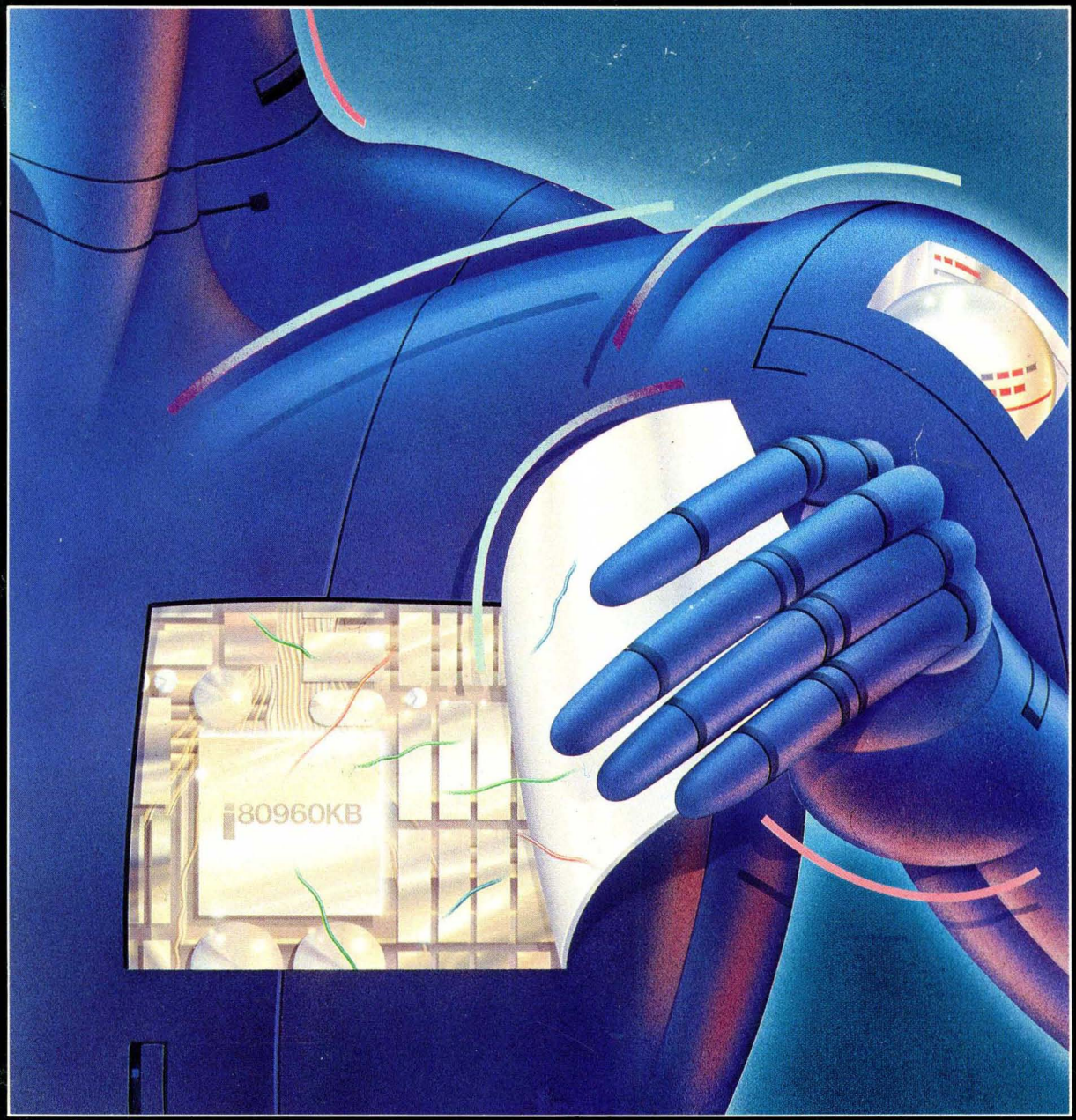
intel

80960KB Hardware Designer's Reference Manual



Order Number: 270564-001

LITERATURE

To order Intel literature write or call:

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

Toll Free Number:
(800) 548-4725*

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

1988 HANDBOOKS

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

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

*Good in the U.S. and Canada

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

LITERATURE SALES ORDER FORM

NAME: ______________________________

COMPANY: ______________________________

ADDRESS: ______________________________

CITY: ______________ STATE: ________ ZIP: ________

COUNTRY: ______________________________

PHONE NO.: (______) ______________________

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

Subtotal ________

Must Add Your
Local Sales Tax ________

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

Total ________

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

☐ Visa ☐ MasterCard ☐ American Express Expiration Date ________

Account No. ______________________________

Signature: ______________________________

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

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

European Literature Order Form in back of book.

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

Prices good until 12/31/88.

Source HB

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

80960KB HARDWARE DESIGNER'S REFERENCE MANUAL

1988

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

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

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

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

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

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

*MULTIBUS is a patented Intel bus.

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

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

TABLE OF CONTENTS

Figures

Tables

Preface

PREFACE

This manual serves as the definitive hardware reference guide for system designs using the 80960KB and 80960KA processors (for clarity, references to the 80960KB apply to both the 80960KA and 80960KB processors). Hardware designers can use this manual as a guideline for developing microprocessor systems. Readers of this manual should be familiar with the operating principles of microprocessors and with the 80960KB data sheet.

This manual presents the 80960KB system design from a hardware perspective. Other information on the software architecture, instruction set, and programming of the 80960KB processor can be found in the *80960KB CPU Programmer's Reference Manual.*

Together with the *80960KB Hardware Designer's Reference Manual*, these publications provide a complete description of the 80960KB system for hardware and software designers.

MANUAL ORGANIZATION

This manual, divided into five chapters, describes how to build a hardware system using the 80960KB processor. The list below shows a synopsis of each chapter.

- Chapter 1 briefly introduces the 80960KB component architecture.
- Chapter 2 presents an overview of the 80960KB hardware system design, which includes a system configuration illustrating the various components that constitute a 80960KB system.
- Chapter 3 describes the local bus and the interface to the 80960KB processor. This chapter includes detailed signal descriptions and discusses timing generation, arbitration, interrupt handling, and initialization.
- Chapter 4 discusses techniques for designing memory subsystems.
- Chapter 5 presents guidelines on how to interface I/O devices to the local bus.

Wherever appropriate, design examples are included in the chapters. These designs are based upon functional 80960KB boards and systems, and are simplified for ease of understanding. The simplified versions of these designs have not been tested except for the figures that show the part numbers.

NOTATION CONVENTIONS

This manual uses the following style conventions.

- Integer numbers are presented in decimal notation unless otherwise indicated by the subscript "H" for hexidecimal or "B" for binary.
- An active low signal is represented by a line over the signal name. For example, $\overline{\text{READY}}$ is an active low signal.

Introduction to the 80960KB Microprocessor

1

CHAPTER 1
INTRODUCTION TO THE 80960KB MICROPROCESSOR

The 80960KB is the first 32-bit microprocessor designed especially for embedded applications. At an operating frequency of 20 MHz, this high performance processor can sustain an instruction execution rate of seven and one-half million instructions per second (MIPS), and burst rates of 20 MIPS[1]. The 80960KB processor enhances embedded system performance by integrating special features to eliminate the need for additional peripheral devices and the associated software overhead. For instance, the 80960KB processor offers an on-chip floating-point processing unit, an improved interrupt handling capability, and support for debugging and tracing.

This chapter describes the architectural attributes and enhancements of the 80960KB processor for embedded computing.

ARCHITECTURAL ATTRIBUTES FOR EMBEDDED COMPUTING

For over a decade, Intel has designed a large variety of 8- and 16-bit microcontrollers to fit the needs of embedded applications. Based on this experience, several architectural attributes shared by both microcontrollers and microprocessors can be implemented that benefit embedded applications and enhance microprocessor performance. Because the 80960KB processor incorporates these attributes (listed below) in its architecture, embedded applications are easy to design, perform well, and get to market fast.

- Simple load/store design
- Large general-purpose register sets
- Boolean and bit-field instructions
- Small number of operations and addressing modes
- Simplified instruction format
- Minimum cycle operation

Load/Store Design

In the 80960 family architecture, operations are register-to-register, with only LOAD and STORE instructions accessing memory. This attribute simplifies the instruction set and shortens cycle time.

The 80960KB processor uses LOAD and STORE instructions to access memory. It further minimizes accesses to memory by providing a 512-byte, direct-mapped instruction cache. When a memory access is required, the processor can perform a burst transaction that accesses up to four data words with one word transferred every clock cycle.

---

[1]DEC VAX 11/780 equals 1 MIPS

Large General-Purpose Register Sets

Because the instructions operate on operands within registers, the 80960 family uses many registers. The 80960KB processor features large, versatile register sets. For maximum flexibility, each processor provides 32 32-bit registers and four 80-bit floating-point registers.

There are two types of general-purpose registers: local and global. The processor automatically accesses the 16 local registers when a procedure call is performed. Multiple sets of local registers are stored on-chip to further increase the efficiency of this register set, as shown in Figure 1-1. The register cache holds up to four local register frames, which means that up to three procedure calls can be made without having to access the procedure stack resident in memory.

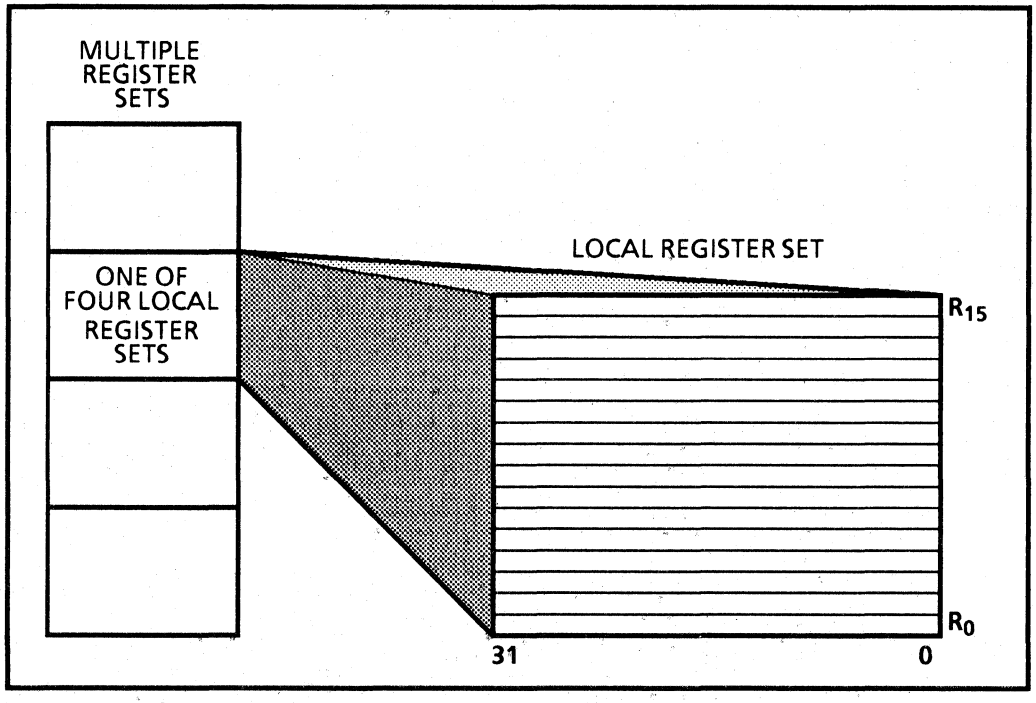


Figure 1-1: Local Register Set

The 20 global registers retain their contents across procedure boundaries. The global registers consist of sixteen 32-bit registers (G_{15} through G_0) and four 80-bit registers (FP_3 through FP_0), as shown in Figure 1-2. While all registers can be used for floating-point operations, the 80-bit registers are used for accumulation of extended precision results.

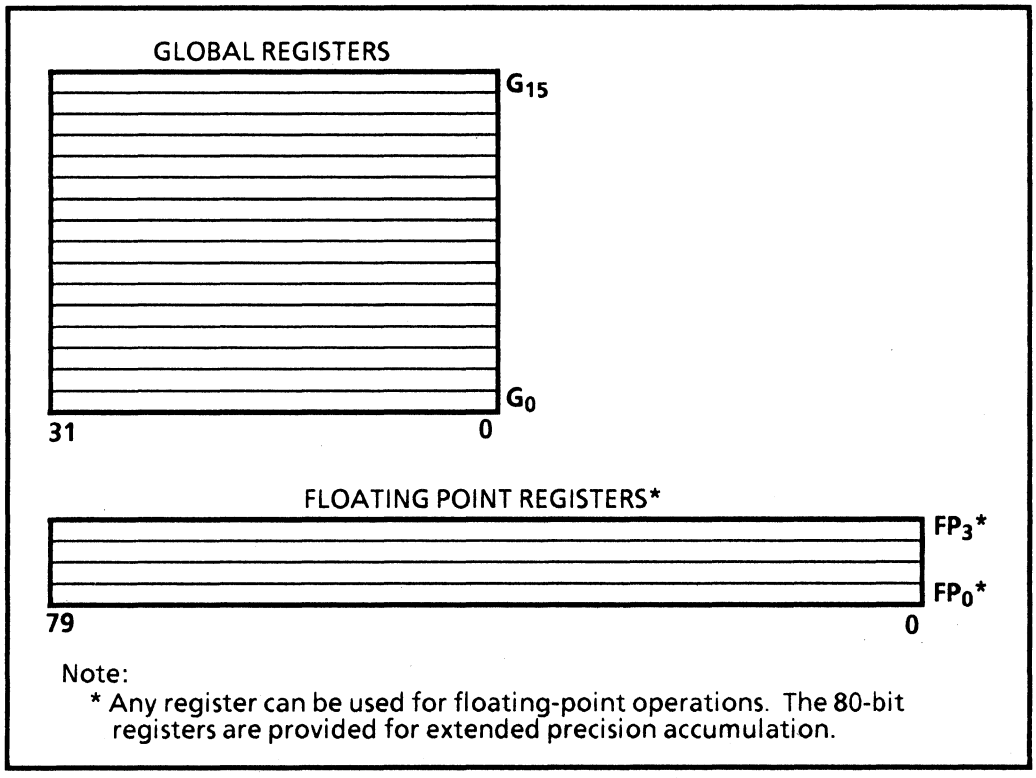


Figure 1-2: Global Register Set

Small Number of Addressing Modes

The 80960 family uses relatively few addressing modes to facilitate a fast, simple interpretation by the control engine. The 80960KB processor provides simple, fast addressing modes, as well as a few complex addressing modes to allow optimizations for code density.

Simplified Instruction Format

A simplified instruction format eases the hardwired decoding of instructions, which again speeds control paths. The 80960KB processor's instruction formats are simple and word aligned; all instructions are one word long except for one class that uses the subsequent word as a 32-bit displacement. To further enhance performance, the instructions do not cross word boundaries. This feature eliminates a pipeline stage (that would have to align instructions) and decreases instruction execution time.

Overlapped Execution

To optimize performance, the 80960KB processor overlaps instruction execution by means of write buffering and register scoreboarding. Write buffering allows a write instruction to proceed as soon as it is placed in the buffer. It does not have to wait for the actual write operation to occur on the L-bus.

Similarly, register scoreboarding is a design technique that allows the 80960KB to continue execution of instructions when it encounters a LOAD instruction. When the LOAD instruction begins, the 80960KB sets a scoreboard bit on the target register. After the target register is loaded with data, the processor resets the bit. While the data is being retrieved, additional instructions that do not reference the target register can be executed. The 80960KB ensures that these additional instructions do not reference the target register by checking the scoreboard bit transparently (no software required). Thus, the scoreboard feature reduces the effect of slow memory speed and provides a useful tool for optimizing procedures.

Minimum cycle operation

The 80960KB processor executes most of the core instructions in a single clock cycle. For these instructions, the 80960KB processor uses hardwired logic rather than microcode to execute the instruction.

The 80960KB also supports a number of important multicycle instructions, such as 32-bit multiply and divide instructions. These auxiliary functions require more than one clock cycle because it is more efficient to use microcode than hardwired logic. On the other hand, the integration of these functions on-chip eliminates much software overhead and the negative effects on code density that would be otherwise required. Thus, the additional functionality of the 80960KB enhances overall system performance while keeping code size small.

ADDITIONAL 80960KB ARCHITECTURAL ENHANCEMENTS

The 80960KB incorporates two useful features: an on-chip floating-point processing and debugging functions. The floating-point unit can be used for applications that require precision, such as machine-control operations. The debugging function significantly decreases development time.

Floating-point Operation

The on-chip floating-point unit of each processor improves the performance of floating-point calculations by eliminating bus overhead used to transfer operands to a coprocessor. The processor provides hardware support for both mandatory and recommended portions of IEEE standard 754 for floating-point arithmetic, exponential, logarithmic, and other transcendental functions. By integrating the floating-point unit on-chip, the 80960KB processor reduces the overall chip count for a system, decreases power consumption, and increases overall performance and reliability.

Debug Capabilities

The processor provides extensive system debug capabilities, an important feature for embedded computing where the ability to instrument an application may be limited. The 80960KB processor allows breakpoint instructions that stop program execution on various events, such as procedure calls, or certain instructions. Another debug facility traces the activity of the processor while it is executing a program. Tracing is done by recording the addresses of instructions that cause trace events to occur. For example, a trace event can occur on the execution of a specific instruction, branch, or procedure call. To ensure that the 80960KB is operating properly, the processor performs a self-test when it is reset. If the self-test is successful, the 80960KB begins operation, otherwise it enters the stopped state.

STANDARD BUS INTERFACE

The advanced features of the 80960KB processor are implemented using a performance-optimized bus interface. The processor uses a high bandwidth local bus (L-bus) that consists of standard signal groups: a 32-bit multiplexed address/data path and control signals for data transactions. Because of the large amount of caching, the L-bus supports burst transactions that transfer up to four successive data words. Transactions on the L-bus can use 8-, 16-, and 32-bit data types and address up to 4G bytes of physical memory. Bus arbitration can be accomplished by simply using the hold request/hold acknowledge protocol.

INTER-AGENT COMMUNICATION/COPROCESSOR CAPABILITIES

The 80960KB processor offers a flexible way to manage interrupts. It accepts interrupts in one of three ways: by communicating with an external interrupt controller using the standard Interrupt/Interrupt Acknowledge signals, by activating the on-chip interrupt controller, or by accepting an inter-agent communication (IAC) message. This allows the 80960KB to act as a coprocessor on a shared bus with another CPU.

SUMMARY

The 80960KB processor optimizes embedded system performance by using a new 32-bit architecture. The 80960 family architecture includes a load/store design, large general purpose register sets, fast addressing modes, a simplified instruction format, and minimized instruction execution cycles.

To further enhance system performance, the 80960KB processor provides floating-point operation, interrupt controller capabilities, and debug functions. By intergrating these functions on-chip, the 80960KB reduces the power requirements and overall chip count for a system.

As a result of the 80960 architecture, the 80960KB processor provides unprecedented performance. For a speed selection of 20 MHz, it can sustain an instruction execution rate of over seven and one-half million MIPS and burst rates of 20 MIPS, speeds comparable to that of super minicomputers. The high instruction execution rates are made possible through a innovative design that incorporates an on-chip instruction cache with burst-transfer capability.

80960KB System Architecture

2

CHAPTER 2
80960KB SYSTEM ARCHITECTURE

This chapter illustrates the flexibility and power of the 80960KB system architecture using the advanced 32-bit 80960KB processor. This chapter examines system configurations from a general perspective to explain the design concepts. Subsequent chapters describe the details of the system design.

OVERVIEW OF A SINGLE PROCESSOR SYSTEM ARCHITECTURE

The central processing module, memory module, and I/O module form the natural boundaries for the hardware system architecture. The modules are connected together by the high bandwidth 32-bit multiplexed L-bus, which can transfer data at a maximum sustained rate of 53M bytes per second for an 80960KB processor operating at 20 MHz.

Figure 2-1 shows a simplified block diagram of a possible system configuration. The heart of this system is the 80960KB processor, which fetches program instructions, executes code, manipulates stored information, and interacts with I/O devices. The high bandwidth L-bus connects the 80960KB processor to memory and I/O modules. The 80960KB processor stores system data and instructions and programs in the memory module. By accessing various peripheral devices in the I/O module, the 80960KB processor supports communication to terminals, modems, printers, disks, and other I/O devices.

80960KB Processor and the L-Bus

The 80960KB processor performs bus operations using multiplexed address and data signals and provides all the necessary control signals. For example, standard control signals, such as Address Latch Enable ($\overline{ALE}$), Address/Data Status ($\overline{ADS}$), Write/Read command (W/$\overline{R}$), Data Transmit/Receive (DT/$\overline{R}$), and Data Enable ($\overline{DEN}$) are provided by the 80960KB processor. The 80960KB processor also generates byte enable signals that specify which bytes on the 32-bit data lines are valid for the transfer.

The L-bus supports burst transactions, which access up to four data words at a maximum rate of one word per clock cycle. The 80960KB processor uses the two low-order address lines to indicate how many words are to be transferred. The 80960KB processor performs burst transactions to load the on-chip 512-byte instruction cache to minimize memory accesses for instruction fetches. Burst transactions can also be used for data accesses.

To transfer control of the bus to an external bus master, the 80960KB processor provides two arbitration signals: hold request (HOLD) and hold acknowledge (HLDA). After receiving HOLD, the processor grants control of the bus to an external bus master by asserting HLDA.

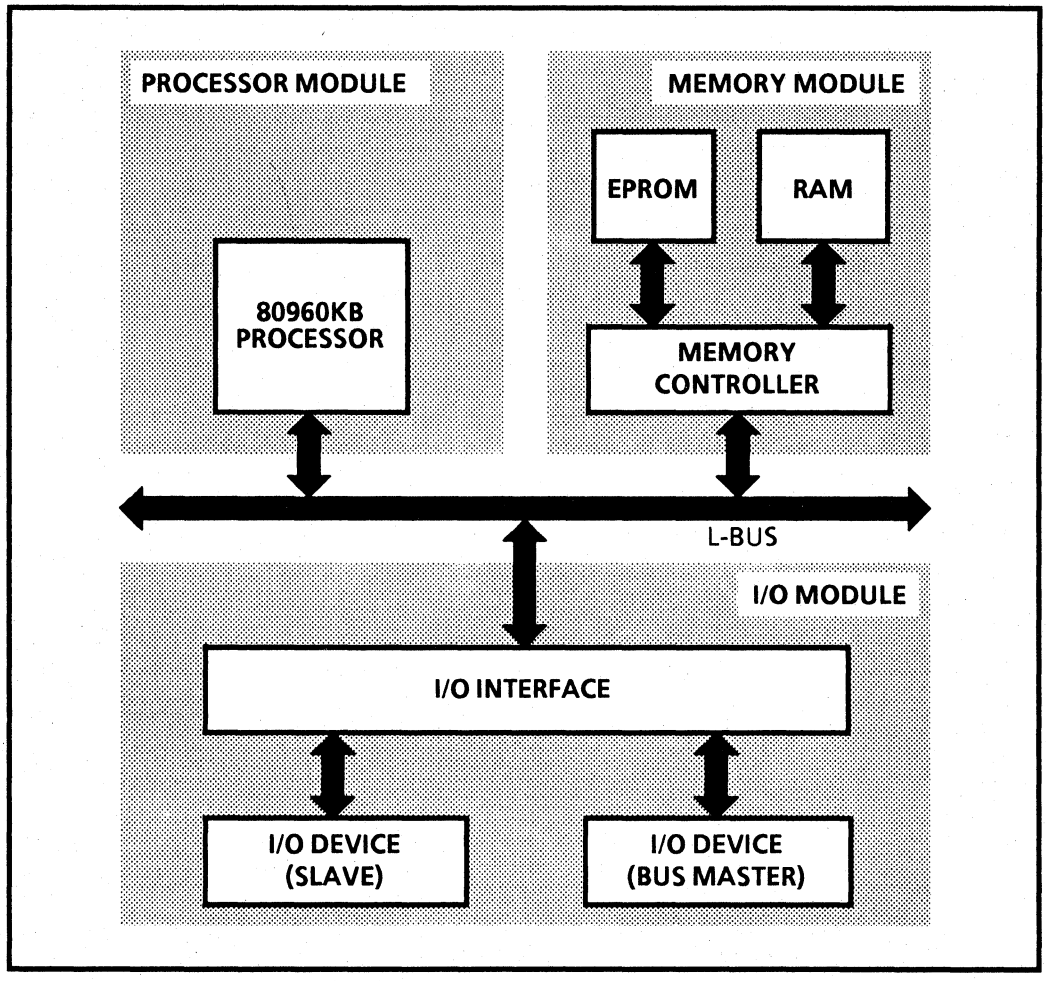


Figure 2-1: Basic 80960KB System Configuration

The 80960KB processor provides a flexible interrupt structure by using an on-chip interrupt controller, an external interrupt controller, or both. The type of interrupt structure is specified by an internal interrupt vector register. For a system with multiple processors, another method is available, called inter-agent communication (IAC) where a processor can interrupt another processor by sending an IAC message.

Complete details of the L-bus and bus operations are discussed in Chapter 3.

Memory Module

A memory module can consist of the memory controller, Erasable Programmable Read Only Memory (EPROM), and static or dynamic Random Access Memory (RAM). The memory controller first conditions the L-bus signals for memory operation. It demultiplexes the address

and data lines, generates the chip select signals from the address, detects the start of the cycle for burst mode operation, and latches the byte enable signals.

The memory controller generates the control signals for EPROM, SRAM, and DRAM. In particular, it provides the control signals, multiplexed row/column address, and refresh control for dynamic RAMs. The controller can be designed to accommodate the burst transaction of the 80960KB processor by using the static column mode or nibble mode features of the dynamic RAM. In addition to supplying the operation signals, the controller generates the $\overline{\text{READY}}$ signal to indicate that data can be transferred to or from the 80960KB processor.

The 80960KB processor directly addresses up to 4G bytes of physical memory. The processor does not allow burst accesses to cross a 16-byte boundary to ease the design of the controller. Each address specifies a four-byte data word within the block. Individual data bytes can be accessed by using the four byte enable signals from the 80960KB processor.

Chapter 4 provides design guidelines for the memory controller.

I/O Module

The I/O module consists of the I/O components and the interface circuit. I/O components can be used to allow the 80960KB processor to use most of its clock cycles for computational and system management activities. Time consuming tasks can be off-loaded to specialized slave-type components, such as the 8259A Programmable Interrupt Controller, or the 82530 Serial Communication Controller. Some tasks may require a master-type component, such as the 82586 Local Area Network Control.

The interface circuit performs several functions. It demultiplexes the address and data lines, generates the chip select signals from the address, produces the I/O read or I/O write command from the processor's W/$\overline{\text{R}}$ signal, latches the byte enable signals, and generates the $\overline{\text{READY}}$ signal. Because these functions are the same as some of the functions of the memory controller, the same logic can be used for both interfaces. For master-type peripherals that operate on a 16-bit data bus, the interface circuit translates the 32-bit data bus to a 16-bit data bus.

The 80960KB processor uses memory-mapped addresses to access I/O devices. This allows the CPU to use many of the same instructions to exchange information for both memory and peripheral devices. Thus, the powerful memory-type instructions can be used to perform 8-, 16-, and 32-bit data transfers.

Chapter 5 describes design guidelines for the I/O interface by examining representative design examples.

SUMMARY

The basic hardware system configuration is modular and flexible. The processor, memory, and I/O modules form the natural boundaries in the basic hardware system architecture. The high-bandwidth L-bus that supports burst transfers is used for the data path between the 80960KB processor and other modules.

This chapter presents an overview for basic hardware system design. The next three chapters discuss the details of the L-bus, memory modules, and I/O modules.

The 80960KB Processor and the Local Bus

3

CHAPTER 3
THE 80960KB PROCESSOR AND THE LOCAL BUS

The 32-bit multiplexed local bus (L-bus) connects the 80960KB processor to memory and I/O and forms the backbone of any 80960KB processor based system. This high bandwidth bus provides burst-transfer capability allowing up to four successive 32-bit data word transfers at a maximum rate of one word every clock cycle. In addition to the L-bus signals, the 80960KB processor uses other signals to communicate to other bus masters. This chapter, which describes these signals and the associated operations, follows the outline shown below:

- L-bus states and their relationship to each other
- L-bus signal groups, which consist of address/data and control
- L-bus read, write, and burst transactions
- L-bus timing analyses and timing circuit generation
- Related L-bus operations such as arbitration, interrupt, and reset operations

OVERVIEW OF THE 80960KB L-BUS

The L-bus forms the data communication path between the various components in a basic 80960KB hardware system. The 80960KB processor utilizes the L-bus to fetch instructions, to manipulate information from both memory and I/O devices, and to respond to interrupts. To perform these functions at a high data rate, the 80960KB processor provides a burst mode, which transfers up to four data words at a maximum rate of one 32-bit word per clock cycle. The 80960KB L-bus has the following features:

- 32-bit multiplexed address/data path
- High data bandwidth relative to the speed selection of the 80960KB processor
- Four byte enables and a four-word burst capability that allow transfers from 1 to 16 bytes in length
- Support for TTL latches and buffers.

BASIC L-BUS STATES

The L-bus has five basic bus states: idle (T_i), address (T_a), data (T_d), recovery (T_r), and wait (T_w). During system operation, the 80960KB processor continuously enters and exits different bus states as shown in Figure 3-1. This state diagram assumes that only one bus master resides on the L-bus.

The processor occupies the T_i state when no address/data transfers are in progress. When a new request is received, the 80960KB processor enters the T_a state to transmit the address.

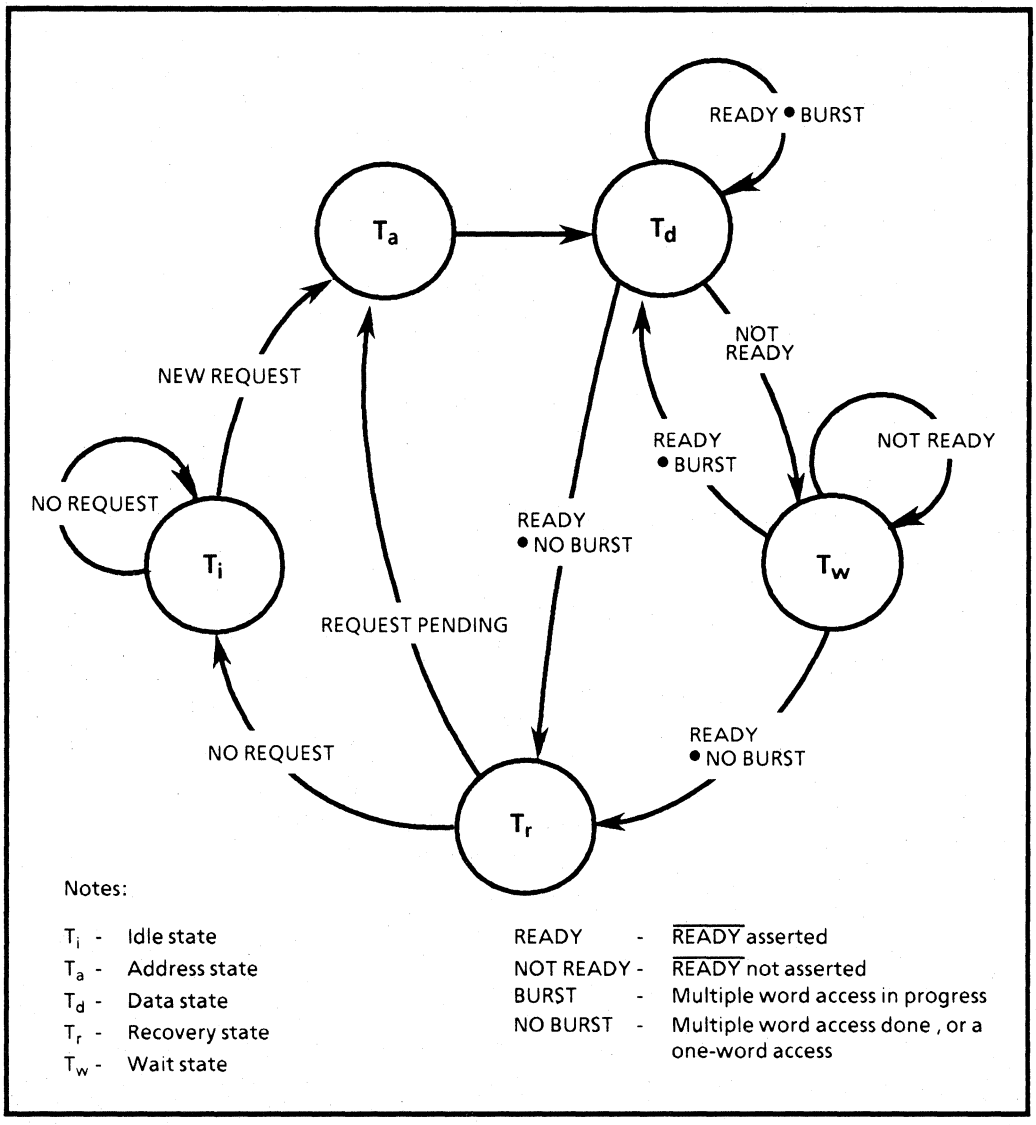


Figure 3-1: Basic L-Bus States

Following a T_a state, the 80960KB processor enters a T_d state to transmit or receive data on the address/data lines provided that the data is ready (indicated by the assertion of $\overline{READY}$ at the input of the processor). If the data is not ready, the processor enters a T_w state and remains in this state until data is ready. T_w states may be repeated as many times as necessary to allow sufficient time for the memory or I/O device to respond.

After a data word is transferred, the 80960KB processor exits the T_d or T_w state for a single word transfer or enters the T_d state again to transfer another data word for a burst transaction. If the next data word is not ready during the next clock cycle for a burst transaction, the processor enters the T_w state again.

When the 80960KB processor completes the data transfer of all the data words (one or up to four), it enters the recovery (T_r) state to allow sufficient time for devices (such as memories) on the bus to recover. The processor returns to the T_i state if no new request is pending, or enters the T_a state if a new request is pending.

L-BUS SIGNAL GROUPS

The L-bus states are used to define some of the L-bus signals. As shown in Figure 3-2, the signals on the L-bus consist of two basic groups: address/data, and control.

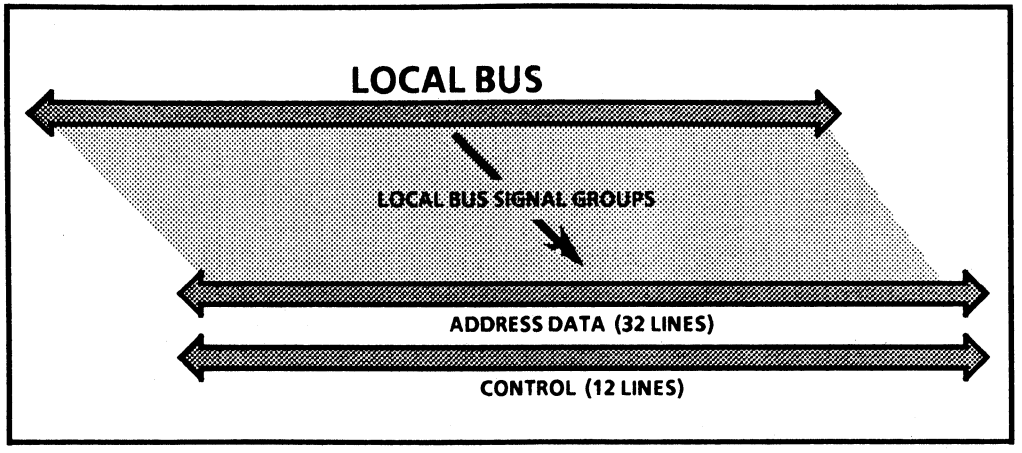


Figure 3-2: L-Bus Signal Groups

Address/Data

The address/data signal group consists of 32 bidirectional lines. These signals are multiplexed and serve a dual purpose depending upon the bus state.

LAD_{31}-LAD_2 — **Local Address/Data$_{31}$** through **Local Address/Data$_2$** represent the **address** signals on the L-bus during the T_a state. LAD_2 is the least significant bit, and LAD_{31} is the most significant address bit. LAD_{31} through LAD_2 contain a physical word address. LAD_1 and LAD_0 specify the number of data words to transfer for a burst transaction. The address/data signals float to a high impedance state when not activated.

SIZE (LAD_1-LAD_0) — The **SIZE** signal indicates whether one, two, three, or four words are transferred during the current transaction. During a T_a state, LAD_1 and LAD_0 represent the word size signals. The encoding is shown in Table 3-1.

Table 3-1: SIZE Signal Decoding

WORD SELECTION	LAD_1	LAD_0
1 WORD	LOW	LOW
2 WORDS	LOW	HIGH
3 WORDS	HIGH	LOW
4 WORDS	HIGH	HIGH

LAD_{31}-LAD_0 **Local Address/Data$_{31}$** through **Local Address/Data$_0$** represent the **data** signals on the L-bus during the T_d and T_w states. LAD_0 is the least significant, and LAD_{31} is the most significant address bit. The address/data signals float to a high impedance state when not activated.

Control

The control signal group consists of 12 signals that permit the transfer of data. These signals can be used to control data buffers, address latches, and other standard interface logic.

$\overline{ALE}$ The **Address Latch Enable** is an active low signal that can be used to latch the address from the 80960KB processor. $\overline{ALE}$ is asserted during the T_a state and deasserted before the beginning of the T_d state. $\overline{ALE}$ floats to a high impedance level when the processor is not operating on the bus (i.e., it is in the idle state), or is at the end of any bus access.

$\overline{ADS}$ **Address/Data Status** is an active low signal that is driven by the 80960KB processor to indicate an address state. $\overline{ADS}$ is asserted during every T_a state and deasserted during the following T_d and T_w states. For a burst transaction, $\overline{ADS}$ is asserted again every T_d (and T_w) state where $\overline{READY}$ was asserted in the prior cycle. The $\overline{ADS}$ signal is an open drain output.

DT/$\overline{R}$ **Data Transmit/Receive** indicates the direction of data flow to or from the L-bus. For a read operation or an interrupt acknowledgement, DT/$\overline{R}$ is low during the T_a, T_w, and T_d states to indicate that data flows into the 80960KB processor. For a write operation, DT/$\overline{R}$ is high during the T_a, T_w, and T_d states to indicate that data flows from the 80960KB processor. DT/$\overline{R}$ never changes states when $\overline{DEN}$ is asserted. The DT/$\overline{R}$ line is an open drain output of the 80960KB processor.

$\overline{DEN}$ **Data Enable** is an active-low signal that can be used to enable data transceivers. $\overline{DEN}$ is asserted during all T_d and T_w states. The $\overline{DEN}$ line is an open drain output of the 80960KB processor.

W/$\overline{R}$ The **Write/Read** signal instructs a memory or I/O device to write or read data on the L-bus. The 80960KB processor asserts W/$\overline{R}$ during a T_a state. The signal remains valid during subsequent T_d and T_w states. W/$\overline{R}$ is an open drain output of the 80960KB processor.

$\overline{BE_3}$-$\overline{BE_0}$ The **Byte Enable** output signals of the 80960KB processor specify which bytes (up to four) on the 32-bit data bus are transferred during the transaction. Table 3-2 shows the decoding scheme.

Table 3-2: Byte Enable Signal Decoding

BYTE ENABLE SIGNAL	ADDRESS LINE SELECTION
$\overline{BE}_0$	LAD_7.LAD_0
$\overline{BE}_1$	LAD_{15}.LAD_8
$\overline{BE}_2$	LAD_{23}.LAD_{16}
$\overline{BE}_3$	LAD_{31}.LAD_{24}

The byte enable signals are valid from the 80960KB processor before data is transferred, as shown in Figure 3-3 (assumes no wait states). The byte enable signals that are valid for the first data word are specified during the T_a state. For a four-word burst transaction, the byte enable signals that are valid for the second word are asserted during the first data state (T_{d0}), for the third word during the second data state (T_{d1}), and for the fourth word during the third data state (T_{d2}). The byte enable signals are undefined during the last data state (T_{d3}) of the last word transferred.

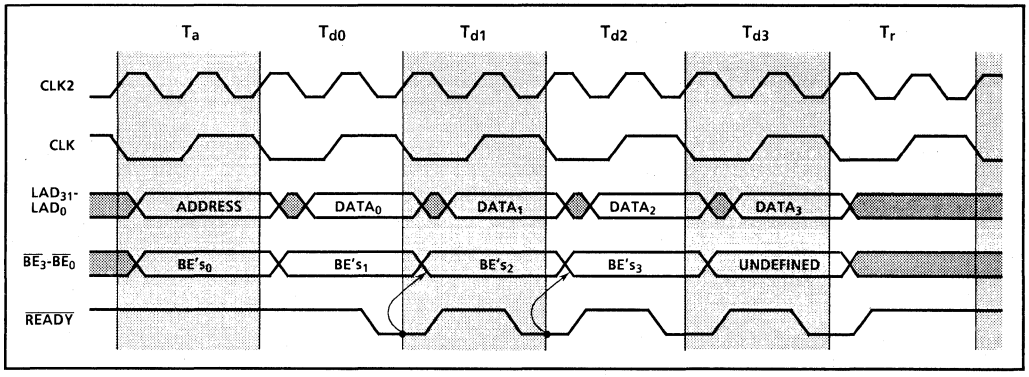


Figure 3-3: Byte Enable Timing Diagram

Although not shown in the diagram, the byte enable signals of each word are latched internally by the 80960KB processor and remain valid during every data or wait state until $\overline{READY}$ is applied. After $\overline{READY}$ is applied the byte enable signals change during the next T_d state or become undefined for the last data transfer.

The 80960KB processor asserts only adjacent byte enables. For example, the 80960KB processor does not perform a bus operation with only $\overline{BE}_0$ and $\overline{BE}_2$ active.

The Byte Enable lines are open drain outputs.

$\overline{READY}$

$\overline{READY}$ signal indicates that the data on the L-bus can be sampled (read) or removed (write) by the 80960KB processor. If $\overline{READY}$ is not asserted following T_a state or in between T_d states, a T_w state is generated. The $\overline{READY}$ is an active-low input signal to the 80960KB processor.

$\overline{\text{LOCK}}$

Bus **Lock** prevents other bus masters from gaining control of the L-bus during a bus operation. It is activated by certain 80960KB processor operations and instructions.

The 80960KB processor uses the bus $\overline{\text{LOCK}}$ signal when it performs a RMW memory operation. When the processor performs a RMW-Read operation, it asserts the $\overline{\text{LOCK}}$ signal during the T_a state and holds $\overline{\text{LOCK}}$ asserted. If the $\overline{\text{LOCK}}$ signal was already asserted, the processor waits until this signal is deasserted before performing the RMW-Read operation. The processor deasserts the $\overline{\text{LOCK}}$ signal during the T_a state when it performs a RMW-Write operation.

The 80960KB processor asserts the $\overline{\text{LOCK}}$ signal during the interrupt acknowledge sequence. $\overline{\text{LOCK}}$ is an input and an open drain output.

CACHE

The **Cacheable** signal specifies whether the data is cacheable. If the 80960KB processor asserts CACHE during the T_a state, then the data is cacheable. The CACHE signal is undefined during the T_d and T_w states. The CACHE signal floats to a high impedance state when the L-bus is not acquired.

Table 3-3 summarizes the L-bus signals.

Table 3-3: Summary of L-Bus Signals

SIGNAL GROUP	SIGNAL SYMBOL	SIGNAL FUNCTION	ACTIVE STATE	DIRECTION	TYPE OF OUTPUT
LOCAL ADDRESS/ DATA	ADDRESS (LAD_{31}-LAD_2)	32-BIT ADDRESS	T_a	O	3-STATE
	DATA (LAD_{31}-LAD_0)	32-BIT DATA	T_d, T_w	I/O	3-STATE
	SIZE (LAD_1-LAD_0)	SPECIFIES NUMBER OF WORDS TO TRANSFER	T_a	O	3-STATE
CONTROL	$\overline{ALE}$	ENABLES ADDRESS LATCH	T_a	O	3-STATE
	$\overline{ADS}$	IDENTIFIES AN ADDRESS STATE	T_a, T_d[1], T_w[1]	O	OPEN DRAIN
	DT/$\overline{R}$	CONTROLS DIRECTION OF DATA FLOW	T_a, T_d, T_w	O	OPEN DRAIN
	$\overline{DEN}$	ENABLES DATA TRANSCEIVER/LATCH	T_d, T_w	O	OPEN DRAIN
	W/$\overline{R}$	READ/WRITE COMMAND	T_a, T_d, T_w	O	OPEN DRAIN
	$\overline{BE_3}$-$\overline{BE_0}$	SPECIFIES WHICH DATA BYTES TO TRANSFER	T_a, T_d[2], T_w[2]	O	OPEN DRAIN
	$\overline{READY}$	INDICATES DATA IS READY TO TRANSFER	T_d, T_w	I	---------
	$\overline{LOCK}$	LOCKS BUS	ANY	I/O	OPEN DRAIN
	CACHE	INDICATES CACHEABLE TRANSACTION	T_a	O	3-STATE

Additional pins are used by the 80960KB processor to control the execution of instructions and to interface to other bus masters. These pins include the arbitration, interrupt, error, and reset signals. Each of these signal groups are explained in separate sections.

L-BUS TRANSACTIONS

The 80960KB processor uses the L-bus signals to perform transactions, which are simply L-bus operations where data is transferred to (or from) the CPU from (or to) another component. During a transaction, the 80960KB processor can transfer up to four words of data for a single address to enhance system throughput. This is especially useful when loading cache memory.

Clock Signal

The 80960KB hardware system typically uses two clock signals, CLK2 and CLK, to synchronize the transitions between L-bus states. CLK2 is the clock input to the 80960KB and is double the specified processor frequency. CLK is the clock input signal to the peripheral devices, and it is the operating frequency of the 80960KB processor. Figure 3-4 shows the relationship between the system CLK2 and CLK.

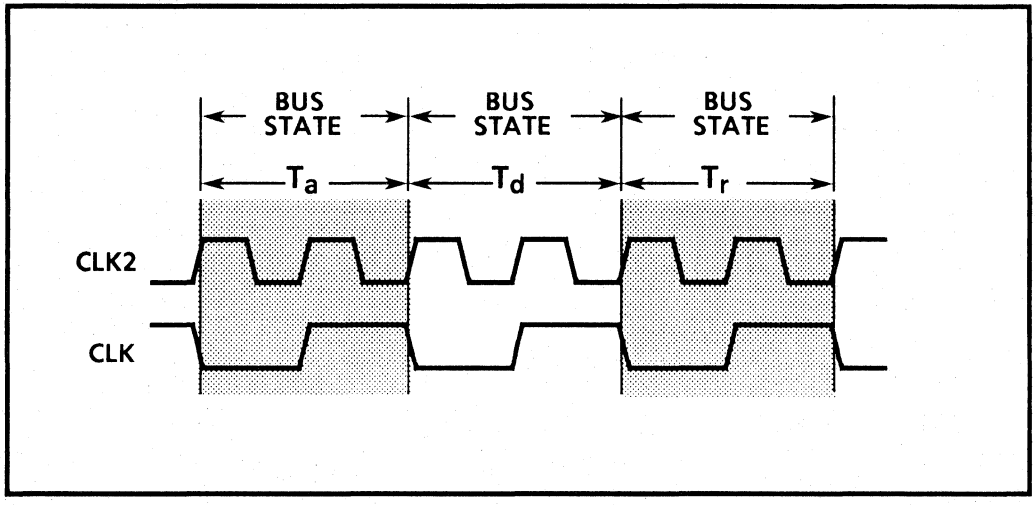


Figure 3-4: Clock Relationships

Basic Read

The basic transaction reads or writes one data word. Figure 3-5 shows a typical timing diagram for a basic read transaction (for exact timings, see the 80960KB processor data sheet). A read transaction may be preceded and succeeded by any type of bus transaction. The following sequence of events explains the flow of the timing diagram. For simplicity, no wait states are shown.

1. The 80960KB processor generates several signals during the T_a state.
   - It transmits the address on the address/data lines. LAD_1 and LAD_0 specify a single word transaction.

- It asserts $\overline{ALE}$. An $\overline{ALE}$ signal can be used to latch the address.
- It asserts $\overline{ADS}$.
- It asserts $\overline{BE}_3$-$\overline{BE}_0$ to specify which bytes are used when reading the data word.
- It brings W/$\overline{R}$ low to denote a read operation.
- It brings DT/$\overline{R}$ signal low. DT/$\overline{R}$ can be used for the direction input to data transceivers.

2. During the T_d state, several actions occur.
   - The 80960KB processor reads the data on the address/data lines.
   - The 80960KB processor asserts $\overline{DEN}$. $\overline{DEN}$ can be used to enable data transceivers. $\overline{READY}$ is asserted by external timing logic and data is transmitted from the storage devices. If $\overline{READY}$ is not asserted, the data transfer is delayed generating a T_w state. The T_w state is repeated, until $\overline{READY}$ is asserted.
3. The T_r state follows the data state. This allows the system components adequate time (one processor clock cycle) to remove their outputs from the bus before the 80960KB processor generates the next address on the address/data lines. During the T_r state W/$\overline{R}$, DT/$\overline{R}$, and $\overline{DEN}$ become inactive.

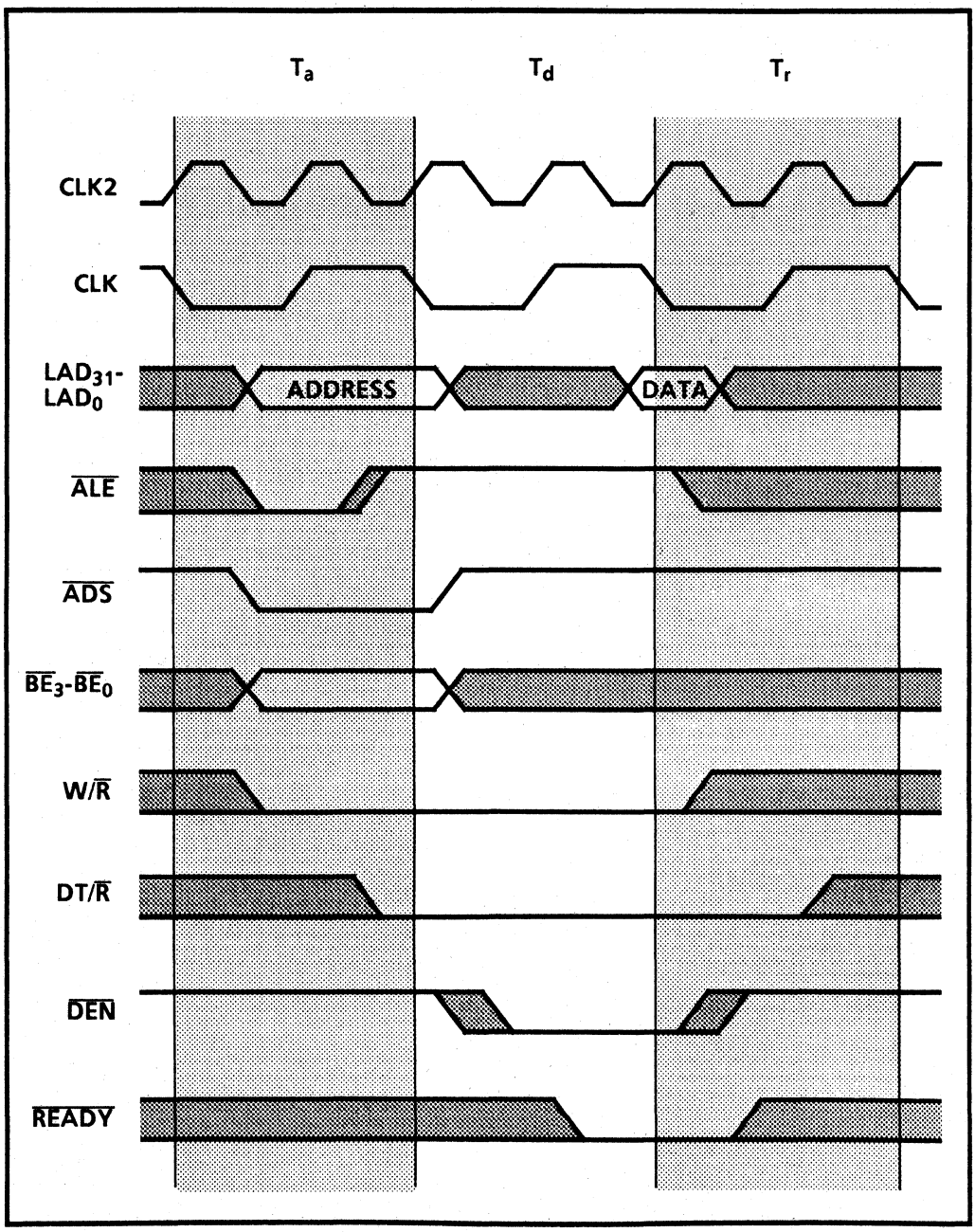


Figure 3-5: 80960KB Processor Read Transaction

Basic Write

Figure 3-6 shows a typical timing diagram for a basic write transaction with one wait state. Like the read transaction, a write operation may be preceded and succeeded by any type of bus transaction. The following sequence of events explains the flow of the timing diagram.

1. Similar to the read transaction, the 80960KB processor generates several signals during the T_a state.
   - It transmits the address on the address/data lines. LAD_1 and LAD_0 specify a single word transaction.
   - It asserts $\overline{ALE}$. An $\overline{ALE}$ signal can be used to latch the address.
   - It asserts $\overline{ADS}$.
   - It asserts $\overline{BE_3}$-$\overline{BE_0}$ to specify which bytes are used when writing the data word.
   - It brings W/$\overline{R}$ high to denote a write operation.
   - It brings DT/$\overline{R}$ signal high. DT/$\overline{R}$ can be used for the direction input to data transceivers.
2. During the T_d state, several actions occur.
   - The 80960KB places the data on the address/data lines.
   - The 80960KB processor asserts $\overline{DEN}$. $\overline{DEN}$ can be used to enable data transceivers.
   - $\overline{READY}$ is not asserted by external timing logic. Consequently, data is held on the LAD lines.
3. During the T_w state $\overline{READY}$ is asserted and the data is written to the storage device. Note that the W/$\overline{R}$, DT/$\overline{R}$, and $\overline{DEN}$ remain constant until the bus state after $\overline{READY}$ is asserted.
4. The T_r state follows the wait state. During the T_r state W/$\overline{R}$, DT/$\overline{R}$, and $\overline{DEN}$ become inactive.

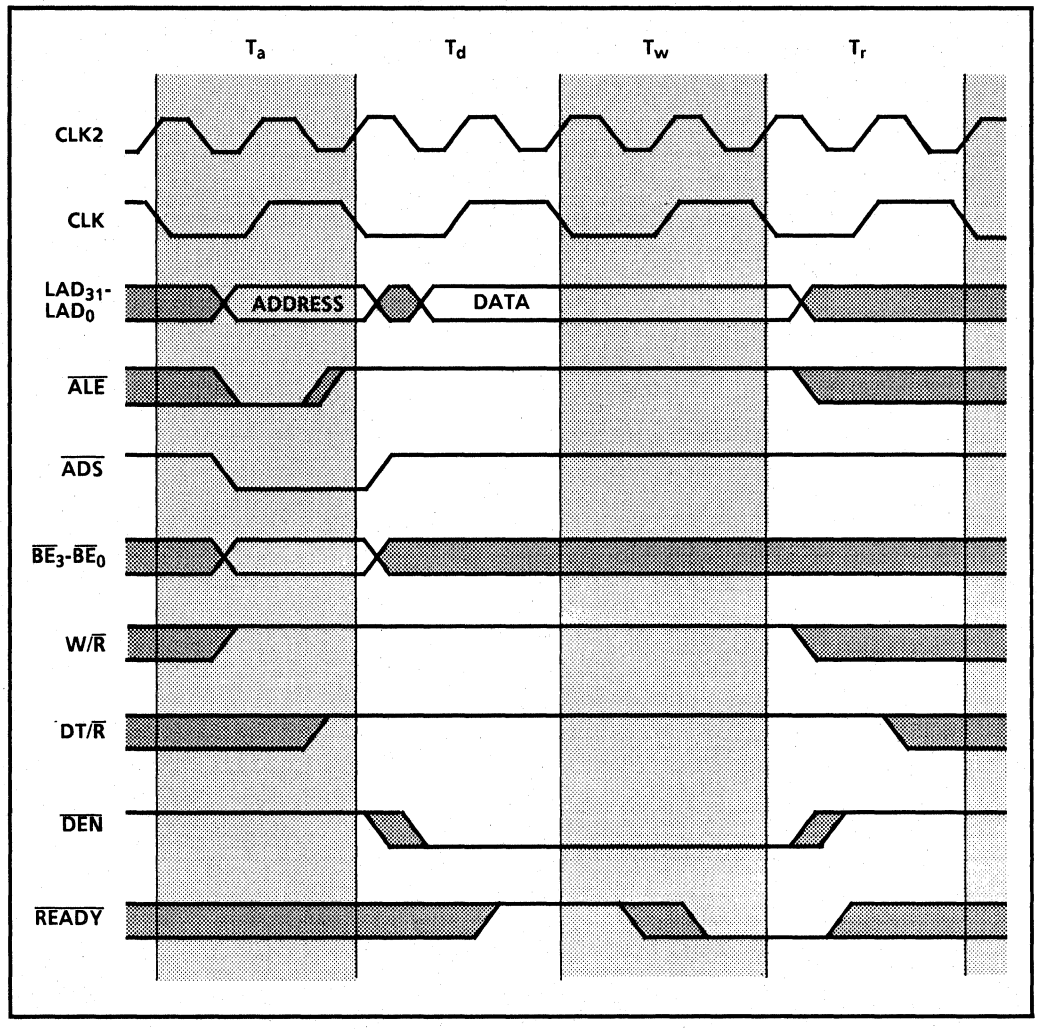


Figure 3-6: 80960KB Processor Write Transaction

Burst

The 80960KB processor supports burst transactions that read or write up to four words at a maximum rate of one word every processor clock cycle. Burst transactions are always contained within a 16-byte boundary. If a transaction crosses a 16-byte boundary, the 80960KB processor automatically splits the transaction into two accesses.

The byte enable signals are valid for each word to allow partial-word write operations for a burst write transaction. The CACHE output signal during a T_a state applies to all words of a burst transaction.

A burst read or write transaction is similar to a basic read or write operation. It differs primarily in the number of data words transferred: the basic transaction always transfers one data word, the burst transaction transfers up to four data words. For a burst transaction, the byte enable signals are applied during the T_a state, and subsequently during every T_d or T_w state before the data word is transferred. Figure 3-7 shows the timing for a three-word burst read transaction without wait states. Figure 3-8 shows the timing for a two-word burst write transaction with a wait state occurring during the transfer of the first word. Note that the byte enable signals remain constant until the data state after $\overline{\text{READY}}$ is asserted.

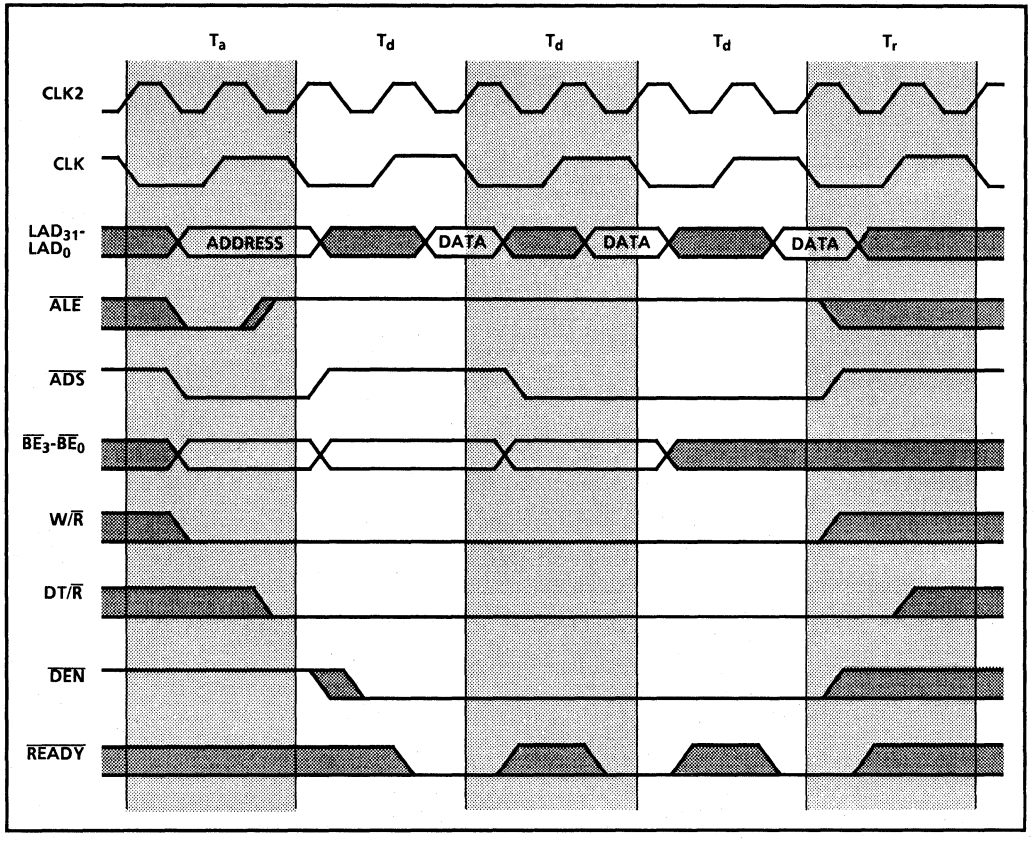


Figure 3-7: 80960KB Processor Burst Read Transaction

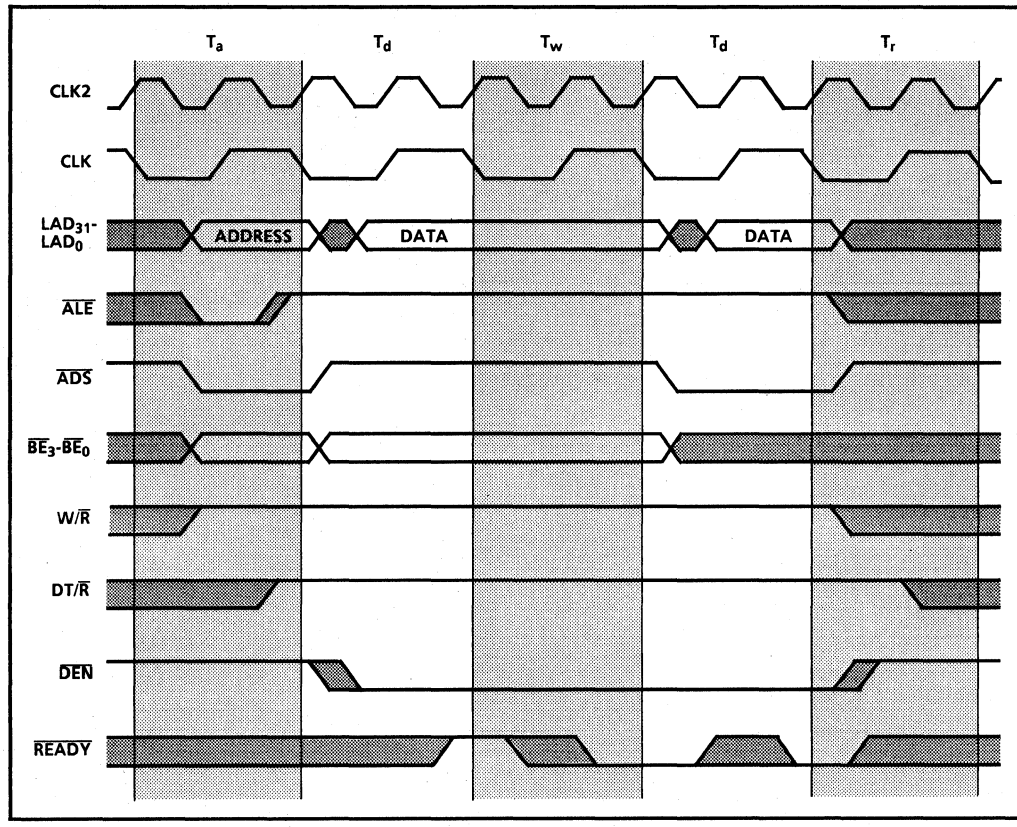


Figure 3-8: 80960KB Processor Burst Write Transaction

TIMING GENERATION

In an 80960KB processor-based system, timing signals must be generated for the clock and reset inputs. To generate these signals, discrete logic should be utilized to minimize skew and maintain the rise and fall times as short as possible. This section describes a typical circuit that synthesizes the clock signal. The RESET timing generation is discussed in the "RESET and Initialization" section on page 3-33.

80960KB Processor Clock Requirements

In order to design a clock generator, the clock input specifications to the 80960KB processor are examined first. The clock (CLK2) waveform is shown in Figure 3-9. The clock pulse is specified by five parameters listed below:

- The clock fall time (t_f)
- The clock low time (t_l)

- The clock rise time (t_r)
- The clock high time (t_h)
- The clock period (t_{cyc})

The time required to go from 90% of the difference between the high and low voltage levels to 10% of the difference (or from low to high) is defined as the clock fall (rise) time. The clock low time specifies the time required for the clock to remain within 10% of the low voltage level. Similarly, the clock high time specifies the required time for the clock pulse to remain within 10% of the high voltage level. The clock period is the sum of $t_f + t_l + t_r + t_h$.

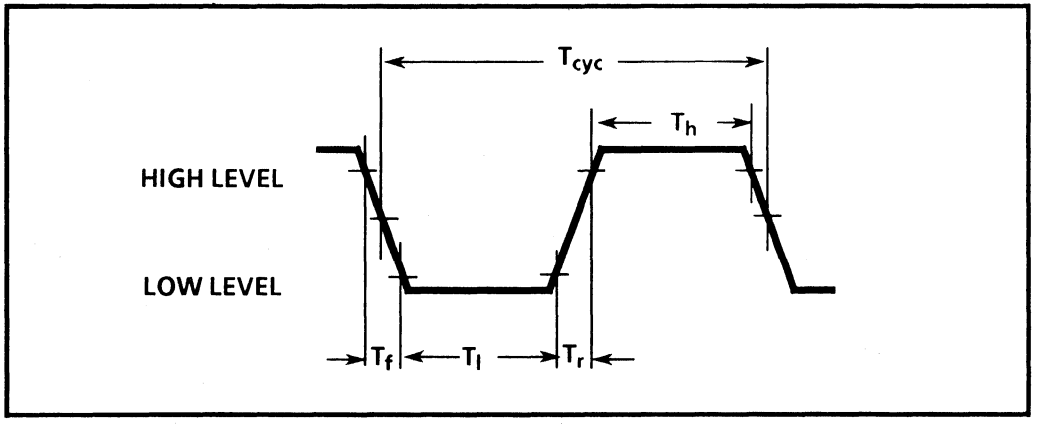


Figure 3-9: System Clock Pulse

The clock generator must have fast enough rise and fall times to comply with the requirements for high and low time and the overall clock period. For example, consider a clock pulse with a 50% duty cycle at 40 MHz. The clock period is specified at minimum of 25 ns, low time at minimum of 8 ns, and high time at minimum of 8 ns. This implies that the sum of the rise and fall time must not be greater than 9 ns. Thus, the clock generator should be designed to have rise and fall times not greater than 4.5 ns each.

Besides specifying a maximum clock rate, the 80960KB processor requires a minimum CLK2 rate of 8 MHz to maintain the state of the internal dynamic cells. Due to this minimum frequency requirement, the 80960KB processor cannot be single-stepped by disabling the clock.

Clock Generation

Figure 3-10 shows an example of a clock generator that produces two clock pulses, one double the frequency of the other with the skew between the pulses in the range of 1 to 3 ns. This particular circuit produces a 40-MHz clock at 50% duty cycle with rise and fall times of less than 4 ns. The circuit design consists of four devices: an oscillator, a pulse shaping network, a synchronous up/down counter, and a NAND gate driver. The output of the 80-MHz hybrid clock oscillator connects to the pulse shaping network (two NAND gates in series), which in turn feeds into the clock input of the up/down counter. This counter produces a 40-MHz CLK2 output signal and a 20-MHz CLK output signal. Because the outputs of the counter are

synchronous, the skew between CLK2 and CLK is typically less than 2 ns. To provide adequate signal margin and maintain fast rise and fall times, the two clock signals are conditioned by the NAND gate driver. The timing waveforms of the clock circuit are shown in Figure 3-11.

If the opposite phase CLK is preferred, U/$\overline{D}$ pin can be connected to V_{CC}.

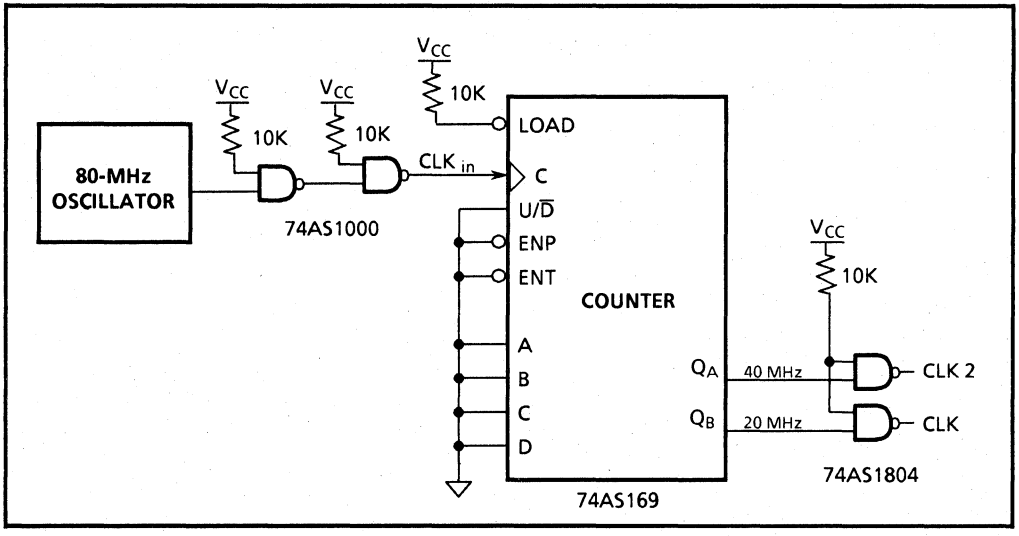


Figure 3-10: Clock Generation Circuit

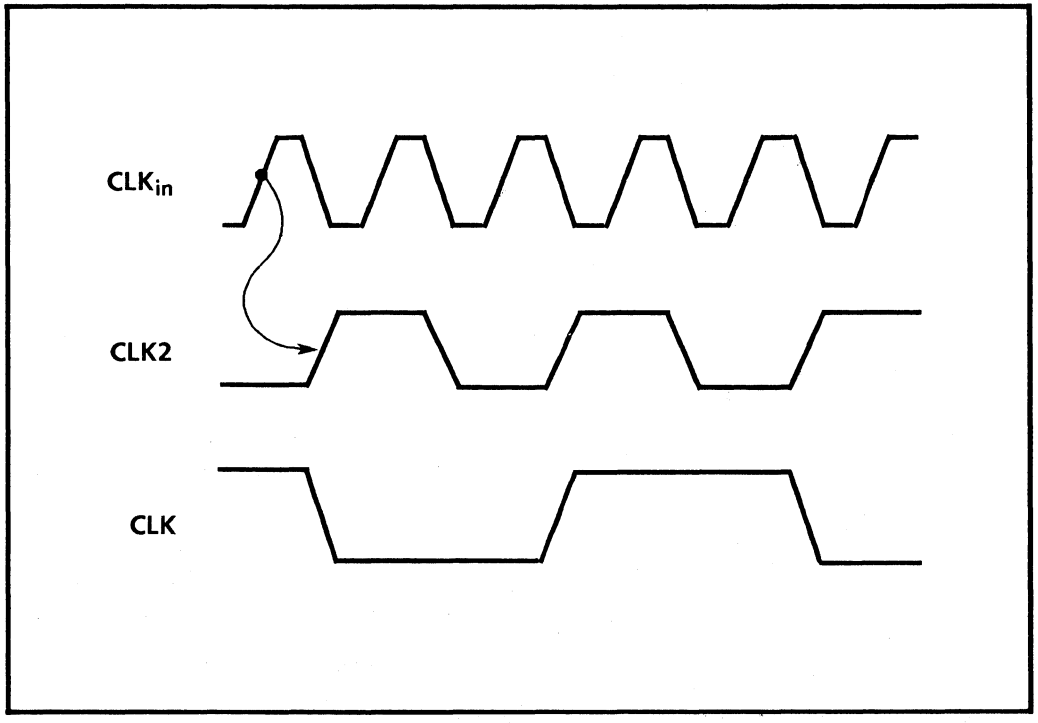


Figure 3-11: Clock Timing Waveforms

The hybrid clock oscillator typically requires 5 ms to stabilize after power is applied. The 80960KB processor cannot begin to execute instructions until after the clock and V_{CC} have reached their DC and AC specifications. The RESET signal can be used to control the start of the CPU execution when power is applied. This is discussed in the "RESET and Initialization" section on page 3-33.

ARBITRATION

When multiple bus masters exist, an arbitration protocol is used to exchange control of the bus. The protocol assumes that there are two bus masters: one that controls the bus by default, and the other that requests control of the bus when it performs an operation, such as a DMA controller. More than two bus masters may exist on the L-bus, but this requires external arbitration logic. There should be no more than two 80960KB processors, however, on an L-bus.

Assuming that there are only two bus masters, this section examines the bus arbitration, bus states, and timing diagrams for different combinations of bus masters, as shown in Table 3-4.

Table 3-4: Combination of Bus Masters

	Bus Master Combination	
	Bus Master that Controls the Bus by Default	**Bus Master that Requests Contol of the Bus**
CASE 1	80960KB PROCESSOR	I/O DEVICE
CASE 2	80960KB PROCESSOR	80960KB PROCESSOR
CASE 3	I/O DEVICE	80960KB PROCESSOR

Single 80960KB Processor on the L-Bus

For the first case, the 80960KB processor controls the L-bus, and a master I/O peripheral, such as a DMA controller, requests control of the bus for operations. The 80960KB processor and the I/O peripheral exchange control of the bus with two signals: the hold request (HOLD) and hold acknowledge (HLDA) signals.

HOLD is an input signal of the 80960KB processor, which indicates that the master I/O peripheral is requesting control of the L-bus. When HOLD is asserted, the 80960KB processor surrenders control of the bus after it completes the current bus transaction. The 80960KB processor acknowledges transfer of control of the L-bus to the other bus master by asserting the HLDA.

State Diagram

Figure 3-12 shows the state diagram for a L-bus with an I/O peripheral bus master. This state diagram consists of the hold state (T_h) in addition to the five basic states described in the "Basic L-Bus State" section on page 3-1. The 80960KB processor enters the T_h state when it surrenders the control of the bus. It can enter the T_h state from the T_i or T_r state. When the 80960KB processor regains control of the L-bus, it enters the T_a state if a new request is pending or a T_i state if no new request is pending.

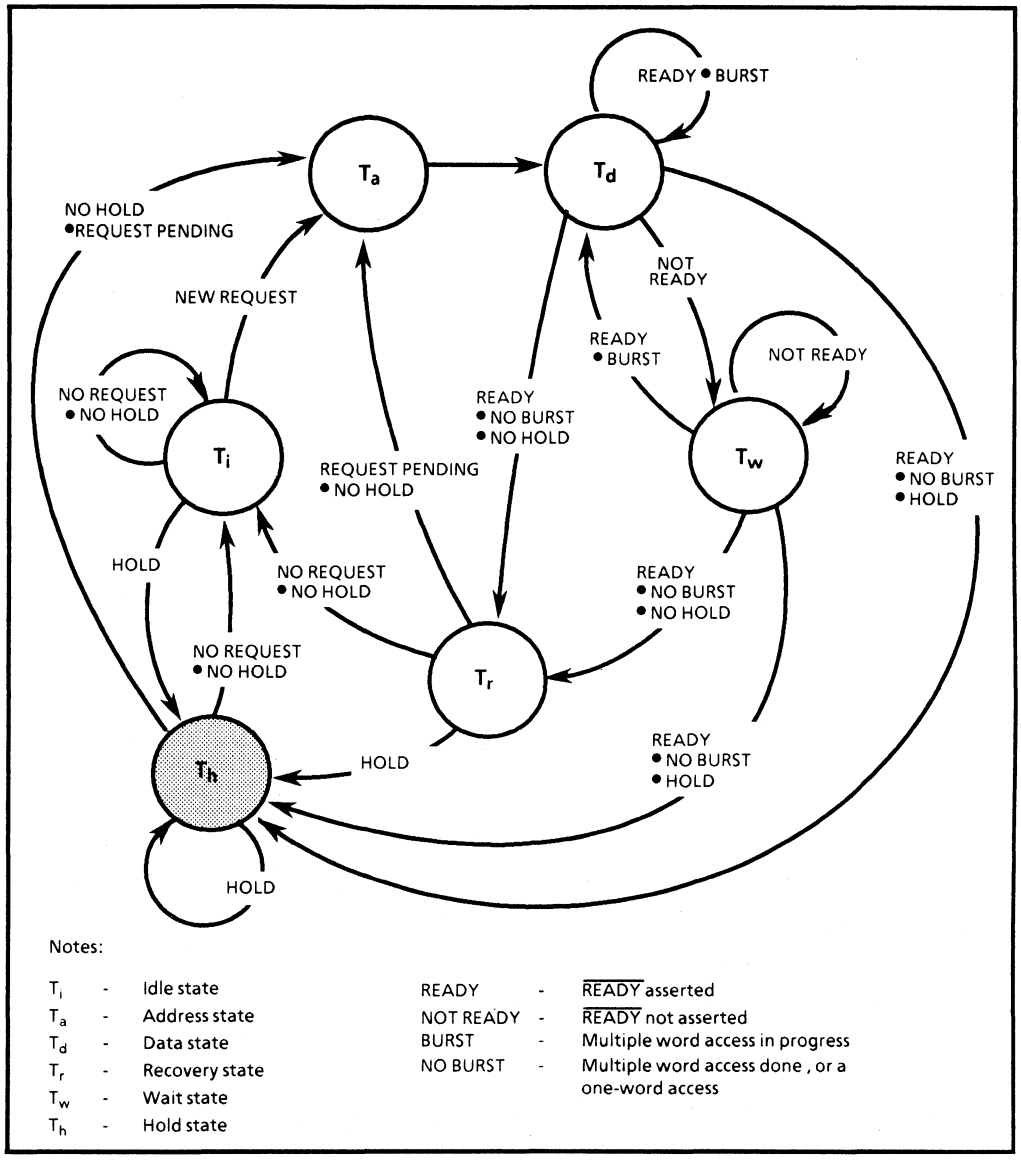


Figure 3-12: L-Bus States with Arbitration

Arbitration Timing

Figure 3-13 shows the arbitration timing diagram. The "T" state represents the last cycle of a transaction in which the $\overline{\text{READY}}$ signal was asserted or a T_i state. The 80960KB processor receives a request to relinquish control of the bus when HOLD is asserted. After the 80960KB processor completes the current transaction, it responds to this request by floating the three-state output signals and deasserting the open drain output signals. The HLDA output signal,

however, remains active and is asserted as the 80960KB processor enters a T_h state. During the T_h state, the CPU ignores all input signals except HOLD and RESET. When the HOLD input signal is deasserted, the 80960KB processor exits the T_h state and deasserts HLDA.

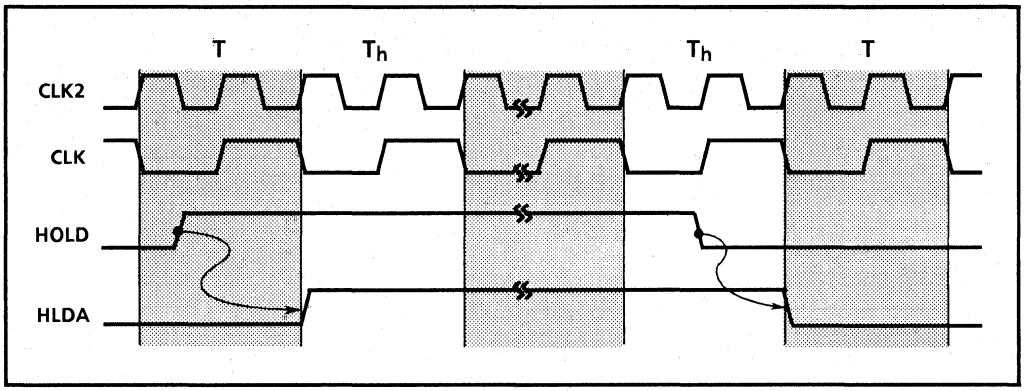


Figure 3-13: Arbitration Timing Diagram for a Bus Master

Two 80960KB Processors on the L-Bus

For the next case, two 80960KB processors reside on the L-bus. During initialization, one is designated as the Primary Bus Master (PBM), the other as the Secondary Bus Master (SBM). The exchange protocol that is used guarantees that neither device is kept off the bus indefinitely. The 80960KB processors use two pins for bus arbitration: the HOLD input pin, and the HLDA output pins. These input and output pins for the SBM are interpreted differently, however.

When the SBM is initialized, the pin normally used for HOLD input signal is interpreted as the hold acknowledge request (HLDAR) input signal. The assertion of HLDAR indicates that the PBM relinquished control of the L-bus. Similarly, the HLDA output signal of the SBM is interpreted as the hold request (HOLDR) output signal. The SBM asserts HOLDR to request acquisition of the L-bus. Thus, bus arbitration between two 80960KB processors can be accomplished by connecting HOLD of the PBM to HOLDR of the SBM, and HLDA of the PBM to the HLDAR of the SBM, as shown in Figure 3-14.

When using the connection shown in Figure 3-14, a delay must be inserted between the input and output signals because the minimum clock-to-output delay is less than the maximum hold time of the input signals. The delay time must be greater than 5 ns, but less than the clock period minus the setup time minus the maximum clock-to-output delay ($5\ ns \leq Delay \leq T_{Period} - T_{Setup} - T_{Clock\text{-}to\text{-}Output}$).

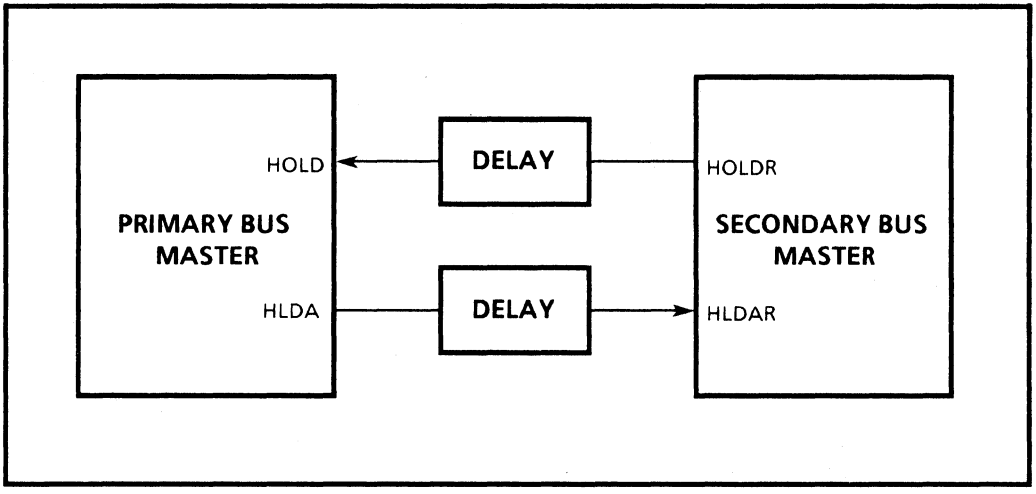


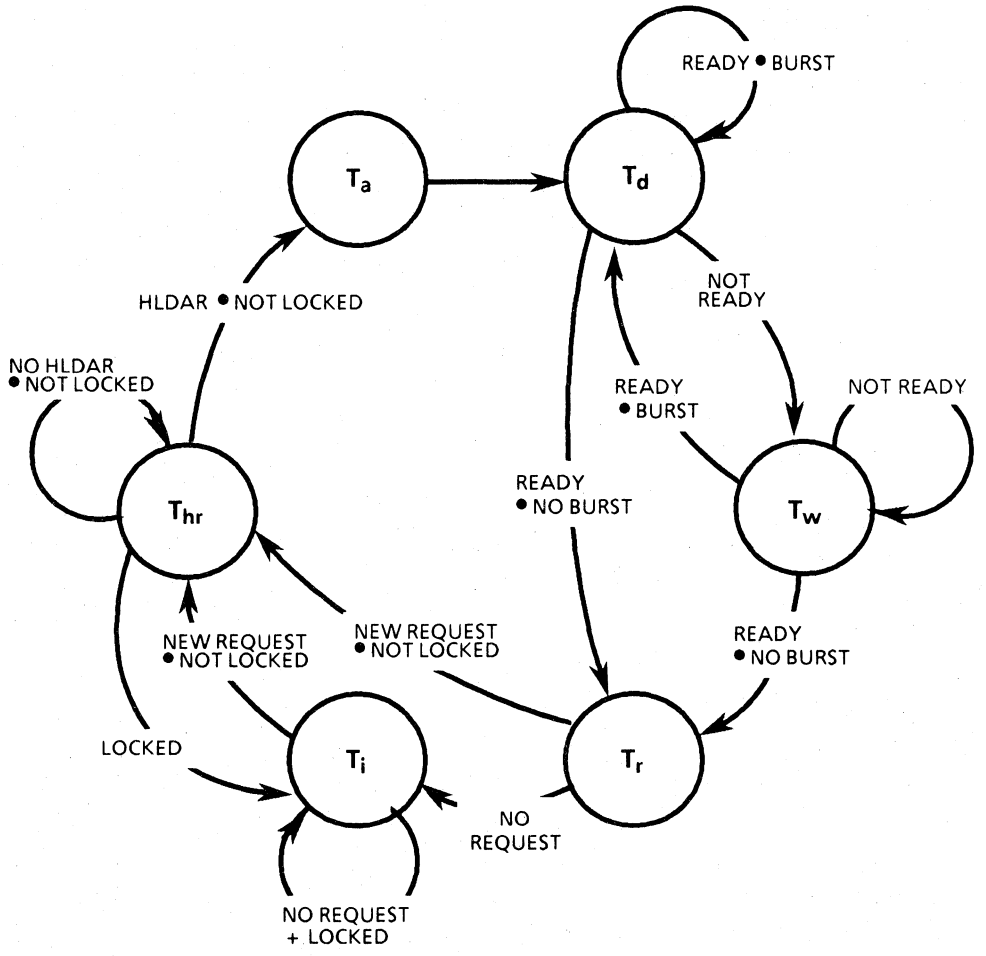
Figure 3-14: Arbitration Connection Between Two 80960KB Processors

Bus states for Two 80960KB Processors

The state diagram for the SBM is shown in Figure 3-15. Because there are two 80960KB processors, the $\overline{LOCK}$ signal is included in the state diagram. The SBM requests control of the L-bus by asserting HOLDR and subsequently enters the hold request (T_{hr}) state provided that the bus is not locked (locked means that $\overline{LOCK}$ is asserted by the PBM and the SBM has a RMW operation pending). The SBM remains in the T_{hr} state until it acquires control of the L-bus by receiving HLDAR. The SBM returns to the T_i state by deasserting HOLDR provided that the following two conditions exist:

- A RMW operation is pending
- The PBM asserted $\overline{LOCK}$ while the SBM was in the T_{hr} state.

The SBM gains control of the bus when HLDAR is asserted provided that the bus is not locked. After gaining control of the L-bus, the SBM performs the operations, and enters a T_w state if necessary. At the end of a transaction, the SBM goes to the T_r state and deasserts HOLDR for at least one processor clock cycle to allow another peripheral bus master to gain access if needed. If another request is pending, the SBM enters the T_{hr} state and asserts HOLDR provided the bus is not locked. The PBM never forces the SBM off the bus.



NOTES:

T_i	Idle state	READY	READY asserted
T_a	Address state	NOT READY	READY not asserted
T_d	Data state	LOCKED	LOCK asserted by another bus master and RMW operation pending for secondary bus master
T_r	Recovery state	HLDAR	HOLD acknowledge requested (request for bus granted)
T_{hr}	Hold request state	BURST	Multiple word access in progress
		NO BURST	Multiple word access done , or a one-word access

Figure 3-15: L-Bus States for Secondary Bus Master

Arbitration Timing for Two 80960KB Processors on the L-Bus

Figure 3-16 shows the timing diagram for acquiring and relinquishing the L-bus by an SBM. The SBM enters into the Hold Request (T_{hr}) state and asserts the HOLDR signal. It remains in the T_{hr} state until HLDAR is asserted, which indicates that the SBM can utilize the L-bus during the next state. When the bus is no longer required, HOLDR is deasserted during the state following the last $\overline{\text{READY}}$ signal. Except for HOLDR, the output signals of the SBM go into a high impedance state or are deasserted for the case of open-drain outputs.

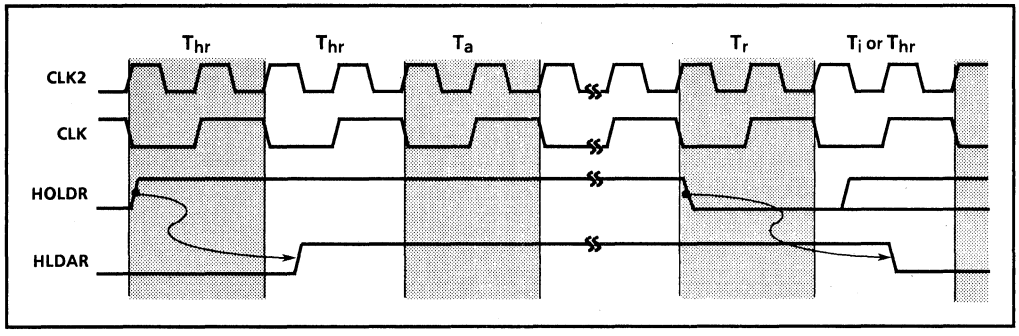


Figure 3-16: Arbitration Timing Diagram for an SBM

Bus Exchange Example Between Two 80960KB Processors

Figure 3-17 shows an example of bus arbitration between a PBM and an SBM using the arbitration signals. Each bus master performs a one-word read and a two-word write transaction to demonstrate the fastest possible bus exchanges.

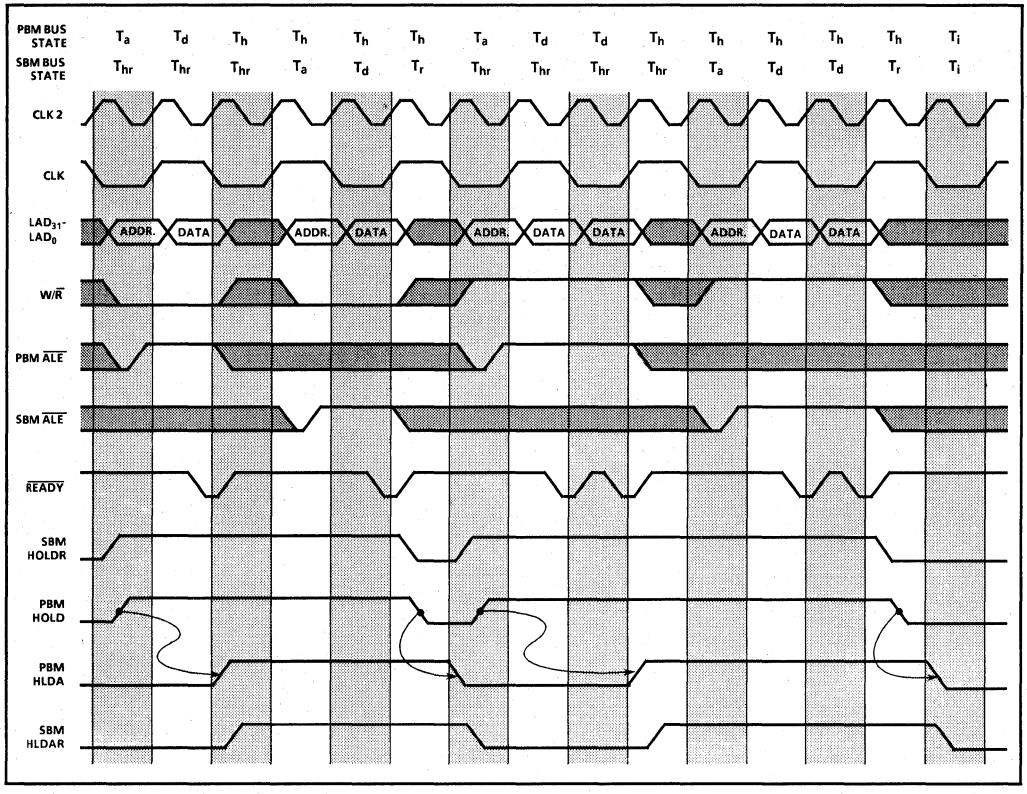


Figure 3-17: Example of a Bus Exchange Transaction

While the PBM is performing a read transaction, the SBM requests control of the L-bus by asserting HOLDR and entering the T_{hr} state. It remains in this state until the PBM grants the request by asserting HLDA after the read transaction is completed. After granting the request, the PBM enters the T_h state and remains in this state until its HOLD signal is deasserted. When the SBM completes the read transaction, it deasserts HOLDR and gives control back to the PBM.

The PBM now performs a two-word write transaction after deasserting the HLDA. The SBM requests control of the bus again by asserting the HOLDR signal and enters the T_{hr} state. When the PBM completes the two-word write transaction, it grants the request by asserting HLDA and enters the T_h state. The SBM receives the signal on the HLDAR input and performs a two-word write transaction. When the SBM completes the transaction, the control of the L-bus is transferred to the PBM, and both the PBM and the SBM enter the T_i state.

A Peripheral Device As the Default Bus Master

Another case exists where a peripheral device controls the L-bus, and the 80960KB processor requests control of the bus to perform operations. This alternative is not advisable because it hinders system performance. The exchange protocol is identical to the one described in the

previous section. The 80960KB processor is an SBM and uses two pins for bus arbitration: the HOLDR input pin and the HLDAR output pin. The state diagram is similar to the one shown in Figure 3-15. The lock conditions are not used for this case, however.

The peripheral device grants control of the L-bus bus asserting HLDAR when the SBM requests use of the L-bus. The peripheral device can obtain control of the L-bus again by deasserting HLDAR. If this occurs, the 80960KB processor surrenders control of the bus after it completes the current transaction, as shown in Figure 3-18. At that time, the 80960KB processor deasserts the HOLDR signal and places the other output signals into a high impedance state or a deasserted open drain level. The 80960KB processor may request access to the L-bus by asserting HOLDR again.

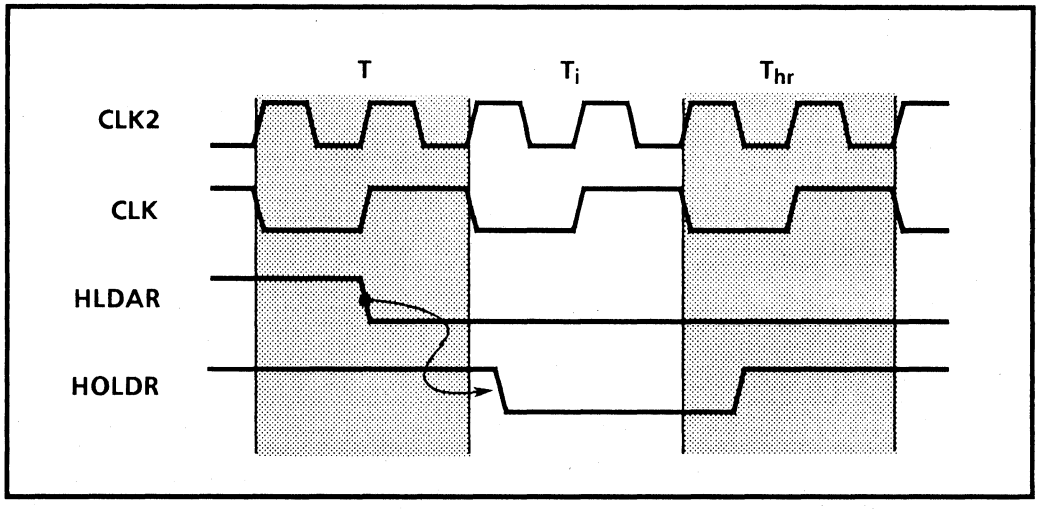


Figure 3-18: Forced Relinquishment Timing Diagram for an SBM

INTER-AGENT COMMUNICATION (IAC)

The IAC mechanism gives 80960KB processors the capability to send and receive messages to one another and to other bus agents. The IAC mechanism is essentially a non-maskable interrupt with pre-defined service routines. These routines are implemented in the 80960KB processor and are used to perform control functions such as purging the instruction cache, setting breakpoint registers, or stopping and starting the processor. By using IAC messages, external agents can remotely control the 80960KB. This allows easy integration of the 80960KB into system environments.

IAC messages can also be used to generate interrupts that behave exactly the same as hardwired interrupts. Since the interrupt vector is encoded in the IAC message, any of the 248 possible interrupt service routines can be invoked. For further information on IAC message definitions see the *80960KB CPU Programmer's Reference Manual.*

Overview of IAC Operations

Figure 3-19 shows a typical example of an IAC operation. In this case, an external processor gains control of the 80960KB by using an IAC operation. The external processor performs two functions: it writes the message in a buffer, called the message buffer; and it asserts the $\overline{IAC}$ pin of the 80960KB processor. Upon receipt of the $\overline{IAC}$ signal, the 80960KB processor stops executing its current process and performs a four-word read of the message buffer. After completing the read operation, the 80960KB processor automatically performs a one-word write operation to a pre-defined address to acknowledge the receipt of the message. The 80960KB processor then proceeds to perform the required action.

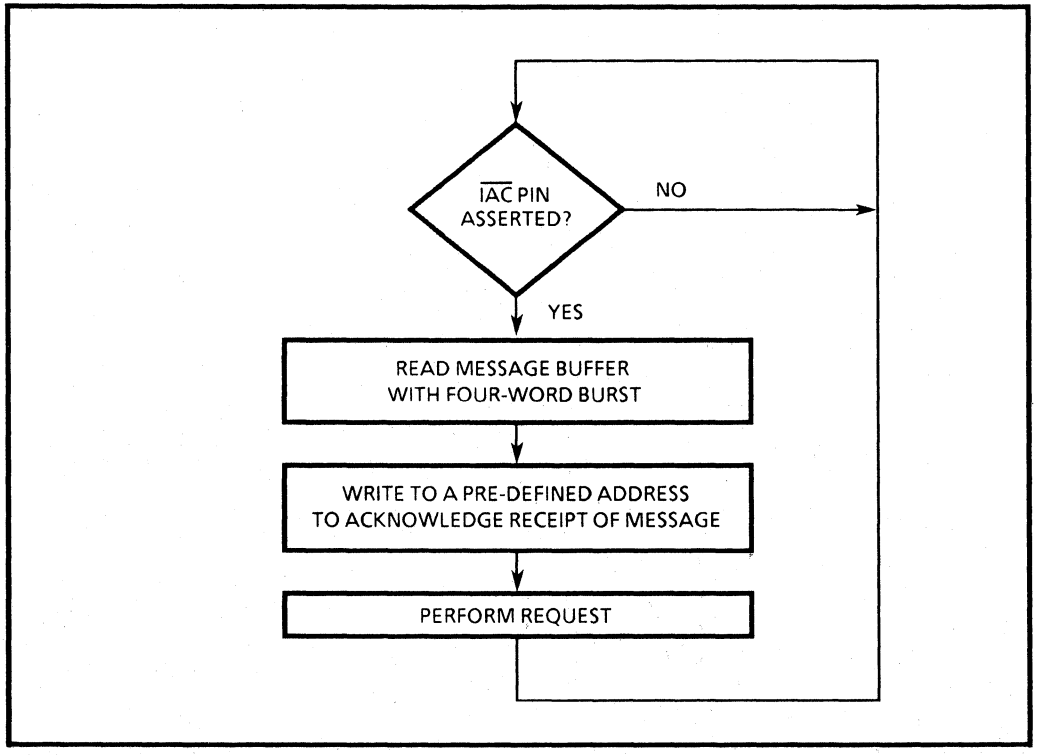


Figure 3-19: Example Flow Chart for an IAC Operation

IAC Messages

The IAC messages are specifically defined and behave much like machine instructions. The 80960KB processor reserves the upper 16M bytes ($FF000000_H$ to $FFFFFFFF_H$) of the 4M-byte address range for IAC message operations.

There are two types of IAC messages: internal and external. Internal IAC messages allow a program to send a command to its own processor. An internal IAC message is sent by writing to address $FF000010_H$. Internal IAC messages cause no L-bus activity.

External IAC messages can be used to send a command to another processor on the L-bus or to a remote processor. A processor sends an external IAC message by writing to a buffer area and causing the $\overline{IAC}$ pin of the receiving 80960KB to be asserted.

When the $\overline{IAC}$ pin is asserted, the recipient processor reads the reserved address to fetch the data from its IAC message buffer. After reading the IAC message buffer, the recipient does a write operation to another reserved address to acknowledge receipt of the IAC message. The $\overline{IAC}$ pin is deasserted as a result of this write operation, and the processor is ready to receive another IAC.

Hardware Requirements for External IAC Messages

To use the external IAC feature of the 80960KB, the following items are needed: a four-word message buffer RAM mapped to a reserved address to store the message, logic to assert the $\overline{IAC}$ pin of the 80960KB, and decoding logic to deassert the $\overline{IAC}$ pin on command from the 80960KB.

Message Buffers

Each 80960KB processor that receives an IAC message must have four 32-bit words of message buffer. This buffer can use special hardware or a reserved area in RAM. For proper operation of the buffer, two requirements must be met: the receiving 80960KB must be able to read this buffer at $FF000010_H$ if the receiving 80960KB's Local Processor Number (LPN) is equal to zero (see the "RESET and Initialization" section on page 3-33 for details of the LPN), or at $FF000030_H$ if the LPN is equal to one; and the sending processor must be able to write this buffer.

$\overline{IAC}$ Pin Logic

When the IAC message buffer receives a message, logic asserts the $\overline{IAC}$ pin and keeps it asserted. After the 80960KB processor reads the IAC message, it performs a one-word write to address FF000000 if its LPN is zero, or FF000020 if its LPN is one. This reserved address serves two functions: it causes external logic to deassert the $\overline{IAC}$ pin, and it maps to a register that contains the current processor priority. If the low order three bits of the data word have a value of 100_B (see Figure 3-20), the external logic should deassert the $\overline{IAC}$ pin on completion of the write operation.

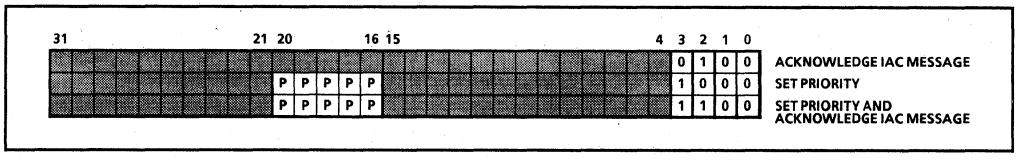


Figure 3-20: Data Settings

EXTERNAL PRIORITY REGISTER

The 80960KB contains an internal register that keeps track of the current priority (a value between 0 and 31) at which it is executing. This priority is used to decide whether or not to service interrupts -- higher priority interrupts are serviced, others are posted for later servicing. In some system designs it may be desirable to have this priority visible outside of the processor. To allow this, the 80960KB provides support for an external priority register. Whenever the priority of the 80960KB changes, the contents of this register are automatically updated.

This feature may be enabled in two steps. If the *Write External Priority* bit is set in the PRCB (see the *80960KB CPU Programmer's Reference Manual*), then the external priority register is updated as a result of a MODPC instruction or whenever an interrupt occurs. If external IAC messages are enabled, then external priority is also updated whenever a result of an IAC is to change processor priority.

Hardware Requirements

The 80960KB expects to write its priority into a 5-bit register mapped to address FF000000 if its LPN is zero, or FF000020 if its LPN is one. To set the priority, the processor performs a one-word write operation in the form shown in Figure 3-20. The priority is contained in bit_{20}-bit_{16}, and bit_3 is asserted to indicate that the priority is changed. It is necessary to use bit_3 as a qualifier to distinguish priority write operations from IAC message acknowledgments, which use the same reserved address.

External Priority and IAC Messages

The external priority register can be used to filter IAC messages. Since the processor always services the $\overline{IAC}$ pin (i.e., it is non-maskable), a low priority IAC message can interrupt a high priority task. To prevent this, a system can associate a priority with each IAC message. This priority can then be compared to the priority stored in the external priority register and used to decide whether or not to accept the IAC message. One way to associate a priority with an IAC message is to encode the message priority into the IAC message destination address as shown in Figure 3-21. The range of reserved addresses shown in Figure 3-21 have been set aside for this purpose.

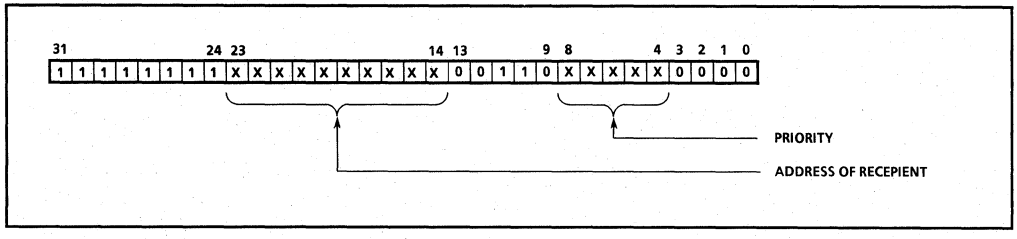


Figure 3-21: Physical Address Interpretation for IAC Messages

INTERRUPTS

The 80960KB processor responds to external events occurring at arbitrary times by means of an interrupt signal. Various sources, which include hardware components and special software instructions, generate an interrupt signal that can suspend execution of the 80960KB processor's current instruction stream. The hardware-generated interrupts are discussed in this section. For complete information on software-generated interrupts, see the *80960KB CPU Programmers Reference Manual.*

The 80960KB processor provides a flexible interrupt structure. The 80960KB processor can be interrupted using any of three methods below:

- Receipt of a signal on any or all of the four direct interrupt input signals ($\overline{INT}_0$, INT_1, INT_2, and $\overline{INT}_3$)
- Receipt of a signal on the interrupt request (INTR) line to obtain an external interrupt vector
- Receipt of an IAC message from a processor program or external source.

The choice of the method is determined by the setting in the on-chip Interrupt Control register. Interrupt signals can occur during any bus state regardless of which method is implemented.

This section provides details on the multiplexed interrupt pins, the three interrupt methods, the Interrupt Control register, synchronization, and interrupt latency.

Interrupt Signals

The interrupt signals are multiplexed on four pins of the 80960KB processor: $\overline{INT}_0/\overline{IAC}$, INT_1, INT_2/INTR, and $\overline{INT}_3/\overline{INTA}$. The on-chip Interrupt Control register determines how these pins are used (see "Interrupt Control Register" section on page 3-30).

$\overline{INT}_0/\overline{IAC}$ — This pin multiplexes the **Interrupt$_0$** and **Inter-Agent Communication** request input signals. The 80960KB processor interprets this input signal as either $\overline{INT}_0$ or $\overline{IAC}$. The $\overline{INT}_0$ signal indicates a request for interrupt service when it is asserted. The $\overline{IAC}$ signal denotes that a message is waiting when it is asserted.

INT_1 — The **Interrupt$_1$** input signal indicates a request for interrupt service when it is asserted.

INT_2/INTR — This pin multiplexes the **Interrupt$_2$** and **Interrupt Request** input signals. The 80960KB processor interprets this input signal as either INT_2 or INTR. The INT_2 signal indicates a request for interrupt service when it is asserted. The INTR signal indicates an interrupt request from an external interrupt controller. The 80960KB processor responds with an interrupt-acknowledge sequence. To ensure an interrupt, the INTR signal must remain asserted until the first cycle of the interrupt-acknowledge transaction.

$\overline{INT}_3/\overline{INTA}$ — This pin multiplexes the **Interrupt$_3$** input signal and **Interrupt Acknowledge** output signal. The 80960KB processor uses this pin as

the $\overline{INT_3}$ input signal or as the $\overline{INTA}$ output signal. The Interrupt Control register setting selects either the combination of INTR/$\overline{INTA}$ or $INT_2/\overline{INT_3}$. The $\overline{INT_3}$ input signal indicates a request for interrupt service when it is asserted. $\overline{INTA}$ acknowledges the interrupt request from an external interrupt controller. The $\overline{INTA}$ signal is latched by the 80960KB processor and remains valid during the T_d state. This signal is open drain output.

Interrupt Control Register

The 80960KB processor uses a 32-bit, on-chip Interrupt Control register to define the function of the multiplexed interrupt pins. This 32-bit Interrupt Control register allocates eight bits for each of the four direct interrupt signals ($\overline{INT_0}$, INT_1, INT_2, and $\overline{INT_3}$). The eight bits contain the vector number for each interrupt signal, as shown in Figure 3-22. The vector number is automatically read when one of the interrupt signals ($\overline{INT_0}$, INT_1, INT_2, and $\overline{INT_3}$) is activated. For example, when an interrupt is signaled on $\overline{INT_0}$, the 80960KB processor uses bit_7-bit_0 of the Interrupt Control register as the vector number.

The 80960KB processor uses the data field corresponding to $\overline{INT_0}$ to determine identification of the $\overline{INT_0}/\overline{IAC}$ input pin; a value of 00_H signifies the $\overline{IAC}$ function. If the data field corresponding to INT_2 has a value of 00_H, the 80960KB processor interprets the INT_2/INTR pin as the INTR input signal, and the $\overline{INT_3}/\overline{INTA}$ pin as the $\overline{INTA}$ output signal. In other words, this setting specifies that the 80960KB processor should use these two pins for communication with an external interrupt controller. If the functions of INTR and $\overline{INTA}$ are selected, the direct interrupt pins ($\overline{INT_0}$ and INT_1) can still be used.

The on-chip Interrupt Control register may be read and written by the Synchronous Load (synld) and Synchronous Move (synmov) instructions at the address $FF000004_H$ (see the *80960KB CPU Programmer's Reference Manual*). The value of the data fields in the Interrupt Control register is $FF000000_H$ after initialization. This setting specifies that the four interrupt pins function as $\overline{INTA}$, INTR, INT_1, and $\overline{IAC}$.

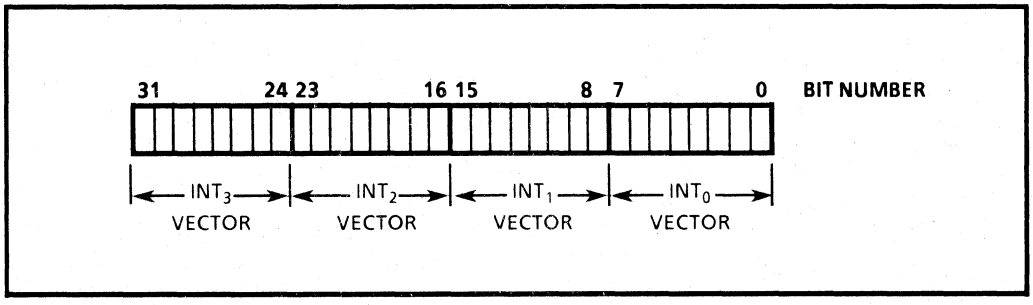


Figure 3-22: Interrupt Control Register

Using the Four Direct Interrupt Pins

The 80960KB processor can be interrupted by asserting any or all of the four interrupt input signals ($\overline{INT}_0$, $\overline{INT}_1$, INT_2, $\overline{INT}_3$). If the signals are simultaneously asserted, the 80960KB assumes that $\overline{INT}_0$ has the highest priority, followed by $\overline{INT}_1$, INT_2, and $\overline{INT}_3$. Software should follow this convention when programming the Interrupt Control register. When the interrupt input signals are asserted, the 80960KB processor utilizes a vector number specified by the Interrupt Control register as an index to an entry in the interrupt table located in memory. For complete software information on this topic, see the *80960KB CPU Programmer's Reference Manual.*

Using an External Interrupt Controller

The 80960KB processor can communicate with an external interrupt controller by performing an interrupt acknowledge sequence using the INTR and $\overline{INTA}$ signals. Figure 3-23 shows an example of the timing of an interrupt acknowledge sequence using the 8259A Programmable Interrupt Controller.

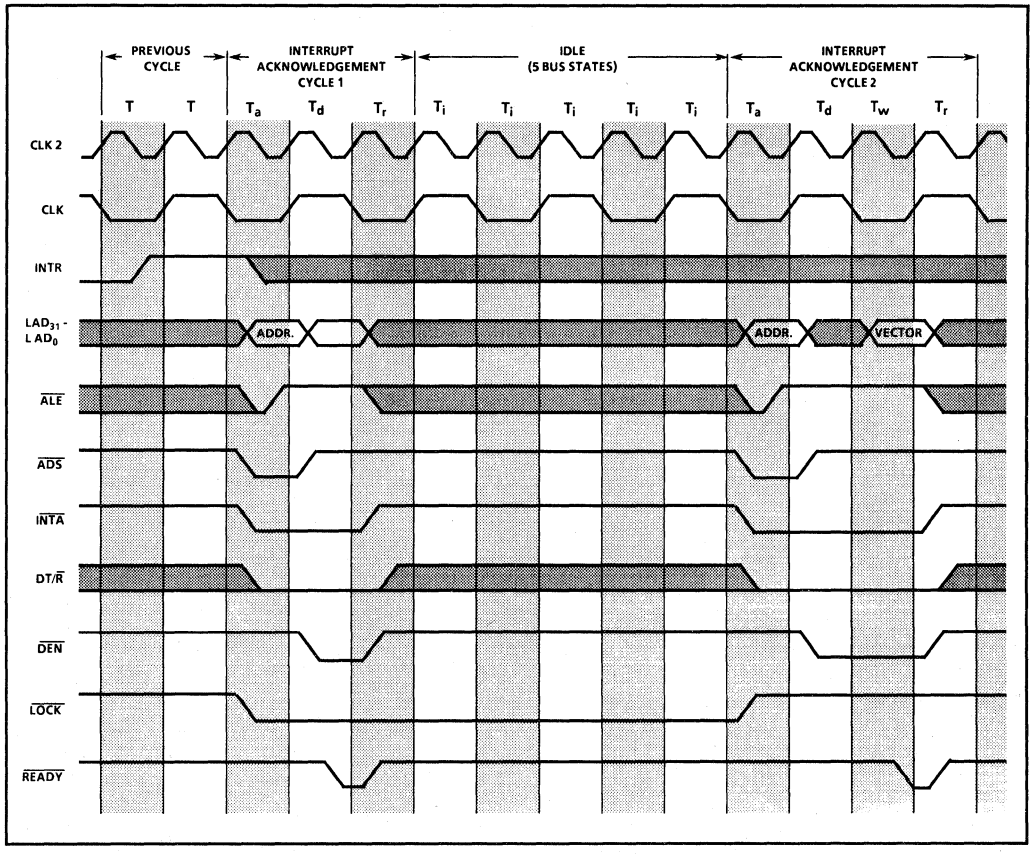


Figure 3-23: Timing Diagram for Interrupt Acknowledge Transaction

INTR is asserted by the 8259A and remains asserted until the 80960KB processor activates the $\overline{INTA}$ signal for the first time. When the 80960KB processor receives an interrupt request, the CPU completes the current transaction (or comes to some interruptible point), and asserts $\overline{INTA}$. $\overline{INTA}$ remains valid through the T_a, T_d, and T_w states. The first assertion of $\overline{INTA}$ triggers the 8259A to resolve priority among its interrupt requests.

To compensate for the timing of the 8259A, the 80960KB processor automatically inserts five T_i states before asserting the $\overline{INTA}$ again to read the interrupt vector. Figure 3-23 shows $\overline{READY}$ asserted without a wait state during the first Interrupt Acknowledgement cycle and with one wait state during the second Interrupt Acknowledgement cycle. In practice, the 8259A would require about four wait states in both cycles. The address during the T_a state for both interrupt acknowledge cycles is $FFFFFFFC_H$. For more details, see the "8259A Programmable Interrupt Controller" section in Chapter 5 on page 5-5.

The 80960KB processor services the interrupt according to its priority. If the interrupt has higher priority than the current activity, the 80960KB processor services it immediately. Otherwise, after reading the interrupt vector, the 80960KB processor posts the interrupt vector in the interrupt table. Typically, the 80960KB processor responds within 4 µs for an interrupt with higher priority than the current process (assuming CLK2 at 40 MHz). If the interrupt has lower priority than the current activity, the interrupt is serviced when its priority is higher than the priority of the subsequent activity of the 80960KB processor.

Using IAC Requests for Interrupts

The 80960KB processor can also be interrupted by an IAC message. The 80960KB processor can send IAC messages to itself by using one of the Synchronous Move instructions. Because this message does not utilize the L-bus when sent to the same processor, no special hardware is required. More details are provided on page 3-25 and in the *80960KB CPU Programmer's Reference Manual.*

Synchronization

The $\overline{INT_0}/\overline{IAC}$, INT_1, INT_2/INTR, and $\overline{INT_3}$ input signals can be either synchronous or asynchronous to the system clock (CLK2). Synchronous interrupt signals must be set up 3 ns prior to the rising edge of CLK2 and held for 10 ns after the rising edge of CLK2. To properly preset the interrupt signals for synchronous operation, $\overline{INT_0}/\overline{IAC}$, INT_1, INT_2/INTR, and $\overline{INT_3}$ must be deasserted for at least one processor clock cycle and asserted for at least one processor clock cycle. These signals may be deasserted and asserted individually.

If the interrupt signals are asynchronous to CLK2, the 80960KB processor internally synchronizes them. For the CPU to recognize the asynchronous interrupt input signals, they must be preset by deasserting them for at least two processor clock cycles, and then asserting them for at least two processor clock cycles. These signals may be deasserted and asserted individually.

RESET AND INITIALIZATION

The system RESET signal provides an orderly way to start or restart the 80960KB processor. When the 80960KB processor detects the low-to-high transition of RESET, it terminates all external activities and places the output pins in the high impedance state or deasserted condition. When the RESET signal falls low again, the 80960KB processor begins the initialization process and later starts fetching instructions from a specific address.

RESET Timing Requirements

To properly reset the 80960KB processor to a known state, the low-to-high transition of RESET must be asserted relative to any rising edge of CLK2 and remain asserted for at least 41 CLK2 cycles, as shown in Figure 3-24. RESET must be deasserted after the rising edge of CLK2, but prior to the next rising edge of CLK2. This establishes the next rising edge of CLK2 as edge A.

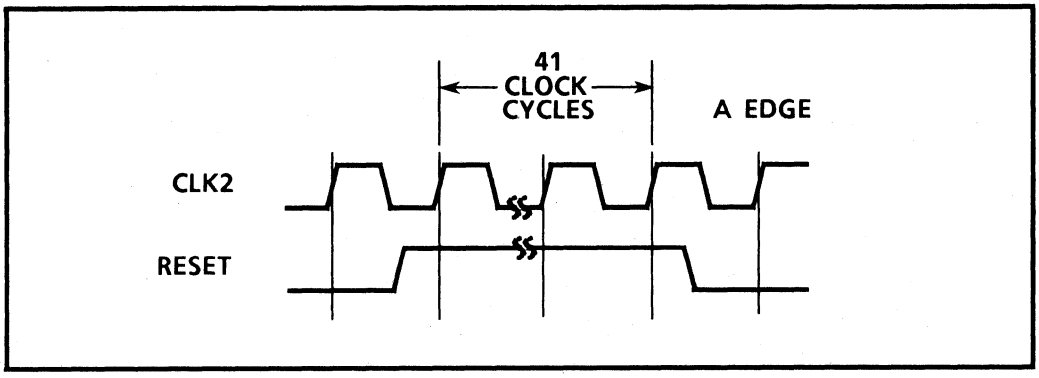


Figure 3-24: RESET Timing Diagram

RESET Timing Generation

The RESET input signal to the 80960KB processor can easily be generated by implementing a synchronization circuit comprised of a two D-type flip-flops, as shown in Figure 3-25.

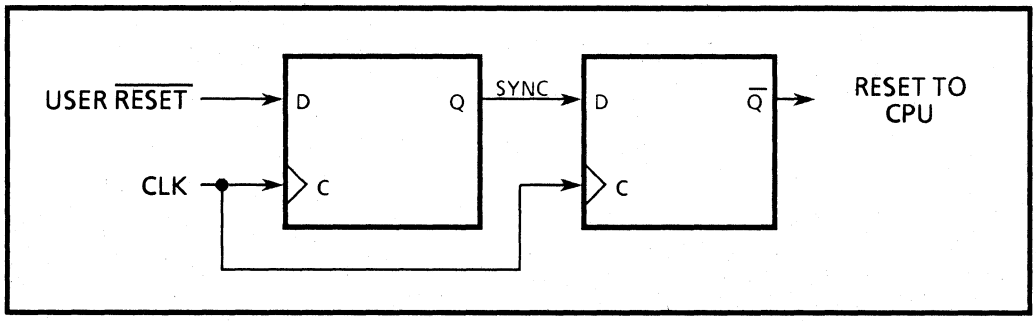


Figure 3-25: Asynchronous RESET Circuit

The user $\overline{\text{RESET}}$ signal is synchronized with the CLK signal by applying CLK to the clock input of both flip-flops. To protect against a metastable $\overline{\text{RESET}}$ signal, the output of the first flip-flop, SYNC, is applied to the input of the second flip-flop. The output of the second flip-flop results in a processor RESET signal. The timing diagram for these signals is shown in Figure 3-26. $\overline{\text{CLK}}$ or CLK2 can be used instead of CLK in Figure 3-26. Using $\overline{\text{CLK}}$ provides an edge A corresponding to the rising edge of CLK.

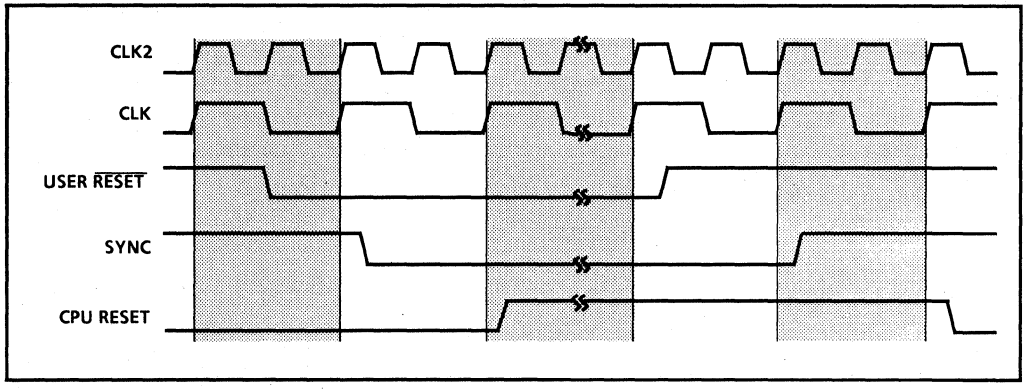


Figure 3-26: Diagram for RESET Timing Generation

This circuit assumes an asynchronous user $\overline{\text{RESET}}$ signal. If the user $\overline{\text{RESET}}$ signal is already synchronous with the CLK signal, the same circuitry can be implemented as shown in Figure 3-27. In this case, however, the output from the first flip-flop is used to generate the processor RESET signal rather than being routed to the input of the second flip-flop.

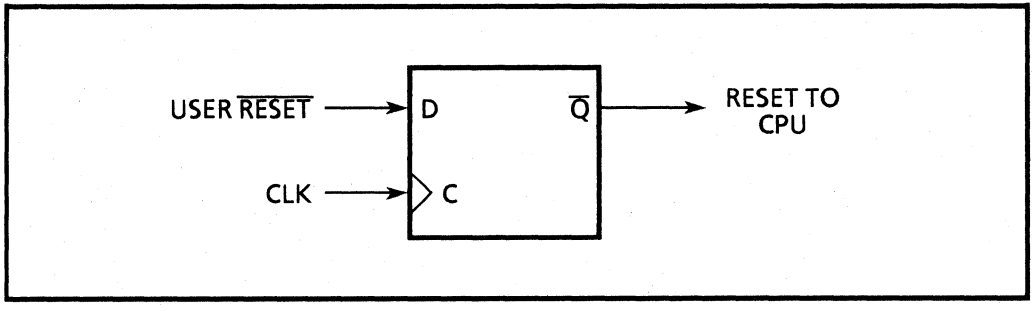


Figure 3-27: Synchronous RESET Circuit

Initialization

The initialization sequence of events is shown in Figure 3-28. When RESET is deasserted after a minimum of 41 CLK2 cycles, several actions take place: two input pins are sampled, the $\overline{\text{FAILURE}}$ output signal (see next section for the pin description) is asserted, and the self-test is performed.

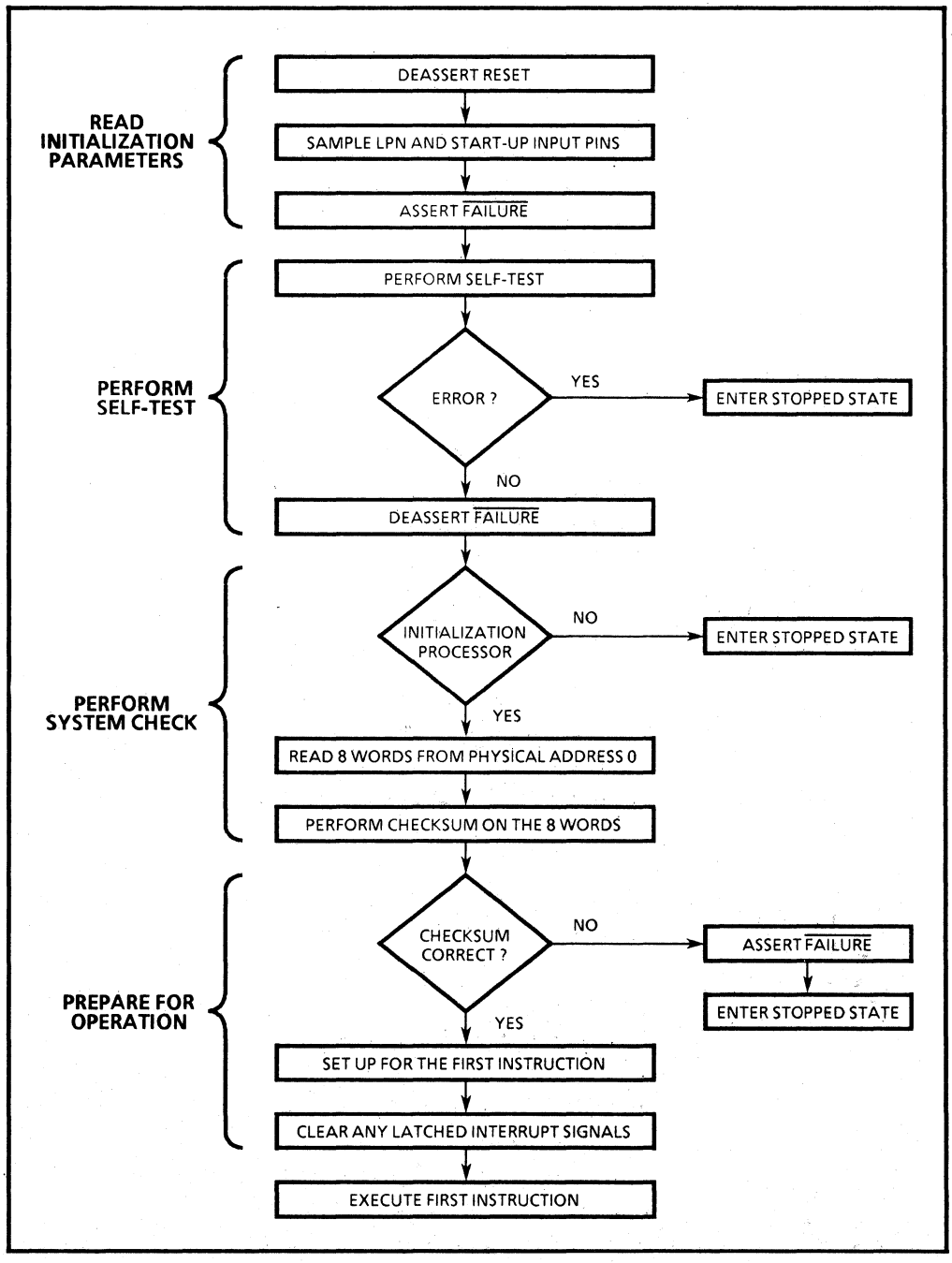


Figure 3-28: Initialization Flow Chart

When RESET is deasserted, the 80960KB processor samples the signals residing on the $\overline{INT_0}/\overline{IAC}$ and the $\overline{BADAC}$ input pins (see the next section for the pin description of $\overline{BADAC}$). At this time, these pins are interpreted as the *Local Processor Number* (LPN) and *Startup* (STARTUP) signals, respectively. The LPN input signal defines whether the 80960KB processor is a PBM (high voltage input level) or a SBM (low voltage input level). The STARTUP input pin indicates whether the 80960KB processor performs initialization (high voltage level) or not (low voltage level). The STARTUP signal is used to allow one or more processors to perform the active initialization. The input voltage levels for the LPN and STARTUP must be setup 3 ns before the rising CLK2 edge prior to edge A and held 10 ns beyond edge C, as shown in Figure 3-29.

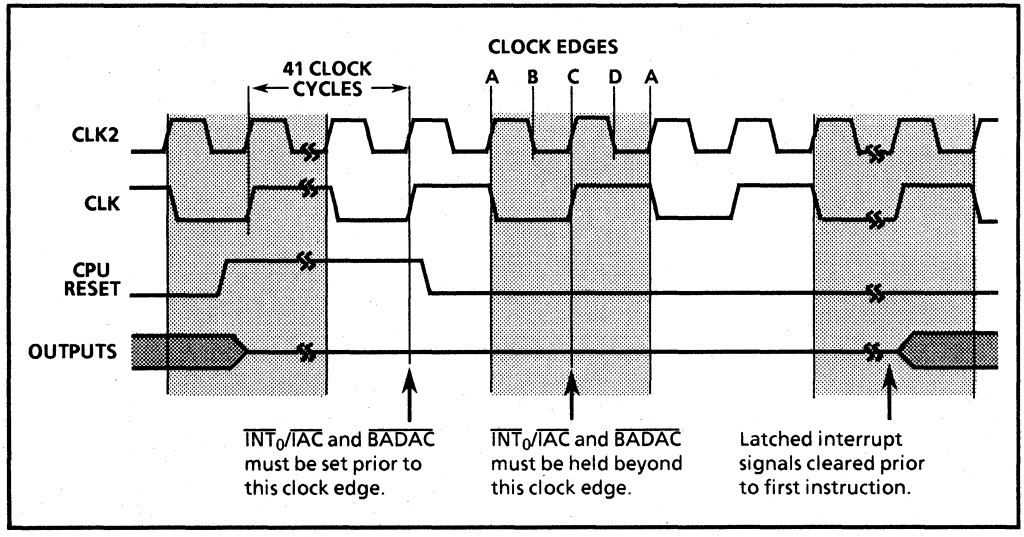


Figure 3-29: RESET Signal Timing Relationship

Besides sampling the two input pins, the 80960KB processor asserts the $\overline{FAILURE}$ output signal a few cycles after RESET is deasserted. The $\overline{FAILURE}$ signal remains asserted while the CPU performs the self-test. If a failure is detected during the self-test, $\overline{FAILURE}$ remains asserted and the CPU enters the stopped state where the processor does nothing. If the self-test completes successfully, the CPU deasserts the $\overline{FAILURE}$ signal.

An 80960KB processor that is designated as the initialization processor proceeds by doing a checksum test of eight words fetched from memory at physical address $0000\ 0000_H$ to ensure that the memory and L-bus are operating properly. If the initial checksum is incorrect, then the $\overline{FAILURE}$ signal is asserted (and remains asserted) and the 80960KB processor enters the stopped state. After a successful checksum test, the 80960KB processor uses some of the words as addresses to initial data structures. Complete details are provided in the *80960KB CPU Programmer's Reference Manual.*

Just prior to executing the first instruction, the 80960KB processor clears any latched interrupt signals.

ERROR SIGNALS

The 80960KB processor provides an input signal ($\overline{\text{BADAC}}$) for notification of an error in the system, and provides an output signal ($\overline{\text{FAILURE}}$) for notification of an error within the processor.

$\overline{\text{BADAC}}$ — When asserted, the **Bad Access** input signal indicates that an unrecoverable error occurred during the current data transfer. If, however, $\overline{\text{BADAC}}$ was asserted after a Synchronous Move or Synchronous Load instruction, the error is recoverable. The 80960KB processor samples the $\overline{\text{BADAC}}$ input signal during the cycle following the one when the last $\overline{\text{READY}}$ is asserted.

$\overline{\text{FAILURE}}$ — The **$\overline{\text{FAILURE}}$** signal indicates that an error occurred during initialization. The 80960KB processor always asserts $\overline{\text{FAILURE}}$ after the activation of the RESET signal. If a failure is detected during a self-test, $\overline{\text{FAILURE}}$ remains asserted. Otherwise, the processor deasserts $\overline{\text{FAILURE}}$ after a successful self-test is performed. If the initial memory checksum is incorrect, the initialization processor asserts $\overline{\text{FAILURE}}$ a second time, and keeps it asserted. $\overline{\text{FAILURE}}$ is an open drain output signal.

SUMMARY

The L-bus is a high speed 32-bit multiplexed bus with burst-transfer capability and is designed to operate with the high performance 80960KB processor. The L-bus consists of two signal groups: address/data, and control. These signal groups are utilized by the 80960KB processor to perform read, write, and burst transactions.

The arbitration, interrupt, and reset operations are related to the L-bus transactions. The arbitration operation transfers control of the L-bus to another bus master. Three methods are available to handle interrupts: by invoking the on-chip interrupt controller, by employing an external interrupt controller using the INTR/$\overline{\text{INTA}}$ signals, by using an IAC message. The reset function sets the 80960KB processor to a known internal state after it successfully completes the self-test. These operations offer power and flexibility to hardware system design using the 80960KB processor.

This chapter focuses on the L-bus and how it relates to the 80960KB processor. The next two chapters develop guidelines on designing memory and peripheral devices in the hardware system.

Memory Interface 4

CHAPTER 4
MEMORY INTERFACE

The high-speed bus interface has many features that enhance high-performance designs. In particular, the burst-transfer feature allows up to four successive 32-bit data word transfers at a maximum rate of one word every processor clock cycle. This chapter outlines approaches for memory designs that use these features, describes memory design considerations, analyzes the timing, and lists a number of useful examples. The concepts illustrated by these examples apply to a wide variety of memory system implementations.

BASIC MEMORY INTERFACE

Figure 4-1 shows the major logic blocks of the memory interface circuit. The data transceivers buffer the data to compensate for any slow devices that may be connected to the 80960KB processor. The address latches demultiplex the address/data signals from the 80960KB processor and latch the address. The address decoder selects the appropriate memory device from the latched address. To accommodate a memory burst transaction, the burst logic decrements the word count, increments the local address lines 3 and 2 (LAD_3 and LAD_2), and generates a CYCLE-IN-PROGRESS signal. The timing control generates a $\overline{READY}$ signal and other specific signals required by a particular memory device. The byte enable latch stores the byte enable signals.

Although not part of the basic memory interface, the DRAM controller, SRAM interface, DRAM, SRAM, and EPROM are included in Figure 4-1 for completeness. In a hardware system the DRAM, SRAM, and EPROM are typically located in separate subsystems.

Although the memory interface circuit can be designed using programmable logic, gate arrays, or other custom logic, the examples use standard components wherever possible to illustrate the design concepts.

Data Transceivers

Standard 8-bit transceivers can be used to provide isolation and additional drive capability for the L-bus. Transceivers can be used to prevent bus contention that can occur if some memories are slow to remove data from the L-bus after a read operation. For example, if a write operation follows a read operation, the 80960KB processor may drive the L-bus before a slow device has removed its output data, potentially causing a current spike on the power and ground lines. Transceivers, however, can be omitted if the data float time of the device is short enough and the load does not exceed the 80960KB device specifications.

The data transceivers can be controlled by two signals from the 80960KB processor: data transmit/receive ($DT/\overline{R}$) and data enable ($\overline{DEN}$). $DT/\overline{R}$ indicates the direction of data flow and $\overline{DEN}$ enables the transceivers.

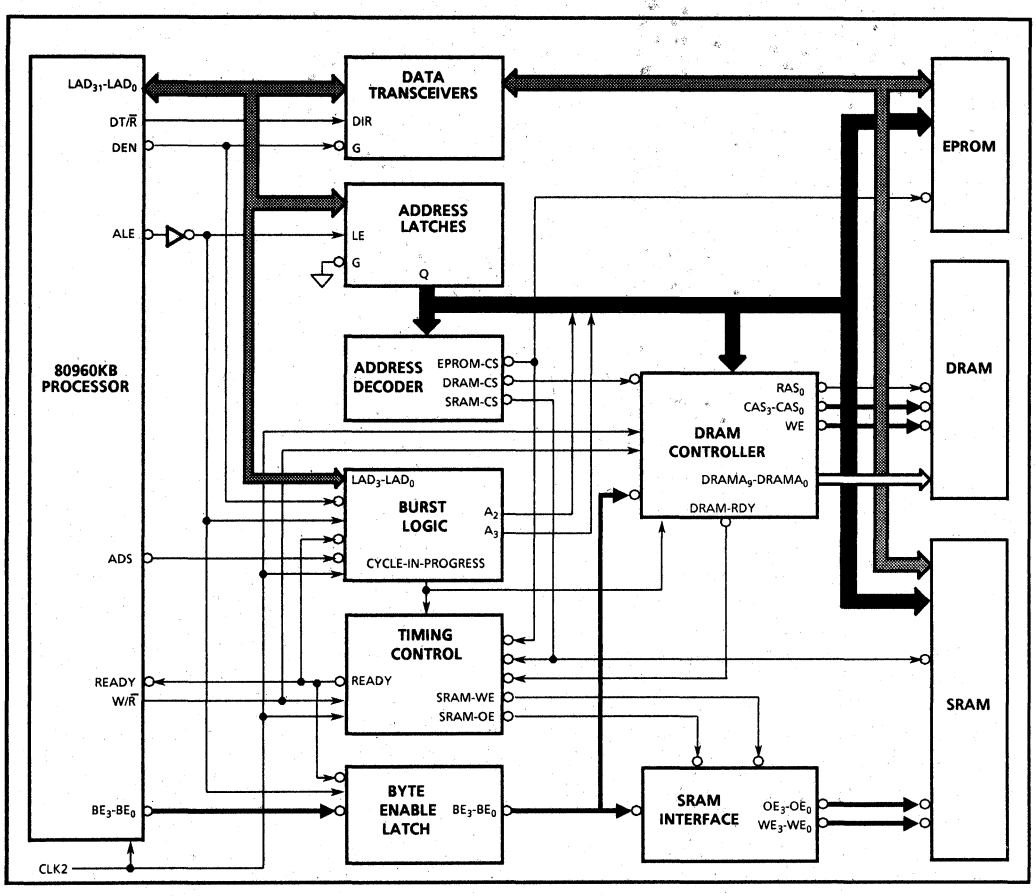


Figure 4-1: Simplified Block Diagram for Memory Interface Logic

Address Latch/Demultiplexer

Conventional transparent latches can be used to demultiplex the address/data lines of the 80960KB processor and to hold the address constant during the memory operation. The latch is controlled by the $\overline{ALE}$ signal from the 80960KB processor. $\overline{ALE}$ passes through an inverter, so that when $\overline{ALE}$ goes low, the address flows through the latch. The low-to-high transition of $\overline{ALE}$ can be used to latch the address. The output enable of the latch can be tied to ground. The lower four address lines (LAD_3-LAD_0) are latched by the burst logic.

Address Decoder

The 80960KB processor accesses both memory and I/O devices by supplying a 32-bit address and a read/write command. The address decoder determines which particular memory or I/O device is selected by decoding the address lines. The following discussion focuses on memory selection, and the "Address Decoder" section in Chapter 5 on page 5-3 discusses I/O device selection using memory-mapped I/O techniques.

The memory address can be divided into regions where one region can apply to EPROM or ROM, another to RAM, and another to the I/O registers. In a 80960KB-based system the ROM address space is likely to start at address 0000 0000_H because the CPU begins execution at this address. The RAM or I/O regions can start at any other address in the 4G-word address range except for addresses $FF000000_H$ through $FFFFFFFF_H$, which the 80960KB processor reserves for inter-agent communication.

Because of the large address range of the 80960KB processor, the address can be divided into word address bits and chip select bits. Typically the higher-order address bits are decoded to generate the selection signal for ROM, RAM, or I/O devices.

The address decoder can be located either before or after the address latches. Usually, it is placed after the latches, so that the chip-select signal does not need to be latched. Figure 4-1 shows the block diagram of the address decoder placed behind the address latches.

Burst Logic

To enhance system performance, the 80960KB processor performs burst transactions that transfer up to four data words at a maximum rate of one word every clock cycle. A DRAM controller can be designed that takes advantage of the burst-transfer capability by using the static column mode or nibble mode features of the DRAM (see the "DRAM Controller" section on page 4-5). This DRAM controller requires a signal, called CYCLE-IN-PROGRESS, to identify the start and end of a memory cycle. The burst logic generates the CYCLE-IN-PROGRESS signal.

Figure 4-2 shows the flow chart for the burst logic. If $\overline{ADS}$ is low and $\overline{DEN}$ is high, then the burst logic latches LAD_3 through LAD_0, and asserts the CYCLE-IN-PROGRESS signal. The burst logic checks the SIZE signals (LAD_1 and LAD_0). If the value of the SIZE signals equal zero, then the burst logic runs one memory cycle, and terminates the CYCLE-IN-PROGRESS signal. If the value of the SIZE signals do not equal zero, the burst logic runs one memory cycle, increments the address (LAD_3 and LAD_2), and decrements the value of the SIZE signals. When this is finished, the burst logic checks the value of the SIZE signals again.

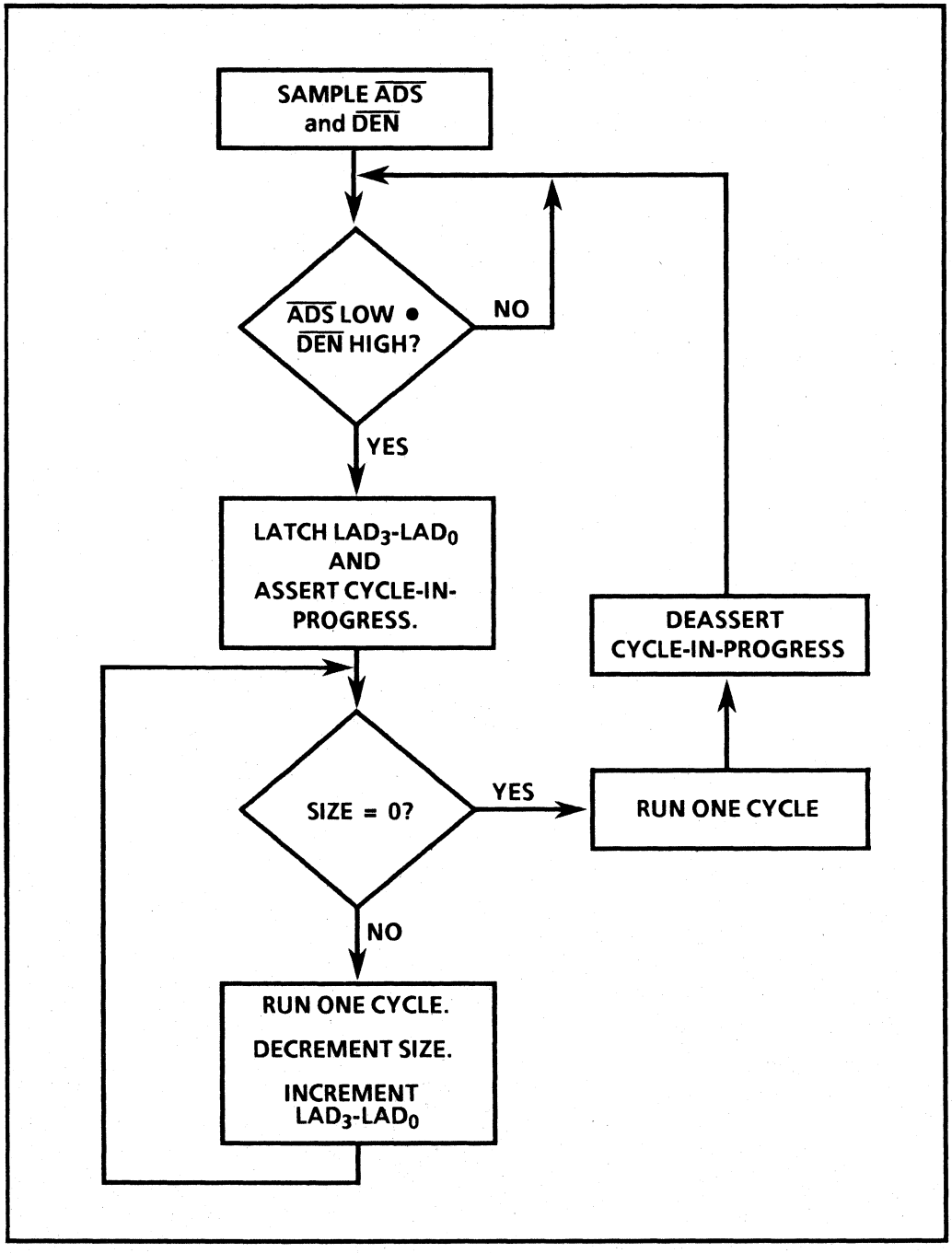


Figure 4-2: Burst Logic Flow Chart

The burst logic can be used with memory devices, such as EPROM, SRAM, or DRAM without static column mode or nibble mode, which do not inherently support a burst transaction. Because the 80960KB processor ensures that a burst transaction cannot exceed four words or cross a 16-byte boundary, incrementing LAD_3 and LAD_2 after a single data word transfer makes the burst transfer transparent to these devices.

Timing Control Logic

The timing control logic accommodates memory devices that cannot transfer information at the maximum bus rate by generating a $\overline{\text{READY}}$ signal when the data is available. The timing control logic consists of a counter and timing logic, as shown in Figure 4-3. The counter produces a 4-bit binary count. The count begins when the CYCLE-IN-PROGRESS signal is asserted. The timing logic asserts $\overline{\text{READY}}$ at the appropriate time based upon the count, the $\overline{\text{EPROM-CS}}$, and the $\overline{\text{SRAM-CS}}$ signals. For a burst transfer, $\overline{\text{READY}}$ resets the counter to properly time a $\overline{\text{READY}}$ signal for the next data transfer. When CYCLE-IN-PROGRESS is deasserted, the clock counting is terminated.

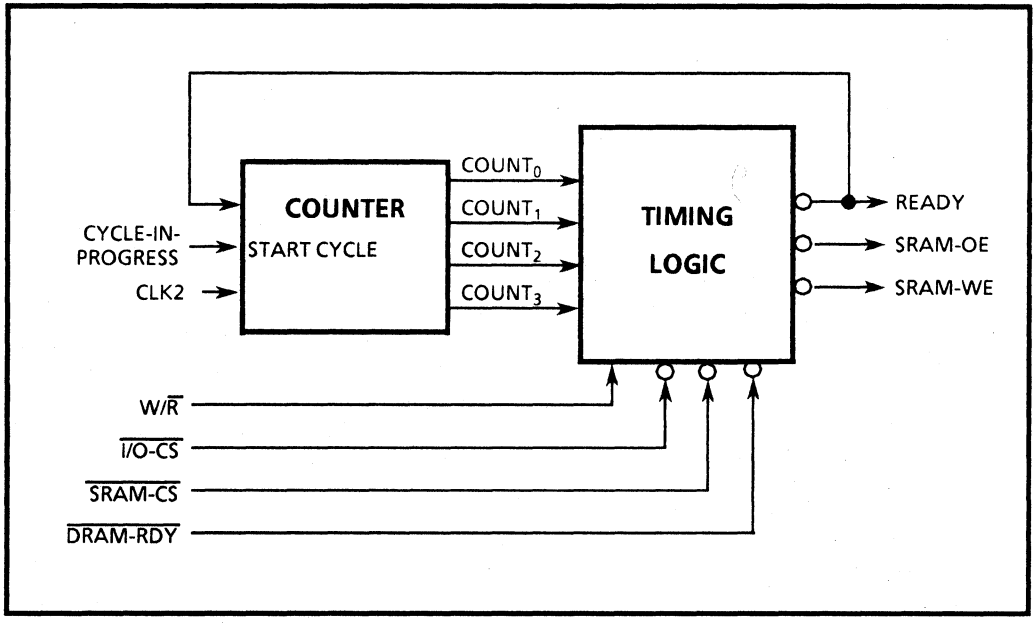


Figure 4-3: Timing Control Logic Block Diagram

Because the timing of DRAM is more complicated, the DRAM controller generates a $\overline{\text{DRAM-RDY}}$ signal to the 80960KB processor. In addition, the clock count, the W/$\overline{\text{R}}$ command, and $\overline{\text{SRAM-CS}}$ signal can also be used to generate $\overline{\text{SRAM-WE}}$ and $\overline{\text{SRAM-OE}}$ signals.

Byte Enable Latch

The byte enable latch holds the byte enable signals constant until the DRAM controller or SRAM interface uses the signals. As mentioned in the "L-Bus Signal Groups" section in Chapter 3 on page 3-4, the byte enable signals specify which bytes (up to four) on the 32-bit data bus are transferred during the data cycle. Each individual byte enable signal selects eight data lines as shown in Table 4-1.

Table 4-1: Byte Enable Signal Decoding

BYTE ENABLE SIGNAL	ADDRESS LINE SELECTION
$\overline{BE}_0$	LAD_7.LAD_0
$\overline{BE}_1$	LAD_{15}.LAD_8
$\overline{BE}_2$	LAD_{23}.LAD_{16}
$\overline{BE}_3$	LAD_{31}.LAD_{24}

The byte enable signals are valid from the 80960KB processor before data is transferred. These signals are asserted during the address cycle for the first data word transfer; they are asserted again during the first data cycle for the second word transfer; the second data cycle for the third word transfer; and the third data cycle for the fourth word transfer. For each word, the byte enable signals remain valid throughout every data or wait cycle until $\overline{READY}$ is asserted. After $\overline{READY}$ is asserted, the byte enable signals change during the next processor clock cycle.

The $\overline{ALE}$ signal can be used to latch the first byte enable signals. $\overline{READY}$ can be used to latch the other byte enable signals for each word.

SRAM INTERFACE

The basic memory interface can be used in conjunction with the SRAM interface to read and write to SRAM. This section describes the SRAM interface and examines the timing.

SRAM Interface Logic

The SRAM interface logic uses the latched byte enable signals, the $\overline{SRAM\text{-}WE}$, and the $\overline{SRAM\text{-}OE}$ signals to generate four output enable signals ($\overline{SRAM\text{-}OE}_3$ through $\overline{SRAM\text{-}OE}_0$) and four write enable signals ($\overline{SRAM\text{-}WE}_3$ through $\overline{SRAM\text{-}WE}_0$), as shown in Figure 4-4. These signals allow the 80960KB processor to write to the data byte that is specified by the byte enable signals. SRAMs with separate $\overline{OE}$ and $\overline{CS}$ signals require only one $\overline{OE}$ signal per bank since the 80960KB ignores unrequested bytes in read operations.

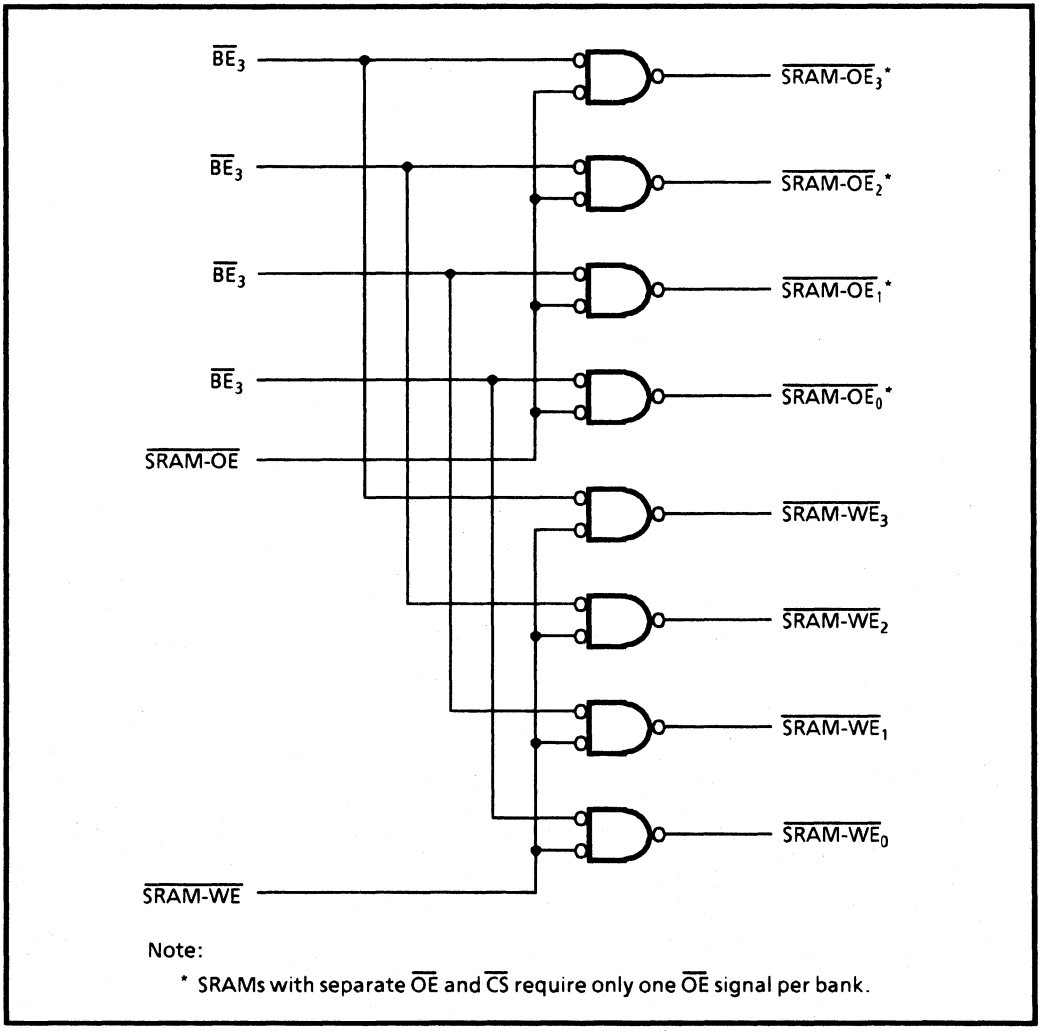


Figure 4-4: Block Diagram for SRAM Interface

SRAM Timing Considerations

This section analyzes the critical timing paths of the SRAM control signals. From the critical path, the timing equations can be derived to determine the memory access time for no wait state operation.

When evaluating critical timing paths, the timing calculations should use worst-case parameter specifications, rather than typical specifications. By using worst-case timing values, reliable operation is assured over all variations in temperature, voltage, and individual device characteristics. These timing values are determined by assuming the maximum propagation delay to latch an address, select a memory device, and pass through data buffers and transceivers.

Figure 4-5 shows the critical timing path for a one-word SRAM read operation. The diagram consists of three time periods: the address setup period ($T_{addrset}$), the memory response period (T_{mem}), and the data return period ($T_{dataset}$). Note that the timing for the read command and output control signals does not enter into the critical timing path.

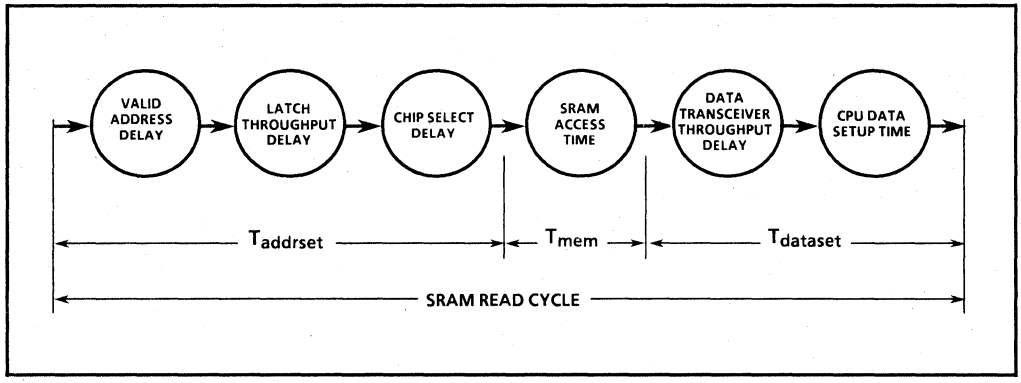


Figure 4-5: Critical Timing Path for SRAM Read Operation

During the $T_{addrset}$ period, the 80960KB processor outputs a valid address that is latched on the low-to-high transition of the $\overline{ALE}$ signal. The address decoder generates the $\overline{SRAM\text{-}CS}$ signal from the latched address. During the T_{mem} period the SRAM responds to the commands and signals and retrieves the data. The access time of the memory determines the duration of the T_{mem} period. T_{mem} can be varied in increments of clock cycles by delaying the $\overline{READY}$ signal.

The data must be available at the address/data pins of the CPU before the end of the data state. The $T_{dataset}$ period must take into account the setup time requirement of the 80960KB processor and the throughput delay of a data transceiver.

For a no wait state operation, the data transfer word must be completed in two system clock (CLK) cycles. The minimum time period for a no wait state operation ($T_{mem\text{-}no\text{-}wait}$) can be determined by using equation 1.

$$T_{mem\text{-}no\text{-}wait} = 2CLK - T_{addrset} - T_{dataset} \quad (1)$$

where: $T_{mem\text{-}no\text{-}wait}$ = Memory access time for no wait state operation

$2CLK$ = Two system clock (CLK) cycles

$T_{addrset}$ = Maximum delay to valid address
+ Maximum throughput delay of address latch
+ Maximum delay to generate chip select

$T_{dataset}$ = Maximum delay through data transceiver
+ Maximum data setup time of CPU

A similar analysis can be done for burst transactions. Equation 1 can be used to determine the access time for no wait state operation of the first word. For subsequent words, equation 2 can be used. In this equation, the address setup time is replaced by delay in the burst logic to change the address (T_{burst}). In this case, the data transfer of each subsequent word must be completed in one system clock (CLK) cycle (no address state). The minimum access time for a no wait state operation ($T_{mem\text{-}no\text{-}wait}$) can be determined by using the lesser value of equation 1 or equation 2.

$$T_{mem\text{-}no\text{-}wait} = CLK - T_{burst} - T_{dataset} \quad (2)$$

where: $T_{mem\text{-}no\text{-}wait}$ = Memory access time for no wait state operation

CLK = One system clock (CLK) cycles

T_{burst} = Maximum delay to change the address

$T_{dataset}$ = Maximum delay through data transceiver + Maximum data setup time of CPU

The memory access time can be extended by delaying the $\overline{READY}$ signal and adding wait states.

The timing analysis described for a SRAM read operation can be used for EPROM timings. If EPROMs are only used to store initialization programs, they are seldom accessed compared to memory devices used to store program data or instructions. Consequently, the addition of wait states during the read cycle does not affect overall system performance.

Figure 4-6 shows the critical timing path for an SRAM write operation. The diagram consists of two time periods: the address setup period ($T_{addrset}$) and the memory response period (T_{mem}).

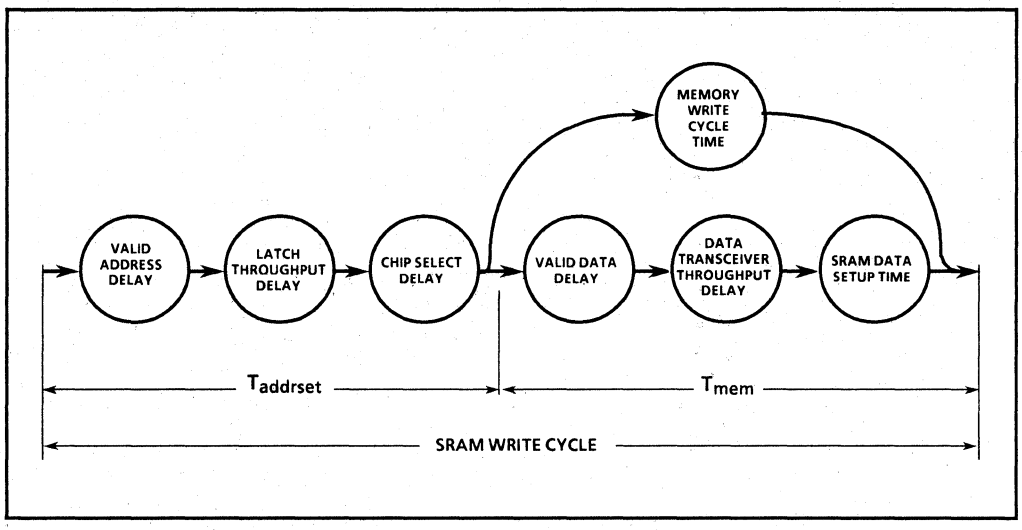


Figure 4-6: Critical Timing Path for SRAM Write Transaction

During the $T_{addrset}$ period, the 80960KB processor outputs a valid address that is latched on the low-to-high transition of $\overline{ALE}$. The address decoder generates the $\overline{SRAM\text{-}CS}$ signal from the latched address.

During the T_{mem} period the SRAM responds to the commands and writes the data. The access time of the memory determines the duration of the T_{mem} period. T_{mem} can be varied in increments of clock cycles by delaying the $\overline{READY}$ signal.

Two timing paths should be considered during the T_{mem} period: the path where data is supplied to the memory, and the path that monitors the memory write cycle time. The first path takes into account the time for the 80960KB processor to generate valid data, the throughput delay of a data transceiver, and the data setup time requirement of the memory. The second path is the memory write cycle specification. The longer of the two paths is the critical timing path.

By examining the timing path required to operate the SRAM, equation 2 can be derived which determines SRAM write cycle time for no wait state operation. The memory cycle time is determined by the lesser value of equation 1 or equation 2.

$$T_{mem\text{-}no\text{-}wait} = 2CLK - T_{addrset} \quad (3)$$

where: $T_{mem\text{-}no\text{-}wait}$ ≥ Maximum delay to valid data
+ Maximum throughput delay of data transceiver
+ Maximum data setup time of memory

2CLK = Two system clock (CLK) cycles

$T_{addrset}$ = Maximum delay to valid address
+ Maximum throughput delay of address latch
+ Maximum delay to generate chip select

The memory access time can be extended by delaying the $\overline{READY}$ signal and generating wait states.

DRAM CONTROLLER

This section provides design guidelines for a DRAM controller. DRAMs offer static column mode and $\overline{CAS}$ before $\overline{RAS}$ refresh features. This section shows guidelines on how to use these features with the burst capability of the 80960KB processor to significantly enhance system throughput.

The DRAM controller multiplexes the address into a row and column address, performs the refresh operation, arbitrates between a refresh request and memory request, and generates the necessary control signals for the DRAM. To implement these functions, the memory controller uses an address multiplexer, arbiter, refresh interval timer, and DRAM timing and control as shown is Figure 4-7.

A standard DRAM controller can be used, but it typically degrades system performance.

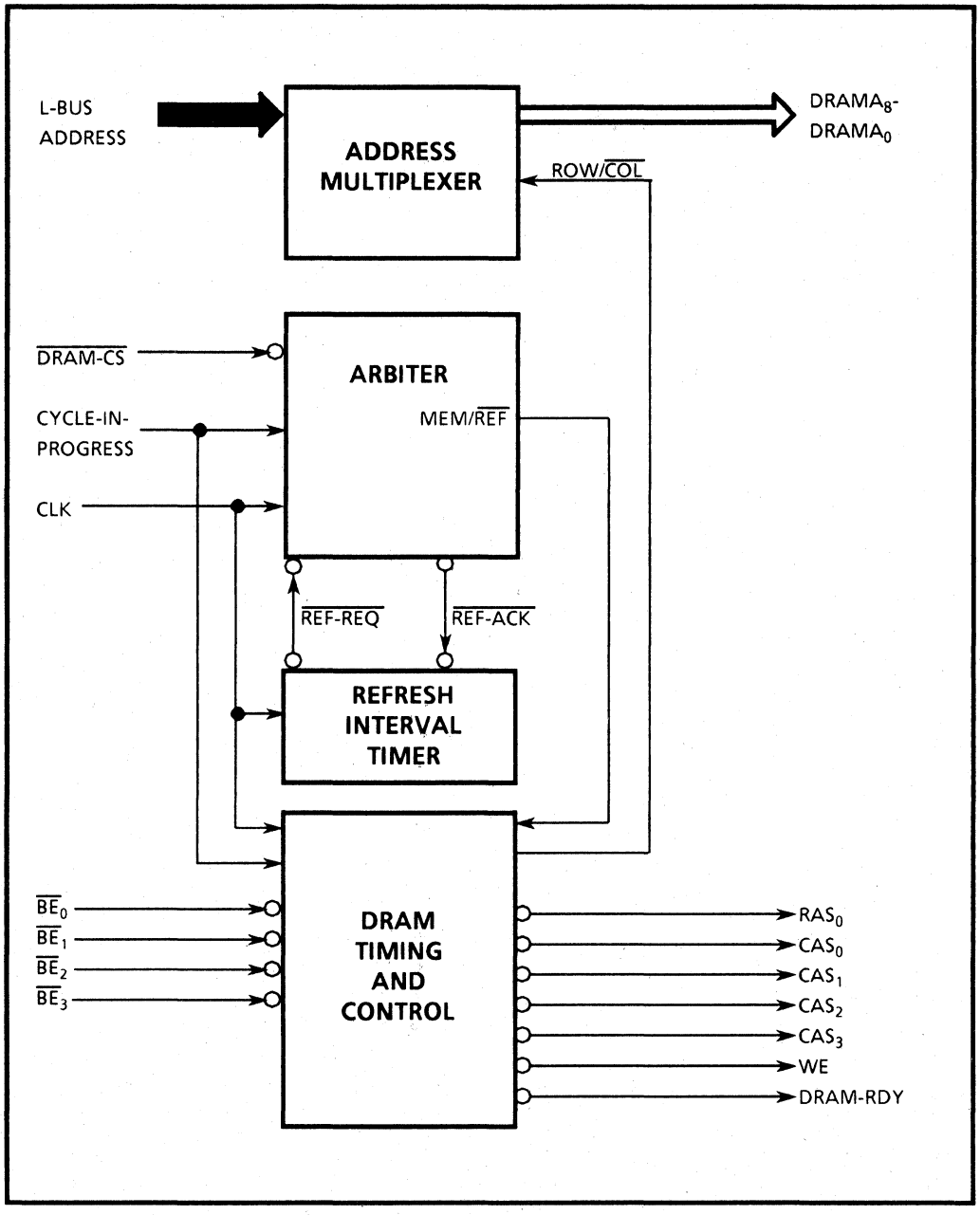


Figure 4-7: DRAM Controller Block Diagram

Address Multiplexer

The address multiplexer divides the DRAM address into a row and column address. The proper selection of a row or column address is accomplished by the row/column select signal (ROW/$\overline{\text{COL}}$) from the DRAM timing and control circuit.

Refresh Interval Timer

The refresh interval timer periodically generates a refresh request ($\overline{\text{REF-REQ}}$) by counting enough bus cycles to equal the refresh interval period. Since a refresh request is processed after a completed operation, the refresh period must take into account the time required to perform a bus operation, as well as the DRAM refresh specification. For example, a 1M-bit DRAM that requires 512 refresh cycles within 8 ms needs a refresh cycle every 15.6 µs. To meet the DRAM specification, the refresh interval timer must generate a refresh request in less than 15.6 µs to compensate for any required time to complete the operation with wait states.

After the $\overline{\text{REF-REQ}}$ signal is generated, the arbiter sends a refresh acknowledge signal $\overline{\text{REF-ACK}}$ back to the interval timer to assure that refresh occurred before generating another $\overline{\text{REF-REQ}}$.

Arbiter

DRAM controller uses an arbiter to decide whether a memory cycle or refresh cycle is performed. In a synchronous design, arbitration is easily performed because memory and refresh cycle requests never occur at or near the same time.

The arbiter monitors memory cycle requests and refresh requests. The arbiter detects a DRAM memory request by decoding two signals: $\overline{\text{DRAM-CS}}$ and CYCLE-IN-PROGRESS. The $\overline{\text{REF-REQ}}$ signal indicates that a refresh cycle must be performed. The arbiter arbitrates between a memory cycle or refresh cycle and generates a Memory/Refresh (MEM/$\overline{\text{REF}}$) signal. The DRAM timing and control block uses the MEM/$\overline{\text{REF}}$ signal to start the generation of the control signals.

When a refresh cycle is performed, the arbiter sends a $\overline{\text{REF-ACK}}$ signal to the refresh timer, which uses this signal to begin another count.

DRAM Timing and Control

The DRAM timing and control circuit is the final logic block and core of the DRAM controller. The functions of this circuit include the following:

- Generating the DRAM control signals ($\overline{\text{RAS}}$, $\overline{\text{CAS}}$, and $\overline{\text{WE}}$) with the proper timing relationships during system operation
- Generating the $\overline{\text{DRAM-RDY}}$ signal
- Performing the refresh function by asserting $\overline{\text{CAS}}$ before $\overline{\text{RAS}}$

- Performing several warm-up cycles required by the DRAM when power is first applied.

The DRAM timing and control logic can be designed to take advantage of the burst-transfer capability of the 80960KB processor by implementing static column mode or nibble mode. With nibble mode, a multiplexed address is applied to the DRAM, and up to four bits of data are quickly transferred by successively toggling the $\overline{CAS}$ pulse. The DRAM timing and control logic can be designed to provide the successive $\overline{CAS}$ pulses by using the CYCLE-IN-PROGRESS and $\overline{DRAM\text{-}RDY}$ signals. Static column mode can also be used to take advantage of the burst capability of the DRAM. Static column mode allows fast access to the bits located in the selected row of the DRAM simply by changing the column address after the first access.

Figure 4-8 shows a flow chart for the DRAM timing and control logic using static column mode. The DRAM timing and control circuit receives a refresh request or a memory request on the MEM/$\overline{REF}$ and CYCLE-IN-PROGRESS input signals. For a memory request, the DRAM timing and control determines whether a read or a write operation is desired from the W/$\overline{R}$ signal from the 80960KB processor.

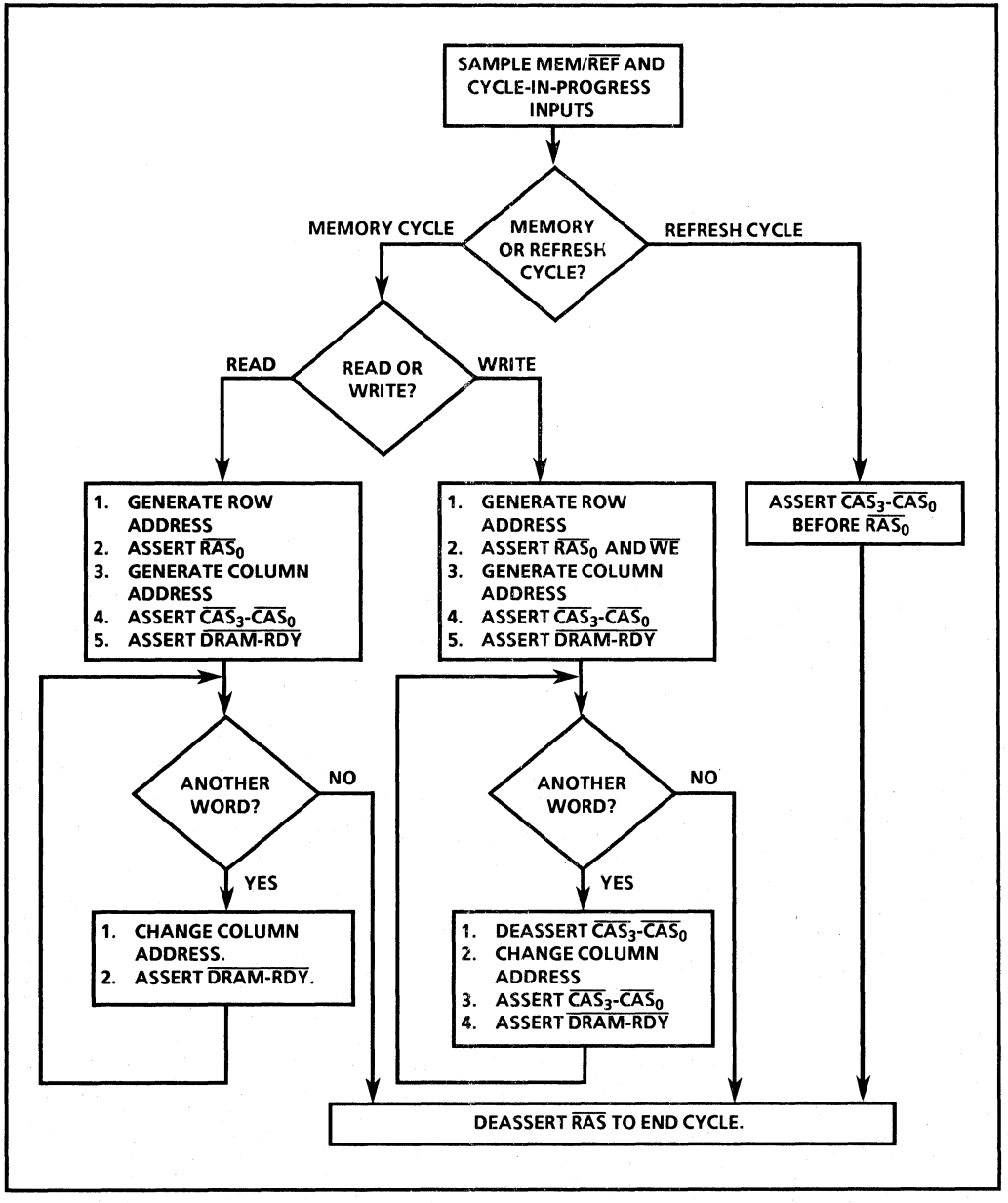


Figure 4-8: Flow Chart for DRAM Timing and Control Logic

For a read operation, the DRAM timing and control logic performs several functions: it brings ROW/$\overline{\text{COL}}$ high to select a row address; it asserts $\overline{\text{RAS}}_0$; it brings ROW/$\overline{\text{COL}}$ low to select the column address; it asserts $\overline{\text{CAS}}_3$ through $\overline{\text{CAS}}_0$ (derived from the four latched byte enable signals); and it generates a $\overline{\text{DRAM-RDY}}$ signal. The $\overline{\text{DRAM-RDY}}$ signal causes the burst logic to increment the address and the 80960KB processor to read the data word.

After completing these functions the DRAM timing and control logic samples the CYCLE-IN-PROGRESS to determine whether to transfer another data word. If so, the DRAM timing and control logic maintains the ROW/$\overline{\text{COL}}$ signal low to select the new column address and generates another $\overline{\text{DRAM-RDY}}$. The DRAM timing and control logic repeats the procedure until all the data words are transferred. Then the DRAM timing and control logic deasserts $\overline{\text{RAS}}_0$ and observes the necessary row precharge specification of the DRAM.

For a write operation, the DRAM timing and control logic performs similar functions on the first word: it asserts $\overline{\text{WE}}$; it brings ROW/$\overline{\text{COL}}$ high to select a row address; it asserts $\overline{\text{RAS}}_0$; it brings ROW/$\overline{\text{COL}}$ low to select the column address; it asserts $\overline{\text{CAS}}_3$ through $\overline{\text{CAS}}_0$ (derived from the four latched byte enable signals); and it generates a $\overline{\text{DRAM-RDY}}$ signal. The $\overline{\text{DRAM-RDY}}$ signal causes the burst logic to increment the address and informs the 80960KB processor that the data word was written.

After completing these functions the DRAM timing and control logic samples the CYCLE-IN-PROGRESS to determine whether to transfer another data word. If so, the DRAM timing and control logic maintains the ROW/$\overline{\text{COL}}$ signal low to select the new column address, deasserts and asserts $\overline{\text{CAS}}_3$ through $\overline{\text{CAS}}_0$ to observe the $\overline{\text{CAS}}$ precharge specification of the DRAM, and generates another $\overline{\text{DRAM-RDY}}$. The DRAM timing and control logic repeats the procedure until all the data words are transferred. Then the DRAM timing and control logic deasserts $\overline{\text{RAS}}_0$ and observes the necessary row precharge specification of the DRAM.

Although only one $\overline{\text{RAS}}$ signal is generated, four $\overline{\text{CAS}}$ signals ($\overline{\text{CAS}}_3$-$\overline{\text{CAS}}_0$) are generated to enable each byte of the L-bus. These $\overline{\text{CAS}}$ signals are triggered by the four write signals generated by the byte enable decoder and correspond to the byte enable signals of the 80960KB processor. For example, $\overline{\text{CAS}}_0$, which is mapped directly from $\overline{\text{BE}}_0$, selects the least-significant data byte (LAD_7-LAD_0).

A single $\overline{\text{WE}}$ control signal and four $\overline{\text{CAS}}$ 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 $\overline{\text{CAS}}$ control signal and four $\overline{\text{WE}}$ control signals, using the $\overline{\text{WE}}$ signals to write the DRAM bytes selectively in write cycles that use fewer than 32 bits. Although the selected bytes are written correctly, the unselected bytes are enabled for a read cycle. These bytes output their data to the unselected bytes of the data bus while the data transceivers output data to every bit of the data bus. When the two devices simultaneously output data to the same bus, bus contention occurs.

The refresh function can be performed simply by asserting $\overline{\text{CAS}}$ signal before asserting $\overline{\text{RAS}}$. The $\overline{\text{CAS}}$ before $\overline{\text{RAS}}$ refresh feature eliminates the need for an external refresh address counter. When the $\overline{\text{CAS}}$ pulse is activated prior to the assertion of the $\overline{\text{RAS}}$ pulse, the DRAM automatically performs a refresh cycle on one row by employing an on-chip address counter. Upon completion of the refresh cycle, the address counter is automatically incremented. The MEM/$\overline{\text{REF}}$ signal from the arbiter can be used by the DRAM timing and control logic block to initiate a $\overline{\text{CAS}}$ before $\overline{\text{RAS}}$ refresh cycle.

Besides generating the $\overline{\text{RAS}}$, $\overline{\text{CAS}}$, and $\overline{\text{WE}}$ signals, the DRAM timing and control logic generates a number of warm-up cycles for the DRAM after reset by issuing several refresh requests.

Timing Considerations for the DRAM Controller

Figure 4-9 shows a typical example of a timing diagram for a two-word read transaction that uses static column mode; similarly, Figure 4-10 is a typical example for a two-word write transaction. The example assumes a memory access time that requires two wait states (T_W) for the initial data word and one wait for the second data word.

The critical timing areas for both read and write transactions are noted by circled numbers in the diagrams, which are explained in the enumerated list below.

1. The delay for the CPU to generate a valid address.
2. The delay for the DRAM timing and control logic to generate the CYCLE-IN-PROGRESS signal.
3. The delay to generate the DRAM row address. This time includes the address latch throughput delay, the multiplexer throughput delay, and the address driver delay.
4. The delay to generate $\overline{\text{RAS}}$, which includes the delay to generate the $\overline{\text{DRAM-CS}}$ signal.
5. The row address hold time after the high-to-low transition of $\overline{\text{RAS}}$.
6. The time required to generate the multiplexer control signal (ROW/$\overline{\text{COL}}$) after the row address hold time is satisfied.
7. The time required to switch from a row to column address plus any driver delays.
8. The delay to generate and drive the $\overline{\text{CAS}}$ signals.
9. For a read transaction, the throughput delay of the data transceivers. For a write transaction, the delay by the CPU to generate valid data.
10. For a read transaction, the data setup time of the CPU. For a write transaction, the throughput delay of the data transceivers.
11. The time required to increment and drive the column address.
12. For a write transaction only, the delay time to bring $\overline{\text{CAS}}$ high (terminate the $\overline{\text{CAS}}$ pulse for the first data byte), to precharge the $\overline{\text{CAS}}$ pulse (required by the DRAM), and to assert $\overline{\text{CAS}}$ again.
13. The $\overline{\text{RAS}}$ precharge time, which must be satisfied before another memory cycle can begin.

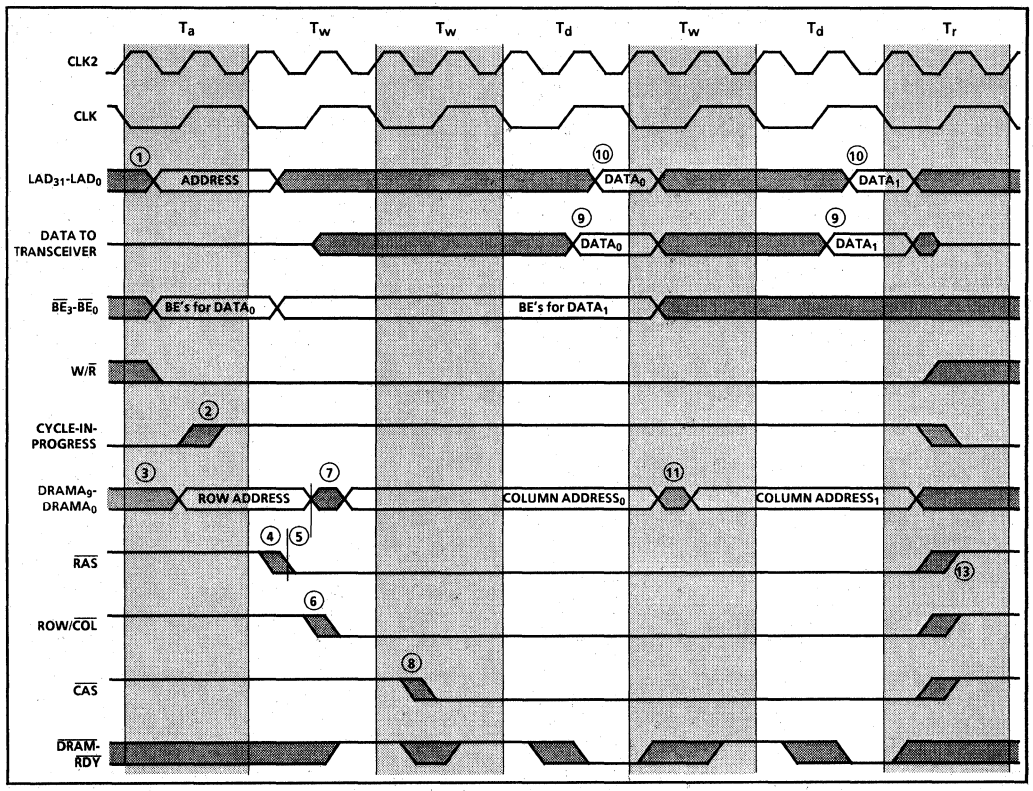


Figure 4-9: Timing Diagram for Two-word DRAM Read Transaction

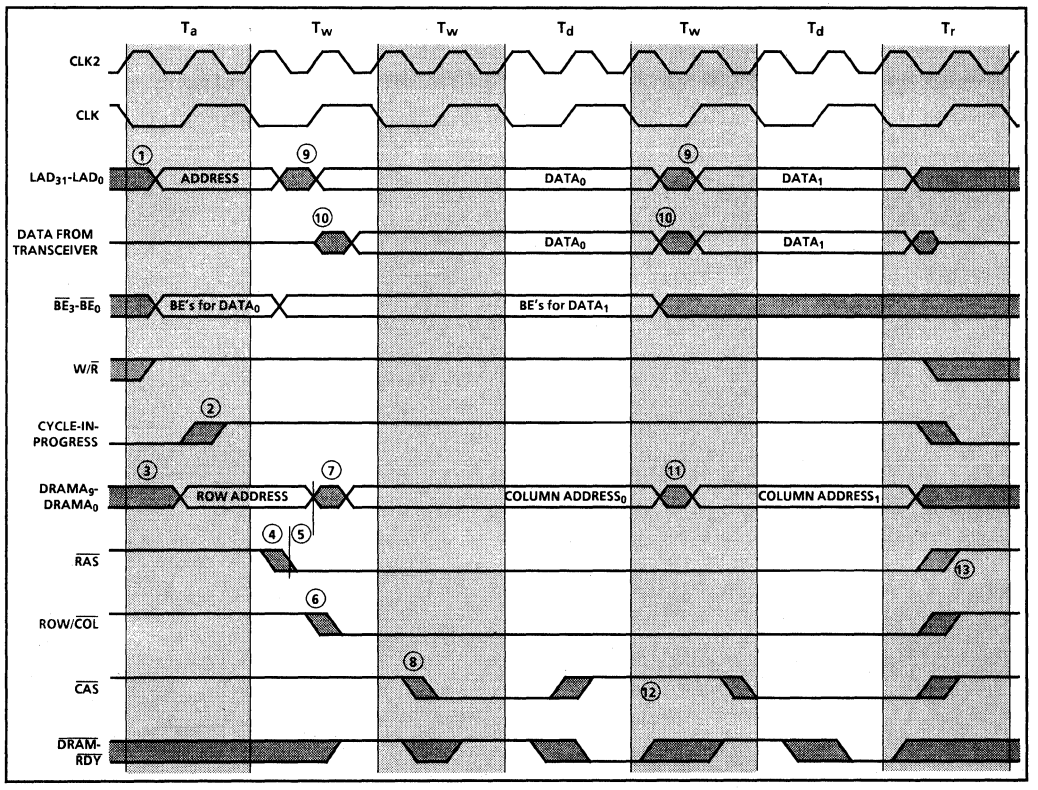


Figure 4-10: Timing Diagram for Two-word DRAM Write Transaction

DRAM Interleaving

Because the DRAM consists of dynamic nodes, a row precharge time is required to recharge the nodes after every memory cycle. This time must be included in the timing evaluation, as noted by the example. To avoid the precharge time delay of the DRAM, the memory array can 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 is ready to perform the next access immediately.

If DRAMs are interleaved (i.e., arranged in multiple banks so that adjacent addresses are in different banks), the DRAM precharge time can be masked for most accesses. With two banks of DRAMs, one for even 32-bit addresses and one for odd 32-bit addresses, all sequential 32-bit accesses can be completed without waiting for the DRAM to precharge.

Even when 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 masked when the 80960KB processor 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.

SUMMARY

The memory interface circuit allows the 80960KB processor to communicate with the memory devices. The basic memory interface logic can be divided into six blocks: the data transceivers, the address latches, the address decoder, the burst logic, the DRAM timing and control logic, and the byte enable latch. The DRAM controller and SRAM interface complete the memory interface circuit. The DRAM controller can be designed to take advantage of the 80960KB processor's burst capability to enhance system performance.

This chapter focuses on the design guidelines for the memory interface design to the 80960KB processor. Chapter 5 develops guidelines on designing peripheral devices in the single-processor hardware system.

I/O Interface 5

CHAPTER 5
I/O INTERFACE

The 80960KB processor supports 8-bit, 16-bit, and 32-bit I/O devices that can be mapped into the 4G-byte memory address space. This chapter describes the design considerations for the interface between the 80960KB processor and I/O components. Several examples illustrate the design concepts.

INTERFACING TO 8-BIT AND 16-BIT PERIPHERALS

The 80960KB processor accesses I/O devices by using a memory-mapped address. Consequently, memory-type instructions can be used to perform input/output operations. For example, the 80960KB processor's LOAD and STORE instructions can be used to move 8-bit and 16-bit data to I/O peripherals. The instructions include those listed below.

- Load Ordinal Byte (reads a byte)
- Load Ordinal Short (reads 16-bit data)
- Store Ordinal Byte (writes a byte)
- Store Ordinal Short (writes 16-bit data)

These instructions perform the transfer on the data bits specified by the two low-order lines of the effective address. See the *80960KB CPU Programmer's Reference Manual* for complete details.

GENERAL SYSTEM INTERFACE

In a typical 80960KB processor system design, a number of slave I/O devices can be controlled through a general system interface. Other I/O devices, particularly those capable of controlling the L-bus, can use the general system interface, but may require additional logic to isolate the bus. This section describes the general system interface and assumes that the 80960KB processor does not perform burst transactions to the I/O devices.

Figure 5-1 shows the major logic blocks of the general system interface. Standard 8-bit data transceivers add drive capability, provide bus isolation, and prevent bus conflicts that may occur with slow I/O components. The address latch demultiplexes the address/data lines and holds the address stable throughout the L-bus transaction. The address decoder generates the I/O chip-select signals from the latched address lines. The timing control block provides the $\overline{\text{READY}}$ signal to the 80960KB processor and the I/O read and I/O write command.

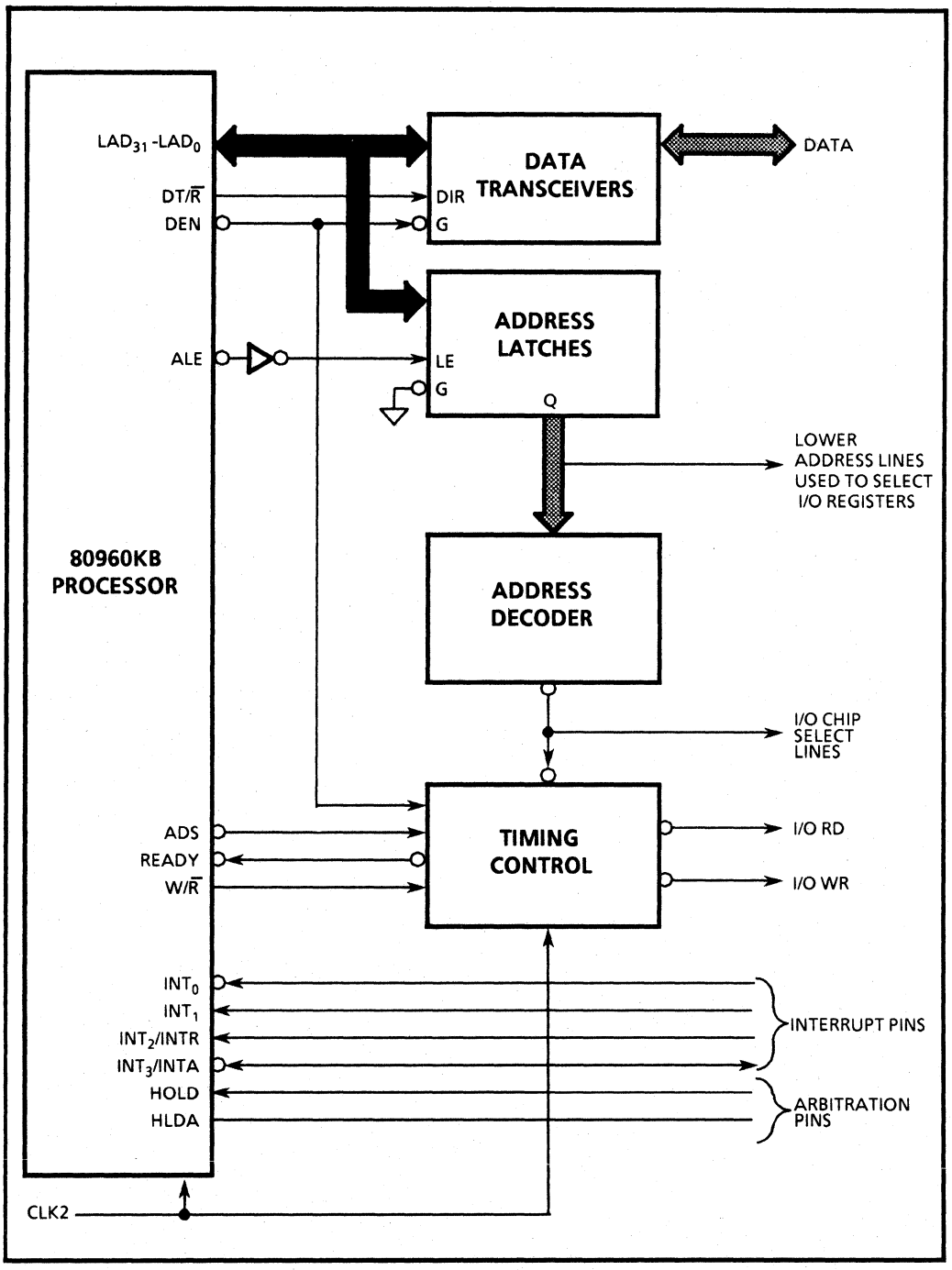


Figure 5-1: Simplified General System Interface

This basic interface circuit is similar to the one used in the basic memory interface described in Chapter 4. For most systems the same data transceivers, address decoders, and address latches can be used to access both memory and I/O devices. The timing control logic can be designed to accommodate both memory and I/O devices.

Data Transceivers

Standard 8-bit transceivers can be used to provide isolation and additional drive capability for the L-bus. Transceivers prevent bus contention that can occur if some devices are slow to remove data from the data bus after a read cycle. For example, if an I/O write cycle follows a I/O read cycle, the 80960KB processor may drive the L-bus before a slow device has removed its outputs from the bus, potentially causing a current spike. Transceivers, however, can be omitted if the data float time of the device is short enough and the load does not exceed the 80960KB device specifications.

The data transceiver can be controlled by two signals from the 80960KB processor: Data Transmit/Receive ($DT/\overline{R}$) and Data Enable ($\overline{DEN}$). $DT/\overline{R}$ indicates the direction of data flow and $\overline{DEN}$ enables the transceivers.

Address Latch/Demultiplexer

Standard transparent latches can be used to demultiplex the address/data lines of the 80960KB processor. The latch is controlled by the $\overline{ALE}$ signal from the 80960KB processor. The $\overline{ALE}$ signal passes through an inverter, such that when $\overline{ALE}$ goes low, the address flows through the latch. The low-to-high transition of $\overline{ALE}$ can be used to latch the address.

If only slave-type peripherals are used in a system, the output enable of the latches can always remain active by connecting it to ground. For systems with DMA devices, the output enable can be used to permit the DMA device to drive a common address bus.

Address Decoder

The address decoder determines which particular I/O device is selected by decoding the address. The I/O address can be any address in the 4G-word address range except for the upper 16M bytes (addresses $FF000000_H$ through $FFFFFFFF_H$), which the 80960KB processor reserves for inter-agent communication and internal I/O. Typically, a small range of address bits is reserved for accessing I/O devices by designating certain higher-order address bits to mean an I/O access.

For example, consider a 32-bit address: A_{31} through A_{15} can indicate an I/O access when A_{31} is set to zero, and A_{30}-A_{15} are set to one; A_{14} through A_5 can be used to specify a particular I/O device; and A_4 through A_2 can be used to access up to 8 registers of the I/O component. A_1 and A_0 are not used by the I/O device. This particular scheme selects up to 1,024 devices, while using only 32K bytes of address space.

The address decoder can be located either before or after the address latches. Usually, it is placed after the latches, so that the chip-select signal does not need to be latched.

Timing Control Logic

The timing control logic accommodates I/O devices that cannot transfer information at the maximum bus rate by generating a $\overline{\text{READY}}$ signal when the data is available. The timing control logic consists of a counter and timing logic, as shown in Figure 5-2. The counter produces a 4-bit binary count. The count is started at the beginning of the operation (determined by $\overline{\text{ADS}}$ and $\overline{\text{DEN}}$) and is stopped by the $\overline{\text{READY}}$ signal. The timing logic asserts the $\overline{\text{READY}}$ signal, the I/O write command ($\overline{\text{I/O-WR}}$), and the I/O read command ($\overline{\text{I/O-RD}}$) based upon the clock count, the I/O chip select signal ($\overline{\text{I/O-CS}}$), and the W/$\overline{\text{R}}$ command.

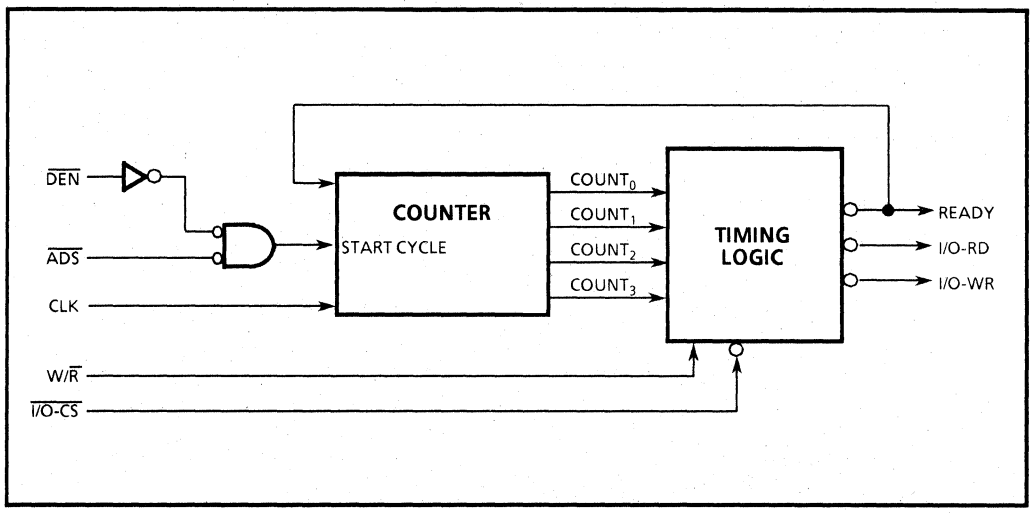


Figure 5-2: Timing Control Block Diagram

For many peripherals, the timing logic can be programmed to assert $\overline{\text{READY}}$ at the appropriate count for the selected device. Specific I/O chip select signals can be used to indicate how many clock cycles to wait before asserting $\overline{\text{READY}}$.

For some I/O peripherals, particularly bus masters, $\overline{\text{READY}}$ cannot be determined by counting clock cycles. For these I/O devices, $\overline{\text{READY}}$ can be supplied by the device and passed on to 80960KB processor.

The timing control block can assert the $\overline{\text{I/O-RD}}$ or $\overline{\text{I/O-WR}}$ signal for I/O devices based upon the clock count. The timing for these signals can be selected for the slowest device to simplify the logic circuit or can be customized for each individual peripheral device to maximize performance.

I/O INTERFACE DESIGN EXAMPLES

The general system interface shown in Figure 5-1 can be used to connect the 80960KB processor to many slave peripherals. The following list includes some common peripherals compatible with this interface:

- 8259A Programmable Interrupt Controller
- 8253, 8254 Programmable Interval Timer
- 8272 Floppy Disk Controller
- 82062, 82064 Fixed Disk Controller
- 82510, Asynchronous Serial Controller
- 8274, 82530 Multi-Protocol Serial Controller
- 8255 Programmable Peripheral Interface
- 8041, 8042 Universal Peripheral Interface

This section provides guidelines and design considerations for interfacing the 80960KB processor to different types of I/O configurations. Specifically, four design examples are examined. The 8259A and 82530 design examples show how to interface the 80960KB processor to a slave-type peripheral device. The 82586 design example shows how a 16-bit bus master reads and writes to the 80960KB processor's system memory. The 82786 design example shows how the 80960KB processor can read or write to graphics memory using a 16-bit data bus.

8259A Programmable Interrupt Controller

The 8259A Programmable Interrupt Controller is designed for use in interrupt-driven microcomputer systems, where it manages up to eight independent interrupts. The 8259A handles interrupt priority resolution and returns an 8-bit vector to the 80960KB processor during an interrupt-acknowledge cycle. Intel *Application Note AP-59* contains detailed information on configurations of the 8259A.

Interface

Figure 5-3 shows the connection of the 80960KB processor to a single 8259A Interrupt Controller. This circuit consists of the general system interface plus a bidirectional buffer. The example assumes that several interrupt requests occur at the same time so that priority resolution is needed.

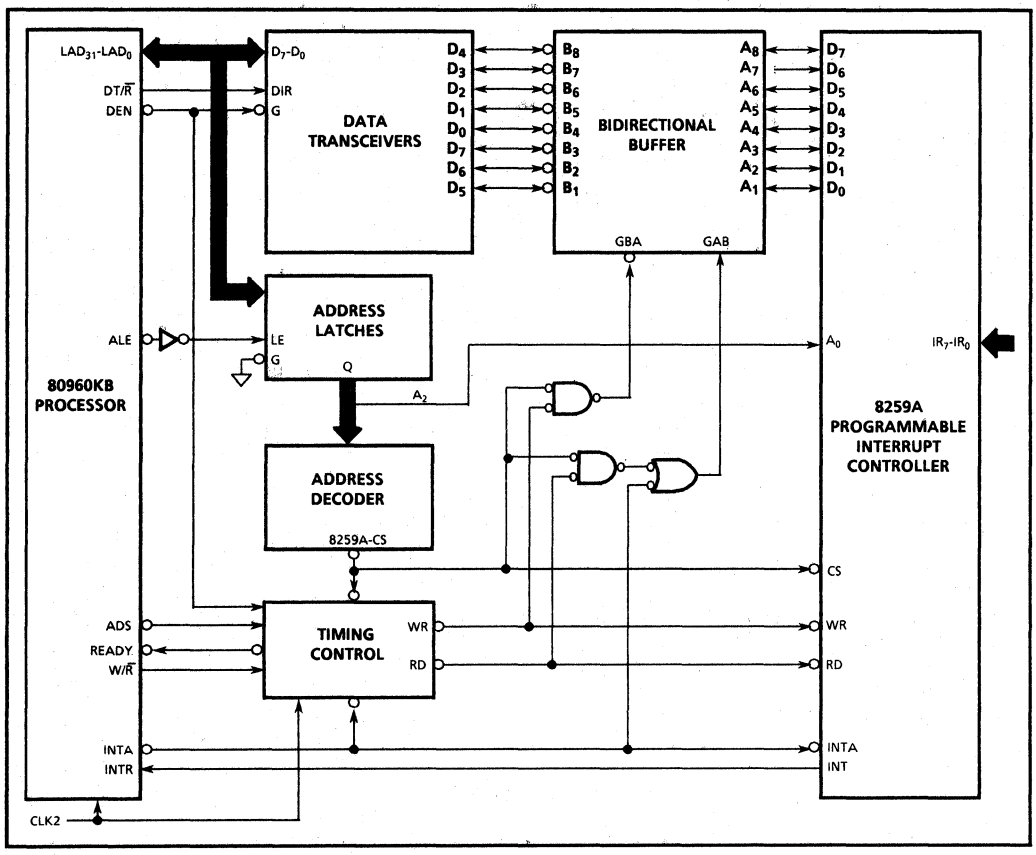


Figure 5-3: Block Diagram for 8259A Interface

The data lines from the 8259A are not directly aligned to the 80960KB processor because of the difference in priority resolution between the devices. Although both devices use an 8-bit interrupt vector, the 80960KB processor implicitly defines the priority by dividing the interrupt vector by eight. The 8259A defines the priority in the lower three bits of the interrupt vector. Furthermore, the highest priority vector of the 80960KB processor has a value of 31 in the upper five bits of the interrupt vector. Whereas, the highest priority interrupt of the 8259A has a value of 0 in the lower three bits of the interrupt vector.

To resolve the priority difference, the interrupt vector from the 8259A can be inverted and rotated left by three bits as shown by the data alignment between the 80960KB processor and 8259A in Figure 5-3. Rotating the data bits in this manner provides two advantages: the interrupt table for the 8259A can be located by contiguous addresses, and the upper two most significant bits of the interrupt vector remain free to group interrupt vectors if additional 8259As are needed.

Care must be exercised, however, when programming the registers of the 8259A. For example, assume that the second initialization command word (ICW2 register) of the 8259A requires a data byte value of 00011111_B. To transfer the correct information, the 80960KB

processor needs to write a data word with the value of 00000111_B because this word is rotated left three places and inverted.

Operation

The 8259A starts the interrupt cycle by generating an interrupt request (INT) to the 80960KB processor, which receives the signal at the INTR input pin. This assumes the Interrupt Control register of the 80960KB processor is set to accommodate an external interrupt controller.

When the 80960KB processor comes to a breakpoint in its execution, it asserts the $\overline{\text{INTA}}$ signal twice. The first $\overline{\text{INTA}}$ signal acknowledges the interrupt request and causes the 8259A to prioritize the interrupt requests it received up to this point. The $\overline{\text{INTA}}$, together with the $\overline{\text{8259A-CS}}$, are applied to the timing control logic to generate a $\overline{\text{READY}}$ signal.

The 80960KB processor automatically asserts the second $\overline{\text{INTA}}$ signal five clock cycles after the assertion of $\overline{\text{READY}}$. After the second assertion of $\overline{\text{INTA}}$, the 80960KB processor reads the interrupt vector from the 8259A.

The bidirectional buffer inverts and passes the 8-bit vector to the 80960KB processor with the appropriate lines rearranged. The output enable signal for the data buffer is controlled by $\overline{\text{INTA}}$ for this operation. After the data transfer is completed, the timing control circuit generates a second $\overline{\text{READY}}$ signal to terminate the interrupt acknowledge cycle.

The same circuitry can be used to read or write to the 8259A registers. In this case, the 80960KB processor selects the 8259A through a memory-mapped address. Local address line 2 (A_2) selects one of two internal registers of the 8259A. The I/O read or I/O write command is generated by the timing control circuit. The data passes through the bidirectional data buffer to or from the selected register of the 8259A.

The direction of data flow through the buffer is controlled by three logic gates shown in Figure 5-3. For an I/O write operation, the I/O write command and $\overline{\text{8259A-CS}}$ signal control the output enable signal of the bidirectional buffer. Similarly, for a read operation, the I/O read command and the $\overline{\text{8259A-CS}}$ signal control the output enable signal of the latch. After the data is transferred, the timing control circuit asserts $\overline{\text{READY}}$.

82530 Serial Communication Controller Example

The 82530 Serial Communication Controller is a dual-channel, multi-protocol controller with on-chip baud rate generators, digital phase locked loops, various data encoding/decodings, and extensive diagnostic capabilities. The 82530 is designed to interface with high-speed serial communications lines using a variety of communication protocols, including asynchronous, synchronous, and HDLC/SDLC protocols. The 82530 contains two independent full-duplex channels.

The general system interface circuit previously described can be used to connect the 80960KB processor to the 82530, as shown in Figure 5-4. The 82530 can send an interrupt request to the 80960KB processor as shown or it can send the interrupt request to an interrupt controller, which in turn sends it to the 80960KB processor. The 80960KB processor responds to the

interrupt request and issues an address. After the address is latched, the address lines are decoded to generate a chip-select ($\overline{82530\text{-}CS}$) signal to activate the 82530.

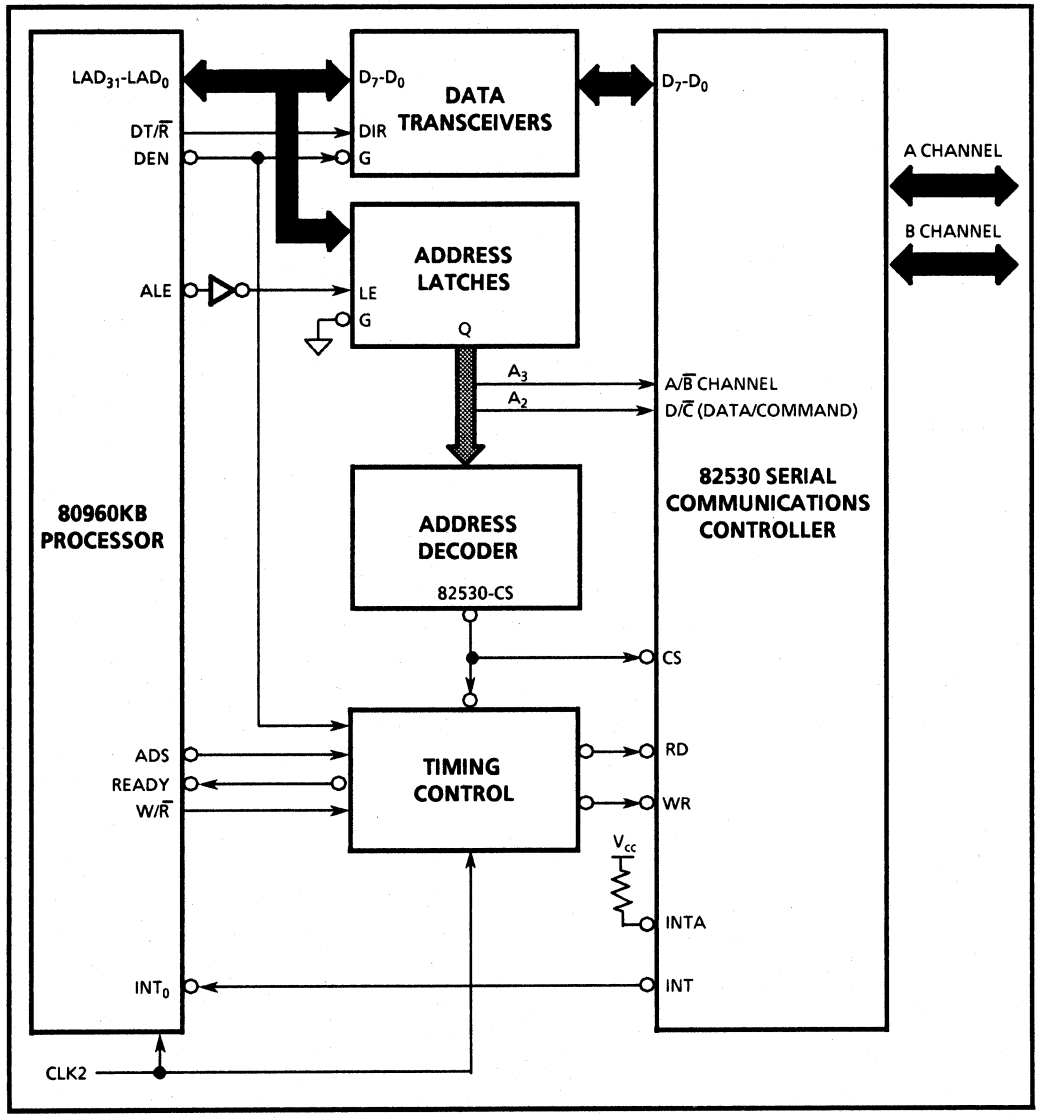


Figure 5-4: Block Diagram for 82530 Interface

The lower two address lines, A_2 and A_3, are used for channel selection and command/data selection. A_2 is connected to the Channel-A/Channel-B ($A/\overline{B}$) select input pin. This selects the channel that performs the serial read or write operations. A_3 is connected to the Data/Command ($D/\overline{C}$) select input pin. This signal defines the type of information transferred to or from the 82530 on the data lines (D_7 through D_0). A high level means data is transferred; a low level indicates a command.

The timing control circuit generates an I/O read or I/O write command based upon the W/$\overline{R}$ command from the 80960KB processor. When the data transfer is completed, the timing control circuit sends a $\overline{READY}$ signal to terminate the transaction.

The baud rate clocks can be programmed in several ways, including use of an external crystal.

82586 Local Area Network Coprocessor Example

The 82586 is an intelligent, high-performance communications controller designed to perform most tasks required for controlling access to a local area network (LAN), such as Ethernet or Starlan. In many applications, the 82586 is the communication manager for a station connected to a LAN controller. Such a station usually includes a host CPU, shared memory, a Serial Interface Unit, a transceiver, and LAN controller link, as shown in Figure 5-5. The 82586 performs all functions associated with data transfer between the shared memory and the LAN link, including:

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

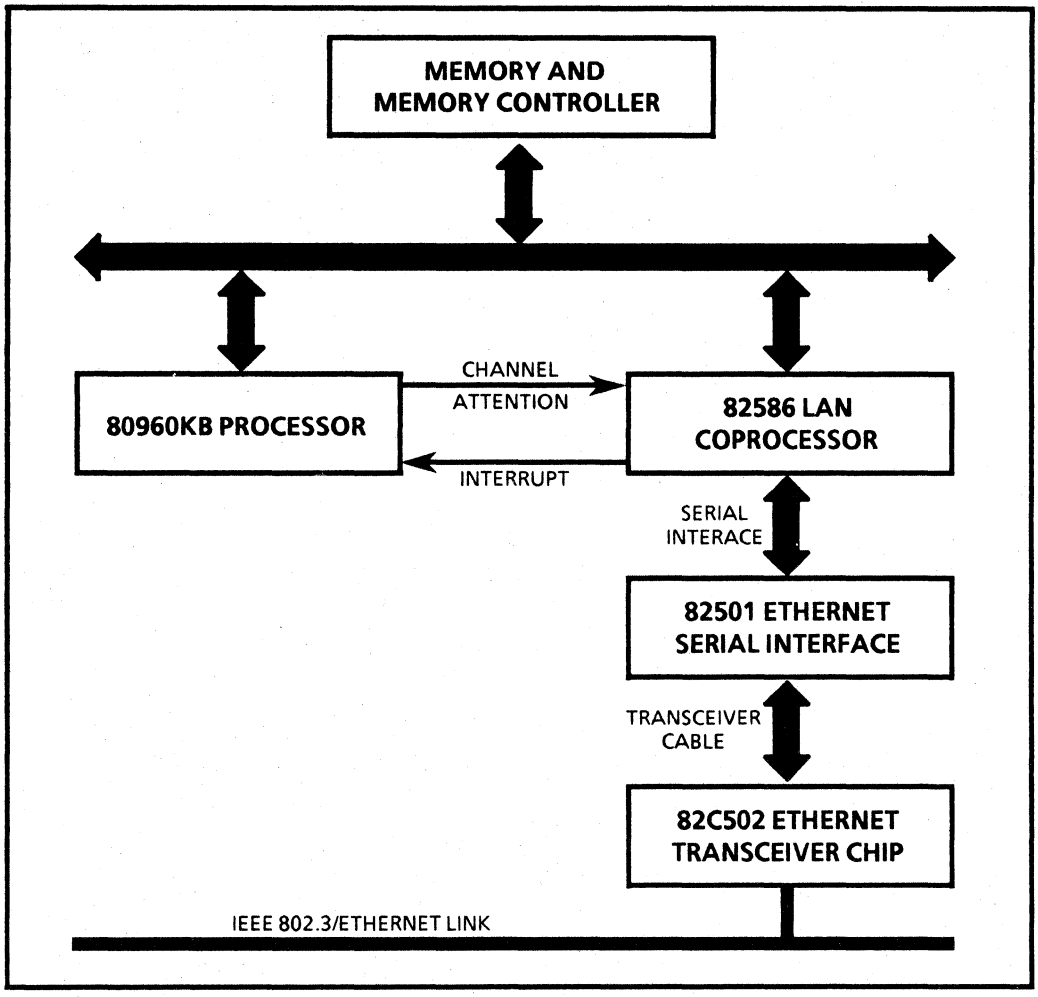


Figure 5-5: LAN Station

The 82586 has two interfaces: a 16-bit bus interface 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 Component User's Manual.*

Interface

There are several ways to design an interface between the 82586 and the 80960KB processor. The chosen design example shows how to interface the 82586 using a shared bus. In this example, the 82586 operates in minimum mode at one-half the processor clock frequency.

The primary function of the interface circuit is to allow the 82586 to read and write 16-bit data using the 32-bit L-bus. This is accomplished by adding the high-order address lines and translating the 16-bit data lines to the 32-bit data lines by using byte enable signals.

Figure 5-6 shows the 82586 interface circuit, which includes the DRAM controller (see the "DRAM Controller" section in Chapter 4 on page 4-11). This interface uses the general system interface circuit plus other logic units that specifically pertain to the 82586: the LAN data transceivers, the byte enable converter, and the LAN address latches. These logic blocks are highlighted by the shaded boxes.

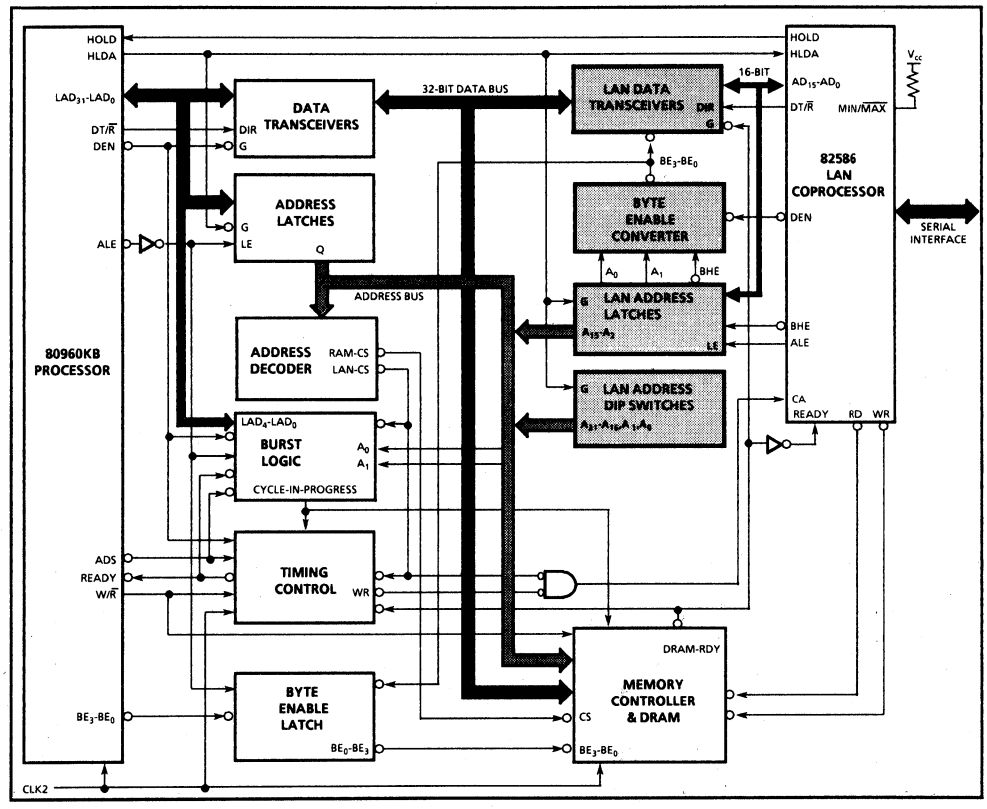


Figure 5-6: Block Diagram for LAN Controller Interface

The LAN data transceivers connect 16 data lines from the 82586 to both the upper and lower 16 bits of the L-bus. The data transfer is controlled by converting A_0, A_1, and the $\overline{BHE}$ to four byte enable signals as shown in Figure 5-7. A_1 selects between the upper and lower 16-bit data lines; A_0 selects the lower data byte for either the upper or lower 16-bit data lines; and the byte high enable signal ($\overline{BHE}$) selects the upper data byte for either the upper or lower 16-bit data lines. Data flows through the buffers when the appropriate byte enable signal is asserted. The direction of the data flow is controlled by the DT/$\overline{R}$ signal of the 82586.

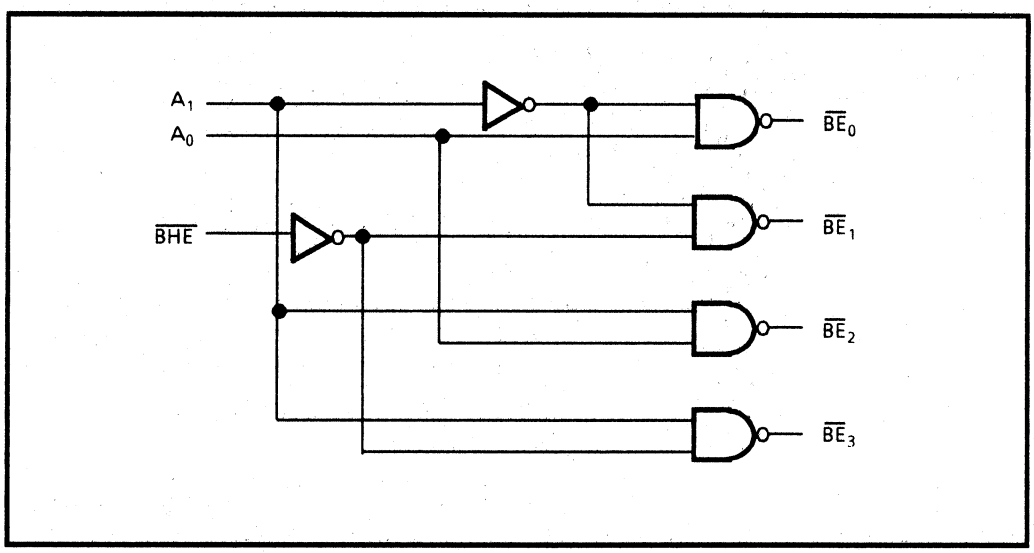


Figure 5-7: Byte Enable Generation Circuit

The LAN address latches are used to demultiplex AD_{15} through AD_0. The address lines and $\overline{BHE}$ are latched by the ALE signal from the 82586. The upper address lines (A_{31} through A_{16}) are generated by hardware programmable DIP switches.

The 82586 begins operation when the Channel Attention (CA) input signal is asserted. This signal is generated by gating the write command and 82586 chip select signal.

Operation

The interaction between the 82586 and the 80960KB processor is described below and is summarized in Figure 5-8.

- The 80960KB processor invokes the 82586 by supplying a memory-mapped address and a write command. The memory-mapped address results in a $\overline{82586\text{-}CS}$ signal, which is gated with a write command to produce the CA signal.
- The 82586 responds by generating a hold request and waits for HLDA.
- The 80960KB processor asserts HLDA, which enables the outputs of the LAN address latches and disables the outputs of the address latches next to the 80960KB processor. The HLDA signal also gives control of the L-bus to the 82586.
- After the 82586 takes control of the bus, it generates a 16-bit address (AD_{15} through AD_0), an ALE signal, and a $\overline{BHE}$ signal. The upper address lines are provided by the programmable DIP switches to produce an address on the L-bus.
- A_1 and A_0 (from the 82586), and $\overline{BHE}$ are decoded to generate four byte enable signals ($\overline{BE_3}$ through $\overline{BE_0}$). $\overline{DEN}$ enables the output of the byte enable converter.
- $DT/\overline{R}$ from the 82586 controls the direction of the data flow through the buffers.

- The read or write signal from the 82586 is applied to the DRAM controller.
- The 82586 accesses DRAM by using the DRAM controller.
- The $\overline{\text{DRAM-RDY}}$ is asserted by the DRAM controller. This action enables the output of the LAN data transceiver and terminates the 82586 memory cycle. The timing control logic passes the $\overline{\text{DRAM-RDY}}$ signal as the $\overline{\text{READY}}$ signal to the 82586.
- The 82586 deasserts HOLD and the 80960KB processor deasserts HLDA. The 80960KB processor regains control of bus.

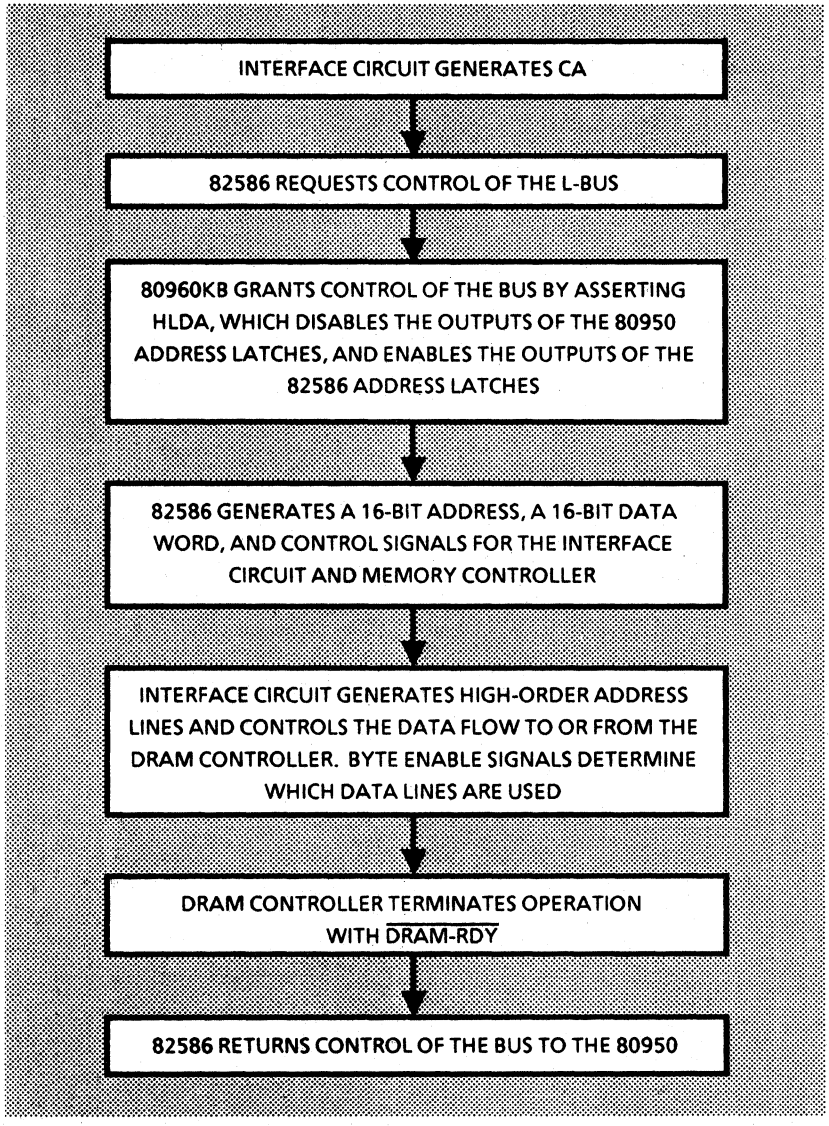


Figure 5-8: Operational Flow Diagram for 82586 Interface

82786 Graphics Coprocessor Example

The 82786 is a high performance graphics processor that provides high quality text and advanced display control. It provides full support for graphics primitives at up to 25 million pixels per second and bit-mapped text up to 25 thousand characters per second. This graphics

processor supports advanced features such as hardware windows, zooming, panning, and scrolling. Intel *Application Note AP-259* and *Application Note AP-270* contain detailed information on 82786.

When using the 82786, it may be necessary for the 80960KB processor to write to graphics memory. The interface design example illustrates how the 80960KB processor can transfer a 32-bit data word to the 16-bit data bus of the 82786.

Interface

There are several ways to design an interface between the 82786 and the 80960KB processor. In this example, the 80960KB processor reads or writes to graphics memory by accessing the 82786 through the interface logic circuit. This example assumes that the 82786 operates in the slave mode, and that the 80960KB processor does not perform burst transfers. The 80960KB processor only performs burst transfers for instructions that specify accesses for more than one word or for instruction fetches.

The interface circuit translates a 32-bit data bus to a 16-bit data bus by dividing the data lines into the upper and lower 16 bits and sequencing the data transmission. When the 80960KB processor writes to graphics memory, the bidirectional transceivers sequence the lower and the upper data bits of the L-bus to the 16-bit data bus of the 82786.

The process is reversed when the 80960KB processor reads from graphics memory. The bidirectional transceivers form a 32-bit data word by latching the first 16-bit data word on the lower data lines, routing the next 16 bits to the upper data lines, and then passing the 32-bit data word on the L-bus.

Figure 5-9 shows the details of the graphics controller interface circuit. This interface uses the general system interface circuit plus the following logic units: the bidirectional transceivers, the data buffer control, the data bus controller, and the address translator. These logic blocks are highlighted by the shaded boxes.

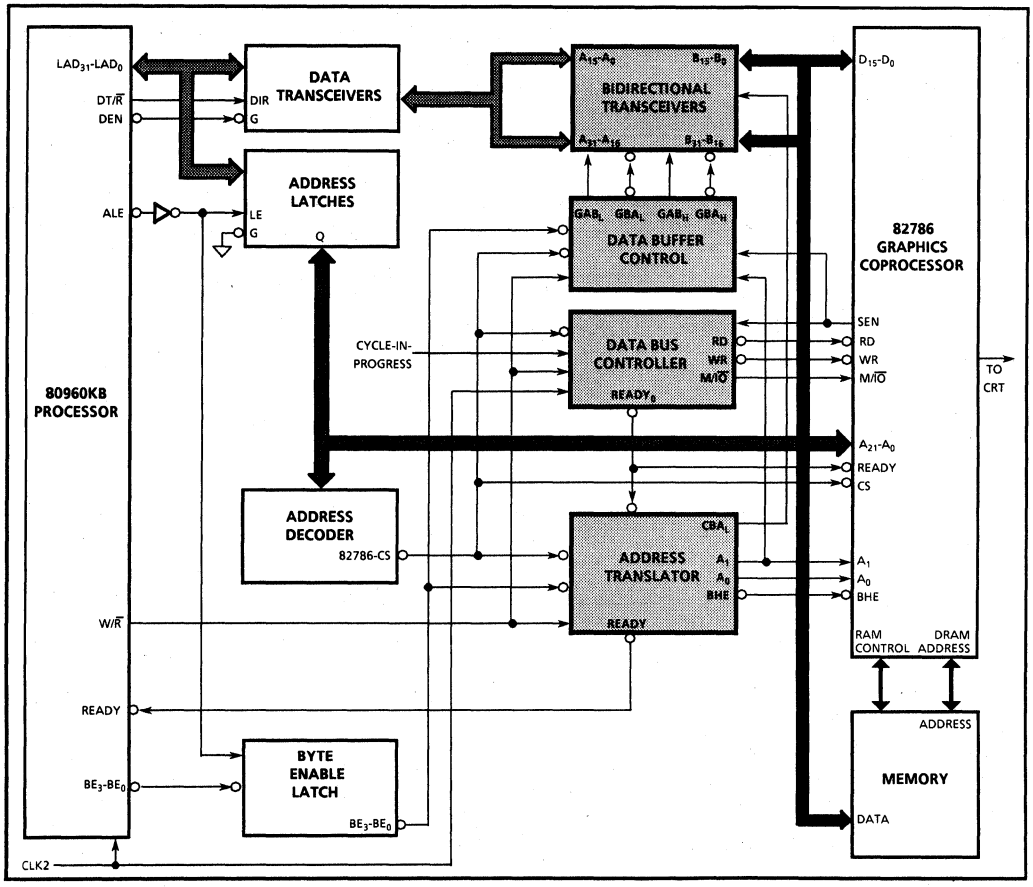


Figure 5-9: Block Diagram for 82786 Interface

The bidirectional transceivers pass data to (from) a 32-bit data bus from (to) a 16-bit data bus. Data is sequenced through the transceivers by the control signals generated by the data buffer control logic.

The data buffer control logic generates the signals that operate and sequence the bidirectional transceivers. The direction signal for data flow through the transceivers is derived from the W/$\overline{R}$ signal of the 80960KB processor. The data buffer control logic generates four output enable signals: GAB_L enables the outputs on the B side for the lower 16 bits; $\overline{GBA}_L$ enables the outputs on the A side for the lower 16 bits; GAB_H enables the outputs on the B side for the higher 16 bits; and $\overline{GBA}_H$ enables the outputs on the A side for the higher 16 bits. These output enable signals are derived from the byte enable signals and are asserted when the slave enable signal (SEN) is activated by the 82786.

The select lines for the bidirectional transceivers allow data to flow from either the latched data or the input pins. These lines, which are not shown, can be hardwired.

The data bus controller provides the read and write commands, memory or I/O signal (M/$\overline{IO}$), and a $\overline{READY}_0$ signal. This circuit generates two read or write commands for every 32-bit

data transfer to or from the 80960KB processor (one for each 16-bit data transfer). The data bus controller starts counting clock cycles when the $\overline{82786\text{-}CS}$ and CYCLE-IN-PROGRESS signals are asserted. At the proper time (based upon clock counts), it asserts the read/write command. The data bus controller produces $\overline{READY}_0$ after receiving the SEN signal from the 82786. $\overline{READY}_0$ resets the count, and another read/write command is generated.

The address translator performs four functions: it converts the four byte enable signals to A_0, A_1, and $\overline{BHE}$; it increments A_1 after receiving $\overline{READY}_0$ for the first 16-bit transfer; it generates the clock signal (CBA_L) that latches the first 16-bit data word in the bidirectional transceivers when the 80960KB processor performs a read operation; and it generates the $\overline{READY}$ signal for the CPU.

Not shown is the cycle detector circuit that generates the CYCLE-IN-PROGRESS signal. This signal can be generated by using the circuit similar to the one shown in Figure 5-2. The start of the cycle can be detected by gating the $\overline{ADS}$ and $\overline{DEN}$ signals. The end of the cycle can be indicated by $\overline{READY}$.

Operation

The interaction between the 82786 and the 80960KB processor is summarized in Figure 5-10. The operation is divided into two 16-bit data movements for both a read and write operation.

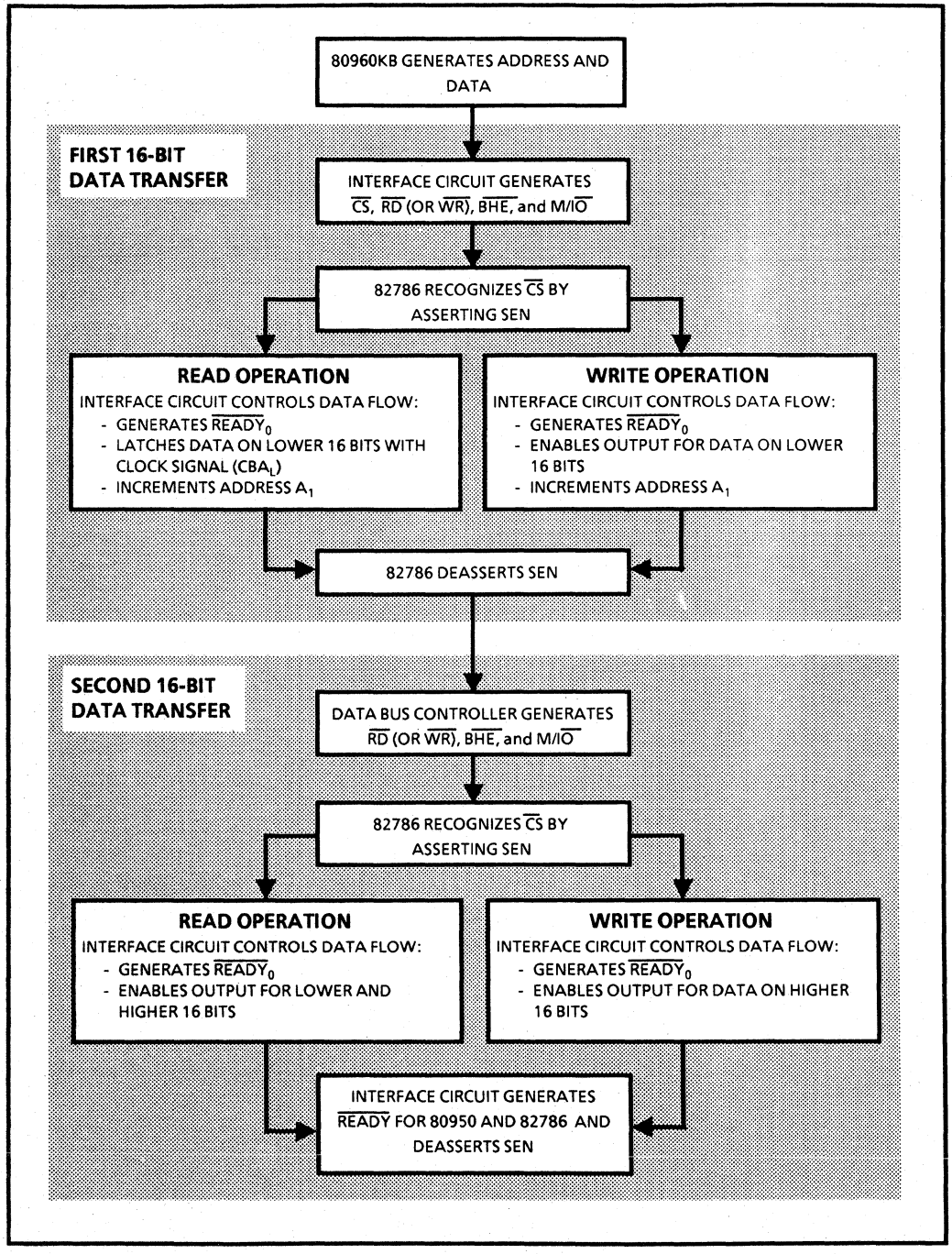


Figure 5-10: Operational Flow Diagram for 82786 Interface Circuit

The 80960KB processor generates a memory-mapped address and data for the desired graphics memory location. It accesses the 82786 by triggering the interface circuit to generate the chip select signal and several operational signals: the read ($\overline{RD}$) or write ($\overline{WR}$) command, $\overline{BHE}$, and the memory or I/O (M/$\overline{IO}$) signal. The 82786 begins the memory operation after it completes the current graphics processing activity. The 82786 acknowledges that it is performing a memory operation by asserting SEN.

After the 82786 asserts SEN, it begins a 16-bit memory read or write operation by translating the address inputs (A_{21} through A_0) to a multiplexed DRAM address, and generating the DRAM control signals. Note that A_1 and A_0 are derived from the byte enable signals.

For a read operation, the data bus controller uses SEN to generate the $\overline{READY}_0$ signal. The assertion of $\overline{READY}_0$ causes the address translator to increment A_1 and to generate CBA_L, which latches the lower 16 data bits on the B inputs of the bidirectional transceivers to the A side.

Similarly, for a write operation, the data bus controller uses SEN to generate the $\overline{READY}_0$ signal. The assertion of $\overline{READY}_0$ causes the address translator to increment A_1. The data buffer control uses SEN and the byte enable signals to produce GAB_L, which enable the outputs for the lower 16 data bits of the bidirectional transceivers.

The 82786 automatically deasserts SEN and the transfer of the first 16 data bits is complete. To transfer the second 16 data bits, the interface circuit requests another memory operation by generating $\overline{RD}$ (or $\overline{WR}$), $\overline{BHE}$, and M/$\overline{IO}$ ($\overline{CS}$ is already asserted). After it completes the current graphics processing activity, the 82786 begins the memory operation and asserts SEN.

For a read operation, the data bus controller uses SEN to generate the $\overline{READY}_0$ signal. The data buffer control uses SEN to assert $\overline{GBA}_H$ and $\overline{GBA}_L$, which enable the outputs for the higher and lower 16 data bits.

For a write operation, the data bus controller uses SEN to generate the $\overline{READY}_0$ signal. The data buffer control uses SEN and the byte enable signals to produce GAB_H, which enable the outputs for the higher 16 data bits of the bidirectional transceivers.

The address translator generates $\overline{READY}$ for the 80960KB processor from the second $\overline{READY}_0$ to terminate the data transfer to the graphics memory.

SUMMARY

The 80960KB processor supports 8-bit, 16-bit, and 32-bit I/O interfaces. A general system interface circuit can be designed that connects to many slave-type peripherals. This interface can be expanded to accommodate a bus master peripheral or a 32-bit to 16-bit data bus translator. These interfaces were illustrated by four design examples.

Index

INDEX

T

W

intel

LITERATURE SALES FORM (EUROPE)

NAME: _____

COMPANY: _____

ADDRESS: _____

_____

_____

PHONE NO.: _____

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

Subtotal _____

Your Local Sales Tax _____

Postage _____

Total _____

PAYMENT

Cheques should be made payable to your local Intel Sales Office.

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 for the attention of the LITERATURE CO-ORDINATOR and returned to your local Intel Sales Office.

UNITED STATES
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

JAPAN
Intel Japan K.K.
5-6 Tokodai, Tsukuba-shi
Ibaraki, 300-26

FRANCE
Intel Paris
1 Rue Edison, BP 303
78054 Saint-Quentin-en-Yvelines Cedex

UNITED KINGDOM
Intel Corporation (U.K.) Ltd.
Pipers Way
Swindon
Wiltshire, England SN3 1RJ

WEST GERMANY
Intel Semiconductor GmbH
Seidlstrasse 27
D-8000 Muenchen 2

HONG KONG
Intel Semiconductor Ltd.
1701-3 Connaught Centre
1 Connaught Road

CANADA
Intel Semiconductor of Canada, Ltd.
190 Attwell Drive, Suite 500
Rexdale, Ontario M9W 6H8

Printed in U.S.A/0388/15K/RRD SM
Embedded Controllers

ISBN 1-55512-033-4