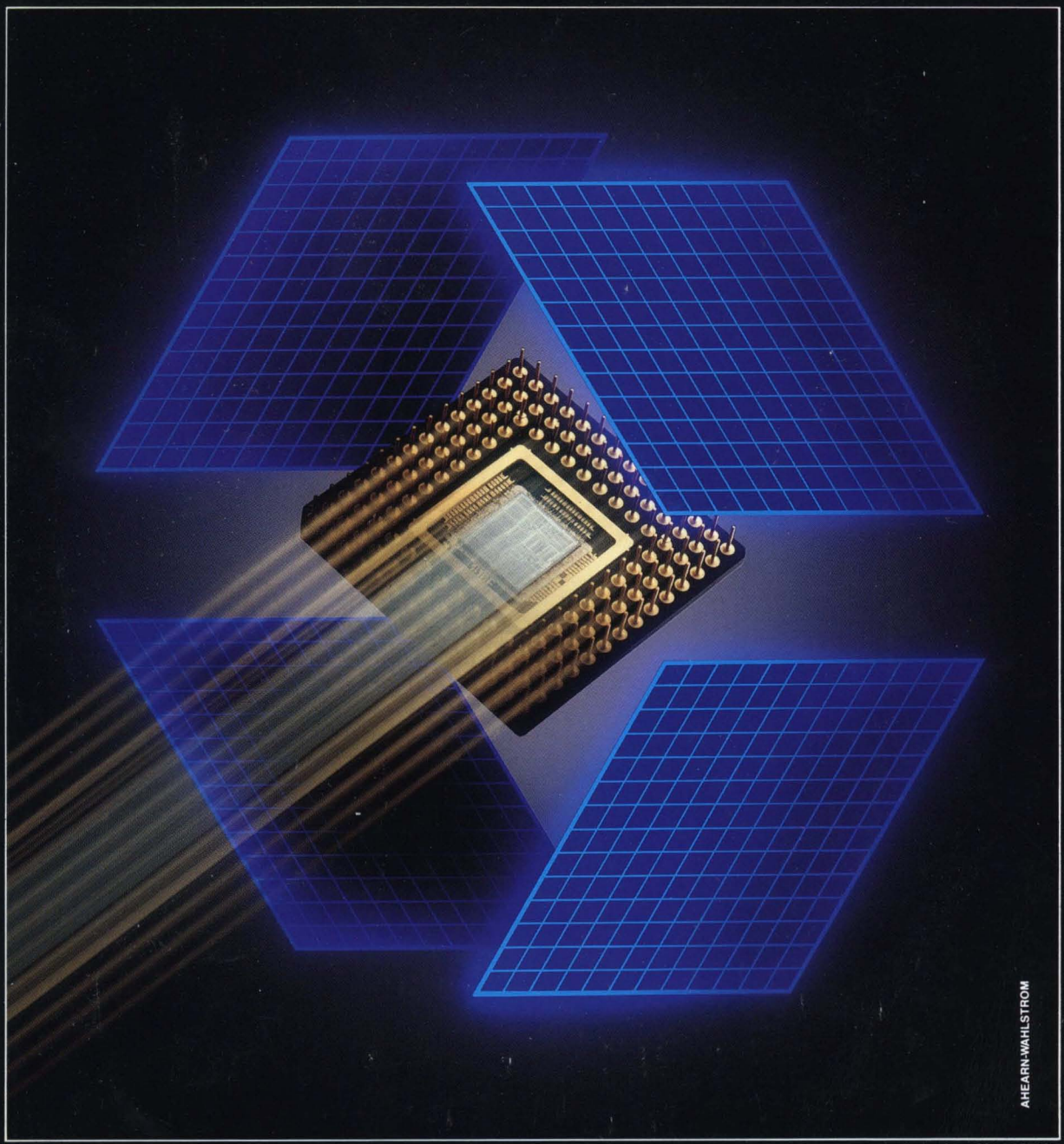
intel

INTEL386™ DX MICROPROCESSOR
HARDWARE REFERENCE MANUAL



Order No. 231732-005

intel

LITERATURE

To order Intel Literature or obtain literature pricing information in the U.S. and Canada call or write Intel Literature Sales. In Europe and other international locations, please contact your ***local*** sales office or distributor.

INTEL LITERATURE SALES
P.O. BOX 7641
Mt. Prospect, IL 60056-7641

In the U.S. and Canada
call toll free
(800) 548-4725
This 800 number is for external customers only.

CURRENT HANDBOOKS

Product line handbooks contain data sheets, application notes, article reprints and other design information. All handbooks can be ordered individually, and most are available in a pre-packaged set in the U.S. and Canada.

TITLE	INTEL ORDER NUMBER	ISBN
SET OF THIRTEEN HANDBOOKS (Available in U.S. and Canada)	**231003**	**N/A**
CONTENTS LISTED BELOW FOR INDIVIDUAL ORDERING:		
COMPONENTS QUALITY/RELIABILITY	210997	1-55512-132-2
EMBEDDED APPLICATIONS	270648	1-55512-123-3
8-BIT EMBEDDED CONTROLLERS	270645	1-55512-121-7
16-BIT EMBEDDED CONTROLLERS	270646	1-55512-120-9
16/32-BIT EMBEDDED PROCESSORS	270647	1-55512-122-5
MEMORY PRODUCTS	210830	1-55512-117-9
MICROCOMMUNICATIONS	231658	1-55512-119-5
MICROCOMPUTER PRODUCTS	280407	1-55512-118-7
MICROPROCESSORS	230843	1-55512-115-2
PACKAGING	240800	1-55512-128-4
PERIPHERAL COMPONENTS	296467	1-55512-127-6
PRODUCT GUIDE (Overview of Intel's complete product lines)	210846	1-55512-116-0
PROGRAMMABLE LOGIC	296083	1-55512-124-1
ADDITIONAL LITERATURE: (Not included in handbook set)		
AUTOMOTIVE HANDBOOK	231792	1-55512-125-x
INTERNATIONAL LITERATURE GUIDE (Available in Europe only)	E00029	N/A
CUSTOMER LITERATURE GUIDE	210620	N/A
MILITARY HANDBOOK (2 volume set)	210461	1-55512-126-8
SYSTEMS QUALITY/RELIABILITY	231762	1-55512-046-6

intel®

U.S. and CANADA LITERATURE ORDER FORM

NAME: _____

COMPANY: _____

ADDRESS: _____

CITY: _____ STATE: _____ ZIP: _____

COUNTRY: _____

PHONE NO.: ( ) _____

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

Subtotal _____

Must Add Your Local Sales Tax _____

Include postage:
Must add 15% of Subtotal to cover U.S. and Canada postage. (20% all other.)

→ Postage _____

Total _____

Pay by check, money order, or include company purchase order with this form ($100 minimum). We also accept VISA, MasterCard or American Express. Make payment to Intel Literature Sales. Allow 2-4 weeks for delivery.

☐ VISA ☐ MasterCard ☐ American Express Expiration Date _____

Account No. _____

Signature _____

Mail To: Intel Literature Sales
P.O. Box 7641
Mt. Prospect, IL 60056-7641

International Customers outside the U.S. and Canada should use the International order form on the next page or contact their local Sales Office or Distributor.

For phone orders in the U.S. and Canada
Call Toll Free: (800) 548-4725

Prices good until 12/31/91.
Source HB

intel®

INTERNATIONAL LITERATURE ORDER FORM

NAME: _____

COMPANY: _____

ADDRESS: _____

CITY: _____ STATE: _____ ZIP: _____

COUNTRY: _____

PHONE NO.: ( ) _____

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

Subtotal _____

Must Add Your Local Sales Tax _____

Total _____

PAYMENT

Cheques should be made payable to your ***local*** Intel Sales Office (see inside back cover).

Other forms of payment may be available in your country. Please contact the Literature Coordinator at your ***local*** Intel Sales Office for details.

The completed form should be marked to the attention of the LITERATURE COORDINATOR and returned to your ***local*** Intel Sales Office.

Intel386™ DX MICROPROCESSOR HARDWARE REFERENCE MANUAL

1991

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

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

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

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

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

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

CHMOS and HMOS are patented processes of Intel Corp.

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

Micro Channel, OS/2 and PS/2 are trademarks of International Business Machines Corporation.

UNIX is a registered trademark of UNIX System Laboratories, Inc.

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

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

CG-072391

CUSTOMER SUPPORT

INTEL'S COMPLETE SUPPORT SOLUTION WORLDWIDE

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

After a customer purchases any system hardware or software product, service and support become major factors in determining whether that product will continue to meet a customer's expectations. Such support requires an international support organization and a breadth of programs to meet a variety of customer needs. As you might expect, Intel's customer support is extensive. It can start with assistance during your development effort to network management. 100 Intel sales and service offices are located worldwide – in the U.S., Canada, Europe and the Far East. So wherever you're using Intel technology, our professional staff is within close reach.

HARDWARE SUPPORT SERVICES

Intel's hardware maintenance service, starting with complete on-site installation will boost your productivity from the start and keep you running at maximum efficiency. Support for system or board level products can be tailored to match your needs, from complete on-site repair and maintenance support to economical carry-in or mail-in factory service.

Intel can provide support service for not only Intel systems and emulators, but also support for equipment in your development lab or provide service on your product to your end-user/customer.

SOFTWARE SUPPORT SERVICES

Software products are supported by our Technical Information Service (TIPS) that has a special toll free number to provide you with direct, ready information on known, documented problems and deficiencies, as well as work-arounds, patches and other solutions.

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

NETWORK SERVICE AND SUPPORT

Today's broad spectrum of powerful networking capabilities are only as good as the customer support provided by the vendor. Intel offers network services and support structured to meet a wide variety of end-user computing needs. From a ground up design of your network's physical and logical design to implementation, installation and network wide maintenance. From software products to turn-key system solutions; Intel offers the customer a complete networked solution. With over 10 years of network experience in both the commercial and Government arena; network products, services and support from Intel provide you the most optimized network offering in the industry.

CONSULTING SERVICES

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

CUSTOMER TRAINING

Intel offers a wide range of instructional programs covering various aspects of system design and implementation. In just three to ten days a limited number of individuals learn more in a single workshop than in weeks of self-study. For optimum convenience, workshops are scheduled regularly at Training Centers worldwide or we can take our workshops to you for on-site instruction. Covering a wide variety of topics, Intel's major course categories include: architecture and assembly language, programming and operating systems, BITBUS™ and LAN applications.

PREFACE

The Intel386™ DX microprocessor is a high-performance 32-bit microprocessor. This manual provides complete hardware reference information for Intel386 DX microprocessor system designs. It is written for system engineers and hardware designers who understand the operating principles of microprocessors and microcomputer systems. Readers of this manual should be familiar with the information in the *Introduction to the 80386* (Intel publication Order Number 231252).

RELATED PUBLICATIONS

In this manual, the Intel386 DX microprocessor is presented from a hardware perspective. Information on the software architecture, instruction set, and programming of the Intel386 DX microprocessor can be found in these related Intel publications:

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

The *386™ DX Microprocessor Data Sheet* contains device specifications for the Intel386 DX microprocessor. Always consult the most recent version of this publication for specific Intel386 DX microprocessor parameter values.

Detailed device specifications on Intel386 DX microprocessor family components can be found in the following publications:

- *387™ DX Math Coprocessor Data Sheet*, Order Number 240448
- *82380 Data Sheet*, Order Number 290128
- *82385 Data Sheet*, Order Number 290143
- *82310/11 Data Sheet*, Order Number 290167
- *82350 Data Sheet*, Order Number 290220
- *82596DX Data Sheet*, Order Number 290219

Together with the *Intel386™ DX Microprocessor Hardware Reference Manual*, these publications provide a complete description of the Intel386 DX microprocessor system for hardware designers, software engineers, and all users of Intel386 DX microprocessor systems.

The Intel386 DX microprocessor is object-code compatible with 3 other processors: the Intel486™, the Intel386 SX and the 376™ microprocessors.

The Intel486 DX processor integrates cache memory, floating point hardware and memory management on-chip while retaining binary compatibility with previous members of the 86 architectural family. Related documentation includes:

- *i486™ Microprocessor Data Sheet*, Order Number 240440
- *i486™ Microprocessor Programmer's Reference Manual*, Order Number 240486

The Intel386 SX processor (16-bit data bus) – The Intel386 DX processor adapted for mid-range personal computers, which are sensitive to the higher system cost of a 32-bit bus. Related documentation includes:

- *386™ SX Microprocessor Data Sheet*, Order Number 240187
- *386™ SX Microprocessor Hardware Reference Manual*, Order Number 240332

The 376 embedded processor (16-bit data bus) – A reduced form of the Intel386 processor, optimized for embedded applications. Related documentation includes:

- *376™ Microprocessor Data Sheet*, Order Number 240182
- *376™ Embedded Processor Programmer's Reference Manual*, Order Number 240314

ORGANIZATION OF THIS MANUAL

The information in this manual is divided into 12 chapters and three appendices. The material begins with a description of the Intel386 DX microprocessor and continues with discussions of hardware design information needed to implement Intel386 DX microprocessor system designs.

- Chapter 1, "System Overview." This chapter provides an overview of the Intel386 DX microprocessor and its supporting devices.
- Chapter 2, "Internal Architecture." This chapter describes the internal architecture of the Intel386 DX microprocessor.
- Chapter 3, "Local Bus Interface." This chapter discusses the Intel386 DX microprocessor local bus interface. This chapter includes Intel386 DX microprocessor signal descriptions, memory and I/O organization, and local bus interface guidelines.
- Chapter 4, "Performance Considerations." This chapter explores the factors that affect the performance of an Intel386 DX microprocessor system.
- Chapter 5, "Coprocessor Hardware Interface." This chapter describes the interface between the Intel386 DX microprocessor and the Intel387™ Numeric Coprocessors. This coprocessor expands the floating-point numerical processing capabilities of the Intel386 DX microprocessor.
- Chapter 6, "Memory Interfacing." This chapter discusses techniques for designing memory subsystems for the Intel386 DX microprocessor.
- Chapter 7, "Cache Subsystems." This chapter describes cache memory subsystems, which provide higher performance at lower relative cost.
- Chapter 8, "I/O Interfacing." This chapter discusses techniques for connecting I/O devices to an Intel386 DX microprocessor system.
- Chapter 9, "MULTIBUS I and Intel386 DX Microprocessor." This chapter describes the interface between an Intel386 DX microprocessor system and the Intel MULTIBUS I multi-master system bus.
- Chapter 10, "MULTIBUS II and Intel386 DX Microprocessor." This chapter describes the interface between an Intel386 DX microprocessor system and the Intel MULTIBUS II multi-master system bus.

- Chapter 11, "Physical Design and Debugging." This chapter contains recommendations for constructing and debugging Intel386 DX microprocessor systems.
- Chapter 12, "Test Capabilities." This chapter describes Intel386 DX microprocessor test procedures.
- Appendix A contains descriptions of the components of the basic memory interface described in Chapter 6.
- Appendix B contains descriptions of the components of the dynamic RAM subsystem described in Chapter 6.

TABLE OF CONTENTS

Figures

Figures

Figures

Tables

System Overview 1

CHAPTER 1
SYSTEM OVERVIEW

The Intel386 DX microprocessor is a 32-bit microprocessor that forms the basis for a high-performance 32-bit system. The Intel386 DX microprocessor incorporates multitasking support, memory management, pipelined architecture, address translation caches, and a high-speed bus interface all on one chip. The integration of these features speeds the execution of instructions and reduces overall chip count for a system. Paging and dynamic data bus sizing can each be invoked selectively, making the Intel386 DX microprocessor suitable for a wide variety of system designs and user applications.

While the Intel386 DX microprocessor represents a significant improvement over previous generations of microprocessors, substantial ties to the earlier processors are preserved. Software compatibility at the object-code level is provided, so that an existing investment in 8086 and 80286 software can be maintained. New software can be built upon existing routines, reducing the time to market for new products. Hardware compatibility is preserved through the dynamic bus-sizing feature.

The Intel386 DX microprocessor is fully supported by a family of peripheral and performance enhancement components. The major components of an Intel386 DX microprocessor system and their functions are shown in Figure 1-1 and Figure 1-2. Table 1-1 describes these components.

1.1 MICROPROCESSOR

The 33-MHz Intel386 DX microprocessor has a peak execution rate of over 16 million native instructions per second. It sustains rates of eight million equivalent VAX instructions per second, a speed comparable to that of most super minicomputers. This achievement is made possible through a state-of-the-art design that includes a pipelined internal architecture, address translation caches, and a high-performance bus.

The Intel386 DX microprocessor features 32-bit wide internal and external data paths and eight general-purpose 32-bit registers. The instruction set offers 8-, 16-, and 32-bit data types, and the processor outputs 32-bit physical addresses directly, for a physical memory capacity of four gigabytes.

The Intel386 DX microprocessor has separate 32-bit data and address paths. A 32-bit memory access can be completed in only two clock cycles, enabling the bus to sustain a throughput of 40 megabytes per second (at 20 MHz). By making prompt transfers between the microprocessor, memory, and peripherals, the high-speed bus design ensures that the entire system benefits from the processor's increased performance.

Pipelined architecture enables the Intel386 DX microprocessor to perform instruction fetching, decoding, execution, and memory management functions in parallel. The six independent units that make-up the Intel386 DX microprocessor pipeline are described

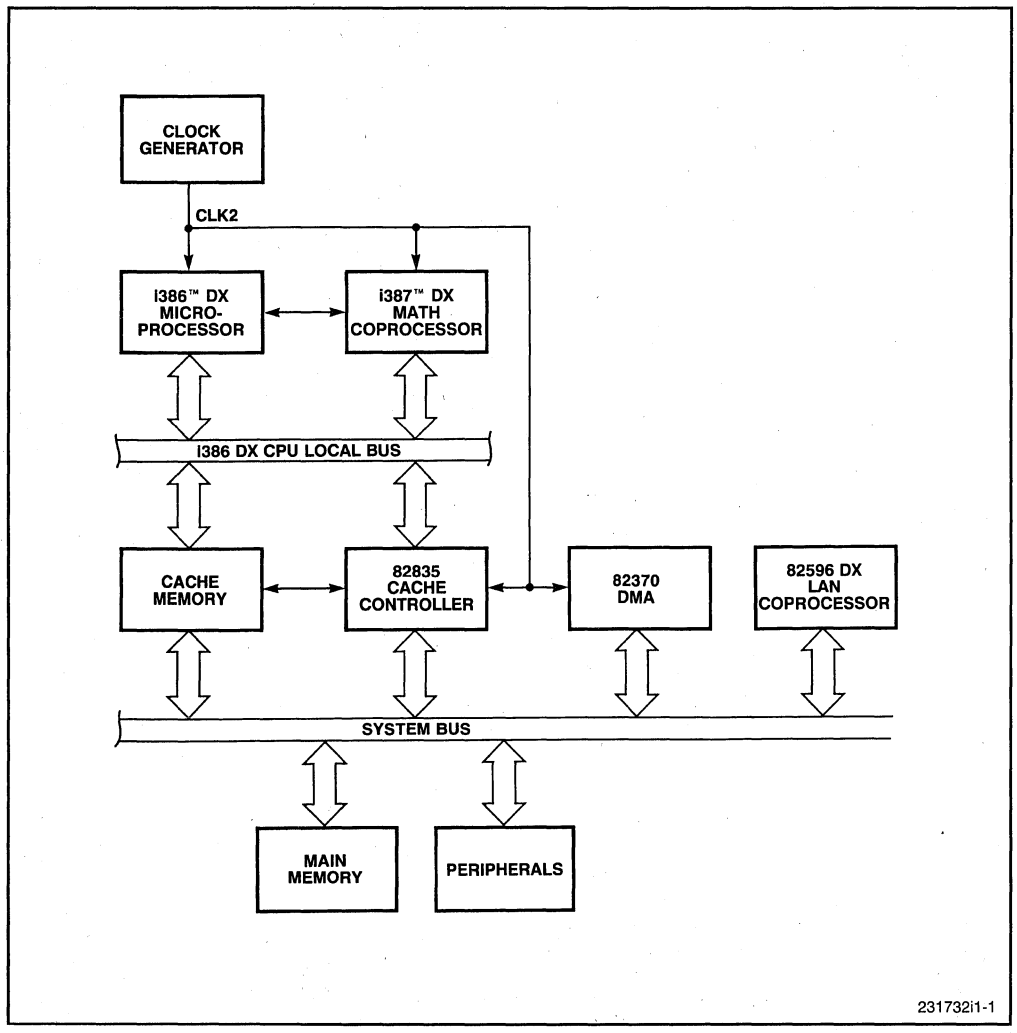


Figure 1-1. Intel386™ DX Microprocessor System Block Diagram

in detail in Chapter 2. Because the Intel386 DX microprocessor prefetches instructions and queues them internally, instruction fetch and decode times are absorbed in the pipeline; the processor rarely has to wait for an instruction to execute.

Pipelining is not unusual in modern microprocessor architecture; however, including the memory management unit (MMU) in the on-chip pipeline is a unique feature of the Intel386 DX Architecture. By performing memory management on-chip, the Intel386 DX microprocessor eliminates the serious access delays typical of implementations that use off-chip memory management units. The benefit is not only high performance but also relaxed memory-access time requirements, hence lower system cost.

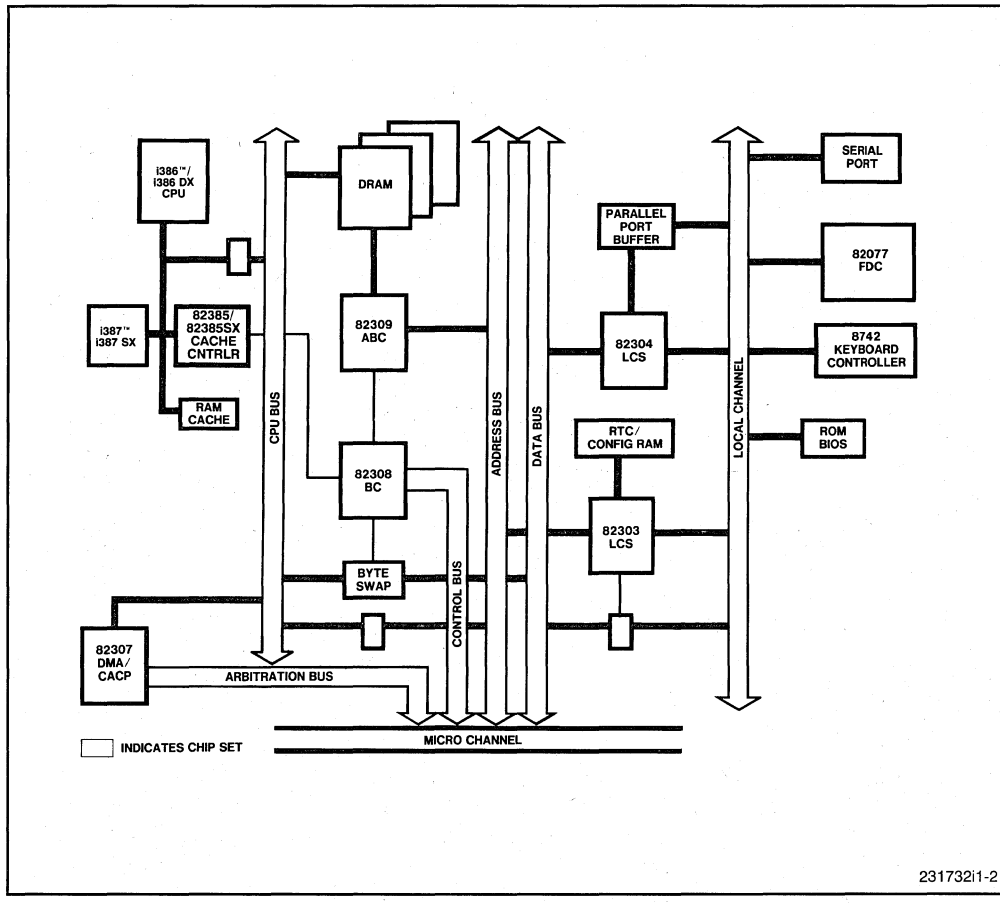


Figure 1-2. Micro Channel-Compatible Solution with 82311 Chip Set

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

The Intel386 DX microprocessor supports virtual 8086 (V86) mode. In this mode, one or more 8086 programs can be integrated into the protected, multitask environment of the Intel386 DX CPU. V86 tasks take advantage of the hardware support of multitasking offered by the Intel386 DX protected-mode, allowing multiple V86 tasks, each executing an 8086 program.

Task switching occurs frequently in real-time multitasking or multiuser systems. To perform task switching efficiently, the Intel386 DX microprocessor incorporates special high-speed hardware. Only a single instruction or an interrupt is needed for the Intel386 DX microprocessor to perform a complete task switch. A 20-MHz Intel386 DX

Table 1-1. Intel386™ Family System Components

Component	Description
Intel386™ DX Microprocessor	32-bit high-performance microprocessor with on-chip memory management and protection
Intel387™ DX Math Coprocessor	Performs numeric instruction in parallel with Intel386 DX microprocessor; expands instruction set
82380 Integrated System Peripheral	Provides 32-bit high-speed direct memory access, interrupt control, and interval timers
82385 Cache Controller	Provides cache directory and management logic
8259A Programmable Interrupt Controller	Provides interrupt control and management
82350 EISA Chip Set	Extends the 32-bit transfer capability of the Intel386 DX microprocessor to the I/O expansion bus
82311 MCA Chip Set	Provides for the design of a very high performance Micro Channel compatible PS/2 system
82596DX LAN Coprocessor	Performs high-level commands, command chaining and interprocessor communications via shared memory

microprocessor can save the state of one task (all registers), load the state of another task (all registers, even segment and paging registers if required), and resume execution in less than 14 microseconds (at 20 MHz). For less sophisticated task and interrupt handling, the latency can be as short as 2.9 microseconds (at 20 MHz).

1.2 COPROCESSORS

The performance of most applications can be enhanced by the use of specialized coprocessors. A coprocessor provides the hardware to perform functions that would otherwise be performed in software. Coprocessors extend the instruction set of the Intel386 DX microprocessor.

The Intel386 DX microprocessor has a numeric coprocessor interface designed for the Intel387 DX math coprocessor. For applications that benefit from high-precision integer and floating-point calculations, the numeric coprocessor provides full support for the IEEE standard for floating-point operations. The Intel387 DX math coprocessor is software compatible with the 80287 and the 8087, earlier numeric coprocessors.

1.3 INTEGRATED SYSTEM PERIPHERAL

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

The 82380 Integrated System Peripheral is a multi-function Intel386 DX microprocessor companion chip. It integrates a 32-bit DMA Controller with other necessary processor support functions needed in an Intel386 DX microprocessor environment. The 82380 is optimized for use with the Intel386 DX microprocessor. It enhances the overall Intel386 DX microprocessor system performance by providing high data throughput as well as efficient bus operation. The 32-bit DMA Controller provides eight independently programmable channels that can transfer data at the full bandwidth of the Intel386 DX microprocessor bus. Other features of the 82380 are listed as follows.

- High performance 32-bit DMA Controller
  - 8 independently programmable channels
  - 32 megabytes per second data transfer rate at 16 MHz
  - 40 megabytes per second data transfer rate at 20 MHz
  - Capable of transferring data between devices with different bus widths
  - Automatic byte assembly/disassembly capability for non-aligned data transfers
  - Buffer chaining capability for transferring data into non-contiguous memory buffers
- 20-Source Interrupt Controller
  - 15 external, 5 internal interrupt requests
  - 82C59A susperset
  - Individually programmable interrupt vectors
- Four 16-bit programmable Interval Timers
  - 82C54 compatible
- Programmable Wait State Generator
  - 0 to 15 wait states for memory and I/O access cycles
- DRAM Refresh Controller
  - Refresh request always has the highest priority among the DMA requests
- Intel386 DX microprocessor Shutdown Detect and Reset Control
  - Software and hardware reset
- Optimized for use with the Intel386 DX microprocessor
  - Resides on local bus for maximum bus bandwidth

1.4 CACHE CONTROLLER

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

The 82385 Cache Controller is a high performance peripheral designed specifically for the Intel386 DX microprocessor. The 82385 allows the Intel386 DX microprocessor to reach its full performance potential by offering the following features:

- Supports a 32-kbyte cache memory organized as either 2-way set associative or direct mapped.
- Integrated cache directory and management logic.
- Utilizes posted writes for zero wait states on write cycles.
- Guarantees cache coherency by bus watching.
- Supports non-cacheable accesses.
- Presents an Intel386 DX microprocessor interface to system resources.
- Dhrystone benchmark shows an average hit rate of 95%.

1.5 EISA CHIP SET

EISA extends the 32-bit transfer capability of the Intel386 DX microprocessor to the I/O expansion bus. The 82350 chip set is a highly integrated solution in a 5-piece chip set using 3 components:

- 82358 EISA Bus Controller
- 82357 Integrated System Peripheral
- 82352 EISA Bus Buffer (3 used)

The 82355 Bus Master Interface Controller (BMIC) is provided for add-in board support. The BMIC provides all of the necessary control signals, address lines and data lines for an EISA bus master to interface to the EISA bus.

The EISA specification and the 82350 chip set are both designed to be 100% backward compatible to the ISA (Industry Standard Architecture) AT-bus. Therefore, software and add-in boards designed for the ISA bus may be used in higher performance EISA systems.

This high performance, high integration 82350 EISA solution is designed to be used with the Intel486 DX and Intel386 SX microprocessors as well as the Intel386 DX microprocessor (up to and including 33 MHz).

1.6 MCA CHIP SET

The 82311 chip set consists of standard peripheral components for implementing an IBM PS/2 compatible motherboard which supports the Micro Channel Architecture. Included in the Micro Channel compatible chip set are seven highly integrated VLSI peripherals including:

- 82303 Local I/O Support Chip
- 82304 Local I/O Support Chip
- 82307 DMA/CACP Controller

- 82308 Micro Channel Bus Controller
- 82309 Address Bus Controller
- 82706 VGA Graphics Controller
- 82077 Floppy Disk Controller

The 82311 solution not only offers Micro Channel compatibility, but also high integration and performance. It features all the peripheral functions required to interface to the CPU, Micro Channel Bus, I/O Peripheral Bus and the Graphics Channel.

The 82311 chip set supports the Intel386 DX microprocessors up to 25 MHz, and the Intel386 SX microprocessor.

1.7 LAN COPROCESSOR

The 82596DX is an intelligent high performance LAN coprocessor that performs high-level commands, command chaining, and interprocessor communications via shared memory. This relieves the host CPU of many tasks associated with network control; all time critical functions are performed independently of the CPU which greatly improves performance.

The high performance 82596DX bus interface combined with the high integration of the serial interface components (82C501AD or 82521TA), allows the integration of the LAN circuitry into the motherboard. This increases overall system performance while reducing system complexity and manufacturing costs.

The 82596 product family comprises of 3 products:

- 82596DX optimized for the Intel386 DX CPU
- 82596SX optimized for the Intel386 SX CPU
- 82596CA optimized for the Intel486 DX CPU

1.8 CLOCK GENERATOR

The Clock Generator circuit (see Figure 3-16) generates timing for the Intel386 DX microprocessor and its support components. The circuit provides both the Intel386 DX microprocessor clock (CLK2) and a half-frequency clock (CLK) to indicate the internal phase of the Intel386 DX microprocessor and to drive 80286-compatible devices that may be included in the system. It can also be used to generate the RESET signal for the Intel386 DX microprocessor and other system components. Both CLK2 and CLK are used throughout this manual to describe execution times.

1.9 8086/80286 FAMILY COMPONENTS

With the appropriate interface, the Intel386 DX microprocessor can use 8086/80286 family components.

The 8259A Programmable Interrupt Controller manages interrupts for an Intel386 DX microprocessor system. Interrupts from as many as eight external sources are accepted by one 8259A; as many as 64 requests can be accommodated by cascading several 8259A chips. The 8259A resolves priority between active interrupts, then interrupts the processor and passes a code to the processor to identify the interrupting source. Programmable features of the 8259A allow it to be used in a variety of ways to fit the interrupt requirements of a particular system.

1.10 INTEL PROGRAMMABLE LOGIC DEVICES

Intel manufactures a line of high speed PLDs (Programmable Logic Devices) specifically designed for use with microprocessors such as the Intel386 DX. These devices called μPLDs (Microcomputer Programmable Logic Devices) include the 85C220 and the 85C508/85C509. Some of the design examples in this manual use these devices.

Internal Architecture 2

CHAPTER 2
INTERNAL ARCHITECTURE

The internal architecture of the Intel386 DX microprocessor consists of six functional units that operate in parallel. Fetching, decoding, execution, memory management, and bus accesses for several instructions are performed simultaneously. This parallel operation is called pipelined instruction processing. With pipelining, each instruction is performed in stages, and the processing of several instructions at different stages may overlap as illustrated in Figure 2-1. The six-stage pipelined processing of the Intel386 DX microprocessor results in higher performance and an enhanced throughput rate over non-pipelined processors.

The six functional units of the Intel386 DX microprocessor are identified as follows:

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

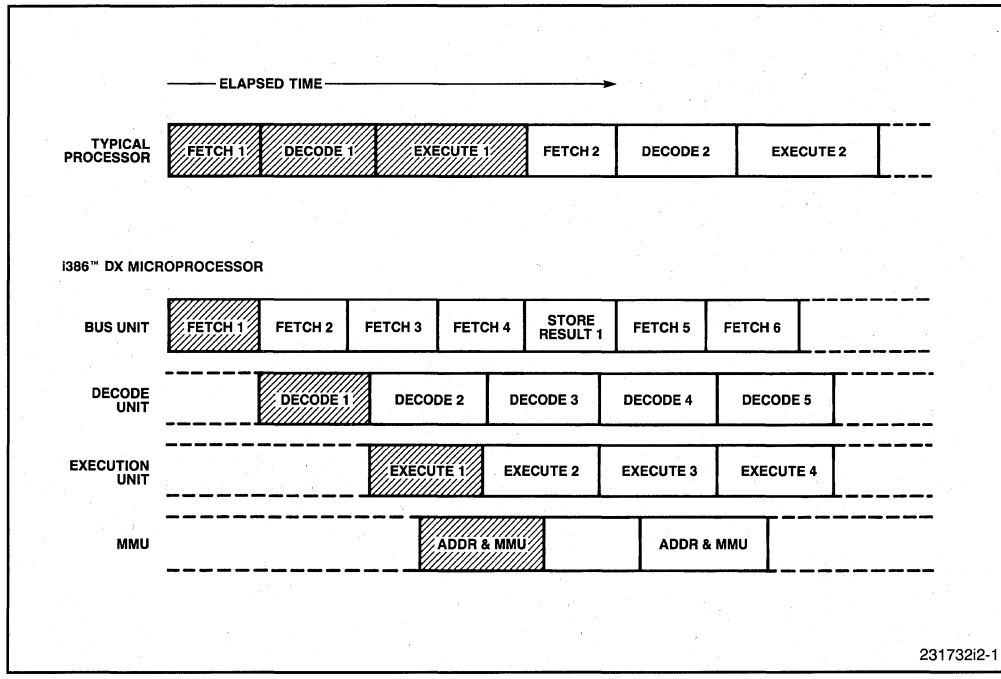


Figure 2-1. Instruction Pipelining

The Execution Unit in turn consists of three subunits:

- Control Unit
- Data Unit
- Protection Test Unit

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

2.1 BUS INTERFACE UNIT

The Bus Interface Unit provides the interface between the Intel386 DX microprocessor and its environment. It accepts internal requests for code fetches (from the Code Prefetch Unit) and data transfers (from the Execution Unit), and prioritizes the requests. At the same time, it generates or processes the signals to perform the current bus cycle. These signals include the address, data, and control outputs for accessing external memory and I/O. The Bus Interface Unit also controls the interface to external bus masters and coprocessors.

2.2 CODE PREFETCH UNIT

The Code Prefetch Unit performs the program look ahead function of the Intel386 DX microprocessor. When the Bus Interface Unit is not performing bus cycles to execute an instruction, the Code Prefetch Unit uses the Bus Interface Unit to fetch sequentially along the instruction byte stream. These prefetched instructions are stored in the 16-byte Code Queue to await processing by the Instruction Decode Unit.

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

2.3 INSTRUCTION DECODE UNIT

The Instruction Decode Unit takes instruction stream bytes from the Prefetch Queue and translates them into microcode. The decoded instructions are then stored in a three-deep Instruction Queue (FIFO) to await processing by the Execution Unit. Immediate data and opcode offsets are also taken from the Prefetch Queue. The decode unit works in parallel with the other units and begins decoding when there is a free slot in the FIFO and there are bytes in the prefetch queue. Opcodes can be decoded at a rate of one byte per clock. Immediate data and offsets can be decoded in one clock regardless of their length.

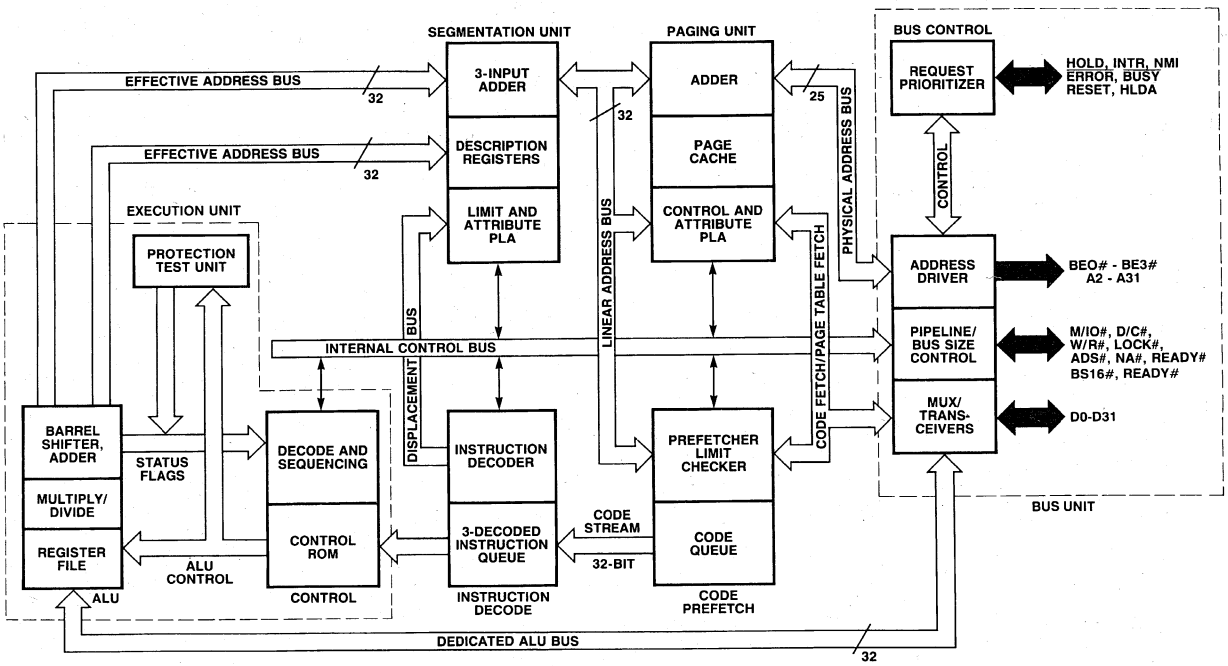


Figure 2-2. Intel386™ DX Microprocessor Functional Units

2.4 EXECUTION UNIT

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

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

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

2.5 SEGMENTATION UNIT

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

2.6 PAGING UNIT

When the Intel386 DX microprocessor paging mechanism is enabled, the Paging Unit translates linear addresses generated by the Segmentation Unit or the Code Prefetch Unit into physical addresses. (If paging is not enabled, the physical address is the same as the linear address, and no translation is necessary.) The Page Descriptor Cache stores recently used Page Directory and Page Table entries in its Translation Lookaside Buffer (TLB) to speed this translation. The Paging Unit forwards physical addresses to the Bus Interface Unit to perform memory and I/O accesses.

Local Bus Interface 3

CHAPTER 3
LOCAL BUS INTERFACE

Local bus operations are considered in this chapter. The Intel386 DX microprocessor performs a variety of bus operations in response to internal conditions and external conditions (interrupt servicing, for example). The function and timing of the signals that make up the local bus interface are described, as well as the sequences of particular local bus operations.

The high-speed bus interface of the Intel386 DX microprocessor provides high performance in any system. At 33 MHz, the Intel386 DX CPU bus transfers up to 66 Mbytes/sec. of data. At the same time, the bus control inputs and status outputs of the Intel386 DX microprocessor allow for adaptation to a wide variety of system environments.

The Intel386 DX microprocessor communicates with external memory, I/O, and other devices through a parallel bus interface. This interface consists of a data bus, a separate address bus, five bus status pins, and three bus control pins as follows:

- The bidirectional data bus consists of 32 pins (D31–D0). Either 8, 16, 24, or 32 bits of data can be transferred at once.
- The address bus, which generates 32-bit addresses, consists of 30 address pins (A31–A2) and four byte-enable pins (BE3#–BE0#). Each byte-enable pin corresponds to one of four bytes of the 32-bit data bus. The address pins identify a 4-byte location, and the byte-enable pins select the active bytes within the 4-byte location.
- The bus status pins establish the type of bus cycle to be performed. These outputs indicate the following conditions:

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

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

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

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

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

The following pins are used to control the execution of instructions in the Intel386 DX microprocessor and to interface external bus masters. The Intel386 DX microprocessor provides both a standard interface to communicate with other bus masters and a special interface to support a numerics coprocessor.

- The CLK2 input provides a double-frequency clock signal for synchronous operation. This signal is divided by two internally, so the Intel386 DX microprocessor fundamental frequency is half the CLK2 signal frequency. For example, a 20-MHz Intel386 DX microprocessor uses a 40-MHz CLK2 signal.
- The RESET input forces the Intel386 DX microprocessor to a known reset state.
- The HOLD signal can be generated by another bus master to request that the Intel386 DX microprocessor release control of the bus. The Intel386 DX microprocessor responds by activating the Hold Acknowledge (HLDA) signal as it relinquishes control of the local bus.
- The Maskable Interrupt (INTR) and Non-Maskable Interrupt (NMI) inputs cause the Intel386 DX microprocessor to interrupt its current instruction stream and begin execution of an interrupt service routine.
- The BUSY#, ERROR#, and Processor Extension Request (PEREQ) signals make up the interface to an external numeric coprocessor. BUSY# and ERROR# are status signals from the coprocessor; PEREQ allows the coprocessor to request data from the Intel386 DX microprocessor.

All of the Intel386 DX microprocessor bus interface pins are summarized in Table 3-1.

3.1 BUS OPERATIONS

There are seven types of bus operations:

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

Each bus cycle is initiated when the address is valid on the address bus, and bus status pins are driven to states that correspond to the type of bus cycle, and ADS# is driven low. Status pin states that correspond to each bus cycle type are shown in Table 3-2. Notice that the signal combinations marked as invalid states may occur when ADS# is false (high). These combinations will never occur if the signals are sampled on the CLK2 rising edge when ADS# is low, and the Intel386 DX microprocessor internal CLK is high (as indicated by the CLK output of the clock generator circuit shown in Figure 3-16). Bus status signals must be qualified with ADS# asserted (low) to identify the bus cycle.

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

Table 3-1. Summary of Intel386™ DX Microprocessor Signal Pins

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

3.1.1 Bus States

The Intel386 DX microprocessor uses a double-frequency clock input (CLK2) to generate its internal processor clock signal (CLK). As shown in Figure 3-1, each CLK cycle is two CLK2 cycles wide.

Notice that the internal Intel386 DX microprocessor matches the external CLK signal. The CLK signal is permitted to lag CLK2 slightly, but will never lead CLK2, so that it can be used reliably as a phase status indicator. All Intel386 DX microprocessor inputs are sampled at CLK2 rising edges. Many Intel386 DX microprocessor signals are sampled every other CLK2 rising edge; some are sampled on the CLK2 edge when CLK is high, while some are sampled on the CLK2 edge when CLK is low. The maximum data transfer rate for a bus operation, as determined by the Intel386 DX microprocessor internal clock, is 32 bits for every two CLK cycles, or 66 megabytes per second (CLK2 = 66 MHz, internal CLK = 33 MHz).

Table 3-2. Bus Cycle Definitions

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

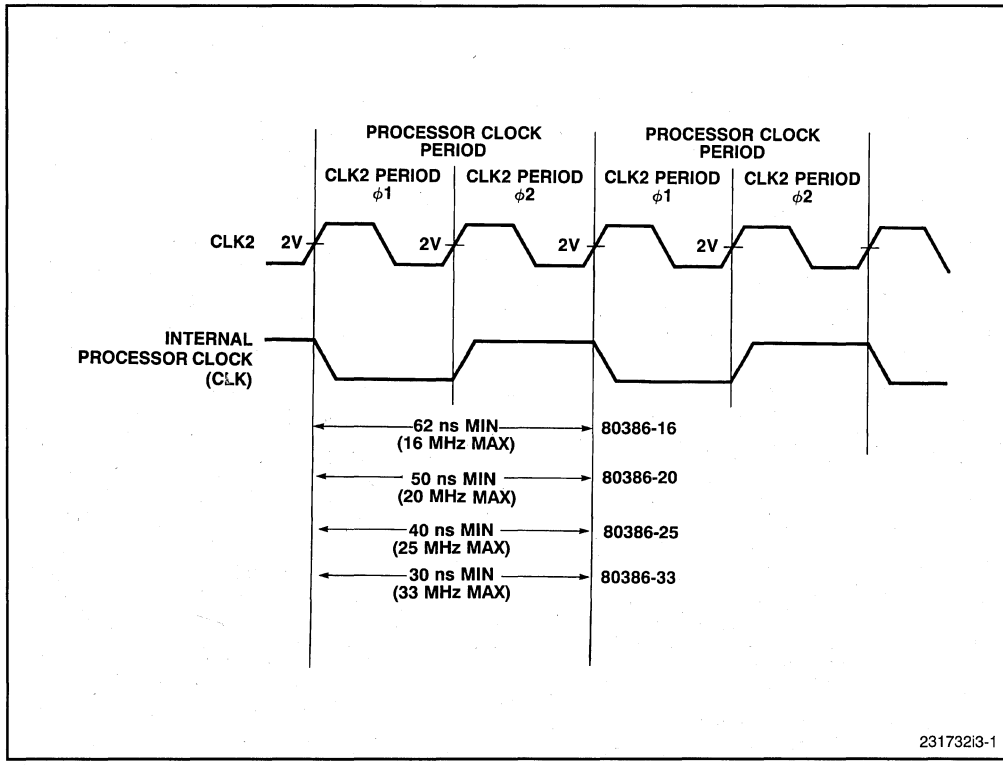
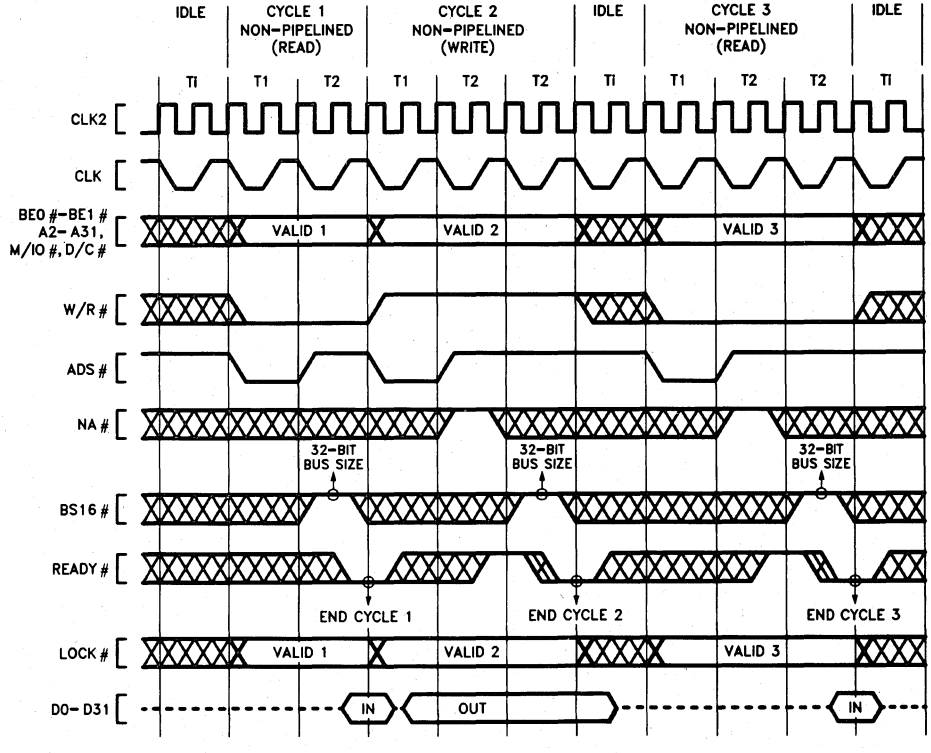


Figure 3-1. CLK2 and CLK Relationship

Each bus cycle is comprised of at least two bus states, T1 and T2. Each bus state in turn consists of two CLK2 cycles, which can be thought of as Phase 1 and Phase 2 of the bus state. Figure 3-2 shows bus states for some typical read and write cycles. During the first bus state (T1), address and bus status pins go active. During the second bus state (T2), external logic and devices respond. If the READY# input of the Intel386 DX microprocessor is sampled low at the end of the second CLK cycle, the bus cycle terminates. If READY# is high when sampled, the bus cycle continues for an additional T2 state, called a wait state, and READY# is sampled again. Wait states are added until READY# is sampled low.

When no bus cycles are needed by the Intel386 DX microprocessor (no bus requests are pending, the Intel386 DX microprocessor remains in the idle bus state (Ti). The relationship between T1, T2, and Ti is shown in Figure 3-3.



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

231732i3-2

Figure 3-2. Intel386™ DX CPU Bus States Timing Example

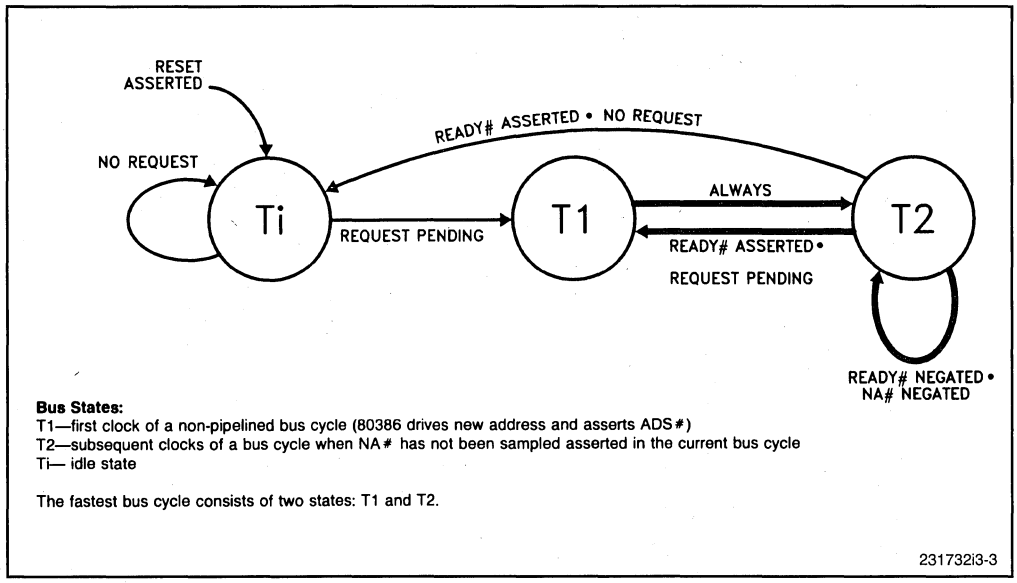


Figure 3-3. Bus State Diagram (Does Not Include Address Pipelining)

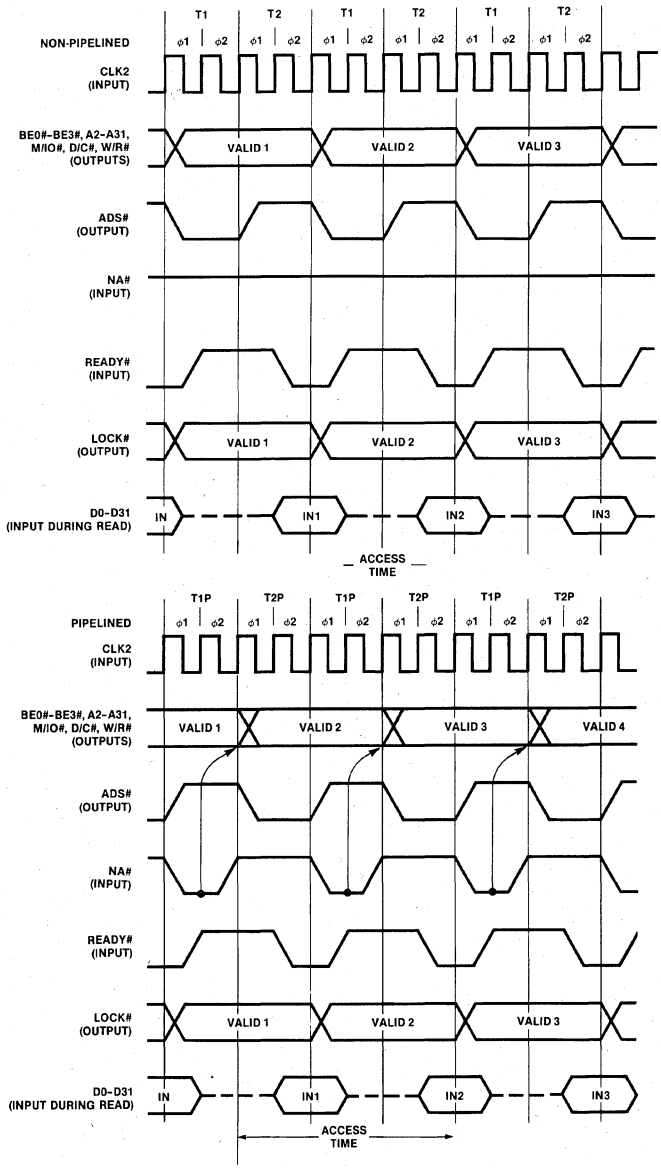
3.1.2 Address Pipelining

In the Intel386 DX microprocessor, the timing address and status outputs can be controlled so the outputs become valid before the end of the previous bus cycle. This technique, which allows bus cycles to be overlapped, is called address pipelining. Figure 3-4 compares non-pipelined address cycles to pipelined address cycles.

Address pipelining increases bus throughput without decreasing allowable memory or I/O access time, thus allowing high bandwidth with relatively inexpensive components. In addition, using pipelining to address slower devices can yield the same throughput as addressing faster devices with no pipelining. A 20-MHz Intel386 DX microprocessor can transfer data at the maximum rate of 40 megabytes per second while allowing an address access time of three CLK cycles (150 nanoseconds at CLK = 20 MHz neglecting signal delays); without address pipelining, the access time is only two CLK cycles (100 nanoseconds at CLK = 20 MHz). When address pipeline is activated following an idle bus cycle, performance is decreased slightly because the first bus cycle cannot be pipelined. This condition is explained fully in Chapter 4.

3.1.3 32-Bit Data Bus Transfers and Operand Alignment

The Intel386 DX microprocessor can address up to four gigabytes (2^{32} bytes, addresses 00000000H-FFFFFFFFH) of physical memory and up to 64 kilobytes (2^{16} bytes, addresses 00000000H-0000FFFFH) of I/O. The Intel386 DX microprocessor maintains separate physical memory and I/O spaces.



231732i3-4

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

The programmer views the address space (memory or I/O) of the Intel386 DX microprocessor as a sequence of bytes. Words consist of two consecutive bytes, and double words consist of four consecutive bytes. However, in the system hardware, address space is implemented in four sections. Each of the four 8-bit portions of the data bus (D0–D7, D8–D15, D16–D23, and D24–D31) connects to a section. When the Intel386 DX microprocessor reads a doubleword, it accesses one byte from each section. The Intel386 DX microprocessor automatically translates the programmers' view of consecutive bytes into this hardware implementation (see Figure 3-5).

The Intel386 DX microprocessor memory spaces and I/O space are organized physically as sequences of 32-bit doublewords (2^{30} 32-bit memory locations and 2^{14} 32-bit I/O ports maximum). Each doubleword starts at a physical address that is a multiple of four, and has four individually addressable bytes at consecutive addresses.

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

Data can be transferred in quantities of 32 bits, 24 bits, 16 bits, or 8 bits for each bus cycle of a data transfer. Table 3-3 shows which bytes of a 32-bit doubleword can be transferred in a single bus cycle. If a data transfer can be completed in a single cycle, the

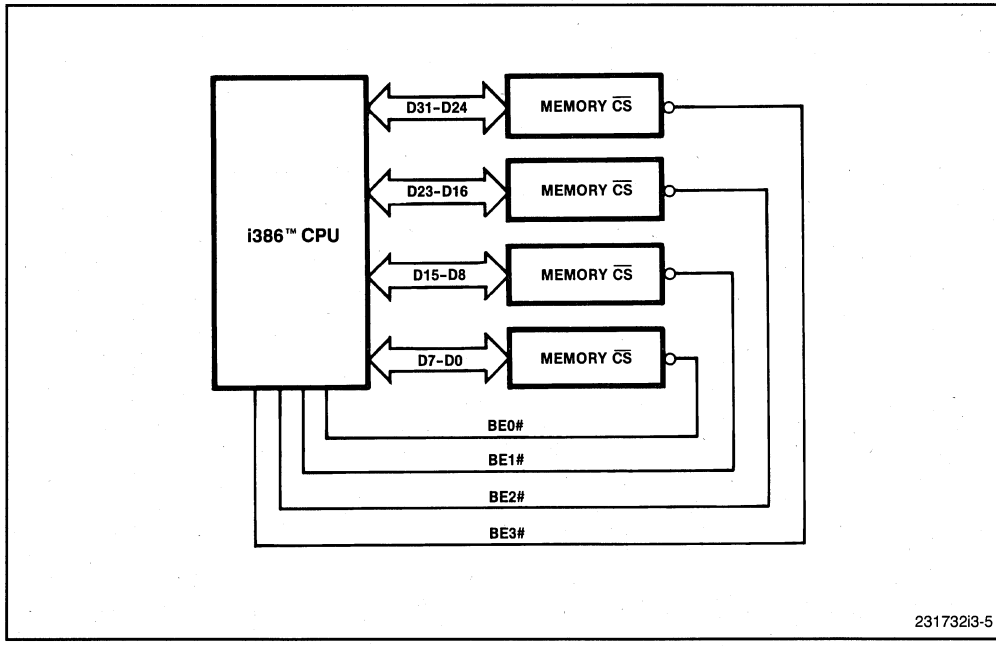


Figure 3-5. Consecutive Bytes in Hardware Implementation

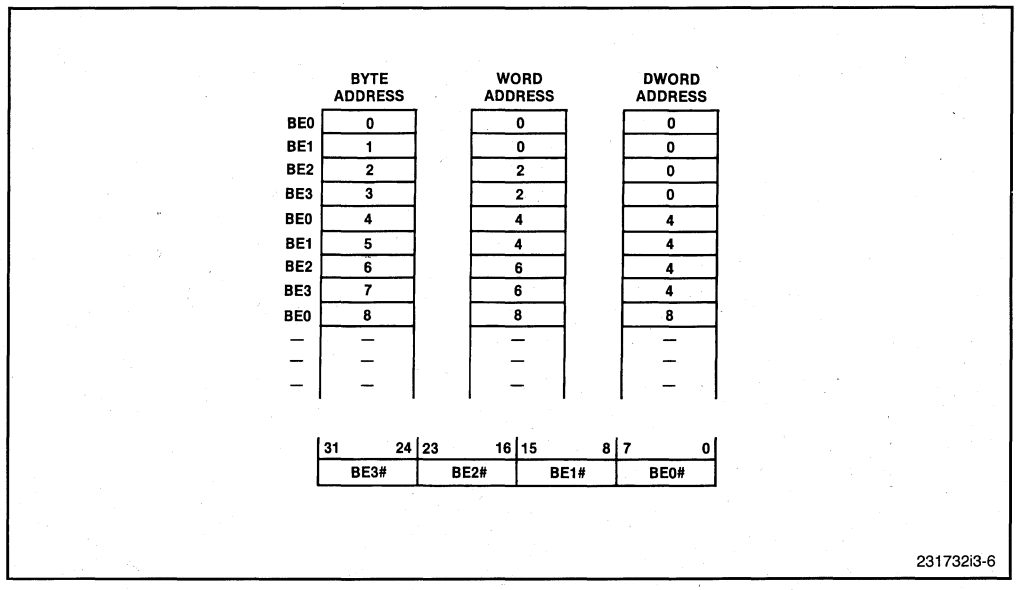


Figure 3-6. Address, Data Bus, and Byte Enables for 32-Bit Bus

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

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

transfer is said to be aligned. For example, a word transfer involving D23–D8 and activating BE1# and BE2# is aligned.

Transfers of words and doublewords that overlap a doubleword boundary of the Intel386 DX microprocessor are called misaligned transfers. These transfers require two bus cycles, which are automatically generated by the Intel386 DX microprocessor. For example, a word transfer at (byte) address 0003H requires two byte transfers: the first transfer activates doubleword address 0004H and uses D7–D0, and the second transfer activates doubleword address 0000H and uses D31–D24.

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

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

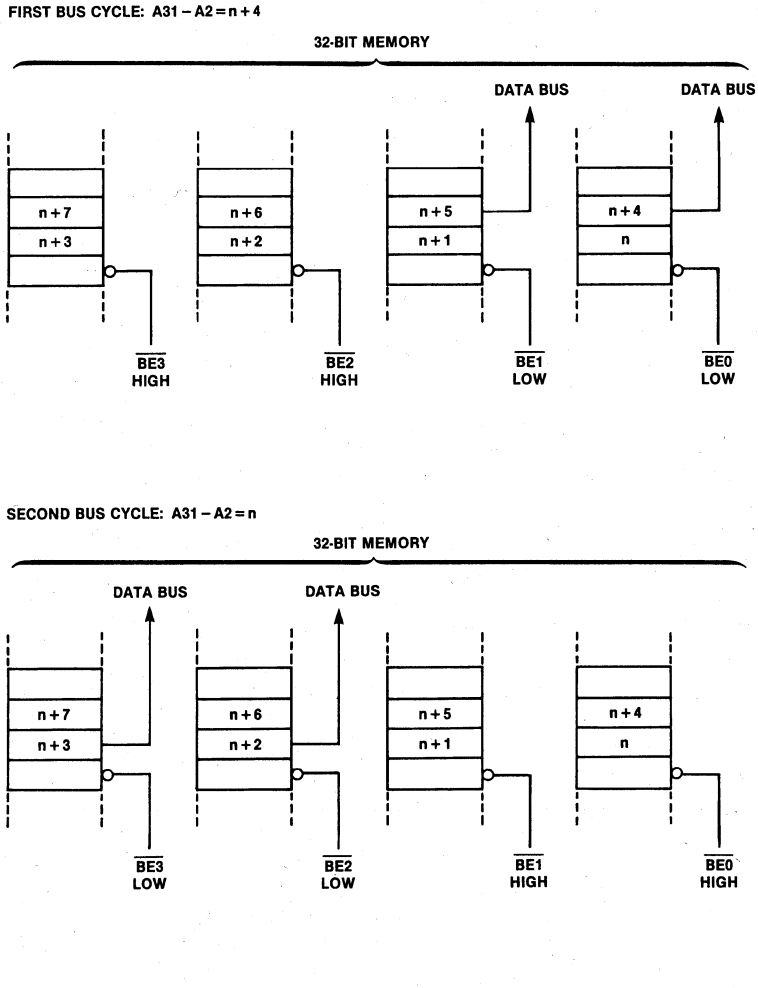


Figure 3-7. Misaligned Transfer

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

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

Because the Intel386 DX microprocessor operates on only bytes, words, and doublewords, certain combinations of BE3#–BE0# are never produced. For example, a bus cycle is never performed with only BE0# and BE2# active because such a transfer would be an operation on two noncontiguous bytes at the same time. A single 3-byte transfer will never occur, but a 3-byte transfer followed or preceded by a 1-byte transfer can occur for some misaligned doubleword transfers.

3.1.4 Read Cycle

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

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

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

- The Intel386 DX microprocessor initiates the cycle by driving ADS# low. The states of the address bus (A31–A2), byte enable pins (BE3#–BE0#), and bus status outputs (M/IO#, D/C#, W/R#, and LOCK#) at the CLK2 edge when ADS# is sampled low determine the type of bus cycle to be performed. For a read cycle,
  - W/R# is low
  - M/IO# is high for a memory read, low for an I/O read
  - For a memory read, D/C# is high if data is to be read, low if an instruction is to be read. Immediate data is included in an instruction.
  - LOCK# is low if the bus cycle is a locked cycle. In a read-modify-write sequence, both the memory data read cycle and the memory data write cycle are locked. No other bus master should be permitted to control the bus between two locked bus cycles.

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

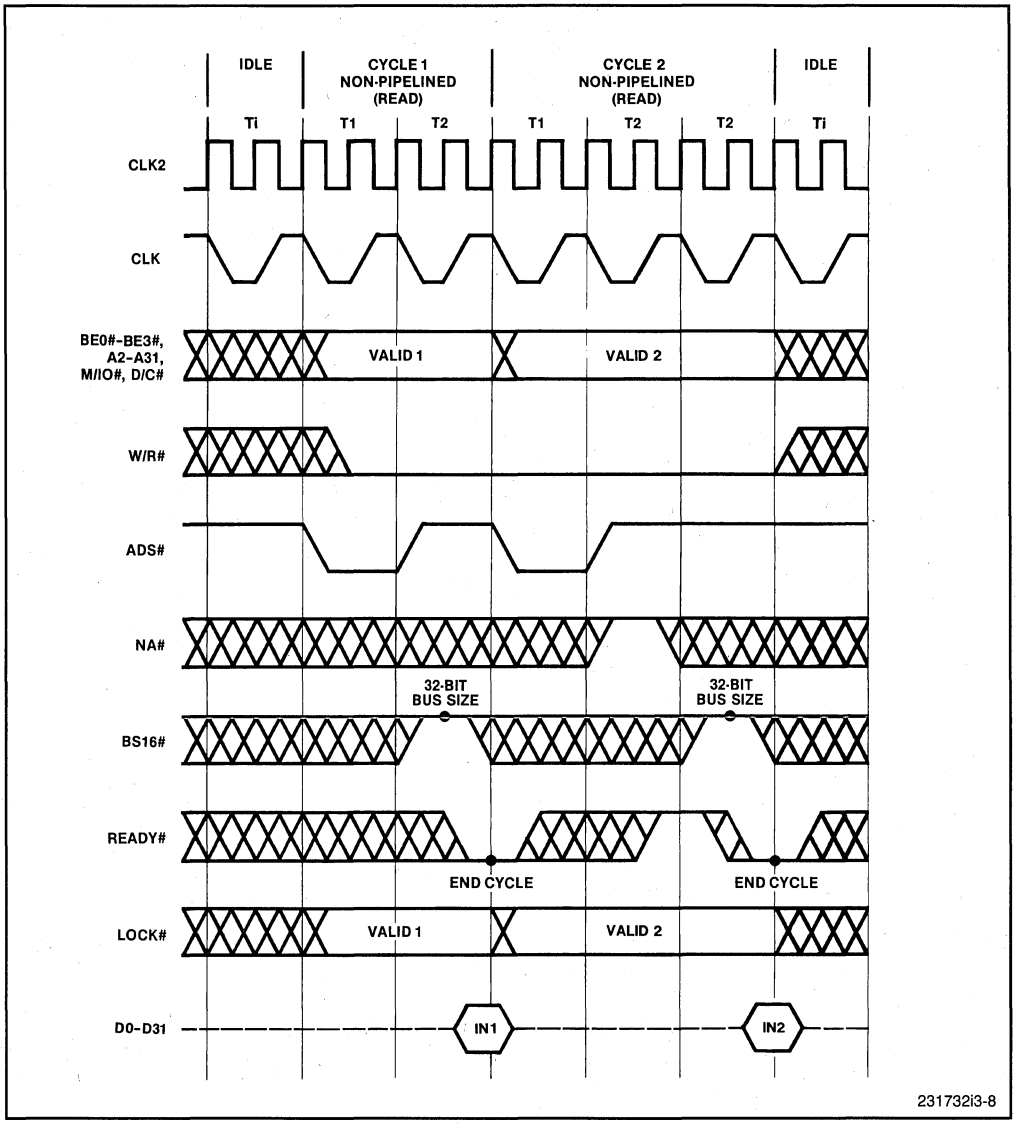


Figure 3-8. Non-Pipelined Address Read Cycles

- At the end of T2, READY# is sampled. If READY# is low, the Intel386 DX microprocessor reads the input data on the data bus.
- If READY# is high, wait states (one CLK cycle) are added until READY# is sampled low. READY# is sampled at the end of each wait state.
- Once READY# is sampled low, the Intel386 DX microprocessor reads the input data, and the read cycle terminates. If a new bus cycle is pending, it begins on the next CLK cycle.

3.1.5 Write Cycle

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

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

- The Intel386 DX microprocessor initiates the cycle by driving ADS# low. The states of the address bus (A31–A2), byte enable pins (BE3#–BE0#), and bus status outputs (M/IO#, D/C#, W/R#, and LOCK#) at the CLK edge when ADS# is sampled low to determine the type of bus cycle to be performed. For a write cycle,
  - W/R# is high
  - M/IO# is high for a memory write, low for an I/O write
  - D/C# is high
  - LOCK# is low if the bus cycle is a locked cycle. In a read-modify-write sequence, both the memory data read cycle and the memory data write cycle are locked. No other bus master should be permitted to control the bus between two locked bus cycles.

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

3.1.6 Pipelined Address Cycle

Address pipelining allows bus cycles to be overlapped, increasing the amount of time available for the memory or I/O device to respond. The NA# input of the Intel386 DX microprocessor controls address pipelining. NA# is generated by logic in the system to indicate that the address bus is no longer needed (for example, after the address has been latched). If the system is designed so that NA# goes active before the end of the cycle, address pipelining may occur.

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

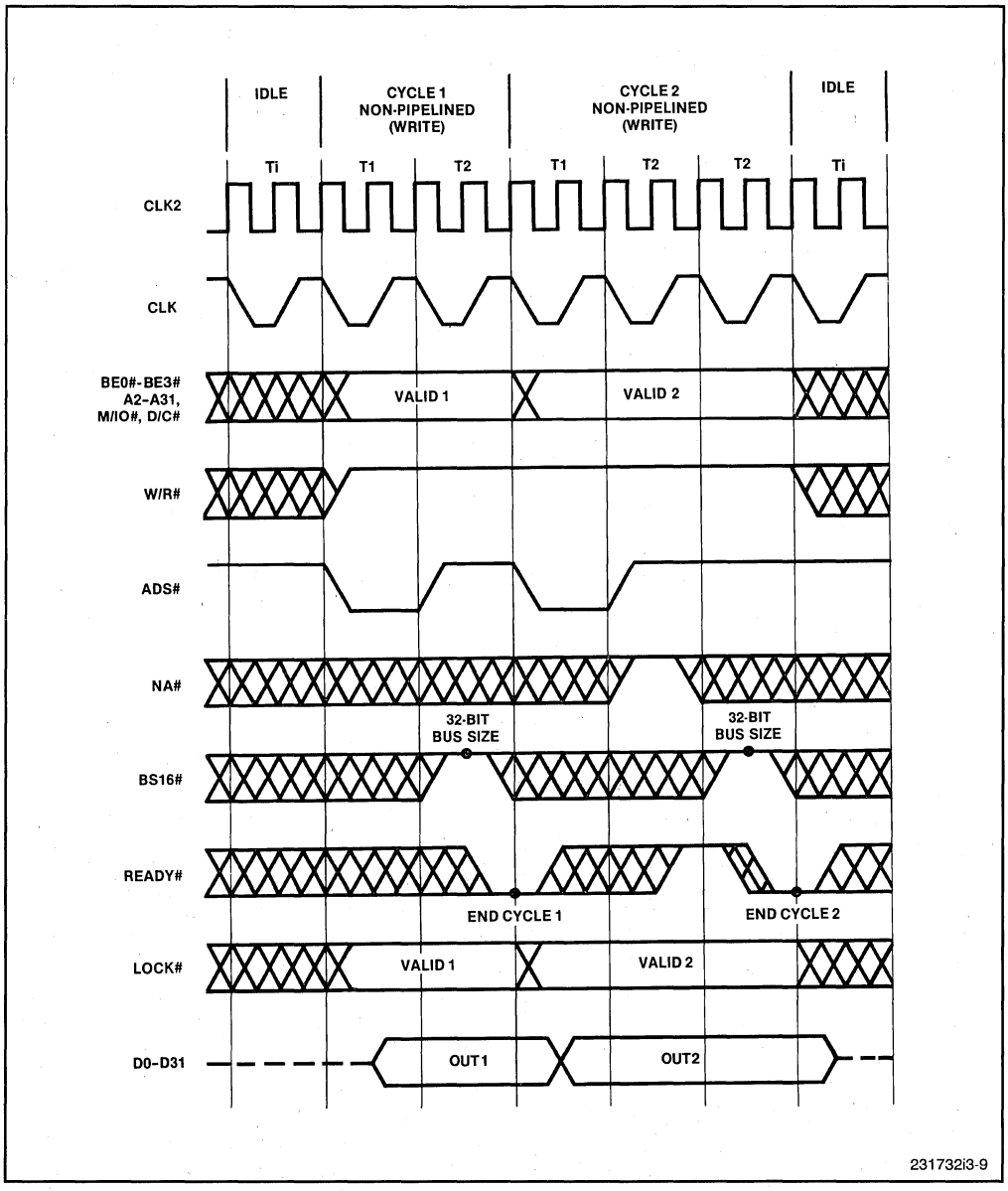
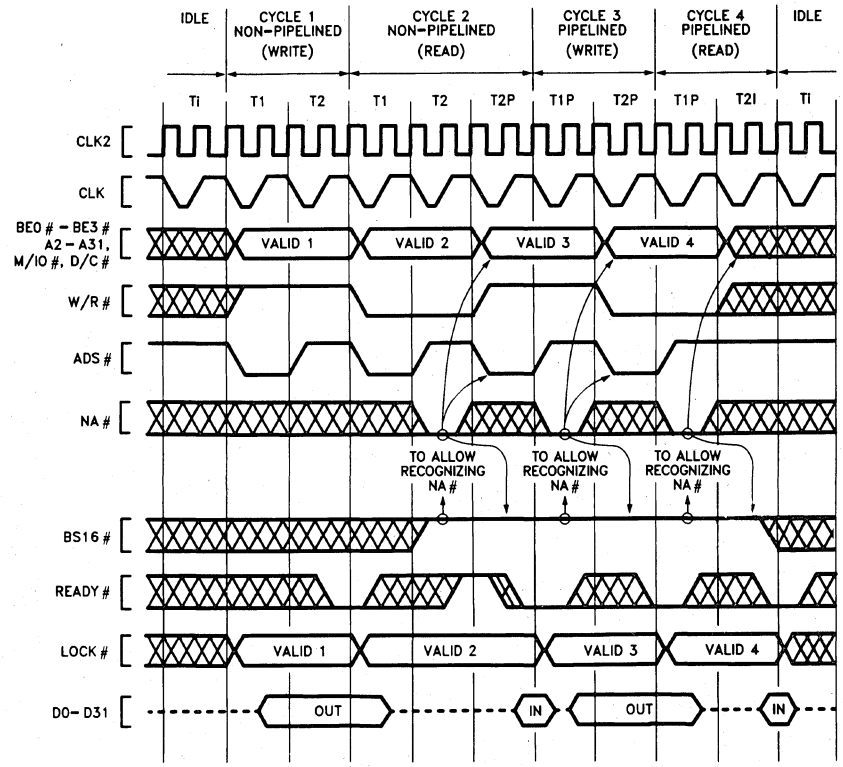


Figure 3-9. Non-Pipelined Address Write Cycles

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



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

231732i3-10

Figure 3-10. Pipelined Address Cycles

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

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

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

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

A complete discussion of the considerations for using address pipelining can be found in the *386™ DX Microprocessor Data Sheet* (Order Number 231630).

3.1.7 Interrupt Acknowledge Cycle

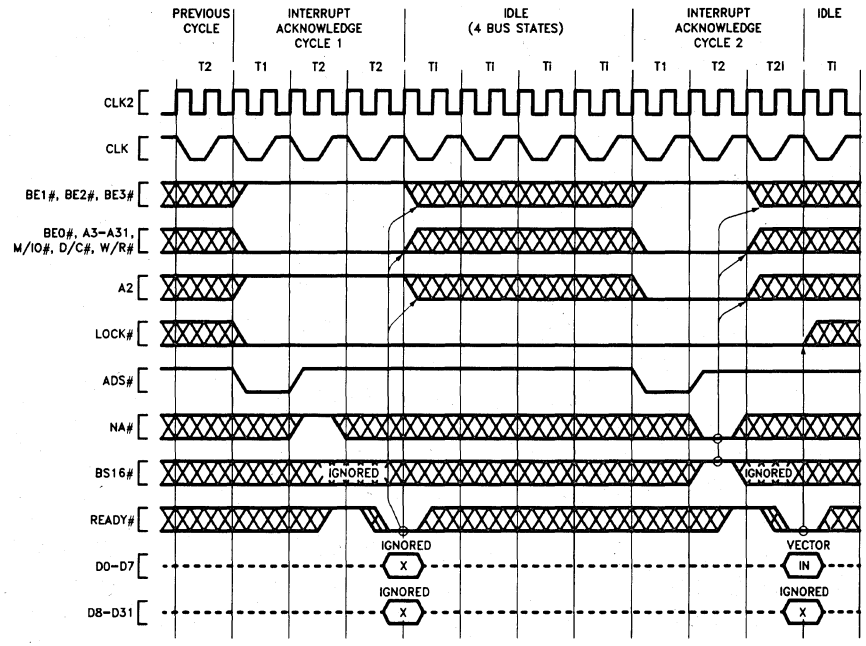
An unmasked interrupt causes the Intel386 DX microprocessor to suspend execution of the current program (after it completes the instruction it is executing) and perform instructions from another program called an interrupt service routine. Interrupts are described in detail in Section 3.4.

The 8259A Programmable Interrupt Controller is a system component that coordinates the interrupts of several devices (eight interrupts for a single 8259A; up to 64 interrupts with eight cascaded 8259As). When a device signals an interrupt request, the 8259A activates the INTR input to the Intel386 DX microprocessor.

Interrupt acknowledge cycles are special bus cycles designed to activate the 8259A INTA input. INTA signals the 8259A to output a service-routine vector on the data bus. The Intel386 DX microprocessor performs two back-to-back interrupt acknowledge cycles in response to an active INTR input (as long as the interrupt flag of the Intel386 DX microprocessor is enabled).

Interrupt acknowledge cycles are similar to regular bus cycles in that the Intel386 DX microprocessor bus outputs signals at the start of each bus cycle and an active READY# terminates each bus cycle. The cycles are shown in Figure 3-11.

- ADS# is driven low to start each bus cycle.
- Control signals M/IO#, D/C#, and W/R# are driven low to signal two interrupt-acknowledge bus cycles. These signals must be decoded to generate the INTA input signal for the 8259A. The decoding logic is usually included in the bus controller logic for the particular design. Bus controller designs are discussed in Chapters 6 and 8.
- LOCK# is active from the beginning of the first cycle to the end of the second. HOLD requests from other bus masters are not recognized until after the second interrupt acknowledge cycle.
- The address driven during the first cycle is 4; during the second cycle, the address is 0. BE3#, BE2#, and BE1# are high, BE0# is low, and A31–A3 are low for both cycles; A2 is high for the first cycle and low for the second.
- The Intel386 DX microprocessor floats D31–D0 for both cycles; however, at the end of the second cycle, the service routine vector at the 8259A outputs is read by the Intel386 DX microprocessor on pins D7–D0.
- READY# must go low to terminate each cycle.



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

231732i3-11

Figure 3-11. Interrupt Acknowledge Bus Cycles

System logic must delay READY# to extend the cycle to the minimum pulse-width requirement of the 8259A Programmable Interrupt Controller. In addition, the Intel386 DX microprocessor inserts four Ti states between the two cycles to match the recovery time of the 8259A.

3.1.8 Halt/Shutdown Cycle

The halt condition in the Intel386 DX microprocessor occurs in response to a HLT instruction. The shutdown condition occurs when the Intel386 DX microprocessor is processing a double fault and encounters a protection fault; the Intel386 DX microprocessor cannot recover and shuts down. Halt or shutdown cycles result from these conditions. Externally, a shutdown cycle differs from a halt cycle only in the resulting address bus outputs.

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

- M/IO# and W/R# are driven high, and D/C# is driven low to indicate a halt or shutdown cycle.
- All address bus outputs are driven low. For a halt condition, BE2# is active; for a shutdown condition, BE0# is active. These signals are used by external devices to respond to the halt or shutdown cycle.

READY# must be asserted to complete the halt or shutdown cycle. The Intel386 DX microprocessor will remain in the halt or shutdown condition until...

- NMI goes high; Intel386 DX microprocessor services the interrupt
- RESET goes high; Intel386 DX microprocessor is reinitialized

In the halt condition (but not in the shutdown condition), if maskable interrupts are enabled, an active INTR input will cause the Intel386 DX microprocessor to end the halt cycle to service the interrupt. The Intel386 DX microprocessor can service processor extension (PEREQ input) requests and HOLD (HOLD input) requests while in the halt or shutdown condition.

3.1.9 BS16 Cycle

The Intel386 DX microprocessor can perform data transfers for both 32-bit and 16-bit data buses. A control input, BS16#, allows the bus size to be specified for each bus cycle. This dynamic bus sizing gives the Intel386 DX microprocessor flexibility in using 16-bit components and buses.

The BS16# input causes the Intel386 DX microprocessor to perform data transfers for a 16-bit data bus (using data bus signals D15–D0) rather than a 32-bit data bus. The Intel386 DX microprocessor automatically performs two or three cycles for data transfers larger than 16 bits and for misaligned (odd-addressed) 16-bit transfers.

BS16# must be supplied by external hardware, either through chip select decoding or directly from the addressed device. BS16# is sampled at the start of Phase 2 only in CLK cycle as long as ADS# is not active. If BS16# and READY# are sampled low in the same CLK cycle, the Intel386 DX microprocessor assumes a 16-bit data bus.

The BS16# control input affects the performance of a data transfer only for data transfers in which 1) BE0# or BE1# is active and 2) BE2# or BE3# is active at the same time. In these transfers, the Intel386 DX microprocessor must perform two bus cycles using only the lower half of the data bus.

If a BS16 cycle requires an additional bus cycle, the Intel386 DX microprocessor will retain the current address for the second cycle. Address pipelining cannot be used with BS16 cycles because address pipelining requires that the next address be generated on

the bus before the end of the current bus cycle. Therefore, because both signals are sampled at the same sampling window, BS16# must be active before or at the same time as NA# to guarantee 16-bit operation. Once NA# is sampled active in a bus cycle and BS16# is not active at that time, BS16# must be negated for the remainder of the bus cycle.

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

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

3.1.10 16-Bit Byte Enables and Operand Alignment

For a 16-bit data bus, the Intel386 DX microprocessor views memory and I/O as sequences of 16-bit words. For this configuration, the Bus High Enable (BHE#), A0 or Bus Low Enable (BLE#), and A1 signals are needed. BHE# and BLE# are byte enables that correspond to two banks of memory in the same way that BE3#–BE0# correspond to four banks. A1 is added to A31–A2 to generate the addresses of 2-byte locations instead of 4-byte locations. Figure 3-14 compares the addressing configurations of 32-bit and 16-bit data buses.

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

When BS16# is sampled active, the states of BE3#–BE0# determine how the Intel386 DX microprocessor responds:

- BS16# has no effect if activated for a bus cycle in which BE3# and BE2# are inactive.

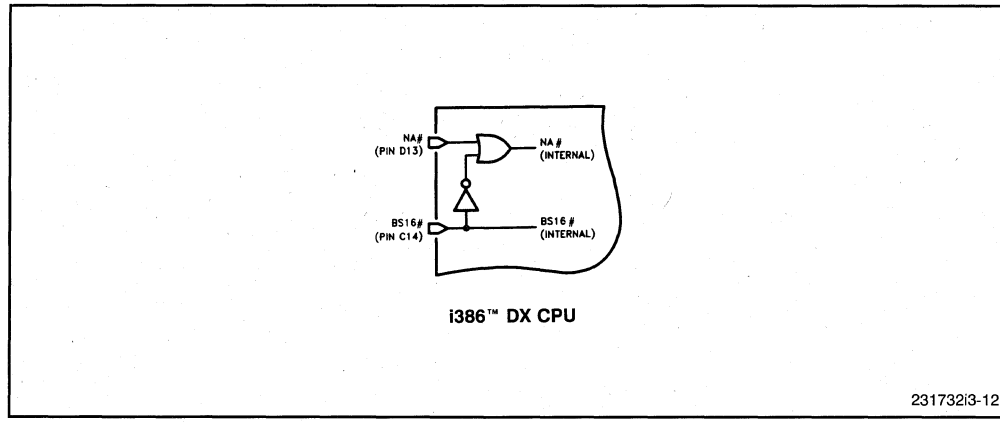


Figure 3-12. Internal NA# and BS16# Logic

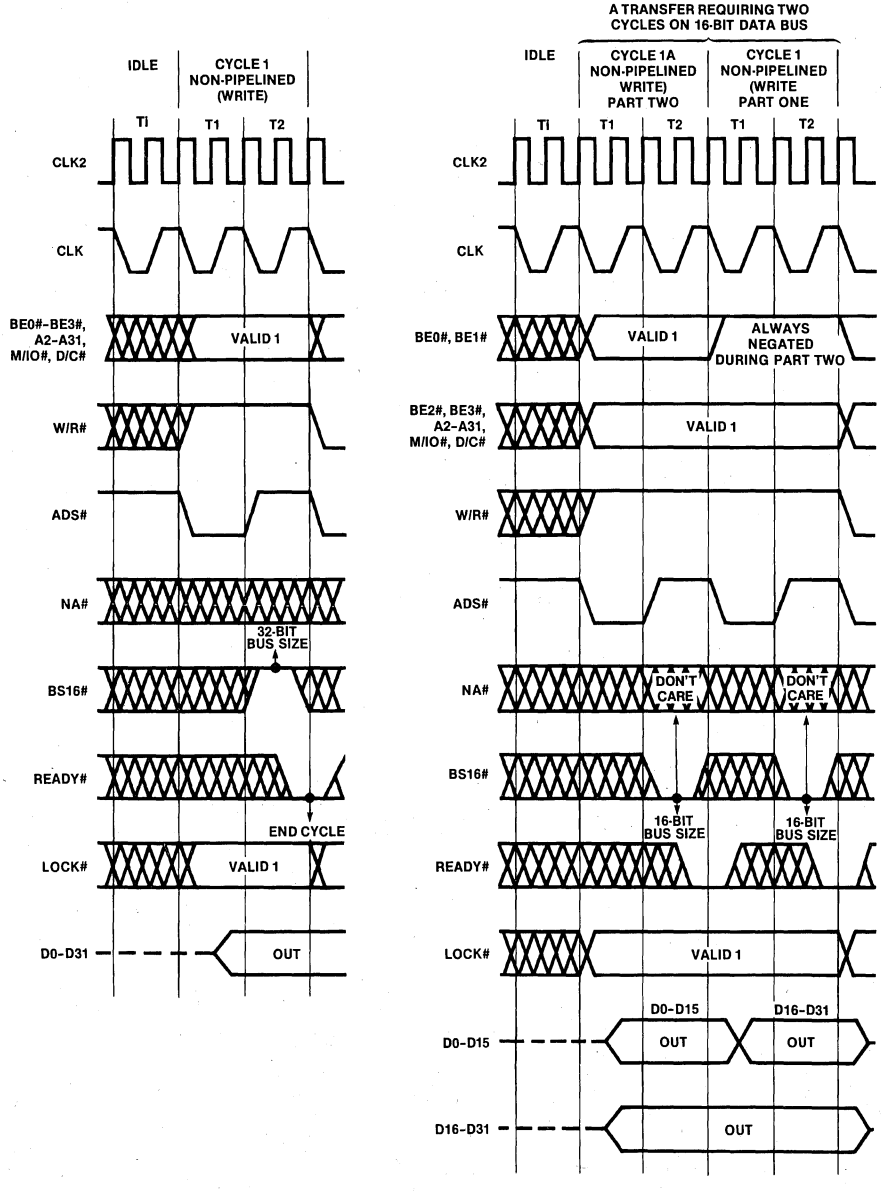


Figure 3-13. 32-Bit and 16-Bit Bus Cycle Timing

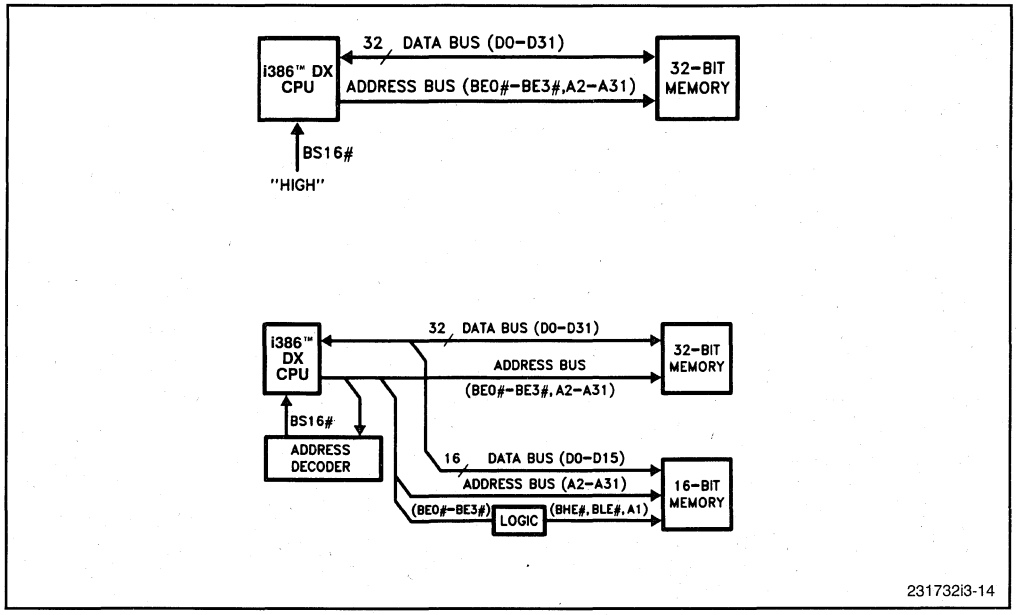


Figure 3-14. 32-Bit and 16-Bit Data Addressing

- If BE0# and BE1# are both inactive during a BS16 cycle, and either BE2# or BE3# is active,
  - For a write cycle, data on D31–D16 is duplicated on D15–D0, regardless of the state of BS16#. (This duplication occurs because BS16# is sampled late in the cycle but data must be available early.)
  - For a read cycle, data that would normally be read on D31–D24 is read on D15–D8, and data that would normally be read on D23–D16 is read on D7–D0.
- If BE0# or BE1# is active, and BE2# or BE3# is active, two bus cycles are required. The two cycles are identical except BE0# and BE1# are inactive in the second cycle and
  - For a write cycle, the data that was on D31–D16 in the first cycle is copied onto D15–D0.
  - For a read cycle, data that would normally be read on D31–D24 is read on D15–D8, and data that would normally be read on D23–D16 is read on D7–D0.

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

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

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

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

Intel386™ DX Microprocessor Signals				16-Bit Bus Signals			Comments
BE3#	BE2#	BE1#	BE0#	A1	BHE#	BLE# (A0)	
H*	H*	H*	H*	x	x	x	x—no active bytes
H	H	H	L	L	H	L	
H	H	L	H	L	L	H	
H	H	L	L	L	L	L	
H	L	H	H	H	H	L	
H*	L*	H*	L*	x	x	x	x—not contiguous bytes
H	L	L	H	L	L	H	
H	L	L	L	L	L	L	
L	H	H	H	H	L	H	
L*	H*	H*	L*	x	x	x	x—not contiguous bytes
L*	H*	L*	H*	x	x	x	x—not contiguous bytes
L*	H*	L*	L*	x	x	x	x—not contiguous bytes
L	L	H	H	H	L	L	
L*	L*	H*	L*	x	x	x	x—not contiguous bytes
L	L	L	H	L	L	H	
L	L	L	L	L	L	L	

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

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

Table 3-6. Byte Enables during BS16 Cycles

First Cycle				Second Cycle			
BE3#	BE2#	BE1#	BE0#	BE3#	BE2#	BE1#	BE0#
High	High	High	Low	No second cycle			
High	High	Low	High	No second cycle			
High	High	Low	Low	No second cycle			
High	Low	High	High	No second cycle			
High	Low	Low	High	High	Low	High	High
High	Low	Low	Low	High	Low	High	High
Low	High	High	High	No second cycle			
Low	Low	High	High	No second cycle			
Low	Low	Low	High	Low	Low	High	High
Low	Low	Low	Low	Low	Low	High	High

3.2 BUS TIMING

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

All Intel386 DX microprocessor signals have setup and hold time requirements relative to CLK2. The timings of certain signals relative to one another depends on whether address pipelining is used. These facts must be considered when determining external logic needed to facilitate bus cycles.

The analyses that follow are based on the assumption that a 20-MHz Intel386 DX microprocessor is used. If the processor is operated at a different frequency, the timings will change accordingly. Example worst-case signal parameter values from the *386™ DX Microprocessor Data Sheet* (Order Number 231630) are used; **consult the most recent data sheet to confirm these values.** Also note that delay times and setup times must be factored into the timing of system response and interaction with the Intel386 DX microprocessor to ensure comfortable margins for all critical timings.

3.2.1 Read Cycle Timing

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

Four CLK2 cycles (at 20 MHz)	100 nanoseconds
– A31–A2 output delay (maximum)	– 30 nanoseconds
– D31–D0 input setup (minimum)	– 11 nanoseconds
	59 nanoseconds

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

Non-pipelined value	59 nanoseconds
+ One CLK cycle (2 CLK2 cycles)	+ 50 nanoseconds
	109 nanoseconds

For both cases above, each wait state in the bus cycle adds 50 nanoseconds.

3.2.2 Write Cycle Timing

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

Four CLK2 cycles (at 20 MHz)	100 nanoseconds
– A31–A2 output delay (maximum)	– 30 nanoseconds
	70 nanoseconds
(With address pipelining)	(+ 50 nanoseconds)
(With N wait states)	(+ N* 50 nanoseconds)

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

Three CLK2 cycles (at 20 MHz)	75 nanoseconds
– D31–D0 output delay (maximum)	– 38 nanoseconds
	37 nanoseconds
(With N wait states)	(+ N*50 nanoseconds)

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

One CLK2 cycle	25 nanoseconds
+D31–D0 hold time (minimum)	+ 4 nanoseconds
	29 nanoseconds
(Wait states do not affect this parameter)	

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

3.2.3 READY# Signal Timing

The ready signal is ignored during the first CLK period (T1) of each bus cycle and sampled on the last rising edge of CLK2 in each CLK period thereafter until it is found active.

The amount of time from the output of valid address signals to the assertion of READY# to end a bus cycle determines how quickly external logic must generate the READY# signal. READY# must meet the Intel386 DX microprocessor setup time. In a nonpipelined address cycle, READY# signal timing is as follows:

Four CLK2 cycles	100 nanoseconds
– A31-A2 output delay (maximum)	– 30 nanoseconds
–READY# setup (minimum)	– 12 nanoseconds
	58 nanoseconds
(With address pipelining)	(+ 50 nanoseconds)
(With N wait states)	(+ N* 50 nanoseconds)

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

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

3.3 CLOCK GENERATION

3.3.1 Clock Timing

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

The internal CLK signal of the Intel386 DX microprocessor is matched to the external CLK output by the falling edge of the RESET signal. This operation is described with the RESET function in Section 3.7.

The skew between CLK2 and CLK signals is maintained at 0 nanoseconds (regardless of clock frequency). For closely timed interfaces, peripheral devices must be timed by CLK2. Devices that cannot be operated at the double-clock frequency must use the CLK output. The Intel386 DX microprocessor interface to these devices must allow for the CLK2-to-CLK skew.

The phase of the CLK output is useful for determining the beginning of a bus cycle. Because each CLK2 cycle is 25 nanoseconds (at CLK2 = 40 MHz), and bus status signal delays may be as much as 32 nanoseconds, it is impossible to tell from these status signals alone which CLK2 cycle begins the bus cycle, and therefore when to expect valid address signals. The phase of CLK can be used to make this determination. ADS# should be sampled on rising CLK2 transitions when CLK is high, i.e., at the end of phase 2 (see Figure 3-15).

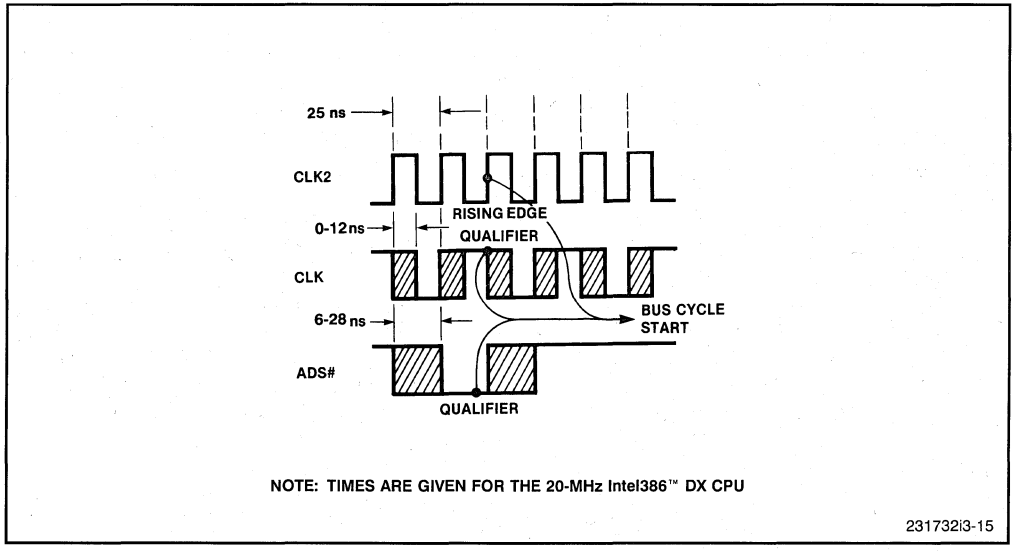


Figure 3-15. Using CLK to Determine Bus Cycle Start

3.3.2 Crystal Oscillator Clock Generator

Figure 3-16 shows a clock generator circuit for the Intel386 DX microprocessor. It is implemented with TTL and CMOS components. It provides the CLK2, CLK, and RESET signals needed by the Intel386 DX microprocessor and local bus controller. The CLK2 signal provides the fundamental timing for the Intel386 DX microprocessor and is generated by a CMOS crystal oscillator. This oscillator must have a CMOS output buffer guaranteed compatible with the Intel386 DX microprocessor CLK2 input. A 74F109 flip-flop divides CLK2 by two to generate CLK. CLK has the same phase as the Intel386 DX microprocessor internal clock, going low during phase 1 and high during phase 2. A 74F379 generates a synchronous RESET output, ensuring that the falling edge of RESET occurs during phase 2.

This circuit is recommended for designs up to 25 MHz. The timings of the 74F109 and 74F379 prohibit the use of this circuit at 33 MHz. In this case, the function of the circuit may be implemented in an E-series PAL or 85C220-80 PLD.

The recommended circuit does not implement ADS# synchronization. The ADS# synchronizer is not necessary if all registered PLDs in the local bus controller use CLK2. Clocking PLDs with CLK2 (rather than CLK) is the preferred method since it minimizes skew between the processor and its surrounding logic. If it is necessary to have an address status signal synchronous to CLK, then an ADS# synchronizer may be built using a delay line and flip-flop, as shown in Figure 3-17.

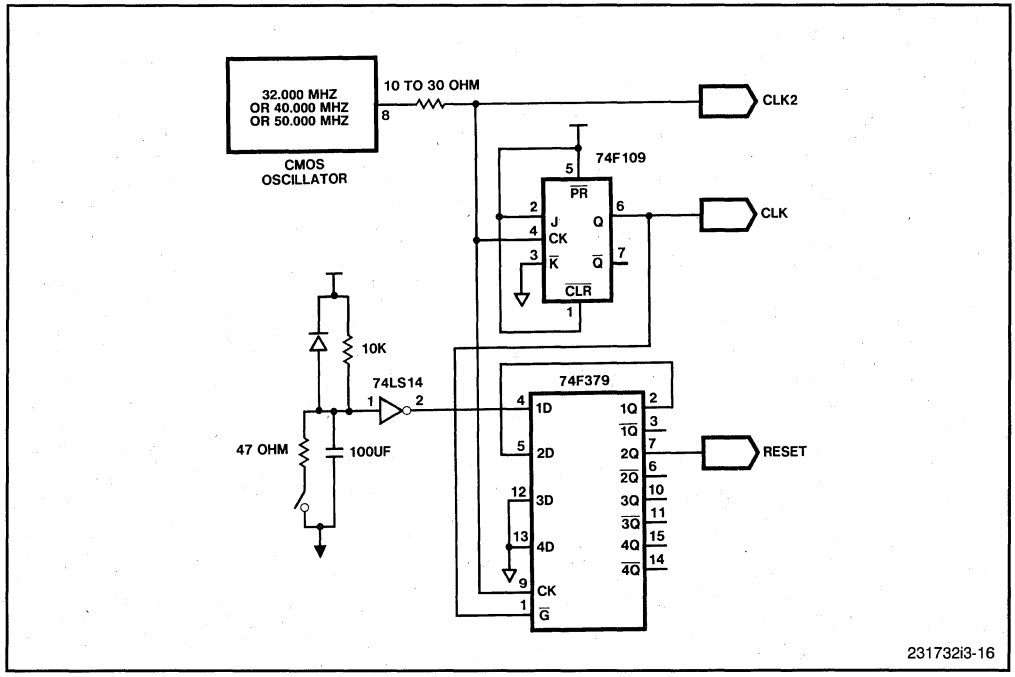


Figure 3-16. Clock Generator

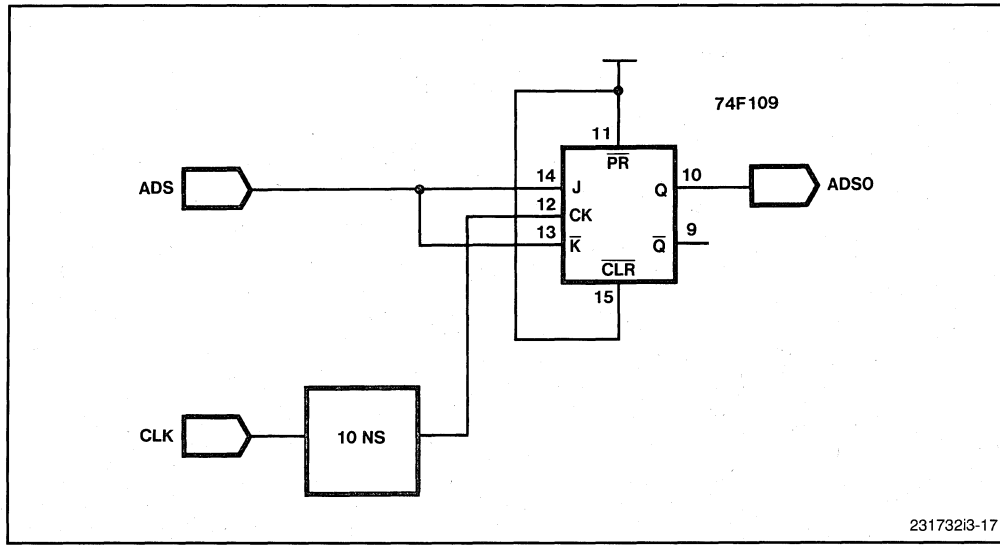


Figure 3-17. ADS# Synchronizer

An alternative method of generating CLK2 is to use a TTL oscillator coupled to a 74ACT244 buffer. Although a typical 74ACT244 datasheet does not guarantee an output compatible with the Intel386 DX microprocessor CLK2 input, some manufacturers devices have been observed to have sufficiently fast rise/fall times (4 ns at CL = 80 pf), as well as the necessary swing, to meet the Intel386 DX microprocessor CLK2 requirements. This approach is recommended if the Intel386 DX microprocessor must run synchronously with an external clock (i.e., EFI). Similarly, if a CMOS level swing is required on CLK, a 74AC109 flip-flop may be used instead of the 74F109.

3.4 INTERRUPTS

Both hardware-generated and software-generated interrupts can alter the programmed execution of the Intel386 DX microprocessor. A hardware-generated interrupt occurs in response to an active input on one of two Intel386 DX microprocessor interrupt request inputs (NMI or INTR). A software-generated interrupt occurs in response to an INT instruction or an exception (a software condition that requires servicing). For complete information on software-generated interrupts, see the *386™ DX Microprocessor Programmer's Reference Manual.*

In response to an interrupt request, the Intel386 DX microprocessor processes the interrupt (saves the processor state on the stack, plus task information if a task switch is required) and services the interrupt (transfers program execution to one of 256 possible interrupt service routines). Entry-point descriptors to service routines or interrupt tasks are stored in a table (Interrupt Descriptor Table or IDT) in memory. To access a particular service routine, the Intel386 DX microprocessor must obtain a vector, or index, to the table location that contains the corresponding descriptor. The source of this vector

depends on the type of interrupt; if the interrupt is maskable (INTR input active), the vector is supplied by the 8259A Interrupt Controller. If the interrupt is nonmaskable (NMI input active), location 2 in the IDT is used automatically.

The NMI request and the INTR request differ in that the Intel386 DX microprocessor can be programmed to ignore INTR requests (by clearing the interrupt flag of the Intel386 DX microprocessor). An NMI request always provokes a response from the Intel386 DX microprocessor unless the Intel386 DX microprocessor is already servicing a previous NMI request. In addition, an INTR request causes the Intel386 DX microprocessor to perform two interrupt-acknowledge bus cycles to fetch the service-routine vector. These bus cycles are not required for an NMI request, because the vector location for an NMI request is fixed.

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

- The incoming interrupt is an INTR request, and the Intel386 DX microprocessor is programmed to ignore maskable interrupts. (The Intel386 DX microprocessor is automatically programmed to ignore maskable interrupts when it receives any interrupt request. This condition may be changed by the interrupt service routine.) In this case, the INTR request will be serviced only if it is still active when maskable interrupts are reenabled.
- The incoming interrupt is an NMI, and the Intel386 DX microprocessor is servicing a previous NMI. In this case, the NMI is saved automatically to be processed after the IRET instruction in the NMI service routine has been executed. Only one NMI can be saved; any others that occur while the Intel386 DX microprocessor is servicing a previous NMI will not be recognized.

If neither of the above conditions is true, and an interrupt occurs while the Intel386 DX microprocessor is servicing a previous interrupt, the new interrupt is processed and serviced immediately. The Intel386 DX microprocessor then continues with the previous service routine. The last interrupt processed is the first one serviced.

If an NMI request and an INTR request arrive at the Intel386 DX microprocessor simultaneously, the NMI request is processed first. Multiple hardware interrupts arriving at the 8259A are processed according to their priority and are sent to the Intel386 DX microprocessor INTR input one at a time.

3.4.1 Non-Maskable Interrupt (NMI)

The NMI input of the Intel386 DX microprocessor generally signals a catastrophic event, such as an imminent power loss, a memory error, or a bus parity error. This input is edge-triggered (on a low-to-high transition) and asynchronous. A valid signal is low for eight CLK2 periods before the transition and high eight CLK2 periods after the transition. The NMI signal can be asynchronous to CLK2.

An NMI request automatically causes the Intel386 DX microprocessor to execute the service routine corresponding to location 2 in the IDT. The Intel386 DX microprocessor will not service subsequent NMI requests until the current request has been serviced. The Intel386 DX microprocessor disables INTR requests (although these can be reenabled in the service routine) in Real Mode. In Protected Mode, the disabling of INTR requests depends on the gate in IDT location 2.

3.4.2 Maskable Interrupt (INTR)

The INTR input of the Intel386 DX microprocessor allows external devices to interrupt Intel386 DX microprocessor program execution. To ensure recognition by the Intel386 DX microprocessor, the INTR input must be held high until the Intel386 DX microprocessor acknowledges the interrupt by performing the interrupt acknowledge sequence. The INTR input is sampled at the beginning of every instruction; it must be high at least eight CLK2 periods prior to the instruction to guarantee recognition as a valid interrupt. This requirement reduces the possibility of false inputs from voltage glitches. In addition, maskable interrupts must enabled in software for interrupt recognition. The INTR input may be asynchronous to CLK2.

The INTR signal is usually supplied by the 8259A Programmable Interrupt Controller, which in turn is connected to devices that require interrupt servicing. The 8259A, which is controlled by commands from the Intel386 DX microprocessor (the 8259A appears as a set of I/O ports), accepts interrupt requests from devices connected to the 8259A, determines the priority for transmitting the requests to the Intel386 DX microprocessor, activates the INTR input, and supplies the appropriate service routine vector when requested.

An INTR request causes the Intel386 DX microprocessor to execute two back-to-back interrupt acknowledge bus cycles, as described earlier in Section 3.1.7.

3.4.3 Interrupt Latency

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

- If interrupts are masked, an INTR request will not be recognized until interrupts are reenabled.
- If an NMI is currently being serviced, an incoming NMI request will not be recognized until the Intel386 DX microprocessor encounters the IRET instruction.
- If the Intel386 DX microprocessor is currently executing an instruction, the instruction must be completed. An interrupt request is recognized only on an instruction boundary. (However, Repeat String instructions can be interrupted after each iteration.)

- Saving the Flags register and CS:EIP registers (which contain the return address) requires time.
- If interrupt servicing requires a task switch, time must be allowed for saving and restoring registers.
- If the interrupt service routine saves registers that are not automatically saved by the Intel386 DX microprocessor, these instructions also delay the beginning of interrupt servicing.

The longest latency occurs when the interrupt request arrives while the Intel386 DX microprocessor is executing a long instruction such as multiplication, division, or a task-switch in the Protected mode.

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

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

3.5 BUS LOCK

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

Any bus cycles that must be performed back-to-back without any intervening bus cycles by other bus masters should be locked. The use of a semaphore is one example of this precept. The value of a semaphore indicates a condition, such as the availability of a device. If the Intel386 DX microprocessor reads a semaphore to determine that a device is available, then writes a new value to the semaphore to indicate that it intends to take control of the device, the read cycle and write cycle should be locked to prevent another bus master from reading from or writing to the semaphore in between the two cycles. The erroneous condition that could result from unlocked cycles is illustrated in Figure 3-18.

The LOCK# output of the Intel386 DX microprocessor signals the other bus masters that they may not gain control of the bus. In addition, an Intel386 DX microprocessor with LOCK# asserted will not recognize a HOLD request from another bus master.

3.5.1 Locked Cycle Activators

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

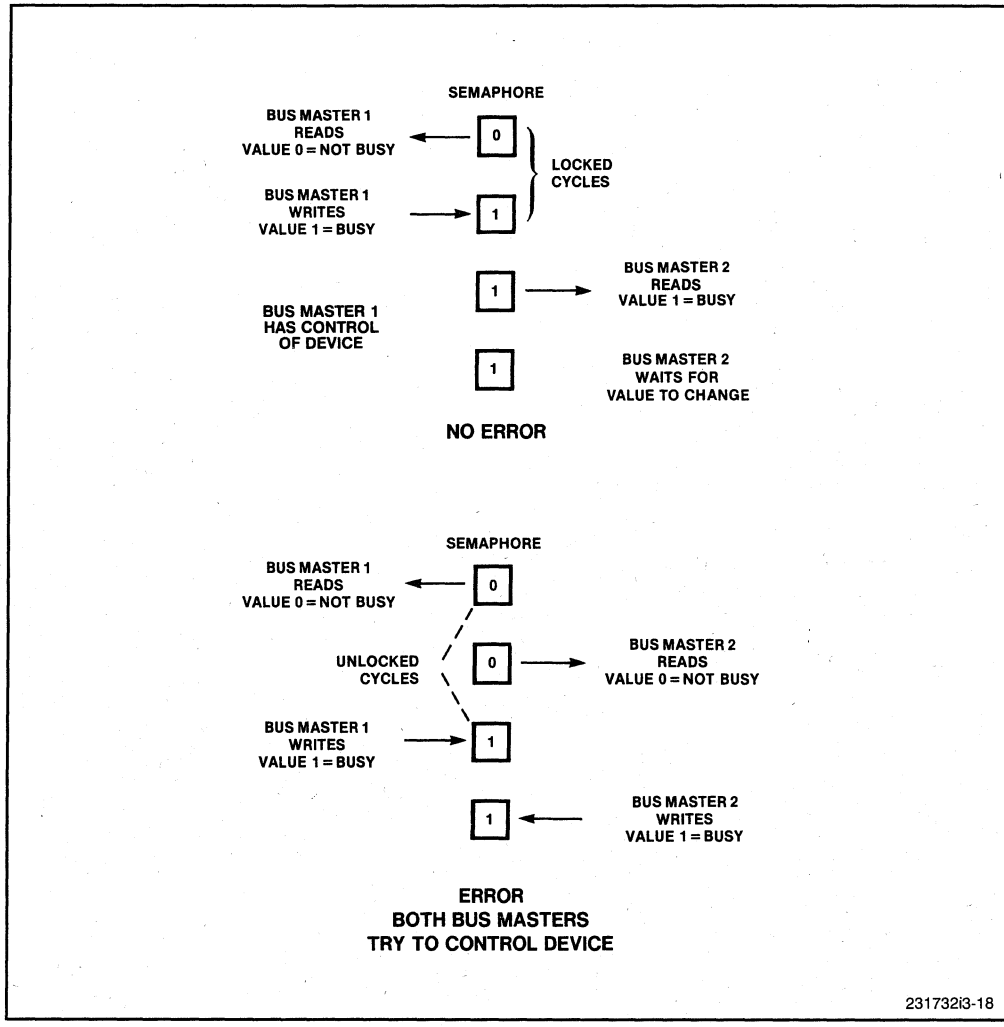


Figure 3-18. Error Condition Caused by Unlocked Cycles

3.5.2 Locked Cycle Timing

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

LOCK# is activated and deactivated on these CLK2 edges whether or not address pipelining is used. If address pipelining is used, LOCK# will remain active until after the address bus and bus cycle status signals have been asserted for the pipelined cycle. Consequently, the LOCK# signal can extend into the next memory access cycle that does not need to be locked. (See Figure 3-19.) The result is that the use of the bus by another bus master is delayed by one bus cycle.

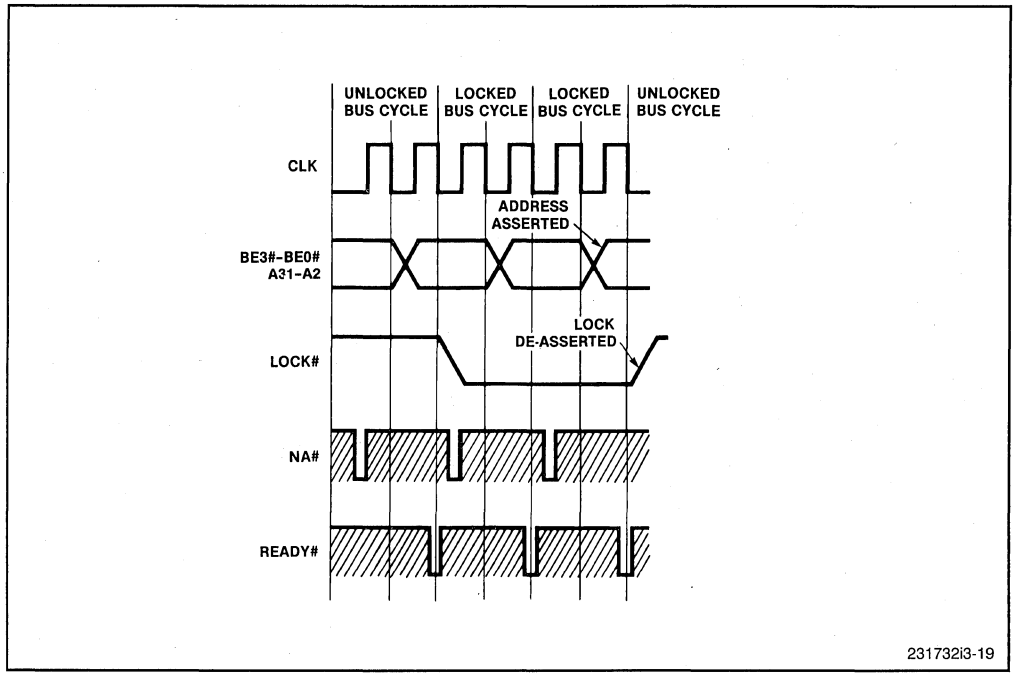


Figure 3-19. LOCK# Signal during Address Pipelining

3.5.3 LOCK# Signal Duration

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

The longest duration of LOCK# in real mode is two bus cycles plus approximately two clocks. This occurs during the XCHG instruction and in LOCKed read-modify-write operations. The longest duration of LOCK# in protected mode is five bus cycles plus approximately fifteen clocks. This occurs when an interrupt (hardware or software interrupt) occurs and the Intel386 DX microprocessor performs a LOCKed read of the gate in the IDT (8 bytes), a read of the target descriptor (8 bytes), and a write of the accessed bit in the target descriptor.

3.6 HOLD/HLDA (Hold Acknowledge)

The Intel386 DX microprocessor provides on-chip arbitration logic that supports a protocol for transferring control of the local bus to other bus masters. This protocol is implemented through the HOLD input and HLDA output.

3.6.1 HOLD/HLDA Timing

To gain control of the local bus, the requesting bus master drives the Intel386 DX microprocessor HOLD input active. This signal must be synchronous to the CLK2 input of the Intel386 DX microprocessor. The Intel386 DX microprocessor responds by completing its current bus cycle (plus a second locked cycle or a second cycle required by BS16#). Then the Intel386 DX microprocessor sets all outputs but HLDA to the three-state OFF condition to effectively remove itself from the bus and drives HLDA active to signal the requesting bus master that it may take control of the bus.

The requesting bus master must maintain HOLD active until it no longer needs the bus. When HOLD goes low, the Intel386 DX microprocessor drives HLDA low and begins a bus cycle (if one is pending).

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

When the Intel386 DX microprocessor receives an active HOLD input, it completes the current bus cycle before relinquishing control of the bus. Figure 3-20 shows the state diagram for the bus including the HOLD state.

During the HOLD state, the Intel386 DX microprocessor can continue executing instructions in its Prefetch Queue. Program execution is delayed if a read cycle is needed while the Intel386 DX microprocessor is in the HOLD state. The Intel386 DX microprocessor can queue one write cycle internally, pending the return of bus access; if more than one write cycle is needed, program execution is delayed until HOLD is released and the Intel386 DX microprocessor regains control of the bus.

HOLD has priority over most bus cycles, but HOLD is not recognized between two interrupt acknowledge cycles, between two repeated cycles of a BS16 cycle, or during locked cycles. For the Intel386 DX microprocessor, HOLD is recognized between two cycles required for misaligned data transfers; for the 8086 and 80286 HOLD it is not recognized. This difference should be considered if critical misaligned data transfers are not locked.

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

All inputs are ignored while the Intel386 DX microprocessor is in the HOLD state, except for the following:

- HOLD is monitored to determine when the Intel386 DX microprocessor may regain control of the bus.
- RESET takes precedence over the HOLD state. An active RESET input will reinitialize the Intel386 DX microprocessor.
- One NMI request is recognized and latched. It is serviced after HOLD is released.

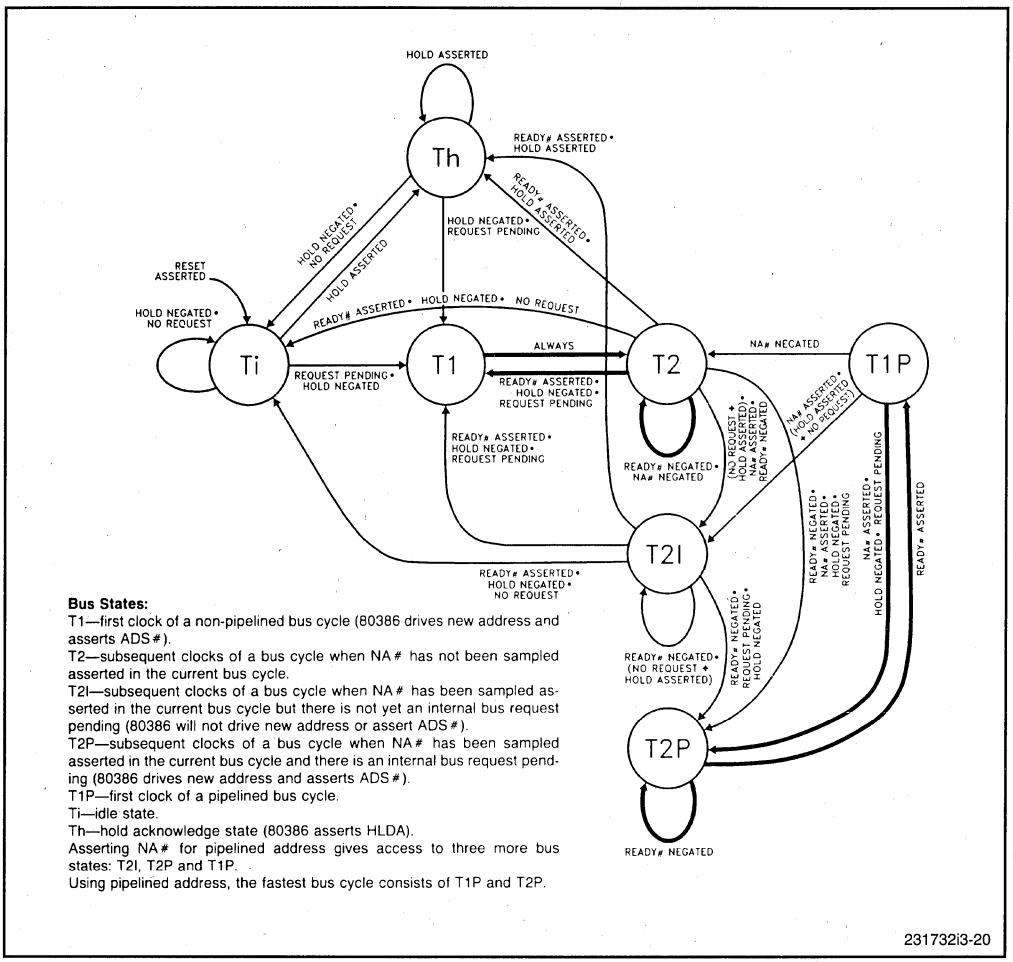


Figure 3-20. Bus State Diagram with HOLD State

3.6.2 HOLD Signal Latency

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

The minimum possible latency occurs when the Intel386 DX microprocessor receives the HOLD input during an idle cycle. HLDA is asserted on the CLK2 rising edge following the HOLD active input (synchronous to CLK2).

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

3.6.3 HOLD State Pin Conditions

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

3.7 RESET

RESET starts or restarts the Intel386 DX microprocessor. When the Intel386 DX microprocessor detects a low-to-high transition on RESET, it terminates all activities. When RESET goes low again, the Intel386 DX microprocessor is initialized to a known internal state and begins fetching instructions from the reset address.

3.7.1 RESET Timing

The clock generator generates the RESET signal to initialize the Intel386 DX microprocessor and other system components.

The RESET input of the Intel386 DX microprocessor must remain high for at least 15 CLK2 periods to ensure proper initialization (at least 80 CLK2 periods if self-test is to be performed). The CLK output of the clock generator is initialized with the rising edge of RESET. When RESET goes low, the Intel386 DX microprocessor adjusts the falling edge of its internal clock (CLK) to coincide with the start of the first CLK2 cycle after the high-to-low transition of RESET. The clock generator times the high-to-low edge of RESET (synchronous to CLK2) so that the phase of the internal CLK of the Intel386 DX microprocessor matches the phase of the CLK output of the clock generator. This relationship is shown in Figure 3-21.

On the high-to-low transition of RESET, the BUSY# pin is sampled. If BUSY# is low, the Intel386 DX microprocessor will perform a self-test lasting approximately 2^{20} + 60 CLK2 cycles before it begins executing instructions. The Intel386 DX microprocessor continues with initialization after the test, regardless of the test results.

The Intel386 DX microprocessor fetches its first instruction from linear address 0FFFFFFF0H, sometime between 350 and 450 CLK2 cycles after the high-to-low transition of RESET (or, if self-test is performed, after completion of self-test). Because paging is disabled, linear address 0FFFFFFF0H is the same as physical address 0FFFFFFF0H. This location normally contains a JMP instruction to the beginning of the bootstrap program.

3.7.2 Intel386 DX Microprocessor Internal States

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

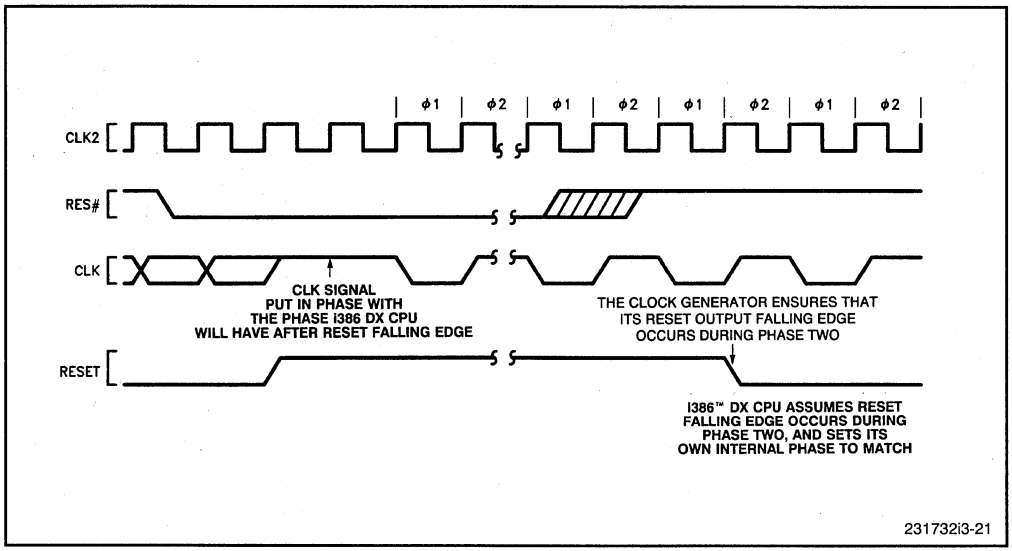


Figure 3-21. RESET, CLK, and CLK2 Timing

The Intel386 DX microprocessor samples its ERROR# input during initialization to determine the type of processor extension present in the system. This sampling occurs at some time at least 20 CLK2 periods after the high-to-low transition of RESET and before the first instruction fetch. If the ERROR# input is low, the Intel386 DX microprocessor assumes that an Intel387 DX math coprocessor is being used, and the programmer must issue a command (FINIT) to reset the ERROR# input after initialization. If ERROR# is high no processor extension is assumed.

3.7.3 Intel386 DX Microprocessor External States

RESET causes the Intel386 DX microprocessor output pins to enter the states shown in Table 3-7. Data bus pins enter the three-state condition.

Prior to its first instruction fetch, the Intel386 DX microprocessor makes no internal requests to the bus, and, therefore, will relinquish bus control if it receives a HOLD request (see Section 3.6 for a complete description of HOLD cycles).

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

Table 3-7. Output Pin States during RESET

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

Performance Considerations 4

CHAPTER 4
PERFORMANCE CONSIDERATIONS

System performance measures how fast a microprocessing system performs a given task or set of instructions. Through increased processing speed and data throughput, a Intel386 DX microprocessor operating at the heart of a system can improve overall performance immensely. The design of supporting logic and devices for efficient interaction with the Intel386 DX microprocessor is also important in optimizing system performance.

This chapter describes considerations for achieving high performance in Intel386 DX microprocessor-based systems. A variety of examples illustrate the potential performance levels for a number of applications. Two general methods can be used to match the speed of the Intel386 DX CPU to external devices: bus cycle timing or caches. Bus cycle timing includes wait states and pipelining option. Chapter 7 discusses caches.

4.1 WAIT STATES AND PIPELINING

Because a system may include devices whose response is slow relative to the Intel386 DX microprocessor bus cycle, the overall system performance is often less than the potential performance of the Intel386 DX microprocessor. Two techniques for accommodating slow devices are wait states and address pipelining. The designer must consider how to use one or both of these techniques to minimize the impact of device performance on system performance.

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

Wait states are extra CLK cycles added to the Intel386 DX microprocessor bus cycle. External logic generates wait states by delaying the READY# input to the Intel386 DX microprocessor. For an Intel386 DX microprocessor operating at 33 MHz, one wait state adds 30 nanoseconds to the time available for the memory to respond. Each wait state increases the bus cycle time by 50 percent of the zero wait-state cycle time; however, overall system performance does not vary in direct proportion to the bus cycle increase. The second column of Table 4-1 shows the performance impact (based on an example simulation) for memory accesses requiring different numbers of wait states; one wait state results in an overall performance decrease of 19 percent.

Unlike a wait state, address pipelining increases the time that a memory has to respond by one CLK cycle without lengthening the bus cycle. This extra CLK cycle eliminates the output delay of the Intel386 DX microprocessor address and status outputs. Address pipelining overlaps the address and status outputs of the next bus cycle with the end of the current bus cycle, lengthening the address access time by one or more CLK cycles from the point of view of the accessed memory device. An access that requires two wait

Table 4-1. Intel386™ DX Microprocessor Performance with Wait States and Pipelining

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

states without address pipelining would require one wait state with address pipelining. The third column of Table 4-1 shows performance with pipelining for different wait-state requirements.

Address pipelining is advantageous for most bus cycles, but if the next address is not available before the current cycle ends, the Intel386 DX microprocessor cannot pipeline the next address, and the bus timing is identical to a non-pipelined bus cycle. Also, the first bus cycle after an idle bus must always be non-pipelined because there is no previous cycle in which to output the address early. If the next cycle is to be pipelined, the first cycle must be lengthened by at least one wait state so that the address can be output before the end of the cycle.

With the Intel386 DX microprocessor, address pipelining is optional so that bus cycle timing can be closely tailored to the access time of the memory device; pipelining can be activated once the address is latched externally or not activated if the address is not latched.

The Intel386 DX microprocessor NA# input controls address pipelining. When the system no longer requires the Intel386 DX microprocessor to drive the address of the current bus cycle (in most systems, when the address has been latched), the system can activate the Intel386 DX microprocessor NA# input. The Intel386 DX microprocessor outputs the address and status signals for the next bus cycle on the next CLK cycle.

The system must activate the NA# signal without knowing which device the next bus cycle will access. In an optimal Intel386 DX microprocessor system, address pipelining should be used even for fast memory that does not require pipelining, because if a fast memory access is followed by a pipelined cycle to slower memory, one wait state is saved. If a fast memory access is followed by another fast memory access, the extra time is not used, and no processor time is lost. Therefore, all devices in a system must be able to accept both pipelined and non-pipelined cycles.

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

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

A third and less common approach to accommodating memory speed is reducing the Intel386 DX microprocessor operating frequency. Because a slower clock frequency increases the bus cycle time, fewer wait states may be required for particular memory devices. At the same time, however, system performance depends directly on the Intel386 DX microprocessor clock frequency; execution time increases in direct proportion to the increase in clock period (reduction in clock frequency).

The design and application determine whether frequency reduction makes sense. In some instances, a slight reduction in clock frequency reduces the wait-state requirement and increases system performance. Table 4-2 shows that a 25-MHz Intel386 DX microprocessor operating with zero wait states yields better performance than a 33-MHz Intel386 DX microprocessor operating with two wait states.

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

Number of Wait States	33 MHz Without Pipelining	33 MHz with Pipelining	25 MHz Without Pipelining	25 MHz With Pipelining
0	1.00	0.91	0.76	0.69
1	0.81	0.76	0.61	0.56
2	0.66	0.63	0.50	0.48
3	0.57	–	0.43	–

5 Coprocessor Hardware Interface

CHAPTER 5
COPROCESSOR HARDWARE INTERFACE

A numeric coprocessor enhances the performance of an Intel386 DX microprocessor system by performing numeric instructions in parallel with the Intel386 DX microprocessor. The Intel386 DX microprocessor automatically passes on these instructions to the coprocessor as it encounters them.

The Intel387 DX math coprocessor performs 32-bit data transfers and interfaces directly with the Intel386 DX microprocessor. The Intel387 DX math coprocessor supports the instruction set of both the 80287 and the 8087, offering additional enhancements that include full compatibility with the IEEE Floating-Point Standard, 754-1985. The performance of a 16-MHz Intel387 DX math coprocessor is about eight times faster than that of a 5-MHz 80287.

The Intel386 DX microprocessor samples its ERROR# input during initialization to determine if a coprocessor is present. Very little logic or board space is required to support a numerics option. The math coprocessor can be an option. A socket can be designed on the board such that the Intel387 DX math coprocessor is a user installed option.

Data transfers to and from a coprocessor are accomplished through I/O addresses 800000F8H and 800000FCH; these addresses are automatically generated by the Intel386 DX microprocessor for coprocessor instructions and allow simple chip-select generation using A31 (high) and M/IO# (low). Because A31 is high for coprocessor cycles, the coprocessor addresses lie outside the range of the programmed I/O address space and are easy to distinguish from programmed I/O addresses. Coprocessor usage is independent of the I/O privilege level of the Intel386 DX microprocessor.

The Intel386 DX microprocessor has three input signals for controlling data transfer to and from an Intel387 DX math coprocessor: BUSY#, Processor Extension Request (PEREQ), and ERROR#. These signals, which are level-sensitive and may be asynchronous to the CLK2 input of the Intel386 DX microprocessor, are described as follows:

- BUSY# indicates that the coprocessor is executing an instruction and therefore cannot accept a new one. When the Intel386 DX microprocessor encounters any coprocessor instruction except FNINIT and FNCLEX, the BUSY# input must be inactive (high) before the coprocessor accepts the instruction. A new instruction therefore cannot overrun the execution of the current coprocessor instruction. (Certain Intel387 DX math coprocessor instructions can be transferred when BUSY# is active (low). These instructions are queued and do not interfere with the current instruction.)

- PEREQ indicates that the coprocessor needs to transfer data to or from memory. Because the coprocessor is never a bus master, all input and output data transfers are performed by the Intel386 DX microprocessor. PEREQ always goes inactive before BUSY# goes inactive.

- ERROR# is asserted after a coprocessor math instruction results in an error that is not masked by the coprocessor's control register. The data sheets for the Intel387 DX math coprocessor describe these errors and explain how to mask them under program control. If an error occurs, ERROR# goes active before BUSY# goes inactive, so that the Intel386 DX microprocessor can take care of the error before performing another data transfer.

5.1 Intel387 DX MATH COPROCESSOR INTERFACE

The Intel387 DX math coprocessor achieves significant enhancements in performance and instruction capabilities over the 80287. To achieve maximum speed, the interface with the Intel386 DX microprocessor is synchronous and includes a full 32-bit data bus. Detailed information on other Intel387 DX math coprocessor enhancements can be found in the *387™ DX Math Coprocessor Data Sheet.*

The Intel387 DX math coprocessor is designed to run either fully synchronously or pseudosynchronously with the Intel386 DX microprocessor. In the pseudosynchronous mode, the interface logic of the Intel387 DX math coprocessor runs with the clock signal of the Intel386 DX microprocessor, whereas internal logic runs with a different clock signal.

5.1.1 Intel387 DX Math Coprocessor Connections

The connections between the Intel386 DX microprocessor and the Intel387 DX math coprocessor are shown in Figure 5-1 and are described as follows:

- The Intel387 DX math coprocessor BUSY#, ERROR#, and PEREQ outputs are connected to corresponding Intel386 DX microprocessor inputs.
- The Intel387 DX math coprocessor RESETIN input is connected to the system's RESET signal.
- The Intel387 DX math coprocessor Numeric Processor Select chip-select inputs (NPS1# and NPS2) are connected directly to the Intel386 DX microprocessor M/IO# and A31 outputs, respectively. For coprocessor cycles, M/IO# is always low; A31, high.
- The Intel387 DX math coprocessor Command (CMD0#) input differentiates data from commands. This input is connected directly to the Intel386 DX microprocessor A2 output. The Intel386 DX microprocessor outputs address 800000F8H when writing a command or reading status, address 800000FCH when writing or reading data.
- All 32 bits (D31–D0) of the Intel386 DX microprocessor data bus connect directly to the data bus of the Intel387 DX math coprocessor. Because the data lines are connected directly, any local data bus transceivers must be disabled when the Intel386 DX microprocessor reads data from the Intel387 DX math coprocessor.

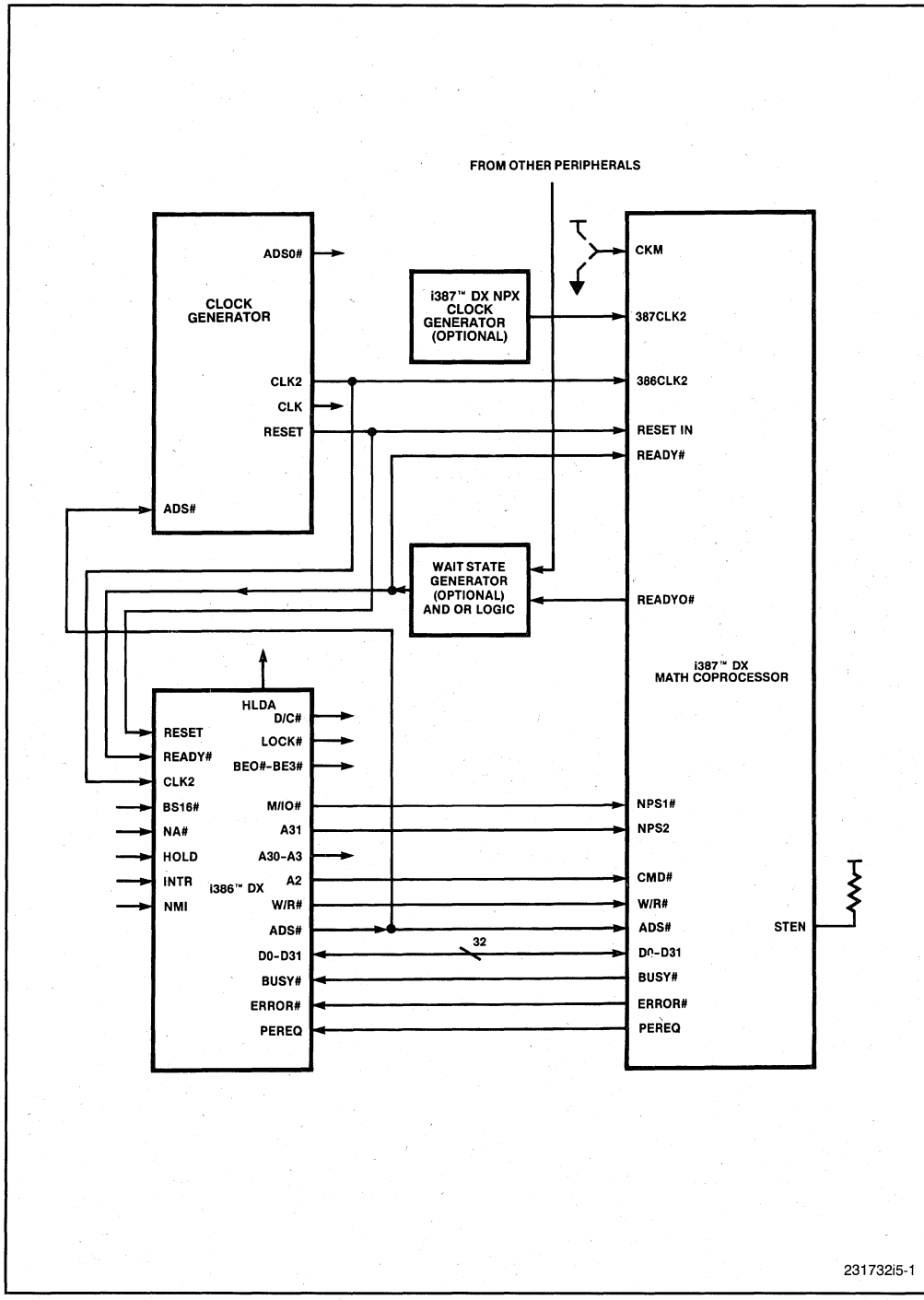


Figure 5-1. Intel386™ DX CPU System with Intel387™ DX Math Coprocessor

- The Intel387 DX math coprocessor READY#, ADS#, and W/R# inputs are connected to the corresponding pins on the Intel386 DX microprocessor. READY# and ADS# are used by the Intel387 DX math coprocessor to track bus activity and determine when W/R#, NPS1#, NPS2, and Status Enable (STEN) can be sampled.
- Status Enable (STEN) serves as a chip select for the Intel387 DX math coprocessor. This pin is high to enable the Intel387 DX math coprocessor, and may be driven low to float all Intel387 DX math coprocessor outputs. STEN may be used to do onboard testing (using the overdrive method). STEN may also be used to activate one Intel387 DX math coprocessor at a time, in systems with multiple Intel387 DX math coprocessors. If not needed, STEN should be pulled high.
- Ready Out (READYO#) can be used to acknowledge Intel387 DX math coprocessor bus cycles. The Intel387 DX math coprocessor activates READYO# at such a time that write cycles are terminated after two clocks and read cycles are terminated after three clocks. READYO# can be connected to the Intel386 DX microprocessor READY# input through logic that ORs READY# signals from other devices. Alternatively, READYO# can be left disconnected, and external logic can be used to acknowledge Intel387 DX math coprocessor bus cycles.

5.1.2 Intel387 DX Math Coprocessor Bus Cycles

When the Intel386 DX microprocessor encounters a coprocessor instruction, it automatically generates one or more I/O cycles to addresses 800000F8H and 800000FCH. The Intel386 DX microprocessor will perform all necessary bus cycles to memory and transfer data to and from the Intel387 DX math coprocessor. All Intel387 DX math coprocessor transfers are 32 bits wide. If the memory subsystem is only 16 bits wide, the Intel386 DX microprocessor automatically performs the necessary conversion before transferring data to or from the Intel387 DX math coprocessor. Since the Intel387 DX math coprocessor is a 32-bit device, BS16# must not be asserted during Intel387 DX math coprocessor communication cycles.

Read cycles (transfers from the Intel387 DX math coprocessor to the Intel386 DX microprocessor) require at least one wait state, whereas write cycles to the Intel387 DX math coprocessor require no wait states. This requirement is automatically reflected in the state of the READYO# output of the Intel387 DX math coprocessor, which can be used to generate the necessary wait state.

5.1.3 Intel387 DX Math Coprocessor Clock Input

The Intel387 DX math coprocessor can be operated in two modes. In either mode, the CLK2 signal must be connected to the 386CLK2 input of the Intel387 DX math coprocessor because the interface to the Intel386 DX microprocessor is always synchronous. The state of the Intel387 DX math coprocessor CKM input determines its mode:

- In synchronous mode, CKM is high and the 387CLK2 input is not connected. The Intel387 DX math coprocessor operates from the CLK2 signal. Operation of the Intel387 DX math coprocessor is fully synchronous with that of the Intel386 DX microprocessor.

- In pseudo-synchronous mode, CKM is low and a frequency source for the 387CLK2 input must be provided. Only the interface logic of the Intel387 DX math coprocessor is synchronous with the Intel386 DX microprocessor. The internal logic of the Intel387 DX math coprocessor operates from the 387CLK2 clock source, whose frequency may be 10/16 to 14/10 times the speed of CLK2. Figure 5-2 depicts pseudo-synchronous operation.

5.2 LOCAL BUS ACTIVITY WITH THE Intel387 DX MATH COPROCESSOR

The Intel387 DX math coprocessor uses two distinct methods to interact with the Intel386 DX microprocessor:

- The Intel386 DX microprocessor initiates coprocessor operations during the execution of a coprocessor instruction (an ESC instruction). These interactions occur under program control.
- The coprocessor uses the PEREQ signal to request the Intel386 DX microprocessor to initiate operand transfers to or from system memory. These operand transfers occur when the Intel387 DX math coprocessor requests them; thus, they are asynchronous to the instruction execution of the Intel386 DX microprocessor.

When the Intel386 DX microprocessor executes an ESC instruction that requires transfers of operands to or from the coprocessor, the Intel386 DX microprocessor automatically sets an internal memory address base register, memory address limit register, and direction flag. The coprocessor can then request operand transfers by driving PEREQ active. These requests occur only when the coprocessor is executing an instruction (when BUSY# is active).

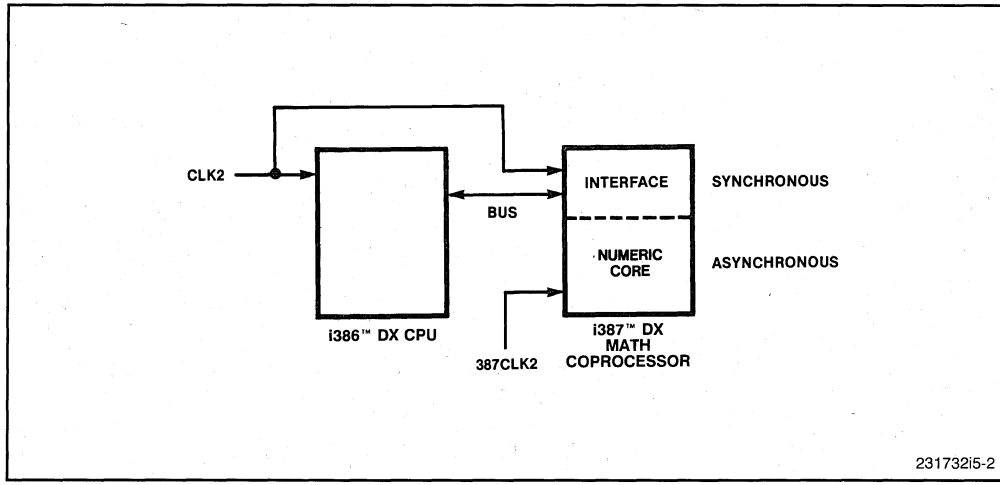


Figure 5-2. Pseudo-Synchronous Interface

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

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

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

5.3 80287/Intel387 DX MATH COPROCESSOR RECOGNITION

In systems that provide a math coprocessor, it is necessary for both hardware and software to correctly determine the presence and identity of the coprocessor.

5.3.1 Hardware Recognition of the NPX

The Intel386 DX microprocessor samples its ERROR# input some time after the falling edge of RESET and before executing the first ESC instruction. The Intel387 DX math coprocessor keeps its ERROR# output in active state after hardware reset. Subsequently if ERROR# was sampled active, the Intel386 DX CPU employs the 32-bit protocol of the Intel387 DX math coprocessor.

5.3.2 Software Recognition of the NPX

Figure 5-3 shows an example of a recognition routine that determines whether a math coprocessor is present, and distinguishes between the Intel387 SX/DX coprocessors and the 8087/80287. This routine can be executed on any Intel386 DX, Intel386 SX, 80286, or 8086 microprocessor hardware configuration that has a math coprocessor socket.

Even though the Intel386 DX microprocessor uses the value of ERROR# after RESET to select microcode which conforms to the Intel387 DX 32-bit protocol, the software designer should not use Intel reserved bits to determine the presence or identity of coprocessors. To assure compatibility with future processors a software recognition test is necessary.

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

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


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



LOC  OBJ                  LINE    SOURCE

                             1 +1 $title('Test for presence of a Numerics Chip, Revision 1.0')
                             2
                             3            name    Test_NPX
                             4
----                         5    stack   segment stack 'stack'
0000 (100                    6            dw      100 dup (?)
     ????
     )
00C8 ????                    7    sst     dw              ?
----                         8    stack   ends
                             9
----                        10    data    segment public 'data'
0000 0000                   11    temp    dw      0h
----                        12    data    ends
                            13
                            14    dgroup  group   data, stack
                            15    cgroup  group   code
                            16
----                        17    code    segment public 'code'
                            18            assume cs:cgroup, ds:dgroup
                            19
0000                        20    start:
                            21    ;
                            22    ;       Look for an 8087, 80287, i387 SX or i387 DX NPX.
                            23    ;       Note that we cannot execute WAIT on 8086/88 if no 8087 is present.
                            24    ;
0000                        25    test_npx:
0000 9BDBE3                 26            fninit                  ; Must use non-wait form
0003 BE0000          R      27            mov     si,offset dgroup:temp
0006 C7045A5A               28            mov     word ptr [si],5A5AH ; Initialize temp to non-zero value
000A 9BDD3C                 29            fnstsw  [si]            ; Must use non-wait form of fstsw
                            30                                    ; It is not necessary to use a WAIT instruction
                            31                                    ; after fnstsw or fnstcw. Do not use one here.
000D 803C00                 32            cmp     byte ptr [si],0 ; See if correct status with zeroes was read
0010 752A                   33            jne     no_npx          ; Jump if not a valid status word, meaning no NPX
                            34    ;
                            35    ;       Now see if ones can be correctly written from the control word.
                            36    ;
0012 9BD93C                 37            fnstcw  [si]            ; Look at the control word; do not use WAIT form
                            38                                    ; Do not use a WAIT instruction here!
0015 8B04                   39            mov     ax,[si]         ; See if ones can be written by NPX
0017 253F10                 40            and     ax, 103fh       ; See if selected parts of control word look OK
001A 3D3F00                 41            cmp     ax,3fh          ; Check that ones and zeroes were correctly read
001D 751D                   42            jne     no_npx          ; Jump if no NPX is installed
                            43    ;
                            44    ;       Somne numerics chip is installed. NPX instructions and WAIT are now safe.
                            45    ;       See if the NPX is an 8087, 80287, i387 SX or i387 DX NPX
                            46    ;       This code is necessary if a denormal exception handler is used or the
                            47    ;       new i387 DX NPX instructions will be used.
                            48    ;
```

Figure 5-3. Software Routine to Recognize the Coprocessor

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


LOC  OBJ                  LINE    SOURCE

001F 9BD9E8                 49            fld1                    ; Must use default control word from FNINIT
0022 9BD9EE                 50            fldz                    ; Form infinity
0025 9BDEF9                 51            fdiv                    ; 8087/287 says + inf = -inf
0028 9BD9C0                 52            fld     st              ; Form negative infinity
002B 9BD9E0                 53            fchs                    ; i387 SX/DX NPX says +inf <> -inf
002E 9BDED9                 54            fcompp                  ; See if they are the same and remove them
0031 9BDD3C                 55            fstsw   [si]            ; Look at status from FCOMPP
0034 8B04                   56            mov     ax,[si]
0036 9E                     57            sahf                    ; See if the infinities matched
0037 7406                   58            je      found_87_287    ; Jump if 8087/287 is present
                            59    ;
                            60    ;       An i387 SX/DX NPX is present.  If denormal exceptions are used for an 8087/287,
                            61    ;       they must be masked.  The i387 SX/DX NPX will automatically normalize denormal
                            62    ;       operands faster than an exception handler can.
                            63    ;
0039 EB0790                 64            jmp     found_387 SX_DX
003C                        65    no_npx:
                            66    ;       set up for no NPX
                            67    ;               ---
                            68    ;
003C EB0490                 69            jmp exit
003F                        70    found_87_287:
                            71    ;       set up for 87/287
                            72    ;               ---
                            73    ;
003F EB0190                 74            jmp exit
0042                        75    found_387:
                            76    ;       set up for 387 SX_DX
                            77    ;               ---
                            78    ;
0042                        79    exit:
----                        80    code    ends
                            81            end     start,ds:dgroup,ss:dgroup:sst

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

Figure 5-3. Software Routine to Recognize the Coprocessor (Contd.)

Memory Interfacing 6

CHAPTER 6
MEMORY INTERFACING

The Intel386 DX microprocessor high-speed bus interface has many features that contribute to high-performance memory interfaces. This chapter outlines approaches to designing memory systems that utilize these features, describes memory design considerations, and lists a number of useful examples. The concepts illustrated by these examples apply to a wide variety of memory system implementations.

6.1 MEMORY SPEED VERSUS PERFORMANCE AND COST

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

To realize the performance potential of the Intel386 DX microprocessor, a system must use relatively fast memory. A high-performance processor coupled with low-performance memory provides no better throughput than a less expensive low-performance processor. Fast memory devices, however, cost more than slow memory devices.

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

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

6.2 BASIC MEMORY INTERFACE

The high performance and flexibility of the Intel386 DX microprocessor local bus interface plus the availability of programmable and semi-custom logic (programmable logic arrays, for example) make it practical to design custom bus control logic that meets the requirements of a particular system. Standard logic components can generate the bus control signals needed to interface the Intel386 DX microprocessor with memory and I/O devices. The basic memory interface is discussed in this chapter; the basic I/O interface is presented in Chapter 8.

The block diagram of the basic memory interface is shown in Figure 6-1. The bus control logic provides the control signals for the address latches, data buffers, and memory devices; it also returns READY# active to end the Intel386 DX microprocessor bus cycle and NA# to control address pipelining. The address decoder generates chip-select signals and the BS16# signal based on the address outputs of the Intel386 DX microprocessor. This interface is suitable for accessing ROMs, EPROMs, and static RAMs (SRAMs).

6.2.1 TTL Devices

TTL devices are specified by number (function), but not by family (speed). Virtually any family of a device can be used if it meets the performance requirements of the application. For example, a 74x00 device might be implemented with a 74F00 or 74AS00.

Table 6-1 lists the most common families of TTL devices, and some of their relative performance specifications. Generally, the F and AS families provide the highest performance.

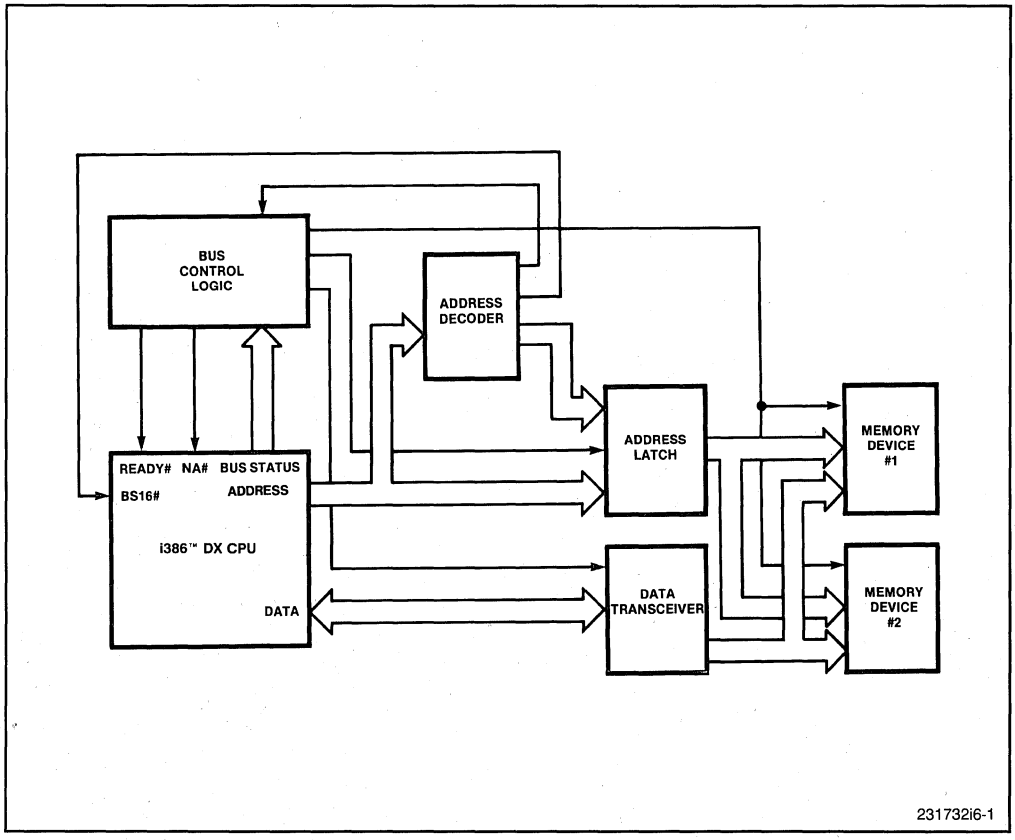


Figure 6-1. Basic Memory Interface Block Diagram

Table 6-1. Common Logic Families*

74xxx	The original TTL family. Now obsolete.
74Lxxx	Low-power version of standard TTL. Very slow and now obsolete.
74LSxxx	Low-power Schottky TTL. Lower power and higher speed than standard TTL. Widely used in microprocessor systems.
74Sxxx	Schottky TTL. High speed and high power consumption. Now obsoleted by newer families.
74ALSxxx	Advanced low-power Schottky TTL. An improved version of LS TTL, providing faster speed, lower power, and better-defined specifications.
74ASxxx	Advanced Schottky TTL. A replacement for Schottky TTL, with higher speeds and lower power.
74Fxxx	Fairchild Advanced Schottky TTL. A competitor of AS.
There are also several families of CMOS logic that are pin-compatible with the TTL families and use the same type number designations. CMOS logic has the advantage of very low power consumption. Unlike TTL, however, the power consumption of CMOS logic increases linearly with switching frequency.	
There are several major families of CMOS logic, as follows:	
74Cxxx	CMOS equivalents of standard TTL devices, but considerably slower. Obsoleted by newer families.
74HCxxx	High-speed CMOS logic. Speeds comparable to LS TTL.
74HCTxxx	High-speed CMOS logic with TTL-compatible input thresholds. Widely used as a low-power LS TTL replacement.
74ACxxx	Advanced CMOS logic, with higher speeds than HC. This nomenclature is used for several families, including Fairchild's FACT family and the ACL family from TI, Signetics, and Phillips.
74ACTxxx	Same as 74AC, with TTL-compatible thresholds

**Microprocessor Based Design.* Michael Slater; © 1987PLD Devices

6.2.2 PLD Devices

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

1. PLD pinout is determined by the designer, which can simplify board layout by moving signals as required.

2. PLDs are inexpensive as compared to dedicated bus controllers.

EPLDs have the following additional advantages:

1. Programmability/erasability allows EPLD functions to be changed easily, simplifying prototype development.

2. Since EPLDs are implemented in CMOS technology, they can consume an order of magnitude less power than bipolar PLDs. Power-conscious applications can benefit greatly from using EPLDs.

3. Since the EPROM cell size is an order of magnitude smaller than an equivalent bipolar fuse, EPLDs can implement more functions in the same package. This higher integration can result in a lower overall component count for a design. The added flexibility can also mean that an extremely low number of "raw" (unprogrammed) devices need to be stocked versus bipolar PLDs.

4. Once an EPLD design has been tested, plastic OTP (One-Time Programmable) versions of the device can be used in a production environment.

PLDs and EPLDs have the following tradeoffs:

1. Most PLDs do not have buried (not connected to outputs) registers. For some state machine applications, this means using an otherwise available output pin to store the current state.

2. The drive capability of CMOS EPLDs may be insufficient for some applications. While the trend is towards use of CMOS throughout a system, in cases where high current levels are required, some additional buffering is required with EPLDs.

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

The sum-of-products is then routed to a configurable macrocell. The macrocell in Figure 6-3 can be configured as a combinational output or registered output. The output can be active high or active low. A separate AND term controls the output buffer.

Designing with PLDs consists of determining where Xs must be placed in the AND array and how to configure the macrocell. This task is simplified by logic compilers, such as iPLS II (Intel's Programmable Logic Software II) or ABEL. Logic compilers accept input in the form of sum-of-product equations and translate the input into a JEDEC programming file that can be used by programming hardware/software.

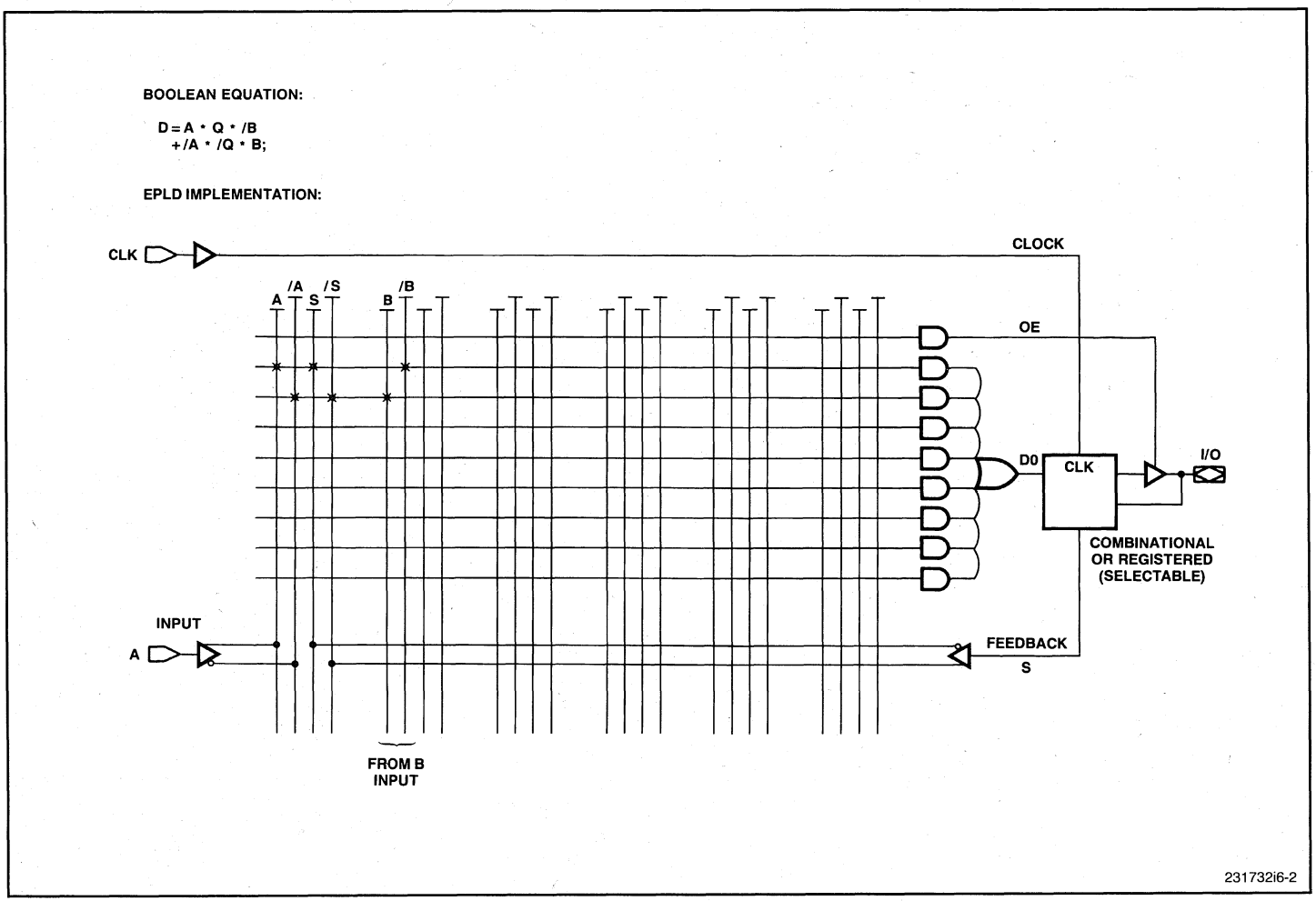


Figure 6-2. PLD Equation and Device Implementation

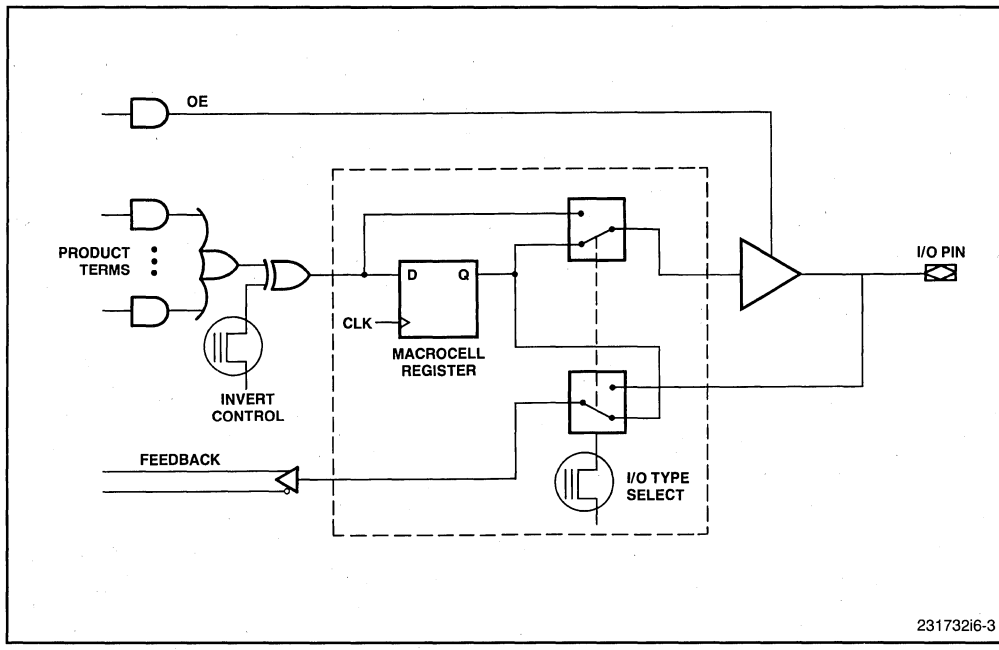


Figure 6-3. 85C220 EPLD Macrocell Architecture

PLDs and other Programmable Logic Devices are specified by part number. Different manufacturers use different numbering schemes. Intel PLDs are described in the *Programmable Logic Handbook*. One EPLD in particular is shown in this chapter, the 85C220. The 85C220 is a 20-pin upgrade to many common bipolar PLDs and is shown in this chapter implementing state machine functions.

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

6.2.3 Address Latch

Latches maintain the address for the duration of the bus cycle and are necessary to pipeline addresses because the address for the next bus cycle appears on the address lines before the current cycle ends. In this example, 74x373 latches are used. Although the Intel386 DX microprocessor can be run without address pipelining to eliminate the need for address latching, the system will usually run less efficiently.

6.2.4 Address Decoder

Address decoders, which convert the Intel386 DX microprocessor address into chip-select signals, can be located before or after the address latches. If it is placed before the latches, the chip-select signal becomes valid as early as possible but must be latched

along with the address. Therefore, the number of address latches needed is determined by the location of the address decoder as well as the number of address bits and chip-select signals required by the interface. Chip-select signals can be routed to the bus control logic to set the correct number of wait states for the accessed device.

The decoder consists of two one-of-four decoders, one for memory address decoding and one for I/O address decoding. In general, the number of decoders needed depends on the memory mapping complexity. The 85C508 EPLD performs both address decoding and latching functions in a single device. In this basic example, the A31 output is sufficient to determine which memory device is to be selected.

6.2.5 Data Transceiver

Standard 8-bit transceivers (74x245, in this example) provide isolation and additional drive capability for the Intel386 DX microprocessor data bus. Transceivers are necessary to prevent the contention on the data bus that occurs if some devices are slow to remove read data from the data bus after a read cycle. If a write cycle follows a read cycle, the Intel386 DX microprocessor may drive the data bus before a slow device has removed its outputs from the bus, potentially causing reliability problems. Transceivers can be omitted only if the data float time of the device is short enough and the load on the Intel386 DX microprocessor data pins meets device specifications.

A bus interface must include enough transceivers to accommodate the device with the most inputs and outputs on the data bus. Normally, 32-bit-wide memories, which require four 8-bit transceivers, are used in Intel386 DX microprocessor systems.

The 74x245 transceiver is controlled through two input signals:

- Data Transmit/Receive (DT/R#)—When high, this input enables the transceiver for a write cycle. When low, it enables the transceiver for a read cycle. This signal is just a latched version of the Intel386 DX microprocessor W/R# output.
- Data Enable (DEN#)—When low, this input enables the transceiver outputs. This signal is generated by the bus control logic.

6.2.6 Bus Control Logic

Bus control logic is shown in Figure 6-4. The bus controller is implemented in two PLDs. One PLD (IOPLD1) follows the Intel386 DX microprocessor bus cycles and generates the overall bus cycle timing. The second PLD (IOPLD2) generates most of the bus control signals. The equations for these PLDs are listed in Appendix A of this manual.

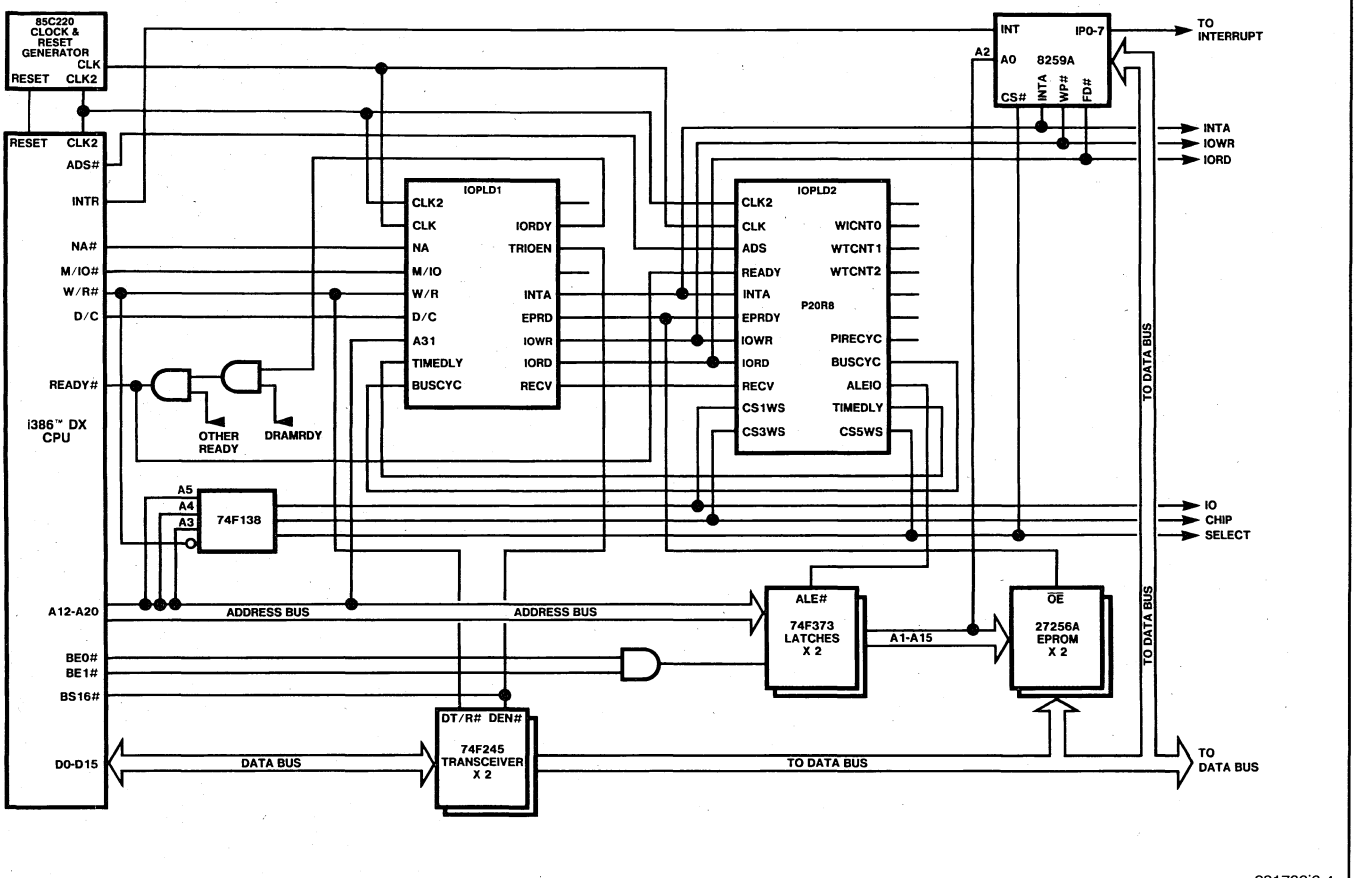


Figure 6-4. I/O Controller Schematic

The bus controller decodes the Intel386 DX microprocessor status outputs (W/R#, M/IO#, and D/C#) and activates a command signal for the type of bus cycle requested. The command signal corresponds to the bus cycle types (described in Chapter 3) as follows:

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

The bus controller also controls the READY# input to the Intel386 DX microprocessor that ends each bus cycle. The bus control PLD counts wait states and returns TIMEDLY# after the number of wait states required by the accessed device. The design of this portion of the bus controller depends on the requirements of the system; relatively simple systems need less wait-state logic than more complex systems. The basic interface described here uses a PLD device to generate TIMEDLY#; other designs may use counters and/or shift registers.

6.2.7 EPROM Interface

Figure 6-5 shows the signal timing for bus cycles from an Intel386 DX microprocessor operating at 20 MHz to a 27256 EPROM, which has a 250-nanosecond access time.

In the EPROM interface, the OE# input of each EPROM devices is connected directly to the EPRD# signal from the bus controller. The wait state requirement is calculated by adding up worst-case delays and comparing the total with the Intel386 DX microprocessor bus cycle time.

The bus cycle timings can be calculated from the waveforms in Figure 6-5. In the following example, the timings for I/O accesses are calculated for CLK2 = 40 MHz, clock circuitry and IOPLD1 implemented using an 85C220-66 (12 ns) PLD and IOPLD2 implemented in a 20R8 PLD. All times are in nanoseconds. Check the most recent *386™ DX Microprocessor Data Sheet* to confirm all parameter values.

tAR: Address stable before Read (EPRD# fall)

(2 x CLK2 period) − PLD RegOut Max (ALEIO) − Latch Enable Max + PLD RegOut Min (EPRD#)

(2 x 25) − 12 − 13 + 1.5

= 26.5 nanoseconds

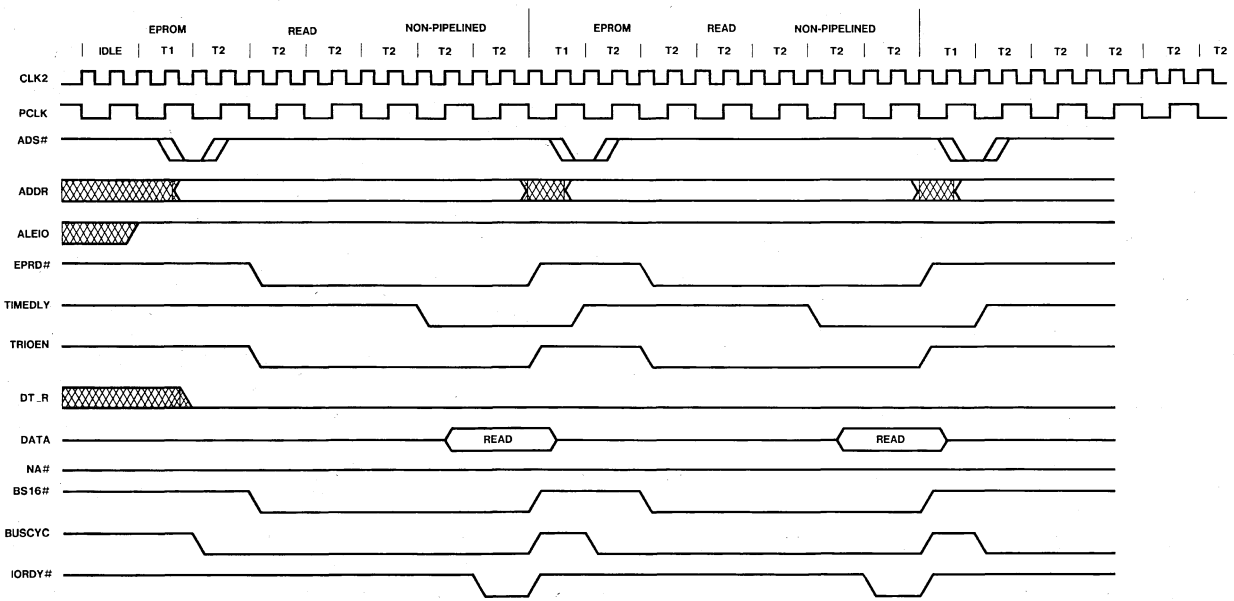


Figure 6-5. 250 Nanosecond EPROM Timing Diagram

tRR: Read (EPRD#) pulse width

(10 x CLK2 period) − PLD RegOut Skew (EPRD# low to high)
(10 x 25) − 4

= 246 nanoseconds

tRA: Address hold after Read (EPRD# rise)

(0 x CLK2 period) − PLD RegOut Max (EPRD#) + PLD RegOut Min (ALEIO)
+ Latch Enable Min
(0 x 25) − 6 + 2
+ 5

= 1 nanoseconds

tAD: Data delay from Address

(12 x CLK2 period) − PLD RegOut Max − Latch Enable Max
− xcvr. prop. Max − Intel386 DX Microprocessor Data Setup Min
(12 x 25) − 12 − 13
− 6 − 11

= 258 nanoseconds

tRD: Data delay from Read (EPRD#)

(10 x CLK2 period) − PLD RegOut Max (EPRD#) − xcvr. prop Max
− Intel386 DX Microprocessor Data Setup Min
(10 x 25) − 6 − 6
− 11

= 227 nanoseconds

6.2.8 16-Bit Interface

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

For example, if BS16# is asserted for EPROM accesses, only two byte-wide EPROMs are needed. Overall performance is reduced because 32-bit data accesses and all code prefetches from the EPROMs are slower (requiring two bus cycles instead of one). However, this reduction is acceptable in certain applications. A system that uses

EPROMs only for power-on initialization and runs programs entirely from SRAM or DRAM has only a power-on time increase over the 32-bit EPROM system; its main programs run at the same speed as the 32-bit system.

The Intel386 DX microprocessor BS16# input directs the Intel386 DX microprocessor to perform data transfers on only the lower 16 bits of the data bus. In systems in which 16-bit memories are used, the address decoder logic must generate the BS16# signal for 16-bit accesses. Since NA# cannot be asserted during a bus cycle in which BS16# is asserted (because the current address may be needed for additional cycles), the decoder logic should also guarantee that the NA# signal is not generated. When the Intel386 DX microprocessor samples BS16# active and NA# inactive, it automatically performs any extra bus cycles necessary to complete a transfer on a 16-bit bus. The Intel386 DX microprocessor response is determined by the size and alignment of the data to be transferred, as described in Chapter 3.

6.3 DYNAMIC RAM (DRAM) INTERFACE

This section presents a dynamic RAM (DRAM) memory subsystem design that is both cost-effective and fast. The design can be adapted for a wide variety of speed and system requirements to provide high throughput at minimum cost. The DRAM design in this section illustrates DRAM subsystem design concepts and analysis. This system would be suitable for use as the main memory of an 82385 cache system described in Chapter 7. Because the 82385 cache controller provides the majority of memory requests in zero wait states, the performance of the main memory is less critical.

6.3.1 Interleaved Memory

DRAMs provide relatively fast access times at a low cost per bit; therefore, large memory systems can be created at low cost. However, DRAMs have the disadvantage that they require a brief idle time between accesses to precharge; if this idle time is not provided, the data in the DRAM can be lost. If back-to-back accesses to the same bank of DRAM chips are performed, the second access must be delayed by the precharge time. To determine if additional idle states will be needed, compare the DRAM cycle time to the cycle length of the Intel386 DX microprocessor. To avoid this delay, memory should be arranged so that each subsequent memory access is most likely to be directed to a different bank. In this configuration, wait time between accesses is not required because while one bank of DRAMs performs the current access, another bank precharges and will be ready to perform the next access immediately.

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

Even if random accesses are made, two DRAM banks allow 50 percent of back-to-back accesses to be made without waiting for the DRAMs to precharge. The precharge time is also avoided when the Intel386 DX microprocessor has no bus accesses to be performed. During these idle bus cycles, the most recently accessed DRAM bank can precharge so that the next memory access to either bank can begin immediately.

The DRAM memory system design described here uses two interleaved banks of DRAMs. The DRAM controller keeps track of the precharge time for each bank while allowing memory accesses to begin as soon as possible.

6.3.2 DRAM Memory Performance

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

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

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

Table 6-2. DRAM Memory Performance

Intel386™ DX Microprocessor Clock Rate	DRAM Access Time (Nanoseconds)	Bus Cycle Wait-States	
		Interleaved Piped:Unpiped	Same Bank
16 MHz	80	0* :1 *	1 *
16 MHz	100	0* :1 *	1 *
16 MHz	150	1 :2	3
20 MHz	70	0* :1 *	2
20 MHz	100	1 :2	3
25 MHz	60	0* :1 *	2 *
25 MHz	80	1 :2	3
33 MHz	60	1+*:2	4 *
33 MHz	80	2 :3	5

*Add one additional wait-state to these times for write accesses.
+ Effective 0 wait states can be achieved by implementing a cache.

NOTE: The numbers for the 16 MHz 100-nanosecond DRAM are based on the assumption that no data transceivers are used.

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

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

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

6.3.3 DRAM Controller

The design in this chapter is a 3-CLK pipelined DRAM controller. The timing analysis is done at 20 MHz. The design can be scaled to match the speed of your design. Other variations for DRAM control are discussed following the sample system.

6.3.3.1 3-CLK DRAM CONTROLLER

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

The DRAM chips used to create the DRAM banks can be of any length (N), and they can be one, four or eight bits wide. If Nx1 DRAM chips are used, 64 chips are required for the two banks; if Nx4 DRAM chips are used, 16 chips are required; if Nx8 DRAM chips are used, only 8 chips are required. The banks in Figure 6-6 are made from eight 256x8 DRAM modules, but another type of DRAM can be substituted easily.

Two Row Address Strobe (RAS) signals are generated by the controller, one for each bank. The top bank is activated by RAS0# and contains the DRAM memory locations for which the Intel386 DX microprocessor address bit A2 is low. The bottom bank is activated by RAS1#, which corresponds to Intel386 DX microprocessor addresses for which A2 is high.

Each of the 32 data lines of the Intel386 DX microprocessor are connected to one DRAM data bit from each bank. If Nx1 DRAMs are used, the corresponding data line is connected to both the Din and Dout pins. If Nx4 or Nx8 DRAMs are used, each data line is connected only to the corresponding I/O pin.

Each bank has four Column Address Strobes (CAS#), one for each byte of the Intel386 DX microprocessor data bus. The Intel386 DX microprocessor Byte Enable Signals (BE3#–BE0#) map to the active bank's CAS signals. CAS0# is generated by OR-ing

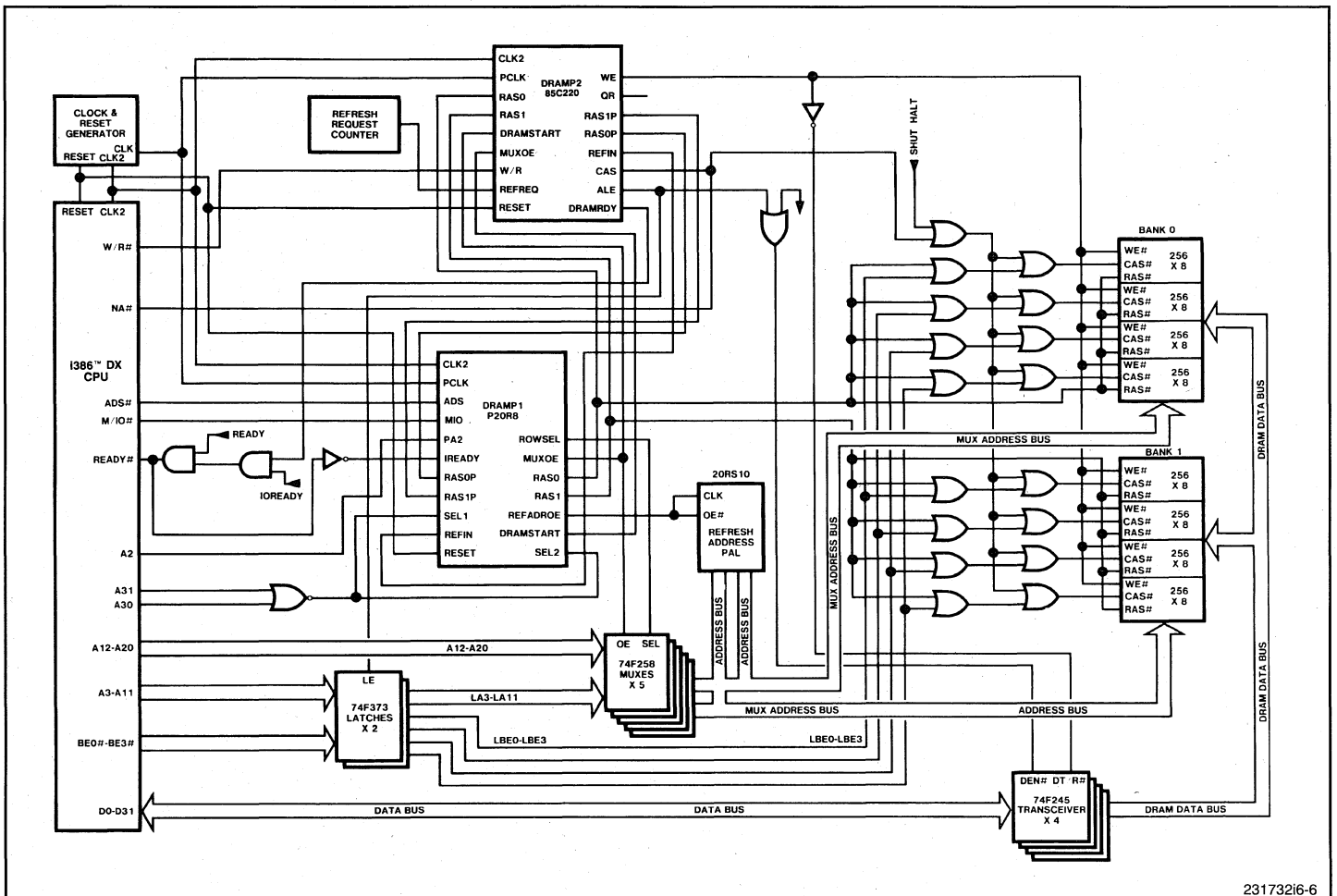


Figure 6-6. 3-CLK DRAM Controller Schematic

the RAS# from that bank and BE0# with the CAS# signal to enable the least-significant byte (D7–D0). Similarly, CAS3# is generated by RAS#, BE3# and CAS# and enables the most significant byte (D31–D24).

The Write Enable (WE#) and the multiplexed address signals are connected to every DRAM module in both banks. For drive considerations the multiplexed address is generated separately for each bank.

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

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

Four 74F245 octal transceivers buffer the DRAM from the data bus. Most DRAMs used in the 3-CLK design require these transceivers to meet the read-data float time. When a DRAM read cycle is followed immediately by a Intel386 DX microprocessor write cycle, the Intel386 DX microprocessor drives its data bus one CLK2 period after the read cycle completes. If the data transceivers are omitted, the CAS inactive delay plus the DRAM output buffer turn-off time (t-OFF) must be less than a CLK2 period to avoid data bus contention.

Two PLDs are used to monitor the Intel386 DX microprocessor status signals and generate the appropriate control signals for the DRAM, multiplexer, and transceivers. PLD codes and pin descriptions for the 3-CLK design are listed in Appendix B of this manual. These PLDs, DRAMP1 and DRAMP2 contain state machines to perform the following functions:

- DRAMP1
  - Performs bus cycle tracking
  - Monitors the Intel386 DX microprocessor DRAM chip select logic
  - Signals start of DRAM cycles to DRAMP2
  - Generates the RAS# signals and the Address Mux select signal (ROWSEL)
  - Controls refresh cycle arbitration and controls the address output enables for refresh cycles
- DRAMP2
  - Receives and stores DRAM refresh requests from the refresh counter
  - Keeps track of DRAM banks requiring precharge time

- Provides the data transceiver and address latch control signals
- Produces the CAS# and WE# DRAM signals
- Generates the READY# signal to end DRAM bus cycles

A DRAM read or write access is requested when all the chip-select signal inputs to DRAMP1 are sampled active simultaneously. These signals become active when all of the following conditions exist at once:

- M/IO#, W/R#, and D/C# outputs of the Intel386 DX microprocessor indicate either a memory read, memory write, or code fetch.
- The bus is idle or the current bus cycle is ending (READY# active).
- ADS# is active.
- A31 is low (in this design, the lower half (two gigabytes) of the Intel386 DX microprocessor memory space is mapped to the DRAM controller).

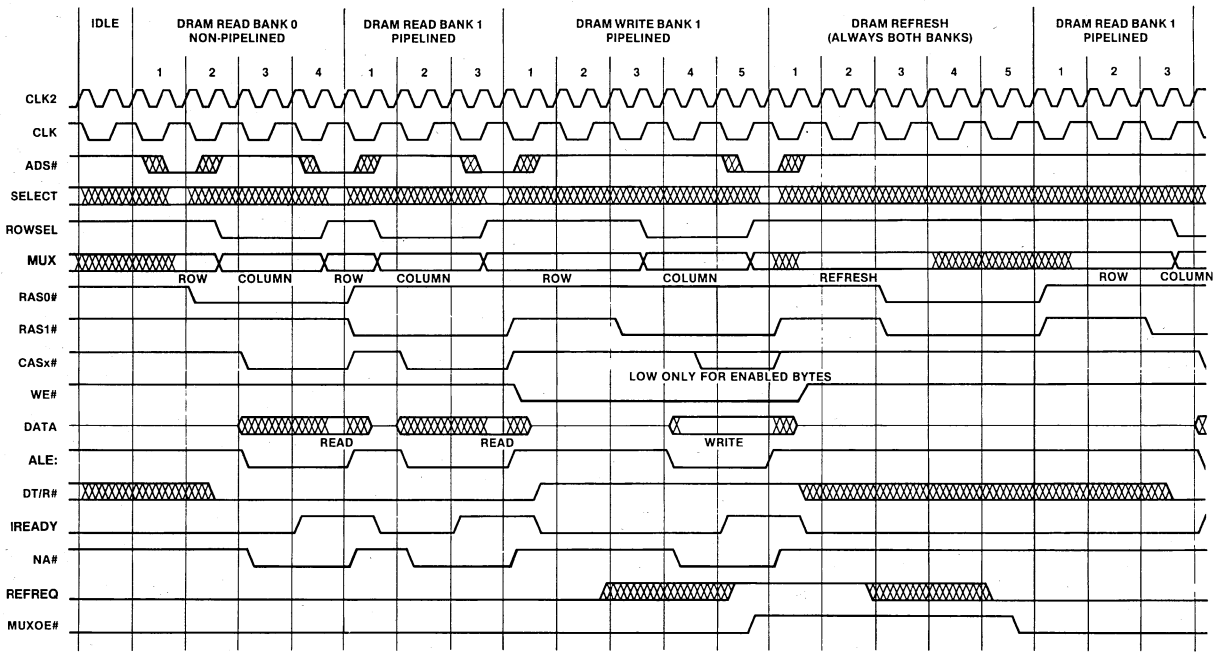
If DRAMP1 is not already performing a cycle, it begins the access immediately. However, if the DRAM controller is performing a refresh cycle, or if it is waiting for the DRAM bank to precharge, the request is latched and performed when the controller is not busy.

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

Figure 6-7 shows the timing of DRAM control signals for the 3-CLK design for the following five sequential DRAM cycles:

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

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



231732i6-7

Figure 6-7. 3-CLK DRAM Controller Cycles

6.3.3.2 DRAM TIMING ANALYSIS

Figure 6-7 shows the signals for bus cycles from a Intel386 DX microprocessor to a DRAM subsystem. This figure will be used for determining the worst case logic timings for a Intel386 DX microprocessor operating at 20 MHz.

In this example, the timing for DRAM accesses are calculated for CLK2 = 40 MHz, DRAMP2 implemented using an 85C220 (12 ns) EPLD, DRAMP1 implemented with a P2OR8 PLD, and Refresh Address implemented with a P2ORS10 PLD. For a registered PLD to change states on each clock edge its maximum clock to output delay plus its minimum setup time must be less than the time between clock edges. This is because the register outputs are fed back and used as input variables in determining the PLD terms.

The 3-CLK DRAM controller in Figure 6-6 performs reads and writes in 3 pipelined clocks (1 wait state). Successive accesses to the same bank will require two clocks of RAS# precharge. Typical DRAM read and write cycles are shown in Figure 6-8 and 6-9.

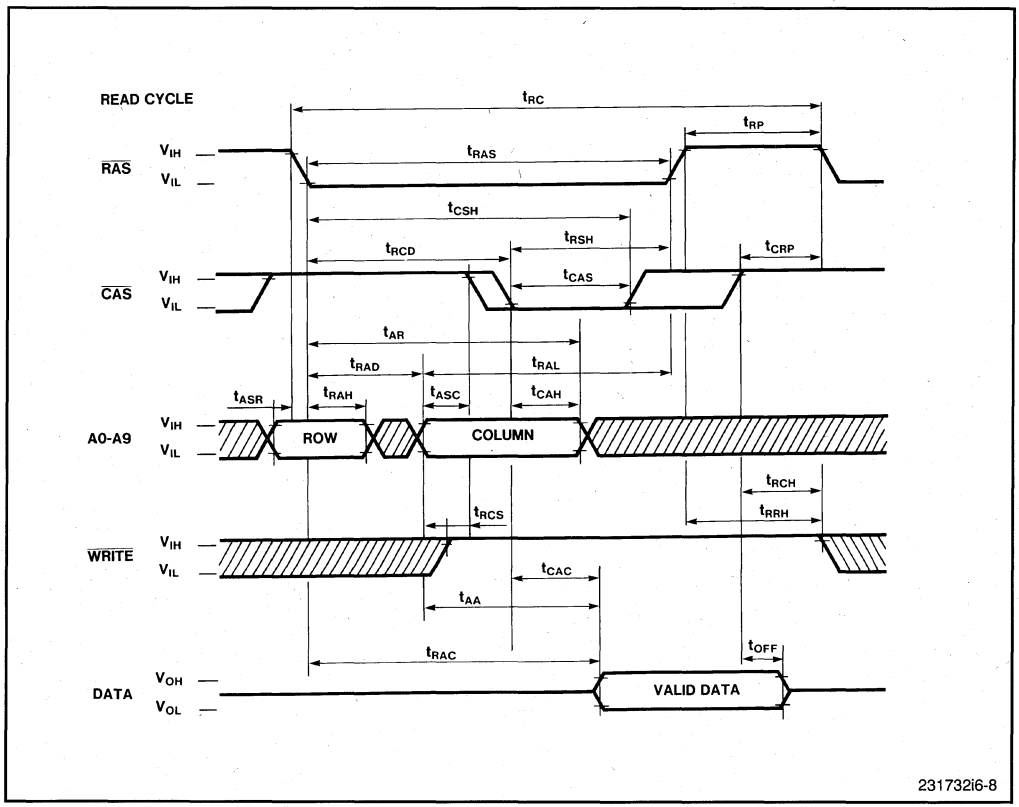


Figure 6-8. Timing Waveforms (Read Cycle)

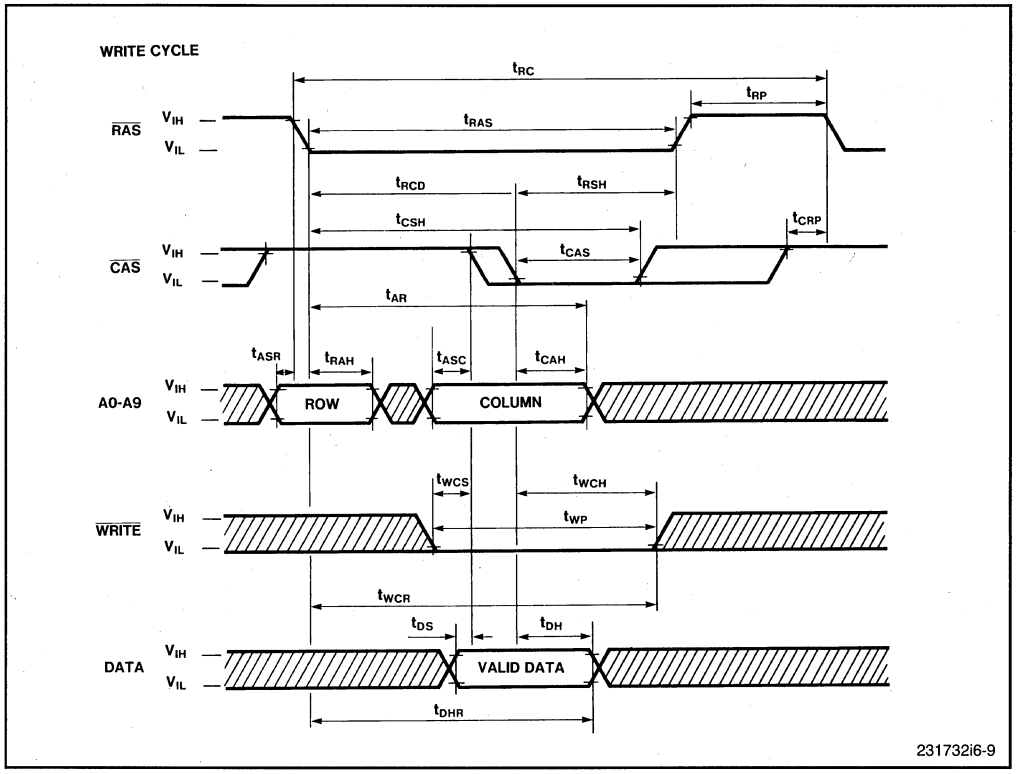


Figure 6-9. Timing Waveforms (Write Cycle)

6.3.3.3 LOGIC DELAY

Logic delay is the time required for an output to change with respect to an input. Devices usually have the following specifications: typical delays, maximum delays and minimum delays. For this chapter we will use worst case logic delays for the commercial operating range.

6.3.3.4 ADDRESS BUS TIMINGS

The first timings to consider are the address set up and hold times for RAS# and CAS#. The address becomes active from the start of phase one and are valid after the address valid delay (T6). The address then passes through the MUX before getting to the DRAM. RAS# is generated on the following phase one CLK2 edge by DRAMP1. The worst case address set up time occurs when the address is delayed the maximum time and RAS# is delayed the minimum amount of time.

T_{ASR} : Row address setup time

= (2 × CLK2 period) − Intel386 DX microprocessor Address Valid Delay Max (T6) − MUX Prop Max (I to Z) + PLD RegOut Min (RAS#)

= 50 − 30 − 6 + 2

= 16 nanoseconds

The worst case address hold time from RAS# occurs when the ROWSEL signal is at a minimum and RAS# delay is at a maximum.

T_{RAH} : Row address hold time

= (1 × CLK2 period) + PLD RegOut Min (ROWSEL) + Mux Prop Min (S to Z) − PLD RegOut Max (RAS#)

= 25 + 2 + 4 − 12

= 19 nanoseconds

Because CAS# is generated a CLK2 cycle later for write cycles, worst case considerations are the column address setup time for read cycles and the column address hold time for write cycles.

T_{ASC} : Column address setup time (read cycles)

= (1 × CLK2) − PLD RegOut Max (ROWSEL) − Mux Prop Max (S to Z) + PLD RegOut Min (CAS#) + (2 × Or-gate Prop Min)

= 25 − 12 − 11 + 1.5 + 6

= 9.5 nanoseconds

T_{CAH} : Column address hold time (write cycles)

= (2 × CLK2) − PLD RegOut Max (CAS#) − (2 × Or-gate Prop Max) + PLD RegOut Min (ROWSEL) + Mux Prop Min (S to Z)

= 50 − 6 − 12 + 2 + 4

= 38 nanoseconds

The write enable (WE#) signal is generated two CLK2 before CAS# on write cycle and one CLK2 before CAS# on a read cycle.

T_{RCS} : Read command setup time

= (1 × CLK2) − PLD RegOut Max (WE#) + PLD RegOut Min (CAS#) + (2 × Or-gate Prop Min)

= 25 − 6 + 1.5 + 6

= 26.5 nanoseconds

6.3.3.5 DATA BUS TIMINGS

The next timings to consider are the data path delays. These calculations include data buffers.

T_{RAC} : Read data access from RAS

= (6 × CLK2) − PLD RegOut Max (RAS#) − Intel386 DX microprocessor Data Setup Min (T21) − Xcvr Prop Max

= 150 − 12 − 11 − 7

= 120 nanoseconds

T_{CAS} : Read data access from CAS#

= (4 × CLK2) − PLD RegOut Max (CAS#) − (2 × Or-gate Prop Max) − Intel386 DX microprocessor Data Setup Min − Xcvr Prop Max

= 100 − 6 − 12 − 11 − 7

= 64 nanoseconds

T_{OFF} : Min output data hold time from CAS#. This is done to assure that the Intel386 DX microprocessor Data Hold Time (T22) will be met

For the inactive CAS# edge:

= PLD RegOut Min (CAS#) + (2 × Or-gate Prop Min) + Xcvr Prop Min − Intel386 DX microprocessor Read Data hold time (T22)

= 1.5 + 6 + 2.5 − 6

= 4.0 nanoseconds Therefore T_{OFF} can be 0 and the read data hold time will still be met

For the inactive DEN# edge:

= PLD RegOut Min (ALE#) + Or-gate Prop Min + Xcvr Disable Min

= 1.5 + 3 + 2

= 6.5 nanoseconds Which meets the Intel386 DX Microprocessor read data hold time (T22) of 6 nanoseconds at 20 MHz.

For write cycles the maximum Intel386 DX microprocessor write data valid delay is 38 nanoseconds measured from the start of clock phase two. CAS# is delayed two CLK2 periods till the start of the next clock phase two to assure the data will be valid.

T_{DS} : Write data setup to CAS#

= (2 × CLK2 period) − Intel386 DX microprocessor Write Data Valid Delay (T12) − Xcvr Prop Max + PLD RegOut Min (CAS#) + (2 × Or-gate Prop Min)

= 50 − 38 − 7 + 1.5 + 6

= 12.5 nanoseconds

T_{DH} : Data hold time from CAS# active

= (3 × CLK2 period) − PLD RegOut Max (CAS#) − (2 × Or-gate Prop Max) + (PLD RegOut Min (ALE) + OR-gate Prop Min) + Xcvr Disable Min

= 75 − 6 − 12 + (1.5 +3) + 2

= 63.5 nanoseconds

6.3.3.6 AVOIDING DATA BUS CONTENTION

Using data transceivers allows write cycles to follow read cycles without additional wait states. Care must be taken to disable the transceivers before changing their direction. Figure 6-10 shows the timing for a read cycle directly followed by a write cycle.

On the Intel386 DX microprocessor data bus side of the transceivers the read data stops driving the bus when DEN# goes inactive and the data buffer disables its output.

T_{XOP} : Transceiver stops driving the processor side of the data bus

= PLD RegOut Max (ALE#) + Or-gate Prop Max + Xcvr Disable Max

= 6 + 6 + 7.5

= 19.5 nanoseconds or 5.5 nanoseconds before the start of phase two

The Intel386 DX microprocessor does not start driving write data until a minimum of 4 ns (T12) after the start of phase two so no contention will occur.

On the DRAM side of the transceivers the read cycle ends when CAS# goes inactive and the DRAMs turn off. The write cycle data will be driven onto the bus at the start of phase one of the next clock after DEN# becomes active.

T_{OFF} : DRAM data turn off from CAS#

= (2 × CLK2 period) − PLD Reg Out Max (CAS#) − (2 × Or-gate Prop Max) + (PLD RegOut Min (ALE) + Or-gate Prop Min) + Xcvr Enable Min

= 50 − 6 − 12 + (1.5+3) + 3

= 39.5 nanoseconds

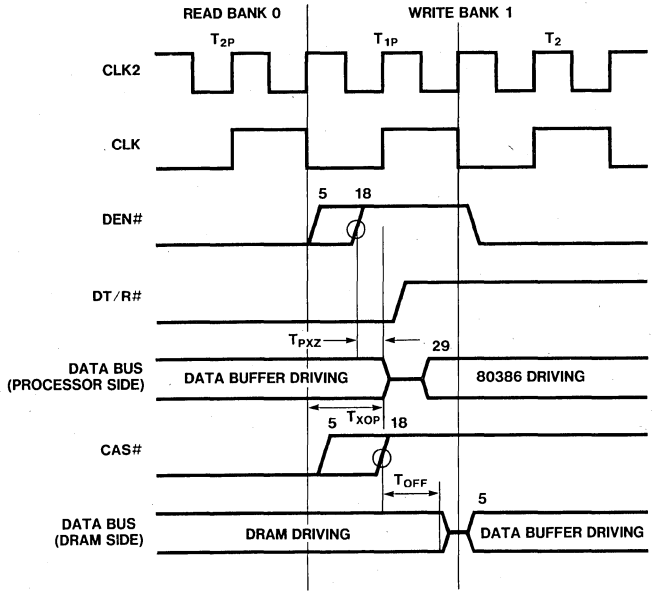


Figure 6-10. Avoiding Data Bus Contention

The DRAMs must be able to turn off the output drivers following a read cycle in 39.5 nanoseconds to avoid the bus contention with the data being, written on the next cycle.

The direction of the data transceivers must be changed while DEN# is inactive and the outputs have been disabled.

Figure 6-6 has separate CAS# lines for each bank. This is to allow a, (T_{CRP}). CAS# to RAS# precharge time for alternate bank accesses.

6.3.3.7 CONTROL SIGNAL TIMINGS

In addition to the DRAM memory access signals, the DRAM controller must generate the NA# and READY# inputs for the Intel386 DX microprocessor.

Once a bus cycle is in progress and the current address has been valid for at least one entire bus state, the NA# input is sampled at the end of every phase one until the bus cycle is acknowledged. Once NA# is sampled asserted the address and status bits for the

current bus cycle can no longer be assumed valid. The 3-clock DRAM controller does not assert NA# in the first T2 but in the second T2 making it a T2p. NA# is asserted at the beginning of phase one.

T_{NA} setup

= (1 × CLK2 period) − PLD RegOut Max (CAS#)

= 25 − 6

= 19 nanoseconds The Intel386 DX microprocessor requires NA# setup time (T15) to be 9 ns.

NA# remains asserted till READY# is returned to the Intel386 DX microprocessor and the cycle ends. Asserting NA# in the next clock cycle is not necessary and only serves to extend the hold time.

When READY# is asserted during a read cycle or an interrupt acknowledge cycle the Intel386 DX microprocessor latches the input data. During write cycles READY# causes the bus cycle to terminate. The Intel386 DX microprocessor READY# setup and hold times are specified in relation to the end of phase two.

$T_{READY\ setup}$:

= (2 × CLK2 period) − (2× And-gate Prop Max) − PLD RegOut Max (DRAM-RDY#)

= 50 − 14 − 8

= 30 nanoseconds Meets Intel386 DX microprocessor READY# setup time (T19)

$T_{READY\ hold}$:

= PLD RegOut Min (DRAMRDY#) + (2× And-gate Prop Min)

= 1.5 + 6

= 7.5 nanoseconds Meets Intel386 DX microprocessor READY# hold time (T20)

6.3.3.8 LOGIC PATHS

When performing worst case logic delay analysis, it is often necessary to consider the maximum delay of one signal path and the minimum delay of another separate signal path. However, when two or more signals are generated from the same device, or signal paths have common elements in their delay paths, it is more realistic to consider the signal skew than to consider the theoretical maximum skew.

For example consider the minimum RAS# pulse width specification. Instead of:

T_{RAS} = (6 × CLK2) − PLD RegOut Max (RAS# active) + PLD RegOut Min (RAS# inactive)

It would be more realistic to consider

T_{RAS} = (6 × CLK2) − PLD RegOut Skew (RAS# active-inactive)

Where the skew depends on:

- The capacitance
- The opposite going signal edges

The skew would even be less for the same signal between two positive or two negative edges. For example TRC, the RAS# cycle time, is measured from RAS# active to RAS# active. The timing analysis would be the number of clock cycles minus the RAS# active-active skew.

6.3.3.9 CAPACITIVE LOADING

The delay of a logic device is affected by the capacitive load on the output. Most devices are specified at a given load and include either a delay versus load graph or a nanoseconds per picofarad specification. The Intel386 DX microprocessor data sheet includes a delay versus load graph. From this graph a linear approximation of the relay can be made. The data sheet specifies the delay for a particular load. If the actual load is greater than the specified load, an additional delay factor needs to be calculated.

The Intel386 DX microprocessor specifications are made at the 1.5 volt levels. If the component interfaced is specified at another level, it will be necessary to consider the rise times of signals. The Intel386 DX microprocessor data sheet provides a rise time versus capacitance graph.

6.3.4 DRAM Design Variations

6.3.4.1 3-CLK DESIGN VARIATIONS

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

- The 3-CLK designs can use any length DRAM in Nx1, Nx4, and Nx8 widths.
- The 3-CLK design can use the internal PLD registers or external TTL registers on the RAS and/or CAS signals.
- Data transceivers are optional. If a data transceiver is used, the DRAM read access must meet the Intel386 DX microprocessor read-data setup time. If no data transceiver is used, the DRAM read-data-float time must not interfere with the next Intel386 DX microprocessor cycle, particularly if it is a write cycle, and the Intel386 DX microprocessor data pin loading must not be exceeded.

- The choice of chip-select logic in the design is arbitrary. Other DRAM memory-mapping schemes can be implemented by modifying the address decoding to the DRAM State PLD chip-selects.
- It is possible to deassert RAS# before the end of the cycle to improve the RAS# precharge time.
- For a single DRAM bank rather than two, the user should tie the DRAMP1 PLD A2 input low, leave RAS1# unconnected (only RAS0# is used), and feed the Intel386 DX microprocessor address bit A2 into the address multiplexer. The DRAMP1 PLD equations can be modified to change the RAS1# output to duplicate the RAS0# output for more drive capability, and the A2 input can be used as another chip-select input. When only one bank is used, no accesses can be interleaved, and back-to-back accesses run with three wait states with the 3-CLK design (independent of address pipelining).

6.3.4.2 USING TAP DELAY LINES

To further optimize your memory design it may be necessary to use tap delay lines. Tap delay lines allow signals to be generated in relation to other signals instead of from clock edges. In Figure 6-11, after RAS# is asserted for a memory read, the MUX select is changed to select the column address. The 5 nanosecond delay allows for RAS# address hold time while the 20 nanosecond delay of CAS# allows for column address setup time.

Tap delay lines can be used to satisfy other DRAM parameters, such as minimum RAS# pulse width. Tap delay lines may allow a design to use the least number of wait states.

6.3.4.3 REDUCING THE CLOCK FREQUENCY

Many of the memory system timings are related to the clock frequency. If the limiting factor of a memory system design is due to a timing that is dependent on the frequency, slowing the clock frequency should be considered. A small reduction in clock frequency

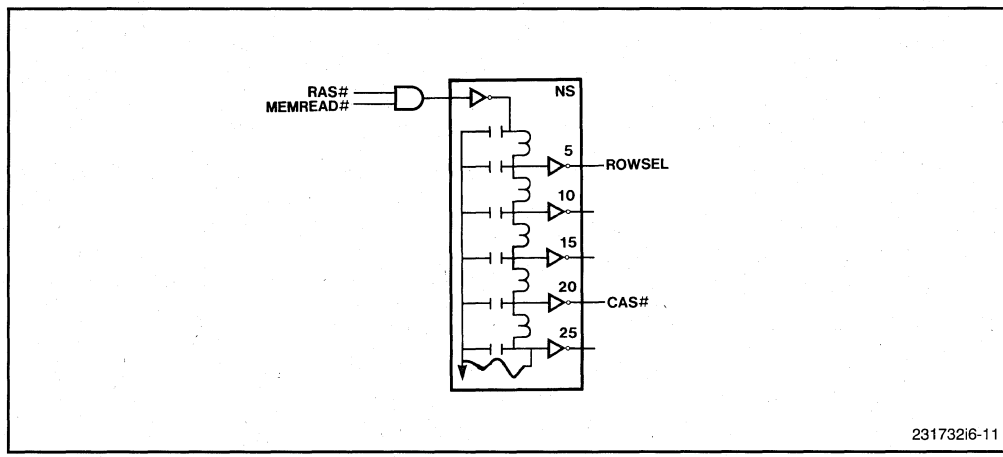


Figure 6-11. Tap Delay Line

may reduce overall system performance less than adding a wait state. Reducing the clock frequency affects the time for both external bus activity and internal computations. The relationship between clock frequency and system performance is approximately linear. Table 4-2 gives relative performance versus wait states and operating frequency.

6.3.5 Refresh Cycles

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

The 3-CLK design uses RAS-only refresh. The address multiplexer is placed in the high impedance state, and the Refresh Address Counter PLD is enabled to output the address of the next row to be refreshed. Then the DRAMP1 PLD activates both RAS0# and RAS1# to refresh the selected row for both banks at once. After the refresh cycle is complete, the Refresh Address Counter PLD increments so that the next refresh cycle refreshes the next sequential row.

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

6.3.5.1 DISTRIBUTED REFRESH

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

The Refresh Interval Counter PLD is programmed to request a single distributed refresh cycle at intervals slightly under 15.625 microseconds. The counter requests a new refresh cycle after a preset number of CLK cycles. This number is dependent on the CLK frequency and can be calculated as follows for a 20-MHz CLK signal:

$$20\ \text{MHz} \times 15.625\ \text{microseconds} - 5/256 = 312.48$$
$$= 312\ \text{CLK cycles}$$

The term 5/256 is subtracted to allow for the time it takes the DRAMP1 PLD to respond to the request. Refresh requests are always given highest priority; however, if a DRAM access is already in progress, it must finish before the refresh cycle can start. The 3-CLK

controller responds within 1-5 CLKs of the refresh request. The maximum latency (the difference between the longest and shortest responses) for the design is therefore 5 CLKs. This time is spread out among all 256 accesses, so 5/256 is subtracted in the above equations to account for the latency period. The counter immediately resets itself after it reaches the maximum count, regardless of this latency period.

Distributed refresh has two advantages over other types of refresh:

- Refresh cycles are spread out, guaranteeing that the Intel386 DX microprocessor access is never delayed very long for refresh cycles. Most programs execute in approximately the same time, regardless of when they are run with respect to DRAM refreshes.
- Distributed refresh hardware is typically simpler than hardware required for other types of refresh.

6.3.5.2 BURST REFRESH

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

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

6.3.5.3 DMA REFRESH USING THE 82380 DRAM REFRESH CONTROLLER

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

DMA refresh can be used for both 3-CLK and 2-CLK designs. To activate both banks, the 82380's Refresh Request (TOUT1/REF#) is ANDed with the Intel386 DX microprocessor's Hold Acknowledge (HLDA) to qualify for a valid refresh operation. The output of this ANDed signal is connected to the RFRQ input of the DRAMP1 PLD (see Figure 6-12). The DRAM State PLD must be modified to ignore chip selects. This modification is needed to prevent the PLD from attempting to run a normal access cycle after the refresh cycle is complete.

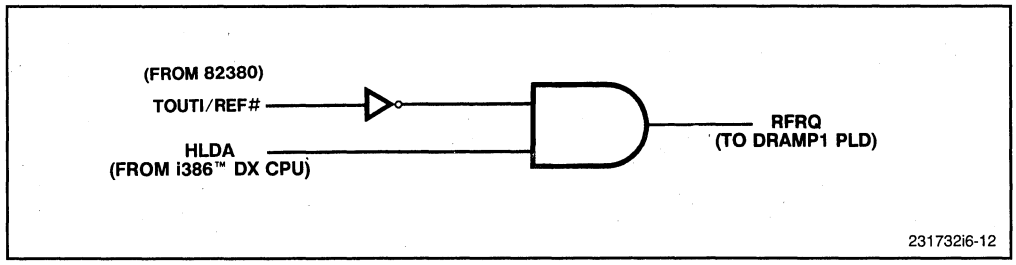


Figure 6-12. Refresh Request Generation

In addition, the DRAMP2 PLD must be modified so that the Ready (RDY) signal is generated on refresh accesses. Finally, the OE# input of the address multiplexer should be tied low so that it never enters the high impedance state, and the row address should include the least significant address bits (A10:3).

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

6.3.6 Initialization

Once the system is initialized, the integrity of the DRAM data and states is maintained, even during a Intel386 DX microprocessor halt or shutdown state or hardware reset, because all DRAM system functions are performed in hardware.

The controller PLDs contain some state and counter information that is not implicitly reset during a power-up or hardware reset. The state machines are designed so that they enter the idle state within 18 CLK2 cycles regardless of whether they powerup in a valid state. The counters can start in any state. Thus, even though the state machines and counters can powerup into any state, they are ready for operation before the Intel386 DX microprocessor begins its first bus access.

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

- Performing several dummy DRAM cycles as part of the Intel386 DX microprocessor initialization process. Setting up the Intel386 DX microprocessor registers and performing a REP LODS instruction is one way to perform these dummy cycles.
- Activating the RFRQ signal, using external logic, for a preset amount of time, causing the DRAM control hardware to run several refresh cycles.

Cache Subsystems 7

CHAPTER 7
CACHE SUBSYSTEMS

Operating at 33 MHz, the Intel386 DX microprocessor can perform a complete bus cycle in only 60 nanoseconds, for a maximum bandwidth of 66 megabytes per second. To sustain this maximum speed, the Intel386 DX microprocessor must be matched with a high-performance memory system. The system must be fast enough to complete bus cycles with no wait states and large enough to allow the Intel386 DX microprocessor to execute large application programs.

Traditional memory systems have been implemented with dynamic RAMs (DRAMs), which provide a large amount of memory for a small amount of board space and money. However, low-cost DRAMs that can complete random read-write cycles in 60 nanoseconds are not commonly available. Faster static RAMs (SRAMs) can meet the bus timing requirement, but they offer a relatively small amount of memory at a higher cost. Large SRAM systems can be prohibitively expensive.

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

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

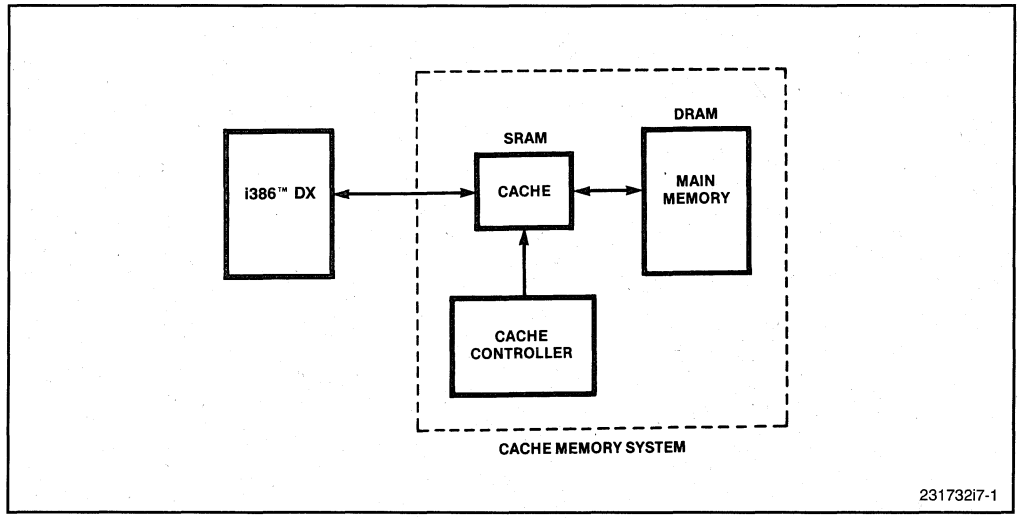


Figure 7-1. Cache Memory System

7.1 INTRODUCTION TO CACHES

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

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

Section 7.7 of this chapter introduces the 82385 High Performance 32-Bit Cache Controller. The 82385 Cache Controller integrates a cache directory and all cache management logic on one chip.

7.1.1 Program Locality

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

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

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

7.1.2 Block Fetch

The block fetch uses program locality to increase the hit rate of a cache. The cache controller partitions the main memory into blocks. Typical block sizes (also known as line size) are 2, 4, 8, or 16 bytes. A 32-bit processor usually uses two or four words per

block. When a needed word is not in the cache, the cache controller moves not only the needed word from the main memory into the cache, but also the entire block that contains the needed word.

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

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

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

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

7.2 CACHE ORGANIZATIONS

7.2.1 Fully Associative Cache

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

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

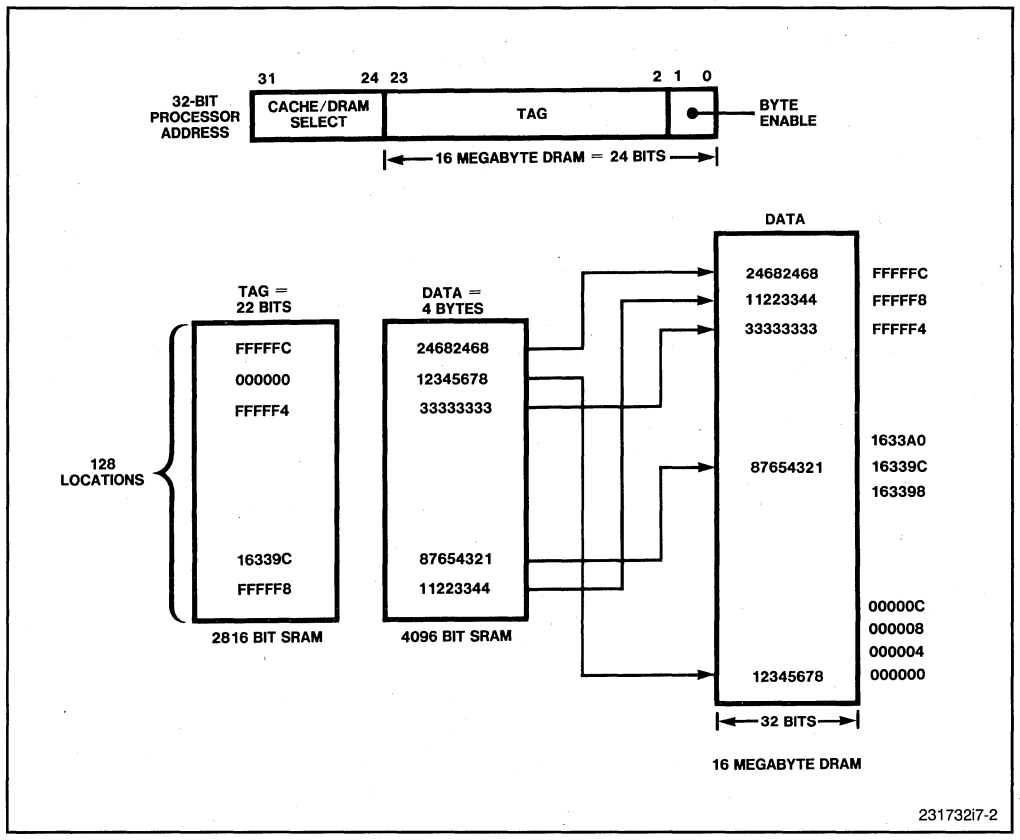


Figure 7-2. Fully Associative Cache Organization

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

7.2.2 Direct Mapped Cache

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

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

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

For example, consider a 64-kilobyte direct mapped cache that contains 16K 32-bit locations and caches 16 megabytes of main memory. The cache index field must include 14 bits to select one of the 16K blocks in the cache, plus 2 bits (or 4 byte Enables) to select a byte from the 4-byte block. The tag field must be 8 bits wide to identify one of the 256 blocks that can occupy the selected cache location. The remaining 8 bits of the 32-bit Intel386 DX microprocessor address are decoded to select the cache subsystem from among other memories in the memory space. The direct-mapped cache organization is shown in Figure 7-3.

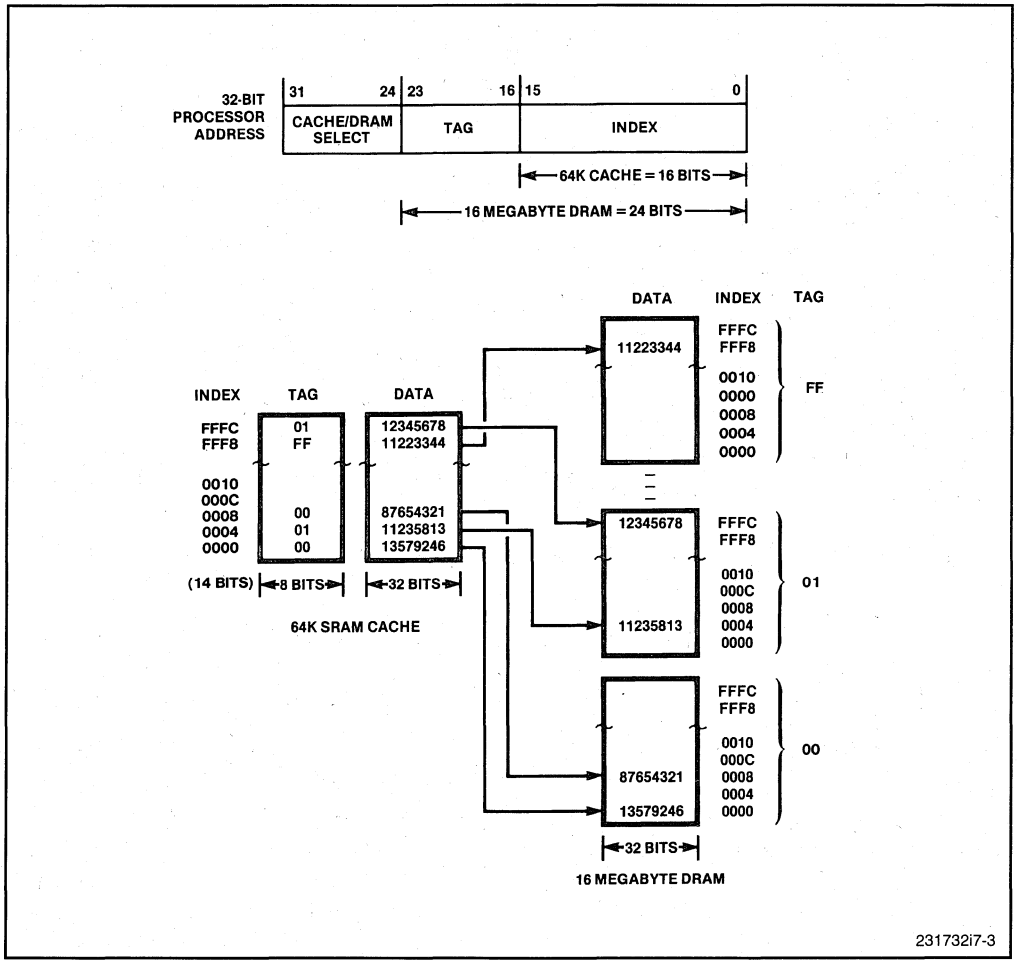


Figure 7-3. Direct Mapped Cache Organization

In a system such as shown in Figure 7-3, a request for the byte of data at the address 12FFE8H in the main memory is handled as follows:

1. The cache controller determines the cache location from the 14 most significant bits of the index field (FFE8H).

2. The controller compares the tag field (12H) with the tag stored at location FFE8H in the cache.

3. If the tag matches, the processor reads the least significant byte from the data in the cache.

4. If the tag does not match, the controller fetches the 4-byte block at address 12FFE8H in the main memory and loads it into location FFE8H of the cache, replacing the current block. The controller must also change the tag stored at location FFE8H to 12H. The processor then reads the least significant byte from the new block.

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

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

7.2.3 Set Associative Cache

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

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

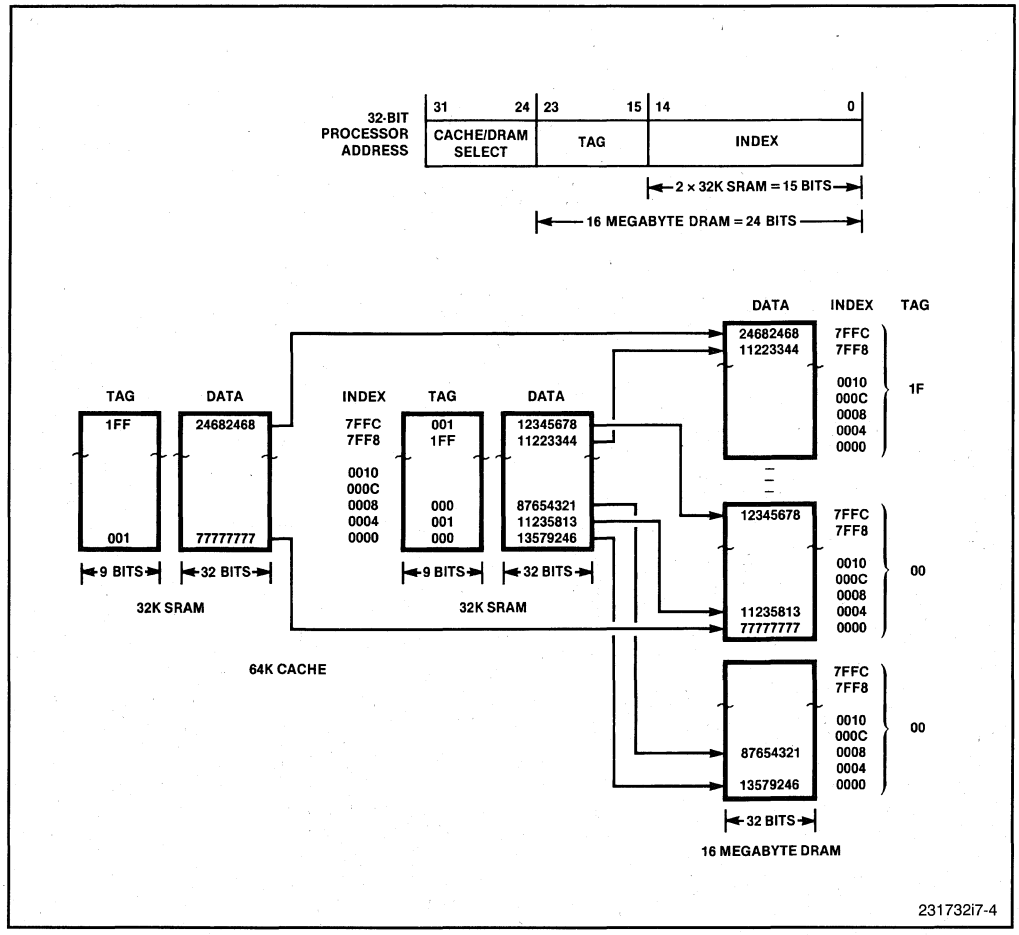


Figure 7-4. Two-Way Set Associative Cache Organization

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

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

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

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

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

7.3 CACHE UPDATING

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

7.3.1 Write-Through System

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

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

7.3.2 Buffered Write-Through System

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

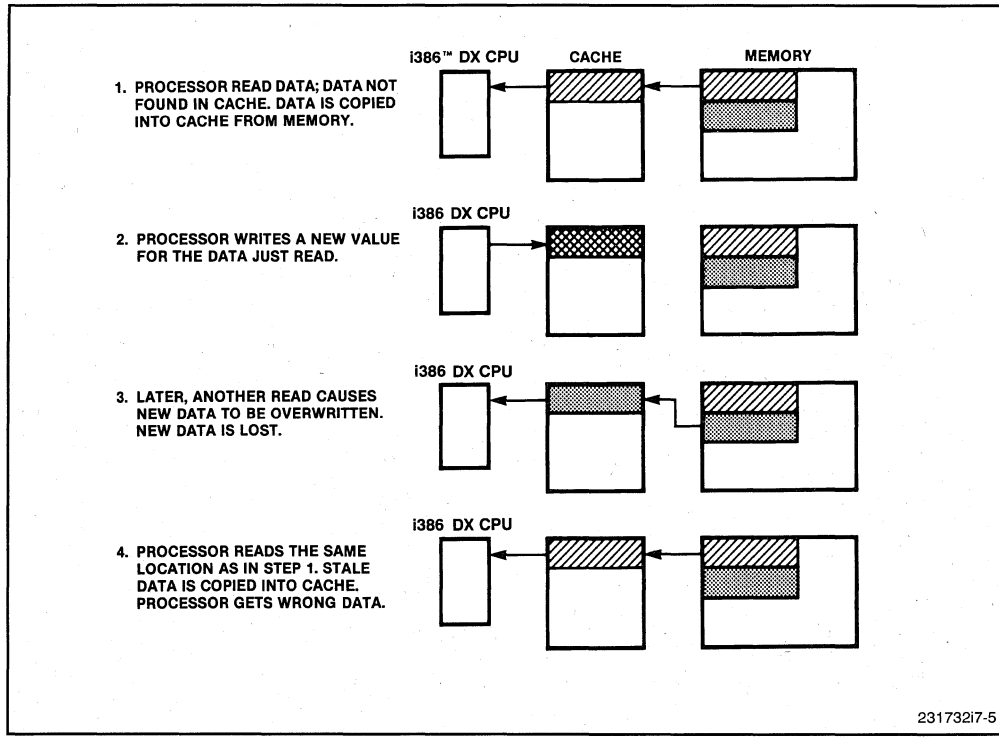


Figure 7-5. Stale Data Problem

7.3.3 Write-Back System

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

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

- Write-back cache controller logic is more complex than write-through. When a write-back system must write an altered block to memory, it must reconstruct the write address from the tag and perform the write-back cycle as well as the requested access.

- All altered blocks must be written to the main memory before another device can access these blocks in main memory.

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

7.3.4 Cache Coherency

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

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

- Hardware transparency—Hardware guarantees cache coherency by ensuring that all accesses to memory mapped by a cache are seen by the cache. This is accomplished either by routing the accesses of all devices to the main memory through the same

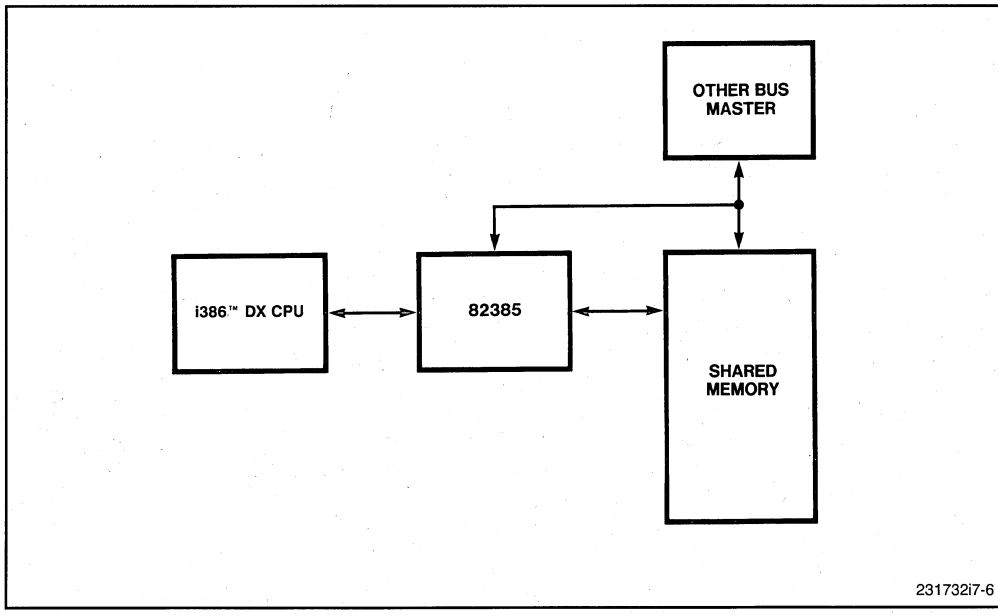


Figure 7-6. Bus Watching

cache or by copying all cache writes both to the main memory and to all other caches that share the same memory (a technique known as broadcasting). Hardware transparent systems are illustrated in Figure 7-7.

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

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

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

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

7.4 EFFICIENCY AND PERFORMANCE

The measurement of cache effectiveness is divided into two topics: efficiency and performance. Cache efficiency is its ability to maintain the most used code and data requested by the microprocessor. Efficiency is measured in terms of hit rate. Performance is a measurement of the speed in which a microprocessor can perform a given

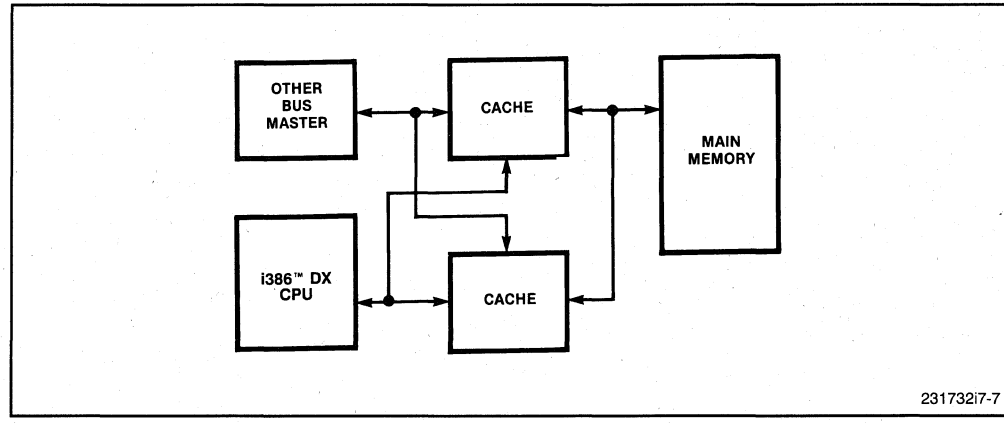


Figure 7-7. Hardware Transparency

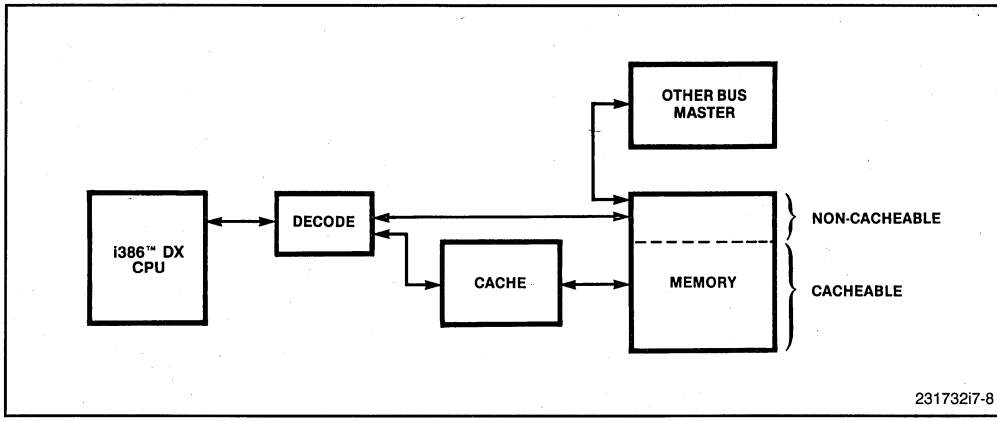


Figure 7-8. Non-Cacheable Memory

task, and is measured in effective wait-states. Hit rate is but one of many factors which affect performance. Write policy, update policy, and coherency methods are performance factors as well.

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

Table 7-1. Cache Hit Rates

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

7.5 CACHE AND DMA

Cache coherency is an issue one must consider when placing a DMA controller in an Intel386 DX microprocessor system. Because the DMA controller has access to main memory, it can potentially introduce stale data. As was mentioned before, stale data can be avoided in the following ways:

- Implementing bus watching (snooping). In this approach, the DMA controller writes to main memory, and the cache controller monitors DMA cycles and automatically invalidates any cache location altered by DMA.
- Implementing a transparent cache, in which memory accesses from both the Intel386 DX microprocessor and the DMA controller are directed through the cache.
- Restrict DMA cycles to non-cacheable areas of memory.

The first method has a distinct advantage: since the DMA controller does not access the cache directly, the Intel386 DX microprocessor can read from the cache while the DMA controller is moving data to the main memory. Although bus watching is difficult to implement in a discrete cache design, the 82385 integrates this function and performs zero waitstate bus watching. The overall memory bandwidth is increased since the Intel386 DX microprocessor can access its cache at the same time as the DMA controller accesses main memory.

The second approach has the advantage of requiring minimal hardware, but has the disadvantage that the Intel386 DX microprocessor must be placed in HOLD during DMA transfers. The third approach is useful if a separate, dual-ported memory can be used as the non-cacheable memory, and the DMA device is tightly coupled to this memory. In all approaches, the cache should be made software transparent, so that DMA cycles do not require special actions by software to insure cache coherency.

7.6 CACHE EXAMPLE

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

7.6.1 Example Design

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

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

The line size is four bytes, which is most convenient for the 32-bit data bus of the Intel386 DX microprocessor. An 8-byte line size would transfer twice as much data for every DRAM access, but would require a wider bus as well as more SRAMs, DRAMs, and transceivers. In such cases, one must weigh the additional cost against the additional performance.

The cache in this example stores both code and data, rather than only code. Code-only caches are easier to implement because there are no write accesses. They can be useful if data accesses are infrequent. In general, however, most programs make frequent data accesses. The code prefetch function of the Intel386 DX microprocessor makes the access time for code less critical to overall performance, since opcodes returned to the processor more quickly may only reside in the code queue longer.

7.6.2 Example Cache Memory Organization

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

The 32-bit address from the Intel386 DX microprocessor is divided into the following three fields:

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

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

The Intel386 DX microprocessor bits A31–A2 are interpreted as follows:

1. Select bit A31 is low during cache/DRAM cycles.
2. Index bits A15–A2 select the cache location.
3. Tag bits A30–A16 are compared with the tag information stored in the cache to determine of the block in the cache is the block needed by the Intel386 DX microprocessor.

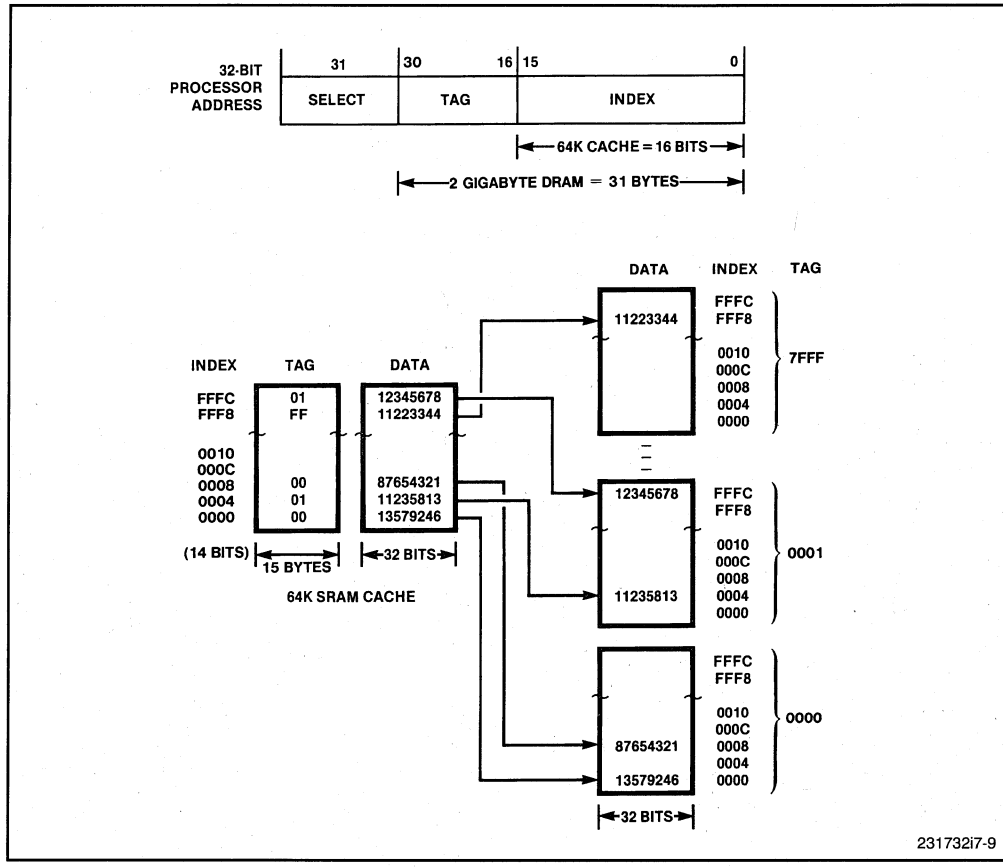


Figure 7-9. Example of Cache Memory Organization

a. If the tag matches, the Intel386 DX microprocessor either reads the data in the cache or writes new data into the cache. In the case of a write, the data is also written to the main memory.

b. If the tag does not match during a read cycle, a doubleword is read from main memory, stored in the cache, and simultaneously presented to the Intel386 DX microprocessor. The new tag is also stored in the cache. During a write, if the tag does not match, the cache is not updated.

7.7 82385 CACHE CONTROLLER

The 82385 Cache Controller enables the Intel386 DX microprocessor to reach its full performance potential by reducing the average number of wait states to nearly zero. It does this by keeping a copy of the most frequently accessed code and data from main memory in its zero wait state cache memory. When the Intel386 DX microprocessor subsequently requests this data, the 82385 will respond with zero wait states.

The 82385 resides on the Intel386 DX microprocessor local bus and interfaces directly to the Intel386 DX microprocessor. It presents a functional Intel386 DX microprocessor bus (called the 82385 local bus) for the system interface. This dual bus structure and the 82385's ability to "snoop" the system interface allows the Intel386 DX microprocessor to run locally out of the cache while another bus master has control of the 82385 local bus.

7.7.1 Bus Structure with the 82385

Figure 7-10 shows the bus structure of a typical Intel386 DX microprocessor system. The Intel386 DX microprocessor local bus consists of the physical Intel386 DX microprocessor address, data, and control buses. The local address and data buses are buffered/latched to become the system address and data buses. The local control bus is decoded by bus control logic to generate the various system bus read and write commands.

The addition of an 82385 creates two distinct buses: the actual Intel386 DX microprocessor local bus and the 82385 local bus (Figure 7-11). The 82385 local bus is functionally equivalent to the Intel386 DX microprocessor local bus, with system resources interfacing to it in the same manner as they would with the Intel386 DX microprocessor local

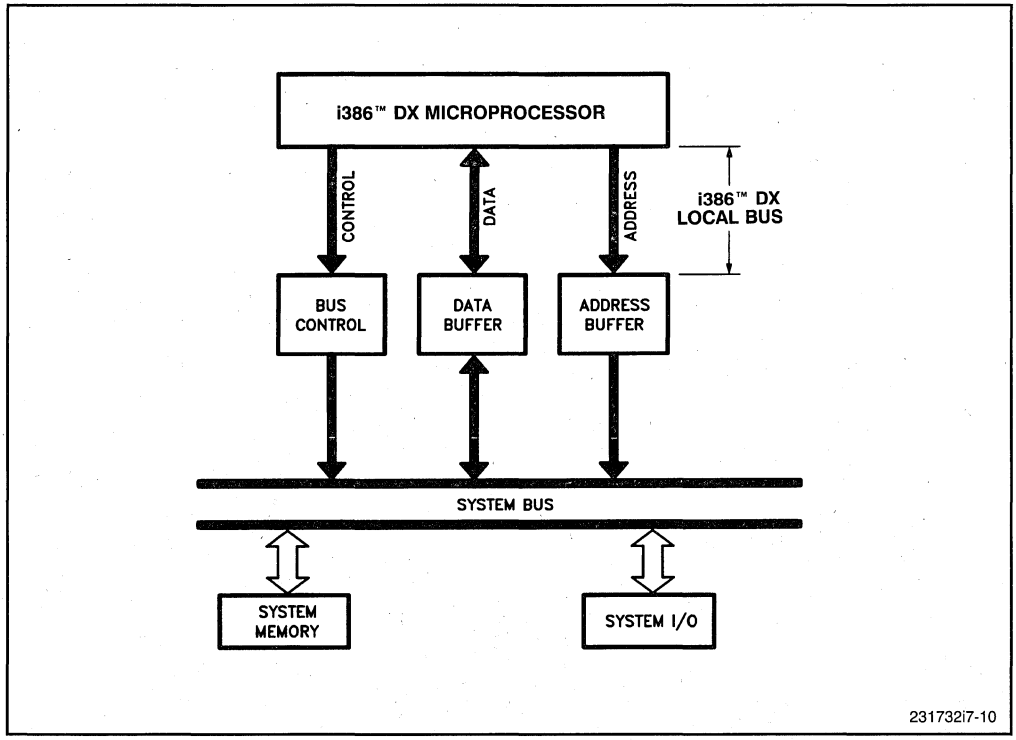


Figure 7-10. Intel386™ DX Microprocessor System Bus Structure

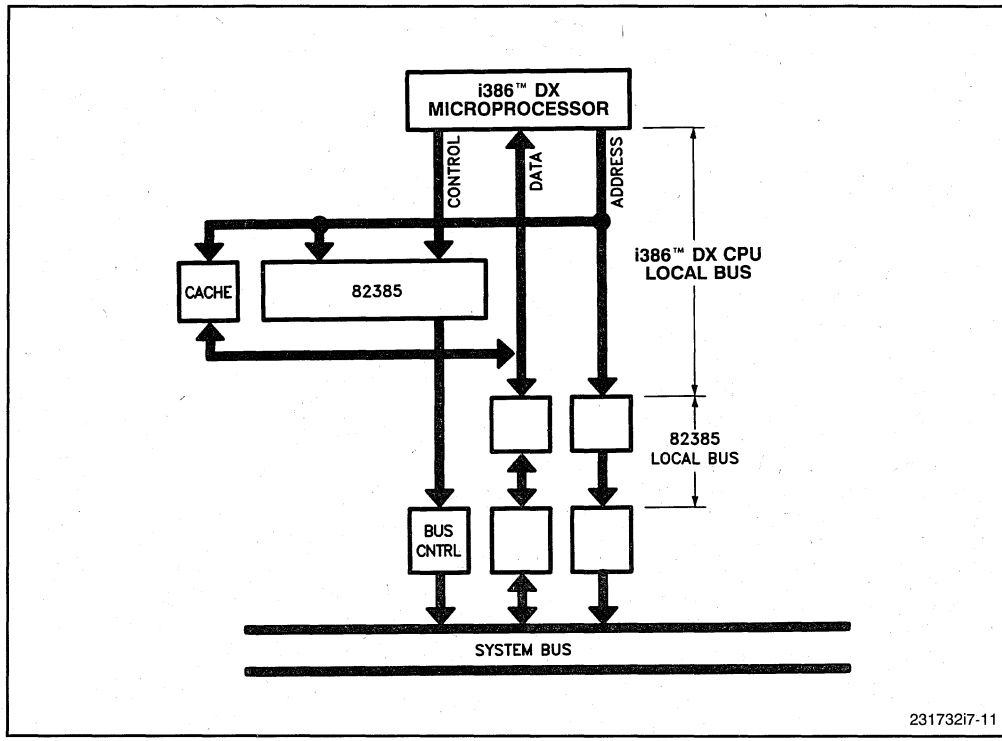


Figure 7-11. Intel386™ DX Microprocessor/82385 System Bus Structure

bus. The 82385 local bus is not simply a buffered version of the Intel386 DX microprocessor local bus, but rather is distinct from and able to operate in parallel with the Intel386 DX microprocessor local bus. The 82385 directly interfaces to the Intel386 DX microprocessor on the Intel386 DX microprocessor local bus.

7.7.2 82385/Intel386 DX Microprocessor Interface

The 82385 directly interfaces to the Intel386 DX microprocessor. It has three inputs which are used to decode Intel386 DX microprocessor local bus accesses, non-cacheable memory accesses, and 16-bit accesses. It runs fully synchronously with the Intel386 DX microprocessor and returns data with a full 32-bit data bus. The Intel387 DX math coprocessor also resides on the Intel386 DX microprocessor local bus.

7.7.2.1 Intel386 DX MICROPROCESSOR INTERFACE

The connections between the Intel386 DX microprocessor and the 82385 are shown in Figure 7-12. As can be seen, the 82385 interfaces directly to the Intel386 DX microprocessor. The 82385 setup specifications for the Intel386 DX microprocessor address and control bus are designed to meet the Intel386 DX microprocessor output delays.

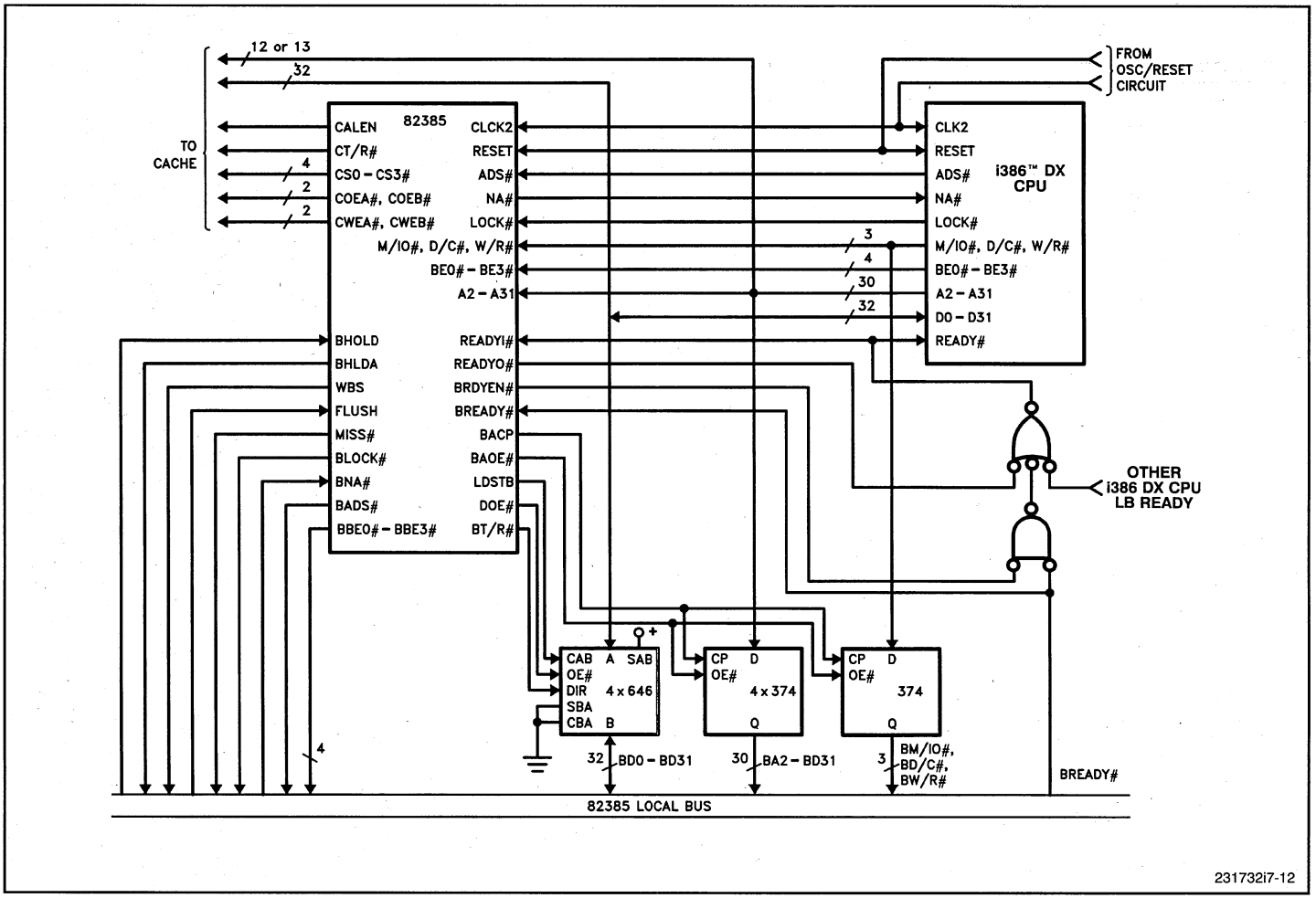


Figure 7-12. Intel386™ DX Microprocessor/82385 Interface

7.7.2.2 Intel387 DX MATH COPROCESSOR INTERFACE

Coprocessor cycles are indicated when the Intel386 DX microprocessor generates I/O cycles to addresses 800000F8H and 800000FCH. The 82385 monitors the Intel386 DX microprocessor M/IO# and A31 signals to determine when the coprocessor is being accessed. When a coprocessor access is encountered by the 82385, the cycle is effectively ignored (the 82385 remains idle) during the cycle.

Care must be taken in designs which allow the Intel387 DX math coprocessor to be an option. Any time that the 82385 recognizes a coprocessor access by the Intel386 DX microprocessor, it will remain idle until the cycle is terminated. Therefore, if the Intel386 DX microprocessor executes a coprocessor cycle without the Intel387 DX math coprocessor being present, the 82385 must see a READYI# to indicate the completion of the cycle. Therefore, these cycles must be locally terminated.

7.7.2.3 82385 SYSTEM CONFIGURATION INPUTS

The 82385 offers three inputs which are used to allow various system configurations. The inputs allow for Intel386 DX microprocessor local bus accesses (LBA#), for non-cacheable memory accesses (NCA#), and 16-bit accesses (X16#). These 82385 inputs are required to be activated in the first state where addresses are valid (T1 or first T2P) and must remain valid until addresses change from the Intel386 DX microprocessor (after the last T2 or T1P).

Non-Cacheable Accesses—NCA#. NCA# allows areas of memory to be mapped as non-cacheable. Memory mapped I/O and dual-ported memory are typical examples of areas which are generally non-cacheable.

16-Bit Transfers—X16#. 16-bit transfers can be managed by the Intel386 DX microprocessor by using its BS16# input. The 82385 can accommodate these transfers by the use of its X16# input. If X16# is activated, the access is treated as non-cacheable. The Intel386 DX microprocessor byte enables (BE0#–BE3#) are monitored by the 82385 to determine if it must lock two halves of a 16-bit transfer.

Intel386 DX Microprocessor Local Bus Cycles—LBA#. The 82385 LBA# input allows devices to reside on the Intel386 DX microprocessor local bus. Certain I/O ports or some memory space might be desired to be locally specific to the Intel386 DX microprocessor. The Intel387 DX math coprocessor resides on the Intel386 DX microprocessor local bus, but the 82385 internally recognizes coprocessor accesses (M/IO# low and A31 high). The Intel387 DX math coprocessor does not need to be externally decoded as a local bus device.

7.7.3 82385 Cache Organization

The cache directory and management logic are integrated into the 82385. The cache data memory consists of external SRAMs which are used to store the actual code and data. The 82385 supplies all of the necessary control signals to access the cache data memory. Via a configuration input, the 82385 can be designed as either a direct mapped cache or a two-way set associative cache.

7.7.3.1 DIRECT MAPPED ORGANIZATION

The recommended SRAM configuration for the direct mapped mode is the use of four 8K × 8 SRAMs. The 82385 will logically regard this as one bank of 8K × 32 (a total of 32 kbytes). The design can further be configured to use four bi-directional buffers (such as 74AS245s) between the SRAMs and the Intel386 DX microprocessor local data bus (see Figures 7-13 and 7-14). The buffers may be used if the SRAMs do not have an output enable or if the capacitive loading on the SRAM data pins requires substantial derating of the SRAM output enable time.

7.7.3.2 TWO-WAY SET ASSOCIATIVE ORGANIZATION

In the two-way set associative mode, the 82385 logically views the cache data memory as if it were two banks of 4K × 32 (a total of 32 kbytes). Each bank is then accessed as a cache "way" by the 82385. The typical design will incorporate eight 4K × 4 SRAMs for each bank (ideally 4K × 8 SRAMs would be used). Again, this can further be configured to use bi-directional buffers (see Figures 7-15 and 7-16).

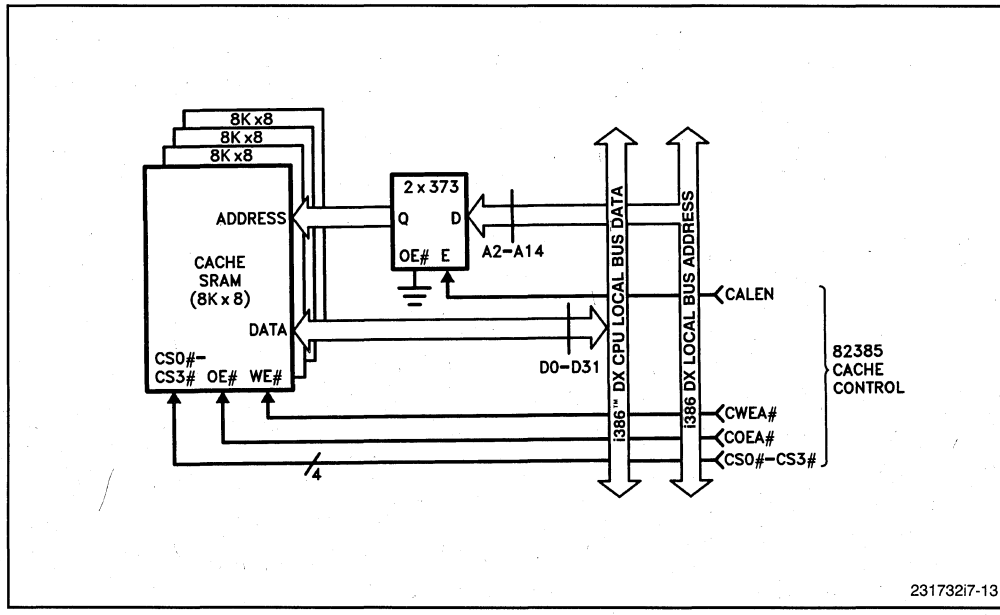


Figure 7-13. Direct Mapped Cache without Data Buffers

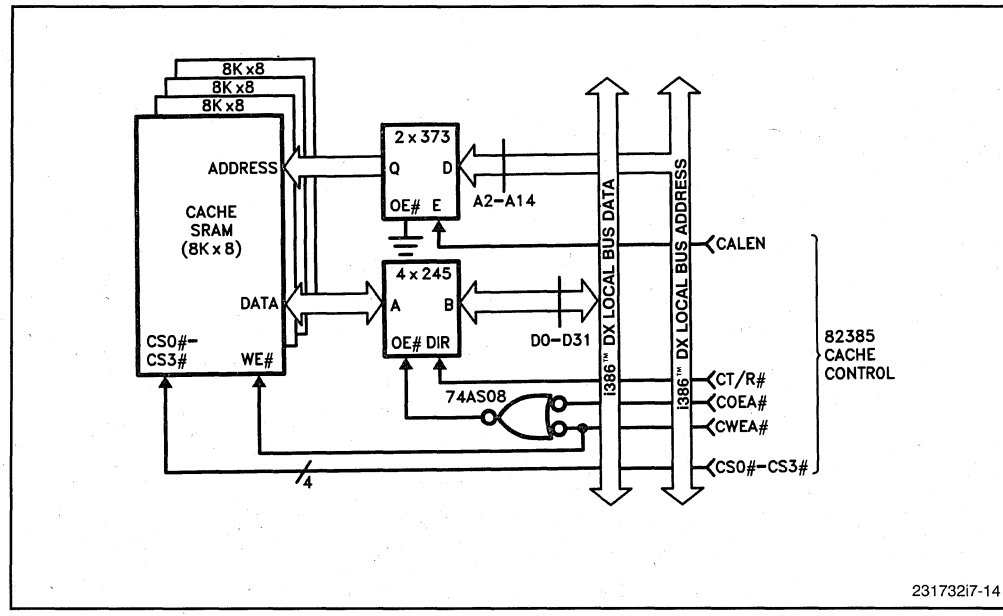


Figure 7-14. Direct Mapped Cache with Data Buffers

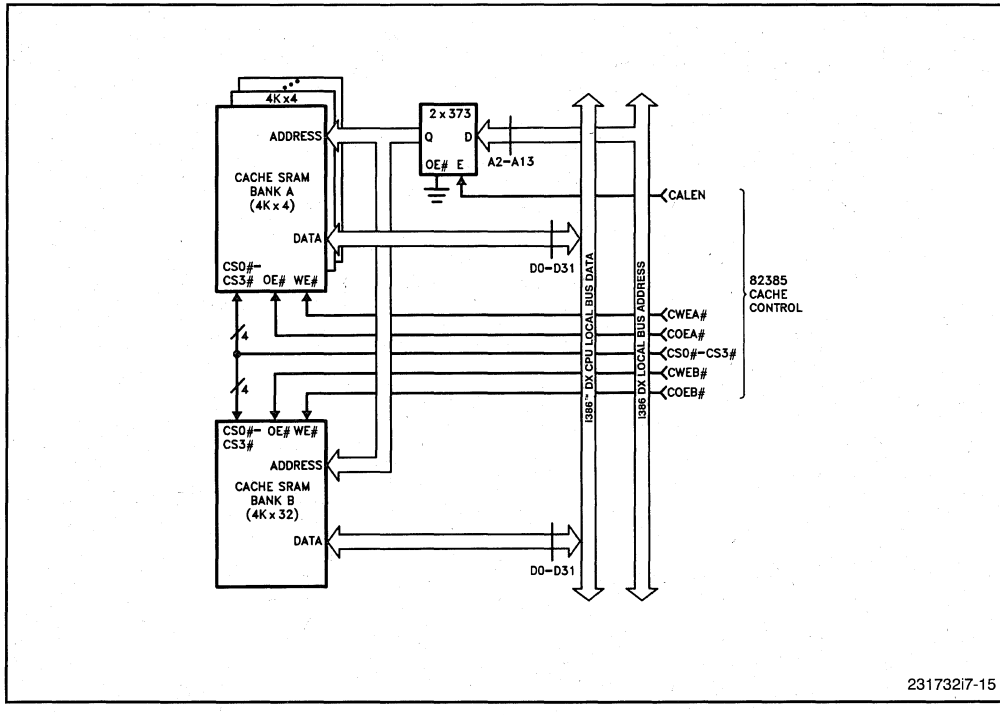


Figure 7-15. Two-Way Set Associative Cache without Data Buffers

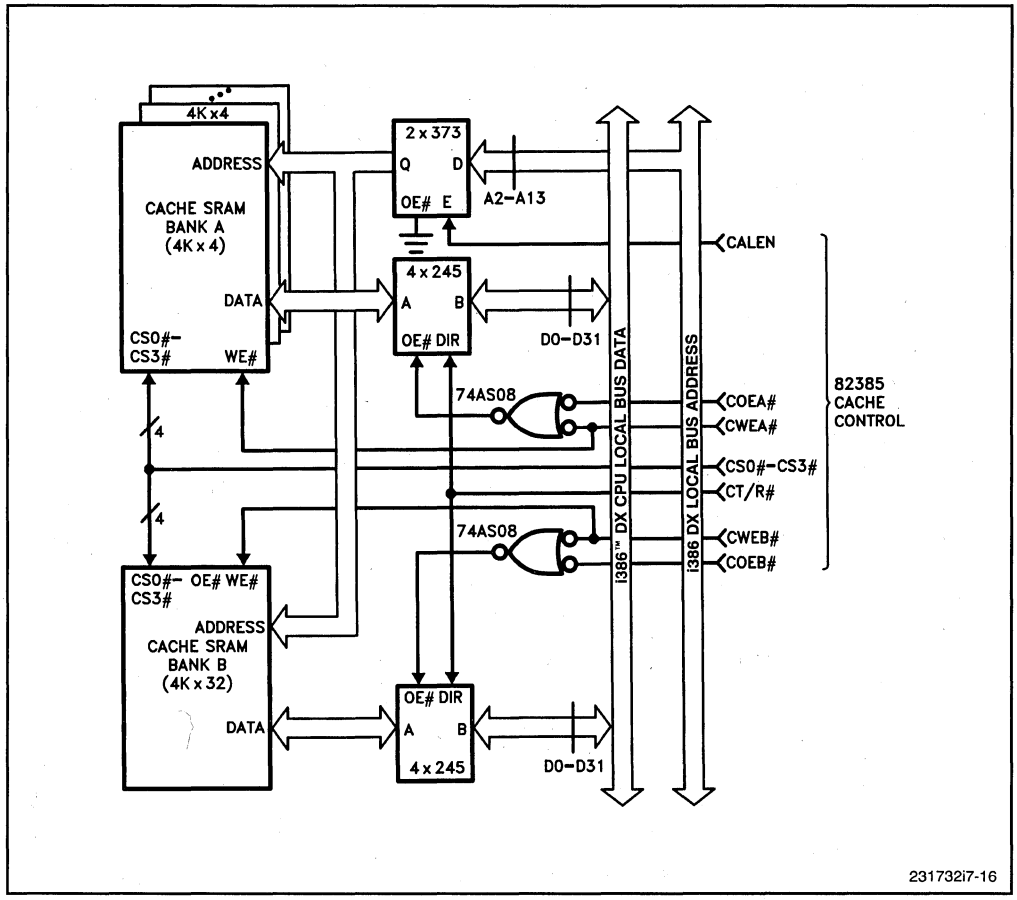


Figure 7-16. Two-Way Set Associative Cache with Data Buffers

7.7.3.3 CACHE SRAM TIMING EQUATIONS

In order to determine the required timing specifications for the SRAM being used with the 82385, it is necessary to complete a timing analysis. The following is a list of equations which can be used to determine these specifications.

Read Cycles

- Address Access Time (With Buffers)

  The smaller of:

  4xCLK2 Period – 386 Min Data Setup (t21) – 385 Max CALEN Delay (t21b) – 74AS373 C-to-Q Max Delay – 74AS245 A-to-B Max Delay

  4xCLK2 Period – 386 Min Data Setup (t21) – 385 Max Addr Valid Delay (t6) – 74AS373 D-to-Q Max Delay – 74AS245 A-to-B Max Delay

- Address Access Time (Without Buffers)

  The smaller of:

  4xCLK2 Period – 386 Min Data Setup (t21) – 385 Max CALEN Delay (t21b) – 74AS373 C-to-Q Max Delay

  4xCLK2 Period – 386 Min Data Setup (t21) – 385 Max Addr Valid Delay (t6) – 74AS373 D-to-Q Max Delay

- Chip Select Access Time (With Buffers)

  4xCLK2 Period – 386 Min Data Setup (t21) – 385 Max CS(0-3)# Delay (t23) – 74AS245 A-to-B Max Delay

- Chip Select Access Time (Without Buffers)

  4xCLK2 Period – 386 Min Data Setup (t21) – 385 Max CS(0-3)# Delay (t23)

- Output Enable to Data Valid (Direct Mapped Without Buffers)

  2xCLK2 Period – 386 Min Data Setup (t21) – 385 Max COE# Delay (t25a)

- Output Enable to Data Valid (Two-Way Without Buffers)

  2xCLK2 Period – 386 Min Data Setup (t21) – 385 Max COE# Delay (t25b)

In 82385 configurations which use buffers for the cache-data memory, the output enable time for the SRAM is effectively the address access time since there is no output enable on the SRAM itself. The 82385 controls the direction and enabling of the 74AS245 buffers.

Write Cycles

- Address Valid to End of Write

  The smaller of:

  3xCLK2 Period + 385 CWE# Min Delay (t22a) – 385 Max CALEN Delay (t21b) – 74AS373 C-to-Q Max Delay

  3xCLK2 Period + 385 CWE# Min Delay (t22a) – 386 Max Addr Valid Delay (t6) – 74AS373 C-to-Q Max Delay

- Data Setup Time (With Buffers)

  385 CWE# Min Pulse (t22b) – 74AS08 Max Prop Delay – 74AS245 Enable to Data Max Delay

- Data Hold Time (With Buffers)

  74AS08 Min Prop Delay – 74AS245 Enable to Data Min Delay

- Data Hold Time (Without Buffers)

  The smaller of:

  1xCLK2 Period + 386 Min Data Hold (t22) – 385 CWE# Max Delay (t22a)

  1xCLK2 Period + 386 Min Data Valid (t12) – 385 CWE# Max Delay (t22a)

7.7.4 System Interface

The 82385 presents the 82385 local bus for the system interface. Since the 82385 local bus is functionally equivalent to the Intel386 DX microprocessor local bus, the system interface is virtually identical. There are some timing differences that need to be understood. These relate to the data setup time and the ready setup time.

7.7.4.1 READ DATA SETUP

At 33 MHz, the read data setup time for the Intel386 DX microprocessor is 5 ns. This does not take into account any buffers in the data path which add to the setup. With an 82385 cache system, the need to update the cache memory for read miss cycles changes the data setup. The equation to determine the data setup for the buffered cache organization is given by:

74xx646 Max Propagation Delay + 74xx245 Max Propagation Delay

+ SRAM Min Write Setup + One CLK2 Period – 82385 CWE# Min Delay (82385 t22a)

The BREADY# signal is used by the 82385 to determine when the cache update can be completed for a read-miss cycle. When BREADY# is activated at the end of a cache-read miss access, it tells the 82385 to trigger the rising edge of CWE# which updates the cache-data SRAMs. For this reason, the BREADY# setup (82385 t37a) for a cache-read miss is 13 ns at 33 MHz.

7.7.5 Special Design Notes

The ability to disable the cache is useful for system memory diagnostic purposes. While the 82385 itself does not have a specific cache enable/disable feature, the FLUSH input can effectively be used to disable the cache. By keeping the FLUSH input active, all Intel386 DX microprocessor memory accesses (which have LBA# inactive) will be forwarded to the system bus. The FLUSH pin invalidates all of the directory tags which makes the cycles read misses. In addition, no coherency issues will exist when the cache is enabled since it was previously flushed.

Certain architectures which use the Intel386 DX microprocessor implement a special function for the A20 address line. When the Intel386 DX microprocessor is running in Real Mode, the A20 line is kept active low (via a registered I/O port) so that regardless

of the state of A20, the memory subsystem will always see it inactive. In Protected Mode, the true state of A20 is forwarded to the system. Beginning with the 82385 (B) step, 6 ns of time will be available from the Intel386 DX microprocessor A20 valid specification to the 82385 A20 setup specification. This allows a logic gate (such as a 74AS08) to reside between the Intel386 DX microprocessor and the 82385 for the A20 line. The A20 address line can be manipulated between the Intel386 DX microprocessor and the 82385 in order to handle any possible coherency issues.

I/O Interfacing 8

CHAPTER 8
I/O INTERFACING

The Intel386 DX microprocessor supports 8-bit, 16-bit, and 32-bit I/O devices that can be mapped into either the 64-kilobyte I/O address space or the 4-gigabyte physical memory address space. This chapter presents the issues to consider when designing an interface to an I/O device. Mapping as well as timing considerations are described. Several examples illustrate the design concepts.

8.1 I/O MAPPING VERSUS MEMORY MAPPING

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

- The address decoding required to generate chip selects for I/O-mapped devices is often simpler than that required for memory-mapped devices. I/O-mapped devices reside in the I/O space of the Intel386 DX microprocessor (64 kilobytes); memory-mapped devices reside in a much larger memory space (4 gigabytes) that makes use of more address lines.

- Memory-mapped devices can be accessed using any Intel386 DX microprocessor instruction, so I/O-to-memory, memory-to-I/O, and I/O-to-I/O transfers as well as compare and test operations can be coded efficiently. I/O-mapped devices can be accessed only through the IN, OUT, INS, and OUTS instructions. All I/O transfers are performed via the AL (8-bit), AX (16-bit), or EAX (32-bit) registers. The first 256 bytes of the I/O space are directly addressable. The entire 64-kilobyte I/O space is indirectly addressable through the DX register.

- Memory mapping offers more flexibility in protection than I/O mapping does. Memory-mapped devices are protected by memory management and protection features. A device can be inaccessible to a task, visible but protected, or fully accessible, depending on where the device is mapped in the memory space. Paging provides the same protection levels for individual 4-kilobyte pages and indicates whether a page has been written to. The I/O privilege level of the Intel386 DX microprocessor protects I/O-mapped devices by either preventing a task from accessing any I/O devices or by allowing a task to access all I/O devices. A virtual-8086-mode I/O permission bitmap can be used to select the privilege level for a combination of I/O bytes.

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

The Intel386 DX microprocessor can operate with 8-bit, 16-bit, and 32-bit peripherals. The interface to a peripheral device depends not only upon data width, but also upon the signal requirements of the device and its location within the memory space or I/O space.

8.2.1 Address Decoding

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

One possible technique for decoding memory-mapped I/O addresses is to map the entire I/O space of the Intel386 DX microprocessor into a 64-kilobyte region of the memory space. The address decoding logic can be configured so that each I/O device responds to both a memory address and an I/O address. Such a configuration is compatible for both software that uses I/O instructions and software that assumes memory-mapped I/O.

Address decoding can be simplified by spacing the addresses of I/O devices so that some of the lower address lines can be omitted. For example, if devices are placed at every fourth address, the Intel386 DX microprocessor Byte Enable outputs (BE3#–BE0#) can be ignored for I/O accesses and each device can be connected directly to the same eight data lines. The 64-kilobyte I/O space is large enough to allow the necessary freedom in allocating addresses for individual devices.

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

8.2.2 8-Bit I/O

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

In a write cycle, if BE3# and/or BE2# is active but not BE1# or BE0#, the write data on the top half of the data bus is duplicated on the bottom half. If the addresses of two devices differ only in the values of BE3#–BE0# (the addresses lie within the same doubleword boundaries), BE3#–BE0# must be decoded to provide a chip select signal that prevents a write to one device from erroneously performing a write to the other. This chip select can be generated using an address decoder PLD such as the 85C508 device or TTL logic.

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

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

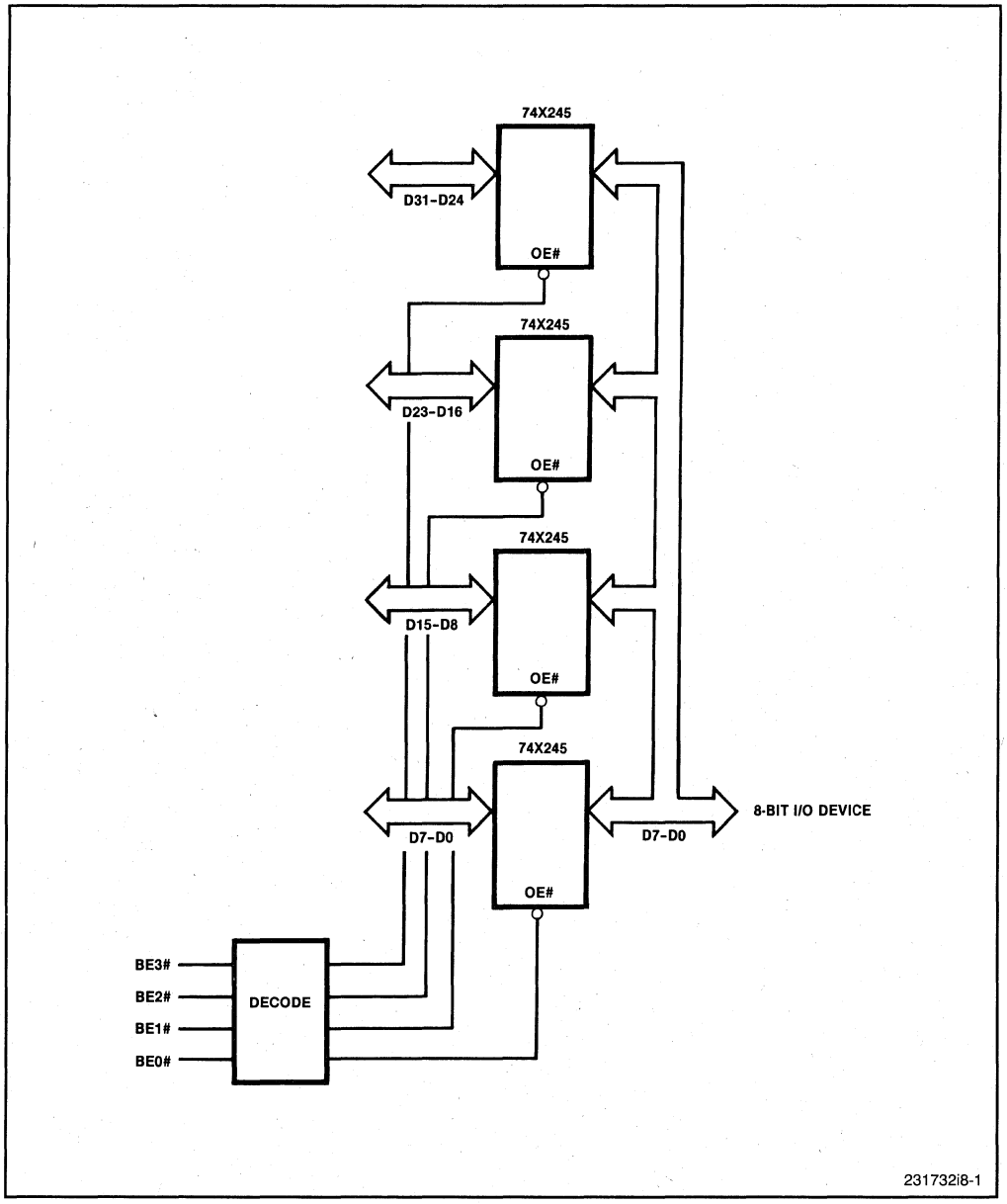


Figure 8-1. 32-Bit to 8-Bit Bus Conversion

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

8.2.3 16-Bit I/O

To avoid extra bus cycles and to simplify device selection, 16-bit I/O devices should be assigned to even addresses. If I/O addresses are located on adjacent word boundaries, address decoding must generate the Bus Size 16 (BS16#) signal so that the Intel386 DX microprocessor performs a 16-bit bus cycle. If the addresses are located on every other word boundary (every doubleword address), BS16# is not needed.

8.2.4 32-Bit I/O

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

8.2.5 Linear Chip Selects

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

8.3 BASIC I/O INTERFACE

In a typical Intel386 DX microprocessor system design, a number of slave I/O devices can be controlled through the same local bus interface. Other I/O devices, particularly those capable of controlling the local bus, require more complex interfaces. This section presents a basic interface for slave peripherals.

The high performance and flexibility of the Intel386 DX microprocessor local bus interface plus the increased availability of programmable and semi-custom logic make it feasible to design custom bus control logic that meets the requirements of particular system.

The basic I/O interface shown in Figure 8-3 can be used to connect the Intel386 DX microprocessor to virtually all slave peripherals. The following list includes some common peripherals compatible with this interface:

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

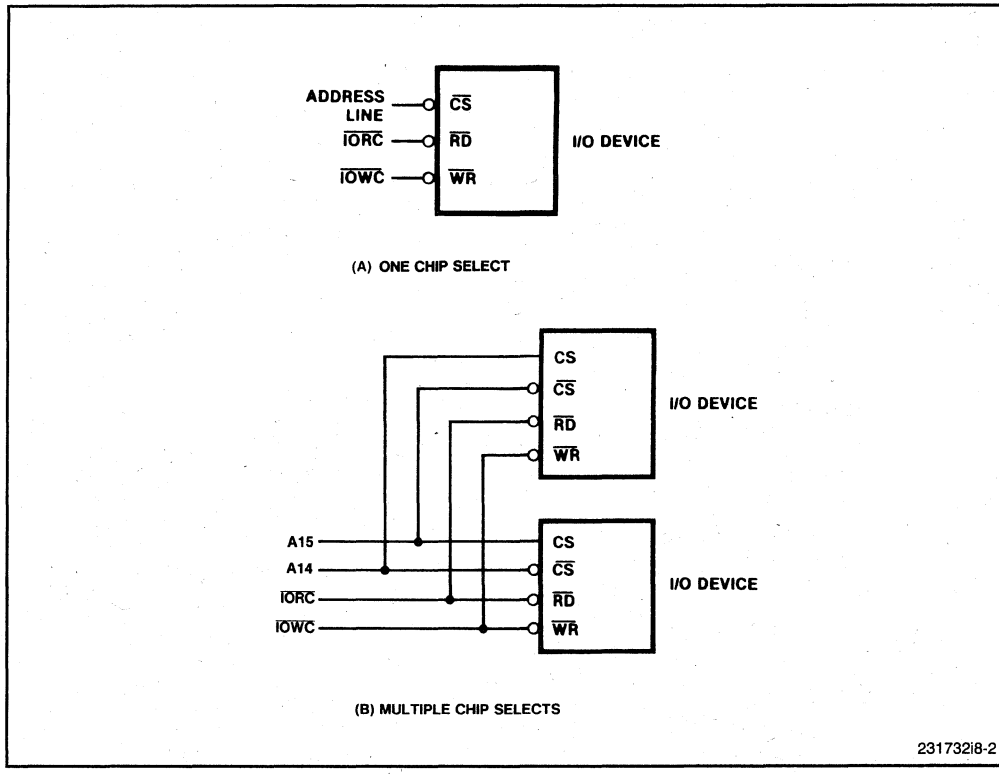


Figure 8-2. Linear Chip Selects

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

8.3.1 Address Latch

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

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

8.3.2 Address Decoder

In this example, the address decoder, which converts the Intel386 DX microprocessor address into chip-select signals, is located before the address latches. In general, the decoder may also be placed after the latches. If it is placed before the latches, the

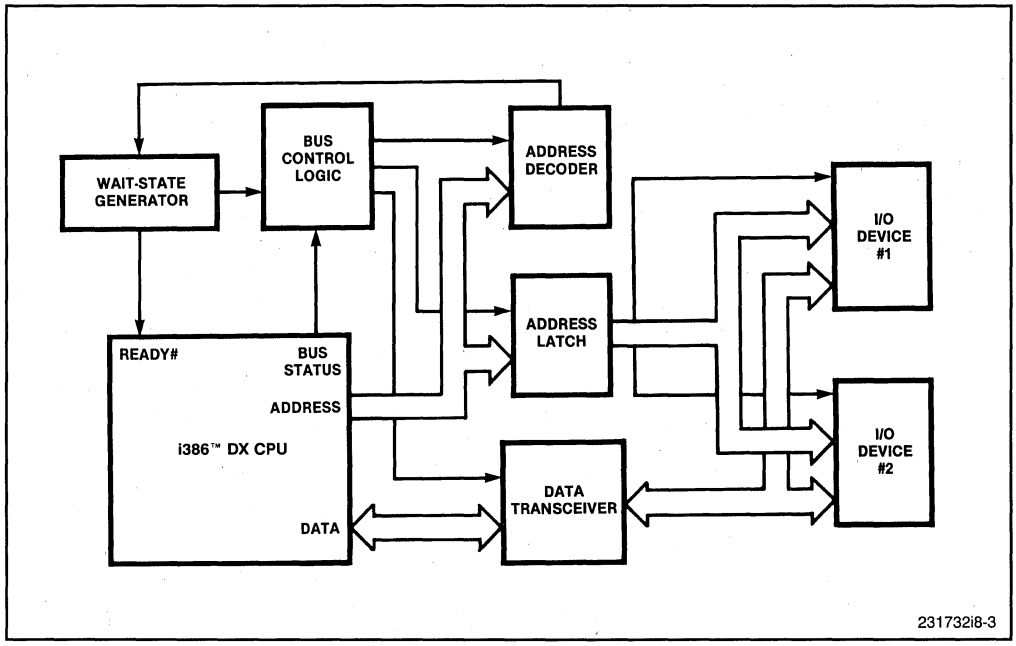


Figure 8-3. Basic I/O Interface Block Diagram

chip-select signal becomes valid as early as possible but must be latched along with the address. Therefore, the number of address latches needed is determined by the location of the address decoder as well as the number of address bits and chip-select signals required by the interface. The chip-select signals are routed to the bus control logic to set the correct number of wait states for the accessed device.

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

8.3.3 Data Transceiver

Standard 8-bit transceivers (74x245, in this example) provide isolation and additional drive capability for the Intel386 DX microprocessor data bus. Transceivers are necessary to prevent the contention on the data bus that occurs if some devices are slow to remove read data from the data bus after a read cycle. If a write cycle follows a read cycle, the Intel386 DX microprocessor may drive the data bus before a slow device has removed its outputs from the bus, potentially causing bus contention problems. Transceivers can be omitted only if the data float time of the device is short enough and the load on the Intel386 DX microprocessor data pins meets device specifications.

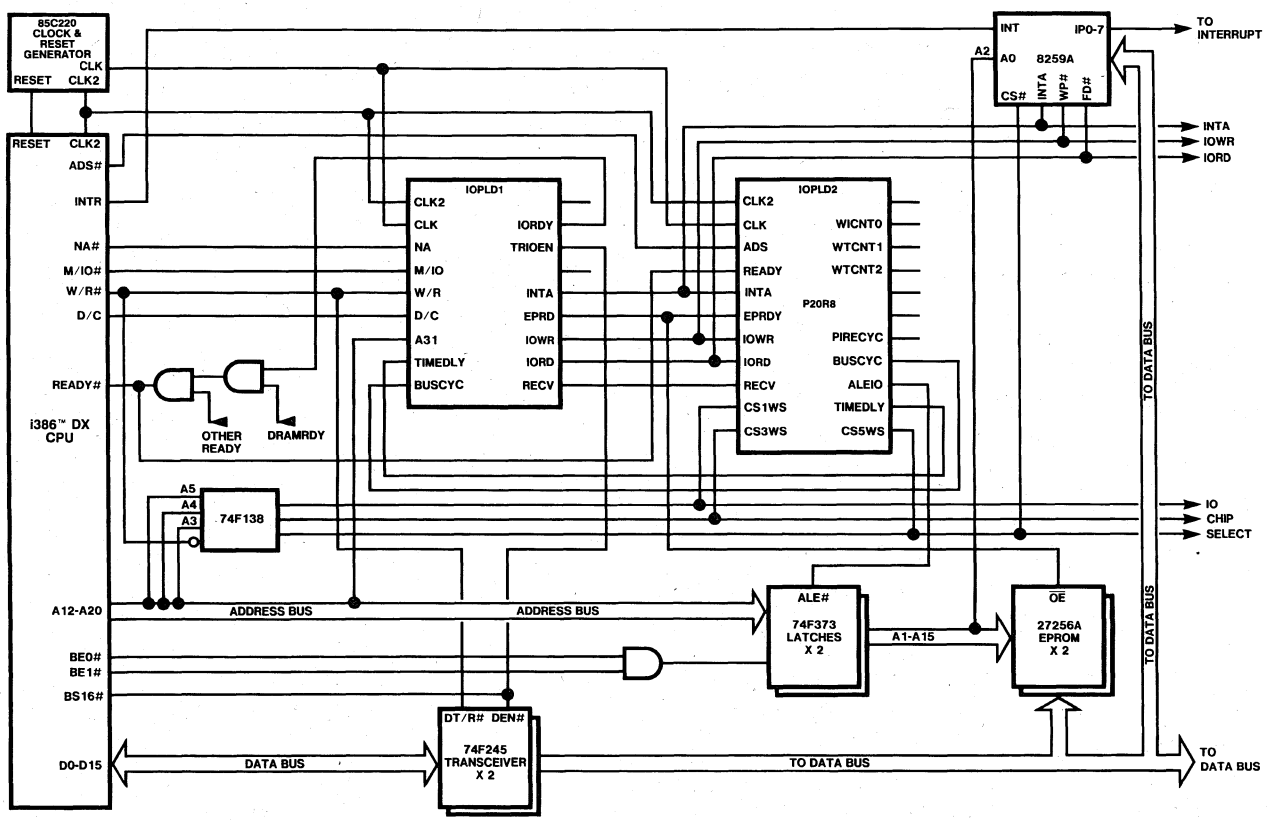


Figure 8-4. I/O Controller Schematic

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

The 74x245 transceiver is controlled through two input signals:

- Data Transmit/Receive (DT/R#)—When high, this input enables the transceiver for a write cycle. When low, it enables the transceiver for a read cycle. This signal is just a latched version of the Intel386 DX microprocessor W/R# output.
- Data Enable (DEN#)—When low, this input enables the transceiver outputs. This signal is generated by the bus control logic.

Note that in a system using the 82380, the data transceivers must be disabled whenever the Intel386 DX microprocessor performs a read access to one of the internal registers of the 82380. Otherwise, both the 82380 and the data transceivers will be driving the local bus which causes data contention. This can be avoided by decoding the 82380 address space in the bus controller logic. Together with the bus cycle definition signals (W/R#, M/IO#), the data transceivers can be disabled by deactivating the DEN# signal.

8.3.4 Bus Control Logic

The bus control logic for the basic I/O interface is the same as the logic for the memory interface described in Section 6.2. The bus controller decodes the Intel386 DX microprocessor status outputs (W/R#, M/IO#, and D/C#) and activates a command signal for the type of bus cycle requested. The command signal corresponds to the bus cycle types (described in Chapter 3) as follows:

- EPROM data read and memory code read cycles generate the Memory Read Command (EPRD#) output. EPRD# commands the selected memory device to output data.
- I/O read cycles generate the I/O Read Command (IORD#) output. IORD# commands the selected I/O device to output data.
- I/O write cycles generate the I/O Write Command (IOWR#) output. IOWR# commands the selected memory device to receive the data on the data bus.

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

The bus controller also controls the READY# input to the Intel386 DX microprocessor that ends each bus cycle. The IOPLD2 bus control PLD counts wait states and returns TIMEDLY# after the number of wait states required by the accessed device. The design of this portion of the bus controller depends on the requirements of the system; relatively simple systems need less wait-state logic than more complex systems. The basic interface described here uses a PLD device to generate TIMEDLY#; other designs may use counters and/or shift registers.

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

8.4 TIMING ANALYSIS FOR I/O OPERATIONS

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

Timing for Intel386 DX microprocessor I/O cycles is identical to memory cycle timing in most respects; in particular, timing depends on the design of the interface. The worst-case timing values are calculated by assuming the maximum delay in the address latches, chip select logic, and command signals, and the longest propagation delay through the data transceivers (if used). These calculations yield the minimum possible access time for an I/O access for comparison with the access time of a particular I/O device. Wait states must be added to the basic worst-case values until read and write cycle times exceed minimum device access times.

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

The CSWS signal requires a maximum decoder delay of:

(4 x CLK2)	– Intel386 DX microprocessor Addr Valid	– PLD setup
(4 x 25)	– 30	– 15

= 55 nanoseconds
(CLK2 = 40 MHz)

The timings of the other signals can be calculated from the waveforms in Figure 8-5. In the following example, the timings for I/O accesses are calculated for CLK2 = 40 MHz, 85C220-66 (12 ns) EPLDs to implement IOPLD1 and Clock circuitry, and a 20R8 PLD to implement IOPLD2. All times are in nanoseconds.

tAR: Address stable before Read (IORD# fall)
tAW: Address stable before Write (IOWR# fall)

(2 x CLK2) + PLD RegOut Min	–PLD RegOut Max	–Latch Enable Max
(2 x 25) + 1.5	– 12	– 13

= 26.5 nanoseconds

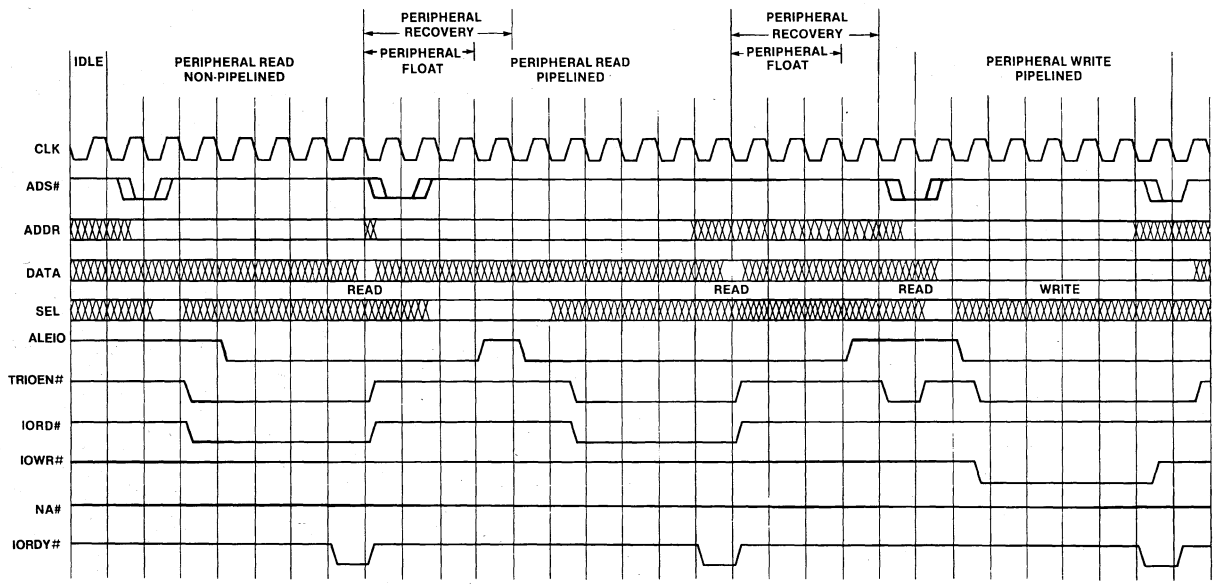


Figure 8-5. Basic I/O Timing Diagram

tRR: Read (IORD#) pulse width,
TWW: Write (IOWR#) Pulse Width

(10 x CLK2) − PLD RegOut Skew
(10 x 25) − 2
= 248 nanoseconds

tRA: Address hold after Read (IORD# rise)

(2 x CLK2) − PLD RegOut Max + PLD RegOut Min + Latch Enable Min
(2 x 25) − 6 + 2 + 5

= 53 nanoseconds

tAD: Data delay from Address

(12 x CLK2) − PLD RegOut Max + Latch Enable Max − xcvr. prop Min − Intel386 DX microprocessor Data Setup Min
(12 x 25) − 6 − 13 − 6 − 11
= 264 nanoseconds

tRD: Data delay from Read (IORD#)

(10 x CLK2) − PLD RegOut Max − xcvr. prop Min − Intel386 DX microprocessor Data Setup Min
(10 x 25) − 6 − 6 − 11

= 227 nanoseconds

tDW: Data setup before write (IOWR# rise)

(10 x CLK2) − PLD RegOut Max − xcvr. Enable Max + PLD RegOut Min
(10 x 25) − 12 − 11 + 1.5

= 228.5 nanoseconds

Many peripherals require a minimum recovery time between back-to-back accesses. This recovery time is usually provided in software by a series of NOP instructions. A JMP to the next instruction also provides a delay because it flushes the Intel386 DX microprocessor Prefetch Queue; this method has a more predictable execution time than the NOP method.

In Intel386 DX microprocessor systems, the instructions that provide recovery time are executed more quickly than in earlier systems. For software compatibility with earlier microprocessor generations, hardware must guarantee the recovery time. However, the circuitry to delay bus commands selectively for the specific instance of back-to-back accesses to a particular device is typically more complex than the frequency of such accesses justifies. Therefore, the preferred solution is to delay all I/O cycles by the minimum recovery time. Because most I/O accesses are relatively infrequent, performance is not degraded.

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

8.5 BASIC I/O EXAMPLES

In this section, two examples of the interface to slave I/O devices are presented. Typically, several of these devices exist on the Intel386 DX microprocessor local bus. The basic I/O interface presented above is used for both examples.

8.5.1 8274 Serial Controller

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

Figure 8-6 shows connections from the basic I/O interface through which the Intel386 DX microprocessor communicates with the 8274. The 8274 is accessed as a sequence of four 8-bit I/O addresses (I/O-mapped or memory-mapped). The Serial I/O (SERIO#) signal is a chip select generated by address decoding logic. RD# and WR# signals are provided by the bus control logic. DB7–DB0 inputs connect to the lower eight outputs of the data transceiver (D7–D0).

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

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

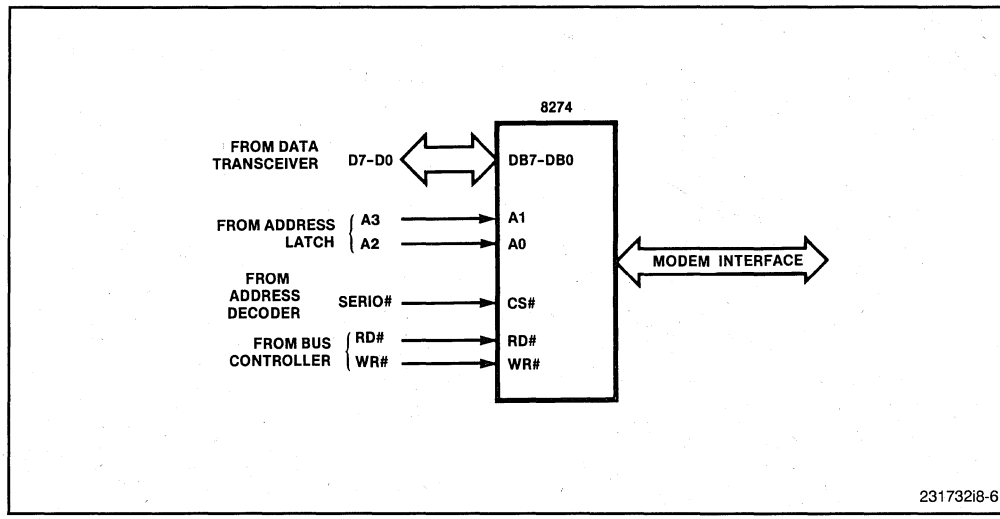


Figure 8-6. 8274 Interface

8.5.2 82380 Programmable Interrupt Controller

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

The 82380 PIC handles interrupt priority resolution and returns a preprogrammed service routine vector to the Intel386 DX microprocessor during an interrupt acknowledge cycle. It consists of three 82C59A compatible banks. The 82380 Data Sheet contains detailed information on the 82380 PIC.

When an interrupt occurs, the 82380 PIC activates its Interrupt (INT) output, which is connected to the Interrupt Request (INTR) input of the Intel386 DX microprocessor. The Intel386 DX microprocessor automatically executes two back-to-back interrupt acknowledge cycles, as described in Chapter 3. The 82380 PIC will automatically terminate the interrupt acknowledge cycles by driving its READYO# signal. Each acknowledge cycle will be extended by five wait states. Also, four idle states are inserted by the Intel386 DX microprocessor between two consecutive interrupt acknowledge cycles.

8.5.2.1 CASCADED INTERRUPT CONTROLLERS TO THE 82380 PIC

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

For a cascaded interrupt request, the 82380 PIC will output an 8-bit cascade address on the data bus during the first interrupt acknowledge cycle. A simple circuit can latch the 8-bit address and encode it to drive the CAS signals (CAS2#–CAS0#) of the slave controllers. During the second interrupt acknowledge cycle, the 82380 will not drive the data bus; instead, the selected slave controller will put the interrupt vector on the data bus for the Intel386 DX microprocessor.

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

8.5.3 8259A Interrupt Controller

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

The 8259A handles interrupt priority resolution and returns a preprogrammed service routine vector to the Intel386 DX microprocessor during an interrupt-acknowledge cycle.

8.5.3.1 SINGLE INTERRUPT CONTROLLER

Figure 8-7 shows the connections from the basic I/O interface used for the Intel386 DX microprocessor and a single 8259A. Programmable Interrupt Controller (PIC#) is a chip-select signal from the address decoding logic. INTA#, RD#, and WR# are generated by the bus control logic. BD7–BD0 are connected to the lower eight outputs of the

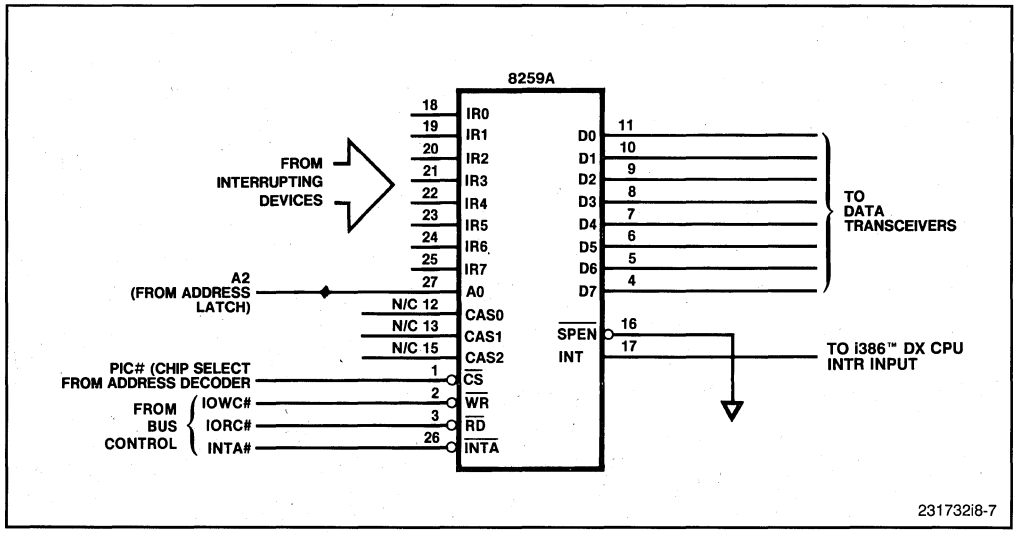


Figure 8-7. Single 8259A Interface

data transceiver. The A2 bit, connected to the 8259A A0 input, is used by the Intel386 DX microprocessor to distinguish between the two interrupt acknowledge cycles; 8259A register addresses must therefore be located at two consecutive double-word boundaries.

When an interrupt occurs, the 8259A activates its Interrupt (INT) output, which is connected to the Interrupt Request (INTR) input of the Intel386 DX microprocessor. The Intel386 DX microprocessor automatically executes two back-to-back interrupt-acknowledge cycles, as described in Chapter 3. The 8259A timing requirements are as follows:

- Each interrupt-acknowledge cycle must be extended by at least one wait state. Wait-state generator logic must provide for this extension.
- Four idle bus cycles must be inserted between the two interrupt-acknowledge cycles. The Intel386 DX microprocessor automatically inserts these idle cycles.

8.5.3.2 CASCADED INTERRUPT CONTROLLERS

Several 8259As can be cascaded to handle up to 64 interrupt requests. In a cascaded configuration, one 8259A is designated as the master controller; it receives input from the other 8259As, called slave controllers. The interface between the Intel386 DX microprocessor and multiple cascaded 8259As is an extension of the single-8259A interface with the following additions:

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

Each slave controller resolves priority between up to eight interrupt requests and transmits a single interrupt request to the master controller. The master controller, in turn, resolves interrupt priority between up to eight slave controllers and transmits a single interrupt request to the Intel386 DX microprocessor.

The timing of the interface is basically the same as that of a single 8259A. During the first interrupt-acknowledge cycle, all the 8259As freeze the states of their interrupt request inputs. The master controller outputs the cascade address to select the slave controller that is generating the request with the highest priority. During the second interrupt-acknowledge cycle, the selected slave controller outputs an interrupt vector to the Intel386 DX microprocessor.

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

8.5.3.3 HANDLING MORE THAN 64 INTERRUPTS

If an Intel386 DX microprocessor system requires more than 64 interrupt request lines, a third level of 8259As in polled mode can be added to the configuration described above. When a third-level controller receives an interrupt request, it drives one of the

interrupt request inputs to a slave controller active. The slave controller sends an interrupt request to the master controller, and the master controller interrupts the Intel386 DX microprocessor. The slave controller then returns a service-routine vector to the Intel386 DX microprocessor. The service routine must include commands to poll the third level of interrupt controllers to determine the source of the interrupt request.

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

8.6 80286-COMPATIBLE BUS CYCLES

Some devices (the 82258, for example) require an 80286-compatible interface in order to communicate with the Intel386 DX microprocessor. An 80286-compatible interface must generate the following signals:

- Address bits A1 and A0, and Byte High Enable (BHE#) from the Intel386 DX microprocessor BE3#–BE0# outputs
- Bus cycle definition signals S0# and S1# from the Intel386 DX microprocessor M/IO#, W/R#, and D/C# outputs
- Address Latch Enable (ALE#), Device Enable (DEN), and Data Transmit/Receive (DT/R#) signals
- I/O Read Command (IORC#) and I/O Write Command (IOWC#) signals for I/O cycles
- Memory Read Command (MRDC#) and Memory Write Command (MWTC#) signals for memory cycles
- Interrupt Acknowledge (INTA#) signal for interrupt-acknowledge cycles

In the following example, the interface is constructed using the 80286-compatible bus controller (82288) and bus arbiter (82289). The 82289, along with the bus arbiters of other processing subsystems, coordinates control of the bus between the Intel386 DX microprocessor and other bus masters. The 82288 provides the control signals to perform bus cycles. Communication between the Intel386 DX microprocessor and these devices is accomplished through PLDs that are programmed to perform all necessary signal translation and generation. Latching and buffering of the data and address buses is performed by TTL logic.

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

- A0/A1 generator—Generates the lower address bits from Intel386 DX microprocessor BE0#-BE3# outputs
- Address decoder—Determines the device the Intel386 DX microprocessor will access
- Address latches—Connect directly to Intel386 DX microprocessor address pins A19–A2 and the outputs of the A0/A1 generator
- Data transceivers—Connect directly to Intel386 DX microprocessor data pins D15–D0

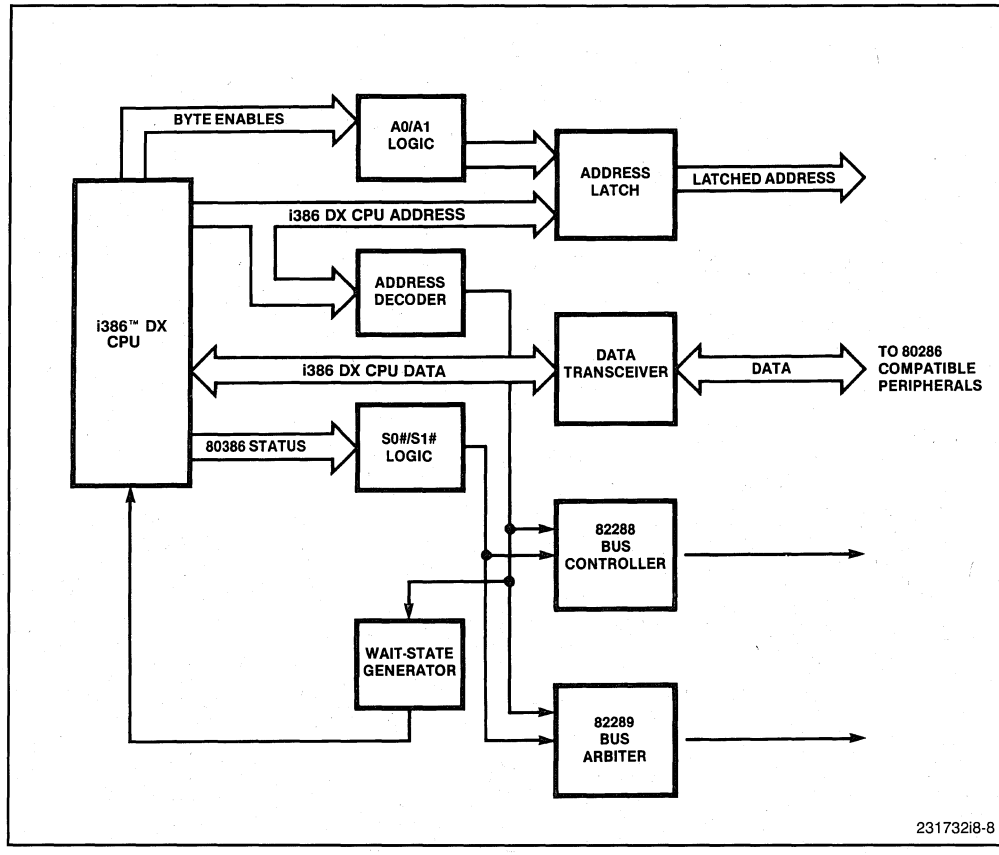


Figure 8-8. 80286-Compatible Interface

- S0#/S1# generator—Translates Intel386 DX microprocessor outputs into the S0# and S1# signals
- Wait-state generator—Controls the length of the Intel386 DX microprocessor bus cycle through the READY# signal
- 82288 Bus Controller—Generates the bus command signals
- 82289 Bus Arbiter—Arbitrates contention for bus control between the Intel386 DX microprocessor and other bus masters

8.6.1 A0/A1 Generator

The A0, A1, and BHE# signals are 80286-compatible. These signals are generated from the Intel386 DX microprocessor byte enables (BE0#–BE3#) as shown in Table 8-2. The truth table can be implemented with the logic shown in Figure 8-9.

Table 8-2. A0, A1, and BHE# Truth Table

Intel386™ DX Microprocessor Signals				16-Bit Bus Signals			Comments
BE3#	BE2#	BE1#	BE0#	A1	BHE#	BLE# (A0)	
H*	H*	H*	H*	x	x	x	x—no active bytes
H	H	H	L	L	H	L	
H	H	L	H	L	L	H	
H	H	L	L	L	L	L	
H	L	H	H	H	H	L	
H*	L*	H*	L*	x	x	x	x—not contiguous bytes
H	L	L	H	L	L	H	
H	L	L	L	L	L	L	
L	H	H	H	H	L	H	
L*	H*	H*	L*	x	x	x	x—not contiguous bytes
L*	H*	L*	H*	x	x	x	x—not contiguous bytes
L*	H*	L*	L*	x	x	x	x—not contiguous bytes
L	L	H	H	H	L	L	
L*	L*	H*	L*	x	x	x	x—not contiguous bytes
L	L	L	H	L	L	H	
L	L	L	L	L	L	L	

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

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

8.6.2 S0#/S1# Generator

S0# and S1# are 80286-compatible status signals that must be provided for the 82288 and 82289. The S0#/S1# logic in Figure 8-10 generates these signals from Intel386 DX microprocessor status outputs (D/C#, M/IO#, and W/R#) and wait-state generator outputs. WS1 and WS2 are wait-state generator outputs that correspond to the first and second wait states of the Intel386 DX microprocessor bus cycle. These signals ensure that S0# and S1# are valid for two CLK cycles.

8.6.3 Wait-State Generator

The wait-state generator PLD shown in Figure 8-11 controls the READY# input of the Intel386 DX microprocessor. For local bus cycles, the wait-state generator produces signal outputs that correspond to each wait state of the Intel386 DX microprocessor bus

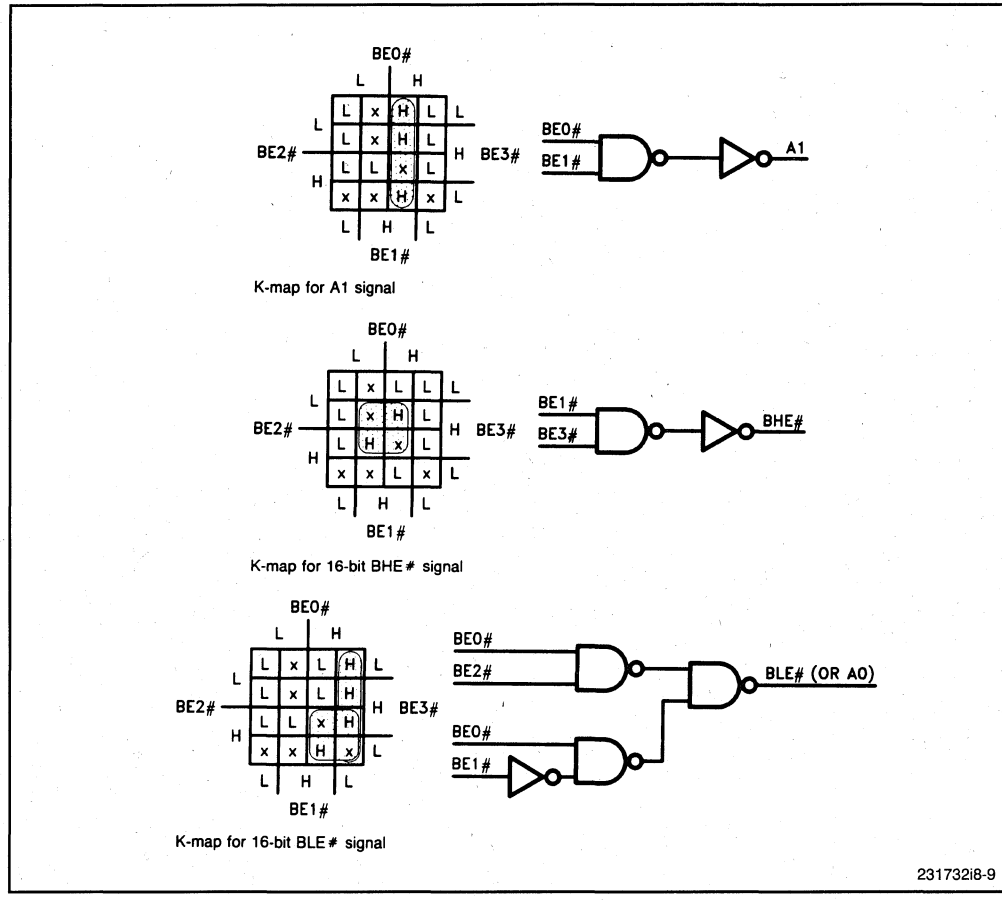


Figure 8-9. A0, A1, and BHE# Logic

cycle, and the PLD READY# output uses these signals to set READY# active after the required number of wait states. Two of the wait-state signals, WS1 and WS2, are also used to generate S0# and S1#, as shown in Figure 8-10.

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

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

RDYPLSE := ARDY * PCLK

/READY := ARDY * PCLK + RDYPLSE

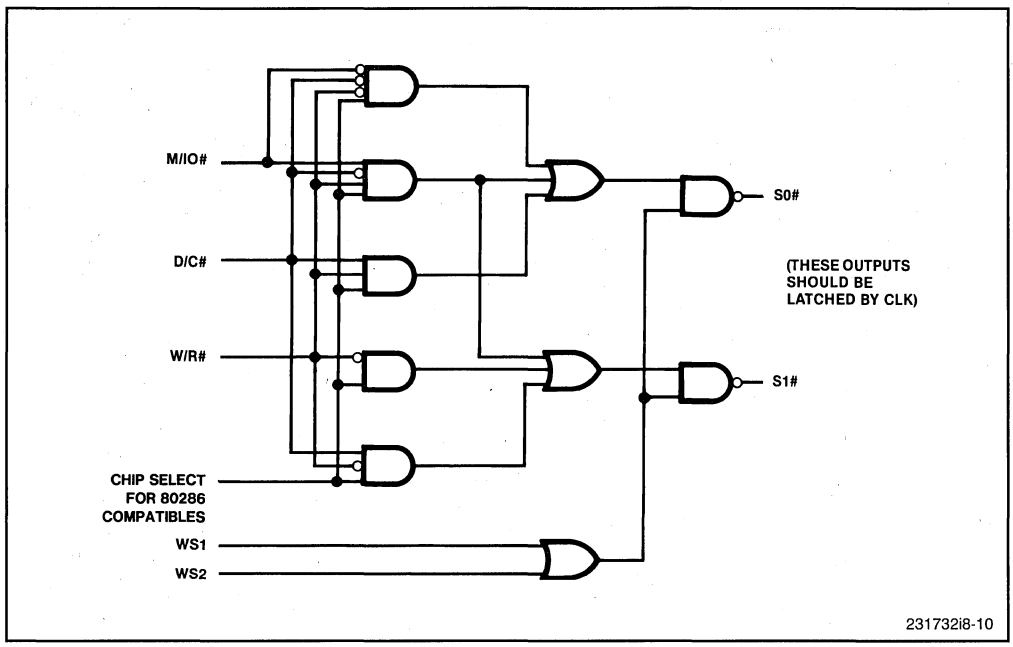


Figure 8-10. S0#/S1# Generator Logic

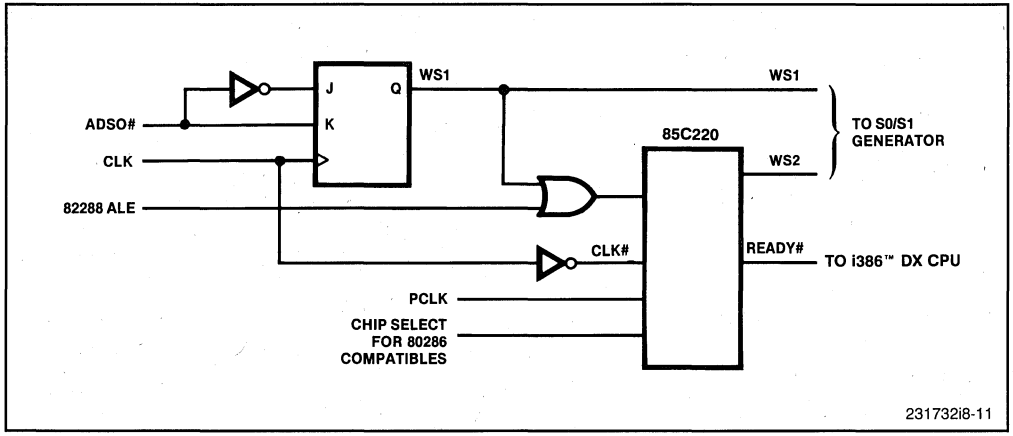


Figure 8-11. Wait-State Generator Logic

8.6.4 Bus Controller and Bus Arbiter

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

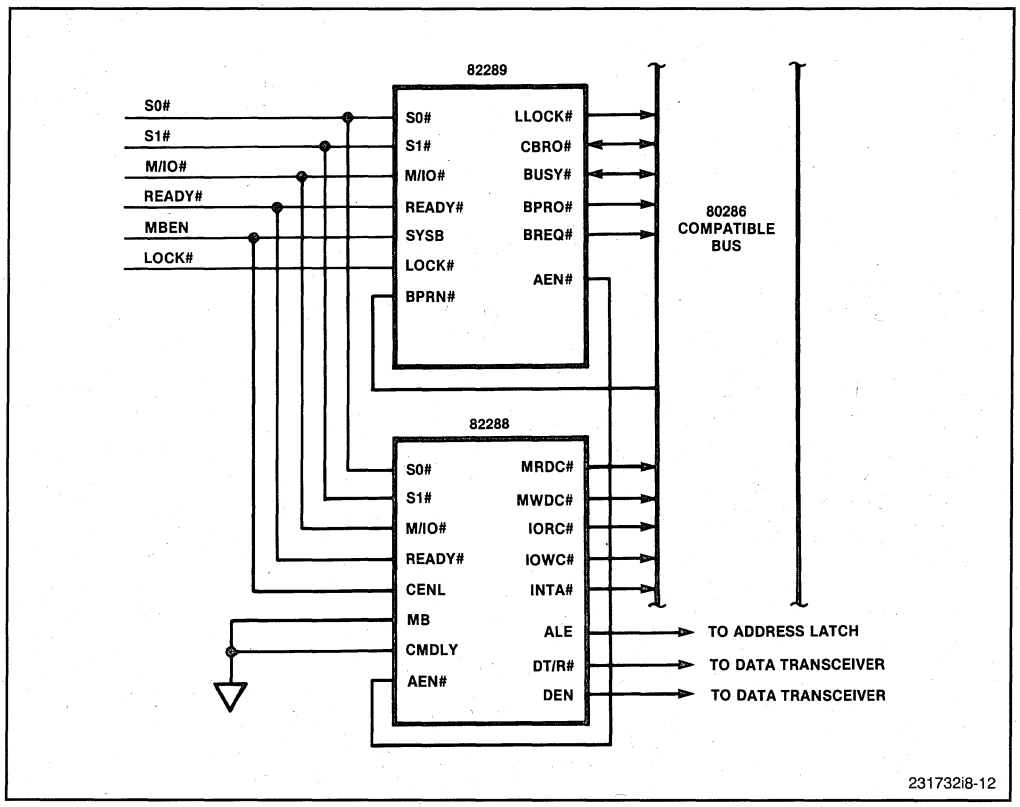


Figure 8-12. 82288 and 82289 Connections

8.6.5 82380 Integrated System Peripheral

The 82380 is designed for easy interface to the Intel386 DX microprocessor. It consists of a set of signals to interface directly to the Intel386 DX microprocessor local bus. Figure 8-13 depicts a typical system configuration with the Intel386 DX microprocessor.

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

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

8.6.6 82586 LAN Coprocessor

The 82586 is an intelligent, high-performance communications controller designed to perform most tasks required for controlling access to a local area network (LAN). In most applications, the 82586 is the communication manager for a station connected to a

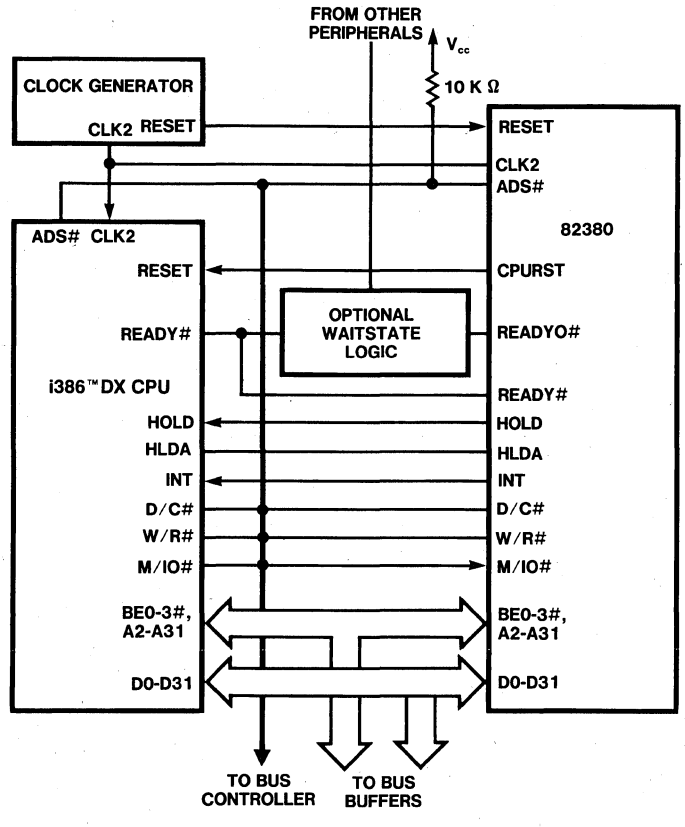


Figure 8-13. Intel386™ DX Microprocessor/82380 Interface

LAN. Such a station usually includes a host CPU, shared memory, a Serial Interface Unit, a transceiver, and a LAN link (see Figure 8-14). The 82586 performs all functions associated with data transfer between the shared memory and the LAN link, including:

- Framing
- Link management
- Address filtering
- Error detection
- Data encoding
- Network management

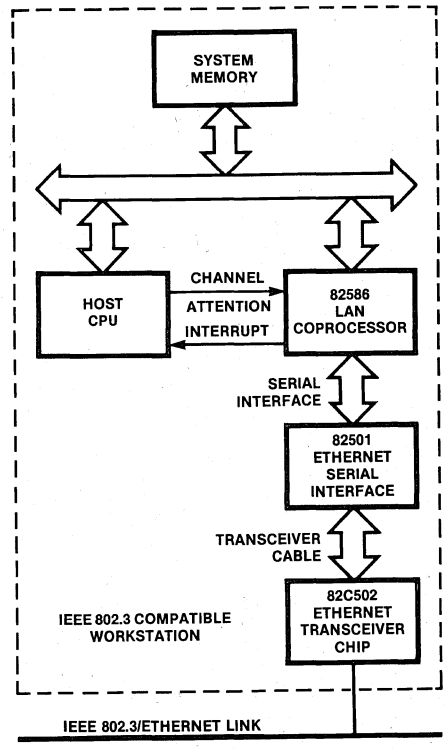


Figure 8-14. LAN Station

- Direct memory access (DMA)
- Buffer chaining
- High-level (user) command interpretation

The 82586 has two interfaces: a bus interface to the Intel386 DX microprocessor local bus and a network interface to the Serial Interface Unit. The bus interface is described here. For detailed information on using the 82586, refer to the *Local Area Networking (LAN) Component User's Manual.*

The 82586, which is a master on the Intel386 DX microprocessor local bus, communicates directly with the Intel386 DX microprocessor through the Channel Attention (CA) and interrupt (INT) signals. There are several ways to design an interface between the

82586 and the Intel386 DX microprocessor. In general, higher performance interfaces (requiring less servicing time from the Intel386 DX microprocessor) are more expensive. Four types of interfaces are described in this section:

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

8.6.6.1 DEDICATED CPU

Dedicating a CPU to control the 82586 results in a high-performance, high-cost interface. The CPU, typically an 80186, an 80188, or a microcontroller, executes the data link layer (a functional division) of software and sometimes the network, transport, and session layers as well. (For definitions of these layers, see the *Local Area Networking (LAN) Components User's Manual*). The dedicated CPU relieves the Intel386 DX microprocessor of these layers and provides a high-level, message-oriented interface that can be treated in software as a standard I/O device. In hardware, the interface is mapped into a dual-port memory.

8.6.6.2 DECOUPLED DUAL-PORT MEMORY

A decoupled dual-port memory interface, shown in Figure 8-15, contains two sections of memory:

- Intel386 DX microprocessor core memory—typically DRAM that provides executable memory space for the operating system
- 82586 communication channel memory—typically dual-ported SRAM that contains the commands and buffers of the 82586

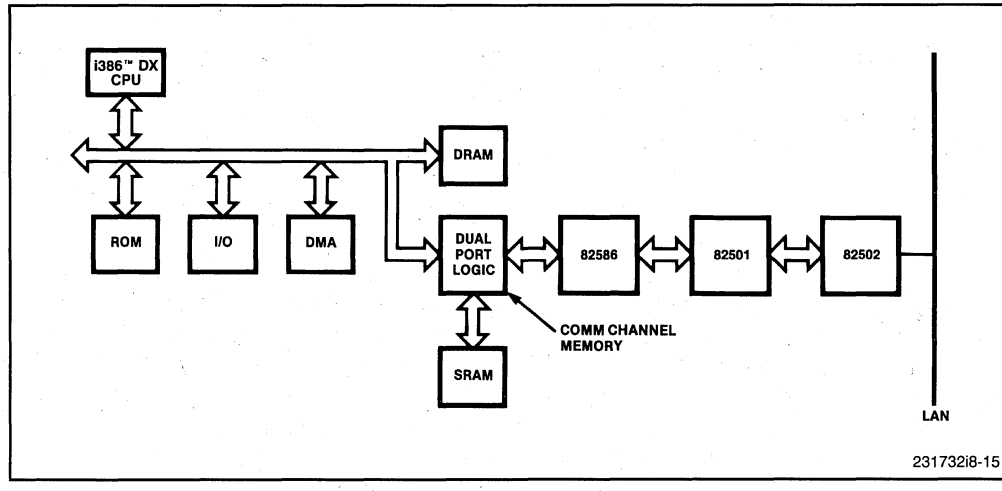


Figure 8-15. Decoupled Dual-Port Memory Interface

Only the dual-ported SRAM is shared; the 82586 cannot access the Intel386 DX microprocessor core memory. The Intel386 DX microprocessor and 82586 operate in parallel except when both require access to the SRAM. In this instance, one processor must wait while the other completes its access. At all other times, the two devices are decoupled.

This interface requires at least one level of data copying to move data between the Intel386 DX microprocessor core memory and the 82586 communication channel memory. However, usually the data must be copied to separate the frame header information.

8.6.6.3 COUPLED DUAL-PORT MEMORY

In a coupled dual-port memory interface, the Intel386 DX microprocessor and the 82586 share a common memory space as illustrated in Figure 8-16. The 82586, with 24 address bits, can address up to 16 megabytes of memory. If the Intel386 DX microprocessor memory is larger than 16 megabytes, some memory is inaccessible to the 82586; this memory must be taken into account in the system design.

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

8.6.6.4 SHARED BUS

In a shared bus interface (Figure 8-17), the Intel386 DX microprocessor and the 82586 share a common address and data bus. The HOLD and HLDA signals provide bus arbitration. When one device enters the hold state in response to the HOLD input, the other device can access the bus.

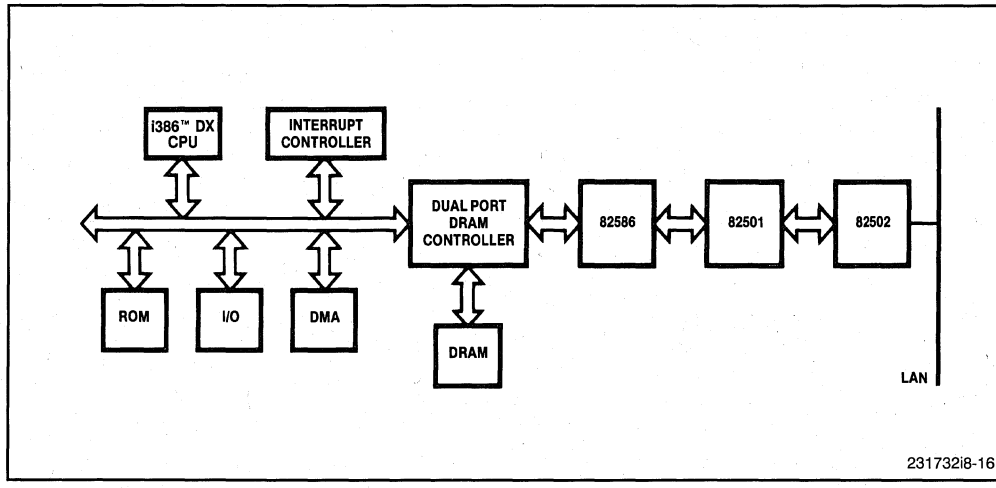


Figure 8-16. Coupled Dual-Port Memory Interface

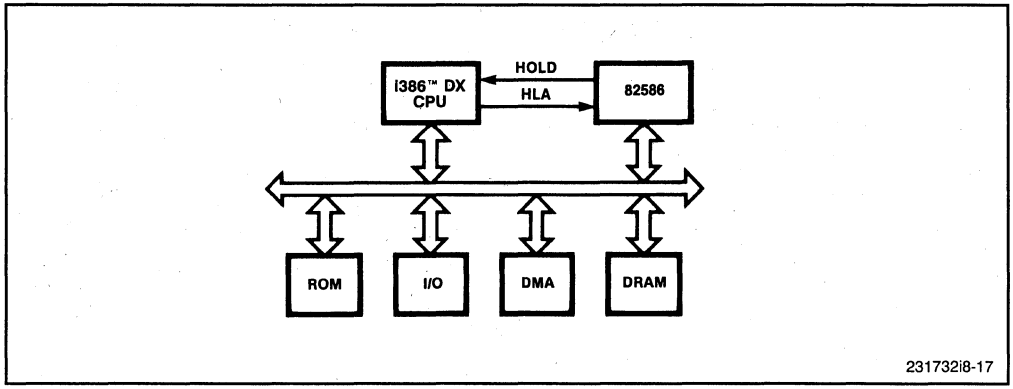


Figure 8-17. Shared Bus Interface

The shared bus interface is probably the simplest and least expensive interface. However, the performance of the Intel386 DX microprocessor may drop tremendously because the Intel386 DX microprocessor must wait for the 82586 to complete its bus operation before it can access the bus. This wait can be several hundred CLK cycles.

9

MULTIBUS I and Intel386 DX Microprocessor

CHAPTER 9
MULTIBUS I AND Intel386 DX MICROPROCESSOR

Previous chapters have presented single-bus systems in which a single Intel386 DX microprocessor connects to memory, I/O, and coprocessors. This chapter introduces the system bus, which connects several single-bus systems to create a powerful multiprocessing system. Two examples of multiprocessing system buses are the Intel MULTIBUS I, discussed in this chapter, and the Intel MULTIBUS II, discussed in Chapter 10.

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

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

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

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

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

9.1 MULTIBUS I (IEEE 796)

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

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

One method of constructing an interface between the Intel386 DX microprocessor and the MULTIBUS I is to generate all MULTIBUS I signals using only TTL and PLD devices. A simpler method is to use the 80286-compatible interface described in Chapter 8. The latter option is described in the MULTIBUS I interface example in this chapter.

9.2 MULTIBUS I INTERFACE EXAMPLE

The MULTIBUS I interface presented in the following example consists of the 80286-compatible, 82289 Bus Arbiter and 82288 Bus Controller. The 82289, along with the bus arbiters of other processing subsystems, coordinates control of the MULTIBUS I; the 82288 provides the control signals to perform MULTIBUS I accesses. Communication between the Intel386 DX microprocessor and these devices is accomplished through PLDs that are programmed to perform all necessary signal translation and generation. Latching and buffering of the data and address buses is performed by TTL logic.

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

- A0/A1 generator—Generates the lower address bits from Intel386 DX microprocessor BE0#–BE3# outputs
- Address decoder—Determines whether the bus cycle requires a MULTIBUS I access
- MULTIBUS I address latches—Connect directly to Intel386 DX microprocessor address pins A23–A2 and the outputs of the A0/A1 generator
- MULTIBUS I data latch/transceivers—Connect directly to Intel386 DX microprocessor data pins D15–D0
- S0#/S1# generator—Translates Intel386 DX microprocessor outputs into the S0# and S1# signals
- Wait-state generator—Controls the length of the Intel386 DX microprocessor bus cycle through the READY# signal
- 82288 Bus Controller—Generates the MULTIBUS I command signals
- 82289 Bus Arbiter—Arbitrates contention for bus control between the Intel386 DX microprocessor and other MULTIBUS I masters

These elements of the 80286-compatible interface are described in detail in Chapter 8. The block diagram in Figure 9-1 does not include the Intel386 DX microprocessor local bus interface and local resources. In a complete system, some logic (for example, the address decoder) is common to both MULTIBUS I and local bus interfaces. The following discussion includes only the logic necessary for the MULTIBUS I interface.

9.2.1 Address Latches and Data Transceivers

MULTIBUS I allows up to 24 address lines and 16 data lines. In this example, the MULTIBUS addresses are located in a 256-kilobyte range between F00000H and F3FFFFH, so that all 24 address lines are used. The 16 data lines correspond to the lower half of the Intel386 DX microprocessor data bus.

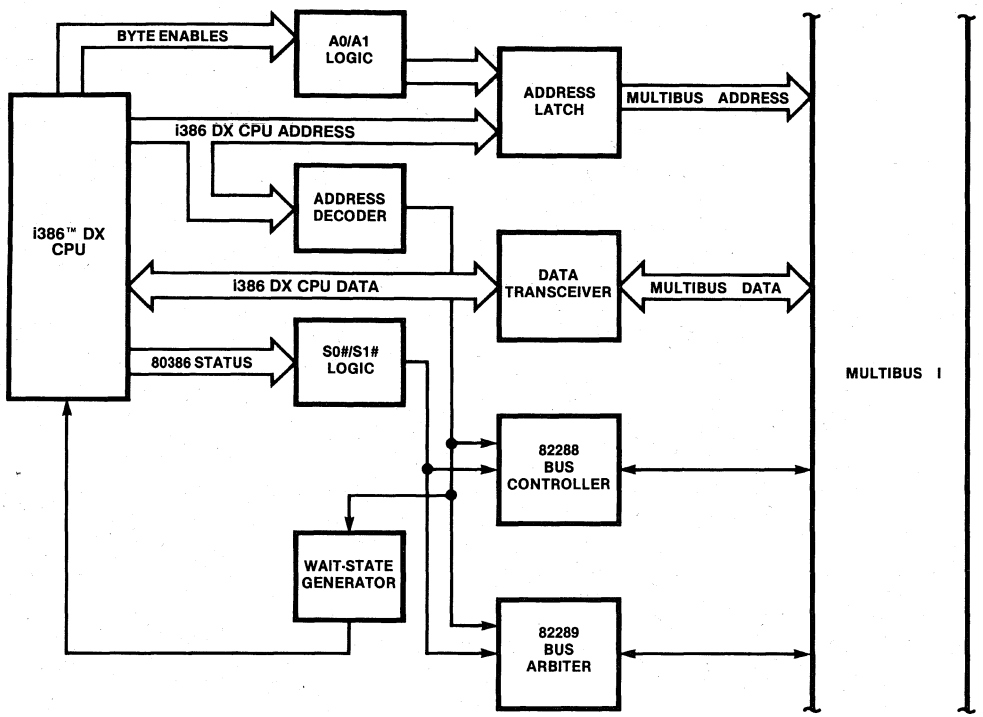


Figure 9-1. Intel386™ DX Microprocessor/MULTIBUS I Interface

Inverting address latches convert the Intel386 DX microprocessor address outputs to the active-low MULTIBUS I address bits. MULTIBUS I address bits are numbered in hexadecimal so that A23–A0 on the Intel386 DX microprocessor bus become ADR17#–ADR0# on the MULTIBUS I (as shown in Figure 9-4). The BHE# signal is latched to provide the MULTIBUS I BHEN# signal.

MULTIBUS I requires address outputs to be valid for at least 50 nanoseconds after the MULTIBUS I command goes inactive; therefore, the address on all bus cycles is latched. The Address Enable (AEN#) output of the 82289 Bus Arbiter, which goes active when the 82289 has control of the MULTIBUS I, is an output enable for the MULTIBUS I latches. The ALE# output of the 82288 latches the Intel386 DX microprocessor address for the MULTIBUS I, as shown in Figure 9-2.

Inverting latch/transceivers are needed to provide active-low MULTIBUS I data bits. MULTIBUS I data bits are numbered in hexadecimal, so D15–D0 convert to DATF#–DAT0#. Data is latched only on write cycles. For MULTIBUS I write cycles, the 82288 ALE#, DEN, and DT/R# inputs can control the address latches and data latch/ transceivers. For MULTIBUS I read cycles, the local bus RD# signal can control the latch/ transceivers. If DEN were used, data contention on the Intel386 DX microprocessor local bus would result when a MULTIBUS I read cycle immediately followed a local write cycle.

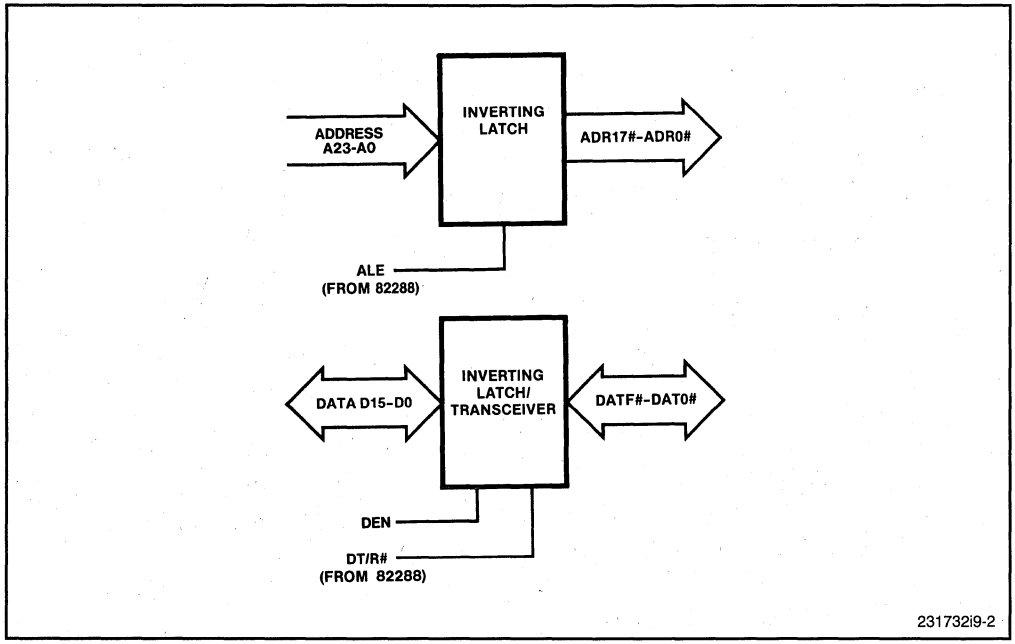


Figure 9-2. MULTIBUS I Address Latches and Data Transceivers

9.2.2 Address Decoder

A MULTIBUS I system typically has both shared and local memory. I/O devices can also be located either on MULTIBUS I or a local bus. Therefore, the address space of the Intel386 DX microprocessor must be allocated between MULTIBUS I and the local bus, and address decoding logic must be used to select one bus or the other.

The following two signals are needed for MULTIBUS I selection:

- Bus Size 16 (BS16#) must be returned active to the Intel386 DX microprocessor to ensure a 16-bit bus cycle. Additional terms for other devices requiring a 16-bit bus can be added to the BS16# PLD equation.
- MULTIBUS Enable (MBEN) selects the 82288 Bus Controller and the 82289 Bus Arbiter on the MULTIBUS I interface. Other outputs of the decoder PLD are programmed to select memory and I/O devices on the local bus.

The decoding of addresses to select either the local bus or the MULTIBUS I is straight forward. In the following example, the system uses the first 64 megabytes of the Intel386 DX microprocessor memory address space, requiring 26 address lines. The MULTIBUS I memory is allocated to the addresses from F00000H to F3FFFFH. The same PLD equation generates the two PLD outputs BS16# and MBEN:

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

I/O resources residing on MULTIBUS I can be memory-mapped into the memory space of the Intel386 DX microprocessor or I/O-mapped into the I/O address space independent of the physical location of the devices on MULTIBUS I. The addresses of memory-mapped I/O devices must be decoded to generate I/O read or I/O write commands for memory references that fall within the I/O-mapped regions of the memory space. This technique is discussed in Chapter 8 along with the tradeoffs between memory-mapped I/O and I/O-mapped I/O.

9.2.3 Wait-State Generator

The wait-state generator controls the READY# input of the Intel386 DX microprocessor. For local bus cycles, the wait-state generator produces signal outputs that correspond to each wait state of the Intel386 DX microprocessor bus cycle, and the PLD READY# output uses these signals to set READY# active after the required number of wait states. Two of the wait-state signals, WS1 and WS2, are also used to generate S0# and S1#.

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

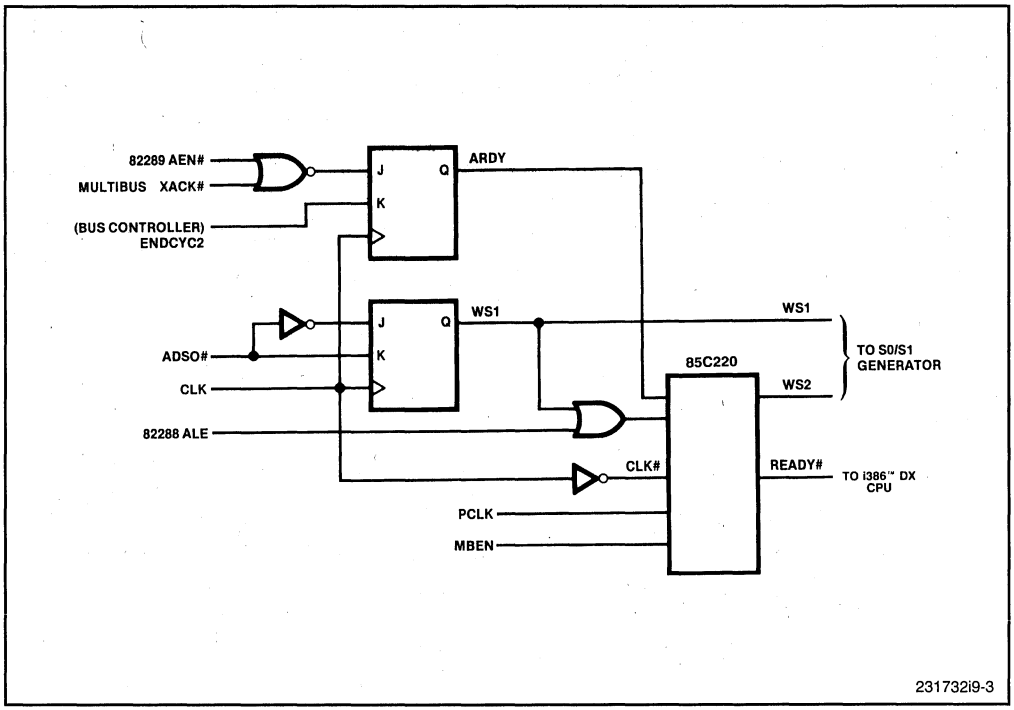


Figure 9-3. Wait-State Generator Logic

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

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

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

RDYPLSE := ARDY * MBEN * PCLK

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

9.2.4 Bus Controller and Bus Arbiter

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

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

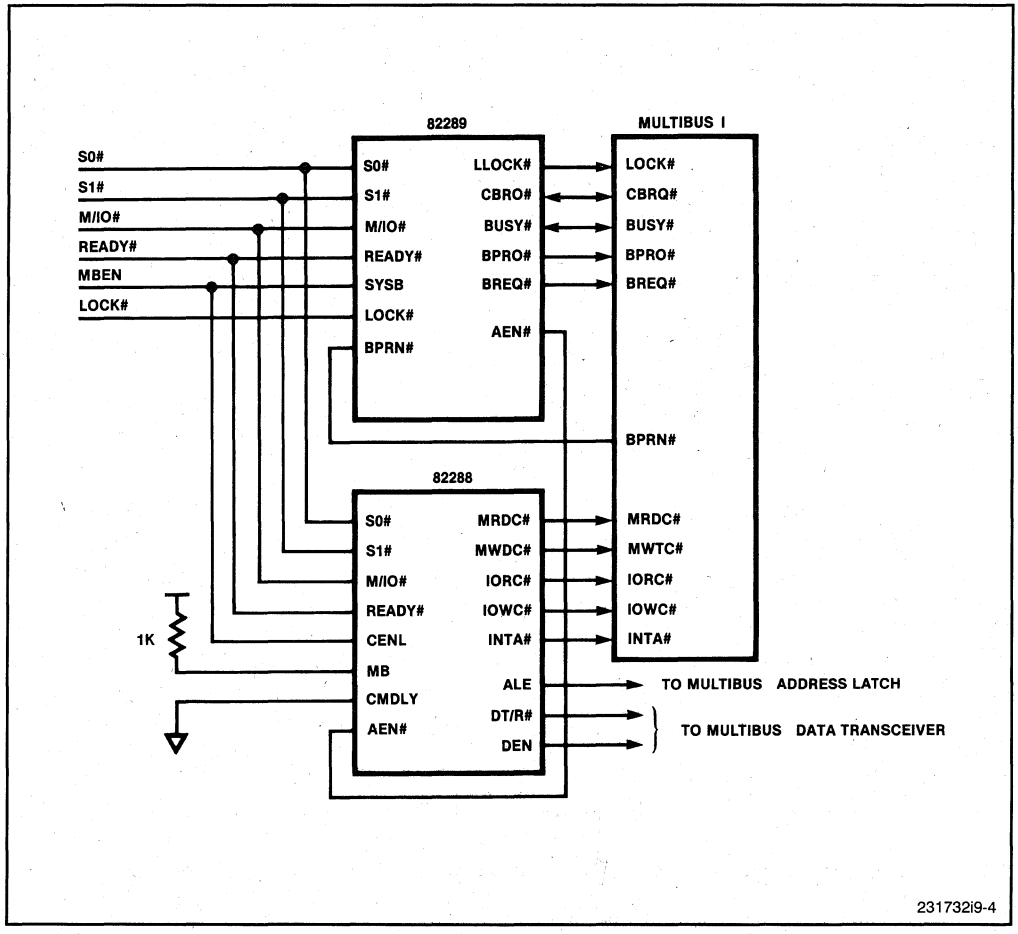


Figure 9-4. MULTIBUS Arbiter and Bus Controller

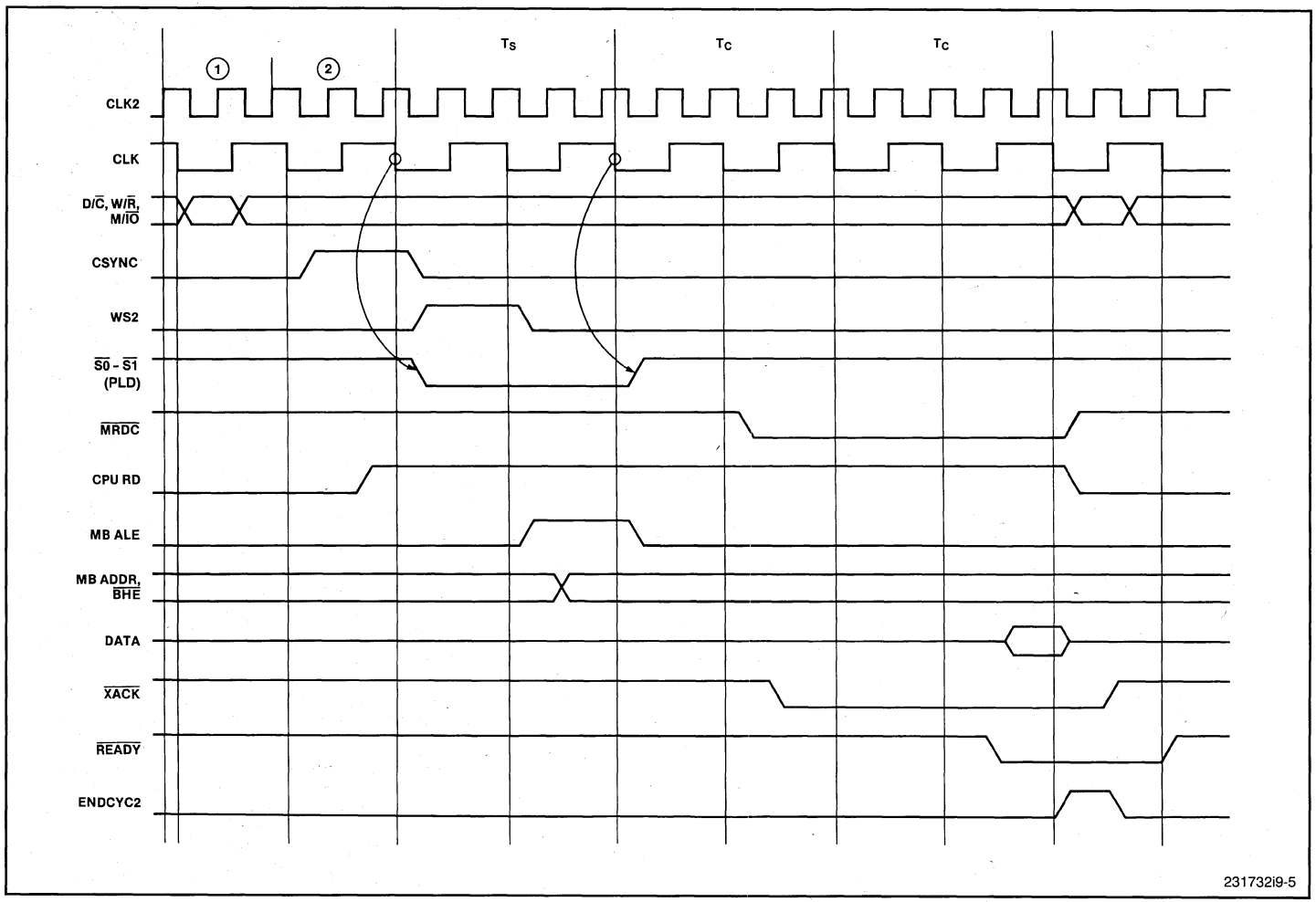


Figure 9-5. MULTIBUS I Read Cycle Timing

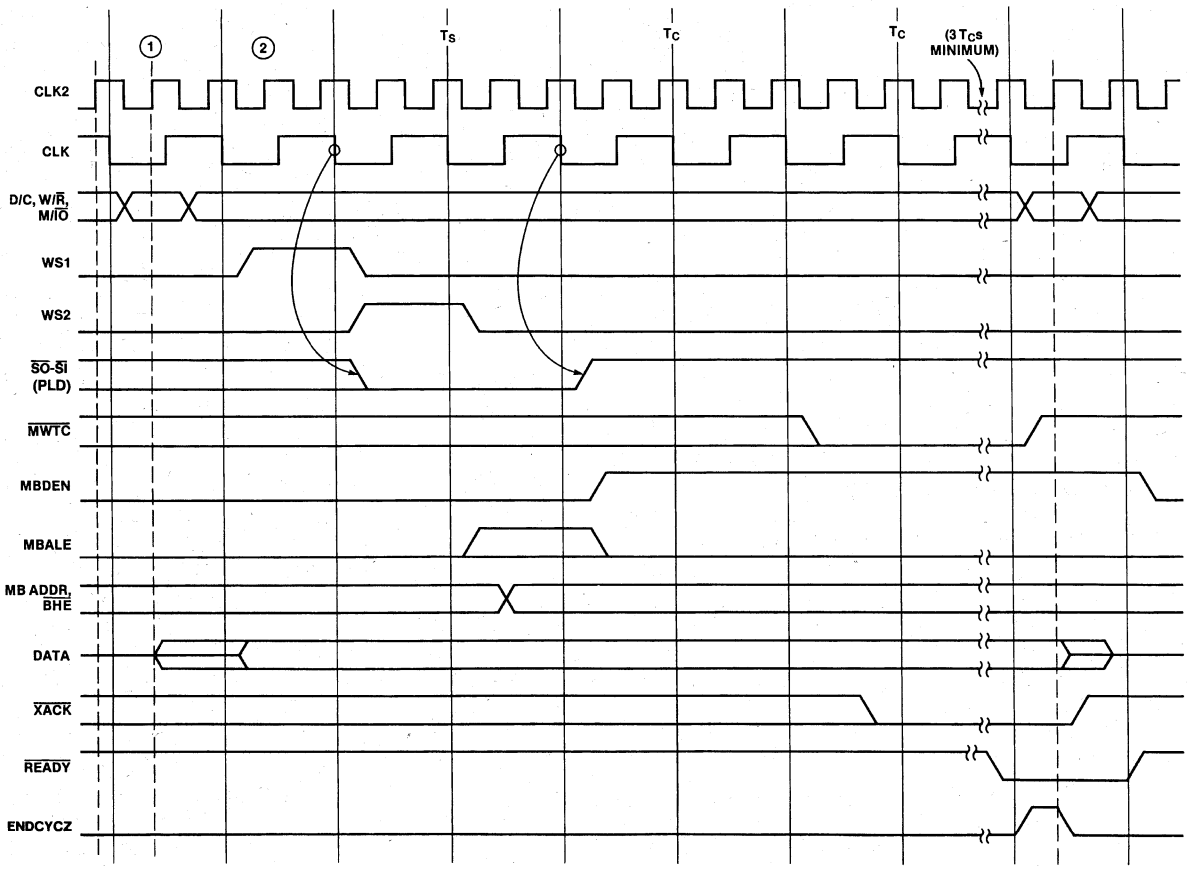


Figure 9-6. MULTIBUS I Write Cycle Timing

9.3 TIMING ANALYSIS OF MULTIBUS I INTERFACE

The timing specifications for the MULTIBUS I are explained in the *MULTIBUS® I Specification*, Order Number 9800683. Table 9-1 lists the MULTIBUS I parameters that relate to the Intel386 DX microprocessor system. These calculations are based on the assumption that 74ALS580 latches and 74F544 transceivers are used for the MULTIBUS I address and data interface.

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

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

9.4 82289 BUS ARBITER

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

Table 9-1. MULTIBUS I Timing Parameters

Timing Parameter	MULTIBUS Specification	Intel386™ DX Microprocessor System Timing
tAS Address setup before command active	50 ns minimum	125 ns (2 CLK cycles) − 20 ns (ALE max delay) − 22 ns (74ALS580 max. delay) + 3 ns (Command min. delay) 86 ns min.
tDS Write data setup before command active	50 ns minimum	125 ns (2 CLK cycles) − 30 ns (DEN max. delay) − 12 ns (74F544 max. delay) + 3 ns (Command min. delay) 86 ns min.
tAH Address hold after command inactive	50 ns minimum	187.5ns (3 CLK cycles) − 25 ns (Command inactive max. delay) + 3 ns (ALE max. delay) 165.5ns min.

Each processing subsystem contains its own 82289 Bus Arbiter. The Bus Arbiter directs its processor onto the bus and allows higher and lower priority bus masters to access the bus. Once the bus arbiter gains control of MULTIBUS I, the Intel386 DX microprocessor can access system resources. The bus arbiter handles bus contention in a manner that is transparent to the Intel386 DX microprocessor.

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

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

9.4.1 Priority Resolution

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

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

The parallel priority technique requires external logic to recognize the BPRN# inputs from all bus arbiters and return the BPRO# signal active to the requesting bus arbiter that has the highest priority. The number of bus arbiters accommodated with this technique depends on the complexity of the decoding logic.

Priority resolution logic need not be included in the design of a single processing subsystem with a MULTIBUS I interface. The bus arbiter takes control of MULTIBUS I when the BPRN# signal goes active and relinquishes control when BPRN# goes inactive. As long as external logic exists to control the BPRN# inputs of all bus arbiters, a subsystem can be designed independent of the priority resolution circuit.

9.4.2 82289 Operating Modes

Following a MULTIBUS I cycle, the controlling bus arbiter can either retain bus control or release control so that another bus master can access the bus. Three modes for relinquishing bus control are as follows:

- Mode 1—The bus arbiter releases the bus at the end of each cycle.
- Mode 2—The bus arbiter retains control of the bus until another bus master (of any priority) requests control.
- Mode 3—The bus arbiter retains control of the bus until a higher priority bus master requests control.

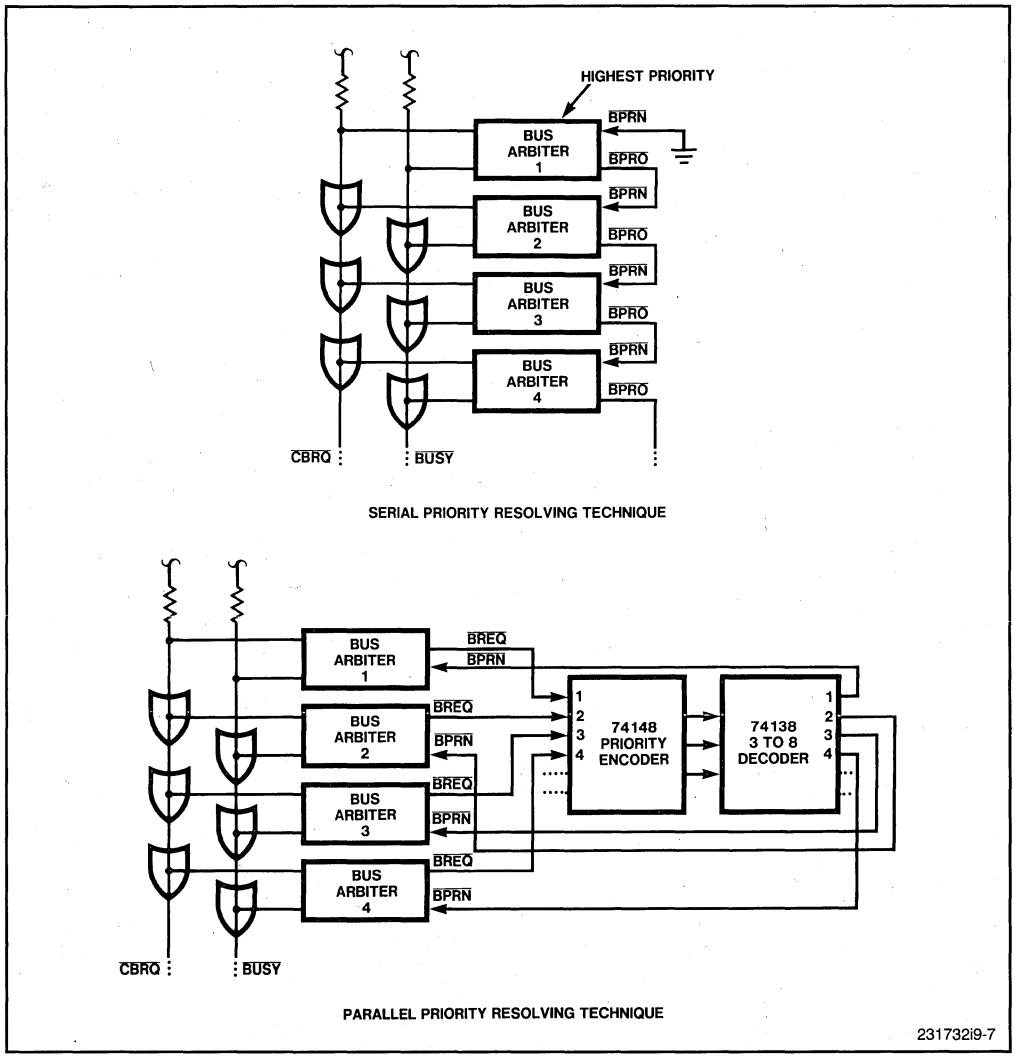


Figure 9-7. Bus Priority Resolution

In addition, the bus arbiter can switch between modes 2 and 3, based on the type of bus cycle.

Figure 9-8 shows the strapping configurations required to implement each of these four techniques.

The operating mode of one bus arbiter affects the throughput of both the individual subsystem as well as other subsystems on MULTIBUS I. This is because the delay required to transfer MULTIBUS I control from one bus arbiter to another affects all

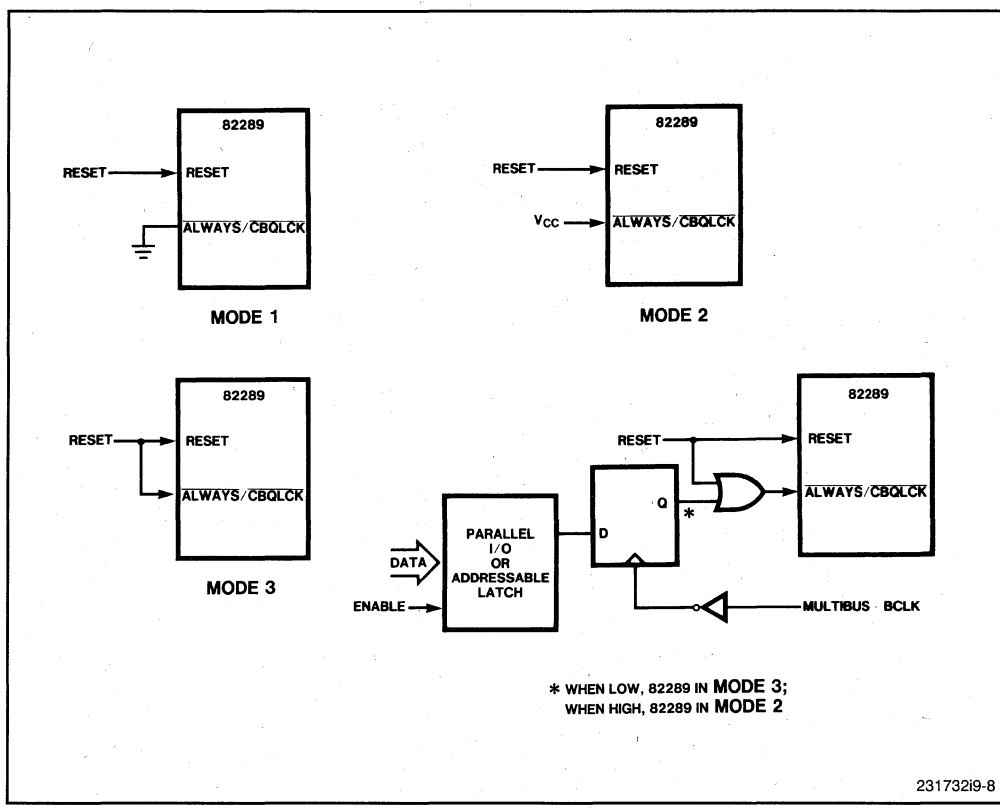


Figure 9-8. Operating Mode Configurations

subsystems waiting to use MULTIBUS I. Therefore, the most efficient operating mode depends on how often a subsystem accesses MULTIBUS I and how this frequency compares to that of the other subsystems.

- Mode 1 is adequate for a subsystem that needs MULTIBUS I access only occasionally. By releasing MULTIBUS I after each bus cycle, the subsystem minimizes its impact on other subsystems that use MULTIBUS I.

- Mode 2 is suited for a subsystem that is one of several subsystems that are all equally likely to require MULTIBUS I. The performance decrease caused by the delay necessary to take control of MULTIBUS I is distributed evenly to all subsystems.

- Mode 3 should be used for a subsystem that uses MULTIBUS I frequently. The delay required for taking control of MULTIBUS I and the consequent performance decrease is shifted to subsystems that use MULTIBUS I less often.

- Switching between modes 2 and 3 is useful if the subsystem demand for MULTIBUS I is unknown or variable.

9.4.3 MULTIBUS I Locked Cycles

Locked bus cycles for the local bus are described in Chapter 3. In locked bus cycles, the Intel386 DX microprocessor asserts the LOCK# signal to prevent another bus master from intervening between two bus cycles. In the same manner, an Intel386 DX microprocessor processing subsystem can assert the LLOCK# output of its bus arbiter to prevent other subsystems from gaining control of MULTIBUS I. A locked cycle overrides the normal operating mode of the bus arbiter (one of the four modes mentioned in Section 9.4.2.

Locked MULTIBUS I cycles are typically used to implement software semaphores (described in Section 3.5) for critical code sections or critical real-time events. Locked cycles can also be used for high-performance transfers within one instruction.

The Intel386 DX microprocessor initiates a locked MULTIBUS I cycle by asserting its LOCK# output to the 82289 bus arbiter. The bus arbiter outputs its LLOCK# signal to the MULTIBUS I LOCK# status line and holds LLOCK# active until the LOCK# signal from the Intel386 DX microprocessor goes inactive. The LLOCK# signal from the bus arbiter must be connected to the MULTIBUS I LOCK# status line through a tristate driver controlled by the AEN# output of the bus arbiter.

9.5 OTHER MULTIBUS I DESIGN CONSIDERATIONS

Additional design considerations are presented in this section. These considerations include provisions for interrupt handling, 8-bit transfers, timeout protection, and power failure handling on MULTIBUS I.

9.5.1 Interrupt-Acknowledge on MULTIBUS I

When an interrupt is received by the Intel386 DX microprocessor, the Intel386 DX microprocessor generates an interrupt-acknowledge cycle (described in Chapter 3) to fetch an 8-bit interrupt vector from the 8259A Programmable Interrupt Controller. The 8259A can be located on either MULTIBUS I or a local bus.

Multiple 8259As can be cascaded (one master and up to eight slaves) to process up to 64 interrupts. Three configurations are possible for cascaded interrupt controllers:

- All of the interrupt controllers for one Intel386 DX microprocessor reside on the local bus of that processor, and all interrupt-acknowledge cycles are directed to the local bus.
- All slave interrupt controllers (those that connect directly to interrupting devices) reside on MULTIBUS I. The master interrupt controller may reside on either the local bus or MULTIBUS I. In this case, all interrupt-acknowledge cycles are directed to MULTIBUS I.

- Some slave interrupt controllers reside on local buses, and other slave interrupt controllers reside on MULTIBUS I. In this case, the appropriate bus for the interrupt-acknowledge cycle depends on the cascade address generated by the master interrupt controller.

In the first two configurations, no decoding is needed because all interrupt acknowledge cycles are directed to one bus. However, if a system contains a master interrupt controller residing on a local bus and at least one slave interrupt controller residing on MULTIBUS I, address decoding must select the bus for each interrupt-acknowledge cycle.

The interrupt-acknowledge cycle must be considered in the design of this decoding logic. The Intel386 DX microprocessor responds to an active INTR input by performing two bus cycles. During the first cycle, the master interrupt controller determines which, if any, of its slave controllers should return the interrupt vector and drive sits cascade address pins (CAS0#, CAS1#, CAS2#) to select that slave controller. During the second cycle, the Intel386 DX microprocessor reads an 8-bit vector from the selected interrupt controller and uses this vector to service the interrupt.

In a system that has slave controllers residing on MULTIBUS I, the circuit shown in Figure 9-9 can be used to decode the three cascade address pins from the master controller to select either MULTIBUS I or the local bus for the interrupt-acknowledge cycle. If MULTIBUS I is selected, the 82289 Bus Arbiter is enabled. The 82289 in turn requests control of MULTIBUS I and enables the address and data transceivers when the request is granted.

The bus-select signal must become valid for the second interrupt-acknowledge cycle. The master controller's cascade address outputs become valid within 565 nanoseconds after the INTA# output from the bus control logic goes active. Bus-select decoding requires 30 nanoseconds, for a total of 595 nanoseconds from INTA# to bus-select valid. The four idle bus cycles that the Intel386 DX microprocessor automatically inserts between the two interrupt-acknowledge cycles provides some of this time. The wait-state generator must add wait states to the first interrupt-acknowledge cycle to provide the rest of the time needed for the bus-select signal to become valid.

The cascade address outputs are gated onto A8, A9, and A10 of the address bus through three-state drivers during the second interrupt-acknowledge cycle. Bus control logic must generate a Master Cascade Enable (MCE) signal to enable these drivers. This signal must remain valid long enough for the cascade address to be captured in MULTIBUS I address latches; however it must be de-asserted before the Intel386 DX microprocessor drives the address bus.

9.5.2 Byte Swapping during MULTIBUS I Byte Transfers

The MULTIBUS I standard specifies that all byte transfers must be performed on the lower eight data lines (MULTIBUS I DAT0#–DAT7#), regardless of the address of the data. An Intel386 DX microprocessor subsystem must swap data from eight of its upper 24 data lines (D8–D15, D16–D23, or D24–D31) to its lower eight data lines (D0–D7) before transferring data to MULTIBUS I, and swap data from its lower data lines to the

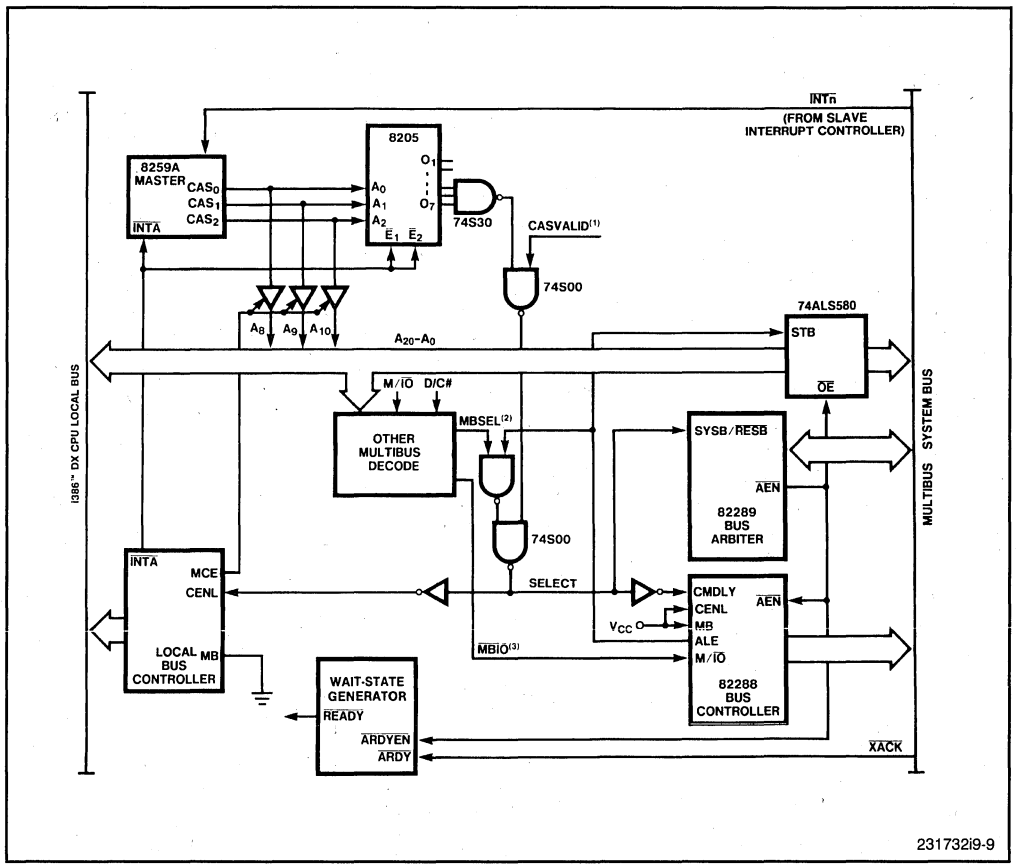


Figure 9-9. Bus-Select Logic for Interrupt Acknowledge

appropriate upper data lines when reading a byte from MULTIBUS I. This byte-swapping requirement maintains compatibility between 8-bit, 16-bit, and 32-bit systems sharing the same MULTIBUS I.

The BS16# signal is generated and returned to the Intel386 DX microprocessor for all MULTIBUS I cycles. The Intel386 DX microprocessor automatically swaps data between the lower half (D15–D0) and the upper half (D31–D16) of its data bus and adds an extra bus cycle as necessary to complete the data transfer. Therefore, only the logic to swap data from D15–D8 to D7–D0 is needed to meet the byte-swapping requirement of MULTIBUS I.

Figure 9-10 illustrates a circuit that performs the byte-swapping function. The Output Enable (OE#) inputs of the data latch/transceivers are conditioned by the states of the BHE# and A0 outputs of the address decoder.

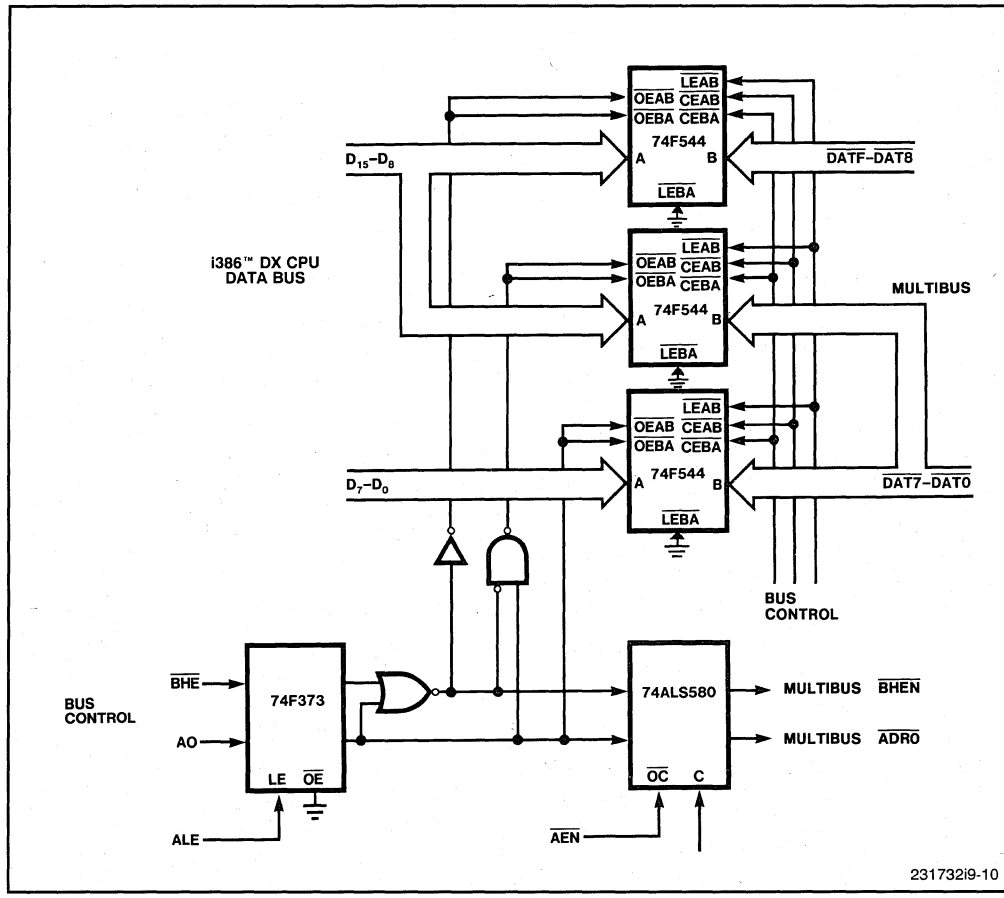


Figure 9-10. Byte-Swapping Logic

9.5.3 Bus Timeout Function for MULTIBUS I Accesses

The MULTIBUS I XACK# signal terminates an Intel386 DX microprocessor bus cycle by driving the wait-state generator logic. However, if the Intel386 DX microprocessor addresses a nonexistent device on MULTIBUS I, the XACK# signal is never generated. Without a bus-timeout protection circuit, the Intel386 DX microprocessor waits indefinitely for an active READY# signal and prevents other processors from using MULTIBUS I.

Figure 9-11 shows an implementation of a bus-timeout circuit that ensures that all MULTIBUS I cycles eventually end. The ALE# output of the bus controller activates a one-shot that outputs a 1-millisecond pulse. The rising edge of the pulse activates the TIMEOUT# signal if READY# does not go active within 1 millisecond to clear the TIMEOUT# flip-flop. The TIMEOUT# signal is input to the wait-state generator logic to activate the READY# signal. When READY# goes active, it is returned to clear the TIMEOUT# signal.

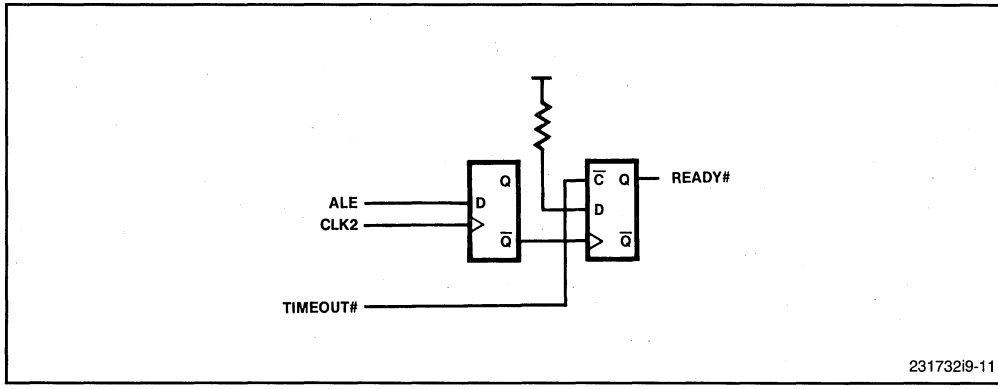


Figure 9-11. Bus-Timeout Protection Circuit

9.5.4 MULTIBUS I Power Failure Handling

The MULTIBUS I interface includes a Power Fail Interrupt PFIN signal to signal an impending system power failure. Typically, PFIN# is connected to the non-maskable interrupt (NMI) request input of each Intel386 DX microprocessor. The NMI service routine can direct the Intel386 DX microprocessor to save its environment immediately, before falling voltages and the MULTIBUS I Memory Protect (MPRO#) signal prevent any further memory activity. In systems with memory backup power or nonvolatile memory, the saved environment can be recovered on power up.

The power-up sequence of the Intel386 DX microprocessor can check the state of the MULTIBUS I Power Fail Sense Latch (PFSN#) to see if a previous power failure has occurred. If this signal is active (low), the Intel386 DX microprocessor can branch to a power-up routine that resets the latch using the Power Fail Sense Reset signal (PFSR#), restores the previous Intel386 DX microprocessor environment, and resumes execution.

Further guidelines for designing Intel386 DX microprocessor systems with power failure features are contained in the Intel *MULTIBUS® I Specification.*

9.6 iLBX™ BUS EXPANSION

The iLBX (Local Bus Expansion) is a high-performance bus interface standard that permits the modular expansion of an Intel386 DX microprocessor-based system. An iLBX interface links the Intel386 DX microprocessor system board with additional boards containing memory, I/O subsystems, and other peripheral devices or bus masters. Any board that conforms to the iLBX standard can be added to the system as the user's needs dictate. For a 16-MHz Intel386 DX microprocessor-based system, a typical iLBX access cycle requires six wait states.

The *iLBX™ Bus Specification* describes the iLBX Local Bus Expansion standard in detail.

The iLBX bus interface requires the generation of A1, A0, and BHE# from the Intel386 DX microprocessor BE3#–BE0# outputs. The iLBX connector contains 24 address bits (AB23–AB0) and 16 data bits (DB15–DB0), which are taken from the buffered address lines (A23–A0), and data lines (D15–D0) of the Intel386 DX microprocessor local bus. BHE# is inverted and buffered to provide the Byte High Enable (BHEN) signal.

The Read/Write (R/W#), Data Strobe (DSTB#), and Address Strobe (ASTB#) controls are generated from local bus control signals using the logic shown in Figure 9-12. R/W# is a delayed, inverted version of the W/R# output of the Intel386 DX microprocessor. DSTB# goes active when either RD# or WR# from the local bus control goes active. ASTB# and DSTB# are delayed to allow adequate setup time for BHEN. In this example, the WS2 signal, which is active during the third CLK cycle of the Intel386 DX microprocessor bus cycle, provides the delay.

A chip-select output of address decoding logic goes active for accesses to the memory and I/O locations allocated to the iLBX bus and selects the iLBX address and data buffers. Command signals from the local bus control logic enable the outputs of the iLBX transceivers.

When an iLBX cycle is complete, the Acknowledge (ACK#) signal is returned over the iLBX bus. This signal must be synchronized and incorporated into the wait-state generator logic to provide the READY# signal.

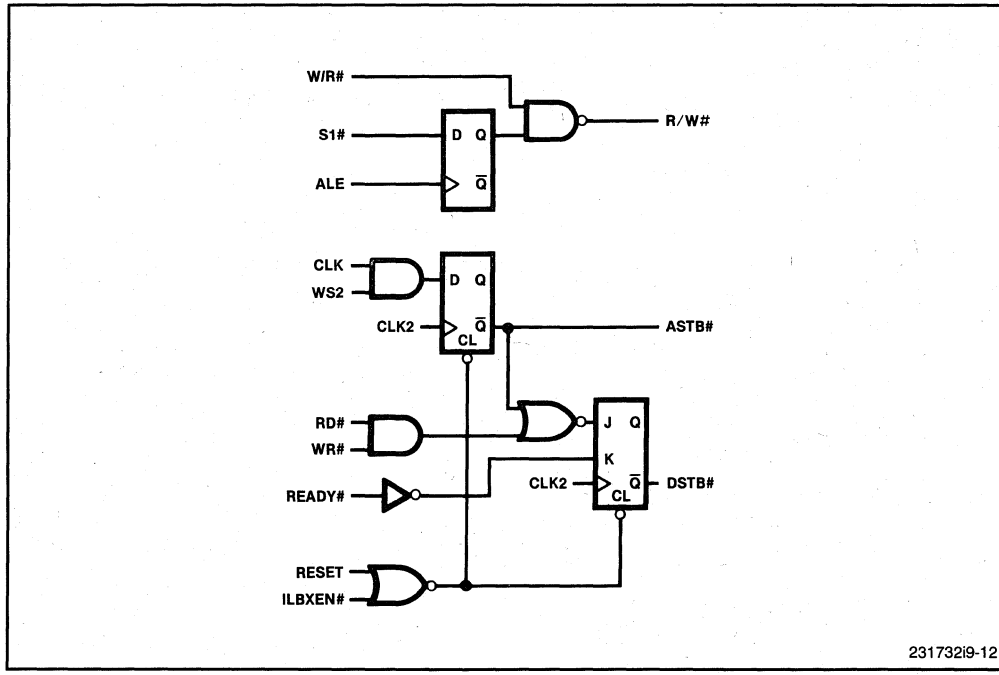


Figure 9-12. iLBX™ Signal Generation

9.7 DUAL-PORT RAM WITH MULTIBUS I

A dual-port RAM is a memory subsystem that can be accessed by both the Intel386 DX microprocessor, through its local bus, and other processing subsystems, through the MULTIBUS I system bus. Dual-port RAM offers some of the advantages of both local resources and system resources. It is an effective solution when using only local memory or only system memory would decrease system cost and/or performance significantly.

The Intel386 DX microprocessor accesses dual-port RAM through its high-speed local bus, leaving MULTIBUS I free for other system operations. Other processing subsystems can pass data to and from the Intel386 DX microprocessor through the dual-port RAM using MULTIBUS I.

If necessary, dual-port RAM can be mapped to reserve address ranges for the exclusive use of the Intel386 DX microprocessor. The Intel386 DX microprocessor and the other processing subsystems need not use the same address mapping for dual-port RAM.

The disadvantage of dual-port RAM is that its design is more complex than that of either local or system memory. Dual-port RAM requires arbitration logic to ensure that only one of the two buses gains access at one time.

9.7.1 Avoiding Deadlock with Dual-Port RAM

The MULTIBUS-LOCK# signal and the Intel386 DX microprocessor LOCK# signal mediate contention when both the Intel386 DX microprocessor and a MULTIBUS I device attempt to access dual-port RAM. However, locked cycles to dual-port RAM can potentially result in deadlock. Deadlock arises when the Intel386 DX microprocessor performs locked cycles to ensure back-to-back accesses to dual-port RAM and MULTIBUS I.

Suppose the Intel386 DX microprocessor locks an access to dual-port RAM followed by a MULTIBUS access, to ensure that the accesses are performed back-to-back. (This could happen only in protected mode during interrupt processing when the IDT is in the dual-port RAM and the target descriptor is in MULTIBUS RAM.) At the same time the Intel386 DX microprocessor performs the first locked cycle, another device gains control of MULTIBUS I for the purpose of accessing dual-port RAM. The Intel386 DX microprocessor cannot gain control of MULTIBUS I to complete the locked operation, and the other device cannot relinquish control of MULTIBUS I because it cannot complete its access to dual-port RAM. Each device therefore enters an interminable wait state.

Two approaches can be used to avoid deadlock:

- Requiring software to be free of locked accesses to dual-port RAM.
- Designing hardware to negate the LOCK# signal for transfers between dual-port RAM and MULTIBUS I. If this approach is used, software writers must be informed that such transfers will not be locked even though software dictates locked cycles.

10 MULTIBUS II and Intel386 DX Microprocessor

CHAPTER 10
MULTIBUS II AND Intel386 DX MICROPROCESSOR

Standard bus interfaces guarantee compatibility between existing and newly developed systems. This compatibility safeguards a user's hardware investment against obsolescence even in the face of rapidly advancing technology. The MULTIBUS I standard interface has proven its value in providing flexibility for the expansion of existing systems and the integration of new designs. The MULTIBUS II standard interface extends Intel's Open Systems design strategy into the world of 32-bit microprocessing systems.

10.1 MULTIBUS II STANDARD

The MULTIBUS II standard is a processor-independent bus architecture that features a 32-bit parallel system bus with a maximum throughput of 40 megabytes per second, high-speed local bus access to off-board memory, a low-cost serial system bus, and full multiprocessing support. MULTIBUS II achieves these features through five specialized Intel buses:

- Parallel System Bus (iPSB)
- Local Bus Extension (iLBX II)
- Serial System Bus (iSSB)
- Multi-channel DMA I/O Bus
- System Expansion I/O Bus (iSBX™)

The DMA I/O Bus and the iSBX are carried over directly from MULTIBUS I architecture. See the *MULTIBUS® I Architectural Specification* for a full description of these buses. The multiple bus structure provides the following important advantages over a single, generalized bus:

- Each bus is optimized for a specific function.
- The buses perform operations in parallel.
- Buses that are not needed for a particular system can be omitted, avoiding unnecessary costs.

10.2 PARALLEL SYSTEM BUS (iPSB)

The Parallel System Bus (iPSB) is optimized for interprocessor data transfer and communication. Its burst transfer capability provides a maximum sustained bandwidth of 40 megabytes per second for high-performance data transfers.

The iPSB supports four address spaces per bus agent (a board that encompasses a functional subsystem). The conventional I/O and memory address spaces are included, plus two other address spaces that support advanced functions:

- A 255 address message space supports message passing. Typically, a microprocessor performs interprocessor communications inefficiently. Message passing allows two bus agents to exchange a block of data at full bus bandwidth without supervision from a microprocessor. An intelligent bus interface capable of message passing shifts the burden of interprocessor communication away from the processor, thus enhancing overall system performance.

- An interconnect space allows geographic addressing, which is the identification of any bus agent (board) by slot number. Every MULTIBUS II system contains a Central Services Module (CSM) that provides system services, such as uniform initialization and bus timeout detection, for all bus agents residing on the iPSB bus. The CSM may use the registers of the interconnect space of each bus agent to configure the agent dynamically. Stake pin jumpers, DIP switches, and other hardware configuration devices can be eliminated.

Because the Intel386 DX microprocessor can access only memory space or I/O space, the message space and interconnect space may be mapped into the memory space or the I/O space. Decoding logic provides chip select signals for the devices implementing the message space and the interconnect space, as well as devices in the memory space and the I/O space.

Three types of bus cycles define activity on the iPSB bus:

- Arbitration Cycle—Determines the next owner of the bus. This cycle consists of a resolution phase, in which competing bus agents determine priority for bus control, and an acquisition phase, in which the agent with the highest priority initiates a transfer cycle.

- Transfer Cycle—Performs a data transfer between the bus owner and another bus agent. This cycle consists of a request phase, in which address control signals are driven, and a reply phase, in which the two agents perform a handshake to synchronize the data transfer. The reply phase is repeated and data transfers continue until the bus owner ends the transfer cycle.

- Exception Cycle—Indicates that an exception (error) has occurred during a transfer cycle. This cycle consists of a signal phase, in which an exception signal from one bus agent causes all other bus agents to terminate any arbitration and transfer cycles in progress, and a recovery phase, in which the exception signals go inactive. A new arbitration cycle can begin on the clock cycle after the recovery phase.

Figure 10-1 shows how the timing of these cycles overlap.

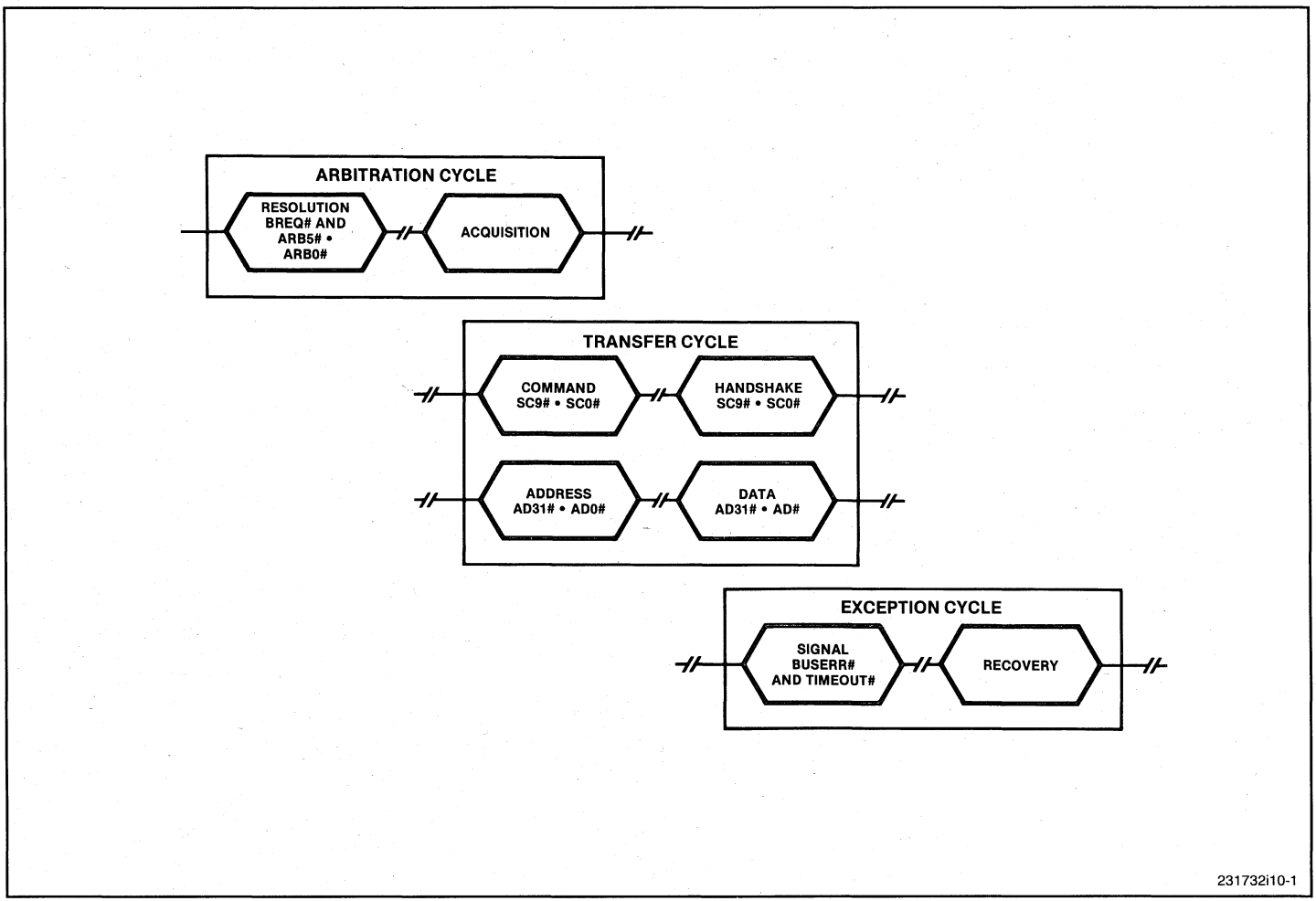


Figure 10-1. iPSB Bus Cycle Timing

10.2.1 iPSB Interface

Each bus agent must provide a means of transferring data between its Intel386 DX microprocessor, its interconnect registers, and the iPSB bus. The location of bus interface logic to meet this requirement is shown in Figure 10-2. A full-featured subsystem may also include provisions for the message passing protocols used by the iPSB bus.

The iPSB interface may be conveniently implemented by a Bus Arbiter/Controller (BAC), a Message Interrupt Controller (MIC), and miscellaneous logic. The BAC coordinates direct interaction with the other devices on the iPSB bus, while the MIC works through the BAC to send and receive interrupt messages. Other logic is needed for address decoding, parity checking, and control signal generation.

The BAC and MIC are implemented in Intel gate arrays. In addition, Intel has developed an advanced CMOS device, the Message Passing Coprocessor (MPC), that integrates the functions of the BAC and the MIC plus parity checking and full message passing (solicited and unsolicited), all in one package. Detailed information on the MPC 82389 is available in the *Microprocessor and Peripheral Handbook*, Order Number 230843.

10.2.1.1 BAC SIGNALS

The BAC provides arbitration and system control logic for the arbitration, transfer, and exception cycles defined by the MULTIBUS II architecture. Through the BAC, the bus agent functions as either a requestor or a replier in a transfer cycle. In all cases, the device requiring iPSB bus access (either the Intel386 DX microprocessor or the MIC) is completely isolated from the iPSB; the BAC provides all direct interaction.

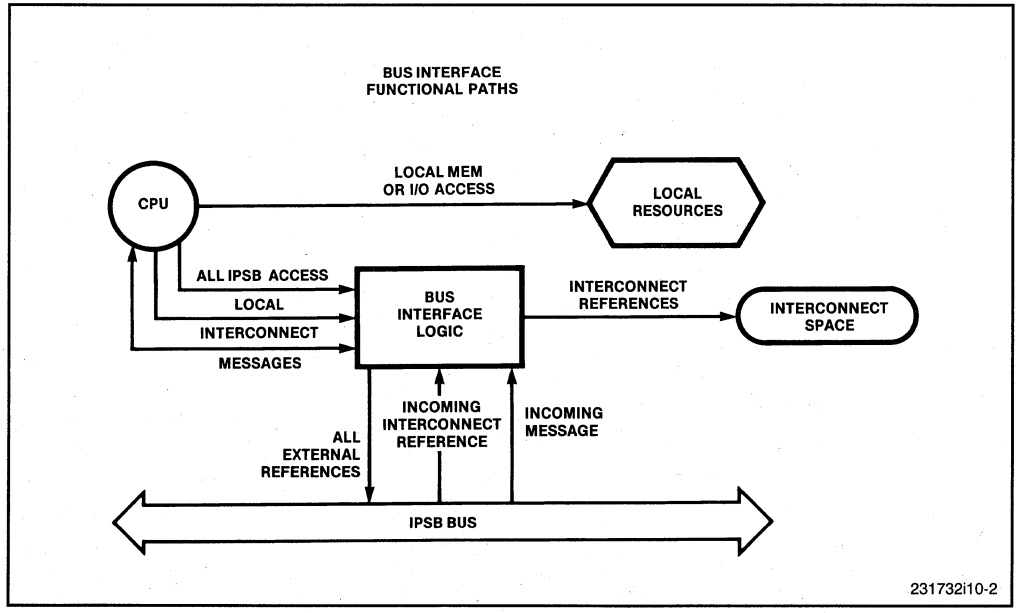


Figure 10-2. iPSB Bus Interface

The BAC signals can be divided into three functional groups:

- iPSB interface
- Local bus interface
- Register interface with the Intel386 DX microprocessor

The iPSB interface signals perform mainly arbitration and system control. Five bidirectional Arbitration signals (ARB5–ARB0) are used during reset to read a cardslot ID and arbitration ID from the CSM, and during arbitration cycles to output the arbitration ID for priority resolution. Bus Request (BREQ#) is a bidirectional signal. Each bus agent asserts BREQ# to request control of the bus and samples BREQ# to determine if other agents are also contending for bus control.

Bus Error (BUSERR#) is a bidirectional signal that a bus agent outputs to all other bus agents when it detects a parity error during a transfer cycle. Bus Timeout (TIMOUT #) is output by the CSM to all bus agents when a bus cycle fails to end within a prescribed time period.

Ten System Control signals (SC9#–SC0#) coordinate transfer cycles. The *MULTIBUS® II Architectural Specification* defines each of these signals. Directional enables (SCOEH and SCOEL) are provided for transceivers to buffer these bidirectional signals. External logic checks byte parity on the multiplexed address and data bus (AD31–AD0) and sets the Parity inputs (PAR3–PAR0) accordingly.

Other iPSB signals are Reset (RST#), Reset-Not-Complete (RSTNC#), and ID Latch (LACHn#, n = slot number). These signals are used only during reset.

Local bus interface signals pertain to the communication between the BAC and the Intel386 DX microprocessor or between the BAC and the MIC. These signals indicate to the BAC when to request bus control and what type of bus cycle to drive when it gains bus control.

Four control signals are necessary for each of the two devices connected to the BAC. The signals that connect to the Intel386 DX microprocessor are REQUESTA, GRANTA, READYA, and SELECTA; those that connect to the MIC are REQUESTB, GRANTB, READYB, and SELECTB.

To request bus control, the Intel386 DX microprocessor or the MIC activates one of the REQUEST signals. The corresponding GRANT signal is returned by the BAC when it has bus control. Data width and address space selections are encoded on the WIDTH1#, WIDTH0#, SPACE1#, and SPACE0# inputs, while WR# dictates either a write cycle or a read cycle. These five inputs translate directly to SC6#–SC2# outputs during the request phase of a transfer cycle. READYA or READYB indicates that WIDTH0#, WIDTH1#, SPACE0#, SPACE1#, and WR# can be read by the BAC to drive the transfer cycle.

LASTINA or LASTINB controls the end-of-cycle signal for burst transfers. The LOCK# input is activated for locked transfers.

The bus agent that receives a transfer cycle from the bus owner must have its BAC enabled by an active SELECT input. Errors detected by the replying agent are encoded by its MIC on the AGERR2–AGERR0 inputs to its BAC so that the BAC can drive the SC7#–SC5# lines accordingly. If an error occurs, the requesting agent notifies the Intel386 DX microprocessor through the EINT signal.

The register interface signals control register operations between the Intel386 DX microprocessor and the BAC. Three 5-bit registers (Arbitration ID, Slot ID, and Error Port) are addressed through RSEL1 and RSEL0. Data is transferred on RIO4–RIO0; the direction of transfer is indicated by RRW.

10.2.1.2 MIC SIGNALS

The MIC coordinates interrupt handling for a bus agent on the iPSB bus. Interrupts are implemented as virtual interrupts in the message space. To send an interrupt message, the Intel386 DX microprocessor writes four bytes to the MIC to indicate the source, destination, and type of message. The MIC then coordinates the message transfer. The MIC of the receiving bus agent reads the 4-byte message and stores it in a 4-deep message queue to be read by the Intel386 DX microprocessor.

The MIC signals are divided into three groups:

- iPSB interface
- Local bus interface
- BAC interface

The iPSB interface consists of the multiplexed address/data bus (AD31#–AD0#). Although the MIC gains access to the iPSB bus through the BAC, the MIC drives the address/data bus directly. As a requesting agent, the MIC drives the address and data at the appropriate times. As a receiving agent, the MIC monitors the address/data bus for its address. When it recognizes its address, the MIC selects its BAC to perform the required handshake and read the message into the message queue. Then, the MIC interrupts the Intel386 DX microprocessor to indicate that the message is pending in the queue. The Intel386 DX microprocessor reads the message and services the interrupt accordingly.

The local bus interface consists of seven register/ports, addressed through A2–A0, through which the MIC and the Intel386 DX microprocessor communicate. Data is transferred over D7–D0, and WR# and RD# determine the direction of transfer. Other signals include the MIC Chip Select (CS#), a WAIT# signal for adding wait states to the Intel386 DX microprocessor cycle, and a Message Interrupt (MINT) to signal an interrupt condition to the Intel386 DX microprocessor.

The BAC interface includes REQUESTB, READYB, SELECTB, and GRANTB. These signals have already been described with the other BAC signals.

While the BAC and the MIC together provide the backbone for an iPSB interface, other logic provides buffering and control to round out the interface. An 8751 Microcontroller coordinates Intel386 DX microprocessor access to the interconnect space. An address

decoder distinguishes between local, interconnect, and iPSB accesses. PLDs control the buffering of signals between the Intel386 DX microprocessor, BAC, MIC, 8751 Microcontroller, and iPSB bus.

10.3 LOCAL BUS EXTENSION (iLBX II)

The iLBX II bus extension is a high-speed execution bus designed for quick access to off-board memory. One iLBX II bus extension can support either two processing subsystems (called the primary requesting agent and the secondary requesting agent) plus four memory subsystems, or a single processing subsystem plus five memory subsystems. A MULTIBUS II system may contain more than one iLBX II bus extension to meet its memory requirements.

The iLBX II bus extension features a 26-bit address bus and a separate 32-bit data bus. Because these paths are separate, the extension allows pipelining of transfer cycles; the request phase of a transfer cycle can overlap the reply phase of the previous cycle.

Other features of the iLBX II bus extension are:

- A unidirectional handshake for fast data transfers
- Mutual exclusion capability to control multiported memory
- Interconnect space (for each bus agent) through which the primary requesting agent initializes and configures all other bus agents.

10.4 SERIAL SYSTEM BUS (iSSB)

The Serial System Bus (iSSB) provides a simple, low-cost alternative to the Parallel System Bus (iPSB) bus. In applications that do not require the high performance of the iPSB bus, the iSSB bus can provide some cost reduction. In systems containing both the iPSB bus and the iSSB bus, the iSSB bus provides an alternate path for interface control, diagnostics, or redundancy.

The iSSB bus can contain up to 32 bus agents distributed over a maximum of 10 meters. Bus control is determined through an access protocol called Carrier Sense Multiple Access with Collision Detection (CSMA/CD). This protocol allows agents to transmit data whenever they are ready. In case of simultaneous transmission by two or more bus agents, the iSSB invokes a deterministic collision resolution algorithm to grant fair access to all agents.

From the application point of view, the error detection capability of the iSSB bus, coupled with an intelligent bus agent interface (able to retransmit) makes the iSSB bus as reliable as the iPSB bus, even though the iSSB bus may be up to 10 meters long.

11 Physical Design and Debugging

CHAPTER 11
PHYSICAL DESIGN AND DEBUGGING

To maximize the performance of high-speed Intel386 DX processor systems, it is recommended that optimum design guidelines be followed. This chapter outlines the basic design issues, ranging from power and ground issues to achieving proper thermal environment for Intel386 DX microprocessor.

11.1 GENERAL DESIGN GUIDELINES

The performance and proper operation of any high-speed system greatly depends upon appropriate physical layout. This section gives an overview of design guidelines for layout which are significant to both higher- and lower-frequency system design implementation.

The ever-increasing improvement of integrated circuit technology has led to an enormous increase in performance. The Intel386 DX microprocessor, with an operating frequency of 33 MHz (CLK2 = 66 MHz) and a corresponding fast edge rate, presents a challenge to the conventional interconnection technologies. This challenge applies especially to system designers who are responsible for providing suitable, high frequency interconnections at the systems level.

At higher frequencies, the interconnections in a circuit behave like transmission lines which degrade the system's overall speed and distort its output waveforms.

In laying out a conventional printed circuit board, there is freedom in defining the length, shape and sequence of interconnections. However with high-speed devices such as the Intel386 DX processor, this task should be carried out with careful planning, evaluation, and testing of the wiring patterns. It is critical to understand the physical properties of transmission lines.

11.2 POWER DISSIPATION AND DISTRIBUTION

The Intel386 DX microprocessor uses fast one-micro CHMOS IV process. The main difference between the previous HMOS microprocessors and the new ones is that power dissipation is primarily capacitive and that there is almost no DC power dissipation. As power dissipation is directly proportional to frequency, accommodating high-speed signals on printed circuit boards and through the interconnections is very critical. The power dissipation of the VLSI device in operation is expressed by the sum of the power dissipation of the circuit elements, which include internal logic gates, I/O buffers and cache RAMs. It is also a function of the operating conditions.

The worst-case power dissipation of any VLSI device is estimated in the following manner:

1. To estimate typical power dissipation for each circuit element:
   P_G : Typical power dissipation for internal logic gates (mW)
   $P_{I/O}$: Typical power dissipation for I/O buffers (mW)

2. To estimate total typical power dissipation:
   $P_T = P_G + P_{I/O}$ (mW) ... (1)
   where P_T is the total typical power dissipation (mW)

3. To estimate the worst case power dissipation:
   $P_d = P_T \times C_v$ (mW) ... (2)
   where P_d is the worst case power dissipation (mW) and C_v is a multiplier that is dependent upon power supply voltage.

Internal logic power dissipation varies with operating frequency and to some extent with wait-states and software. It is directly proportional to the supply voltage. Process variations in manufacturing also affect the internal logic power dissipation, although to a lesser extent than with the NMOS processes.

The I/O buffer power dissipation, which accounts for roughly 10 to 25 percent of the overall power dissipation, varies with the frequency and the supply voltage. It is also affected by the capacitive bus load. The capacitive bus loadings for all output pins is specified in the Intel386 DX processor data sheet. The Intel386 DX processor's output valid delays will increase if these loadings are exceeded. The addressing pattern of the software can affect I/O buffer power dissipation by changing the effective frequency at the address pins. The frequency variations at the data pins tends to be smaller; thus, a varying data pattern should not cause a significant change in the total power dissipation.

To calculate the total power dissipated by the board, the following formulas can be used:

To calculate the maximum statistical power:

$$P_{typical1} + P_{typical2} + \ldots [(P_{max1} - P_{typical1})^2 + (P_{max2} - P_{typical2})^2 + \ldots$$

where $P_{typical1}$ and P_{max1} are the typical and maximum power dissipation of each of the integrated circuits on the board. The Intel386 DX processor should be placed closer to fan or where the airflow is unrestricted.

11.2.1 Power and Ground Planes

Today's high-speed CMOS logic devices are susceptible to the ground noise and the problems that this noise creates in digital system design. This noise is a direct result of the fast switching speed and high drive capability of these devices, which are requisites in

high-performance systems. Logic designers can use techniques designed to minimize this problem. One technique is to reduce capacitance loading on signal lines and provide optimum power and ground planes.

Power and ground lines have inherent inductance and capacitance, which affect the total impedance of the entire system. Higher impedances reduce current and therefore offer reduced power consumption, while low impedances (ground planes) help minimize problems like noise and cross talk. Hence, it is very important for a designer to have a controlled impedance design where high speed signals are involved. The formula for impedance is as follows:

$\text{Impedance} = (L/C)^{1/2}$.

The total characteristic impedance for the power supply can be reduced by adding more lines. For multi-layer boards, power and ground planes must be used in the Intel386 DX microprocessor designs.

The effect of adding more lines to reduce impedance is illustrated in Figure 11-1 which shows that two lines in parallel have half the impedance of a single line.

To reduce impedance even further, more lines should be added. To lower the impedance, an infinite number of lines or a plane should be used. Planes also provide the best distribution of power and ground.

The Intel386 DX microprocessor has 20 power (V_{CC}) and 21 ground (V_{SS}) pins. All power and ground pins must be connected to their respective planes. Ideally, the Intel386 DX microprocessor should be placed at the center of the board to take full advantage of these planes. Although Intel386 DX CPU generally demands less power than the conventional devices, the possibility of power surges is increased due to processors higher operating frequency and its wide address and data buses. Peak-to-peak noise on V_{CC} relative to V_{SS} should be maintained at no more than 400 mV, and preferably to no more than 200 mV.

Although power and ground planes are preferable to power and ground traces, double-layer boards present a need for routing of the power and ground traces.

The inductive effect of a printed-circuit board (PCB) trace can be reduced by bypassing (or decoupling). Careful layout procedures should be observed to minimize inductances. Figure 11-2 shows methods for reducing the inductive effects of PCB traces. The power and ground trace layout has a low series inductance as shown in Figure 11-2. This is because the loop area between the integrated circuits (ICs) and the decoupling capacitors is small and the power and ground traces are closer. This results in lower characteristic impedance, which in turn reduces the line voltage drop.

Another placement technique is called orthogonal arrangement, which requires more area than the previous technique but produces similar results. This arrangement is shown in Figure 11-3. These techniques also reduce the electromagnetic interference (EMI), which will be discussed in Section 11.3.3.1.

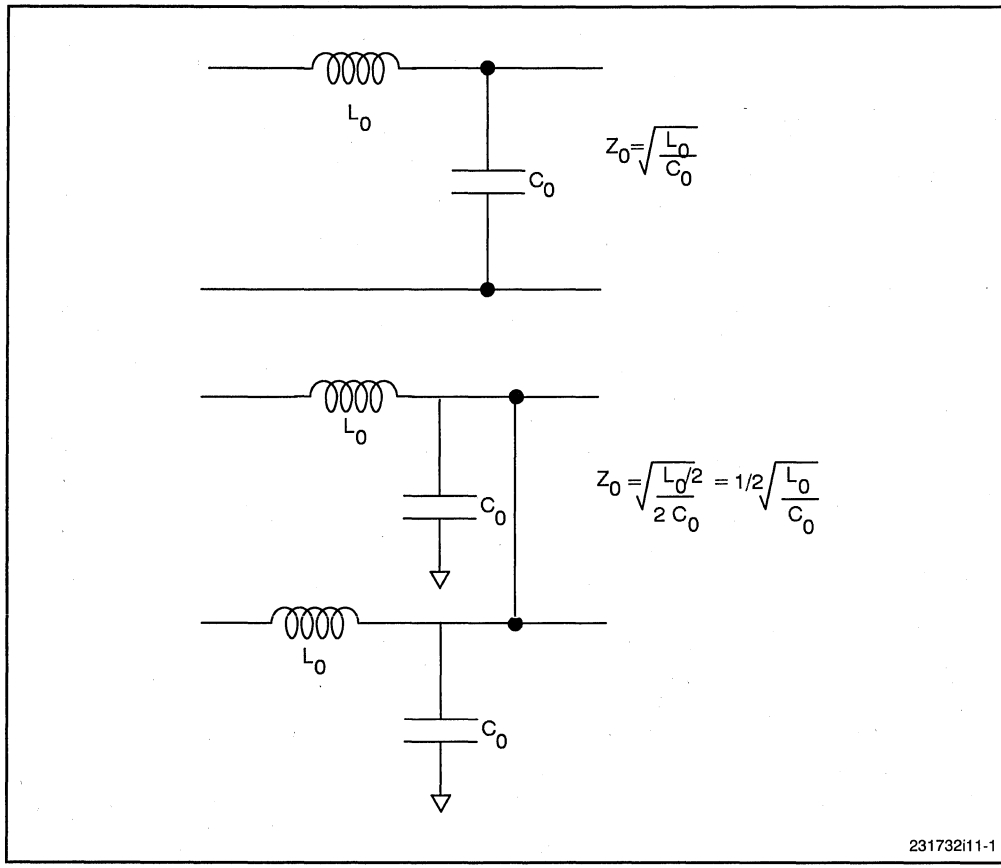


Figure 11-1. Reduction in Impedance

11.3 DECOUPLING CAPACITORS

The advanced, high-speed CMOS logic families available today have much faster edge rates than do the older, slower logic technologies. The switching speeds and drive capability needed to provide high performance systems are also associated with increased noise levels. Some noise levels are inconsequential because they fall within the switching times of the other devices. However, the switching activity of one device can propagate to other devices through the power supply. For example, in the TTL NAND gate shown in Figure 11-4, both the Q3 and the Q4 transistors are ON for a short time while output is switching. This increased loading causes a negative spike on V_{CC} and a positive spike on V_{SS}.

In synchronous systems where several gates switch simultaneously, the result is a significant amount of noise on the power and ground lines. This noise can be removed by decoupling the power supply. First, it is necessary to match the power supply's impedance to that of the individual components. Any power supply presents a low source impedance to other circuits, whether they are individual components on the same board

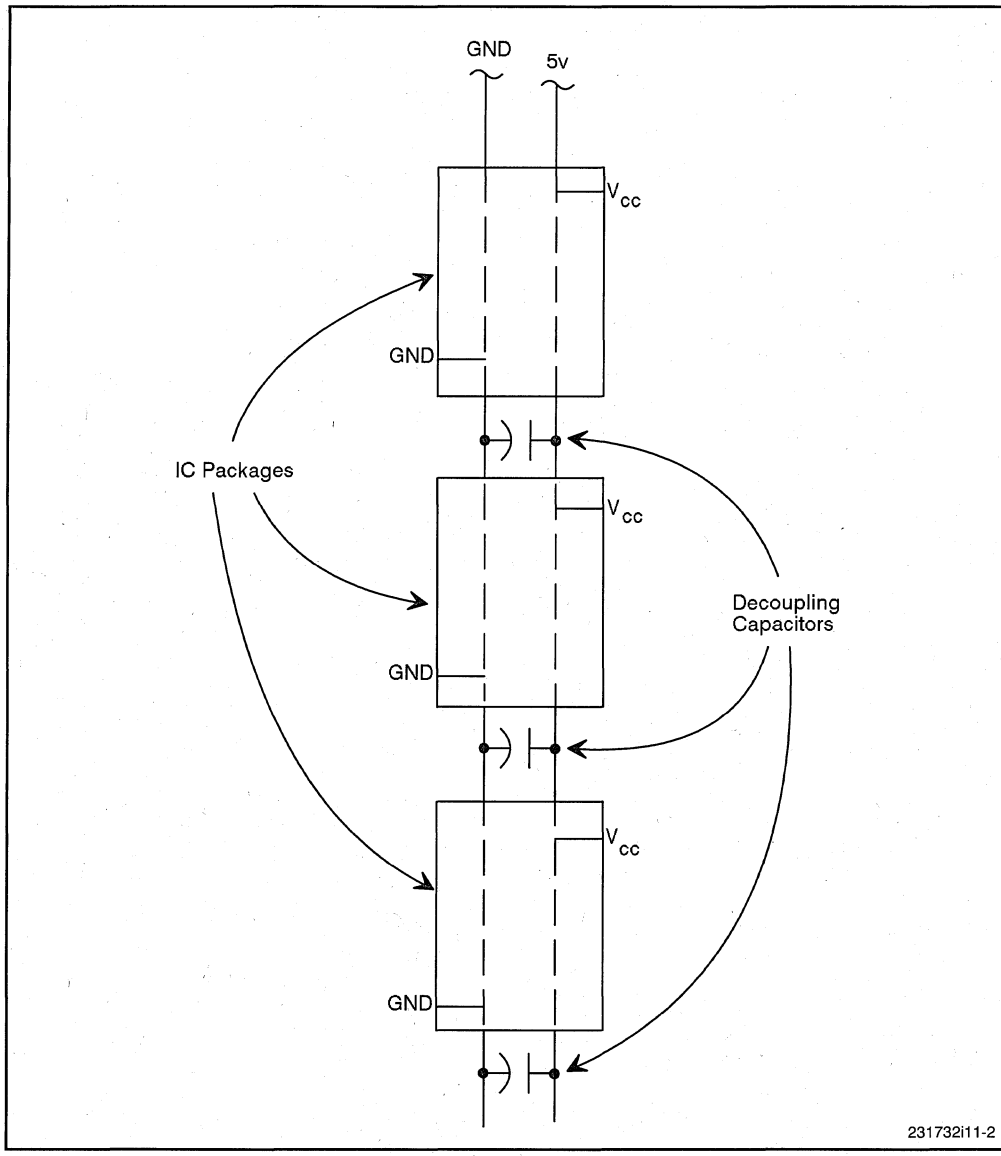


Figure 11-2. Typical Power and Ground Trace Layout for Double-Layer Boards

or other boards in a multi-board system. It is necessary to match the supply's impedance to that of the components in order to lessen the potential for voltage drops that can be caused by IC edge rates, ground- or signal-level shifting, or noise induced currents or voltage reflections.

A mismatch can be minimized by using a suitable high-frequency capacitor for bulk decoupling of major circuitry sections, or for decoupling entire pc boards in multi-board

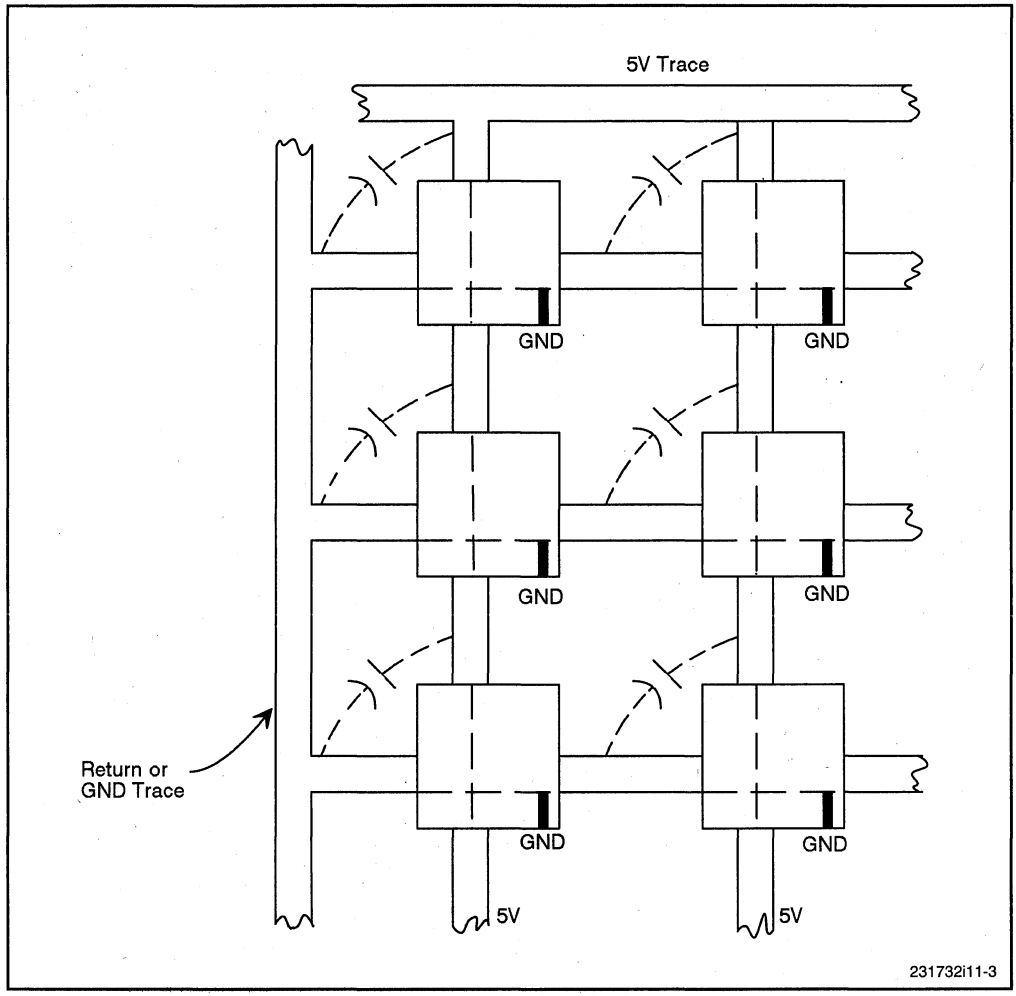


Figure 11-3. Orthogonal Arrangement

systems. This capacitor is typically placed at the supply's entry point to the board. It should be an aluminum or tantalum-electrolytic type capacitor having a low equivalent series capacitance and low equivalent series inductance. The capacitance value is typically 10 to 47 μF. An additional 0.1 μF capacitor may be needed if supply noise is still a problem.

A second type of decoupling is used for the rest of the ICs on the board. Additional decoupling capacitors can be used across the devices between V_{CC} and V_{SS} lines. The voltage spikes that occur due to switching of gates are reduced because the extra current required during switching is supplied by the decoupling capacitors. These capacitors should be placed close to their devices since the inductance of lengthier connection traces reduce their effectiveness.

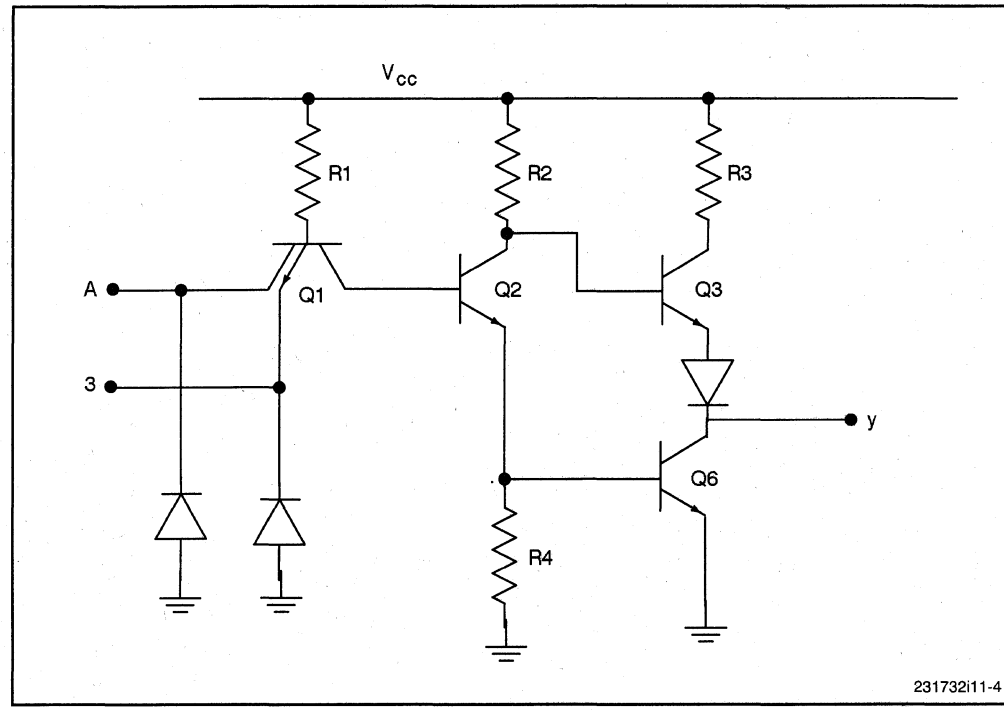


Figure 11-4. Circuit without Decoupling

Most popular logic families require that a capacitor of 0.01 μF to 0.1 μF (RF grade) be placed between every two to five packages, depending on the application. For high-speed CMOS logic, a good rule of thumb is to place one of these bypasses between every two to three ICs, depending on the supply voltage, the operating-speed and EMI requirements. The capacitors should be evenly distributed throughout the board to be most effective. In addition, the board should be decoupled from the external supply line with a 10 to 47 μF capacitor. In some cases, it might be helpful to add a 1 μF tantalum capacitor at major supply trace branches, particularly on large PCBs.

Surface mount (chip) capacitors are preferable for decoupling the Intel386 DX microprocessor because they exhibit lower inductance and require less total board space. They should be connected as shown in Figure 11-5. These capacitors reduce inductance, which keeps the voltage spikes to a minimum. Surface mount capacitors should be used to keep the leads as short as possible.

Inductance is also reduced by the parallel inductor relationships of multiple pins. Six leaded capacitors are required to match the effectiveness of one chip capacitor, but because only a limited number can fit around Intel386 DX CPU, the configuration shown in Figure 11-6 is recommended if leaded capacitors are used.

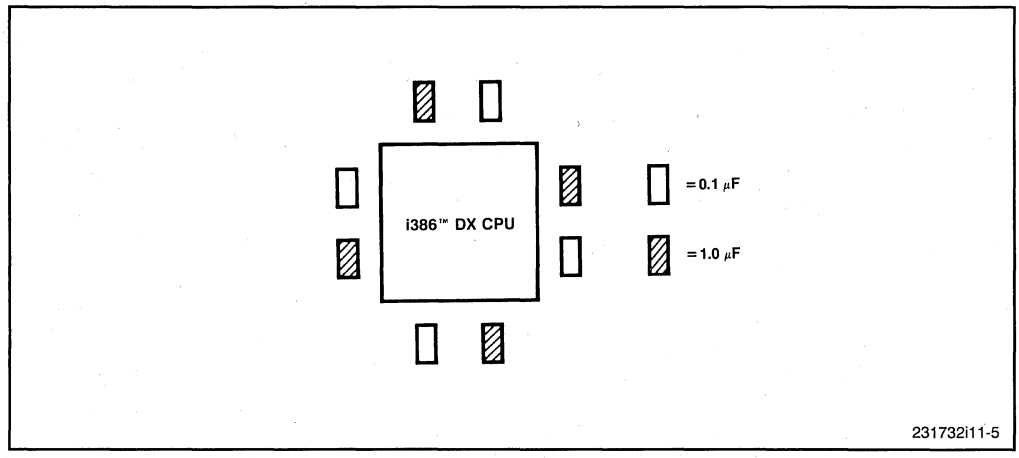


Figure 11-5. Decoupling with Surface Mount Capacitors

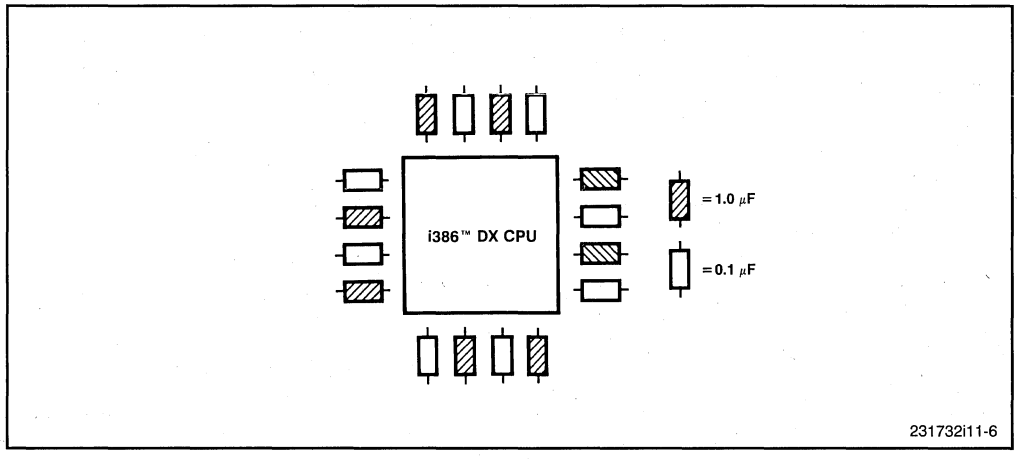


Figure 11-6. Decoupling with Leaded Capacitors

11.4 HIGH FREQUENCY DESIGN CONSIDERATIONS

The overwhelming concern in dealing with high speed technologies is the management of transmission lines. As the edge rates of the signals increase, the physical interconnections between devices behave like transmission lines. Although transmission-line theory is straightforward, the difference between ordinary interconnections and transmission lines is fairly complex. Transmission lines have distributed elements which are hard to define and designers tend to over compensate for the effects of these elements.

Efficient Intel386 DX CPU designs require the identification of the transmission lines over back plane wiring, printed circuit board traces, etc. Once this task is accomplished, designer's next concern should be to deal with problems associated with electromagnetic propagation, impedance control, propagation delay and coupling.

The following sections discuss the negative effects of a transmission line that occur when operating at higher frequencies. In higher frequency design the reflection and cross talk effects are inevitable; it is impossible to design optimum systems without accounting for these effects. Later sections include a discussion of techniques that can minimize these effects.

11.4.1 Transmission Line Effects

As a general rule, any interconnection is considered to be a transmission line when the time required for the signal to travel the length of the interconnection is greater than one-eighth of the signal rise time (True K.M., *"Reflection: Computations and Waveforms, The Interface Handbook,"* Fairchild Corp., Mountain View, CA., 1975, Ch. 3). The rise time can be either rise time or fall time, whichever is smaller, and it corresponds to the linear ramp amplitude from 0% to 100%. Normally the rise times are specified between 10% to 90% or 20% to 80% amplitude points. The respective values are multiplied by 1.25 or 1.67 to obtain the linear-ramp duration from 0% to 100% amplitude.

For example in a PCB using G-10 and polymide (the two main dielectric systems available for printed circuit boards) signals travel at approximately 5 to 6 inches per nanosecond (ns).

If $t_r v/l < 8$ then the signal path is not a transmission line but it is a lumped element (True K.M., *"Reflection: Computations and Waveforms, The Interface Handbook,"* Fairchild Corp., Mountain View, CA., 1975, Ch. 3).

where

t_r = rise time 0%–100%
v = speed of propagation (5 to 6 inches/sec)
l = length of interconnection (one-way only)

The calculation is given by:

$6t_r/l \leq 8$ so $l \geq 6t_r/8 = (6 \times 4 \times 1.67)/8 \approx 5.01$ inches

This calculation is based on the fact that the maximum rise time of the signals for the Intel386 DX processor is 4 ns. For $l >= 5.01$ inches, interconnections will act as transmission lines.

Every conductor that carries an AC signal and acts as a transmission line has a distributed resistance, an inductance and a capacitance which combine to produce the characteristic impedance (Z0). The value of Z0 depends upon physical attributes such as cross-sectional area, the distance between the conductors and other ground or signal conductors, and the dielectric constant of the material between them. Because the characteristic impedance is reactive, its effect increases with frequency.

11.4.1.1 TRANSMISSION LINE TYPES

Although many different types of transmission lines (conductors) exist, those most commonly used on the printed circuit boards are micro strip lines, strip lines, printed circuit traces, side-by-side conductors and flat conductors.

11.4.1.1.1 Micro Strip Lines

The micro strip trace consists of a signal plane that is separated from a ground plane by a dielectric as shown in Figure 11-7. G-10 fiber-glass epoxy, which is most common, has an $e_r = 5$ where e_r is the dielectric constant of the insulation.

w = the width of signal line (inches)
t = the thickness of copper (.0015 inches for 1 oz Cu/.003 inches for 2 oz Cu)
h = the height of dielectric for controlled impedance (inches)

The characteristic impedance, Z0, is a function of dielectric constant and the geometry of the board. This is theoretically given by the following formula:

$$Z0 = [87/\sqrt{(e_r + 1.41)}] \ln (5.98h/.8w+t) \text{ ohms}$$

where e_r is the relative dielectric constant of the board material and h, w and t are the dimensions of the strip. Knowing the line width, the thickness of Cu and the height of dielectric, the characteristic impedance can be easily calculated.

The propagation delay (tpd) associated with the trace is a function of the dielectric only. This is calculated as follows:

$$tpd = 1.017 \sqrt{(0.475e_r + 0.67)} \text{ ns/ft}$$

For G-10 fiber-glass epoxy boards ($e_r = 5.0$), the propagation delay of micro strip is calculated to be 1.77 ns/ft.

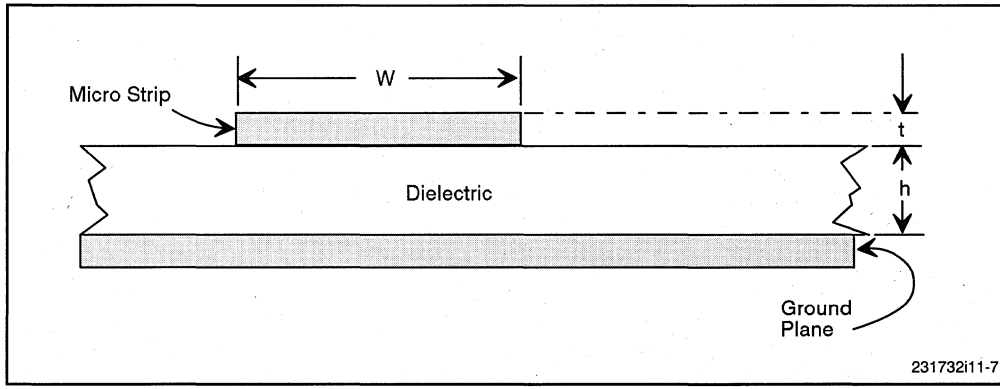


Figure 11-7. Micro Strip Lines

11.4.1.1.2 Strip Lines

A strip line is a strip conductor centered in a dielectric medium between two voltage planes. The characteristic impedance is given theoretically by the equation below:

$Z0 = (60/\sqrt{e_r}\ \ln (5.98b/\pi\ (0.8\ W + t))$ ohms

where b = distance between the planes for controlled impedance as shown in Figure 11-8.

The propagation delay is given by the following formula

$tpd = 1.017\ \sqrt{e_r}$ ns/ft

For G-10 fiberglass epoxy boards ($e_r = 5.0$), the propagation delay of the strip lines is 2.27 ns/ft.

Typical values of the characteristic impedance and propagation delay of these types of lines are as follows:

Z0 = 50 ohms
tpd = 2 ns/ft (or 6″/ns)

The three major effects of transmission line phenomenon are impedance mismatch, coupling and skew.

The following section will discuss them briefly and provide solutions to minimize their effects.

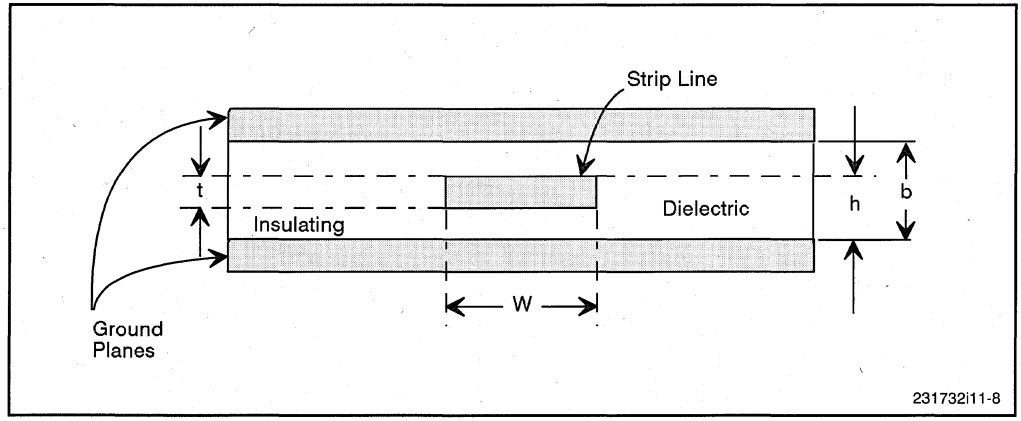


Figure 11-8. Strip Lines

11.4.2 Impedance Mismatch

As mentioned earlier, the impedance of a transmission line is a function of the geometry of the line, its distance from the ground plane, and the loads along the line. Any discontinuity in the impedance will cause reflections.

Impedance mismatch occurs between the transmission line characteristic impedance and the input or output impedances of the devices that are connected to the line. The result is that the signals are reflected back and forth on the line. These reflections can attenuate or reinforce the signal depending upon the phase relationships. The results of these reflections include overshoot, undershoot, ringing and other undesirable effects.

At slower edge rates, the effects of these reflections are not severe. However at faster rates, the rise time of the signal is short with respect to the propagation delay. Thus it can cause problems as shown in Figure 11-9.

Overshoot occurs when the voltage level exceeds the maximum (upper) limit of the output voltage, while undershoot occurs when the level exceeds the minimum (lower) limit. These conditions can cause excess current on the input gates which results in permanent damage to the device.

The amount of reflection voltage can be easily estimated. Figure 11-10 shows a system exhibiting reflections.

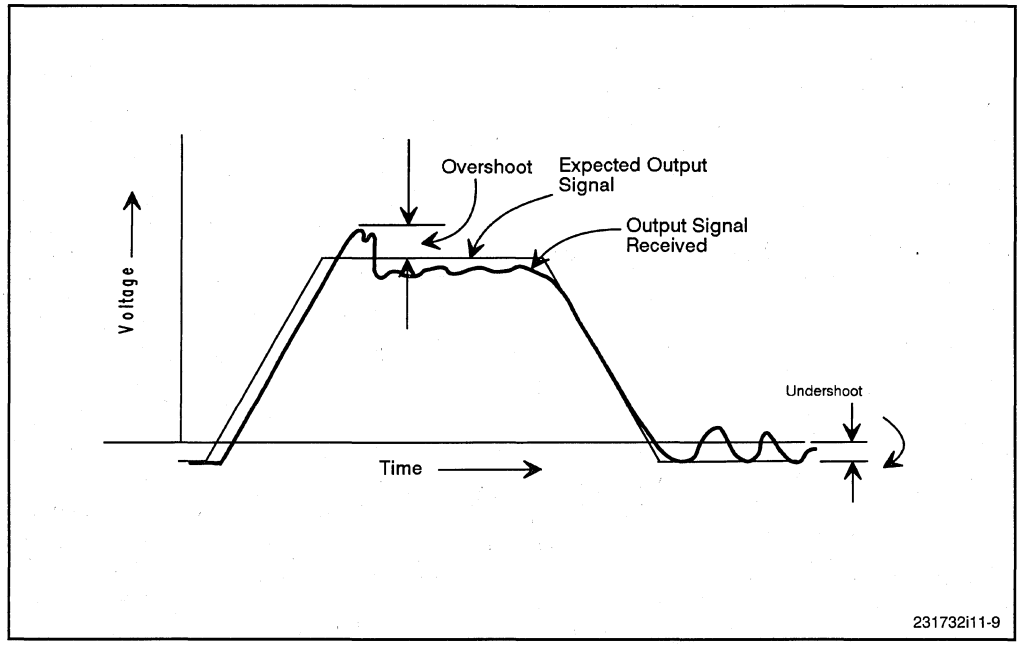


Figure 11-9. Overshoot and Undershoot Effects

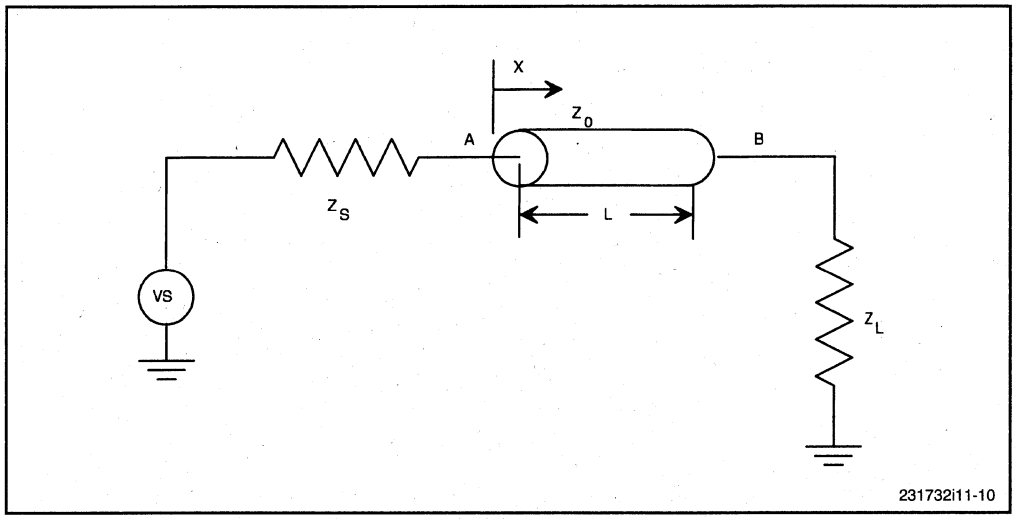


Figure 11-10. Loaded Transmission Line

The magnitude of a reflection is usually represented in terms of a reflection coefficient. This is illustrated in the following equations:

$T = v_r/v_i$ = Reflected voltage/Incident voltage

$T_{load} = (Z_{load} - Z_0)/(Z_{load} + Z_0)$

$T_{source} = (Z_{source} - Z_0)/(Z_{source} + Z_0)$

Reflection voltage V_r is given by V_i, the voltage incident at the point of the reflection, and the reflection coefficient.

The model transmission line can now be completed. In Figure 11-10, the voltage seen at point A is given by the following equation:

$V_a = V_s * Z_0/(Z_0 + Z_s)$

This voltage V_a enters the transmission line at "A" and appears at "B" delayed by t_{pd}.

$V_b = V_a\ (t - x/v)\ H(t - x/v)$

where x = distance along the transmission line from point "A" and H(t) is the unit stop function. The waveform encounters the load Z_L, and this may cause reflection. The reflected wave enters the transmission line at "B" and appears at point "A" after time delay (t_{pd}):

$V_{r1} = T_{load} \cdot V_b$

This phenomenon continues infinitely, but it is negligible after 3 or 4 reflections. Hence:

$$V_{r2} = T_{source} \cdot V_{r1}$$

Each reflected waveform is treated as a separate source that is independent of the reflection coefficient at that point and the incident waveform. Thus the waveform from any point and on the transmission line and at any given time is as follows:

$$\begin{aligned} V(x,t) = \left(\frac{Z_0}{Z_0+Z_s}\right) \Bigg\{ & V_s\left(t-\frac{x}{v}\right) H\left(t-\frac{x}{v}\right) \\ & + T_l \left[V_s\left(t-\left(\frac{2L-x}{v}\right)\right) H\left(t-\left(\frac{2L-x}{v}\right)\right)\right] \\ & + T_l T_s \left[V_s\left(t-\left(\frac{2L+x}{v}\right)\right) H\left(t-\left(\frac{2L+x}{v}\right)\right)\right] \\ & + T_l^2 T_s \left[V_s\left(t-\left(\frac{4L-x}{v}\right)\right) H\left(t-\left(\frac{4L-x}{v}\right)\right)\right] \\ & + T_l^2 T_s^2 \left[V_s\left(t-\left(\frac{4L+x}{v}\right)\right) H\left(t-\left(\frac{4L+x}{v}\right)\right)\right] \\ & + \ldots \Bigg\} \end{aligned}$$

Each reflection is added to the total voltage through the unit step function H(t). The above equation can be rewritten as follows:

$$\begin{aligned} V(x,t) = \left(\frac{Z_0}{Z_0+Z_s}\right) \{ & V_s(t-t_{pd}x)\ H(t-t_{pd}x) \\ & + T_l\ [V_s(t-t_{pd}(2L-x))\ H(t-t_{pd}(2L-x))] \\ & + T_l T_s\ [V_s(t-t_{pd}(2L+x))\ H(t-t_{pd}(2L+x))] \\ & + \ldots \} \end{aligned}$$

The lattice diagram is a convenient visual tool for calculating the total voltage due to reflections as described in the equations above. Two vertical lines are drawn to represent points A and B on the horizontal dimension x. The vertical dimension then represents time. A waveform will travel back and forth between points A and B of the transmission line in time, producing the lattice diagram shown in Figure 11-11. The voltage at a given point is the sum of all the individual reflected voltages up to that time. Notice that at each endpoint two waves are converging—the incident wave and the reflected wave. Therefore, the voltage at the endpoints A or B at the time of the waveform reflection would be calculated by summing both the incident and reflected waves up to and including the point in question.

As an example, let the simple configuration shown in Figure 11-10 be assumed. Assume the following:

$V_s = 3.70\ H(t)\ V$
$Z_0 = 75\ \Omega$
$Z_{source} = 30\ \Omega$
$Z_{load} = 100\ \Omega$

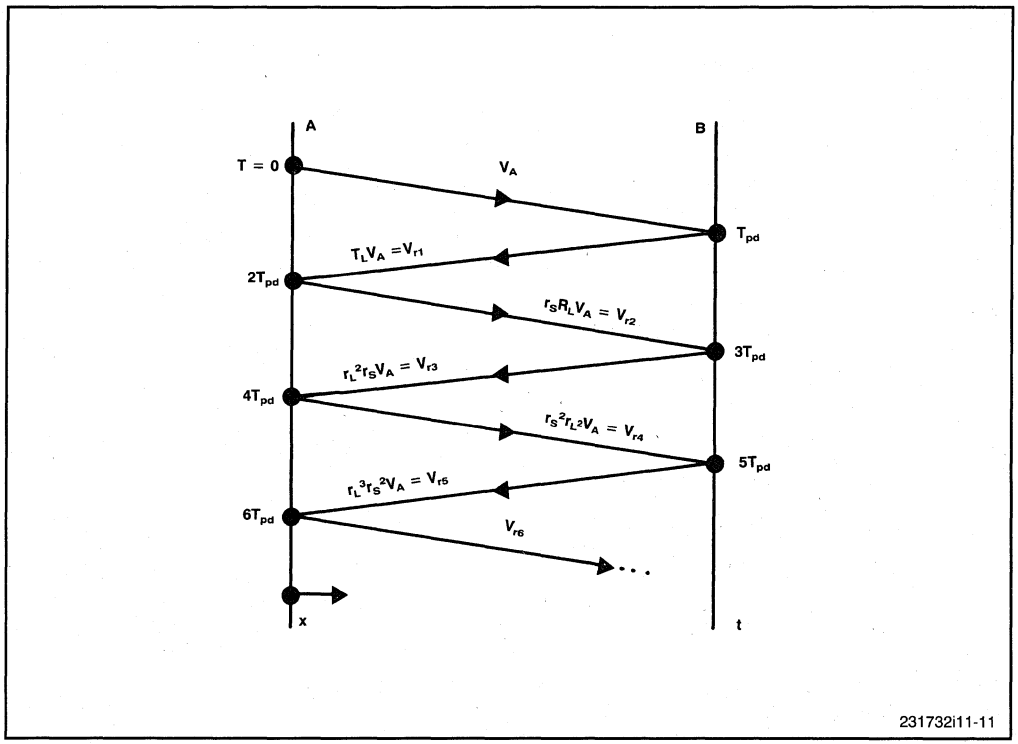


Figure 11-11. Lattice Diagram

The appropriate reflection coefficients can be calculated as follows:

$\Gamma_{source} = (30 - 75)/(30 + 75) = -0.42857$
$\Gamma_{load} = (100 - 75)/(100 + 75) = 0.14286$
$V_a = V_s \times 75/(75 + 30) = 2.64286$ V
$V_{r1} = 2.64286 \times 0.14286 = 0.37755$ V
$V_{r2} = 0.37755 \times -0.42857 = -0.16181$ V
$V_{r3} = -0.16181 \times 0.14286 = -0.02312$ V
$V_{r4} = -0.02312 \times -0.42857 = 0.00991$ V
$V_{r5} = 0.00991 \times 0.14286 = 0.00142$ V
$V_{r6} = 0.00142 \times -0.42857 = -0.00061$ V
$V_{r7} = -0.00061 \times 0.14286 = -0.00009$ V

Figure 11-12 shows the corresponding lattice diagram.

Thus the voltage at point B can be tabulated as shown in Table 11-1.

Impedance discontinuity problems are managed by imposing limits and control during the routing phase of the design. Design rules must be observed to control trace geometry, including specification of the trace width and spacing for each layer. This is very important because it ensures the traces are smooth and constant without sharp turns.

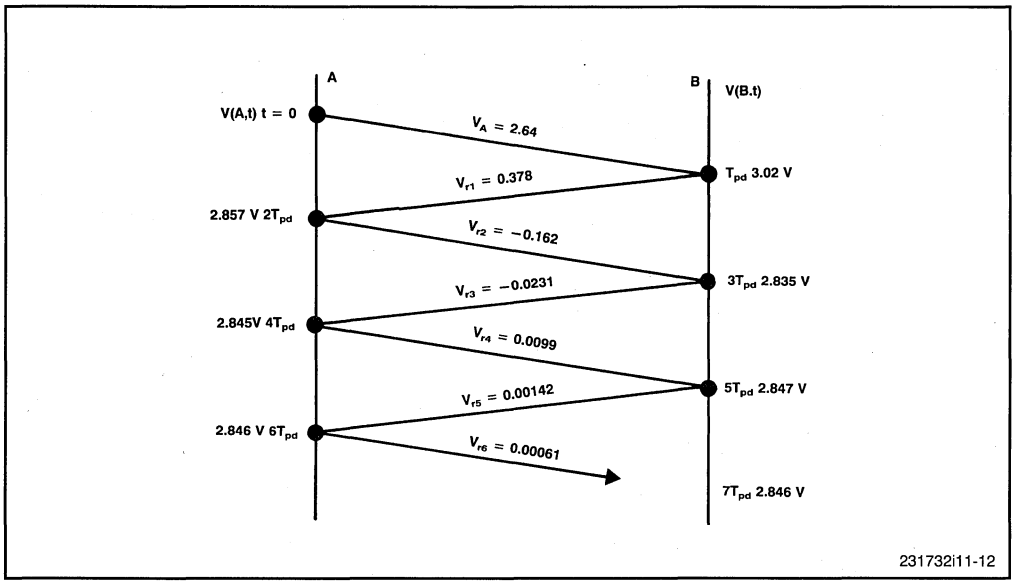


Figure 11-12. Lattice Diagram Example

Table 11-1. Voltage at End Points A and B

t_{pd}	V(A,t)	V(B,t)
0	2.64286	0
1t_{pd}	2.64286	3.02041
2t_{pd}	2.85860	3.02041
3t_{pd}	2.85860	2.83549
4t_{pd}	2.84539	2.83549
5t_{pd}	2.84539	2.84681
6t_{pd}	2.84620	2.84681
7t_{pd}	2.84620	2.84611

There are several techniques which can be employed to further minimize the effects caused by an impedance mismatch during the layout process:

1. Impedance matching
2. Daisy chaining
3. Avoidance of 90° corners.
4. Minimization of the number of vias.

11.4.2.1 IMPEDANCE MATCHING

Impedance matching is the process of matching the impedance of the source or load with that of the trace. It is accomplished with a technique called termination. The reflection, overshoot and undershoot of signals are reduced by terminating the remote end of

the transmission line from the source. The terminating impedance combines with the destination input circuitry to produce a load that closely matches the characteristic impedance of the line (board traces have characteristic impedances in the range of 30 ohms to 200 ohms).

The calculation of characteristic impedance was already discussed. Impedance of the printed circuit board backplane connectors have the impedance in the same range as the traces (i.e., 30 to 200 ohms).

The characteristic impedance of a backplane may change, depending upon the length of the conductors or when using twisted pairs of co-axial cable in place of PC traces. Backplane impedance is also affected by the number of boards plugged into the backplane.

11.4.2.1.1 Need for Termination

The transmission line should be terminated when the tpd exceeds one-third of tr (rise time). If the tpd < 1/3 tr, the line can be left unterminated, provided the capacitive coupling between the traces does not cause cross-talk.

Termination thus eliminates impedance mismatches, increases noise immunity, suppresses RFI/EMI, and helps to ensure that signals reach their destination with minimum of distortion. There are five methods for terminating traces on the board:

1. Series
2. Parallel
3. Thevenin
4. A.C.
5. Active

Terminations can be costly, because they require additional components and power. With passive terminations, extra drivers are needed to deliver more current to the line. With active terminations extra power is needed which increases the power dissipation of the system.

11.4.2.1.2 Series Termination

One way of controlling ringing on longer lines is with the series termination technique also known as damping. This is accomplished by placing a resistor in series with the transmission line at the driving device end. The receiver has no termination. The value of the impedance looking into the driving device ($R_{driver} + R_{line} = Z_0$) should approximate the impedance of the line as closely as possible. In this circuit the ringing dampens out when the reflection coefficient goes to zero. Figure 11-13 illustrates the series termination.

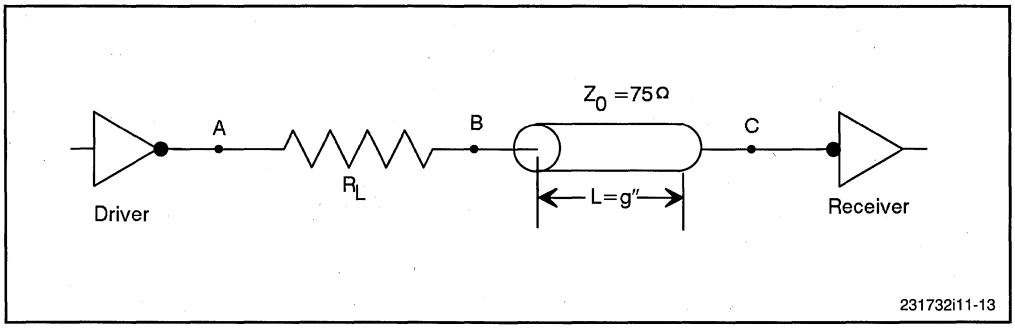


Figure 11-13. Series Termination

One main advantage of series termination is that only logic power dissipation results so that lower overall power is required than with other techniques. There is one penalty, however, in that the distributed loading along the transmission line cannot be used because only half of the voltage waveform is travelling down the line. There is no limit on the number of loads that can be placed at the end of the series terminated connection. However, the drop in voltage across the series terminating resistor limits loading to a maximum of 10 loads.

11.4.2.1.3 Parallel Terminated Lines

Parallel termination is achieved by placing a resistor of an appropriate value between the input of the loading device and the ground as shown in Figure 11-14. To determine an appropriate value, the currents required by all inputs and the leakage currents of the drivers should be summed. A resistor should be selected so that its addition to the circuit does not exceed the output capacity of the weakest driver.

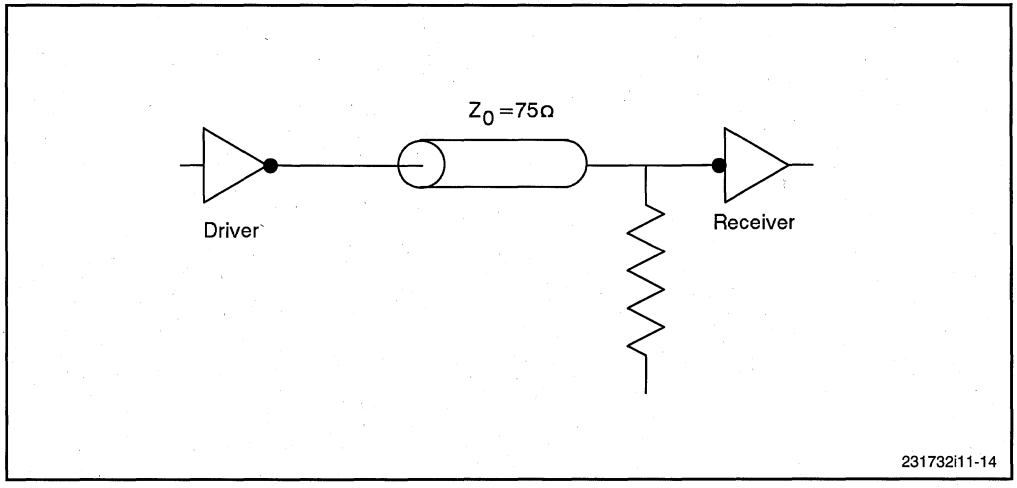


Figure 11-14. Parallel Termination

Since the input impedance of the device is high compared to the characteristic line impedance, the resistor and the line function as a single impedance with a magnitude that is defined by the value of the resistor.

When the resistor matches the line impedance, the reflection coefficient at the load approaches zero, and no reflection will occur. One useful approach is to place the termination as close to the loading device as possible.

Parallel terminated lines are used to achieve optimum circuit performance and to drive distributed loads (an important benefit of using parallel terminations).

There are two significant advantages of using the parallel termination. First, it provides an undistributed waveform along the entire line. Second, when a long line is loaded in parallel termination, it does not affect the rise and fall time or the propagation delay of the driving device. Note that parallel termination can also be used with wire wrap and backplane wiring where the characteristic impedance is not exactly defined. If the designer approximates the characteristic impedance, the reflection coefficient will be very small. This results in minimum overshoot and ringing. Parallel termination is not recommended for characteristic impedances of less than 100 ohms because of large d.c. current requirements.

11.4.2.1.4 Thevenins Equivalent Termination

This technique is an extension of parallel termination technique. It consists of connecting one resistor from the line to the ground and another from the line to the V_{CC}. Each resistor has a value of twice the characteristic impedance of the line, so the equivalent resistance matches the line impedance. This scheme is shown in Figure 11-15.

If there were no logic devices present, the line would be placed half way between the V_{CC} and the V_{SS}. When the logic device is driving the line, a portion of the required current is provided by the resistors, so the drivers can supply less current than needed in parallel termination. The resistor value can be adjusted to bias the lines towards the V_{CC} or the V_{SS}. Ordinarily it is adjusted such that the two are equal, providing balanced performance. The Thevenin's circuit provides good overshoot suppression and noise immunity.

Due to power dissipation, this technique is best suited for bipolar and mix MOS devices and is not suitable for pure CMOS implementations. The reasons for not using Thevenin's equivalent for the pure CMOS system design are as follows:

First CMOS circuits have very high impedance to both ground and V_{CC}, and their switching threshold is 50% of the supply voltage. Second, besides dissipating more power, multiple input crossings may occur which creates output oscillations.

The main problem with Thevenin termination is high power dissipation through the termination resistors in relationship to the total power consumption of all of the CMOS devices on the board. For this reason, most designers prefer series terminations for CMOS to CMOS connections as this does not introduce any additional impedance from the signal to the ground. The main advantage of the series termination technique, apart

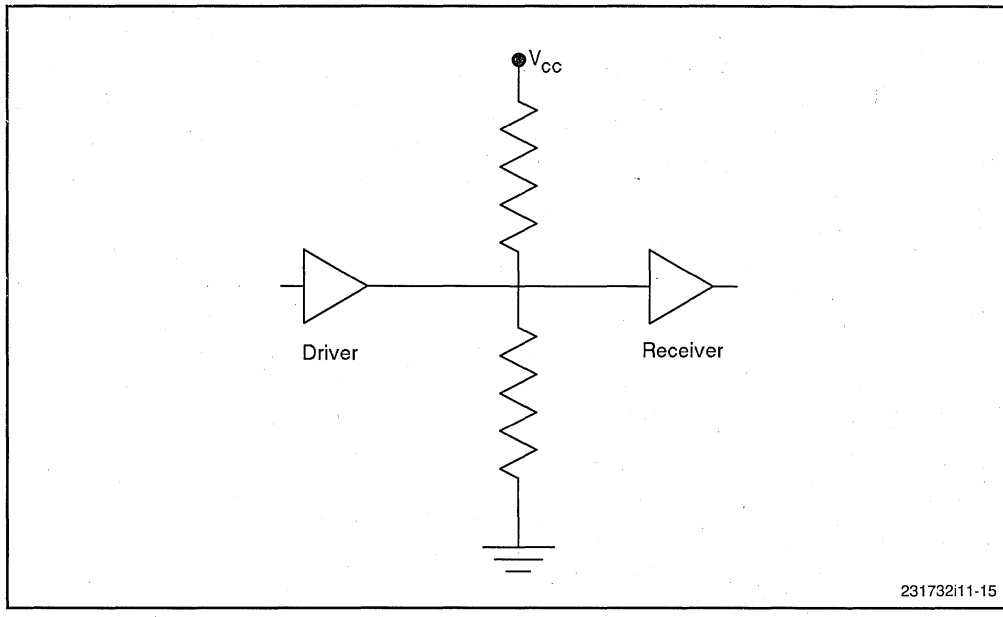


Figure 11-15. Thevenins Equivalent Circuit

from its reduced power consumption, is its flexibility. The received signal amplitude can be adjusted to match the switching threshold of the receiver simply by changing the value of the terminating resistor. This is a very useful technique for interconnecting the logic devices with long lines.

11.4.2.1.5 A.C. Termination

Another technique that can be used for designs which cannot tolerate the high power dissipation of parallel termination and the delays created by series termination is A.C. termination. It consists of a resistor and a capacitor connected in series from the line to the ground. It is similar to the parallel termination technique in functionality except that the capacitor blocks the D.C. component of the signal and thus reduces the power dissipation. This technique is shown in Figure 11-16.

The main disadvantage of A.C. termination is that it requires two components. Further, the optimum value for the RC time constant of the termination network is not easy to calculate. One usually begins with a resistive value which is slightly larger than the characteristic line impedance. It is then critical to determine the capacitor value. If the value of the RC time constant is small, the RC circuit will act as an edge generator and will create overshoots and undershoots. Increasing the capacitor value reduces the overshoot and undershoot, but it increases power consumption. As a rule of thumb, the RC time constant should be greater than twice the line delay. The power dissipation of A.C. termination is a function of the frequency.

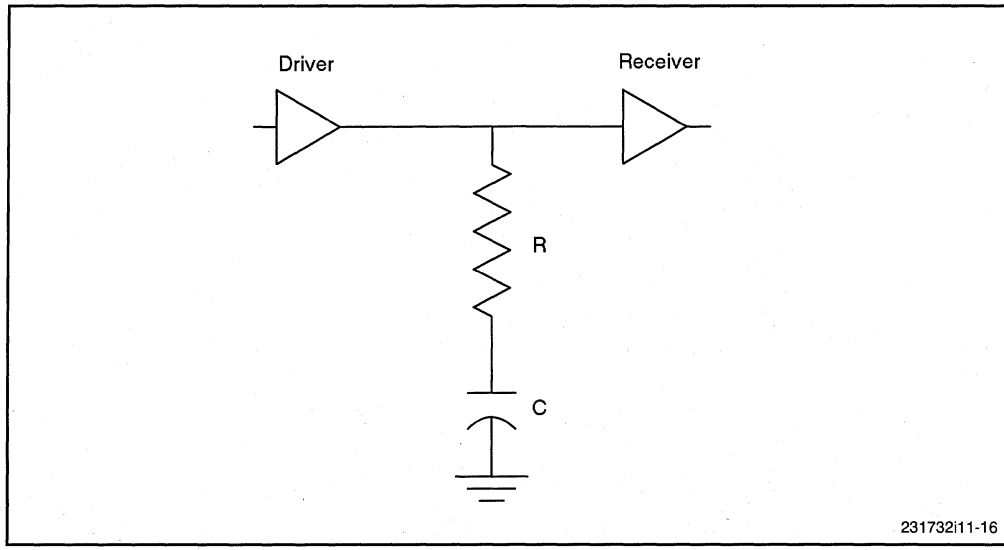


Figure 11-16. A.C. Termination

11.4.2.1.6 Active Termination

These terminations consist of resistors that are connected between the inputs and outputs of buffer drivers as shown in Figure 11-17.

The main advantage of this technique is that it can tolerate large impedance variations and this tolerance is valuable when tri-state drivers are connected to backplane buses. However, the terminations are costly, and the signals that are produced are not as clean

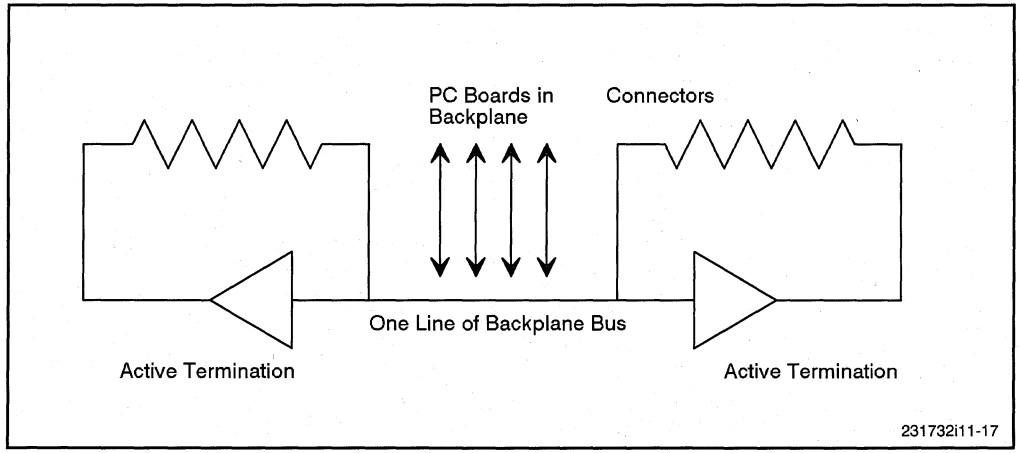


Figure 11-17. Active Termination

as other terminations. A common solution is to place active terminations at both ends of the bus. This helps to maintain the uniform drive levels along the entire length of the bus, and it reduces crosstalk and ringing.

Table 11-2 shows the comparisons of different termination techniques.

11.4.2.1.7 Impedance Matching Example

Previous sections discuss the techniques for calculating characteristic impedances (using transmission line theory) and the termination procedures used match impedances. This section describes an impedance matching example that utilizes these techniques. Figure 11-18 shows a simple interconnection which acts like a transmission line as shown by the calculations.

In this example the different values are given as follows:

Z_s = source impedance = 10 ohms
trs = source rise-time = 3 nsec (normalized to 0% to 100%)
Z_1 = load impedance = 10 Kohms
trl = load rise-time = 3 nsec (normalized to 0% to 100%)
l = length of interconnection = 9 in.
trace = micro-strip
e_r = dielectric constant = 5.0
h = .008 in.
w = .01 in.
t = .0015 in. (1 oz. Cu)
v = 6 in./nsec

The interconnection will act as a transmission line if (as was shown in Section 11.4.1).

$l \geq (t_r \cdot V)/8$
$(t_r \cdot V)/8 = 2.25$

The value of l = 9 in., thus the interconnection acts like a transmission line. The impedance of the transmission line is calculated as follows:

$Z0 = 87/\sqrt{(e_r + 1.41)} \times \ln(5.98h/(.8w+t))$
$= 34.39 \ln 5.035 = 55.6$ ohms

Table 11-2. Comparison of Various Termination Techniques

Termination	# of Extra Components	R_L Power Consumption		Prop Delay
Series	1	Z_0-Zout	Low	Yes
Parallel	1	Z_0	High	No
Thevenin	2	$2Z_0$	High	No
A.C.*	2	Z_0	Medium	No
Active	1	$2Z_0$	Medium	No

* A.C. also uses a capacitor of 200 pf to 500 pf.

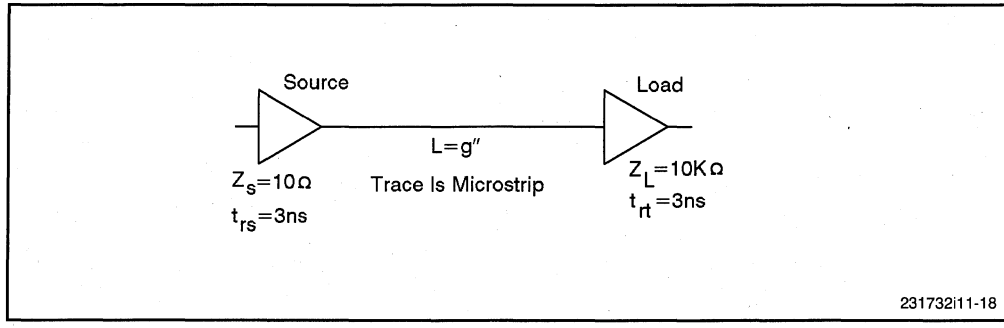


Figure 11-18. Impedance Mismatch Example

Since Z_s = 10 ohms, the termination techniques described previously will be needed to match the difference of 45.6 ohms. One method is to use a series terminating resistor of 45.6 ohms or use A.C. termination where R=55.6 ohms and C=0.3 μF. The terminated circuit of Figure 11-18 is shown in Figure 11-19.

11.4.2.2 DAISY CHAINING

In laying out PC boards, a stub or T-connection is another source of signal reflection. These types of connections act as inductive loads in the signal path. In daisy chaining, a single trace is run from the source, and the loads are distributed along this trace. This is shown in Figure 11-20.

An alternative way to this technique is to run multiple traces from the source to each load. Each trace will have unique reflections. These reflections are then transmitted down other traces when they return to the source. In such cases a separate termination is required for each branch. To eliminate these T-connections, high-frequency designs are routed as daisy chains.

Along the chain, each gate provides its own impedance load, thus it is, necessary to distribute these loads evenly along the length of the chain. Hence, the impedance along the chain will change in a series of steps and it is easier to match. The overall speed of this line is faster and predictable.

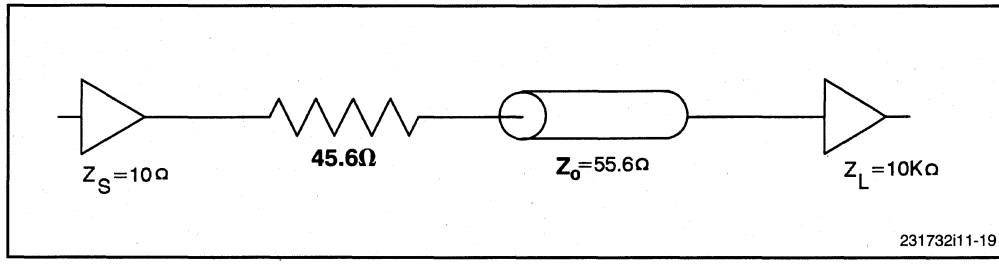


Figure 11-19. Use of Series Termination to Avoid Impedance Mismatch

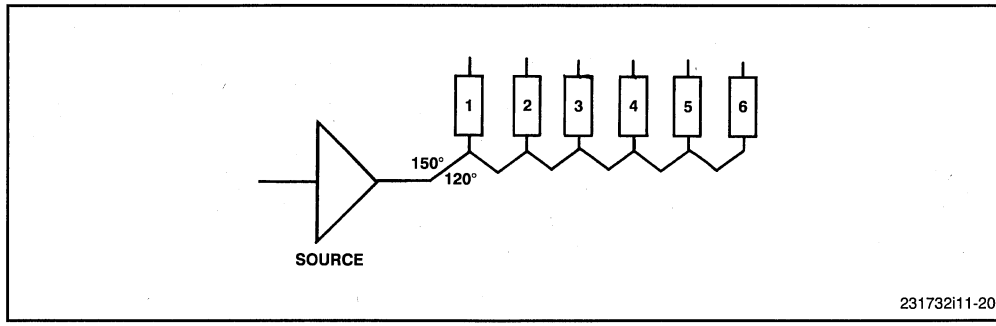


Figure 11-20. Daisy Chaining

11.4.2.3 90-DEGREE ANGLES

Another major cause of reflections are 90-degree angles in the signal paths, which cause an abrupt change in the signal direction. It promotes signal reflection. For high-frequency layout of designs, avoid 90-degree angles and use 45 or 120-degree angles as shown in Figure 11-21.

11.4.2.4 VIAS (FEED THROUGH CONNECTIONS)

Another impedance source that degrades high frequency circuit performance is vias. Expert layout techniques can eliminate vias to avoid reflection sites on PCBs.

11.4.3 Interference

Previous sections discussed reflections in high-frequency design, their causes, and techniques to minimize them. The following sections discuss additional issues related to high

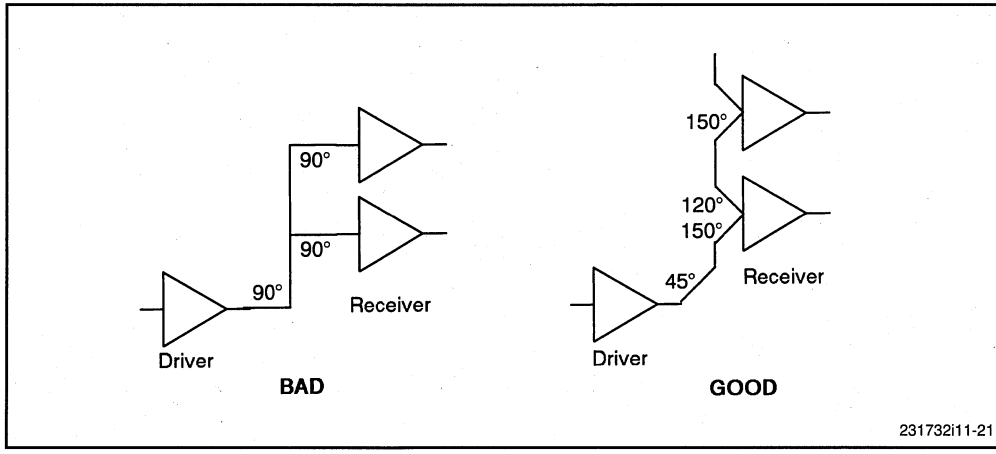


Figure 11-21. Avoiding 90-Degree Angles

frequency design, including interference. In general, interference occurs when electrical activity in one conductor causes transient voltage to appear in another conductor. Two main factors increase the interference in any circuit:

1. Variation of current and voltage in the lines causes frequency interference. This interference increases with frequency.

2. Coupling occurs when conductors are in close proximity.

Two types of interference are observed in high frequency circuits:

1. Electromagnetic Interference (EMI)

2. Electrostatic Interference (ESI)

11.4.3.1 ELECTROMAGNETIC INTERFERENCE (CROSS-TALK)

Cross-talk is a problem at high operating frequencies: when operating frequency increases, signal wavelength becomes comparable to the length of some of the interconnections on the PC board. Cross-talk is a phenomenon of a signal in one trace which induces another similar signal in an adjacent trace. There are two types of couplings between parallel traces which determine the amount of cross-talk in a circuit. These are called inductive coupling and radiative coupling.

Inductive coupling occurs when current in one trace produces current in a parallel trace. This current decreases with increasing distance from the source. Hence, closely spaced wires or traces will incur the greatest degree of inductive coupling. Both the traces in this case act like normal conductors.

Radiative coupling occurs when two parallel traces act as a dipole antenna which radiates signals that parallel wires can pick up. This results in the corruption of the signal that is already present in the trace. The intensity of this type of coupling is directly proportional to the current present in the trace. However, it is inversely proportional to the distance between the radiating source and the receiver.

11.4.3.2 MINIMIZING CROSS-TALK

When laying out a board for a high frequency processor-based system, several guidelines should be followed to minimize cross-talk.

One source of cross-talk is the presence of a common impedance path. Figure 11-22 shows a typical layout which does not have the same earth ground and signal ground.

To reduce cross-talk, it is necessary to minimize the common impedance paths, which are Z2, Z3 and Z4 shown primarily as ground impedances. During current switching, the ground line voltage drops causing noise emission. By enlarging the ground conductor

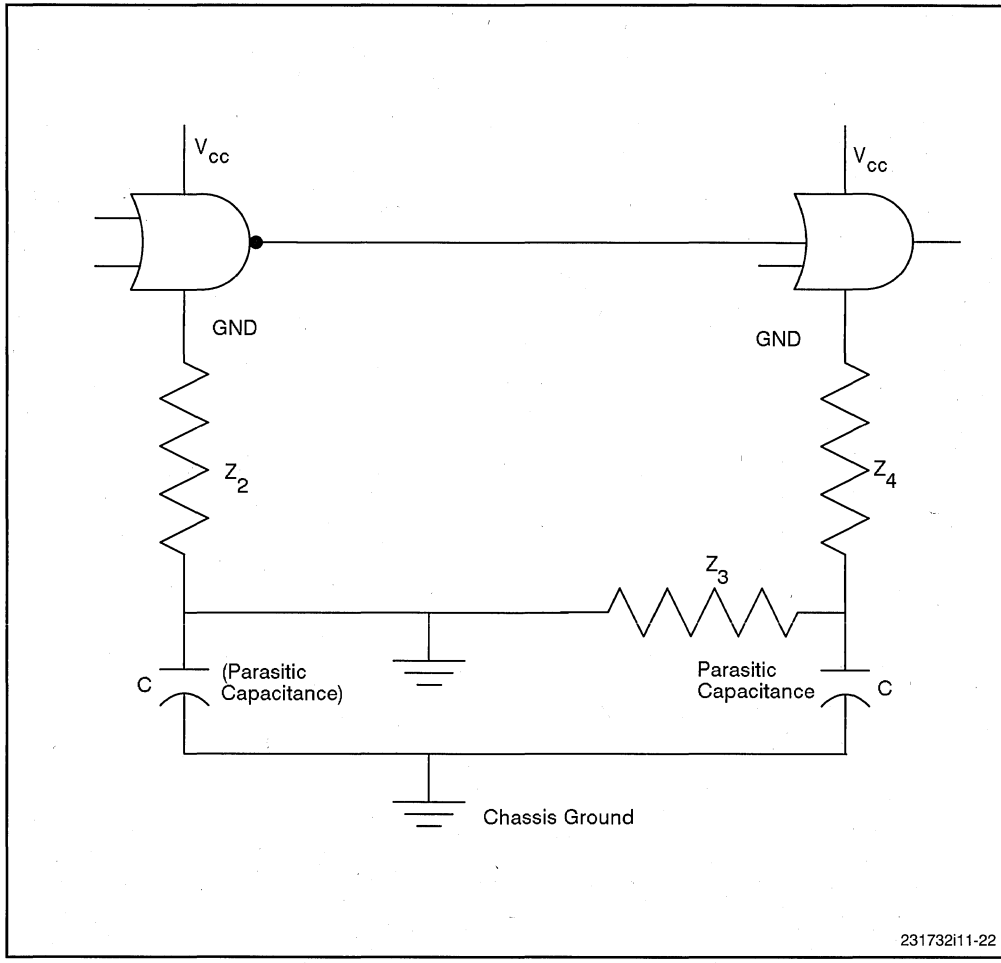


Figure 11-22. Typical Layout

(which reduces its effective impedance), this noise can be minimized. This technique also provides a secondary advantage in that it forms a shield which reduces the emissions of other circuit traces, particularly in multi-layer circuit boards.

The impedances Z2 through Z4 depend upon thickness of copper pc-board foil, the circuit switching speeds, and the effective lengths of the traces. The current flowing through these common impedance paths radiates more noise as its value increases. The amount of voltage generated by these switching currents and multiplied by the impedance is difficult to predict.

An effective way of reducing EMI is to decouple the power supply by adding bypass capacitors between V_{CC} and ground. This technique is similar to the general technique discussed earlier (the goal of the previous technique was to maintain correct logic levels).

The design of effective decoupling and bypass schemes centers on maximizing the charge stored in the circuit bypass loops while minimizing the inductances in these loops. Some other precautions that can minimize the EMI are as follows:

- Running a ground line in between two adjacent signal lines. The extra line should be grounded at both ends.
- The address and data buses can also be separated by ground lines. This technique may be expensive due to large number of address and data lines.
- Removing closed-loop signal paths that can create inductive noise (as shown in Figure 11-23).

Minimizing cross-talk involves first examining the circuit's interconnection with its nearest neighbors since parallel and adjacent lines can interact and cause EMI. It is necessary to maximize the distance between adjacent parallel wires.

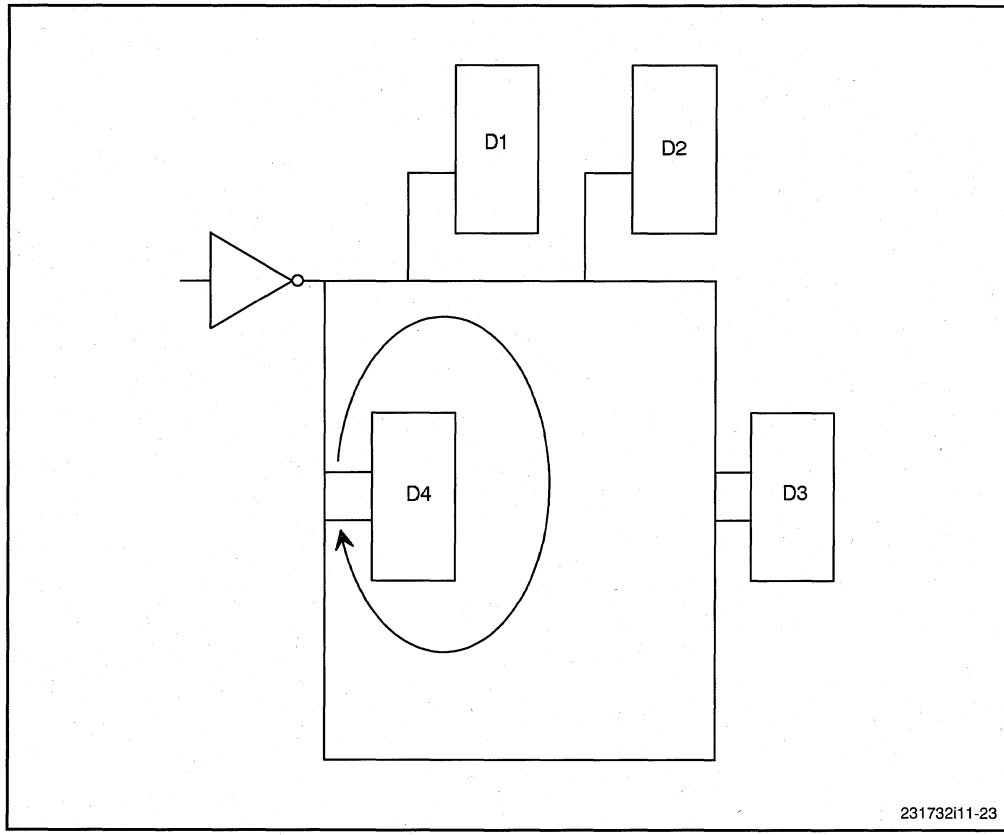


Figure 11-23. Closed Loop Signal Paths are Undesirable

11.4.3.3 ELECTROSTATIC INTERFERENCE

We have discussed two types of coupling, namely inductive and radiative coupling which are responsible for creating electromagnetic interference. A third, known as capacitive coupling, occurs when two equipotential parallel traces are separated by a dielectric and act as a capacitor. According to the standard capacitor equation, the electric field between the two capacitor surfaces varies with the permitivity of the dielectric and with the area of the parallel conductors.

Electrostatic interference (ESI) is caused by this type of coupling. The charge built on one plate of the capacitor induces opposite charge on the other. To minimize the ESI, the following steps should be taken.

- Separate the signal lines so that the effect of capacitive coupling is negated.
- Run a ground line in between the two lines to cancel the electrostatic fields.

For high-frequency designs, a rule of thumb is to include ground planes under each signal layer. Ground planes limit the cross-talk caused by a capacitive coupling between small sections of adjacent layers that are at equipotentials. Additionally, when the width and the thickness of signal lines and their distance from the ground is constant, the effect of capacitive coupling upon impedance remains uniform within (+ −) 5 percent across the board. Using fixed impedance does not reduce capacitive coupling, but it does simplify the modeling of propagation delays and coupling effects. In addition, capacitive coupling can cause interference between layers so the wires should be routed orthogonally on neighboring board layers.

11.4.4 Propagation Delay

The propagation delay of a circuit is a function of the loads on the line, the impedance, and the length of the line segments. The term propagation delay means the propagation delay in the entire circuit, including the delay in the transmission line (which is a function of its dielectric constant).

Also, the printed-circuit interconnections add to the propagation delay of the signal on a wire. These interconnections not only decrease the operating speed of the circuits, but also cause reflections which produce undershoot and overshoot.

When the propagation delays in a circuit are significant, the design must compensate for signal skew. Signal skew occurs when the wire lengths (and thus the propagation delays) between each source and each corresponding load are unequal.

Another negative aspect of propagation delay is that it may generate a race condition. This condition occurs when two signals must reach the same destination within one clock pulse of one another. To avoid race conditions, it is necessary to have the signals travel through same-length traces. If one route is shorter, then the signals will arrive at different times, causing race conditions.

One way to minimize this is by decreasing the lengths of the interconnections. Overall route lengths are shorter in multilayer printed-circuit boards than in double layer boards because ground and power traces are not present. In addition to adding ground planes, a routing program can help to shorten the paths.

The guidelines discussed so far are prominent at higher operating frequencies. Debugging an Intel386 DX processor-based system at higher frequencies requires careful planning of the layout and the physical design. The following sections cover latch-up and thermal characteristics which are system design considerations that stem from the device itself.

11.5 LATCH-UP

Latch-up is a condition in CMOS devices which occurs when V_{CC} becomes shorted to V_{SS}. Much attention has been directed at eliminating this phenomenon under normal conditions. It is necessary for board designers to be aware of latch-up, of its causes, and of how to prevent it.

Latch-up is triggered when the voltage limits on the I/O pins are exceeded, causing the internal PN junction to become forward-biased. The following steps ensure the prevention of latch-up.

- Observe the maximum input voltage rating of I/O pins.
- Never apply power to Intel386 DX processor pin or to any device connected to it before applying power to the Intel386 DX processor.
- Use good termination techniques to prevent overshoot and undershoot.
- Ensure a proper layout to minimize reflections and to reduce noise on the signals.

11.6 CLOCK CONSIDERATIONS

11.6.1 Requirements

For performance at high frequencies, the clock signal (CLK2) for the Intel386 DX microprocessor must be free of noise and within the specifications listed in the Intel386 DX microprocessor data sheet. These requirements can be met by following these guidelines:

- Construct the Clock Generating circuit as shown in Figure 11-24.
- Terminate the CLK2 output to obtain a clean signal.
- Avoid placing too many loads on a single driver and carefully plan the traces to minimize reflection.
- Use an oscilloscope to verify the waveform of CLK2 against the specification in the Intel386 DX microprocessor data sheet.

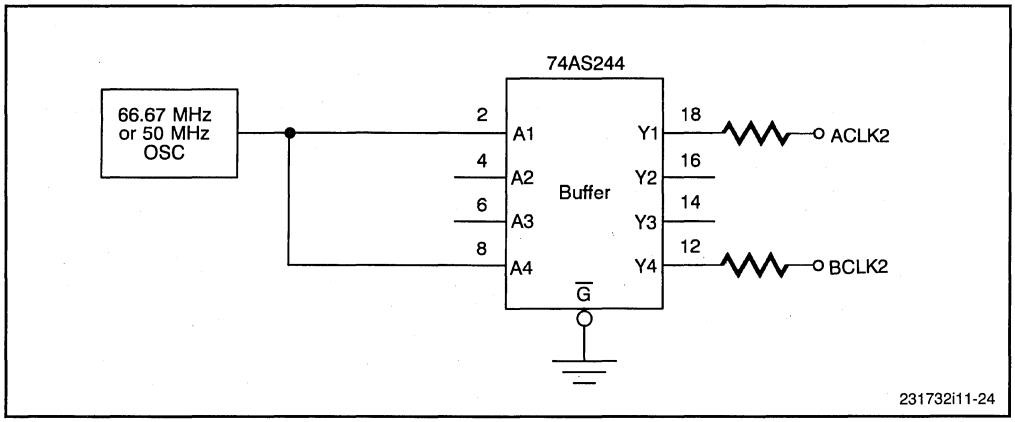


Figure 11-24. Typical Intel386™ DX Microprocessor Clock Circuit

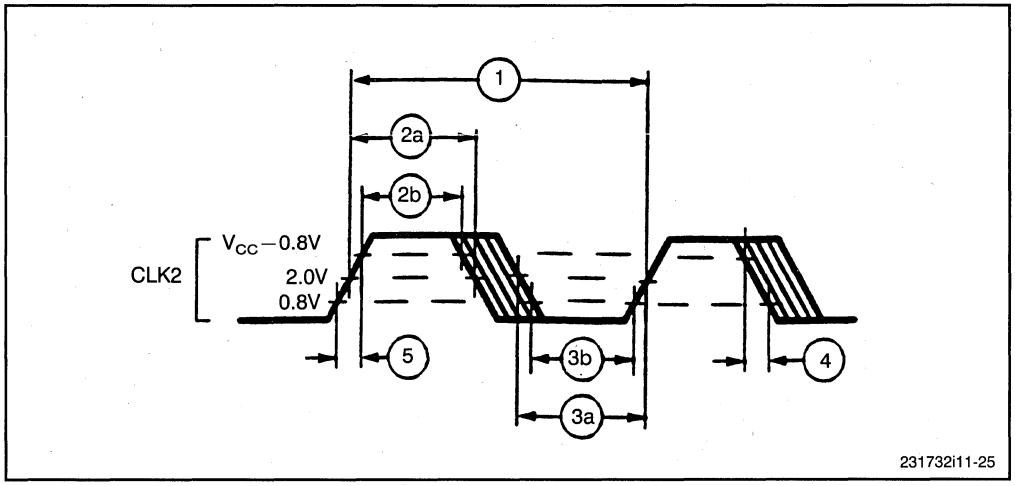


Figure 11-25. CLK2 Timing Diagram

The clock input requirements for Intel386 DX microprocessor systems are more stringent than those for many commonly used TTL devices. The clock timings are shown in Figure 11-25 and Table 11-3.

11.6.2 Routing

Achieving the proper clock routing around a 33-MHz printed circuit board is delicate because a myriad of problems, some of them subtle, can arise if certain design guide lines are not followed. For example fast clock edges cause reflections from high impedance terminations. These reflections can cause significant signal degradations in the systems operating at 33 MHz. This section covers some design guidelines which should be observed to properly layout the clock lines for efficient Intel386 DX processor operation.

Table 11-3. Timing Specifications for CLK2

Symbol	Parameter	25 MHz i386™ DX CPU		Unit	33 MHz i386 DX CPU		Unit	Notes
		Min	Max		Min	Max		
	Operating Frequency	4	25	MHz	8	33.3	MHZ	Half of CLK2 Frequency
t1	CLK2 Period	20	125	ns	15.0	62.5	ns	
t2a	CLK2 High Time	7		ns	6.25		ns	at 2V
t2b	CLK2 High Time	4		ns	4.5		ns	at 3.7V
t3a	CLK2 Low Time	7		ns	6.25		ns	at 2V
t3b	CLK2 Low Time	5		ns	4.5		ns	at 0.8V
t4	CLK2 Fall Time		7	ns		4	ns	3.7V to 0.8V
t5	CLK2 Rise Time		7	ns		4	ns	0.8V to 3.7V

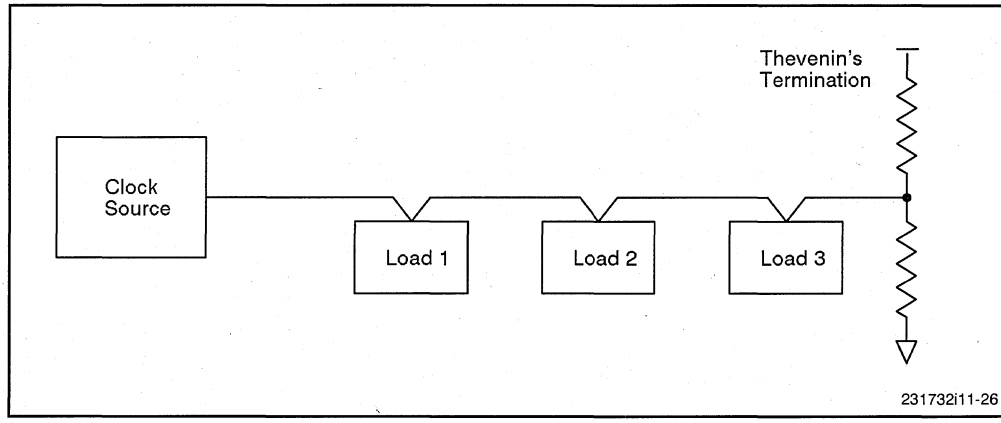


Figure 11-26. Clock Routing

Since the rise/fall time of the clock signal is typically in the range of 2–4 ns, the reflections at this speed could result in undesirable noise and unacceptable signal degradation. The degree of reflection depends on the impedance of the traces of the clock connections. These reflections can be optimized by using proper terminations and by keeping the length of the traces as short as possible. The preferred method is to connect all of the loads via a single trace as shown in Figure 11-26, thus avoiding the extra stubs associated with each load. The loads should be as close to one another as possible. Multiple clock sources should be used for distributed loads.

A less desirable method is the star connection layout in which the clock traces branch to the load as closely as possible (Figure 11-27). In this layout, the stubs should be kept as short as possible. The maximum allowable length of the traces depends upon the frequency and the total fanout, but the length of the traces in the star connection should be equal. Lengths of less than one inch are recommended.

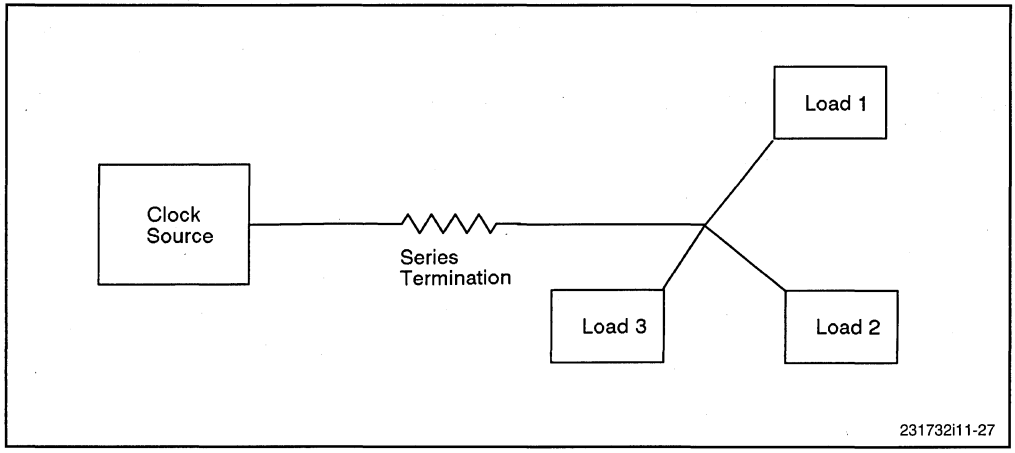


Figure 11-27. Star Connection

11.7 THERMAL CHARACTERISTICS

The thermal specification for the Intel386 DX microprocessor defines the maximum case temperature. This section describes how to ensure that an Intel386 DX microprocessor system meets this specification.

Thermal specifications for the Intel386 DX microprocessor are designed to guarantee a tolerable temperature at the surface of the Intel386 DX microprocessor chip. This temperature (called the junction temperature) can be determined from external measurements using the known thermal characteristics of the package. Two equations for calculating junction temperature are as follows:

$T_j = T_a + (\theta_{ja} * PD)$ and

$T_j = T_c + (\theta_{jc} * PD)$

where

T_j = junction temperature

$_a$ = ambient temperature

T_c = case temperature

θ_{ja} = junction-to-ambient temperature coefficient

θ_{jc} = junction-to-case temperature coefficient

PD = power dissipation (worst-case $I_{cc} * V_{cc}$)

Case temperature calculations offer several advantages over ambient temperature calculations:

- Case temperature is easier to measure accurately than ambient temperature because the measurement is localized to a single point (top center of the package).
- The worst-case junction temperature (T_j) is lower when calculated with case temperature for the following reasons:
  - The junction-to-case thermal coefficient (θ_{jc}) is lower than the junction-to-ambient thermal coefficient (θ_{ja}); therefore, calculated junction temperature varies less with power dissipation (PD).
  - θ_{jc} is not affected by air flow in the system; θ_{ja} varies with air flow.

With the case-temperature specification, the designer can either set the ambient temperature or use fans to control case temperature. Finned heat sinks or conductive cooling may also be used in environments where the use of fans is precluded. To approximate the case temperature for various environments, the two equations above should be combined by setting the junction temperature equal for both, resulting in this equation:

$$T_a = T_c - ((\theta_{ja} - \theta_{jc}) * PD)$$

The current data sheet should be consulted to determine the values of θ_{ja} (for the system's air flow) and ambient temperature that will yield the desired case temperature. Whatever the conditions are, the case temperature is easy to verify.

11.8 DEBUGGING CONSIDERATIONS

This section outlines an approach to building and debugging Intel386 DX microprocessor hardware incrementally. In a short time, a complete Intel386 DX microprocessor-based system can be built and working. This approach does not have to be followed to the letter, but it provides several valuable debugging concepts and useful hints. Use these guidelines in conjunction with the Intel386 DX microprocessor data sheet, which contains detailed information about the Intel386 DX microprocessor.

11.8.1 Hardware Debugging Features

Even before a system is built, debugging can be made easier by planning a suitable environment for the Intel386 DX microprocessor. The Intel386 DX microprocessor board (whether it is a printed circuit board or a wire-wrap board) must have power and ground planes. The user should provide a decoupling capacitor between V_{cc} and GND next to each IC on the board. All Intel386 DX microprocessor V_{cc} and GND pins should be connected individually to the appropriate power or ground plane; multiple power or ground pins should not be daisy-chained.

Room in the system should be included for the following physical features to aid debugging:

- Two switches: one for generating the RESET signal to the Intel386 DX microprocessor and one for tying the READY# signal high (negated).
- Connections for a logic analyzer on major control signals:
  Inputs to the Intel386 DX microprocessor:
  Ready (READY#)
  Next Address (NA#)
  Bus Size 16 (BS16#)
  Data Bus (D0–D31)

  Outputs from the Intel386 DX microprocessor:
  Address Strobe (ADS#)
  Write/Read (W/R#), Data/Control (D/C#), Memory/IO (M/IO#), Lock (LOCK#)
  Address Bus (A2–A31)
  Byte Enable (BE0#–BE3#)

  Logic analyzer connection points should be provided to all Intel386 DX microprocessor address outputs (A2–A31 and BE0#–BE3#) even if there are not enough logic analyzer inputs to accommodate all of them. Initially, only BE0#, BE1#, BE2#, BE3#, and the output of the address decoder circuit should be connected. The single output of an address decoder circuit represents many bits of address information. If the address decoder does not work as expected, more of the logic analyzer inputs should be moved to the Intel386 DX microprocessor address pins.
- Buffers and visual indicators (such as LEDs) for three or four of the critical Intel386 DX microprocessor control signals. A visual indicator for the ADS# output, for example, will light when the system is performing bus cycles.

11.8.2 Bus Interface

During initial debugging, bus-cycle operation should be simplified. The Intel386 DX microprocessor bus interface is flexible enough to be tested in stages. To simplify bus control, the initial testing should be performed with a non-pipelined address. The NA# input should be tied high (negated) to guarantee no address pipelining. The only signals that need to be controlled are the READY# input and the BS16# input.

The READY# input on the Intel386 DX microprocessor lets the user delay the end of any bus cycle for as long as necessary. For each CLK cycle after T2 that READY# is not sampled active, a wait state is added. READY# can be used to provide extra time (wait states) for slow memories or peripherals. Wait state requirements are a function of the device being addressed. Therefore, the address decoder must determine how many wait states, if any, to add to each bus cycle. The address decoder circuit (usually in conjunction with a shift register) must generate the READY# signal when it is time for the bus cycle to end. It is critical for the system to generate the READY# signal; if it does not, the Intel386 DX microprocessor will wait forever for the bus cycle to end.

EPROMs, static RAMs, and peripherals all interface in much the same way. The EPROM interface is the simplest because EPROMs are read-only devices. RAM interfaces must support byte addressability during RAM write cycles. Therefore, RAM write enables for each byte of the 32-bit data bus must be controlled separately.

The BS16# signal must be activated when the current bus cycle communicates over a 16-bit bus. An address decoder circuit can be used to determine if BS16# must be asserted during the current bus cycle.

11.8.3 Simplest Diagnostic Program

To start debugging Intel386 DX microprocessor hardware, the user should make a set of EPROMs containing a simple program, such as a 4-byte diagnostic that loops. Such a program is shown in Figure 11-28. Because the program is four bytes long, it will exercise all 32 bits of the data bus. This program tests only the code prefetch ability of the Intel386 DX microprocessor.

In generating this program, the user should take into account the initial values of the Intel386 DX microprocessor CS register (F000H) and IP register (FFF0H) after reset. The software entry point (label START in Figure 11-28) must match the CS:IP location.

The Intel386 DX microprocessor is initially in Real Mode (the mode that emulates the 8086) after reset. With this simple diagnostic code, it will remain in Real Mode. In Real Mode, CS:IP generates the physical code fetch address directly, without any descriptors, by shifting CS left by 4 bits and adding IP thus:

```
(CS)        F000
(IP)      + FFF0
          -------
          FFFF0
```

```
                        ASSUME CS:SIMPLEST_CODE

0000                    SIMPLEST_CODE  SEGMENT
FFF0                                   ORG 0FFF0H

FFF0 90                 START:         NOP
FFF1 90                                NOP
FFF2 EB FC                             JMP START

FFF4                    SIMPLEST_CODE  ENDS
                                       END
```

Figure 11-28. 4-Byte Diagnostic Program

Also, after reset (until the Intel386 DX microprocessor executes an intersegment JMP or CALL instruction), the physical base address of the code segment is set internally to FFFF0000H. Therefore, the physical address of the first code fetch after reset is always FFFFFFF0H. The simple diagnostic program must begin at this location.

11.8.4 Building and Debugging a System Incrementally

When designing an Intel386 DX microprocessor system, the designer plans the entire system. The core portions must be tested, however, before building the entire system. Beginning with only the Intel386 DX microprocessor and the clock generator, the following steps outline an approach that enables the designer to build up a system incrementally:

1. Install the clock generator. Check that the CLK2 signal is clean. Connect the CLK2 signal to the Intel386 DX microprocessor.

2. Connect the RESET output to the Intel386 DX microprocessor RESET input, and with CLK2 running, check that the state of the Intel386 DX microprocessor during RESET is correct.

3. Tie the Intel386 DX microprocessor INTR, NMI, and HOLD input pins low. Tie the READY# pin high so that the first bus cycle will not end. Reset the Intel386 DX microprocessor, and check that the Intel386 DX microprocessor is emitting the correct signals to perform its first code fetch from physical address FFFFFFF0H. Connect the address latch, and verify that the address is driven at its outputs.

4. Connect the address decoding hardware to the Intel386 DX microprocessor, and check that after reset, the Intel386 DX microprocessor is attempting to select the EPROM devices in which the initial code to be executed will be stored.

5. Connect the data transceiver to the system, and check that after reset, the transceiver control pins are being driven for a read cycle. Connect all address pins of the EPROM sockets, and check that after reset, they are receiving the correct address for the first code fetch cycle.

Intel's iPPS programmer for EPROMs supports dividing an object module into four EPROMs, as is necessary for a 32-bit data bus to EPROM. The programmer can also divide an object module into two EPROMs for a 16-bit data bus to the EPROMs. (In this case, the BS16# input to the Intel386 DX microprocessor must be asserted during all bus cycles communicating with the EPROMs).

When the clock generator, Intel386 DX microprocessor, address decoder, address latch, data transceiver, and READY# generation logic (including wait-state generation) are functioning, the Intel386 DX microprocessor is capable of running the software in the EPROMs. Now the simple debug program described above can be run to see whether the parts of the system work together.

After installing the EPROMs, the READY# line should be tied high (negated) so that the Intel386 DX microprocessor begins its first bus cycle after reset and then continues to add wait states. While the system is in this state, the circuit should be probed to verify signal states, using a voltmeter or oscilloscope probe.

The programmer should check whether the address latches have latched the first address and whether the address decoder is applying a chip-select signal to the EPROMs. The EPROMs should be emitting the first four opcode bytes of the first code to be executed (90H, 90H, EBH, FCH for the 4-byte program of Figure 11-28), and the opcode should be propagating through the data transceivers to the Intel386 DX microprocessor data pins.

Then the READY# input should be connected to the READY# generation logic, the Intel386 DX microprocessor, and the results should be tested when the simple program runs. Because the program loops back on itself, it runs continuously. At this point, the system has progressed to running multiple bus cycles, so a logic analyzer is needed to observe the dynamic behavior of the system.

When the EPROMs programmed with the simple 4-byte diagnostic program are installed and the Intel386 DX microprocessor is executing the code, the LED indicator for ADS# (if included in the system) glows, because ADS# is generated for each bus cycle by the Intel386 DX microprocessor. It is necessary to check that the EPROMs are selected for each code fetch cycle. After system operation is verified with the simple program, more complex programs can be run.

11.8.5 Other Simple Diagnostic Software

Other simple programs can be used to check the other operations the system must perform. The program described here is longer than the 4-byte program illustrated previously; it tests the abilities to write data into RAM and read the data back to the Intel386 DX microprocessor.

This second diagnostic program, shown in Figure 11-29, is also suitable for placing into EPROMS. Because this diagnostic loops back to itself, the ADS# LED should glow continuously, just as it does when running the 4-byte program.

The program in Figure 11-29 is based on the assumption that hardware exists to report whether the data being read back from RAM is correct. This hardware consists of a writable output latch that can display a byte value written to it. The byte value written is a function of the RAM data comparison test. If the data is correct, the byte value written is AAH (10101010); if the data is incorrect, 55H (01010101) is written.

This diagnostic program is not comprehensive, but it does exercise EPROM, RAM, and an output latch to verify that the basic hardware works.

The program is short (45 bytes) to be easily understood. Because it is short and because it loops continuously, a logic analyzer or even an oscilloscope can be used to observe system activity.

```
                PAGE    66,132
;
;       EQUATES
;
LATCH           EQU     0C8H            ;PRESUMES A HARDWARE
                                        ;LATCH IS AT I/O ADDR C8H
GOOD_SIGNAL     EQU     0AAH
BAD_SIGNAL      EQU     055H
;
;       CODE TO VERIFY ABILITY TO WRITE AND READ RAM CORRECTLY
;
                ASSUME CS:INITIAL_CODE
INITIAL_CODE    SEGMENT

                ORG     0F000H          ;THIS IS INTENDED TO BE LOCATED
                                        ;AT PHYSICAL ADDRESS FFFFF000H
TST_LOOP:       MOV     BX, 0000H       ;INITIALIZE BASE REGISTER TO 0
                MOV     DS, BX          ;INITIALIZE DS REGISTER TO 0
                MOV     [BX], 5473H     ;WRITE 5473H TO RAM ADDR 0 AND 1
                MOV     [BX]+2, 2961H   ;WRITE 2961H TO RAM ADDR 2 AND 3
                JMP     READ            ;JMP TO FORCE CPU TO BREAK
                                        ;PRE-FETCH QUEUE AND FETCH THE
                                        ;NEXT INSTRUCTION AGAIN. THIS
                                        ;PREVENTS THE RAM DATA WRITTEN
                                        ;FROM JUST LINGERING ON THE DATA
                                        ;BUS UNTIL THE READ OCCURS
READ:           CMP     [BX], 5473H     ;READ DATA FROM RAM ADDR 0 AND 1
                                        ;AND COMPARE WITH VALUE WRITTEN
                JNE     BADRAM          ;IF DATA DOESN'T MATCH, THEN JMP
                CMP     [BX]+2, 2961H   ;READ DATA FROM RAM ADDR 2 AND 3
                                        ;AND COMPARE WITH VALUE WRITTEN
                JNE     BADRAM          ;IF DATA DOESN'T MATCH, THEN JMP

                MOV     AL, GOOD_SIGNAL
                OUT     LATCH, AL       ;SIGNAL THAT DATA WAS CORRECT
                JMP     TST_LOOP

BADRAM:         MOV     AL, BAD_SIGNAL
                OUT     LATCH, AL       ;SIGNAL THAT DATA WAS BAD
                JMP     TST_LOOP

                ORG     0FFF0H          ;POSITION THE FOLLOWING INSTRUCTION
                                        ;AT OFFSET 0FFF0H
START:          JMP     TST_LOOP        ;INTRA-SEGMENT JUMP (WITHIN
                                        ;SEGMENT)
                                        ;THIS IS INTENDED TO BE THE FIRST
                                        ;INSTRUCTION EXECUTED, SO IT MUST
                                        ;BE LOCATED AT PHYSICAL ADDRESS
                                        ;FFFFFFF0H.
INITIAL_CODE    ENDS
                END
```

Figure 11-29. More Complex Diagnostic Program

This program can be written in ASM86 assembly language. Because the primary purpose of this program is to exercise the system hardware quickly, the Intel386 DX microprocessor is not tested extensively, and Protected Mode is not enabled.

The diagnostic software verifies the ability of the system to perform bus cycles. The Intel386 DX microprocessor fetches code from the EPROMs, implying that EPROM read cycles function correctly. Instructions in the program explicitly generate bus cycles to write and read RAM. The data value read back from RAM is checked for correctness, then a byte (AAH if the data is correct, 55H if it is not) is output to the 8-bit output latch. The program then loops back to its beginning and starts over.

After the source code is assembled, the resulting object code should be as shown in Figure 11-30.

11.8.6 Debugging Hints

The debugging approach described in this section is incremental; it lets the programmer debug the system piece by piece. If even the simple 4-byte program does not run, a logic analyzer can be used to determine where the problem is. At the very least, the Intel386 DX microprocessor should be initiating a code fetch cycle to EPROM.

The Intel386 DX microprocessor stops running only for one of three reasons:

- The READY# signal is never asserted to terminate the bus cycle.
- The HALT instruction is encountered, so the Intel386 DX microprocessor enters a HALT state.
- The Intel386 DX microprocessor encounters a shutdown condition. In Real Mode operation (as in the simple diagnostic program), a shutdown usually indicates that the Intel386 DX microprocessor is reading garbage on the data bus.

If the Intel386 DX microprocessor stops running, the cause can be determined easily if the system contains simple hardware decoders with associated LEDs to visually indicate halt and shutdown conditions. The Intel386 DX microprocessor emits specific codes on its W/R#, D/C#, M/IO#, and address outputs to indicate halt or shutdown. A circuit to decode these signals can be tested by executing a HLT instruction (F4H) to see if the halt LED is turned on. The shutdown LED cannot be tested in the same way, but its decoder is so similar to the halt decoder that if the halt decoder works, the shutdown decoder should also work.

If the shutdown LED comes on and the Intel386 DX microprocessor stops running, the data being read in during code fetch cycles is garbled. The programmer should check the EPROM contents, the wiring of the address path and data path, and the data transceivers. The 4-byte diagnostic program should be used to investigate the system. This program should work before more complex software is used.

If neither the halt LED nor the shutdown LED is on when the Intel386 DX microprocessor stops running, the READY# generation circuit has not activated READY# to complete the bus cycle. The Intel386 DX microprocessor is adding wait states to the

```
                                        PAGE   66,132
                                ;
                                ;       EQUATES
                                ;

= 00C8                          LATCH           EQU    0C8H
= 00AA                          GOOD_SIGNAL     EQU    0AAH
= 0055                          BAD_SIGNAL      EQU    055H

                                ;
                                ;       CODE TO VERIFY ABILITY TO WRITE
                                ;       AND READ RAM CORRECTLY
                                ;
                                        ASSUME CS:INITIAL_CODE
0000                            INITIAL_CODE    SEGMENT

F000                                    ORG    0F000H

F000  BB 0000                   TST_LOOP:       MOV    BX, 0000H
F003  8E DB                                     MOV    DS, BX
F005  C7 07 5473                                MOV    [BX], 5473H
F009  C7 47 02 2961                             MOV    [BX]+2, 2961H
F00E  EB 01 90                                  JMP    READ

F011  81 3F 5473                READ:           CMP    [BX], 5473H
F015  75 0D                                     JNE    BADRAM
F017  81 7F 02 2961                             CMP    [BX]+2, 2961H
F01C  75 06                                     JNE    BADRAM

F01E  B0 AA                                     MOV    AL, GOOD_SIGNAL
F020  E6 C8                                     OUT    LATCH, AL
F022  EB DC                                     JMP    TST_LOOP

F024  B0 55                     BADRAM:         MOV    AL, BAD_SIGNAL
F026  E6 C8                                     OUT    LATCH, AL
F028  EB D6                                     JMP    TST_LOOP

FFF0                                    ORG    0FFF0H
FFF0  E9 F000 R                 START:          JMP    TST_LOOP

FFF3                            INITIAL_CODE    ENDS
                                                END

Warning Severe
Errors  Errors
0       0
```

Figure 11-30. Object Code for Diagnostic Program

cycle, waiting for the READY# signal to go active. The address at the address latch outputs and the states of the W/R#, D/C#, and M/IO# signals should be checked to narrow the investigation to a specific part of the READY# generation circuit. Then the circuit should be investigated with the logic analyzer.

Once the basic system is built and debugged, more software and further enhancements can be added to the system. The incremental approach described applies to these additions. Systematic, step-by-step testing and debugging is the surest way to build a reliable Intel386 DX microprocessor-based system.

Test Capabilities 12

CHAPTER 12
TEST CAPABILITIES

The Intel386 DX microprocessor contains built-in features that enhance its testability. These features are derived from signature analysis and proprietary test techniques. All the regular logic blocks of the Intel386 DX microprocessor, or about half of all its internal devices, can be tested using these built-in features.

The Intel386 DX microprocessor testability features include aids for both internal and board-level testing. This chapter describes these features.

12.1 INTERNAL TESTS

Allowances have been made for two types of internal tests: automatic self-test and Translation Lookaside Buffer (TLB) tests. The automatic self-test is controlled completely by the Intel386 DX microprocessor. The designer needs only to initiate the test and check the results. The TLB tests must be externally developed and applied. The Intel386 DX microprocessor provides an interface that makes this test development simple.

12.1.1 Automatic Self-Test

The Intel386 DX microprocessor can automatically verify the functionality of its three major Programmable Logic Arrays (PLAs) (the Entry Point, Control, and Test PLAs) and the contents of its Control ROM (CROM). The automatic self-test is initiated by setting the BUSY# input active during initialization (as described in Chapter 3). The test result is stored in the EAX register of the Intel386 DX microprocessor.

The self-test progresses as follows (see Figure 12-1):

1. Normal PLA or CROM inputs are disabled.

2. A pseudo-random count sequence, generated by an internal Linear Feedback Shift register (LFSR), provides all possible combinations of PLA and CROM inputs.

3. PLA and CROM outputs for each input combinations are directed to a parallel-load LFSR.

4. Through the action of this LFSR, a signature of all output results is accumulated.

5. After all input combinations have been sequenced, the final contents of the LFSR are XORed with a signature constant stored in the Intel386 DX microprocessor. If the LFSR contents match the signature constant, the result will be all zeroes, indicating functional PLA and CROM.

6. The result is loaded into the EAX register.

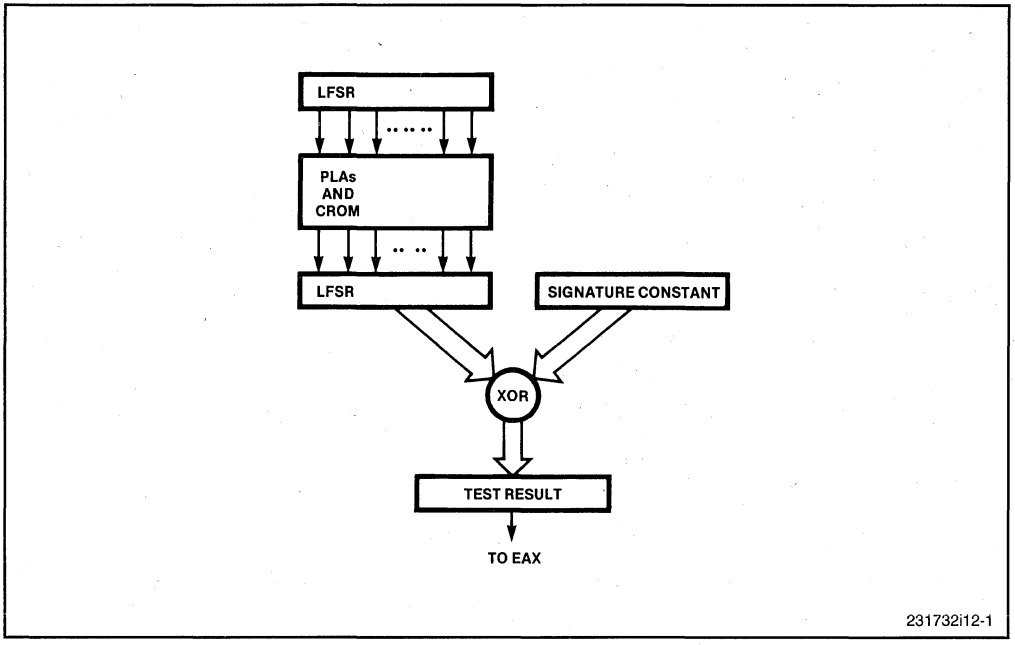


Figure 12-1. Intel386™ DX Microprocessor Self-Test

The self-test provides 100-percent coverage of single-bit faults, which statistically comprise a high percentage of total faults.

12.1.2 Translation Lookaside Buffer Tests

The on-chip Page Descriptor Cache of the Intel386 DX microprocessor stores its data in the TLB. (Cache operation is discussed fully in Chapter 7.) The linear-to-physical mapping values for the most recent memory accesses are stored in the TLB, thus allowing fast translation for subsequent accesses to those locations. The TLB consists of:

- Content-addressable memory (CAM)—holds 32 linear addresses (Page Directory and Page Table fields only) and associated tag bits (used for data protection and cache implementation)
- Random access memory (RAM)—holds the 32 physical addresses (upper 20 bits only) that correspond to the linear addresses in the CAM
- Logic-implements the four-way cache and includes a 2-bit replacement pointer that determines to which of the four sets a new entry is directed during a write to the TLB.

To translate a linear address to a physical address, the Intel386 DX microprocessor tries to match the Page Directory and Page Table fields of the linear address with an entry in the CAM. If a hit (a match) occurs, the corresponding 20 bits of physical address are

retrieved from the RAM and added to the 12 bits of the Offset field of the linear address, creating a 32-bit physical address. If a miss (no match) occurs, the Intel386 DX microprocessor must bring the Page Directory and Page Table values into the TLB from memory.

The Intel386 DX microprocessor provides an interface through which to test the TLB. Two 32-bit test registers of the Intel386 DX microprocessor are used to write and read the contents of the TLB through the MOV TREG, reg and MOV reg, TREG instructions. An Intel386 DX microprocessor program can be used to generate test patterns which are applied to the TLB through automatic test machines or assembly language programs.

The paging mechanism of the Intel386 DX microprocessor must be disabled during a test of the TLB. The internal response is therefore not identical to that of normal operation, but the main functionality of the TLB can be verified.

Test register #6 is used as the command register for TLB accesses; test register #7 is used as the data register. Addresses and commands are written to the TLB through the command register. Data is read from or written to the TLB through the data register.

The two test operations that may be performed on the TLB are:

- Write the physical address contained in the data register and the linear address and tag bits contained in the command register into a TLB location designed by the data register.
- Look up a TLB entry using the linear address and tag bits contained in the command register. If a hit occurs, copy the corresponding physical address into the data register, and set the value of the hit/miss bit in the data register. If a miss occurs, clear the hit/miss bit. In this case, the physical address in the data register is undefined.

A command is initiated by writing to the command register. The command register has the format shown in Figure 12-2 (top). The two possible commands are distinguished by the state of bit 0 in the command register. If bit 0 = 1, a TLB lookup operation is performed. If bit 0 = 0, a TLB write is performed.

The tag bits (not including the linear address) consist of the following:

Bit	Name	Definition
11	Valid (V)	Entry is valid
10	Dirty (D)	Entry has been changed
9	Not Dirty (D#)	Entry has not been changed
8	User (U)	Entry is accessible to User privilege level
7	Not User (U#)	Entry is not accessible to User privilege level
6	Writable (W)	Entry may be changed
5	Not Writable (W#)	Entry may not be changed

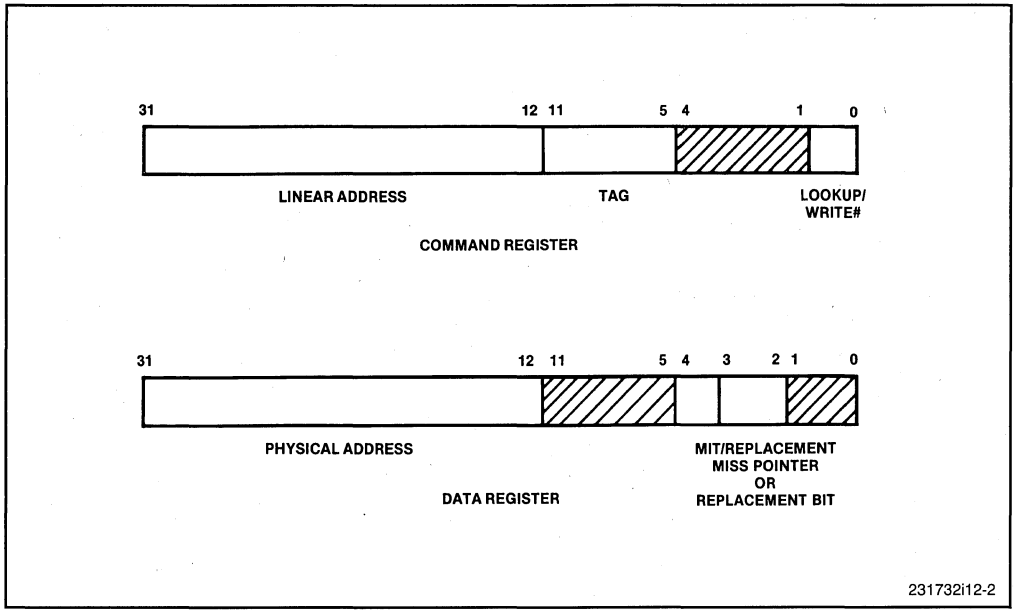


Figure 12-2. TLB Test Registers

The complement of the Dirty, User, and Writable bits are provided to force a hit or miss for TLB lookups. A lookup operation with a bit and its complement both low is forced to be a miss; if both bits are high, a hit is forced. A write operation must always be performed with a bit and its complement bit having opposite values.

The data register has the format shown in Figure 12-2 (bottom). The replacement pointer indicates which of the four sets of the TLB is to receive write data. Its value is changed according to a proprietary algorithm after every TLB hit. For testing, a TLB write may use the replacement pointer value that exists in the TLB, or it may use the value in bits 3 and 2 of the data register. If data register bit 4 = 0, the existing replacement pointer is used. If bit 4 = 1, bits 3 and 2 of the data register are used.

The TLB write operation progresses as follows:

1. The physical address, replacement bit, and replacement pointer value (optional) are written to the data register.

2. The linear address and tag values are written to the command register, as well as a 0 value for bit 0.

It is important not to write the same linear address to more than one TLB entry. Otherwise, hit information returned during a TLB lookup operation is undefined.

The TLB lookup operation progresses as follows:

- The linear address and tag values are written to the command register, as well as a 1 value for bit 0.
- New values for the hit/miss bit and replacement pointer are written to bits 4-2 in the data register. If the hit/miss bit (bit 4) is 1, bits 31-12 contain the physical address from the TLB. Otherwise, bits 31-12 are undefined.

For more information on how to write routines to test the TLB, refer to the *386™ DX Microprocessor Programmer's Reference Manual.*

12.2 BOARD-LEVEL TESTS

For board-level testing, it is often desirable to isolate areas of the board from the interactions of other devices. The Intel386 DX microprocessor can be forced to a state in which all but two of its pins are effectively removed from their circuits. This state is accomplished through the HOLD and HLDA pins.

When the HOLD input of the Intel386 DX microprocessor is asserted, the Intel386 DX microprocessor places all of its outputs except for HLDA in the three-state condition. HLDA is then driven high. The Intel386 DX microprocessor remains in this condition until HOLD is de-asserted. Note that RESET being asserted takes priority over HOLD requests.

The Intel386 DX microprocessor completes its current bus cycle before responding to the HOLD input. Detailed information on HOLD/HLDA response is given in Chapter 3.

Local Bus Control PLD Descriptions

A

APPENDIX A
LOCAL BUS CONTROL PLD DESCRIPTIONS

The bus controller is implemented in two PLDs. One PLD (called IOPLD1) follows the Intel386 DX microprocessor status lines and initiates I/O and EPROM accesses. The second PLD (IOPLD2) contains the bus cycle tracking state machine and determines the number of wait states for I/O system accesses.

EPROMs and peripherals are usually arranged with 16-bit data bus interfaces. This subsystem asserts BS16# for all accesses to the I/O and EPROMs. Because all accesses are BS16#, pipelined cycles cannot be requested. This system can coexist with a subsystem that uses pipelining, provided the pipelined system keeps NA# asserted until the end of the cycle. The DRAM subsystem described in Chapter 6 will meet this requirement.

The PLDs are clocked by CLK2. They could also be clocked by CLK. Using CLK2 has the following advantages over using CLK:

- The skew from clock to command signal is reduced, so higher performance is possible with slower devices.
- The Intel386 DX microprocessor ADS# and READY# signals can be sampled directly.
- The PLD can provide delays in 25 nanosecond, rather than 50 nanosecond, increments.

The advantages of using CLK to clock the PLDs are as follows:

- A slower PLD device could be used.
- One PLD input is saved because only CLK, rather than CLK and CLK2, is needed.

Because CLK2 is used to clock the PLDs, the choice of PLDs is limited by the frequency of the processor.

IOPLD1 FUNCTIONS

IOPLD1 is implemented as two state machines. The first state machine enables the data transceivers between the processor and the peripherals. The transceivers remain active until the end of the bus cycle. The second state machine determines the type of cycle that has been initiated. Once a cycle has been started, the state machine waits for the TIMEDLY# signal from IOPLD2 before continuing the cycle.

IOPLD2 FUNCTIONS

The IOPLD2 has two functions. First, the PLD contains the bus cycle tracking state machine. The BUSCYC# signal is used by IOPLD1 for determining the start of bus cycles. Second, the PLD counter determines the number of wait states from IOPLD1 initiating a bus cycle until the time it returns TIMEDLY# to IOPLD1. If peripheral

devices requiring different numbers of wait states are in the system, the TIMEDLY# state machine must check the chip select wait state pins (CS1WS#, CS3WS#, CS5WS#). These signals are generated from the mapping of the I/O devices. The 8259A interface has not been built or tested.

PLD EQUATIONS

The equations for IOPLD1, IOPLD2, and the RESET/CLOCK PLDs are shown in Figures A-1, A-2, and A-3, respectively. These equations are shown in a high-level PLD language (ABEL, by Data I/O) that allows the PLD to be described as a series of states rather than equations. This language frees the designer of the tedious task of implementing the state machine and reducing the logical equations manually. The language saves time not only in the initial design, but also in debugging the state machines. The automated term reduction of the high-level PLD language allows the designer to explore many implementations quickly, which is a useful feature for complex PLD designs. The PLD equations generated by ABEL are included to allow the conversion to a different PLD programming language.

```
module iopld1; flag '-r3'; flag '-ul';
title 'eprom/io controller intel corporation'

''This 85C220 generates IORD#, IOWR#, EPRD#, and INTA# for
''the peripheral subsystem. It decodes and responds to
''the following bus cycles: i/o read, i/o write, memory
''read (with A31 high), interrupt acknowledge, halt,
''and shutdown.

U7 device 'E0320';
h,l,c,x = 1,0,.C.,.X.;

gnd pin 10;
vcc pin 20;
oe pin 11;

clk2 pin 1;        ''80386 CLK2
clk pin 2;         ''low during phase 1, high during phase 2
na pin 3;          ''low to begin bus cycles
mio pin 4;         ''high during memory cycles, low for i/o
wr pin 5;          ''high for write, low for read
dc pin 6;          ''high for data cycles, low for control cycles
pa31 pin 7;        ''processor address A31
timedly pin 8;     ''time delay input
buscyc pin 9;      ''low during active bus cycles

recv pin 12;       ''low during float and recovery
iord pin 13;       ''low to read io
iowr pin 14;       ''low to write to
eprd pin 15;       ''low to read eproms
inta pin 16;       ''low for interrupt acknowledge
trioen pin 18;     ''low to enable io transceiver
iordy pin 19;      ''low to indicate ready

idle     = [1,1,1,1,1,1];
ioread1  = [0,1,1,1,1,1];
ioread2  = [0,1,1,1,0,1];
iowrite1 = [1,0,1,1,1,1];
iowrite2 = [1,1,1,1,0,1];
epread1  = [1,1,0,1,1,1];
epread2  = [1,1,0,1,0,1];
intak1   = [1,1,1,0,1,1];
intak2   = [1,1,1,0,0,1];
recover  = [1,1,1,1,1,0];

''io transceiver enable

state_diagram [trioen];
state 1:   ''idle
    if (na & !buscyc & !mio & !pa31 & recv & clk) then 0
    else if (na & !buscyc & mio & pa31 & recv & clk) then 0
    else 1;

state 0:  ''enable transceiver between processor and peripherals
    if (!iordy & clk) then 1
    else if (buscyc & clk) then 1
    else 0;
```

Figure A-1. IOPLD1 Equations

```
''io state machine

state_diagram [iord,iowr,eprd,inta,iordy,recv];
state idle:
  case na & !buscyc & pa31 & !wr & clk: epread1;
       na & !buscyc & !pa31 & !mio & dc & wr & clk: iowrite1;
       na & !buscyc & !pa31 & !mio & dc & !wr & clk: ioread1;
       na & !buscyc & !pa31 & !mio & !dc & !wr & clk: intak1;
       na & !buscyc & mio & !dc & wr & clk: iowrite2; ''halt
  endcase;

state epread1:  if (!timedly & clk) then epread2 else epread1;
state epread2:  if (clk) then idle else epread2;
state iowrite1: if (!timedly & clk) then iowrite2 else iowrite1;
state iowrite2: if (!mio & clk) then recover
                else if (mio & clk) then idle
                else iowrite2;
state ioread1:  if (!timedly & clk) then ioread2 else ioread1;
state ioread2:  if (clk) then recover else ioread2;
state intak1:   if (!timedly & clk) then intak2 else intak1;
state intak2:   if (clk) then recover else intak2;
state recover:  if (!timedly & clk) then idle else recover;

test_vectors ([clk2,clk,na,mio,wr,dc,pa31,timedly,buscyc,oe] ->
              [iord,iowr,eprd,inta,iordy,recv]);

    [c,h,h,h,h,h,h,h,h,l] -> [h,h,h,h,h,h];    ''idle
    [c,h,h,h,h,h,h,h,h,l] -> [h,h,h,h,h,h];    ''idle
    [c,h,h,h,h,h,h,h,h,l] -> [h,h,h,h,h,h];    ''idle

    [c,h,h,h,l,l,h,h,l,l] -> [h,h,l,h,h,h];    ''eprom read
    [c,h,h,h,l,l,h,h,l.l] -> [h,h,l,h,h,h];    ''eprom read
    [c,h,h,h,l,l,h,h,l,l] -> [h,h,l,h,h,h];    ''eprom read
    [c,h,h,h,l,l,h,h,l,l] -> [h,h,l,h,h,h];    ''eprom read
    [c,h,h,h,l,l,h,l,l,l] -> [h,h,l,h,l,h];    ''eprom read
    [c,h,h,h,h,h,h,h,h,l] -> [h,h,h,h,h,h];    ''idle

    [c,h,h,l,l,h,l,h,l,l] -> [l,h,h,h,h,h];    ''io read
    [c,h,h,l,l,h,l,h,l,l] -> [l,h,h,h,h,h];    ''io read
    [c,h,h,l,l,h,l,h,l,l] -> [l,h,h,h,h,h];    ''io read
    [c,h,h,l,l,h,l,h,l,l] -> [l,h,h,h,h,h];    ''io read
    [c,h,h,l,l,h,l,l,l,l] -> [l,h,h,h,h,l,h];  ''io read
    [c,h,h,h,h,h,h,h,h,l] -> [h,h,h,h,h,l];    ''recovery
    [c,h,h,h,h,h,h,h,h,l] -> [h,h,h,h,h,l];    ''recovery
    [c,h,h,h,h,h,h,l,h,l] -> [h,h,h,h,h,h];    ''recovery

    [c,h,h,l,h,h,l,h,l,l] -> [h,l,h,h,h,h];    ''io write
    [c,h,h,l,h,h,l,h,l,l] -> [h,l,h,h,h,h];    ''io write
    [c,h,h,l,h,h,l,h,l,l] -> [h,l,h,h,h,h];    ''io write
    [c,h,h,l,h,h,l,h,l,l] -> [h,l,h,h,h,h];    ''io write
    [c,h,h,l,h,h,l,l,l,l] -> [h,h,h,h,l,h];    ''io write
    [c,h,h,l,h,h,h,h,h,l] -> [h,h,h,h,h,l];    ''recovery
    [c,h,h,h,h,h,h,h,h,l] -> [h,h,h,h,h,l];    ''recovery
    [c,h,h,h,h,h,h,l,h,l] -> [h,h,h,h,h,h];    ''recovery

    [c,h,h,l,l,l,l,h,l,l] -> [h,h,h,l,h,h];    ''interrupt ack
    [c,h,h,l,l,l,l,h,l,l] -> [h,h,h,l,h,h];    ''interrupt ack
    [c,h,h,l,l,l,l,h,l,l] -> [h,h,h,l,h,h];    ''interrupt ack
    [c,h,h,l,l,l,l,h,l,l] -> [h,h,h,l,h,h];    ''interrupt ack
    [c,h,h,l,l,l,l,l,l,l] -> [h,h,h,l,l,h];    ''interrupt ack
    [c,h,h,h,h,h,h,h,h,l] -> [h,h,h,h,h,l];    ''recovery
    [c,h,h,h,h,h,h,h,h,l] -> [h,h,h,h,h,l];    ''recovery
    [c,h,h,h,h,h,h,l,h,l] -> [h,h,h,h,h,h];    ''recovery

    [c,h,h,h,h,l,l,h,l,l] -> [h,h,h,h,l,h];    ''halt or shutdown
```

Figure A-1. IOPLD1 Equations (Contd.)

```
    [c,h,h,h,h,h,h,h,h,l] -> [h,h,h,h,h,h];    ''idle
    [c,h,h,h,h,h,h,h,h,l] -> [h,h,h,h,h,h];    ''idle

    end iopld1;

eprom/io controller intel corporation
Equations for Module iopld1

Device U7

- Reduced Equations:

    !trioen := (!clk & !trioen
                # !buscyc & iordy & !trioen
                # !buscyc & clk & mio & na & pa31 & recv & trioen
                # !buscyc & clk & !mio & na & !pa31 & recv & trioen);

    !iord := (!clk & eprd & inta & !iord & iowr & recv
              # eprd & inta & !iord & iordy & iowr & recv
              # !buscyc & clk & dc & eprd & inta & iordy & iowr & !mio &
                na & !pa31 & recv & !wr);

    !iowr := (!clk & eprd & inta & iord & iordy & !iowr & recv
              # eprd & inta & iord & iordy & !iowr & recv & timedly
              # !buscyc & clk & dc & eprd & inta & iord & iordy & iowr &
                !mio & na & !pa31 & recv & wr);

    !eprd := (!clk & !eprd & inta & iord & iowr & recv
              # !eprd & inta & iord & iordy & iowr & recv
              # !buscyc & clk & inta & iord & iordy & iowr & na & pa31 &
                recv & !wr);

    !inta := (!clk & eprd & !inta & iord & iowr & recv
              # eprd & !inta & iord & iordy & iowr & recv
              # !buscyc & clk & !dc & eprd & iord & iordy & iowr & !mio &
                na & !pa31 & recv & !wr);

    !iordy := (!clk & eprd & iord & !iordy & iowr & recv
               # clk & eprd & !inta & iord & iordy & iowr & recv & !timedly
               # !clk & eprd & inta & !iordy & iowr & recv
               # clk & eprd & inta & !iord & iordy & iowr & recv & !timedly
               # clk & eprd & inta & iord & iordy & !iowr & recv & !timedly
               # !clk & inta & iord & !iordy & iowr & recv
               # clk & !eprd & inta & iord & iordy & iowr & recv & !timedly
               # !buscyc & clk & !dc & eprd & inta & iord & iordy & iowr &
                 mio & na & recv & wr);

    !recv := (!clk & eprd & inta & iord & iordy & iowr & !recv
              # eprd & inta & iord & iordy & iowr & !recv & timedly
              # clk & eprd & !inta & iord & !iordy & iowr & recv
              # clk & eprd & inta & !iord & !iordy & iowr & recv
              # clk & eprd & inta & !iordy & iowr & !mio & recv);
```

Figure A-1. IOPLD1 Equations (Contd.)

```
module iopld2; flag '-r3';
title 'eprom/io wait state timer and bus cycle tracking'
"This P20R8 determines the number of wait states and recovery
"states for i/o reads, i/o writes, eprom reads, and interrupt
"acknowledge cycles. Choose the number of wait states for each
"peripheral by the chip selects

U8    device 'p20r8';
h,l,c,x = 1,0,.C.,.X.;

oe pin 13;

clk2 pin 1;         "80386 CLK2
clk pin 2;          "low during phase 1, high during phase 2
iord pin 3;         "low to read io
iowr pin 4;         "low to write io
eprd pin 5;         "low to read eproms
inta pin 6;         "low for interrupt acknowledge
recv pin 7;         "low during float and recovery
ads pin 8;          "low to begin bus cycles
ready pin 9;        "low to end bus cycles
cs1ws pin 10;       "from decoder: 1 wait state chip select
cs3ws pin 11;       "from decoder: 3 wait state chip select
cs5ws pin 14;       "from decoder: 5 wait state chip select

wtcnt0 pin 22;      "wait state counter bit 0
wtcnt1 pin 21;      "wait state counter bit 1
wtcnt2 pin 20;      "wait state counter bit 2
timedly pin 15;     "time delay output
aleio pin 16;       "high to make address latch transparent
buscyc pin 17;      "low during active bus cycles
pipecyc pin 18;     "low after pipelined bus cycles


idle  = [1,1,1,1];
time1 = [1,1,1,0];
time2 = [1,1,0,1];
time3 = [1,1,0,0];
time4 = [1,0,1,1];
time5 = [1,0,1,0];
time6 = [1,0,0,1];
time7 = [1,0,0,0];
timeup= [0,1,1,1];

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""


"io address latch enable

equations !aleio := (!iord & clk) #
                    (!iowr & clk) #
                    (!inta & clk) #
                    (!aleio & !clk);

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
```

Figure A-2. IOPLD2 Equations

```
"io cycle timer

state_diagram [timedly,wtcnt2,wtcnt1,wtcnt0];
state idle
     if ((!iord # !iowr # !inta) & clk) then time3
     else if (!eprd & clk) then time2
     else if (!recv & clk) then time2
     else idle;

state time7: if clk then time6 else time7;
state time6: if clk then time5 else time6;
state time5: if clk then time4 else time5;
state time4: if clk then time3 else time4;
state time3: if clk then time2 else time3;
state time2: if clk then time1 else time2;
state time1: if clk then timeup else time1;
state timeup: if (iord & iowr & eprd & inta & clk) then idle else timeup;

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

"bus cycle tracking

state_diagram [buscyc,pipecyc]
state [1,1]:  "idle
     if (!ads & clk) then [0,1]
     else [1,1];

state [0,1]:  "active
     if (!ready & ads & clk) then [1,1]
     else if (!ready & !ads & clk) then [1,0]
     else [0,1];

state [1,0]:  "pipelined
     if (clk) then [0,1]
     else [1,0];

state [0,0]:  "illegal
     goto [1,1];

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
```

Figure A-2. IOPLD2 Equations (Contd.)

```
test_vectors ([clk2,clk,iord,iowr,eprd,inta,recv] → [timedly]);

     [c,h,h,h,h,h,h] → [h];     "idle
     [c,h,h,h,h,h,h] → [h];     "idle
     [c,h,h,h,h,h,h] → [h];     "idle
     [c,h,h,h,h,h,h] → [h];     "idle

     [c,h,l,h,h,h,h] → [h];     "io read
     [c,h,l,h,h,h,h] → [h];     "io read
     [c,h,l,h,h,h,h] → [h];     "io read
     [c,h,l,h,h,h,h] → [h];     "io read
     [c,h,h,h,h,h,h] → [h];     "idle

     [c,h,l,h,h,h,h] → [h];     "interrupt acknowledge
     [c,h,l,h,h,h,h] → [h];     "interrupt acknowledge
     [c,h,l,h,h,h,h] → [h];     "interrupt acknowledge
     [c,h,l,h,h,h,h] → [l];     "interrupt acknowledge
     [c,h,h,h,h,h,l] → [h];     "idle

     [c,h,h,l,h,h,h] → [h];     "io write
     [c,h,h,l,h,h,h] → [h];     "io write
     [c,h,h,l,h,h,h] → [h];     "io write
     [c,h,h,l,h,h,h] → [l];     "io write
     [c,h,h,h,h,h,h] → [h];     "idle

     [c,h,h,h,l,h,h] → [h];     "eprom read
     [c,h,h,h,l,h,h] → [h];     "eprom read
     [c,h,h,h,l,h,h] → [l];     "eprom read
     [c,h,h,h,h,h,h] → [h];     "idle


test_vectors ([clk2,clk,ads,ready] → [buscyc,pipecyc])

     [c,l,h,h] → [x,h];    "idle-busy-idle
     [c,h,h,l] → [x,h];
     [c,l,h,h] → [h,h];
     [c,h,l,h] → [l,h];
     [c,l,h,l] → [l,h];
     [c,h,h,l] → [h,h];

     [c,l,l,h] → [h,h];    "idle-busy-pipe-busy
     [c,h,l,h] → [l,h];
     [c,l,h,l] → [l,h];
     [c,h,l,l] → [h,l];
     [c,l,l,h] → [h,l];
     [c,h,h,h] → [l,h];

     [c,h,h,l] → [h,h];    "idle-busy-busy-idle
     [c,l,h,h] → [h,h];
     [c,h,l,h] → [l,h];
     [c,l,l,h] → [l,h];
     [c,h,h,h] → [l,h];
     [c,l,h,h] → [l,h];
     [c,h,h,h] → [l,h];
     [c,l,h,c] → [l,h];
     [c,h,h,l] → [h,h];
     [c,l,h,h] → [h,h];


                end iopal2;
```

Figure A-2. IOPLD2 Equations (Contd.)

```
eprom/io wait state timer and bus cycle tracking
Equations for Module iotime

Device U8

- Reduced Equations:

    aleio := !(!aleio & !clk # clk & !inta # clk & !iord # clk &
                !iowr);

    timedly := !(!clk & !timedly & wtcnt0 & wtcnt1 & wtcnt2
                 # clk & timedly & !wtcnt0 & wtcnt1 & wtcnt2
                 # !eprd & !timedly & wtcnt0 & wtcnt1 & wtcnt2
                 # !inta & !timedly & wtcnt0 & wtcnt1 & wtcnt2
                 # !iord & !timedly & wtcnt0 & wtcnt1 & wtcnt2
                 # !iowr & !timedly & wtcnt0 & wtcnt1 & wtcnt2);

    wtcnt2 := !(!clk & timedly & !wtcnt2
                # timedly & !wtcnt0 & !wtcnt2
                # timedly & !wtcnt1 & !wtcnt2);

    wtcnt1 := !(!clk & timedly & !wtcnt1
                # timedly & !wtcnt0 & !wtcnt1
                # clk & timedly & wtcnt0 & wtcnt1 & !wtcnt2
                # clk & !inta & timedly & wtcnt0 & wtcnt1
                # clk & !iord & timedly & wtcnt0 & wtcnt1
                # clk & !iowr & timedly & wtcnt0 & wtcnt1
                # clk & !eprd & timedly & wtcnt0 & wtcnt1
                # clk & !recv & timedly & wtcnt0 & wtcnt1);

    wtcnt0 := !(!clk & timedly & !wtcnt0
                # clk & timedly & wtcnt0 & !wtcnt1
                # clk & timedly & wtcnt0 & !wtcnt2
                # clk & !inta & timedly & wtcnt0
                # clk & !iord & timedly & wtcnt0
                # clk & !iowr & timedly & wtcnt0);

    buscyc := !(buscyc & clk & !pipecyc
                # !ads & buscyc & clk
                # !buscyc & !clk & pipecyc
                # !buscyc & pipecyc & ready);

    pipecyc := !(buscyc & !clk & !pipecyc
                 # !ads & !buscyc & clk & pipecyc & !ready);
```

Figure A-2. IOPLD2 Equations (Contd.)

```
module clock; flag '-r3'; flag '-ul';
title 'clock generator intel corporation'

''This 85C220 divides the doulbe frequency CLK2 input
''by two to generate a single frequency CLK. It also
''provides synchronous reset outputs from an asynch-
''ronous reset input. The lowest two bits of the DRAM
''refresh timer are included in this PLD.

U2 device 'E0320';
h,l,c,x = 1,0,.C.,.X.;

gnd pin 10;
vcc pin 20;
oe pin 11;

clk2 pin 1;        "80386 CLK2
res pin 2;         "asynchronous reset input
refreq pin 3;      "low to high transition requests refresh cycle

clk pin 12;        "low during phase 1, high during phase 2
reseth pin 13;     "high during reset, changes during phase 2
resetl pin 14;     "low during reset, changes during phase 1
tc0 pin 15;        "bit 0 of refresh timer
tc1 pin 16;        "bit 1 of refresh timer
tcout pin 17;      "carry from bits 0 and 1 of refresh timer
tload pin 18;      "low to load refresh timer
resync pin 19;     "flip flop in reset synchronizer

rtime0 = [0,0];
rtime1 = [0,1];
rtime2 = [1,0];
rtime3 = [1,1];

"clk2 divide by two
equations clk := !clk;

"reset synchronizer
equations resync := (res & !clk) # (resync & clk);
equations reseth := (resync & !clk) # (reseth & clk);
equations resetl := !reseth;

"lowest 2 bits of refresh timer
state_diagram [tc1,tc0];
state rtime0: if (clk) then rtime1 else rtime0;
state rtime1: if (clk) then rtime2 else rtime1;
state rtime2: if (clk) then rtime3 else rtime2;
state rtime3: if (clk) then rtime0 else rtime3;

"refresh timer carry and load
state_diagram [tcout,tload]
state [0,1]:   "idle
    if (tc1 & tc0 & !refreq & !clk) then [1,1]
    else if (tc1 & tc0 & refreq & !clk) then [0,0]
    else [0,1];

state [1,1]:   "increment refresh timer
    goto [0,1];

state [0,0]:   "load refresh timer
    goto [0,1];

state [1,0]:   "illegal
    goto [1,1];
```

Figure A-3. RESET/CLOCK PLD Equations

```
test_vectors ([clk2,refreq] → [clk,tc1,tc0,tcout,tload])

     [c,l] → [l,l,l,l,h];          "count 0
     [c,l] → [h,l,l,l,h];          "count 0
     [c,l] → [l,l,h,l,h];          "count 1
     [c,l] → [h,l,h,l,h];          "count 1
     [c,l] → [l,h,l,l,h];          "count 2
     [c,l] → [h,h,l,l,h];          "count 2
     [c,l] → [l,h,h,l,h];          "count 3
     [c,l] → [h,h,h,h,h];          "count 3, carry
     [c,l] → [l,l,l,l,h];          "count 0
     [c,l] → [h,l,l,l,h];          "count 0
     [c,l] → [l,l,h,l,h];          "count 1
     [c,l] → [h,l,h,l,h];          "count 1
     [c,l] → [l,h,l,l,h];          "count 2
     [c,l] → [h,h,l,l,h];          "count 2
     [c,l] → [l,h,h,l,h];          "count 3
     [c,l] → [h,h,h,h,h];          "count 3, carry
     [c,h] → [l,l,l,l,h];          "count 0
     [c,h] → [h,l,l,l,h];          "count 0
     [c,h] → [l,l,h,l,h];          "count 1
     [c,h] → [h,l,h,l,h];          "count 1
     [c,h] → [l,h,l,l,h];          "count 2
     [c,h] → [h,h,l,l,h];          "count 2
     [c,h] → [l,h,h,l,h];          "count 3
     [c,h] → [h,h,h,l,l];          "count 3, load
     [c,h] → [l,l,l,l,h];          "count 0
     [c,h] → [h,l,l,l,h];          "count 0


test_vectors ([clk2,res] → [clk,resync,reseth,reset1])

     [c,h] → [l,x,x,x];
     [c,h] → [h,h,x,x];
     [c,h] → [l,h,x,x];
     [c,h] → [h,h,h,x];
     [c,h] → [l,h,h,l];
     [c,h] → [h,h,h,l];
     [c,l] → [l,h,h,l];          "res input negated
     [c,l] → [h,l,h,l];
     [c,l] → [l,l,h,l];
     [c,l] → [h,l,l,l];"reseth falling edge occurs during phase 2
     [c,l] → [l,l,l,h];"reset1 rising edge occurs during phase 1
     [c,l] → [h,l,l,h];
end clock;
```

Figure A-3. RESET/CLOCK PLD Equations (Contd.)

```
clock generator     intel corporation
Equations for Module clock

Device U2

- Reduced Equations:

    clk := !(clk);

    resync := !(!clk & !res # clk & !resync);

    reseth := !(!clk & !resync # clk & !reseth);

    resetl := !(reseth);

    tc1 := !(clk & tc0 & tc1 # !clk & !tc1 # !tc0 & !tc1);

    tc0 := !(clk & tc0 # !clk & !tc0);

    tcout := !(!tcout & !tload
              # tcout & tload
              # clk & tload
              # !tc0 & tload
              # !tc1 & tload
              # refreq & tload);

    tload := !(!clk & refreq & tc0 & tc1 & !tcout & tload);
```

Figure A-3. RESET/CLOCK PLD Equations (Contd.)

DRAM PLD Descriptions

B

APPENDIX B
DRAM PLD DESCRIPTIONS

This section describes the inputs, outputs, and functions of each of the PLDs in the DRAM design described in Chapter 6. The terms Start-Of-Phase and Middle-Of-Phase used to describe PLD input sampling times refer to the Intel386 DX microprocessor internal CLK phase and are defined in Figure B-1.

The setup, hold, and propagation delay times for each PLD input and output can be determined from the PLD data sheets. In a few cases, the setup and hold times during certain events must be violated; in these cases, the PLD equations mask these inputs so they are not sampled. Because the states are fully registered and because inputs are masked when their setup or hold times cannot be guaranteed, no hazards exist.

DRAM PLDs

The DRAM PLDs determine when to run a new DRAM cycle and tracks the state of the DRAM through the cycle. The inputs sample DRAM requests from the processor (or any other bus master) as well as requests for refresh. The outputs store state information and generate the two RAS signals and two multiplexer control signals.

The equations for the DRAMP1 are shown in Figure B-2. The DRAMP1 is implemented in a 20R8 PLD. The equations for DRAMP2 are shown in Figure B-3. The DRAMP2 is implemented in an 85C220 EPLD.

For a PLD to change states on each clock edge, its maximum clock to output delay plus its minimum setup time must be less than the time between clock edges.

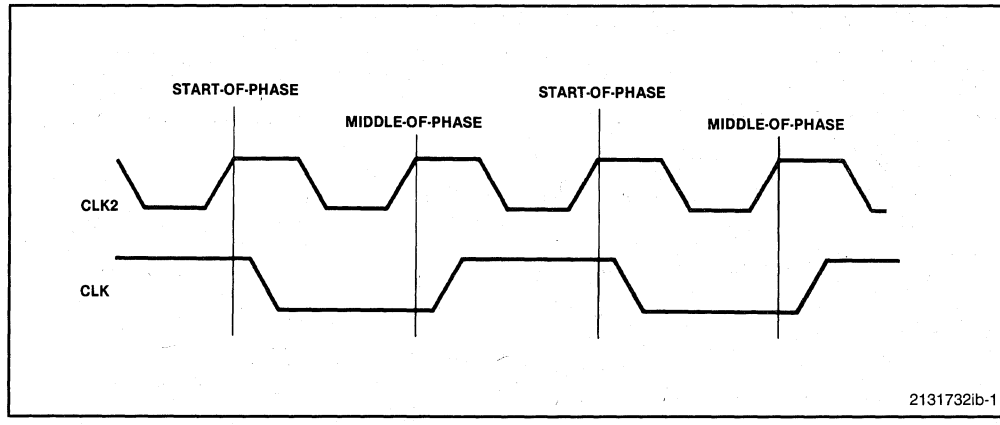


Figure B-1. PLD Sampling Edges

```
module     DRAM_CONTROLLER_FOR_80386 flag '-r3','-t2"
title '80386 Interleaved DRAM Controller pal-1 pipelined 1ws'
U33    device       'P20R8';

" Constants:
     DN              =      1;
     OFF             =      0;
     h               =      1;
     l               =      0;
     x               =      .X.;      " ABEL 'don't care' symbol
     c               =      .C.;      " ABEL 'clocking input' symbol

"Pin names:
  "Control
    clk2          pin 1;   "80386 Double-Frequency system clock
    oe            pin 13; "Output Enable (tie active low)

  "Inputs
    pclk          pin 2;   "processor/phase clock
    ads           pin 3;   "ads from 80386
    mio           pin 4;   "M/IO from 80386
    pa2           pin 5;   "A2 from 80386
    iready        pin 6;   "inverse of Ready into the 80386
    ras0p         pin 7;   "RAS0 precharged from PLD2
    sel1          pin 9;   "High for DRAM address
    ras1p         pin 8;   "RAS1 precharged from PLD2
    refin         pin 10; "request for start refresh
    reset         pin 11; " RESET from clock circuitry
    sel2          pin 14; " High for DRAM address cycles

  "Outputs
    ras0          pin 17; "DRAM RAS output for bank 0
    ras1          pin 18; "DRAM RAS output for bank 1
    rowsel        pin 20; "DRAM address mux select
    muxoe         pin 19; "DRAM address mux output enable
    dramstart     pin 15; "DSTART for PLD2 and refresh eq.
    refadroe      pin 16; " output enable for refresh address PLD
    pipecyc       pin 21; " low during pipe cycle
    buscyc        pin 22; " low when bus cycle active

        idle      =        [1,1];
        startdram =        [0,1];
        col_den   =        [0,0];
        col_den2  =        [1,0];

        ras0idle = [1];
        ras1idle = [1];
        ras0act = [0];
        ras1act = [0];

        select = [sel1 & sel2] ;

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
```

Figure B-2. DRAMP1 PLD Equations

```
"bus cycle tracking

state_diagram [buscyc,pipecyc]
state [1,1]:  "idle
     if (!ads & pclk) then [0,1]
     else [1,1];

state [0,1]:  "active
     if (iready & ads & pclk) then [1,1]
     else if (iready & !ads & pclk) then [1,0]
     else [0,1];

state [1,0]:  "pipelined
     if (pclk) then [0,1]
     else [1,0];

state [0,0]:  "illegal
     goto [1,1];

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

start_diagram [dramstart,rowsel];

state idle:  "wait for DRAM or refresh cycles
     if  (pclk & !ads & mio & select & !muxoe & !pa2 & ras0p &
!refin ) then startdram
     else if (pclk & !ads & mio & select & !muxoe & pa2 & ras1p &
!refin) then startdram
     else if (pclk & ads & mio & select & !muxoe & ras1p & ras0p
& !buscyc & !iready & !refin) then startdram
     else idle;

 state startdram:         " assert dramstart
     goto col_den;

 state col_den:  " change rowsel to select column on mux.
     if (pclk) then col_den2
     else col_den;

state col_den2:  "
     if (pclk) then idle
     else col_den2;

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

state_diagram [ras0];

state ras0idle:  "wait for DRAM or refresh cycles
     if (pclk & !ads & mio & select & !muxoe & !pa2 & ras0p &
!refin) then ras0act
     else if (pclk & select & mio & !muxoe & !pa2 & ras0p &
!buscyc & !refin & !iready) then ras0act
     else if (pclk & muxoe & !refadroe) then ras0act
     else ras0idle;

 state ras0act: " assert ras for bank 0
     if (pclk & reset) then ras0idle
     else if (pclk & iready) then ras0idle
     else if (pclk & muxoe & refadroe) then ras0idle
     else ras0act;

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
```

Figure B-2. DRAMP1 PLD Equations (Contd.)

```
state_diagram [ras1];

state ras1idle:  "wait for DRAM or refresh cycles
     if (pclk & !ads & mio & select & !muxoe & pa2 & ras1p &
!refin) then ras1act
     else if(pclk & select & mio & !muxoe & pa2 & ras1p & !buscyc
& !refin & !iready) then ras1act
     else if (pclk & muxoe & !refadroe) then ras1act
     else ras1idle;

state ras1act:  " assert ras for bank 1
     if (pclk & reset) then ras1idle
     else if (pclk & iready) then ras1idle
     else if (pclk & muxoe & refadroe) then ras0idle
     else ras1act;

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

state_diagram [muxoe,refadroe];

state [0,1]:    " wait for refin
     if (pclk & refin & ras1 & ras0 & dramstart ) then [1,1]
     else [0,1];

state [1,1]:    " turn off row/column mux oe
     if (pclk & reset) then [0,1]
     else if (pclk & ras0 & ras1) then [1,0]
     else if (pclk & refadroe & !ras0 & !ras1) then [0,1]
     else [1,1];
state [1,0]:    " turn on refresh address pal's oe
     if (pclk & reset) then [0,1]
     else if (pclk & !ras0 & !ras1) then [1,1]
     else [1,0];

state [0,0]:    " illegal
     goto [0,1];

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

test_vectors
([clk2,pclk,ads,mio,pa2,sel1,sel2,ras0p,ras1p,iready,refin,reset]

->                [dramstart,ras0,ras1,rowsel,muxoe,refadroe]);
"           s s                 d     r   r
"           e e r r r r r       r     o m e
"c p         l l a a e e e       a r r w u f
"l c a m p e e s s a f s         m a a s x a
"k l d i a c c 0 1 d i e         s s s e o o
"2 k s o 2 t t p p y n t         t 0 1 l e e
[c,h,h,h,h,h,h,h,h,1,1,h]  ->  [h,h,h,h,x,h]; "reset
[c,1,h,h,h,h,h,h,h,1,1,h]  ->  [h,h,h,h,x,h]; "reset
                                              "read bank 0
[c,h,h,h,h,h,h,h,h,1,1,1]  ->  [h,h,h,h,1,h]; "idle/rasidle
[c,1,x,h,h,h,h,h,h,1,1,1]  ->  [h,h,h,h,1,h]; "idle/rasidle
[c,h,1,h,1,h,h,h,h,1,1,1]  ->  [1,1,h,h,1,h]; "dramstart/ras0act
[c,1,x,h,1,h,h,h,h,1,1,1]  ->  [1,1,h,1,1,h]; " change MUX select
[c,h,h,h,1,h,h,1,h,1,1,1]  ->  [h,1,h,1,1,h]; " dramstart/ras0act
[c,1,h,h,1,h,h,1,h,1,1,1]  ->  [h,1,h,1,1,h]; " MUX select
[c,h,h,h,1,h,h,1,h,1,1,1]  ->  [h,1,h,h,1,h]; " ras0act
[c,1,x,h,1,h,h,1,h,x,1,1]  ->  [h,1,h,h,1,h]; " ras0act
                                              " read bank 1
```

Figure B-2. DRAMP1 PLD Equations (Contd.)

```
[c,h,l,h,h,h,h,l,h,h,l,l]   →   [l,h,l,h,l,h]; " dramstart/rasiact
[c,l,x,h,h,h,h,l,h,x,l,l]   →   [l,h,l,l,l,h]; " change MUX select
[c,h,h,h,h,h,h,h,l,l,l,l]   →   [h,h,l,l,l,h]; " dramstart/rasiact
[c,l,h,h,h,h,h,h,l,l,l,l]   →   [h,h,l,l,l,h]; " MUX select
[c,h,h,h,h,h,h,h,l,l,l,l]   →   [h,h,l,h,l,h]; " rasiact
[c,l,x,h,x,h,h,h,l,x,l,l]   →   [h,h,l,h,l,h]; " rasiact
                                                " read bank 1
[c,h,l,h,h,h,h,h,l,h,l,l]   →   [h,h,h,h,l,h]; " back to back read
[c,l,x,h,h,h,h,h,h,l,x,l,l]   →   [h,h,h,h,l,h]; " wait for precharge
[c,h,l,h,h,h,h,h,h,l,l,l,l]   →   [h,h,h,h,l,h]; " back to back read
[c,l,x,h,h,h,h,h,h,l,l,l,l]   →   [h,h,h,h,l,h]; " wait for precharge

[c,h,l,h,h,h,h,h,h,l,l,l,l]   →   [h,h,h,h,l,h]; " back to back read
[c,l,x,h,h,h,h,h,h,l,l,l,l]   →   [h,h,h,h,l,h]; " wait for precharge

[c,h,h,h,h,h,h,h,h,l,l,l]   →   [l,h,l,h,l,h]; " dramstart/rasiact
[c,l,h,h,h,h,h,h,h,l,l,l]   →   [l,h,l,l,l,h]; " change MUX select
[c,h,h,h,h,h,h,h,h,l,l,l,l]   →   [h,h,l,l,l,h]; " dramstart/rasiact
[c,l,h,h,h,h,h,h,h,l,l,l,l]   →   [h,h,l,l,l,h]; " MUX select
[c,h,h,h,h,h,h,h,h,l,l,l,l]   →   [h,h,l,h,l,h]; " rasiact
[c,l,x,h,x,h,h,h,l,x,l,l]   →   [h,h,l,h,l,h]; " rasiact

"    ras0 read then refresh
[c,h,l,h,l,h,h,h,h,h,l,l]   →   [l,l,h,h,l,h]; "idle/rasidle
[c,l,x,h,l,h,h,h,h,l,h,l]   →   [l,l,h,l,l,h]; " change MUX select
[c,h,h,h,l,h,h,l,h,l,h,l]   →   [h,l,h,l,l,h]; " dramstart/ras0act
[c,l,h,h,l,h,h,l,h,l,h,l]   →   [h,l,h,l,l,h]; " MUX select
[c,h,h,h,l,h,h,l,h,l,h,l]   →   [h,l,h,h,l,h]; " ras0act
[c,l,h,h,l,h,h,l,h,x,h,l]   →   [h,l,h,h,l,h]; " ras0act
[c,h,h,h,l,h,h,l,h,h,h,l]   →   [h,h,h,h,l,h]; " ras0act
[c,l,h,h,l,h,h,l,h,x,h,l]   →   [h,h,h,h,l,h]; " ras0act
[c,h,h,h,h,h,h,h,h,l,h,l]   →   [h,h,h,h,h,h]; " Refin act. Muxoe h
[c,l,h,h,h,h,h,h,h,l,h,l]   →   [h,h,h,h,h,h]; "
[c,h,h,h,h,h,h,h,h,l,h,l]   →   [h,h,h,h,h,l]; " act refadroe
[c,l,h,h,h,h,h,h,h,l,h,l]   →   [h,h,h,h,h,l]; "
[c,h,h,h,h,h,h,h,h,l,h,l]   →   [h,l,l,h,h,l]; " act RAS
[c,l,h,h,h,h,h,h,h,l,x,l]   →   [h,l,l,h,h,l]; "
[c,h,h,h,h,h,h,h,h,l,x,l]   →   [h,l,l,h,h,h]; " deact muxoe
[c,l,h,h,h,h,h,h,l,l,l,x,l]   →   [h,l,l,h,h,h]; "
[c,h,h,h,h,h,h,h,l,l,l,l,l]   →   [h,h,h,h,l,h]; " deact RAS
[c,l,h,h,h,h,h,h,l,l,l,l,l]   →   [h,h,h,h,l,h]; "
[c,h,h,h,h,h,h,h,l,l,l,l,l]   →   [h,h,h,h,l,h]; " mempend
[c,l,h,h,h,h,h,h,h,h,h,l,l,l]   →   [h,h,h,h,l,h]; "
[c,h,h,h,h,h,h,h,h,h,l,l,l]   →   [h,h,h,h,l,h]; "
 end   DRAM_CONTROLLER_FOR_80386;
```

Figure B-2. DRAMP1 PLD Equations (Contd.)

```
Device U33

- Reduced Equations:

     buscyc := !(buscyc & pclk & !pipecyc
                 # !buscyc & !pclk & pipecyc
                 # !buscyc & !iready & pipecyc
                 # !ads & buscyc & pclk);

pipecyc := !(buscyc & !pclk & !pipecyc
             # !ads & !buscyc & iready & pclk & pipecyc);

dramstart := !(!dramstart & !pclk
               # !dramstart & rowsel
               # !buscyc & !iready & mio & !muxoe & pclk &
                ras0p & ras1p & !refin & rowsel & sel1 & sel2
               # !ads & mio & !muxoe & pa2 & pclk & ras1p &
                !refin & rowsel & sel1 & sel2
               # !ads & mio & !muxoe & !pa2 & pclk & ras0p &
                !refin & rowsel & sel1 & sel2);
rowsel := !(!pclk & !rowsel # !dramstart);
ras0 := !(!iready & !ras0 & !refadroe & !reset
          # !iready & !muxoe & !ras0 & !reset
          # !pclk & !ras0
          # muxoe & pclk & ras0 & !refadroe
          # !buscyc & !iready & mio & !muxoe & !pa2 & pclk &
           ras0 & ras0p & !refin & sel1 & sel2
          # !ads & mio & !muxoe & !pa2 & pclk & ras0 & ras0p &
           !refin & sel1 & sel2);
ras1 := !(!iready & !ras1 & !refadroe & !reset
          # !iready & !muxoe & !ras1 & !reset
          # !pclk & !ras1
          # muxoe & pclk & ras1 & !refadroe
          # !buscyc & !iready & mio & !muxoe & pa2 & pclk &
           ras1 & ras1p & !refin & sel1 & sel2
          # !ads & mio & !muxoe & pa2 & pclk & ras1 & ras1p &
           !refin & sel1 & sel2);

muxoe := !(!muxoe & !refadroe
           # pclk & !ras0 & !ras1 & refadroe
           # muxoe & pclk & reset
           # !dramstart & !muxoe
           # !muxoe & !ras0
           # !muxoe & !ras1
           # !muxoe & !refin
           # !muxoe & !pclk);

refadroe := !(muxoe & ras1 & !refadroe & !reset
              # muxoe & ras0 & !refadroe & !reset
              # muxoe & !pclk & !refadroe
              # muxoe & pclk & ras0 & ras1 & !reset);
```

Figure B-2. DRAMP1 PLD Equations (Contd.)

```
module     DRAM_CONTROLLER_FOR_80386;
flag '-r3','-t2','-ul'

title
'80386 Interleaved DRAM Controller pld-2 pipelined 1ws
Intel Corporation.'

     U34    device      'E0320';

" Constants:

     ON          =     1;
     OFF         =     0;
     h           =     1;
     l           =     0;
     x           =     .X.;              " ABEL 'don't care' symbol
     c           =     .C.;              " ABEL 'clocking input' symbol

"Pin names:

  "Control
    clk2         pin 1;   " 80386 Double-Frequency system clock
    oe           pin 11;  " Output Enable (tie active low)

  "Inputs
    pclk         pin 2;   " system clock
    w_r          pin 7;   " write/read# from 80386
    muxoe        pin 6;   " from dramp1 PLD
    dramstart    pin 5;   " from dramp1 PLD
    ras0         pin 4;   " from dramp1 PLD
    ras1         pin 3;   " from dramp1 PLD
    refreq       pin 8;   " from refresh counter
    reset        pin 9;

  "Outputs
    dramrdy      pin 12;  " READY for dram cycles
    ale          pin 13;  " Address latch enable
    cas          pin 14;  " cas for drams
    refin        pin 15;  " refresh request to dramp1 PLD
    ras0p        pin 16;  " precharge counter to dramp1 PLD
    ras1p        pin 17;  " precharge counter to dramp1 PLD
    qr           pin 18;  "
    we           pin 19;  " write enable
```

Figure B-3. DRAMP2 PLD Equations

```
state_diagram [refin,qr];
                state [0,1]:                                                    "idle
                        if (refreq  & pclk) then [1,1]
                        else [0,1];
                state [1,1]:            " request refresh
                        if (pclk & reset) then [0,1]
                        else if ( !ras0 & !ras1 & pclk ) then [0,0]
                        else [1,1];
                state [0,0]:            " wait for request to be negated
                        if (pclk & reset) then [0,1]
                        else if ( !refreq & pclk) then [0,1]
                        else [0,0];
                state [1,0]:            " illegal
                        goto [0,1];

state_diagram [ras0p];
                state 1:
                        if (pclk & !ras0) then 0
                        else 1;
                state 0:
                        if (pclk & ras0) then 1
                        else 0;

state_diagram [ras1p];
                state 1:
                        if (pclk & !ras1) then 0
                        else 1;
                state 0:
                        if (pclk & ras1) then 1
                        else 0;

state_diagram [we];

                state 1: "read cycle
                        if (!pclk & !dramstart & w_r) then 0
                        else if (!pclk & !dramstart & !w_r) then 1
                        else 1;
                 state 0: " write cycle
                        if (!pclk & !dramstart & !w_r) then 1
                        else if (!pclk & !dramstart & w_r ) then 0
                        else 0;


state_diagram [dramrdy];
                state [1]:  " not ready or inactive
                        if (pclk & !muxoe & !ale) then [0]
                        else [1];
                state [0]:    "
                        if (pclk ) then [1]
                        else [0];
```

Figure B-3. DRAMP2 PLD Equations (Contd.)

```
state_diagram [cas,ale];
                state [1,1]:
                        if (pclk & !ras0 & ras1 & !w_r) then [0,0]
                        else if (pclk & ras0 & !ras1 & !w_r) then [0,0]
                        else if (pclk & !ras0 & ras1 & w_r) then [1,0]
                        else if (pclk & ras0 & !ras1 & w_r) then [1,0]
                        else [1,1];

                state [1,0]:        " wait for valid write data
                        goto  [0,0];

                state [0,0]:
                        if (reset & pclk) then [1,1]
                        else if (pclk & !dramrdy) then [1,1]
                        else [0,0];
                state [0,1]:  "invalid state
                        goto [1,1];
```

Figure B-3. DRAMP2 PLD Equations (Contd.)

```
test_vectors  ([clk2,pclk,w_r,dramstart,ras0,ras1,muxoe,refreq,reset] ->
  [cas,ras0p,ras1p,dramrdy,refin,we]);

 "       d          r              r r r r
 "       r      m   e  r           a a e e
 "c p    a r r  u   f  e         c s s a f
 "l c    m a a  x   r  s         a 0 1 d i w
 "k l w  s s s  o   e  e         s p p y n e
 "2 k r  t 0 1  e   q  t                          read cycle
 [c,h,l, h,h,h ,x ,h ,h ]   -> [h,h,h,h,x,x];     "reset
 [c,l,l, h,h,h ,x ,h ,h ]   -> [h,h,h,h,x,h];     "reset
 [c,h,l, h,h,h ,l ,l ,l ]   -> [h,h,h,h,l,h];     "idle/rasidle
 [c,l,l, x,x,h ,l ,l ,l ]   -> [h,h,h,h,l,h];     "idle/rasidle
 [c,h,l, l,l,h ,l ,l ,l ]   -> [l,l,h,h,l,h];     " startdram/ras0act
 [c,l,l, l,l,h ,l ,l ,l ]   -> [l,l,h,h,l,h];     " col_den/ras0act
 [c,h,l, h,l,h ,l ,l ,l ]   -> [l,l,h,l,l,h];     " ras0act
 [c,l,l, h,l,h ,l ,l ,l ]   -> [l,l,h,l,l,h];     " ras0act
 [c,h,l, h,l,h ,l ,l ,l ]   -> [h,l,h,h,l,h];     " ras0act
 [c,l,l, h,l,h ,l ,l ,l ]   -> [h,l,h,h,l,h];     " ras0act

 [c,h,l, h,h,h ,l ,l ,l ]   -> [h,h,h,h,l,h];     " idle/rasidle
 [c,l,l, h,h,h ,l ,l ,l ]   -> [h,h,h,h,l,h];     " idle/rasidle
 [c,l,l, h,h,h ,l ,l ,l ]   -> [h,h,h,h,l,h];     " idle/rasidle

 "                                                  write cycle
 [c,h,h, h,h,h ,l ,l ,l ]   -> [h,h,h,h,l,h];     "idle/rasidle
 [c,l,h, x,x,h ,l ,l ,l ]   -> [h,h,h,h,l,l];     "idle/rasidle
 [c,h,h, l,l,h ,l ,l ,l ]   -> [h,l,h,h,l,l];     " startdram/ras0act
 [c,l,h, l,l,h ,l ,l ,l ]   -> [l,l,h,h,l,l];     " col_den/ras0act
 [c,h,h, h,l,h ,l ,l ,l ]   -> [l,l,h,l,l,l];     " ras0act
 [c,l,h, h,l,h ,l ,l ,l ]   -> [l,l,h,l,l,l];     " ras0act
 [c,h,h, h,l,h ,l ,l ,l ]   -> [h,l,h,h,l,l];     " ras0act
 [c,l,h, h,l,h ,l ,l ,l ]   -> [h,l,h,h,l,l];     " ras0act

 [c,h,h, h,h,h ,l ,l ,l ]   -> [h,h,h,h,l,l];     " idle/rasidle
 [c,l,h, h,h,h ,l ,l ,l ]   -> [h,h,h,h,l,l];     " idle/rasidle
 [c,l,h, h,h,h ,l ,l ,l ]   -> [h,h,h,h,l,l];     " idle/rasidle



   end  DRAM_CONTROLLER_FOR_80386;
```

Figure B-3. DRAMP2 PLD Equations (Contd.)

```
80386 Interleaved DRAM Controller pld-1 pipelined 1ws
Equations for Module DRAM_CONTROLLER_FOR_80386

Device U34

 - Reduced Equations:

      !refin := (!refin & !refreq
                # !qr
                # pclk & !ras0 & !ras1 & refin
                # pclk & refin & reset
                # !pclk & !refin);

      !qr := (!pclk & !qr & !refin
                # !qr & !refin & refreq & !reset
                # pclk & qr & !ras0 & !ras1 & refin & !reset);

      !ras0p := (!pclk & !ras0p # pclk & !ras0);

      !ras1p := (!pclk & !ras1p # pclk & !ras1);

      !we := (dramstart & !we # pclk & !we # !dramstart & !pclk & w_r);

      !dramrdy := (!dramrdy & !pclk # !ale & dramrdy & !muxoe & pclk);

      !cas := (!ale & dramrdy & !reset
                # !ale & !pclk
                # !ale & cas
                # cas & pclk & ras0 & !ras1 & !w_r
                # cas & pclk & !ras0 & ras1 & !w_r);

      !ale := (!ale & dramrdy & !reset
                # !ale & !pclk
                # !ale & cas
                # cas & pclk & ras0 & !ras1
                # cas & pclk & !ras0 & ras1);
```

Figure B-3. DRAMP2 PLD Equations (Contd.)

REFRESH ADDRESS COUNTER PLD

The Refresh Address Counter PLD maintains the address of the next DRAM row to be refreshed. After every refresh cycle, the PLD increments this address. Table B-1 shows the inputs and outputs of the Refresh Address Counter PLD.

PLD equations are shown in Figure B-4. Ten bits of row address are provided using a 20RS10 PLD. For a system operating at any speed, standard-PLD speeds are sufficient.

Table B-1. Refresh Address Counter PLD Pin Description

PLD Controls			
Name	**Connects From**	**PLD Usage**	
CLOCK		PLD register clock	
OE		outputs enable on refresh	
PLD Inputs			
Name	**Connects From**	**PLD Usage**	**Sampled**
NC0 NC1 NC2 NC3 NC4 NC5 NC6 NC7	Not connected	Not used	Never
PLD Outputs			
Name	**Connects To**	**PLD Usage**	**Changes State**
Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8	Muxed Addr 0 Muxed Addr 1 Muxed Addr 2 Muxed Addr 3 Muxed Addr 4 Muxed Addr 5 Muxed Addr 6 Muxed Addr 7 Muxed Addr 8	Implements 9-bit counter	Any Clock

```
module refaddr;
flag '-r3';
title 'refresh address counter pal NDM intel corporation'

"increments the address by one until 9 bits are 01FFH

U35  device 'p20rs10';
h,l,c,x = 1,0,.C.,.X.;

gnd pin 12;
vcc pin 24;
oe pin 13;

clk pin 1;        "refresh increment signal

RA0 pin 14;        "bit 0 of refresh address
RA1 pin 15;         "bit 1 of refresh address
RA2 pin 16;        "bit 2 of refresh address
RA3 pin 17;        "bit 3 of refresh address
RA4 pin 18;        "bit 4 of refresh address
RA5 pin 19;        "bit 5 of refresh address
RA6 pin 20;        "bit 6 of refresh address
RA7 pin 23;        "bit 7 of refresh address
RA8 pin 22;        "bit 8 of refresh address

equations RA0 := !RA0;

state_diagram [RA1]
        state [0]:
                if (RA0) then 1
        else [0];
        state [1]:
                if (RA0) then [0]
                else [1];

state_diagram [RA2]
        state [0]:
                if (RA1 & RA0) then 1
                else [0];
        state [1]:
                if (RA1 & RA0) then [0]
                else [1];

state_diagram [RA3]
        state [0]:
                if (RA2 & RA1 & RA0) then 1
                else [0];
        state [1]:
                if (RA2 & RA1 & RA0) then [0]
                else [1];

state_diagram [RA4]
        state [0]:
                if (RA3 & RA1 & RA0) then 1
                else [0];
        state [1]:
                if (RA3 & RA2 & RA1 & RA0) then [0]
                else [1];
```

Figure B-4. Refresh Address Counter PLD Equations

```
state_diagram [RA5]
        state [0]:
                if (RA4 & RA3 & RA2 & RA1 & RA0) then 1
                else [0];
        state [1]:
                if (RA4 & RA3 & RA2 & RA1 & RA0) then [0]
                else [1];

state_diagram [RA6]
        state [0]:
                if (RA5 & RA4 & RA3 & RA2 & RA1 &RA0) then 1
                else [0];
        state [1]:
                if (RA5 & RA4 & RA3 & RA2 & RA1 & RA0) then [0]
                else [1];

state_diagram [RA7]
        state [0]:
                if (RA6 & RA5 & RA4 & RA3 & RA2 & RA1 & RA0) then 1
                else [0];
        state [1]:
                if (RA6 & RA5 & RA4 & RA3 & RA2 & RA1 & RA0) then [0]
                else [1];

state_diagram [RA8]
        state [0]:
                if (RA7 & RA6 & RA5 & RA4 & RA3 & RA2 & RA1 &
RA0) then 1
                else [0];
        state [1]:
                if (RA7 & RA6 & RA5 & RA4 & RA3 & RA2 & RA1 &
RA0) then [0]
                else [1];

test_vectors ([clk] → [RA8,RA7,RA6,RA5,RA4,RA3,RA2,RA1,RA0])
                [c] → [x,0,0,0,0,0,0,0,0];                    "0
                [c] → [x,x,x,x,x,x,x,x,1];
                [c] → [x,x,x,x,x,x,x,1,0];
                [c] → [x,x,x,x,x,x,x,x,x];
                [c] → [x,x,x,x,x,x,x,x,x];
                [c] → [x,x,x,x,x,x,x,x,x];                    "5
                [c] → [x,x,x,x,x,x,x,x,x];
                [c] → [x,x,x,x,x,x,x,x,x];
                [c] → [x,x,x,x,x,x,x,x,x];
                [c] → [x,x,x,x,x,x,x,x,x];                    "10
                [c] → [0,0,0,0,0,1,0,1,0];
                [c] → [x,x,x,x,x,x,x,x,x];
                [c] → [x,x,x,x,x,x,x,x,x];
                [c] → [x,x,x,x,x,x,x,x,x];
                [c] → [x,x,x,x,x,x,x,x,x];
                [c] → [x,x,x,x,x,1,1,1,1];                    "15

end refaddr;
```

Figure B-4. Refresh Address Counter PLD Equations (Contd.)

```
refresh address counter pal    intel corporation
Equations for Module refaddr

Device U35

- Reduced Equations:

     RA0 := !(RA0);

     RA1 := !(RA0 & RA1 # !RA0 & !RA1);

     RA2 := !(RA0 & RA1 & RA2 # !RA0 & !RA2 # !RA1 & !RA2);

     RA3 := !(RA0 & RA1 & RA2 & RA3
              # !RA0 & !RA3
              # !RA1 & !RA3
              # !RA2 & !RA3);

     RA4 := !(RA0 & RA1 & RA2 & RA3 & RA4
              # !RA0 & !RA4
              # !RA1 & !RA4
              # !RA2 & !RA4
              # !RA3 & !RA4);

     RA5 := !(RA0 & RA1 & RA2 & RA3 & RA4 & RA5
              # !RA0 & !RA5
              # !RA1 & !RA5
              # !RA2 & !RA5
              # !RA3 & !RA5
              # !RA4 & !RA5);

     RA6 := !(RA0 & RA1 & RA2 & RA3 & RA4 & RA5 & RA6
              # !RA0 & !RA6
              # !RA1 & !RA6
              # !RA2 & !RA6
              # !RA3 & !RA6
              # !RA4 & !RA6
              # !RA5 & !RA6);

     RA7 := !(RA0 & RA1 & RA2 & RA3 & RA4 & RA5 & RA6 & RA7
              # !RA0 & !RA7
              # !RA1 & !RA7
              # !RA2 & !RA7
              # !RA3 & !RA7
              # !RA4 & !RA7
              # !RA5 & !RA7
              # !RA6 & !RA7);

     RA8 := !(RA0 & RA1 & RA2 & RA3 & RA4 & RA5 & RA6 & RA7 & RA8
              # !RA0 & !RA8
              # !RA1 & !RA8
              # !RA2 & !RA8
              # !RA3 & !RA8
              # !RA4 & !RA8
              # !RA5 & !RA8
              # !RA6 & !RA8
              # !RA7 & !RA8);
```

Figure B-4. Refresh Address Counter PLD Equations (Contd.)

DOMESTIC SALES OFFICES

ALABAMA

Intel Corp.
5015 Bradford Dr., #2
Huntsville 35805
Tel: (205) 830-4010
FAX: (205) 837-2640

ARIZONA

†Intel Corp.
410 North 44th Street
Suite 500
Phoenix 85008
Tel: (602) 231-0386
FAX: (602) 244-0446

Intel Corp.
7225 N. Mona Lisa Rd.
Suite 215
Tucson 85741
Tel: (602) 544-0227
FAX: (602) 544-0232

CALIFORNIA

†Intel Corp.
21515 Vanowen Street
Suite 116
Canoga Park 91303
Tel: (818) 704-8500
FAX: (818) 340-1144

†Intel Corp.
300 N. Continental Blvd.
Suite 100
El Segundo 90245
Tel: (213) 640-6040
FAX: (213) 640-7133

Intel Corp.
1 Sierra Gate Plaza
Suite 280C
Roseville 95678
Tel: (916) 782-8086
FAX: (916) 782-8153

†Intel Corp.
9665 Chesapeake Dr.
Suite 325
San Diego 92123
Tel: (619) 292-8086
FAX: (619) 292-0628

†Intel Corp.*
400 N. Tustin Avenue
Suite 450
Santa Ana 92705
Tel: (714) 835-9642
TWX: 910-595-1114
FAX: (714) 541-9157

†Intel Corp.*
San Tomas 4
2700 San Tomas Expressway
2nd Floor
Santa Clara 95051
Tel: (408) 986-8086
TWX: 910-338-0255
FAX: (408) 727-2620

COLORADO

Intel Corp.
4445 Northpark Drive
Suite 100
Colorado Springs 80907
Tel: (719) 594-6622
FAX: (303) 594-0720

†Intel Corp.*
600 S. Cherry St.
Suite 700
Denver 80222
Tel: (303) 321-8086
TWX: 910-931-2289
FAX: (303) 322-8670

CONNECTICUT

†Intel Corp.
301 Lee Farm Corporate Park
83 Wooster Heights Rd.
Danbury 06810
Tel: (203) 748-3130
FAX: (203) 794-0339

FLORIDA

†Intel Corp.
800 Fairway Drive
Suite 160
Deerfield Beach 33441
Tel: (305) 421-0506
FAX: (305) 421-2444

†Intel Corp.
5850 T.G. Lee Blvd.
Suite 340
Orlando 32822
Tel: (407) 240-8000
FAX: (407) 240-8097

Intel Corp.
11300 4th Street North
Suite 170
St. Petersburg 33716
Tel: (813) 577-2413
FAX: (813) 578-1607

GEORGIA

†Intel Corp.
20 Technology Parkway
Suite 150
Norcross 30092
Tel: (404) 449-0541
FAX: (404) 605-9762

ILLINOIS

†Intel Corp.*
Woodfield Corp. Center III
300 N. Martingale Road
Suite 400
Schaumburg 60173
Tel: (708) 605-8031
FAX: (708) 706-9762

INDIANA

†Intel Corp.
8910 Purdue Road
Suite 350
Indianapolis 46268
Tel: (317) 875-0623
FAX: (317) 875-8938

IOWA

Intel Corp.
1930 St. Andrews Drive N.E.
2nd Floor
Cedar Rapids 52402
Tel: (319) 393-5510

KANSAS

†Intel Corp.
10985 Cody St.
Suite 140
Overland Park 66210
Tel: (913) 345-2727
FAX: (913) 345-2076

MARYLAND

†Intel Corp.*
10010 Junction Dr.
Suite 200
Annapolis Junction 20701
Tel: (301) 206-2860
FAX: (301) 206-3677
(301) 206-3678

MASSACHUSETTS

†Intel Corp.*
Westford Corp. Center
3 Carlisle Road
2nd Floor
Westford 01886
Tel: (508) 692-0960
TWX: 710-343-6333
FAX: (508) 692-7867

MICHIGAN

†Intel Corp.
7071 Orchard Lake Road
Suite 100
West Bloomfield 48322
Tel: (313) 851-8096
FAX: (313) 851-8770

MINNESOTA

†Intel Corp.
3500 W. 80th St.
Suite 360
Bloomington 55431
Tel: (612) 835-6722
TWX: 910-576-2867
FAX: (612) 831-6497

MISSOURI

†Intel Corp.
3300 Rider Trail South
Suite 170
Earth City 63045
Tel: (314) 291-1990
FAX: (314) 291-4341

NEW JERSEY

Intel Corp.
Arbor Circle South
8 Campus Drive
Parsippany 07054
Tel: (201) 455-1868
FAX: (201) 644-0680

†Intel Corp.*
Lincroft Office Center
125 Half Mile Road
Red Bank 07701
Tel: (908) 747-2233
FAX: (908) 747-0983

NEW YORK

Intel Corp.*
850 Crosskeys Office Park
Fairport 14450
Tel: (716) 425-2750
TWX: 510-253-7391
FAX: (716) 223-2561

†Intel Corp.*
2950 Express Dr., South
Suite 130
Islandia 11722
Tel: (516) 231-3300
TWX: 510-227-6236
FAX: (516) 348-7939

†Intel Corp.
300 Westage Business Center
Suite 230
Fishkill 12524
Tel: (914) 897-3860
FAX: (914) 897-3125

Intel Corp.
Seventeen State Street
14th Floor
New York 10004
Tel: (212) 248-8086
FAX: (212) 248-0888

NORTH CAROLINA

†Intel Corp.
5800 Executive Center Dr.
Suite 105
Charlotte 28212
Tel: (704) 568-8966
FAX: (704) 535-2236

†Intel Corp.
5540 Centerview Dr.
Suite 215
Raleigh 27606
Tel: (919) 851-9537
FAX: (919) 851-8974

OHIO

†Intel Corp.*
3401 Park Center Drive
Suite 220
Dayton 45414
Tel: (513) 890-5350
TWX: 810-450-2528
FAX: (513) 890-8658

†Intel Corp.*
25700 Science Park Dr.
Suite 100
Beachwood 44122
Tel: (216) 464-2736
TWX: 810-427-9298
FAX: (804) 282-0673

OKLAHOMA

Intel Corp.
6801 N. Broadway
Suite 115
Oklahoma City 73162
Tel: (405) 848-8086
FAX: (405) 840-9819

OREGON

†Intel Corp.
15254 N.W. Greenbrier Pkwy.
Building B
Beaverton 97006
Tel: (503) 645-8051
TWX: 910-467-8741
FAX: (503) 645-8181

PENNSYLVANIA

†Intel Corp.*
925 Harvest Drive
Suite 200
Blue Bell 19422
Tel: (215) 641-1000
FAX: (215) 641-0785

†Intel Corp.*
400 Penn Center Blvd.
Suite 610
Pittsburgh 15235
Tel: (412) 823-4970
FAX: (412) 829-7578

PUERTO RICO

†Intel Corp.
South Industrial Park
P.O. Box 910
Las Piedras 00671
Tel: (809) 733-8616

TEXAS

†Intel Corp.
8911 N. Capital of Texas Hwy.
Suite 4230
Austin 78759
Tel: (512) 794-8086
FAX: (512) 338-9335

†Intel Corp.*
12000 Ford Road
Suite 400
Dallas 75234
Tel: (214) 241-8087
FAX: (214) 484-1180

†Intel Corp.*
7322 S.W. Freeway
Suite 1490
Houston 77074
Tel: (713) 988-8086
TWX: 910-881-2490
FAX: (713) 988-3660

UTAH

†Intel Corp.
428 East 6400 South
Suite 104
Murray 84107
Tel: (801) 263-8051
FAX: (801) 268-1457

VIRGINIA

†Intel Corp.
9030 Stony Point Pkwy.
Suite 360
Richmond 23235
Tel: (804) 330-9393
FAX: (804) 330-3019

WASHINGTON

†Intel Corp.
155 108th Avenue N.E.
Suite 386
Bellevue 98004
Tel: (206) 453-8086
TWX: 910-443-3002
FAX: (206) 451-9556

Intel Corp.
408 N. Mullan Road
Suite 102
Spokane 99206
Tel: (509) 928-8086
FAX: (509) 928-9467

WISCONSIN

Intel Corp.
330 S. Executive Dr.
Suite 102
Brookfield 53005
Tel: (414) 784-8087
FAX: (414) 796-2115

CANADA

BRITISH COLUMBIA

Intel Semiconductor of
Canada, Ltd.
4585 Canada Way
Suite 202
Burnaby V5G 4L6
Tel: (604) 298-0387
FAX: (604) 298-8234

ONTARIO

†Intel Semiconductor of
Canada, Ltd.
2650 Queensview Drive
Suite 250
Ottawa K2B 8H6
Tel: (613) 829-9714
FAX: (613) 820-5936

†Intel Semiconductor of
Canada, Ltd.
190 Attwell Drive
Suite 500
Rexdale M9W 6H8
Tel: (416) 675-2105
FAX: (416) 675-2438

QUEBEC

†Intel Semiconductor of
Canada, Ltd.
1 Rue Holiday
Suite 115
Tour East
Pt. Claire H9R 5N3
Tel: (514) 694-9130
FAX: 514-694-0064

†Sales and Service Office
*Field Application Location

CG/SALE/022891

DOMESTIC DISTRIBUTORS

ALABAMA

Arrow Electronics, Inc.
1015 Henderson Road
Huntsville 35805
Tel: (205) 837-6955
FAX: 205-751-1581

Hamilton/Avnet Computer
4930 I Corporate Drive
Huntsville 35805

Hamilton/Avnet Electronics
4940 Research Drive
Huntsville 35805
Tel: (205) 837-7210
FAX: 205-721-0356

MTI Systems Sales
4950 Corporate Drive
Suite 120
Huntsville 35806
Tel: (205) 830-9526
FAX: (205) 830-9557

Pioneer/Technologies Group, Inc.
4825 University Square
Huntsville 35805
Tel: (205) 837-9300
FAX: 205-837-9358

ALASKA

Hamilton/Avnet Computer
1400 W. Benson Blvd., Suite 400
Anchorage 99503

ARIZONA

†Arrow Electronics, Inc.
4134 E. Wood Street
Phoenix 85040
Tel: (602) 437-0750
TWX: 910-951-1550

Hamilton/Avnet Computer
30 South McKemy Avenue
Chandler 85226

Hamilton/Avnet Computer
90 South McKemy Road
Chandler 85226

†Hamilton/Avnet Electronics
505 S. Madison Drive
Tempe 85281
Tel: (602) 231-5140
TWX: 910-950-0077

Hamilton/Avnet Electronics
30 South McKemy
Chandler 85226
Tel: (602) 961-6669
FAX: 602-961-4073

Wyle Distribution Group
4141 E. Raymond
Phoenix 85040
Tel: (602) 249-2232
TWX: 910-371-2871

CALIFORNIA

Arrow Commercial System Group
1502 Crocker Avenue
Hayward 94544
Tel: (415) 489-5371
FAX: (415) 489-9393

Arrow Commercial System Group
14242 Chambers Road
Tustin 92680
Tel: (714) 544-0200
FAX: (714) 731-8438

†Arrow Electronics, Inc.
19748 Dearborn Street
Chatsworth 91311
Tel: (213) 701-7500
TWX: 910-493-2086

†Arrow Electronics, Inc.
9511 Ridgehaven Court
San Diego 92123
Tel: (619) 565-4800
FAX: 619-279-8062

†Arrow Electronics, Inc.
521 Weddell Drive
Sunnyvale 94086
Tel: (408) 745-6600
TWX: 910-339-9371

†Arrow Electronics, Inc.
2961 Dow Avenue
Tustin 92680
Tel: (714) 838-5422
TWX: 910-595-2860

Hamilton/Avnet Computer
3170 Pullman Street
Costa Mesa 92626

Hamilton/Avnet Computer
1361B West 190th Street
Gardena 90248

Hamilton/Avnet Computer
4103 Northgate Blvd.
Sacramento 95834

Hamilton/Avnet Computer
4545 Viewridge Avenue
San Diego 92123

Hamilton/Avnet Computer
1175 Bordeaux Drive
Sunnyvale 94089

Hamilton/Avnet Electronics
21150 Califa Street
Woodland Hills 91367

†Hamilton/Avnet Electronics
3170 Pullman Street
Costa Mesa 92626
Tel: (714) 641-4150
TWX: 910-595-2638

†Hamilton/Avnet Electronics
1175 Bordeaux Drive
Sunnyvale 94086
Tel: (408) 743-3300
TWX: 910-339-9332

†Hamilton/Avnet Electronics
4545 Ridgeview Avenue
San Diego 92123
Tel: (619) 571-7500
TWX: 910-595-2638

†Hamilton/Avnet Electronics
21150 Califa St.
Woodland Hills 91376
Tel: (818) 594-0404
FAX: 818-594-8233

†Hamilton/Avnet Electronics
10950 W. Washington Blvd.
Culver City 20230
Tel: (213) 558-2458
TWX: 910-340-6364

†Hamilton/Avnet Electronics
1361B West 190th Street
Gardena 90248
Tel: (213) 217-6700
TWX: 910-340-6364

†Hamilton/Avnet Electronics
4103 Northgate Blvd.
Sacramento 95834
Tel: (916) 920-3150

Pioneer/Technologies Group, Inc.
134 Rio Robles
San Jose 95134
Tel: (408) 954-9100
FAX: 408-954-9113

Wyle Distribution Group
124 Maryland Street
El Segundo 90254
Tel: (213) 322-8100

Wyle Distribution Group
7431 Chapman Ave.
Garden Grove 92641
Tel: (714) 891-1717
FAX: 714-891-1621

†Wyle Distribution Group
2951 Sunrise Blvd., Suite 175
Rancho Cordova 95742
Tel: (916) 638-5282

†Wyle Distribution Group
9525 Chesapeake Drive
San Diego 92123
Tel: (619) 565-9171
TWX: 910-335-1590

†Wyle Distribution Group
3000 Bowers Avenue
Santa Clara 95051
Tel: (408) 727-2500
TWX: 408-988-2747

†Wyle Distribution Group
17872 Cowan Avenue
Irvine 92714
Tel: (714) 863-9953
TWX: 910-371-7127

†Wyle Distribution Group
26677 W. Agoura Rd.
Calabasas 91302
Tel: (818) 880-9000
TWX: 372-0232

COLORADO

Arrow Electronics, Inc.
7060 South Tucson Way
Englewood 80112
Tel: (303) 790-4444

Hamilton/Avnet Computer
9605 Maroon Circle, Ste. 200
Engelwood 80112

†Hamilton/Avnet Electronics
9605 Maroon Circle
Suite 200
Englewood 80112
Tel: (303) 799-0663
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 Computer
Commerce Industrial Park
Commerce Drive
Danbury 06810

†Hamilton/Avnet Electronics
Commerce Industrial Park
Commerce Drive
Danbury 06810
Tel: (203) 797-2800
TWX: 710-456-9974

†Pioneer/Standard Electronics
112 Main Street
Norwalk 06851
Tel: (203) 853-1515
FAX: 203-838-9901

FLORIDA

†Arrow Electronics, Inc.
400 Fairway Drive
Suite 102
Deerfield Beach 33441
Tel: (305) 429-8200
FAX: 305-428-3991

†Arrow Electronics, Inc.
37 Skyline Drive
Suite 3101
Lake Marv 32746
Tel: (407) 323-0252
FAX: 407-323-3189

Hamilton/Avnet Computer
6801 N.W. 15th Way
Ft. Lauderdale 33309

Hamilton/Avnet Computer
3247 Spring Forest Road
St. Petersburg 33702

†Hamilton/Avnet Electronics
6801 N.W. 15th Way
Ft. Lauderdale 33309
Tel: (305) 971-2900
FAX: 305-971-5420

†Hamilton/Avnet Electronics
3197 Tech Drive North
St. Petersburg 33702
Tel: (813) 573-3930
FAX: 813-572-4329

†Hamilton/Avnet Electronics
6947 University Boulevard
Winter Park 32792
Tel: (407) 628-3888
FAX: 407-678-1878

†Pioneer/Technologies Group, Inc.
337 Northlake Blvd., Suite 1000
Alta Monte Springs 32701
Tel: (407) 834-9090
FAX: 407-834-0865

Pioneer/Technologies Group, Inc.
674 S. Military Trail
Deerfield Beach 33442
Tel: (305) 428-8877
FAX: 305-481-2950

GEORGIA

Arrow Commercial System Group
3400 C. Corporate Way
Deluth 30139
Tel: (404) 623-8825
FAX: (404) 623-8802

†Arrow Electronics, Inc.
4250 E. Rivergreen Parkway
Deluth 30136
Tel: (404) 497-1300
TWX: 810-766-0439

Hamilton/Avnet Computer
5825 D. Peachtree Corners E.
Norcross 30092

†Hamilton/Avnet Electronics
5825 D Peachtree Corners
Norcross 30092
Tel: (404) 447-7500
TWX: 810-766-0432

Pioneer/Technologies Group, Inc.
3100 F Northwoods Place
Norcross 30071
Tel: (404) 448-1711
FAX: 404-446-8270

ILLINOIS

†Arrow Electronics, Inc.
1140 W. Thorndale
Itasca 60143
Tel: (708) 250-0500
TWX: 708-250-0916

Hamilton/Avnet Computer
1130 Thorndale Avenue
Bensenville 60106

†Hamilton/Avnet Electronics
1130 Thorndale Avenue
Bensenville 60106
Tel: (708) 860-7780
TWX: 708-860-8530

MTI Systems Sales
1100 W. Thorndale
Itasca 60143
Tel: (708) 773-2300

†Pioneer/Standard Electronics
2171 Executive Dr., Suite 200
Addison 60101
Tel: (708) 495-9680
FAX: 708-495-9831

INDIANA

†Arrow Electronics, Inc.
7108 Lakeview Parkway West Drive
Indianapolis 46268
Tel: (317) 299-2071
FAX: 317-299-0255

Hamilton/Avnet Computer
485 Gradle Drive
Carmel 46032

Hamilton/Avnet Electronics
485 Gradle Drive
Carmel 46032
Tel: (317) 844-9333
FAX: 317-844-5921

†Pioneer/Standard Electronics
9350 Priority Way
West Drive
Indianapolis 46250
Tel: (317) 573-0880
FAX: 317-573-0979

†Certified Technical Distributor

CG/SALE/022891

DOMESTIC DISTRIBUTORS (Contd.)

IOWA

Hamilton/Avnet Computer
915 33rd Avenue SW
Cedar Rapids 52404

Hamilton/Avnet Electronics
915 33rd Avenue, S.W.
Cedar Rapids 52404
Tel: (319) 362-4757

KANSAS

Arrow Electronics, Inc.
8208 Melrose Dr., Suite 210
Lenexa 66214
Tel: (913) 541-9542
FAX: 913-541-0328

Hamilton/Avnet Computer
15313 W. 95th Street
Lenexa 61219

†Hamilton/Avnet Electronics
15313 W. 95th
Overland Park 66215
Tel: (913) 888-8900
FAX: 913-541-7951

KENTUCKY

Hamilton/Avnet Electronics
805 A. Newtown Circle
Lexington 40511
Tel: (606) 259-1475

MARYLAND

†Arrow Electronics, Inc.
8300 Guilford Drive
Suite H, River Center
Columbia 21046
Tel: (301) 995-6002
FAX: 301-381-3854

Hamilton/Avnet Computer
6822 Oak Hall Lane
Columbia 21045

†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) 290-8150
FAX: 301-290-6474

†Pioneer/Technologies Group, Inc.
9100 Gaither Road
Gaithersburg 20877
Tel: (301) 921-0660
FAX: 301-921-4255

MASSACHUSETTS

Arrow Electronics, Inc.
25 Upton Dr.
Wilmington 01887
Tel: (508) 658-0900
TWX: 710-393-6770

Hamilton/Avnet Computer
10 D Centennial Drive
Peabody 01960

†Hamilton/Avnet Electronics
10D Centennial Drive
Peabody 01960
Tel: (508) 532-9838
FAX: 508-596-7802

†Pioneer/Standard Electronics
44 Hartwell Avenue
Lexington 02173
Tel: (617) 861-9200
FAX: 617-863-1547

Wyle Distribution Group
15 Third Avenue
Burlington 01803
Tel: (617) 272-7300
FAX: 617-272-6809

MICHIGAN

†Arrow Electronics, Inc.
19880 Haggerty Road
Livonia 48152
Tel: (313) 665-4100
TWX: 810-223-6020

Hamilton/Avnet Computer
2215 S.E. A-5
Grand Rapids 49508

Hamilton/Avnet Computer
41650 Garden Rd., Ste. 100
Novi 48050

Hamilton/Avnet Electronics
2215 29th Street S.E.
Space A5
Grand Rapids 49508
Tel: (616) 243-8805
FAX: 616-698-1831

Hamilton/Avnet Electronics
41650 Garden Brook
Novi 48050
Tel: (313) 347-4271
FAX: 313-347-4021

†Pioneer/Standard Electronics
4505 Broadmoor S.E.
Grand Rapids 49508
Tel: (616) 698-1800
FAX: 616-698-1831

†Pioneer/Standard Electronics
13485 Stamford
Livonia 48150
Tel: (313) 525-1800
FAX: 313-427-3720

MINNESOTA

†Arrow Electronics, Inc.
5230 W. 73rd Street
Edina 55435
Tel: (612) 830-1800
TWX: 910-576-3125

Hamilton/Avnet Computer
12400 Whitewater Drive
Minnetonka 55343

†Hamilton/Avnet Electronics
12400 Whitewater Drive
Minnetonka 55434
Tel: (612) 932-0600
TWX: 910-576-2720

†Pioneer/Standard Electronics
7625 Golden Triange Dr.
Suite G
Eden Prairie 55343
Tel: (612) 944-3355
FAX: 612-944-3794

MISSOURI

†Arrow Electronics, Inc.
2380 Schuetz
St. Louis 63141
Tel: (314) 567-6888
FAX: 314-567-1164

Hamilton/Avnet Computer
739 Goddard Avenue
Chesterfield 63005

†Hamilton/Avnet Electronics
741 Goddard
Chesterfield 63005
Tel: (314) 537-1600
FAX: 314-537-4248

NEW HAMPSHIRE

Hamilton/Avnet Computer
2 Executive Park Drive
Bedford 03102

Hamilton/Avnet Computer
444 East Industrial Park Dr.
Manchester 03103

NEW JERSEY

†Arrow Electronics, Inc.
4 East Stow Road
Unit 11
Marlton 08053
Tel: (609) 596-8000
FAX: 609-596-9632

†Arrow Electronics
6 Century Drive
Parsipanny 07054
Tel: (201) 538-0900
FAX: 201-538-0900

Hamilton/Avnet Computer
1 Keystone Ave., Bldg. 36
Cherry Hill 08003

Hamilton/Avnet Computer
10 Industrial Road
Fairfield 07006

†Hamilton/Avnet Electronics
1 Keystone Ave., Bldg. 36
Cherry Hill 08003
Tel: (609) 424-0110
FAX: 609-751-2552

†Hamilton/Avnet Electronics
10 Industrial
Fairfield 07006
Tel: (201) 575-3390
FAX: 201-575-5839

†MTI Systems Sales
9 Law Drive
Fairfield 07006
Tel: (201) 227-5552
FAX: 201-575-6336

†Pioneer/Standard Electronics
14-A Madison Rd.
Fairfield 07006
Tel: (201) 575-3510
FAX: 201-575-3454

NEW MEXICO

Alliance Electronics Inc.
10510 Research Avenue
Albuquerque 87123
Tel: (505) 292-3360
FAX: 505-292-6537

Hamilton/Avnet Computer
5659 Jefferson, N.E. Suites A & B
Albuquerque 87109

†Hamilton/Avnet Electronics
5659A Jefferson N.E.
Albuquerque 87109
Tel: (505) 765-1500
FAX: 505-243-1395

NEW YORK

†Arrow Electronics, Inc.
3375 Brighton Henrietta Townline Rd.
Rochester 14623
Tel: (716) 427-0300
TWX: 510-253-4766

Arrow Electronics, Inc.
20 Oser Avenue
Hauppauge 11788
Tel: (516) 231-1000
TWX: 510-227-6623

Hamilton/Avnet Computer
933 Motor Parkway
Haupauge 11788

Hamilton/Avnet Computer
2060 Townline
Rochester 14623

†Hamilton/Avnet Electronics
933 Motor Parkway
Hauppauge 11788
Tel: (516) 231-9800
TWX: 510-224-6166

†Hamilton/Avnet Electronics
2060 Townline Rd.
Rochester 14623
Tel: (716) 272-2744
TWX: 510-253-5470

Hamilton/Avnet Electronics
103 Twin Oaks Drive
Syracuse 13206
Tel: (315) 437-0288
TWX: 710-541-1560

†MTI Systems Sales
38 Harbor Park Drive
Port Washington 11050
Tel: (516) 621-6200
FAX: 510-223-0846

Pioneer/Standard Electronics
68 Corporate Drive
Binghamton 13904
Tel: (607) 722-9300
FAX: 607-722-9562

Pioneer/Standard Electronics
40 Oser Avenue
Hauppauge 11787
Tel: (516) 231-9200
FAX: 510-227-9869

†Pioneer/Standard Electronics
60 Crossway Park West
Woodbury, Long Island 11797
Tel: (516) 921-8700
FAX: 516-921-2143

†Pioneer/Standard Electronics
840 Fairport Park
Fairport 14450
Tel: (716) 381-7070
FAX: 716-381-5955

NORTH CAROLINA

†Arrow Electronics, Inc.
5240 Greensdairy Road
Raleigh 27604
Tel: (919) 876-3132
TWX: 510-928-1856

Hamilton/Avnet Computer
3510 Spring Forest Road
Raleigh 27604

†Hamilton/Avnet Electronics
3510 Spring Forest Drive
Raleigh 27604
Tel: (919) 878-0819
TWX: 510-928-1836

Pioneer/Technologies Group, Inc.
9401 L-Southern Pine Blvd.
Charlotte 28210
Tel: (919) 527-8188
FAX: 704-522-8564

Pioneer Technologies Group, Inc.
2810 Meridian Parkway
Suite 148
Durham 27713
Tel: (919) 544-5400
FAX: 919-544-5885

OHIO

Arrow Commercial System Group
284 Cramer Creek Court
Dublin 43017
Tel: (614) 889-9347
FAX: (614) 889-9680

†Arrow Electronics, Inc.
6238 Cochran Road
Solon 44139
Tel: (216) 248-3990
TWX: 810-427-9409

Hamilton/Avnet Computer
7764 Washington Village Dr.
Dayton 45459

Hamilton/Avnet Computer
30325 Bainbridge Rd., Bldg. A
Solon 44139

†Hamilton/Avnet Electronics
7760 Washington Village Dr.
Dayton 45459
Tel: (513) 439-6733
FAX: 513-439-6711

†Hamilton/Avnet Electronics
30325 Bainbridge
Solon 44139
Tel: (216) 349-5100
TWX: 810-427-9452

Hamilton/Avnet Computer
777 Brooksedge Blvd.
Westerville 43081
Tel: (614) 882-7004
FAX: 614-882-8650

Hamilton/Avnet Electronics
777 Brooksedge Blvd.
Westerville 43081
Tel: (614) 882-7004

MTI Systems Sales
23400 Commerce Park Road
Beachwood 44122
Tel: (216) 464-6688

†Pioneer/Standard Electronics
4433 Interpoint Boulevard
Dayton 45424
Tel: (513) 236-9900
FAX: 513-236-8133

†Pioneer/Standard Electronics
4800 E. 131st Street
Cleveland 44105
Tel: (216) 587-3600
FAX: 216-663-1004

†Certified Technical Distributor

DOMESTIC DISTRIBUTORS (Contd.)

OKLAHOMA

Arrow Electronics, Inc.
4719 South Memorial Dr.
Tulsa 74145

†Hamilton/Avnet Electronics
12121 E. 51st St., Suite 102A
Tulsa 74146
Tel: (918) 252-7297

OREGON

†Almac Electronics Corp.
1885 N.W. 169th Place
Beaverton 97005
Tel: (503) 629-8090
FAX: 503-645-0611

Hamilton/Avnet Computer
9409 Southwest Nimbus Ave.
Beaverton 97005

†Hamilton/Avnet Electronics
9409 S.W. Nimbus Ave.
Beaverton 97005
Tel: (503) 627-0201
FAX: 503-641-4012

Wyle
9640 Sunshine Court
Bldg. G, Suite 200
Beaverton 97005
Tel: (503) 643-7900
FAX: 503-646-5466

PENNSYLVANIA

Arrow Electronics, Inc.
650 Seco Road
Monroeville 15146
Tel: (412) 856-7000

Hamilton/Avnet Computer
2800 Liberty Ave., Bldg. E
Pittsburgh 15222

Hamilton/Avnet Electronics
2800 Liberty Ave.
Pittsburgh 15238
Tel: (412) 281-4150

Pioneer/Standard Electronics
259 Kappa Drive
Pittsburgh 15238
Tel: (412) 782-2300
FAX: 412-963-8255

†Pioneer/Technologies Group, Inc.
Delaware Valley
261 Gibralter Road
Horsham 19044
Tel: (215) 674-4000
FAX: 215-674-3107

TENNESSEE

Arrow Commercial System Group
3635 Knight Road
Suite 7
Memphis 38118
Tel: (901) 367-0540
FAX: (901) 367-2081

TEXAS

Arrow Electronics, Inc.
3220 Commander Drive
Carrollton 75006
Tel: (214) 380-6464
FAX: (214) 248-7208

Hamilton/Avnet Computer
1807A West Braker Lane
Austin 78758

Hamilton/Avnet Computer
Forum 2
4004 Beltline, Suite 200
Dallas 75244

Hamilton/Avnet Computer
4850 Wright Rd., Suite 190
Stafford 77477

†Hamilton/Avnet Electronics
1807 W. Braker Lane
Austin 78758
Tel: (512) 837-8911
TWX: 910-874-1319

†Hamilton/Avnet Electronics
4004 Beltline, Suite 200
Dallas 75234
Tel: (214) 308-8111
TWX: 910-860-5929

†Hamilton/Avnet Electronics
4850 Wright Rd., Suite 190
Stafford 77477
Tel: (713) 240-7733
TWX: 910-881-5523

†Pioneer/Standard Electronics
1826-D Kramer
Austin 78758
Tel: (512) 835-4000
FAX: 512-835-9829

†Pioneer/Standard Electronics
13710 Omega Road
Dallas 75244
Tel: (214) 386-7300
FAX: 214-490-6419

†Pioneer/Standard Electronics
10530 Rockley Road
Houston 77099
Tel: (713) 495-4700
FAX: 713-495-5642

†Wyle Distribution Group
1810 Greenville Avenue
Richardson 75081
Tel: (214) 235-9953
FAX: 214-644-5064

UTAH

Hamilton/Avnet Computer
1585 West 2100 South
Salt Lake City 84119

†Hamilton/Avnet Electronics
1585 West 2100 South
Salt Lake City 84119
Tel: (801) 972-2800
TWX: 910-925-4018

†Wyle Distribution Group
1325 West 2200 South
Suite E
West Valley 84119
Tel: (801) 974-9953

WASHINGTON

†Almac Electronics Corp.
14360 S.E. Eastgate Way
Bellevue 98007
Tel: (206) 643-9992
FAX: 206-643-9709

Hamilton/Avnet Computer
17761 Northeast 78th Place
Redmond 98052

†Hamilton/Avnet Electronics
17761 N.E. 78th Place
Redmond 98052
Tel: (206) 881-6697
FAX: 206-867-0159

Wyle Distribution Group
15385 N.E. 90th Street
Redmond 98052
Tel: (206) 881-1150
FAX: 206-881-1567

WISCONSIN

Arrow Electronics, Inc.
200 N. Patrick Blvd., Ste. 100
Brookfield 53005
Tel: (414) 792-0150
FAX: 414-792-0156

Hamilton/Avnet Computer
20875 Crossroads Circle
Suite 400
Waukesha 53186

†Hamilton/Avnet Electronics
28875 Crossroads Circle
Suite 400
Waukesha 53186
Tel: (414) 784-4510
FAX: 414-784-9509

CANADA

ALBERTA

Hamilton/Avnet Computer
2816 21st Street Northeast
Calgary T2E 6Z2

Hamilton/Avnet Electronics
2816 21st Street N.E. #3
Calgary T2E 6Z3
Tel: (403) 230-3586
FAX: 403-250-1591

Zentronics
6815 #8 Street N.E.
Suite 100
Calgary T2E 7H
Tel: (403) 295-8818
FAX: 403-295-8714

BRITISH COLUMBIA

†Hamilton/Avnet Electronics
8610 Commerce Ct.
Burnaby V5A 4N6
Tel: (604) 420-4101
FAX: 604-437-4712

Zentronics
108-11400 Bridgeport Road
Richmond V6X 1T2
Tel: (604) 273-5575
FAX: 604-273-2413

ONTARIO

Arrow Electronics, Inc.
36 Antares Dr., Unit 100
Nepean K2E 7W5
Tel: (613) 226-6903
FAX: 613-723-2018

†Arrow Electronics, Inc.
1093 Meyerside, Unit 2
Mississauga L5T 1M4
Tel: (416) 673-7769
FAX: 416-672-0849

Hamilton/Avnet Computer
Canada System Engineering Group
3688 Nashua Drive
Units 7 & 8
Mississuaga L4V 1M5

Hamilton/Avnet Computer
3688 Nashua Drive
Units 9 & 10
Mississuaga L4V 1M5

Hamilton/Avnet Computer
6845 Rexwood Road
Units 7, 8, & 9
Mississuaga L4V 1R2

Hamilton/Avnet Computer
190 Colonade Road
Nepean K2E 7J5

†Hamilton/Avnet Electronics
6845 Rexwood Road
Units 3-4-5
Mississauga L4T 1R2
Tel: (416) 677-7432
FAX: 416-677-0940

†Hamilton/Avnet Electronics
190 Colonnade Road South
Nepean K2E 7L5
Tel: (613) 226-1700
FAX: 613-226-1184

†Zentronics
1355 Meyerside Drive
Mississauga L5T 1C9
Tel: (416) 564-9600
FAX: 416-564-8320

†Zentronics
155 Colonnade Road
Unit 17
Nepean K2E 7K1
Tel: (613) 226-8840
FAX: 613-226-6352

QUEBEC

Arrow Electronics Inc.
1100 St. Regis
Dorval H9P 2T5
Tel: (514) 421-7411
FAX: 514-421-7430

Arrow Electronics, Inc.
500 Boul. St-Jean-Baptiste
Suite 280
Quebec G2E 5R9
Tel: (418) 871-7500
FAX: 418-871-6816

Hamilton/Avnet Computer
2795 Rue Halpern
St. Laurent H4S 1P8

†Hamilton/Avnet Electronics
2795 Halpern
St. Laurent H2E 7K1
Tel: (514) 335-1000
FAX: 514-335-2481

†Zentronics
520 McCaffrey
St. Laurent H4T 1N3
Tel: (514) 737-9700
FAX: 514-737-5212

†Certified Technical Distributor

EUROPEAN SALES OFFICES

FINLAND

Intel Finland OY
Ruosilantie 2
00390 Helsinki
Tel: (358) 0 544 644
TLX: 123332

FRANCE

Intel Corporation S.A.R.L.
1, Rue Edison-BP 303
78054 St. Quentin-en-Yvelines Cedex
Tel: (33) (1) 30 57 70 00
TLX: 699016

ISRAEL

Intel Semiconductor Ltd.
Atidim Industrial Park-Neve Sharet
P.O. Box 43202
Tel-Aviv 61430
Tel: (972) 03-498080
TLX: 371215

ITALY

Intel Corporation Italia S.p.A.
Milanofiori Palazzo E
20094 Assago
Milano
Tel: (39) (02) 89200950
TLX: 341286

NETHERLANDS

Intel Semiconductor B.V.
Postbus 84130
3099 CC Rotterdam
Tel: (31) 10.407.11.11
TLX: 22283

SPAIN

Intel Iberia S.A.
Zurbaran, 28
28010 Madrid
Tel: (34) (1) 308.25.52
TLX: 46880

SWEDEN

Intel Sweden A.B.
Dalvagen 24
171 36 Solna
Tel: (46) 8 734 01 00
TLX: 12261

SWITZERLAND

Intel Semiconductor A.G.
Zuerichstrasse
8185 Winkel-Rueti bei Zuerich
Tel: (41) 01/860 62 62
TLX: 825977

UNITED KINGDOM

Intel Corporation (U.K.) Ltd.
Pipers Way
Swindon, Wiltshire SN3 1RJ
Tel: (44) (0793) 696000
TLX: 444447/8

WEST GERMANY

Intel GmbH
Dornacher Strasse 1
8016 Feldkirchen bei Muenchen
Tel: (49) 089/90992-0
FAX: (49) 089/904/3948

Intel GmbH
Abraham Lincoln Strasse 16-18
6200 Wiesbaden
Tel: (49) 06121/7605-0
TLX: 4-186183

Intel GmbH
Zettachring 10A
7000 Stuttgart 80
Tel: (49) 0711/7287-280
TLX: 7-254826

EUROPEAN DISTRIBUTORS/REPRESENTATIVES

AUSTRIA

Bacher Electronics G.m.b.H.
Rotenmuehlgasse 26
1120 Wien
Tel: (43) (0222) 83 56 46
TLX: 31532

BELGIUM

Inelco Belgium S.A.
Av. des Croix de Guerre 94
1120 Bruxelles
Oorlogskruisenlaan, 94
1120 Brussel
Tel: (32) (02) 216 01 60
TLX: 64475 or 22090

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: (358) (0) 6926022
TLX: 124224

FRANCE

Almex
Zone industrielle d'Antony
48, rue de l'Aubepine
BP 102
92164 Antony Cedex
Tel: (33) (1) 40 96 54 00
TLX: 250067

LEX Electronics
73-79, Rue des Solets
Silic 585
94663 Rungis Cedex
Tel: (33) (1) 49 78 48 78
TWX: 200485

Metrologie
Tour d'Asnieres
4, av. Laurent-Cely
92606 Asnieres Cedex
Tel: (33) (1) 47 90 62 40
TLX: 611448

Tekelec-Airtronic
Cite des Bruyeres
Rue Carle Vernet - BP 2
92310 Sevres
Tel: (33) (1) 45 34 75 35
TLX: 204552

IRELAND

Micro Marketing Ltd.
Glenageary Office Park
Glenageary
Co. Dublin
Tel: (21) (353) (01) 856288
FAX: (21) (353) (01) 857364
TLX: 31584

ISRAEL

Eastronics Ltd.
11 Rozanis Street
P.O.B. 39300
Tel-Aviv 61392
Tel: (972) 03-475151
TLX: 33638

ITALY

Intesi
Divisione ITT Industries GmbH
Viale Milanofiori
Palazzo E/5
20090 Assago (MI)
Tel: (39) 02/824701
TLX: 311351

Lasi Elettronica S.p.A.
V. le Fulvio Testi, 126
20092 Cinisello Balsamo (MI)
Tel: (39) 02/2440012
TLX: 352040

Telcom S.r.l.
Via M. Civitali 75
20148 Milano
Tel: (39) 02/4049046
TLX: 335654

ITT Multicomponents
Viale Milanofiori E/5
20090 Assago (MI)
Tel: (39) 02/824701
TLX: 311351

Silverstar
Via Dei Gracchi 20
20146 Milano
Tel: (39) 02/49961
TLX: 332189

NETHERLANDS

Koning en Hartman
Elektrotechniek B.V.
Energieweg 1
2627 AP Delft
Tel: (31) (1) 15/609906
TLX: 38250

NORWAY

Nordisk Elektronikk (Norge) A/S
Postboks 123
Smedsvingen 4
1364 Hvalstad
Tel: (47) (02) 84 62 10
TLX: 77546

PORTUGAL

ATD Portugal LDA
Rua Dr. Faria de Vasconcelos, 3 A
1900 Lisboa
Tel: 351 1 847 22 00
FAX: 351 1 847 21 97

SPAIN

ATD Electronica, S.A.
Plaza Ciudad de Viena, 6
28040 Madrid
Tel: (34) (1) 234 40 00
TLX: 42477

Metrologia Iberica, S.A.
Ctra. de Fuencarral, n.80
28100 Alcobendas (Madrid)
Tel: (34) (1) 653 86 11

SWEDEN

Nordisk Elektronik AB
Torshamnsgatan 39
Box 36
164 93 Kista
Tel: (46) 08-03 46 30
TLX: 105 47

SWITZERLAND

Industrade A.G.
Hertistrasse 31
8304 Wallisellen
Tel: (41) (01) 8328111
TLX: 56788

TURKEY

EMPA Electronic
Lindwurmstrasse 95A
8000 Muenchen 2
Tel: (49) 089/53 80 570
TLX: 528573

UNITED KINGDOM

Access Electronic Components Ltd.
Jubilee House, Jubilee Road
Letchworth, Herts SG6 1QH
Tel: (0462) 480888
FAX: (0462) 682467

Bytech Components Ltd.
12A Cedarwood
Chineham Business Park
Crockford Lane
Basingstoke
Hants RG24 0WD
Tel: (0256) 707107
FAX: 0256-707162

Conformix
Rapid House
Oxford Road
High Wycombe
Bucks HP11 2EE
Tel: (0494) 474147
FAX: (0494) 452144

Bytech Systems
Unit 3
The Western Centre
Western Road
Bracknell
Berks RG12 1RW
Tel: (0344) 55333
FAX: (0344) 867270

Jermyn
Vestry Estate
Otford Road
Sevenoaks
Kent TN14 5EU
Tel: (0732) 450144
FAX: (0732) 451251

MMD Ltd.
3 Bennet Court
Bennet Road
Reading
Berks RG2 0QX
Tel: (0734) 313232
FAX: (0734) 313255

Metro Systems
Rapid House
Oxford Road
High Wycombe
Bucks HP11 2EE
Tel: 0494 474171
FAX: 0494 21860

Micro Marketing
Taney Hall
Eglington Terrace
Dundrum
Dublin 14
Eire
Tel: 0001 989 400
FAX: 0001 989 828

Rapid Silicon
3 Bennet Court
Bennet Road
Reading
Berks RG2 0QX
Tel: 0734 752266
FAX: 0734 312728

WEST GERMANY

Electronic 2000 AG
Stahlgruberring 12
8000 Muenchen 82
Tel: (49) 089/42001-0
TLX: 522561

ITT Multikomponent GmbH
Postfach 1265
Bahnhofstrasse 44
7141 Moeglingen
Tel: (49) 07141/4879
TLX: 7264472

Jermyn GmbH
Im Dachsstueck 9
6250 Limburg
Tel: (49) 06431/508-0
TLX: 415257-0

Metrologie GmbH
Meglingerstrasse 49
8000 Muenchen 71
Tel: (49) 089/78042-0
TLX: 5213189

Proelectron Vertriebs GmbH
Max Planck Strasse 1-3
6072 Dreieich
Tel: (49) 06103/30434-3
TLX: 417903

YUGOSLAVIA

H.R. Microelectronics Corp.
2005 de la Cruz Blvd., Ste. 223
Santa Clara, CA 95050
U.S.A.
Tel: (1) (408) 988-0286
TLX: 387452

Rapido Electronic Components S.p.a.
Via C. Beccaria, 8
34133 Trieste
Italia
Tel: (39) 040/360555
TLX: 460461

CG/SALE/022891

INTERNATIONAL SALES OFFICES

AUSTRALIA

Intel Australia Pty. Ltd.
Unit 13
Allambie Grove Business Park
25 Frenchs Forest Road East
Frenchs Forest, NSW, 2086
Tel: 61-2975-3300
FAX: 61-2975-3375

BRAZIL

Intel Semiconductores do Brazil LTDA
Avenida Paulista, 1159-CJS 404/405
01311 - Sao Paulo - S.P.
Tel: 55-11-287-5899
TLX: 11-37-557-ISDB
FAX: 55-11-287-5119

CHINA/HONG KONG

Intel PRC Corporation
15/F, Office 1, Citic Bldg.
Jian Guo Men Wai Street
Beijing, PRC
Tel: (1) 500-4850
TLX: 22947 INTEL CN
FAX: (1) 500-2953

Intel Semiconductor Ltd.*
10/F East Tower
Bond Center
Queensway, Central
Hong Kong
Tel: (852) 844-4555
FAX: (852) 868-1989

INDIA

Intel Asia Electronics, Inc.
4/2, Samrah Plaza
St. Mark's Road
Bangalore 560001
Tel: 91-812-215773
TLX: 953-845-2646 INTEL IN
FAX: 091-812-215067

JAPAN

Intel Japan K.K.
5-6 Tokodai, Tsukuba-shi
Ibaraki, 300-26
Tel: 0298-47-8511
TLX: 3656-160
FAX: 0298-47-8450

Intel Japan K.K.*
Hachioji ON Bldg.
4-7-14 Myojin-machi
Hachioji-shi, Tokyo 192
Tel: 0426-48-8770
FAX: 0426-48-8775

Intel Japan K.K.*
Bldg. Kumagaya
2-69 Hon-cho
Kumagaya-shi, Saitama 360
Tel: 0485-24-6871
FAX: 0485-24-7518

Intel Japan K.K.*
Kawa-asa Bldg.
2-11-5 Shin-Yokohama
Kohoku-ku, Yokohama-shi
Kanagawa, 222
Tel: 045-474-7661
FAX: 045-471-4394

Intel Japan K.K.*
Ryokuchi-Eki Bldg.
2-4-1 Terauchi
Toyonaka-shi, Osaka 560
Tel: 06-863-1091
FAX: 06-863-1084

Intel Japan K.K.
Shinmaru Bldg.
1-5-1 Marunouchi
Chiyoda-ku, Tokyo 100
Tel: 03-3201-3621
FAX: 03-3201-6850

Intel Japan K.K.
Green Bldg.
1-16-20 Nishiki
Naka-ku, Nagoya-shi
Aichi 450
Tel: 052-204-1261
FAX: 052-204-1285

KOREA

Intel Korea, Ltd.
16th Floor, Life Bldg.
61 Yoido-dong, Youngdeungpo-Ku
Seoul 150-010
Tel: (2) 784-8186
FAX: (2) 784-8096

SINGAPORE

Intel Singapore Technology, Ltd.
101 Thomson Road #08-03/06
United Square
Singapore 1130
Tel: (65) 250-7811
FAX: (65) 250-9256

TAIWAN

Intel Technology Far East Ltd.
Taiwan Branch Office
8th Floor, No. 205
Bank Tower Bldg.
Tung Hua N. Road
Taipei
Tel: 886-2-716-9660
FAX: 886-2-717-2455

INTERNATIONAL DISTRIBUTORS/REPRESENTATIVES

ARGENTINA

Dafsys S.R.L.
Chacabuco, 90-6 Piso
1069-Buenos Aires
Tel: 54-1-34-7726
FAX: 54-1-34-1871

AUSTRALIA

Email Electronics
15-17 Hume Street
Huntingdale, 3166
Tel: 011-61-3-544-8244
TLX: AA 30895
FAX: 011-61-3-543-8179

NSD-Australia
205 Middleborough Rd.
Box Hill, Victoria 3128
Tel: 03 8900970
FAX: 03 8990819

BRAZIL

Elebra Componentes
Rua Geraldo Flausina Gomes, 78
7 Andar
04575 - Sao Paulo - S.P.
Tel: 55-11-534-9641
TLX: 55-11-54593/54591
FAX: 55-11-534-9424

CHINA/HONG KONG

Novel Precision Machinery Co., Ltd.
Room 728 Trade Square
681 Cheung Sha Wan Road
Kowloon, Hong Kong
Tel: (852) 360-8999
TWX: 32032 NVTNL HX
FAX: (852) 725-3695

INDIA

Micronic Devices
Arun Complex
No. 65 D.V.G. Road
Basavanagudi
Bangalore 560 004
Tel: 011-91-812-600-631
011-91-812-611-365
TLX: 9538458332 MDBG

Micronic Devices
No. 516 5th Floor
Swastik Chambers
Sion, Trombay Road
Chembur
Bombay 400 071
TLX: 9531 171447 MDEV

Micronic Devices
25/8, 1st Floor
Bada Bazaar Marg
Old Rajinder Nagar
New Delhi 110 060
Tel: 011-91-11-5723509
011-91-11-589771
TLX: 031-63253 MDND IN

Micronic Devices
6-3-348/12A Dwarakapuri Colony
Hyderabad 500 482
Tel: 011-91-842-226748

S&S Corporation
1587 Kooser Road
San Jose, CA 95118
Tel: (408) 978-6216
TLX: 820281
FAX: (408) 978-8635

JAPAN

Asahi Electronics Co. Ltd.
KMM Bldg. 2-14-1 Asano
Kokurakita-ku
Kitakyushu-shi 802
Tel: 093-511-6471
FAX: 093-551-7861

CTC Components Systems Co., Ltd.
4-8-1 Dobashi, Miyamae-ku
Kawasaki-shi, Kanagawa 213
Tel: 044-852-5121
FAX: 044-877-4268

Dia Semicon Systems, Inc.
Flower Hill Shinmachi Higashi-kan
1-23-9 Shinmachi, Setagaya-ku
Tokyo 154
Tel: 03-3439-1600
FAX: 03-3439-1601

Okaya Koki
2-4-18 Sakae
Naka-ku, Nagoya-shi 460
Tel: 052-204-2916
FAX: 052-204-2901

Ryoyo Electro Corp.
Konwa Bldg.
1-12-22 Tsukiji
Chuo-ku, Tokyo 104
Tel: 03-3546-5011
FAX: 03-3546-5044

KOREA

J-Tek Corporation
Dong Sung Bldg. 9/F
158-24, Samsung-Dong, Kangnam-Ku
Seoul 135-090
Tel: (822) 557-8039
FAX: (822) 557-8304

Samsung Electronics
Samsung Main Bldg.
150 Taepyung-Ro-2KA, Chung-Ku
Seoul 100-102
C.P.O. Box 8780
Tel: (822) 751-3680
TWX: KORSST K 27970
FAX: (822) 753-9065

MEXICO

SSB Electronics, Inc.
675 Palomar Street, Bldg. 4, Suite A
Chula Vista, CA 92011
Tel: (619) 585-3253
TLX: 287751 CBALL UR
FAX: (619) 585-8322

Dicopel S.A.
Tochtli 368 Fracc. Ind. San Antonio
Azcapotzalco
C.P. 02760-Mexico, D.F.
Tel: 52-5-561-3211
TLX: 177 3790 Dicome
FAX: 52-5-561-1279

PSI S.A. de C.V.
Fco. Villa esq. Ajusco s/n
Cuernavaca – Morelos
Tel: 52-73-13-9412
FAX: 52-73-17-5333

NEW ZEALAND

Email Electronics
36 Olive Road
Penrose, Auckland
Tel: 011-64-9-591-155
FAX: 011-64-9-592-681

SAUDI ARABIA

AAE Systems, Inc.
642 N. Pastoria Ave.
Sunnyvale, CA 94086
U.S.A.
Tel: (408) 732-1710
FAX: (408) 732-3095
TLX: 494-3405 AAE SYS

SINGAPORE

Electronic Resources Pte, Ltd.
17 Harvey Road
#03-01 Singapore 1336
Tel: (65) 283-0888
TWX: RS 56541 ERS
FAX: (65) 289-5327

SOUTH AFRICA

Electronic Building Elements
178 Erasmus St. (off Watermeyet St.)
Meyerspark, Pretoria, 0184
Tel: 011-2712-803-7680
FAX: 011-2712-803-8294

TAIWAN

Micro Electronics Corporation
12th Floor, Section 3
285 Nanking East Road
Taipei, R.O.C.
Tel: (886) 2-7198419
FAX: (886) 2-7197916

Acer Sertek Inc.
15th Floor, Section 2
Chien Kuo North Rd.
Taipei 18479 R.O.C.
Tel: 886-2-501-0055
TWX: 23756 SERTEK
FAX: (886) 2-5012521

*Field Application Location

DOMESTIC SERVICE OFFICES

ALASKA

Intel Corp.
c/o TransAlaska Network
1515 Lore Rd.
Anchorage 99507
Tel: (907) 522-1776

Intel Corp.
c/o TransAlaska Data Systems
c/o GCI Operations
520 Fifth Ave., Suite 407
Fairbanks 99701
Tel: (907) 452-6264

ARIZONA

*Intel Corp.
410 North 44th Street
Suite 500
Phoenix 85008
Tel: (602) 231-0386
FAX: (602) 244-0446

*Intel Corp.
500 E. Fry Blvd., Suite M-15
Sierra Vista 85635
Tel: (602) 459-5010

ARKANSAS

Intel Corp.
c/o Federal Express
1500 West Park Drive
Little Rock 72204

CALIFORNIA

*Intel Corp.
21515 Vanowen St., Ste. 116
Canoga Park 91303
Tel: (818) 704-8500

*Intel Corp.
300 N. Continental Blvd.
Suite 100
El Segundo 90245
Tel: (213) 640-6040

*Intel Corp.
1900 Prairie City Rd.
Folsom 95630-9597
Tel: (916) 351-6143

*Intel Corp.
9665 Chesapeake Dr., Suite 325
San Diego 92123
Tel: (619) 292-8086

**Intel Corp.
400 N. Tustin Avenue
Suite 450
Santa Ana 92705
Tel: (714) 835-9642

**Intel Corp.
2700 San Tomas Exp., 1st Floor
Santa Clara 95051
Tel: (408) 970-1747

COLORADO

*Intel Corp.
600 S. Cherry St., Suite 700
Denver 80222
Tel: (303) 321-8086

CONNECTICUT

*Intel Corp.
301 Lee Farm Corporate Park
83 Wooster Heights Rd.
Danbury 06811
Tel: (203) 748-3130

FLORIDA

**Intel Corp.
800 Fairway Dr., Suite 160
Deerfield Beach 33441
Tel: (305) 421-0506
FAX: (305) 421-2444

*Intel Corp.
5850 T.G. Lee Blvd., Ste. 340
Orlando 32822
Tel: (407) 240-8000

GEORGIA

*Intel Corp.
20 Technology Park, Suite 150
Norcross 30092
Tel: (404) 449-0541

5523 Theresa Street
Columbus 31907

HAWAII

**Intel Corp.
Honolulu 96820
Tel: (808) 847-6738

ILLINOIS

**†Intel Corp.
Woodfield Corp. Center III
300 N. Martingale Rd., Ste. 400
Schaumburg 60173
Tel: (708) 605-8031

INDIANA

*Intel Corp.
8910 Purdue Rd., Ste. 350
Indianapolis 46268
Tel: (317) 875-0623

KANSAS

*Intel Corp.
10985 Cody, Suite 140
Overland Park 66210
Tel: (913) 345-2727

KENTUCKY

Intel Corp.
133 Walton Ave., Office 1A
Lexington 40508
Tel: (606) 255-2957

Intel Corp.
896 Hillcrest Road, Apt. A
Radcliff 40160 (Louisville)

LOUISIANA

Hammond 70401
(serviced from Jackson, MS)

MARYLAND

**Intel Corp.
10010 Junction Dr., Suite 200
Annapolis Junction 20701
Tel: (301) 206-2860

MASSACHUSETTS

**Intel Corp.
Westford Corp. Center
3 Carlisle Rd., 2nd Floor
Westford 01886
Tel: (508) 692-0960

MICHIGAN

*Intel Corp.
7071 Orchard Lake Rd., Ste. 100
West Bloomfield 48322
Tel: (313) 851-8905

MINNESOTA

*Intel Corp.
3500 W. 80th St., Suite 360
Bloomington 55431
Tel: (612) 835-6722

MISSISSIPPI

Intel Corp.
c/o Compu-Care
2001 Airport Road, Suite 205F
Jackson 39208
Tel: (601) 932-6275

MISSOURI

*Intel Corp.
4203 Earth City Exp., Ste. 131
Earth City 63045
Tel: (314) 291-1990

Intel Corp.
Route 2, Box 221
Smithville 64089
Tel: (913) 345-2727

NEW JERSEY

**Intel Corp.
300 Sylvan Avenue
Englewood Cliffs 07632
Tel: (201) 567-0821

*Intel Corp.
Lincroft Office Center
125 Half Mile Road
Red Bank 07701
Tel: (908) 747-2233

NEW MEXICO

Intel Corp.
Rio Rancho 1
4100 Sara Road
Rio Rancho 87124-1025
(near Albuquerque)
Tel: (505) 893-7000

NEW YORK

*Intel Corp.
2950 Expressway Dr. South
Suite 130
Islandia 11722
Tel: (516) 231-3300

Intel Corp.
300 Westage Business Center
Suite 230
Fishkill 12524
Tel: (914) 897-3860

Intel Corp.
5858 East Molloy Road
Syracuse 13211
Tel: (315) 454-0576

NORTH CAROLINA

*Intel Corp.
5800 Executive Center Drive
Suite 105
Charlotte 28212
Tel: (704) 568-8966

**Intel Corp.
5540 Centerview Dr., Suite 215
Raleigh 27606
Tel: (919) 851-9537

OHIO

**Intel Corp.
3401 Park Center Dr., Ste. 220
Dayton 45414
Tel: (513) 890-5350

*Intel Corp.
25700 Science Park Dr., Ste. 100
Beachwood 44122
Tel: (216) 464-2736

OREGON

**Intel Corp.
15254 N.W. Greenbrier Pkwy.
Building B
Beaverton 97006
Tel: (503) 645-8051

PENNSYLVANIA

*†Intel Corp.
925 Harvest Drive
Suite 200
Blue Bell 19422
Tel: (215) 641-1000
1-800-468-3548
FAX: (215) 641-0785

**†Intel Corp.
400 Penn Center Blvd., Ste. 610
Pittsburgh 15235
Tel: (412) 823-4970

*Intel Corp.
1513 Cedar Cliff Dr.
Camp Hill 17011
Tel: (717) 761-0860

PUERTO RICO

Intel Corp.
South Industrial Park
P.O. Box 910
Las Piedras 00671
Tel: (809) 733-8616

TEXAS

**Intel Corp.
Westech 360, Suite 4230
8911 N. Capitol of Texas Hwy.
Austin 78752-1239
Tel : (512) 794-8086

**†Intel Corp.
12000 Ford Rd., Suite 401
Dallas 75234
Tel: (214) 241-8087

**Intel Corp.
7322 SW Freeway, Suite 1490
Houston 77074
Tel: (713) 988-8086

UTAH

Intel Corp.
428 East 6400 South
Suite 104
Murray 84107
Tel: (801) 263-8051
FAX: (801) 268-1457

VIRGINIA

*Intel Corp.
9030 Stony Point Pkwy.
Suite 360
Richmond 23235
Tel: (804) 330-9393

WASHINGTON

**Intel Corp.
155 108th Avenue N.E., Ste. 386
Bellevue 98004
Tel: (206) 453-8086

CANADA

ONTARIO

**Intel Semiconductor of
Canada, Ltd.
2650 Queensview Dr., Ste. 250
Ottawa K2B 8H6
Tel: (613) 829-9714

**Intel Semiconductor of
Canada, Ltd.
190 Attwell Dr., Ste. 102
Rexdale (Toronto) M9W 6H8
Tel: (416) 675-2105

QUEBEC

**Intel Semiconductor of
Canada, Ltd.
1 Rue Holiday
Suite 115
Tour East
Pt. Claire H9R 5N3
Tel: (514) 694-9130
FAX: 514-694-0064

CUSTOMER TRAINING CENTERS

CALIFORNIA

2700 San Tomas Expressway
Santa Clara 95051
Tel: 1-800-328-0386

MARYLAND

10010 Junction Dr.
Suite 200
Annapolis Junction 20701
Tel: 1-800-328-0386

SYSTEMS ENGINEERING OFFICES

MINNESOTA

3500 W. 80th Street
Suite 360
Bloomington 55431
Tel: (612) 835-6722

NEW YORK

2950 Expressway Dr., South
Islandia 11722
Tel: (506) 231-3300

*Carry-in locations
**Carry-in/mail-in locations

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 Corporation S.A.R.L.
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

GERMANY
Intel GmbH
Dornacher Strasse 1
8016 Feldkirchen bei Muenchen

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 U.S.A./15K/0891/RRD MW
MICROPROCESSORS

ISBN 1-55512-153-5