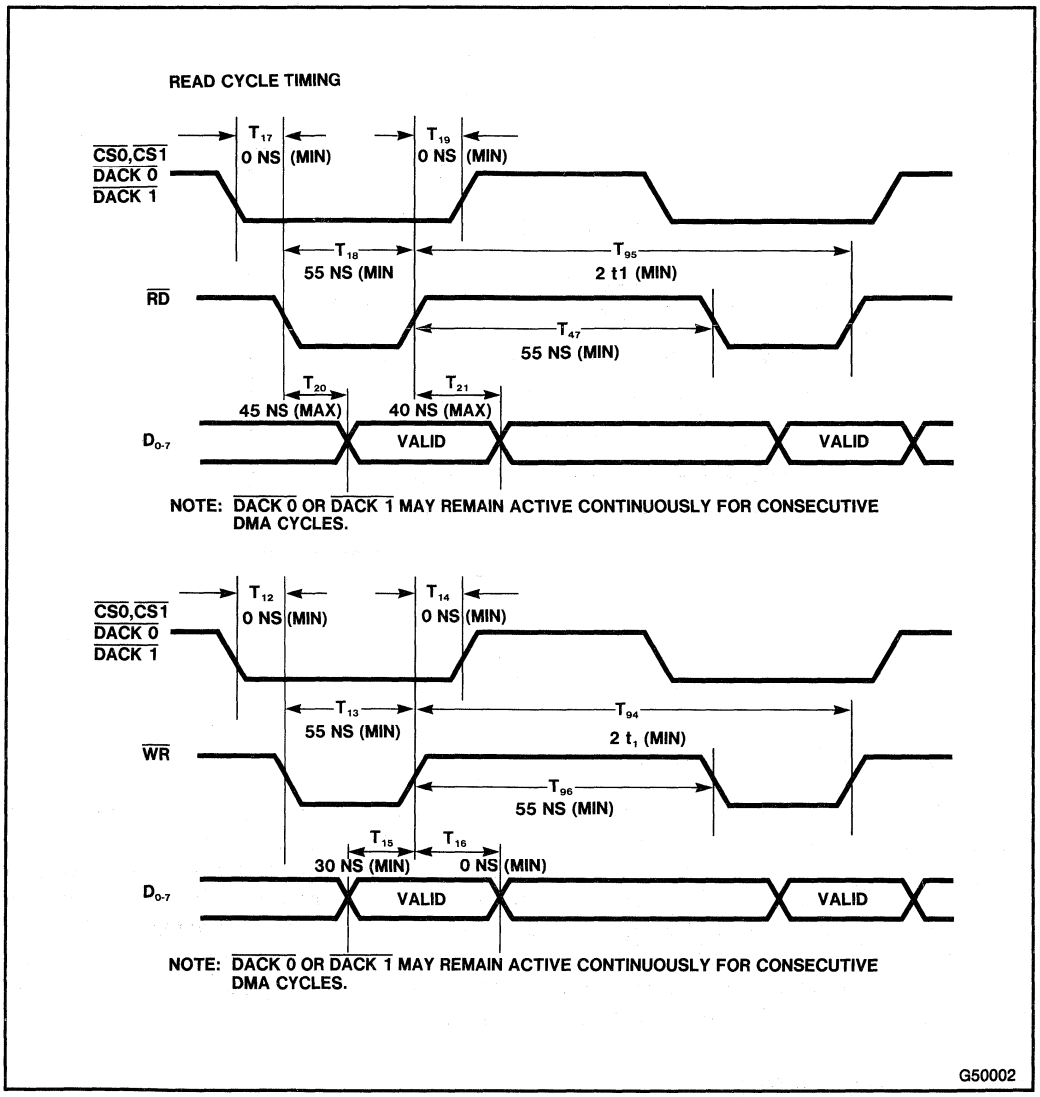


Figure 4-3. Command/Status Transfers

In order to request a transfer to memory the 82590 activates a DRQ signal to the specified channel of the DMA controller. In response, the DMA controller activates the corresponding $\overline{DACK}$ signal and starts the data transfer. The 82590 outputs data onto the bus during the time that the $\overline{RD}$ line is activated by the DMA controller.

When the 82590 is configured to assert the $\overline{EOP}$ signal to terminate DMA data transfers, the $\overline{EOP}$ signal is activated to the DMA controller as shown in Figure 4-4. The DMA controller samples the state of $\overline{EOP}$ using the rising edge of $\overline{RD}$ or $\overline{WR}$.

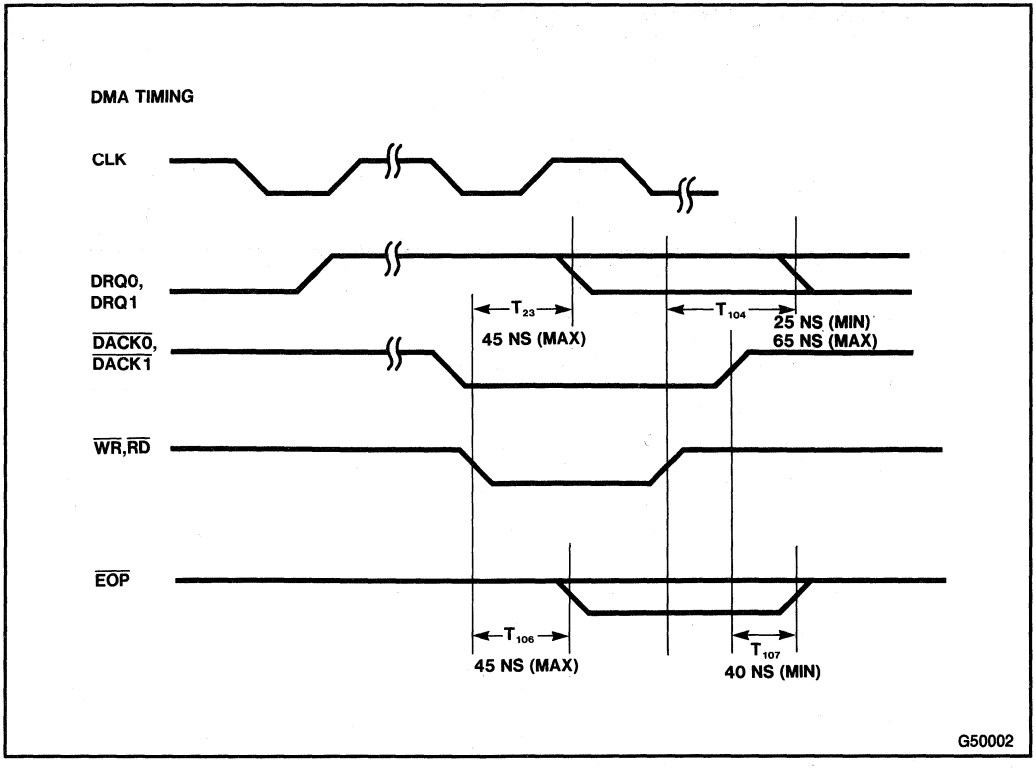


Figure 4-4. Data Transfer

In Tightly Coupled Interface (TCI) the 82590 uses the DRQ signal in addition to the $\overline{EOP}$ signal to inform the DMA controller of state of the serial side process (refer to Section 7.5).

4.3 INTERRUPT

The 82590 reports on completion of a Port 0 or Port 1 event by updating a set of status registers in the respective port and raising the Interrupt signal. The CPU acknowledges the interrupt by issuing an appropriate interrupt acknowledge command to the respective port.

Programming Model 5

CHAPTER 5
PROGRAMMING MODEL

Figure 5-1 shows the 82590 on-chip registers that are directly accessible through the 8-bit I/O ports: Port 0 and Port 1. Port 0 and Port 1 are accessed via the D_0-D_7 lines.

Figure 5-2 shows the Port 0 commands, and Figure 5-3 shows the Port 1 commands. The two separate I/O ports can be accessed at two different addresses selected by $\overline{CS0}$ and $\overline{CS1}$, or at one address selected by $\overline{CS0}$. When the hardware does not support two chip-select signals, port switch commands are used to access both ports alternately at one address. If SWT-TO-PORT-1 command is executed while in Port 0, the active port is logically switched to Port 1. Software overhead associated with port switching is eliminated if two chip select signals are supplied in hardware.

Some Port 0 commands require the associated parameters to be transferred from memory to the 82590. These commands (parametric commands) are IA-Setup, Configure, MC-Setup, Transmit, and Retransmit. After one of these commands is written to Port 0, the 82590 requests the DMA controller to move the parameters from memory to the 82590. Therefore, the CPU must initialize an appropriate parameter block in memory and program the DMA controller before issuing a command to the 82590. A parameter block consists of two bytes of byte count and actual parameters (see Figure 5-4). If there are *n* bytes of parameters in the parameter block, the Byte Count field must contain *n* in it. If the 82592 is used in 16-bit mode, parameter blocks must be set up as 16-bit-wide blocks as shown in Figures 5-5 and 5-6. When the Byte Swap feature is used, the high order bytes and the low order bytes are swapped within parameter blocks (see Figure 5-7).

The Dump command or a frame reception causes the 82590 to request the DMA controller to move data from the 82590 to memory. The Dump result or the received frame is transferred to the memory space to which the DMA controller was initialized.

The 82590 can be configured to have four or six bytes of status registers in Port 0 (see Figure 5-1). When configured to four bytes of status registers, the first three status registers (STATUS 0 through 2) contain the information about the last command executed or the last frame received. The last status register, STATUS 3, contains the state of the 82590. When the 82590 is configured to six bytes of status registers, the two additional bytes are used to report a more complete status of the most recently received frame.

Two Port 1 commands, Set-Count-Val (set counter value) and Set-Count-Con (set counter configuration), require two or three bytes of parameters to be transferred from the CPU to Port 1. The CPU transfers these parameters by performing two or three consecutive write cycles at Port 1. For Port 1 parameter transfers, the CPU directly performs these I/O write cycles instead of having the DMA controller transfer the parameters as in the case for Port 0.

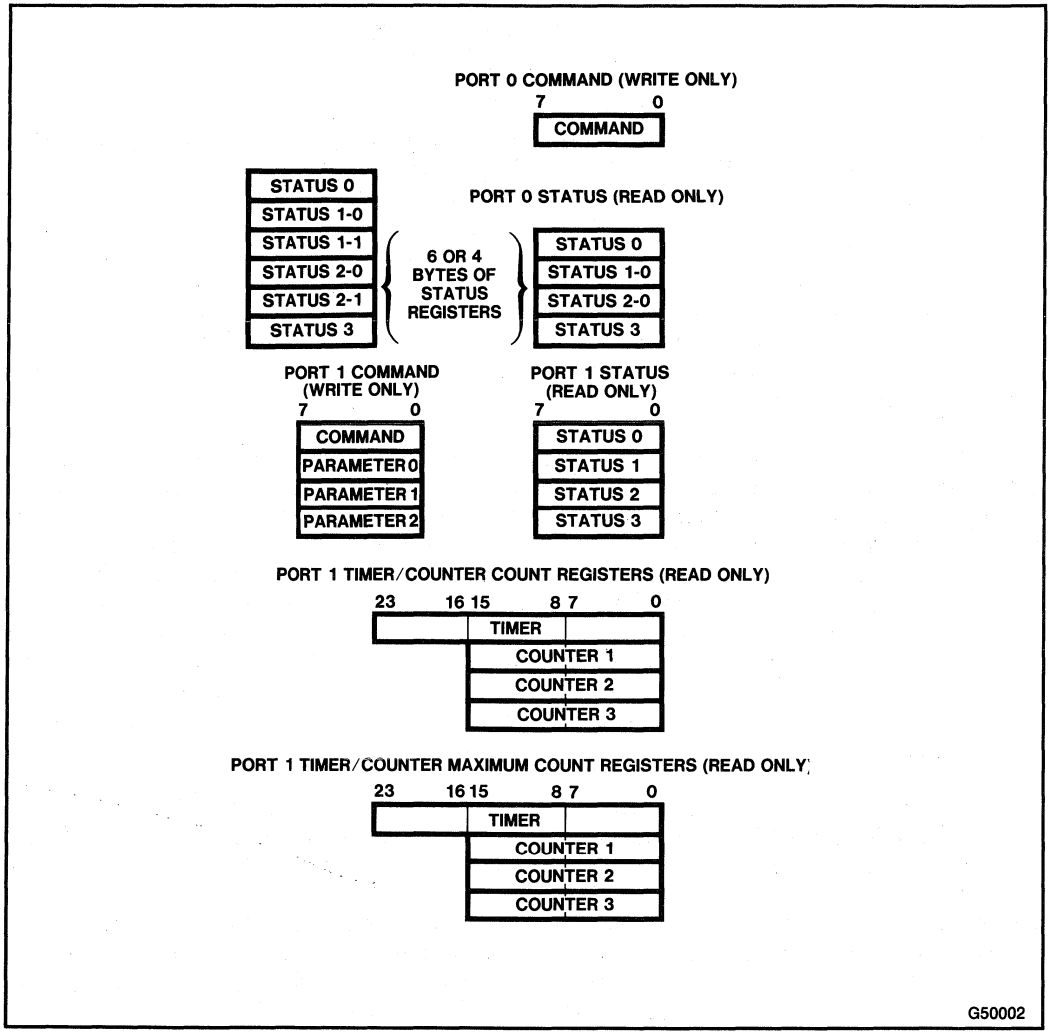


Figure 5-1. Programming Model—Directly Accessible Registers (Accessible Through 8-Bit I/O Port(s))

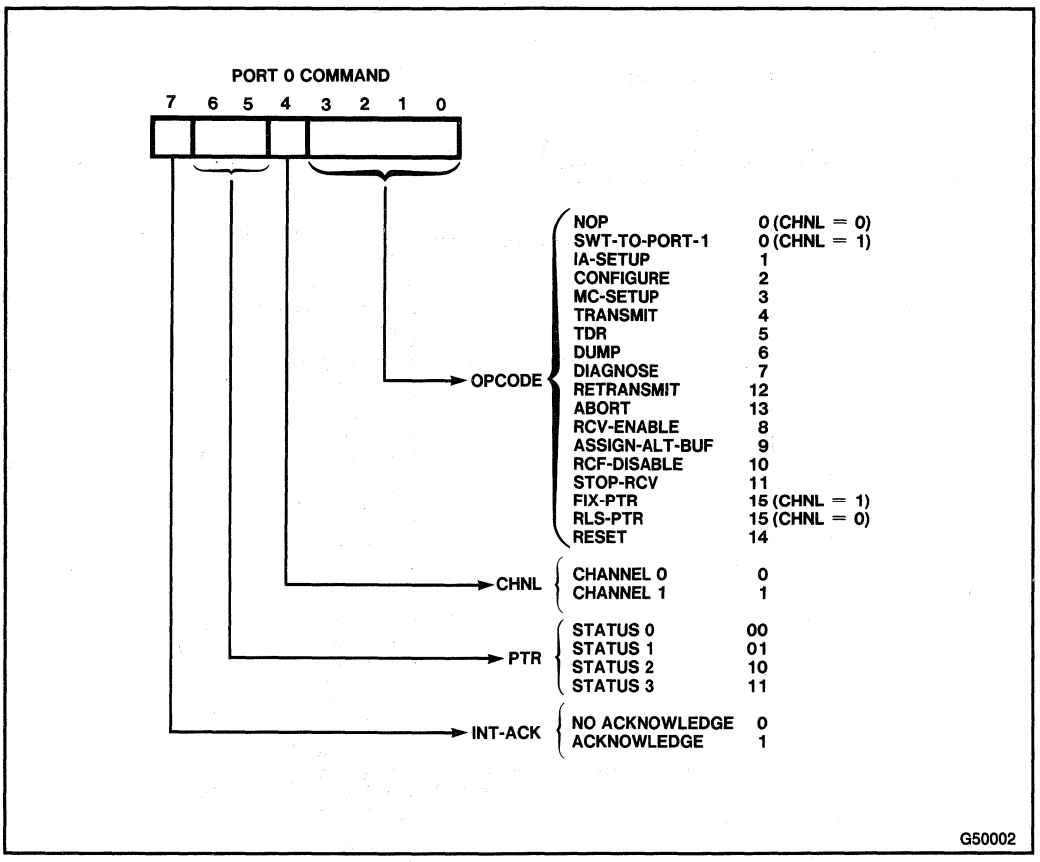


Figure 5-2. Port 0 Commands

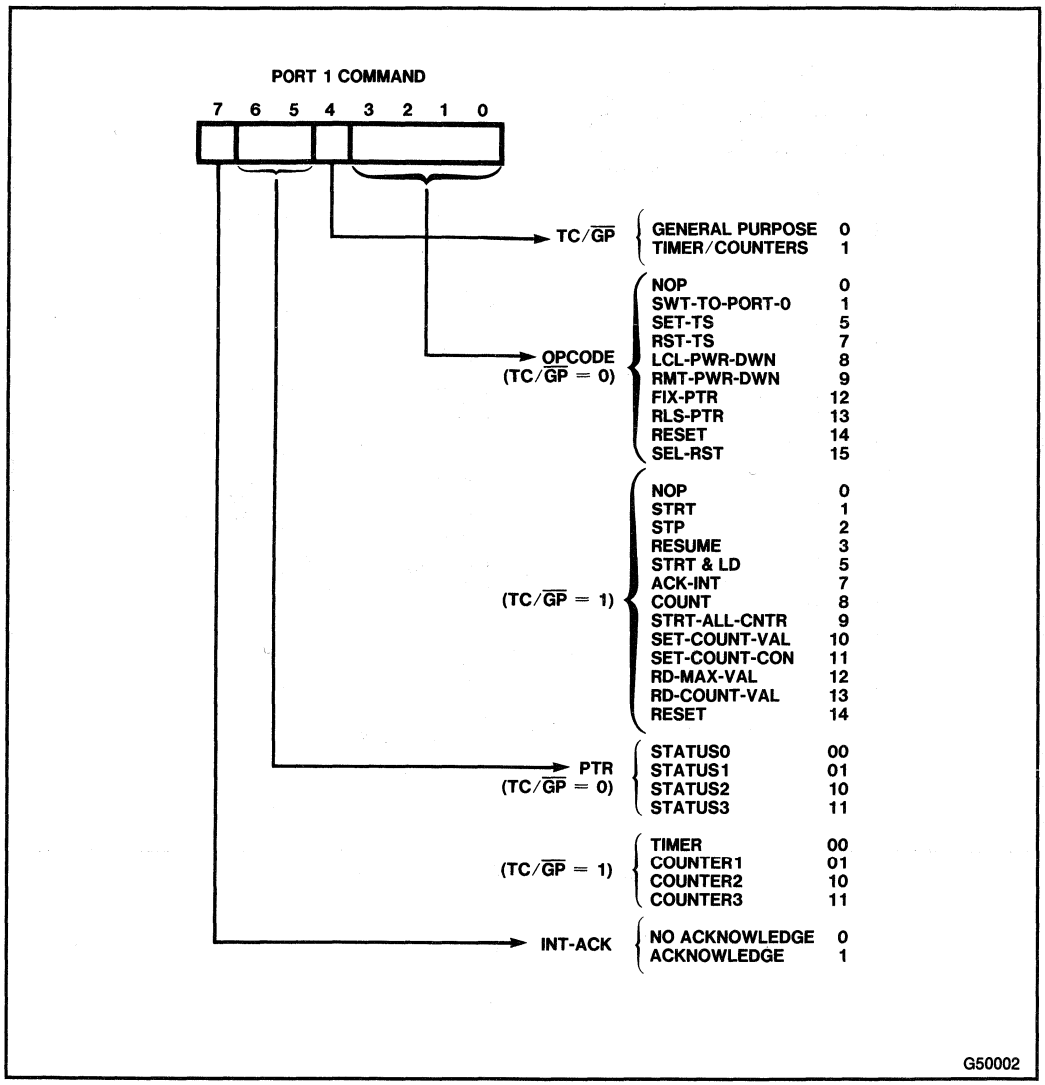


Figure 5-3. Port 1 Commands

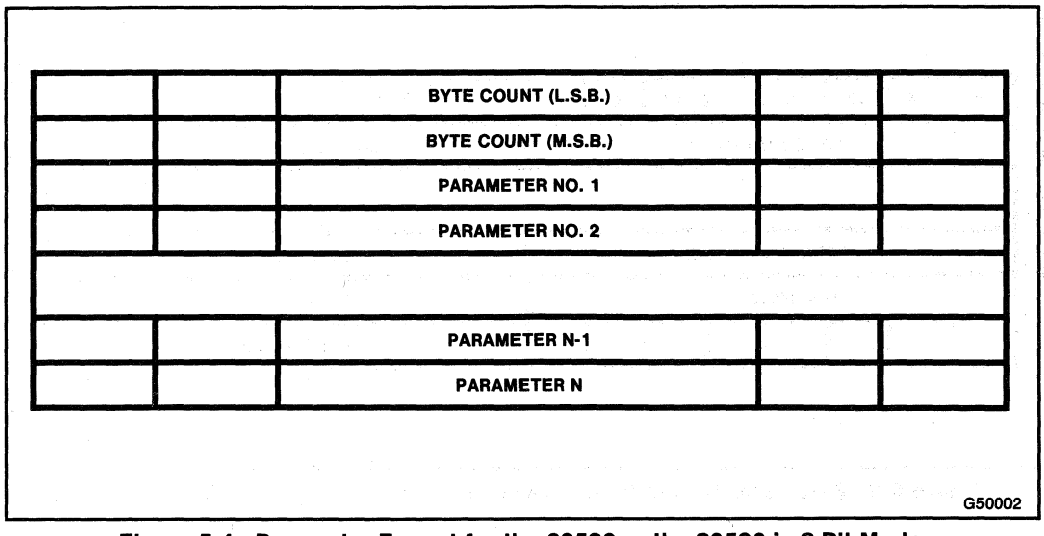


Figure 5-4. Parameter Format for the 82590 or the 82592 in 8-Bit Mode

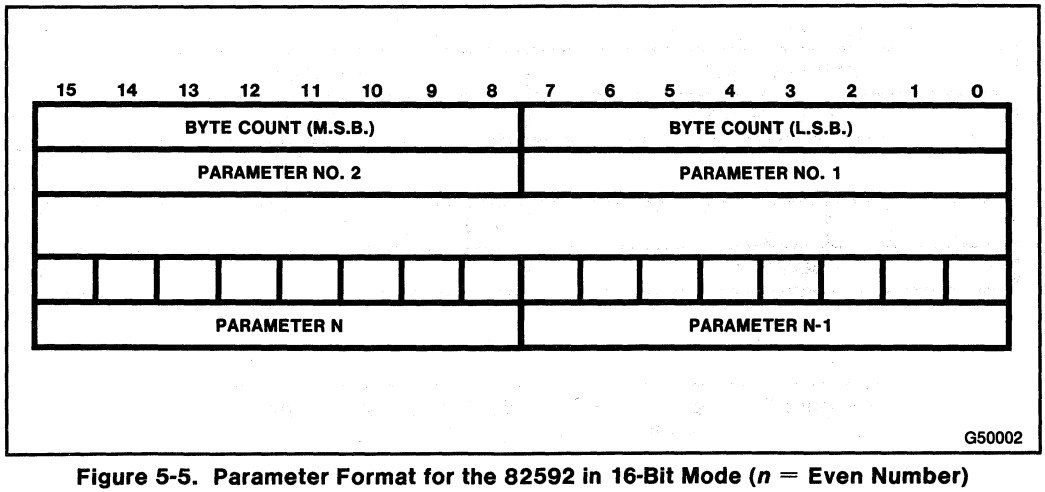


Figure 5-5. Parameter Format for the 82592 in 16-Bit Mode (*n* = Even Number)

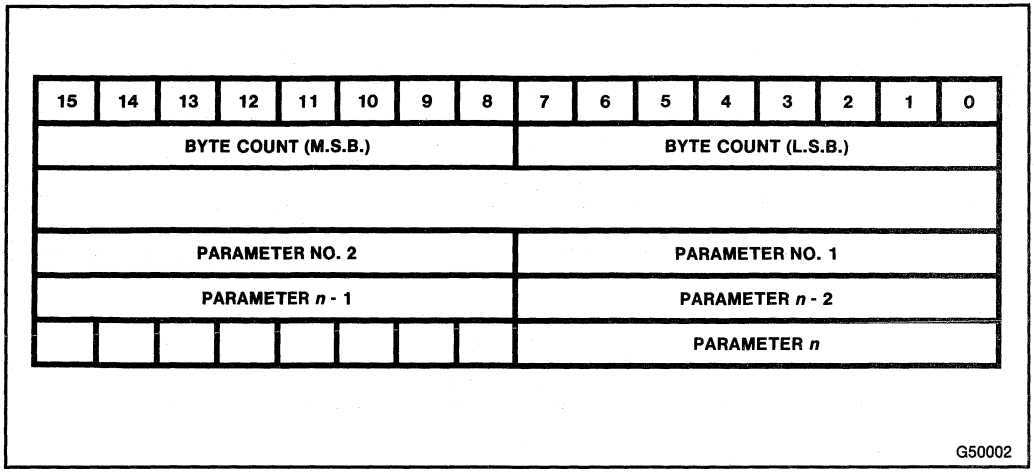


Figure 5-6. Parameter Format for the 82592 in 16-Bit Mode (*n* = an Odd Number)

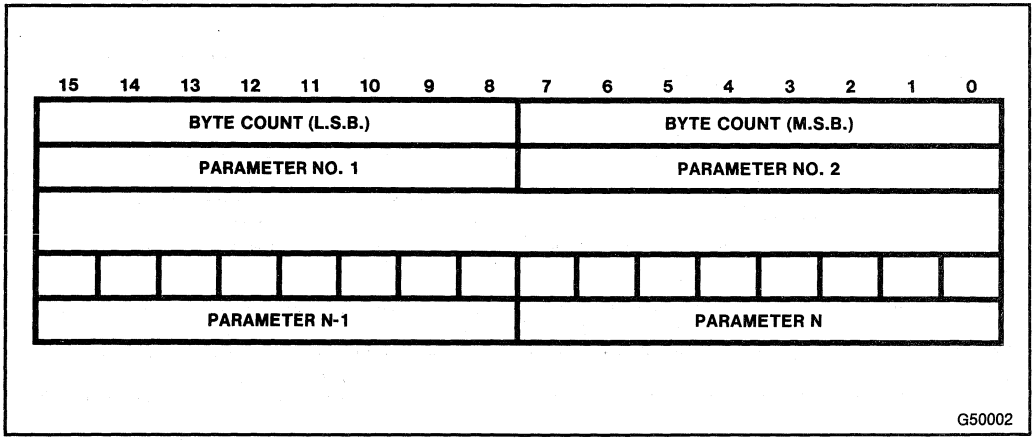


Figure 5-7. Parameter Format for the 82592 Configured to Byte Swap in 16-Bit Mode

Initialization / Configuration

6

CHAPTER 6
INITIALIZATION/CONFIGURATION

6.1 INITIALIZATION

A hardware or software reset brings the 82590 to its initial state where the 82590 is put into the idle state—with the transmitter, receive clock generator, interrupts, etc. inactive. In the case of the 82592, a reset places the device in 8-bit mode (see Figure 6-1). In order to use it in 16-bit mode, a CONFIGURE command must be executed with a byte count of 0 (see Figure 6-2 for the memory structure). The 82592 or the 82590 is then ready to be configured for a specific configuration.

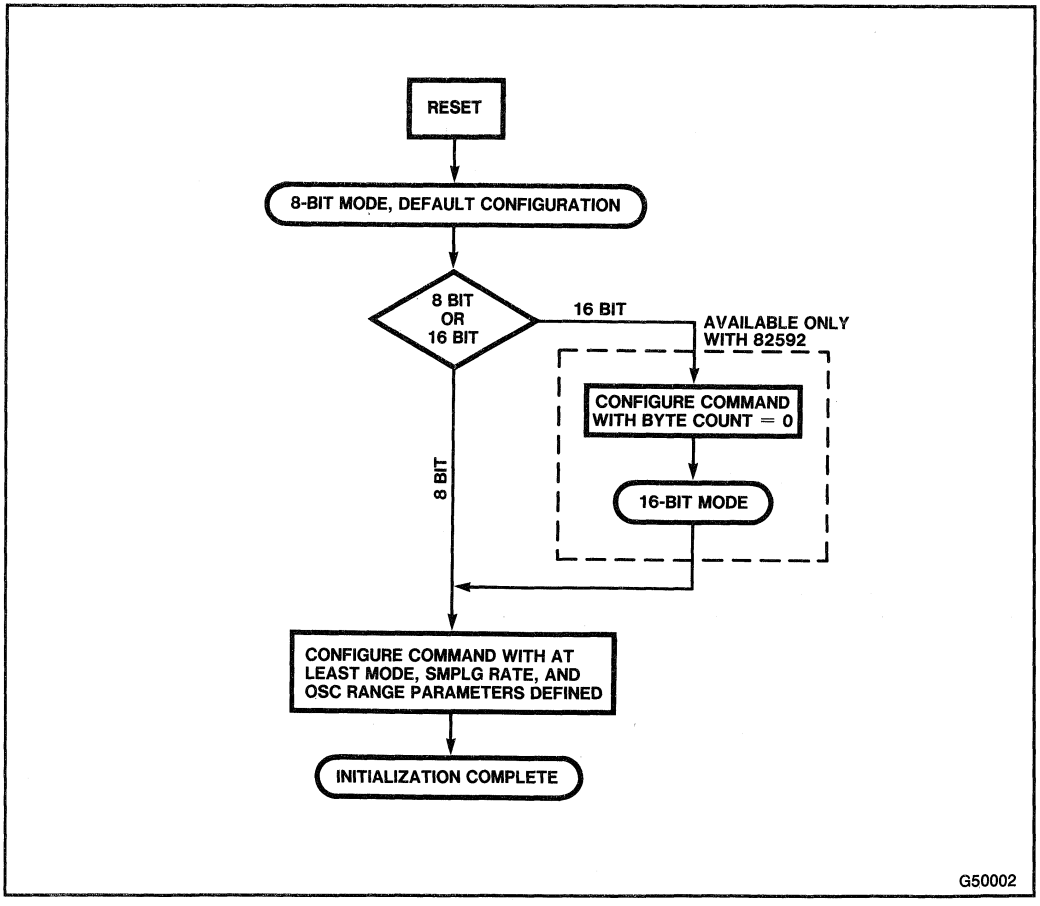


Figure 6-1. 82592 Bus Width Initialization Flowchart

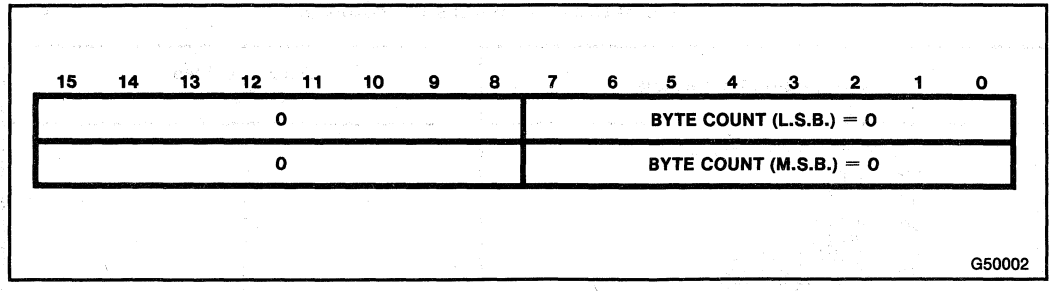


Figure 6-2. Memory Structure for the 82592 when Configured for 16-Bit Mode

6.2 CONFIGURATION

The default configuration parameters are shown in Table 6-1.

Table 6-1. Default Configuration

Configurable Parameters	Default Values on Reset
1. Serial Interface	*
2. Sampling Rate	*
3. Oscillator Range	*
4. FIFO Limit	*
5. Chaining	*
6. Buffer Length/TCI	0
7. Preamble Length	7
8. Address Length	6
9. No Source Address Insertion	0
10. Promiscuous Mode	0
11. Broadcast Disable	0
12. CRC-16/CRC-32	0
13. No CRC Insertion	0
14. Padding	0
15. Bitstuffing/EOC	0
16. Minimum Frame Size	64
17. Interframe Spacing	96
18. Slot Time	512
19. Number of Retries	15
20. Linear Priority	0
21. Backoff Method	0
22. Manchester/NRZ(1)	0
23. Diff. Manch/Manchester	0
24. CRS Filter	0
25. Internal CRS	0
26. CDT Filter	0
27. CDBBC	0
28. CDBCV (Internal CDT)	0
29. Loopback	00
30. Transmit on No CRS	0

*These parameters must be configured after RESET.

Table 6-1. Default Configuration (Cont'd.)

Configurable Parameters	Default Values on Reset
31. Exponential Priority	0
32. StarLAN	1
33. 8X/10X	1
34. Jabber	1
35. CDBSAC	1
36. CRC in Memory	1
37. Auto Retransmit	1
38. Length Field Detection	1
39. Preamble until CRS	1
40. DCR	0
41. DCR Slot Number	0
42. DCR Number of Stations	63
43. Full Duplex	0
44. Multi-IA	0
45. Disable Backoff	0
46. DPLL Integrator Length	7
47. MC All	0
48. Monitor	00
49. Counter Function	0
50. Disable Monitor Interrupt	0
51. FIFO Banking	00 (32-byte Transmit FIFO & 32-byte Receive FIFO)
52. Port 1 Enable	0
53. Byte Swap	0
54. CLK Divider	1
55. Status Length	0 (4 bytes)
56. Receive EOP	0
57. Transmit EOP	0

All but five parameters have a default value after reset. Figure 6-3 shows the configuration block in memory. The first two bytes contain the number of bytes (byte count) which follow. Note that all parameters not having a default value on reset are contained in the first byte after the byte count field. The maximum byte count is 15. Values greater than 15 are interpreted as modulo 16 (meaning only the 4 least significant bits are used). After reset a configure command with a minimum byte count of 1 is essential for operation of the 82590.

6.3 CONFIGURATION PARAMETERS

1. Serial Interface Mode—Byte 2, Bit 6

This bit selects between High-Integration and High-Speed modes. It also selects the function of the $\overline{TxC}$/X1, $\overline{RxC}$/X2, and $\overline{TCLK}/\overline{CRS}$ pins.

7	6	5	4	3	2	1	0	BYTE NO. (DECIMAL)
		BYTE COUNT (L.S.B.)						0
		BYTE COUNT (M.S.B.)						1
CHNG	SERIAL MODE	SMPLG RATE	OSC RANGE	FIFO LIMIT				2
BUFFER LENGTH/TCI								3
LOOPBACK		PREAMBLE LENGTH		NO SA INS	ADDRESS LENGTH			4
BOF METD	EXPONENTIAL PRIORITY			DIFMAN /MAN	LINEAR PRIORITY			5
INTERFRAME SPACING								6
SLOT TIME (LOW)								7
MAXIMUM RETRY NUMBER				CDBBC	SLOT TIME (HIGH)			8
PAD	BIT STUFF	CRC16/ CRC32	NO CRC INSER	Tx ON NO CRS	MAN/ NRZ(I)*	BC DIS	PRM MODE	9
CDTSRC	CDT FILTER			CRSSRC	CRS FILTER			10
MINIMUM FRAME LENGTH								11
STRLAN	8X/10X	JABBER	CDBSAC	AUTOTX	CRCINM	LNGFLD	PRECRS	12
DCR	FDX	DCR SLOT NUMBER						13
DISBOF	MULIA	DCR NUMBER OF STATIONS						14
MONINT	CNTFUN	MONITOR		MC ALL	DPLL INT LEN			15
RX EOP	TX EOP	STTLEN	CLKDIV	BSWAP	PORT 1	FIFO BANK		16

*NRZ IN HIGH-SPEED MODE; NRZI IN HIGH-INTEGRATION MODE

G50002

Figure 6-3. Configure Block

In High-Integration Mode the 82590 generates its transmit and receive clocks from the crystal connected between the X1 and X2 pin or the clock signal provided at the X1 pin; encodes/decodes the data waveform (either Manchester, Differential Manchester, or NRZI); and generates the Carrier Sense and (optionally) the Collision Detect from the incoming RXD signal. The maximum data rate supported in this mode is 4 Mb/s.

In High-Speed Mode the maximum data rate supported is 20 Mb/s. However, the clock generation, including receive data decoding and receive clock recovery, must be done externally. The Data Encoding, Carrier Sense, and Collision Detect are also done externally in most cases.

0—High Integration

1—High Speed

Default setting: none

2. Sampling Rate—Byte 2, Bit 5

Valid only in High-Integration Mode. It specifies the ratio between the frequency of the sampling (fast) clock and the data bit rate. This bit must be 0 in High-Speed Mode.

0—8X or 10X

1—16X or 18X

See for 8X/10X below for a complete truth table.

Default setting: none

3. Oscillator Range—Byte 2, Bit 4

Valid only in High-Integration Mode. It specifies the frequency range of the sampling (fast) clock. This bit must be 0 in High-Speed Mode.

0—high range (1 to 32 MHz)

1—low range (DC to 1 MHz)

Default setting: none

4. FIFO Limit—Byte 2, Bits 3-0

Specifies the byte count in the 16-byte FIFOs at which the 82590 is to resume requests for transfer of data to/from memory. This parameter is used for the Transmit FIFO. The receive FIFO limit is 16 minus this parameter.

When the FIFO length is configured for 32 bytes this parameter is internally multiplied by two to generate the actual FIFO limit. When the FIFO length is configured for 48 bytes the parameter is multiplied by three (see FIFO Banking below).

Default setting: none

5. Chaining—Byte 2, Bit 7

Together with the Buffer Length, Transmit EOP, and Receive EOP bits, determines the mode in which the 82590 receives frames. The 82590 supports Single Buffer, Buffer Switching, Frame Switching, and Tightly Coupled Interface modes. The details of these receive modes are found in Sections 2.3 and 7.4. Table 6-2 summarizes the configurations of these bits (also see Buffer Length, Transmit EOP, and Receive EOP below).

Table 6-2. Receive Mode and EOP Signaling

Chaining	RX EOP (bit 7–byte 15)	TX EOP (bit 6–byte 15)	Receive Mode and EOP Signaling
0	0	0	Single Buffer Reception (82588 compatible) $\overline{EOP}$ signal never asserted
0	0	1	Single Buffer Reception $\overline{EOP}$ signal asserted upon detecting a collision
0	1	0	Frame Switching Reception $\overline{EOP}$ signal asserted at the end of reception (only one DMA channel used)
0	1	1	Frame Switching Reception $\overline{EOP}$ signal asserted upon detecting a collision or at the end of reception Tightly Coupled Interface (If Buffer Length/TCI = C0H or 80H)
1	0	0	Buffer Switching Reception (82588 compatible) $\overline{EOP}$ signal never asserted
1	0	1	Frame Switching Reception $\overline{EOP}$ signal asserted upon detecting a collision
1	1	0	Frame Switching Reception $\overline{EOP}$ signal asserted at the end of bad reception
1	1	1	Frame Switching Reception $\overline{EOP}$ signal asserted upon detecting a collision or at the end of bad reception

If the chaining bit is not set, and if both the Transmit and Receive EOP bits are also not set, each frame is received into one continuous buffer using one DMA channel. This is the Single Buffer mode compatible with the 82588. If the chaining bit is not set but the Receive EOP bit is set, the 82590 performs Frame Switching with one DMA channel, using the EOP signal. However, if both of the Transmit and Receive EOP bits are set while the Chaining bit is not set, the 82590 can be placed in the Tightly Coupled Interface mode by configuring the Buffer Length parameter to 80H or C0H (see Parameter 6 below).

If the chaining bit is set and if both the Transmit and Receive EOP bits are not set, the 82590 switches the DMA channel during reception whenever the number of bytes or words transferred to memory equals the Buffer Length parameter. This configuration provides the Buffer Switching mode compatible with the 82588.

If the chaining bit is set, and if at least one of the Transmit or Receive EOP bits is set, the 82590 switches the DMA channel between frame receptions implementing Frame Switching mode.

Default setting: none

6. Buffer Length/TCI—Byte 3, Bits 7-0

When the 82590 is configured for Buffer Switching reception, this parameter specifies the length of the buffer, after which the DMA channel is switched. It is specified in units of 4 bytes for the 82590 and 8 bytes for the 82592. A buffer length of 0 is interpreted as 1024 bytes in the case of the 82590 and 2048 bytes in the case of the 82592.

If the Chaining bit is not set, and if both the Transmit and Recieve EOP bits are set, 80H or C0H specified in the Buffer Length field places the 82590 in Tightly Coupled Interface (TCI) mode. When the TCI is enabled by 80H, End-of-Frame interrupts are reported to the CPU as frames are received by the 82590. When the TCI is enabled by C0H, the End-of-Frame interrupts are disabled. Refer to Section 7.5.1 for more details.

When the 82590 is configured for Frame Switching reception, this field must contain 0.

Default setting: 0

7. Preamble Length—Byte 4, Bits 5–4

Selects the length of the Preamble not including the SFD.

00–1 byte

01–3 bytes

10–7 bytes

11–15 bytes

Default setting: 10–7 bytes

8. Address Length—Byte 4, Bits 0–2

Determines the length, in bytes, of the address that the 82590 refers to. This applies to source, destination, Multicast, and broadcast addresses. An address length of 7 is treated as 0.

Default setting: 6 bytes

9. No Source Address Insertion—Byte 4, Bit 3

0—Source address is inserted by the 82590 during transmission

1—Source address insertion is disabled (source address is provided by the user in the transmit block

Default setting: 0

10. Promiscuous Mode—Byte 9, Bit 0

When set, causes the 82590 to receive all frames longer than 6 bytes, regardless of their destination address.

0—Promiscuous mode off

1—Promiscuous mode on

Default setting: 0

11. Broadcast Disable—Byte 9, Bit 1

Disables reception of frames with a Broadcast destination address or Multicast address of all 1's. The Promiscuous Mode setting overrides the Broadcast Disable.

0—Broadcast enabled

1—Broadcast disabled

Default setting: 0

12. CRC-16/$\overline{\text{CRC-32}}$—Byte 9, Bit 5

Specifies which CRC polynomial is used for CRC generation and checking.

0—32-bit Autodin - II CRC

1—16-bit CCITT CRC

The 32-bit Autodin-II polynomial is $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$.

The 16-bit CCITT polynomial is $X^{16} + X^{12} + X^5 + 1$

Default setting: 0

13. No CRC Insertion—Byte 9, Bit 4

Specifies whether or not the CRC is inserted after the information field during transmission. This capability allows CRC generation external to the 82590, enabling the user to verify the CRC checking mechanism during reception. It is also useful in a simple store-and-forward bridge where a received CRC must be retransmitted without recalculation of CRC (see CRC in Memory configuration below).

Note that the 82590 always checks for a valid CRC during reception (i.e., CRC check cannot be disabled).

0—CRC is calculated and inserted by the 82590 after the LLC Data/Padding field

1—No CRC insertion (CRC is provided by the user)

Default setting: 0

14. Padding—Byte 9, Bit 7

Determines the padding function together with Bitstuffing/EOC, CDBBC, and Length Field configuration bits as shown in Table 6-3.

Table 6-3. Padding

BITSTF	CDBBC	Pad	Length Field	Padding
0	X	X	0	No Padding
1	X	0	X	No Padding
0	0	X	1	Padding in EOC Framing Up to Slot Time
0	1	0	1	Padding in EOC Framing Up to Slot Time
0	1	1	1	Padding in EOC Framing Until Receive Signature is Calculated for CDBBC
1	0	1	X	Padding in HDLC Framing Up to Slot Time
1	1	1	0	Padding in HDLC Framing Up to Slot Time
1	1	1	1	Padding in EOC Framing Until Receive Signature is Calculated for CDBBC

Padding is done for frames shorter than the slot time. Frame length is measured from the beginning of the Destination Address to the end of the FCS and then compared with the slot time. The short frames are padded with pad bytes (the HDLC flags) until they become longer than the slot time or until the receive signature for the CDBBC is calculated (see Section 2.4 for details of the CDBBC). The latter case occurs only when CDBBC, Padding, and Length Field are all enabled.

The 82590 supports two different schemes for padding. One is the padding scheme used in the HDLC framing and the other one is the padding scheme used in the End of Carrier framing. The Bitstuffing/EOC configuration bit selects one of these two padding schemes. The Padding bit enables padding in the HDLC framing and the Length Field bit enables padding in the EOC framing. In order for Padding or Length Field to enable its respective padding scheme, bitstuffing/EOC must be configured appropriately. Except for one case where both Padding and Length Field are set simultaneously along with CDBBC.

When the Padding bit is set in the HDLC bitstuffing mode, the frames shorter than the slot time are padded with HDLC flags to a byte boundary longer than the slot time (up to a few bits more than the slot time). However, if both CDBBC and Length Field are also set, those short frames are padded until the receive signature for the CDBBC is calculated. In both cases, the pad bytes are added after the FCS field and not taken into account for CRC calculation. Since the first HDLC flag following the FCS field serves as the end of frame indication, the pad bytes are effectively removed by the 82590 during reception.

See Length Field below for descriptions of padding in the End of Carrier framing.

Default setting: 0

15. Bitstuffing/$\overline{\text{EOC}}$—Byte 9, Bit 6

Specifies the frame delineation method.

0—End of Carrier Framing (IEEE 802.3 compatible)

1—Bitstuffing Framing (HDLC, with 01111110 flag and zero bit insertion/deletion)

In the End of Carrier Framing mode, the 82590 checks for the entire SFD bit sequence (10101011) to detect the beginning of a frame. The end of a frame is indicated by Carrier becoming inactive.

In the bitstuffing mode, the 82590 checks for the entire HDLC flag pattern before accepting it as a valid flag. Only one flag is allowed following the preamble bits. Two consecutive flags will be interpreted as a beginning and a closing flag to the 82590, i.e., a frame with no data.

The Bitstuffing framing offers high reliability especially for networks susceptible to line ringing.

Default setting: 0

16. Minimum Frame Size—Byte 11, Bits 0–7

Specifies the minimum receive frame size, not including preamble, in bytes. The 82590 sets a status bit if a shorter frame is received. Note that frames shorter than 6 bytes are discarded with no status update.

Default setting: 64 bytes

17. Interframe Spacing—Byte 6, Bits 7–0

Specifies the time period, in bit times, that the 82590 must defer after detecting the later of two events: (a) The last bit has been transmitted, or (b) Carrier Sense became inactive. The minimum value is 12, and any lower value will default to 12. The 82590 implements a retriggerable IFS timer with its retriggering window of one-half IFS, according to the IEEE 802.3 standard (if the IFS parameter is less than 24, then the retriggering window is equal to IFS). The IFS timer is retriggerable only when it is started by Carrier Sense becoming inactive.

Default setting: 96

18. Slot Time—Byte 7, Bits 0–7; Byte 8, Bits 0–2

The slot time for the network, in bit times. This value is used in calculating backoff, Linear Priority, Padding Length, Jabber Timeout, and DCR delays. Zero is interpreted as 2^{11} (2048).

For IEEE 802.3 networks, the slot time should be set at 512. However, it may be programmed to any number between 1 and 211. The user may change it to optimize the network to specific application environments.

Default setting: 512

19. Number of Retries—Byte 8, Bits 7–4

The maximum number of retries, due to collisions, that the 82590 performs.

Default setting: 15

20. Linear Priority—Byte 5, Bit 0–2

The number of slot times that the 82590 waits after Interframe Spacing or after Backoff, before enabling transmission. A higher number reduces the priority. Stations with this parameter equal to 0, the highest priority, conform to the IEEE 802.3 backoff algorithm.

Default setting: 0

21. Backoff Method—Byte 5, Bit 7

Determines the method for starting the backoff timer.

0—According to the IEEE 802.3 standard; immediately after the end of jam transmission

1—After the interframe spacing period expires

Default setting: 0

22. Manchester/$\overline{\text{NRZ(I)}}$—Byte 9, Bit 2

Specifies the data encoding/decoding method. It depends on the Serial Interface configuration bit.

Default setting: 0

Table 6-4. Manchester/NRZ(I) Configuration Bit

Manchester/NRZ(I)	High Integration	High Speed
0	NRZI (*)	NRZ
1	Manchester	Manchester (**)

* Bitstuffing should also be applied
** Manchester encoding is performed on transmitted data (TxD). NRZ data must, however, be provided for reception (External Manchester decoding)

23. Differential Manchester/$\overline{\text{Manchester}}$—Byte 5, Bit 3

Chooses Manchester or Differential Manchester when the Manchester/NRZ(I) bit is set for Manchester. This bit must be 0 in NRZ or NRZI mode.

0—Manchester

1—Differential Manchester

Default setting: 0

24. CRS Filter—Byte 10, Bits 2–0

Specifies the required width of the Carrier Sense signal, in bit times, before it is recognized as being active. Carrier sense deactivation is recognized immediately, i.e., only negative-going glitches are filtered. This capability is useful in noisy cable environments.

Default setting: 0

25. Internal CRS—Byte 10, Bit 3

Valid only in High-Speed Mode. Specifies whether Carrier Sense is to be performed internally or externally. External Carrier Sense is fed through the $\overline{CRS}$ pin. If set, presense of the receive clock is interpreted as Carrier Sense active. Absence of the receive clock means Carrier Sense inactive. This bit must be zero in High Integration mode.

0—External

1—Internal

Default setting: 0

26. CDT Filter—Byte 10, Bits 4–6

Specifies the required width of CDT, in bit times, before it is recognized that a collision has occurred. Only negative going glitches are filtered.

Default setting: 0

27. CDBCV (Internal CDT)—Byte 10, Bit 7

Determines, in High-Integration mode, whether Collision Detection By Code Violation (CDBCV) is performed in addition to external collision detection via the $\overline{CDT}$ pin. In High Integration mode, collisions are sensed by the $\overline{CDT}$ pin regardless of this configuration bit. In High Speed mode, this bit specifies whether collision detect is to be performed internally or externally via the $\overline{CDT}$ pin. Internal collision detect in High-Speed mode interprets presence of Carrier Sense during transmission as a collision.

Default setting: 0

Table 6-5. CDBCV Configuration Bit

CDBCV	High Integration	High Speed
0	CDBCV off	External Collision Detection
1	CDBCV on	Internal Collision Detection

28. CDBBC—Byte 8, Bit 3

Available both in High-Integration Mode and in High-Speed Mode. Specifies whether Collision Detect by Bit Comparison is to be performed.

0—CDBBC off

1—CDBBC on

Default setting: 0

29. Loopback—Byte 4, Bits 7-6

Configures loopback operation of the 82590.

00—Loopback off
01—Internal Loopback
10—External Loopback. The LPBK pin not activated
11—External Loopback. The LPBK pin activated.

In Internal Loopback mode, the 82590 disconnects itself from the serial link and logically connects TxD to RxD and $\overline{\text{TxC}}$ to $\overline{\text{RxC}}$. The $\overline{\text{TxC}}$ frequency is internally divided by four during internal loopback operation.

In External Loopback mode, the 82590 transmits and receives simultaneously, at full rate. This allows checking of external hardware as well as the serial link to the transceiver interface. The LPBK pin may be used to inform the external hardware of establishing transmit-to-receive interconnection.

Default setting: 00

30. Transmit on No CRS—Byte 9, Bit 3

Determines whether transmission, even if no Carrier Sense is returned from the transceiver, is allowed. This option is used with transceivers that do not return carrier during transmission.

0—Stop transmission if Carrier Sense is not returned. Carrier Sense must be returned active by the time the SFD transmission is completed. In case Carrier Sense is not returned, the transmission is terminated following jam transmission.

1—Transmit on no Carrier Sense. Transmission is continued regardless of the state of Carrier Sense.

Default setting: 0

31. Exponential Priority—Byte 5, Bits 6–5

Extends the range from which the random number for backoff is selected. The Backoff Time is calculated as follows:

Backoff Time = RX Slot Time, where $0 \leq R < 2\,[\min(10, N+K)]$

R is a random integer in the specified range, K is the exponential priority parameter, and N is the number of retries. For IEEE 802.3, K must be 0.

Default setting: 0

32. $\overline{\text{StarLAN}}$—Byte 12, Bit 7

Valid only in High-Integration mode. Enables the 82590 to recognize the IEEE 802.3 1BASE5 (StarLAN) Collision Presense Signal (see Figure 2-10) and also to change the $\overline{\text{RTS}}$ timing such that the signal becomes inactive two bit times after the last TxD bit transition. This bit must be 1 in High-Speed mode.

0—Enable

1—Disable

Default setting: 1

33. 8X/$\overline{\text{10X}}$—Byte 12, Bit 6

Valid only when Manchester encoding is selected in High-Integration Mode. It specifies the ratio between the frequency of the serial sampling (fast) clock and the data bit rate. This bit must be one in High-Speed mode.

0—10X or 18X, according to Sampling Rate and Manchester/NRZ(I) configuration bits. The 82590 can use 10X or 18X only when configured for Manchester or Differential Manchester. The Manchester/NRZ(I) bit override this bit to 1.

1—8X or 16X, according to Sampling Rate configuration bit.

Default setting: 1

Table 6-6. Sampling Rate

Manchester/$\overline{\text{NRZ(I)}}$	Sampling Rate	8X/$\overline{\text{10X}}$	Actual Sampling Rate
0	0	x	8X
0	1	x	16X
1	0	0	10X
1	0	1	8X
1	1	0	18X
1	1	1	16X

34. Jabber—Byte 12, Bit 5

Enables the 82590 to detect abnormally long transmission and reception. The timeout period set for the internal Jabber timer is 24 times the Slot time. If transmission is longer than the timeout value, the 82590 terminates the frame transmission and then appends the jam sequence. If reception does not complete within the timeout value, it is aborted as soon as the timer expires.

0—Enable

1—Disable

Default setting: 1

35. $\overline{\text{CDBSAC}}$—Byte 12, Bit 4

Collision Detection By Source Address Comparison (CDBSAC) compares the source address field of the transmitted frame to the source address of the received frame. If the source addresses are different, it assumes that a collision has occurred. The configuration bit, No Source Address Insertion, must be 0 to use this feature.

0—Enable

1—Disable

Default setting: 1

36. $\overline{\text{CRC in Memory}}$—Byte 12, Bit 2

Enables the 82590 to transfer the CRC fields to memory during reception (see Section 9.3.8 for memory structure).

0—Enable

1—Disable

Default setting: 1

37. $\overline{\text{Auto Retransmit}}$—Byte 12, Bit 3

If the 82590 detects a collision during transmission of the Preamble sequence, it automatically retransmits the frame without CPU or DMA intervention.

0—Enable

1—Disable

Default setting: 1

38. $\overline{\text{Length Field}}$—Byte 12, Bit 1

Enables padding in the End of Carrier framing (see Table 6-3).

The frames shorter than the slot time are extended with pad bytes to the nearest byte boundary greater than or equal to the slot time. The pad bytes are inserted between the LLC data field and the FCS field, and are taken into account for CRC calculation.

If the data field of a received frame contains more bytes than the byte count indicated in the Length field, those extra bytes (i.e., pad bytes) are stripped automatically.

The 82590 configured for this feature is capable of detecting the following erroneous conditions:

1. The frame is too short to include the Length field.
2. The LLC data field is shorter than the byte count indicated in the Length field.
3. The LLC data field is longer than the byte count indicated in the Length field and the frame is longer than the minimum frame size. This is an erroneous condition only when the 82590 is not configured for the CDBBC.
4. The LLC data field is longer than the byte count indicated in the Length field, and the frame is shorter than the minimum frame size.

0—Enable

1—Disable

Default setting: 1

39. $\overline{\text{Preamble until CRS}}$—Byte 12, Bit 0

Enables the 82590 to transmit the Preamble sequence until Carrier is returned in the receive channel. After the Carrier is detected, the 82590 will transmit the configured number of preamble bits.

0—Enable

1—Disable

Default setting: 1

40. DCR—Byte 13, Bit 7

Enables the 82590 to operate in Deterministic Collision Resolution (DCR) mode.

0—Disable

1—Enable

Default setting: 0

41. DCR Slot Number—Byte 13, Bits 5–0

Valid only when DCR bit is set; otherwise, it must be zero. Specifies the station's slot number. Zero prevents the station from transmitting during the collision resolution period. The DCR Slot Number parameter must be smaller than or equal to the DCR Number of Stations parameter (Parameter 42).

Default setting: 0

42. DCR Number of Stations—Byte 14, Bits 5–0

Valid only when DCR bit is set; otherwise, must be all ones. Specifies the number of stations operating in DCR mode.

Default setting: 63

43. Full Duplex—Byte 13, Bit 6

Enables the 82590 to operate in Full Duplex mode. The 82590 does not use the CSMA/CD protocol and can start transmitting a frame regardless of its receiver state. It can also receive a frame while it is transmitting. The IFS timer (deferring algorithm) is never started after reception or transmission. No collisions are allowed in this mode of operation.

0—Disable

1—Enable

Default setting: 0

44. Multi-IA—Byte 14, Bit 6

Enables the 82590 to receive multiple, individual address (IA) frames using the hashing mechanism also used for multicast address filtering. The IA frames must have a destination address with zero in its least significant bit position.

0—Disable

1—Enable

Default setting: 0

45. Disable Backoff—Byte 14, Bit 7

Disables the backoff algorithm implemented in the 82590. Retransmission can take place as soon as the wait period calculated by the deferring algorithm is expired. The Diagnose command must not be executed when this feature is enabled.

0—Backoff Enabled

1—Backoff Disabled

Default setting: 0

46. DPLL Integrator Length—Byte 15, Bits 2–0

Valid only in High-Integration mode and in NRZI encoding; otherwise, it must be 7. This parameter specifies the integrator length of the Digital Phase-Lock Loop (DPLL). Zero provides a 82588 compatible DPLL.

Default setting: 7

47. MC All—Byte 15, Bit 3

Enables the 82590 to receive all frames that have a multicast address in the destination address fields. The multicast (MC) address must have one in its least significant bit position.

0—Disable

1—Enable

Default setting: 0

48. Monitor—Byte 15, Bits 5–4

These two bits configure the receive and monitor functions of the 82590. The receive function, by definition, enables the 82590 to receive the frames which pass the address filtering (IA address match, MC address match, etc.). Those frames must have the SFD field and must be longer than the absolute minimum frame length of 5 bytes (6 bytes in case of Multicast Address filtering). Contents and status of the received frames are transferred to memory and interrupts are generated. The monitor function, on the other hand, enables the 82590 to simply evaluate the incoming frames (or active carriers). The 82590 can monitor the frames which pass or do not pass the address filtering. It can also monitor frames (active carriers) which do not have the SFD fields. Contents of the monitored frames are never transferred to memory. The 82590 may be configured to increment appropriate event counters or to transfer only status bytes of the monitored frames to memory. Whenever the status bytes are transferred to memory, interrupts are generated.

00—The monitor function is completely disabled. The 82590 receives only those frames which are longer than its absolute minimum frame size, and pass its address filtering.

01—The monitor function monitors those frames which are not received by the 82590. The 82590 receives only those frames which are longer than the absolute minimum frame size, and pass the address filtering. Therefore, the monitor function enables the 82590 to monitor the rest of frames which are not received, depending on the Counter Function bit.

10—The receive function is completely disabled. If the Counter Function bit is zero, the 82590 monitors only those frames which are longer than the absolute minimum frame size and pass the address filtering. If the Counter Function bit is one, the 82590 monitors all frames, including the frames (active carriers) which do not have the SFD.

11—The receive function is completely disabled. The Counter Function bit must be 1 to use this configuration. The 82590 monitors all frames, including the frames (active carriers) which do not have the SFD.

Default setting: 00

49. Counter Function—Byte 15, Bit 6

Configures the functions of the three internal event counters. These event counters are incremented exclusively in both of the following configurations, i.e, only one of the three counters is incremented at a time.

0—Counter 1 counts good frames, Counter 2 counts CRC errors, and Counter 3 counts Alignment errors.

1—Counter 1 counts good frames, Counter 2 counts errored frames resulted from collisions, and Counter 3 counts errored frames which are not counted by Counter 2. There are normally three conditions for which Counter 2 is incremented: (1) an active Collision Detect indicated by the 82590's internal or external collision detect mechnism(s), (2) a frame received or monitored is shorter than the Minimum Frame Size, and (3) a frame (active carrier) monitored, regardless of its length, did not have a Start Frame Delimiter (SFD). However, if the Monitor bits (Byte 14, Bits 5-4) are configured to 10 (see Monitor above), Counter 2 is incremented only by the condition (1).

Default setting: 0

50. Disable Monitor Interrupt—Byte 15, Bit 7

Valid only when the monitor function is enabled—at least one of the Monitor bits (Byte 14, Bits 5–4) is set. When this bit is set, the 82590 does not generate an interrupt as a result of monitoring as long as it can maintain the statistics of monitoring in the event counters. If a monitored frame is not the kind of frames which can be counted by the event counters, the 82590 transfers status bytes (and byte count in TCI mode) of the frame to memory and generates an interrupt.

When this bit is not set, the 82590 generates an interrupt as a result of monitoring. Status bytes (and byte count) of each monitored frame are transferred to memory and an interrupt is generated.

0—Monitor Interrupts are generated.

1—Monitor Interrupts are not generated.

Default setting: 0

Table 6-7. FIFO Banking

FIFO Banking	Transmit FIFO Length	Receive FIFO Length
11	16 bytes	16 bytes
01	16 bytes	48 bytes
10	48 bytes	16 bytes
00	32 bytes	32 bytes

Table 6-8. Port 1 Enable

Port 1 Enable	$\overline{DACK}/\overline{CS1}/\overline{EOP}$	$\overline{DACK0}/\overline{DACK1}$
0	$\overline{DACK1}$	$\overline{DACK0}$
1	$\overline{CS1}/\overline{EOP}$	$\overline{DACK}$

51. FIFO Banking—Byte 16, Bits 1–0

Specify the lengths of the Transmit and Receive FIFOs

Default setting: 00

52. Port 1 Enable—Byte 16, Bit 2

Configures the $\overline{DACK1}/\overline{CS1}/\overline{EOP}$ pin to function as $\overline{CS1}/\overline{OP}$, and the $\overline{DACK0}/\overline{DACK}$ pin as $\overline{DACK}$. If transmit EOP or receive EOP is enabled (see below), the $\overline{CS1}/\overline{EOP}$ pin is an I/O pin.

INSERT TABLE 6-8

Default setting: 0

53. Byte Swap—Byte 16, Bit 3

Valid only when the 82592 is configured to a 16-bit data path. Enables the 82592 to change the byte order used for DMA transfers. The low order bytes are transferred on the D8-15 lines and the high order bytes on the D0-7 lines. Command/status bytes are, however, always transferred on the D0-7 lines. This bit must be zero for the 82590 or the 82592 configured to 8-bit mode.

0—Disable

1—Enable

Default setting: 0

54. CLK Divider—Byte 16, Bit 4

Valid only for the case where the CLKSRC pin is strapped HIGH. The CLKSRC pin is available with the 82590 or the 82592 packaged in a 44-pin PLCC. When the CLKSRC is strapped HIGH, the timing for the parallel subsystem is internally generated from the clock generator provided in the serial subsystem. This configuration bit enables the internal divide-by-two prescaler, so the clock frequency of the parallel subsystem is one-half that of the clock generator.

0—Disable

1—Enable

Default setting: 1

55. Status Length—Byte 16, Bit 5

Specifies whether the 82590 is to have 4 or 6 bytes of status registers in Port 0.

0—4 bytes

1—6 bytes

Default setting: 0

56. Transmit EOP—Byte 16, Bit 6

Enables the 82590 to assert the $\overline{\text{EOP}}$ (End Of Process) signal to the DMA controller when a collision is detected within the Collision Window (Slot Time). The 82590 does not generate an interrupt to request retransmission. In order to implement automatic retransmission, the DMA controller must be programmed to Auto-Initialize mode.

When the 82590 is configured for Tightly Coupled Interface (TCI) mode, the $\overline{\text{EOP}}$ signal is asserted at the end of every frame transmission and the quality of each transmission is indicated by the DRQ line.

0—EOP not asserted

1—EOP asserted

Default setting: 0

57. Receive EOP—Byte 16, Bit 7

If the chaining bit is not set, this bit enables the 82590 to assert the $\overline{\text{EOP}}$ signal at the end of a frame reception. The signal is asserted regardless of the quality of frame reception.

If the chaining bit is set, this bit enables the 82590 to assert the $\overline{EOP}$ signal at the end of a bad frame reception. Automatic buffer reclamation can be implemented by programming the DMA controller to Auto-Initialize mode.

When the 82590 is configured for Tightly Coupled Interface (TCI) mode, the $\overline{EOP}$ signal is asserted at the end of every frame reception and the quality of each reception is indicated by the DRQ line.

0—EOP not asserted

1—EOP asserted

Default setting: 0

Controlling the 82590 Through Port 0

7

CHAPTER 7
CONTROLLING THE 82590 THROUGH PORT 0

This chapter specifies how the 82590 is controlled through Port 0. It starts with the definition of the Port 0 command and status registers. It then specifies the behavior of the 82590 during execution of operations and reception of frames. Chapter 10 provides the detailed format and meaning of all the Port 0 operations and status.

The 82590 performs OPERATIONS, such as transmitting a frame, enabling reception, configuring the chip, or performing diagnostics, through Port 0. A Port 0 operation is initiated by the CPU, by writing a COMMAND into the Port 0 command register in the 82590. There are three groups of operations: EXECUTION, RECEPTION, and STATUS POINTERS CONTROL (in addition to a Software Reset). The first two groups correspond to two logical units of the 82590: the EXECUTION UNIT and the RECEIVE UNIT. The third group provides control for the status registers.

There is a set of Port 0 EVENTS that may occur asynchronously in response to Port 0 commands. These events include: end of incoming frame, frame aborted, execution completed for command, and execution aborted. The 82590 reports on these events by updating a set of Port 0 status registers and raising the INTERRUPT signal. Depending on its configuration, the 82590 has 4- or 6-bytes of status registers, all of which are read by the CPU from the same address.

Following an event, status is updated and interrupt is raised only when the interrupt signal is low. When the interrupt is high the CPU can read the status. The interrupt is cleared by a command from the CPU whenever the INTERRUPT ACKNOWLEDGE bit is set.

The 82590 interacts with the DMA controller via two REQUEST/ACKNOWLEDGE lines (referred to as two CHANNELS). The 82590 requests the transfer of bytes, and the DMA performs and acknowledges the transfer. Frames and parameters are transferred via the two channels from/to memory. Commands allocate a specific channel for each operation.

The 82590 can be configured to support receive buffer CHAINING. In this mode two channels must be allocated to the reception of frames, and the 82590 switches channels when a buffer is full (Buffer Switching mode) or when a frame is received (Frame Switching mode). It issues a "Request Alternate Buffer" interrupt in order to allow the CPU to set up the alternate buffer on the other channel.

7.1 COMMAND REGISTER

This section specifies the format of the 82590 command set. A command is given to the 82590 by writing it into the command register (see Figure 7-1). The command can be issued at any time, but in case it is not accepted, the operation is treated like a NOP and is ignored by the 82590 (although the INT and the PTR will be updated).

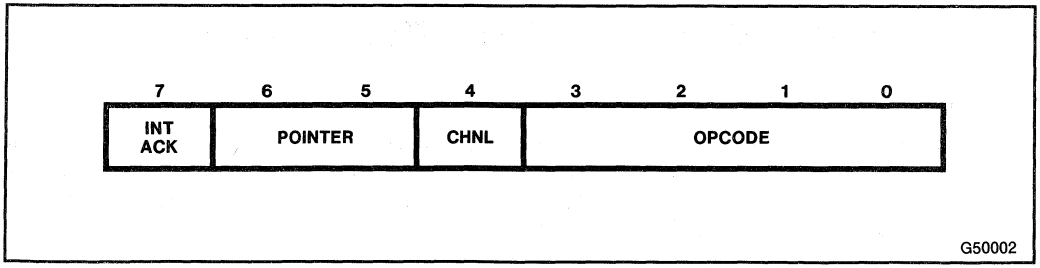


Figure 7-1. Port 0 Command Register

7.1.1 Operations (Bits 0–3)

The OPERATION field initiates a specific operation. There are three groups of operations that can be initiated independently: EXECUTION, RECEPTION, or STATUS POINTER CONTROL.

The 82590 implements the following EXECUTION operations (shown with their codes):

- NOP 0 (0)
- SWITCH-TO-PORT1 0 (1)
- IA-SETUP 1
- CONFIGURE 2
- MC-SETUP 3
- TRANSMIT 4
- TDR 5
- DUMP 6
- DIAGNOSE 7
- RETRANSMIT 12
- ABORT 13

Execution operations except ABORT are accepted only when the Execution unit is IDLE. ABORT is only accepted if the Execution unit is active.

The following are RECEPTION operations (shown with their codes):

- RCV-ENABLE 8
- ASSIGN ALT BUF (chaining only) 9
- RCV-DISABLE 10
- STOP-RCV 11

RCV-ENABLE is accepted only when the Receive Unit is IDLE. The other receive operations are accepted only when the Receive Unit is not IDLE.

The following are the STATUS POINTER CONTROL OPERATIONS:

- FIX PTR 15 (1)
- RLS PTR 15 (0)

The last operation is software reset:

- RESET 14

RESET can always be issued. It has the same affect as a hardware reset.

Operations that are not accepted are simply ignored (although INT and PTR are updated).

7.1.2 Channel Allocation

The CH field selects the channel for the operation that is requested. Note: this field applies only for IA-SETUP, CONFIGURE, MC-SETUP, TRANSMIT, RETRANSMIT, DUMP, and RCV-ENABLE. If the selected channel is already allocated, the operation is ignored (acts like NOP).

7.1.3 Setting the Pointer

The PTR field specifies the new value of the internal pointer. The pointer selects the next status register to be read. It is advanced automatically in modulo 4 when status is read. Automatic advancement may be disabled by issuing the FIX PTR command.

7.1.4 Acknowledging Interrupt (Bit 7)

The INT-ACK bit, if set, causes the interrupt hardware signal and the interrupt bit to be cleared. This is the only way to clear the interrupt bit and reset the interrupt signal other than by a reset.

7.2 STATUS REGISTERS

The 82590 can be configured to have 4 or 6 bytes of status registers in Port 0.

When configured to 4 bytes of status registers, the registers are numbered 0 to 3 (see Figure 7-2). The register that the internal pointer points to, is read by an I/O read operation at Port 0. STATUS 1 and STATUS 2 are also called "result registers."

The 82590 provides the information about the last operation that was executed, or the last frame received, in the first three status registers. The fourth status register provides the state of the chip itself.

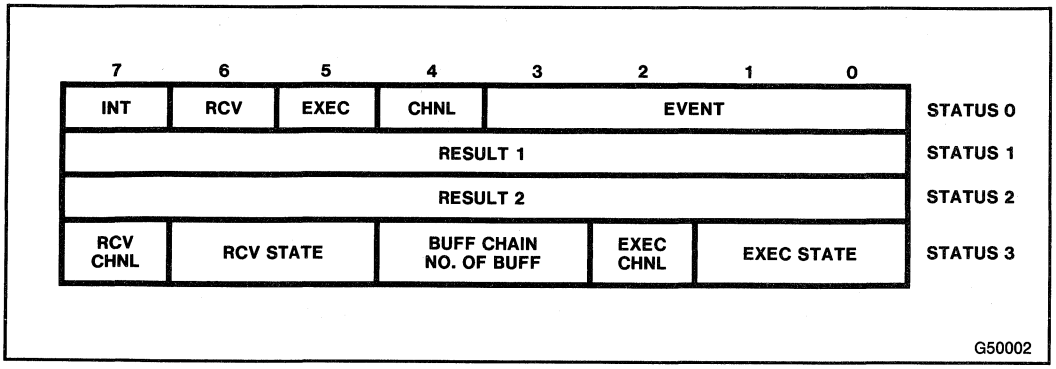


Figure 7-2. Port 0 Status Registers—4 Bytes

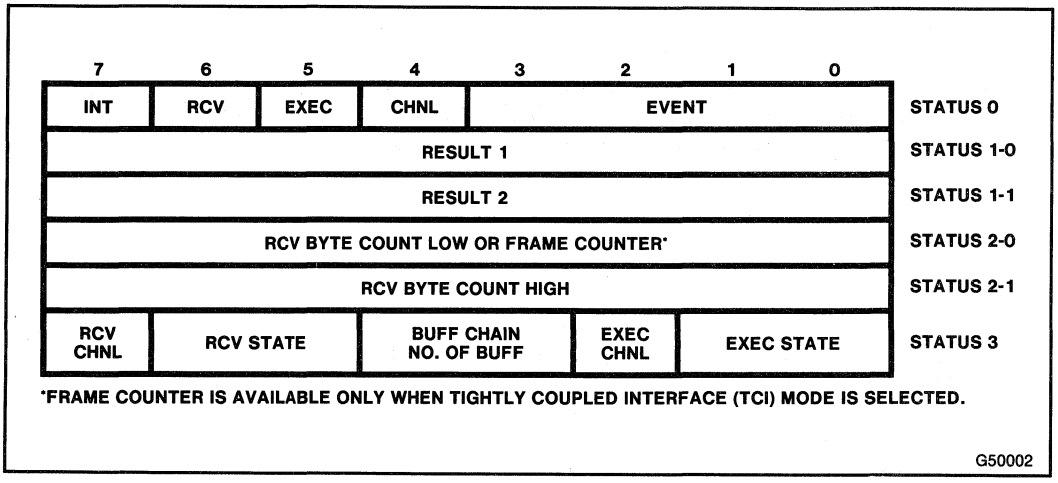


Figure 7-3. Port 0 Status Registers—6 Bytes

STATUS 0, STATUS 1, and STATUS 2 are updated only when INT is not set. STATUS 3 is updated whenever an update is needed, unless it is pointed to by the pointer.

When configured to 6 bytes of status registers, the registers are numbered 0, 1-0, 1-1, 2-0, 2-1, and 3 (see Figure 7-3). In this configuration, STATUS 1-1 and STATUS 2-1 cannot be directly pointed to by the internal pointer. The pointer can only point to STATUS 0, STATUS 1-0, STATUS 2-0, and STATUS 3 and is advanced in round robin among these registers. Whenever the pointer is pointing to STATUS 1-0 or STATUS 2-0, two read cycles can be performed at that pointer location. The first read cycle is to read STATUS 10 or STATUS 2-0 and the second read cycle is to read STATUS 1-1 or STATUS 2-2. If the pointer is fixed at STATUS 1-0 or STATUS 2-0, read cycles are repeatedly performed between STATUS 1-0 and STATUS 1-1 or between STATUS 2-0 and STATUS 2-1. If the pointer is not fixed, six consecutive read cycles allow the CPU to examine all Port 0 status registers.

In the 6-byte status configuration, the first three status bytes are used in the same way as in the 4-byte status configuration, reporting the result of the last operation that was executed. However, for frame reception, two additional bytes are used to report more complete result in this configuration. After a frame is received, status of the frame reception is contained in STATUS 1-0 and STATUS 1-1 and the byte count of the received frame is in STATUS 2-0 and STATUS 2-1. In the 4-byte status configuration, only the byte count of the received frame is contained in STATUS 2 and STATUS 3 and status of the frame reception is stored in memory. When the 82590 is in the Tightly Coupled Interface mode, STATUS 2-0 is used to report number of frames received (see Figure 7-4). The last status register, STATUS 3, provides the state of the chip itself.

7.2.1 STATUS 0

The first status register (STATUS 0) includes the interrupt bit and the cause of the interrupt. Information about the relevant channel is also provided.

7.2.1.1 EVENT (BITS 0–3)

The event field specifies the cause of an interrupt.

The following events may occur (shown with their codes):

- IA-SET-DONE 1
- CONFIGURE-DONE 2
- MC-SETUP-DONE 3
- TRANSMIT-DONE 4
- TDR-DONE 5
- DUMP-DONE 6
- DIAGNOSE-PASSED 7
- END OF FRAME 8
- REQUEST ALT BUFFER 9 (chaining only)
- RECEPTION ABORTED 10

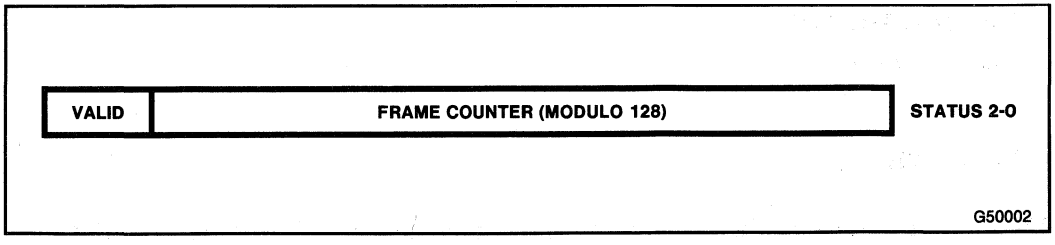


Figure 7-4. STATUS 2-0 Format in TCI Mode

- RETRANSMIT-DONE 12
- EXECUTION-ABORTED 13
- DIAGNOSE-FAILED 15

7.2.1.2 CHANNEL (BIT 4)

The CH bit indicates which channel was allocated for the operation that caused the event. Valid for all events except for REQUEST-ALT-BUFFER, TDR-DONE, and DIAGNOSE-PASSED, AND DIAGNOSE-FAILED.

7.2.1.3 EXECUTION (BIT 5)

The EX bit indicates the completion of an execution operation. Event #13 indicates the abortion of the operation.

7.2.1.4 RECEIVE (BIT 6)

The RCV bit indicates that the reception of a frame was completed, aborted by the Abort command, or not completed because a new buffer is required to allow reception to proceed. Event #8 to 10 provide details.

7.2.1.5 INTERRUPT (BIT 7)

The INT bit is set together with the hardware interrupt signal. Setting the INT bit indicates the occurrence of an event. This bit is cleared by any command whose INT-ACK bit is set.

7.2.2 STATUS 1 (1–0) Through STATUS 2 (2–1)

These registers hold the result of a completed operation. These registers are updated only when any of the following events occur:

- TRANSMIT-DONE
- TDR-DONE
- RETRANSMIT-DONE
- END-OF-FRAME

7.2.3 STATUS 3

The last status register holds the state of the 82590: three lower bits for the Execution unit, five higher bits for the Receiver unit.

7.2.3.1 EXECUTION STATE (BITS 0–1)

The EX-CH field specifies the state of the Execution Unit. It can take one of the following values:

0—IDLE: The Execution Unit is disabled

2—ACTIVE: The Execution Unit is actively executing an operation

3—ABORT IN PROGRESS: The execution of an operation was aborted, but the completion event was not yet issued

7.2.3.2 EXECUTION CHANNEL (BIT 2)

The EX-CH bit indicates which channel is allocated for transferring parameters from/to memory. It is only valid when the Execution Unit is active, and performing a parametric execution.

7.2.3.3 BUFFER CHAINING—NUMBER OF BUFFERS (BIT 3–4)

When configured to "Buffer Chaining" it specifies the number of available buffers for reception. It can take one of the following values:

0—No buffers are available

1—One buffer is available (automatic on RCV enable)

2—Two buffers are available

7.2.3.4 RECEIVE STATE (BITS 5–6)

The RCV STATE field specifies the state of the Receive Unit. It can take one of the following values:

0—IDLE	The Receive Unit is disabled
1—READY	The Receive Unit is set up for an incoming frame
2—ACTIVE	The Receive Unit is actively transferring bytes to memory (provided buffer is available)
3—STOP IN PROGRESS	The Receive Unit is active and will be disabled following the reception of current frame

7.2.3.5 RECEIVE CHANNEL (BIT 7)

The RCV-CH bit indicates which channel is allocated for transferring incoming data to system memory. It is valid only when the Receive Unit is not IDLE. When configured for

Buffer Chaining, at least one buffer should be available. The 82590 assumes that one buffer is available whenever the receiver is enabled. The DMA controller must be initialized and ready to transfer data prior to enabling the receiver.

7.3 PERFORMING EXECUTION OPERATIONS

The 82590 has a repertoire of eleven Port 0 execution operations. These operations perform transmission, configuration, diagnostics, and abortion. All the operations are initiated by writing a command to the Port 0 command register. This causes the Execution Unit to become Active. Some of the operations read parameters from memory.

The parameters for IA-Setup, Configure, MC-Setup, Transmit, and Retransmit are organized in a block that starts with a 16-bit byte count. The byte count specifies the length of the rest of the block. Before beginning the operation, the DMA pointer of the selected channel must point to the first byte of the byte count. There is no restriction on the memory structure of the frame so long as the 82590 receives the next byte of the frame upon each DMA request. Transferring the bytes is the job of the DMA controller or the CPU. If the DMA does not keep up with transmission (i.e., causes an underrun), the transmission is terminated and an underrun is indicated.

The 82590 requests two bytes of the byte count on the allocated channel and determines the length of the parameter block. It then requests the parameters and starts the execution of the operation.

Upon completion of the operation (when interrupt is LOW) the Execution unit does the following: it updates STATUS 0, it updates RESULT 1 and 2 (for Configure, Transmit, TDR, and Retransmit), it raises the interrupt signal, and it goes IDLE.

The ABORT operation causes a request for abortion of the operation that is in progress. In some cases the operation can no longer be aborted (e.g., for a Transmit operation when the entire frame was already transmitted). In these cases the Abort operation is ignored. Nonparametric executions such as TDR and DIAGNOSE cannot be aborted.

The DUMP operation causes a set of internal registers to be written to memory over the allocated channel. It is completed by updating STATUS 0 and raising the interrupt.

7.4 FRAME RECEPTION

The 82590 receives, and passes to memory, all frames whose addresses match the individual, multicast, or broadcast address. Reception of frames is pipelined, i.e., the reception of a frame can start before the end of frame processing of the previous frame is completed. The 82590 is configured to either chaining, consecutive, or frame switching mode. In consecutive mode the entire incoming frame is passed to a single buffer in memory via the allocated channel. In chaining mode the frame is received and deposited in a series of fixed size buffers using the two available DMA channels alternately. The chained buffer size is configurable in steps of 4 bytes—to a maximum of 1 Kilobyte. In frame switching mode each frame is received into one continuous buffer as in the consecutive mode, but the DMA channel is switched to the alternate at the end of each reception.

7.4.1 Consecutive Mode (Single Buffer Mode)

Before the start of reception, the DMA pointer of the specified channel points to the first byte of the available receive buffer. Transferring the bytes is the job of the DMA controller or the CPU.

If the receiver is ready and a frame arrives, the 82590 requests the transfer of bytes to memory, using DMA. After transferring all the bytes (addresses, control, and information) the 82590 transfers two bytes of status to memory. Upon completion of the reception, and when interrupt is LOW, the Receive Unit does the following: it updates appropriate status registers; raises the interrupt signal; and goes to READY state. The DMA Controller would be pointing to the byte following the frame status at this point.

If the receiver is not ready when the first byte of the frame arrives, the frame is not received and no status is updated. Disabling reception after the first byte has passed to memory causes the rest of the frame to be discarded. An interrupt with a Receive Aborted event is then issued.

Reception can begin even when the interrupt that indicates the completion of execution or reception is still pending. This capability allows pipelining of the transfer to memory of a frame by the 82590 and the DMA controller, with the CPU handling the completion of the previous execution or reception (including the interrupt latency). The entire frame gets deposited in a single memory buffer with a 64 kbyte maximum size.

7.4.2 Chaining Mode (Multiple Buffer Mode)

Before the start of reception, the DMA pointer of the first DMA channel points to the first byte of the available buffer. If the receiver is ready (one buffer allocated) and a frame arrives, the 82590 generates an interrupt with a Request Alternate Buffer event, notifying the CPU to allocate an additional buffer. Simultaneously, the 82590 requests the transfer of bytes to memory. Note that if an additional buffer is available when the first byte arrives, then the Request Alternate Buffer event is not issued.

When the currently used buffer is full, and an additional buffer is available, the 82590 switches the DMA channel to access the new buffer. If an additional buffer is not available the receiver deasserts the DMA Request signal and suspends further data transfers to memory. The CPU should then provide another buffer or disable reception. If a new buffer is provided in time, data transfers to memory continue. If a Disable Receive command is issued, then the rest of the frame is discarded.

This process of switching from channel to channel continues until the end of the frame arrives. At this point, two frame status bytes are transferred to memory, a buffer switch occurs, appropriate status registers are updated, and if a new buffer is available the 82590 is ready to receive the next frame. Otherwise, reception is suspended. In both cases an End of Frame event is issued.

If the receiver is not ready when the first byte of the frame arrives, then the entire frame is discarded and status is not updated. Disabling reception after the first byte was passed to memory causes the rest of the frame to be discarded, and an interrupt with a Receive Aborted event to be issued.

If reception is suspended (Ready, but with no buffers assigned) when the first byte of the frame arrives, the receiver interrupts the CPU for a buffer. The user may provide a buffer or disable Reception in this case.

7.4.3 Frame Switching Mode

In this mode each frame is received into one continuous buffer, like the Single Buffer reception, but buffer switching is automatically performed at the end of each frame reception. When implemented with one DMA channel, the End of Process (EOP) signal is used to request the DMA controller to reinitialize the DMA channel. When implemented with two DMA channels, the operation is very similar to the way Buffer Switching is done. As one frame is received using one DMA channel, the 82590 generates the Request Alternate Buffer interrupt in order to be prepared for the next frame reception. The CPU programs the alternate DMA channel and allocates a new buffer to the 82590. When the 82590 finishes receiving the current frame, it immediately switches to the alternate DMA channel. As soon as the next frame arrives, it is received in the new buffer using the alternate DMA channel. This process is repeated on every frame.

7.5 TIGHTLY COUPLED INTERFACE

In this mode the 82590 provides a tightly coupled interface to a DMA controller in order to execute some of the time-critical processes of the CSMA/CD protocol without any CPU intervention. By using the 82590's companion chip, the 82560, or by implementing the Tightly Coupled Interface (TCI) in a DMA controller, operations such as automatic retransmission, continuous back-to-back frame reception, and transmit and/or receive buffer chaining can be accomplished.

7.5.1 DMA Interface

The 82590 provides the status of the current active transmission or reception to the DMA controller by using the DRQ and $\overline{EOP}$ signals at the end of every DMA cycle. The status is encoded according to Table 7-1. As long as the 82590 generates DRQ High and $\overline{EOP}$ Floating at the rising edge of $\overline{RD}$ or $\overline{WR}$, the DMA controller repeats DMA transfers. If the transmission is completed without collisions or if the reception is good, then DRQ and $\overline{EOP}$ both become Low at the end of a DMA transfer which follows the last DMA data transfer. If the transmission encountered a collision or if the reception had an error, DRQ becomes High and $\overline{EOP}$ becomes Low. The DMA controller must decode these signals appropriately and must reinitialize the DMA channel so it can retransmit the same frame or reclaim the otherwise wasted buffer. It is the DMA controller's responsibility to reprogram itself for the next appropriate operation.

Table 7-1. Transmit/Receive Status Encoding on DRQ and $\overline{EOP}$

DRQ	$\overline{EOP}$	Status Information
0	Hi-Z	Idle
1	Hi-Z	DMA Transfer
0	0	Transmission or Reception Terminated OK
1	0	Transmission or Reception Aborted

7.5.2 CPU Interface

When the 82590 is configured for the Tightly Coupled Interface, retransmission upon collision, buffer reclamation upon bad reception, and transmit and receive buffer management are automatically handled by the DMA controller. In order to implement these functions with the DMA controller, the 82590's interrupt function is modified in the TCI mode.

7.5.2.1 INTERRUPT DUE TO COMMAND EXECUTION

After a Transmit command is issued to the 82590, the 82590 may generate an interrupt to the CPU only upon one of the following events:

- All chained Transmit commands are executed without errors and the 82590 successfully reached the end of the transmit chain.
- A transmission attempt failed due to a reason other than a collision.
- A transmission attempt failed due to the maximum number of collisions encountered.
- A transmission attempt was aborted by an Abort command.

All other commands besides the Transmit command cause the 82590 to generate interrupts at the end of every execution as in the non-TCI mode.

7.5.2.2 INTERRUPT DUE TO FRAME RECEPTION

The TCI mode makes the End-of-Frame interrupt a non-time-critical interrupt. In other words, the 82590 in this mode is capable of receiving frames continuously without having the CPU service the End-of-Frame interrupt(s). When a frame is received, the 82590's interrupt logic may be in one of the following states. Operations performed by the 82590 are described for each state.

1. The INT line is Low. There are no interrupts pending.

   In this case STATUS 0 is updated with End-of-Frame. The INT line is then raised.

2. STATUS 0 already contains End-of-Frame in the Event field, and the INT line is already High.

   In this case the 82590 will not update STATUS 0. STATUS 0 will hold the same status as before, and the INT line will continuously remain High.

3. STATUS 0 contains an interrupt event resulted from a command execution, and the INT line is already High.

   The 82590 will not update STATUS 0 with End-of-Frame in this case. STATUS 0 will contain the same command execution event in the Event field until the interrupt is acknowledged. While the interrupt is pending, the 82590 is able to receive incoming frames continuously as long as it has enough memory space and timely DMA services. When the command execution interrupt is acknowledged by the CPU, the INT line will become LOW. If a frame reception is not in progress, the 82590 will immediately update STATUS 0 with End-of-Frame and raise the INT line. This will inform the CPU of queued End-of-Frame interrupt(s). The actual number of frames received before the End-of-Frame interrupt is generated can be found in a RESULT register. If a frame reception is in progress at the time the command execution interrupt is acknowledged, the 82590 will post End-of-Frame and raise the INT line after the reception is completed.

Note that a command cannot be executed unless a pending interrupt due to the previous command is acknowledged. However, frames can be received regardless of the command or receive interrupt status.

It is also possible to disable the End-of-Frame interrupt in the TCI mode. When this configuration is selected, the 82590 normally does not generate an interrupt at the end of frame reception. It is the DMA controller's responsibility to decode the DRQ and $\overline{EOP}$ lines appropriately and generate interrupts to the CPU. The 82590 in this mode generates the End-of-Frame interrupt only when a reception is aborted by an Abort command.

Controlling the 82590 Through Port 1

8

CHAPTER 8
CONTROLLING THE 82590 THROUGH PORT 1

8.1 COMMAND REGISTER

This section specifies the format of the Port 1 command set. The Port 1 commands allow the user to control such 82590 features as local and remote power-down modes, the 24-bit general purpose timer, and the three 16-bit event counters. Figure 8-1 shows the format of the Port 1 Command Register. It is write only and occupies only one I/O address location. Depending on the OPERATION field specified in the command byte, two or three additional write cycles may have to be performed at the same address in order to load parameters into the 82590. One such example is the SET-VAL command which initializes the timer or counter value. If the command is issued for the timer, three additional consecutive write cycles must be performed in order to define all 24 bits for the timer. If it is issued for the counter, only two consecutive write cycles are needed.

8.1.1 TC/$\overline{GP}$ Select (Bit 4)

All other command fields in the Port 1 command byte, OPERATION, POINTER, and INT ACK, are multiplexed by this bit between the timer/counter (TC) and the general purpose (GP) groups.

8.1.2 Operations (Bits 0–3)

The OPERATION field initiates a specific operation. Two groups of the operations are multiplexed by the TC/$\overline{GP}$ bit.

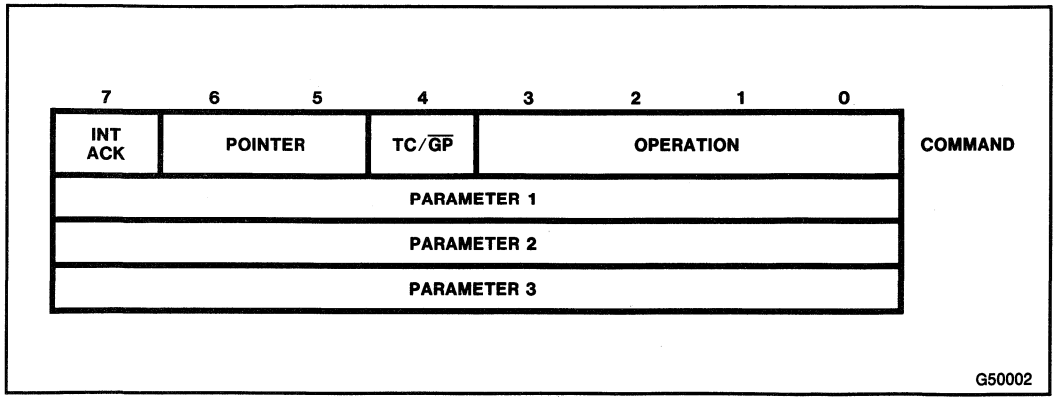


Figure 8-1. Port 1 Command Register

When the TC/$\overline{GP}$ bit is zero, the following general purpose commands shown below with their codes are implemented. The operation codes that are not shown are reserved by Intel and must not be used.

- NOP 0
- SWT-TO-PORT-0 1
- SET-TS 5
- RESET-TS 7
- LCL-PWR-DWN 8
- RMT-PWR-DWN 9
- FIX-PTR 12
- RLS-PTR 13
- RESET 14
- SEL-RST 15

When the TC/$\overline{GP}$ bit is one, the following timer/counter commands shown below are implemented. Those commands that are marked with asterisks require additional write cycles to load the required parameters into the 82590. The operation codes that are not shown are reserved by Intel and must not be used.

- NOP 0
- START 1
- STOP 2
- RESUME 3
- LD&START 5
- ACK-INT 7
- COUNT 8
- START-ALL-COUNTERS 9
- SET-VAL* 10
- SET-CONF* 11
- RD-MAX-COUNT-VAL 12
- RD-COUNT-VAL 13
- RESET 14

8.1.3 Setting the Pointer

The PTR field specifies the new value of the internal pointer. When the TC/$\overline{GP}$ bit is 0, the pointer selects the next status register to be read.

- STATUS 0 0
- STATUS 1 or 1-0 1
- STATUS 2 or 2-0 2
- STATUS 3 3

When the TC/$\overline{GP}$ bit is 1, the pointer specifies one of the timer or event counters for the timer/counter command given in the OPERATION field.

- TIMER 0
- EVENT COUNTER 1 1
- EVENT COUNTER 2 2
- EVENT COUNTER 3 3

The internal pointer is advanced (modulo 4) automatically when a status register is read or when all bytes of the specified timer or event counter are read. Automatic advancement for the status registers may be disabled by issuing the FIX PTR command.

8.1.4 Acknowledging Interrupt (Bit 7)

The INT-ACK bit is used to acknowledge (clear) the interrupt hardware signal and the interrupt bit in the Port 1 status register. For counter/timer interrupts, this bit must be used together with the ACK-INT command. For interrupts due to general purpose operations in Port 1, the INT-ACK bit is used alone to acknowledge the interrupt.

8.2 STATUS AND TIMER/COUNTER REGISTERS

There are three banks of read-only registers in Port 1: one bank of four status registers, one bank of four timer/counter count registers, and one bank of four timer/counter maximum count registers (see Figure 8-2). When a read cycle is attempted at Port 1, a byte is normally read from the status register bank in a similar way a status byte is read from Port 0. However, if one of the bank switch commands, RD-COUNT-VAL or RD-MAX-COUNT-VAL, had been issued to the Port 1 command register, the read cycle is operated from one of the other two timer/counter banks. RD-COUNT-VAL selects the bank of the timer/counter count registers and RD-MAX-COUNT-VAL selects the bank of the timer/counter maximum count registers. Once the Port 1 is selected for one of these timer/counter register banks, the selection is maintained until a new Port 1 general purpose command is issued to the Port 1 command register. The least significant byte of the timer or counter is read first. After reading the entire 24 or 16 bits, the internal pointer is advanced to the next timer or counter register. All timer and counter values are read after nine consecutive read cycles.

In the status bank, the 82590 provides the interrupt information in STATUS 0, timer/counter states in STATUS 1, timer/counter configuration in STATUS 2 and in bits 0 through 3 of STATUS 3, and the port 1 machine states in bits 4 through 7 of STATUS 3.

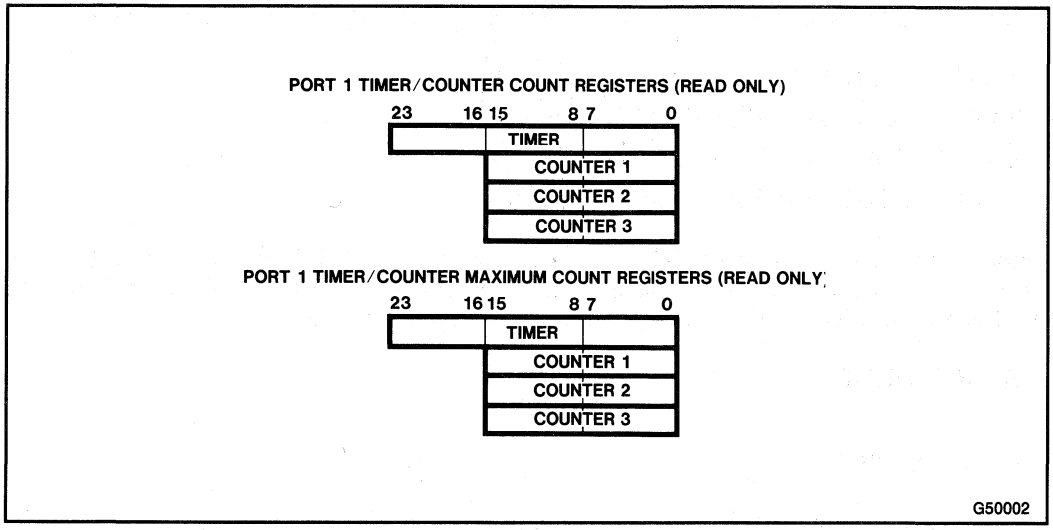


Figure 8-2. Port 1 Status and Timer/Counter Register Bank

8.2.1 STATUS 0

The first status register, STATUS 0, includes the interrupt bit and the cause of the pending interrupt. The event field is multiplexed by the TC and GP bits. STATUS 0 contains valid information only when the TC or GP bit is set.

8.2.1.1 INTERRUPT (BIT 7)

The INT bit is set together with the hardware interrupt signal. The cause of the active interrupt is indicated in the event field. This bit is cleared by issuing an appropriate interrupt acknowledge command to the PORT 1 command register.

8.2.1.2 GP (BIT 5)

The GP (General Purpose) bit indicates that the pending interrupt is caused by the Port 1 general purpose operation.

8.2.1.3 TC (BIT 4)

The TC (Timer/Counter) bit indicates that the pending interrupt is caused by the timer or one of the event counters.

8.2.1.4 EVENT (BITS 3–0)

The EVENT field is multiplexed by the TC and GP bits.

When the TC bit is one, the following timer/counter events shown below are reported.

- TIMER EXPIRED BIT 0 = 1
- EVENT COUNTER 1 EXPIRED BIT 1 = 1
- EVENT COUNTER 2 EXPIRED BIT 2 = 1
- EVENT COUNTER 3 EXPIRED BIT 3 = 1

These four events are independently reported. Therefore, the CPU may find more than one bit being set when it reads this field. When the GP bit is one, the following general purpose event is reported. There is only one general purpose event defined.

- REM-PWR-UP 9

8.2.2 STATUS 1

The second status register, STATUS 1, contains the states of the timer and event counters. Two bits are used for each timer or event counter as shown in Figure 8-3. The state of the timer or counter is decoded as follows:

- IDLE 00
- RUNNING 01
- STOPPED 1X

The RUNNING state is entered only from the IDLE or STOPPED state by executing one of the following commands: START or RESUME. The timer or event counter configured to Sticky Mode (see Section 10.2) goes into the IDLE state upon reaching the maximum count. If a STOP command is issued while in the RUNNING state, it enters the STOPPED state.

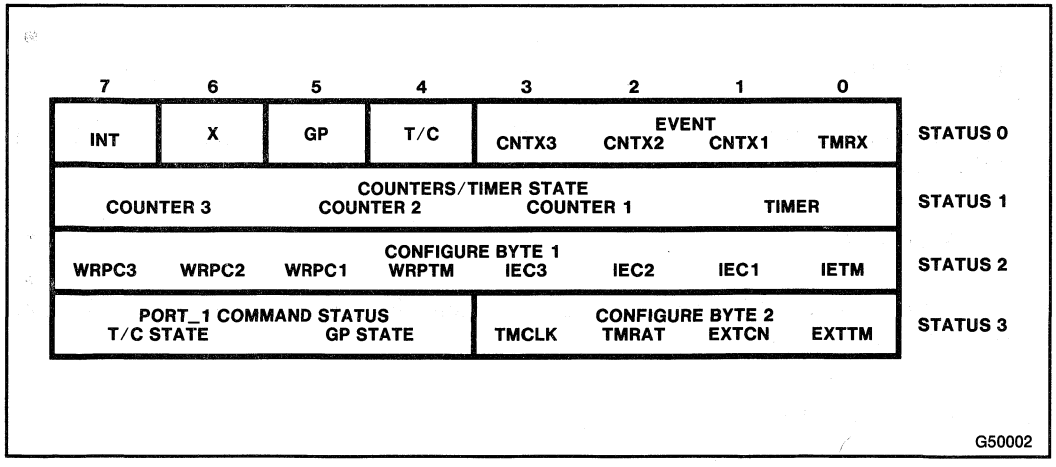


Figure 8-3. Port 1 Status Registers

8.2.3 STATUS 2 and STATUS 3 Bits 0–3

Bits in these registers indicate the current configuration of the timer and the event counters. The format is identical to that of the SET-CONFIGURE command. See Section 10.2 for details.

8.2.4 STATUS 3 Bits 4–7

These bits indicate the state of Port 1.

8.2.4.1 TC STATE (BITS 7–6)

The Port 1 is in one of the following three states after receiving a timer or counter command.

- IDLE 00
- PROCESSING 10
  (only after a timer command)
- BUSY 11

Timer and counter commands are accepted only when the TC state is IDLE. Whenever a timer command is issued, the TC state becomes PROCESSING. The PROCESSING state notifies the CPU that the command is actively being executed. Duration of the PROCESSING state depends on the system clock to timer clock ratio. The state returns to IDLE once the execution is completed. Counter commands are executed immediately; therefore, the TC state machine does not enter the intermediate PROCESSING state. The BUSY state is entered when the SET-VALUE or SET-CONFIGURE command is issued. These commands require subsequent write cycles to load parameters. The BUSY state indicates the CPU that it must write more parameter(s). Once all parameters are written, the state becomes PROCESSING or IDLE, depending if it was issued for the timer or one of the counters.

8.2.4.2 GP STATE (BITS 5–4)

The Port 1 enters one of the following GP states after receiving a general purpose command.

- ACTIVE 00
- LOCAL POWER DOWN 10
- REMOTE POWER DOWN 01

The ACTIVE state indicates the normal operational state of the 82590.

8.2.5 Timer/Counter Count Registers

These registers contain current values of the timer and event counters (see Figure 8-2). The least significant byte is read first. The pointer is advanced to the next event counter or to the timer register after all bytes of the current event counter or timer register are read. In order to read all registers in this bank, the CPU must perform nine byte-wide read cycles.

8.2.6 Timer/Counter Maximum Count Registers

These registers contain the maximum values configured for the timer and event counters (see Figure 8-2). The least significant byte is read first. The pointer is advanced to the next event counter or timer register after all bytes of the current event counter or timer register are read. In order to read all registers in this bank, the CPU must perform nine byte-wide read cycles.

Port 0 Operations and Status 9

CHAPTER 9
PORT 0 OPERATIONS AND STATUS

This section specifies all the Port 0 operations and events. The 4-bit op-code is shown in parentheses.

The value of byte count (where applicable) does not include the two bytes of the field itself. All shaded areas are reserved by Intel.

9.1 OPERATIONS

9.1.1 NOP (0, 00)

This operation does not affect the 82590. It has no parameters and no result. It is used to manipulate the pointer and/or to acknowledge interrupts.

9.1.2 SWT-TO-PORT-1 (1, 00)

This operation logically switches the I/O port selected by $\overline{CS0}$ to Port 1. All subsequent command write or status read cycles are directed to Port 1. In order to switch the I/O port back to Port 0, the SWT-TO-PORT-0 command must be executed in Port 1.

9.1.3 IA-SETUP (01)

This operation sets up the individual address of the 82590. This address is inserted as the source address during transmission and used for address filtering during reception. Formats of the parameters required for this command are shown in Figures 9-1 through 9-3. The Byte Count should match the 'Address Length' configuration parameter. The length of the address (in bytes) that is set up is determined by the minimum of the Address Length configuration parameter or the value of the Byte Count field.

Only the four least significant bits of the byte count are used. (NOTE: The first bit of the first byte must be a 0.)

An interrupt with the event IA-SETUP-DONE is issued when this operation is done unless ABORT is executed during the parameter transfer.

9.1.4 CONFIGURE (02)

This operation configures the 82590. For the 82592, the Configure command with no parameters, issued immediately after a reset places the device in the 16-bit mode. Formats of the configuration parameters are shown in Figures 9-4 and 9-5. The length of the configuration block that is modified is determined by the byte count. If the byte count is less than 15, then only part of the parameters are updated. Only the four least significant bits of the byte count are used. The details of the configuration parameters are found in Section 6.0.

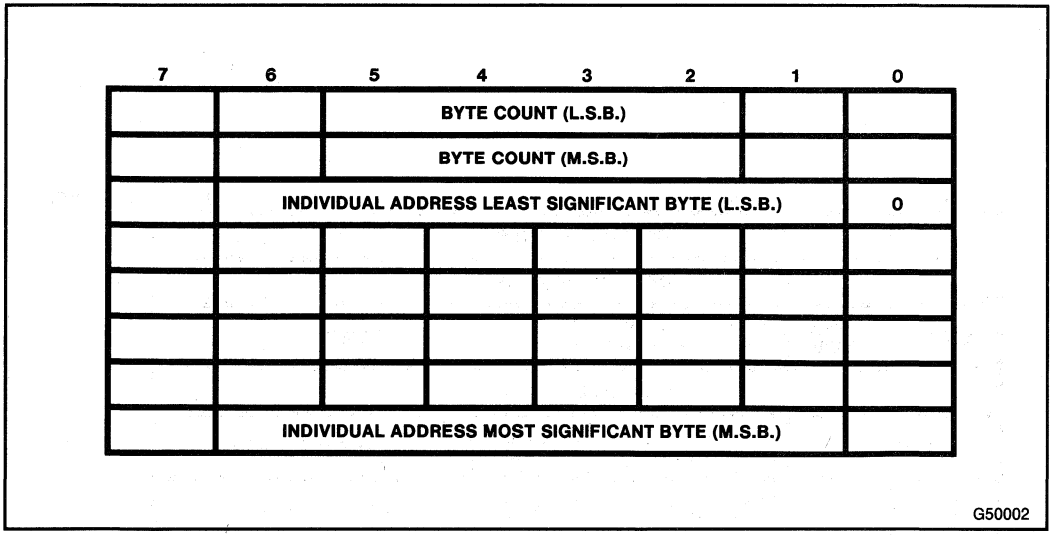


Figure 9-1. IA Parameter Format for the 82590 or the 82592 in 8-Bit Mode

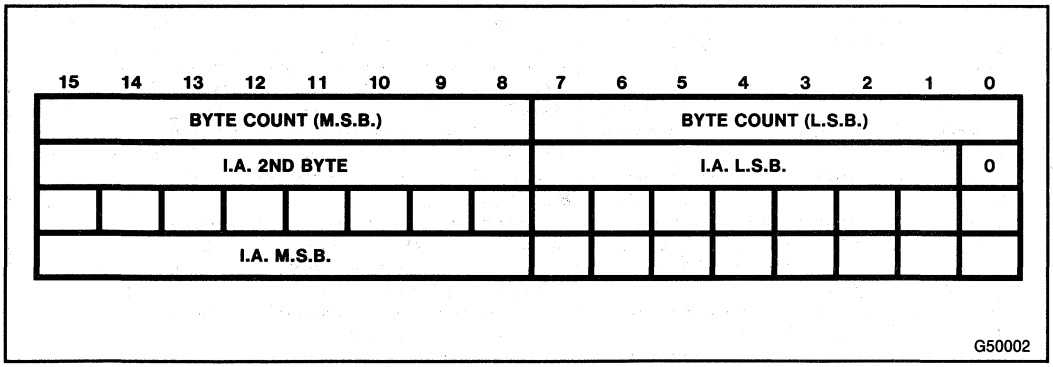


Figure 9-2. IA Parameter Format for the 82592 in 16-Bit Mode

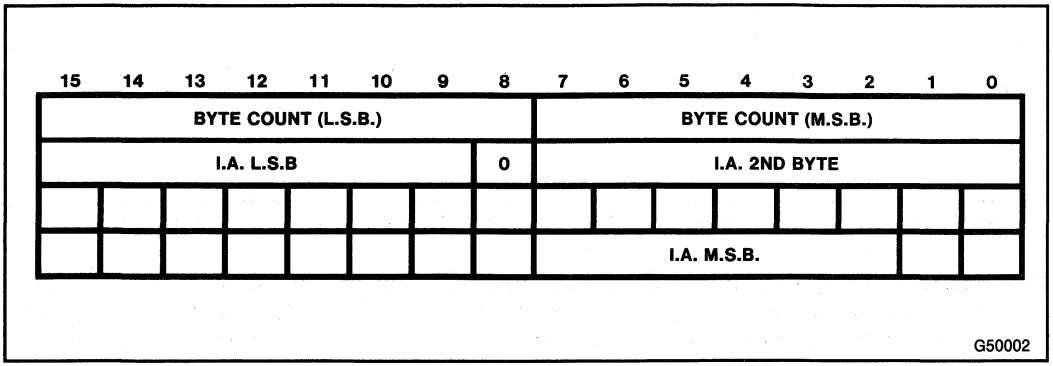


Figure 9-3. IA Parameter Format for the 82592 Configured to Byte Swap in 16-Bit Mode

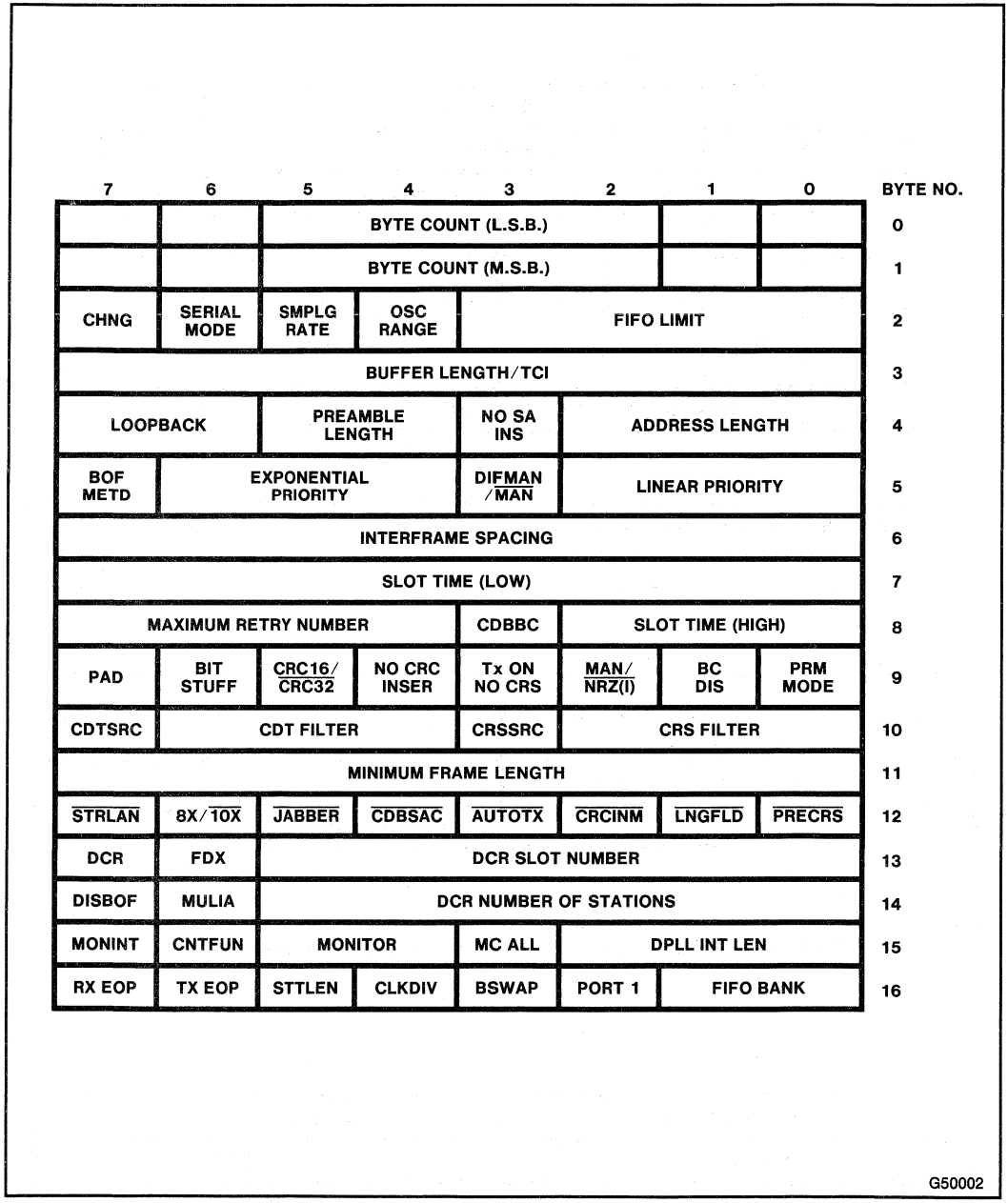
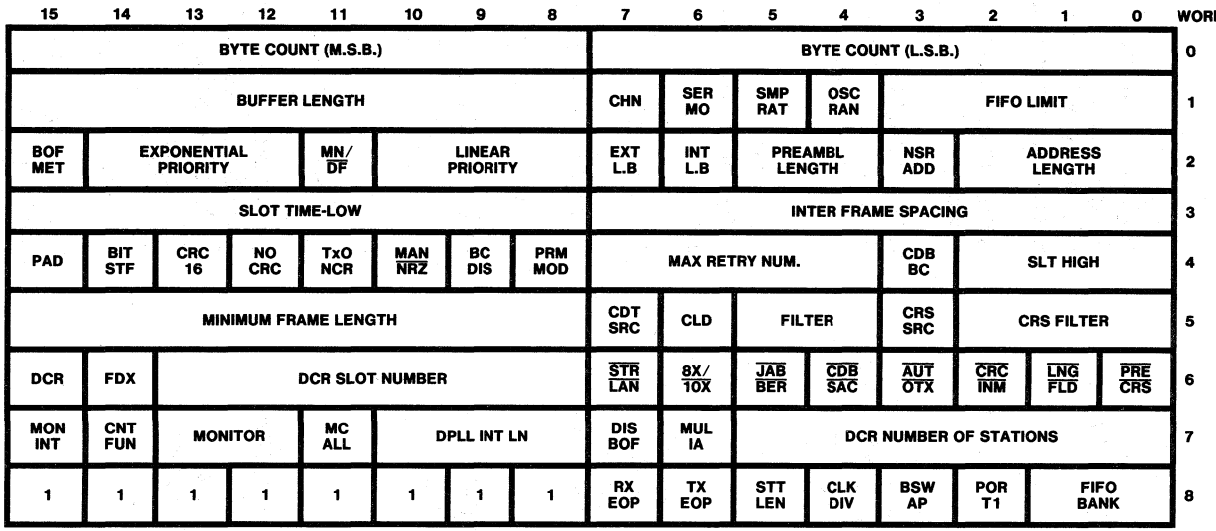


Figure 9-4. Configure Parameter Format for the 82590 or the 82592 in 8-Bit Mode

Note that three bits (Mode, SMPLG RATE, and OSC RANGE) affect operation only during the first Configure command following a reset. (In case of the 82592 in 16-bit mode, the second Configure command since the first one must be issued without parameters to place the device in 16-bit mode.)



NOTE: IF THE 82592 HAS ALREADY BEEN CONFIGURED TO ALTERNATE BYTE ORDER, THE HIGH ORDER BYTES AND THE LOW ORDER BYTES MUST BE SWAPPED.

G50002

Figure 9-5. Configure Parameter Format for the 82592 in 16-Bit Mode

The interpretation of the fields is as follows:

Bytes 0,1	Byte Count	
Byte 2	FIFO Limit (Bits 0–3)	FIFO limit
	OSC RANGE (Bit 4)	High or low frequency range of oscillator
	SMPLG RATE (bit 5)	Sampling rate
	MODE (Bit 6)	High-Integration or High-Speed mode
	CHAINING (Bit 7)	Receive Buffer chaining
Byte 3	BUFFER LENGTH/TCI	Buffer length used for chaining or TCI mode
Byte 4	ADR LEN (Bits 0–2)	Address length
	NO SCR ADDR INS (Bit 3)	Address control location
	PREAM LEN (Bits 4–5)	Preamble length
	LPBK (Bit 6–7)	Loopback
Byte 5	LIN PRIO (Bits 0–2)	Linear priority
	DIF MAN/MAN (Bit 3)	Differential Manchester or Manchester encoding
	EXP PRIO (Bits 4–6)	Exponential Priority
	BOF METD (Bit 7)	Exponential backoff method
Byte 6	INTERFRAME SPACING	Interframe spacing
Byte 7	SLOT TIME (L)	Slot time, low byte
Byte 8	SLOT TM (H) Bits 0–2)	Slot time, high part
	CDBBC (Bit 3)	Collision Detection By Bit Comparison
	RETRY NUM (Bits 4–7)	Number of transmission retries on collision
Byte 9	PRM (Bit 0)	Promiscuous mode
	BC DIS (Bit 1)	Broadcast disable
	MANCH/NRZ(I) (Bit 2)	Manchester or NRZ(I) encoding
	TON NCRS (Bit 3)	Transmit on No CRS
	NCRC INS (Bit 4)	No CRC insertion
	CRC-16/CRC-32 (Bit 5)	CRC type
	BIT STF (Bit 6)	Bit stuffing
	PAD (Bit 7)	Padding
Byte 10	CRSF (Bits 0–2)	Carrier Sense filter bits
	CRS SRC (Bit 3)	Carrier Sense source
	CDTF (Bits 4–6)	Collision Detect filter bits
	CDT SRC (Bit 7)	Collision Detect source
Byte 11	MIN FRAME LEN	Minimum frame length

Byte 12	$\overline{\text{PRECRS}}$ (Bit 0)	Preamble until CRS
	$\overline{\text{LNGFLD}}$ (Bit 1)	Length Field
	$\overline{\text{CRCINM}}$ (Bit 2)	CRC in Memory
	$\overline{\text{AUTOTX}}$ (Bit 3)	Auto Retransmit
	$\overline{\text{CDBSAC}}$ (Bit 4)	Collision Detection By Source Address Comparison
	$\overline{\text{JABBER}}$ (Bit 5)	Jabber
	8X/$\overline{\text{10X}}$ (Bit 6)	8X/16X or 10X/18X
	$\overline{\text{STRLAN}}$ (Bit 7)	StarLAN
Byte 13	DCR SLOT NUMBER (Bits 0–5)	DCR Slot Number
	FDX (Bit 6)	Full Duplex
	DCR (Bit 7)	Deterministic Collision Resolution
Byte 14	DCR NUMBER OF STATIONS (Bits 0–5)	DCR Number of Stations
	MULIA (Bit 6)	Multiple Individual Address
	DISBOF (Bit 7)	Disable Backoff
Byte 15	DPLL INT LN (Bits 0–2)	DPLL Integrator Length
	MC ALL (Bit 3)	Receive All Multicast Frames
	MONITOR (Bits 4–5)	Monitor
	CNT FUNC (Bit 6)	Counter Function
	MON INT (Bit 7)	Disable Monitor Interrupt
Byte 16	FIFO BANK (Bits 0–1)	FIFO Banking
	PORT 1 (Bit 2)	Enable Port 1
	ALTBO (Bit 3)	Alternate Byte Order
	CLKDIV (Bit 4)	CLK Divider
	ST LEN (Bit 5)	Status Length
	TX EOP (Bit 6)	Transmit EOP
	RX EOP (Bit 7)	Receive EOP

A Configure-Done interrupt is issued when the operation is done unless ABORT is executed during the parameter transfer.

9.1.5 MC-SETUP (03)

This operation clears and sets up a new 64-bit Multicast Address hash table. The parameter block may include a number of MC-addresses (see Figures 9-6 and 9-7). Each MC Address in the list contributes to one bit in the 64-bit Hash Table.

The first byte of the nth MC address in the parameter block begins at byte number n x AL + 1, where AL is the Address Length configuration parameter and n is an integer counting from zero. Note that the least significant bit in the first byte of each MC address must be a one.

The number of MC addresses that are set up during the operation is equal to Byte Count divided by AL, rounded down, i.e., if the last MC address is incomplete it is ignored.

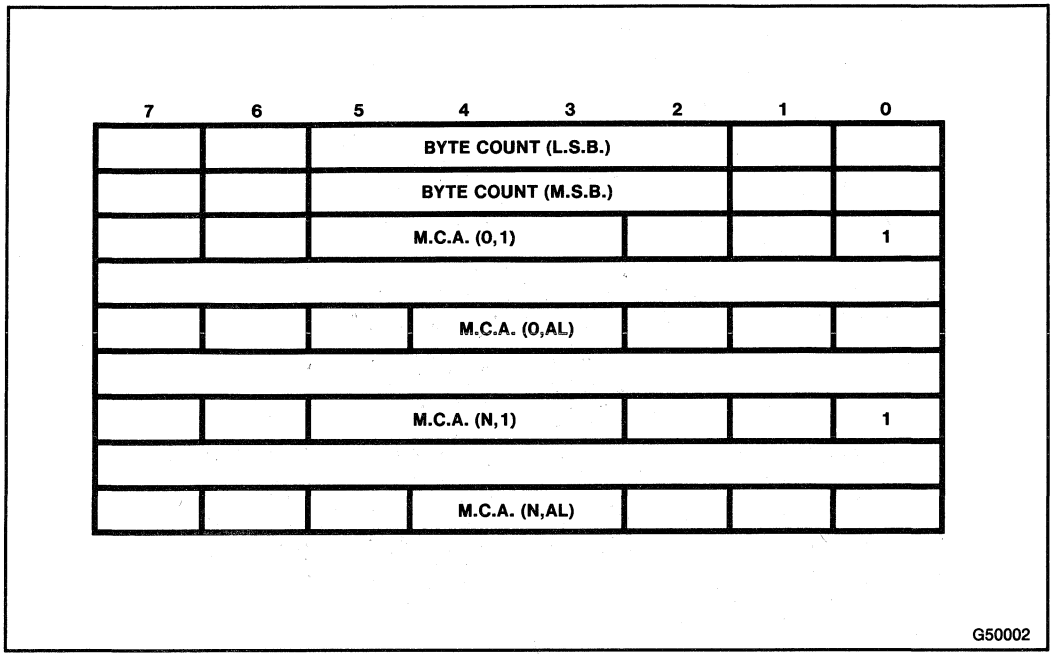


Figure 9-6. MC-SETUP Parameter Format for the 82590 or the 82592 in 8-Bit Mode

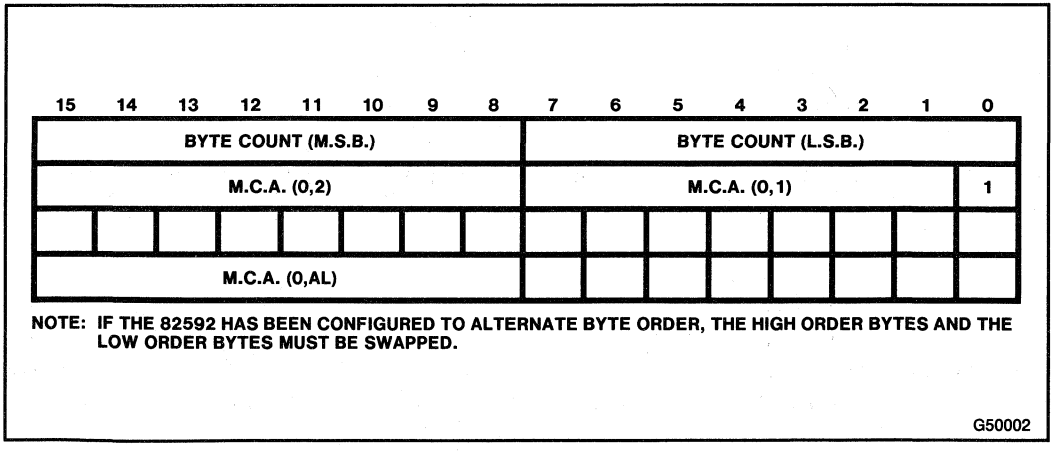


Figure 9-7. MC-SETUP Parameter Format for the 82592 in 16-Bit Mode

When the 82590 is configured to recognize multiple Individual Addresses (Mulit-IA), the MC-Setup command is also used to set up the Hash Table for the Individual Addresses. Note that the least significant bit in the first byte of each IA address must be a zero.

The MC-Setup-Done interrupt is issued after completing this operation unless ABORT is executed during the parameter transfer.

9.1.6 TRANSMIT (04)

This operation transmits one frame. The parameter block includes the following parameters:

- **Destination Address**—The length is determined by the Address Length configuration parameter.
- **Source Address**—This field is needed if the No Source Address Insertion configuration parameter is one. The length is determined by the Address Length configuration parameter.
- **Information (Length)**—The first two bytes of the information field are defined as the Length field by the IEEE 802.3 Standard. If the Length Field configuration is enabled, the 82590 treats these two bytes as the byte count of the following LLC Data Field.
- **Information (LLC Data)**—The length is determined by the Byte Count minus Destination Address minus Source Address (if any) minus CRC (if any).
- **CRC**—This field is needed if the No-CRC-Insertion configuration parameter is one. The length is determined by the CRC-16/32 parameter.

A typical transmit operation parameter block includes the Destination Address and Information fields as shown in Figure 9-8 for the 82590 or the 82592 in 8-bit mode.

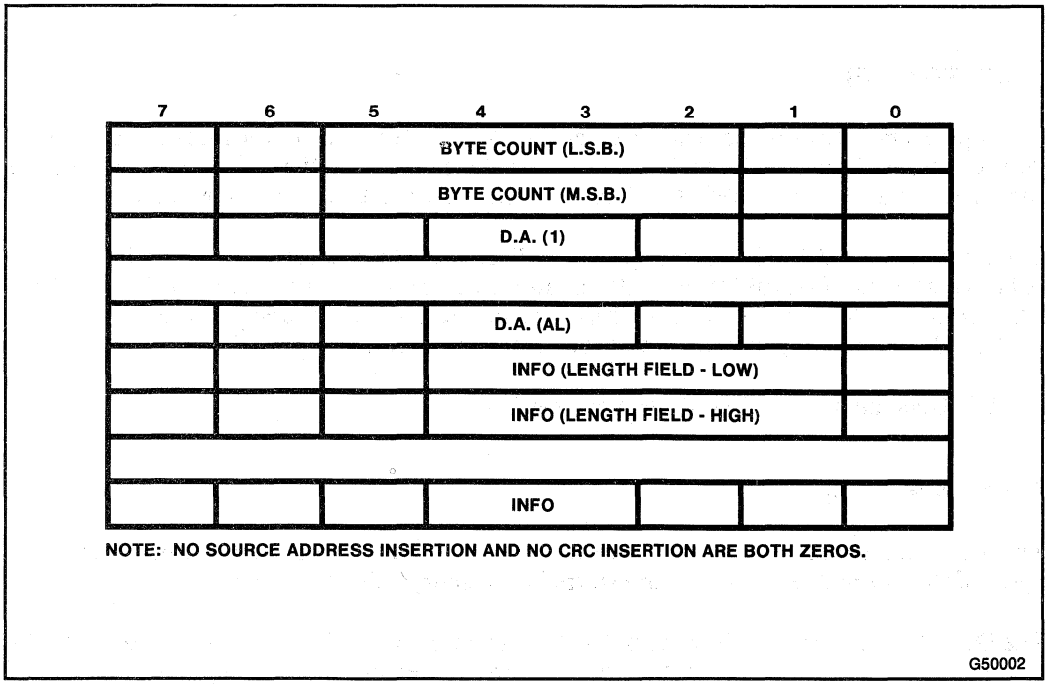


Figure 9-8. Transmit Parameter Format for the 82590 or the 82592 in 8-Bit Mode

The transmit operation will either complete the execution or be aborted by a specific ABORT operation. A Transmit-Done or Execution Aborted interrupt is issued upon completion of this operation.

If the Auto Transmit configuration is enabled, the 82590 automatically retransmits the frame whenever the collision is detected during preamble transmission. Retransmission is attempted until the maximum retry count is reached.

If the transmit EOP configuration is enabled, the $\overline{EOP}$ signal is asserted when the 82590 detects a collision during transmission. Together with the DMA controller programmed for Auto-Initialize mode, the 82590 automatically retransmits the frame until the maximum retry count is reached.

When the Tightly Coupled Interface is enabled, the 82590 can transmit consecutive frames without having the CPU issue the Transmit command each time. The CPU can place multiple transmit frames in memory, with each frame separated from the next by a Transmit command byte. The command byte immediately follows the last byte of the transmit frame. It must contain one of two commands: Transmit (04H) or NOP (00H). Refer to Figure 9-9 and 9-10.

During the DMA cycle the 82590 loads the command byte which follows the last data byte of a frame. If the command byte contains the Transmit command, the 82590 will behave as if another Transmit command has been issued by the CPU, and will repeat the frame transmission as soon as the deferring time expires. A NOP in the command byte indicates that the preceding frame was the last frame to be transmitted and no other frames follow.

9.1.7 TDR (05)

This operation activates the Time Domain Reflectometer, which is a mechanism used to detect open or short circuits on the link. The distance between the station and the open or short circuits is also measured. The TDR command does not require any parameters.

The bit stream used for TDR is a special TDR frame which consists of: Preamble, Source Address, a series of 01101110 bit pattern, and a jam pattern. The maximum length of the TDR frame is 2048 bits. If a collision is detected during transmission of the TDR frame, the 82590 transmits a jam pattern and then stops the transmission.

A TDR-Done interrupt is issued upon completion.

9.1.8 DUMP (06)

This operation causes dumping of internal registers. There are no parameters. Sixty-nine bytes of the internal registers are transferred to memory using the assigned channel.

The DUMP operation will either complete the execution or be aborted by a specific ABORT operation. A DUMP-DONE or Execution Aborted interrupt is issued upon completion of this operation.

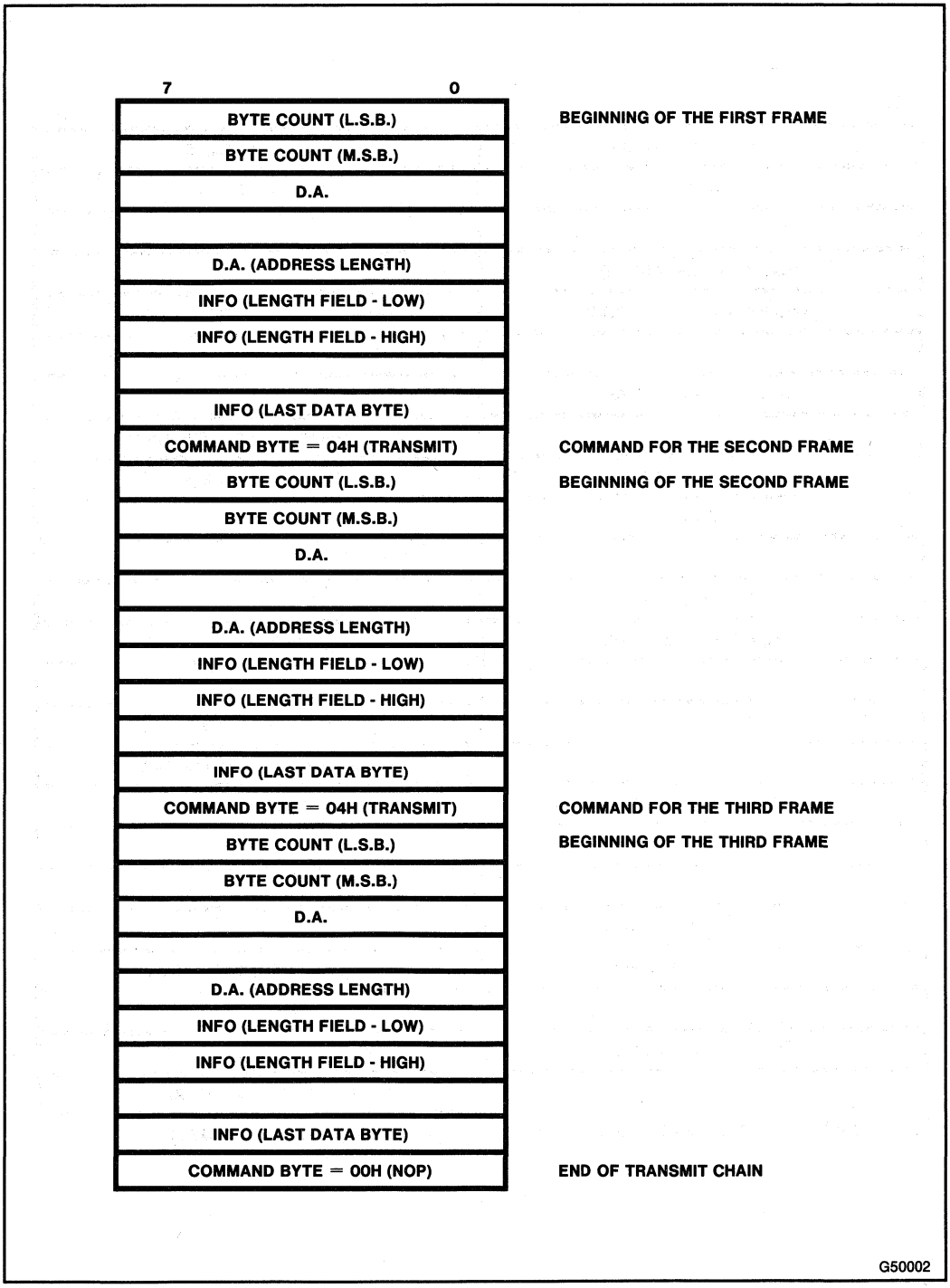


Figure 9-9. Transmit Chaining Data Structure for the 82590 or the 82592 in 8-Bit Mode

15 8	7 0
BYTE COUNT (M.S.B.)	BYTE COUNT (L.S.B.)
D.A.	D.A.
D.A. (ADDRESS LENGTH)	
INFO (LENGTH FIELD - HIGH)	INFO (LENGTH FIELD - LOW)
INFO (LAST DATA BYTE)	
	COMMAND BYTE = 04H (TRANSMIT)
BYTE COUNT (M.S.B.)	BYTE COUNT (L.S.B.)
D.A.	D.A.
D.A. (ADDRESS LENGTH)	
INFO (LENGTH FIELD - HIGH)	INFO (LENGTH FIELD - LOW)
	INFO (LAST DATA BYTE)
	COMMAND BYTE = 04H (TRANSMIT)
BYTE COUNT (M.S.B.)	BYTE COUNT (L.S.B.)
D.A.	D.A.
D.A. (ADDRESS LENGTH)	
INFO (LENGTH FIELD - HIGH)	INFO (LENGTH FIELD - LOW)
INFO (LAST DATA BYTE)	
	COMMAND BYTE = 04H (TRANSMIT)

NOTE: THE SECOND FRAME HAS AN ODD NUMBER OF INFORMATION BYTES IN THIS EXAMPLE.

G50002

Figure 9-10. Transmit Chaining Data Structure for the 82592 in 16-Bit Mode

9.1.9 DIAGNOSE (07)

The DIAGNOSE operation tests the following internal timers and shift registers:

- Exponential Backoff Random Number Generator
- Exponential Backoff Time-out Counter
- Slot Time Period Counter
- Collision Number Counter
- Exponential Backoff Shift Register
- Exponential Backoff Mask Logic
- Time Trigger Logic

9.1.10 RETRANSMIT (12)

This operation is almost identical to the TRANSMIT operation. The only difference is that RETRANSMIT does not reset the retry counter and the exponential backoff mechanism.

The Retransmit operation will either complete the Execution or be aborted by a specific ABORT operation. A RETRANSMIT-DONE or Execution Aborted interrupt is issued upon completion of this operation.

9.1.11 ABORT (13)

This operation attempts to abort execution of the current operation. It is valid only for: IA-SETUP, CONFIGURE, MC-SETUP, TRANSMIT, RETRANSMIT, and DUMP. In order to abort any of these operations successfully, the command must be given to the 82590 before the parameter transfer is completed.

9.1.12 RCV-ENABLE (8)

This operation enables reception. It is ignored if the state of the Receive Unit is not Idle.

The 82590 receives only frames that pass the address filtering. The frame and its status are placed in memory (by the DMA Controller or CPU) and the byte count of the frame is placed in the Result Registers. The completion of frame reception causes an End-of-Frame event.

9.1.13 ASSIGN ALTERNATE BUFFER (9)

This operation assigns an alternate buffer. To assign a new receive buffer, the 82590 must be in the chaining mode. The command is accepted only when the state of the Receive Unit is not IDLE and when there are less than two channels available.

This operation may also be issued when an execution is in progress and the DMA channel is transferring command parameters. In this case, the ASSIGN ALT BUFFER command causes the execution to be aborted (the same as ABORT operation).

This operation has no parameters and no results. It is usually issued by the CPU following a Request Alternate Buffer event.

9.1.14 RCV-DISABLE (10)

This operation disables the reception. If a frame reception is already in progress when the command is issued, the reception is immediately stopped and a Reception Aborted interrupt is generated. It has no parameters and no results.

9.1.15 STOP-RCV (11)

This operation disables the reception gracefully. If a frame reception is already in progress, the reception is continued until the end of the frame. If the reception is not in progress, then this command acts like the RCV-DISABLE command.

9.1.16 RESET (14)

This command resets the chip. It has the same effect as a hardware reset.

9.1.17 FIX PTR (1, 15)

This operation stops the cyclic progress of the read pointer which points to one of the Port 0 STATUS registers. The pointer is assigned to the register specfied in the Pointer Field of the Command.

The channel bit must be set to one. After this command, all subsequent read cycles are performed at the selected register. Note that STATUS 3 is not updated by the 82590 when it is pointed to by the read pointer.

9.1.18 RLS PRT (0, 15)

This operation releases the assignment of the read pointer to a specific status register and restores the cyclic progress.

The channel bit must be set to zero. This command with the pointer specifying STATUS 0 is executed automatically on reset.

9.2 ILLEGAL COMMANDS

9.2.1 Illegal Executions

9.2.1.1 NONPARAMETRIC EXECUTIONS (TDR, DIAGNOSE, ABORT)

TDR and Diagnose will be rejected if the Execution Unit is not idle.

Abort is rejected whenever it is issued under the following conditions:

1. Execution Unit is not active.
2. A nonparametric command is being executed.
3. Transfer of parameters has already been completed.

9.2.1.2 PARAMETRIC EXECUTIONS

Will be rejected if one of the following exists:

1. Execution Unit is not idle.
2. The execution channel equals the receive channel, and the Receive Unit is not idle. The Receive Unit is configured for nonchaining mode or has at least one channel assigned to it.
3. The 82590 is configured for chaining and the Receive Unit is either active or has been assigned both channels.
4. Retransmit if Tx EOP configure bit is on.

9.2.2 Illegal RCV Commands

9.2.2.1 RECEIVE ENABLE

Will be rejected if one of the following exists:

1. Receive Unit is not IDLE.
2. The RCV Channel equals the execution channel, and the Execution Unit is already performing a parametric execution.

9.2.2.2 ASSIGN ALTERNATE BUFFER

Will be rejected if one of the following exists:

1. The 82590 is not configured for Buffer Chaining.
2. The Receive Unit is in IDLE state.
3. An alternate buffer is already assigned.

9.2.2.3 RECEIVE DISABLE AND STOP RECEIVE

Will be rejected if the RCV Machine is in IDLE state.

9.3 EVENT STATUS

This section specifies Port 0 status that indicates the occurrence of Port 0 events. The status information is always accompanied by an interrupt bit.

9.3.1 IA-SETUP-DONE (01)

This event indicates the completion of an IA-SETUP operation.

9.3.2 CONFIGURE-DONE (02)

This event indicates the completion of a CONFIGURE operation. The first result register, RESULT 1, provides information about the component type and stepping. The format is shown in Figure 9-11.

The fields are interpreted as:

- **STEP NUMBER**—Stepping number of the component. The A-0 Stepping is zero.
- **82590**—The component configured is the 82590.
- **82592 (8-Bit)**—The component configured is the 82592 in 8-bit mode.
- **82592 (16-Bit)**—The component configured is the 82592 in 16-bit mode.

9.3.3 MC-SETUP-DONE (03)

This event indicates the completion of an MC-SETUP operation.

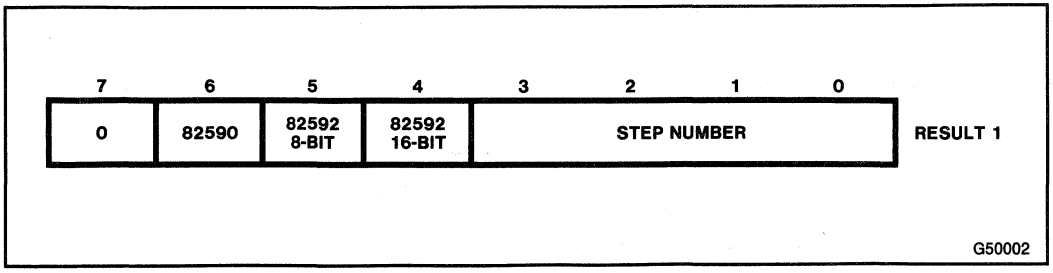


Figure 9-11. Configure Result Format

9.3.4 TRANSMIT-DONE (04)

This event indicates the completion of the TRANSMIT operation. The two result registers provide status information of the execution. Formats of the result registers are shown in Figure 9-12.

The fields are interpreted as:

- **NO. OF COLLISIONS**—Number of collisions the frame has experienced (modulo 16).
- **MAX COLL**—In Non-DCR mode: Retransmission attempts repeatedly failed due to collisions. The configured number of retries is exhausted.

  In DCR mode: A collision is detected during the collision resolution period.
- **HRT BEAT**—Collision detect test passed after the previous transmission.
- **FRTL** (*)—The frame was too long (more than 24 times the Slot Time) and caused the jabber mechanism to stop the transmission.
- **TX DEF**—Transmit deferred due to link activity.
- **LTCOL** (*)—A late collision (a collision after the Slot Time is elapsed) is detected.
- **UNDERRUN** (*)—Indicates that the DMA did not keep up with the transmission data rate.
- **LOST CTS** (*)—Clear to Send was lost during transmission.
- **LOST CRS** (*)—Carrier Sense was lost during transmission, or was not activated by the end of preamble transmission. The transmit on NoCRS (Parameter 30, Section 6.3) feature does not affect status reporting of this bit.
- **TX OK**—Transmission executed OK.
- **COLL** (*)—The last transmission experienced a collision.

If one of the bits marked (*) is set, the TX-OK bit is reset.

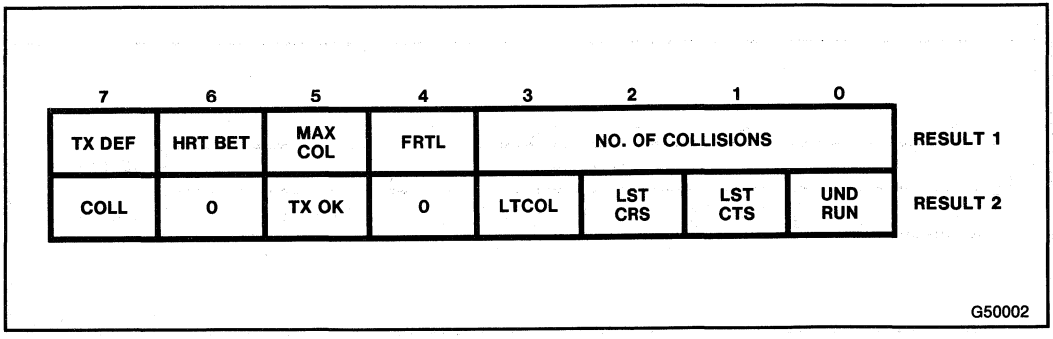


Figure 9-12. Transmit Result

9.3.5 TDR-DONE (05)

This event indicates the completion of a TDR operation. The result registers indicate whether there is a problem, which type, and where. Figure 9-13 shows the formats of the result registers.

The fields are interpreted as:

- **TDR**—Twice the radial distance in bit times between the 82590 and the location of the problem.
- **SHORT**—Indicates there is a short circuit on the transmission line (Carrier Sense signal deasserted during TDR frame transmission).
- **OPEN**—Indicates the transmission line has an open circuit or is not properly terminated (Collision Detect signal became active during TDR frame transmission).
- **XCVR**—Indicates a transceiver problem (Carrier Sense remained inactive during entire TDR frame transmission).
- **OK**—Indicates that no problem was found.

9.3.6 DUMP-DONE (06)

This event indicates that the DUMP operation is completed. The result registers are not updated. The contents of 69 internal registers are transferred to memory using the allocated channel. Figure 9-14 shows the format of the register dump for the 82590 or the 82592 in 8-bit mode. Figure 9-15 shows the format for the 82592 in 16-bit mode.

The interpretation of the fields is as follows:

Bytes 01H to 0FH: These bytes correspond to the Configure command parameter bytes.

Bytes 10H to 15H: The Individual Address Register content.

IAR0–IAR5: IAR0 is the Individual Address least significant byte. The Individual Address is right justified in the IAR space. Unused bytes are filled with all ones.

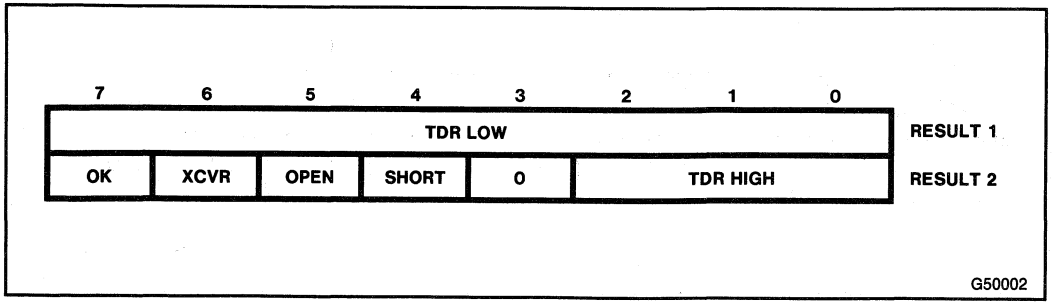


Figure 9-13. TDR Result

7	6	5	4	3	2	1	0	BYTE X/D
CHNG	SERIAL MODE	SMPLG RATE	OSC RANGE	FIFO LIMIT				01/01
BUFFER LENGTH/TCI								02/02
LOOPBACK		PREAMBLE LENGTH		NO SA INS	ADDRESS LENGTH			03/03
BOF METD	EXPONENTIAL PRIORITY			DIF MAN/$\overline{\text{MAN}}$	LINEAR PRIORITY			04/04
INTER FRAME SPACING								05/05
SLOT TIME (LOW)								06/06
MAXIMUM RETRY NUMBER				CDBBC	SLOT TIME (HIGH)			07/07
PAD	BIT STUFF	CRC16/$\overline{\text{CRC32}}$	NO CRC INSER	Tx ON NO CRS	MAN/$\overline{\text{NRZ (I)}}$	BC DIS	PRM MODE	08/08
CDTSRC	CDT FILTER			CRSSRC	CRS FILTER			09/09
MINIMUM FRAME LENGTH								0A/10
$\overline{\text{STRLAN}}$	8X/$\overline{\text{10X}}$	$\overline{\text{JABBER}}$	$\overline{\text{CDBSAC}}$	$\overline{\text{AUTOTX}}$	$\overline{\text{CRCINM}}$	$\overline{\text{LNGFLD}}$	$\overline{\text{PRECRS}}$	0B/11
DCR	FDX	DCR SLOT NUMBER						0C/12
DISBOF	MULIA	DCR NUMBER OF STATIONS						0D/13
MONINT	CNTFUN	MONITOR		MC ALL	DPLL INT LEN			0E/14
RX EOP	TX EOP	STTLEN	CLKDIV	BSWAP	PORT 1	FIFO BANK		0F/15
I.A.R 0								10/16
I.A.R 1								11/17
I.A.R 2								12/18
I.A.R 3								13/19
I.A.R 4								14/20
I.A.R 5								15/21
TX DEF	HRT BET	MAX COL	FRTL	NO. OF COLLISIONS				16/22
COLL	0	TX OK	0	LTCOL	LST CRS	LST CTS	UND RUN	17/23

G50002

Figure 9-14. Dump Result for the 82590 or the 82592 in 8-Bit Mode

TX CRC 0								18/24
TX CRC 1								19/25
TX CRC 2								1A/26
TX CRC 3								1B/27
RX CRC 0								1C/28
RX CRC 1								1D/29
RX CRC 2								1E/30
RX CRC 3								1F/31
TMPR 0								20/32
TMPR 1								21/33
TMPR 2								22/34
TMPR 3								23/35
TMPR 4								24/36
TMPR 5								25/37
SRT FRM	NO EOF	TOO LNG	1	NO SFD	NAD MCH	IA MCH	RCLD	26/38
0	0	RCV OK	LEN ERR	CRC ERR	ALG ERR	0	OVER RUN	27/39
HASHR 0								28/40
HASHR 1								29/41
HASHR 2								2A/42
HASHR 3								2B/43
HASHR 4								2C/44
HASHR 5								2D/45
HASHR 6								2E/46
HASHR 7								2F/47
X	X	X	X	X	X	X	X	30/48
O.K.	XCVR	OPEN	SHORT	X	X	X	X	31/49
X	X	X	X	X	X	X	X	32/50

G50002

Figure 9-14. Dump Result for the 82590 or the 82592 in 8-Bit Mode (Cont'd.)

1	1	1	1	1	1	X	X	33/51
RECEIVE LENGTH FIELD LOW								34/52
RECEIVE LENGTH FIELD HIGH								35/53
STTR 0								36/54
STTR 1								37/55
STTR 2								38/56
STTR 3								39/57
1	1	1	1	1	1	BUFFER COUNTER L		3A/58
BUFFER COUNTER H (FRAME COUNTER)								3B/59
RECEIVE BYTE COUNTER LOW								3C/60
RECEIVE BYTE COUNTER HIGH								3D/61
0	0	1	1	1	1	0	1	3E/62
0	1	0	1	1	0	1	0	3F/63
1	1	0	1	1	0	0	1	40/64
EXECUTION BYTE COUNTER LOW								41/65
EXECUTION BYTE COUNTER HIGH								42/66
EXECUTION DATA INPUT BUFFER 0								43/67
EXECUTION DATA INPUT BUFFER 1								44/68
EXECUTION DATA INPUT BUFFER 2								45/69

G50002

Figure 9-14. Dump Result for the 82590 or the 82592 in 8-Bit Mode (Cont'd.)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	WORD X/D
BUFFER LENGTH								CHN	SER MO	SMP RAT	OSC RAN	FIFO LIMIT				01/01
BOF MET	EXPONENTIAL PRIORITY			MN/ $\overline{DF}$	LINEAR PRIORITY			EXT L.B	INT L.B	PREAMBL LENGTH		NSR ADD	ADDRESS LENGTH			02/02
SLOT TIME - LOW								INTER FRAME SPACING								03/03
PAD	BIT STF	CRC 16	NO CRC	TxO NCR	MAN $\overline{NRZ}$	BC DIS	PRM MOD	MAX RETRY NUM.				CDB BC	SLT HIGH			04/04
MINIMUM FRAME LENGTH								CDT SRC	CLD FILTER			CRS SRC	CRS FILTER			05/05
DCR	FDX	DCR SLOT NUMBER						$\overline{STR}$ $\overline{LAN}$	8X/ $\overline{10X}$	$\overline{JAB}$ $\overline{BER}$	$\overline{CDB}$ $\overline{SAC}$	$\overline{AUT}$ $\overline{OTX}$	$\overline{CRC}$ $\overline{INM}$	$\overline{LNG}$ $\overline{FLD}$	$\overline{PRE}$ $\overline{CRS}$	06/06
RCV I_D	CNT FUN	MONITOR		MC ALL	DPLL INT LN			DIS BOF	MUL IA	DCR NUMBER OF STATIONS						07/07
0	0	0	0	0	0	0	0	RX EOP	TX EOP	STT LEN	CLK DIV	BSW AP	POR T1	FIFO BANK		08/08
I.A.R. 1								I.A.R. 0								09/09
I.A.R. 3								I.A.R. 2								0A/10
I.A.R. 5								I.A.R. 4								0B/11
CLL	0	TX OK	0	0	LST CRS	LST CTS	UND RUN	TX DEF	HRT BET	MAX COL	0	NUM. OF COLLIS.				0C/12
TX CRC 1								TX CRC 0								0D/13
TX CRC 3								TX CRC 2								0E/14
RX CRC 1								RX CRC 0								0F/15
RX CRC 3								RX CRC 2								10/16
TMPR 1								TMPR 0								11/17
TMPR 3								TMPR 2								12/18
TMPR 6								TMPR 4								13/19

G50002

Figure 9-15. Dump Result for the 82592 in 16-Bit Mode

0	0	RCV OK	LEN ERR	CRC ERR	ALG ERR	0	OVR RUN	SRT FRM	NO EOF	TOO LNG	1	NO SFD	NAD MCH	IA MCH	RCV CLD	14/20
HASHR 1								HASHR 0								15/21
HASHR 3								HASHR 2								16/22
HASHR 6								HASHR 4								17/23
HASHR 8								HASHR 7								18/24
OK	XCR	OPN	SRT	X	X	X	X	X	X	X	X	X	X	X	X	19/25
1	1	1	1	1	1	X	X	X	X	X	X	X	X	X	X	1A/26
RECEIVE LENGTH FIELD HIGH								RECEIVE LENGTH FIELD LOW								1B/27
1	1	1	1	1	1	1	1	STTR 0								1C/28
1	1	1	1	1	1	1	1	STTR 1								1D/29
1	1	1	1	1	1	1	1	STTR 2								1E/30
1	1	1	1	1	1	1	1	STTR 3								1F/31
1	1	1	1	1	1	1	1	1	1	1	1	1	1	B.CNT.L		20/32
1	1	1	1	1	1	1	1	BUFFER CNTER H (FRAME COUNTER)								21/33
1	1	1	1	1	1	1	1	RECEIVE BYTE COUNTER LOW								22/34
1	1	1	1	1	1	1	1	RECEIVE BYTE COUNTER HIGH								23/35
1	1	1	1	1	1	1	1	0	0	1	1	1	1	0	1	24/36
1	1	1	1	1	1	1	1	0	1	0	1	1	0	1	0	25/37
1	1	1	1	1	1	1	1	1	1	0	1	1	0	0	1	26/38
EXECUTION BYTE COUNTER HIGH								EXECUTION BYTE COUNTER LOW								27/39
EXECUTION DATA INPUT BUFFER 0																28/40
EXECUTION DATA INPUT BUFFER 1																29/41
EXECUTION DATA INPUT BUFFER 2																30/42

G50002

Figure 9-15. Dump Result for the 82592 in 16-Bit Mode (Cont'd.)

Bytes 16H to 17H: Transmit status registers. These two bytes are status bytes of the last Transmit or Retransmit command.

Bytes 18H to 1BH: Content of the Transmit CRC generator. TX CRC 0 is the least significant byte. The content depends on the most recently performed activity prior to the Dump command.

1. After RESET—All 1's.
2. After successful transmission—All 0's.
3. After MC-SETUP—the CRC value calculated from the last MC address in the MC address list.
4. After unsuccessful transmission, the value depends on where the transmission was stopped:
   Stopped during Preamble—All 1's.
   Stopped after Preamble—according to the CRC of the frame transmitted.

   Note: For 16-bit CRC only TX CRC 0 and TX CRC 1 are valid.

Bytes 1CH to 1FH: Content of the receive CRC checker. RX CRC 0 is the least significant byte. The content depends on the most recently performed activity prior to the Dump command.

Bytes 20H to 25H: Temporary Registers, TMPR O to TMPR 5. These six bytes contain the last six bytes of a received frame including the FCS (Frame Check Sequence). These bytes are, however, not in the order they are received. In order to arrange these bytes in the received order, the following fomula must be used:

$$\text{BYTE}(k) = \text{TMPR}[(1 + k + 5) \bmod 6],$$

where 1 is the number of bytes in the frame, k is the byte index (0 for the last byte of the frame and -5 for the 6th byte from the last).

Bytes 26H to 27H: Receive Status Register. These two bytes are the status bytes of the most recently received frame.

Bytes 28H to 2F: Hash Table.

Byte 31H: The TDR status registers. After a reset, the XCVR, OPEN, SHORT bits are 0 and the OK bit is 1.

Bytes 34H to 35H: Content of the 82590 receive length field counter. It usually contains zeros, but if the frame was truncated, it is different from all zeros.

Bytes 36H to 39H: Contents of the 82590 status registers.

Bytes 3AH to 3BH: Content of the receive buffer counter.

Bytes 3CH to 3DH: Content of the receive byte counter.

Bytes 41H to 42H: Content of the execution byte counter.

Bytes 43H to 45H: Content of the input data buffers.

The interpretation of the fields is as follows:

Words 01H to 08H: These Words correspond to the Configure command parameter bytes.

Words 09H to 0BH: The Individual Address Register content.

IAR0–IAR5: IAR0 is the Individual Address least significant byte. The Individual Address is right justified in the IAR space. Unused bytes are filled with all ones.

Word 0CH: Transmit status registers. These two bytes are status bytes of the last Transmit or Retransmit command.

Words 0DH to 0EH: Content of the Transmit CRC generator. TX CRC 0 is the least significant byte. The content depends on the most recently performed activity prior to the Dump command.

1. After RESET—All 1's.
2. After successful transmission—All 0's.
3. After MC-SETUP—the CRC value calculated from the last MC address in the MC address list.
4. After unsuccessful transmission, the value depends on where the transmission was stopped:
   Stopped during Preamble—All Ones.
   Stopped after Preamble—according to the CRC of the frame transmitted.

   NOTE: For 16-bit CRC only TX CRC 0 and TX CRC 1 are valid.

Words 0FH to 10H: Content of the receive CRC checker. RX CRC 0 is the least significant byte. The content depends on the most recently performed activity prior to the Dump command.

Words 11H to 13H: Temporary Registers, TMPR O to TMPR 5. These six bytes contain the last six bytes of a received frame including the FCS (Frame Check Sequence). These bytes are, however, not in the order they are received. In order to arrange these bytes in the received order, the following fomula must be used:

$$\text{BYTE}(k) = \text{TMPR}[(1 + k + 5) \bmod 6],$$

where 1 is the number of bytes in the frame, k is the byte index (0 for the last byte of the frame and -5 for the 6th byte from the last).

Word 14H: Receive Status Register. These two bytes are the status bytes of the most recently received frame.

Words 15H to 18H: Hash Table.

Word 19H: The TDR status registers. After a reset, the XCVR, OPEN, SHORT bits are zero and the OK bit is one.

Word 1BH: Content of the 82592 receive length field counter. It usually contains zeros, but if the frame was truncated, it is different from all zeros.

Words 1CH to 1FH: Content of the 82592 status registers.

Word 20H: Content of the receive buffer counter (low).

Word 21: Content of the Receive Buffer counter (high) or the frame counter in TCI mode.

Words 22H and 23H: Content of the receive byte counter.

Word 27H: Content of the execution byte counter.

Words 28H to 30H: Content of the input data buffers.

9.3.7 DIAGNOSE-PASSED (07)

This event indicates that the self-test procedure was completed successfully.

9.3.8 End of Frame

This event indicates the completion of a frame reception. The frame is stored in memory as shown in Figure 9-16, 9-17, or 9-18, depending on the component type and configuration. When configured to store the CRC bytes in memory, the 82590 places the CRC bytes between the information field and the status bytes. If the CRC bytes are to be stored in memory, the status bytes immediately follow the information field. When configured for the Tightly Coupled Interface (TCI) mode, the 82590 appends two bytes of the byte count field. The byte count field contains the number of bytes that are transferred in memory, not including the byte count field itself. When the 82592 in 16-bit mode is configured for Byte Swapping, the high and low order bytes are swapped. The low order bytes are transferred on the D8 through D15 lines and the high order bytes on the D0 through D7 lines.

The status bits are interpreted as follows:

- **NO EOF**—No End of Frame flag (in bitstuffing only).
- **SRT FRM**—Received frame is shorter than Minimum Frame Length parameter.
- **TOO LNG**—Received frame was longer than 24 times the Slot Time and was truncated upon jabber timeout. This bit is valid only when the jabber configuration is enabled.
- **NO SFD**—No SFD detected. This bit is valid only in Monitor mode.
- **NAD MCH**—No Address Match. This bit indicates that the frame monitored did not pass any of the address filtering methods. This bit is valid only in Monitor mode.
- **$\overline{\text{IA MCH}}$**—The destination address of the received frame matched the station's individual address.
- **RCV CLD**—A collision is detected during reception.
- **OVERRUN**—The DMA did not keep up with incoming bits.

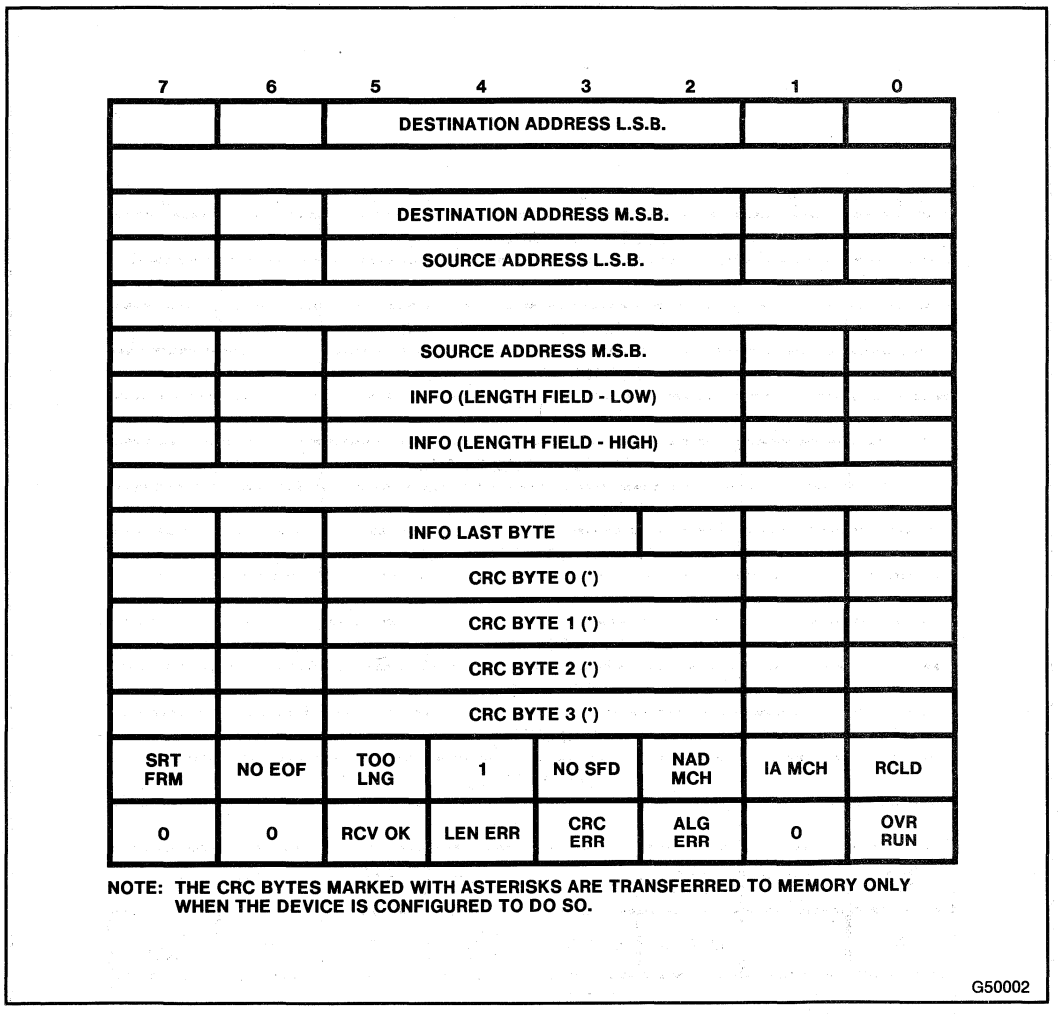


Figure 9-16. Receive Format for the 82590 or the 82592 in 8-Bit Mode (Tightly Coupled Interface Not Enabled)

- **ALG ERR**—Alignment error.
- **CRC ERR**—CRC error.
- **LEN ERR**—Length error. This bit is valid only when the 82590 is configured to use the length field.
- **RCV OK**—Frame received OK.

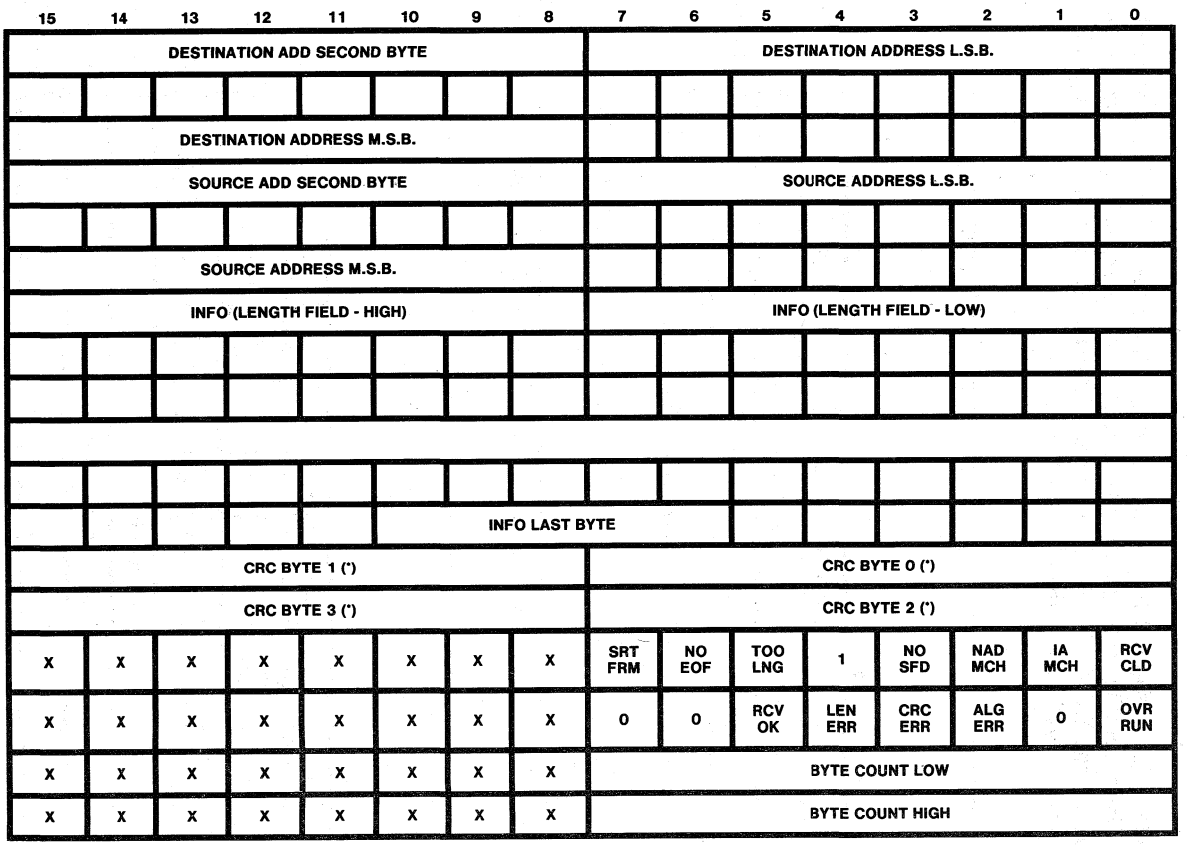
When the 82590 is configured to have four bytes of status registers, STATUS 1 and STATUS 2 store the byte count of the received frame (see Figure 9-19). These registers contain the number of bytes that are transferred to memory, from the beginning of the destination address field to the end of the status field.

7	6	5	4	3	2	1	0
		DESTINATION ADDRESS L.S.B.					
		DESTINATION ADDRESS M.S.B.					
		SOURCE ADDRESS L.S.B.					
		SOURCE ADDRESS M.S.B.					
		INFO (LENGTH FIELD - LOW)					
		INFO (LENGTH FIELD - HIGH)					
		INFO LAST BYTE					
		CRC BYTE 0 (*)					
		CRC BYTE 1 (*)					
		CRC BYTE 2 (*)					
		CRC BYTE 3 (*)					
SRT FRM	NO EOF	TOO LNG	1	NO SFD	NAD MCH	IA MCH	RCLD
0	0	RCV OK	LEN ERR	CRC ERR	ALG ERR	0	OVR RUN
BYTE COUNT LOW							
BYTE COUNT HIGH							

NOTE: THE CRC BYTES MARKED WITH ASTERISKS ARE TRANSFERRED TO MEMORY ONLY WHEN THE DEVICE IS CONFIGURED TO DO SO.

G50002

Figure 9-17. Receive Format for the 82590 or the 82592 in 8-Bit Mode (Tightly Coupled Interface Enabled)



NOTE: THE CRC BYTES MARKED WITH ASTERISKS ARE TRANSFERRED TO MEMORY ONLY WHEN THE DEVICE IS CONFIGURED TO DO SO.

G50002

Figure 9-18. Receive Format for the 82592 in 16-Bit Mode (Tightly Coupled Interface Enabled, Byte Swap Not Enabled)

When the 82590 is configured to have six bytes of status registers, status of the received frame is reported in the additional two registers (see Figure 9-20). This configuration allows the CPU to read the status of reception directly from the on-chip registers.

When the 82590 is configured for Tightly Coupled Interface, STATUS 2 is used as the Frame Count field (see Figure 9-21). It contains the number of frames that have been received from the moment the RCV-ENABLE command was executed. It counts both good and bad frames. The value is valid only when the valid bit is set.

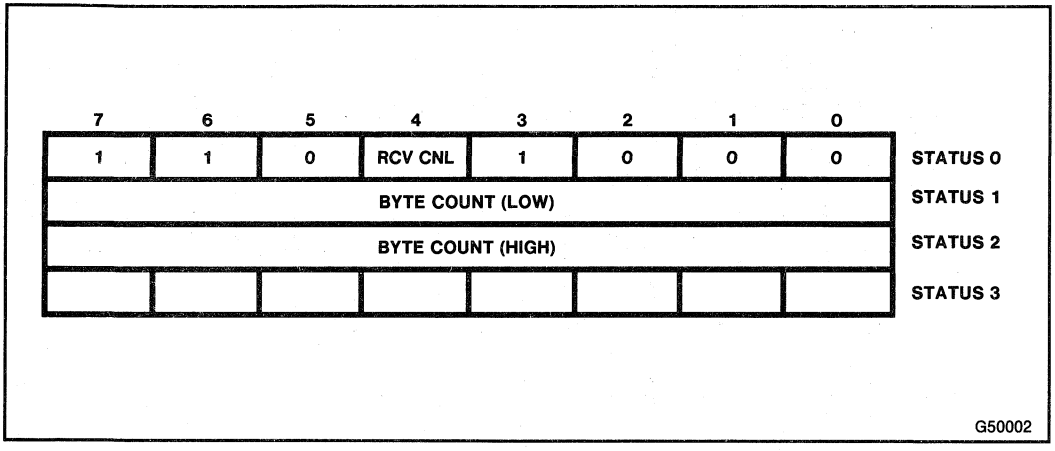


Figure 9-19. Receive Status (4-Byte Status Configuration)

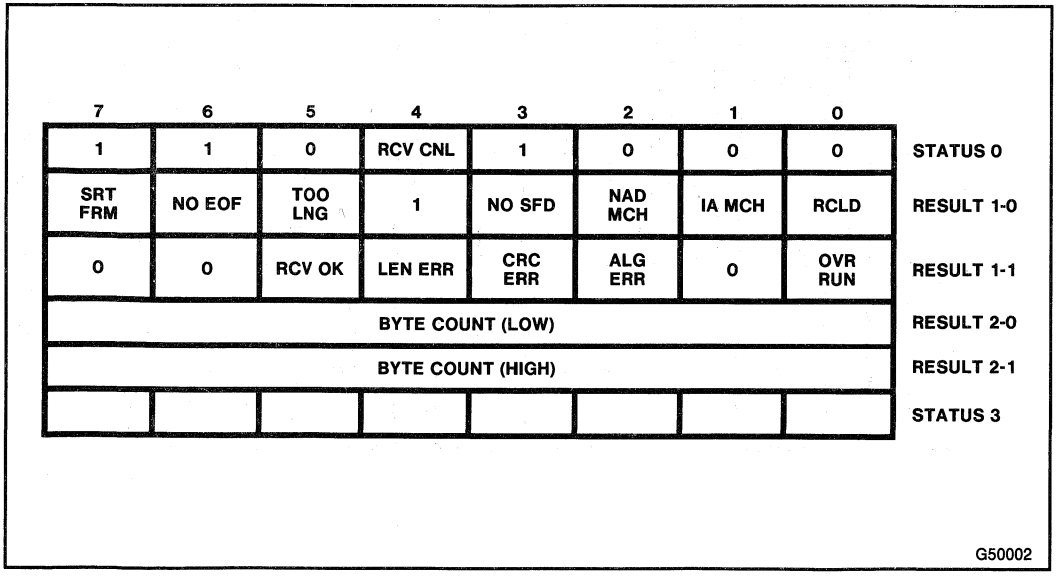


Figure 9-20. Receive Status (6-Byte Status Configuration)

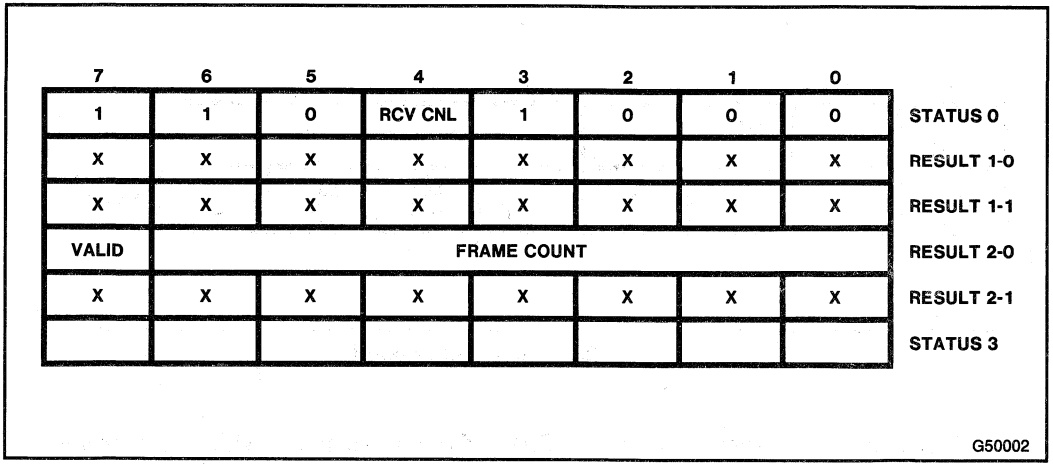


Figure 9-21. Receive Status in Tightly Coupled Interface Mode

9.3.9 Request Alternate Buffer (9)

This event is issued in Chaining mode when an additional alternate buffer is needed during a frame reception.

9.3.10 Reception Aborted (10)

This event is issued as a result of a RCV-DISABLE operation that causes part of a frame to be discarded.

9.3.11 RETRANSMIT-DONE (~~11~~ 12)

This event indicates that the RETRANSMIT operation was completed. The format of the result registers is identical to that of the TRANSMIT-DONE, except that the No. of Collisions field provides meaningful information. See Figure 9-22.

The fields are interpreted as:

- **NO. OF COLLISIONS**—Number of collisions the frame has experienced (modulo 16).
- **MAX COLL**—In Non-DCR mode: Retransmission attempts repeatedly failed due to collisions. The configured number of retries is exhausted.

  In DCR mode: A collision is detected during the collision resolution period.
- **HRT BEAT**—Collision detect test passed after the previous transmission.
- **FRTL (*)**—The frame was too long (more than 24 times the Slot Time) and caused the jabber mechanism to stop the transmission.
- **TX DEF (*)**—Transmit deferred due to link activity.
- **LTCOL (*)**—A late collision (a collision after the Slot Time is elapsed) is detected.

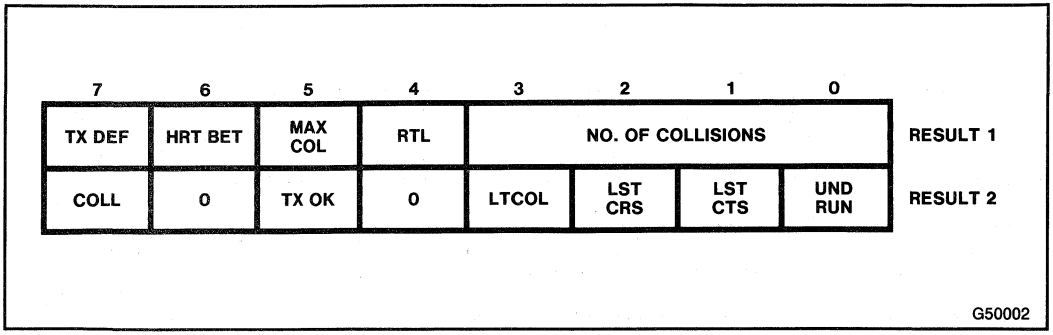


Figure 9-22. Retransmit Result

- **UNDERRUN** (*)—Indicates that the DMA did not keep up with the transmission data rate.
- **LOST CTS** (*)—Clear to Send was lost during transmission.
- **LOST CRS** (*)—Carrier Sense was lost during transmission, or was not set until end of preamble.
- **TX OK**—Transmission executed OK.
- **COLL** (*)—The last transmission experienced a collision.

The setting of the bits marked with (*) will cause the TX OK bit to be reset.

9.3.12 Execution-Aborted (13)

This event indicates that the execution of the last operation was aborted. The reason for the abortion could be either an ABORT or an ASSIGN-ALT-BUF command.

9.3.13 Diagnose-Failed (15)

This event indicates that the self-check procedure was completed unsuccessfully.

Port 1 Operations

10

CHAPTER 10
PORT 1 OPERATIONS

This section describes the Port 1 command operations in detail. The 4-bit op-codes are shown in parentheses.

10.1 GENERAL PURPOSE OPERATIONS

10.1.1 NOP (00)

This operation does not affect the 82590. It has no parameters and no result. It is used to manipulate the pointer and/or to acknowledge interrupts.

10.1.2 SWT-TO-PORT-0 (01)

This operation returns the control of the command/status port (selected by CS0 in hardware) to Port 0.

10.1.3 SET-TS (05)

This operation puts all output pins of the 82590 in the High Impedance state.

10.1.4 RST-TS (07)

This operation releases all output pins from the High Impedance state.

10.1.5 LCL-PWR-DWN (08)

This operation places the 82590 in Local Power Down mode by internally applying a reset signal to the state machines and turning off clock signals. All configuration and setup parameters are, however, preserved internally. Once the 82590 is powered down, the internal pointer of Port 1 is fixed at STATUS 3. STATUS 3 of Port 1 is, therefore, the only register that the CPU can read during power down. Only NOP or a hardware reset signal is accepted by the 82590 during power down. When the Port 1 NOP command ($\overline{GP} = 0$) is issued, the internal clocks of the 82590 are started. Once the clocks are started, the RESET or SEL-RST command can be issued to Port 1 to power up the device.

The 82590 can be powered up in two different ways. If a hardware or software reset is used, it is completely reinitialized as it powers up. If the SEL-RST command is used, then the

82590 powers up with all of the preserved configuration and setup parameters. The 82590 does not have to be reconfigured in the latter case. The detailed steps for power up are shown below:

1. Issue a NOP command with the TC/$\overline{GP}$ bit set to zero. This starts the internal clock signals.
2. If the on-chip crystal oscillator is used in High-Integration mode, wait for 32 milliseconds. This wait time allows the clock signals to stabilize within the device.
3. Provide a hardware reset or issue a software RESET or SEL-RST command to Port 1.

10.1.6 RMT-PWR-DWN (09)

This operation places the 82590 in Remote Power Down mode by internally applying a reset signal to most of the state machines and turning off clock signals. The crystal oscillator and a part of the receive circuits remains active, so the 82590 can be powered up by receiving a certain frame. All configuration and setup parameters are preserved internally. Once the 82590 is powered down, the internal pointer is fixed at STATUS 3. STATUS 3 is, therefore, the only register that the CPU can read during power down. Only the Port 1 NOP command or a hardware reset signal is accepted by the 82590 during power down. When the Port 1 NOP command is issued, the internal clocks of the 82590 are started. Once the clocks are started, the RESET or SEL-RST command can be issued to Port 1 to power up the device.

The 82590 in Remote Power Down listens to the link. When it senses active carrier, it starts evaluating the incoming frame. In order for the frame to cause power up, it must satisfy the following requirements:

1. It must have the Individual Address of the 82590 in the Destination Address field and have a perfect address match.
2. The Length field must contain 8000 (Hexadecimal).
3. The frame must be received without errors.

When a received frame satisfies all of the above requirements, the parallel subsystem is turned on and the REM-PWR-UP interrupt is generated.

If desired, the 82590 in Remote Power Down can be powered up locally. The power up procedure is the same as the one for the local power down mode. See LCL-PWR-DWN above for details.

10.1.7 FIX-PTR (12)

This operation stops the cyclic progress of the read pointer of the Port 1 Status register bank. The pointer is assigned to the register specified in the pointer field of the command.

10.1.8 RLS-PTR (13)

This operation releases the assignment of the read pointer to a specific status register and restores the cyclic progress of the pointer.

10.1.9 RESET (14)

This command resets the chip. It has the same effect as a hardware reset.

10.1.10 SEL-RST (15)

The selective reset operation resets the 82590 while preserving its configuration and setup parameters as well as the most recent contents of the status registers.

10.2 TIMER/COUNTER OPERATIONS

10.2.1 NOP (00)

This operation does not affect the 82590. It has no parameters and no result. It is used to manipulate the pointer and/or to acknowledge general purpose (GP) interrupts.

10.2.2 START (01)

This operation starts the timer or event counter specified in the pointer field. In case of the timer, it is loaded with the maximum value and then decremented by the timer clock. In case of an event counter, it is loaded with zero and then incremented whenever its responsible event occurs. This command can be issued regardless of the current timer or event counter state.

10.2.3 STOP (02)

This operation stops the timer or event counter specified in the pointer field. The Count register of the timer or event counter stops counting and holds the last value that was counted. If the state of the timer or event counter is already IDLE or STOPPED, this command has no effects.

10.2.4 RESUME (03)

This operation restarts the timer or event counter specified in the pointer field. Counting is resumed from the value contained in the count register. If the state of the timer is IDLE, the timer is loaded with all 1's and then started. If the state of the event counter is IDLE, the event counter is loaded with 0 and then started.

10.2.5 LD&START (05)

This is a special command which allows counter diagnostics (see Section 10.3 for details). The event counters are started after they are initialized to all 1's.

10.2.6 ACK-INT (07)

This command acknowledges the interrupt event caused by the timer or event counter specified in the pointer field. When issued with the INT-ACK bit set, this command deasserts the interrupt signal and also reset the specified event bit in STATUS 0. For the timer and event counter interrupts that are enabled by the IE bits, this command must be issued together with the INT-ACK bit set. For the timer and event counter interrupts that are not enabled, issuing this command alone only clears the specified event bit in STATUS 0.

10.2.7 COUNT (08)

Valid only when the timer and event counters are configured to External Source and are already in the RUNNING state. This command decrements the timer by one or increments the event counter by one. A specific timer or event counter is specified by the pointer field.

10.2.8 START-ALL-COUNTERS (09)

This operation starts all event counters simultaneously. The pointer field specified with this command has no effect. The event counters are all loaded with zero and then incremented individually as their responsible events occur. This command can be issued regardless of the current states of the event counters.

10.2.9 SET-VAL (10)

This operation initializes the Maximum Count register of the timer or event counter specified in the pointer field. The command requires three additional write cycles for the 24-bit timer and two for the 16-bit event counters. The least significant byte is written first. The Port 1 status remains as BUSY and cannot be used for any other operations until all required bytes are written.

10.2.10 SET-CONF (11)

This operation configures all the timer and event counters. The command requires two additional write cycles to load the configuration parameters. The second byte of configuration parameters is accepted by the 82590 if none of the timer commands have been executed before this command and after a hardware or software reset. When this command is given after a timer command is executed, the second configuration byte has no effects to the 82590. The pointer field given with this command has no effects. Figure 10-1 shows the format of the configuration parameters.

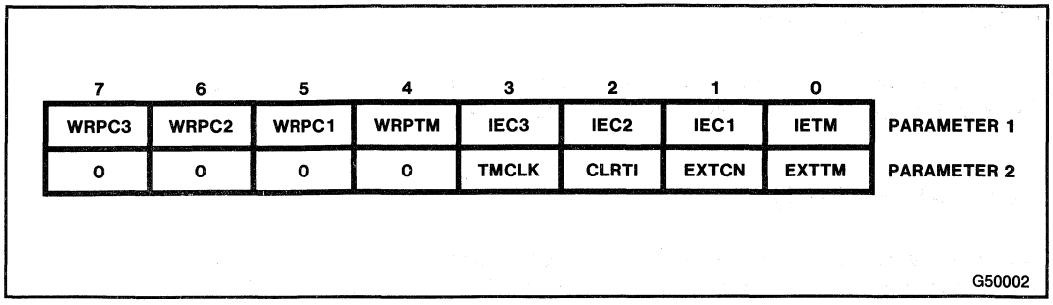


Figure 10-1. Timer/Counter Configuration Parameters

10.2.10.1 WRP(TM-C3)—PARAMETER 1, BITS 7–4

Determines each timer or event counter to operate in Wrap Around or Sticky mode. In Wrap Around mode, the timer is loaded automatically with the maximum value upon reaching zero and the event counters with zero upon reaching the maximum values. Comparison and reinitialization are done during the same timer clock or counter event; therefore, the timer does not store zero and the event counters do not store the maximum count values. In Sticky mode, the timer stops at FFFFFF (Hexadecimal) in Non-External Source mode and at 0 in External Source mode. The event counters stop at the maximum values in Sticky mode.

0—sticky

1—wrap around

Default Setting: 1

10.2.10.2 IE(TM-C3)—PARAMETER 1, BITS 3–0

Enables an interrupt event generated by the timer or event counter to assert the interrupt hardware signal. The timer generates an interrupt event when its count register reaches zero. The event counters generate interrupt events when their count registers reach the maximum values. If the IE bit is set, the interrupt signal is asserted as soon as its corresponding interrupt event is reported in STATUS 0. If the IE bit is not set, the interrupt signal is not asserted. The interrupt event is, however, reported in STATUS 0 regardless of IE bit.

0—Disable

1—Enable

Default Setting: 1

10.2.10.3 EXTTM—PARAMETER 2, BIT 0

Places the timer in External Source mode. This mode allows the timer to be decremented by the COUNT command and not by the timer clock.

0—Timer is decremented by the timer clock.

1—External Event (Timer is decremented by the COUNT command)

Default Setting: 0

10.2.10.4 EXTCN—PARAMETER 2, BIT 1

Places all the event counters in External Source mode. This mode allows the event counters to be incremented by the COUNT command and not by the network events specified by the COUNTER-FUNC bit.

0—Event Counters are incremented by the network events.

1—External Event (Event Counters are incremented by the COUNT command)

Default Setting: 0

10.2.10.5 CLRTI—PARAMETER 2, BIT 2

Must be determined according to the frequency ratio between the timer clock and the system clock. If the ratio is more than two (i.e., the frequency of the timer is more than twice the frequency of the system clock), then this bit must be 0; otherwise, it may be 0 or 1. When this bit is 0, the timer value which is read by the RD-COUNT-VAL command is updated every eight timer clocks. When this bit is one, it is updated every four timer clocks. The timer COUNT register is, however, decremented on every timer clock regardless of this bit. The bit is internally used to synchronize and coordinate the data flow between the timer COUNT register and the register which holds the value for the RD-COUNT-VAL command.

Default Setting: 0

10.2.10.6 TMCLK—PARAMETER 2, BIT 3

Determines the source of the timer clock. If this bit is zero, the timer clock is the transmit clock (TCLK in High-Integration mode and TXC in High-Speed mode). If one, the timer clock is the system clock. This bit may be changed only when the timer is in the IDLE state.

Default Setting: 0

10.2.11 RD-MAX-COUNT-VAL (12)

This operation switches the read port of Port 1 to the bank of the timer/counter maximum counter registers. The pointer field specifies the first timer or event counter to be read. The least significant byte of the timer or event counters is read first.

10.2.12 RD-COUNT-VAL (13)

This operation switches the read port of Port 1 to the bank of the timer/counter registers. The pointer field specifies the first timer or event counter to be read. The least significant byte of the timer or event counters is read first.

10.2.13 RESET (14)

This command resets the chip. It has the same affect as a hardware reset.

10.2.14 Timer/Event Counter Diagnostics

There are no explicit diagnostics commands implemented for the timer and event counters. However, the timer and event counters can be tested by configuring to External Count mode (EXTCN and EXTTM bits set) and using the COUNT and READ-VAL commands. In order to test the timer, the flowchart shown in Figure 10-2 can be used. This procedure exercises all bits of the timer register with minimum programming. Figure 10-3 shows a similar flowchart for testing event counters.

10.3 ILLEGAL COMMANDS

This section summarizes the illegal Port 1 operations.

1. LD&START for the timer is illegal and not accepted by the 82590.
2. All commands except for NOP and RESET are rejected in the power down modes.
3. The INT-ACK bit has no effects if the INT bit is not set.
4. The timer and event counter commands are not accepted unless the Port 1 (STATUS 3, Bits 7-6) is in the IDLE state.
5. The COUNT command is rejected unless the timer and event counters are configured to External Source mode.
6. All non-specified commands are illegal.

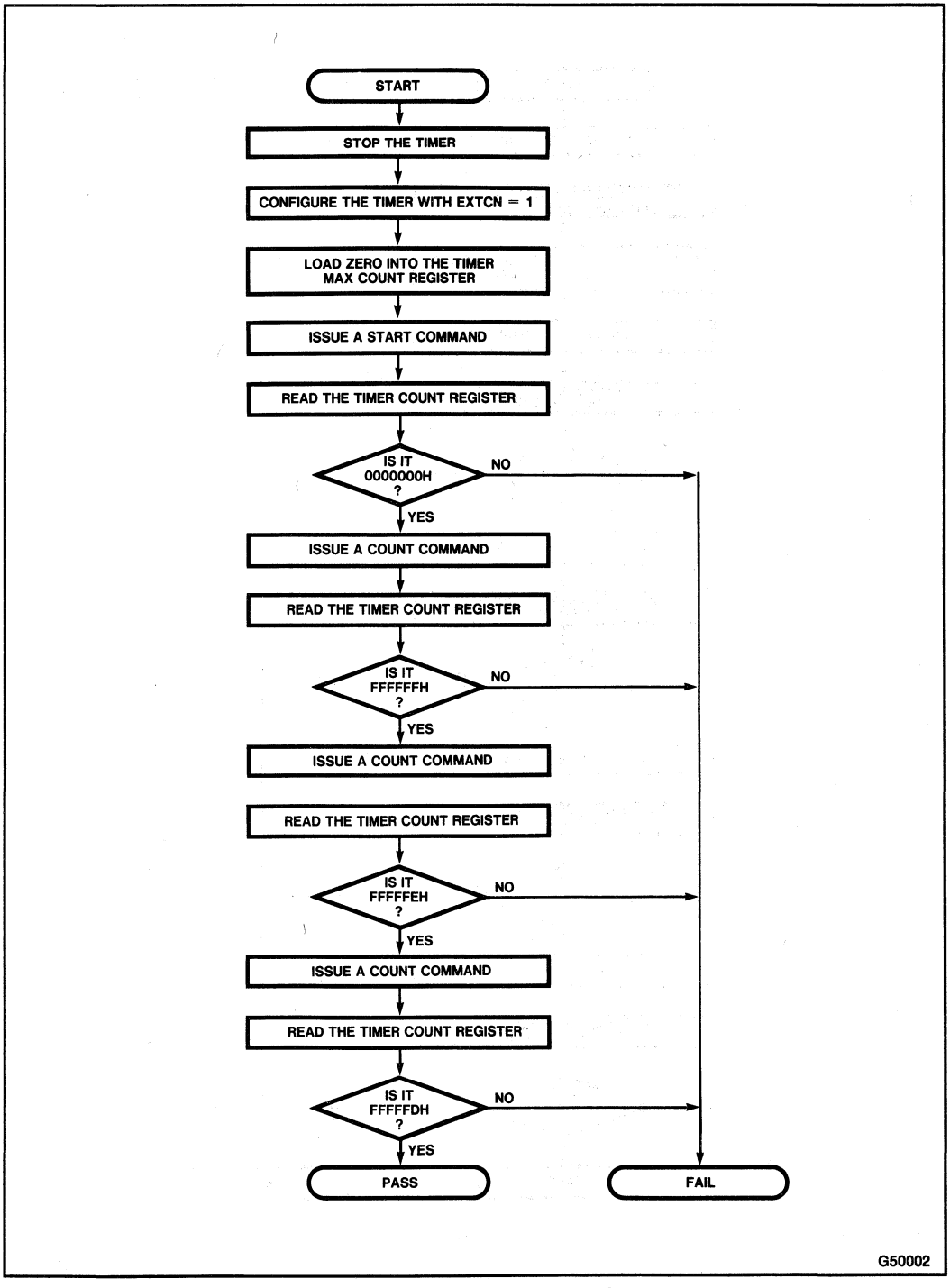


Figure 10-2. Timer Diagnostics

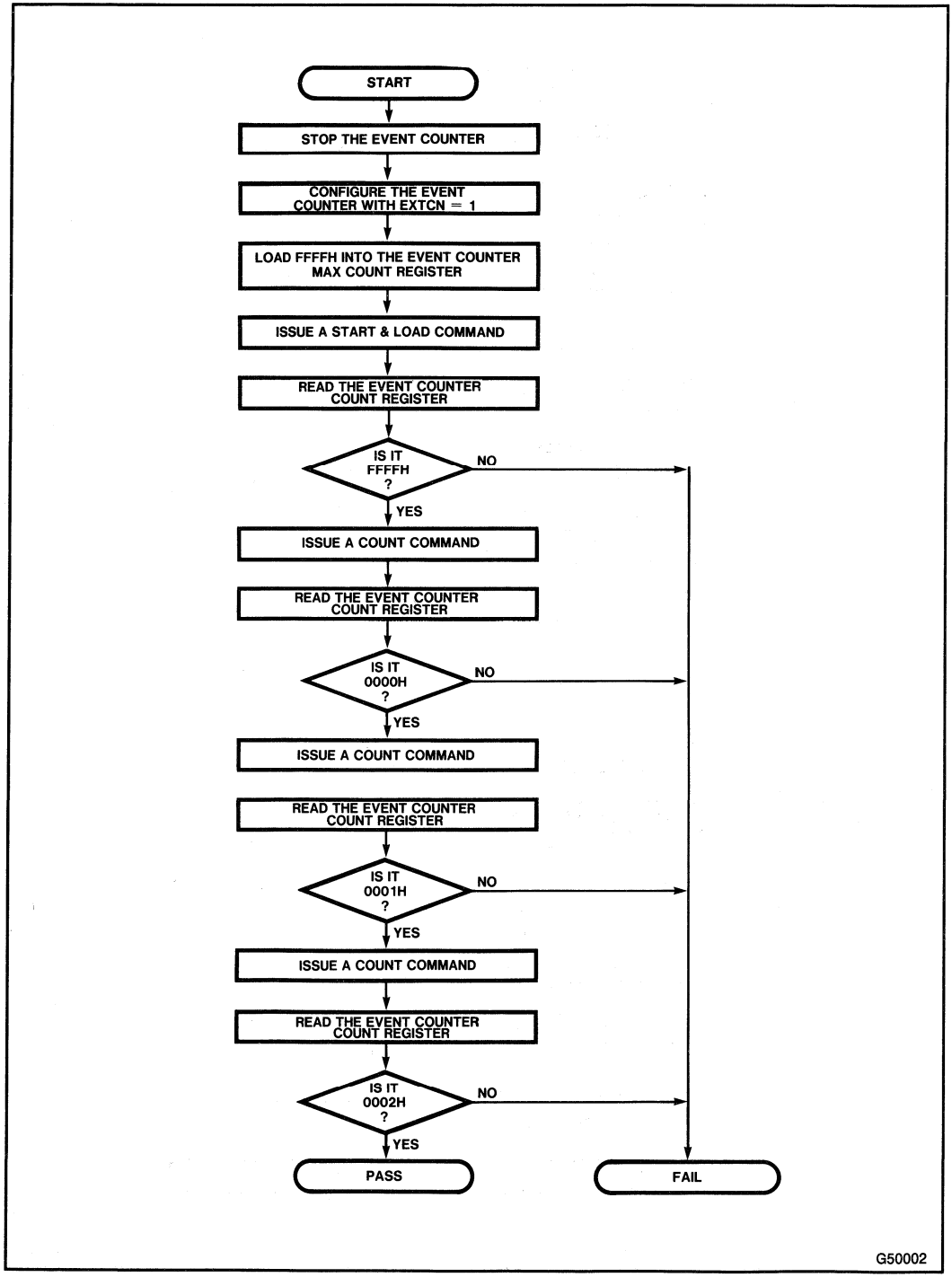


Figure 10-3. Event Counter Diagnostics

Performance Considerations 11

CHAPTER 11
PERFORMANCE CONSIDERATIONS

This section elaborates on data transfer limitations, bus utilization, and interframe spacing considerations. Note, the analysis is only for steady state, and therefore, should only be regarded as a rough guideline.

11.1 TRANSFER RATE LIMITATIONS

The system interface of the 82590 must be able to keep up with the serial link. Assuming f_s is the serial data bit rate, then $f_s/8$ is the serial data rate in bytes/sec. The data rate of the system interface depends on the system clock (f_p) and the number of clocks required to transfer one byte (n).

In order for the interface to keep up with the serial link, the following must hold:

$$\frac{f_p \times 8}{f_s \times n} \quad \text{is greater than 1}$$

The 82590 uses two on-chip FIFOs to increase burstiness of data transfers on the system bus. There are a total of 64 bytes that can be used for the FIFOs. During configuration these 64 bytes can be divided into one of four possible configurations: 32 and 32 bytes, 16 and 48 bytes, 48 and 16 bytes, and 16 and 16 bytes for the transmit and receive FIFOs respectively. FIFO limits are also specified during configuration. The receive FIFO, which is filled by the serial subsystem and emptied by the parallel subsystem, requests data transfers as soon as the number of bytes it contains becomes greater than the specified receive FIFO limit. Data transfers are continued in a burst until the receive FIFO becomes empty. The transmit FIFO, which is filled by the parallel subsystem and emptied by the serial subsystem, requests data transfers as soon as the number of bytes it contains becomes less than the specified transmit FIFO limit. Burst data transfers are continued until the transmit FIFO becomes full.

The FIFO limits must be carefully set in order to establish optimal interfaces between the two subsystems. If the transmit FIFO limit is set too high, or if the receive FIFO limit is set too low, the parallel subsystem will be constantly accessing the system bus, creating inefficient system bus usage. If the transmit FIFO limit is set too low, or if the receive FIFO limit is set too high, data transfer underrun or overrun may occur. The variables of concern in setting the FIFO limits are the system clock (f_p), the serial data bit rate (f_s), and the bus latency time (N_L). The bus latency time is the time it takes to acquire the bus. The transmit FIFO limit is directly configurable, and the receive FIFO limit is internally calculated from the transmit FIFO limit (see Chapter 6, Parameter 4). In order to prevent underruns the maximum latency time (N_{Lmax}) must be less than the time it takes to empty the transmit FIFO which is filled with bytes up to the FIFO limit.

$$\frac{N_{Lmax}}{f_p} \quad \textit{must be less than} \quad \frac{\text{LIMIT}}{f_s/8}$$

i.e.,

$$\text{LIMIT} \quad \text{must be greater than} \quad \frac{f_s \times N_{Lmax}}{f_p \times 8}$$

In order to assure an interval of N_I clocks for which the CPU has access to the system bus, the time it takes to empty the transmit FIFO from being full to its limit, plus the minimum latency time (N_{Lmix}), must be greater than the time between bursts.

$$\frac{L - LIMIT}{f_s/8} + \frac{NLMIN}{f_p} \quad \text{must be greater than} \quad \frac{N_I}{f_p},$$

where L is the length of the FIFO.

i.e.,

$$\text{LIMIT} \quad \text{is less than} \quad \frac{f_s \times (N_I - N_{Lmin})}{f_p \times 8}$$

11.2 BUS UTILIZATION

Bus utilization is defined as the fraction of time that the bus is not busy servicing the 82590 (including bus latency time). The following analysis, again, is based on 'steady state,' and therefore, should be regarded as a guideline only.

The complement of the CPU utilization is the time that the CPU is busy filling the FIFO (T_F), divided by the cycle time from the FIFO being full until it is full again.

$$CPUUTIL = 1 - \frac{T_F}{\frac{L - LIMIT}{f_s/8} + T_F + N_L}$$

To calculate T_F consider the fact that during one cycle the number of bytes entered into the FIFO equals the number removed.

$$L - LIMIT + N_L \times f_s/8 + T_F \times f_s/8 = T_F \times f_p/n$$

i.e.,

$$T_F = \frac{L - LIMIT + N_L \times f_s/8}{f_p/n - f_s/8}$$

Therefore, by substituting T_F into CPUUTIL:

$$CPUUTIL = 1 - \frac{f_s \times 8}{f_p \times 8}$$

This same expression could be derived directly, as it represents the complement of the ratio of the time to transfer a byte to/from memory, to the time to transfer a byte from/to the link. Note, this expression applies only to the period of transfer, not taking into account the events between buffers or frames.

11.3 BUFFER SIZE

The 82590 provides support for chaining buffers in memory during reception of frames. The 82590 compares the number of bytes transferred to memory to a programmable buffer limit register. When the count equals the limit and the alternate channel is set up, the 82590 switches to the alternate DMA channel, resets the counter, and issues a Request Alternate Buffer interrupt. The CPU interrupt service routine takes care of updating the DMA controller parameters, issuing an Alternate Buffer Assigned, and acknowledging the interrupt. The minimum buffer size depends on the maximum CPU service time which is composed of the following: the time from issuing the interrupt until the CPU services it, the time the CPU needs to sort out the reason for the interrupt, and the time it takes for the CPU to respond. This time depends on the system (not on the 82590); but, note that the data transfer may contend for the bus. Note also, activity on the execution channel changes the buffer size considerations.

There are several constraints on the size of the buffer (not taking into account the affect of a partially filled last buffer). The advantage of using short buffers is that, assuming frames of random length, half a buffer will be fragmented (per frame). The smaller the buffer the less memory is wasted. However, the buffer has a minimum size that is determined by the fact that, in order to prevent an overrun, the CPU service time must be shorter than the time to fill a buffer.

The advantages of longer buffers is less CPU overhead for buffer switching and memory overhead for storing buffer descriptors. The maximum buffer length of the 82590 is 1 kilobyte.

11.4 INTERFRAME SPACING

Interframe spacing (IFS) is a configuration parameter that is enforced by the link management. The minimum IFS is intended to ensure recovery time between completion of transmit or receive to receive, and between transmit to transmit. The formula for the 82590 receive recovery time is :

$$\text{IFS for IFS} \leq 23$$

$$\frac{\text{IFS}}{2} \text{ for IFS} > 23$$

The 82590 supports pipelining between received frames. This feature relaxes the constraints on the IFS. In consecutive mode (i.e., not chaining) the CPU service time, following a frame, must be shorter than the sum of the IFS and the minimum length of the frame.

Therefore, if the CPU service time is short enough, and the minimum frame long enough, the IFS may be as short as 12 bit times, which is the minimum configurable.

In chaining mode the constraint is more severe, and the IFS time must be longer than the CPU service time. This is due to a worst case where the last byte of the frame is placed in the first byte of the buffer.

The 82590 can be programmed to perform automatic retransmission if a collision is detected during preamble transmission. The 82590 does not perform retransmission by itself if a collision is detected after the preamble transmission.

If the 82590 is not programmed to assert the $\overline{\text{EOP}}$ signal on detecting a collision, it only notifies the CPU about a collision, stops transmission after sending jam, and disables transmission for backoff time, which can be equal to the IFS time in the minimum case.

If the chip is programmed to use the $\overline{\text{EOP}}$ signal, then it does not interrupt the CPU to report a collision. It handles the retransmission with the DMA controller which is programmed for Auto-Initialize mode.

It is desirable (but not mandatory) that the station be able to retransmit within the deferring time (i.e., IFS time). In order to ensure this, the IFS time must be longer than the CPU service time.

If the 82590 is configured for Tightly Coupled Interface mode, then all of the time-critical processes, such as retransmission within the deferring time, buffer switching, continuous frame reception, and buffer reclamation are handled by the 82590 and the DRAM controller without CPU intervention.

Applications 12

CHAPTER 12
APPLICATIONS

The 82590 can be used in a variety of applications. When it is used in High-Integration mode it implements most of the Data Link and Physical Layer functions required by the IEEE 802.3 TYPE 1BASE5 (StarLAN) and the IBM PC Network—Baseband and Broadband. When it is used in High-Speed mode, it can work with the 82C501 and the 82502 for IEEE 802.3 TYPE 10BASE5 (Ethernet) and TYPE 10BASE2 (Cheapernet or Thin Ethernet) implementation. If the desired network requires determinism, the Deterministic Collision Resolution (DCR) feature can be used. The linear and exponential priority schemes are also very useful for serial backplane applications.

Figure 12-1 shows a block diagram of an 82590/82560 StarLAN adapter board. Figure 12-2 shows a block diagram of an 82592/82560 Ethernet/Cheapernet adapter board. For both of these designs the 82560 provides the key functions of connecting the local bus to the host bus, providing DMA sevices with the Tightly Coupled Interface to the 82590 or the 82592, and generating appropriate memory control signals.

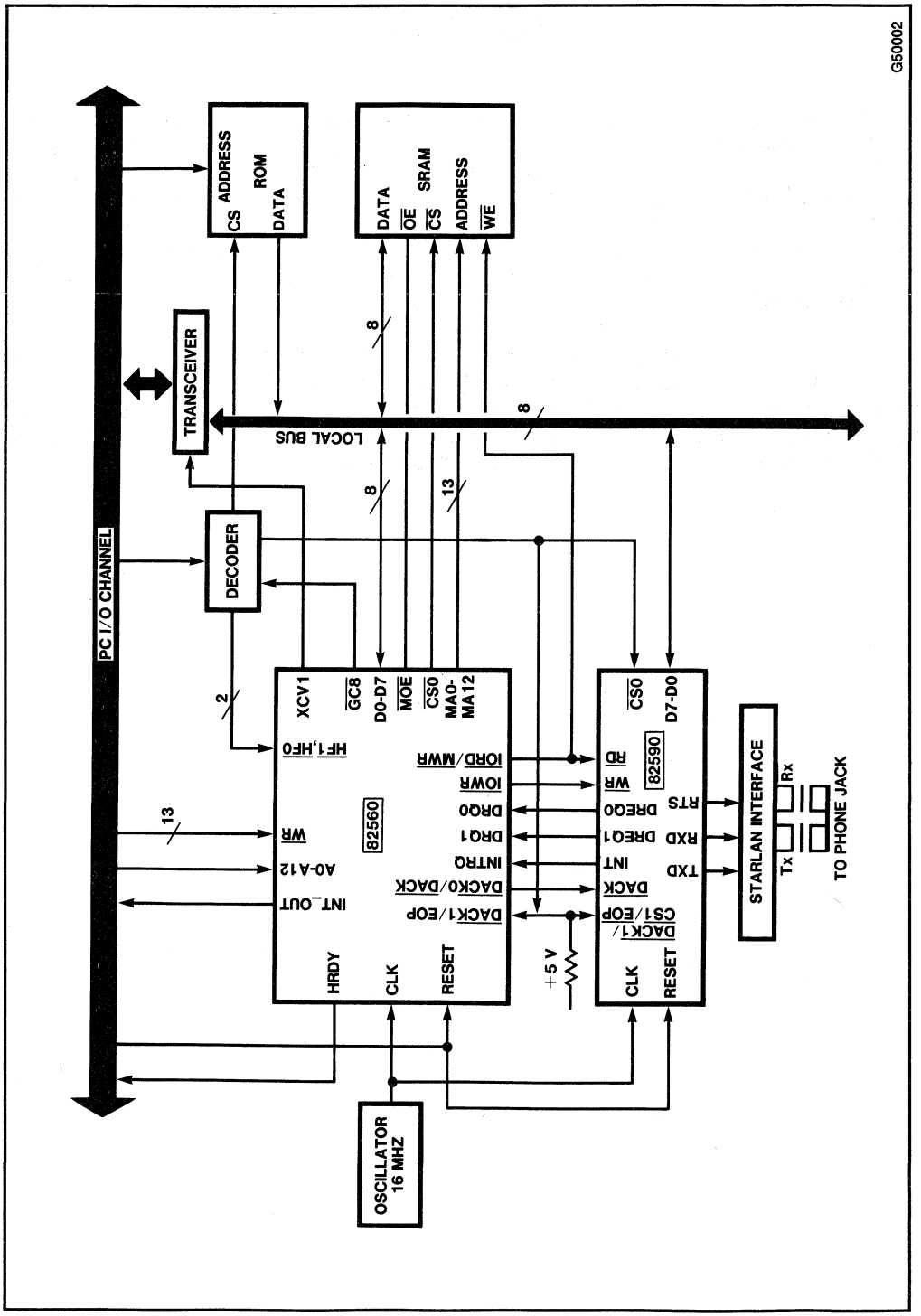


Figure 12-1. 82590/82560 StarLAN Adapter Board

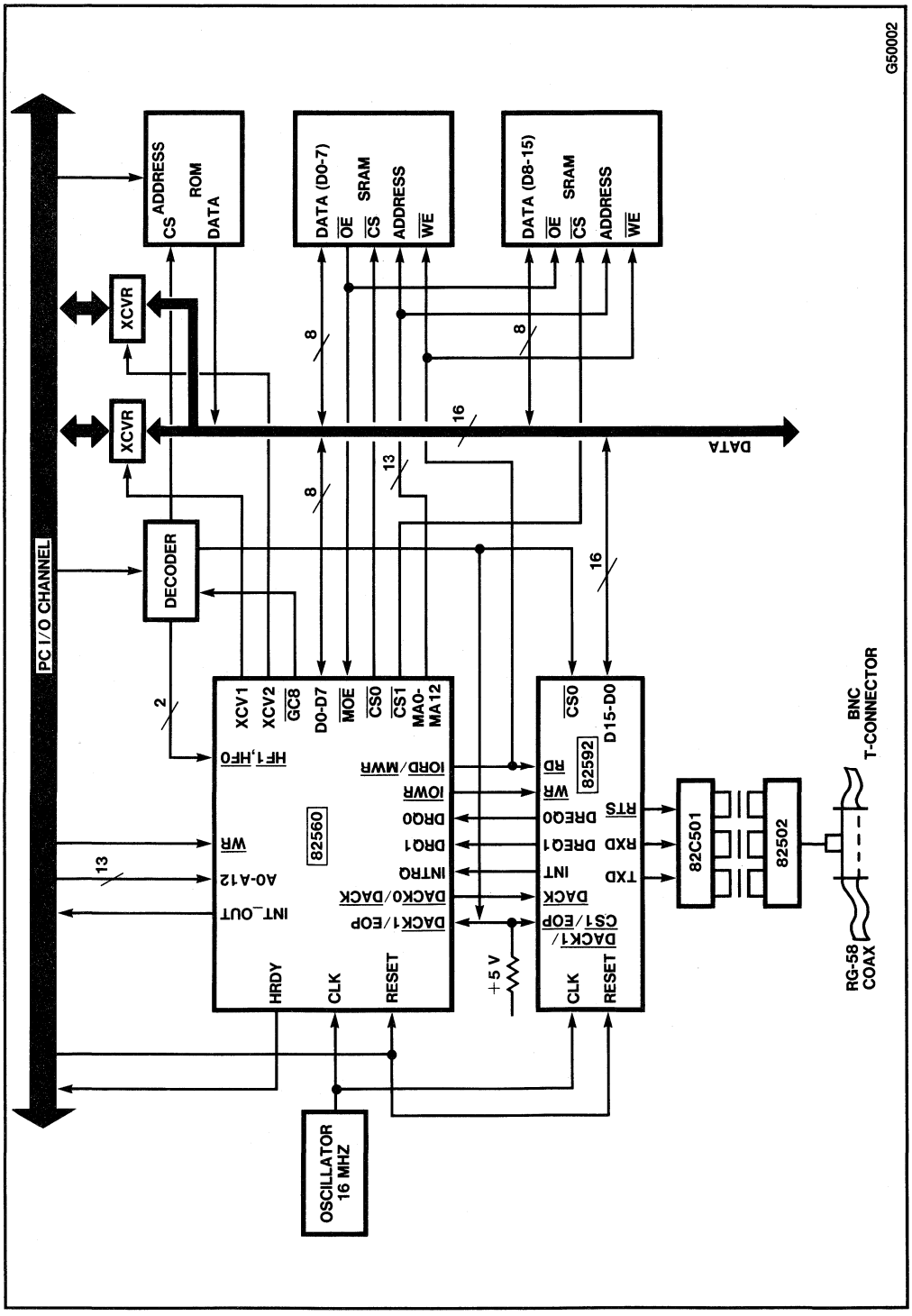


Figure 12-2. 82592/82560 Ethernet/Cheapernet Adapter Board

Local Area Network Data Sheets

7

82586
LOCAL AREA NETWORK COPROCESSOR

- **Performs Complete CSMA/CD Medium Access Control Functions Independently of CPU**
  - **— High Level Command Interface**
- **Supports Established and Emerging LAN Standards**
  - **— IEEE 802.3/Ethernet (10BASE5)**
  - **— IEEE 802.3/Cheapernet (10BASE2)**
  - **— IBM PC Network**
  - **— IEEE 802.3/StarLAN (1BASE5)**
  - **— Proprietary CSMA/CD Networks up to 10 Mbps**
- **On-Chip Memory Management**
  - **— Automatic Buffer Chaining**
  - **— Buffer Reclaim After Receipt of Bad Frames**
  - **— Save Bad Frames, Optionally**
- **Interfaces to 8-bit and 16-bit Microprocessors**
- **48 Pin DIP and 68 Pin PLCC**

  (see "Intel Packaging" Document, Order Number: 231369-001)
- **Supports Minimum Component Systems**
  - **— Shared Bus Configuration**
  - **— Interface to iAPX 186 and 188 Microprocessors without Glue**
- **Supports High Performance Systems**
  - **— Bus Master, with On-Chip DMA**
  - **— 5 MBytes/Sec Bus Bandwidth**
  - **— Compatible with Dual Port Memory**
  - **— Back to Back Frame Reception at 10 Mbps**
- **Network Management**
  - **— CRC Error Tally**
  - **— Alignment Error Tally**
  - **— Location of Cable Faults**
- **Self-Test Diagnostics**
  - **— Internal Loopback**
  - **— External Loopback**
  - **— Internal Register Dump**
  - **— Backoff Timer Check**

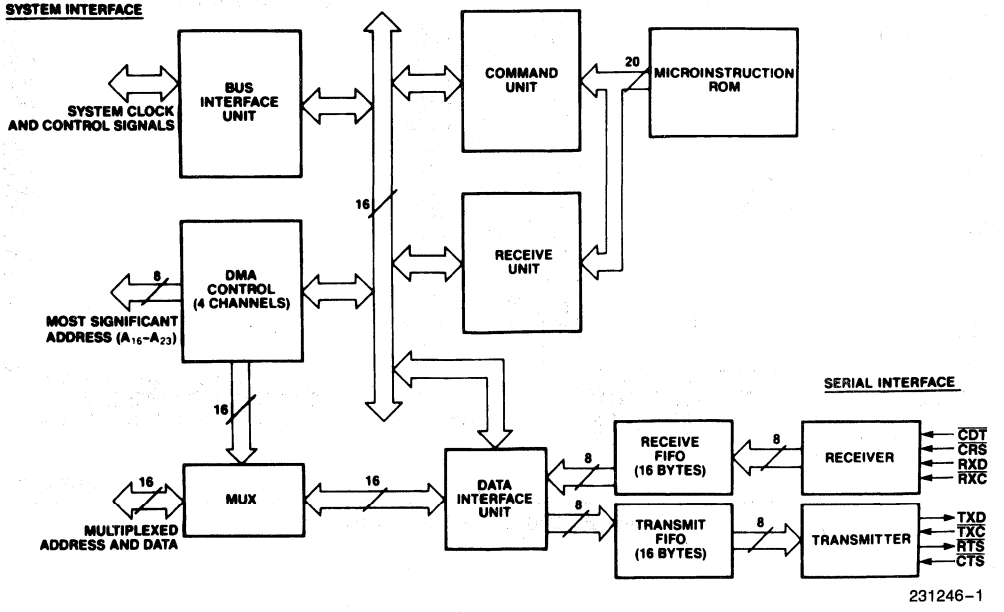
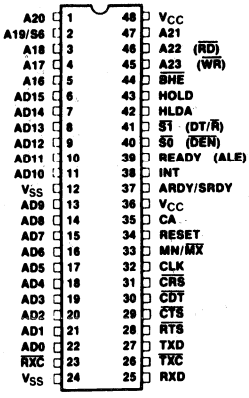


Figure 1. 82586 Functional Block Diagram

*IBM is a trademark of International Business Machines Corp.

March 1988
Order Number: 231246-006



231246-2

NOTE:
The symbols in parentheses correspond to minimum mode.

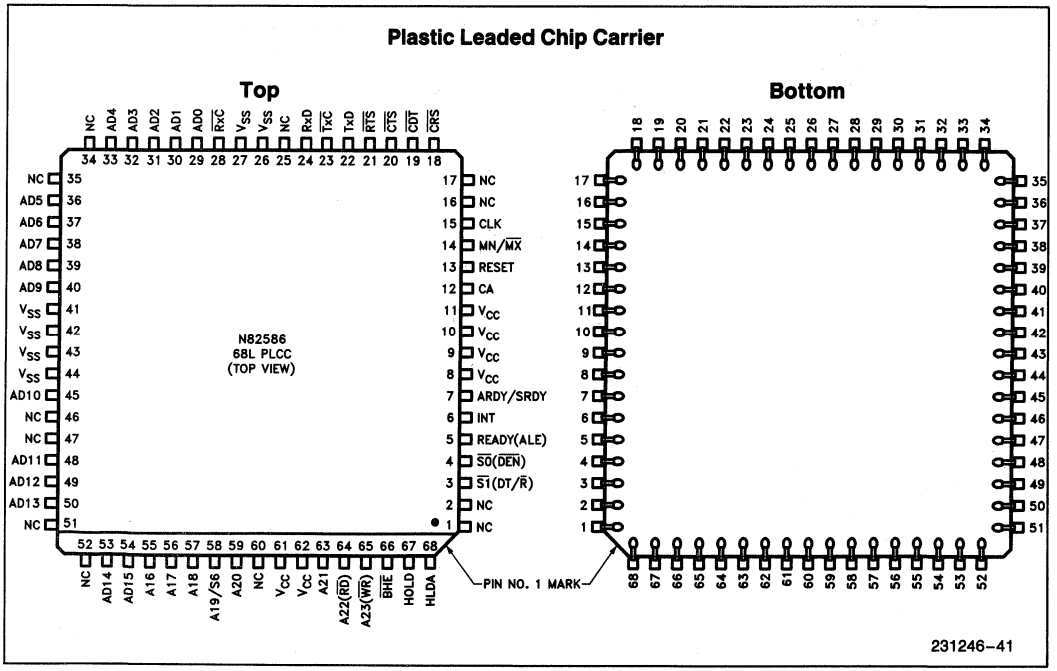


Figure 2. 82586 Pinout Diagrams

The 82586 is an intelligent, high performance Local Area Network coprocessor, implementing the CSMA/CD access method (Carrier Sense Multiple Access with Collision Detection). It performs all time-critical functions independently of the host processor, which maximizes performance and network efficiency.

The 82586 performs the full set of IEEE 802.3 CSMA/CD media access control and channel interface functions including: framing, preamble generation and stripping, source address generation, destination address checking, CRC generation and checking, short frame detection. Any data rate up to 10 Mb/s can be used.

The 82586 features a powerful host system interface. It automatically manages memory structures with command chaining and bidirectional data chaining. An on-chip DMA controller manages 4 channels transparently to the user. Buffers containing errored or collided frames can be automatically recovered. The 82586 can be configured for 8-bit or 16-bit data path, with maximum burst transfer rate of 2 or 4 Mbyte/sec. respectively. Memory address space is 16 Mbyte maximum.

The 82586 provides two independent 16 byte FIFOs, one for receiving and one for transmitting. The threshold for block transfer to/from memory is programmable, enabling the user to optimize bus overhead for a given worst case bus latency.

The 82586 provides a rich set of diagnostic and network management functions including: internal and external loopbacks, exception condition tallies, channel activity indicators, optional capture of all frames regardless of destination address, optional capture of errored or collided frames, and time domain reflectometry for locating faults in the cable.

The 82586 can be used in either baseband or broadband networks. It can be configured for maximum network efficiency (minimum contention overhead) for any length network operating at any data rate up to 10 Mbps. The controller supports address field lengths of 1, 2, 3, 4, 5, or 6 bytes. It can be configured for either the IEEE 802.3/Ethernet or HDLC method of frame delineation. Both 16-bit and 32-bit CRC are supported.

The 82586 is fabricated in Intel's reliable HMOS II 5V technology and is available in a 48 pin DIP or 68 pin PLCC package.

Table 1. 82586 Pin Description

Symbol	48 Pin DIP Pin No.	68 Pin PLCC Pin No.	Type Level	Name and Function
V_{CC}, V_{CC}	48, 36	8, 9, 10, 11, 61, 62		System Power: +5V Power Supply.
V_{SS}, V_{SS}	12, 24	26, 27, 41, 42, 43, 44		System Ground.
RESET	34	13	I TTL	RESET is an active HIGH internally synchronized signal, causing the 82586 to terminate present activity immediately. The signal must be HIGH for at least four clock cycles. The 82586 will execute RESET within ten system clock cycles starting from RESET HIGH. When RESET returns LOW, the 82586 waits for the first CA to begin the initialization sequence.
TxD	27	22	0 TTL	Transmitted Serial Data output signal. This signal is HIGH when not transmitting.
$\overline{TxC}$	26	23	I *	Transmit Data Clock. This signal provides timing information to the internal serial logic, depending upon the mode of data transfer. For NRZ mode of operation, data is transferred to the TxD pin on the HIGH to LOW clock transition.
RxD	25	24	I TTL	Received Data Input Signal.
$\overline{RxC}$	23	28	I *	Received Data Clock. This signal provides timing information to the internal shifting logic depending upon the mode of data transfer. For NRZ data, the state of the RxD pin is sampled on the HIGH to LOW clock transition.

*See D.C. Characteristics.

Table 1. 82586 Pin Description (Continued)

Symbol	48 Pin DIP Pin No.	68 Pin PLCC Pin No.	Type Level	Name and Function
$\overline{RTS}$	28	21	0 TTL	Request To Send signal. When LOW, notifies an external interface that the 82586 has data to transmit. It is forced HIGH after a Reset and while the Transmit Serial Unit is not sending data.
$\overline{CTS}$	29	20	I TTL	Active LOW Clear To Send input enables the 82586 transmitter to actually send data. It is normally used as an interface handshake to $\overline{RTS}$. This signal going inactive stops transmission. It is internally synchronized. If $\overline{CTS}$ goes inactive, meeting the setup time to $\overline{TxC}$ negative edge, transmission is stopped and $\overline{RTS}$ goes inactive within, at most, two TxC cycles.
$\overline{CRS}$	31	18	I TTL	Active LOW Carrier Sense input used to notify the 82586 that there is traffic on the serial link. It is used only if the 82586 is configured for external Carrier Sense. When so configured, external circuitry is required for detecting serial link traffic. It is internally synchronized. To be accepted, the signal must stay active for at least two serial clock cycles.
$\overline{CDT}$	30	19	I TTL	Active LOW Collision Detect input is used to notify the 82586 that a collision has occurred. It is used only if the 82586 is configured for external Collision Detect. External circuitry is required for detecting the collision. It is internally synchronized. To be accepted, the signal must stay active for at least two serial clock cycles. During transmission, the 82586 is able to recognize a collision one bit time after preamble transmission has begun.
INT	38	6	0 TTL	Active HIGH Interrupt request signal.
CLK	32	15	I MOS	The system clock input from the 80186 or another symmetrical clock generator.
MN/$\overline{MX}$	33	14	I TTL	When HIGH, MN/$\overline{MX}$ selects $\overline{RD}$, $\overline{WR}$, ALE $\overline{DEN}$, DT/$\overline{R}$ (Minimum Mode). When LOW, MN/$\overline{MX}$ selects A22, A23, READY, $\overline{S0}$, $\overline{S1}$ (Maximum Mode). Note: This pin should be static during 82586 operation.
AD0–AD15	6–11, 13–22	29–33, 36–40, 45, 48, 49, 50, 53, 54	I/O TTL	These lines form the time multiplexed memory address (t1) and data (t2, t3, tW, t4) bus. When operating with an 8-bit bus, the high byte will output the address only during T1. AD0–AD15 are floated after a RESET or when the bus is not acquired.
A16–A18 A20–A23	1, 3–5 45–47	55–57, 59, 63–65	0 TTL	These lines constitute 7 out of 8 most significant address bits for memory operation. They switch during t1 and stay valid during the entire memory cycle. The lines are floated after RESET or when the bus is not acquired. Address lines A22 and A23 are not available for use in minimum mode.
A19/S6	2	58	0 TTL	During t1 it forms line 19 of the memory address. During t2 through t4 it is used as a status indicating that this is a Master peripheral cycle, and is HIGH. Its timing is identical to that of AD0–AD15 during write operation.

Table 1. 82586 Pin Description (Continued)

Symbol	48 Pin DIP Pin No.	68 Pin PLCC Pin No.	Type Level	Name and Function
HOLD	43	67	0 TTL	HOLD is an active HIGH signal used by the 82586 to request local bus mastership at the end of the current CPU bus transfer cycle, or at the end of the current DMA burst transfer cycle. In normal operation, HOLD goes inactive before HLDA. The 82586 can be forced off the bus by HLDA going inactive. In this case, HOLD goes inactive within four clock cycles in word mode and eight clock cycles in byte mode.
HLDA	42	68	I TTL	HLDA is an active HIGH Hold Acknowledge signal indicating that the CPU has received the HOLD request and that bus control has been relinquished to the 82586. It is internally synchronized. After HOLD is detected as LOW, the processor drives HLDA LOW. Note, CONNECTING V_{CC} TO HLDA IS NOT ALLOWED because it will cause a deadlock. Users wanting to give permanent bus access to the 82586 should connect HLDA with HOLD.
CA	35	12	I TTL	The CA pin is a Channel Attention input used by the CPU to initiate the 82586 execution of memory resident Command Blocks. The CA signal is synchronized internally. The signal must be HIGH for at least one system clock period. It is latched internally on HIGH to LOW edge and then detected by the 82586.
$\overline{BHE}$	44	66	0 TTL	The Bus High Enable signal ($\overline{BHE}$) is used to enable data onto the most significant half of the data bus. Its timing is identical to that of A16–A23. With a 16-bit bus it is LOW and with an 8-bit bus it is HIGH. Note: after RESET, the 82586 is configured to 8-bit bus.
READY	39	5	I TTL	This active HIGH signal is the acknowledgement from the addressed memory that the transfer cycle can be completed. While LOW, it causes wait states to be inserted. This signal must be externally synchronized with the system clock. The Ready signal internal to the 82586 is a logical OR between READY and SRDY/ARDY.
ARDY/SRDY	37	7	I TTL	This active HIGH signal performs the same function as READY. If it is programmed at configure time to SRDY, it is identical to READY. If it is programmed to ARDY, the positive edge of the Ready signal is internally synchronized. Note, the negative edge must still meet setup and hold time specifications, when in ARDY mode. The ARDY signal must be active for at least one system clock HIGH period for proper strobing. The Ready signal internal to the 82586 is a logical OR between READY (in Maximum Mode only) and SRDY/ARDY. Note that following RESET, this pin assumes ARDY mode.

Table 1. 82586 Pin Description (Continued)

Symbol	48 Pin DIP Pin No.	68 Pin PLCC Pin No.	Type Level	Name and Function
$\overline{S0}$, $\overline{S1}$	40,41	4, 3	0 TTL	Maximum mode only. These status pins define the type of DMA transfer during the current memory cycle. They are encoded as follows: $\overline{S1}$ $\overline{S0}$ 0 0 Not Used 0 1 Read Memory 1 0 Write Memory 1 1 Passive Status is active from the middle of t4 to the end of t2. They return to the passive state during t3 or during tW when READY or ARDY is HIGH. These signals can be used by the 8288 Bus Controller to generate all memory control and timing signals.* Any change from the passive state, signals the 8288 to start the next t1 to t4 bus cycle. These pins are pulled HIGH and floated after a system RESET and when the bus is not acquired.
$\overline{RD}$	46	64	0 TTL	Used in minimum mode only. The read strobe indicates that the 82586 is performing a memory read cycle. $\overline{RD}$ is active LOW during t2, t3 and tW of any read cycle. This signal is pulled HIGH and floated after a RESET and when the bus is not acquired.
$\overline{WR}$	45	65	0 TTL	Used in minimum mode only. The write strobe indicates that the 82586 is performing a write memory cycle. $\overline{WR}$ is active LOW during t2, t3 and tW of any write cycle. It is pulled HIGH and floats after RESET and when the bus is not acquired.
ALE	39	5	0 TTL	Used in minimum mode only. Address Latch Enable is provided by the 82586 to latch the address into the 8282/8283 address latch. It is a HIGH pulse, during t1 ('clock low') of any bus cycle. Note that ALE is never floated.
$\overline{DEN}$	40	4	0 TTL	Used in minimum mode only. Data ENable is provided as output enable for the 8286/8287 transceivers in a stand-alone (no 8288) system. $\overline{DEN}$ is active LOW during each memory access. For a read cycle, it is active from the middle of t2 until the beginning of t4. For a write cycle, it is active from the beginning of t2 until the middle of t4. It is pulled HIGH and floats after a system RESET or when the bus is not acquired.
DT/$\overline{R}$	41	3	0 TTL	Used in minimum mode only. DT/$\overline{R}$ is used in non-8288 systems using an 8286/8287 data bus transceiver. It controls the direction of data flow through the Transceiver. Logically, DT/$\overline{R}$ is equivalent to $\overline{S1}$. It becomes valid in the t4 preceding a bus cycle and remains valid until the final t4 of the cycle. This signal is pulled HIGH and floated after a RESET or when the bus is not acquired.

NOTE:
*8288 does not support 10 MHz operation.

82586/HOST CPU INTERACTION

Communication between the 82586 and the host is carried out via shared memory. The 82586's on-chip DMA capability allows autonomous transfer of data blocks (buffers, frames) and relieves the CPU of byte transfer overhead. The 82586 is optimized to interface the iAPX 186, but due to the small number of hardware signals between the 82586 and the CPU, the 82586 can operate easily with other processors. The 82586/host interaction is explained separately in terms of the logical interface and the hardware bus interface.

The 82586 consists of two independent units: Command Unit (CU) and Receive Unit (RU). The CU executes commands from shared memory. The RU handles all activities related to frame reception. The CU and RU enable the 82586 to engage in the two types of activities simultaneously: the CU may be fetching and executing commands out of memory, and the RU may be storing received frames in memory. CPU intervention is only required after the CU executes a sequence of commands or the RU stores a sequence of frames.

The only hardware signals that connect the CPU and the 82586 are INTERRUPT and CHANNEL ATTENTION (see Figure 3). Interrupt is used by the 82586 to draw the CPU's attention to a change in the contents of the SCB. Channel Attention is used by the CPU to draw the 82586's attention.

82586 SYSTEM MEMORY STRUCTURE

The Shared Memory structure consists of four parts: Initialization Root, System Control Block (SCB), Command List, and Receive Frame Area (RFA) (see Figure 4).

The Initialization Root is at a predetermined location in the memory space, (0FFFFF6H), known to both the host CPU and the 82586. The root is accessed at initialization and points to the System Control Block.

The System Control Block (SCB) functions as a bidirectional mail drop between the host CPU, CU and RU. It is the central element through which the CPU and the 82586 exchange control and status information. The SCB consists of two parts, the first of which entails instructions from the CPU to the 82586. These include: control of the CU and RU (START, ABORT, SUSPEND, RESUME), a pointer to the list of commands for the CU, a pointer to the receive frame area, and a set of Interrupt acknowledge bits. The second entails status information keyed by the 82586 to the CPU, including: state of the CU and RU (e.g. IDLE, ACTIVE READY, SUSPENDED, NO RECEIVE RESOURCES), interrupts bits (command completed, frame received, CU not ready, RU not ready), and statistics (see Figure 4).

The Command List serves as a program for the CU. Individual commands are placed in memory units called a Command Block, or CB. CB's contain command specific parameters and command specific statuses. Specifically, these high level commands are called Action Commands (e.g. Transmit, Configure).

A specific command, Transmit, causes transmission of a frame by the 82586. The Transmit command block includes Destination Address, Length Field, and a pointer to a list of linked buffers that holds the frame to be constructed from several buffers scattered in memory. The Command Unit performs with-

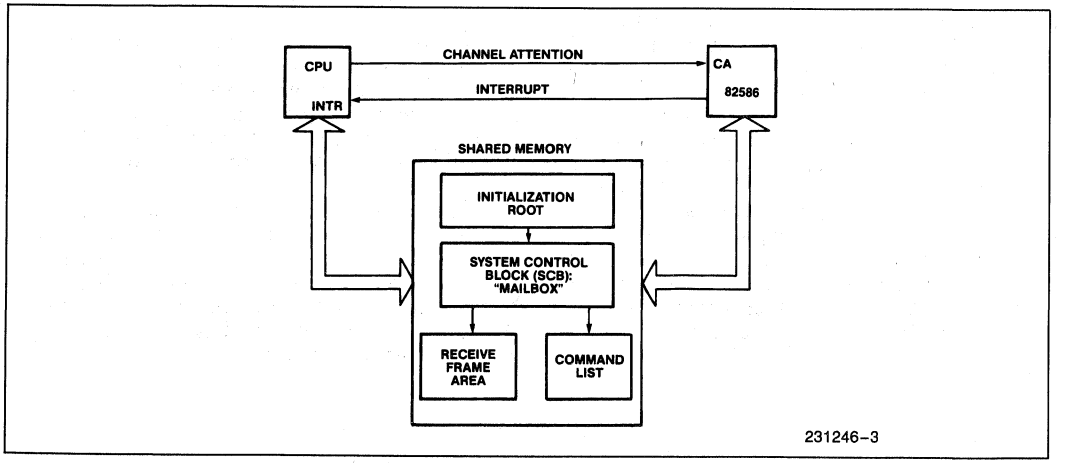


Figure 3. 82586/Host CPU Interaction

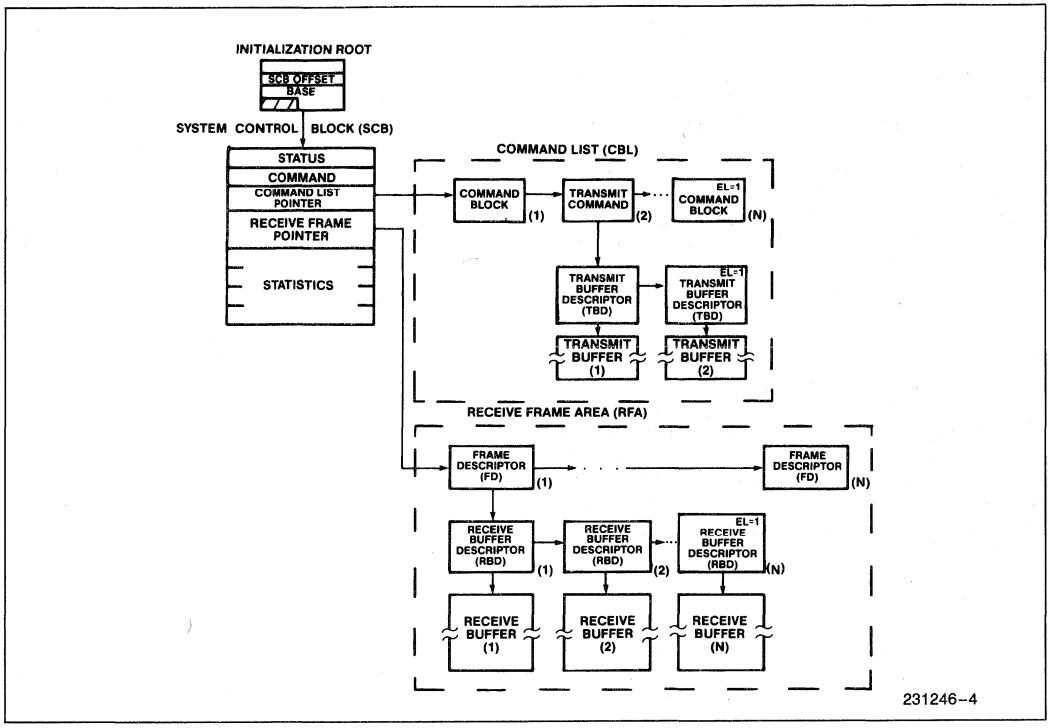


Figure 4. 82586 Shared Memory Structure

out the CPU intervention, the DMA of each buffer and the prefetching of references to new buffers in parallel. The CPU is notified only after successful transmission or retransmission.

The Receive Frame Area is a list of Free Frame Descriptors (Descriptors not yet used) and a list of buffers prepared by the user. It is conceptually distinct from the Command List. Frames arrive without being solicited by the 82586. The 82586 must be prepared to receive them even if it is engaged in other activities and to store them in the Free Frame Area. The Receive Unit fills the buffers upon frame reception and reformats the Free Buffer List into received frame structures. The frame structure is virtually identical to the format of the frame to be transmitted. The first frame descriptor is referenced by SCB. A Frame Descriptor and the associated Buffer Descriptor wasted upon receiving a Bad Frame (CRC or Alignment errored, Receive DMA overrun errored, or Collision fragmented frame) are automatically reclaimed and returned to the Free Buffer List, unless the chip is configured to Save Bad Frames.

Receive buffer chaining (i.e. storing incoming frames in a linked list of buffers) improves memory utilization significantly. Without buffer chaining, the user must allocate consecutive blocks of the maximum frame size (1518 bytes in Ethernet) for each frame. Taking into account that a typical frame size may be about 100 bytes, this practice is very inefficient. With buffer chaining, the user can allocate small buffers and the 82586 uses only as many as needed.

In the past, the drawback of buffer chaining was the CPU processing overhead and the time involved in the buffer switching (especially at 10 Mb/s). The 82586 overcomes this drawback by performing buffer management on its own for both transmission and reception (completely transparent to the user).

The 82586 has a 22-bit memory address range in minimum mode and 24-bit memory address range in maximum mode. All memory structures, the System Control Block, Command List, Receive Descriptor List, and all buffer descriptors must reside within one 64K-byte memory segment. The Data Buffers can be located anywhere in the memory space.

TRANSMITTING FRAMES

The 82586 executes high level action commands from the Command List in external memory. Action commands are fetched and executed in parallel with the host CPU's operation, thereby significantly improving system performance. The general action commands format is shown in Figure 5.

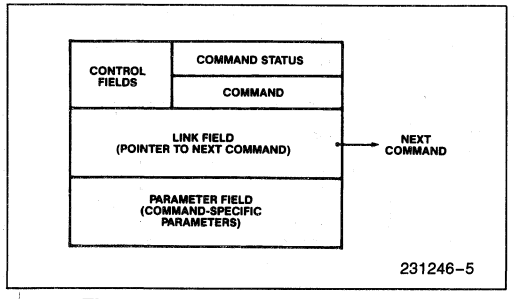


Figure 5. Action Command Format

Message transmission is accomplished by using the Transmit command. A single Transmit command contains, as part of the command-specific parameters, the destination address and length field for the transmitted frame along with a pointer to a buffer area in memory containing the data portion of the frame. (See Figure 15.) The data field is contained in a memory data structure consisting of a Buffer Descriptor (BD) and Data Buffer (or a linked list of buffer descriptors and buffers) as shown in Figure 6. The BD contains a Link Field which points to the next BD on the list and a 24-bit address pointing to the Data Buffer itself. The length of the Data Buffer is specified by the Actual Count field of the BD.

Using the BD's and Data Buffers, multiple Data Buffers can be 'chained' together. Thus, a frame with a long Data Field can be transmitted using multiple (shorter) Data buffers chained together. This chaining technique allows the system designer to develop efficient buffer management policies.

The 82586 automatically generates the preamble (alternating 1's and 0's) and start frame delimiter, fetches the destination address and length field from the Transmit command, inserts its unique address as the source address, fetches the data field from

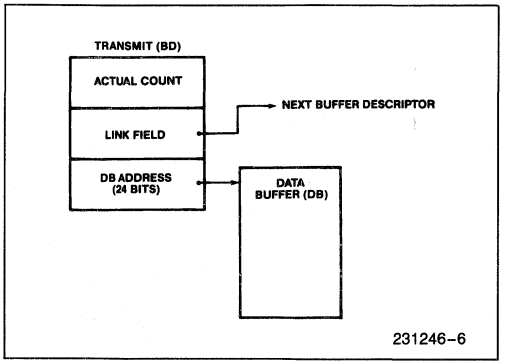


Figure 6. Data Buffer Descriptor and Data Buffer Structure

buffers pointed to by the Transmit command, and computes and appends the CRC at the end of the frame. See Figure 7.

The 82586 can be configured to generate either the Ethernet or HDLC start and end frame delimiters. In the Ethernet mode, the start frame delimiter is 10101011 and the end frame delimiter indicated by the lack of a signal after transmitting the last bit of the frame check sequence field. When in the HDLC mode, the 82586 will generate the 01111110 'flag' for the start and end frame delimiters and perform the standard 'bit stuffing/stripping'. In addition, the 82586 will optionally pad frames that are shorter than the specified minimum frame length by appending the appropriate number of flags to the end of the frame.

In the event of a collision (or collisions), the 82586 manages the entire jam, random wait and retry process, reinitializing DMA pointers without CPU intervention. Multiple frames can be sent by linking the appropriate number of Transmit commands together. This is particularly useful when transmitting a message that is larger than the maximum frame size (1518 bytes for Ethernet).

RECEIVING FRAMES

In order to minimize CPU overhead, the 82586 is designed to receive frames without CPU supervision. The host CPU first sets aside an adequate

PREAMBLE	START FRAME DELIMITER	DEST ADDR	SOURCE ADDR	LENGTH FIELD	DATA FIELD	FRAME CHECK SEQUENCE	END FRAME DELIMITER

Figure 7. Frame Format

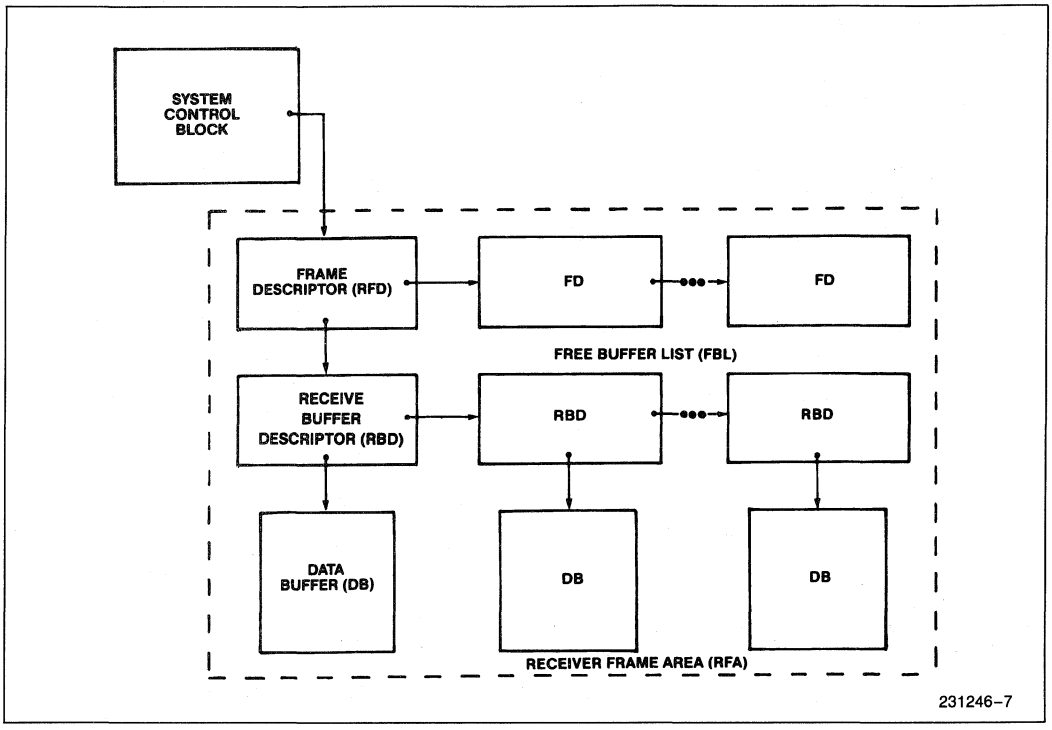


Figure 8. Receive Frame Area Diagram

amount of receive buffer space and then enables the 82586's Receive Unit. Once enabled, the RU 'watches' for any of its frames which it automatically stores in the Receive Frame Area (RFA). The RFA consists of a Receive Descriptor List (RDL) and a list of free buffers called the Free Buffer List (FBL) as shown in Figure 8. The individual Receive Frame Descriptors that make up the RDL are used by the 82586 to store the destination and source address, length field and status of each frame that is received. (Figure 9.)

The 82586, once enabled, checks each passing frame for an address match. The 82586 will recognize its own unique address, one or more multicast addresses or the broadcast address. If a match occurs, it stores the destination and source address and length field in the next available RFD. It then begins filling the next free Data Buffer on the FBL (which is pointed to by the current RFD) with the data portion of the incoming frame. As one DB is filled, the 82586 automatically fetches the next DB on the FBL until the entire frame is received. This buffer chaining technique is particularly memory efficient because it allows the system designer to set aside buffers that fit a frame size that may be much shorter than the maximum allowable frame.

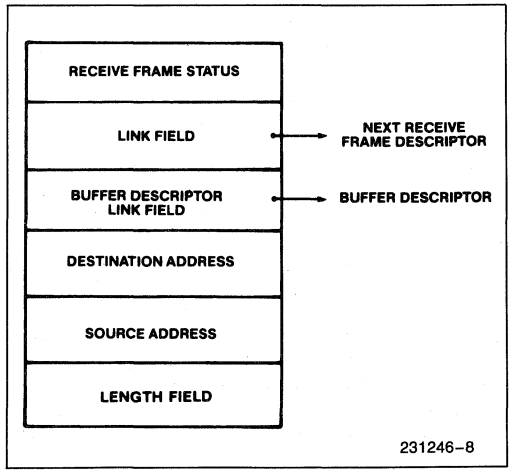


Figure 9. Receive Frame Descriptor

Once the entire frame is received without error, the 82586 performs the following housekeeping tasks:

- Updates the Actual Count field of the last Buffer Descriptor used to hold the frame just received with the number of bytes stored in its associated Data Buffer.

- Fetches the address of the next free Receive Frame Descriptor.
- Writes the address of the next free Buffer Descriptor into the next free Receive Frame Descriptor.
- Posts a 'Frame Received' interrupt status bit in the SCB.
- Interrupts the CPU.

In the event of a frame error, such as a CRC error, the 82586 automatically reinitializes its DMA pointers and reclaims any data buffers containing the bad frame. As long as Receive Frame Descriptors and data buffers are available, the 82586 will continue to receive frames without further CPU help.

82586 NETWORK MANAGEMENT AND DIAGNOSTIC FUNCTIONS

The behavior of data communication networks is typically very complex due to their distributed and asynchronous nature. It is particularly difficult to pinpoint a failure when it occurs. The 82586 was designed in anticipation of these problems and includes a set of features for improving reliability and testability.

The 82586 reports on the following events after each frame transmitted:

- Transmission successful.
- Transmission unsuccessful; lost Carrier Sense.
- Transmission unsuccessful; lost Clear-to-Send.
- Transmission unsuccessful; DMA underrun because the system bus did not keep up with the transmission.
- Transmission unsuccessful; number of collisions exceeded the maximum allowed.

The 82586 checks each incoming frame and reports on the following errors, (if configured to 'Save Bad Frame'):

- CRC error: incorrect CRC in a well aligned frame.
- Alignment error: incorrect CRC in a misaligned frame.
- Frame too short: the frame is shorter than the configured value for minimum frame length.
- Overrun: the frame was not completely placed in memory because the system bus did not keep up with incoming data.
- Out of buffers: no memory resources to store the frame, so part of the frame was discarded.

NETWORK PLANNING AND MAINTENANCE

To perform proper planning, operation, and maintenance of a communication network, the network management entity must accumulate information on network behavior. The 82586 provides a rich set of network-wide diagnostics that can serve as the basis for a network management entity.

Network Activity information is provided in the status of each frame transmitted. The activity indicators are:

- Number of collisions: number of collisions the 82586 experienced in attempting to transmit this frame.
- Deferred transmission: indicates if the 82586 had to defer to traffic on the link during the first transmission attempt.

Statistics registers are updated after each received frame that passes the address filtering, and is longer than the Minimum Frame Length configuration parameter.

- CRC errors: number of frames that experienced a CRC error and were properly aligned.
- Alignment errors: number of frames that experienced a CRC error and were misaligned.
- No-resources: number of correct frames lost due to lack of memory resources.
- Overrun errors: number of frame sequences lost due to DMA overrun.

The 82586 can be configured to Promiscuous Mode. In this mode it captures all frames transmitted on the Network without checking the Destination Address. This is useful in implementing a monitoring station to capture all frames for analysis.

The 82586 is capable of determining if there is a short or open circuit anywhere in the Network using the built in Time Domain Reflectometer (TDR) mechanism.

STATION DIAGNOSTICS

The chip can be configured to External Loopback. The transmitter to receiver interconnection can be placed anywhere between the 82586 and the link to locate faults, for example: the 82586 output pins, the Serial Interface Unit, the Transceiver cable, or in the Transceiver.

The 82586 has a mechanism recognizing the transceiver 'heart beat' signal for verifying the correct operation of the Transceiver's collision detection circuitry.

82586 SELF TESTING

The 82586 can be configured to Internal Loopback. It disconnects itself from the Serial Interface Unit, and any frame transmitted is received immediately. The 82586 connects the Transmit Data to the Receive Data signal and the Transmit Clock to the Receive Clock.

The Dump Command causes the chip to write over 100 bytes of its internal registers to memory.

The Diagnose command checks the exponential Backoff random number generator internal to the 82586.

CONTROLLING THE 82586

The CPU controls operation of the 82586's Command Unit (CU) and Receive Unit (RU) of the 82586 via the System Control Block.

THE COMMAND UNIT (CU)

The Command Unit is the logical unit that executes Action Commands from a list of commands very similar to a CPU program. A Command Block (CB) is associated with each Action Command.

The CU can be modeled as a logical machine that takes, at any given time, one of the following states:

- IDLE—CU is not executing a command and is not associated with a CB on the list. This is the initial state.
- SUSPENDED—CU is not executing a command but (different from IDLE) is associated with a CB on the list.
- ACTIVE—CU is currently executing an Action Command, and points to its CB.

The CPU may affect the CU operation in two ways: issuing a CU control Command or setting bits in the COMMAND word of the Action Command.

THE RECEIVE UNIT (RU)

The Receive Unit is the logical unit that receives frames and stores them in memory.

The RU is modeled as a logical machine that takes, at any given time, one of the following states:

- IDLE—RU has no memory resources and is discarding incoming frames. This is the initial RU state.
- NO-RESOURCES—RU has no memory resources and is discarding incoming frames. This state differs from the IDLE state in that RU accumulates statistics on the number of frames it had to discard.
- SUSPENDED—RU has free memory resources to store incoming frames but discard them anyway.

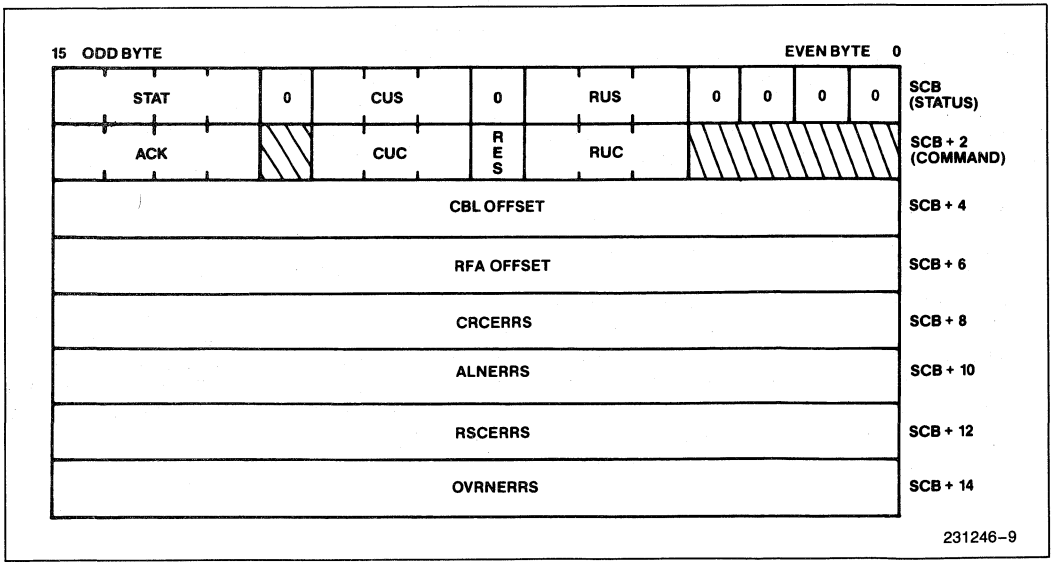


Figure 10. System Control Block (SCB) Format

- READY—RU has free memory resources and stores incoming frames.

The CPU may affect RU operation in three ways: issuing an RU Control Command, setting bits in Frame Descriptor, FD, COMMAND word of the frame currently being received, or setting EL bit of Buffer Descriptor, BD, of the buffer currently being filled.

SYSTEM CONTROL BLOCK (SCB)

The System Control Block is the communication mail-box between the 82586 and the host CPU. The SCB format is shown in Figure 10.

The host CPU issues Control Commands to the 82586 via the SCB. These commands may appear at any time during routine operation, as determined by the host CPU. After the required Control Command is setup, the CPU sends a CA signal to the 82586.

SCB is also used by the 82586 to return status information to the host CPU. After inserting the required status bits into SCB, the 82586 issues an Interrupt to the CPU.

The format is as follows:

STATUS word: Indicates the status of the 82586. This word is modified only by the 82586. Defined bits are:

CX	(Bit 15)	• A command in the CBL having its 'I' (interrupt) bit set has been executed.
FR	(Bit 14)	• A frame has been received.
CNR	(Bit 13)	• The Command Unit left the Active state.
RNR	(Bit 12)	• The Receive Unit left the Ready state.
CUS	(Bits 8–10)	• (3 bits) this field contains the status of the Command Unit. Valid values are: 0 — Idle 1 — Suspended 2 — Active 3–7 — Not Used
RUS	(Bits 4—6)	• (3 bits) this field contains the status of the Receive Unit. Valid values are: 0 — Idle 1 — Suspended 2 — No Resources 3 — Not Used 4 — Ready 5–7 — Not Used

COMMAND word: Specifies the action to be performed as a result of the CA. This word is set by the CPU and cleared by the 82586. Defined bits are:

ACK-CX	(Bit 15)	• Acknowledges the command executed event.
ACK-FR	(Bit 14)	• Acknowledges the frame received event.
ACK-CNA	(Bit 13)	• Acknowledes that the Command Unit became not ready.
ACK-RNR	(Bit 12)	• Acknowledges that the Receive Unit became not ready.
CUC	(Bits 8–10)	• (3 bits) this field contains the command to the Command Unit.
	0	• NOP (doesn't affect current state of the unit).
	1	• Start execution of the first command on the CBL. If a command is in execution, then complete it before starting the new CBL. The beginning of the CBL is in CBL OFFSET.
	2	• Resume the operation of the command unit by executing the next command. This operation assumes that the command unit has been previously suspended.
	3	• Suspend execution of commands on CBL after current command is complete.
	4	• Abort execution of commands immediately.
	5–7	• Reserved, illegal for use.
RUC	(Bits 4–6)	• (3 bits) This field contains the command to the receive unit. Valid values are:
	0	• NCP (does not alter current state of unit).
	1	• Start reception of frames. If a frame is being received, then complete reception before starting. The beginning of the RFA is contained in the RFA OFFSET.
	2	• Resume frame receiving (only when in suspended state.)
	3	• Suspend frame receiving. If a frame is being received, then complete its reception before suspending.
	4	• Abort receiver operation immediately.
	5–7	• Reserved, illegal for use.
RESET	(Bit 7)	• Reset chip (logically the same as hardware RESET).

CBL-OFFSET:

Gives the 16-bit offset address of the first command (Action Command) in the command list to be executed following CU-START. Thus, the 82586 reads this word only if the CUC field contained a CU-START Control Command.

RFA-OFFSET:

Points to the first Receive Frame Descriptor in the Receive Frame Area.

CRCERRS:

CRC Errors - contains the number of properly aligned frames received with a CRC error.

ALNERRS:

Alignment Errors - contains the number of misaligned frames received with a CRC error.

RSCERRS:

Resource Errors - records the number of correct incoming frames discarded due to lack of memory resources (buffer space or received frame descriptors).

OVRNERRS:

Overrun Errors - counts the number of received frame sequences lost because the memory bus was not available in time to transfer them.

ACTION COMMANDS

The 82586 executes a 'program' that is made up of action commands in the Command List. As shown in Figure 5, each command contains the command field, status and control fields, link to the next action command in the CL, and any command-specific parameters. This command format is called the Command Block.

The 82586 has a repertoire of 8 commands:

NOP
Setup Individual Address
Configure
Setup Multicast Address
Transmit
TDR
Diagnose
Dump

NOP

This command results in no action by the 82586, except as performed in normal command processing. It is present to aid in Command List manipulation.

NOP command includes the following fields:

STATUS word (written by 82586):

C	(Bit 15)	• Command Completed
B	(Bit 14)	• Busy Executing Command
OK	(Bit 13)	• Error Free Completion

COMMAND word:

EL	(Bit 15)	• End of Command List
S	(Bit 14)	• Suspend After Completion
I	(Bit 13)	• Interrupt After Completion
CMD	(Bits 0–2)	• NOP = 0

LINK OFFSET: Address of next Command Block

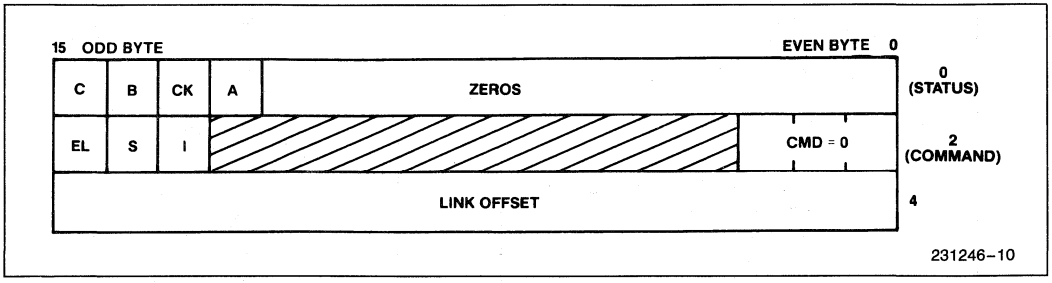


Figure 11. The NOP Command Block

IA-SETUP

This command loads the 82586 with the Individual Address. This address is used by the 82586 for recognition of Destination Address during reception and insertion of Source Address during transmission.

The IA-SETUP command includes the following fields:

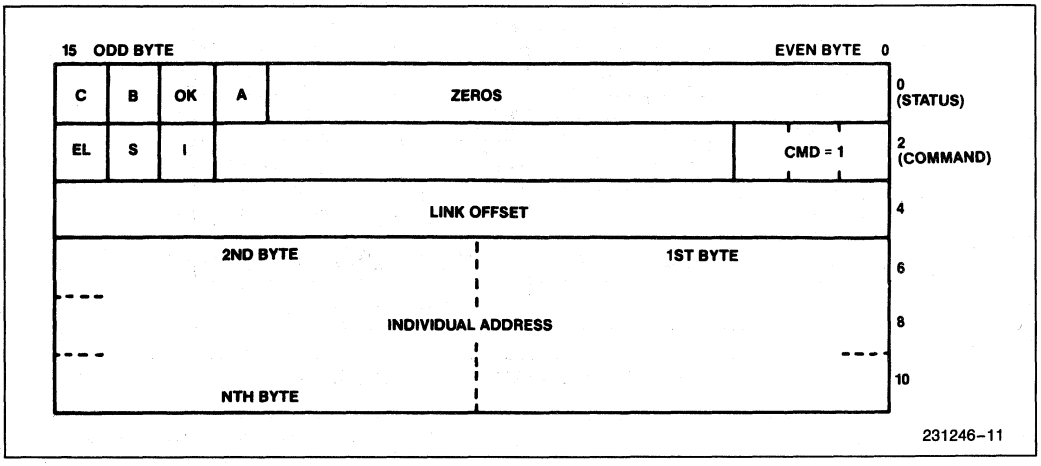


Figure 12. The IA-SETUP Command Block

STATUS word (written by 82586).

C	(Bit 15)	• Command Completed
B	(Bit 14)	• Busy Executing Command
OK	(Bit 13)	• Error Free Completion
A	(Bit 12)	• Command Aborted

COMMAND word:

EL	(Bit 15)	• End of Command List
S	(Bit 14)	• Suspend After Completion
I	(Bit 13)	• Interrupt After Completion
CMD	(Bits 0–2)	• IA-SETUP = 1

LINK OFFSET: Address of next Command Block

INDIVIDUAL ADDRESS: Individual Address parameter

The least significant bit of the Individual Address parameter must be zero for IEEE 802.3/Ethernet. However, no enforcement of 0 is provided by the 82586. Thus, an Individual Address with least significant bit 1, is possible.

CONFIGURE

The CONFIGURE command is used to update the 82586 operating parameters.

The CONFIGURE command includes the following fields:

STATUS word (written by 82586):

C	(Bit 15)	• Command Completed
B	(Bit 14)	• Busy Executing Command
OK	(Bit 13)	• Error Free Completion
A	(Bit 12)	• Command Aborted

COMMAND word:

EL	(Bit 15)	• End of Command List
S	(Bit 14)	• Suspend After Completion
I	(Bit 13)	• Interrupt After Completion
CMD	(Bits 0–2)	• Configure = 2

LINK OFFSET: Address of next Command Block

Byte 6–7:

BYTE CNT	(Bits 0–3)	• Byte Count, Number of bytes including this one, holding the parameters to be configured. A number smaller than 4 is interpreted as 4. A number greater than 12 is interpreted as 12.

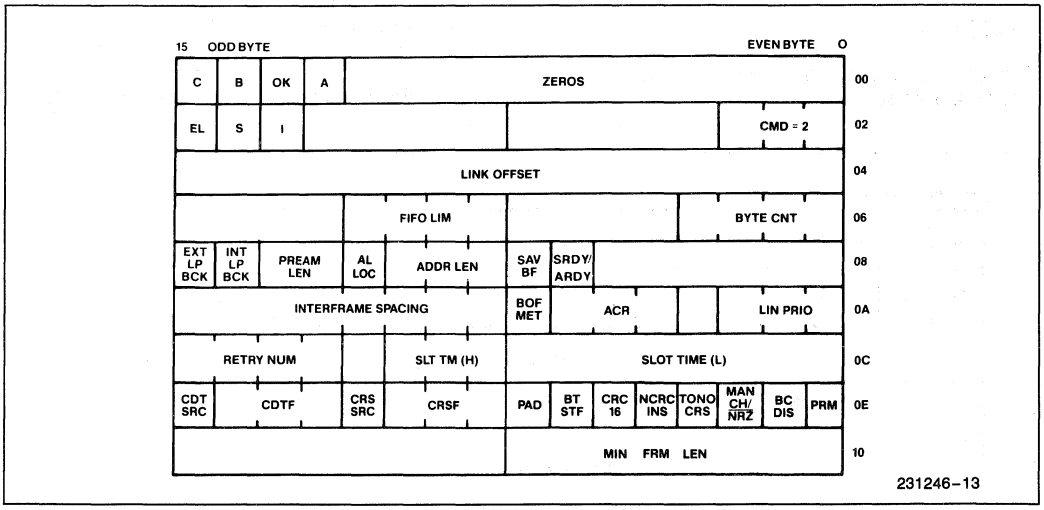


Figure 13. The CONFIGURE Command Block

FIFO-LIM	(Bits 8–11)	• Value of FIFO Threshold.

Byte 8–9:

SRDY/ARDY	(Bit 6)	
	0	• SRDY/ARDY pin operates as ARDY (internal synchronization).
	1	• SRDY/ARDY pin operates as SRDY (external synchronization).
SAV-BF	(Bit 7)	
	0	• Received bad frames are not saved in memory.
	1	• Received bad frames are saved in memory.
ADD-LEN	(Bits 8–10)	• Number of address byes. NOTE: 7 is interpreted as 0.
AL-LOC	(Bit 11)	
	0	• Address and Length Fields separated from data and associated with Transmit Command Block or Receive Frame Descriptor. For transmitted Frame, Source Address is inserted by the 82586.
	1	• Address and Length Fields are part of the Transmit/Receive data buffers, including Source Address (which is not inserted by the 82586).
PREAM-LEN	(Bits 12–13)	• Preamble Length including Beginning of Frame indicator: 00 - 2 bytes 01 - 4 bytes 10 - 8 bytes 11 - 16 bytes
INT-LPBCK	(Bit 14)	• Internal Loopback
EXT-LPBCK	(Bit 15)	• External Loopback. NOTE: Bits 14 and 15 configured to 1, cause Internal Loopback.

Byte 10–11:

LIN-PRIO	(Bits 0–2)	• Linear Priority
ACR	(Bits 4–6)	• Accelerated Contention Resolution (Exponential Priority)
BOF-MET	(Bit 7)	• Exponential Backoff Method 0 - IEEE 802.3/Ethernet 1 - Alternate Method

INTER FRAME SPACING	(Bits 8–15)	• Number indicating the Interframe Spacing in TxC period units.

Byte 12–13:

SLOT-TIME (L)	(Bits 0–7)	• Slot Time Number, Low Byte
SLT-TM (H)	(Bits 8–10)	• Slot Time Number, High Bits
RETRY-NUM	(Bits 12–15)	• Maximum Number of Transmission Retries on Collisions

Byte 14–15:

PRM	(Bit 0)	• Promiscuous Mode
BC-DIS	(Bit 1)	• Broadcast Disable
MANCH/$\overline{NRZ}$	(Bit 2)	• Manchester or NRZ • Encoding/Decoding
	0	• NRZ
	1	• Manchester
TONO-CRS	(Bit 3)	• Transmit on No Carrier Sense
	0	• Cease Transmission if $\overline{CRS}$ Goes Inactive During Frame Transmission
	1	• Continue Transmission Even if no Carrier Sense
NCRC-INS	(Bit 4)	• No CRC Insertion
CRC-16	(Bit 5)	• CRC Type:
	0	• 32 bit Autodin II CRC Polynomial
	1	• 16 bit CCITT CRC Polynomial
BT-STF	(Bit 6)	• Bitstuffing:
	0	• End of Carrier Mode (Ethernet)
	1	• HDLC like Bitstuffing Mode
PAD	(Bit 7)	• Padding
	0	• No Padding
	1	• Perform Padding by Transmitting Flags for Remainder of Slot Time
CRSF	(Bits 8–9)	• Carrier Sense Filter in Bit Times
CRS-SRC	(Bit 11)	• Carrier Sense Source
	0	• External
	1	• Internal
CDTF	(Bits 12–14	• Collision Detect Filter in Bit Times
CDT-SRC	(Bit 15)	• Collision Detect Source
	0	• External
	1	• Internal

Byte 16:

MIN-FRM-	(Bits 0–7)	• Minimum Number of Bytes in a Frame

CONFIGURATION DEFAULTS

The default values of the configuration parameters are compatible with the IEEE 802.3/Ethernet Standards. RESET configures the 82586 according to the defaults shown in Table 2.

Table 2. 82586 Default Values

Preamble Length (Bytes)	=	8
Address Length (Bytes)	=	6
Broadcast Disable	=	0
CRC-16/CRC-32	=	0
No CRC Insertion	=	0
Bitstuffing/EOC	=	0
Padding	=	0
Min-Frame-Length (Bytes)	=	64
Interframe Spacing (Bits)	=	96
Slot Time (Bits)	=	512
Number of Retries	=	15
Linear Priority	=	0
Accelerated Contention Resolution	=	0
Exponential Backoff Method	=	0
Manchester/$\overline{NRZ}$	=	0
Internal CRS	=	0
CRS Filter	=	0
Internal CDT	=	0
CDT Filter	=	0
Transmit On No CRS	=	0
FIFO THRESHOLD	=	8
SRDY/$\overline{ARDY}$	=	0
Save Bad Frame	=	0
Address/Length Location	=	0
INT Loopback	=	0
EXT Loopback	=	0
Promiscuous Mode	=	0

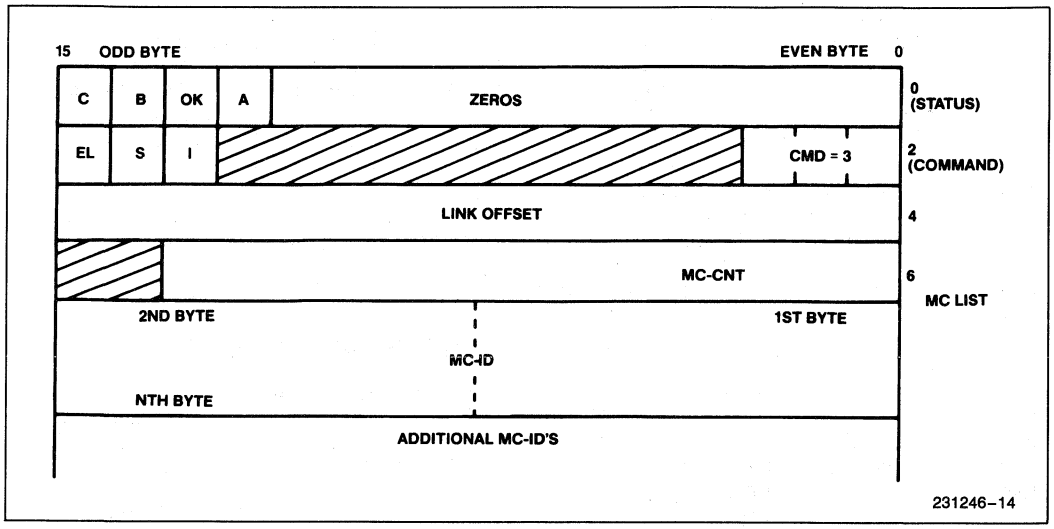


Figure 14. The MC-SETUP Command Block

MC-SETUP

This command sets up the 82586 with a set of Multicast Addresses. Subsequently, incoming frames with Destination Addresses from this set are accepted.

The MC-SETUP command includes the following fields:

STATUS word (written by 82586):

C	(Bit 15)	• Command Completed
B	(Bit 14)	• Busy Executing Command
OK	(Bit 13)	• Error Free Completion
A	(Bit 12)	• Command Aborted

COMMAND word:

EL	(Bit 15)	• End of Command List
S	(Bit 14)	• Suspend After Completion
I	(Bit 13)	• Interrupt After Completion
CMD	(Bits 0–2)	• MC-SETUP = 3

LINK OFFSET: Address of next Command Block

MC-CNT: A 14-bit field indicating the number of bytes in the MC-LIST field. MC-CNT is truncated to the nearest multiple of Address Length (in bytes). Issuing a MC-SETUP command with MC-CNT = 0 disables reception of any incoming frame with a Multicast Address.

MC-LIST: A list of Multicast Addresses to be accepted by the 82586. Note that the most significant byte of an address is followed immediately by the least significant byte of the next address. Note also that the least significant bit of each Multicast Address in the set must be a one.

The Transmit-Byte-Machine maintains a 64-bit HASH table used for checking Multicast Addresses during reception.

An incoming frame is accepted if it has a Destination Address whose least significant bit is a one, and after hashing points to a bit in the HASH table whose value is one. The hash function is selecting bits 2 to 7 of the CRC register. RESET causes the HASH table to become all zeros.

After the Transmit-Byte-Machine reads a MC-SETUP command from TX-FIFO, it clears the HASH table and reads the bytes in groups whose length is determined by the ADDRESS length. Each group is hashed using CRC logic and the bit in the HASH table to which bits 2–7 of the CRC register point is set to one. A group that is not complete has no effect on the HASH table. Transmit-Byte-Machine notifies CU after completion.

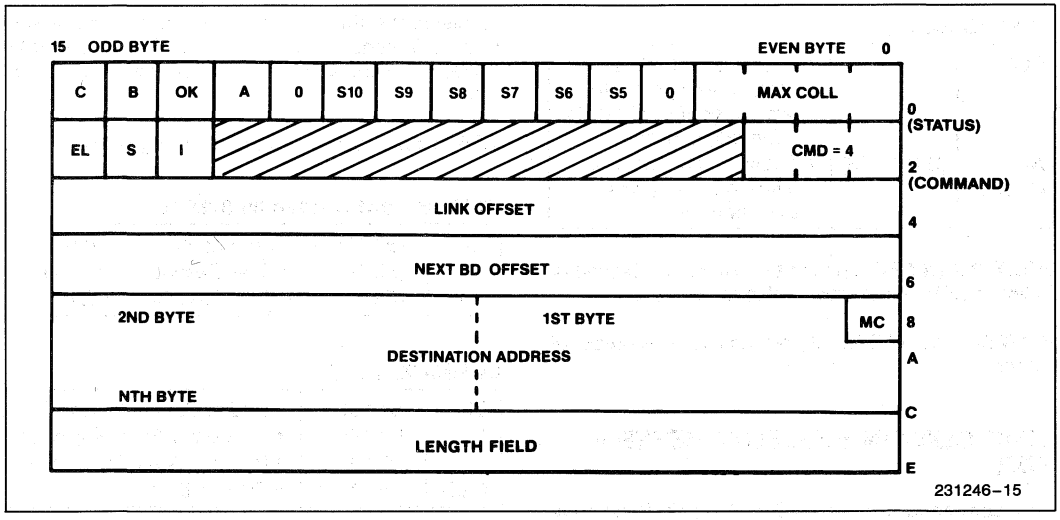


Figure 15. The Transmit Command Block

TRANSMIT

The TRANSMIT command causes transmission (and if necessary retransmission) of a frame.

TRANSMIT CB includes the following fields:

STATUS word (written by 82586):

C	(Bit 15)	• Command Completed
B	(Bit 14)	• Busy Executing Command
OK	(Bit 13)	• Error Free Completion
A	(Bit 12)	• Command Aborted
S10	(Bit 10)	• No Carrier Sense signal during transmission (between beginning of Destination Address and end of Frame Check Sequence).
S9	(Bit 9)	• Transmission unsuccessful (stopped) due to loss of Clear-to-Send signal.
S8	(Bit 8)	• Transmission unsuccessful (stopped) due to DMA underrun, (i.e. data not supplied from the sytem for transmission).
S7	(Bit 7)	• Transmission had to Defer to traffic on the link.
S6	(Bit 6)	• Heart Beat, indicates that during Interframe Spacing period after the previous transmission, a pulse was detected on the Collision Detect pin.
S5	(Bit 5)	• Transmission attempt stopped due to number of collisions exceeding the maximum number of retries.
MAX-COLL	(Bits 3–0)	• Number of Collisions experienced by this frame. S5 = 1 and MAX-COLL = 0 indicates that there were 16 collisions.

COMMAND word:

EL	(Bit 15)	• End of Command List
S	(Bit 14)	• Suspend After Completion
I	(Bit 13)	• Interrupt After Completion
CMD	(Bits 0–2)	• TRANSMIT = 4

LINK OFFSET: Address of next Command Block

TBD OFFSET: Address of list of buffers holding the information field. TBD-OFFSET = 0FFFFH indicates that there is no Information field.

DESTINATION ADDRESS: Destination Address of the frame.

LENGTH FIELD: Length field of the frame.

STATUS word:

EOF		• Indicates that this is the Buffer Descriptor of the last buffer of this frame's Information Field.
ACT-COUNT	(Bits 0–13)	• Actual number of data bytes in buffer (can be even or odd).

NEXT BD OFFSET: points to next Buffer Descriptor in list. If EOF is set, this field is meaningless.

BUFFER ADDRESS: 24-bit absolute address of buffer.

TIME DOMAIN REFLECTOMETER - TDR

This command performs a Time Domain Reflectometer test on the serial link. By performing the command, the user is able to identify shorts or opens and their location. Along with transmission of 'All Ones,' the 82586 triggers an internal timer. The timer measures the time elapsed from transmission start until 'echo' is obtained. 'Echo' is indicated by Collision Detect going active or Carrier Sense signal drop.

TDR command includes the following fields:

STATUS word (written by 82586):

C	(Bit 15)	• Command Completed
B	(Bit 14)	• Busy Executing Command
OK	(Bit 13)	• Error Free Completion

COMMAND word:

EL	(Bit 15)	• End of Command List
S	(Bit 14)	• Suspend After Completion
I	(Bit 13)	• Interrupt After Completion
CMD	(Bits 0–2)	• TDR = 5

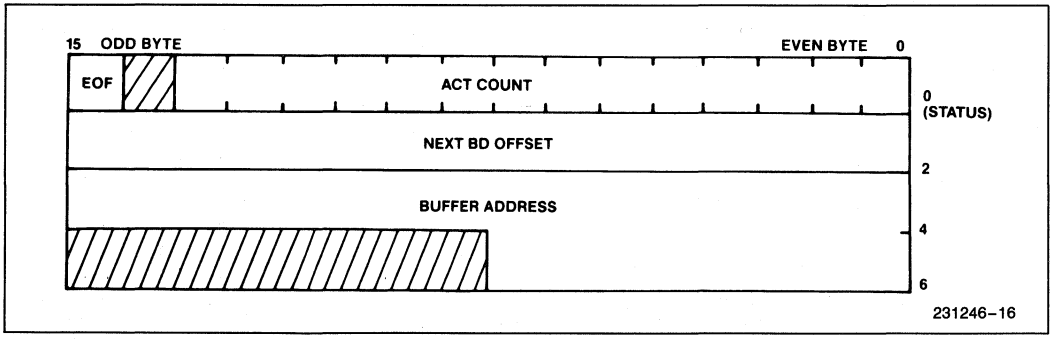


Figure 16. The Transmit Buffer Description

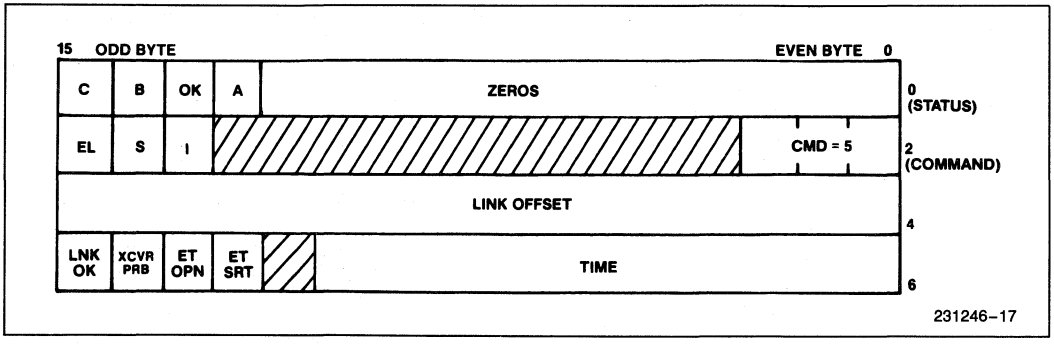


Figure 17. The TDR Command Block

LINK OFFSET: Address of next Command Block

RESULT word:

LNK-OK	(Bit 15)	• No Link Problem Identified
XCVR-PRB	(Bit 14)	• Transceiver Cable Problem identified (valid only in the case of a Transceiver that does not return Carrier Sense during transmission).
ET-OPN	(Bit 13)	• Open on the link identified (valid only in the case of a Transceiver that returns Carrier Sense during transmission).
ET-SRT	(Bit 12)	• Short on the link identified (valid only in the case of a Transceiver that returns Carrier Sense during transmission).
TIME	(Bits 0–10)	• Specifying the distance to a problem on the link (if one exists) in transmit clock cycles.

DUMP

This command causes the contents of over a hundred bytes of internal registers to be placed in memory. It is supplied as a self diagnostic tool, as well as to supply registers of interest to the user.

DUMP command includes the following fields:

STATUS word (written by 82586):

C	(Bit 15)	• Command Completed
B	(Bit 14)	• Busy Executing Command
OK	(Bit 13)	• Error Free Completion

COMMAND word:

EL	(Bit 15)	• End of Command List
S	(Bit 14)	• Suspend After Completion
I	(Bit 13)	• Interrupt After Completion
CMD	(Bits 0–2)	• DUMP = 6

LINK OFFSET: Address of next Command Block

BUFFER OFFSET: This word specifies the offset portion of the memory address which points to the top of the buffer allocated for the dumped registers contents. The length of the buffer is 170 bytes.

DUMP AREA FORMAT

Figure 18 shows the format of the DUMP area. The fields are as follows:

Bytes 00H to 0AH: These bytes correspond to the 82586 CONFIGURE command field.

Bytes 0CH to 11H: The Individual Address Register content. IARO is the Individual Address least significant byte.

Bytes 12H to 13H: Status word of last command block (only bits 0–13).

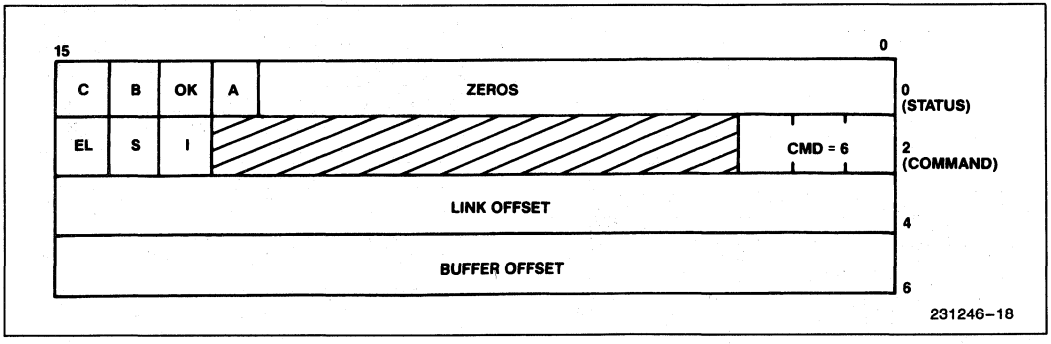


Figure 18. The DUMP Command Block

Bytes 14H to 17H: Content of the Transmit CRC generator. TXCRCRO is the least significant byte. The contents are dependent on the activity before the DUMP command:

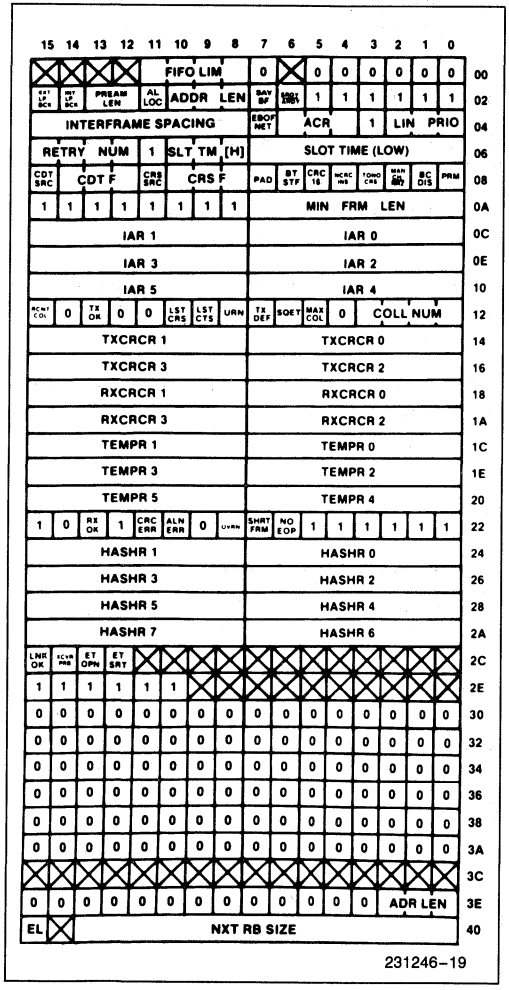
After RESET - 'All Ones.'

After successful transmission - 'All Zeros'.

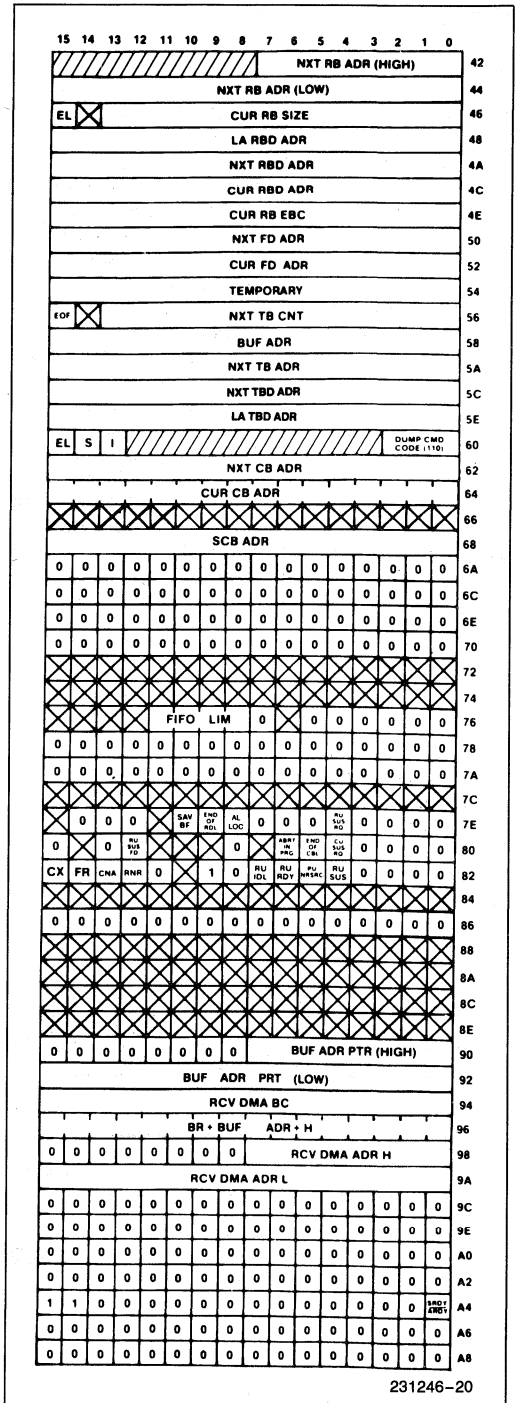
After MC-SETUP command - Generated CRC value of the last MC address, on MC-LIST.

After unsuccessful transmission, depends on where it stopped.

NOTE:
For 16-bit CRC only TXCRCRO and TXCRCR1 are valid.



231246-19



231246-20

Figure 19. The DUMP Area

Bytes 18H to 1BH: Contents of Receive CRC Checker. RXCRCRO is the least significant byte. The contents are dependent on the activity performed before the DUMP command:

After RESET - 'All Ones.'

After good frame reception—

1. For CRC-CCITT - 0ID0FH
2. For CRC-Autodin-II - C704DD7BH

After Bad Frame reception - corresponds to the received information.

After reception attempt, i.e. unsuccessful check for address match, corresponds to the CRC performed on the frame address.

NOTE:
Any frame on the serial link modifies this register contents.

Bytes 1CH to 21H: Temporary Registers.

Bytes 22H to 23H: Receive Status Register. Bits 6, 7, 8, 10, 11 and 13 assume the same meaning as corresponding bits in the Receive Frame Descriptor Status field.

Bytes 24H to 2BH: HASH TABLE.

Bytes 2CH to 2DH: Status bits of the last time TDR command that was performed.

NXT-RB-SIZE: Let N be the last buffer of the last received frame, then NXT-RB-SIZE is the number of bytes of available in the N + 1 buffer. EL - The EL bit of the Receive Buffer Descriptor.

NXT-RB-ADR: Let N be the last Receive Buffer used, then NXT-RB-ADR is the BUFFER-ADDRESS field in the N + 1 Receive-Buffer Descriptor, i.e. the pointer to the N + 1 Receive Buffer.

CUR-RB-SIZE: The number of bytes in the last buffer of the last received frame. EL - The EL bit of the last buffer in the last received frame.

LA-RBD-ADR: Look Ahead Buffer Descriptor, i.e. the pointer to N + 2 Receiver Buffer Descriptor.

NXT-RBD-ADR: Next Receive Buffer Descriptor Address. Similar to LA-RBD-ADR but points to N + 1 Receive Buffer Descriptor.

CUR-RBD-ADR: Current Receive Buffer Descriptor Address. Similar to LA-RBD-ADR, but point to Nth Receive Buffer Descriptor.

CUR-RB-EBC: Current Receive Buffer Empty Byte Count Let N be the currently used Receive Buffer. Then CUR-RB-EBC indicates the Empty part of the buffer, i.e. the ACT-COUNT of buffer N is given by the difference between its SIZE and the CUR-RB-EBC.

NXT-FD-ADR: Next Frame Descriptor Address. Define N as the last Receive Frame Descriptor with bits C = 1 and B = 0, then NXT-FD-ADR is the address of N + 2 Receive Frame Descriptor (with B = C = 0) and is equal to the LINK-ADDRESS field in N + 1 Receive Frame Descriptor.

CUR-FD-ADR: Current Frame Descriptor Address. Similar to next NXT-FD-ADR but refers to N + 1 Receive Frame Descriptor (with B = 1, C = 0).

Bytes 54H to 55H: Temporary register.

NXT-TB-CNT: Next Transmit Buffer Count. Let N be the last transmitted buffer of the TRANSMIT command executed recently, the NXT-TB-CNT is the ACT-COUNT field in the Nth Transmit Buffer Descriptor. EOF - Corresponds to the EOF bit of the Nth Transmit Buffer Descriptor. EOF = 1 indicates that the last buffer accessed by the 82586 during Transmit was the last Transmit Buffer in the data buffer chain associated with the Transmit Command.

BUF-ADR: Buffer Address. The BUF-PTR field in the DUMP-STATUS Command Block.

NXT-TB-AD-L: Next Transmit Buffer Address Low. Let N be the last Transmit Buffer in the transmit buffer chain of the TRANSMIT Command performed recently, then NXT-TB-AD-L are the two least significant bytes of the Nth buffer address.

LA-TB-ADR: Look Ahead Transmit Buffer Descriptor Address. Let N be the last Transmit Buffer in the transmit buffer chain of the TRANSMIT Command performed recently, then LA-TBD-ADR is the NEXT-BD-ADDRESS field of the Nth Buffer Descriptor.

NXT-TBD-ADR: Next Transmit Buffer Descriptor Address. Similar in function to LA-TBD-ADR but related to Transmit Buffer Descriptor N-1. Actually, it is the address of Transmit Buffer Descriptor N.

Bytes 60H, 61H: This is a copy of the 2nd word in the DUMP-STATUS command presently executing.

NXT-CB-ADR: Next Command Block Address. The LINK-ADDRESS field in the DUMP Command Block presently executing. Points to the next command.

CUR-CB-ADR: Current Command Block Address. The address of the DUMP Command Block currently executing.

SCB-ADR: Offset of the System Control Block (SCB).

Bytes 7EH, 7FH:

RU-SUS-RQ (Bit 4) - Receive Unit Suspend Request.

Bytes 80H, 81H:

CU-SUS-RQ (Bit 4) - Command Unit Suspend Request.

END-OF-CBL (Bit 5) - End of Command Block List. If "1" indicates that DUMP-STATUS is the last command in the command chain.

ABRT-IN-PROG (Bit 6) - Command Unit Abort Request.

RU-SUS-FD (Bit 12) - Receive Unit Suspend Frame Descriptor Bit. Assume N is the Receive Frame Descriptor used recently, then RU-SUS-FD is equivalent to the S bit of N + 1 Receive Frame Descriptor.

Bytes 82H, 83H:

RU-SUS (Bit 4) - Receive Unit in SUSPENDED state.

RU-NRSRC (Bit 5) - Receive Unit in NO RESOURCES state.

RU-RDY (Bit 6) - Receive Unit in READY state.

RU-IDL (Bit 7) - Receive Unit in IDLE state.

RNR (Bit 12) - RNR Interrupt in Service bit.

CNA (Bit 13) - CNA Interrupt in Service bit.

FR (Bit 14) - FR Interrupt in Service bit.

CX (Bit 15) - CX Interrupt in Service bit.

Bytes 90H to 93H:

BUF-ADR-PTR - Buffer pointer is the absolute address of the bytes following the DUMP Command block.

Bytes 94H to 95H:

RCV-DMA-BC - Receive DMA Byte Count. This field contains number of bytes to be transferred during the next Receive DMA operation. The value depends on AL-LOCation configuration bit.

1. If AL-LOCation = 0 then RCV-DMA-BC = (2 times ADDR-LEN plus 2) if the next Receive Frame Descriptor has already been fetched.
2. If AL-LOCation = 1 then it contains the size of the next Receive Buffer.

BR + BUF − PTR + 96H - Sum of Base Address plus BUF − PTR field and 96H.

RCV-DMA-ADR - Receive DMA absolute Address. This is the next RCV-DMA start address. The value depends on AL-LOCation configuration bit.

1. If AL-LOCation = 0, then RCV-DMA-ADR is the Destination Address field located in the next Receive Frame Descriptor.
2. If AL-LOCation = 1, then RCV-DMA-ADR is the next Receive Data Buffer Address.

The following nomenclature has been used in the DUMP table:

0	• The 82586 writes zero in this location.
1	• The 82586 writes one in this location.
X	• The 82586 writes zero or one in this location.
///	• The 82586 copies this location from the corresponding position in the memory structure.

DIAGNOSE

The DIAGNOSE Command triggers an internal self test procedure of backoff related registers and counters.

The DIAGNOSE command includes the following:

STATUS word (written by 82586):

C	(Bit 15)	• Command Completed
B	(Bit 14)	• Busy Executing Command
OK	(Bit 13)	• Error Free Completion
FAIL	(Bit 11)	• Indicates that the Self Test Procedured Failed

COMMAND word:

EL	(Bit 15)	• End of Command List
S	(Bit 14)	• Suspend After Completion
I	(Bit 13)	• Interrupt After Completion
CMD	(Bits 0–2)	• DIAGNOSE = 7

LINK OFFSET: Address of next Command Block.

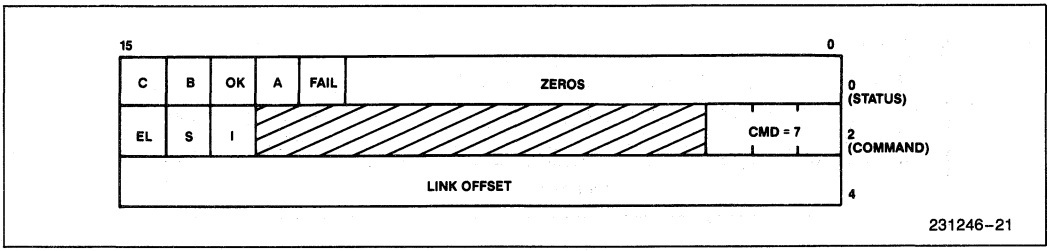


Figure 20. The DIAGNOSE Command Block

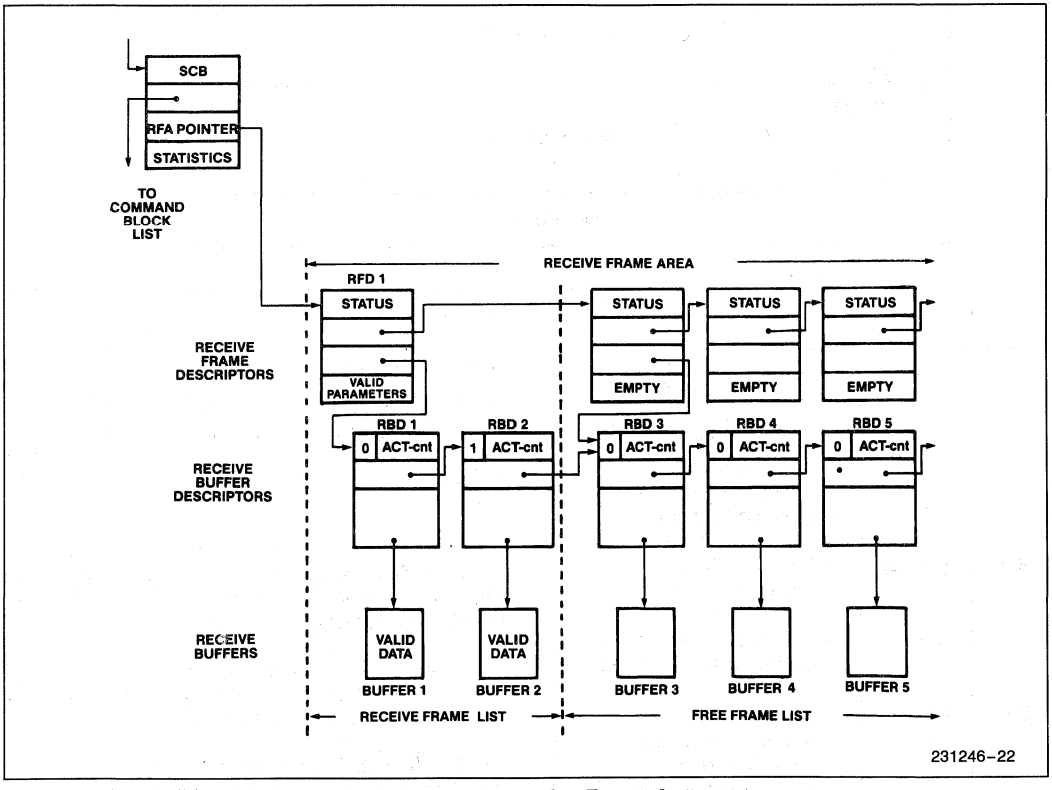


Figure 21. The Receive Frame Area

RECEIVE FRAME AREA (RFA)

The Receive Frame Area, RFA, is prepared by the host CPU, data is placed into the RFA by the 82586 as frames are received. RFA consists of a list of Receive Frame Descriptors (FD), each of which is associated with a frame. RFA-OFFSET field of SCB points to the first FD of the chain; the last FD is identified by the End-of-Listing flag (EL). See Figure 21.

FRAME DESCRIPTOR (FD) FORMAT

The FD includes the following fields:

STATUS word (set by the 82586):

C	(Bit 15)	• Completed Storing Frame.
B	(Bit 14)	• FD was Consumed by RU.

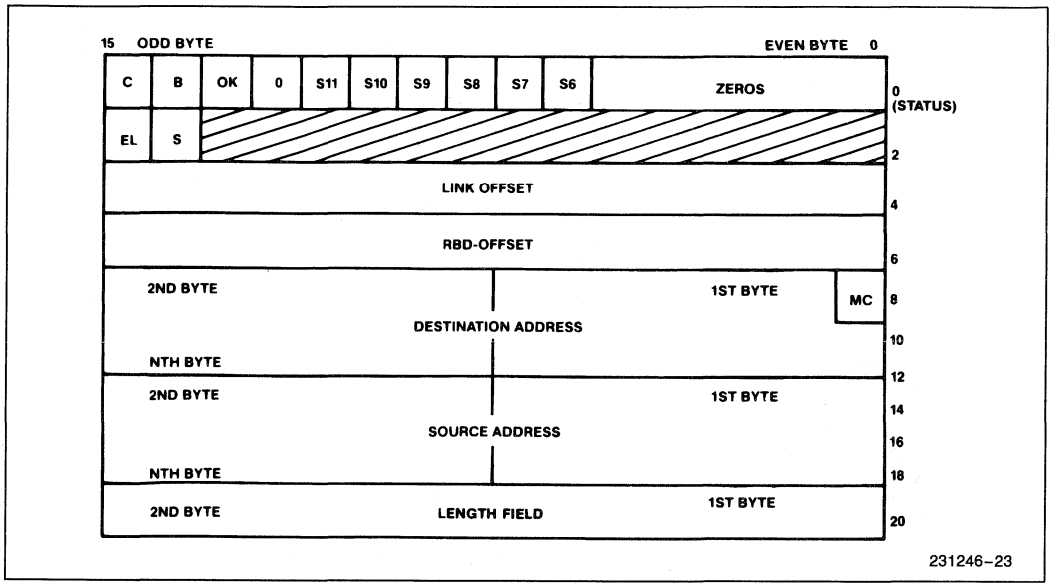


Figure 22. The Frame Descriptor (FD) Format

OK	(Bit 13)	• Frame received successfully. If this bit is set, then all others will be reset; if it is reset, then the other bits will indicate the nature of the error.
S11	(Bit 11)	• Received Frame Experienced CRC Error.
S10	(Bit 10)	• Received Frame Experienced an Alignment Error.
S9	(Bit 9)	• RU ran out of resources during reception of this frame.
S8	(Bit 8)	• RCV-DMA Overrun.
S7	(Bit 7)	• Received frame had fewer bits than configured Minimum Frame Length.
S6	(Bit 6)	• No EOF flag detected (only when configured to Bitstuffing).

COMMAND word:

EL	(Bit 15)	• Last FD in the List.
S	(Bit 14)	• RU should be suspended after receiving this frame.

LINK OFFSET: Address of next FD in list.

RBD-OFFSET: (initially prepared by the CPU and later may be updated by 82586): Address of the first RBD that represents the Information Field. RBD-OFFSET = 0FFFFH means there is no Information Field.

DESTINATION ADDRESS (written by 82586): Contains Destination Address of received frame. The length in bytes, it is determined by the Address Length configuration parameter.

SOURCE ADDRESS (written by 82586): Contains Source Address of received frame. Its length is the same as DESTINATION ADDRESS.

LENGTH FIELD (written by 82586): Contains the 2 byte Length or Type Field of received frame.

RECEIVE BUFFER DESCRIPTOR FORMAT

The Receive Buffer Descriptor (RBD) holds information about a buffer; size and location, and the means for forming a chain of RBDs, (forward pointer and end-of-frame indication).

The Buffer Descriptor contains the following fields.

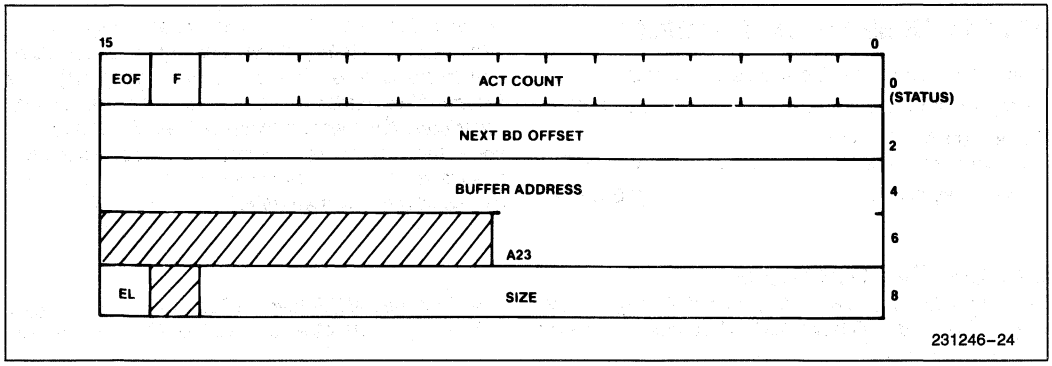


Figure 23. The Receive Buffer Descriptor (RBD) Format

STATUS word (written by the 82586).

EOF	(Bit 15)	• Last buffer in received frame.
F	(Bit 14)	• ACT COUNT field is valid.
ACT COUNT	(Bits 0–13)	• Number of bytes in the buffer that are actually occupied.

NEXT RBD OFFSET: Address of next BD in list of BD's.

BUFFER ADDRESS: 24-bit absolute address of buffer.

EL/SIZE:

EL	(BIT 15)	• Last BD in list.
SIZE	(Bits 0–13)	• Number of bytes the buffer is capable of holding.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .....0°C to 70°C
Storage Temperature.....−65°C to 150°C
Voltage on Any Pin with
Respect to Ground.....−1.0V to +7V
Power Dissipation.....3.0 Watts

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS

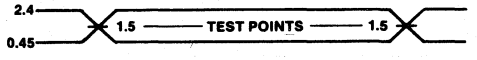
T_A = 0°C to 70°C, T_C = 0°C to 105°C, V_{CC} = 5V ±10%, CLK has MOS levels (See V_{MIL}, V_{MIH}, V_{MOL}, V_{MOH}). $\overline{TxC}$ and $\overline{RxC}$ have 82C501 compatible levels (V_{MIL}, V_{TIH}, V_{RIH}). All other signals have TTL levels (see V_{IL}, V_{IH}, V_{OL}, $_{OH}$).

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage (TTL)	−0.5	+0.8	V	
V_{IH}	Input High Voltage (TTL)	2.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage (TTL)		0.45	V	I_{OL} = 2.5 mA
V_{OH}	Output High Voltage (TTL)	2.4		V	I_{OH} − 400 μA
V_{MIL}	Input Low Voltage (MOS)	−0.5	0.6	V	
V_{MIH}	Input High Voltage (MOS)	3.9	V_{CC} + 0.5	V	
V_{TIH}	Input High Voltage ($\overline{TxC}$)	3.3	V_{CC} + 0.5	V	
V_{RIH}	Input High Voltage ($\overline{RxC}$)	3.0	V_{CC} + 0.5	V	
V_{MOL}	Output Low Voltage (MOS)		0.45	V	I_{OL} 2.5 mA
V_{MOH}	Output High Voltage (MOS)	V_{CC} − 0.5		V	I_{OH} − 400 μA
I_{LI}	Input Leakage Current		±10	μA	0 ≤ V_{IN} ≤ V_{CC}
I_{LO}	Output Leakage Current		±10	μA	0.45 ≤ V_{OUT} ≤ V_{CC}
C_{IN}	Capacitance of Input Buffer		10	pF	FC = 1 MHz
C_{OUT}	Capacitance of Output Buffer		20	pF	FC = 1 MHz
I_{CC}	Power Supply Current		550 450	mA	T_A = 0°C T_A = 70°C

SYSTEM INTERFACE A.C. TIMING CHARACTERISTICS

T_A = 0°C to 70°C, T_C = 0°C to 105°C, V_{CC} = 5V ±10%. Figures 24 and 25 define how the measurements should be done.

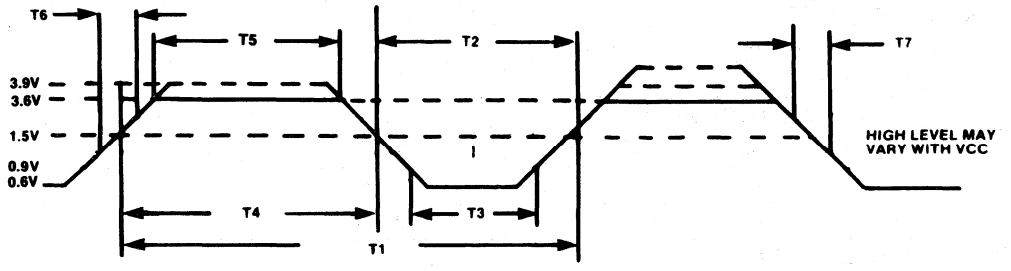
INPUT AND OUTPUT WAVEFORMS FOR A.C. TESTS



231246-25

AC Testing Inputs are Driven at 2.4V for a Logic 1 and 0.45 for a Logic 0. Timing measurements are made at 1.5V for both a Logic 1 and 0

Figure 24. TTL Input/Output Voltage Levels for Timing Measurements



231246-26

MOS I/O measurements are taken at 0.1 and 0.9 of the voltage swing

Figure 25. System Clock CMOS Input Voltage Levels for Timing Measurements

INPUT TIMING REQUIREMENTS*

Symbol	Parameter	82586-6 (6 MHz)		82586 (8 MHz)		82586-10 (10 MHz)		Comments
		Min	Max	Min	Max	Min	Max	
T1	CLK Cycle Period	166	2000	125	2000	100	200	
T2	CLK Low Time at 1.5V	73	1000	55	1000	44	1000	
T3	CLK Low Time at 0.9V			42.5	1000	42.5	1000	
T4	CLK High Time at 1.5V	73		55		44		
T5	CLK High Time at 3.6V			42.5		42.5		
T6	CLK Rise Time		15		15		12	Note 1
T7	CLK Fall Time		15		15		12	Note 2
T8	Data in Setup Time	20		20		15		
T9	Data in Hold Time	10		10		10		
T10	Async RDY Active Setup Time	20		20		15		Note 3
T11	Async RDY Inactive Setup Time	35		35		25		Note 3
T12	Async RDY Hold Time	15		15		15		Note 3
T13	Synchronous Ready/Active Setup	35		35		35		
T14	Synchronous Ready Hold Time	0		0		0		
T15	HLDA Setup Time	20		20		20		Note 3
T16	HLDA Hold Time	10		10		5		Note 3
T17	Reset Setup Time	20		20		20		Note 3
T18	Reset Hold Time	10		10		10		Note 3
T19	CA Pulse Width	1 T1		1 T1		1 T1		
T20	CA Setup Time	20		20		20		Note 3
T21	CA Hold Time	10		10		10		Note 3

OUTPUT TIMINGS**

Symbol	Parameter	Min	Max	Min	Max	Min	Max	Comments
T22	DT/R Valid Delay	0	60	0	60	0	44	
T23	$\overline{WR}$, $\overline{DEN}$ Active Delay	0	70	0	70	0	56	
T24	$\overline{WR}$, $\overline{DEN}$ Inactive Delay	10	65	10	65	10	45	
T25	Int. Active Delay	0	85	0	85	0	70	Note 4
T26	Int. Inactive Delay	0	85	0	85	0	70	Note 4
T27	Hold Active Delay	0	85	0	85	0	70	Note 4
T28	Hold Inactive Delay	0	85	0	85	0	70	Note 4
T29	Address Valid Delay	0	55	0	55	0	50	
T30	Address Float Delay	0	50	0	50	12	50	
T31	Data Valid Delay	0	55	0	55	0	50	Note 7
T32	Data Hold Time	0		0		0		
T33	Status Active Delay	10	60	10	60	10	45	

OUTPUT TIMINGS** (Continued)

Symbol	Parameter	82582-6 (6 MHz)		82586 (8 MHz)		82586-10 (10 MHz)		Comments
		Min	Max	Min	Max	Min	Max	
T34	Status Inactive Delay	10	70	10	70	10	50	Note 8
T35	ALE Active Delay	0	45	0	45	0	35	Note 5
T36	ALE Inactive Delay	0	45	0	45	0	37	Note 5
T37	ALE Width	T2-10		T2-10		T2-10		Note 5
T38	Address Valid to ALE Low	T2-40		T2-30		T2-25		
T39	Address Hold to ALE Inactive	T4-10		T4-10		T4-10		
T40	$\overline{RD}$ Active Delay	10	95	10	95	10	95	
T41	$\overline{RD}$ Inactive Delay	10	70	10	70	10	70	
T42	$\overline{RD}$ Width	2T1-50		2T1-50		2T1-46		
T43	Address Float to $\overline{RD}$ Active	10		10		0		
T44	$\overline{RD}$ Inactive to Address Active	T1-40		T1-40		T1-34		
T45	$\overline{WR}$ Width	2T1-40		2T1-40		2T1-34		
T46	Data Hold After $\overline{WR}$	T2-25		T2-25		T2-25		
T47	Control Inactive After Reset	0	60	0	60	0	60	Note 6

*All units are in ns.
**CL on all outputs is 20-200 pF unless otherwise specified.

NOTES:
1. 1.0V to 3.5V
2. 3.5V to 1.0V
3. To guarantee recognition at next clock
4. CL = 50 pF
5. CL = 100 pF
6. Affects:
   MIN MODE: $\overline{RD}$, $\overline{WR}$, DT/$\overline{R}$, $\overline{DEN}$
   MAX MODE: $\overline{S0}$, $\overline{S1}$
7. High address lines (A16-A24, BHE) become valid one clock before T1 only on first memory cycle after the 82586 acquired the bus.
8. $\overline{S1}$, S0 go inactive just prior to T4.

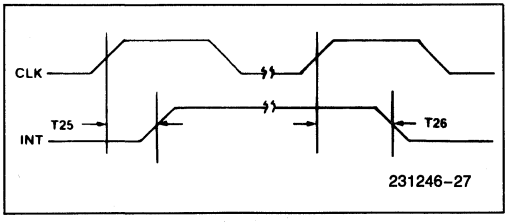


Figure 26. INT Output Timing

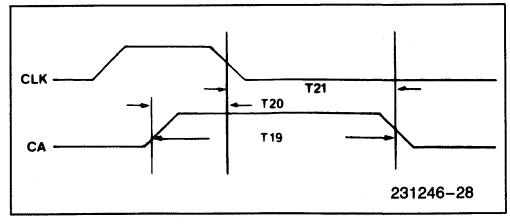


Figure 27. CA Input Timing

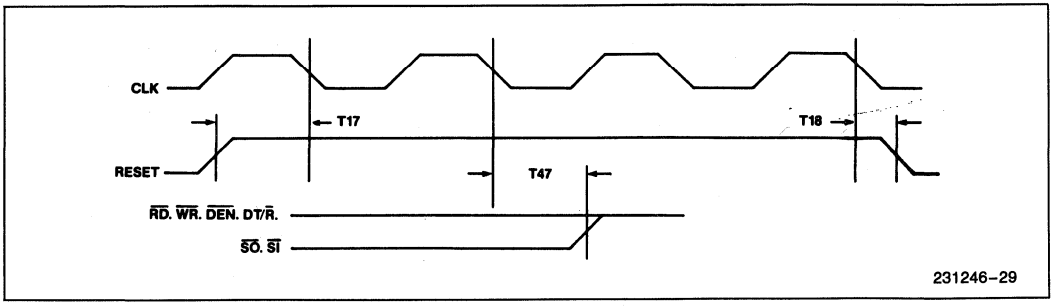


Figure 28. RESET Timing

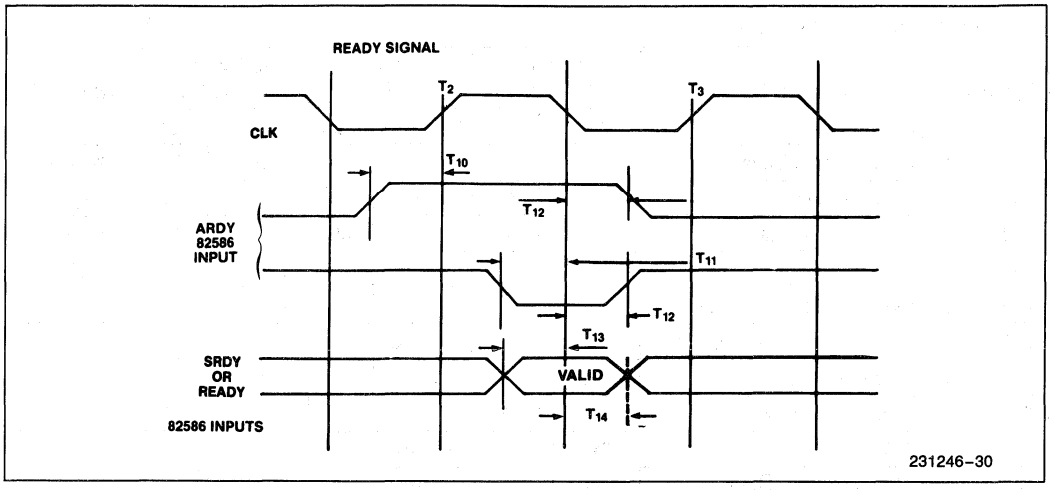


Figure 29. ARDY and SRDY Timings Relative to CLK

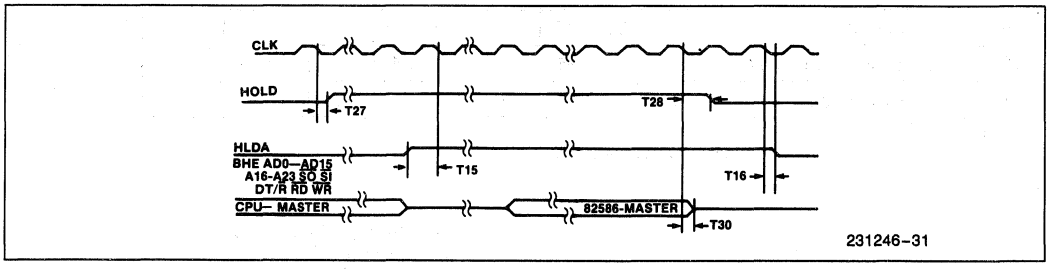


Figure 30. HOLD/HLDA Timing Relative to CLK

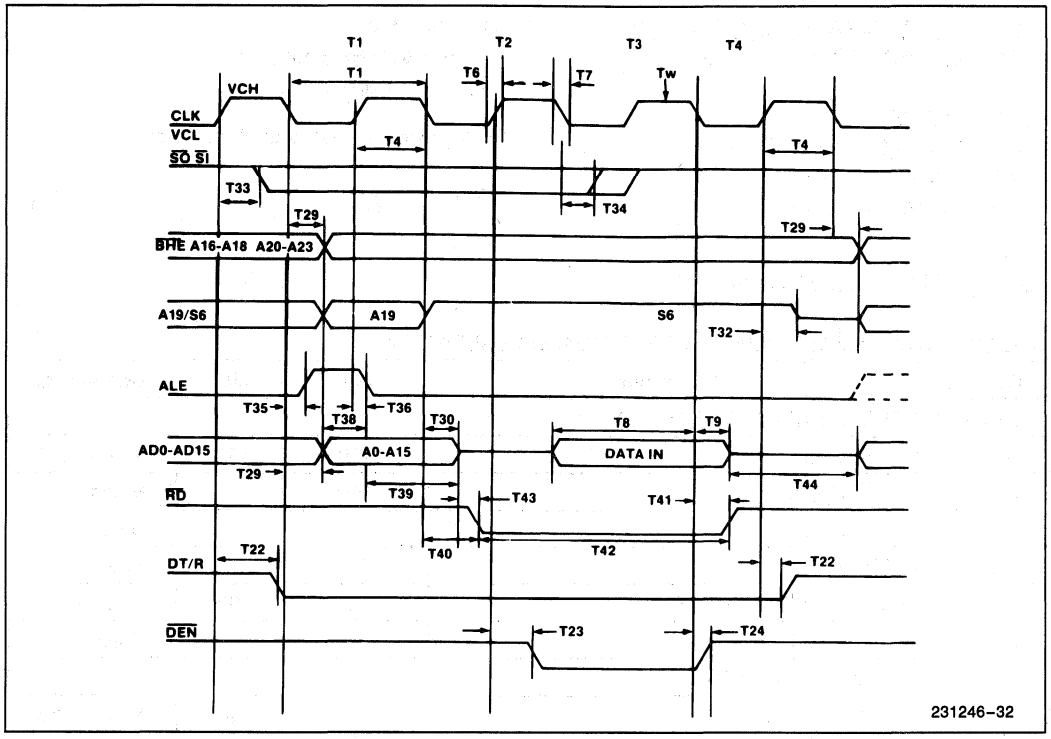


Figure 31. Read Cycle Timing

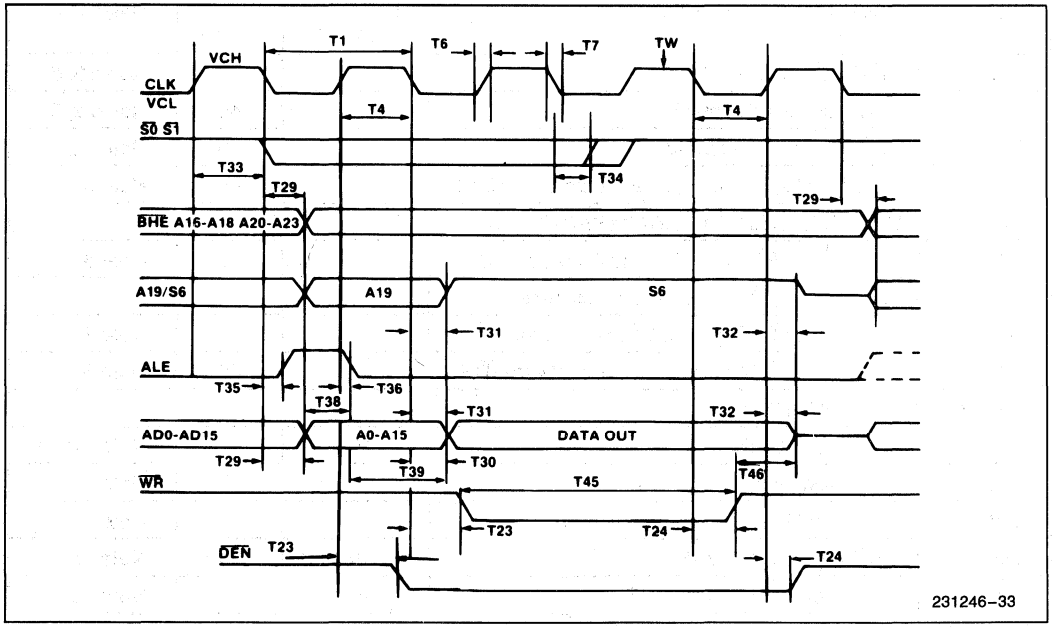


Figure 32. Write Cycle Timing

SERIAL INTERFACE A.C. TIMING CHARACTERISTICS

CLOCK SPECIFICATION

Applies for $\overline{TxC}$, $\overline{RxC}$ for NRZ:

f min = 100 kHz ± 100 ppm

f max = 10 MHz ± 100 ppm

for Manchester:

f min = 500 kHz ± 100 ppm

f max = 10 MHz ± 100 ppm

for Manchester, symmetry is needed:

$$T51, T52 = \frac{1}{2f} \pm 5\%$$

A.C. CHARACTERISTICS

TRANSMIT AND RECEIVE TIMING PARAMETER SPECIFICATION*

Symbol	Parameter	Min	Max	Comments
TRANSMIT CLOCK PARAMETERS				
T48	$\overline{TxC}$ Cycle	100	1000	Notes 14, 2
T48	$\overline{TxC}$ Cycle	100		Notes 14, 3
T49	$\overline{TxC}$ Rise Time		5	Note 14
T50	$\overline{TxC}$ Fall Time		5	Note 14
T51	$\overline{TxC}$ High Time @ 3.0V	40	1000	Note 14
T52	$\overline{TxC}$ Low Time @0.9V	40		Notes 14, 4
TRANSMIT DATA PARAMETERS				
T53	TxD Rise Time		10	Notes 5, 13
T54	TxD Fall Time		10	Notes 5, 13
T55	TxD Transition-Transition	Min (T51, T52) − 7		Notes 2, 5
T56	$\overline{TxC}$ Low to TxD Valid		40	Notes 3, 5
T57	$\overline{TxC}$ Low to TxD Transition		40	Notes 2, 5
T58	$\overline{TxC}$ High to TxD Transition		40	Notes 2, 5
T59	$\overline{TxC}$ Low to TxD High at the Transmission End		40	Note 5
REQUEST TO SEND/CLEAR TO SEND PARAMETERS				
T60	$\overline{TxC}$ Low to $\overline{RTS}$ Low. Time to Activate $\overline{RTS}$		40	Note 6
T61	$\overline{CTS}$ Valid to $\overline{TxC}$ Low. $\overline{CTS}$ Setup Time	45		
T62	$\overline{TxC}$ Low to $\overline{CTS}$ Invalid. $\overline{CTS}$ Hold Time	20		Note 7
T63	$\overline{TxC}$ Low to $\overline{RTS}$ High, time to Deactivate $\overline{RTS}$		40	Note 6
RECEIVE CLOCK PARAMETERS				
T64	$\overline{RxC}$ Clock Cycle	100		Notes 15, 3
T65	$\overline{RxC}$ Rise Time		5	Note 15
T66	$\overline{RxC}$ Fall Time		5	Note 15
T67	$\overline{RxC}$ High Time @ 2.7V	36	1000	Note 15
T68	$\overline{RxC}$ Low Time @0.9V	40		Note 15

*All units are in ns.

A.C. CHARACTERISTICS (Continued)

TRANSMIT AND RECEIVE TIMING PARAMETER SPECIFICATION* (Continued)

Symbol	Parameter	Min	Max	Comments
RECEIVE DATA PARAMETERS				
T69	RxD Setup Time	30		Note 1
T70	RxD Hold Time	30		Note 1
T71	RxD Rise Time		10	Note 1
T72	RxD Fall Time		10	Note 1
CARRIER SENSE/COLLISION DETECT PARAMETERS				
T73	$\overline{CDT}$ Valid to $\overline{TxC}$ High Ext. Collision Detect Setup Time	30		Note 12
T74	$\overline{TxC}$ High to $\overline{CDT}$ Inactive. $\overline{CDT}$ Hold Time	20		Note 12
T75	$\overline{CDT}$ Low to Jamming Start			Note 8
T76	$\overline{CRS}$ Valid to $\overline{TxC}$ High Ext. Carrier Sense Setup Time	30		Note 12
T77	$\overline{TxC}$ High to $\overline{CRS}$ Inactive. $\overline{CRS}$ Hold Time	20		Note 12
T78	$\overline{CRS}$ Low to Jamming Start			Note 9
T79	Jamming Period			Note 10
T80	$\overline{CRS}$ Inactive Setup Time to $\overline{RxC}$ High End of Receive Frame	60		
T81	$\overline{CRS}$ Active Hold Time from $\overline{RxC}$ High	3		
INTERFRAME SPACING PARAMETER				
T82	Inter Frame Delay			Note 11

*All units are in ns.

NOTES:
1. TTL levels
2. Manchester only
3. NRZ only
4. Manchester requires 50% duty cycle
5. 1 TTL load + 50 pF
6. 1 TTL load + 100 pF
7. Abnormal end of transmission. $\overline{CTS}$ expires before $\overline{RTS}$
8. Programmable value:
   T75 = NCDF × T48 + (12.5 to 23.5) × T48 if collision occurs after preamble
   NCDF—The collision detection filter configuration value
9. Programmable value:
   T78 = NCSF × T48 + (12.5 to 23.5) × T48
   NCSF—The carrier sense filter configuration value
   TBD is a function of internal/external carrier sense bit
10. T79 = 32 × T48
11. Programmable value:
    T82 = NIFS × T48
    NIFS—the IFS configuration value
*12. To guarantee recognition on the next clock
13. Applies to TTL levels
14. 82C501 compatible levels, see Figure 34
15. 82C501 compatible levels, see Figure 35

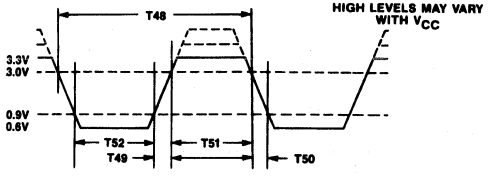
A.C. TIMING CHARACTERISTICS

Input and Output Waveforms for AC Tests

231246-34

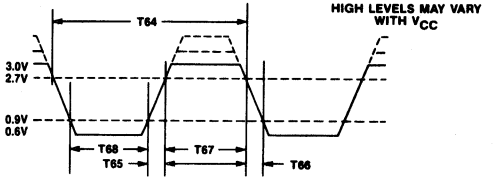
AC testing inputs are driven at 2.4V for a Logic 1 and 0.45 for a Logic 0. Timing measurements are made at 1.5V for both a Logic 1 and 0.

Figure 33. TTL Input/Output Voltage Levels for Timing Measurements



231246-35

Figure 34. $\overline{TxC}$ Input Voltage Levels for Timing Measurements



231246-36

Figure 35. $\overline{RxC}$ Input Voltage Levels for Timing Measurements

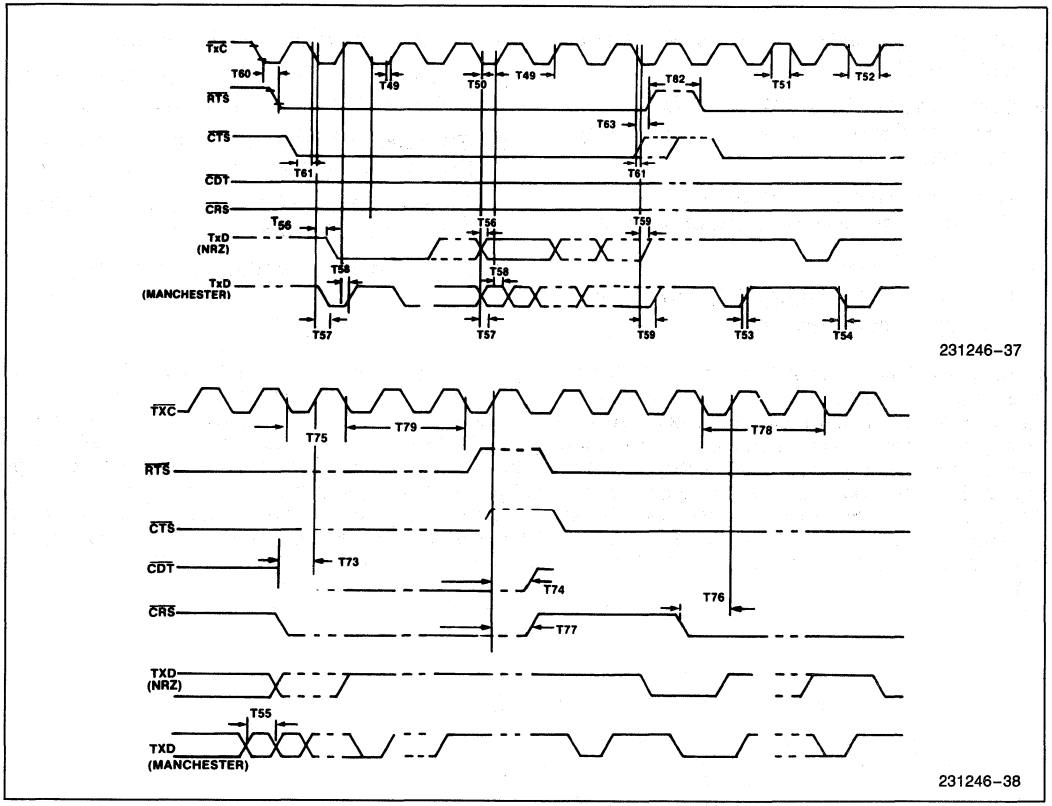


Figure 36. Transmit and Control and Data Timing

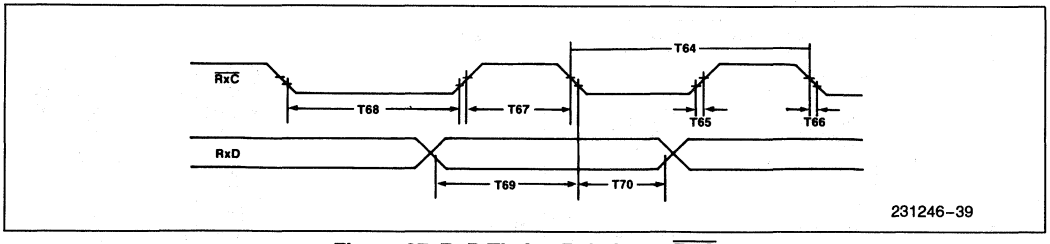


Figure 37. RxD Timing Relative to $\overline{RxC}$

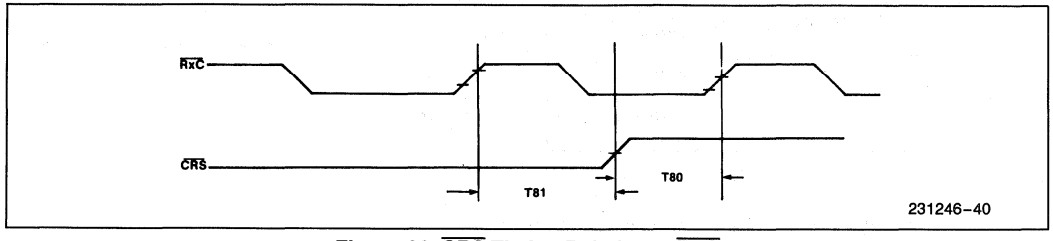
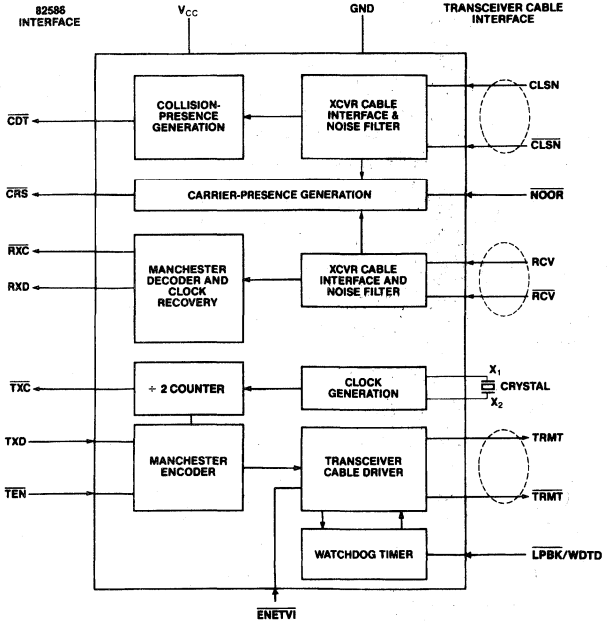


Figure 38. $\overline{CRS}$ Timing Relative to $\overline{RxC}$

82C501 ETHERNET SERIAL INTERFACE

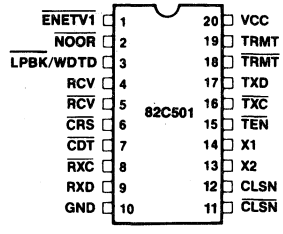
- CHMOS Replacement for Intel 82501 or SEEQ 8023A
- Conforms to IEEE 802.3 10BASE5 (Ethernet) and 10BASE2 (Cheapernet) Specifications
- Direct Interface to the 82586 LAN Coprocessor and 82502 Transceiver
- 10 Mbps Operation
- Manchester Encoding/Decoding and Receive Clock Recovery
- 10 MHz Transmit Clock Generator
- Drives/Receives 802.3 10BASE5 Access Unit Interface (Transceiver) Cable
- Defeatable Watchdog Timer Circuit to Prevent Continous Transmissions
- Diagnostic Loopback for Network Node Fault Detection and Isolation
- Replaces 8 to 12 MSI Components

The 82C501 Ethernet Serial Interface (ESI) chip is designed to work directly with the 82586 LAN Coprocessor in IEEE 802.3 (10BASE5 and 10BASE2), 10 Mbps, Local Area Network applications. The major functions of the 82C501 are to generate the 10 MHz transmit clock for the 82586, perform Manchester encoding/decoding of the transmitted/received frames, and provide the electrical interface to the Ethernet transceiver cable (AUI). Diagnostic loopback control enables the 82C501 to route the signal to be transmitted from the 82586 through its Manchester encoding and decoding circuitry and back to the 82586. The combined loopback capabilities of the 82586 and 82C501 result in highly effective fault detection and isolation through sequential testing of the communications interface. A (defeatable) on-chip watchdog timer circuit prevents the station from locking up in a continuous transmit mode. The 82C501 is socket compatible with the predecessor 82501 and SEEQ 8023A.



231926-1

Figure 1. 82C501 Functional Block Diagram



231926-2

Figure 2. Pin Configuration

September 1987
Order Number: 231926-003

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
$\overline{\text{ENETVI}}$	1	I	**ETHERNET VERSION 1.0:** An active low, MOS-level input. When $\overline{\text{ENETV1}}$ is asserted, the TRMT/$\overline{\text{TRMT}}$ pair remains at high differential voltage at the end of transmission. This operation is compatible with the Ethernet Version 1.0 specification. If the $\overline{\text{ENETV1}}$ pin is left floating, an internal pull-up resistor biases the input inactive high.
$\overline{\text{NOOR}}$	2	I	**CRS 'OR':** An active low, MOS-level input. When $\overline{\text{NOOR}}$ is asserted a valid signal on the collision-presence pair (CLSN/$\overline{\text{CLSN}}$) will not force $\overline{\text{CRS}}$ active low. With $\overline{\text{NOOR}}$ active $\overline{\text{CRS}}$ can only be asserted by the presence of valid data bits on the RCV/$\overline{\text{RCV}}$ pair. If the $\overline{\text{NOOR}}$ pin is floating, an internal pull-up resistor biases the input inactive high.
$\overline{\text{LPBK}}$/ WDTD	3	I	**LOOPBACK/WATCHDOG TIMER DISABLE:** An active low, TTL-level control signal enables the loopback mode. In loopback mode serial data on the TXD input is routed through the 82C501 internal circuits and back to the RXD output without driving the TRMT/$\overline{\text{TRMT}}$ output pair to the transceiver cable. During loopback $\overline{\text{CDT}}$ is asserted at the end of each transmission to simulate the SQE test. **WATCHDOG TIMER DISABLE:** An input voltage of 10 to 16 through a 1 KΩ resistor will disable the on-chip watchdog timer.
RCV $\overline{\text{RCV}}$	4 5	I I	**RECEIVE PAIR:** A differentially driven input pair which is tied to the receive pair of the Ethernet transceiver cable. The first transition on RCV will be negative-going to indicate the beginning of a frame. The last transition should be positive-going to indicate the end of the frame. The received bit stream is assumed to be Manchester encoded.
$\overline{\text{CRS}}$	6	O	**CARRIER SENSE:** An active low, MOS-level output to notify the 82586 that there is activity on the coaxial cable. The signal is asserted when valid data or a collision-presence signal from the transceiver is present. It is deasserted at the end of a frame; or when the end of the collision-presence signal is detected, synchronous with $\overline{\text{RXC}}$. After transmission, once deasserted, $\overline{\text{CRS}}$ will not be reasserted again for a period of 5 μs minimum or 7 μs maximum, regardless of any activity on the collision-presence signal (CLSN/$\overline{\text{CLSN}}$) and RCV/$\overline{\text{RCV}}$ inputs.
$\overline{\text{CDT}}$	7	O	**COLLISION DETECT:** An active-low, MOS-level signal which drives the $\overline{\text{CDT}}$ input of the 82586 controller. It is asserted as long as there is activity on the collision pair (CLSN/$\overline{\text{CLSN}}$), and during SQE (heartbeat) test in loopback.
$\overline{\text{RXC}}$	8	O	**RECEIVE CLOCK:** A 10 MHz MOS level clock output with 5 ns rise and fall times. This output is connected to the 82586 receive clock input $\overline{\text{RXC}}$. There is a maximum 1.4 μs delay at the beginning of a frame reception before the clock recovery circuit gains lock. During idle (no incoming frames) $\overline{\text{RXC}}$ is forced low.
RXD	9	O	**RECEIVE DATA:** A MOS-level output tied directly to the RXD input of the 82586 controller and sampled by the 82586 at the negative edge of $\overline{\text{RXC}}$. The bit stream received from the transceiver cable is Manchester decoded prior to being transferred to the controller. This output remains high during idle.
GND	10		**GROUND:** Reference.
CLSN $\overline{\text{CLSN}}$	12 11	I I	**COLLISION PAIR:** A differentially driven input pair tied to the collision-presence pair of the Ethernet transceiver cable. The collision-presence signal is a 10 MHz square wave. The first transition at CLSN is negative-going to indicate the beginning of the signal; the last transition is positive-going to indicate the end of the signal.
X_1 X_2	14 13	I I	**CLOCK CRYSTAL:** 20 MHz crystal inputs. When X_2 is floated, X_1 can be used as an external MOS level input clock.

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{TEN}$	15	I	**TRANSMIT ENABLE:** An active low, TTL level signal synchronous to $\overline{TXC}$ that enables data transmission to the transceiver cable and starts the watchdog timer. $\overline{TEN}$ can be driven by the $\overline{RTS}$ from the 82586.
$\overline{TXC}$	16	O	**TRANSMIT CLOCK:** A 10 MHz MOS level clock output with 5 ns rise and fall times. This clock is connected directly to the $\overline{TXC}$ input of the 82586.
TXD	17	I	**TRANSMIT DATA:** A TTL-level input signal that is directly connected to the serial data output, TXD, of the 82586.
TRMT $\overline{TRMT}$	19 18	O O	**TRANSMIT PAIR:** A differential output driver pair that drives the transmit pair of the transceiver cable. The output bit stream is Manchester encoded. Following the last transmission, which is always positive at TRMT, the differential voltage is slowly reduced to zero volts in a series of steps. If $\overline{ENETV1}$ is asserted this voltage stepping is disabled.
V_{CC}	20		**POWER:** 5V ±10%.

FUNCTIONAL DESCRIPTION

Clock Generation

A 20 MHz parallel resonant crystal is used to control the clock generation oscillator which provides the basic 20 MHz clock source. An internal divide-by-two counter generates the 10 MHz ± 0.01% clock required by the IEEE 802.3 specification.

It is recommended that a crystal meeting the following specifications be used:

- Quartz Crystal
- 20.00 MHz ± 0.002% @ 25°C
- Accuracy ±0.005% Over Full Operating Temperature, 0 to 70°C
- Parallel resonant with 20 pF Load Fundamental Mode

Several vendors have these crystals available; either off the shelf or custom made. Two possible vendors are:

1. M-Tron Industries, Inc
   Yankton, SD 57078

2. Crystek Corporation
   100 Crystal Drive
   Ft Myers, FL 33907

For best operation the total crystal load capacitance should be about 20 pF. Considering the internal capacitance of the 82C501, and board capacitance, a typical configuration would have a 30 to 35 pF capacitor connected between the X_1 and X_2 input and ground. Stray capacitance of the board will bring the total capacitance to 20 pF. The total length of the line on each side of the crystal (between X_1 and X_2, the crystal, and the capacitor) should be less than one inch.

An external, 20 MHz, MOS-level clock may be applied to pin X_1 while pin X_2 is left floating.

TRANSMIT SECTION

Manchester Encoder and Transceiver Cable Driver

The 20 MHz clock is used to Manchester encode data on the TXD input line. The clock is also divided by two to produce the 10-MHz clock required by the 82586 for synchronizing its $\overline{RTS}$ and TXD signals. See Figure 3. (Note that the 82586 $\overline{RTS}$ is tied to the 82C501 $\overline{TEN}$ input as shown in Figure 4.)

Data encoding and transmission begins with $\overline{TEN}$ going low. Since the first bit is a '1', the first transition on the transmit output TRMT is always negative. Transmission ends with the $\overline{TEN}$ going high. The last transition is always poisitve at TRMT and may occur at the center of the bit cell (last bit = 1) or at the boundary of the bit cell (last bit = 0). A 1-bit delay is introduced by the 82C501 between its TXD input and TRMT/$\overline{TRMT}$ output as shown in Figure 3. If the signal applied to the $\overline{ENETV1}$ input is inactive high the TRMT output is slowly brought to its high state 200 ns after the last transmit data transition. The TRMT/$\overline{TRMT}$ differential voltage will become less than 40 mV within 8 μs after the last data transition. The undershoot for return to idle is less than 100 mV differentially. This mode of operation is compatible with the IEEE 802.3 tranceiver. See Figure 4.

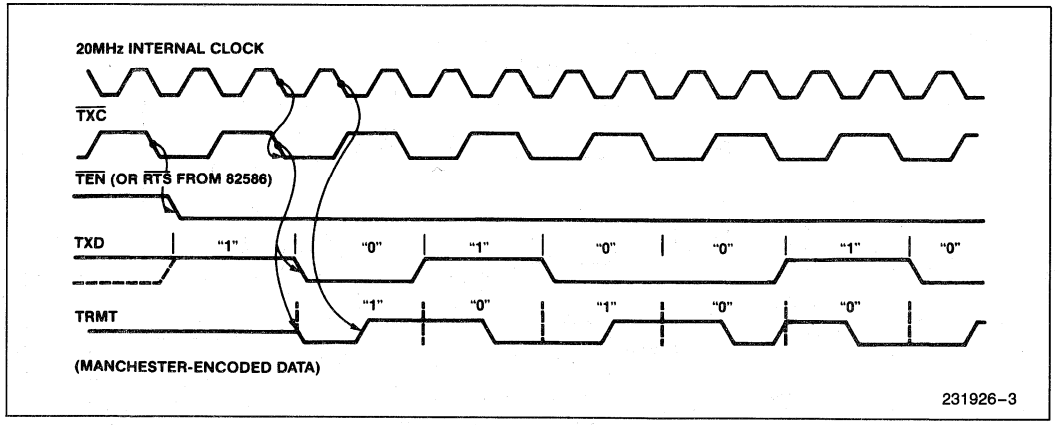


Figure 3. Start of Transmission and Manchester Encoding

If an active signal is present at the $\overline{\text{ENETV1}}$ input at the end of transmission, the TRMT/$\overline{\text{TRMT}}$ pair output will remain a high differential voltage. As a result there will be a positive differential voltage during the entire transmit idle time. This mode of operation is compatible with the Ethernet Version 1.0 specification.

Immediately after the end of a transmission all signals on the receive pair are inhibited for 5 μs minimum to 7 μs maximum. This dead time is required for proper operation of the SQE (heartbeat) test.

An internal watchdog timer is started when $\overline{\text{TEN}}$ is asserted low at the beginning of the frame. The duration of the watchdog timer is 25 ms ± 15%. If the transmission terminates (by deasserting the $\overline{\text{TEN}}$) before the timer expires, the timer is reset (and ready for the next transmission). If the timer expires before the transmission ends the frame is aborted. The frame is aborted by disabling the output driver for the TRMT/$\overline{\text{TRMT}}$ pair. RXD and $\overline{\text{RXC}}$ are not affected. The watchdog timer is reset only when the $\overline{\text{TEN}}$ is deasserted.

The cable driver is a differential circuit requiring external pulldown resistors of 240Ω ±5%. In addition, high-voltage protection of +10V maximum, and short circuit protection to ground is provided.

To provide additional high voltage protection, if the cable is shorted, an isolation transformer can be used to isolate the TRMT and $\overline{\text{TRMT}}$ outputs. Transmit circuit inductance (including the IEEE 802.3 transceiver transformers) should be twenty-seven microhenrys minimum.

RECEIVE SECTION

Cable Interface and Noise Filter

The 82C501 input circuits can be driven directly from the Ethernet transceiver cable receive pair. In this case the cable is terminated with a resistor of 78Ω ± 6% for proper impedance matching. See Figure 4.

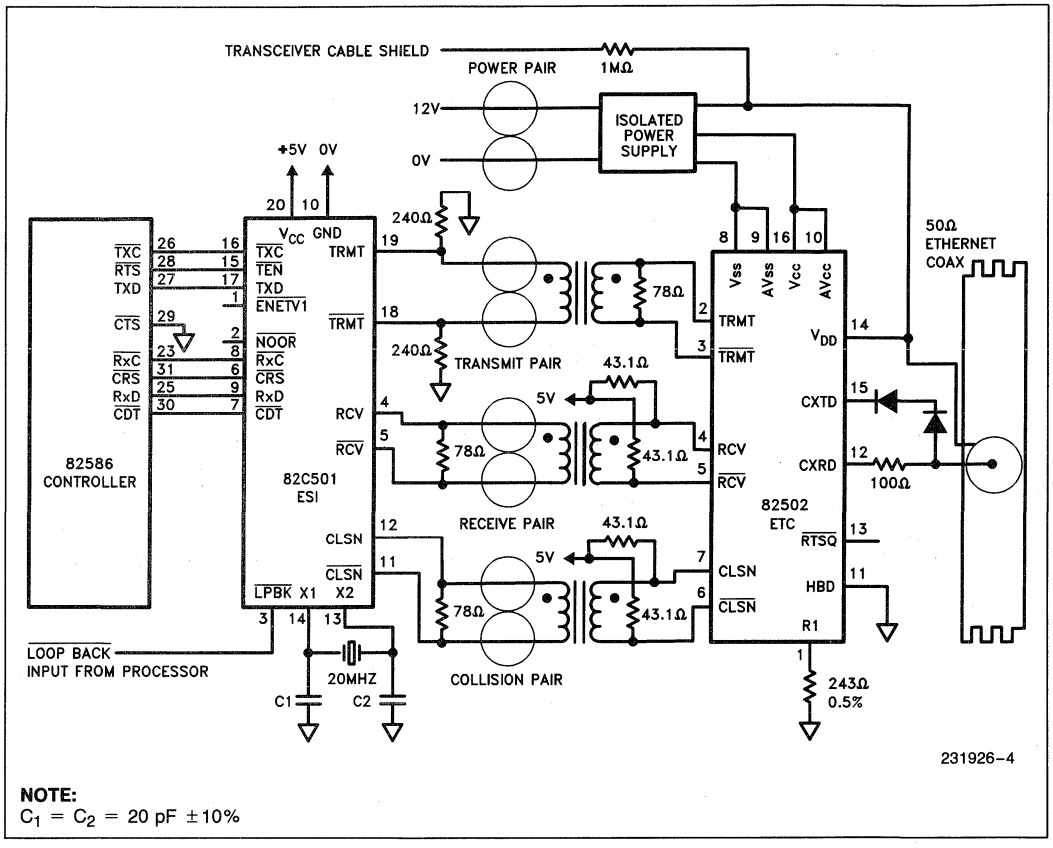
The signal received on the RCV/$\overline{\text{RCV}}$ pair from the transceiver defines both the $\overline{\text{RXC}}$ and RXD outputs to the 82586. The $\overline{\text{RXC}}$ and RXD signals are recovered from the encoded RCV/$\overline{\text{RCV}}$ pair signal by Manchester decode circuitry.

The input circuits can also be driven with ECL voltage levels. In either case, the input common mode voltage must be in the range of 0-V_{CC} volts to allow for wide driver supply variation at the transceiver. The input terminals have a 10V minimum overvoltage protection. To provide additional high voltage protection, if the cable is shorted, an isolation transformer can be used to isolate the RCV and $\overline{\text{RCV}}$ inputs.

A noise filter is provided at the RCV/$\overline{\text{RCV}}$ input pair to prevent spurious signals from improperly triggering the receiver circuitry. The noise filter has the following characteristics.

A negative pulse which is narrower than 10 ns, or is less than 150 mV in amplitude, is rejected during idle.

At the beginning of a reception the filter is turned off by the first negative pulse whose amplitude is greater than 275 mV and is wider than 30 ns.



NOTE:
$C_1 = C_2 = 20$ pF ±10%

Figure 4. 82586/82C501/Transceiver Cable/82502 Interface

As soon as the first valid negative pulse is recognized by the noise filter the data threshold is lowered and the $\overline{CRS}$ signal is asserted to inform the 82586 controller of the beginning of a reception. The $\overline{RXC}$ will be held low for a maximum of 1.4 μs while the synchronizer acquires a lock on the RCV signal.

If no negative transition occurs within 160 ns from the last positive transition end of frame is assumed.

After the end of a frame $\overline{CRS}$ is not inhibited. If there is a negatvie pulse whose amplitude is greater than 275 mV and is wider than 30 ns, or an equivalent sized undershoot from the transceiver, then $\overline{CRS}$ will be reactivated.

Manchester Decoder and Clock Recovery

The filtered data enters the clock recovery and decoder circuits. A Digital Bit Synchronizer (DBS) is used to extract the received clock from the data; beginning from the third negative transition of the incoming data. The DBS will acquire lock within the first fourteen bit times, as seen from the RCV/$\overline{RCV}$ inputs. During that period of time the $\overline{RXC}$ is held low. The maximum, bit-cell timing distortion *(jitter)* tolerated by the DBS is ± 12 ns for the preamble, and ± 18 ns for data.

The RCV/$\overline{RCV}$ input is decoded from Manchester to NRZ and transferred synchronously with $\overline{RXC}$ to the 82586.

COLLISION-PRESENCE SECTION

The CLSN/$\overline{CLSN}$ input signal is a 10 MHz ± 15% square-wave generated by the transceiver whenever two or more data frames are superimposed on the coaxial cable. The maximum asymmetry in the CLSN/$\overline{CLSN}$ signal is 60/40% for low-to-high or

high-to-low levels. The signal is filtered for noise rejection in the same manner as RCV/$\overline{\text{RCV}}$. The noise filter rejects signals with an amplitude less than 150 mV and narrower than 10 ns. The filter turns off at the first negative pulse with an amplitude greater than 275 mV and wider than 30 ns. The filter will remain off indicating that a valid collision-presence signal is present as long as the negative CLSN/$\overline{\text{CLSN}}$ signal pulses have an amplitude greater than 275 mV. The filter returns to the 'on' state if the signal's amplitude becomes less than 150 mV, or if no negative transition occurs within 160 ns after the last positive transition. The collision filter can be immediately turned off again at the first valid signal pulse.

The common-mode voltage and external termination are identical to the RCV/$\overline{\text{RCV}}$ input. (See Figure 4.) Like the RCV/$\overline{\text{RCV}}$ input, the CLSN/$\overline{\text{CLSN}}$ input also has a 10V maximum protection and additional clamping against low-energy, high-voltage noise signals. To provide additional high voltage protection, if the cable is shorted, an isolation transformer can be used to isolate the CLSN and $\overline{\text{CLSN}}$ inputs.

A valid collision-presence signal will assert the 82C501 $\overline{\text{CDT}}$ output which can be directly tied to the $\overline{\text{CDT}}$ input of the 82586 controller. During normal operation the 82C501 logically 'OR's the collision-presence signal with internal valid data on (RCV/$\overline{\text{RCV}}$ pair) signal to generate $\overline{\text{CRS}}$ output. If, however, the $\overline{\text{NOOR}}$ input is asserted low, this 'OR' function is removed and $\overline{\text{CRS}}$ is only asserted by the presence of valid data on the RCV/$\overline{\text{RCV}}$ pair. This mode of operation is required for repeater design.

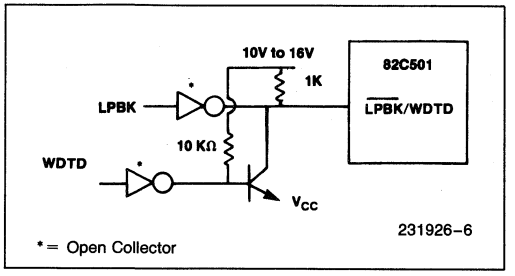
During the time that valid collision-presence transitions are present on the CLSN/$\overline{\text{CLSN}}$ input invalid data transitions may be present on the receive data pair due to the superposition of signals from two or more stations transmitting simultaneously. It is possible for RCV/$\overline{\text{RCV}}$ to lose transitions for a few bit times due to perfect cancellation of the signals; this may cause the 82C501 to abort the reception.

When a valid collision-presence signal is present the $\overline{\text{CRS}}$ signal is asserted (along with $\overline{\text{CDT}}$). However, if this collision-presence signal arrives within 5 to 7 μs from the last transmission only $\overline{\text{CDT}}$ is generated so that the 82586 recognizes the active $\overline{\text{CDT}}$ as a valid SQE (heartbeat) test signal.

Internal Loopback

When asserted, $\overline{\text{LPBK}}$ causes the 82C501 to route serial data from its TXD input through its transmit logic (retiming and Manchester encoding); returning it through the receive logic (Manchester decoding and receive clock generation) to RXD output. The internal routing prevents the data from passing through the output drivers and onto the transmit output pair TRMT/$\overline{\text{TRMT}}$. When in loopback mode all of the transmit and receive circuits, including the noise filter, are tested except for the transceiver cable output driver and input receivers. Also, at the end of each frame transmitted in loopback mode the 82C501 generates the SQE test (heartbeat) signal within 1 μs after the end of the frame. Thus, the collsion circuits, including the noise filter, are also tested in loopback mode.

The watchdog timer remains enabled in loopback mode, terminating test frames that exceed its time-out period. The watchdog can be inhibited by connecting $\overline{\text{LPBK}}$ to a 1 kΩ resistor connected to 10V to 16V. The loopback feature can still be used to test the integrity of the 82C501 by using the circuit shown in Figure 5.



LPBK	WDTD	Function
1	X	LPBK mode
0	0	Normal mode
0	*1	Normal mode with watchdog timer disabled

Figure 5. Watchdog Timer Disable

The 82C501 operates as a full-duplex device, being able to transmit and receive simultaneously. By combining the internal and external loopback modes of the 82586, and the internal loopback and normal modes of the 82C501, incremental testing of an 82586/82C501-based interface can be performed under program control for systematic fault detection and fault isolation.

Interface Example

The 82C501 is designed to work directly with the 82586 controller in IEEE 802.3 10 Mbps, as well as other 10 Mbps LAN applications. The control and data signals connect directly between the two devices without the need for additional external logic. The complete 82586/82C501 Ethernet Transceiver cable/82502 interface is shown in Figure 4. The 82C501 provides the driver and receivers needed to directly connect to the transceiver cable; requiring only terminating resistors on each input signal pair and 240Ω pull-down resistors.

If additional high voltage protection is desired, a pulse transformer may be included for Ethernet applications. IEEE 802.3 10BASE5 (Ethernet) specifications require at least 16V protection for the Transmit, Receive, and Collision pairs. In 10BASE2 (Cheapernet) a pulse transformer is required to be inserted between the DTE (82586/82C501) and the transceiver (82502). Through the use of jumpers, the same transformer can be used for an Ethernet connection at minimal additional cost.

The pulse transformer should have the following characteristics:

1. Minimum inductance of 50 microhenries; 70 preferred.
2. 2000V isolation between primary and secondary windings.
3. 2000V isolation between primaries of separate transformers.

Since Ethernet Version 1.0 transceivers may require a positive differential on the TRMT pair during idle, check with the transceiver vendor before including the pulse transformer.

ABSOLUTE MAXIMUM RATING

Ambient Temperature Under Bias ....0°C to +70°C
Case Temperature Under Bias .....0°C to +90°C
Storage Temperature .....−65°C to +140°C
All Output and Supply Voltages .....−0.5V to +7V
All Input Voltages .....−1.0V to +6.0V(1)
Power Dissipation.....0.5W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS T_A = 0°C to +70°C, T_C = 0°C to +90°C, V_{CC} = 5V ± 10%

Symbol	Parameter		Min	Max
V_{IL}	Input Low Voltage	TTL MOS	−0.5V −0.5V	0.8V 0.6V
V_{IH}	Input High Voltage	TTL MOS	2.0V 3.9V	V_{CC} + 0.5V V_{CC} + 0.5V
V_{IDF}	Input Differential Voltage		±300 mV	±1500 mV
V_{CM}	Input Common Mode Voltage		0V	V_{CC}
V_{OCM}	Common Mode Output(2)		0.5V	5.0V
V_{OL}	Output Low Voltage @ I_{OL} = 4 mA			0.45V
V_{OH}	Output High Voltage (MOS) @ I_{OH} = −500 μA		3.9V	
V_{ODF}	Differential Output Swing(3)		±0.55V	±1.2V
I_{LI}	Input Leakage Current(4) @ V_{IN} = 0V to V_{CC}			±50 μA
C_{IN}	Input Capacitance @ fc = 1 MHz			10 pF
C_{OUT}	Output Capacitance @ fc = 1 MHz			20 pF
I_{CC}	Power Supply Current @ T_A = 70°C(5)			100 mA
V_{REJECT}	Differential Reject Voltage			150 mV
V_{ACCEPT}	Differential Accept Voltage		275 mV	

NOTES:
1. The voltage levels for CLSN/$\overline{\text{CLSN}}$, RCV/$\overline{\text{RCV}}$ inputs are −1.5V to +10V.
2. The load is a 78Ω ±6% resistor in parallel with a 27 μH ±1% inductor.
3. DC measurement values.
4. Only TXD, $\overline{\text{TEN}}$, and X_1 conform to this spec. $\overline{\text{NOOR}}$, $\overline{\text{ENETV1}}$, and $\overline{\text{LPBK}}$/WDTD inputs source/sink a maximum of 1 mA.
5. Part of the power is dissipated through the pulldown resistors connected to TRMT/$\overline{\text{TRMT}}$ outputs.

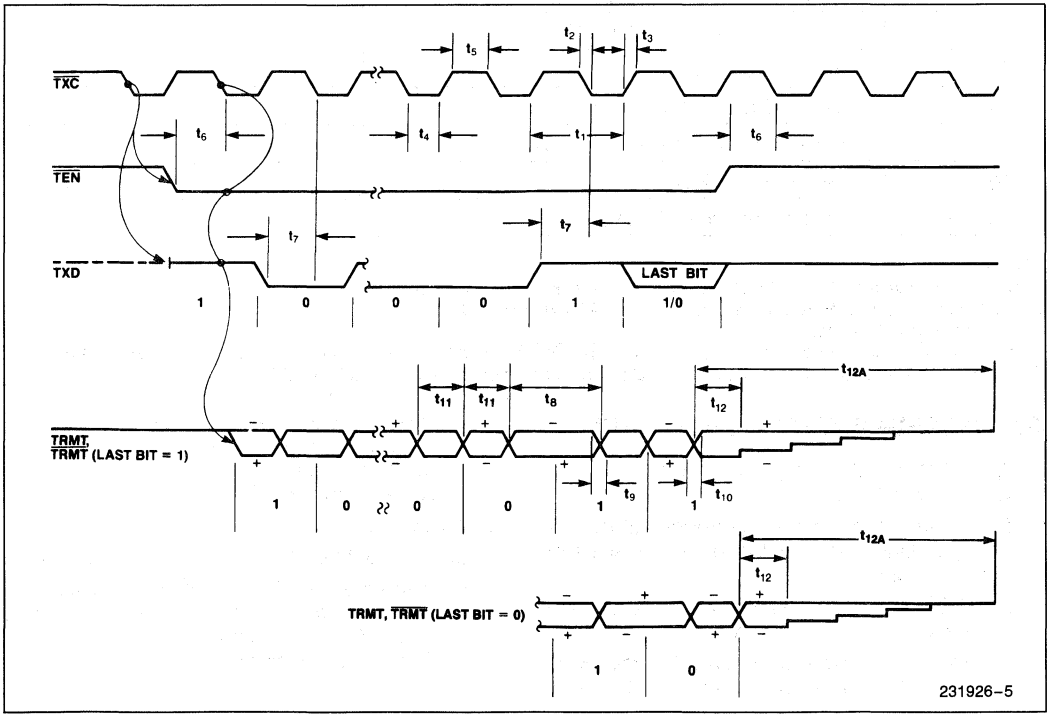
A.C. MEASUREMENT CONDITIONS

1. T_A = 0°C to +70°C, T_C = 0°C to +90°C, V_{CC} = 5V ±10%.
2. The AC measurements are performed at the following voltage levels for these various kinds of inputs and outputs:
   a) MOS and TTL, as defined in the DC characteristics.
   b) Differential Inputs and Outputs: the 50% points of the total swing are used for delay measurement. The rise and fall times are measured at the 20% and 80% points, except when the IEEE 802.3 CSMA/CD spec indicates otherwise.
3. AC Loads:
   a) MOS: a 20 pF total capacitance to ground.
   b) Differential: a 10 pF total capacitance from each terminal to ground, and a load resistor of 78Ω ±6% in parallel with a 27 μH ±1% inductor between terminals.

TRANSMIT TIMING(3)

Symbol	Parameter	Min	Max	Unit
t_1	$\overline{TXC}$ Cycle Time	99.99	100.01	ns
t_2	$\overline{TXC}$ Fall Time(1)		5	ns
t_3	$\overline{TXC}$ Rise Time(1)		5	ns
t_4	$\overline{TXC}$ Low Time (at 0.9V)	40		ns
t_5	$\overline{TXC}$ High Time (at 3.0V)	40		ns
t_6	Transmit Enable/Disable to $\overline{TXC}$ Low	45		ns
t_7	TXD Stable to $\overline{TXC}$ Low	45		ns
t_8	Bit Cell Center to Bit Cell Center of Transmit Pair Data	99.5	100.5	ns
t_9	Transmit Pair Data Fall Time(2)	1.0	5.0	ns
t_{10}	Transmit Pair Data Rise Time(2)	1.0	5.0	ns
t_{11}	Bit Cell Center to Bit Cell Boundary of Transmit Pair Data	49.5	50.5	ns
t_{12}	$\overline{TRMT}$ held low from Last Positive Transition of Transmit Pair Data during idle:	200		ns
t_{12A}	From Last Positive Transition of Transmit Pair Differential Output Approaches Within 40 mV of zero volts.		8000	ns

NOTES:
1. Rise and fall times are guaranteed by design.
2. Measured per 802.3 Para. 7.4.1.1.
3. t_{22}, t_{23}, t_{24}, t_{39} do not appear.

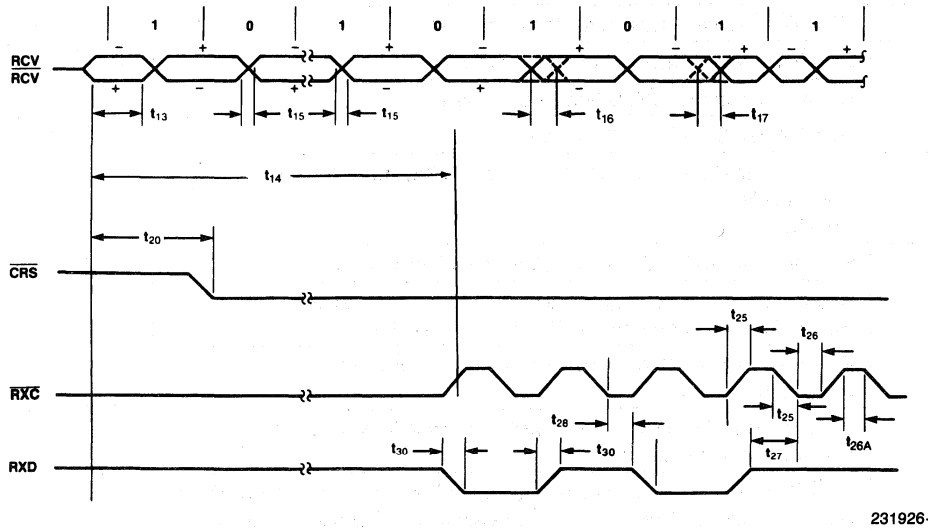


RECEIVE TIMING

Symbol	Parameter	Min	Max	Unit
t_{13a}	RCV/$\overline{RCV}$ Noise Filter-Pulse Width Rejected		10	ns
t_{13b}	RCV/$\overline{RCV}$ Noise Filter-Pulse Width Accepted	30		ns
t_{14}	Duration which the $\overline{RXC}$ is held at Low State		1400	ns
t_{15}	Receive Pair Signal Rise/Fall Time at ± 0.2V		10	ns
t_{16}	Receive Pair Bit Cell Center from Crossover Timing Distortion: In Preamble In Data		 ±12 ±18	 ns ns
t_{17}	Receive Pair Bit Cell Boundary Allowing for Timing Distortion In Data		±18	ns
t_{18}	Receive Idle Time Before the Next Reception can Begin in a Transmitting Station (as Measured from the Deassertion of $\overline{CRS}$)		8	µs
t_{19}	Receive Pair Signal Return to Zero Level from Last Valid Positive Transition	160		ns
t_{20}	$\overline{CRS}$ Assertion Delay from the First Received Valid Negative Transition of Receive Pair Signal		100	ns
t_{21}	$\overline{CRS}$ Deassertion Delay from the Last Valid Positive Transition Received (when no Collision-Presence Signal Exists on the Transceiver Cable)(1)		300	ns
t_{25}	$\overline{RXC}$ Rise/Fall Time(2)		5.0	ns
t_{26}	$\overline{RXC}$ Low Time (at 0.9V)	40		ns
t_{26A}	$\overline{RXC}$ High Time (at 3.0V)	36		ns
t_{27}	Receive Data Stable Before the Negative Edge of $\overline{RXC}$	30		ns
t_{28}	Receive Data Held Valid Past the Negative Edge of $\overline{RXC}$	30		ns
t_{29}	Carrier Sense Active → Inactive Hold Time from $\overline{RXC}$ High	10	40	ns
t_{30}	Receive Data Rise/Fall Time		10	ns
t_{31}	$\overline{CRS}$ Inhibit Time After Frame Transmission(3)	5	7	µs

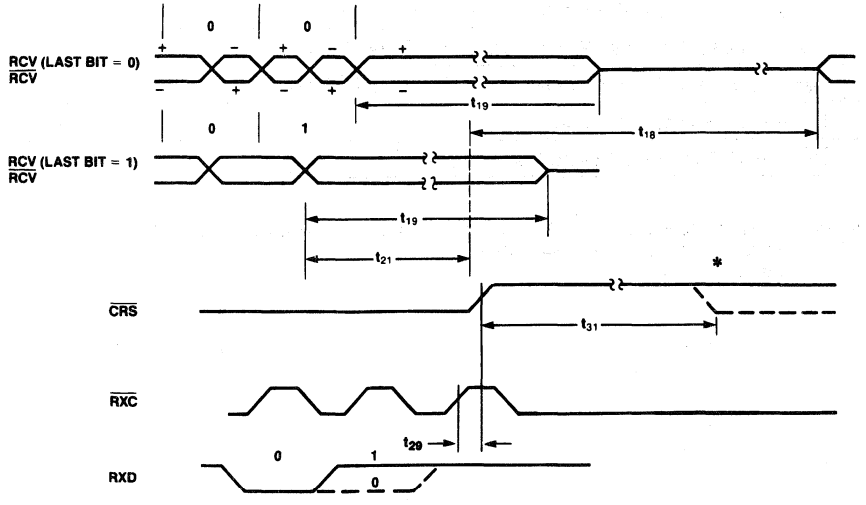
NOTES:
1. $\overline{CRS}$ is deasserted synchronously with the $\overline{RXC}$. This condition is not specified in the IEEE 802.3 specification.
2. Not tested, rise and fall times are guaranteed by design.
3. Required for SQE test.

RECEIVE TIMING: START OF FRAME



231926-7

RECEIVE TIMING: END OF FRAME

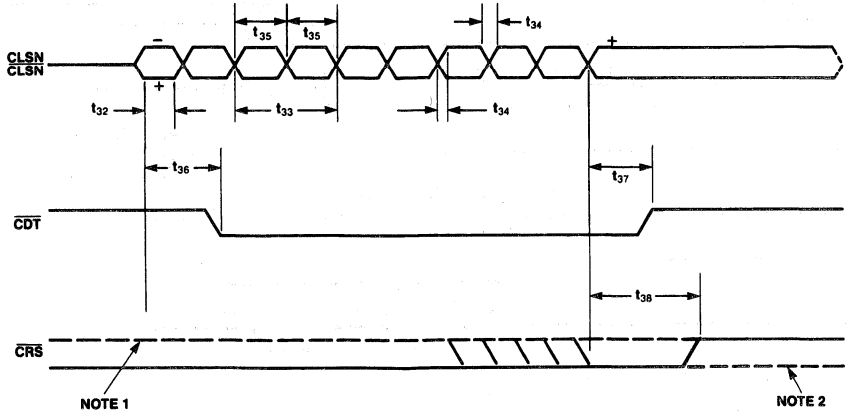


231926-8

***NOTE:**
CRS can be triggered on again by the collision-presence signal.

COLLISION TIMING

Symbol	Parameter	Min	Max	Unit
t_{32a}	CLSN/$\overline{\text{CLSN}}$ Noise Filter-Pulse Width Rejected		10	ns
t_{32b}	CLSN/$\overline{\text{CLSN}}$ Noise Filter-Pulse Width Accepted	30		ns
t_{33}	CLSN/$\overline{\text{CLSN}}$ Cycle Time	85	118	ns
t_{34}	CLSN/$\overline{\text{CLSN}}$ Rise/Fall Time at ± 0.2V		15	ns
t_{35}	CLSN/$\overline{\text{CLSN}}$ Transition Time	35	70	ns
t_{36}	$\overline{\text{CDT}}$ Assertion from the First Valid Negative Edge of Collision Pair Signal		75	ns
t_{37}	$\overline{\text{CDT}}$ Deassertion from the Last Positive Edge of CLSN/$\overline{\text{CLSN}}$ Signal		300	ns
t_{38}	$\overline{\text{CRS}}$ Deassertion from the Last Positive Edge of CLSN/$\overline{\text{CLSN}}$ Signal (if no Post-Collision Signal Remains on the Receive Pair)		450	ns



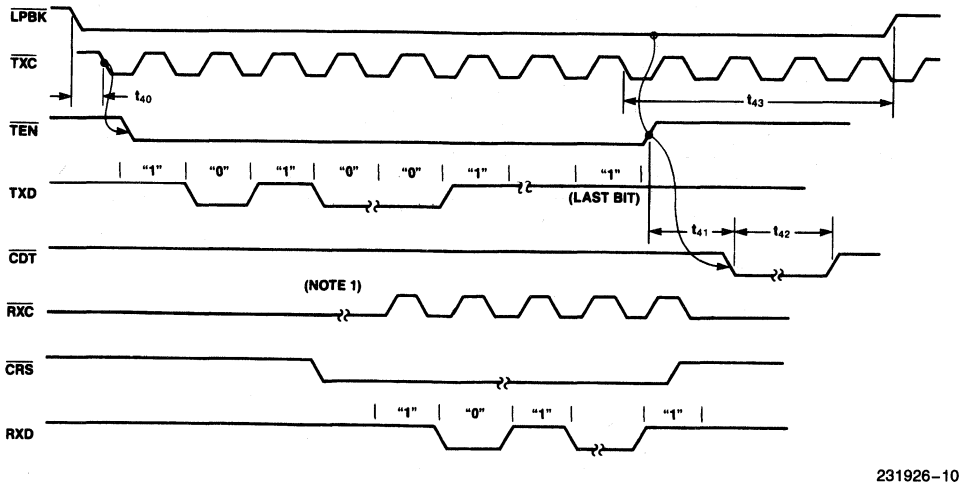
231926-9

NOTES:
1. $\overline{\text{CRS}}$ will be deasserted for a period up to 7 μs maximum when RCV/$\overline{\text{RCV}}$ or CLSN/$\overline{\text{CLSN}}$ terminates, whichever occurs later.
2. $\overline{\text{CRS}}$ will remain asserted after the CLSN/$\overline{\text{CLSN}}$ signal terminates if RCV/$\overline{\text{RCV}}$ signals continue.

LOOPBACK TIMING

Symbol	Parameter	Min	Max	Unit
t_{40}	$\overline{LPBK}$ asserted before the first attempted transmission (1)	500		ns
t_{41}	Simulated collision test delay from the end of each attempted transmission	0.5	1.5	μs
t_{42}	Simulated collision test duration(2)	0.6	1.6	μs
t_{43}	$\overline{LPBK}$ deasserted after the last attempted transmission	5		μs

NOTES:
1. In Loopback mode, $\overline{RXC}$, $\overline{RXC}$ and $\overline{CRS}$ function in the same manner as a normal Receive.
2. SQE test (heartbeat) signal



NOTE:
1. During Loopback, the 82C501 receive circuitry uses 14 bit times while the Digital Bit Synchronizer locks on the data. As a result, the first 14 bits are lost and $\overline{RXC}$ is held low during that time.

TESTABILITY

NOTE:
1. All AC Parameters become valid after the Digital Bit Synchronizer has stabilized: 100 μs after the application of power.

82502
ETHERNET TRANSCEIVER CHIP

- **Conforms to Following Standards:**
  - **— IEEE 802.3, 10BASE5 (Ethernet)**
  - **— IEEE 802.3, 10BASE2 (Cheapernet)**
  - **— Ethernet Version 2.0**
- **Jabber Function**
- **Receive Based Collision Detection**
- **Defeatable Signal Quality Error (Heartbeat) Test**
- **Requires Minimum Board Space**
  - **— On-Chip Voltage Reference**
  - **— 16 Pin DIP**
- **No External Adjustments Required**
- **Reliable CHMOS Technology**

The 82502 Ethernet Transceiver Chip is a CHMOS VLSI device that provides the complete set of transmit, receive and collision detection functions specified by the IEEE 802.3, 10BASE5 (Ethernet) and 10BASE2 (Cheapernet) 10 Mbps baseband standards for the Media Attachment Unit (MAU). The 82502 teams up with Intel's 82586 LAN Coprocessor and the 82C501 Ethernet Serial Interface enabling the designer to implement highly integrated IEEE 802.3 systems.

Three basic functional blocks make up the 82502: transmit, receive and collision detection. The transmit and receive sections transfer data from the transceiver drop (Attachment Unit Interface or AUI) cable to the coaxial cable of the network and vice-versa. The collision detection section senses simultaneous transmissions by two or more network stations (collisions) on the coaxial cable and reacts by sending a 10 MHz signal across the transceiver drop cable to the station that it front ends.

When used in an Ethernet application, the 82502 can drive a transceiver cable up to 50 meters in length (for Cheapernet, there is no transceiver cable). The 82502 provides all active communications circuitry for the transceiver function in the Ethernet/Cheapernet environment. It is an ideal companion to the Intel 82C501.

The 82502 is fabricated in Intel's reliable 10V CHMOS II process.

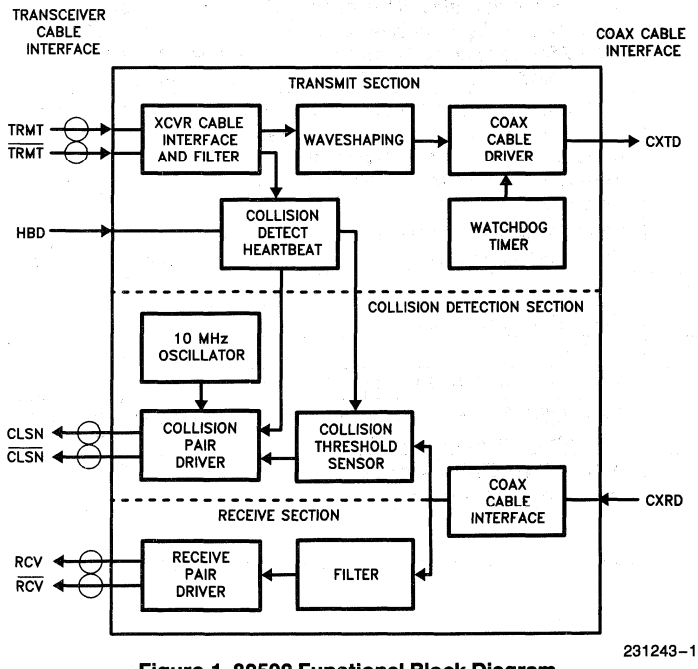


Figure 1. 82502 Functional Block Diagram

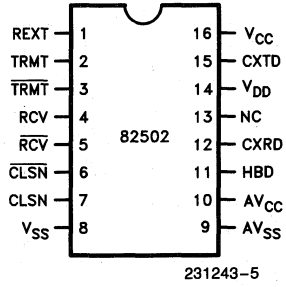


Figure 2. 82502 Pin Diagram

October 1987
Order Number: 231243-005

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
TRMT, $\overline{\text{TRMT}}$	2 3	I I	**TRANSMIT DATA PAIR:** A differentially driven input tied to the transmit pair of the transceiver cable. The transmit pair of the transceiver cable is driven with 10 Mbps Manchester encoded data from the serial interface of the data link (82C501). TRMT/$\overline{\text{TRMT}}$ must be isolated from the transceiver cable by a pulse transformer. The last transition is expected to be positive indicating end of packet.
RCV, $\overline{\text{RCV}}$	4 5	O O	**RECEIVE DATA PAIR:** An output driver pair that generates an ECL AC signal level to drive the transceiver cable receive pair with the 10 Mbps Manchester encoded data received from the coaxial cable of the network. RCV/$\overline{\text{RCV}}$ must be isolated from the transceiver cable by a pulse transformer. The last transition is always positive indicating the end of the packet. The current from the $\overline{\text{RCV}}$ pin is incrementally decreased after the last transition.
CLSN, $\overline{\text{CLSN}}$	7 6	O O	**COLLISION PRESENCE PAIR:** An output driver pair that generates a 10 MHz ECL AC signal level square wave on the collision presence pair of the transceiver cable when: a collision is detected on the coaxial cable of the network, during self-test as the collision circuit heartbeat indication, or after the watchdog timer has expired to indicate that the coaxial cable transmitter is disabled.
CXTD	15	O	**COAXIAL CABLE TRANSMIT DATA:** An output pin that transmits data onto the coaxial cable of the network by sinking current from the center conductor of the coaxial cable. The last data transition at the end of a packet is always low to high.
CXRD	12	I	**COAXIAL CABLE RECEIVE DATA:** An input pin that receives data from the coaxial cable of the network. Typical signal levels (referenced to V_{DD}) received on CXRD are −200 mV for high, −1.8V for a low and 0V during idle. The last data transition received is expected to be positive indicating the end of packet.
HBD	11	I	**HEARTBEAT DISABLE:** A strapping option that when tied low (V_{SS}), allows the transceiver to generate a collision detect heartbeat signal after each packet. A high (V_{CC}) on this pin disables the heartbeat circuitry as well as the 6.4 μs transmit inhibit timer but keeps the collision circuit enabled for use in repeater applications.
REXT	1		**EXTERNAL RESISTOR:** A 243Ω 0.5% resistor is attached between REXT and ground (V_{SS}) to provide precision internal current levels.
NC	13	NC	
V_{CC}*	16		**POWER:** 5 ±10% volts.
V_{SS}*	8		**GROUND**
V_{DD}*	14		**POWER/COAX SHIELD:** 10 ±10% volts.
AV_{CC}*	10		**ANALOG POWER:** 5 ±10% volts. Included to reduce the effects of the current fluctuations in the V_{CC} pin.
AV_{SS}*	9		**ANALOG GROUND:** Included to reduce the effects of current fluctuations in the V_{SS} pin.

***NOTE:**
These voltages are referenced to V_{SS}. The shield of the coaxial cable of the Ethernet channel (V_{DD}) is connected to earth ground.

FUNCTIONAL DESCRIPTION

Transmit Section

Transmit Pair Receiver

The 82502 receives transmit data from an Ethernet Serial Interface (ESI) on the TRMT pins and transmits it onto a 50Ω coax cable (the Ethernet channel). The TRMT pins are connected to the transmit pair of a transceiver cable through a pulse transformer (see Figure 3). The transformer provides the isolation required between the transceiver cable and the coax cable. The DC bias for the TRMT pins is provided internally. The 78 ±5% Ω resistor for the TRMT input termination is external to the device.

A noise filter (squelch) is provided at the TRMT input pair to prevent spurious noise signals received on the pins from being transmitted onto the coaxial cable. The squelch is on during idle and rejects signals not more negative than −160 mV and/or less than 16 ns wide. The squelch is removed if the signal on the TRMT pins becomes more negative than −225 mV for 42 ns or more.

Once the squelch is removed, the transmit signal received on the device's TRMT pins measured differentially may be between ±0.25V minimum and ±1.32V maximum. The squelch will remain off if the received signal is not more positive than −225 mV or if it is more positive for less than 111 ns. The squelch will turn back on if the received signal is more positive than −160 mV for more than 215 ns. When the squelch turns back on a non-retriggerable 6.4 μs transmit inhibit timer is started which disables the transmitter within 250 ns after the last transition for a minimum of 5.5 μs and a maximum of 8 μs.

The transmit data path through the device consists of a receiver, rise and fall time control circuitry, and coaxial cable current driver. The maximum timing distortion introduced by the device's transmit path is ±2 ns per data transition edge (t3) including timing distortion caused by output rise and fall time difference. The maximum steady-state propagation delay of the transmit path is 50 ns.

Coaxial Cable Driver

The CXTD pin transmits data onto the coaxial cable by sinking current from the center conductor of the coaxial cable. The DC component of the output current level is between 37 mA and 45 mA. The AC component is from ±28 mA up to the DC component value. The idle current level is less than 5 μA. Positive current is defined as current into the device. The output resistance is 9 KΩ minimum.

At the start of a frame transmission, no more than 2 bits of the frame information may be received from the TRMT pair and not transmitted onto the coaxial cable. The first bit transmitted by the 82502 may contain phase violations, however all successive bits of the frame will be valid.

Watchdog Timer (Jabber Function)

The 82502 Watchdog Timer will inhibit continuous transmissions onto the coaxial cable of duration greater than 52 ms ±18%. The timer starts timing when the TRMT squelch is turned off to receive valid transmit data on the TRMT pins. If a frame transmission is completed within the 52 ms time slot, the timer is reset 2 μs to 6 μs after the last bit of the frame has been transmitted. Two to six microseconds of reset delay is provided so that a short absence of transmit data will not reset the watchdog timer. If a new transmission is started within the reset delay time, the timer will not be reset.

If the timer times out (i.e., transmit frame duration exceeds 52 ms ±18%) the transmit section is disabled and the active collision signal is transmitted out onto the collision pair until transmit data received on the TRMT pair is discontinued. When the TRMT inputs become idle, the collision signal is turned off and a 420 ms (typical) non-retriggerable timer is started. During this 420 ms, the transmit section is disabled. The transmit section is re-enabled after 420 ms.

Receive Section

Coaxial Cable Receiver

The 82502's coaxial cable receive pins "V_{DD}" and "CXRD" are connected to the coax cable shield and center conductor, respectively.

The shunt capacitance measured between the V_{DD} and CXRD pins is typically 2.0 pF. The shunt resistance is greater than 1 MΩ. The input leakage current on the CXRD pin is between −1 μA and 20 μA. These conditions apply in both the power off and power on states.

A noise filter is provided at the V_{DD} and CXRD inputs to reject noise signals during idle. Signals received on the coax whose DC component (frequency less than 1 MHz) does not drop below −120 mV will be rejected. The noise filter will be disabled if the signal's DC component drops below −350 mV. A maximum of 4 bits will be received from the coaxial cable and may not be transmitted onto the RCV pair. The first bit transmitted may contain phase violations, however all successive bits of the frame will be valid.

The coaxial cable receiver compensation circuit is provided such that the combination of the coaxial cable and the receive circuit do not result in more than ±6 ns of edge jitter (t10) for untapped minimum and maximum length Ethernet cables in a noise-free environment.

The noise filter will remain off if the received signal's AC component is not positive for more than 136 ns, and the DC component remains below −350 mV. While the filter is off, the received data will be transmitted onto the RCV pair with less than 50 ns steady state propagation delay. The timing distortion on the RCV pair output is less than ±2 ns per edge for a 400 mV peak-to-peak 5 MHz sinusoidal input from the coaxial cable.

The noise filter will turn back on if the received signal's AC component is positive for more than 210 ns, or if the DC component is more positive than −120 mV. The RXD driver is disabled within 275 ns after the last received data transition.

Receive Pair Driver

The RCV outputs are connected to the receive pair of a transceiver cable through a pulse transformer.

The receive pair (RXD/$\overline{\text{RXD}}$) transceiver cable driver is a current driver. The output pins require 43.2Ω ±1% pull-up resistors to V_{CC}. The differential output voltage level is between ±580 mV minimum and ±1V maximum, for a combined differential inductive load of 25 μH minimum and either 73Ω or 83Ω ±1% resistive load.

Between 200 ns and 325 ns after the last output data transition on the RCV/$\overline{\text{RCV}}$ pins, the 82502 begins incrementally decreasing the $\overline{\text{RCV}}$ output current, using a series of small steps until it becomes idle. The maximum idle current levels on the RCV and $\overline{\text{RCV}}$ pins are 0.2 mA and 7 mA, respectively.

The purpose of the stepping sequence is to guarantee that the RCV/$\overline{\text{RCV}}$ differential output signal (measured with a 78 ±5% terminating resistor in parallel with a minimum magnetizing inductance of 25 μH) will remain high for a minimum of 200 ns following the last output transition, will not undershoot more than 80 mV and will limit the current stored in the inductor to 4 mA after 80 bit times.

Collision Detect Section

The collision presence outputs are connected to the collision pair of a transceiver through a pulse transformer. The collision presence outputs are used by the device to indicate one of three conditions: (1) there are simultaneous transmission attempts by two or more stations on the coax, (2) self test after a transmission proving that the collision circuitry is functional or (3) a frame transmission time exceeds the 52 ms watchdog timer time limit. The active collision presence signals is a 10 MHz ±15% ECL level square wave transmitted on the collision pair of the transceiver cable. The duty cycle of the square wave is better than 40/60 or 60/40. The collision pair driver has the same output signal characteristics as the receive pair driver.

The collision threshold DC level set inside the 82502 corresponds to −1.495 ±5% volts on the coaxial cable. When the collision filter's impulse response is taken into account, the threshold corresponds to a collision threshold between −1.492V and −1.58V. If the DC average signal level detected on the coaxial cable is more negative than the threshold, the on-chip oscillator transmits an active collision presence signal onto the collision pair. The delay in the device from the moment it detects a collision on the CXRD pin to the collision presence signal transmission onto the collision pair is less than 0.9 μs. With a 500 meter coaxial cable, it is less than 2.9 μs including the propagation delay of the coaxial cable.

Self Test

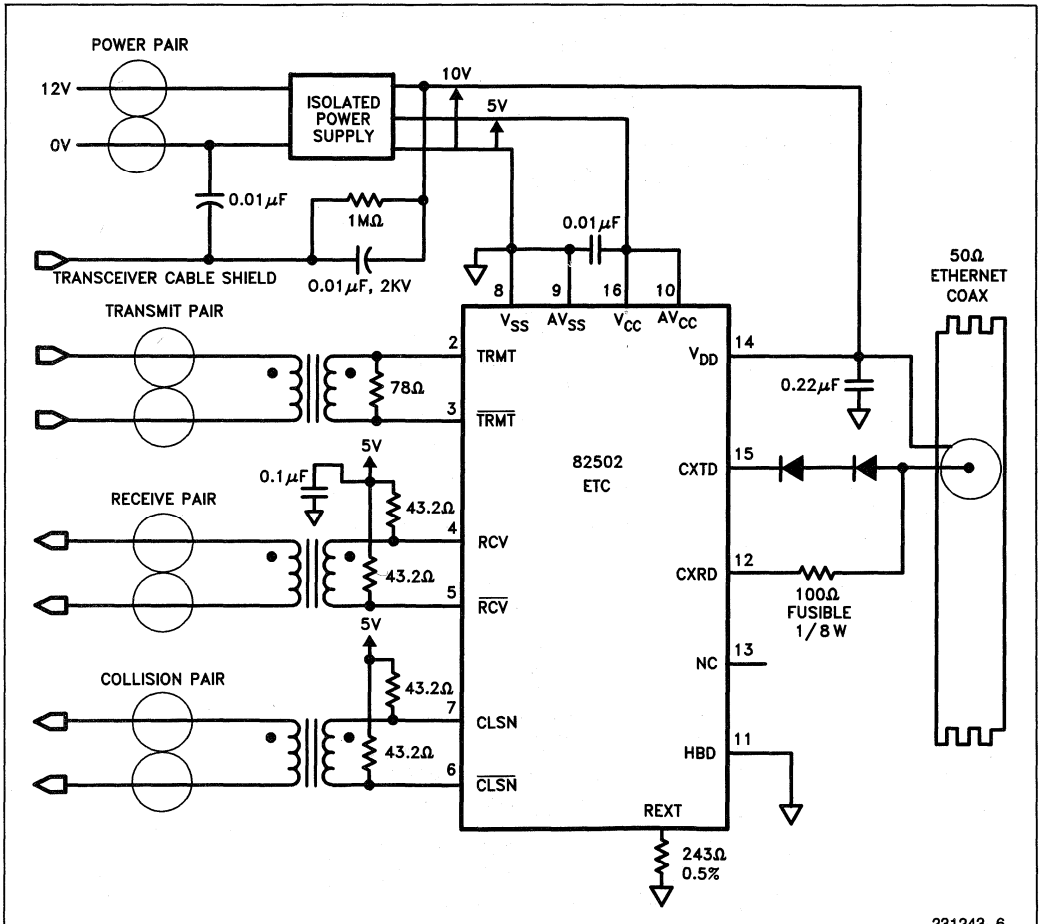
If the HBD input is strapped low, the self test feature of the 82502 is enabled. After each frame transmission, the device transmits a collision presence signal (heartbeat) onto the collision presence pair to indi-

cate that the collision circuitry is operational. The heartbeat begins between 600 ns to 1600 ns following the last positive transition on the transmit pair. The signal's duration is between 0.5 μs and 1.5 μs.

If the HBD input is strapped high, the self test feature as well as the 6.4 μs transmit inhibit timer are disabled.

Design Example

Figure 3 shows the schematic of an IEEE 802.3, 10BASE5 (Ethernet) compatible transceiver based on the 82502. Power to the 82502 is supplied by an isolated DC-to-DC converter which generates +5V ±10% and +10V ±10%. (These voltage levels are referenced to V_{SS}; remember that V_{DD} is connected to earth ground.) Small signal, high conductance diodes which have very small capacitance (less than 3 pF at $V_d = 0$) are connected between the CXTD input pin and the center conductor to reduce the capacitance load on the channel. A 100Ω fusible resistor on the CXRD pin is to protect the network from being brought down in the event of a short circuit fault between the CXRD pin and GND.



NOTES:
1. Pulse transformer inductance ≥ 50 μH, typically 75 μH
2. Isolated power supply: Pulse Engineering Inc. part no. PE-64369
   Reliability Inc. part nos. 2E12R10-5, 2VA12U10-5, and 2VA5U10-5.
3. Isolated power supply may require additional decoupling capacitors on its inputs and outputs.
4. If CXRD/CXTD is not terminated properly (e.g., left floating), the state of CLSN/$\overline{\text{CLSN}}$ is unknown.

Figure 3. Typical 10BASE5 (Ethernet) Transceiver Implementation Using the 82502

Applications

The 82502 is intended for use in high performance, 10 Mbps LAN applications such as IEEE 802.3 10 Base 5 (Ethernet). Ethernet requires that the 82502 transceiver chip be located in a tap box attached directly to the coaxial cable of the network. A drop cable up to 50 meters in length connects the transceiver tap box to the data terminal equipment (DTE), see Figure 4.

In Cheapernet applications, a.k.a. IEEE 802.3 10 Base 2, the 82502 would be located inside the DTE, and transformer coupled to the 82C501, see Figure 5. In both applications, the IEEE specifications require that a DC isolated power supply power the 82502.

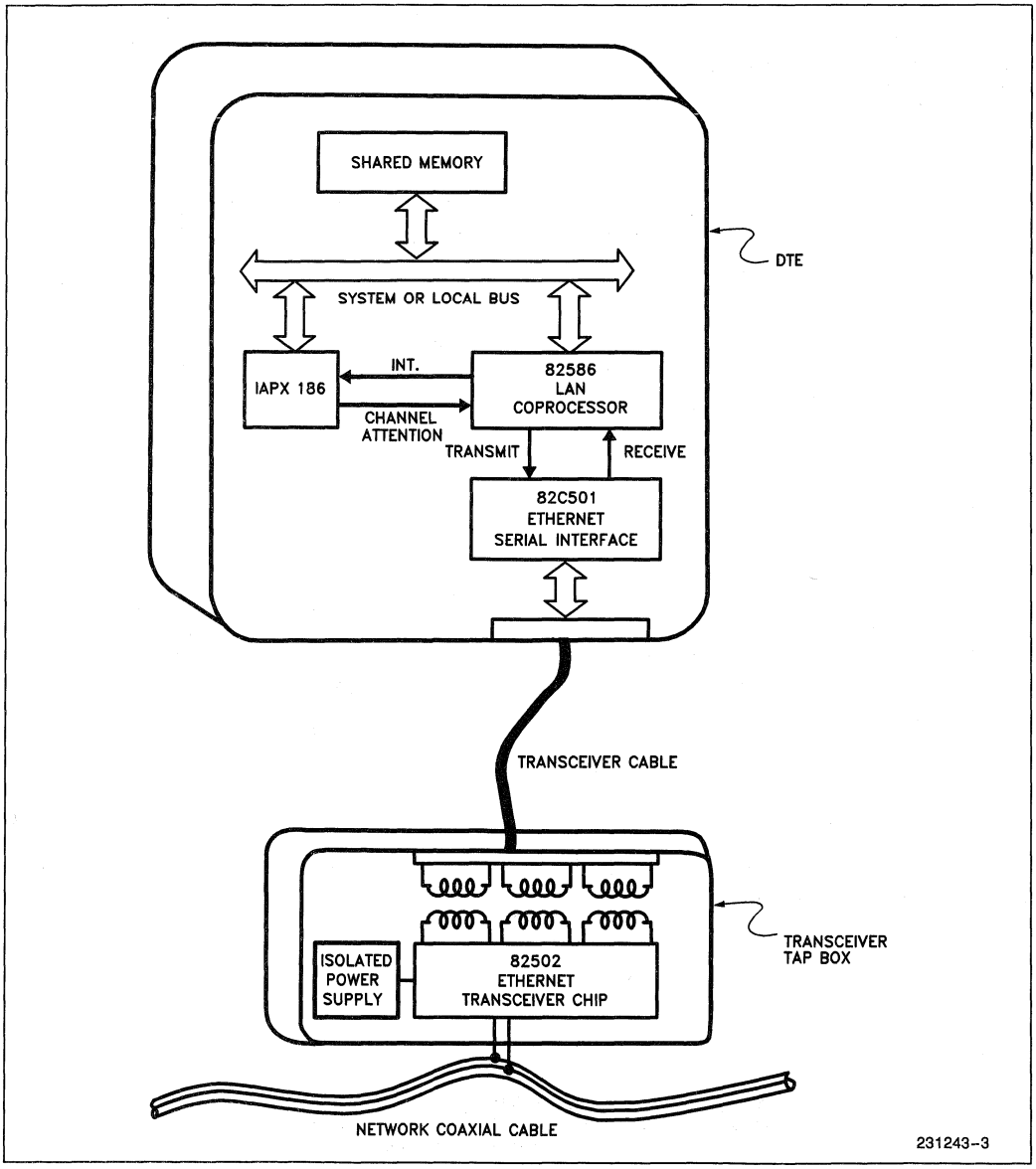


Figure 4. IEEE 802.3 10 Base 5 (Ethernet) configuration supports 100 users per segment, each segment being 500 meters long.

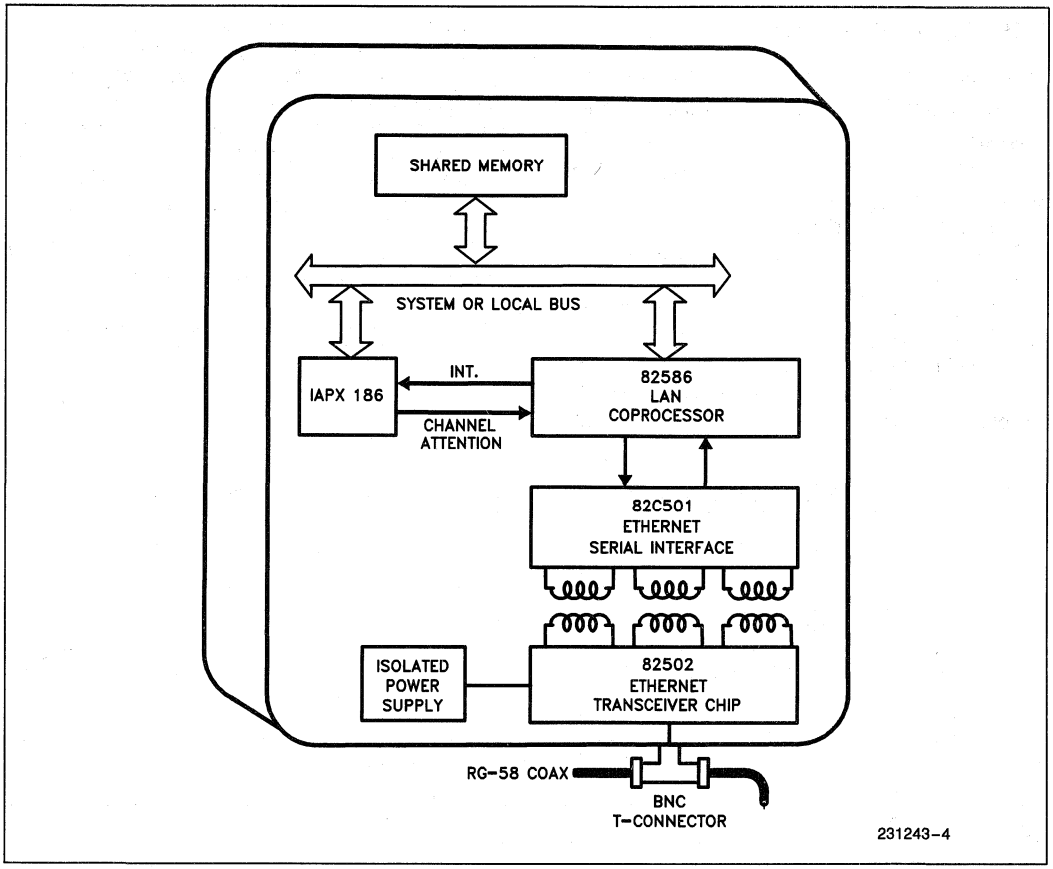


Figure 5. IEEE 802.3 10 Base 2 (Cheapernet) configuration supports 30 users per segment, each segment being 185 meters long.

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias ..... −10°C to +80°C
Storage Temperature ..... −65°C to +150°C
V_{DD} with Respect to V_{SS} ..... −0.5V to +15V
V_{CC} with Respect to V_{SS} ..... −0.5V to +11V
All Input and Output Voltages (Except CXTD and CXRD) with Respect to V_{SS} ..... −0.5V to +11V
CXTD Output Voltage with Respect to V_{SS} ..... −0.5V to +15V
CXRD Input Voltage with Respect to V_{DD} ..... 0V to −15V, > 0V if $I_{CXRD} \leq 200$ mA
Power Dissipation 1.0W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

Operating maximum ratings assume the device transmitting 90% of its operating time.

D.C. CHARACTERISTICS T_A = 0°C–70°C, T_{CMAX} = 115°C, V_{CC} = 5V ±10%, V_{DD} = 10V ±10%

Symbol	Parameter	Limits			Units	Conditions
		Min	Typ	Max		
V_{IDF}	Transceiver Cable Input Differential Voltage	±250		±1400	mV	Base to peak
I_{AVEC}	CXTD Output Current DC Component (Including the effects of timing distortion)	37		45	mA	CXTD: 3.3V to 10.3V
I_{ACC}	CXTD Output Current AC Component	±28	±31	$\pm I_{AVEC}$	mA	
I_{IDC}	CXTD Idle Output Current			5	μA	CXTD: 0V to 11V
V_{ICAC}	Coax Receiver A.C. Voltage	±200			mV	Base to peak
I_{ILC}	Input Leakage Current CXRD	−1		20	μA	
V_{ODF}	Differential Output Swing RCV and CLSN	±0.58		±1.0	V	R_L = 73Ω to 83 Ω ±1% R_{PULLUP} = 43.2Ω ±1% L ≥ 25 μH
I_{IDT}	Output Idle Current Transceiver Cable	3		7 0.2	mA mA	$\overline{RCV}$ and $\overline{CLSN}$ RCV and CLSN
I_{CC}	V_{CC} and AV_{CC} Supply Current			23	mA	
I_{DD}	V_{DD} Supply Current			43	mA	
R_I	Shunt Resistance between V_{DD} and CXRD	1			MΩ	
C_I	Shunt Capacitance between V_{DD} and CXRD		2	3	pF	f = 10 MHz

A.C. CHARACTERISTICS

A.C. Measurement Conditions

I. T_A = 0°C to 70°C, T_{CMAX} = 115°C, V_{CC} = 5V ±10%, V_{DD} = 10V ±10%

II. The A.C. measurements are taken at the following voltage levels for the various kinds of inputs and outputs:

A. Differential Transceiver Cable Inputs and Outputs:

All delay and timing distortion measurements are referenced to the 0V points measured differentially. Rise and fall times are referenced to the 20% and 80% points. The differential voltage swing at the inputs is at least ±275 mV with rise and fall times of 0 to 10 ns.

B. Coaxial Cable Inputs and Outputs:

Delay measurements are referenced to the 50% points of the signals' total swing. V_{DD} is used as the voltage reference. Rise and fall times are measured at the 20% and 80% points. The differential voltage swing at the inputs is at least ±200 mV. The input signal is either a trapezoidal signal with 25 ±5 ns rise/fall times or a sine wave.

In order to simulate a collision, DC voltage of −2.2V referenced to V_{DD} is applied to the CXRD pin.

C. Redundant Transmit Squelch Output:

Delay measurements are referenced to the 50% point of the total swing.

III. The A.C. loading for various outputs is as follows:

A. Transceiver cable drive pins:

RCV, $\overline{RCV}$, CLSN and $\overline{CLSN}$ are each loaded with 10 pF to ground and 43.2Ω to V_{CC}. A parallel load of 25 μH minimum and 78Ω is connected between RCV and $\overline{RCV}$. The same load is present on CLSN and $\overline{CLSN}$.

B. Coaxial Cable Driver Pins:

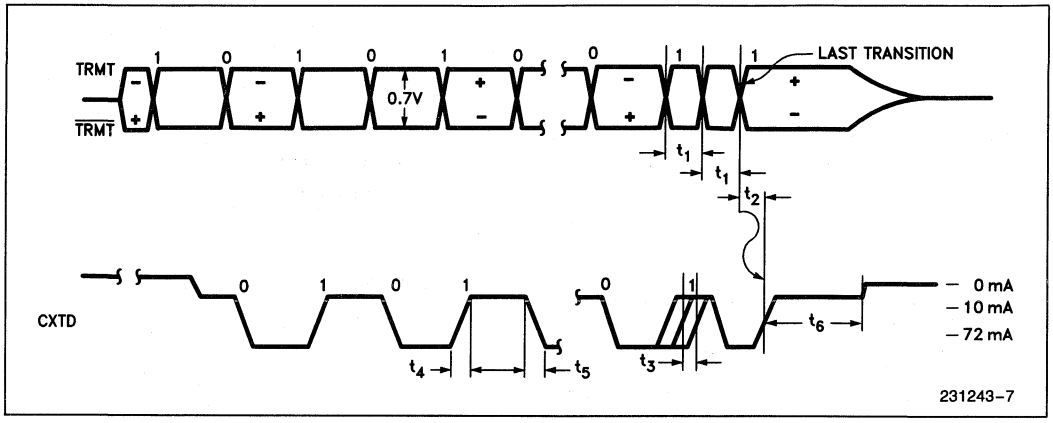
A 25Ω resistor is connected between V_{DD} and CXRD. One small signal diode is connected between the CXRD and CXTD pins such that it conducts when the 82502 is transmitting.

TRANSMIT TIMING

Symbol	Parameter	Limits			Units
		Min	Typ	Max	
t_1	TRMT/$\overline{TRMT}$ Input Pulse Width	44		106	ns
t_2	Transmit Steady State Delay			50	ns
t_3	CXTD Output Edge Jitter			±2	ns
t_4	CXTD Rise Time	20		30	ns
t_5	CXTD Fall Time	20		30	ns
t_6	Last Data Transition to CXTD Disabled			250	ns

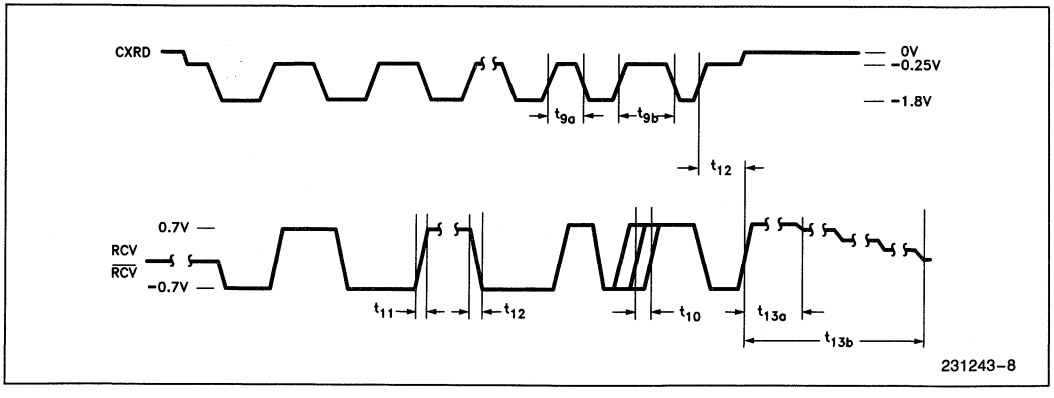
NOTE:
Timings t_7 and t_8 have been deleted.

TRANSMIT TIMING (Continued)



RECEIVE TIMING

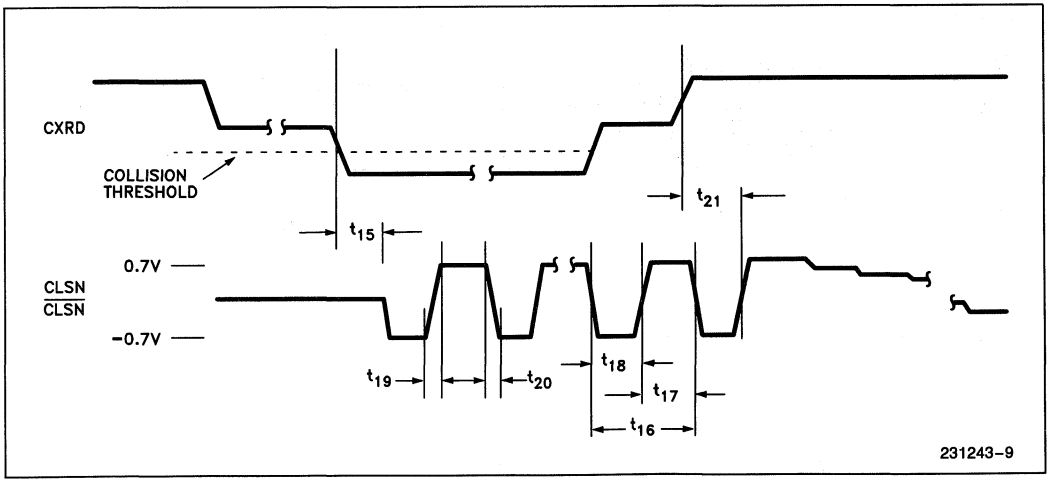
Symbol	Parameter	Limits			Units
		Min	Typ	Max	
t_9	CXRD Input Pulse Width a) Manchester Short Pulse b) Manchester Long Pulse	 28 68		 72 132	 ns ns
t_{10}	RCV/$\overline{\text{RCV}}$ Output Edge Jitter			±6	ns
t_{11}	RCV/$\overline{\text{RCV}}$ Rise Time		3	6	ns
t_{12}	RCV/$\overline{\text{RCV}}$ Fall Time		3	6	ns
t_{13}	Last RCV/$\overline{\text{RCV}}$ Transition to Idle a) RCV/$\overline{\text{RCV}}$ Held High from Last Postive Transition b) RCV/$\overline{\text{RCV}}$ Becomes Idle	 200		 8000	 ns ns
t_{14}	Receive Steady-State Propagation Delay			50	ns



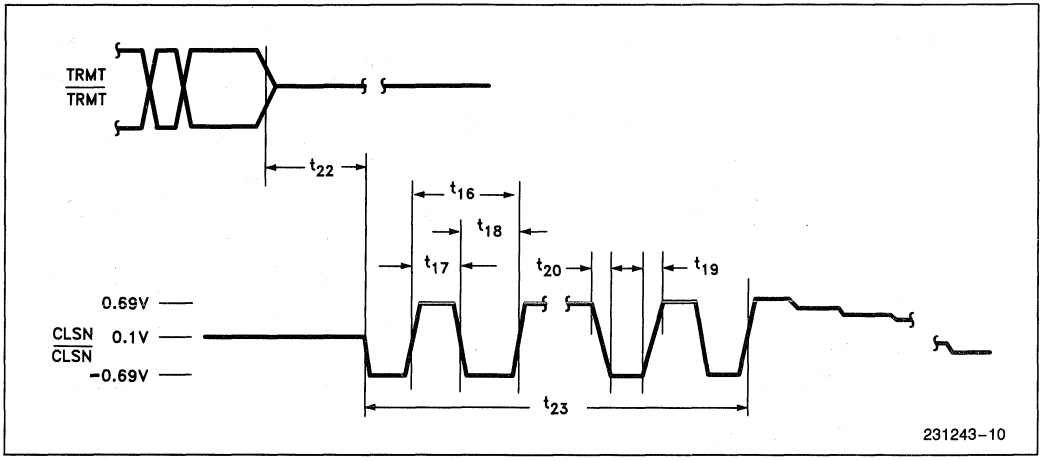
COLLISION AND HEARTBEAT TIMING

Symbol	Parameter	Limits			Units
		Min	Typ	Max	
t_{15}	Delay from Collision on CXRD to First Negative Transition on CLSN/$\overline{\text{CLSN}}$			900	ns
t_{16}	CLSN/$\overline{\text{CLSN}}$ Cycle Time	86		117	ns
t_{17}	CLSN/$\overline{\text{CLSN}}$ High Time	35		70	ns
t_{18}	CLSN/$\overline{\text{CLSN}}$ Low Time	35		70	ns
t_{19}	CLSN/$\overline{\text{CLSN}}$ Rise Time		3		ns
t_{20}	CLSN/$\overline{\text{CLSN}}$ Fall Time		3		ns
t_{21}	Delay from End of Collision to Last CLSN/$\overline{\text{CLSN}}$ Transition			2	μs
t_{22}	Last TRMT/$\overline{\text{TRMT}}$ Transition to Start of Heartbeat Packet	600		1600	ns
t_{23}	Heartbeat Packet Duration	500		1500	ns

COLLISION TIMING



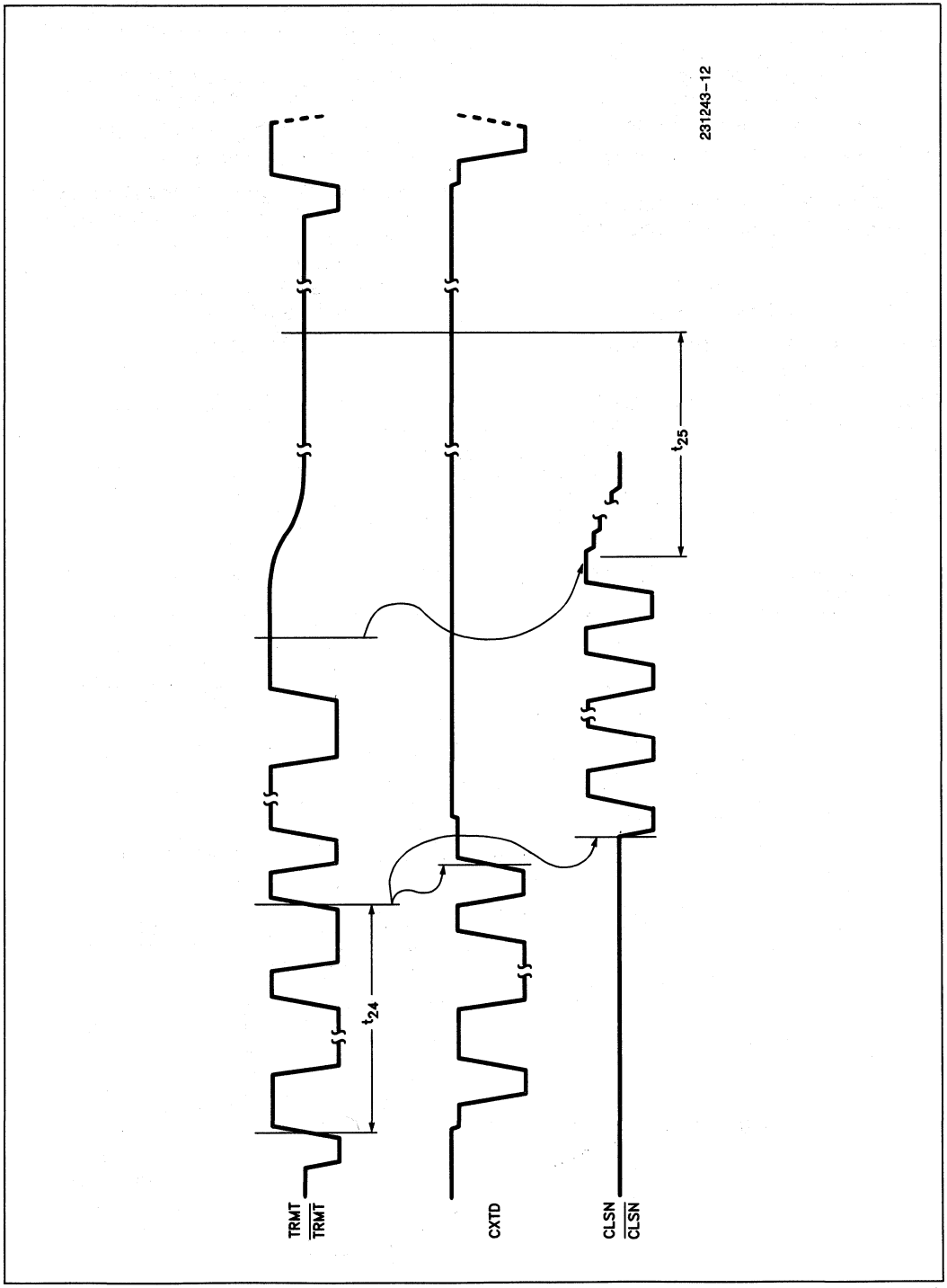
HEARTBEAT TIMING



WATCHDOG TIMER TIMING

Symbol	Parameter	Limits			Units
		Min	Typ	Max	
t_{24}	Watchdog Timer Time Out Time	43	52	61	ms
t_{25}	Transmitter Reset Period after CLSN/$\overline{\text{CLSN}}$ Transmission due to Watchdog Timer Time Out	340	420	550	ms

WATCHDOG TIMER TIMING



82588
High Integration LAN Controller

- **Integrates ISO Layers 1 and 2**
  - **— CSMA/CD Medium Access Control (MAC)**
  - **— On-Chip Manchester, NRZI Encoding/Decoding**
  - **— On-Chip Logic Based Collision Detect and Carrier Sense**
- **Supports Mid-Range Industry Standard LANs**
  - **— StarLAN (IEEE 802.3 1BASE5)**
  - **— IBM/PC Network-Baseband and Broadband**
- **High Level Command Interface Offloads the CPU**
- **Efficient Memory Use Via Multiple Buffer Reception**
- **User Configurable**
  - **— Up to 2 Mb/s Bit Rates with On-chip Encoder/Decoder (High Integration Mode)**
  - **— Up to 5 Mb/s with External Encoder/ Decoder (High Speed Mode)**
- **No TTL Glue Required with iAPX 186 and 188 Microprocessors**
- **Network Management and Diagnostics**
  - **— Short or Open Circuit Localization**
  - **— Station Diagnostics (External Loopback)**
  - **— Self Test Diagnostics Internal Loopback User Readable Registers**

The 82588 is a highly integrated CSMA/CD controller designed for cost sensitive, mid-range Local Area Network (LAN) applications, such as personal computer networks.

At data rates of up to 2 Mb/s, the 82588 provides a highly integrated interface and performs: CSMA/CD Data Link Control, Manchester, Differential Manchester or NRZI encoding/decoding, clock recovery; Carrier Sense, and Collision Detection. This mode is called "High Integration Mode." In the 82588 "High Speed Mode", the user can transfer data at a rate of up to 5 Mb/s. In this mode the physical link functions are done external to the 82588.

The 82588 is available in a 28 pin DIP and 44 lead PLCC package and fabricated in Intel's reliable HMOS II 5 volt technology.

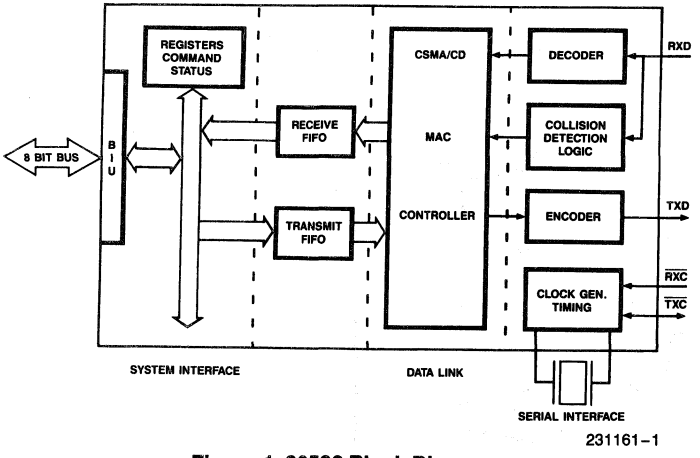


Figure 1. 82588 Block Diagram

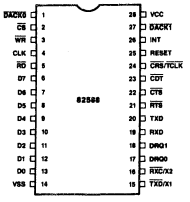


Figure 2. 82588 Pin Configuration (DIP)

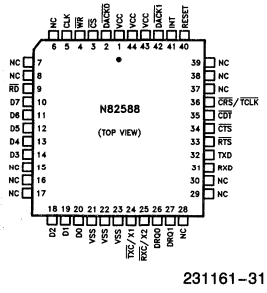


Figure 3. 82588 Pin Configuration (PLCC)

September 1987
Order Number: 231161-004

Table 1. Pin Description

Symbol	Pin No.		Type	Name and Function
	DIP	PLCC		
D7 D6 D5 D4 D3 D2 D1 D0	6 7 8 9 10 11 12 13	10 11 12 13 14 18 19 20	I/O	**DATA BUS:** The Data Bus lines are bi-directional three state lines connected to the system's Data Bus for the transfer of data, commands, status and parameters.
$\overline{RD}$	5	9	I	**READ:** Together with $\overline{CS}$, $\overline{DACK0}$ or $\overline{DACK1}$, Read controls data or status transfers out of the 82588 registers.
$\overline{WR}$	3	4	I	**WRITE:** Together with $\overline{CS}$, $\overline{DACK0}$ or $\overline{DACK1}$, Write controls data or command transfers into the 82588 registers.
$\overline{CS}$	2	3	I	**CHIP SELECT:** When this signal is LOW, the 82588 is selected by the CPU for transfer of command or status. The direction of data flow is determined by the $\overline{RD}$ or $\overline{WR}$ inputs.
CLK	4	5	I	**CLOCK:** System clock. TTL compatible signal.
RESET	25	40	I	**RESET:** A HIGH signal on this pin will cause the 82588 to terminate current activity. This signal is internally synchronized and must be held HIGH for at least four Clock cycles.
INT	26	41	O	**INTERRUPT:** Active HIGH signal indicates to the CPU that the 82588 is requesting an interrupt.
DRQ0	17	26	O	**DMA REQUEST (CHANNEL 0):** This pin is used by the 82588 to request a DMA transfer. DRQ0 remains HIGH as long as 82588 requires data transfers. Burst transfers are done by having the signal active for multiple transfers.
DRQ1	18	27	O	**DMA REQUEST (CHANNEL 1):** This pin is used by the 82588 to request a DMA transfer. DRQ1 remains HIGH as long as 82588 requires data transfers. Burst transfers are done by having the signal active or multiple transfers.
$\overline{DACK0}$	1	2	I	**DMA ACKNOWLEDGE (CHANNEL 0):** When LOW, this input signal from the DMA Controller notifies the 82588 that the requested DMA cycle is in progress. This signal acts like chip select for data and parameter transfer using DMA channel 0.
$\overline{DACK1}$	27	42	I	**DMA ACKNOWLEDGE (CHANNEL 1):** When LOW, this input signal from the DMA controller notifies the 82588 that the requested DMA cycle is in progress. This signal acts like chip select for data and parameter transfer using DMA channel 1.

Table 1. Pin Description (Continued)

Symbol	Pin No.		Type	Name and Function
	DIP	PLCC		
				High Integration Mode
X1/X2	15/16	24/25	I	**OSCILLATOR INPUTS:** These inputs may be used to connect a quartz crystal that controls the internal clock generator for the serial unit. X1 may also be driven by a MOS level clock whose frequency is 8 or 16 times the bit rate of Transmit/Receive data. X2 must be left floating if X1 has an external MOS clock.
				High Speed Mode
$\overline{TxC}$	15	24	I	**TRANSMIT CLOCK:** This signal provides timing information to the internal serial logic, depending upon the mode of data transfer. For NRZ encoding, data is transferred to the TxD pin on the HIGH to LOW clock transition. For Manchester encoding the transmitted bit center is aligned with the $\overline{TxC}$ LOW to HIGH transition.
$\overline{RxC}$	16	25	I	**RECEIVE CLOCK:** This signal provides timing information to the internal serial logic. NRZ data should be provided for reception (RxD). The state of the RxD pin is sampled on the HIGH to LOW transition of $\overline{RxC}$. The operating mode of the 82588 is defined when configuring the chip.
$\overline{TCLK}/\overline{CRS}$	24	36	I	In High Speed Mode, this pin is Carrier Sense, input CRS, and is used to notify the 82588 that there is activity on the serial link.
			O	In High Integration Mode, this pin is Transmit Clock, $\overline{TCLK}$, and is used to output the transmit clock.
$\overline{CDT}$	23	35	I	**COLLISION DETECT:** This input notifies the 82588 that a collision has occurred. It is sensed only if the 82588 is configured for external Collision Detect (external circuitry is then required for detecting the collision).
RxD	19	31	I	**RECEIVE DATA:** This pin receives serial data.
TxD	20	32	O	**TRANSMIT DATA:** This pin transmits data to the Serial Link. This signal is HIGH when not transmitting.
$\overline{RTS}$	21	33	O	**REQUEST TO SEND:** When this signal is LOW, the 82588 notifies an external interface that it has data to transmit. It is forced HIGH after a reset and when transmission is stopped.
$\overline{CTS}$	22	34	I	**CLEAR TO SEND:** CTS enables the 82588 to start transmitting data. Raising this signal to HIGH stops the transmission.
VCC	28	1, 43, 44		**POWER:** +5V Supply
VSS	14	21, 22, 23		Ground

Table 1. Pin Description (Continued)

Symbol	Pin No.		Type	Name and Function
	DIP	PLCC		
NC		6 7 8 15 16 17 28 29 30 37 38 39		**NO CONNECT:** These pins are reserved for future use.

FUNCTIONAL DESCRIPTION

High Integration Mode

The 82588 LAN Controller is a highly integrated CSMA/CD controller for cost sensitive LAN applications such as personal computer networks. Included on chip is a programmable CSMA/CD controller, an NRZI and Manchester encoder/decoder with clock recovery, and two collision detection mechanisms. With the addition of simple transceiver line drivers or RF Modem, the 82588 performs all the major functions of the ISO Physical and Data Link Layers.

CSMA/CD Controller

The 82588 on-chip CSMA/CD controller is programmable, which allows it to operate in a variety of LAN environments, including industry standards such as StarLAN (IEEE 802.3 1BASE5) and the 2 Mb/s IBM PC Network (both baseband and broadband transmission). Programmable parameters include:

- Framing (End of Carrier of SDLC)
- Address field length
- Station priority
- Interframe spacing
- Slot time
- CRC-32 OR CRC-16

Encoder/Decoder

The on-chip NRZI and Manchester encoder/decoder supports data rates up to 2 Mb/s. Manchester encoding is typically used in baseband applications and NRZI is used in broadband applications.

Collision Detection

One of the 82588's unique features is its on-chip logic based collision detection. To ensure a high probability of collision detection two mechanisms are provided. The Code Violation method defines a collision when a transition edge occurs outside the area of normal transitions as specified by either the Manchester or NRZI encoding methods. Bit Comparison method compares the signature of the transmitted frame to the received frame signature (re-calculated by the 82588 while listening to itself). If the signatures are identical the frame is assumed to have been transmitted without a collision.

System Interface

In addition to providing the functions necessary for interfacing to the LAN, the 82588 has a friendly system interface that eases the design effort. First, the 82588 has a high level command interface; that is the CPU sends the 82588 commands such as Transmit or Configure. This means the designer does not have to write low level software to perform these tasks, and it offloads the CPU in the application. Second, the 82588 supports an efficient memory structure called Multiple Buffer Reception in which buffers are chained together while receiving frames. This is an important feature in applications with limited memory, such as personal computers. Third, the 82588 has two independent sixteen byte FIFO's, one for reception and one for transmission. The FIFO's allow the 82588 to tolerate bus latency. Finally the 82588 provides an eight byte data path that supports up to 4 Mbytes/second using external DMA.

Network Management & Diagnostics

The 82588 provides a rich set of diagnostic and network management functions including: internal and external loopback, channel activity indicators, optional capture of all frames regardless of destination address (Promiscuous Mode), capture of collided frames, (if address matches), and time domain reflectometry for locating fault points in the network cable. The 82588 register Dump command ensures reliable software by dumping the content of the 82588 registers into the system memory.

The next section will describe the 82588 system bus interface, the 82588 network interface, and the 82588 internal architecture.

82588/Host CPU Interaction

The CPU communicates with the 82588 through the system's memory and 82588's on-chip registers. The CPU creates a data structure in the memory, programs the external DMA controller with the start address and byte count of the block, and issues the command to the 82588.

The 82588 is optimized for operating with the iAPX 186/188, but due to the small number of hardware signals between the 82588 and the CPU, the 82588 can operate easily with other processors. The data bus is 8 bits wide and there is no address bus.

Chip Select and Interrupt lines are used to communicate between the 82588 and the host as shown in the Figure 3. Interrupt is used by the 82588 to draw the CPU's attention. The Chip Select is used by the CPU to draw the 82588's attention.

There are two kinds of transfer over the bus: Command/Status and data transfers. Command/Status transfers are always performed by the CPU. Data transfers are requested by the 82588, and are typically performed by a DMA controller. The table given in Figure 4 shows the Command/Status and data transfer control signals.

The CPU writes to 82588 using $\overline{CS}$ and $\overline{WR}$ signals. The CPU reads the 82588 status register using $\overline{CS}$ and $\overline{RD}$ signals.

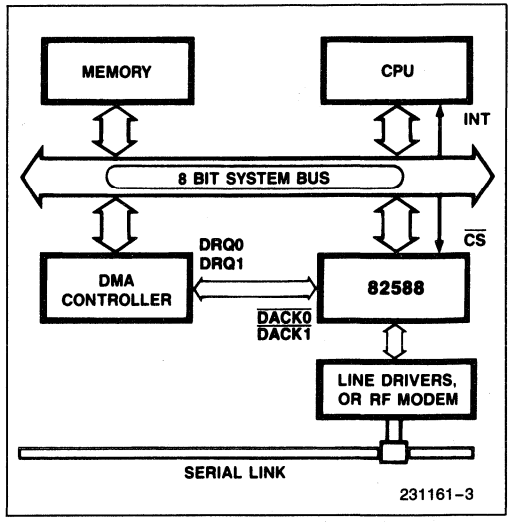


Figure 3. 82588/HOST CPU Interaction

To initiate an operation like Transmit or Configure (see Figure 5), a Write command from CPU to 82588 is issued by the CPU. A Read operation from CPU gives the status of the 82588. Although there are four status registers they're read using the same port in a round robin fashion (Figure 6).

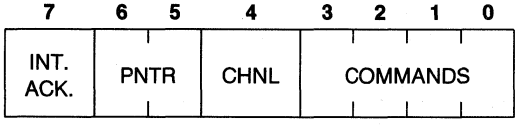
Any parameters or data associated with a command are transferred between the memory and 82588 using DMA. The 82588 has two data channels, each having Request and Acknowledge lines. Typically one channel is used to receive data and other to transmit data and perform all the other initialization and maintenance operations like Configure, Address Set-Up, Diagnose, etc. The channels are identical and can be used interchangeably.

When the 82588 requires access to the memory for parameter or data transfer it activates the DMA request lines and uses the DMA controller to achieve the data transfer. Upon the completion of an operation, the 82588 interrupts the CPU. The CPU then reads results of the operation (the status of the 82588).

Pin Name			Function
$\overline{CS}$*	$\overline{RD}$	$\overline{WR}$	
1	x	x	No transfer to/from Command/Status
0	1	1	
0	0	0	Illegal
0	0	1	Read from status register
0	1	0	Write to Command register
$\overline{DACK0}$[$\overline{DACK1}$]*	$\overline{RD}$	$\overline{WR}$	
1	x	x	No DMA transfer
0	1	1	
0	0	0	Illegal
0	0	1	Data Read from DMA channel 0 [or 1]
0	1	0	Data Write to DMA channel 0 [or 1]

* Only one of $\overline{CS}$, $\overline{DACK0}$ and $\overline{DACK1}$ may be active at any time.

Figure 4. Databus Control Signals and Their Functions



COMMAND REGISTER

COMMANDS		VALUE	COMMANDS		VALUE
NOP	—	0	ABORT	—	13
IA-SETUP	—	1	RECEIVER-ENABLE	—	8
CONFIGURE	—	2	ASSIGN NEXT BUF	—	9
MC-SETUP	—	3	RECEIVE-DISABLE	—	10
TRANSMIT	—	4	STOP-RECEPTION	—	11
TDR	—	5	RESET	—	14
DUMP	—	6	FIX PTR	—	15 (CHNL=1)
DIAGNOSE	—	7	RLS PTR	—	15 (CHNL=0)
RETRANSMIT	—	12			

Figure 5. Command Format and Operation Values

	7	6	5	4	3	2	1	0
Status 0	INT	RCV	EXEC	CHNL	EVENT			
Status 1				RESULT 1				
Status 2				RESULT 2				
Status 3	RCV CHNL	RCV STATE		BUFF CHNG NO. OF BUF		EXEC CHNL	EXEC STATE	

EVENTS		VALUE (STATUS 0)
IA-SETUP-DONE	—	1
CONFIGURE-DONE	—	2
MC-SETUP-DONE	—	3
TRANSMIT-DONE	—	4
TDR-DONE	—	5
DUMP-DONE	—	6
DIAGNOSE-PASSED	—	7
END OF FRAME	—	8
REQUEST NEXT BUFFER	—	9
RECEPTION ABORTED	—	10
RETRANSMIT-DONE	—	12
EXECUTION-ABORTED	—	13
DIAGNOSE-FAILED	—	15

Figure 6. Status Registers and Event Values

Transmitting a Frame

To transmit a frame, the CPU prepares a Transmit Data Block in memory as shown in Figure 7. Its first two bytes specify the length of the rest of the block. The next few bytes (Up to 6 bytes long) contain the destination address of the node it is being sent to. The rest of the block is the data field. The CPU programs the DMA controller with the start address of the block, length of the block and other control information and then issues the Transmit command to the 82588.

Upon receiving the command, the 82588 fetches the first two bytes of the block to determine the length of the block. If the link is free, and the first data byte was fetched, the 82588 begins transmitting the preamble and concurrently fetches the bytes from the Transmit Data Block and loads them into a 16 byte FIFO to keep them ready for transmitting. The FIFO is a buffer between the serial and parallel part of the 82588. The on-chip FIFOs help the 82588 to tolerate system bus latency as well as provide efficient usage of system bandwidth.

The destination address is sent out after the preamble. This is followed by the source or the station individual address, which is stored earlier on the 82588 using the IA-SETUP command. After that, the entire information field is transmitted followed by a CRC field calculated by the 82588. If during the transmission of the frame, a collision is encountered, then the transmission is aborted and a jam pattern is sent out after completion of the preamble. The 82588 generates an Interrupt indicating the experience of a collision and the frame has to be re-transmitted. Retransmission is done by the CPU exactly as the Transmit command except the Re-Transmit command keeps track of the number of collisions encountered. When the 82588 gets the Retransmit command and the exponential back-off time is expired, the 82588 transmits the frame again. The transmitted frame can be coded to either Manchester, Differential Manchester or NRZI methods.

Collision Detection

The 82588 eliminates the need for external collision detection logic, in most applications, while easing or eliminating the need for complex transceivers. Two algorithms are used for collision detection: Bit Comparison and Code Violation. The Bit Comparison Method is useful in Broadband networks where there are separate transmit and receive channels. Bit Comparison compares the "signature" of the transmitted data and received data at the end of the collision window in any network configuration. This algorithm calculates the CRC after a programmable number of transmitted bits, holds this CRC in a register, and compares it with received data's CRC. A CRC or "signature" difference indicates a collision. The code violation is detected if the encoding of the received data has any bit that does not fit the encoding rules. The code violation method is useful in short bus topology and serial backplane applications where bit attenuation over the bus is negligible.

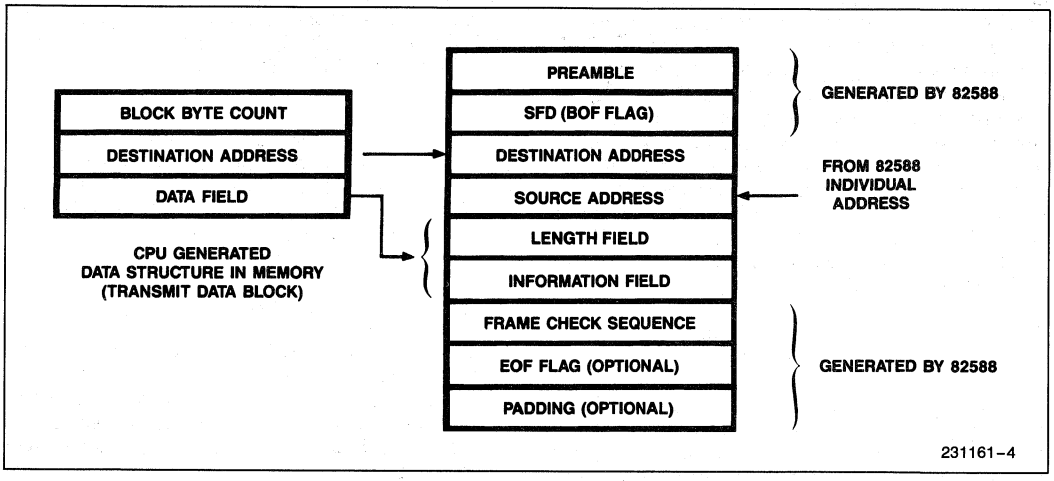


Figure 7. The 82588 Frame Structure and location of Data element in System Memory

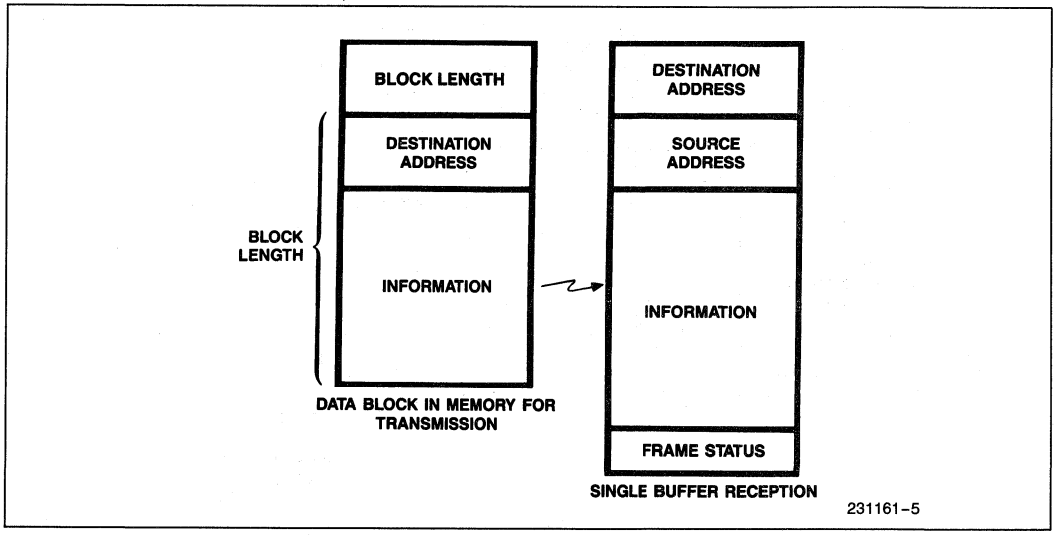


Figure 8. Single Buffer Reception

Receiving a Frame

The 82588 can receive a frame when its receiver has been enabled. The received frame is decoded by either on-chip Manchester, Differential Manchester or NRZI decoders in High Integration Mode and NRZI in High Speed Mode. The 82588 checks for an address match for an individual address, a Mulitcast address or a Broadcast address. In the Promiscuous mode the 82588 receives all frames. Only when the address match is successful does the 82588 transfer the frame to the memory using the DMA controller. Before enabling the receiver, the CPU makes a memory buffer area available to the Receive Unit and programs the starting address of the DMA controller. The received frame is transferred to the memory buffer in the format shown in Figure 8. This method of reception is called "Single Buffer" reception. The entire frame is contained in one continuous buffer. Upon completion of reception the total number of bytes written into the memory buffer is loaded into status registers 1 and 2 and the status of the reception itself is appended to the received frame. An interrupt to the CPU follows.

If the frame size is unknown, memory usage can be optimized by using "Multiple Buffer" reception.

This way the user does not have to allocate large memory space for short frames. Instead, the 82588 can dynamically allocate memory space as it receives frames. This method requires both DMA channels alternately to receive the frame. As the frame reception starts, the 82588 interrupts the CPU and automatically requests assignment of the next sequential buffer. The CPU does this and loads the second DMA channel with the next buffer information so that the 82588 can immediately switch to the other channel as soon as the current buffer is full. When the 82588 switches from the first to the second buffer it again interrupts the CPU requesting it to allocate another buffer on the other (previous) channel in advance. This process continues until the entire frame is received. The received frame is spread over multiple memory buffers. The link between the buffers is easily maintained by the CPU using a buffer chain descriptor structure in memory (see Figure 9).

This dynamic (pre) allocation of memory buffers results in efficient use of available storage when handling frames of widely differing sizes. Since the buffers are pre-allocated one block in advance, the system is not time critical.

80188 Based System

Figure 10 shows a high performance, high-integration configuration of the 82588 with the 80188 in a typical iAPX188-based microcomputer. The 80188 controls the 82588, as well as providing DMA control services for data transfer, using its on-chip two channel DMA controller.

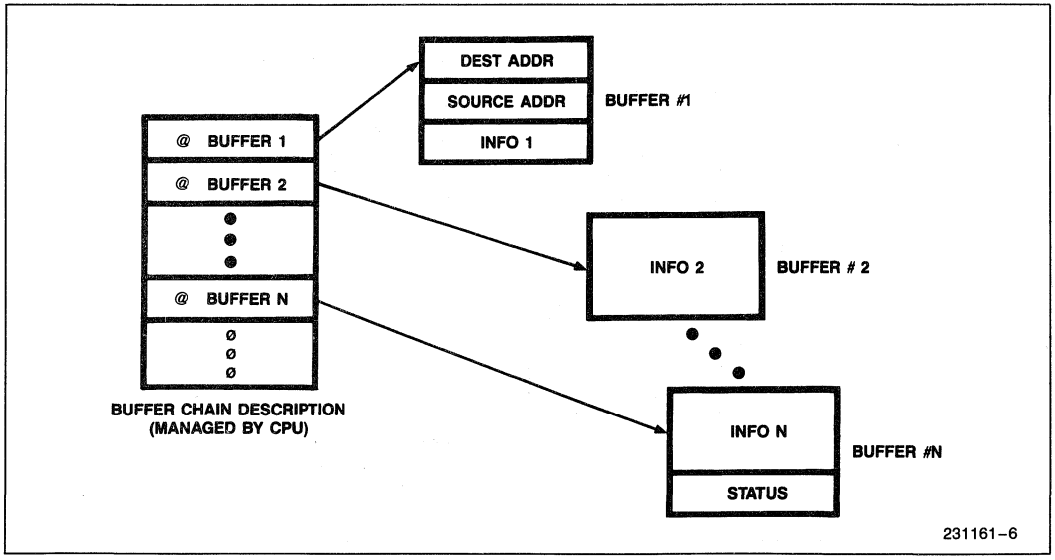


Figure 9. Multiple Buffer Reception

Link Interface

The Serial Interface Mode configuration parameter selects either a highly integrated Direct Link interface (High Integration Mode) or a highly flexible Transceiver Interface (High Speed Mode).

Application

In the High Integration Mode it is possible to connect the 82588 on a very short "Wired OR" link, on a longer twisted pair cable, or a broadband connection.

Twisted Pair Connection

The link consists of a twisted pair that interconnects the 82588. The transmit data pin is connected via a driver and the receive data pin is connected via a buffer. The twisted pair must be properly terminated to prevent reflections.

In the minimum configuration, TxD and RxD are connected to the twisted pair and $\overline{CTS}$ is grounded. The 82588 may control the driver with the $\overline{RTS}$ pin. It is also possible to use external circuitry for performing collision detection, and feeding it to the 82588 through the $\overline{CDT}$ pin.

Broadband Connection

The 82588 supports data communications over a broadband link in both its modes. Proper MODEM interface should be provided. Collision Detection by Bit Comparison, in High Interface Mode, can be applied to transmission over broadband links.

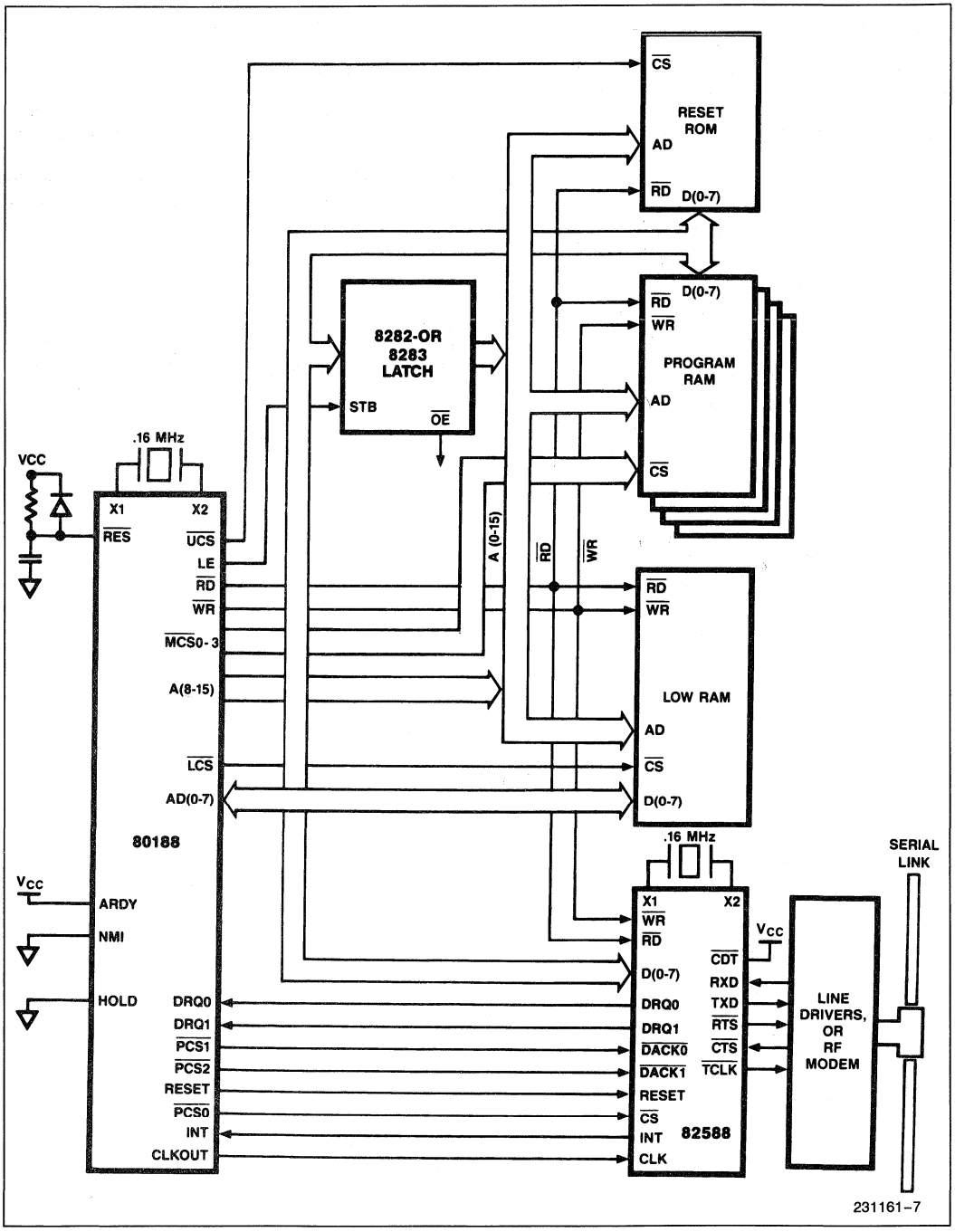


Figure 10. 80188 Based System

Absolute Maximum Ratings*

Ambient Temperature Under Bias ....0°C to +70°C
Storage Temperature .....−65°C to +150°C
Voltage on Any Pin With
Respect to Ground .....−1.0V to 7V
Power Dissipation .....1.7 Watts

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

D.C. Characteristics

(T_A = 0°C to +70°C; T_C (DIP) = 52°C to 108°C, T_C (PLCC) = 63°C to 116°C; VCC = +5V ±10%)

$\overline{TxC}$, $\overline{RxC}$ have MOS levels (See VMIL, VMIH). All other signals have TTL levels (See VIL, VIH, VOL, VOH).

Symbol	Parameter	Min	Max	Units	Test Conditions
VIL	Input Low Voltage (TTL)	−0.5	+0.8	V	
VIH	Input High Voltage (TTL)	2.0	VCC + 0.5	V	
VOL	Output Low Voltage (TTL)		0.45	V	IOL = 2.0 mA
VOH	Output High Voltage (TTL)	2.4		V	IOH = −400 μA
VMIL	Input Low Voltage (MOS)	−0.5	0.6	V	
VMIH	Input High Volatge (MOS)	3.9	VCC + 0.5	V	
ILI	Input Leakage Current		+10	μA	0 = VIN = VCC
ILO	Output Leakage Current		±10	μA	0.45 = VOUT = VCC
ICC	Power Supply Current		400 300	mA mA	T_A = 0°C T_A = +70°C

A.C. Characteristics

(T_A = 0°C to +70°C; T_C (DIP) = 52°C to 108°C, T_C (PLCC) = 63°C to 116°C; VCC = +5V ±10%)

System Clock Parameters

Symbol	Parameter	Min	Max	Units	Test Conditions
T1	CLK Cycle Period	125		ns	
T2	CLK Low Time	53	1000	ns	*5
T3	CLK High Time	53		ns	*6
T4	CLK Rise Time		15	ns	*1
T5	CLK Fall Time		15	ns	*2

A.C. Characteristics (Continued)

Symbol	Parameter	Min	Max	Units	Test Conditions
Reset Parameters					
T6	Reset Active to Clock Low	20		ns	*3
T8	Reset Pulse Width	4T1		ns	
T9	Control Inactve After Reset		T1	ns	
Interrupt Timing Parameters					
T10	CLK High to Interrupt Active		85	ns	*4
T11	$\overline{WR}$ Idle to Interrupt Idle		85	ns	*4
Write Parameters					
T12	$\overline{CS}$ or $\overline{DACK0}$ or $\overline{DACK1}$ Setup to $\overline{WR}$ Low	0		ns	
T13	$\overline{WR}$ Pulse Width	95		ns	
T14	$\overline{CS}$ or $\overline{DACK0}$ or $\overline{DACK1}$ Hold After $\overline{WR}$ High	0		ns	
T15	Data Setup to $\overline{WR}$ High	75		ns	
T16	Data Hold After $\overline{WR}$ High	0		ns	
Read Parameters					
T17	$\overline{CS}$ or $\overline{DACK0}$ or $\overline{DACK1}$ Setup to $\overline{RD}$ Low	0		ns	
T18	$\overline{RD}$ Pulse Width	95		ns	
T19	$\overline{CS}$ or $\overline{DACK0}$ or $\overline{DACK1}$ Address Valid After $\overline{RD}$ High	0		ns	
T20	$\overline{RD}$ Low to Data Valid		80	ns	*7
T21	Data Float After $\overline{RD}$ High		55	ns	*7
DMA Parameters					
T22	CLK Low to DRQ0 or DRQ1 Active		85	ns	*4
T23	$\overline{WR}$ or $\overline{RD}$ Low to DRQ0 or DRQ1 Inactive		60	ns	*4

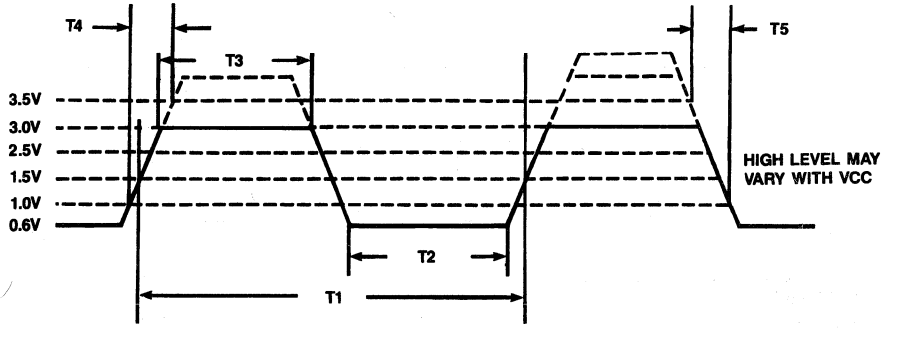
NOTES:
*1—0.8V–2.0V
*2—2.0V–0.8V
*3—to guarantee recognition at next clock
*4—CL = 50 pF
*5—measured at 1.5V
*6—measured at 1.5V
*7—CL = 20 pF–200 pF

A.C. TESTING INPUT/OUTPUT WAVEFORM

231161-8

AC Testing Inputs are Driven at 2.4V for a Logic 1 and 0.45V for a Logic Q Timing Measurements are Made at 1.5V for Both a Logic 1 and Q:
Rise and Fall Time of Input/Output Signals are Measured Between 0.8V to 2.0V Respectively.

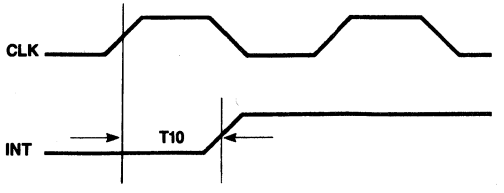
TTL Input/Output Voltage Levels for Timing Measurements



231161-9

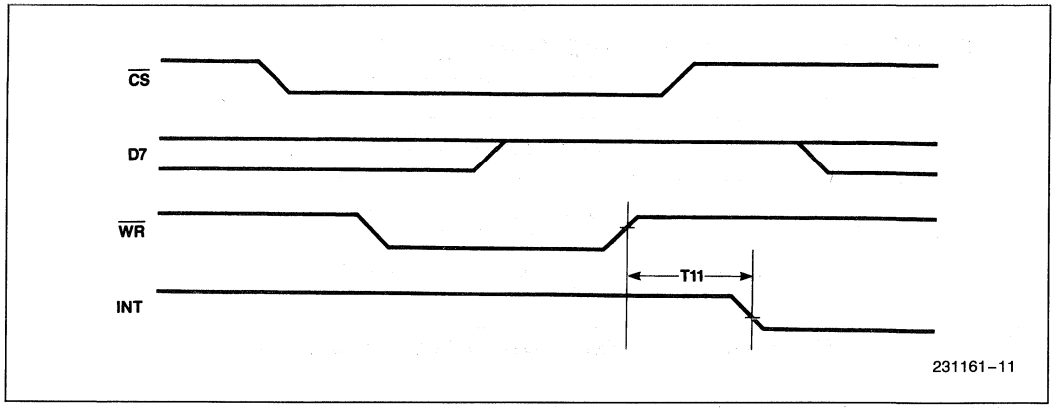
Rise and Fall Time of Input Signals are Measured Between 1.0V to 3.5V Respectively.

Clocks MOS Input Voltage Levels for Timing Measurements

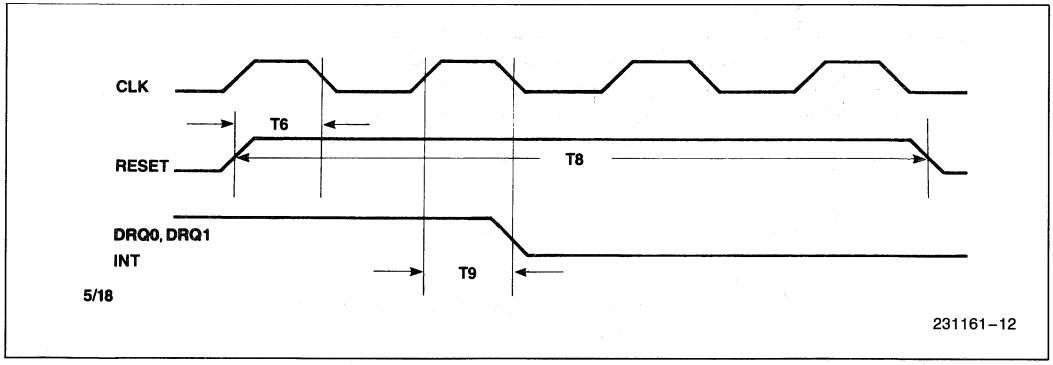


231161-10

Interrupt Timing (Going Active)



Interrupt Timing (Going Inactive)



Reset Timing

Serial Interface A.C. Timing Characteristics
High Integration Mode

$\overline{TFC}$ is the crystal or serial clock input at the X1 pin. When a serial clock is provided at the X1 pin, the maximum capacitive load allowed on the X2 pin is 15 pF.
$\overline{TFC}$ Frequency Range:

For Oscillator Frequency = 1 to 16 MHz (High)		
	×8 Sampling	×16 Sampling
$\overline{TCLK}$ Frequency	0.125−2 MHz	62.5 kHz−1 MHz
T29 = $\overline{TCLK}$ Cycle Time	8 × T24	16 × T24
T30 = $\overline{TCLK}$ High Time	T24 (Typically)	T24 (Typically)
T31 = $\overline{TCLK}$ Low time	7 × T24 (Typically)	15 × T24 (Typically)

For Oscillator Frequency = 0 to 1 MHz (Low)*		
	×8 Sampling	×16 Sampling
$\overline{TCLK}$ Frequency	0−0.125 MHz	0−6.25 kHz
T29 = $\overline{TCLK}$ Cycle Time	8 × T24	16 × T24
T30 = $\overline{TCLK}$ High Time	T25 (Typically)	T25 (Typically)
T31 = $\overline{TCLK}$ Low Time	7 × T24 + T26 (Typically)	15 × T24 + T26 (Typically)
*A non-symmetrical clock should be provided so that T25 is less than 1000 ns. T24 = Serial Clock Period T25 = Serial Clock High Time T26 = Serial Clock Low Time		

High Speed Mode

- Applies for $\overline{TxC}$, $\overline{RxC}$
- f max = 5 MHz ±100 ppm
- For Manchester, symmetry is required: T_{63}, $T_{64} = \frac{1}{2f} \pm 5\%$

High Integration Mode

Symbol	Parameter	Min	Max	Units	Test Conditions
External (Fast) Clock Parameters					
T24	Fast Clock Cycle	62.5		ns	*1
T25	$\overline{TFC}$ High Time	20	1000	ns	*1, *14
T26	$\overline{TFC}$ Low Time	20		ns	*1
T27	$\overline{TFC}$ Rise Time		5	ns	*1
T28	$\overline{TFC}$ Fall Time		5	ns	*1
Transmit Clock Parameters					
T29	Transmit Clock Cycle	500		ns	*3, *12
T30	$\overline{TCLK}$ High Time	*8	1070	ns	*3
T31	$\overline{TCLK}$ Low Time	*9			*3
T32	$\overline{TCLK}$ Rise Time		15	ns	*3
T33	$\overline{TCLK}$ Fall Time		15	ns	*3

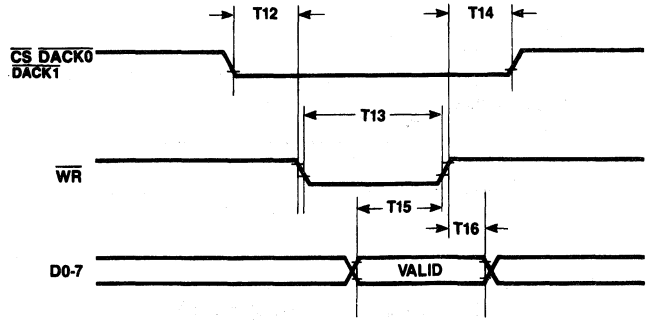
High Integration Mode (Continued)

Symbol	Parameter	Min	Max	Units	Test Conditions
Transmit Data Parameters (Manchester, Differential Manchester)					
T34	TxD Transition-Transition	4T24-10		ns	*12
T35	$\overline{TCLK}$ Low to TxD Transition Half Bit Cell		*10		*2, *12
T36	$\overline{TCLK}$ Low to TxD Transition Full Bit Cell		*11		*2, *12
T37	TxD Rise Time		15	ns	*2
T38	TxD Fall Time		15	ns	*2
Transmit Data Parameters (NRZI)					
T39	TxD Transition-Transition	8T24-10		ns	*12
T40	$\overline{TCLK}$ Low to TxD Transition		*10		*2, *12
T41	$\overline{TxD}$ Rise Time		15	ns	*2
T42	TxD Fall Time		15	ns	*2
$\overline{RTS}$, $\overline{CTS}$, Parameters					
T43	$\overline{TCLK}$ Low To $\overline{RTS}$ Low		*10		*3, *12
T44	$\overline{CTS}$ Low to $\overline{TCLK}$ Low $\overline{CTS}$ Setup Time	65		ns	
T45	$\overline{TCLK}$ low to $\overline{RTS}$ High		*10		*3, *12
T46	$\overline{TCLK}$ Low to $\overline{CTS}$ Invalid. $\overline{CTS}$ Hold Time	20		ns	*4, *13
T47	$\overline{CTS}$ High to $\overline{TCLK}$ Low. $\overline{CTS}$ Setup Time to Stop Transmission	65		ns	*4
IFS Parameters					
T48	Interframe Delay	*5			
Collision Detect Parameter					
T49	$\overline{CDT}$ Low to $\overline{TCLK}$ High. External Collision Detect Setup Time	50		ns	*13
T50	$\overline{CDT}$ High to $\overline{TCLK}$ Low	50		ns	*13
T51	$\overline{TCLK}$ High to $\overline{CDT}$ Inactive. $\overline{CDT}$ Hold Time	20		ns	*13

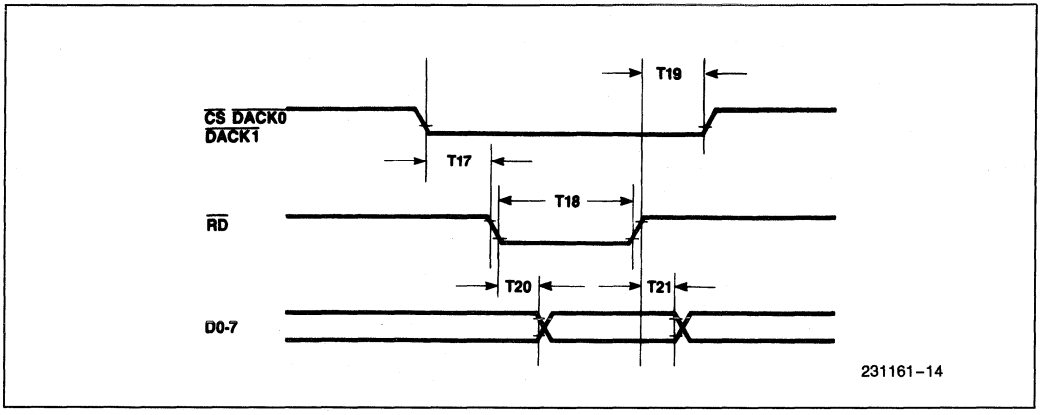
High Integration Mode (Continued)

Symbol	Parameter	Min	Max	Units	Test Conditions
Collision Detect Parameters (Continued)					
T52	$\overline{CDT}$ Low to Jamming Start		*6		
T53	Jamming Period	*7			
Received Data Parameters (Manchester)					
T54	RxD Transition-Transition	4T24		ns	*12
Received Data Parameters (Manchester)					
T55	RxD Rise Time		10	ns	*1
T56	RxD Fall Time		10	ns	*1
Received Data Parameters (NRZI)					
T57	RxD Transition-Transition	8T24		ns	*12
T58	RxD Rise Time		10	ns	*1
T59	RxD Fall Time		10	ns	*1

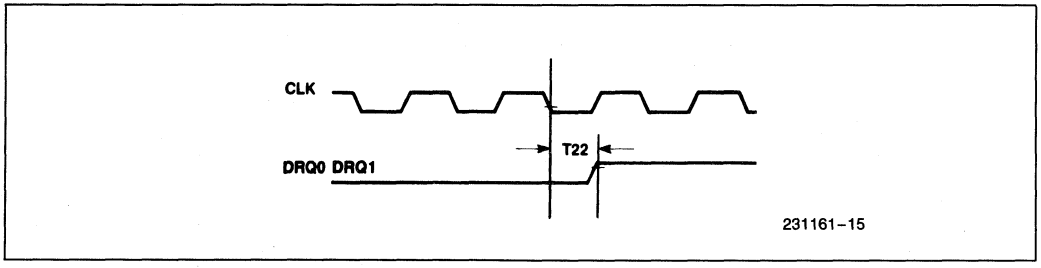
NOTES:
*1—MOS levels.
*2—1 TTL load + 50 pF.
*3—1 TTL load + 100 pF.
*4—Abnormal end to transmission: $\overline{CTS}$ expires before $\overline{RTS}$.
*5—Programmable value: T48 = NIFS × T29 (ns) NIFS—the IFS configuration value.
If NIFS is less than 12, then it is enforced to 12.
*6—Programmable value:
T52 = NCDF × T29 + (12 to 15) × T29 (if collision occurs after preamble).
*7—T53 = 32 × T29
*8—Depends on T24 frequency range:
High Range: T24 − 10
Low Range: T25 − 10
*9—T31 = T29 − T30 − T32 −T33
*10—2T24 + 40 ns
*11—6T24 + 40 ns
*12—For ×16 sampling clock parameter minimum value should be multiplied by a factor of 2.
*13—To guarantee recognition on the next clock.
*14—62.5 ns minimum in Low Range.



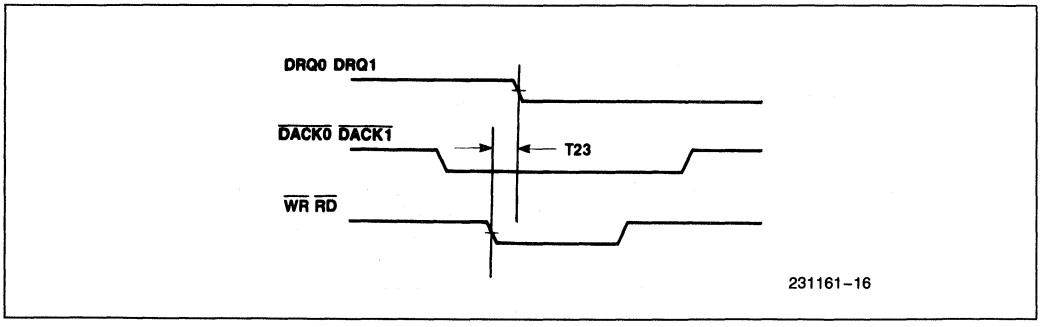
Write Timing



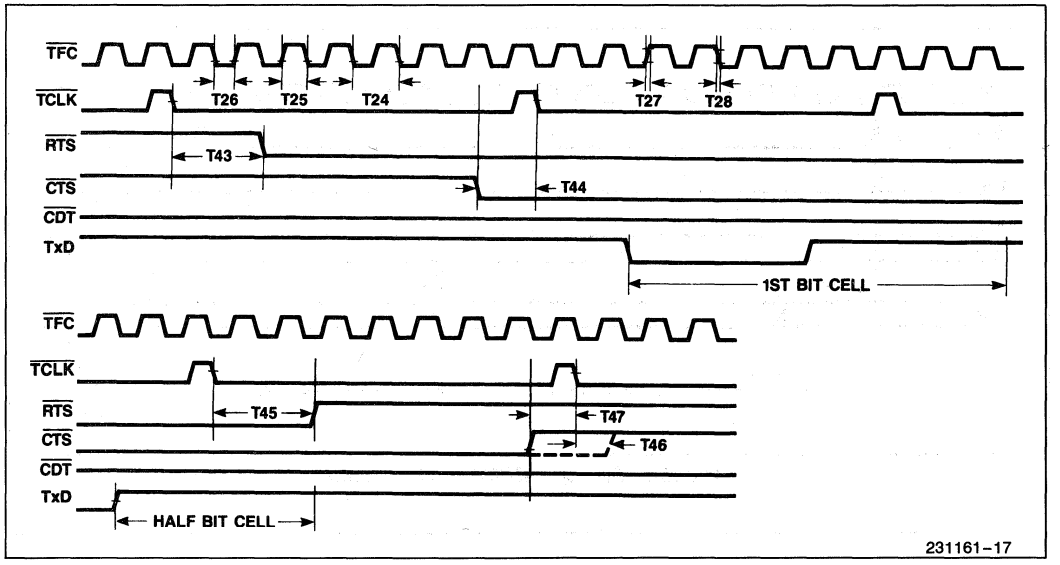
Read Timing



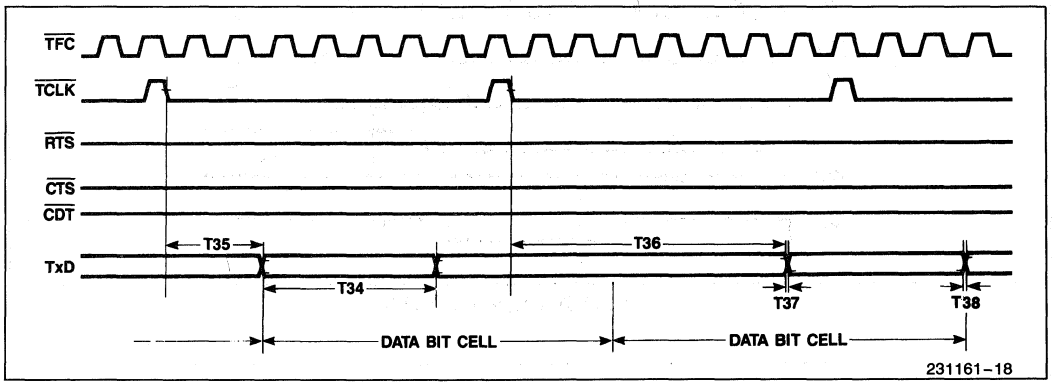
DMA Request (Going Active)



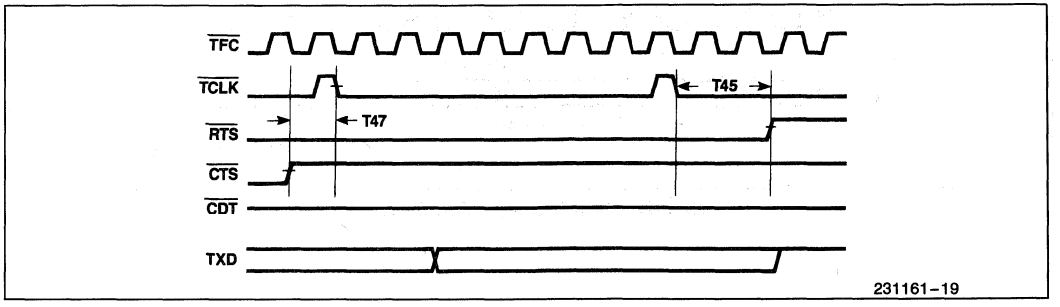
DMA Request (Going Inactive)



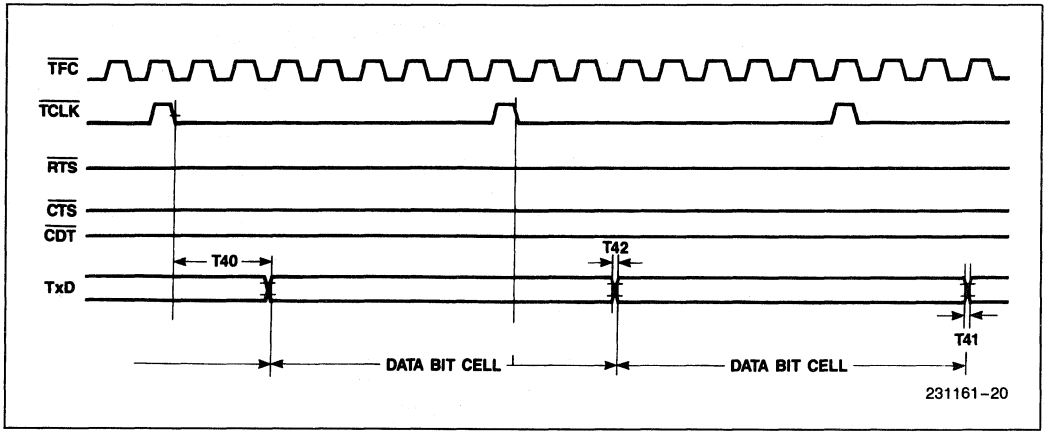
Transmit Timings: Clocks $\overline{RTS}$ and $\overline{CTS}$



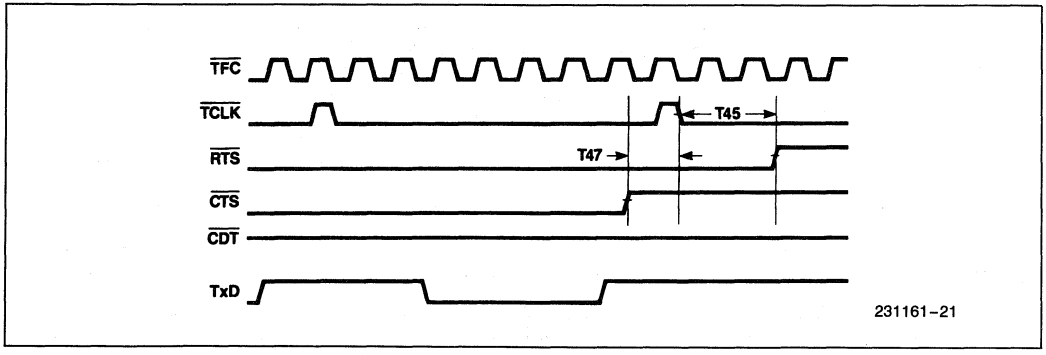
Transmit Timings—Manchester Data Encoding



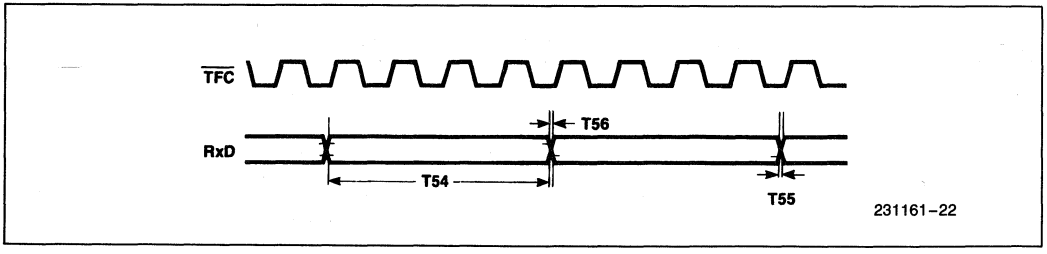
Transmit Timings—Lost $\overline{CTS}$



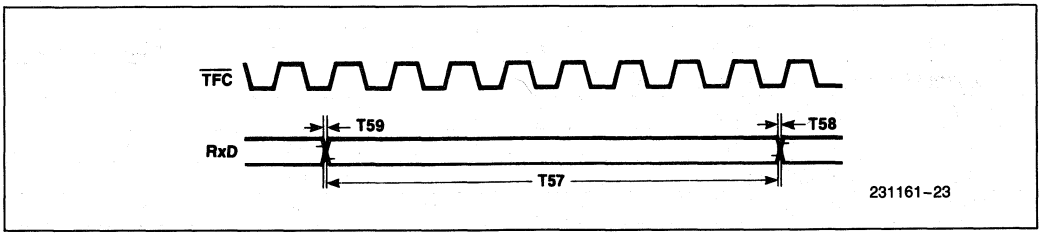
Transmit Timings—NRZI Data Encoding



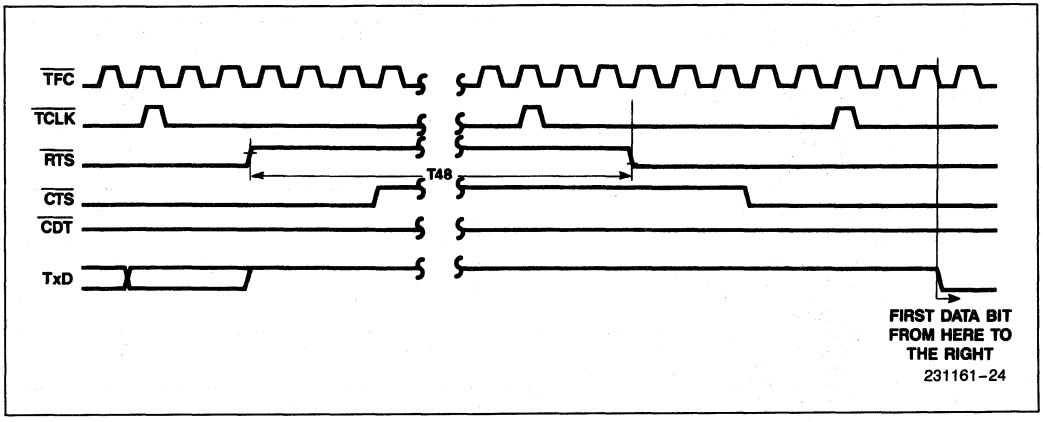
Transmit Timings—Lost $\overline{CTS}$



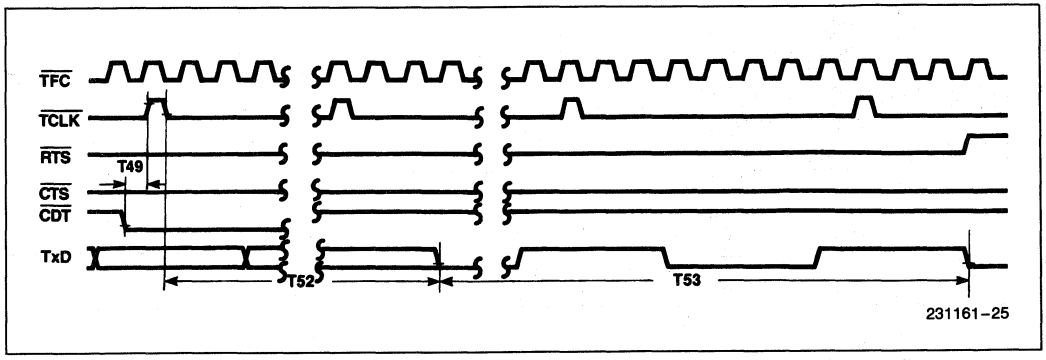
Receive Data Timings (Manchester)



Receive Data Timings (NRZI)



Transmit Timings—Interframe Spacing



Transmit Timings—Collision Detect and Jamming

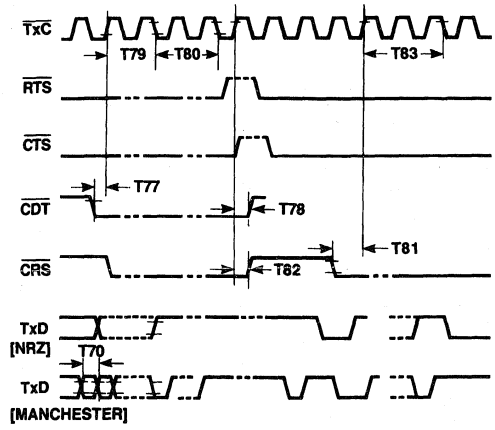
High Speed Mode

Symbol	Parameter	Min	Max	Units	Test Conditions
Transmit/Receive Clock Parameters					
T60	$\overline{RxC}$ $\overline{TxC}$ Cycle	200	*13	ns	
T61	$\overline{TxC}$ Rise Time		10	ns	*1
T62	$\overline{TxC}$ Fall Time		10	ns	*1
T63	$\overline{TxC}$ High	80	1000	ns	*1, *3
T64	$\overline{TxC}$ Low	80		ns	*1, *3
Transmit Data Parameters					
T65	TxD Rise Time		20	ns	*4
T66	TxD Fall Time		20	ns	*4
T67	$\overline{TxC}$ Low to TxD Valid		60	ns	*4, *6
T68	$\overline{TxC}$ Low to TxD Transition		60	ns	*2, *4
T69	$\overline{TxC}$ High to TxD Transition		60	ns	*2, *4
T70	TxD Transition— Transition	70			*2, *4
T71	$\overline{TxC}$ Low to TxD High (At the Transmission End)		60	ns	*4
$\overline{RTS}$, $\overline{CTS}$ Parameters					
T72	$\overline{TxC}$, Low to $\overline{RTS}$ Low Time to Activate $\overline{RTS}$		60	ns	*5
T73	$\overline{CTS}$ Low to $\overline{TxC}$ Low $\overline{CTS}$ Setup Time	65		ns	
T74	$\overline{TxC}$ Low to $\overline{RTS}$ High		60	ns	*5
T75	$\overline{TxC}$ Low to $\overline{CTS}$ Invalid	20		ns	
T75A	$\overline{CTS}$ High to $\overline{TxC}$ Low $\overline{CTS}$ Set-up Time to Stop Transmission	65		ns	*7
Interframe Spacing Parameters					
T76	Inter Frame Delay	*9			
$\overline{CRS}$, $\overline{CDT}$, Parameters					
T77	$\overline{CDT}$ Low to $\overline{TxC}$ High External Collision Detect Setup Time	45		ns	
T78	$\overline{TxC}$ High to $\overline{CDT}$ Inactive $\overline{CDT}$ Hold Time	20		ns	*14
T79	$\overline{CDT}$ Low to Jamming Start		*10		
T80	Jamming Period	*11			
T81	$\overline{CRS}$ Low to $\overline{TxC}$ High Carrier Sense Setup Time	45		ns	*14
T82	$\overline{TxC}$ High to $\overline{CRS}$ Inactive $\overline{CRS}$ Hold Time	20		ns	*14

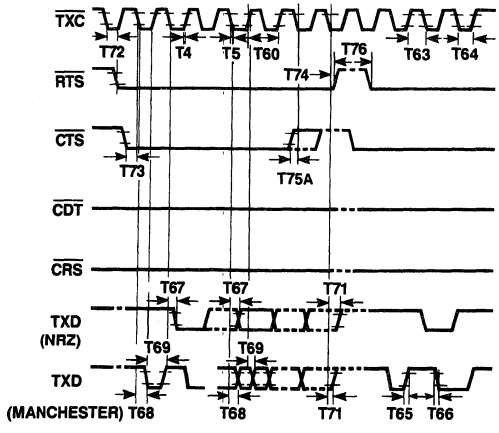
High Speed Mode (Continued)

Symbol	Parameter	Min	Max	Units	Test Conditions
$\overline{CRS}$, $\overline{CDT}$, Parameters (Continued)					
T83	$\overline{CRS}$ High to Jaming (Internal Collision Detect)		*12		
T84	$\overline{CRS}$ High to $\overline{RxC}$ High. End of Receive Packet	80		ns	
T85	$\overline{RxC}$ High to $\overline{CRS}$ High. End of Receive Packet.	20		ns	
Receive Clock Parameters					
T86	$\overline{RxC}$ Rise Time		10	ns	*1
T87	$\overline{RxC}$ Fall Time		10	ns	*1
T88	$\overline{RxC}$ High Time	80		ns	*1
T89	$\overline{RxC}$ Low Time	80		ns	*1
Received Data Parameters					
T90	RxD Setup Time	45		ns	*1
T91	RxD Hold Time	45		ns	*1
T92	RxD Rise Time		20	ns	*1
T93	RxD Fall Time		20	ns	*1

NOTES:
*1 — MOS levels.
*2 — Manchester only.
*3 — Manchester. Needs 50% duty cycle.
*4 — 1 TTL load + 50 pF.
*5 — 1 TTL load + 100 pF.
*6 — NRZ only.
*7 — Abnormal end to transmissions: $\overline{CTS}$ expires before $\overline{RTS}$.
*8 — Normal end to transmission.
*9 — Programmable value.
T76 = NIFS × T60 (ns)
NIFS - the IFS configuration value.
If NIFS is less than 12, then NIFS is enforced to 12.
*10 — Programmable value:
T79 = NCDF × T60 + (12 to 15) × T60 (ns) (if collision occurs after preamble).
*11 — T80 = 32 × T60
*12 — Programmable value:
NCSF × TTRC + (12 to 15) × TTRC
T83 = NCSF × T60 + (12 to 15) × T60
NCDF - collision detect filter configuration value.
*13 — 2000 ns if configured for Manchester encoding.
*14 — To guarantee recognition on the next clock.

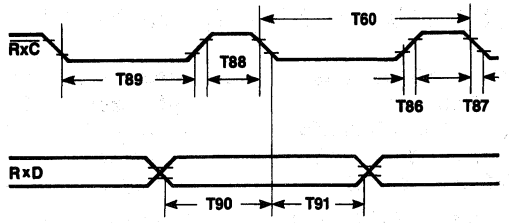


231161-26

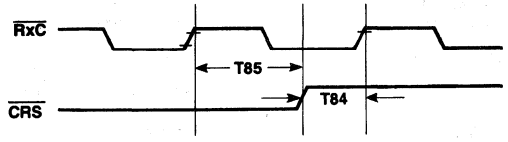


231161-27

Transmit Data Waveforms



Receive Data Waveforms (NRZ)

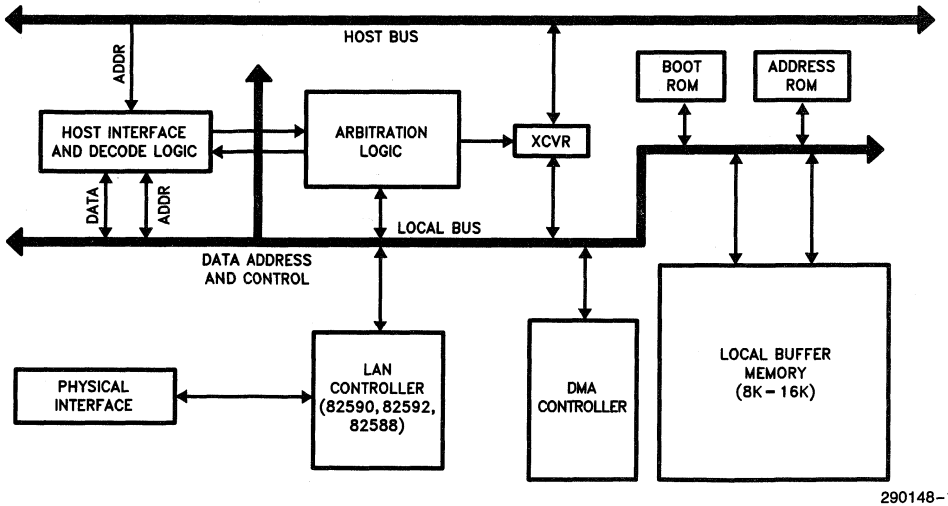


Receive Data Waveforms

82560
ARCHITECTURAL OVERVIEW
HOST INTERFACE AND MEMORY MANAGER

- **Host Interface to IBM PC/XT/AT Bus for 82590, 82592 and 82588 LAN Controllers. Allows the LAN Adapter's Local Memory to be Mapped to Host Memory or to Host I/O Address Space**
- **Tightly Coupled Interface to 82590 and 82592 LAN Controllers**
  - **— Automatic Retransmission upon Collision**
  - **— Automatic Buffer Reclamation Upon Receive Errors**
- **Transmit and Receive, Buffer Management Schemes Allow Continuous Back-to-Back Frame Reception and Transmit Chaining**
- **Integrated Peripheral Functions Optimized for Non-Intelligent, Buffered Adapter Architectures**
  - **— Interrupt Control Logic**
  - **— Programmable Wait State Generator**
- **Supports 8- or 16-Bit-Wide Static Memory, with Capability to Address up to 16K Bytes**
- **Advanced Programmable DMA Controller**
- **Multi-Port, Priority-Based Arbitration**
- **CHMOS III Technology for Increased Reliability and Reduced Power Consumption**
- **68-Lead PLCC Package**

The 82560 Host Interface and Memory Manager is a companion chip for Intel's Advanced 82590 and 82592 CSMA/CD LAN controllers, and for the 82588. The 82560 interfaces these controllers to the IBM PC/XT/AT bus, and provides on-chip DMA support for data transmission and reception. It integrates all the functions required to implement a non-intelligent, locally buffered LAN adapter for the PC/XT/AT environment. When used in the "Tightly Coupled Interface" with the 82590 or 82592, the 82560 performs retransmission and receive buffer reclamation without CPU intervention. The buffer management schemes implemented in the Tightly Coupled Interface allow transmit command chaining and continuous back-to-back frame reception. The 82560 can support 16K bytes of static buffer memory. The buffer memory can be mapped into either the host memory address space or the host I/O space—accessed sequentially through a single I/O port. The 82560 also includes a multi-port arbiter which arbitrates requests for access to devices residing on the adapter's local bus. When combined with the 82590 or 82592, the 82560 provides a low-cost, high-performance solution for LAN adapters in industry standard CSMA/CD networks.



290148-1

Figure 1. Non-Intelligent, Buffered Adapter Architecture

October 1987
Order Number: 290148-001

FUNCTIONAL OVERVIEW

The 82560 is a multi-port memory controller using interrupt logic and DMA to implement a non-intelligent, buffered LAN adapter for the IBM PC/XT/AT bus. This type of adapter uses on-board memory as a buffer to store frames during transmission and reception. It also uses on-board DMA to transfer data between its local memory and the LAN controller. A block diagram of the buffered, non-intelligent LAN adapter is shown in Figure 1. The architecture is termed non-intelligent because it does not use an on-board CPU to process the transmit or receive frames. The host CPU processes the frames and programs the DMA and LAN controllers. The host interface logic, arbitration logic, and the bus transceivers connect the host bus to the adapter's local bus. They also control all host accesses to the local bus. The local memory is shared by the host and the LAN controller. It stores information that the host wishes to transfer to the LAN controller, and information received by the LAN controller which should be read by the host. The memory can be shared in two ways—mapping into the host memory space, or mapping into the host I/O space. The DMA controller transfers data between the local buffer memory and the LAN controller. The host CPU may use either string move instructions or a system DMA channel to move data into the buffer memory. The host also accesses the LAN controller registers, the DMA controller registers, the boot ROM, and the address ROM through the local bus.

The 82560 integrates the host interface, arbitration logic and DMA into one component. It replaces 20–30 MSI and SSI components (see Figure 2). It also provides a Tightly Coupled Interface to the 82590 and 82592 LAN controllers, and a sophisticated buffer management scheme, which allows the 82590/82560 combination to handle time-critical processes such as retransmission, buffer reclamation, and continuous back-to-back frame reception without host CPU intervention. This improves the overall data throughput in the network; and, consequently, system performance. The following discussion describes the 82560 interface to the PC/XT/AT bus, its support of locally buffered memory, and the operation of DMA and the Tightly Coupled Interface (including the buffer management schemes they support).

HOST INTERFACE

The host interface port connects the 82560 to the PC-bus through external decode logic and bus transceivers. The external decode logic generates the $\overline{HF0}$ and $\overline{HF1}$ signals indicating the kind of access the host desires. When the request is detected by the 82560 it deasserts the HRDY signal, thereby suspending the host cycle. HRDY is reactivated when the local device being accessed by the host is ready to accept (Write cycle) or output (Read cycle) data. The request undergoes arbitration, and, if granted, the 82560 activates the $\overline{XCV0}$ and $\overline{XCV1}$ signals. The XCV signals control the transceiver(s) which interfaces the host data bus to the local data bus. By using one or both transceiver control signals the 82560 can support an 8- or 16-bit-wide bus. Once the arbiter grants the host request, the 82560 begins the local bus cycle by generating the appropriate address and control signals.

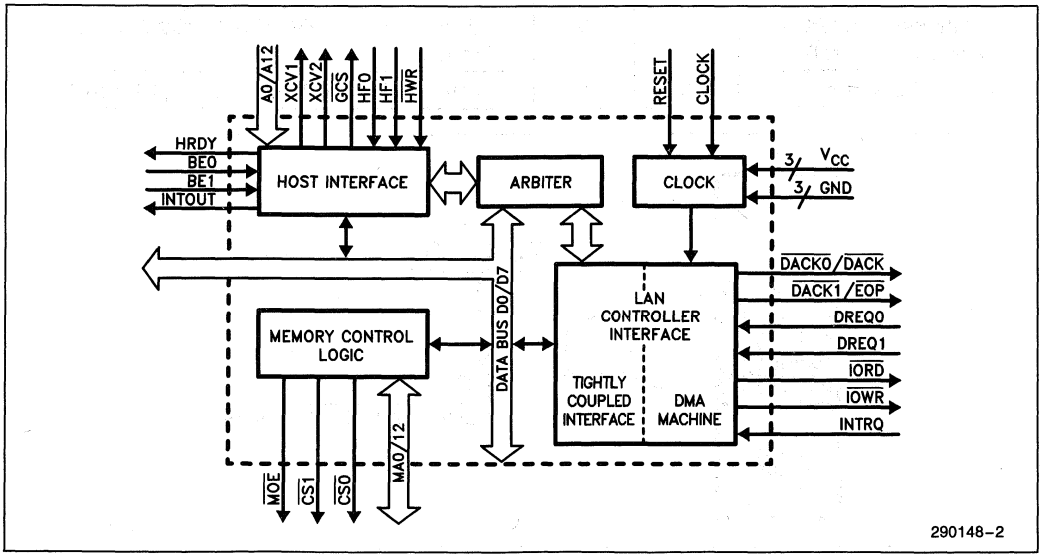


Figure 2. 82560 Block Diagram

The host CPU can access the internal registers of the 82560, the external memory controlled by the 82560, or other devices—such as Boot ROM or external Latch—which share the same bus as the 82560. Table 1 lists the various access types that can be requested by the host.

The 82560 also supports devices on the local bus other than memory and the LAN controller. These devices can be accessed in two ways: by using the General Chip Select ($\overline{GCS}$) signal, or by using the Special Chip Select (SCS) addresses in the 82560 register space. The first method typically supports EPROMs, external latches, and similar devices. The $\overline{GCS}$ signal is used as a qualifier by the external decode logic before it generates the Chip Select signal for the appropriate device. The second method is used for accessing the registers of controllers which use the 82560 DMA channels; e.g., the 82590, 82592, or 82588. Each address in the SCS port provides a unique combination of the $\overline{DACK0}$, $\overline{DACK1}$, and $\overline{CS}$-pin output states. The CPU activates the chip select of the device being accessed by asserting or deasserting the appropriate signals.

The host CPU can access the 82560 registers and other devices on the local bus at any time. However, local memory can only be accessed by the host after the 82560 memory control registers are initialized.

The host accesses local memory in two ways: Page Access or Sequential Access. On power-up, or after reset, the memory access mode is undefined, and host access to local memory is disabled. The 82560 must be configured for the appropriate memory access mode before local memory can be accessed by the host.

In Page Access mode the local memory is mapped into the host memory space. In this mode the host can directly access local memory through a fixed size window which can be moved around in local memory space. This window is referred to as a "page". Figure 3 shows the paging scheme. The page size can be up to 8K bytes, and can be located anywhere in local memory. The page size can be made smaller by grounding one or more of the address lines (A0–A12). The exact location of the page in the local memory is defined by a page register. By reprogramming the page register the user can relocate the page in local memory.

In Sequential Access mode the memory is mapped into the host I/O address space. Data is transferred between host and local memory using host DMA or string I/O instructions. The 82560 can be programmed to support memory accesses through a single I/O port. The I/O port is defined by an address programmed into an 82560 register. The

Table 1. Host Access Types

Access Type	HF0	HF1	Address	Cycle Status Indications
82560 Registers	0	1	10H	HRDY
Local Memory Access	1	0	User Defined	HRDY, Memory Control Signals, XCV Signals
GCS Access (Boot ROM) (General Chip Select)	0	0	User Defined	HRDY, GCS, IOWR/IORD
Special Chip Select	0	1	<10H	HRDY, DACK Lines, IOWR/IORD

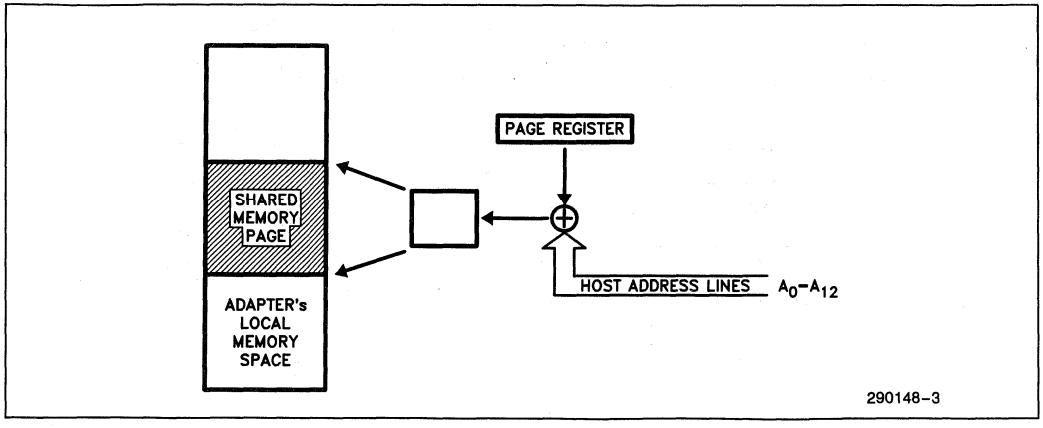


Figure 3. Page Mechanism

82560 maintains the current address, which is updated each time a memory cycle is run. The host does not directly access the local memory. It outputs the I/O address onto the A0–A12 address lines, with the HF lines indicating a memory access. If the I/O address matches the address programmed into the 82560, then the 82560 executes the local memory cycle by outputting the current address onto the memory address lines MA0–12.

ARBITER

All requests for access to devices on the local bus, whether by the host or by the 82560 DMA, undergo arbitration. The host requests are indicated on the HF lines; the DMA requests are indicated on the DRQ lines. Figure 4 shows the basic arbitration cycle of the 82560. Arbitration for the local bus is pipelined. It can take place at any time when the 82560 is idle, or one clock before the end of the current local bus cycle. All requests are latched in on the falling edge of the 82560 clock. Arbitration is completed within one clock cycle. The resultant local bus cycle is started on the next clock. If more than one request is active, arbitration is resolved on a rotating priority basis. Priority is given to the device which has control of the local bus.

The 82560 deactivates its HRDY line when a host request is detected; the request is then arbitrated. If the request is granted, the appropriate local bus cycle is begun. When the device being accessed is ready to accept or output data, the HRDY line is reactivated and the host CPU can then perform the transfer of data. The DMA requests are acknowledged once DMA has been granted access to the local bus. The acknowledge lines are kept inactive until the DMA is granted access to the local bus.

DMA MACHINE

The 82560 provides two DMA channels. Each channel can address up to 512K bytes, and has request and acknowledge lines and address registers. The DMA normally operates in the Demand mode, and becomes active in response to a DMA request being granted. The requests come in on the activating lines and, if granted, are acknowledged by the DACK lines. The DMA can be configured for the 82590, in which case it is tightly coupled to the 82590 parallel subsystem, using EOP and DRQ lines to determine the status of the current transmit or receive operation. $\overline{\text{DACK0}}$ becomes $\overline{\text{DACK}}$, the acknowledge for both channels; and $\overline{\text{DACK1}}$ becomes $\overline{\text{EOP}}$, an input. The DRQ and the $\overline{\text{EOP}}$ lines indicate the status of the frame transmission or reception. This allows the 82560 to handle certain time-critical events, during transmission or reception, without CPU intervention.

The DMA channels operate in flyby mode, with data being read from the source and written to the destination simultaneously. Data is transferred on a demand basis. The 82560 will continue to transfer data if a DRQ line is active (and no other device is requesting control of the local bus) and no End of Process ($\overline{\text{EOP}}$) indication is provided. The arbiter will allow back-to-back DMA cycles if no other requests are active; however, normally DMA cycles will be interleaved with other local bus cycles.

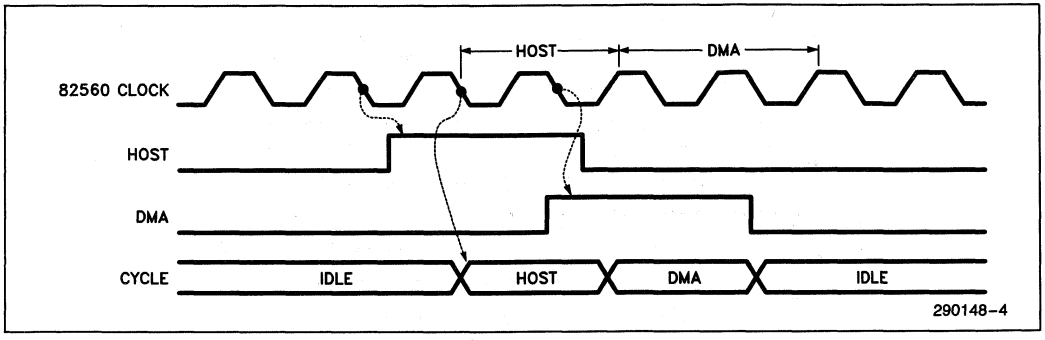


Figure 4. 82560 Arbitration Cycle

Each DMA channel has a Control, Current Address, Base Address, and a Stop register. The Current Address register maintains the address which will be accessed during the next DMA cycle. It is automatically incremented when a DMA cycle is completed successfully. The Base Address register points to the beginning of the memory structure used to store a frame. It is used to reload the Current Address register when a collision or receive error occurs while in the Tightly Coupled Interface mode. The Stop register points to the location following the last byte of the frame. When the Current Address Register becomes equal to the Stop register, the DMA is disabled and the 82560 generates an interrupt to the CPU.

The DACK lines can also be asserted or deasserted under CPU control by accessing Special Chip Select (SCS) addresses in the 82560 register space. It provides the host with the option of accessing the LAN controller port(s) through the 82560.

TIGHTLY COUPLED INTERFACE

The Tightly Coupled Interface is a hardware interface between the 82560 and the 82590. This interface allows most transmission and reception events to be processed without CPU intervention. It allows the implementation of such time-critical CSMA/CD processes as automatic retransmission, buffer reclamation, and continuous frame reception. The basic interface is a two-wire DMA handshake between the 82560 and the 82590, this occurs over the DRQ and $\overline{EOP}$ pins. The 82590 provides the status of the current transmit or receive process, or requests another DMA cycle at the end of each DMA cycle. When configured for the Tightly Coupled Interface, the 82560 and the 82590 use a specific interrupt scheme to minimize CPU overhead and to improve data throughput. The 82590 will not generate interrupts upon events that can be handled by the 82560 without CPU intervention. Figure 5 illustrates the Tightly Coupled Interface mechanism. Table 2 lists the various combinations of the DRQ and EOP signals, and the events they represent.

Table 2. DMA Handshake Encoding

DRQ	$\overline{EOP}$	Event Status
0	0	Frame Transmitted or Received OK
0	1	Idle
1	0	Frame Transmission or Reception Aborted
1	1	Another DMA Transfer Requested

If both DRQ and $\overline{EOP}$ are sampled high, the Current Address Register of the channel is incremented and the DMA cycle is repeated. If a frame is transmitted or received without errors, both DRQ and $\overline{EOP}$ are low at the end of the DMA cycle and the 82560 will generate an interrupt. If DRQ is high and $\overline{EOP}$ is low, a collision occurred during transmission, or an error occurred during reception. In this case the Current Address Register will be reloaded with the value in the Base Address Register; and, once again, it will point to the beginning of the frame structure in memory.

The $\overline{DACK1}/\overline{CS1}/\overline{EOP}$ pin of the 82590 is multiplexed and requires external logic to derive the $\overline{EOP}$ and $\overline{CS1}$ signals (see 82590 data sheet). Because the 82560 integrates this logic, its $\overline{DACK1}/\overline{EOP}$ pin can be connected directly to the $\overline{DACK1}/\overline{CS1}/\overline{EOP}$ pin of the 82590, and its $\overline{DACK0}/\overline{DACK}$ pin can be connected directly to the $\overline{DACK}$ pin of the 82590.

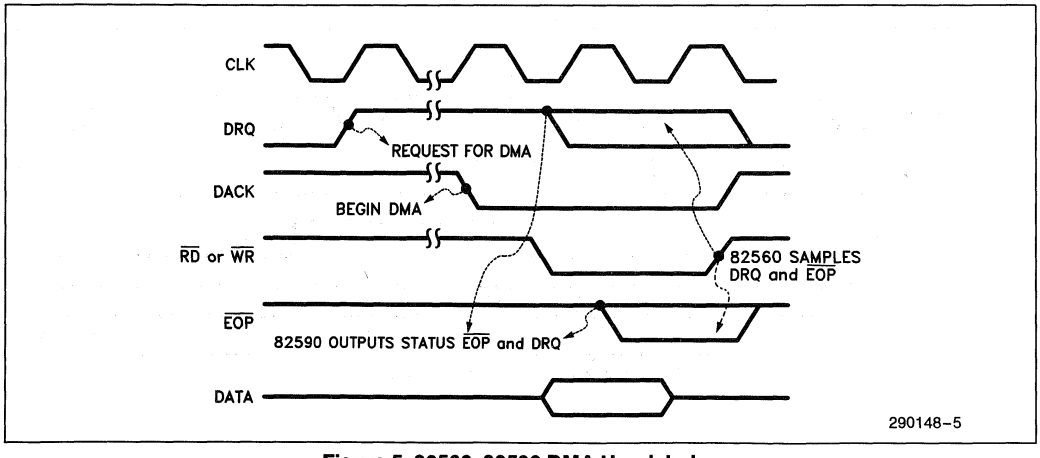


Figure 5. 82560, 82590 DMA Handshake

TRANSMIT COMMAND CHAINING

The 82590 can transmit consecutive frames without requiring the CPU to issue the Transmit command each time. This improves data throughput during transmission and eliminates CPU overhead. The CPU can place multiple transmit frames in memory, with each frame separated from the next by a Command byte. The Command byte immediately follows the last byte of the transmit frame. It must contain one of two commands: Transmit (04H) or NOP (00H). During the DMA cycle the 82590 loads the Command byte which follows the DMA of the last byte of the frame. In the same DMA cycle the 82590 indicates Transmit Complete on the DRQ and $\overline{EOP}$ lines. The Transmit Complete signal causes the 82560 to load the value of the Current Address register into the Base Address register, so the Base Address register now points to a location immediately preceding the beginning of the next frame. If the Command byte contains the Transmit command, the 82590 will behave as if another Transmit command has been issued by the CPU, and will repeat the frame transmission procedure. A NOP in the Command byte indicates the preceding frame was the last frame to be transmitted and no other transmit frames follow. Figure 6 illustrates the Transmit Chaining process.

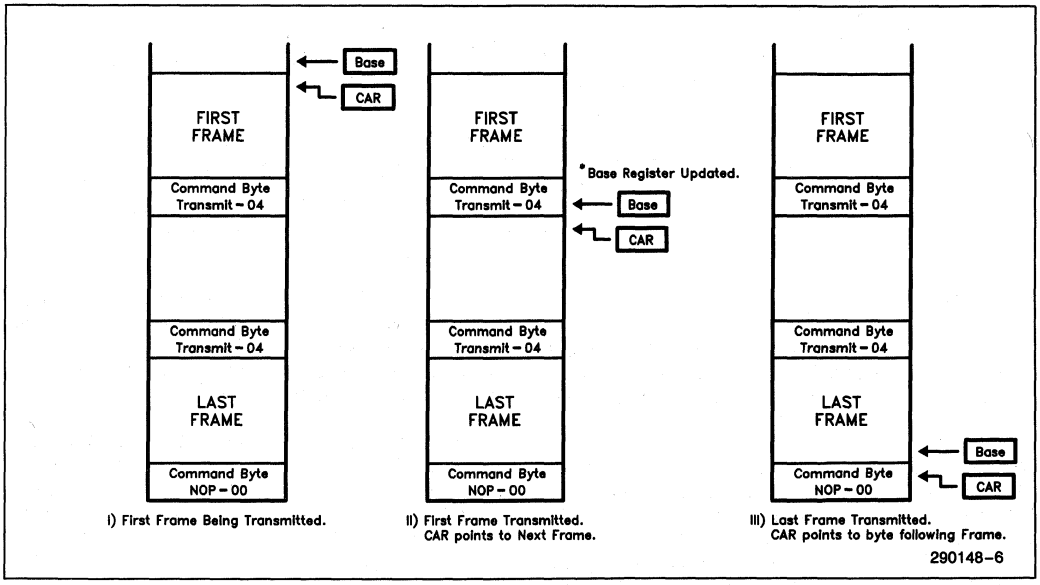


Figure 6. Transmit Chaining Process

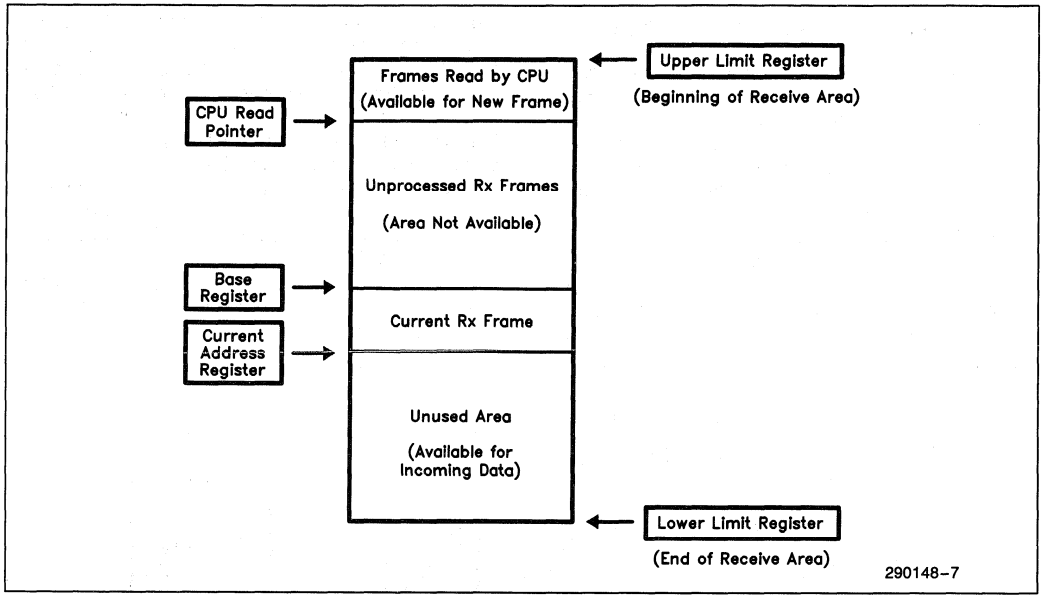


Figure 7. Ring Buffer Implementation

BUFFER MANAGEMENT

The 82560 arranges a portion of the local memory into a ring buffer with consecutive frames received in adjacent portions of memory. Figure 7 shows the implementation of this scheme.

The beginning and end of the receive area is user definable through Lower and Upper Limit registers. The incoming frames are deposited in this region in adjacent locations in the Receive area. When the DMA reaches the end of the Receive area it will go back to the beginning and start depositing the incoming bytes—if the data from those locations has been read by the CPU. Once the frame is read by the CPU, the memory it had occupied becomes available for other incoming frames. The CPU Read pointer prevents overwriting of frames that have not been read. It points to the location in the receive area which was last read by the CPU. The 82560 will not deposit received data into locations following the location pointed to by the CPU Read pointer. The Base Address register points to the beginning of the current receive structure, which is 3 bytes from the beginning of the actual frame. It is updated each time the 82590 issues the Frame Received OK on the DMA handshake. The Current Address register points to the location where the next incoming byte/word will be deposited.

While receiving in the Tightly Coupled Interface mode, the 82560 stores two bytes of data internally. When the 82590 issues a Frame Received OK signal on the DMA handshake, the 82560 will write the last two bytes, internally stored in temporary registers, to the beginning of the frame. This allows the CPU to process the frame with a minimum of overhead. Figure 8 shows the frame structure used in this implementation.

APPLICATIONS

Figure 9 shows a buffered, non-intelligent StarLAN adapter for the IBM PC bus (using the 82560 and the 82590). Figure 10 shows a buffered, non-intelligent Ethernet adapter for the IBM PC bus (using the 82560, 82592, 82C501, and 82502).

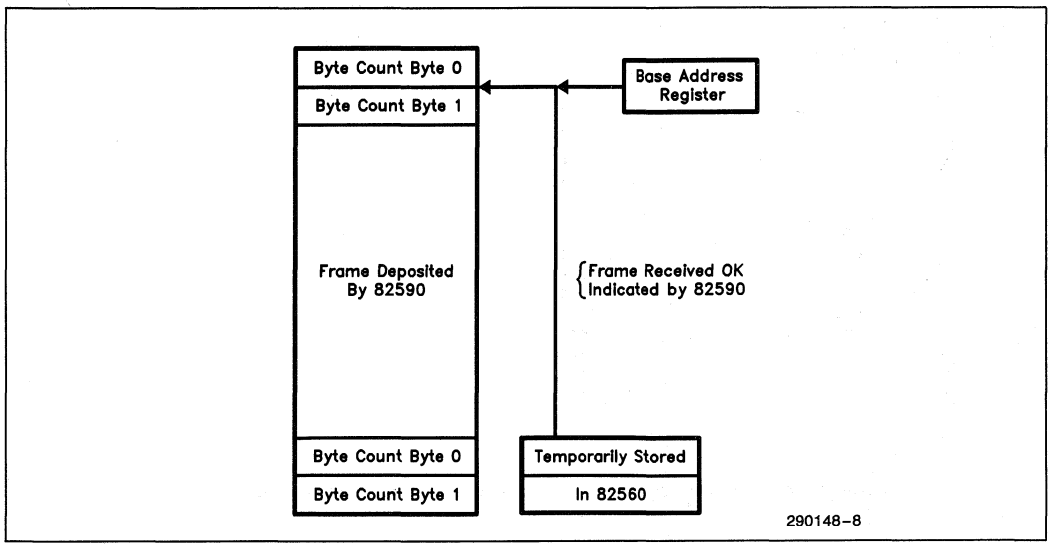


Figure 8. Ring Buffer Frame Structure

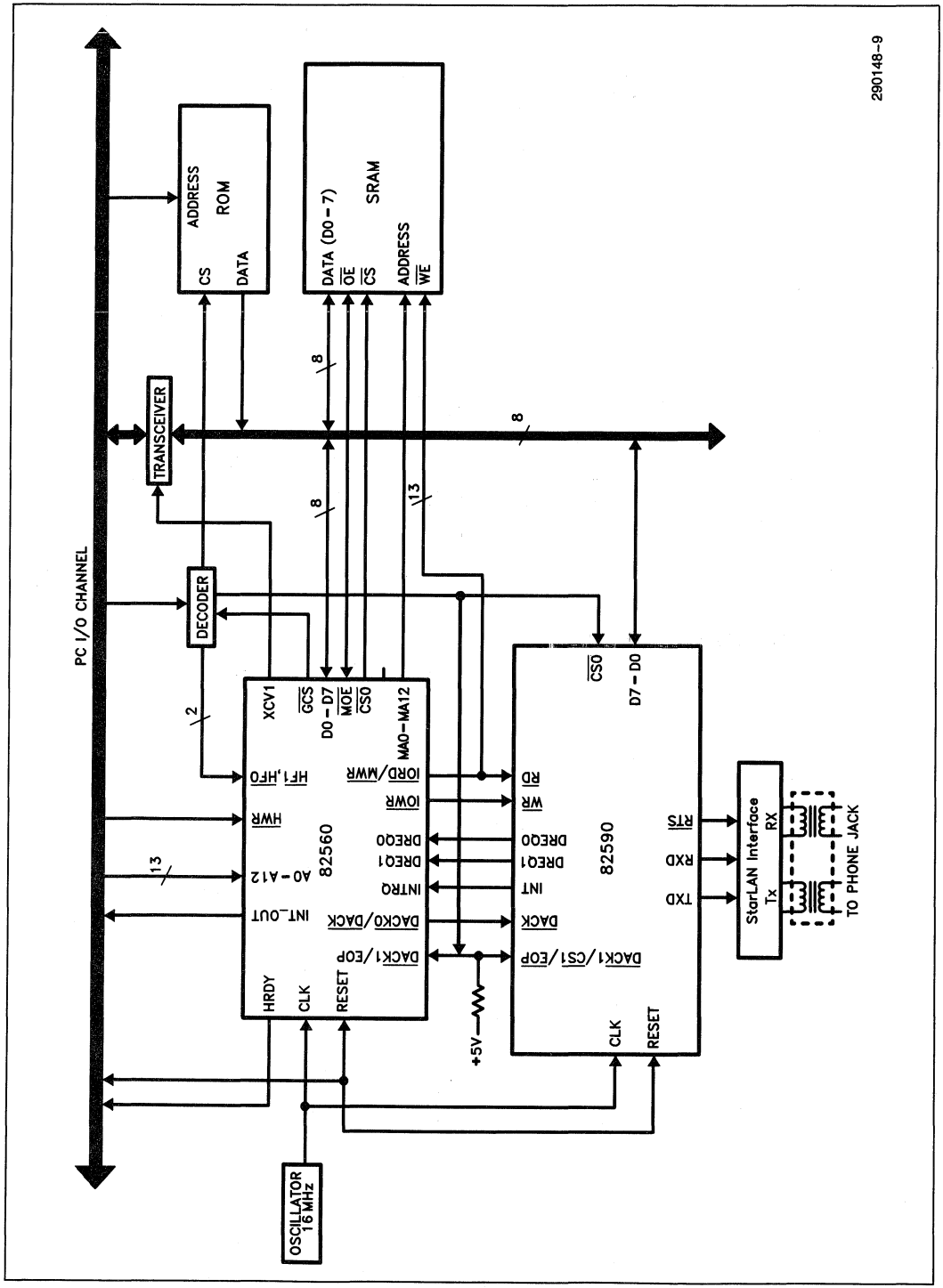


Figure 9. StarLAN Adapter (560, 590)

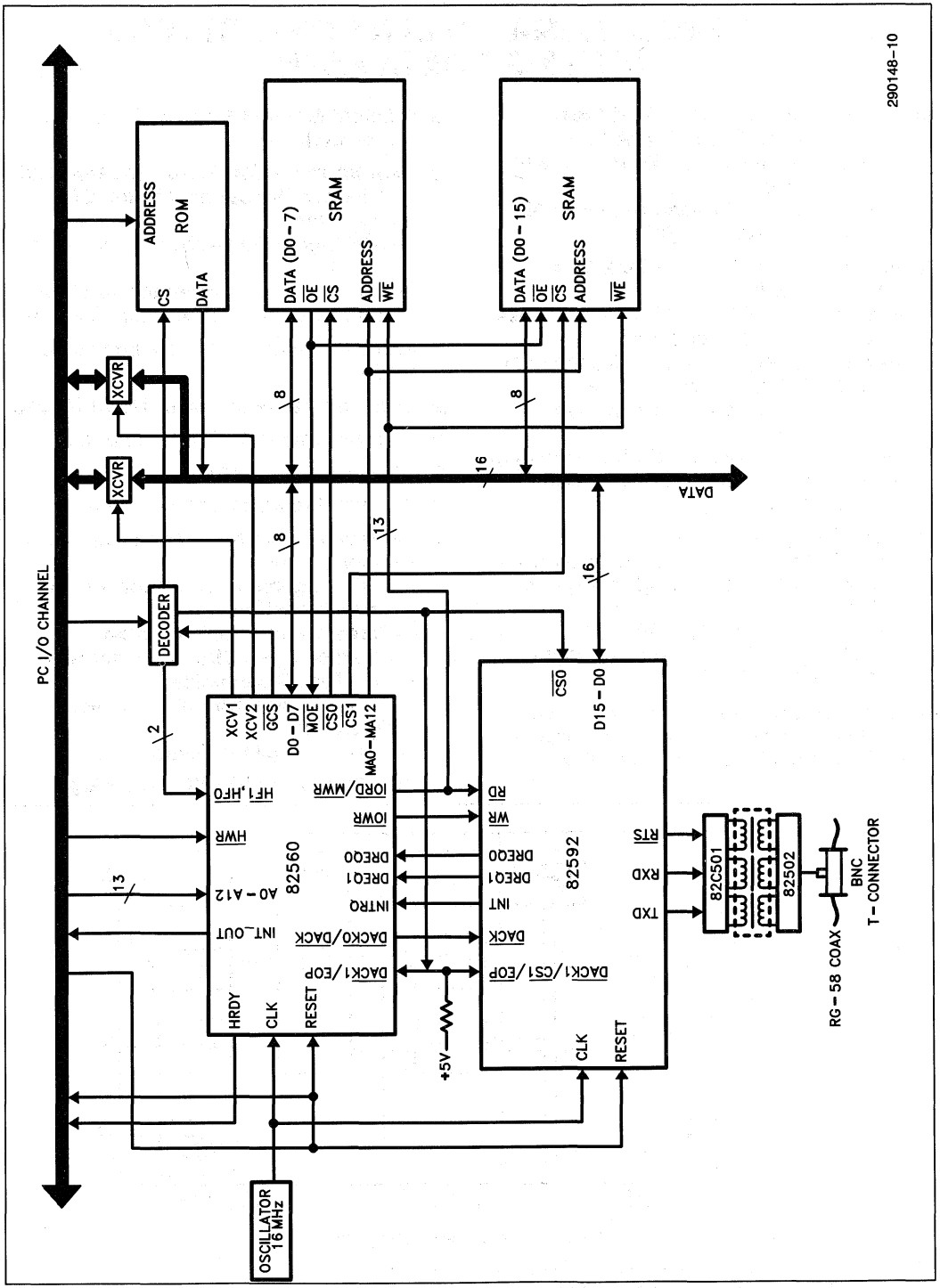
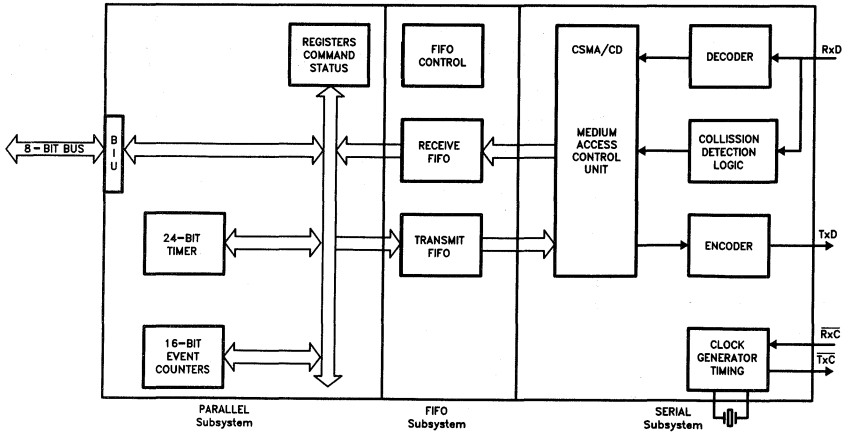


Figure 10. Ethernet Adapter (82560, 82592, 82C501, 82502)

82590
ADVANCED CSMA/CD LAN CONTROLLER WITH 8-BIT DATA PATH

- Supports Industry Standard LANs
  - StarLAN (IEEE 802.3 1BASE5)
  - IBM™ PC Network—Baseband and Broadband
  - Ethernet and Cheapernet (IEEE 802.3 10BASE5 and 10BASE2)
- Integrates Physical and Data Link Layers of OSI Model
  - Complete CSMA/CD Medium Access Control (MAC) Functions
  - Manchester, Differential Manchester, or NRZI Encoding/Decoding
  - On-Chip, Logic-Based Collision Detection
  - IEEE 802.3 or HDLC Frame Delimiting
  - Broadband Ethernet (IEEE 802.3 10BROAD36)
- Two Modes of Operation
  - Bit Rates up to 4 Mb/s with On-Chip Encoder/Decoder (High-Integration Mode)
  - Bit Rates up to 20 Mb/s with External Encoder/Decoder (High-Speed Mode)
- High-Performance System Interface
  - 16-MHz Clock, 2 Clocks per Transfer
  - 64 Bytes of Configurable FIFO
- Efficient Memory Use via Buffer and Frame Chaining
- DMA Interface for Retransmission and Continuous Reception without CPU Intervention
  - $\overline{\text{EOP}}$ Signal Generation for 8237 and 82380
  - Tightly Coupled Interface to 82560 Host Interface and Memory Manager
- 82588 Pin- and Software-Compatible Mode
- Local and Remote Power-Down Modes
- Deterministic Collision Resolution
- 24-Bit General Purpose Timer
- On-Chip Jabber Inhibit Function
- Network Management and Diagnostics
  - Monitor Mode
  - CRC, Alignment, and Short Frame Error Detection
  - Three 16-Bit Event Counters
  - Short or Open Circuit Localization
  - Self-Test Diagnostics
  - Internal and External Loopback Operation
  - Internal Register Dump
- High-Speed CHMOS III Technology



290147-1

Figure 1. 82590 Block Diagram

*IBM, PC, PCAT, PCXT are trademarks of International Business Machines.

February 1988
Order Number: 290147-002

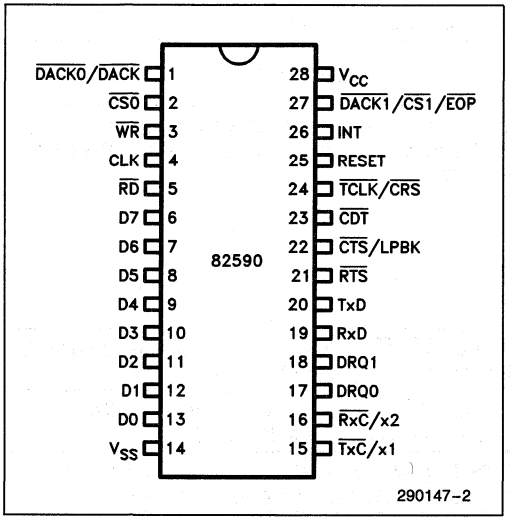


Figure 2. 82590 Pin Configuration (DIP)

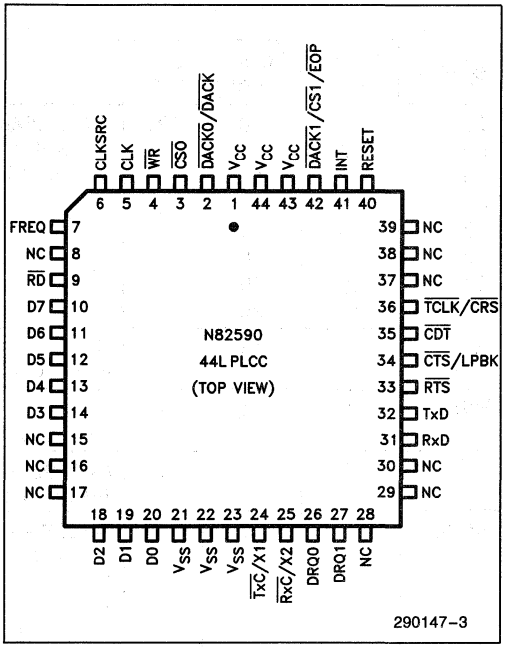


Figure 3. 82590 Pin Configuration (PLCC)

The 82590 is a second-generation, 8-bit data path CSMA/CD controller. Its system interface enables efficient operation with a wide variety of Intel microprocessors (such as iAPX 188, 186, 286, or 386) and industry standard buses (such as the IBM PC I/O channel or Personal System/2™ Micro Channel™). The 82590 can be configured to support a wide variety of industry standard networks, including StarLAN, IBM PC Network, and Ethernet/Cheapernet.

The 82590 provides a natural upgrade path for existing 82588 applications, since it is pin and software compatible with its predecessor. Its rich incremental functionality compared to the 82588 can be utilized by selectively modifying existing software drivers.

Together with the 82560 (Host Interface and Memory Manager) the 82590 offers a complete solution for CSMA/CD LAN adapters oriented to the IBM PC environment. The 82590 fully conforms to existing IEEE 802.3 standards (1BASE5, 10BASE5, 10BASE2, and 10BROAD36). To implement a complete Ethernet/Cheapernet solution, Intel offers the 82C501 Ethernet Serial Interface and the 82502 Ethernet Transceiver chip. Intel also offers the 82592, a 16-bit data path version of the 82590, for higher performance applications.

The 82590 is available in a 28-pin Plastic DIP or a 44-pin PLCC package. It is fabricated with Intel's reliable CHMOS III technology.

Table 1. 82590 Pin Description

Symbol	Pin No. (DIP)	Pin No. (PLCC)	Type	Name and Function
D7 D6 D5 D4 D3 D2 D1 D0	6 7 8 9 10 11 12 13	10 11 12 13 14 18 19 20	I/O	**DATA BUS**—The Data Bus lines are bidirectional, three-state lines connected to the CPU's Data Bus for transfers of data, commands, status, and parameters.
$\overline{RD}$	5	9	I	**READ**—Together with $\overline{CS0}$, $\overline{CS1}$, $\overline{DACK0}$, or $\overline{DACK1}$, Read controls data or status transfers out of the 82590.
$\overline{WR}$	3	4	I	**WRITE**—Together with $\overline{CS0}$, $\overline{CS1}$, $\overline{DACK0}$, or $\overline{DACK1}$, Write controls data or command transfers into the 82590.
$\overline{CS0}$	2	3	I	**CHIP SELECT (PORT 0)**—When LOW, the 82590 is selected by the CPU for command or status transfer through PORT 0.
RESET	25	40	I	**RESET**—A HIGH signal on this pin causes the 82590 to terminate current activity. This signal is internally synchronized and must be held HIGH for at least four Clock (CLK) cycles. When the Clock signal is provided internally (CLKSRC is strapped HIGH), the RESET signal must be held HIGH for at least 50 μs. (PLCC version only.)
INT	26	41	O	**INTERRUPT**—A HIGH signal on this pin notifies the CPU that the 82590 is requesting an interrupt.
DRQ0	17	26	O	**DMA REQUEST (CHANNEL 0)**—This pin is used by the 82590 to request DMA transfer. DRQ0 remains HIGH as long as the 82590 requires DMA transfers. Burst transfers are thus possible. When the 82590 is programmed for Tightly Coupled Interface, the 82590 notifies the DMA controller of the status of transmission or reception, using this pin together with $\overline{EOP}$.
DRQ1	18	27	O	**DMA REQUEST (CHANNEL 1)**—This pin is used by the 82590 to request DMA transfer. DRQ1 remains HIGH as long as the 82590 requires DMA transfers. Burst transfers are thus possible. When the 82590 is programmed for Tightly Coupled Interface, the 82590 notifies the DMA controller of the status of transmission or reception, using this pin together with $\overline{EOP}$.
$\overline{DACK0}$ $\overline{DACK}$	1	2	I	**DMA ACKNOWLEDGE (CHANNEL 0)**—When LOW, this input signal from the DMA controller notifies the 82590 that the requested DMA cycle is in progress. This signal acts similarly to Chip Select for data and parameter transfers, using DMA channel 0. **DMA ACKNOWLEDGE (CHANNELS 0 AND 1)**—When the $\overline{DACK1}/\overline{CS1}/\overline{EOP}$ pin is programmed to $\overline{CS1}/\overline{EOP}$, this pin provides a DMA acknowledge for both channels 0 and 1. Two DMA acknowledge signals from the DMA controller, $\overline{DACK0}$ and $\overline{DACK1}$, must be externally ANDed in this mode of operation.

Table 1. 82590 Pin Description (Continued)

<table>
<tr><th>Symbol</th><th>Pin No. (DIP)</th><th>Pin No. (PLCC)</th><th>Type</th><th>Name and Function</th></tr>
<tr><td>$\overline{\text{DACK1}}$
$\overline{\text{CS1}}/\overline{\text{EOP}}$</td><td>27</td><td>42</td><td>

I

I/O</td><td>This is a multifunction, bidirectional pin which can be programmed to $\overline{\text{DACK1}}$ or $\overline{\text{CS1}}/\overline{\text{EOP}}$ during configuration. When it is configured for $\overline{\text{EOP}}$, it provides an open-drain output.
DMA ACKNOWLEDGE (CHANNEL 1)—When LOW, this input signal from the DMA controller notifies the 82590 that the requested DMA cycle is in progress. This signal acts similarly to Chip Select for data and parameter transfers, using DMA channel 1.
CHIP SELECT (PORT 1)—When LOW, the 82590 is selected by the CPU for command or status transfer through PORT 1.
END OF PROCESS—A LOW output signal requests the DMA controller to terminate the active DMA service.</td></tr>
<tr><td>CLK</td><td>4</td><td>5</td><td>I

I/O</td><td>CLOCK—In the 28-pin DIP, this is only an input pin. A TTL-compatible clock input to this pin provides the timing for the 82590 parallel subsystem.
In the 44-pin PLCC, this pin can be a clock input or output, depending on the state of CLKSRC. If CLKSRC is strapped LOW, this pin is a clock input which provides timing for the 82590 parallel subsystem. If CLKSRC is strapped HIGH, the clock for the 82590 parallel subsystem is generated from the internal clock generator. The CLK pin is then a clock output and provides a clock signal whose frequency can be one-half of or identical to, the frequency of the internally generated parallel subsystem clock, depending on the state of FREQ. Note that the maximum frequency of the clock signal supplied by the CLK pin is 8 MHz.
<table>
<tr><th rowspan="2">CLKSRC</th><th rowspan="2">FREQ</th><th colspan="2">CLK</th><th rowspan="2">Clock for the Parallel Subsystem</th></tr>
<tr><th>Type</th><th>Signal</th></tr>
<tr><td>0 (LOW)</td><td>Don't Care</td><td>I</td><td>Clock</td><td>Clock as Provided on the CLK Pin</td></tr>
<tr><td>1 (HIGH)</td><td>1</td><td>O</td><td>Internal Parallel Subsystem Clock Divided by Two</td><td>Prescaled Clock Generated from the Internal Clock Generator</td></tr>
<tr><td>1</td><td>0</td><td>O</td><td>Internal Parallel Subsystem Clock</td><td>Prescaled Clock Generated from the Internal Clock Generator</td></tr>
</table></td></tr>
<tr><td>CLKSRC</td><td>NA</td><td>6</td><td>I</td><td>CLOCK SOURCE—When strapped LOW, a clock signal on the CLK pin provides timing for the parallel subsystem. When strapped HIGH, timing for the parallel subsystem is internally generated from the clock generator provided in the serial subsystem. The internal prescaler is programmed during configuration to determine the frequency of the clock for the parallel subsystem.</td></tr>
<tr><td>FREQ</td><td>NA</td><td>7</td><td>I</td><td>FREQUENCY—When strapped LOW, CLK has an output frequency equal to that of the internal parallel subsystem clock. When strapped HIGH, CLK has an output frequency one-half that of the internal parallel subsystem clock. The state of this pin is relevant only when CLKSRC is strapped HIGH.</td></tr>
</table>

Table 1. 82590 Pin Description (Continued)

Symbol	Pin No. (DIP)	Pin No. (PLCC)	Type	Name and Function
X1/X2	15/16	24/25	I	**High Integration Mode** **OSCILLATOR INPUTS**—These inputs may be used to connect a quartz crystal which controls the internal clock generator for the serial subsystem. When CLKSRC is strapped HIGH, the clock generator also provides a clock for the parallel subsystem. X1 may also be driven by a MOS-level clock whose frequency is 8, 10, 16, or 18 times the bit rate of Transmit/Receive data. X2 must be left floating if X1 is connected to an external MOS clock.
$\overline{TxC}$	15	24	I	**High Speed Mode** **TRANSMIT CLOCK**—This signal provides the fundamental timing for the serial subsystem. The clock is also used to transmit data synchronously on the TxD pin. For NRZ encoding, data is transferred to the TxD pin on the HIGH to LOW clock transition. For Manchester encoding, the transmitted bit center is aligned with the LOW to HIGH transition.
$\overline{RxC}$	16	25	I	**RECEIVE CLOCK**—This clock is used to synchronously sample data on the RxD pin. Only NRZ data format is supported for reception. The state of the RxD pin is sampled on the HIGH to LOW transition.
$\overline{TCLK}/\overline{CRS}$	24	36	I O	**CARRIER SENSE**—In High Speed Mode this pin is Carrier Sense, $\overline{CRS}$, and is used to notify the 82590 that the serial link is active **TRANSMIT CLOCK**—In High Integration Mode this pin is Transmit Clock, $\overline{TCLK}$.
$\overline{CDT}$	23	35	I	**COLLISION DETECT**—This input notifies the 82590 that a collision has occurred. In High Speed Mode a collision is sensed by this pin only when the 82590 is configured for external Collision Detect (external means are then required for collision detection). In High Integration Mode collisions are sensed by this pin regardless of the internal or external Collision Detect configuration of the 82590.
RxD	19	31	I	**RECEIVE DATA**—This pin receives serial data. It must be HIGH when not receiving.
TxD	20	32	O	**TRANSMIT DATA**—This pin transmits data to the serial link. It is HIGH when not transmitting.
$\overline{RTS}$	21	33	O	**REQUEST TO SEND**—When this signal is LOW the 82590 notifies the channel that it has data to transmit. It is forced HIGH after a reset or when transmission is stopped.
$\overline{CTS}$/LPBK	22	34	I/O	**CLEAR TO SEND**—An active LOW signal which enables the 82590 to start transmitting data. Asserting this signal HIGH stops the transmission. **LOOPBACK**—This pin, in conjunction with a pull-down resistor, can be programmed to provide an active HIGH loopback signal to the external interface device.
V_{CC}	28	1 43 44		**POWER:** +5V ±10%
V_{SS}	14	21 22 23		**GROUND:** 0V

FUNCTIONAL DESCRIPTION

Internal Architecture

The 82590 consists of a parallel subsystem, a serial subsystem, and a FIFO subsystem (see Figure 1).

Parallel Subsystem

The parallel subsystem consists of a bus interface unit (BIU), command and status registers, a 24-bit general purpose timer, and three 16-bit event counters.

The BIU provides an 8-bit data bus interface to the external system bus. It handles all data transfers to and from memory (at speeds up to 8 Mbytes/sec.), accepts commands from the CPU, and provides status to the CPU. There are two separate I/O ports, Port 0 and Port 1; and two separate DMA channels, Channel 0 and Channel 1. Port 0 is the 82588-compatible I/O port through which the CPU issues commands such as Transmit and Receive Enable. The 82590's enhanced features, such as the general purpose timer and event counters, are accessed through Port 1. The two DMA channels are independent of each other and can be used for high-performance operations such as simultaneous transmission and reception.

The 24-bit timer consists of a 24-bit maximum count register, a 24-bit count register, and associated control bits in the command registers. Its clock source can be the transmit clock or the parallel subsystem clock. The timer can be programmed to halt or continue on a terminal count with or without causing an interrupt.

The three 16-bit event counters can be programmed to count valid frames, collided frames, and errored (CRC or Alignment) frames. When these event counters are used in Monitor mode, the 82590 is capable of maintaining the network statistics by itself; i.e., without requesting DMA services or causing interrupts to the CPU.

Serial Subsystem

The serial subsystem consists of a CSMA/CD unit, a data encoder and decoder, collision detect and carrier sense logic, and a clock generator.

The 82590's CSMA/CD unit is highly flexible in implementing the CSMA/CD protocol. It can operate in a variety of IEEE 802.3 and other CSMA/CD LAN environments, including 1BASE5 (StarLAN, 10BASE5 (Ethernet), 10BASE2 (Cheapernet), and the IBM™ PC Network (Baseband and Broadband). The programmable parameters include:

- Framing (IEEE 802.3 Framing or HDLC Framing)
- Address Field Length
- Station Priority
- Interframe Spacing
- Slot Time
- CRC-32 or CRC-16

The CSMA/CD unit also has a mode of operation which implements deterministic collision resolution (DCR). The DCR algorithm is fully compatible with the MULTIBUS™ II Serial System Bus (SSB) specifications.

The encoder and decoder in the serial subsystem is capable of NRZI, Manchester, and Differential Manchester encoding and decoding at bit rates up to 4 Mb/s in High-Integration Mode, and Manchester encoding at bit rates up to 20 Mb/s in High-Speed Mode. A digital phase-lock loop is used in High-Integration Mode to decode the receive data and to generate the synchronous receive clock.

The collision detect and carrier sense logic generate the internal collision detect and carrier sense signals for the CSMA/CD unit.

The 82590 implements several different internal, logic-based collision detect mechanisms. Two of these, Code Violation and Bit Comparison, are also available with the 82588. The Code Violation method defines a collision where a transition edge occurs outside the area of normal transitions (as specified by the data encoding method). For example, if there are no mid-bit cell transitions in the Manchester encoded data, this method interprets that condition as a collision. The Bit Comparison method compares the signature of the transmitted frame to the signature of the received frame. If the signatures are different, a collision is assumed to have occurred. Two other internal collision detect methods implemented in the 82590 are Source Address Comparison and StarLAN CPS (Collision Presence Signal) Recognition. The Source Address Comparison compares the source address field of the transmitted frame to the source address field of the received frame. If the source addresses are different, it assumes that a collision has occurred resulting in data corruption in the source address field. The StarLAN CPS Recognition method looks for the specific collision presence signal defined by the IEEE 802.3 1BASE5 standard. Other abnormal circumstances, such as no carrier for more than one-half slot time in the receive channel during transmission, are interpreted as collisions by the 82590.

In addition to these internal, logic-based collision detection methods, an external means of collision detection can be used in parallel by using the $\overline{CDT}$ input pin.

The clock generator in the serial subsystem is available only in High-Integration Mode and provides timing for the serial subsystem. The clock signal can also be routed to the parallel subsystem, if so desired. The oscillator circuit is designed for use with an external, parallel resonant, fundamental mode crystal. The crystal frequency should be selected at 8×, 10×, 16×, or 18× the required serial bit rate.

FIFO Subsystem

The FIFO subsystem is located between the parallel subsystem and the serial subsystem. It consists of a transmit FIFO, a receive FIFO, and FIFO control logic. The transmit and receive FIFOs are independent of each other and individually provide optimal interfaces between the two subsystems which may have different speeds. There is a total of 64 bytes that can be used for the two separate FIFOs. During configuration these 64 bytes can be divided into one of four possible combinations: 16 and 16 bytes, 16 and 48 bytes, 32 and 32 bytes, or 48 and 16 bytes for the transmit and receive FIFO respectively. The FIFO threshold is also programmed during configuration.

PROGRAMMING MODEL—REGISTER OVERVIEW

Figure 4 shows the 82590 internal registers that are directly accessible through the 8-bit I/O ports: Port 0 and Port 1. The registers enclosed in darker lines are 82588-compatible registers and are accessible only through Port 0.

Figure 5 shows the Port 0 commands. All of the Port 0 commands are compatible with the 82588 except for the NOP command with the channel bit set to 1. If the NOP command is executed with the channel bit set to 1, the active port is switched to Port 1. Port 0, which is selected by $\overline{CS0}$ in hardware, logically becomes Port 1. When the hardware does not support the second chip select, $\overline{CS1}$, this software port switch command is used. Figure 6 shows the Port 1 commands. When the SWT-TO-PORT-0 command is executed, the active port is switched back to Port 0.

The 82590 can be configured to have 4 or 6 bytes of status registers in Port 0 (see Figures 4 and 7). When configured to 4 bytes of status registers, formats of these registers are identical to those of the 82588. The first three status registers (STATUS 0 through 2) contain the information about the last command executed or the last frame received. The last status register, STATUS 3, contains the state of the 82590. When the 82590 is configured to 6 bytes of status registers, the two additional bytes are used to report a more complete status of the most recently received frame.

Status of the timer and event counters is available in the Port 1 status registers as shown in Figure 8.

82590 AND HOST INTERACTION

The CPU interacts with the 82590 through the system's memory and the 82590's on-chip registers. The CPU creates a data structure in memory, programs the external DMA controller with the start address and byte count of the memory block, and issues a command to the 82590.

The chip select and interrupt lines are used to communicate between the 82590 and the CPU as shown in Figure 9. The interrupt signal is used by the 82590 to attract the CPU's attention. The chip select signal is used by the CPU to attract the 82590's attention. Note that the 82590 does not have any address lines.

There are two kinds of transfers over the bus: command/status and data transfers. The command/status transfers are always performed by the CPU. The data transfers are requested by the 82590, and are usually performed by a DMA controller. Table 2 shows the command/status and data transfer control signals. The CPU writes commands to the 82590 using the $\overline{CS0}$ (or $\overline{CS1}$) and $\overline{WR}$ signals, and reads status using the $\overline{CS0}$ (or $\overline{CS1}$) and $\overline{RD}$ signals. When data transfers are performed, $\overline{DACK0}$ or $\overline{DACK1}$ must be asserted by the DMA controller instead of the Chip Select.

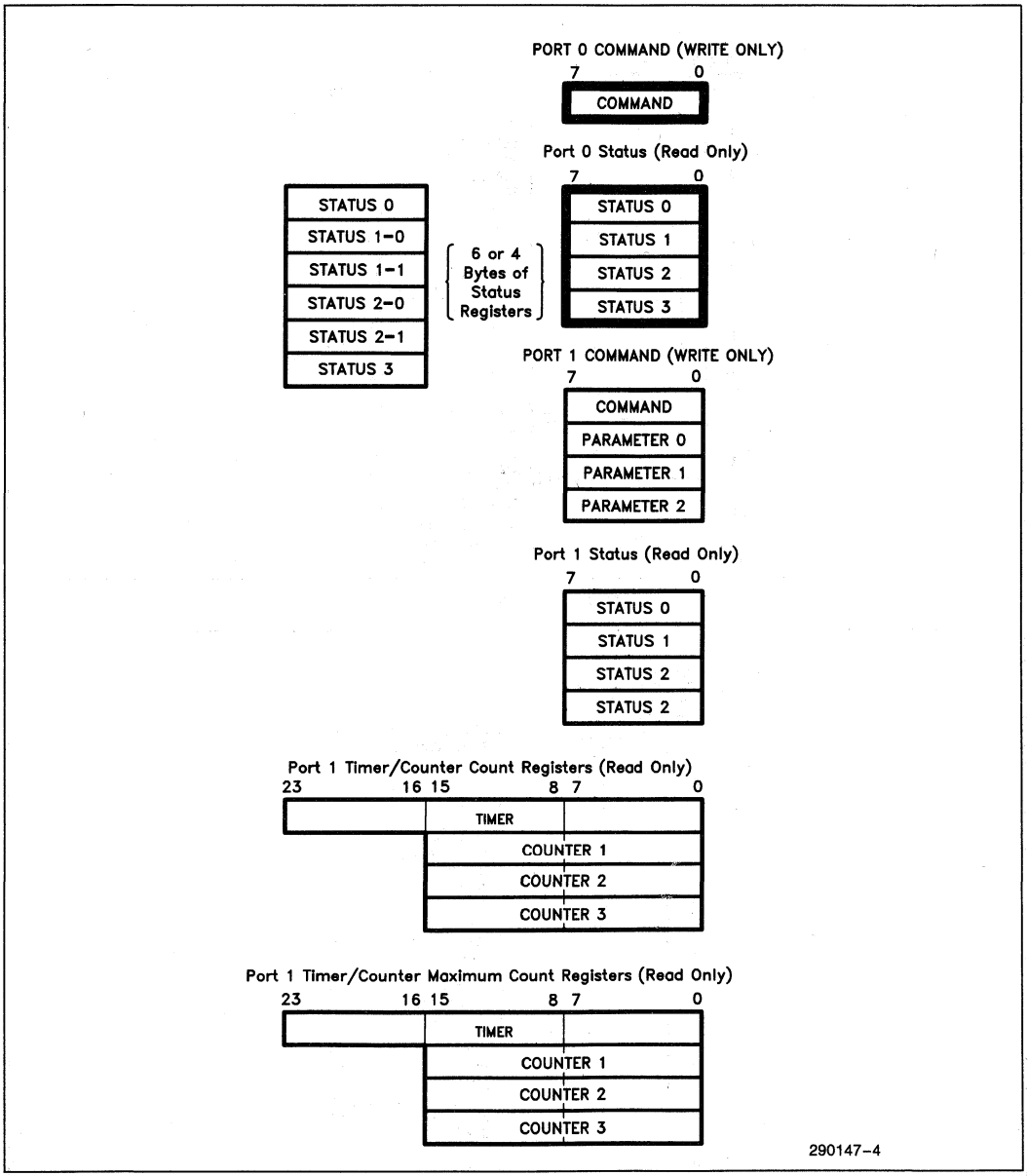


Figure 4. Programming Model—Directly Accessible Registers (Accessible Through 8-Bit I/O Port[s])

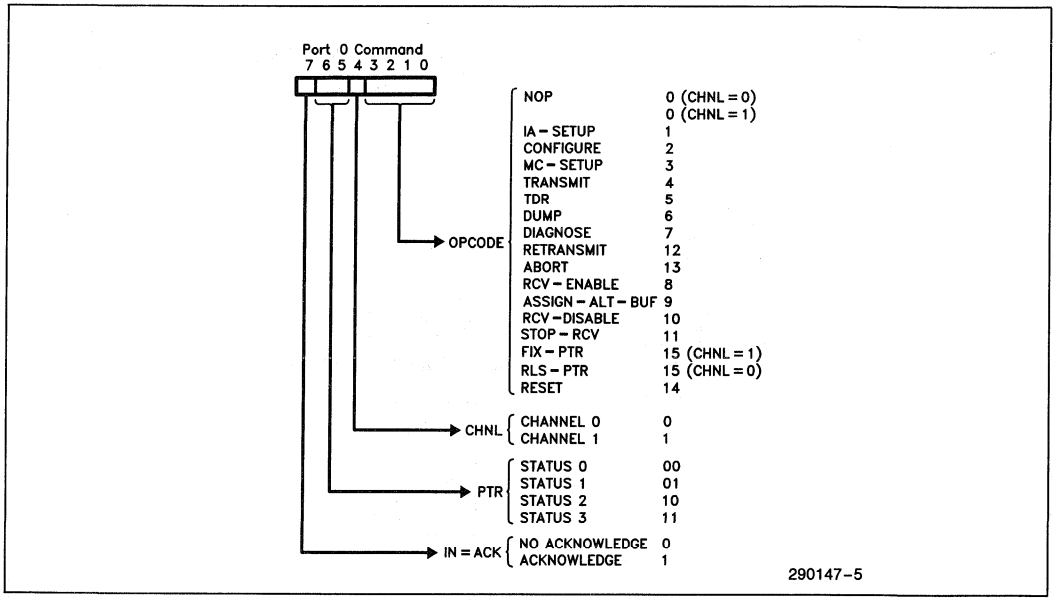


Figure 5. Port 0 Commands

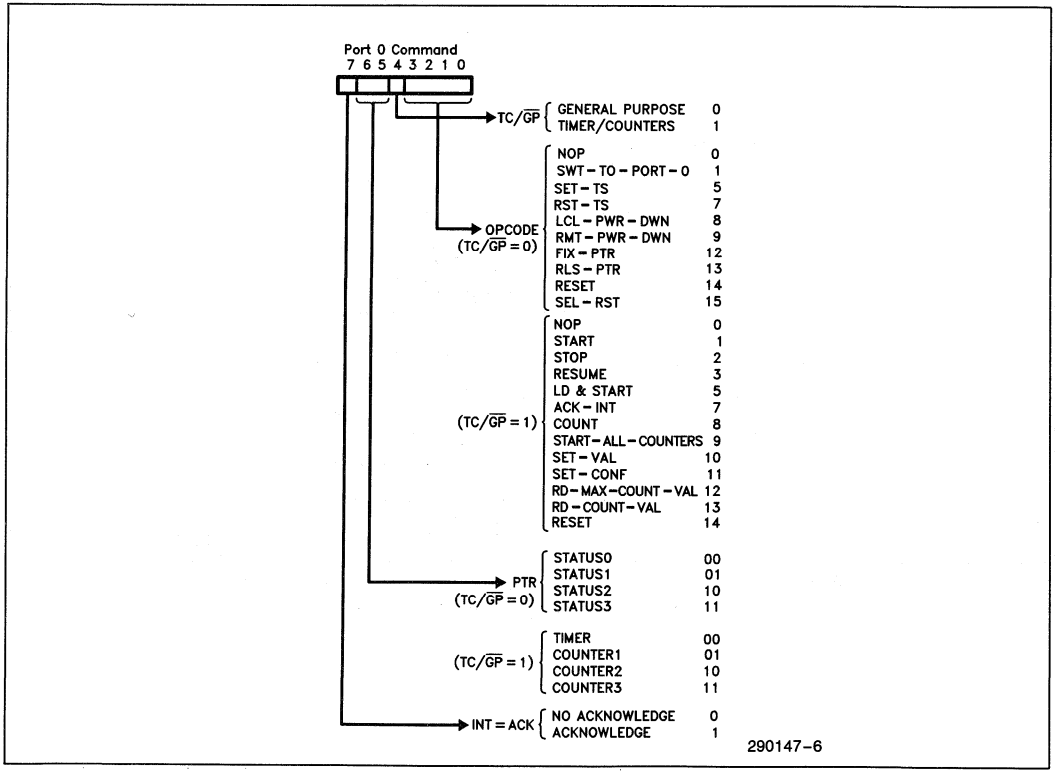
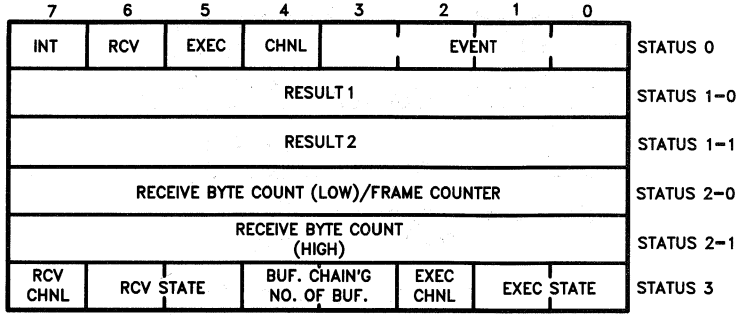


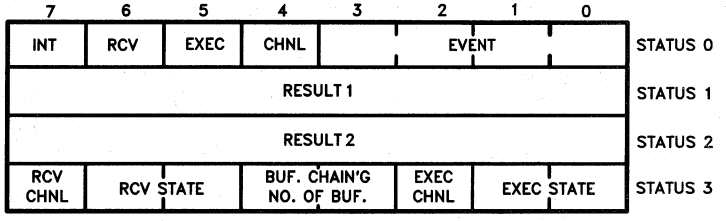
Figure 6. Port 1 Commands

Status Registers—6 Bytes



290147-7

Status Registers—4 Bytes (82588 Compatible Modes)

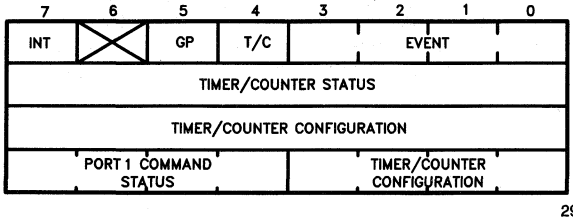


290147-8

Events	Value (Status 0)	Events	Value (Status 0)
CMOS*	0 (CHNL = 1)	Diagnose-Passed	7
IA-Setup-Done	1	End-Of-Frame	8
Configure-Done	2	Request-Next-Buffer	9
MC-Setup-Done	3	Reception-Aborted	10
Transmit-Done	4	Retransmit-Done	12
TDR-Done	5	Execution-Aborted	13
Dump-Done	6	Diagnose-Failed	15

*Available only after Hardware or Software Reset

Figure 7. Port 0 Status Registers



290147-9

Timer/Counter Events (T/C = 1)	Value* (Status 0)
Timer Expired	Bit 0 = 1
Counter 1 Expired	Bit 1 = 1
Counter 2 Expired	Bit 2 = 1
Counter 3 Expired	Bit 3 = 1
General Purpose Event (GP = 1)	**Value* (Status 0)**
REM-PWR-UP	9

*The 82590 may have more than one EVENT bit set by the time the CPU reads the status register.

Figure 8. Port 1 Status Registers

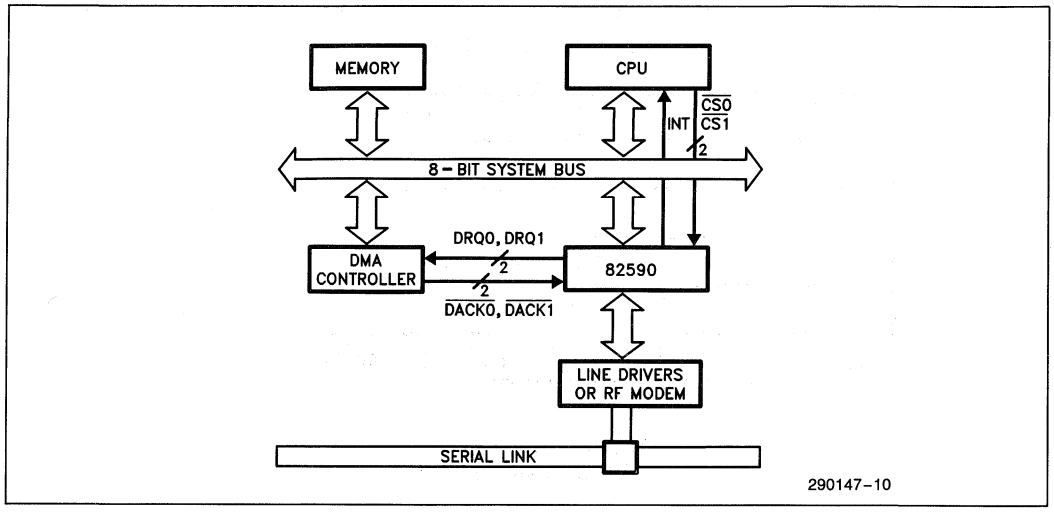


Figure 9. 82590/Host CPU Interaction

Table 2. Data Bus Control Signals and Functions

Pin Name			Function
$\overline{CS0}$ $\overline{CS1}$*	$\overline{RD}$	$\overline{WR}$	
1 0	X 1	X 1	No Transfer To/From Command/Status
0	0	0	Illegal
0	0	1	Read from Status Register
0	1	0	Write to Command Register
$\overline{DACK0}$ $\overline{DACK1}$*	$\overline{RD}$	$\overline{WR}$	
1 0	X 1	X 1	No DMA Transfer
0	0	0	Illegal
0	0	1	Data Read from DMA Channel 0 (or 1)
0	1	0	Data Write to DMA Channel 0 (or 1)

*Only one of $\overline{CS0}$, $\overline{CS1}$, $\overline{DACK0}$, or $\overline{DACK1}$ may be active at any time.

To initiate an operation such as Transmit or Configure (see Figure 5), the command from the CPU must first be written to the 82590. Any parameters or data associated with the command are transferred from memory to the 82590 using DMA. Upon completion of the operation, the 82590 updates the appropriate status registers and sends an interrupt to the CPU.

FRAME TRANSMISSION

To transmit a frame, the CPU prepares a Transmit Data Block in memory as shown in Figure 10. Its first two bytes specify the length of the rest of the block. The next few bytes (up to six) contain the destination address of the station the frame is being sent to. The rest of the block is the data field. The CPU programs the DMA controller with the start address of the block, length of the block, and other control information and then issues a Transmit command to the 82590. Upon receiving this command, the 82590 fetches the first two bytes of the block to determine its length. If the link is free and the first data byte was fetched, the 82590 begins transmitting the preamble and concurrently fetches more bytes from the Transmit Data Block and loads them into the transmit FIFO to keep them ready for transmission.

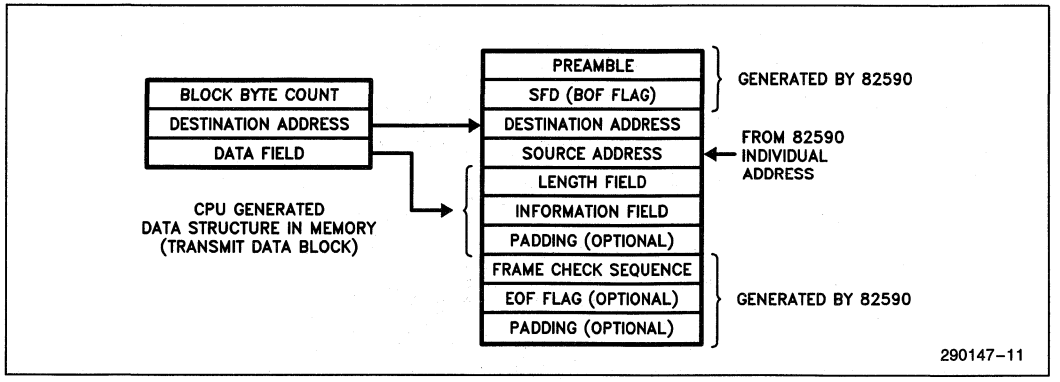


Figure 10. The 82590 Frame Structure and Location of Data Element in System Memory

The destination address is transmitted after the preamble. This is followed by the source or the station individual address, which was previously stored in the 82590 by the IA-Setup command. After this, the entire information field is transmitted, followed by a CRC field calculated by the 82590. If a collision is encountered during transmission of the frame, then the transmission is aborted after a jam pattern is sent. If the collision is detected during preamble or SFD (Start Frame Delimiter) transmission, the 82590 transmits the jam pattern after the SFD is transmitted. An interrupt is then generated to inform the CPU of the unsuccessful transmission due to a collision. The CPU reinitializes the DMA controller and issues a Retransmit command to the 82590. Retransmission is done by the CPU exactly as the Transmit command is done, except the Retransmit command keeps track of the number of collisions encountered. When the 82590 gets the Retransmit command and the backoff timer is expired, it transmits the frame again. Retransmission is repeated until the attempt is successful, or until the preprogrammed retry number expires.

If the 82590 is programmed to generate the $\overline{EOP}$ signal to the 8237 or 82380 DMA controller, or if it is used with a DMA controller which implements the Tightly Coupled Interface, retransmission is performed without CPU intervention.

FRAME RECEPTION

The 82590 can receive frames when its receiver has been enabled. The 82590 checks for an address match for an Individual address, a Multicast address, or a Broadcast address. In the Promiscuous mode the 82590 receives all frames. When the address match is successful, the 82590 transfers the frame to memory using the DMA controller. Before enabling the receiver, it is the CPU's responsibility to make a memory buffer area available to the receiver and to properly program the starting address of the DMA controller. The received frame is transferred to the memory buffer in the format shown in Figure 11. This method of reception is called Single Buffer reception; the entire frame is contained in one continuous buffer. Upon completion of reception, the status of the reception is appended at the end of the received frame in the memory buffer, and the total number of bytes transferred to the memory buffer is loaded into the internal status registers 1 and 2. An interrupt is then generated to inform the CPU of the frame reception.

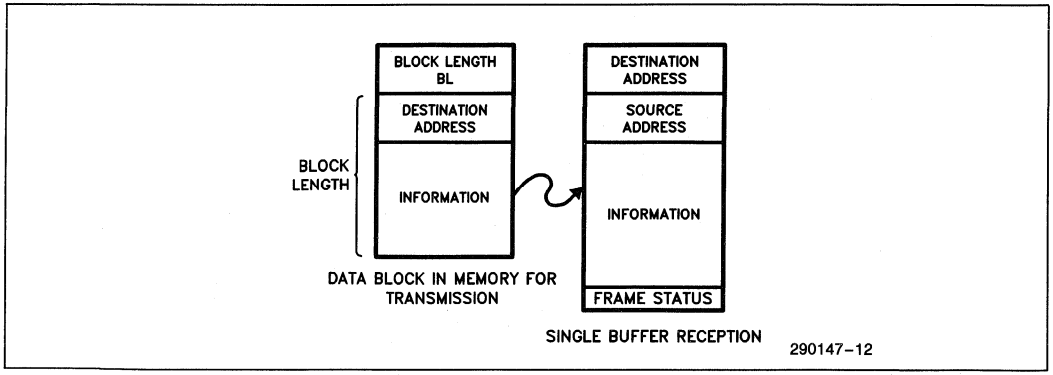


Figure 11. Single Buffer Reception

If the frame size is unknown, memory usage can be optimized by using Multiple Buffer reception. In this mode of operation, the CPU and DMA Controller can dynamically allocate memory space as it receives frames. This method requires both DMA channels to receive the frame alternately. As frame reception begins, the 82590 interrupts the CPU and automatically requests assignment of the next available buffer. The CPU does this and loads the second DMA channel with the next buffers information so the 82590 can immediately switch to the other channel when the current buffer is full. When the 82590 switches from the first to the second buffer it again interrupts the CPU and requests another buffer to be allocated on the previous channel. This process continues until the entire frame is received. The received frame is spread over multiple memory buffers. The link between the buffers is easily maintained by the CPU, using a buffer chain descriptor structure in memory as shown in Figure 12. This dynamic allocation of memory buffers results in efficient use of available storage when handling frames of widely differing sizes.

If the 82590 is programmed to generate the $\overline{EOP}$ signal to the 8237 or 82380 DMA controller, or if it is used with a DMA controller which implements the Tightly Coupled Interface, buffer reclamation and more advanced data structures for the buffer area can significantly improve system performance.

$\overline{EOP}$ SIGNAL TO THE DMA CONTROLLER

The 82590 can be programmed to assert the $\overline{EOP}$ signal to the 8237 or 82380 DMA controller when one or more of the following occurs:

- A collision during transmission
- An error (CRC or alignment) during reception
- A good frame reception

If the 8237 or 82380 is programmed for Auto-initialize mode and if the 82590 is programmed to assert the $\overline{EOP}$ signal on a collision during transmission, the retransmission following a collision is done automatically by the 8237 and the 82590. The 8237 will reinitialize itself automatically and the 82590 will retransmit the same frame from the same memory area without CPU intervention. When the 82590 is programmed for this mode it does not interrupt the CPU upon a collision, and the CPU does not need to issue a Retransmit command to the 82590. The CPU is interrupted only after a successful transmission or retransmission, or after a transmission failure, such as DMA underrun.

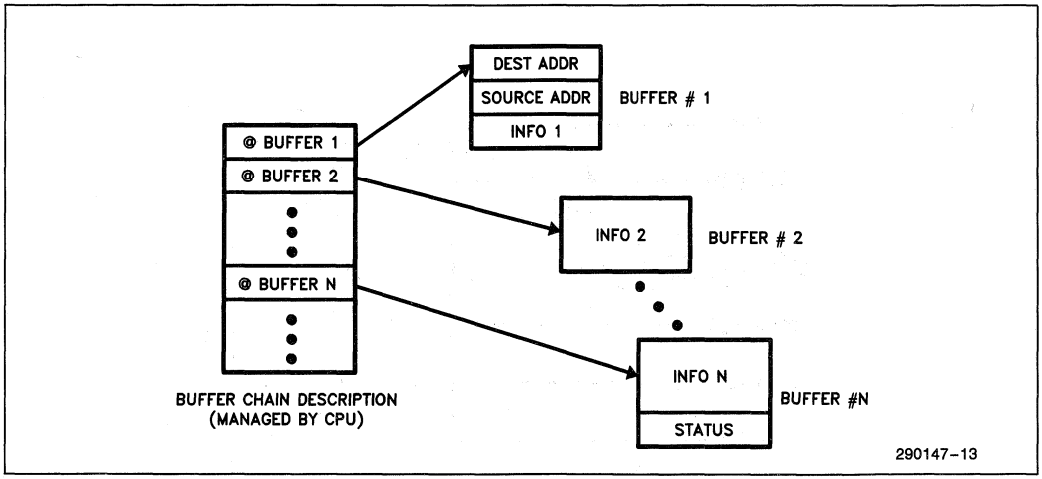


Figure 12. Multiple Buffer Reception

If the 82590 is programmed to assert the $\overline{EOP}$ signal when an error occurs during reception, the 8237 or the 82380 in Auto-initialize mode will be able to reclaim the memory area which would otherwise be wasted for the errored frame reception. If the 82590 is programmed to assert $\overline{EOP}$ at the end of a frame reception, automatic buffer switching can be accomplished by alternating the DMA channels with the 8237 or the 82380. When the 82380 is used, the buffer switching can be done with only one DMA channel.

The $\overline{EOP}$ signal must be derived from the $\overline{DACK1}$/$\overline{CS1}$/$\overline{EOP}$ pin using external logic (see Figure 13).

82590/82560 TIGHTLY COUPLED INTERFACE

The 82590 has a mode of operation called "Tightly Coupled Interface." In this mode the 82590 provides a tightly coupled interface to a DMA controller in order to execute some of the time-critical processes of the CSMA/CD protocol without any CPU intervention. By using the 82590's companion chip, the 82560, or by implementing the Tightly Coupled Interface in a DMA controller, operations such as automatic retransmission, continuous back-to-back frame reception, and transmit and/or receive buffer chaining can be accomplished.

The 82590 provides the status of the current active transmission or reception to the DMA controller by using the DRQ and $\overline{EOP}$ signals at the end of every DMA cycle. The status is encoded according to Table 3. As long as the 82590 generates DRQ High and $\overline{EOP}$ Floating at the rising edge of $\overline{RD}$ or $\overline{WR}$, the DMA controller repeats DMA transfers. If the transmission is completed without collisions or if the reception is good (no collision, no CRC, or no Alignment error), then DRQ and $\overline{EOP}$ both become Low at the end of a DMA transfer which follows the last DMA data transfer. If the transmission encountered a collision or if the reception had an error, DRQ becomes High and $\overline{EOP}$ becomes Low. The DMA controller must decode these signals appropriately and must reinitialize the DMA channel so it can retransmit the same frame or reclaim the otherwise wasted buffer. It is the DMA controller's responsibility to reprogram itself for the next appropriate operation.

The 82560 fully implements the Tightly Coupled Interface and provides very high-performance DMA services for the 82590 with minimal CPU involvement.

NETWORK MANAGEMENT AND DIAGNOSTICS

The 82590 provides a large set of diagnostic and network management functions including: internal and external loopback, monitor mode, optional capture of all frames regardless of destination address (Promiscuous mode), and time domain reflectometry for locating fault points in the network cable. The 82590 Dump command ensures software reliability by dumping the contents of the 82590 internal registers into the system memory.

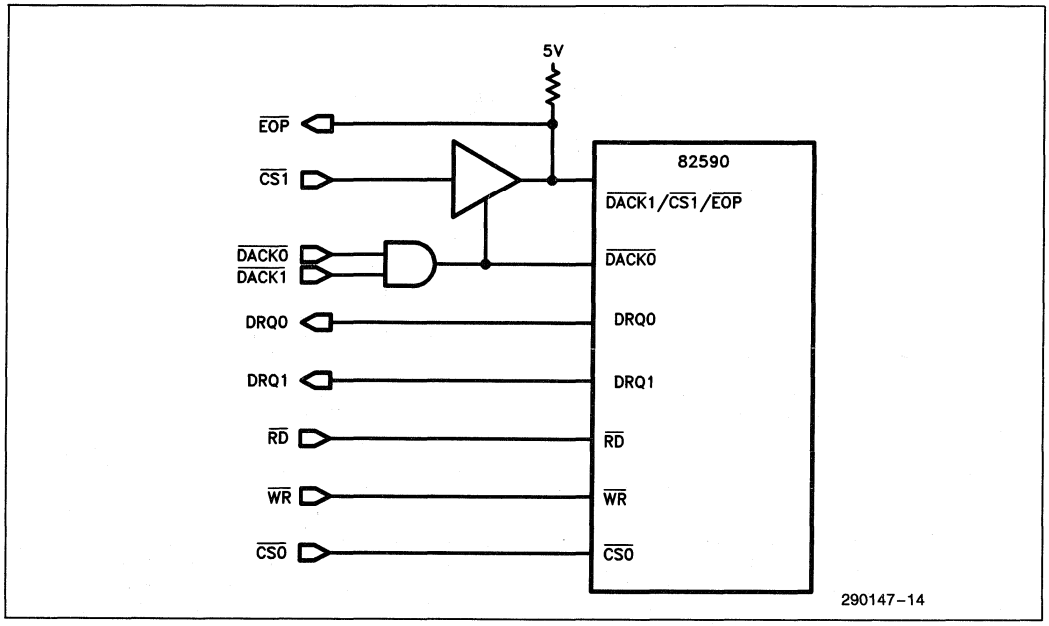


Figure 13. Demultiplexing DACK/CS1/EOP Pin

Table 3. Transmit/Receive Status Encoding on DRQ and EOP

DRQ	EOP	Status Information
0	Hi-Z	Idle
1	Hi-Z	DMA Transfer
0	0	Transmission or Reception Terminated OK
1	0	Transmission or Reception Aborted

OTHER ENHANCEMENTS

Compared to the 82588 the 82590 has a number of functional and performance enhancements. This section lists some of these enhancements which are not covered in other sections.

① **Multi-IA**—The 82590 implements multiple-individual address (Multi-IA) filtering. It can receive more than one IA frame in this mode.

② **Power Down Modes**—Two power down modes, Local Power Down and Remote Power Down, are available. When the 82590 is in Remote Power Down mode, it can be powered up remotely by sending a special frame to it.

③ **Automatic Padding and IEEE 802.3 Length Field**—If a frame to be transmitted is shorter than the configured Slot Time, the 82590 automatically appends pad bytes up to the shortest frame greater than the Slot Time. If the data field of a received frame is longer than the byte count indicated in the Length field, the extra bytes are stripped automatically according to the Length field. Erroneous conditions are detected and reported by the 82590. An example of such conditions is reception of a frame which is shorter than the byte count indicated in the Length field.

④ **Automatic Retransmission on Collision During Preamble**—The 82590 can be programmed to retransmit automatically if it detects a collision during transmission of the preamble.

⑤ **On-Chip Jabber Inhibit Function**—The 82590 can be programmed to provide an on-chip jabber inhibit function.

⑥ **CRC Transfer to Memory**—The 82590 can be programmed to transfer the CRC field of a received frame into memory.

⑦ **Loopback Signal to the 82C501**—The 82590 can be programmed to provide an active High loopback signal to the 82C501 (see Figure 14).

⑧ **StarLAN**—The 82590 can be configured to recognize the IEEE 802.3 1BASE5 Collision Presence Signal (CPS). In this mode it also delays deactivation of the $\overline{RTS}$ signal at the end of a frame transmission in order to insert an end-of-frame marker according to the standard.

APPLICATIONS

The 82590 can be used in a variety of applications. When it is used in High-Integration Mode, it implements most of the Data Link and Physical Layer functions required by the IEEE 802.3 1BASE5 (StarLAN) and the IBM PC Network—Baseband and Broadband. When it is used in High-Speed Mode, it can work with the 82C501 and the 82502 for IEEE 802.3 10BASE5 (Ethernet) and 10BASE2 (Cheapernet) implementations.

If the desired network requires determinism, the 82590's Deterministic Collision Resolution (DCR) method can be used.

Figure 15 shows a block diagram of an 82590/82560 StarLAN adapter board. The 82560 provides the following functions: DMA for the 82590 with Tightly Coupled Interface and dual-port memory control for the static RAM. The 82590 is configured to High-Integration mode to minimize the serial interface logic.

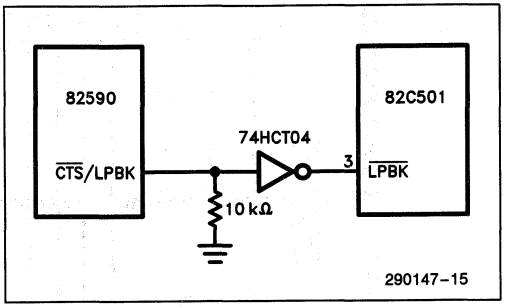


Figure 14. Loopback Output to the 82C501

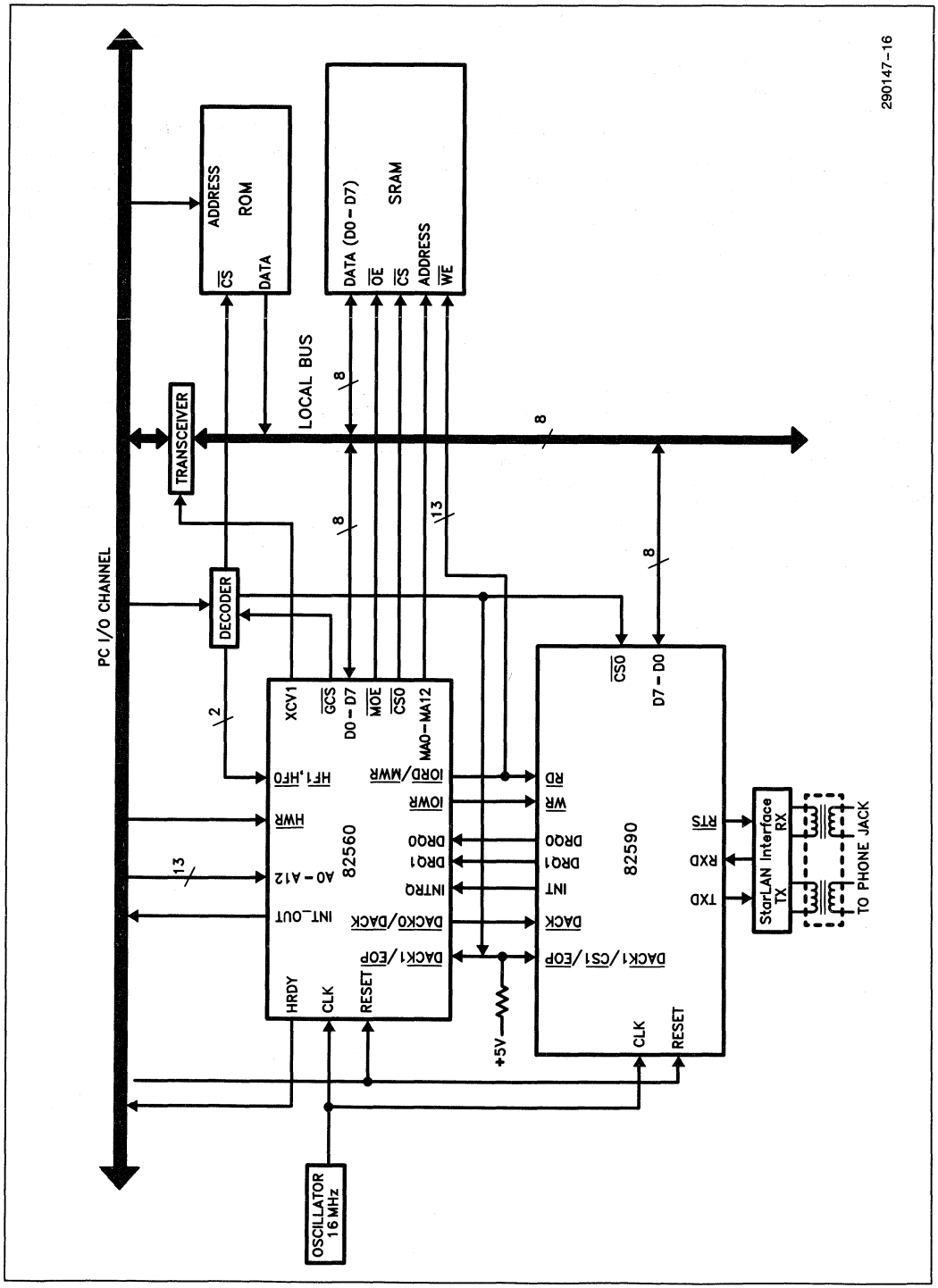


Figure 15. 82590/82560 StarLAN Adapter

ABSOLUTE MAXIMUM RATINGS*

Case Temperature (T_C) Under Bias
- Plastic ..... 0°C to +85°C
- PLCC ..... 0°C to +85°C

Storage Temperature ..... −65°C to +150°C

Voltage on any Pin with Respect to Ground ..... −1V to +7V

Power Dissipation ..... 550 mW

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS

(T_C [Plastic] = 0°C to +85°C, T_C [PLCC] = 0°C to +85°C, V_{CC} = +5V ±10%)

TxC and RxC have MOS levels (see V_{MIL}, V_{MIH}). All other signals have TTL levels (see V_{IL}, V_{IH}, V_{OL}, V_{OH}).

Symbol	Parameter	Min	Max	Units	* Test Conditions
V_{IL}	Input Low Voltage (TTL)	−0.5	0.8	V	
V_{IH}	Input High Voltage (TTL)	2.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage (TTL)		0.45	V	I_{OL} = 2.0 mA
V_{OH}	Output High Voltage (TTL)	2.4		V	I_{OH} = − 400 µA
V_{MIL}	Input Low Voltage (MOS)	−0.5	0.6	V	
V_{MIH}	Input High Voltage (MOS)	3.9	V_{CC} + 0.5	V	
I_{LI}	Input Leakage Current		±10	µA	0V ≤ V_{IN} ≤ V_{CC}
I_{LIO}	I/O Leakage Current		±10	µA	0.45V ≤ V_{OUT} ≤ V_{CC} − 0.45V
C_{IN}	Capacitance of Input Buffer		10	pF	Frequency = 1 MHz
C_{IO}	Capacitance of I/O Buffer		20	pF	Frequency = 1 MHz
I_{CC}	Power Supply Current		100 80	mA	(Note 1)
			5	mA	(Note 2)
			1	mA	(Note 3)
					(Note 4)

ADVANCE INFORMATION SEE INTEL FOR DESIGN-IN INFORMATION

NOTES:
1. System side (CLK) at 16 MHz, serial side ($\overline{TxC}$) at 20 MHz in High-Speed Mode.
2. Remote power-up mode.
3. Power-down mode—all inputs connected to V_{CC} level.
4. General formulas for current are: (a) f(CLK) × 4 + f($\overline{TxC}$) × 2 for High Speed Mode. (b) f(CLK) × 4 + f($\overline{TFC}$) × 0.3 + f($\overline{TFC}$) × 32/SR, where SR is the sampling rate in High-Integration Mode.

A.C. CHARACTERISTICS (C_L on all outputs is 20 pF–125 pF unless otherwise specified.)

Symbol	Parameter	Min	Max	Units	Test Conditions
SYSTEM CLOCK INPUT PARAMETERS					
t_1	CLK Cycle Time	62.5		ns	
t_2	CLK Low Time	30		ns	(Note 5)
t_3	CLK High Time	30		ns	(Note 5)
t_4	CLK Rise Time		5	ns	(Note 1)
t_5	CLK Fall Time		5	ns	(Note 2)
SYSTEM CLOCK OUTPUT PARAMETERS					
t_{98}	CLK Cycle Time	120		ns	(Notes 4, 7)
t_{99}	CLK Low Time	50		ns	(Note 4)
t_{100}	CLK High Time	50		ns	(Note 4)
t_{101}	CLK Rise Time		15	ns	(Notes 1, 4)
t_{102}	CLK Fall Time		15	ns	(Notes 2, 4)
RESET PARAMETERS					
t_6	Reset Active to Clock Low	20		ns	(Note 3)
t_8	Reset Pulse Width	$4t_1$		ns	
t_9	Control Inactive after Reset		125	ns	
INTERRUPT PARAMETERS					
t_{10}	CLK High to Interrupt Active		55	ns	(Note 4)
t_{11}	$\overline{WR}$ Inactive to Interrupt		55	ns	(Note 4)
t_{103}	Int Low to Int High Gap	$2t_1$		ns	(Note 4)
WRITE PARAMETERS					
t_{12}	$\overline{CS0}$, $\overline{CS1}$, $\overline{DACK0}$, or $\overline{DACK1}$ Setup to $\overline{WR}$ Low	0		ns	
t_{13}	$\overline{WR}$ Pulse Width	55		ns	
t_{14}	$\overline{CS0}$, $\overline{CS1}$, $\overline{DACK0}$, or $\overline{DACK1}$ Hold after $\overline{WR}$ High	0		ns	
t_{15}	Data Setup to $\overline{WR}$ High	30		ns	
t_{16}	Data Hold after $\overline{WR}$ High	0		ns	
t_{94}	Write Cycle Time	$2t_1$		ns	
t_{96}	$\overline{WR}$ Inactive Time	55		ns	

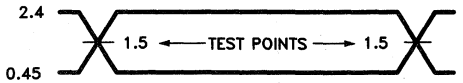
A.C. CHARACTERISTICS (C_L on all outputs is 20 pF–125 pF unless otherwise specified.) (Continued)

Symbol	Parameter	Min	Max	Units	Test Conditions
READ PARAMETERS					
t_{17}	$\overline{CS0}$, $\overline{CS1}$, $\overline{DACK0}$, or $\overline{DACK1}$ Setup to $\overline{RD}$ Low	0		ns	
t_{18}	$\overline{RD}$ Pulse Width	55		ns	
t_{19}	$\overline{CS0}$, $\overline{CS1}$ $\overline{DACK0}$, or $\overline{DACK1}$ Hold after $\overline{RD}$ High	0		ns	
t_{20}	$\overline{RD}$ Low to Data Valid		45	ns	
t_{21}	Data Float after $\overline{RD}$ High		40	ns	
t_{95}	Read Cycle Time	$2t_1$		ns	
t_{97}	$\overline{RD}$ Inactive Time	55		ns	
DMA PARAMETERS					
t_{22}	CLK Low to DRQ0 or DRQ1 Active		55	ns	(Note 4)
t_{23}	$\overline{WR}$ or $\overline{RD}$ Low to DRQ0 or DRQ1 Inactive		45	ns	(Note 4)
t_{104}	$\overline{WR}$ or $\overline{RD}$ Low to DRQ0 or DRQ1 Inactive—82560 Retransmit or Receive Buffer Reclaim	2.5	65	ns	(Note 4)
t_{105}	$\overline{WR}$ or $\overline{RD}$ Low to $\overline{EOP}$ Active		45	ns	(Note 6)
t_{106}	$\overline{EOP}$ Float after $\overline{DACK0}$ or $\overline{DACK1}$ Going Inactive		40	ns	(Note 6)

NOTES:
1. 0.8V to 2.0V.
2. 2.0V to 0.8V.
3. To guarantee recognition on the next clock.
4. C_L = 50 pF.
5. Measured at 1.5V.
6. Open drain I/O pin.
7. None of the A.C. Parameters are related to the CLK output pin.

A.C. TESTING INPUT/OUTPUT WAVEFORMS

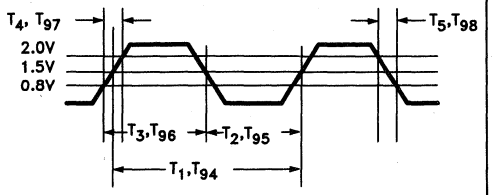
TTL Input/Output Voltage Levels for Timing Measurements



290147-17

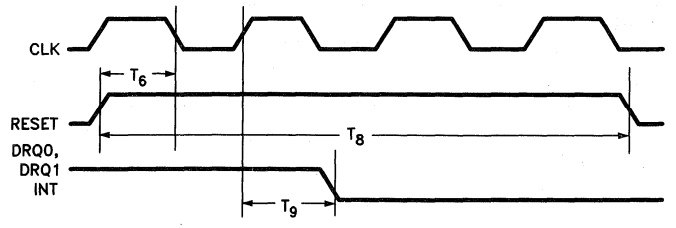
A.C. Testing inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing measurements are made at 1.5V for both a Logic "1" and "0". Rise and fall times are measured between 0.8V and 2.0V.

System Clock I/O Voltage Levels for Timing Measurements



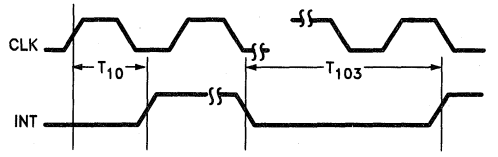
290147-18

Reset Timing



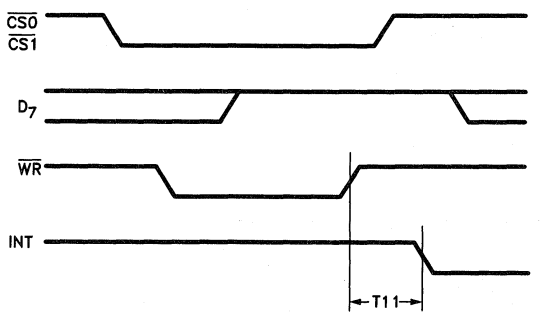
290147-19

Interrupt Timing (Going Active)



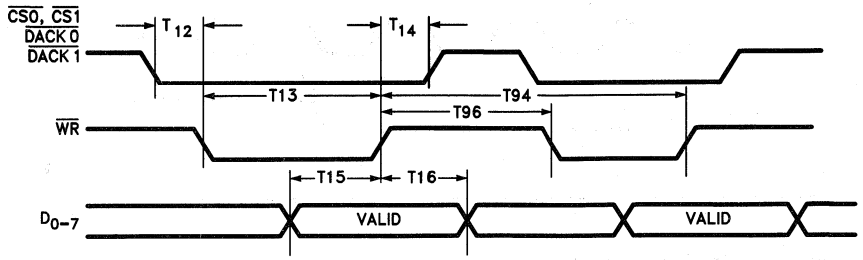
290147-20

Interrupt Timing (Going Inactive)



290147-21

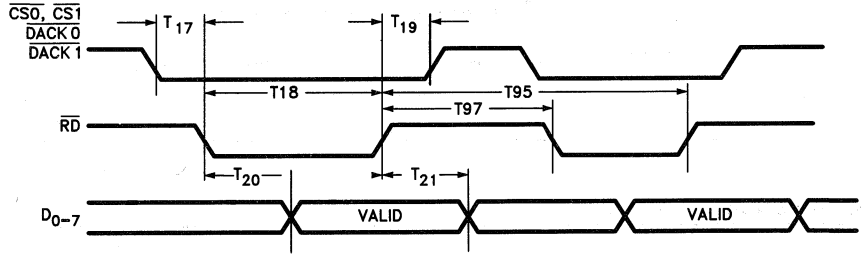
WRITE TIMING



290147-22

NOTE:
DACK0 or DACK1 may remain active continuously for consecutive DMA cycles.

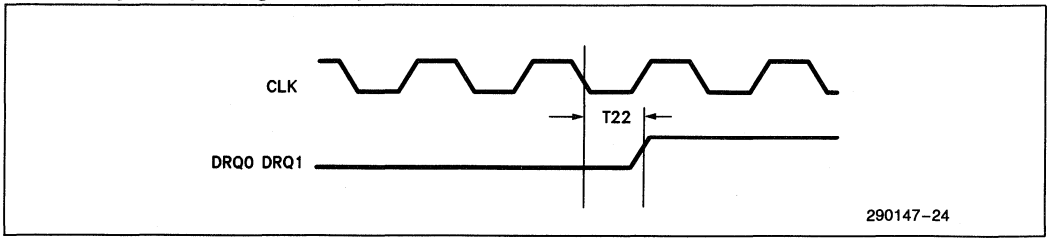
READ TIMING



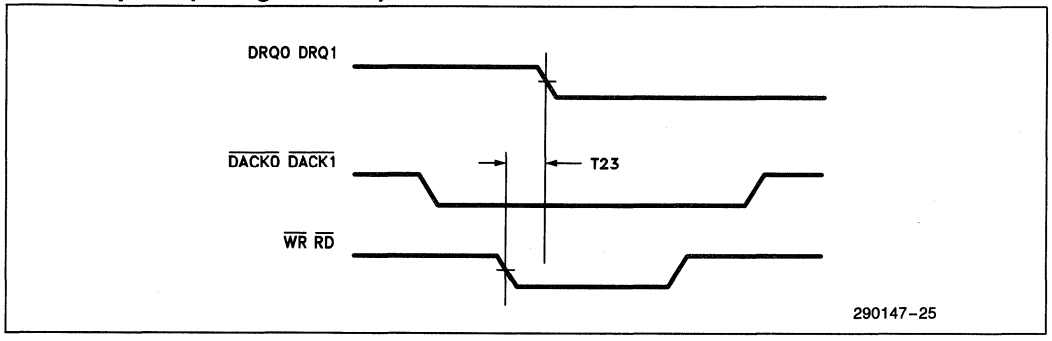
290147-23

NOTE:
DACK0 or DACK1 may remain active continuously for consecutive DMA cycles.

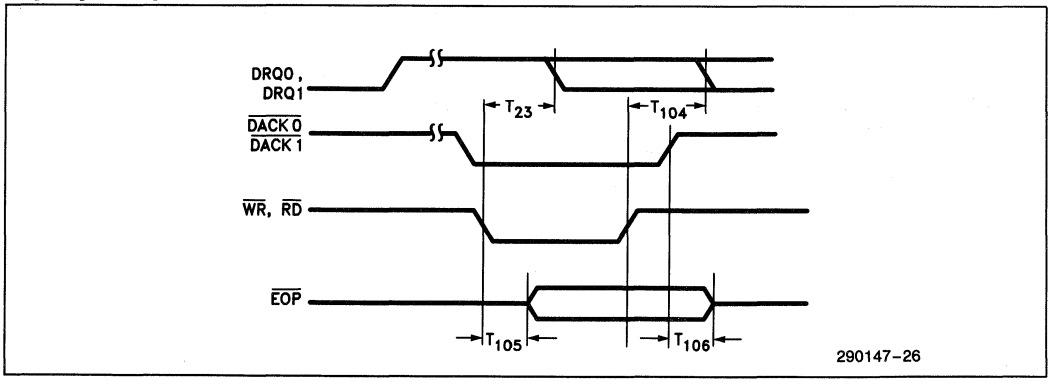
DMA Request (Going Active)



DMA Request (Going Inactive)



Tightly Coupled Interface



SERIAL INTERFACE A.C. TIMING CHARACTERISTICS

$\overline{TFC}$ is the Crystal or Serial Clock Input at X1.

	X8 Sampling	X10 Sampling	X16 Sampling	X18 Sampling
High Integration Mode For $\overline{TFC}$ Frequency = 1 MHz to 32 MHz (High)				
$\overline{TCLK}$ Frequency	0.125 MHz–4 MHz	100 kHz–3.2MHz	62.5 kHz–2 MHz	55.6 kHz–1.78 MHz
t_{29} = $\overline{TCLK}$ Cycle Time	8 × t_{24}	10 × t_{24}	16 × t_{24}	18 × t_{24}
t_{30} = $\overline{TCLK}$ High Time	t_{24} (Typically)	t_{24} (Typically)	t_{24} (Typically)	t_{24} (Typically)
t_{31} = $\overline{TCLK}$ Low Time	7 × t_{24} (Typically)	9 × t_{24} (Typically)	15 × t_{24} (Typically)	17 × t_{24} (Typically)
For $\overline{TFC}$ Frequency = 0 MHz to 1 MHz (Low)				
$\overline{TCLK}$ Frequency	0–0.125 MHz	0–100 kHz	0–62.5 kHz	0–55.6 kHz
t_{29} = $\overline{TCLK}$ Cycle Time	8 × t_{24}	10 × t_{24}	16 × t_{24}	18 × t_{24}
t_{30} = $\overline{TCLK}$ High Time	t_{25} (Typically)	t_{25} (Typically)	t_{25} (Typically)	t_{25} (Typically)
t_{31} = $\overline{TCLK}$ Low Time	7 × t_{24} + t_{26} (Typically)	9 × t_{24} + t_{26} (Typically)	15 × t_{24} + t_{26} (Typically)	17 × t_{24} + t_{26} (Typically)

NOTES:
X10 and X18 are available only for Manchester or Differential Manchester encoding/decoding
t_{24} = Serial Clock Cycle Time
t_{25} = Serial Clock High Time
t_{26} = Serial Clock Low Time

HIGH SPEED MODE

- Applies for $\overline{TxC}$, $\overline{RxC}$
- f_{max} = 20 MHz ±100 ppm
- For Manchester, symmetry is required: t_{64}, $t_{64} = \frac{1}{2f} \pm 5\%$

HIGH INTEGRATION MODE

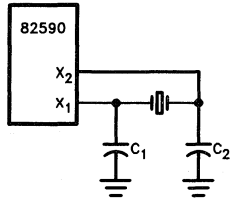
Symbol	Parameter	Min	Max	Units	Test Conditions
EXTERNAL (FAST) CLOCK PARAMETERS					
t_{24}	Fast Clock ($\overline{TFC}$) Cycle Time	31.25		ns	(Notes 1, 16)
t_{25}	$\overline{TFC}$ High Time	(Note 13)		ns	(Notes 1, 7)
t_{26}	$\overline{TFC}$ Low Time	12		ns	(Notes 1, 17)
t_{27}	$\overline{TFC}$ Rise Time		3	ns	(Note 1)
t_{28}	$\overline{TFC}$ Fall Time		3	ns	(Note 1)
TRANSMIT CLOCK PARAMETERS					
t_{29}	Transmit Clock ($\overline{TCLK}$) Cycle Time	(Note 13)		ns	(Note 2)
t_{30}	$\overline{TCLK}$ High Time	(Note 7)		ns	(Note 2)
t_{31}	$\overline{TCLK}$ Low Time	(Note 8)		ns	(Note 2)
t_{32}	$\overline{TCLK}$ Rise Time		10	ns	(Note 2)
t_{33}	$\overline{TCLK}$ Fall Time		10	ns	(Note 2)
TRANSMIT DATA PARAMETERS (MANCHESTER, DIFFERENTIAL MANCHESTER)					
t_{34}	TxD Transition-Transition	(Note 14)		ns	
t_{35}	$\overline{TCLK}$ Low to TxD Mid Bit Cell Transition		(Note 10)	ns	(Note 2)
t_{36}	$\overline{TCLK}$ Low to TxD Bit Cell Boundary Transition		(Note 9)	ns	(Note 2)
t_{37}	TxD Rise Time		10	ns	(Note 2)
t_{38}	TxD Fall Time		10	ns	(Note 2)
TRANSMIT DATA PARAMETERS (NRZI)					
t_{39}	TxD Transition-Transition	(Note 15)		ns	
t_{40}	$\overline{TCLK}$ Low to TxD Transition		(Note 9)	ns	(Note 2)
t_{41}	TxD Rise Time		10	ns	(Note 2)
t_{42}	TxD Fall Time		10	ns	(Note 2)
$\overline{RTS}$, $\overline{CTS}$ PARAMETERS					
t_{43}	$\overline{TCLK}$ Low to $\overline{RTS}$ Low		(Note 9)	ns	(Note 2)
t_{44}	$\overline{CTS}$ Low to $\overline{TCLK}$ Low	35		ns	
t_{45}	$\overline{TCLK}$ Low to $\overline{RTS}$ High		(Note 9)	ns	(Note 2)
t_{46}	$\overline{TCLK}$ Low to $\overline{CTS}$ Invalid $\overline{CTS}$ Hold Time	10		ns	(Notes 3, 12)
t_{47}	$\overline{CTS}$ High to $\overline{TCLK}$ Low; $\overline{CTS}$ Setup Time to Stop Transmission	35		ns	(Note 3)
IFS PARAMETERS					
t_{48}	Interframe Delay	(Note 4)		ns	

HIGH INTEGRATION MODE (Continued)

Symbol	Parameter	Min	Max	Units	Test Conditions
COLLISION DETECT PARAMETERS					
t_{49}	$\overline{CDT}$ Low to $\overline{TCLK}$ High External Collision Detect Setup Time	35		ns	(Note 12)
t_{50}	$\overline{CDT}$ High to $\overline{TCLK}$ High	35		ns	(Note 12)
t_{51}	$\overline{TCLK}$ High to $\overline{CDT}$ Inactive $\overline{CDT}$ Hold Time	10		ns	(Note 12)
t_{52}	$\overline{CDT}$ Low to Jamming Start		(Note 5)	ns	
t_{53}	Jamming Period	(Note 6)		ns	
RECEIVED DATA PARAMETERS (MANCHESTER, DIFFERENTIAL MANCHESTER)					
t_{54}	RxD Transition-Transition	115		ns	(Note 14)
t_{55}	RxD Rise Time		10	ns	
t_{56}	RxD Fall Time		10	ns	
RECEIVED DATA PARAMETERS (NRZI)					
t_{57}	RxD Transition-Transition	240		ns	(Note 11)
t_{58}	RxD Rise Time		10	ns	
t_{59}	RxD Fall Time		10	ns	
EXTERNAL LOOPBACK PARAMETERS					
t_{107}	$\overline{TCLK}$ Low to LPBK High		50	ns	(Note 2)
t_{108}	$\overline{TCLK}$ Low to LPBK Float		50	ns	(Note 2)

NOTES:
1. MOS Levels.
2. 1 TTL Load + 50 pF.
3. Abnormal End of Transmission: $\overline{CTS}$ expires before $\overline{RTS}$.
4. Programmable value: $t_{28} = N_{IFS} \times t_{29}$ (ns)
   N_{IFS}: the IFS configuration value.
   If N_{IFS} is less than 12 then it is enforced to 12.
5. Programmable Values:
   $t_{52} = N_{CDF} \times t_{29}$ + (12 to 15) $\times t_{29}$ (if collision occurs after preamble).
   N_{CDF}: The Collision Detect Filter Configuration Value.
6. $t_{53} = 32 \times t_{29}$
7. Depends on frequency range:
   High Range: $t_{24} - 10$ ns
   Low Range: $t_{25} - 10$ ns
8. $t_{31} = t_{29} - t_{30} - t_{32} - t_{33}$
9. $2 \times t_{24}$ + 40 ns for 8X or 10X
   $4 \times t_{24}$ + 40 ns for 16X
10. $6 \times t_{24}$ + 40 ns for 8X
    $12 \times t_{24}$ + 40 ns for 16X
    $7 \times t_{24}$ + 40 ns for 10X
    $13 \times t_{24}$ + 40 ns for 18X
11. $8 \times t_{24} - 10$ ns for 8X
    $10 \times t_{24} - 10$ ns for 10X
    $16 \times t_{24} - 10$ ns for 16X
    $18 \times t_{24} - 10$ ns for 18X
12. To Guarantee recognition on the next clock.
13. 10 ns for High Range
    30 ns for Low Range
14. $4 \times t_{24} - 10$ ns for 8X
    $5 \times t_{24} - 10$ ns for 10X
    $8 \times t_{24} - 10$ ns for 16X
    $9 \times t_{24} - 10$ ns for 18X
15. $t_{29} - 10$ ns
16. See Figure "CRYSTAL CONNECTION."
17. Maximum capacitance load on the X_2 pin when an external MOS clock is connected to X_1:
    15 pF for DC to 16 MHz
    5 pF for 16 MHz to 32 MHz

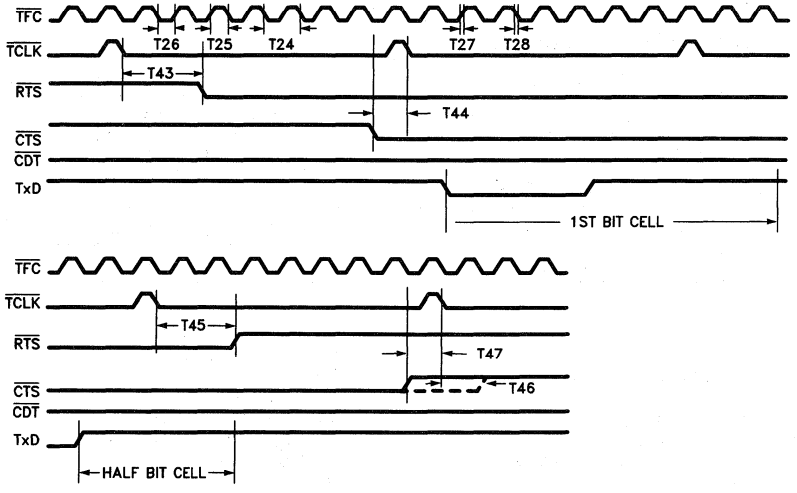
CRYSTAL CONNECTION



290147-27

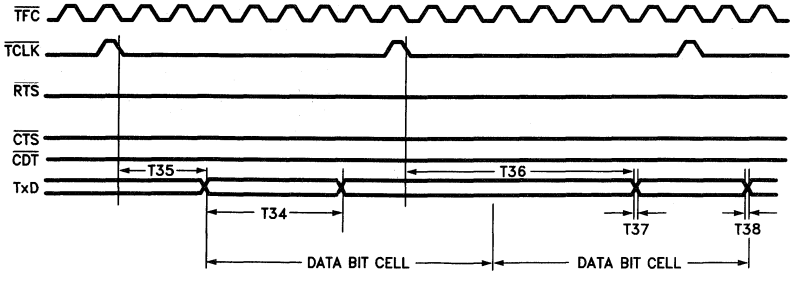
NOTES:
1. High-quality, parallel resonant, fundamental-mode crystals are recommended for maximum accuracy.
2. C1, C2, and stray capacitance of the board should be adjusted so the total capacitance load on the crystal is approx. 15 pF.
3. For IEEE 802.3 applications, the crystal must be accurate to ±35 PPM over a range of 0°C to 70°C.

TRANSMIT TIMINGS—CLOCKS $\overline{RTS}$ AND $\overline{CTS}$



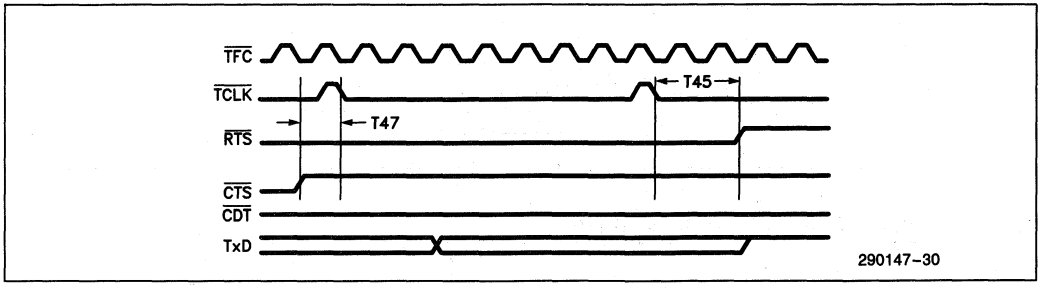
290147-28

TRANSMIT TIMINGS—MANCHESTER DATA ENCODING

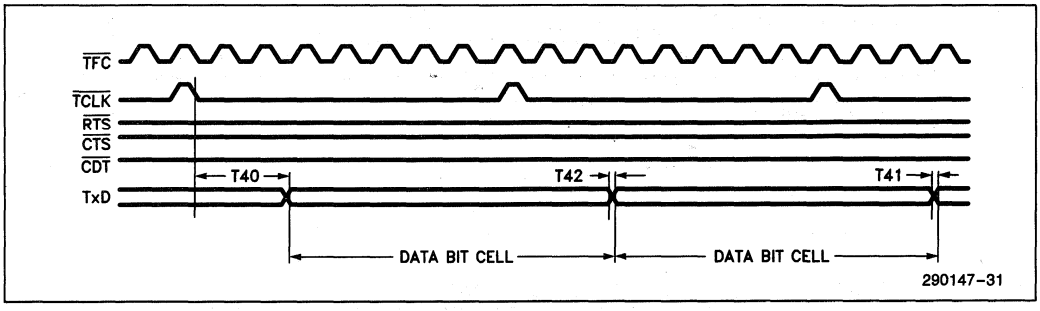


290147-29

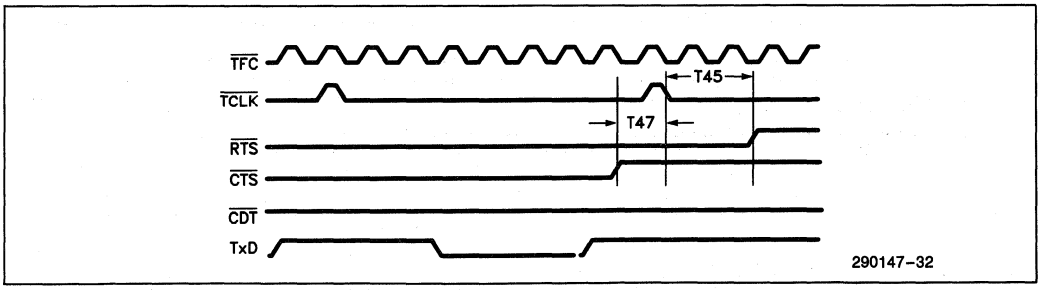
TRANSMIT TIMINGS—LOST $\overline{CTS}$



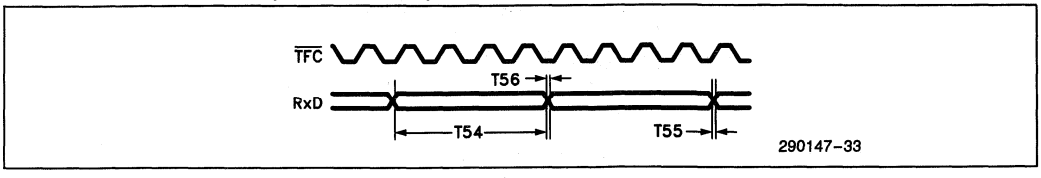
TRANSMIT TIMINGS—NRZI DATA ENCODING



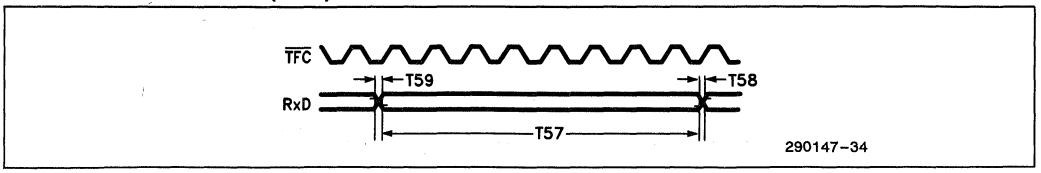
TRANSMIT TIMINGS—LOST $\overline{CTS}$



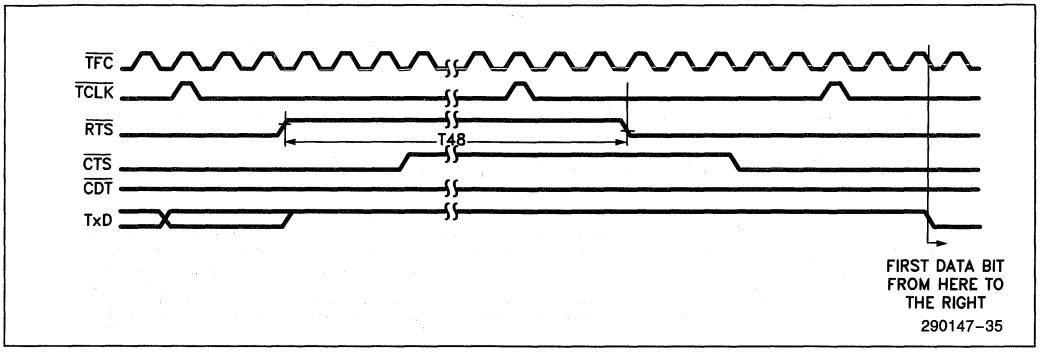
RECEIVE DATA TIMINGS (MANCHESTER)



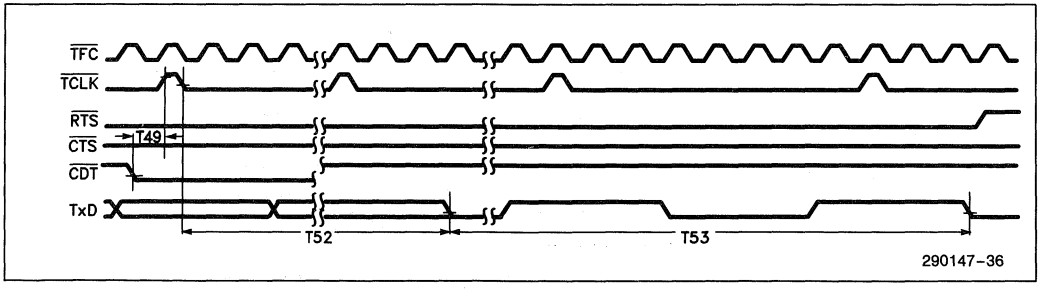
RECEIVE DATA TIMINGS (NRZI)



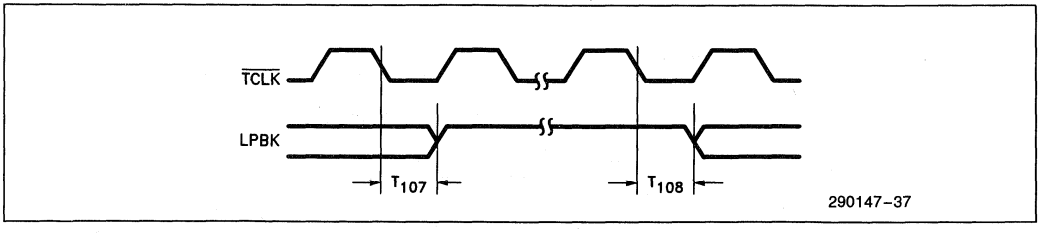
TRANSMIT TIMINGS—INTERFRAME SPACING



TRANSMIT TIMINGS—COLLISION DETECT AND JAMMING



LOOPBACK OUTPUT SIGNAL TIMINGS



HIGH SPEED MODE

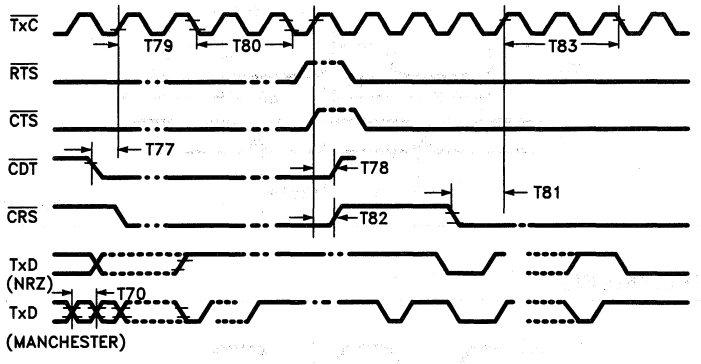
Symbol	Parameter	Min	Max	Units	Test Conditions
TRANSMIT/RECEIVE CLOCK PARAMETERS					
t_{60}	$\overline{RxC}$, $\overline{TxC}$ Cycle Time	50		ns	(Notes 1, 3)
t_{61}	$\overline{TxC}$ Rise Time		5	ns	(Note 1)
t_{62}	$\overline{TxC}$ Fall Time		5	ns	(Note 1)
t_{63}	$\overline{TxC}$ High Time	18		ns	(Notes 1, 3)
t_{64}	$\overline{TxC}$ Low Time	19		ns	(Notes 1, 3)
TRANSMIT DATA PARAMETERS					
t_{65}	TxD Rise Time		10	ns	(Note 4)
t_{66}	TxD Fall Time		10	ns	(Note 4)
t_{67}	$\overline{TxC}$ Low to TxD Valid		25	ns	(Notes 4, 5)
t_{68}	$\overline{TxC}$ Low to TxD Transition		25	ns	(Notes 2, 4)
t_{69}	$\overline{TxC}$ High to TxD Transition		25	ns	(Notes 4, 5)
t_{70}	TxD Transition-Transition	20		ns	(Notes 2, 4)
t_{71}	$\overline{TxC}$ Low to TxD High (At the Transmission End)		25	ns	(Note 4)
$\overline{RTS}$, $\overline{CTS}$ PARAMETERS					
t_{72}	$\overline{TxC}$ Low to $\overline{RTS}$ Low Time to Activate $\overline{RTS}$		25	ns	(Note 4)
t_{73}	$\overline{CTS}$ Low to $\overline{TxC}$ Low $\overline{CTS}$ Setup Time	20		ns	
t_{74}	$\overline{TxC}$ Low to $\overline{RTS}$ High		25	ns	(Note 4)
t_{75}	$\overline{TxC}$ Low to $\overline{CTS}$ Invalid. $\overline{CTS}$ Hold Time	10		ns	(Note 6)
t_{75a}	$\overline{CTS}$ High to $\overline{TxC}$ Low. $\overline{CTS}$ Setup Time to Stop Transmission	20		ns	(Note 6)
INTERFRAME SPACING PARAMETER					
t_{76}	Inter Frame Delay	(Note 8)		ns	

HIGH SPEED MODE (Continued)

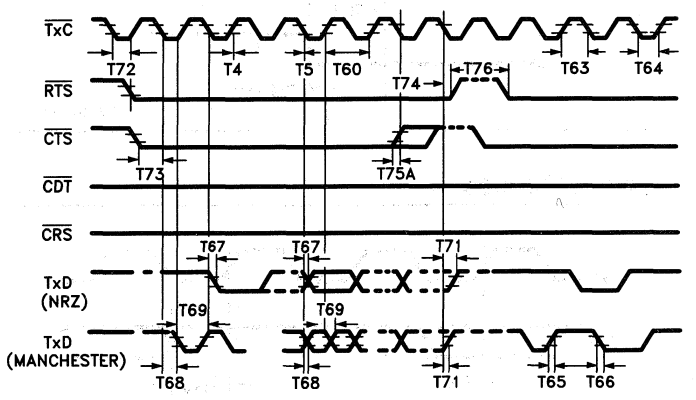
Symbol	Parameter	Min	Max	Units	Test Conditions
$\overline{CRS}$, $\overline{CDT}$ PARAMETERS					
t_{77}	$\overline{CDT}$ Low to $\overline{TxC}$ High; External Collision Detect Setup Time	20		ns	
t_{78}	$\overline{TxC}$ High to $\overline{CDT}$ Inactive; $\overline{CDT}$ Hold Time	10		ns	(Note 12)
t_{79}	$\overline{CDT}$ Low to Jam Start		(Note 9)	ns	
t_{80}	Jamming Period		(Note 10)	ns	
t_{81}	$\overline{CRS}$ Low to $\overline{TxC}$ High; Carrier Sense Setup Time	20		ns	
t_{82}	$\overline{TxC}$ High to $\overline{CRS}$ Inactive; $\overline{CRS}$ Hold Time	10		ns	(Note 12)
t_{83}	$\overline{CRS}$ High to Jamming Start (Internal Collision Detect)		(Note 11)	ns	
t_{84}	$\overline{CRS}$ High to $\overline{RxC}$ High; $\overline{CRS}$ Inactive Setup Time	30		ns	
t_{85}	$\overline{RxC}$ High to $\overline{CRS}$ High; $\overline{CRS}$ Inactive Hold Time	10		ns	
RECEIVE CLOCK PARAMETERS					
t_{86}	$\overline{RxC}$ Rise Time		5	ns	(Note 1)
t_{87}	$\overline{RxC}$ Fall Time		5	ns	(Note 1)
t_{88}	$\overline{RxC}$ High Time	18		ns	(Note 1)
t_{89}	$\overline{RxC}$ Low Time	19		ns	(Note 1)
RECEIVED DATA PARAMETERS					
t_{90}	$\overline{RxD}$ Setup Time	15		ns	(Note 5)
t_{91}	$\overline{RxD}$ Hold Time	15		ns	(Note 5)
t_{92}	$\overline{RxD}$ Rise Time		10	ns	
t_{93}	$\overline{RxD}$ Fall Time		10	ns	
EXTERNAL LOOPBACK PARAMETERS					
t_{109}	$\overline{TxC}$ Low to LPBK High		t_{60}	ns	(Note 4)
t_{110}	$\overline{TxC}$ Low to LPBK High		t_{60}	ns	(Note 4)

NOTES:
1. MOS Levels.
2. Manchester Only.
3. Manchester. Needs 50% duty cycle.
4. 1 TTL Load + 50 pF.
5. NRZ only.
6. Abnormal End of Transmission: $\overline{CTS}$ expires before $\overline{RTS}$.
7. Normal End of Transmission.
8. Programmable value:
   $t_{76} = N_{IFS} \times t_{60}$
   N_{IFS}: the IFS configuration value.
   If N_{IFS} is less than 12 then N_{IFS} is enforced to 12.
9. Programmable Value:
   $t_{79} = N_{CDF} \times t_{60} + (12 \text{ to } 15) \times t_{60}$ (if collision occurs after preamble).
   N_{CDF}: The collision detect filter configuration value.
10. $t_{80} = 32 \times t_{60}$
11. Programmable Value:
    $t_{83} = N_{CDF} \times t_{60} + (12 \text{ to } 15) \times t_{60}$
12. To guarantee recognition on the next clock.

TRANSMIT DATA WAVEFORMS

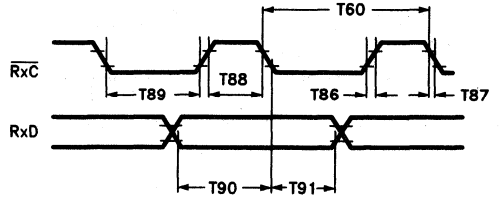


290147-38



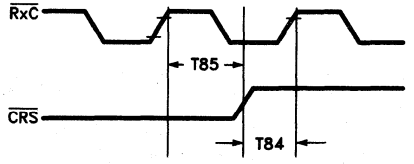
290147-39

RECEIVE DATA WAVEFORMS (NRZ)



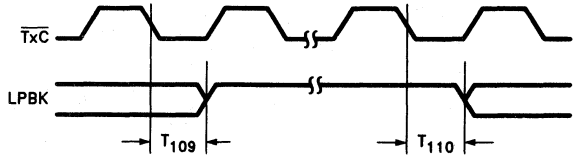
290147-40

RECEIVE DATA WAVEFORMS



290147-41

LOOPBACK OUTPUT SIGNAL TIMINGS



290147-42

82592
ADVANCED CSMA/CD LAN CONTROLLER WITH 16-BIT DATA PATH

- Supports Industry Standard LANs
  - StarLAN (IEEE 802.3 1BASE5)
  - IBM* PC Network—Baseband and Broadband
  - Ethernet and Cheapernet (IEEE 802.3 10BASE5 and 10BASE2)
  - Broadband Ethernet (IEEE 802.3 10BROAD36)
- Integrates Physical and Data Link Layers of OSI Model
  - Complete CSMA/CD Medium Access Control (MAC) Functions
  - Manchester, Differential Manchester, or NRZI Encoding/Decoding
  - On-Chip, Logic-Based Collision Detection
  - IEEE 802.3 or HDLC Frame Delimiting
- Two Modes of Operation
  - Bit Rates Up to 4 Mb/s with On-Chip Encoder/Decoder (High-Integration Mode)
  - Bit Rates Up to 20 Mb/s with External Encoder/Decoder (High-Speed Mode)
- High-Performance System Interface
  - 16-MHz Clock, 2 Clocks per Transfer
  - 64 Bytes of Configurable FIFO
- Efficient Memory Use via Buffer and Frame Chaining
- DMA Interface for Retransmission and Continuous Reception Without CPU Intervention
  - $\overline{\text{EOP}}$ Signal Generation for 8237 and 82380
  - Tightly Coupled Interface to 82560 Host Interface and Memory Manager
- Supports 8- or 16-Bit DMA Transfers
- Local and Remote Power-Down Modes
- Deterministic Collision Resolution
- 24-Bit General Purpose Timer
- On-Chip Jabber Inhibit Function
- Network Management and Diagnostics
  - Monitor Mode
  - CRC, Alignment, and Short Frame Error Detection
  - Three 16-Bit Event Counters
  - Short or Open Circuit Localization
  - Self-Test Diagnostics
  - Internal and External Loopback Operation
  - Internal Register Dump
- High-Speed CHMOS III Technology

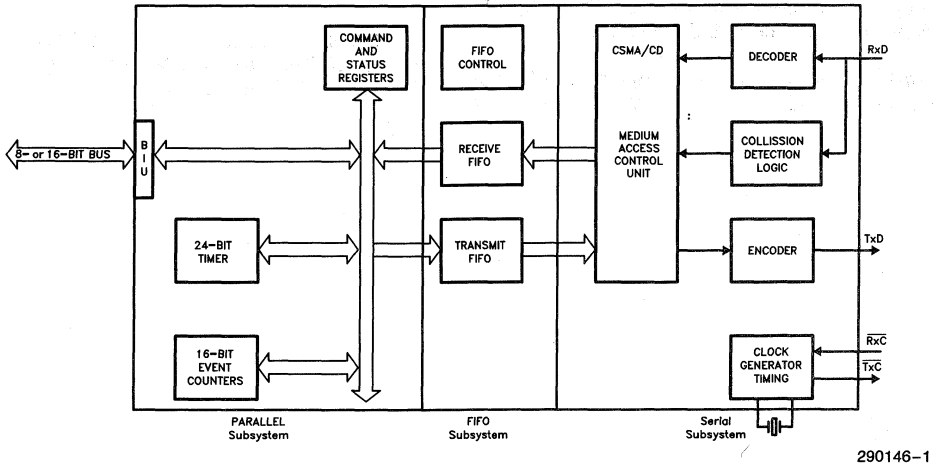


Figure 1. 82592 Block Diagram

*IBM is a trademark of International Business Machines Corporation.

March 1988
Order Number: 290146-002

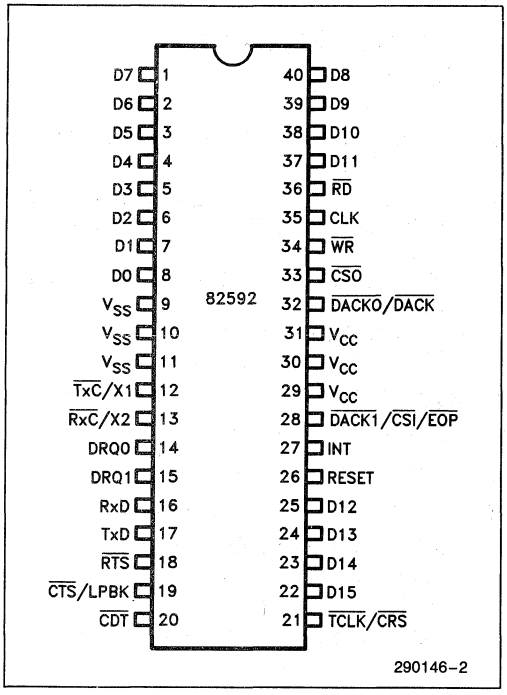


Figure 2. 82592 Pin Configuration (DIP)

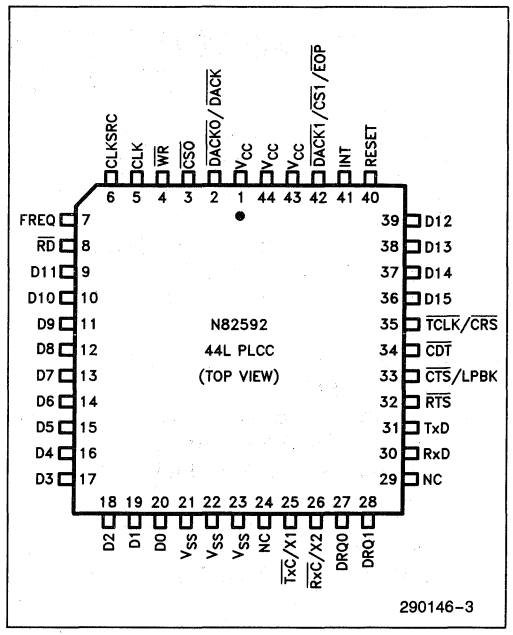


Figure 3. 82592 Pin Configuration (PLCC)

The 82592 is a second-generation, 16-bit data path CSMA/CD controller. Its system interface enables efficient operation with a wide variety of Intel microprocessors (e.g., 80186, 80286, or 80386) and industry standard buses (such as the IBM PC I/O channel or Personal System/2 Micro Channel). The 82592 can be configured to support a wide variety of industry standard networks, including StarLAN, IBM PC Network, and Ethernet/Cheapernet.

Together with the 82560 (Host Interface and Memory Manager) the 82592 offers a complete solution for CSMA/CD LAN adapters oriented to the IBM PC environment. The 82592 fully conforms to existing IEEE 802.3 standards (1BASE5, 10BASE5, 10BASE2, and 10BROAD36). To implement a complete Ethernet/Cheapernet solution, Intel offers the 82C501 Ethernet Serial Interface and the 82502 Ethernet Transceiver chip. Intel also offers the 82590, an 8-bit data path version of the 82592. The 82590 is pin and software compatible with the 82588.

The 82592 is available in a 40-pin Plastic DIP or a 44-pin PLCC package. It is fabricated with Intel's reliable CHMOS III technology.

Table 1. Pin Description

Symbol	Pin No. (DIP)	Pin No. (PLCC)	Type	Name and Function
D15 D14 D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0	22 23 24 25 37 38 39 40 1 2 3 4 5 6 7 8	36 37 38 39 9 10 11 12 13 14 15 16 17 18 19 20	I/O	**DATA BUS:** The Data Bus lines are bidirectional, three-state lines connected to the CPU's Data Bus for transfers of data, commands, status, and parameters.
$\overline{RD}$	36	8	I	**READ:** Together with $\overline{CS0}$, $\overline{CS1}$, $\overline{DACK0}$, or $\overline{DACK1}$, Read controls data or status transfers out of the 82592.
$\overline{WR}$	34	4	I	**WRITE:** Together with $\overline{CS0}$, $\overline{CS1}$, $\overline{DACK0}$, or $\overline{DACK1}$, Write controls data or command transfers into the 82592.
$\overline{CS0}$	33	3	I	**CHIP SELECT (PORT 0):** When LOW, the 82592 is selected by the CPU for command or status transfer through PORT 0.
RESET	26	40	I	**RESET:** A HIGH signal on this pin causes the 82592 to terminate current activity. This signal is internally synchronized and must be held HIGH for at least four Clock (CLK) cycles. When the Clock signal is provided internally (CLKSRC is strapped HIGH), the RESET signal must be held HIGH for at least 50 μs. (PLCC version only.)
INT	27	41	O	**INTERRUPT:** A HIGH signal on this pin notifies the CPU that the 82592 is requesting an interrupt.
DRQ0	14	27	O	**DMA REQUEST (CHANNEL 0):** This pin is used by the 82592 to request DMA transfer. DRQ0 remains HIGH as long as the 82592 requires DMA transfers. Burst transfers are thus possible. When the 82592 is programmed for Tightly Coupled DMA Interface, the 82592 notifies the DMA controller of the status of transmission or reception, using this pin together with $\overline{EOP}$.
DRQ1	15	28	O	**DMA REQUEST (CHANNEL 1):** This pin is used by the 82592 to request DMA transfer. DRQ1 remains HIGH as long as the 82592 requires DMA transfers. Burst transfers are thus possible. When the 82592 is programmed for Tightly Coupled DMA Interface, the 82592 notifies the DMA controller of the status of transmission or reception, using this pin together with $\overline{EOP}$.

Table 1. Pin Description (Continued)

Symbol	Pin No. (DIP)	Pin No. (PLCC)	Type	Name and Function
$\overline{DACK0}$/ $\overline{DACK}$	32	2	I	**DMA ACKNOWLEDGE (CHANNEL 0):** When LOW, this input signal from the DMA controller notifies the 82592 that the requested DMA cycle is in progress. This signal acts similarly to Chip Select for data and parameter transfers, using DMA channel 0. **DMA ACKNOWLEDGE (CHANNELS 0 AND 1):** When the $\overline{DACK1}$/$\overline{CS1}$/$\overline{EOP}$ pin is programmed to $\overline{CS1}$/$\overline{EOP}$, this pin provides a DMA acknowledge for both channels 0 and 1. Two DMA acknowledge signals from the DMA controller, $\overline{DACK0}$ and $\overline{DACK1}$, must be externally ANDed in this mode of operation.
$\overline{DACK1}$ $\overline{CS1}$/$\overline{EOP}$	28	42	I I/O	This is a multifunction, bidirectional pin which can be programmed to $\overline{DACK1}$ or $\overline{CS1}$/$\overline{EOP}$ during configuration. When it is configured for $\overline{EOP}$, it provides an open-drain output. **DMA ACKNOWLEDGE (CHANNEL 1):** When LOW, this input signal from the DMA controller notifies the 82592 that the requested DMA cycle is in progress. This signal acts similarly to Chip Select for data and parameter transfers, using DMA channel 1. **CHIP SELECT (PORT 1):** When LOW, the 82592 is selected by the CPU for command or status transfer through PORT 1. **END OF PROCESS:** A LOW output signal requests the DMA controller to terminate the active DMA service.
CLK	35	5	I I/O	**CLOCK:** In the 40-pin DIP, this is only an input pin. A TTL-compatible clock input to this pin provides the timing for the 82592 parallel subsystem. In the 44-pin PLCC, this pin can be a clock input or output, depending on the state of CLKSRC. If CLKSRC is strapped LOW, this pin is a clock input which provides timing for the 82592 parallel subsystem. If CLKSRC is strapped HIGH, the clock for the 82592 parallel subsystem is generated from the internal clock generator. The CLK pin is then a clock output and provides a clock signal whose frequency can be one-half of, or identical to, the frequency of the internally generated parallel subsystem clock, depending on the state of FREQ. Note that the maximum frequency of the clock signal supplied by the CLK pin is 8 MHz.

CLKSRC	FREQ	CLK		Clock for the Parallel Subsystem
		Type	Signal	
0 (LOW)	Don't Care	I	Clock	Clock as provided on the CLK pin.
1 (HIGH)	1	O	Internal Parallel Subsystem Clock Divided by Two	Prescaled clock generated from the internal clock generator.
1	0	O	Internal Parallel Subsystem Clock	Prescaled clock generated from the internal clock generator

CLKSRC	NA	6	I	**CLOCK SOURCE:** When strapped LOW, a clock signal on the CLK pin provides timing for the parallel subsystem. When strapped HIGH, timing for the parallel subsystem is internally generated from the clock generator provided in the serial subsystem. The internal prescaler is programmed during configuration to determine the frequency of the clock for the parallel subsystem.

Table 1. Pin Description (Continued)

Symbol	Pin No. (DIP)	Pin No. (PLCC)	Type	Name and Function
FREQ	NA	7	I	**FREQUENCY:** When strapped LOW, CLK has an output freqency equal to that of the parallel subsystem clock. When strapped HIGH, CLK has an output frequency one-half that of the parallel subsystem clock. The state of this pin is relevant only when CLKSRC is strapped HIGH.
X1/X2	12/13	25/26	I	**HIGH INTEGRATION MODE OSCILLATOR INPUTS:** These inputs may be used to connect a quartz crystal which controls the internal clock generator for the serial subsystem. When CLKSRC is strapped HIGH, the clock generator also provides a clock for the parallel subsystem. X1 may also be driven by a MOS-level clock whose freqency is 8, 10, 16, or 18 times the bit rate of Transmit/Receive data. X2 must be left floating if X1 is connected to an external MOS clock.
$\overline{TxC}$	12	25	I	**HIGH SPEED MODE TRANSMIT CLOCK:** This signal provides the fundamental timing for the serial subsystem. The clock is also used to transmit data synchronously on the TxD pin. For NRZ encoding, data is transferred to the TxD pin on the HIGH to LOW clock transition. For Manchester encoding, the transmitted bit center is aligned with the LOW to HIGH transition.
$\overline{RxC}$	13	26	I	**RECEIVE CLOCK:** This clock is used to synchronously sample data on the RxD pin. Only NRZ data format is supported for reception. The state of the RxD pin is sampled on the HIGH to LOW transition.
$\overline{TCLK}/\overline{CRS}$	21	35	I O	**CARRIER SENSE:** In High-Speed Mode this pin is Carrier Sense, $\overline{CRS}$, and is used to notify the 82592 that the serial link is active. **TRANSMIT CLOCK:** In High-Integration Mode this pin is Transmit Clock, $\overline{TCLK}$.
$\overline{CDT}$	20	34	I	**COLLISION DETECT:** This input notifies the 82592 that a collision has occurred. In High-Speed Mode a collision is sensed by this pin only when the 82592 is configured for external Collision Detect (external means are then required for collision detection). In High-Integration Mode collisions are sensed by this pin regardless of the internal or external Collision Detect configuration of the 82592.
RxD	16	30	I	**RECEIVE DATA:** This pin receives serial data. It must be HIGH when not receiving.
TxD	17	31	O	**TRANSMIT DATA:** This pin transmits data to the serial link. It is HIGH when not transmitting.
$\overline{RTS}$	18	32	O	**REQUEST TO SEND:** When this signal is LOW the 82592 notifies the channel that it has data to transmit. It is forced HIGH after a reset or when transmission is stopped.
$\overline{CTS}$/LPBK	19	33	I/O	**CLEAR TO SEND:** An active LOW signal which enables the 82592 to start transmitting data. Asserting this signal HIGH stops the transmission. **LOOPBACK:** This pin, in conjunction with a pull-down resistor, can be programmed to provide an active HIGH loopback signal to the external interface device.
V_{CC}	29 30 31	1 43 44		**POWER:** +5V ±10%.
V_{SS}	9 10 11	21 22 23		**GROUND:** 0V.

FUNCTIONAL DESCRIPTION

Internal Architecture

The 82592 consists of a parallel subsystem, a serial subsystem, and a FIFO subsystem (see Figure 1).

PARALLEL SUBSYSTEM

The parallel subsystem consists of a bus interface unit (BIU), command and status registers, a 24-bit general purpose timer, and three 16-bit event counters.

The BIU provides an 8- and/or 16-bit interface to the external system bus. It handles all data transfers to and from memory (at speeds up to 16 Mbytes/sec.), accepts commands from the CPU, and provides status to the CPU. There are two separate 8-bit I/O ports, Port 0 and Port 1; and two separate 8- or 16-bit DMA channels, Channel 0 and Channel 1. The 8-bit I/O ports are interfaced to the CPU via the data lines D_0–D_7. The DMA channels can be configured for an 8- or 16-bit data path during initialization, and are typically interfaced to an external DMA controller. When the 82592 is reset by hardware or software, the DMA channels are initialized for an 8-bit data path. The CPU can then configure the 82592 for a 16-bit data path if desired. Once the DMA channels are configured for a 16-bit data path all subsequent DMA transfers are performed on the data lines D_0–D_{15}. The two DMA channels are independent and can be used for high-performance operations such as simultaneous transmission and reception.

The 24-bit timer consists of a 24-bit maximum count register, a 24-bit count register, and associated control bits in the command registers. Its clock source can be the transmit clock or the parallel subsystem clock. The timer can be programmed to halt or continue on a terminal count with or without causing an interrupt.

The three 16-bit event counters can be programmed to count valid frames, collided frames, and errored (CRC or Alignment) frames. When these event counters are used in Monitor mode, the 82592 is capable of maintaining the network statistics by itself; i.e., without requesting DMA services or causing interrupts to the CPU.

SERIAL SUBSYSTEM

The serial subsystem consists of a CSMA/CD unit, a data encoder and decoder, collision detect and carrier sense logic, and a clock generator.

The 82592's CSMA/CD unit is highly flexible in implementing the CSMA/CD protocol. It can operate in a variety of IEEE 802.3 and other CSMA/CD LAN environments, including 1BASE5 (StarLAN), 10BASE5 (Ethernet), 10BASE2 (Cheapernet), and the IBM PC Network (Baseband and Broadband). The programmable parameters include:

- Framing (IEEE 802.3 Framing or HDLC Framing)
- Address Field Length
- Station Priority
- Interframe Spacing
- Slot Time
- CRC-32 or CRC-16

The CSMA/CD unit also has a mode of operation which implements deterministic collision resolution (DCR). The DCR algorithm is fully compatible with the MULTIBUS™ II Serial System Bus (SSB) specifications.

The encoder and decoder in the serial subsystem is capable of NRZI, Manchester, and Differential Manchester encoding and decoding at bit rates up to 4 Mb/s in High-Integration Mode, and Manchester encoding at bit rates up to 20 Mb/s in High-Speed Mode. A digital phase-lock loop is used in High-Integration Mode to decode the receive data and to generate the synchronous receive clock.

The collision detect and carrier sense logic generate the internal collision detect and carrier sense signals for the CSMA/CD unit.

The 82592 implements several different internal, logic-based collision detect mechanisms. Two of these, Code Violation and Bit Comparison, are also implemented in the 82588 (8-bit NMOS High Integration LAN Controller), and have been used in a variety of applications. The Code Violation method defines a collision where a transition edge occurs outside the area of normal transitions (as specified by the data encoding method). For example, if there are no mid-bit cell transitions in the Manchester encoded data, this method interprets that condition as a collision. The Bit Comparison method compares the signature of the transmitted frame to the signature of the received frame. If the signatures are different, a collision is assumed to have occurred. Two other internal collision detect methods are Source Address Comparison and StarLAN CPS (Collision Presence Signal) Recognition. The Source Address Comparison compares the source address field of the transmitted frame to the source address field of the received frame. If the source addresses are different, it assumes that a collision has occurred resulting in data corruption in the source address field. The StarLAN CPS Recognition method looks for the specific collision presence signal defined by the

IEEE 802.3 1BASE5 standard. Other abnormal circumstances, such as no carrier for more than one-half slot time in the receive channel during transmission, are interpreted as collisions by the 82592.

In addition to these internal, logic-based collision detection methods, an external means of collision detection can be used in parallel by using the $\overline{CDT}$ input pin.

The clock generator in the serial subsystem is available only in High-Integration Mode and provides timing for the serial subsystem. The clock signal can also be routed to the parallel subsystem, if so desired. The oscillator circuit is designed for use with an external, parallel resonant, fundamental mode crystal. The crystal frequency should be selected at 8X, 10X, 16X, or 18X the required serial bit rate.

FIFO SUBSYSTEM

The FIFO subsystem is located between the parallel subsystem and the serial subsystem. It consists of a transmit FIFO, a receive FIFO, and FIFO control logic. The transmit and receive FIFOs are independent of each other and individually provide optimal interfaces between the two subsystems which may have different speeds. There is a total of 64 bytes that can be used for the two separate FIFOs. During configuration these 64 bytes can be divided into one of four possible combinations: 16 and 16 bytes, 16 and 48 bytes, 32 and 32 bytes, or 48 and 16 bytes for the transmit and receive FIFO respectively. The FIFO threshold is also programmed during configuration.

Programming Model—Register Overview

Figure 4 shows the 82592 internal registers that are directly accessible through the 8-bit I/O ports: Port 0 and Port 1.

Figure 5 shows the Port 0 commands, and Figure 6 shows the Port 1 commands. The two separate I/O ports can be accessed at two different addresses selected by $\overline{CS0}$ and $\overline{CS1}$, or at one address selected by $\overline{CS0}$. When the hardware does not support two chip select signals, port switch commands are used to access both ports alternately at one address. If the SWT-TO-PORT-1 command is executed while in Port 0, the port logically becomes Port 1. Software overhead associated with port switching is eliminated if two chip select signals are supplied in hardware.

The 82592 can be configured to have 4 or 6 bytes of status registers in Port 0 (see Figures 4 and 7). When configured to 4 bytes of status registers the first three status registers (STATUS 0 through 2) contain the information about the last command executed or the last frame received. The last status register, STATUS 3, contains the state of the 82592. When the 82592 is configured to 6 bytes of status registers, the two additional bytes are used to report a more complete status of the most recently received frame.

The status of the timer and event counters is available in the Port 1 status registers as shown in Figure 8.

82592 and Host Interaction

The CPU interacts with the 82592 through the system's memory and the 82592's on-chip registers. The CPU creates a data structure in memory, programs the external DMA controller with the start address and byte count of the memory block, and issues a command to the 82592.

The chip select and interrupt lines are used to communicate between the 82592 and the CPU as shown in Figure 9. The interrupt signal is used by the 82592 to attract the CPU's attention. The chip select signal is used by the CPU to attract the 82592's attention. Note that the 82592 does not have any address lines.

There are two kinds of transfers over the bus: command/status and data transfers. The 8-bit command/status transfers are always performed by the CPU. The 8- or 16-bit data transfers are requested by the 82592, and are usually performed by a DMA controller. Table 2 shows the command/status and data-transfer control signals. The CPU writes commands to the 82592 using the $\overline{CS0}$ (or $\overline{CS1}$) and $\overline{WR}$ signals, and reads status using the $\overline{CS0}$ (or $\overline{CS1}$) and $\overline{RD}$ signals. When data transfers are performed, $\overline{DACK0}$ or $\overline{DACK1}$ must be asserted by the DMA controller instead of the Chip Select.

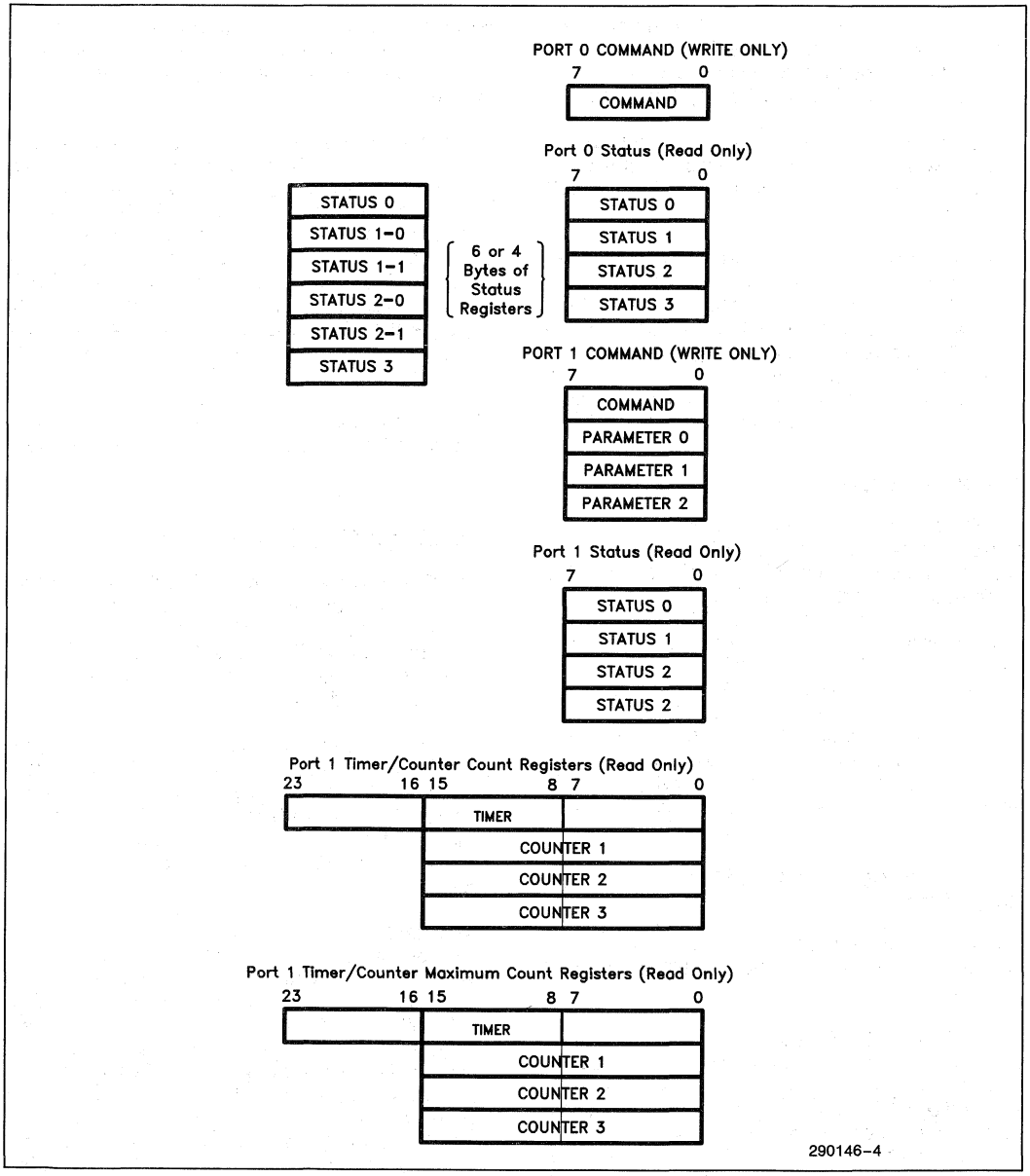


Figure 4. Programming Model—Directly Accessible Registers (Accessible Through 8-Bit I/O Port[s])

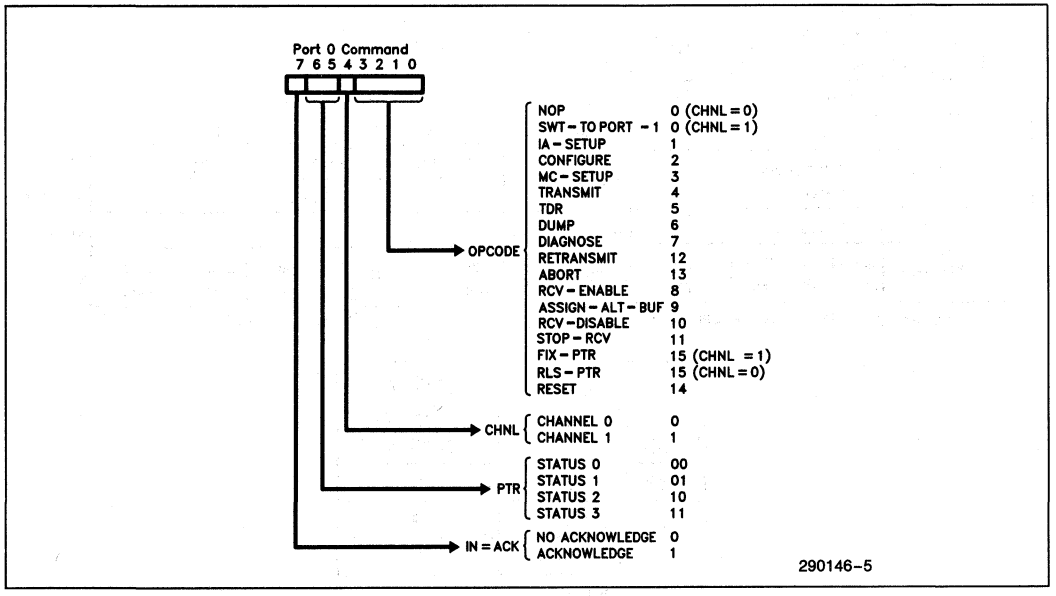


Figure 5. Port 0 Commands

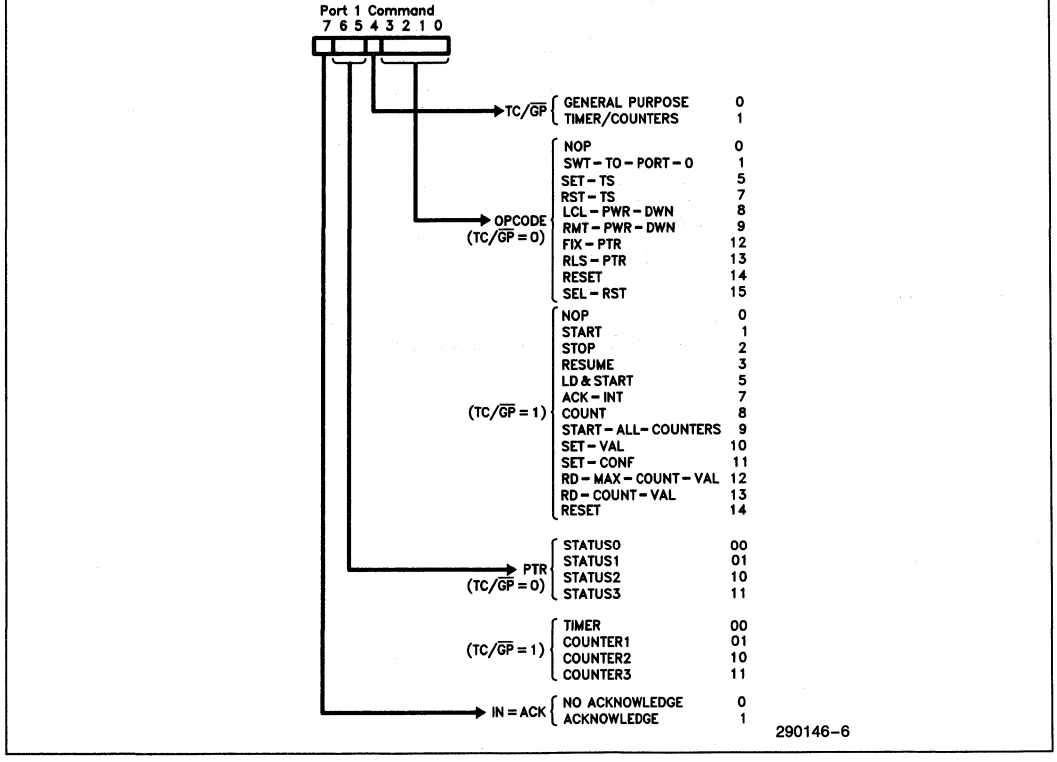
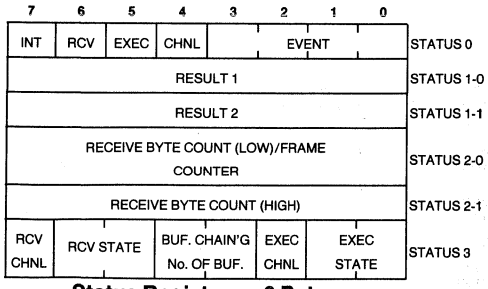
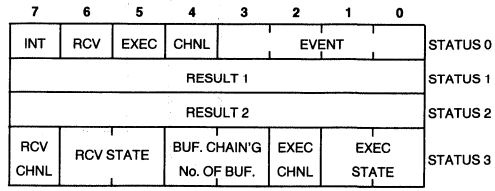


Figure 6. Port 1 Commands



Status Registers—6 Bytes

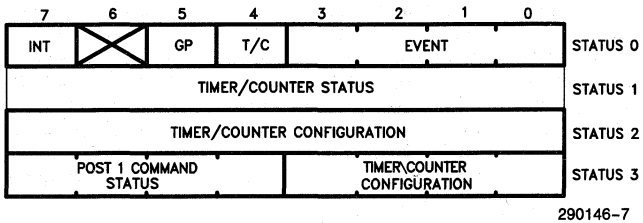


Status Registers—4 Bytes

Events	Value (Status 0)
CMOS*	0 (CHNL = 1)
IA-Setup-Done	1
Configure-Done	2
MC-Setup-Done	3
Transmit-Done	4
TDR-Done	5
Dump-Done	6
Diagnose-Passed	7
End-of-Frame	8
Request-Next-Buffer	9
Reception-Aborted	10
Retransmit-Done	12
Execution-Aborted	13
Diagnose-Failed	15

*Available only after Hardware or Software Reset.

Figure 7. Port 0 Status Registers



Timer/Counter Events (T/C = 1)	Value* (Status 0)
Timer Expired	Bit 0 = 1
Counter 1 Expired	Bit 1 = 1
Counter 2 Expired	Bit 2 = 1
Counter 3 Expired	Bit 3 = 1
General Purpose Event (GP = 1)	**Value* (Status 0)**
REM-PWR-UP	9

*The 82592 may have more than one EVENT bit set by the time the CPU reads the Status register.

Figure 8. Port 1 Status Registers

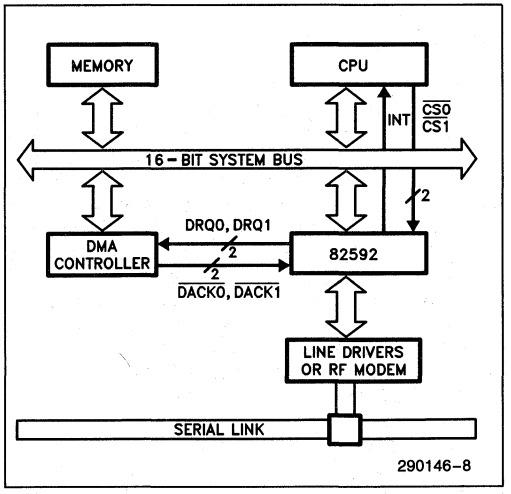


Figure 9. 82592/Host CPU Interaction

Table 2. Data Bus Control Signals and Functions

Pin Name			Function
$\overline{CS0}$ $\overline{CS1}$*	$\overline{RD}$	$\overline{WR}$	
1	X	X	No Transfer to/from Command/Status
0	1	1	
0	0	0	Illegal
0	0	1	Read from Status Register
0	1	0	Write to Command Register
$\overline{DACK0}$ $\overline{DACK1}$*	$\overline{RD}$	$\overline{WR}$	
1	X	X	No DMA Transfer
0	1	1	
0	0	0	Illegal
0	0	1	Data Read from DMA Channel 0 (or 1)
0	1	0	Data Write to DMA Channel 0 (or 1)

*Only one of $\overline{CS0}$, $\overline{CS1}$, $\overline{DACK0}$, or $\overline{DACK1}$ may be active at any time.

To initiate an operation such as Transmit or Configure (see Figure 5), the command from the CPU must first be written to the 82592. Any parameters or data associated with the command are transferred from memory to the 82592 using DMA. Upon completion of the operation, the 82592 updates the appropriate status registers and sends an interrupt to the CPU.

Frame Transmission

To transmit a frame, the CPU prepares a Transmit Data Block in memory as shown in Figure 10. Its first two bytes specify the length of the rest of the block. The next few bytes (up to six) contain the destination address of the station the frame is being sent to. The rest of the block is the data field. The CPU programs the DMA controller with the start address of the block, length of the block, and other control information and then issues a Transmit command to the 82592. Upon receiving this command, the 82592 fetches the first two bytes of the block to determine its length. If the link is free and the first data byte was fetched, the 82592 begins transmitting the preamble and concurrently fetches more bytes from the Transmit Data Block and loads them into the transmit FIFO to keep them ready for transmission.

The destination address is transmitted after the preamble. This is followed by the source or the station individual address, which was previously stored in the 82592 by the IA-Setup command. After this, the entire information field is transmitted, followed by a CRC field calculated by the 82592. If a collision is encountered during transmission of the frame, then the transmission is aborted after a jam pattern is sent. If the collision is detected during preamble or SFD (Start Frame Delimiter) transmission, the 82592 transmits the jam pattern after the SFD is transmitted. An interrupt is then generated to inform the CPU of the unsuccessful transmission due to a collision.

The CPU reinitializes the DMA controller and issues a Retransmit command to the 82592. Retransmission is done by the CPU exactly as the Transmit command is done, except the Retransmit command keeps track of the number of collisions encountered. When the 82592 gets the Retransmit command and the backoff timer is expired, it transmits the frame again. Retransmission is repeated until the attempt is successful, or until the preprogrammed retry number expires.

If the 82592 is programmed to generate the $\overline{EOP}$ signal to the 8237 or 82380 DMA controller, or if it is used with a DMA controller which implements the Tightly Coupled Interface, retransmission is performed without CPU intervention.

Frame Reception

The 82592 can receive frames when its receiver has been enabled. The 82592 checks for an address match for an Individual address, a Multicast address, or a Broadcast address. In the Promiscuous mode the 82592 receives all frames. When the address match is successful, the 82592 transfers the frame to memory using the DMA controller. Before enabling the receiver, it is the CPU's responsibility to make a memory buffer area available to the receiver and to properly program the starting address of the DMA controller. The received frame is transferred to the memory buffer in the format shown in Figure 11. This method of reception is called Single Buffer reception; the entire frame is contained in one continuous buffer. Upon completion of reception, the status of the reception is appended at the end of the received frame in the memory buffer, and the total number of bytes transferred to the memory buffer is loaded into the internal status registers 1 and 2. An interrupt is then generated to inform the CPU of the frame reception.

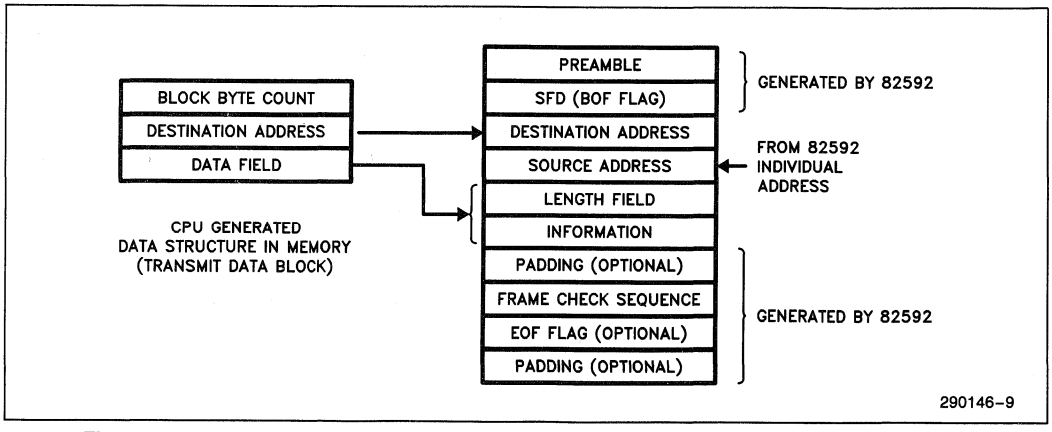


Figure 10. The 82592 Frame Structure and Location of Data Element in System Memory

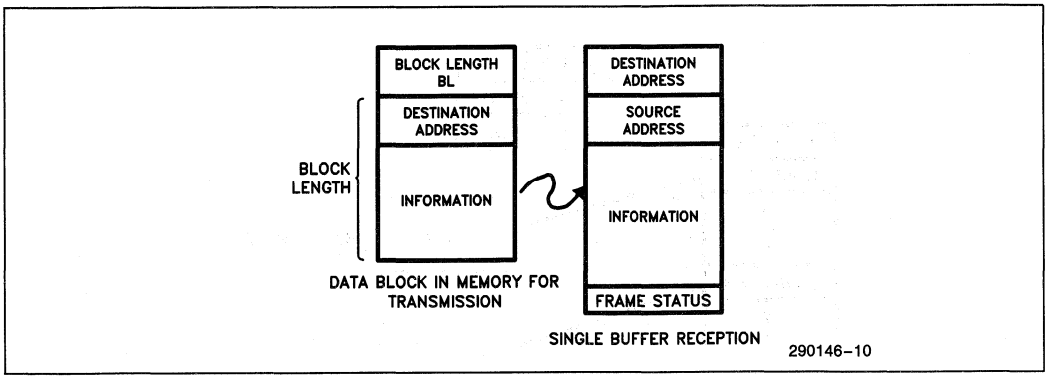


Figure 11. Single Buffer Reception

If the frame size is unknown, memory usage can be optimized by using Multiple Buffer reception. In this mode of operation, the CPU and DMA controller can dynamically allocate memory space as it receives frames. This method requires both DMA channels to receive the frame alternately. As frame reception begins, the 82592 interrupts the CPU and automatically requests assignment of the next available buffer. The CPU does this and loads the second DMA channel with the next buffer's information so the 82592 can immediately switch to the other channel when the current buffer is full. When the 82592 switches from the first to the second buffer it again interrupts the CPU and requests another buffer to be allocated on the previous channel. This process continues until the entire frame is received. The received frame is spread over multiple memory buffers. The link between the buffers is easily maintained by the CPU, using a buffer chain descriptor structure in memory as shown in Figure 12. This dynamic allocation of memory buffers results in efficient use of available storage when handling frames of widely differing sizes.

If the 82592 is programmed to generate the $\overline{EOP}$ signal to the 8237 or 82380 DMA controller, or if it is used with a DMA controller which implements the Tightly Coupled Interface, buffer reclamation and more advanced data structures for the buffer area can significantly improve system performance.

$\overline{EOP}$ Signal to the DMA Controller

The 82592 can be programmed to assert the $\overline{EOP}$ signal to the 8237 or 82380 DMA controller when one or more of the following occurs:

- A collision during transmission
- An error (CRC or alignment) during reception
- A good frame reception

If the 8237 or 82380 is programmed for Auto-initialize mode and if the 82592 is programmed to assert the $\overline{EOP}$ signal on a collision during transmission, the retransmission following a collision is done automatically by the 8237 and the 82592. The 8237 will reinitialize itself automatically and the 82592 will retransmit the same frame from the same memory area without CPU intervention. When the 82592 is programmed for this mode it does not interrupt the CPU upon a collision, and the CPU does not need to issue a Retransmit command to the 82592. The CPU is interrupted only after a successful transmission or retransmission, or after a transmission failure, such as DMA underrun.

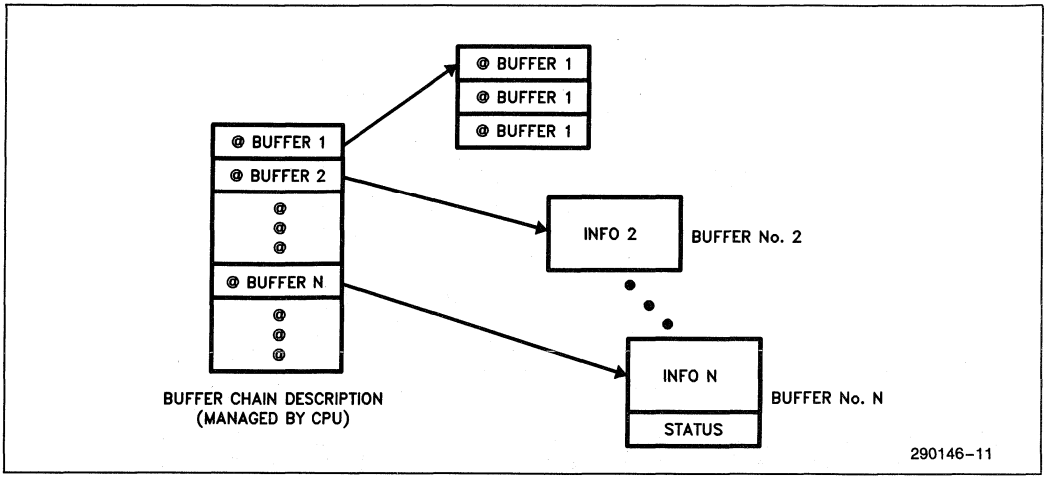


Figure 12. Multiple Buffer Reception

If the 82592 is programmed to assert the $\overline{EOP}$ signal when an error occurs during reception, the 8237 or the 82380 in Auto-initialize mode will be able to reclaim the memory area which would otherwise be wasted for the errored frame reception. If the 82592 is programmed to assert $\overline{EOP}$ at the end of a frame reception, automatic buffer switching can be accomplished by alternating the DMA channels with the 8237 or the 82380. When the 82380 is used, the buffer switching can be done with only one DMA channel.

The $\overline{EOP}$ signal must be derived from the $\overline{DACK1}$/$\overline{CS1}$/$\overline{EOP}$ pin using external logic (see Figure 13).

82592/82560 Tightly Coupled Interface

The 82592 has a mode of operation called "Tightly Coupled Interface." In this mode the 82592 provides a tightly coupled interface to a DMA controller in order to execute some of the time-critical processes of the CSMA/CD protocol without any CPU intervention. By using the 82592's companion chip, the 82560, or by implementing the Tightly Coupled Interface in a DMA controller, operations such as automatic retransmission, continuous back-to-back frame reception, and transmit and/or receive buffer chaining can be accomplished.

The 82592 provides the status of the current active transmission or reception to the DMA controller by using the DRQ and $\overline{EOP}$ signals at the end of every DMA cycle. The status is encoded according to Table 3. As long as the 82592 generates DRQ High and $\overline{EOP}$ Floating at the rising edge of $\overline{RD}$ or $\overline{WR}$, the DMA controller repeats DMA transfers. If the transmission is completed without collisions or if the reception is good (no collision, no CRC, or no Alignment error), then DRQ and $\overline{EOP}$ both become Low at the end of a DMA transfer which follows the last DMA data transfer. If the transmission encountered a collision or if the reception had an error, DRQ becomes High and $\overline{EOP}$ becomes Low. The DMA controller must decode these signals appropriately and must reinitialize the DMA channel so it can retransmit the same frame or reclaim the otherwise wasted buffer. It is the DMA controller's responsibility to reprogram itself for the next appropriate operation.

The 82560 fully implements the Tightly Coupled Interface and provides very-high-performance DMA services for the 82592 with minimal CPU involvement.

Network Management and Diagnostics

The 82592 provides a large set of diagnostic and network management functions including: internal and external loopback, monitor mode, optional capture of all frames regardless of destination address (Promiscuous mode), and time domain reflectometry for locating fault points in the network cable. The 82592 Dump command ensures software reliability by dumping the contents of the 82592 internal registers into the system memory.

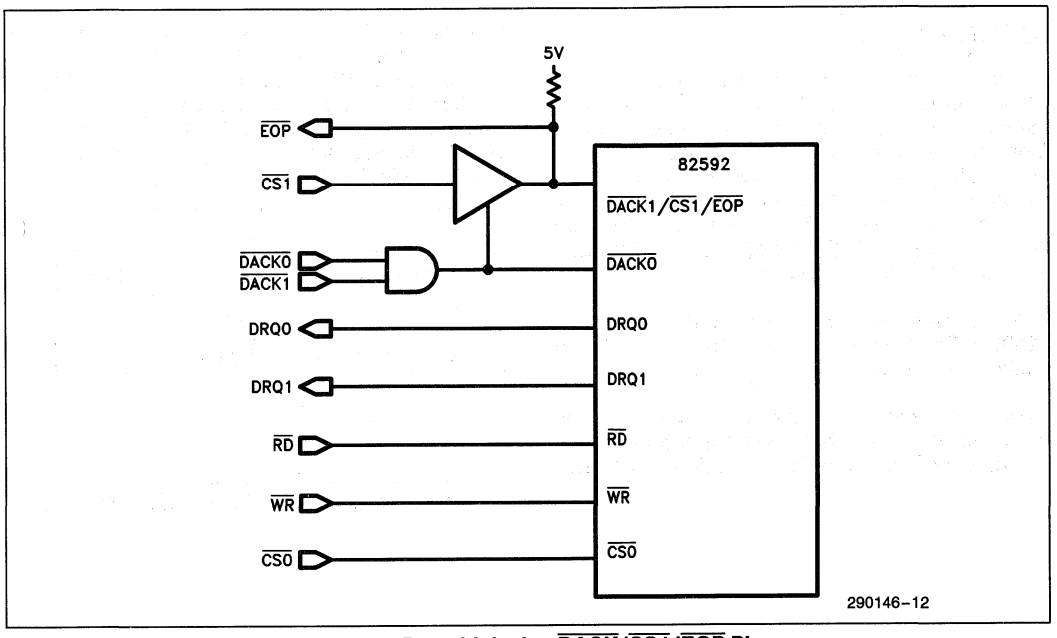


Figure 13. Demultiplexing DACK/CS1/EOP Pin

Table 3. Transmit/Receive Status Encoding on DRQ and EOP

DRQ	EOP	Status Information
0	Hi-Z	Idle
1	Hi-Z	DMA Transfer
0	0	Transmission or Reception Terminated OK
1	0	Transmission or Reception Aborted

Other Enhancements

Compared to the previous generation LAN controllers such as the 82588, the 82592 has many functional and performance enhancements. This section lists some of these enhancements which are not covered in other sections.

1. **Multi-IA**—The 82592 implements multiple-individual address (Multi-IA) filtering. It can receive more than one IA frame in this mode.
2. **Power Down Modes**—Two power down modes, Local Power Down and Remote Power Down, are available. When the 82592 is in Remote Power Down mode, it can be powered up remotely by sending a special frame to it.
3. **Automatic Padding and IEEE 802.3 Length Field**—If a frame to be transmitted is shorter than the configured Slot Time, the 82592 automatically appends pad bytes up to the shortest frame greater than the Slot Time. If the data field of a received frame is longer than the byte count indicated in the Length field, the extra bytes are stripped automatically according to the Length field. Erroneous conditions are detected and reported by the 82592. An example of such conditions is reception of a frame which is shorter than the byte count indicated in the Length field.
4. **Automatic Retransmission on Collision During Preamble**—The 82592 can be programmed to retransmit automatically if it detects a collision during transmission of the preamble.
5. **On-Chip Jabber Inhibit Function**—The 82592 can be programmed to provide an on-chip jabber inhibit function.
6. **CRC Transfer to Memory**—The 82592 can be programmed to transfer the CRC field of a received frame into memory.
7. **Loopback Signal to the 82C501**—The 82592 can be programmed to provide an active High loopback signal to the 82C501 (see Figure 14).
8. **StarLAN**—The 82592 can be configured to recognize the IEEE 802.3 1BASE5 Collision Presence Signal (CPS). In this mode it also delays deactivation of the RTS signal at the end of a frame transmission in order to insert an end-of-frame marker as required by the standard.

APPLICATIONS

The 82592 can be used in a variety of applications. When it is used in High-Integration Mode it implements most of the Data Link and Physical Layer functions required by the IEEE 802.3 1BASE5 (StarLAN) and the IBM PC Network (Baseband and Broadband). When it is used in High-Speed Mode it can work with the 82C501 and the 82502 for IEEE 802.3 10BASE5 (Ethernet) and 10BASE2 (Cheapernet) implementations.

If the desired network requires determinism, the 82592's Deterministic Collision Resolution (DCR) method can be used.

Figure 15 shows a block diagram of an 82592/82560 Ethernet/Cheapernet adapter board. The 82560 provides the Tightly Coupled DMA Interface for the 82592 and dual-port memory control for the static RAM. The 82592 is interfaced to the 82C501 to provide the Ethernet channel and then to the 82502 to provide the Cheapernet channel. Due to the CMOS process used for these chips, such a board uses much less power than a board based on NMOS or bipolar chips.

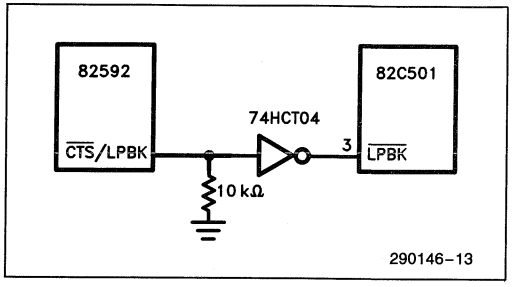


Figure 14. Loopback Output to the 82C501

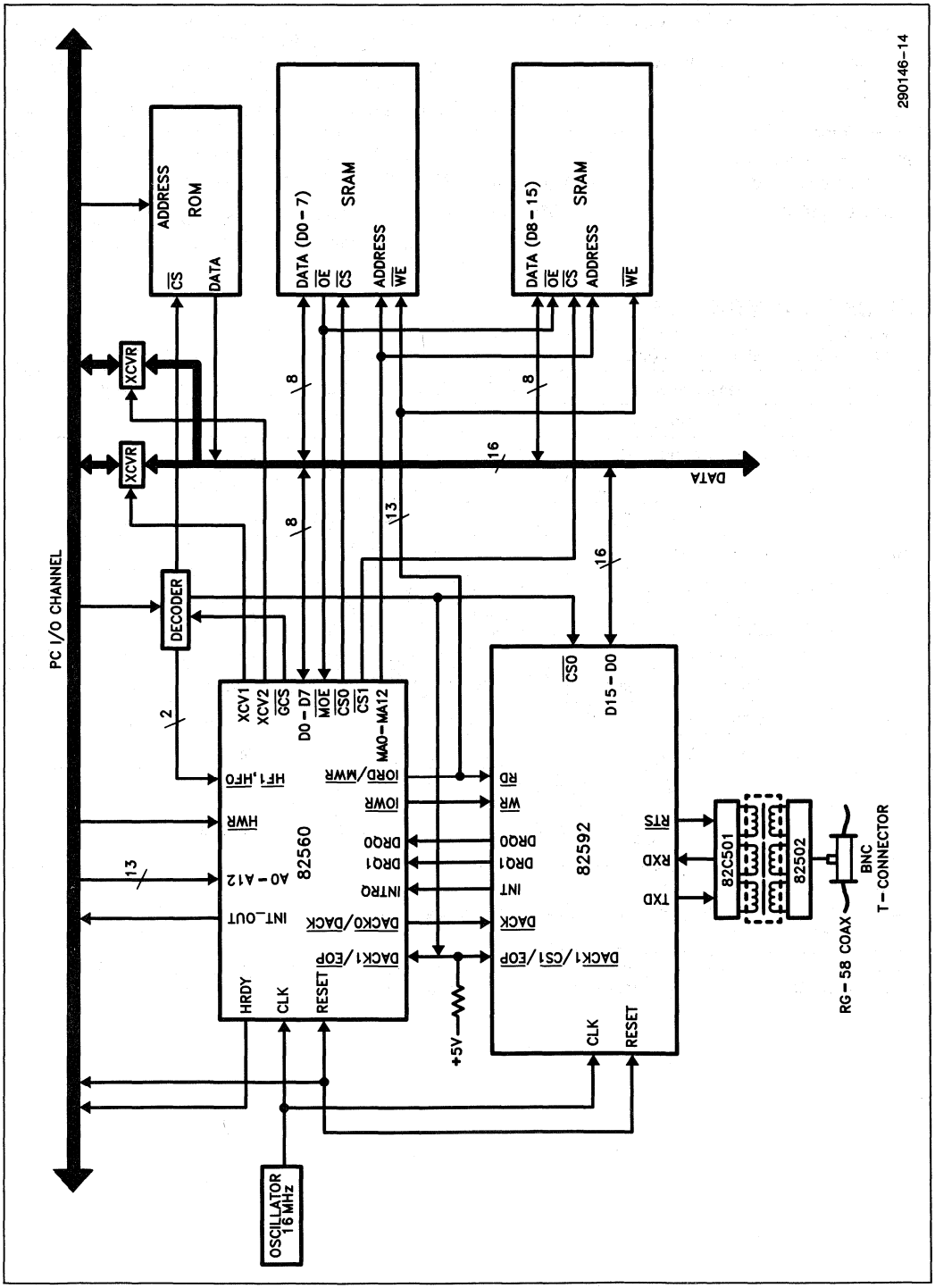


Figure 15. 82592/82560 Ethernet/Cheapernet Adapter

ABSOLUTE MAXIMUM RATINGS*

Case Temperature (T_C) Under Bias
- Plastic ..... 0 to +85°C
- PLCC ..... 0 to +85°C

Storage Temperature ..... −65°C to +150°C

Voltage on Any Pin with Respect to Ground ..... −1V to +7V

Power Dissipation ..... 550 mW

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS

(T_C [Plastic] = 0 to +85°C, T_C [PLCC] = 0 to +85°C, V_{CC} = +5V ±10%)
TxC and RxC have MOS levels (see V_{MIL}, V_{MIH}). All other signals have TTL levels (see V_{IL}, V_{IH}, V_{OL}, V_{OH}).

Symbol	Parameter	Min	Max	Units	* Test Conditions
V_{IL}	Input Low Voltage (TTL)	−0.5	0.8	V	
V_{IH}	Input High Voltage (TTL)	2.0	V_{CC} +0.5	V	
V_{OL}	Output Low Voltage (TTL)		0.45	V	I_{OL} = 2.0 mA
V_{OH}	Output High Voltage (TTL)	2.4		V	I_{OH} = −400 μA
V_{MIL}	Input Low Voltage (MOS)	−0.5	0.6	V	
V_{MIH}	Input High Voltage (MOS)	3.9	V_{CC} +0.5	V	
I_{LI}	Input Leakage Current		±10	μA	0V ≤ V_{IN} ≤ V_{CC}
I_{LIO}	I/O Leakage Current		±10	μA	0.45V ≤ V_{OUT} ≤ V_{CC} −0.45V
C_{IN}	Capacitance of Input Buffer		10	pF	Frequency = 1 MHz
C_{IO}	Capacitance of I/O Buffer		20	pF	Frequency = 1 MHz
I_{CC}	Power Supply Current		100	mA	(Note 1)
			5	mA	(Note 2)
			1	mA	(Note 3)
					(Note 4)

ADVANCE INFORMATION
SEE INTEL FOR DESIGN-IN INFORMATION

NOTES:
1. System side (CLK) at 16 MHz, serial side ($\overline{TxC}$) at 20 MHz in High-Speed Mode.
2. Remote power-up mode.
3. Power-down mode—all inputs connected to V_{CC} level.
4. General formulas for current are: (a) f(CLK) × 4 + f($\overline{T \times C}$) × 2 for High Speed Mode. (b) f(CLK) × 4 + f($\overline{TFC}$) × 0.3 + f($\overline{TFC}$) × 32/SR, where SR is the sampling rate in High-Integration Mode.

A.C. CHARACTERISTICS C_L on all outputs is 20–125 pF unless otherwise specified.

Symbol	Parameter	Min	Max	Units	Test Conditions
SYSTEM CLOCK INPUT PARAMETERS					
t_1	CLK Cycle Time	62.5		ns	
t_2	CLK Low Time	30		ns	(Note 5)
t_3	CLK High Time	30		ns	(Note 5)
t_4	CLK Rise Time		5	ns	(Note 1)
t_5	CLK Fall Time		5	ns	(Note 2)
SYSTEM CLOCK OUTPUT PARAMETERS					
t_{98}	CLK Cycle Time	120		ns	(Notes 4, 7)
t_{99}	CLK Low Time	50		ns	(Note 4)
t_{100}	CLK High Time	50		ns	(Note 4)
t_{101}	CLK Rise Time		15	ns	(Notes 1, 4)
t_{102}	CLK Fall Time		15	ns	(Notes 2, 4)
RESET PARAMETERS					
t_6	Reset Active to Clock Low	20		ns	(Note 3)
t_8	Reset Pulse Width	$4t_1$		ns	
t_9	Control Inactive after Reset		125	ns	
INTERRUPT PARAMETERS					
t_{10}	CLK High to Interrupt Active		55	ns	(Note 4)
t_{11}	$\overline{WR}$ Inactive to Interrupt		55	ns	(Note 4)
t_{103}	Int Low to Int High Gap	$2t_1$		ns	(Note 4)
WRITE PARAMETERS					
t_{12}	$\overline{CS0}$, $\overline{CS1}$, $\overline{DACK0}$, or $\overline{DACK1}$ Setup to $\overline{WR}$ Low	0		ns	
t_{13}	$\overline{WR}$ Pulse Width	55		ns	
t_{14}	$\overline{CS0}$, $\overline{CS1}$, $\overline{DACK0}$, or $\overline{DACK1}$ Hold After $\overline{WR}$ High	0		ns	
t_{15}	Data Setup to $\overline{WR}$ High	30		ns	
t_{16}	Data Hold After $\overline{WR}$ High	0		ns	
t_{94}	Write Cycle Time	$2t_1$		ns	
t_{96}	$\overline{WR}$ Inactive Time	55		ns	

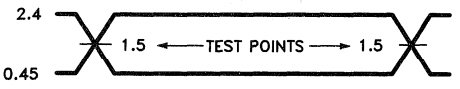
A.C. CHARACTERISTICS C_L on all outputs is 20–125 pF unless otherwise specified. (Continued)

Symbol	Parameter	Min	Max	Units	Test Conditions
READ PARAMETERS					
t_{17}	$\overline{CS0}$, $\overline{CS1}$, $\overline{DACK0}$ or $\overline{DACK1}$ Setup to $\overline{RD}$ Low	0		ns	
t_{18}	$\overline{RD}$ Pulse Width	55		ns	
t_{19}	$\overline{CS0}$, $\overline{CS1}$, $\overline{DACK0}$, or $\overline{DACK1}$ Hold after $\overline{RD}$ High	0		ns	
t_{20}	$\overline{RD}$ Low to Data Valid		45	ns	
t_{21}	Data Float after $\overline{RD}$ High		40	ns	
t_{95}	Read Cycle Time	$2t_1$		ns	
t_{97}	$\overline{RD}$ Inactive Time	55		ns	
DMA PARAMETERS					
t_{22}	CLK Low to DRQ0 or DRQ1 Active		55	ns	(Note 4)
t_{23}	$\overline{WR}$ or $\overline{RD}$ Low to DRQ0 or DRQ1 Inactive		45	ns	(Note 4)
t_{104}	$\overline{WR}$ or $\overline{RD}$ Low to DRQ0 or DRQ1 Inactive—82560 Retransmit or Receive Buffer Reclaim	2.5	65	ns	(Note 4)
t_{105}	$\overline{WR}$ or $\overline{RD}$ Low to $\overline{EOP}$ Active		45	ns	(Note 6)
t_{106}	$\overline{EOP}$ Float after $\overline{DACK0}$ or $\overline{DACK1}$ Going Inactive		40	ns	(Note 6)

NOTES:
1. 0.8V to 2.0V.
2. 2.0V to 0.8V.
3. To guarantee recognition on the next clock.
4. C_L = 50 pF.
5. Measured at 1.5V.
6. Open drain I/O pin.
7. None of the A.C. parameters are related to the CLK output pin.

A.C. TESTING INPUT/OUTPUT WAVEFORMS

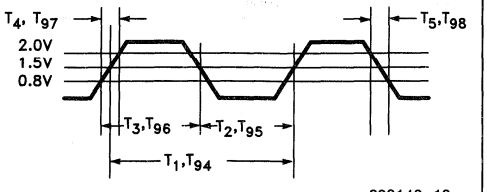
TTL Input/Output Voltage Levels for Timing Measurements



290146-17

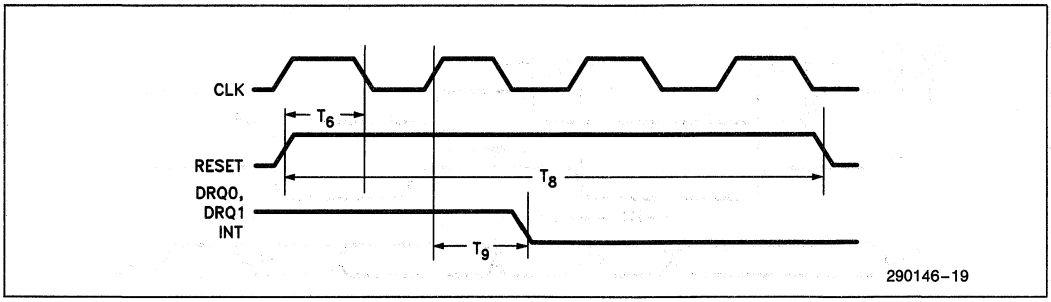
A.C. testing inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing measurements are made at 1.5V for both a Logic "1" and "0". Rise and fall times are measured between 0.8V and 2.0V.

System Clock I/O Voltage Levels for Timing Measurements

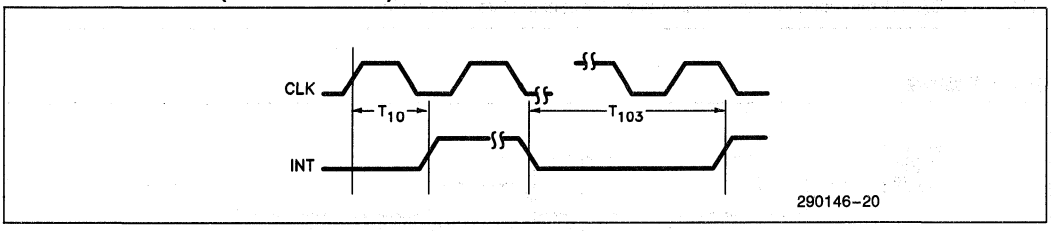


290146-18

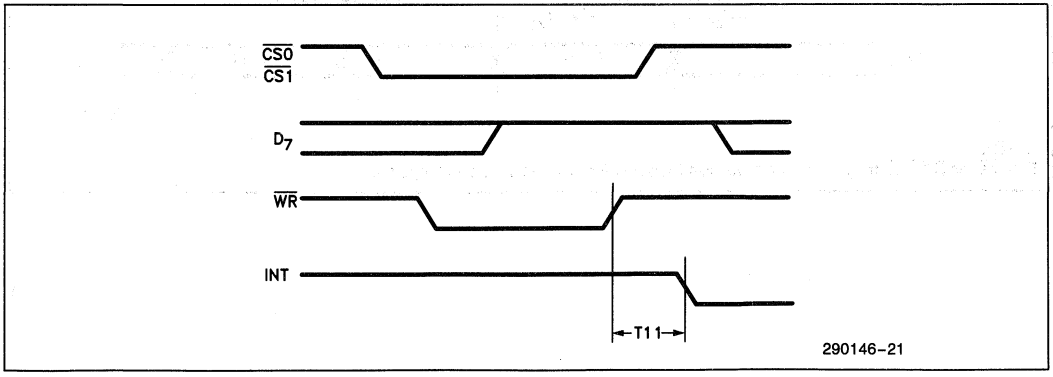
RESET TIMING



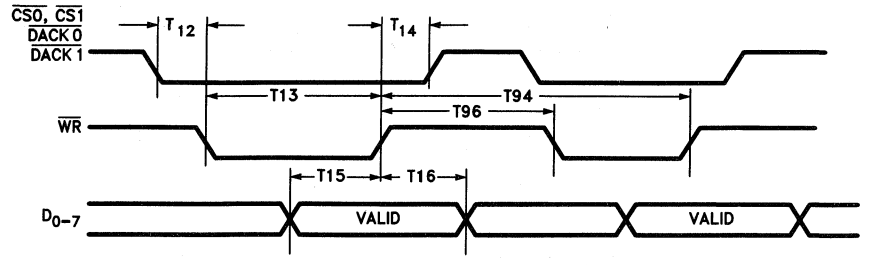
INTERRUPT TIMING (GOING ACTIVE)



INTERRUPT TIMING (GOING INACTIVE)



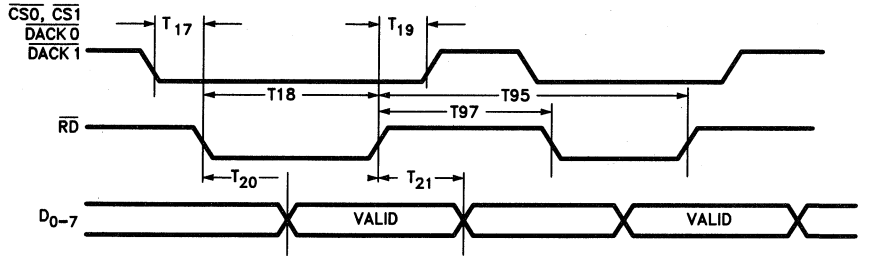
WRITE TIMING



290146-22

NOTE:
DACK0 or DACK1 may remain active continuously for consecutive DMA cycles.

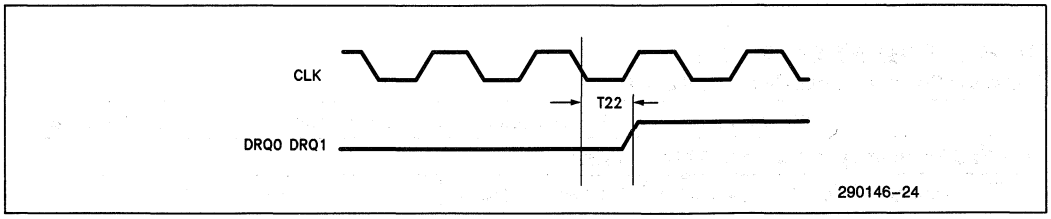
READ TIMING



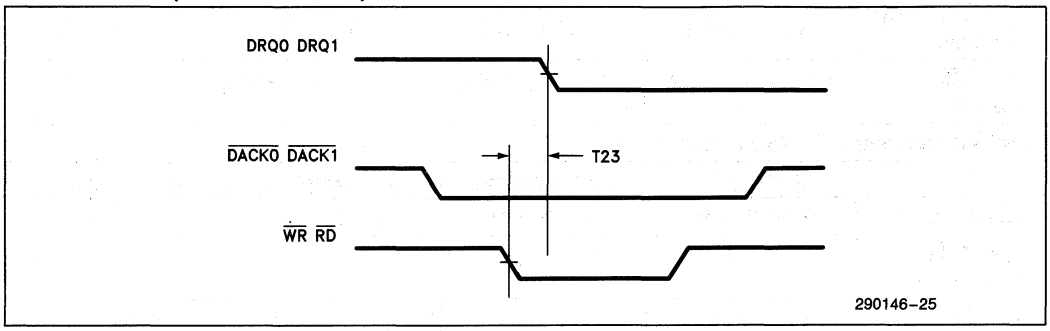
290146-23

NOTE:
DACK0 or DACK1 may remain active continuously for consecutive DMA cycles.

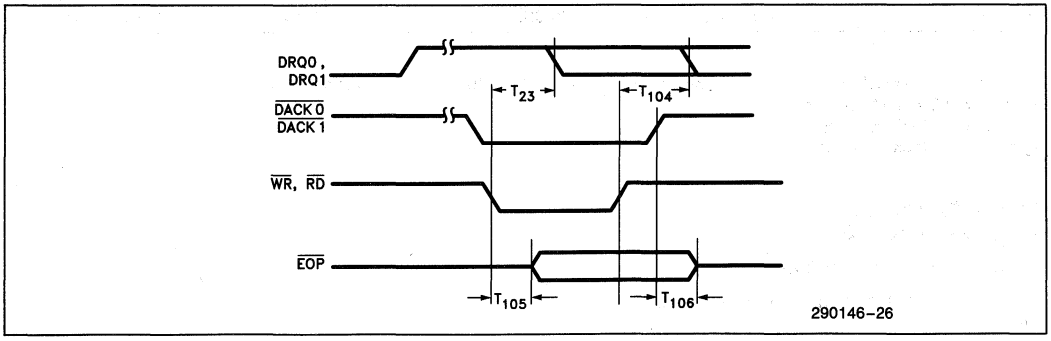
DMA REQUESTS (GOING ACTIVE)



DMA REQUEST (GOING INACTIVE)



TIGHTLY COUPLED INTERFACE



SERIAL INTERFACE A.C. TIMING CHARACTERISTICS

HIGH INTEGRATION MODE

$\overline{TFC}$ is the Crystal or Serial Clock Input at X1.

	X8 Sampling	X10 Sampling	X16 Sampling	X18 Sampling
For $\overline{TFC}$ Frequency = 1 MHz to 32 MHz (High)				
$\overline{TCLK}$ Frequency	0.125 MHz–4 MHz	100 kHz– 3.2 MHz	62.5 kHz–2 MHz	55.6 kHz–1.78 MHz
t_{29} = $\overline{TCLK}$ Cycle Time	8 × t_{24}	10 × t_{24}	16 × t_{24}	18 × t_{24}
t_{30} = $\overline{TCLK}$ High High Time	t_{24} (Typically)	t_{24} (Typically)	t_{24} (Typically)	t_{24} (Typically)
t_{31} = $\overline{TCLK}$ Low Time	7 × t_{24} (Typically)	9 × t_{24} (Typically)	15 × t_{24} (Typically)	17 × t_{24} (Typically)
For $\overline{TFC}$ Frequency = 0 to 1 MHz (Low)				
	X8 Sampling	**X10 Sampling**	**X16 Sampling**	**X18 Sampling**
$\overline{TCLK}$ Frequency	0–0.125 MHz	0–100 kHz	0–62.5 kHz	0–55.6 kHz
t_{29} = $\overline{TCLK}$ Cycle Time	8 × t_{24}	10 × t_{24}	16 × t_{24}	18 × t_{24}
t_{30} = $\overline{TCLK}$ High Time	t_{25} (Typically)	t_{25} (Typically)	t_{25} (Typically)	t_{25} (Typically)
t_{31} = $\overline{TCLK}$ Low Time	7 × t_{24} + t_{26} (Typically)	9 × t_{24} + t_{26} (Typically)	15 × t_{24} + t_{26} (Typically)	17 × t_{24} + t_{26} (Typically)

NOTES:
X10 and X18 are available only for Manchester or Differential Manchester encoding/decoding.
t_{24} = Serial Clock Cycle Time.
t_{25} = Serial Clock High Time.
t_{26} = Serial Clock Low Time.

HIGH SPEED MODE

- Applies for $\overline{TxC}$, $\overline{RxC}$
- f_{max} = 20 MHz ±100 ppm
- For Manchester, symmetry is required:

$t6_4, t_{64} = \frac{1}{2f} \pm 5\%$

HIGH INTEGRATION MODE (Continued)

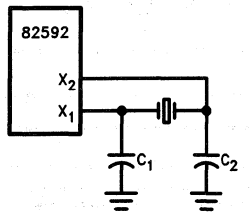
Symbol	Parameter	Min	Max	Units	Test Conditions
EXTERNAL (FAST) CLOCK PARAMETERS					
t_{24}	Fast Clock ($\overline{TFC}$) Cycle Time	31.25		ns	(Notes 1, 16)
t_{25}	$\overline{TFC}$ High Time	(Note 13)		ns	(Note 1, 17)
t_{26}	$\overline{TFC}$ Low Time	12		ns	(Notes 1, 17)
t_{27}	$\overline{TFC}$ Rise Time		3	ns	(Note 1)
t_{28}	$\overline{TFC}$ Fall Time		3	ns	(Note 1)
TRANSMIT CLOCK PARAMETERS					
t_{29}	Transmit Clock ($\overline{TCLK}$) Cycle Time	(Note 13)		ns	(Note 2)
t_{30}	$\overline{TCLK}$ High Time	(Note 7)		ns	(Note 2)
t_{31}	$\overline{TCLK}$ Low Time	(Note 8)		ns	(Note 2)
t_{32}	$\overline{TCLK}$ Rise Time		10	ns	(Note 2)
t_{33}	$\overline{TCLK}$ Fall Time		10	ns	(Note 2)
TRANSMIT DATA PARAMETERS (MANCHESTER, DIFFERENTIAL MANCHESTER)					
t_{34}	TxD Transition-Transition	(Note 14)		ns	
t_{35}	$\overline{TCLK}$ Low to TxD Mid Bit Cell Transition		(Note 10)	ns	(Note 2)
t_{36}	$\overline{TCLK}$ Low to TxD Bit Cell Boundary Transition		(Note 9)	ns	(Note 2)
t_{37}	TxD Rise Time		10	ns	(Note 2)
t_{38}	TxD Fall Time		10	ns	(Note 2)
TRANSMIT DATA PARAMETERS (NRZI)					
t_{39}	TxD Transition-Transition	(Note 15)	ns		
t_{40}	$\overline{TCLK}$ Low to TxD Transition		(Note 9)	ns	(Note 2)
t_{41}	TxD Rise Time		10	ns	(Note 2)
t_{42}	TxD Fall Time		10	ns	(Note 2)
$\overline{RTS}$, $\overline{CTS}$ PARAMETERS					
t_{43}	$\overline{TCLK}$ Low to $\overline{RTS}$ Low		(Note 9)	ns	(Note 2)
t_{44}	$\overline{CTS}$ Low to $\overline{TCLK}$ Low	35		ns	
t_{45}	$\overline{TCLK}$ Low to $\overline{RTS}$ High		(Note 9)	ns	(Note 2)
t_{46}	$\overline{TCLK}$ Low to $\overline{CTS}$ Invalid $\overline{CTS}$ Hold Time	10		ns	(Notes 3, 12)
t_{47}	$\overline{CTS}$ High to $\overline{TCLK}$ Low; $\overline{CTS}$ Setup Time to Stop Transmission	35		ns	(Note 3)
IFS PARAMETERS					
t_{48}	Interframe Delay	(Note 4)		ns	

HIGH INTEGRATION MODE (Continued)

Symbol	Parameter	Min	Max	Units	Test Conditions
COLLISION DETECT PARAMETERS					
t_{49}	$\overline{CDT}$ Low to $\overline{TCLK}$ High External Collision Detect Setup Time	35		ns	(Note 12)
t_{50}	$\overline{CDT}$ High to $\overline{TCLK}$ High	35		ns	(Note 12)
t_{51}	$\overline{TCLK}$ High to $\overline{CDT}$ Inactive $\overline{CDT}$ Hold Time	10		ns	(Note 12)
t_{52}	$\overline{CDT}$ Low to Jamming Start		(Note 5)	ns	
t_{53}	Jamming Period	(Note 6)		ns	
RECEIVED DATA PARAMETERS (MANCHESTER, DIFFERENTIAL MANCHESTER)					
t_{54}	RxD Transition-Transition	115		ns	(Note 14)
t_{55}	RxD Rise Time		10	ns	
t_{56}	RxD Fall Time		10	ns	
RECEIVED DATA PARAMETERS (NRZI)					
t_{57}	RxD Transition-Transition	240		ns	(Note 11)
t_{58}	RxD Rise Time		10	ns	
t_{59}	RxD Fall Time		10	ns	
EXTERNAL LOOPBACK PARAMETERS					
t_{107}	$\overline{TCLK}$ Low to LPBK High		50	ns	(Note 2)
t_{108}	$\overline{TCLK}$ Low to LPBK Float		50	ns	(Note 2)

NOTES:
1. MOS levels.
2. 1 TTL load + 50 pF.
3. Abnormal end of transmission: $\overline{CTS}$ expires before $\overline{RTS}$.
4. Programmable value: $t_{28} = N_{IFS} \times t_{29}$ (ns).
   N_{IFS}: the IFS configuration value.
   If N_{IFS} is less than 12 then it is enforced to 12.
5. Programmable values:
   $t_{52} = N_{CDF} \times t_{29} + (12 \text{ to } 15) \times t_{29}$ (if collision occurs after preamble).
   N_{CDF}: The collision detect filter configuration value.
6. $t_{53} = 32 \times t_{29}$
7. Depends on frequency range:
   High range: $t_{24} - 10$ ns
   Low range: $t_{25} - 10$ ns
8. $t_{31} = t_{29} - t_{30} - t_{32} - t_{33}$
9. $2 \times t_{24} + 40$ ns for 8X or 10X
   $4 \times t_{24} + 40$ ns for 16X
10. $6 \times t_{24} + 40$ ns for 8X
    $12 \times t_{24} + 40$ ns for 16X
    $7 \times t_{24} + 40$ ns for 10X
    $13 \times t_{24} + 40$ ns for 18X
11. $8 \times t_{24} - 10$ ns for 8X
    $10 \times t_{24} - 10$ ns for 10X
    $16 \times t_{24} - 10$ ns for 16X
    $18 \times t_{24} - 10$ ns for 18X
12. To guarantee recognition on the next clock.
13. 10 ns for high range
    30 ns for low range
14. $4 \times t_{24} - 10$ ns for 8X
    $5 \times t_{24} - 10$ ns for 10X
    $8 \times t_{24} - 10$ ns for 16X
    $9 \times t_{24} - 10$ ns for 18X
15. $t_{29} - 10$ ns
16. See Figure "CRYSTAL CONNECTION."
17. Maximum capacitance load on the X_2 pin when an external MOS clock is connected to X_1:
    15 pF for DC to 16 MHz
    5 pF for 16 MHz to 32 MHz

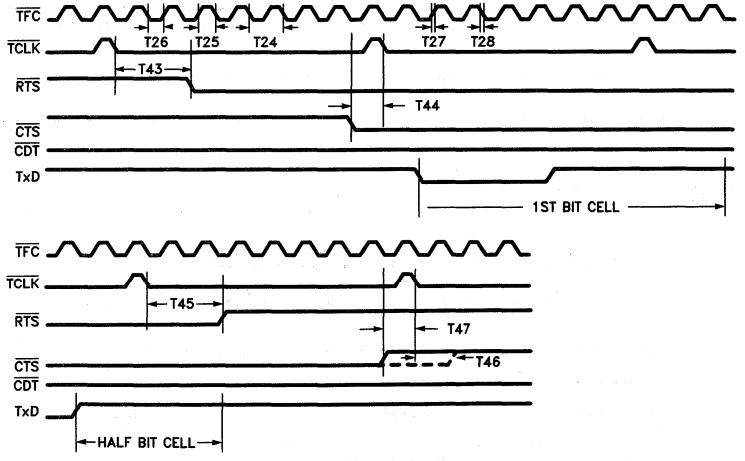
CRYSTAL CONNECTION



290146-27

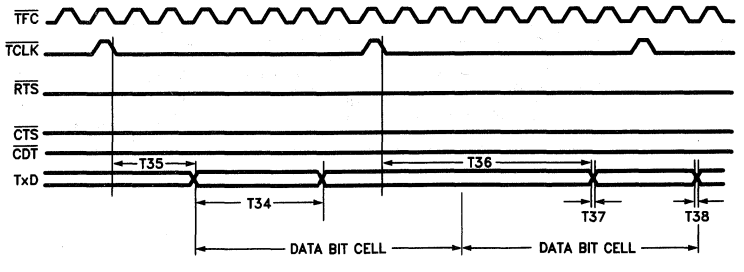
NOTES:
1. High-quality, parallel resonant, fundamental-mode crystals are recommended for maximum accuracy.
2. C1, C2, and stray capacitance of the board should be adjusted so the total capacitance load on the crystal is approx. 15 pF.
3. For IEEE 802.3 applications, the crystal must be accurate to ±35 PPM over a range of 0°C to 70°C.

TRANSMIT TIMINGS—CLOCKS $\overline{RTS}$ AND $\overline{CTS}$



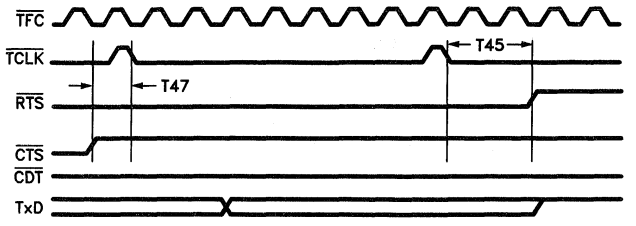
290146-28

TRANSMIT TIMINGS—MANCHESTER DATA ENCODING

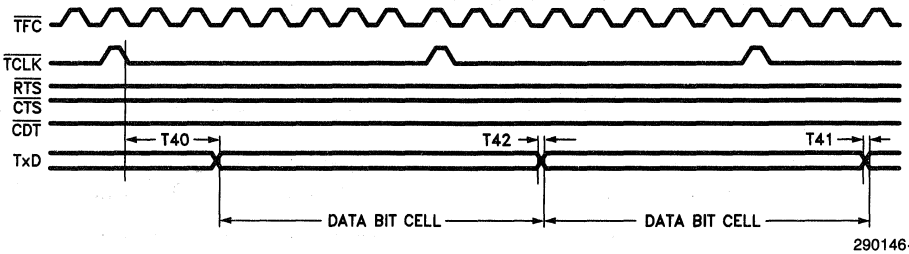


290146-29

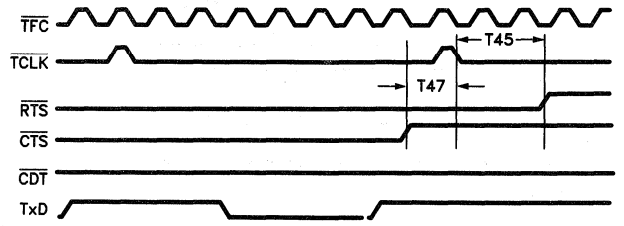
TRANSMIT TIMINGS—LOST $\overline{CTS}$



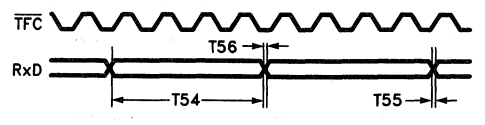
TRANSMIT TIMINGS—NRZI DATA CODING



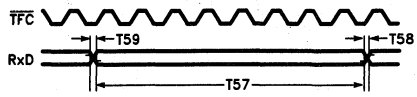
TRANSMIT TIMINGS—LOST $\overline{CTS}$



RECEIVE DATA TIMINGS (MANCHESTER)

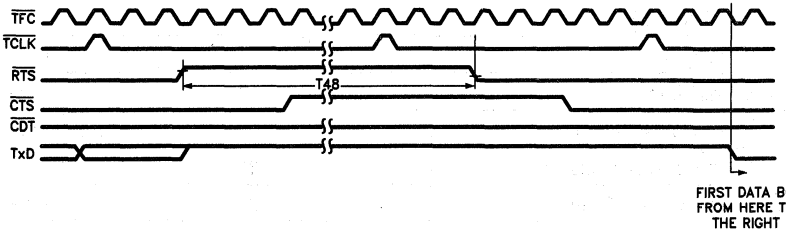


RECEIVE DATA TIMINGS (NRZI)



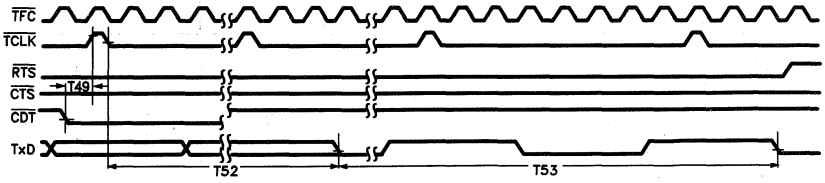
290146-34

TRANSMIT TIMINGS—INTERFRAME SPACING



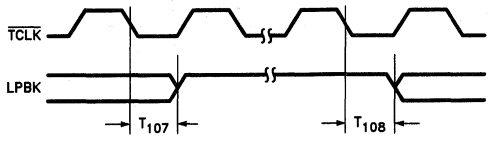
290146-35

TRANSMIT TIMINGS—COLLISION DETECT AND JAMMING



290146-36

LOOPBACK OUTPUT SIGNAL TIMINGS



290146-37

HIGH SPEED MODE

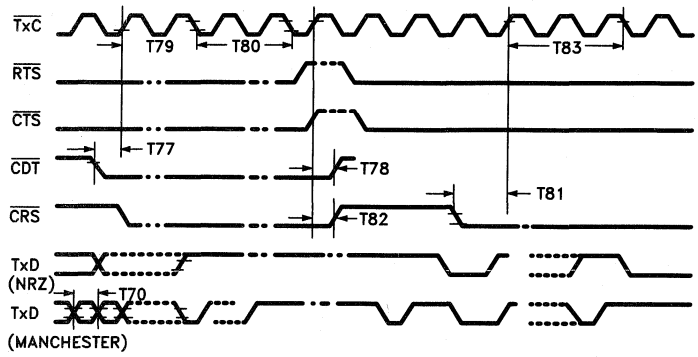
Symbol	Parameter	Min	Max	Units	Test Conditions
TRANSMIT/RECEIVE CLOCK PARAMETERS					
t_{60}	$\overline{RxC}$, $\overline{TxC}$ Cycle Time	50		ns	(Notes 1, 3)
t_{61}	$\overline{TxC}$ Rise Time		5	ns	(Note 1)
t_{62}	$\overline{TxC}$ Fall Time		5	ns	(Note 1)
t_{63}	$\overline{TxC}$ High Time	18		ns	(Notes 1, 3)
t_{64}	$\overline{TxC}$ Low Time	19		ns	(Notes 1, 3)
TRANSMIT DATA PARAMETERS					
t_{65}	TxD Rise Time		10	ns	(Note 4)
t_{66}	TxD Fall Time		10	ns	(Note 4)
t_{67}	$\overline{TxC}$ Low to TxD Valid		25	ns	(Notes 4, 5)
t_{68}	$\overline{TxC}$ Low to TxD Transition		25	ns	(Notes 2, 4)
t_{69}	$\overline{TxC}$ High to TxD Transition		25	ns	(Notes 4, 5)
t_{70}	TxD Transition-Transition	20		ns	(Notes 2, 4)
t_{71}	$\overline{TxC}$ Low to TxD High (At the Transmission End)		25	ns	(Note 4)
$\overline{RTS}$, $\overline{CTS}$ PARAMETERS					
t_{72}	$\overline{TxC}$ Low to $\overline{RTS}$ Low Time to Activate $\overline{RTS}$		25	ns	(Note 4)
t_{73}	$\overline{CTS}$ Low to $\overline{TxC}$ Low $\overline{CTS}$ Setup Time	20		ns	
t_{74}	$\overline{TxC}$ Low to $\overline{RTS}$ High		25	ns	(Note 4)
t_{75}	$\overline{TxC}$ Low to $\overline{CTS}$ Invalid. $\overline{CTS}$ Hold Time	10		ns	(Note 6)
t_{75a}	$\overline{CTS}$ High to $\overline{TxC}$ Low. $\overline{CTS}$ Setup Time to Stop Transmission	20		ns	(Note 6)
INTERFRAME SPACING PARAMETER					
t_{76}	Inter Frame Delay	(Note 8)		ns	

HIGH SPEED MODE (Continued)

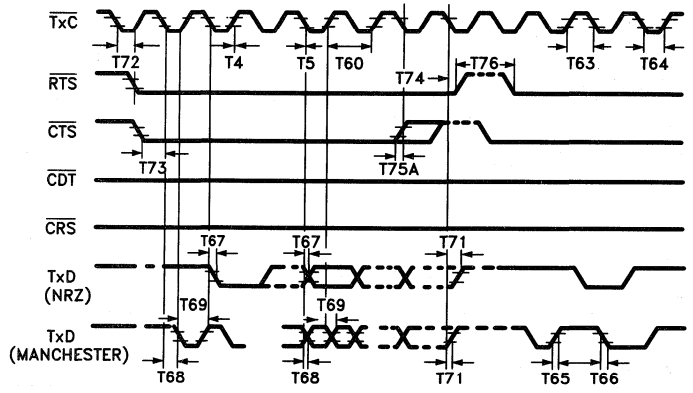
Symbol	Parameter	Min	Max	Units	Test Conditions
$\overline{CRS}$, $\overline{CDT}$ PARAMETERS					
t_{77}	$\overline{CDT}$ Low to $\overline{TxC}$ High. External Collision Detect Setup Time	20		ns	
t_{78}	$\overline{TxC}$ High to $\overline{CDT}$ Inactive. $\overline{CDT}$ Hold Time	10		ns	(Note 12)
t_{79}	$\overline{CDT}$ Low to Jam Start		(Note 9)	ns	
t_{80}	Jamming Period		(Note 10)	ns	
t_{81}	$\overline{CRS}$ Low to $\overline{TxC}$ High. Carrier Sense Setup Time	20		ns	
t_{82}	$\overline{TxC}$ High to $\overline{CRS}$ Inactive. $\overline{CRS}$ Hold Time	10		ns	(Note 12)
t_{83}	$\overline{CRS}$ High to Jamming Start. (Internal Collision Detect)		(Note 11)	ns	
t_{84}	$\overline{CRS}$ High to $\overline{RxC}$ High. $\overline{CRS}$ Inactive Setup Time	30		ns	
t_{85}	$\overline{RxC}$ High to $\overline{CRS}$ High. $\overline{CRS}$ Inactive Hold Time	10		ns	
RECEIVE CLOCK PARAMETERS					
t_{86}	$\overline{RxC}$ Rise Time		5	ns	(Note 1)
t_{87}	$\overline{RxC}$ Fall Time		5	ns	(Note 1)
t_{88}	$\overline{RxC}$ High Time	18		ns	(Note 1)
t_{89}	$\overline{RxC}$ Low Time	19		ns	(Note 1)
RECEIVED DATA PARAMETERS					
t_{90}	RxD Setup Time	15		ns	(Note 5)
t_{91}	RxD Hold Time	15		ns	(Note 5)
t_{92}	RxD Rise Time		10	ns	
t_{93}	RxD Fall Time		10	ns	
EXTERNAL LOOPBACK PARAMETERS					
t_{109}	$\overline{TxC}$ Low to LPBK High		t_{60}	ns	(Note 4)
t_{110}	$\overline{TxC}$ Low to LPBK Float		t_{60}	ns	(Note 4)

NOTES:
1. MOS levels.
2. Manchester only.
3. Manchester. Needs 50% duty cycle.
4. 1 TTL load + 50 pF.
5. NRZ only.
6. Abnormal end of transmission: $\overline{CTS}$ expires before $\overline{RTS}$.
7. Normal end of transmission.
8. Programmable value:
   $t_{76} = N_{IFS} \times t_{60}$
   N_{IFS}: the IFS configuration value.
   If N_{IFS} is less than 12 then NIFS is enforced to 12.
9. Programmable value:
   $t_{79} = N_{CDF} \times t_{60} + (12 \text{ to } 15) \times t_{60}$ (If collision occurs after preamble).
   N_{CDF}: The collision detect filter configuration value.
10. $t_{80} = 32 \times t_{60}$
11. Programmable value:
    $t_{83} = N_{CDF} \times t_{60} + (12 \text{ to } 15) \times t_{60}$
12. To guarantee recognition on the next clock.

TRANSMIT DATA WAVEFORMS

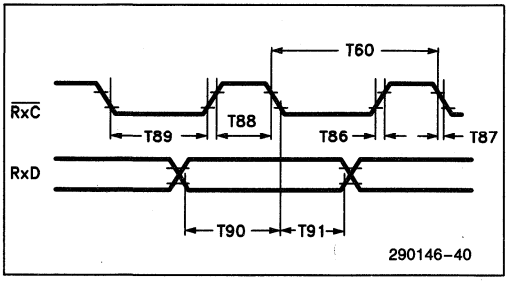


290146-38

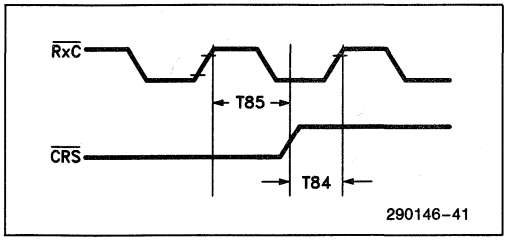


290146-39

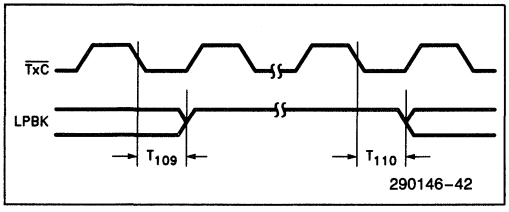
RECEIVE DATA WAVEFORMS (NRZ)



RECEIVE DATA WAVEFORMS



LOOPBACK OUTPUT SIGNAL TIMINGS



Glossary

8

GLOSSARY

Addressing — targets frames to one or more hosts.

ALNERRS — number of misaligned frames received with a CRC error.

ALU — Arithmetic Logic Unit used by the Command Unit and Receive Unit to perform simple arithmetic and logical operations.

ARDY — Asynchronous Ready

Backoff Timer — implements Backoff, Defer, Wait, and Priority algorithms.

BHE — Bus High Enable signal.

Bitstuffing — see zero bit insertion.

BIU — Bus Interface Unit generates or receives all control signals between the 82586 and the local bus.

BOF — Beginning of Frame.

Broadcast Address — all ones destination address. All stations will receive a frame with the Broadcast Address.

CA — Channel Attention signal used by host CPU for 82586 attention calls.

Carrier Sense — indication of activity on the link.

CB — Command Block associated with each Action Command in the Command Unit.

CBL — Command Block Unit.

CDBBC — Collision Detection by Bit Comparison.

Channel Interface Module — the part of the 82586 that handles the network interface.

CLK — a clock signal supplied by the host and used for all Parallel Unit and FIFO timing.

Coaxial Cable Segment — a length of coaxial cable made up from one or more coaxial cable sections and coaxial connectors, terminated at each end in its characteristic impedance. A 500 meter segment is the longest configuration possible without repeaters.

Collision — the result of multiple transmissions overlapping in the physical channel, resulting in garbled data and necessitating retransmission.

Collision Detect — a signal provided by the Physical Layer to the Data Link Layer to indicate that one or more other stations are contending with the local station's transmission. It can be true only during transmission.

CPU — Central Processing Unit.

CRC — Cyclic Redundancy Check. Autodin II (used in IEEE 802.3/Ethernet) 32-bit polynomial is $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$.

CCITT V.41 16-bit polynomial is $X^{16} + X^{12} + X^5 + 1$.

CRCERRS — number of properly aligned frames received with a CRC error.

CS — Chip Select Signal.

CSMA/CD — Carrier Sense Multiple Access with Collision Detection.

CTS — Clear-to-Send signal.

CU — Command Unit. The logical unit that executes Action commands from a list of commands very similar to a CPU program.

CUC — a 3-bit field in SCB containing operation codes specifying Control Commands for the Command Unit.

CUS — a 3 bit field in SCB indicating the current status of the Command Unit. September 1985

Data Interface Unit — a switching matrix which routes data and status signals to the appropriate destinations by interconnecting input and output as required.

Defer — delay pending transmission.

Deference — a process by which a data link controller delays its transmission when the channel is busy to avoid contention with ongoing transmissions.

Destination Address — frame target address.

DMA — Direct Memory Access.

DMA Machine — memory address generator operating on starting addresses and byte counts to transfer information between the 82586 and shared memory.

DSAP — Destination Service Access Point.

Dynamic List Processing — A method for controlling the 82586 where by the 82586's Command and/or Receive Unit remain in the Active State automatically processing commands or frames on a linked list until the end of the list is reached.

EL — End of List bit used by 82586 memory structure to mark the end of a list of commands and buffers.

End of Carrier — Ethernet compatible framing mechanism. The EOF is indicated by dropping the carrier immediately after the FCS field.

EOF — End of Frame.

External Loopback — a method for locating faults anywhere between the 82586 and the link.

FBL — Free Buffer List.

FCS — Frame Check Sequence. A CRC used in detecting bit errors in a received frame.

FD — Frame Descriptor.

FIFO Module — two 16-bit First In First Out storage array situated between the Host Interface Module and the Channel Interface Module.

Frame Check Sequence — see FCS.

Framing — determines the beginning and end of the frame.

82586 Handler — A frame level software module to drive the 82586. This module includes sending and receiving frames, accessing buffers, adding and deleting multicast addresses, initialization, and self test diagnostics. It does not include any Logical Link Control functions.

HASH table — Multicast bits are mapped to one bit of this 64-bit table.

Host Interface Module — the part of the 82586 that handles the bus interface.

IFS — Inter Frame Spacing. Minimum time between two consecutive transmissions or receptions.

Interframe Spacing — the minimum time allowed between two consecutive received frames. This time period allows the Data Link and Physical Link to recover from receiving a message.

Individual Address — an address with the least significant bit = 0.

INT — Interrupt signal from the chip to request the host CPU's attention.

Internal Loopback — the 82586 disconnects itself from the Serial Interface Device and any frame transmitted is received immediately.

ISCP — Intermediate System Configuration Pointer.

Jam — a pattern of extra bits after collision is detected to enforce a collision. The pattern is 32 ones.

LAN — Local Area Network.

LSAP — Link Service Access Point for the data link interface.

Manchester Encoding — a means by which separate data and clock signals can be combined into a single, self-synchronizable data stream, suitable for transmission on a serial channel.

Micro-Instruction ROM — supplies micro-program to the Command and Receive Sequencers.

Multicast Address — an addressing mode used to target a specific group of nodes. A multicast Destination Address is one with the least significant bit = 1.

No-Resources — frames last due to lack of memory resources.

NRZI — Non Return to Zero Inverted. Coding usually used for HDLC schemes.

OK — bit used in action command to designate error free completion.

OVRNERRS — number of received frames lost because the memory bus was not available in time to transfer them.

Padding — HDLC flags added to frames shorter than Slot Time.

Preamble — a sequence of 64 encoded bits which the Physical Layer transmits before each frame to allow synchronization of clocks and other Physical Layer circuitry at other sites on the channel.

Promiscuous Mode — the 82586 captures all frames transmitted on the Network without checking the Destination Address.

Repeater — a device used to extend the length and topology of the physical channel beyond that imposed by a single segment, up to the maximum allowable end-to-end channel length.

RBD — Receive Buffer Descriptors.

RCV-FIFO — Receive FIFO.

$\overline{RD}$ — Read Signal.

Real Address — a single 24-bit entity specifying the physical byte address in memory.

Receive Buffer Chaining — storing incoming frames in a linked list of buffers.

Receive Bit Machine — strips the Preamble and BOF flag from the received frame, determines the End of Frame, deletes zeros inserted by Bitstuffing, computes and verifies the frame CRC.

RFA — Receive Frame Area is a list of Receive Frame Descriptors, (each of which is associated with a frame) and a list of buffer descriptors and their buffers.

RNR — Receive No Resources Interrupt.

RSCERRS — number of received frames discarded due to lack of memory resources.

RTS — Request to Send signal.

RU — Receive Unit. The logical unit that receives frames and stores them in memory.

RUC — a 3-bit field in SCB containing operation codes specifying Control Commands for the Receive Unit.

RUS — a 3-bit field in SCB indicating the current status of the Receive Unit.

SCB — System Control Block. The communication mail-box between the 82586 and the host CPU.

SCP — System Configuration Pointer. The only fixed address in the system.

Segemented Address — is the sum of a 24-bit long real address base and a 16-bit long offset.

SFD — Start of Frame Delimiter.

Slot Time — is the worst case time to detect a collision, i.e. the time it takes for a frame to propagate from one end of the Network to the other, and back again (includes recognition and Jam and retransmission times). The slot time period is used as the time increment (scheduling quantum) for retrys after a collision.

Source Address — senders address.

SRDY — Synchronous Ready.

SSAP — Source Service Access Point.

Static List Processing — A method for controlling the 82586 where by the 82586's Command Unit and Receive Unit goes from the Active state to the Idle state after each command or frame is processed. To have the next command or frame processed, the CPU has to re-start the Command and/or Receive Unit.

TBD — Transmit Buffer Descriptor.

TCLK — Transmit Clock.

TDR — Time Domain Reflectometer. The mechanism used by the 82586 for determining if there is an open or short circuit anywhere in the Network.

Transceiver — transmits and receives the electrical signal to/from the coaxial cable, usually also detects collisions.

Transceiver Cable — a four pair, shielded cable used for the transceiver cable interface.

Transceiver Cable Interface — the electrical, mechanical and logical interface which connects the transceiver to the controller. The standard transceiver cable is a recommended compatibility interface.

TX-FIFO — Transmit FIFO.

Underrun — data transfer rate from shared memory to the 82586 did not keep up with transmission.

$\overline{WR}$ — Write Signal.

Zero Bit Insertion/Deletion (ZBI) — This procedure inserts a zero after any series of five consecutive 1's. These zeros are deleted upon receive. By using ZBI, the HDLC opening/closing flag is unique because it consists of six consecutive 1's: 01111110.

DOMESTIC SALES OFFICES

ALABAMA

†Intel Corp.
5015 Bradford Dr., #2
Huntsville 35805
Tel: (205) 830-4010

ARIZONA

†Intel Corp.
11225 N. 28th Dr.
Suite D-214
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
2nd Floor
Santa Clara 95051
Tel: (408) 986-8086
TWX: 910-338-0255
FAX: 408-727-2620

COLORADO

†Intel Corp.
4445 Northpark Drive
Suite 100
Colorado Springs 80907
Tel: (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
2nd Floor
Danbury 06811
Tel: (203) 748-3130
TWX: 710-456-1199

FLORIDA

†Intel Corp.
6363 N.W. 6th Way, Suite 100
Ft. Lauderdale 33309
Tel: (305) 771-0600
TWX: 510-956-9407
FAX: 305-772-8193

†Intel Corp.
5850 T.G. Lee Blvd.
Suite 340
Orlando 32822
Tel: (305) 240-8000
FAX: 305-240-8097

†Intel Corp.
11300 4th Street North
Suite 170
St. Petersburg 33716
Tel: (813) 577-2413
FAX: 813-578-1607

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.
1930 St. Andrews Drive N.E.
2nd Floor
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.
7833 Walker Drive
Suite 550
Greenbelt 20770
Tel: (301) 441-1020

MASSACHUSETTS

†Intel Corp.*
Westford Corp. Center
3 Carlisle Road
2nd Floor
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
FAX: 201-740-0626

NEW MEXICO

†Intel Corp.
8500 Menaul 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
FAX: 607-723-2677

†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.
15 Myers Corner Road
Suite 25
Hollowbrook Park
Wappinger Falls 12590
Tel: (914) 297-6161
TWX: 510-248-0060

NORTH CAROLINA

†Intel Corp.
5700 Executive 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 73162
Tel: (405) 848-8086

OREGON

†Intel Corp.
15254 N.W. Greenbrier Parkway
Building B
Beaverton 97006
Tel: (503) 645-8051
TWX: 910-467-8741

PENNSYLVANIA

†Intel Corp.*
455 Pennsylvania Avenue
Suite 230
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.*
12000 Ford Road
Suite 400
Dallas 75234
Tel: (214) 241-8087
FAX: 214-484-1180

†Intel Corp.*
7322 S.W. Freeway
Suite 1490
Houston 77074
Tel: (713) 988-8086
TWX: 910-881-2490

UTAH

†Intel Corp.
428 East 6400 South
Suite 104
Murray 84107
Tel: (801) 263-8051

VIRGINIA

†Intel Corp.
1504 Santa Rosa Road
Suite 108
Richmond 23288
Tel: (804) 282-5668

WASHINGTON

†Intel Corp.
155 108th 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. John Boulevard
Pointe Claire H9R 3K2
Tel: (514) 694-9130
TWX: 514-694-9134

†Sales and Service Office
*Field Application Location

DOMESTIC DISTRIBUTORS

ALABAMA

Arrow Electronics, Inc.
1015 Henderson Road
Huntsville 35816
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 35816
Tel: (205) 837-9300
TWX: 810-726-2197

ARIZONA

†Hamilton/Avnet Electronics
505 S. Madison Drive
Tempe 85281
Tel: (602) 968-1461
TWX: 910-950-0077

Kierulff Electronics, Inc.
4134 E. Wood Street
Phoenix 85040
Tel: (602) 437-0750
FAX: 602-252-9109

Wyle Distribution Group
17855 N. Black Canyon Highway
Phoenix 85023
Tel: (602) 866-2888
FAX: 602-866-6937

CALIFORNIA

Arrow Electronics, Inc.
19748 Dearborn Street
Chatsworth 91311
Tel: (818) 701-7500
FAX: 818-772-8930

Arrow Electronics, Inc.
9511 Ridgehaven Court
San Diego 92123
Tel: (619) 565-4800
FAX: 619-279-0862

†Arrow Electronics, Inc.
521 Weddell Drive
Sunnyvale 94089
Tel: (408) 745-6600
FAX: 408-743-4770

Arrow Electronics, Inc.
2961 Dow Avenue
Tustin 92680
Tel: (714) 838-5422
FAX: 714-838-4151

†Avnet Electronics
350 McCormick Avenue
Costa Mesa 92626
Tel: (714) 754-6051
FAX: 714-754-6007

Hamilton/Avnet Electronics
1175 Bordeaux Drive
Sunnyvale 94089
Tel: (408) 743-3300
FAX: 408-745-6679

†Hamilton/Avnet Electronics
4545 Viewridge Avenue
San Diego 92123
Tel: (619) 571-7500
FAX: 619-277-6136

†Hamilton/Avnet Electronics
9650 Desoto Avenue
Chatsworth 91311
Tel: (818) 700-1222, 6500
FAX: 818-700-6553

†Hamilton/Avnet Electronics
4103 Northgate Boulevard
Sacramento 95834
Tel: (916) 920-3150
FAX: 916-925-3478

†Hamilton/Avnet Electronics
3002 G Street
Ontario 91311
Tel: (714) 989-9411
FAX: 714-980-7129

†Hamilton/Avnet Electronics
10950 W. Washington Blvd.
Culver City 90230
Tel: (213) 558-2458
FAX: 213-558-2248

†Hamilton Electro Sales
3170 Pullman Street
Costa Mesa 92626
Tel: (714) 641-4150
FAX: 714-641-4122

CALIFORNIA (Cont'd.)

Kierulff Electronics, Inc.
10824 Hope Street
Cypress 90630
Tel: (714) 220-6300
FAX: 714-821-8420

†Kierulff Electronics, Inc.
1180 Murphy Avenue
San Jose 95131
Tel: (408) 971-2600
FAX: 408-947-3432

†Kierulff Electronics, Inc.
14242 Chamber Rd.
Tustin 92680
Tel: (714) 731-5711
FAX: 714-669-4235

†Kierulff Electronics, Inc.
9800 Variel St.
Chatsworth 91311
Tel: (213) 725-0325
FAX: 818-407-0803

Wyle Distribution Group
26677 W. Agoura Rd.
Calabasas 91302
Tel: (818) 880-9000
FAX: 818-880-5510

†Wyle Distribution Group
17872 Cowan Avenue
Irvine 92714
Tel: (714) 863-9953
FAX: 714-863-0473

Wyle Distribution Group
11151 Sun Center Drive
Rancho Cordova 95670
Tel: (916) 638-5282
FAX: 916-638-1491

†Wyle Distribution Group
9525 Chesapeake Drive
San Diego 92123
Tel: (619) 565-9171
TWX: 910-371-9592
FAX: 619-565-9171 ext. 274

†Wyle Distribution Group
3000 Bowers Avenue
Santa Clara 95051
Tel: (408) 727-2500
FAX: 408-727-5896

Wyle Military
18910 Teller Avenue
Irvine 92715
Tel: (714) 851-9958
TWX: 910-371-9127
FAX: 714-851-8366

Wyle Systems
7382 Lampson Avenue
Garden Grove 92641
Tel: (714) 891-1717
FAX: 714-895-9038

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
FAX: 203-797-2866

†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) 475-4297
TWX: 510-955-9456

†Arrow Electronics, Inc.
1530 Bottlebrush N.E.
Palm Bay 32905
Tel: (305) 725-1480

†Hamilton/Avnet Electronics
6801 N.W. 15th Way
Ft. Lauderdale 33309
Tel: (305) 971-2900
TLX: 510-956-3097

Hamilton/Avnet Electronics
3245 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
FAX: 305-628-3888 ext. 40

†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
FAX: 404-242-6827

Hamilton/Avnet Electronics
5825 D. Peachtree Corners East
Norcross 30092
Tel: (404) 447-7500
TWX: 810-766-0432

Pioneer Electronics
3100 F. Northwoods Place
Norcross 30071
Tel: (404) 448-1711
FAX: 404-446-8270

ILLINOIS

†Arrow Electronics, Inc.
2000 E. Alonquin Street
Schaumberg 60173
Tel: (312) 397-3440
FAX: 312-397-3550

†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
FAX: 312-250-0916

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

INDIANA (Cont'd.)

Hamilton/Avnet Electronics
485 Gradle Drive
Carmel 46032
Tel: (317) 844-9333
FAX: 317-844-5921

†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
FAX: 913-541-7951

Pioneer Electronics
10551 Lackman Rd.
Lenexa 66215
Tel: (913) 492-0500
FAX: 913-492-7832

KENTUCKY

Hamilton/Avnet Electronics
805-A Newtown Circle
Lexington 40511
Tel: (606) 259-1475
FAX: 606-252-3238

MARYLAND

Arrow Electronics, Inc.
9300 Guilford Rd., Suite H
Rivers Center
Columbia 21046
Tel: (301) 995-6002
TWX: 710-236-9005
FAX: 301-381-3854

†Hamilton/Avnet Electronics
6822 Oak Hall Lane
Columbia 21045
Tel: (301) 995-3500
FAX: 301-995-3593

†Mesa Technology Corp.
9720 Patuxent Woods 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
100 Centennial Drive
Peabody 01960
Tel: (617) 532-3701
TWX: 710-393-0382

Kierulff Electronics, Inc.
13 Fortune Dr.
Billerica 01821
Tel: (617) 667-8331
TWX: 710-390-1449
FAX: 617-663-1754

Pioneer Northeast Electronics
44 Hartwell Avenue
Lexington 02173
Tel: (617) 861-9200
FAX: 617-863-1547

MICHIGAN

Arrow Electronics, Inc.
755 Phoenix Drive
Ann Arbor 48108
Tel: (313) 971-8220
FAX: 313-971-2633

†Hamilton/Avnet Electronics
32487 Schoolcraft Road
Livonia 48150
Tel: (313) 522-4700
TWX: 810-242-8775
FAX: 313-522-2624

Hamilton/Avnet Electronics
2215 29th Street S.E.
Space A5
Grand Rapids 49508
Tel: (616) 243-8805
TWX: 810-273-6921
FAX: 616-243-0028

MICHIGAN (Cont'd.)

Pioneer Electronics
4505 Broadmoor Ave. S.E.
Grand Rapids 49508
Tel: (616) 698-1800
FAX: 616-698-1831

†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
FAX: 612-830-1856

Hamilton/Avnet Electronics
12400 White Water Drive
Minnetonka 55343
Tel: (612) 932-0600
FAX: 612-932-0613

†Pioneer Electronics
10203 Bren Road East
Minnetonka 55343
Tel: (612) 935-1444
FAX: 612-935-1921

MISSOURI

†Arrow Electronics, Inc.
2380 Schuetz
St. Louis 63146
Tel: (314) 567-6888
FAX: 314-567-1164

††Hamilton/Avnet Electronics
13743 Shoreline Court East
Earth City 63045
Tel: (314) 344-1200
FAX: 314-291-8889

Kierulff Electronics, Inc.
11804 Borman Dr.
St. Louis 63146
Tel: (314) 997-4956
FAX: 314-567-0860

NEW HAMPSHIRE

†Arrow Electronics, Inc.
3 Perimeter Road
Manchester 03103
Tel: (603) 668-6968
FAX: 603-668-3484

Hamilton/Avnet Electronics
444 E. Industrial Drive
Manchester 03103
Tel: (603) 624-9400
FAX: 603-624-2402

NEW JERSEY

†Arrow Electronics, Inc.
6000 Lincoln Drive East
Marlton 08053
Tel: (609) 596-8000
FAX: 609-596-5632

†Arrow Electronics, Inc.
6 Century Drive
Parsipanny 07054
Tel: (201) 538-0900
FAX: 201-538-4962

†Hamilton/Avnet Electronics
1 Keystone Ave., Bldg. 36
Cherry Hill 08003
Tel: (609) 424-0110
TWX: 710-940-0262
FAX: 609-751-8624

†Hamilton/Avnet Electronics
10 Industrial
Fairfield 07006
Tel: (201) 575-3390
FAX: 201-575-5839

†Pioneer Northeast Electronics
45 Route 46
Pinebrook 07058
Tel: (201) 575-3510
FAX: 201-575-3454

†MTI Systems Sales
37 Kulick Rd.
Fairfield 07006
Tel: (201) 227-5552
FAX: 201-575-6336

†Microcomputer System Technical Distributor Center

DOMESTIC DISTRIBUTORS

NEW MEXICO

Alliance Electronics Inc.
11030 Cochiti S.E.
Albuquerque 87123
Tel: (505) 292-3360
FAX: 505-292-6537

Hamilton/Avnet Electronics
2524 Baylor Drive S.E.
Albuquerque 87106
Tel: (505) 765-1500
FAX: 505-243-1395

NEW YORK

Arrow Electronics, Inc.
25 Hub Drive
Melville 11747
Tel: (516) 694-6800
TWX: 510-224-6126
FAX: 516-391-1401

†Arrow Electronics, Inc.
3375 Brighton-Henrietta Townline Rd.
Rochester 14623
Tel: (716) 427-0300
FAX: 716-427-0735

Arrow Electronics, Inc.
20 Oser Avenue
Hauppauge 11788
Tel: (516) 231-1000
FAX: 516-231-1072

Hamilton/Avnet Electronics
2060 Townline Rd.
Rochester 14623
Tel: (716) 475-9130
FAX: 716-475-9119

†Hamilton/Avnet Electronics
103 Twin Oaks Drive
Syracuse 13206
Tel: (315) 437-2641
FAX: 315-432-0740

†Hamilton/Avnet Electronics
933 Motor Parkway
Hauppauge 11788
Tel: (516) 231-9800
FAX: 516-434-7426

†MTI Systems Sales
38 Harbor Park Drive
P.O. Box 271
Port Washington 11050
Tel: (516) 621-6200
FAX: 516-625-3039

†Pioneer Northeast Electronics
68 Corporate Drive
Binghamton 13904
Tel: (607) 722-9300
FAX: 607-722-9562

†Pioneer Northeast Electronics
60 Crossway Park West
Woodbury, Long Island 11797
Tel: (516) 921-8700
TWX: 510-221-2184
FAX: 516-921-2143

†Pioneer Northeast Electronics
840 Fairport Park
Fairport 14450
Tel: (716) 381-7070
FAX: 716-381-5955

NORTH CAROLINA

†Arrow Electronics, Inc.
5240 Greens Dairy Road
Raleigh 27604
Tel: (919) 876-3132
FAX: 919-876-3132, ext. 200

†Hamilton/Avnet Electronics
3510 Spring Forest Drive
Raleigh 27609
Tel: (919) 878-0819
TWX: 510-928-1836

NORTH CAROLINA (Cont'd.)

Pioneer Electronics
9801 A-Southern Pine Blvd.
Charlotte 28217
Tel: (704) 527-8188
TWX: 810-621-0366

OHIO

Arrow Electronics, Inc.
7620 McEwen Road
Centerville 45459
Tel: (513) 435-5563
FAX: 513-435-2049

†Arrow Electronics, Inc.,
6238 Cochran Road
Solon 44139
Tel: (216) 248-3990
FAX: 216-248-1106

Hamilton/Avnet Electronics
777 Brooksedge Blvd.
Westerville 43081
Tel: (614) 882-7004
FAX: 614-882-8650

†Hamilton/Avnet Electronics
954 Senate Drive
Dayton 45459
Tel: (513) 439-6700
FAX: 513-439-6711

†Hamilton/Avnet Electronics
30325 Bainbridge Rd., Bldg. A
Solon 44139
Tel: (216) 349-5100
FAX: 216-349-1894

†Pioneer Electronics
4433 Interpoint Blvd.
Dayton 45424
Tel: (513) 236-9900
FAX: 513-236-8133

†Pioneer Electronics
4800 E. 131st Street
Cleveland 44105
Tel: (216) 587-3600
TWX: 810-422-2211
FAX: 216-587-3906

OKLAHOMA

Arrow Electronics, Inc.
3158 S. 108 East Ave., Ste. 210
Tulsa 74146
Tel: (918) 665-7700
FAX: 918-665-7700

OREGON

†Almac Electronics Corp.
1885 N.W. 169th Place
Beaverton 97006
Tel: (503) 629-8090
FAX: 503-645-0611

†Hamilton/Avnet Electronics
6024 S.W. Jean Road
Bldg. C, Suite 10
Lake Oswego 97034
Tel: (503) 635-7848
FAX: 503-636-1327

Wyle Distribution Group
5250 N.E. Elam Young Parkway
Suite 600
Hillsboro 97124
Tel: (503) 640-6000
FAX: 503-640-5846

PENNSYLVANIA

Arrow Electronics, Inc.
650 Seco Road
Monroeville 15146
Tel: (412) 856-7000
FAX: 412-856-5777

Hamilton/Avnet Electronics
2800 Liberty Ave., Bldg. E
Pittsburgh 15222
Tel: (412) 281-4150
FAX: 412-281-8662

PENNSYLVANIA (Cont'd.)

Pioneer Electronics
259 Kappa Drive
Pittsburgh 15238
Tel: (412) 782-2300
TWX: 710-795-3122
FAX: 412-963-8255

†Pioneer Electronics
261 Gibralter Road
Horsham 19044
Tel: (215) 674-4000
TWX: 510-665-6778
FAX: 215-674-3107

TEXAS

†Arrow Electronics, Inc.
3220 Commander Drive
Carrollton 75006
Tel: (214) 380-6464
FAX: 214-248-7208

†Arrow Electronics, Inc.
10899 Kinghurst Dr.
Suite 100
Houston 77099
Tel: (713) 530-4700
FAX: 713-568-8518

†Arrow Electronics, Inc.
2227 W. Braker Lane
Austin 78758
Tel: (512) 835-4180
FAX: 512-832-9875

†Hamilton/Avnet Electronics
1807 W. Braker Lane
Austin 78758
Tel: (512) 837-8911
FAX: 512-339-6232

†Hamilton/Avnet Electronics
2111 W. Walnut Hill Lane
Irving 75038
Tel: (214) 550-6111
FAX: 214-550-6172

†Hamilton/Avnet Electronics
4850 Wright Road, Ste. 190
Stafford 77477
Tel: (713) 240-7733
FAX: 713-240-0582

Kierulff Electronics, Inc.
2010 Merritt Drive
Garland 75040
Tel: (214) 840-0110
FAX: 214-278-0928

†Pioneer Electronics
1826-D Kramer Lane
Austin 78758
Tel: (512) 835-4000
FAX: 512-835-9829

†Pioneer Electronics
13710 Omega Road
Dallas 75244
Tel: (214) 386-7300
FAX: 214-490-6419

†Pioneer Electronics
5853 Point West Drive
Houston 77036
Tel: (713) 988-5555
FAX: 713-988-1732

UTAH

†Hamilton/Avnet Electronics
1585 West 2100 South
Salt Lake City 84119
Tel: (801) 972-2800
FAX: 801-974-9675

Kierulff Electronics, Inc.
1946 W. Parkway Blvd.
Salt Lake City 84119
Tel: (801) 973-6913
FAX: 801-972-0200

Wyle Distribution Group
1325 West 2200 South
Suite E
Salt Lake City 84119
Tel: (801) 974-9953
FAX: 801-972-2524

WASHINGTON

†Almac Electronics Corp.
14360 S.E. Eastgate Way
Bellevue 98007
Tel: (206) 643-9992
FAX: 206-643-9709

Arrow Electronics, Inc.
14320 N.E. 21st Street
Bellevue 98007
Tel: (206) 643-4800
FAX: 206-746-3740

Hamilton/Avnet Electronics
14212 N.E. 21st Street
Bellevue 98005
Tel: (206) 453-5874
FAX: 206-643-0086

Wyle Distribution Group
1750 132nd Ave., N.E.
Bellevue 98005
Tel: (206) 453-8300
FAX: 206-453-4071

WISCONSIN

†Arrow Electronics, Inc.
200 N. Patrick Blvd., Ste. 100
Brookfield 53005
Tel: (414) 792-0150
FAX: 414-792-0156

Hamilton/Avnet Electronics
2975 Moorland Road
New Berlin 53151
Tel: (414) 784-4510
FAX: 414-784-9509

Kierulff Electronics, Inc.
2238-E W. Bluemound Rd.
Waukesha 53186
Tel: (414) 784-8160
FAX: 414-784-0409

CANADA

ALBERTA

Hamilton/Avnet Electronics
2816 21st Street N.E.
Calgary T2E 6Z2
Tel: (403) 250-9380
FAX: 403-250-1591

Zentronics
6815 8th Street, N.E., Ste. 100
Calgary T2E 7H7
Tel: (403) 295-8838
FAX: 403-295-8714

BRITISH COLUMBIA

Hamilton/Avnet Electronics
2550 Boundary Rd., Ste. 115
Burnaby V5M 3Z3
Tel: (604) 437-6667
FAX: 604-437-4712

Zentronics
106-11400 Bridgeport Road
Richmond V6X 1T2
Tel: (604) 273-5575
FAX: 604-273-2413

MANITOBA

Zentronics
60-1313 Border Street
Winnipeg R3H 0X4
Tel: (204) 694-1957
FAX: 204-633-9255

ONTARIO

Arrow Electronics Inc.
1093 Meyerside Drive
Unit 2
Mississauga LST 1M4
Tel: (416) 672-7769
FAX: 416-672-0489

Arrow Electronics Inc.
Nepean K2E 7W5
Tel: (613) 226-6903
FAX: 613-723-2018

†Hamilton/Avnet Electronics
6845 Rexwood Road
Units 3-5
Mississauga L4V 1R2
Tel: (416) 677-7432
FAX: 416-677-0940

Hamilton/Avnet Electronics
3688 Nashua Dr.
Units 9 and 10
Mississauga L4V 1M5
Tel: (416) 677-0484
FAX: 416-677-0627

†Hamilton/Avnet Electronics
190 Colonnade Road South
Nepean K2E 7J5
Tel: (613) 226-1700
FAX: 613-226-1184

†Zentronics
8 Tilbury Court
Brampton L6T 3T4
Tel: (416) 451-9600
FAX: 416-451-3320

†Zentronics
155 Colonnade Road
Unit 17
Nepean K2E 7K1
Tel: (613) 226-8840
FAX: 613-226-6350

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
FAX: 514-341-4821

Arrow Electronics Inc.
909 Charest Blvd.
Quebec 61N 269
Tel: (418) 687-4231
FAX: 418-687-5348

Hamilton/Avnet Electronics
2795 Rue Halpern
St. Laurent H4S 1P8
Tel: (514) 335-1000
FAX: 514-335-2481

Zentronics
817 McCafrey St.
St. Laurent H4T 1N4
Tel: (514) 737-9700
FAX: 514-737-5212

†Microcomputer System Technical Distributor Center

EUROPEAN SALES OFFICES

DENMARK

Intel
Glentevej 61, 3rd Floor
2400 Copenhagen NV
Tel: (01) 19 80 33
TLX: 19567

FINLAND

Intel
Ruosilantie 2
00390 Helsinki
Tel: +358 0 544 644
TLX: 123332

FRANCE

Intel
1, Rue Edison-BP 303
78054 St. Quentin-en-Yvelines Cedex
Tel: (1) 30 57 70 00
TLX: 699016

Intel
Immeuble BBC
4, Quai des Etroits
69005 Lyon
Tel: 78 42 40 89
TLX: 305153

WEST GERMANY

Intel*
Seidlstrasse 27
8000 Muenchen 2
Tel: 089/5389-0
TLX: 5-23177

Intel
Hohenzollern Strasse 5
3000 Hannover 1
Tel: 0511/344081
TLX: 9-23625

Intel
Abraham Lincoln Strasse 16-18
6200 Wiesbaden
Tel: 06121/7605-0
TLX: 4-186183

Intel
Zettachring 10A
7000 Stuttgart 80
Tel: (0711) 72 87-280
TLX: 7254826

ISRAEL

Intel*
Atidim Industrial Park-Neve Sharet
P.O. Box 43202
Tel-Aviv 61430
Tel: 03-499080
TLX: 371215

ITALY

Intel*
Milanofiori Palazzo E
20090 Assago
Milano
Tel: (02) 824 40 71
TLX: 341286

NETHERLANDS

Intel*
Marten Meesweg 93
3068 AV Rotterdam
Tel: (31)-(0)10-421.23.77
TLX: 22283

NORWAY

Intel
Hvamveien 4-P.O. Box 92
2013 Skjetten
Tel: (6) 842-420
TLX: 78018

SPAIN

Intel
Zurbaran, 28
28010 Madrid
Tel: 410 40 04
TLX: 46880

SWEDEN

Intel*
Dalvagen 24
171 36 Solna
Tel: +46 8 734 01 00
TLX: 12261

SWITZERLAND

Intel*
Talackerstrasse 17
8065 Zurich
Tel: 01/829 29 77
TLX: 57989

UNITED KINGDOM

Intel*
Pipers Way
Swindon, Wiltshire SN3 1RJ
Tel: (0793) 69 60 00
TLX: 444447/8

EUROPEAN DISTRIBUTORS/REPRESENTATIVES

AUSTRIA

Bacher Electronics Ges.m.b.H.
Rothenmuehlgasse 26
A-1120 Wien
Tel: (0222) 83 56 46-0
TLX: 131532

BELGIUM

Inelco Belgium S.A.
Av. des Croix de Guerre 94
1120 Bruxelles
Oorlogskruisenlaan, 94
1120 Brussel
Tel: (02) 216 01 60
TLX: 64475

DENMARK

ITT-Multikomponent
Naverland 29
2600 Glostrup
Tel: 45 (0) 2 45 66 45
TLX: 33 355

FINLAND

OY Fintronic AB
Melkonkatu 24A
00210 Helsinki
Tel: (0) 6926022
TLX: 124224

FRANCE

Generim
Z.A. de Courtaboeuf
Av. de la Baltique-BP 88
91943 Les Ulis Cedex
Tel: (1) 69 07 78 78
TLX: 691700

Jermyn S.A.
73-73, rue des Solets
Silic 585
94663 Rungis Cedex
Tel: (1) 45 60 04 00
TLX: 260967

Metrologie
Tour d'Asnieres
4, av. Laurent-Cely
92606 Asnieres Cedex
Tel: (1) 47 90 62 40
TLX: 611448

Tekelec-Airtronic
Rue Carle Vernet - BP 2
92315 Sevres Cedex
Tel: (1) 45 34 75 35
TLX: 204552

WEST GERMANY

Electronic 2000 AG
Stahlgruberring 12
8000 Muenchen 82
Tel: 089/42001-0
TLX: 522561

EMPA Electronic
Lindwurmstrasse 95A
8000 Muenchen 2
Tel: 089/53/80 570
TLX: 528573

ITT Multikomponent GmbH
Postfach 1265
Bahnhofstrasse 44
7141 Moeglingen
Tel: 07141/487-347
TLX: 7264399

Jermyn GmbH
Im Dachsstueck 9
6250 Limburg Offheim
Tel: 06431/508-0
TLX: 415257-0

Metrologie GmbH
Meglingerstrasse 49
8000 Muenchen 71
Tel: 089/78042-0
TLX: 5213189

Proelectron Vertriebs GmbH
Max Planck Strasse 1-3
6072 Dreieich
Tel: 06103/30 43 43
TLX: 417972

IRELAND

Micro Marketing Ltd.
Glenageary Office Park
Glenageary
Co. Dublin
Tel: (01) 85 63 25
TLX: 31584

ISRAEL

Eastronics Ltd.
11 Rozanis Street
P.O. Box 39300
Tel-Aviv 61392
Tel: 03-475151
TLX: 33638

ITALY

Intesi
Divisione ITT Industries GmbH
Viale Milanofiori
Palazzo E/5
20090 Assago
Milano
Tel: 02/824-701
TLX: 311351

ITALY (Cont'd.)

Lasi Elettronica S.p.a.
V. le Fulvio Tesi, 126
20092 Cinisello Balsamo
Milano
Tel: 02/2440012
TLX: 352040

NETHERLANDS

Koning en Hartman
Postbos 125
2600 AP Delft
Tel: (015) 60 99 06
TLX: 38250

NORWAY

Nordisk Elektronikk (Norge) A/S
Postboks 123
Smedsvingen 4
1364 Hvalstad
Tel: (02) 84 62 10
TLX: 77546

PORTUGAL

Ditram
Avenida Marques de Tomar, 46-A
1000 Lisboa
Tel: (1) 73 48 34
TLX: 14182

SPAIN

ATD Electronica, S.A.
Plaza Ciudad de Viena, 6
28040 Madrid
Tel: 234 40 00
TLX: 42754

ITT-SESA
Calle Miguel Angel, 21-3
28010 Madrid
Tel: 419 09 57
TLX: 27461

SWEDEN

Nordisk Elektronik AB
Huvudstagatan 1
P.O. Box 1409
1727 Solna
Tel: 08-734 97 70
TLX: 10547

SWITZERLAND

Industrade A.G.
Hertistrasse 31
8304 Wallisellen
Tel: (801) 83 05 04 0
TLX: 56788

UNITED KINGDOM

Accent Electronic Components Ltd.
Jubilee House, Jubilee Road
Letchworth, Herts SG6 1TL
Tel: (0462) 686666
TLX: 826293

Bytech Components (Components & DTO)
Unit 3 The Western Centre
Western Road
Bracknell
Berks RG12 1RW
Tel: (0344) 48 22 11
TWX: 848215

Bytech Comway Ltd. (Systems)
Unit 3 The Western Centre
Western Road
Bracknell
Berks RG12 1RW
Tel: (0344) 55333
TLX: 848215

Jermyn
Vestry Estate
Otford Road
Sevenoaks
Kent TN14 5EU
Tel: (0732) 450144
TLX: 95142

MMD (PLDO)
Unit 8 South View Park
Caversham
Reading
Berks RG4 0AF
Tel: (0793) 48 16 66
TLX: 846669

Rapid Silicon
Rapid House
Denmark St.
High Wycombe
Buckinghamshire HP11 2ER
Tel: (0494) 442266
TLX: 837931

Rapid Systems
Rapid House
Denmark St.
High Wycombe
Buckinghamshire HP11 2ER
Tel: (0494) 450244
TLX: 837931

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: AA 20097
FAX: (2) 923-2632

BRAZIL

Intel Semicondutores do Brasil LTDA
Av. Paulista, 1159-CJS 404/405
01311 - Sao Paulo - S.P.
Tel: 55-11-287-5899
TLX: 1153146 SAPI BR
FAX: 55-11-212-7631

CHINA/HONG KONG

Intel PRC Corporation
15/F, Office 1, Citic Bldg.
Jian Guo Men Wai Street
Beijing, PRC
Tel: (1) 500-4850
TLX: 22947 INTEL CN
FAX: (1) 500-2953

Intel Semiconductor Ltd.*
10/F East Tower
Bond Center
Queensway, Central
Hong Kong
Tel: (5) 8444-555
TLX: 63869 ISHLHK HX
FAX: (5) 8681-989

JAPAN

Intel Japan K.K.
5-6 Tokodai, Tsukuba-shi
Ibaraki, 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.
Business Center 16th Floor
61, Yoido-Dong, Young Deung Po-Ku
Seoul 150
Tel: (2) 784-8186, 8286, 8386
TLX: K29312 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

DAFSYS S.R.L.
Chacabuco, 90-4 PISO
1069-Buenos Aires
Tel: 54-1-334-1871
54-1-334-7726
TLX: 25472

Reycom Electronica S.R.L.
Arcos 3631
1429-Buenos Aires
Tel: 54 (1) 701-4462/66
FAX: 54 (1) 11-1722
TLX: 22122

AUSTRALIA

Total Electronics
P.M.B. 250
9 Harker Street
Burwood, Victoria 3125
Tel: 61-3-288-4044
TLX: AA 31261
FAX: 61-3-288-9696

BRAZIL

Elebra Microelectronica
R. Geraldo Flausina Gomes, 78
9 Andar
04575 - Sao Paulo - S.P.
Tel: 55-11-534-9522
TLX: 1154591 ELB
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/HONG KONG

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: 852-0-261-802

CHINA/HONG KONG (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
Flat 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, Trombay Road
Chembur
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: 03-497-4900
FAX: 03-497-4879

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: 052-204-2916
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

Electronic Resources 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
Residencia Kamarata
Locales 4 A17
La Candelaria, Caracas
Tel: 58-2-571-0396
TLX: 28450
FAX: 58-2-572-3321

*Field Application Location

CG-3/4/88

UNITED STATES
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

JAPAN
Intel Japan K.K.
5-6 Tokodai, Tsukuba-shi
Ibaraki, 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.
10/F East Tower
Bond Center
Queensway, Central

CANADA
Intel Semiconductor of Canada, Ltd.
190 Attwell Drive, Suite 500
Rexdale, Ontario M9W 6H8

Printed in USA/10K/488/RRD/PM
Data Communications

ISBN 1-55512-034-2