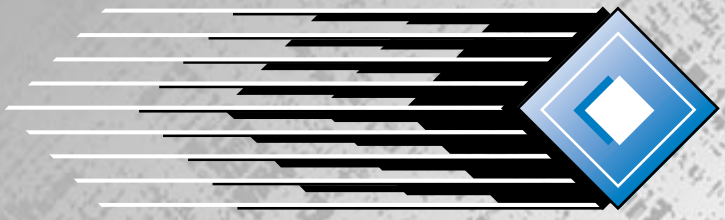
Intel386™ EX Embedded Microprocessor User's Manual

Intel386™ EXTB Embedded Microprocessor

Intel386™ EXTC Embedded Microprocessor



intel®

Intel386™ EX Embedded Microprocessor User's Manual

1996 Order Number 272485-002

Contact your local Intel sales office or your distributor to obtain the latest specifications before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature, may be obtained from:

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

or call 1-800-548-4725

CHAPTER 11
ASYNCHRONOUS SERIAL I/O UNIT

CHAPTER 12
DMA CONTROLLER

APPENDIX D
SYSTEM REGISTER QUICK REFERENCE

FIGURES

FIGURES

FIGURES

FIGURES

FIGURES

FIGURES

TABLES

TABLES

1

GUIDE TO THIS MANUAL

CHAPTER 1
GUIDE TO THIS MANUAL

This manual describes the Intel386™ EX Embedded Processor. It is intended for use by hardware designers familiar with the principles of microprocessors and with the Intel386 processor architecture.

This chapter is organized as follows:

1.1 MANUAL CONTENTS

This manual contains 18 chapters and 5 appendixes, a glossary, and an index. This section summarizes the contents of the remaining chapters and appendixes. The remainder of this chapter describes notational conventions and special terminology used throughout the manual and provides references to related documentation.

Chapter 2 — Architectural Overview — describes the device features and some potential applications.

Chapter 3 — Core Overview — describes the differences between this device and the Intel386™ SX processor core.

Chapter 4 — System Register Organization — describes the organization of the system registers, the I/O address space, address decoding, and addressing modes.

Chapter 5 — Device Configuration — explains how to configure the device for various applications.

Chapter 6 — Bus Interface Unit — describes the bus interface logic, bus states, bus cycles, and instruction pipelining.

Chapter 7 — System Management Mode — describes Intel's System Management Mode (SMM).

Chapter 8 — Clock and Power Management Unit — describes the clock generation circuitry, power management modes, and system reset logic.

Chapter 9 — Interrupt Control Unit — describes the interrupt sources and priority options and explains how to program the interrupt control unit.

Chapter 10 — Timer/Counter Unit — describes the timer/counters and their available count formats and operating modes.

Chapter 11 — Asynchronous Serial I/O (SIO) Unit — explains how to use the universal asynchronous receiver/transmitters (UARTs) to transmit and receive serial data.

Chapter 12 — DMA Controller — describes how the enhanced direct memory access controller allows internal and external devices to transfer data directly to and from the system and explains how bus control is arbitrated.

Chapter 13 — Synchronous Serial I/O (SSIO) Unit — explains how to transmit and receive data synchronously.

Chapter 14 — Chip-select Unit — explains how to use the chip-select channels to access various external memory and I/O devices.

Chapter 15 — Refresh Control Unit — describes how the refresh control unit generates periodic refresh requests and refresh addresses to simplify the interface to dynamic memory devices.

Chapter 16 — Input/Output Ports — describes the general-purpose I/O ports and explains how to configure each pin to serve either as an I/O pin or as a pin controlled by an internal peripheral.

Chapter 17 — Watchdog Timer Unit — explains how to use the watchdog timer unit as a software watchdog, bus monitor, or general-purpose timer.

Chapter 18 — JTAG Test-logic Unit — describes the independent test-logic unit and explains how to test the device logic and board-level connections.

Appendix A — Signal Descriptions — describes the device pins and signals and lists pin states after a system reset and during powerdown, idle, and hold.

Appendix B — Compatibility with PC/AT* Architecture — describes the ways in which the device is compatible with the standard PC/AT architecture and the ways in which it departs from the standard.

Appendix C — Example Code Header Files — contains the header files called by the code examples that are included in several chapters of this manual.

Appendix D — System Register Quick Reference — contains an alphabetical list of registers.

Appendix E — Instruction Set Summary — lists all instructions and their clock counts.

Glossary — defines terms with special meaning used throughout this manual.

Index — lists key topics with page number references.

1.2 NOTATIONAL CONVENTIONS

The following notations are used throughout this manual.

#	The pound symbol (#) appended to a signal name indicates that the signal is active low.
Variables	Variables are shown in italics. Variables must be replaced with correct values.
New Terms	New terms are shown in italics. See the Glossary for a brief definition of commonly used terms.
Instructions	Instruction mnemonics are shown in upper case. When you are programming, instructions are not case sensitive. You may use either upper or lower case.
Numbers	Hexadecimal numbers are represented by a string of hexadecimal digits followed by the character *H*. A zero prefix is added to numbers that begin with *A* through *F*. (For example, *FF* is shown as *0FFH*.) Decimal and binary numbers are represented by their customary notations. (That is, 255 is a decimal number and 1111 1111 is a binary number. In some cases, the letter *B* is added for clarity.)
Units of Measure	The following abbreviations are used to represent units of measure:

A	amps, amperes
Gbyte	gigabytes
Kbyte	kilobytes
KΩ	kilo-ohms
mA	milliamps, milliamperes
Mbyte	megabytes
MHz	megahertz
ms	milliseconds
mW	milliwatts
ns	nanoseconds
pF	picofarads
W	watts
V	volts
µA	microamps, microamperes
µF	microfarads
µs	microseconds
µW	microwatts

Register Bits When the text refers to more that one bit, the range may appear as two numbers separated by a colon (example: 7:0 or 15:0). The first bit shown (7 or 15 in the example) is the most-significant bit and the second bit shown (0) is the least-significant bit.

Register Names Register names are shown in upper case. If a register name contains a lowercase, italic character, it represents more than one register. For example, P*n*CFG represents three registers: P1CFG, P2CFG, and P3CFG.

Signal Names Signal names are shown in upper case. When several signals share a common name, an individual signal is represented by the signal name followed by a number, while the group is represented by the signal name followed by a variable (*n*). For example, the lower chip-select signals are named CS0#, CS1#, CS2#, and so on; they are collectively called CS*n*#. A pound symbol (#) appended to a signal name identifies an active-low signal. Port pins are represented by the port abbreviation, a period, and the pin number (e.g., P1.0, P1.1).

1.3 SPECIAL TERMINOLOGY

The following terms have special meanings in this manual.

Assert and Deassert The terms *assert* and *deassert* refer to the act of making a signal active and inactive, respectively. The active polarity (high/low) is defined by the signal name. Active-low signals are designated by a pound symbol (#) suffix; active-high signals have no suffix. To assert RD# is to drive it low; to assert HOLD is to drive it high; to deassert RD# is to drive it high; to deassert HOLD is to drive it low.

DOS I/O Address Integrated peripherals that are compatible with PC/AT system architecture can be mapped into DOS (or PC/AT) addresses 0H–03FFH. In this manual, the terms *DOS address* and *PC/AT address are synonymous.*

Expanded I/O Address All peripheral registers reside at I/O addresses 0F000H–0FFFFH. PC/AT-compatible integrated peripherals can also be mapped into DOS (or PC/AT) address space (0H–03FFH).

PC/AT Address Integrated peripherals that are compatible with PC/AT system architecture can be mapped into PC/AT (or DOS) addresses 0H–03FFH. In this manual, the terms *DOS address* and *PC/AT address are synonymous.*

Processor and CPU *Processor* refers to the Intel386 EX processor including the integrated peripherals. *CPU* refers to the processor core, which is based on the static Intel386 SX processor.

Reserved Bits

Reserved bits are not used in this device, but they may be used in future implementations. Follow these guidelines to ensure compatibility with future devices:

- Avoid any software dependence on the state of undefined register bits.
- Use a read-modify-write sequence to load registers.
- Mask undefined bits when testing the values of defined bits.
- Do not depend on the state of undefined bits when storing undefined bits to memory or to another register.
- Do not depend on the ability to retain information written to undefined bits.

Set and Clear

The terms *set* and *clear* refer to the value of a bit or the act of giving it a value. If a bit is *set*, its value is "1"; *setting* a bit gives it a "1" value. If a bit is *clear*, its value is "0"; *clearing* a bit gives it a "0" value.

1.4 RELATED DOCUMENTS

The following documents contain additional information that is useful in designing systems that incorporate the Intel386 EX processor. To order documents, please call Intel Literature Fulfillment (1-800-548-4725 in the U.S. and Canada; +44(0) 1793-431155 in Europe).

Document Name	Order Number
Intel386™ EX Embedded Microprocessor datasheet	272420
Intel386™ SX Microprocessor datasheet	240187
Intel386™ SX Microprocessor Programmer's Reference Manual	240331
Intel386™ SX Microprocessor Hardware Reference Manual	240332
Development Tools	272326
Buyer's Guide for the Intel386™ Embedded Processor Family	272520
Intel386™ EX Microprocessor Pin Multiplexing Map	272587
Packaging	240800

You may also want to refer to Standard 1149.1—1990, IEEE Standard Test Access Port and Boundary-Scan Architecture and its supplement, Standard 1149.1a—1993.

1.5 ELECTRONIC SUPPORT SYSTEMS

Intel's FaxBack* service and application BBS provide up-to-date technical information. Intel also maintains several forums on CompuServe and offers a variety of information on the World Wide Web. These systems are available 24 hours a day, 7 days a week, providing technical information whenever you need it.

1.5.1 FaxBack Service

FaxBack is an on-demand publishing system that sends documents to your fax machine. You can get product announcements, change notifications, product literature, device characteristics, design recommendations, and quality and reliability information from FaxBack 24 hours a day, 7 days a week.

1-800-525-3019 (US or Canada)

+44-1793-432509 (Europe)

+65-256-5350 (Singapore)

+852-2-844-4448 (Hong Kong)

+886-2-514-0815 (Taiwan)

+822-767-2594 (Korea)

+61-2-975-3922 (Australia)

1-503-264-6835 (Worldwide)

Think of the FaxBack service as a library of technical documents that you can access with your phone. Just dial the telephone number and respond to the system prompts. After you select a document, the system sends a copy to your fax machine.

Each document has an order number and is listed in a subject catalog. The first time you use FaxBack, you should order the appropriate subject catalogs to get a complete list of document order numbers. Catalogs are updated twice monthly. In addition, daily update catalogs list the title, status, and order number of each document that has been added, revised, or deleted during the past eight weeks. To receive the update for a subject catalog, enter the subject catalog number followed by a zero. For example, for the complete microcontroller and flash catalog, request document number 2; for the daily update to the microcontroller and flash catalog, request document number 20.

The following catalogs and information are available at the time of publication:

1. *Solutions OEM* subscription form
2. Microcontroller and flash catalog
3. Development tools catalog
4. Systems catalog
5. Multimedia catalog
6. Multibus and iRMX® software catalog and BBS file listings

7. Microprocessor, PCI, and peripheral catalog
8. Quality and reliability and change notification catalog
9. iAL (Intel Architecture Labs) technology catalog

1.5.2 Bulletin Board System (BBS)

The bulletin board system (BBS) lets you download files to your computer. The application BBS has the latest *Ap*BUILDER software, hypertext manuals and datasheets, software drivers, firmware upgrades, code examples, application notes and utilities, and quality and reliability data.

The system supports 1200- through 19200-baud modems. Typical modem settings are 14400 baud, no parity, 8 data bits, and 1 stop bit (14400, N, 8, 1).

To access the BBS, use a terminal program to dial the telephone number given below for your area; once you are connected, respond to the system prompts. During your first session, enter your name and location. The system operator will set up your access account within 24 hours. At that time, you can access the files on the BBS.

503-264-7999	U.S., Canada, Japan, Asia Pacific (up to 19.2 Kbaud)
44(0)1793-432955	Europe

NOTE

If you have problems accessing the BBS, use these settings for your modem: 2400, N, 8, 1. Refer to your terminal software documentation for instructions on changing these settings.

1.5.3 CompuServe Forums

The CompuServe forums provide a means for you to gather information, share discoveries, and debate issues. Type “go intel” for access. For information about CompuServe access and service fees, call CompuServe at 1-800-848-8199 (U.S.) or 614-529-1340 (outside the U.S.).

1.5.4 World Wide Web

We offer a variety of information through the World Wide Web (http://www.intel.com/). Select “Embedded Design Products” from the Intel home page.

1.6 TECHNICAL SUPPORT

In the U.S. and Canada, technical support representatives are available to answer your questions between 5 a.m. and 5 p.m. PST. You can also fax your questions to us. (Please include your voice telephone number and indicate whether you prefer a response by phone or by fax). Outside the U.S. and Canada, please contact your local distributor.

1-800-628-8686	U.S. and Canada
916-356-7599	U.S. and Canada
916-356-6100 (fax)	U.S. and Canada

1.7 PRODUCT LITERATURE

You can order product literature from the following Intel literature centers.

1-800-548-4725	U.S. and Canada
708-296-9333	U.S. (from overseas)
44(0)1793-431155	Europe (U.K.)
44(0)1793-421333	Germany
44(0)1793-421777	France
81(0)120-47-88-32	Japan (fax only)

2

ARCHITECTURAL OVERVIEW

CHAPTER 2
ARCHITECTURAL OVERVIEW

The Intel386™ EX embedded processor (Figure 2-1) is based on the static Intel386 SX processor. This highly integrated device retains those personal computer functions that are useful in embedded applications and integrates peripherals that are typically needed in embedded systems. The Intel386 EX processor provides a PC-compatible development platform in a device that is optimized for embedded applications. Its integrated peripherals and power management options make the Intel386 EX processor ideal for portable systems.

The integrated peripherals of the Intel386 EX processor are compatible with the standard desktop PC. This allows existing PC software, including most of the industry's leading desktop and embedded operating systems, to be easily implemented on an Intel386 EX processor-based platform. Using PC-compatible peripherals also allows for the development and debugging of application software on a standard PC platform.

Typical applications using the Intel386 EX processor include automated manufacturing equipment, cellular telephones, telecommunications equipment, fax machines, hand-held data loggers, high-precision industrial flow controllers, interactive television, medical equipment, modems, and smart copiers.

This chapter is organized as follows:

2.1 Intel386 EX EMBEDDED PROCESSOR CORE

The Intel386 EX processor contains a modular, fully static Intel386 CX central processing unit (CPU). The Intel386 CX processor is an enhanced Intel386 SX processor with the addition of System Management Mode (SMM) and two additional address lines. The Intel386 EX processor has a 16-bit data bus and a 26-bit address bus, supporting up to 64 Mbytes of memory address space and 64 Kbytes of I/O address space. The performance of the Intel386 EX processor closely reflects the Intel386 SX CPU performance at the same speeds.

Chapter 3, "CORE OVERVIEW" describes differences between the Intel386 EX processor core and the Intel386 SX processor. Please refer to the *Intel386™ SX Microprocessor Programmer's Reference Manual* (order number 240331) for applications and system programming information; descriptions of protected, real, and virtual-8086 modes; and details on the instruction set.

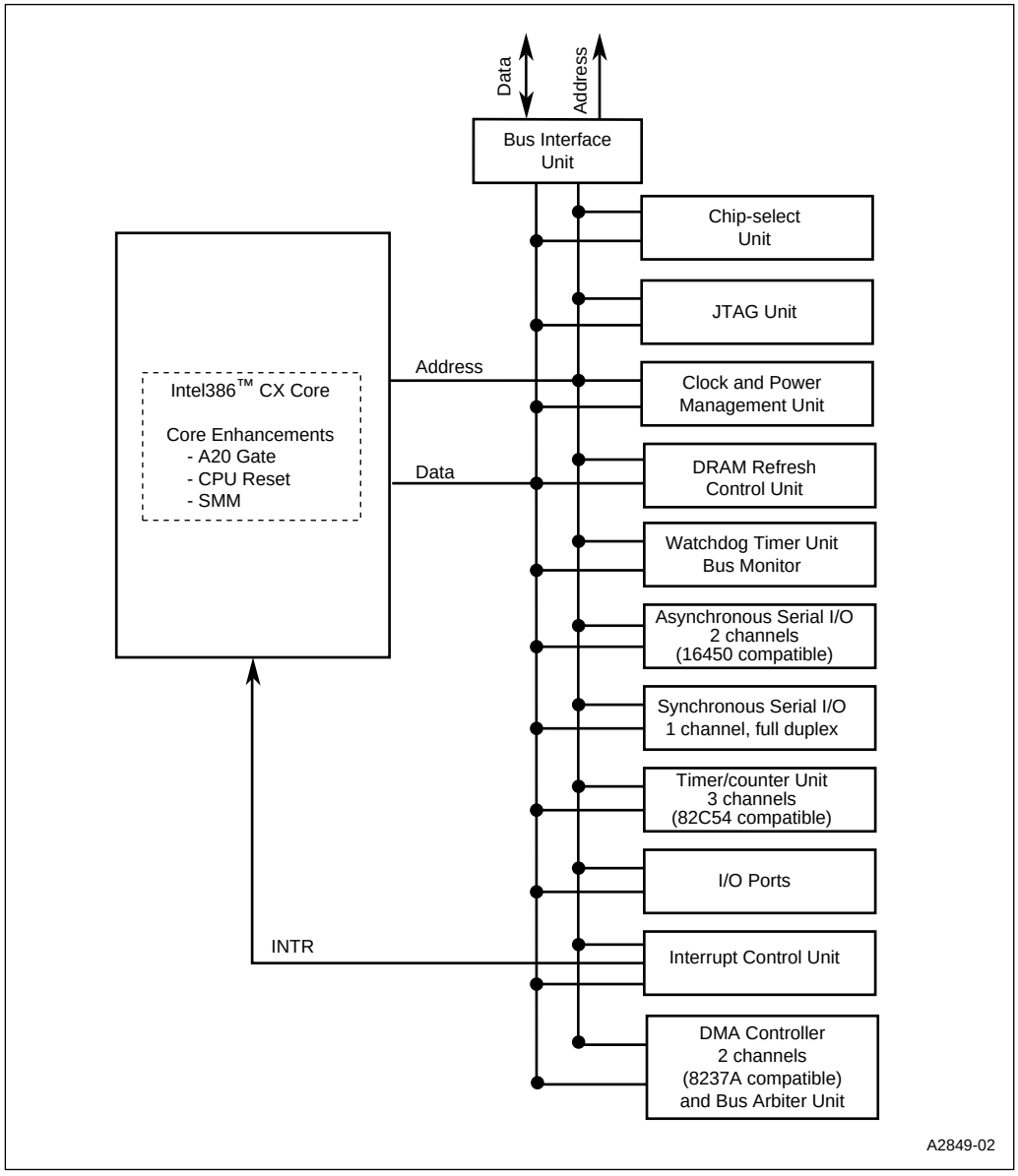


Figure 2-1. Intel386™ EX Embedded Processor Block Diagram

2.2 INTEGRATED PERIPHERALS

The Intel386 EX processor integrates both PC-compatible peripherals (Table 2-1) and peripherals that are specific to embedded applications (Table 2-2).

Table 2-1. PC-compatible Peripherals

Name	Description
Interrupt Control Unit (ICU)	Consists of two 82C59A programmable interrupt controllers (PICs) configured as master and slave. You may cascade up to six external 82C59A PICs to expand the external interrupt lines to 52. Refer to Chapter 9, “INTERRUPT CONTROL UNIT.”
Timer/counter Unit (TCU)	Provides three independent 16-bit down counters. The programmable TCU is functionally equivalent to three 82C54 counter/timers with enhancements to allow remapping of peripheral addresses and interrupt assignments. Refer to Chapter 10, “TIMER/COUNTER UNIT.”
Asynchronous Serial I/O (SIO) Unit	Features two independent universal asynchronous receiver and transmitter (UART) units which are functionally equivalent to National Semiconductor’s NS16450. Each channel contains a baud-rate generator, transmitter, receiver, and modem control unit. Receive and transmit interrupt signals can be connected to the ICU controller and DMA controller. Refer to Chapter 11, “ASYNCHRONOUS SERIAL I/O UNIT.”
Direct Memory Access (DMA) Controller	Transfers internal or external data between any combination of memory and I/O devices for the entire 26-bit address bus. The two independent channels operate in 16- or 8-bit bus mode. Buffer chaining allows data to be transferred into noncontiguous memory buffers. The DMA channels can be tied to any of the serial devices to support high data rates, minimizing processor interruptions. Provides a special two-cycle mode that uses only one channel for memory-to-memory transfers. Bus arbitration logic resolves priority conflicts between the DMA channels, the refresh control unit, and an external bus master. SIO and SSIO interrupts can be connected to DMA for high-speed transfers. Backward compatible with 8237A. Refer to Chapter 12, “DMA CONTROLLER.”

Table 2-2. Embedded Application-specific Peripherals

Name	Description
System Management Mode (SMM)	The Intel386 EX processor provides a mechanism for system management with a combination of hardware and CPU microcode enhancements. An externally generated system management interrupt (SMI#) allows the execution of system-wide routines that are independent and transparent to the operating system. The system management mode (SMM) architectural extensions to the Intel386 CPU are described in Chapter 7, "SYSTEM MANAGEMENT MODE."
Clock and Power Management Unit	An external clock source provides the input frequency. The clock and power management unit generates separate internal clock signals for core and peripherals (half the input frequency), divides the internal clock by two for baud clock inputs to the SIO and SSIO, and divides the internal clock by a programmable divisor to provide a prescaled clock signal (various frequencies) for the TCU and SSIO. Power management provides idle and powerdown modes (idle stops the CPU clock but leaves the peripheral clocks running; powerdown stops both CPU and peripheral clocks). An external clockout signal is also provided. Refer to Chapter 8, "CLOCK AND POWER MANAGEMENT UNIT."
Synchronous Serial I/O (SSIO) unit	Provides simultaneous, bidirectional high speed serial I/O. Consists of a transmit channel, a receive channel, and a baud rate generator. Built-in protocols are not included, because these can be emulated using the CPU. SSIO interrupts can be connected to the DMA unit for high-speed transfers. Refer to Chapter 13, "SYNCHRONOUS SERIAL I/O UNIT."
Chip-select Unit (CSU)	Programmable, eight-channel CSU allows direct access to up to eight devices. Each channel can operate in 16- or 8-bit bus mode and can generate up to 31 wait states. The CSU can interface with the fastest memory or the slowest peripheral device. The minimum address block for memory address-configured channels is 2 Kbytes. The size of these address blocks can be increased by powers of 2 Kbytes for memory addresses and by multiples of 2 bytes for I/O addresses. Supports SMM memory addressing and provides ready generation and programmable wait states. Refer to Chapter 14, "CHIP-SELECT UNIT."
Refresh Control Unit (RCU)	Provides a means to generate periodic refresh requests and refresh addresses. Consists of a programmable interval timer unit, a control unit, and an address generation unit. Bus arbitration logic ensures that refresh requests have the highest priority. The refresh control unit (RCU) is provided for applications that use DRAMs with a simple EPLD-based DRAM controller or PSRAMs that do not need a separate controller. Refer to Chapter 15, "REFRESH CONTROL UNIT."
Parallel I/O Ports	Three I/O ports facilitate data transfer between the processor and surrounding system circuitry. The Intel386 EX processor is unique in that several functions are multiplexed with each other or with I/O ports. This ensures maximum use of available pins and maintains a small package. Each multiplexed pin is individually programmable for peripheral or I/O function. Refer to Chapter 16, "INPUT/OUTPUT PORTS."
Watchdog Timer (WDT) Unit	When enabled, the WDT functions as a general purpose 32-bit timer, a software timer, or a bus monitor. Refer to Chapter 17, "WATCHDOG TIMER UNIT."
JTAG Test-logic Unit	The test-logic unit simplifies board-level testing. Consists of a test access port and a boundary-scan register. Fully compliant with Standard 1149.1–1990, *IEEE Standard Test Access Port and Boundary-Scan Architecture* and its supplement, Standard 1149.1a–1993. Refer to Chapter 18, "JTAG TEST-LOGIC UNIT."

3

CORE OVERVIEW

CHAPTER 3
CORE OVERVIEW

The Intel386™ EX processor core is based upon the Intel386 CX processor, which is an enhanced version of the Intel386 SX processor. This chapter describes the Intel386 CX processor enhancements over the Intel386 SX processor, internal architecture of the Intel386 CX processor, and the core interface on the Intel386 EX processor.

This chapter is organized as follows:

3.1 Intel386 CX PROCESSOR ENHANCEMENTS

The Intel386 CX processor, based on the Intel386 SX processor, adds system management mode and two additional address lines for a total of 26 address lines.

3.1.1 System Management Mode

The Intel386 CX processor core provides a mechanism for system management with a combination of hardware and CPU microcode enhancements. An externally generated System Management Interrupt (SMI#) allows the execution of system wide routines which are independent and transparent to the operating system. The System Management Mode (SMM) architecture extensions to the Intel386 SX processor consist of the following elements:

- Interrupt input pin (SMI#) to invoke SMM
- One output pin to identify execution state (SMIACT#)
- One new instruction (RSM, executable only from SMM) to exit SMM
- SMM also added one to four execution clocks to the following instructions: IN, INS, REP INS, OUT, REP OUT, POPA, HALT, MOV CR0, and SRC. INTR and NMI also need an additional two clocks for interrupt latency. These cycles were added due to the microcode modification for the SMM implementation. Refer to Appendix E for the exact execution times. Otherwise, 100% of the Intel386 SX processor instructions execute on the Intel386 CX processor core.

Please refer to Chapter 7 for more details on System Management Mode.

3.1.2 Additional Address Lines

Two additional address lines were added to the Intel386 CX processor core for a total of 26. This expands the physical address space from 16 Mbytes to 64 Mbytes.

3.2 Intel386 CX PROCESSOR INTERNAL ARCHITECTURE

The internal architecture of the Intel386 CX processor consists of functional units that operate in parallel. Fetching, decoding, execution, memory management and bus accesses for several instructions are performed simultaneously. This parallel operation is called *pipelined instruction* processing. With pipelining, each instruction is performed in stages, and the processing of several instructions at different stages may overlap, as shown in Figure 3-1. The pipelined processing of the Intel386 CX processor results in higher performance and enhanced throughput rate over non-pipelined processors.

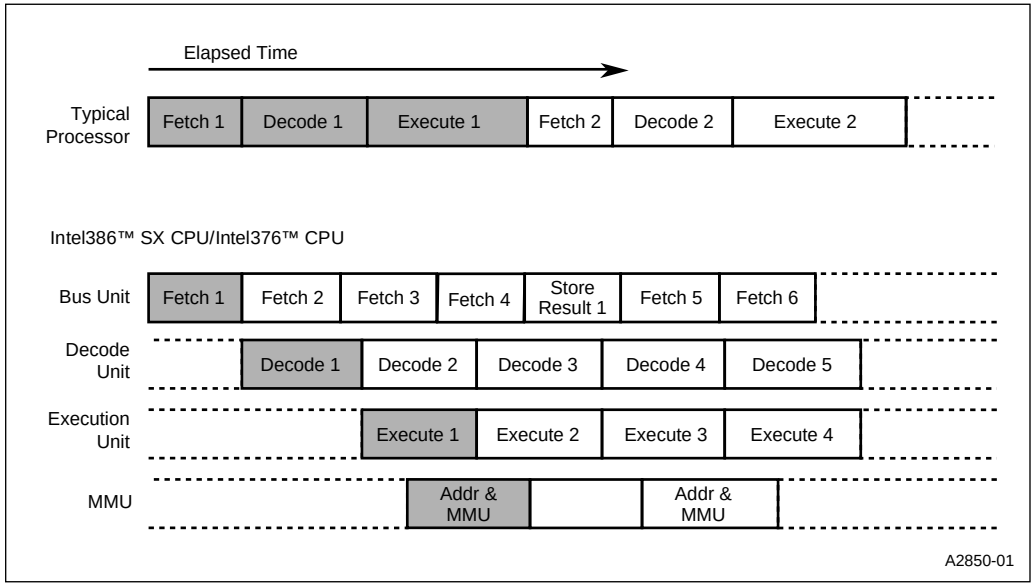


Figure 3-1. Instruction Pipelining

Figure 3-2 shows the internal architecture of the Intel386 CX processor.

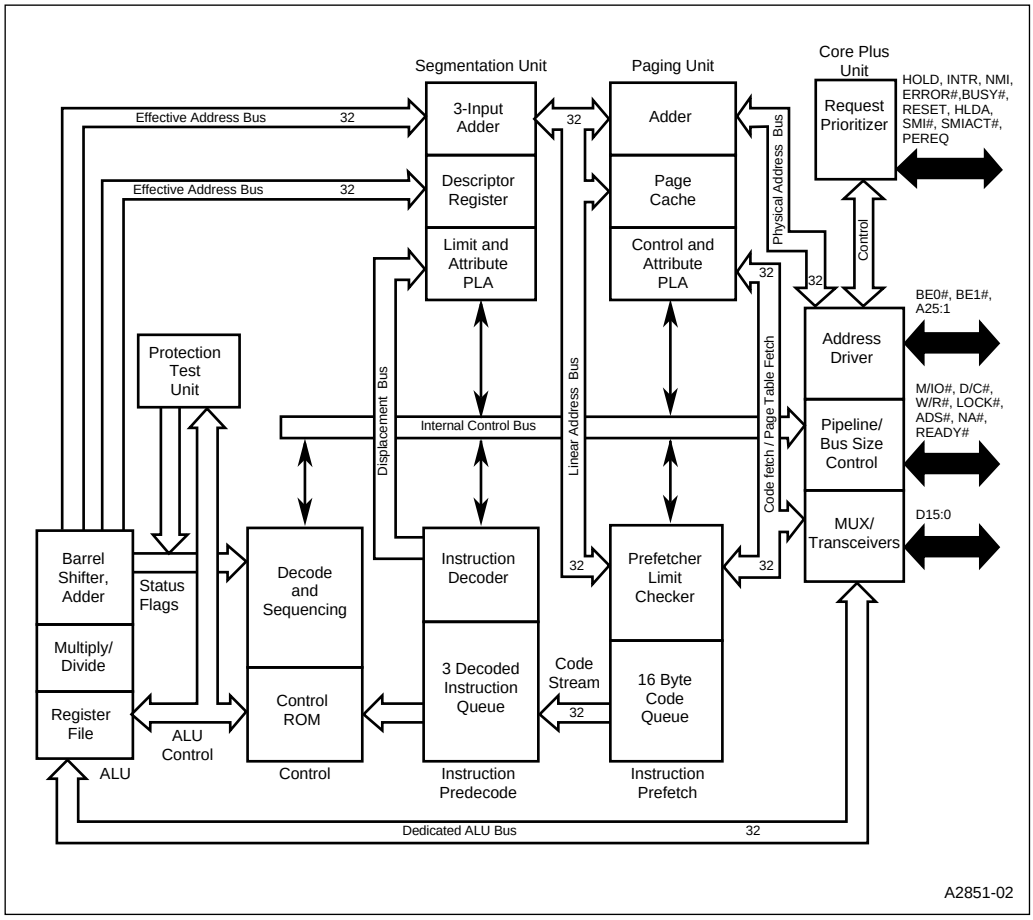


Figure 3-2. The Intel386™ CX Processor Internal Block Diagram

The six functional units of the Intel386 CX processor are:

- Core Bus Unit
- Instruction Prefetch Unit
- Instruction Decode Unit
- Execution Unit
- Segmentation Unit
- Paging Unit

3.2.1 Core Bus Unit

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

3.2.2 Instruction Prefetch Unit

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

Instruction prefetches are given a lower priority than data transfers; assuming zero wait state memory access, prefetch activity never delays execution. On the other hand, when there is no data transfer requested, prefetching uses bus cycles that would otherwise be idle.

3.2.3 Instruction Decode Unit

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

3.2.4 Execution Unit

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

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

To speed the execution of memory reference instructions, the Execution Unit partially overlaps the execution of any memory reference instruction with the previous instruction.

3.2.5 Segmentation Unit

The Segmentation Unit translates logical addresses into linear addresses at the request of the Execution Unit. The on-chip Segment Descriptor Cache stores the currently used segment descriptors to speed this translation. At the same time it performs the translation, the Segmentation Unit checks for bus-cycle segmentation violations. (These checks are separate from the static segmentation violation checks performed by the Protection Test Unit.) The translated linear address is truncated to a 24-bit physical address.

3.2.6 Paging Unit

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

3.3 CORE Intel386 EX PROCESSOR INTERFACE

The Intel386 EX processor peripherals are connected to the Intel386 CX processor core through an internal Bus Interface Unit (BIU). The BIU controls internal peripheral accesses and external memory and I/O accesses. Because it has the BIU between the Intel386 CX processor core and the external bus, the Intel386 EX processor bus timings are not identical to those of the Intel386 CX processor or Intel386 SX processor.

The Intel386 CX processor numeric coprocessor interface is maintained and brought out to the Intel386 EX processor pins. The same I/O addresses used on the Intel386 SX processor are used on the Intel386 EX processor, even though there are more address lines. The A23 line is high for coprocessor cycles. Refer to "Interface To Intel387™ SX Math Coprocessor" on page 6-38 for more details.

4

SYSTEM REGISTER ORGANIZATION

CHAPTER 4
SYSTEM REGISTER ORGANIZATION

This chapter provides an overview of the system registers incorporated in the Intel386™ EX processor, focusing on register organization from an address architecture viewpoint. The chapters that cover the individual peripherals describe the registers in detail.

This chapter is organized as follows:

4.1 OVERVIEW

The Intel386 EX processor has register resources in the following categories:

- Intel386 processor core architecture registers:
  - — General purpose registers
  - — Segment registers
  - — Instruction pointer and flags
  - — Control registers
  - — System address registers (protected mode)
  - — Debug registers
  - — Test registers
- Intel386 EX processor peripheral registers:
  - — Configuration space control registers
  - — Interrupt control unit registers
  - — Timer/counter unit registers
  - — DMA unit registers (8237A-compatible and enhanced function registers)
  - — Asynchronous serial I/O (SIO) registers
  - — Clock generation selector registers

— Power management control registers

— Chip-select unit control registers

— Refresh control unit registers

— Watchdog timer control registers

— Synchronous serial I/O control registers

— Parallel I/O port control registers

4.1.1 Intel386 Processor Core Architecture Registers

These registers are a superset of the 8086 and 80286 processor registers. All 16-bit 8086 and 80286 registers are contained within the 32-bit Intel386 processor core registers. A detailed description of the Intel386 processor architecture base registers can be found in the *Intel386™ SX Microprocessor Programmer's Reference Manual* (order number 240331).

4.1.2 Intel386 EX Processor Peripheral Registers

The Intel386 EX processor contains some peripherals that are common and compatible with the PC/AT* system architecture and others that are useful for embedded applications. The peripheral registers control access to these peripherals and enable you to configure on-chip system resources such as timer/counters, power management, chip selects, and watchdog timer.

All peripheral registers reside physically in the *expanded I/O address space* (addresses 0F000H–0FFFFH). Peripherals that are compatible with PC/AT system architecture can also be mapped into *DOS I/O address space* (addresses 0H–03FFH, 10-bit decode). The following rules apply for accessing peripheral registers after a system reset:

- Registers within the DOS I/O address space are accessible.
- Registers within the expanded I/O address space are accessible only after the expanded I/O address space is enabled.

4.2 I/O ADDRESS SPACE FOR PC/AT SYSTEMS

The Intel386 EX processor's I/O address space is 64 Kbytes. On PC/AT platforms, the DOS operating system and applications assume that only 1 Kbyte of the total 64-Kbyte I/O address space is used. The first 256 bytes (addresses 00000H–00FFH) are reserved for platform (motherboard) I/O resources such as the interrupt and DMA controllers, and the remaining 768 bytes (addresses 0100H–03FFH) are available for "general" I/O peripheral card resources. Since only 1 Kbyte of the address space is supported, add-on I/O peripheral cards typically decode only the lower 10 address lines. Because the upper address lines are not decoded, the 256 platform address locations and the 768 bus address locations are repeated 64 times (on 1-Kbyte boundaries), covering the entire 64-Kbyte address space. (See Figure 4-1.)

Generally, add-on I/O peripheral cards do not use the I/O addresses reserved for the platform resources. Software running on the platform can use any of the 64 repetitions of the 256 address locations reserved for accessing platform resources.

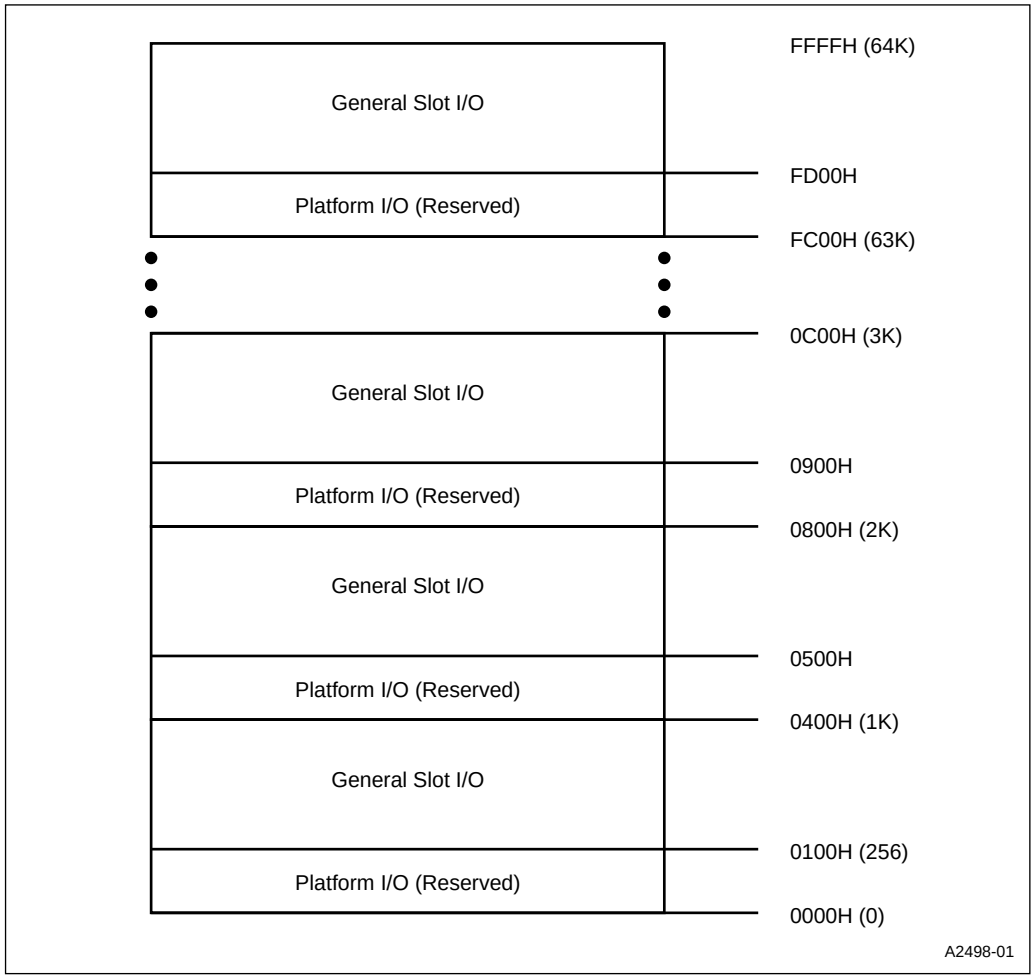


Figure 4-1. PC/AT I/O Address Space (10-bit Decode)

4.3 EXPANDED I/O ADDRESS SPACE

The Intel386 EX processor's I/O address scheme is similar to that of the Extended Industry Standard Architecture (EISA) bus and the Enhanced - Industry Standard Architecture (E-ISA) bus. Both standards maintain backward software compatibility with the ISA architecture. The ISA Platform I/O (0-100H) is accessed with a 16-bit address decode and is located in the first 256 I/O locations. The General Slot I/O that is typically used by add-in boards is repeated throughout the 64 Kbyte I/O address range due to their 10-bit only decode. This allows 63 of the 64 repetitions of the first 256 address locations of every 1 Kbyte block to be allocated to specific slots. Each slot is 4 Kbyte in size, allowing for a total of 16 slots. The partitioning is such that four groups of 256 address locations are assigned to each slot, for a total of 1024 specific address locations per slot.

(See Figure 4-2.) Thus, each slot has 1 Kbyte addresses (in four 256-byte segments) that can potentially contain extended peripheral registers.

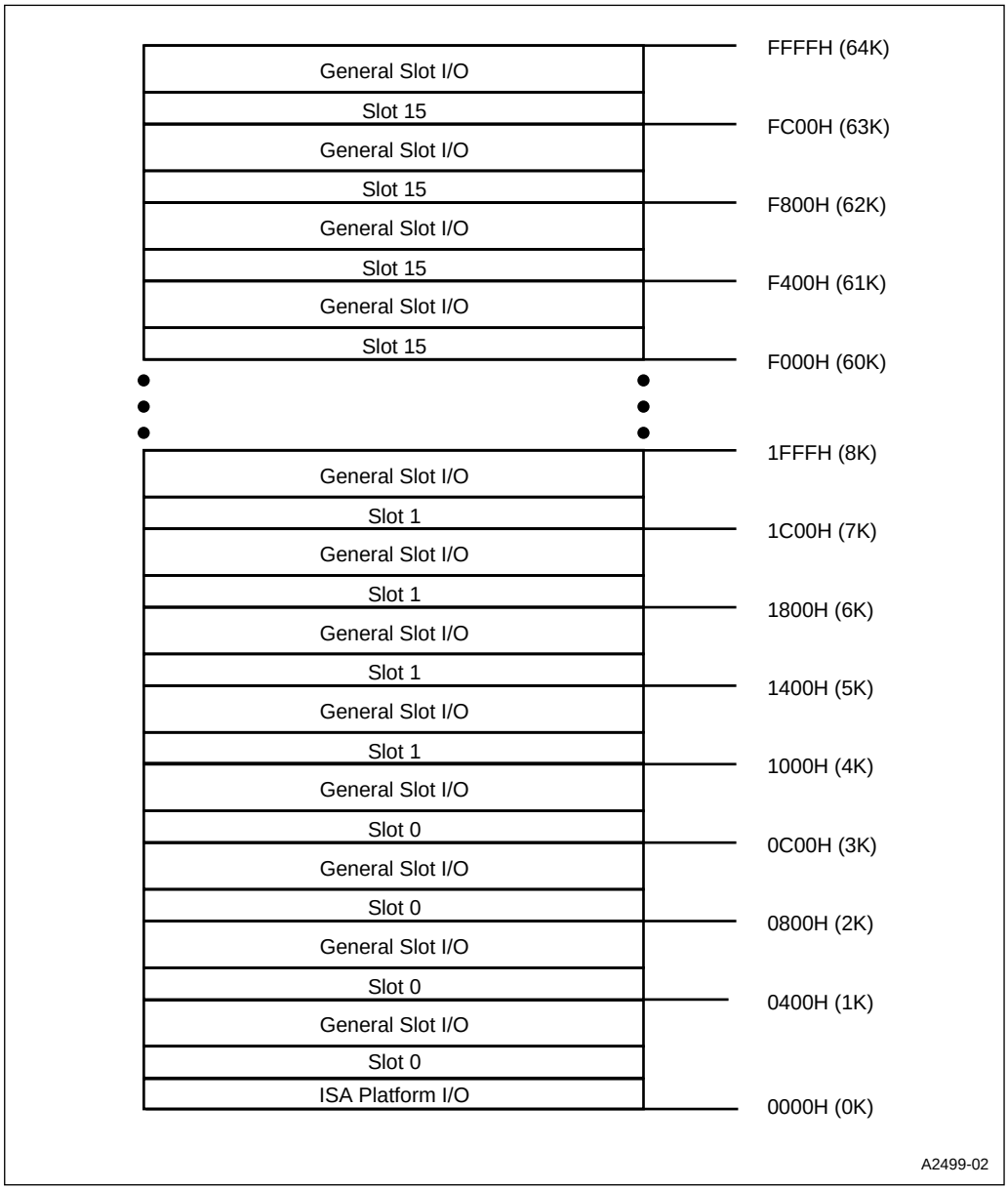


Figure 4-2. Expanded I/O Address Space (16-bit Decode)

The Intel386 EX processor uses slot 15 for the registers needed for integrated peripherals. Using this slot avoids conflicts with other devices in an EISA system, since EISA systems typically do not use slot 15.

4.4 ORGANIZATION OF PERIPHERAL REGISTERS

The registers associated with the integrated peripherals are physically located in slot 15 of the I/O space. There are sixteen 4 Kbyte address slots in I/O space. Slot 0 refers to 0H–0FFFH; slot 15 refers to 0F000H–0FFFFH. Table 4-1 shows the address map for the peripheral registers in slot 15. Note that the I/O addresses fall in address ranges 0F000H–0F0FFH, 0F400H–0F4FFH, and 0F800H–0F8FFH; utilizing the unique sets of 256 I/O addresses in Slot 15.

Table 4-1. Peripheral Register I/O Address Map in Slot 15

Register Description	I/O Address Range
DMA Controller 1	0F000H – 0F01FH
Master Interrupt Controller	0F020H – 0F03FH
Programmable Interval Timer	0F040H – 0F05FH
DMA Page Registers	0F080H – 0F09FH
Slave Interrupt Controller	0F0A0H – 0F0BFH
Math Coprocessor	0F0F0H – 0F0FFH
Chip Select Unit	0F400H – 0F47FH
Synchronous Serial I/O Unit	0F480H – 0F49FH
DRAM Refresh Control Unit	0F4A0H – 0F4BFH
Watchdog Timer Unit	0F4C0H – 0F4CFH
Asynchronous Serial I/O Channel 0 (COM1)	0F4F8H – 0F4FFH
Clock Generation and Power Management Unit	0F800H – 0F80FH
External/Internal Bus Interface Unit	0F810H – 0F81FH
Chip Configuration Registers	0F820H – 0F83FH
Parallel I/O Ports	0F860H – 0F87FH
Asynchronous Serial I/O Channel 1 (COM2)	0F8F8H – 0F8FFH

4.5 I/O ADDRESS DECODING TECHNIQUES

One of the key features of the Intel386 EX processor is that it is configurable for compatibility with the standard PC/AT architecture. In a PC/AT system, the platform I/O resources are located in the slot 0 I/O address space. For the Intel386 EX processor, this means that PC/AT-compatible internal peripherals should be reflected in slot 0 of the I/O space for DOS operating system and application software to access and manipulate them properly.

This discussion leads to the concepts of *DOS I/O space* and *expanded I/O space.*

DOS I/O Space DOS I/O space refers to the lower 1 Kbyte of I/O addresses, where only PC/AT-compatible peripherals can be mapped.

Expanded I/O Space Expanded I/O space refers to the top 4 Kbytes of I/O addresses, where all peripheral registers are physically located. The remainder of this section explains how special I/O address decoding schemes manipulate register addresses within these two I/O spaces.

4.5.1 Address Configuration Register

I/O address locations 22H and 23H in DOS I/O space offer a special case. These address locations are not used to access any peripheral registers in a PC/AT system. The Intel386 SL microprocessor and other integrated PC solutions use them to enable extra address space required for configuration registers specific to these products. On the Intel386 EX processor, these address locations are used to *hide* the peripheral registers in the expanded I/O space. The expanded I/O space can be enabled (registers visible) or disabled (registers hidden).

The 16-bit register at I/O location 22H can also be used to control mapping of various internal peripherals in I/O address space. This register, REMAPCFG, is defined in Figure 4-3.

The remap bits of this register control whether the internal PC compatible peripherals are mapped into the DOS I/O space. Setting the peripheral bit makes the peripheral accessible only in expanded I/O space. Clearing the peripheral bit makes the peripheral accessible in both DOS I/O space and expanded I/O space. To access the REMAPCFG register, you must first enable the expanded I/O address space as described in the next section. At reset, this register is cleared, mapping internal PC/AT-compatible peripherals into DOS I/O space.

Address Configuration Register
REMAPCFG

Expanded Addr: 0022H
PC/AT Address: 0022H
Reset State: 0000H

15							8
ESE	—	—	—	—	—	—	—
7							**0**
—	S1R	S0R	ISR	IMR	DR	—	TR

Bit Number	Bit Mnemonic	Function
15	ESE	0 = Disables expanded I/O space 1 = Enables expanded I/O space
14–7	—	Reserved.
6	S1R	0 = Makes serial channel 1 (COM2) accessible in both DOS I/O space and expanded I/O space 1 = Remaps serial channel 1 (COM2) address into expanded I/O space
5	S0R	0 = Makes serial channel 0 (COM1) accessible in both DOS I/O space and expanded I/O space 1 = Remaps serial channel 0 (COM1) address into expanded I/O space
4	ISR	0 = Makes the slave 82C59A interrupt controller accessible in both DOS I/O space and expanded I/O space 1 = Remaps slave 82C59A interrupt controller address into expanded I/O space
3	IMR	0 = Makes the master 82C59A interrupt controller accessible in both DOS I/O space and expanded I/O space 1 = Remaps master 82C59A interrupt controller address into expanded I/O space
2	DR	0 = Makes the DMA address accessible in both DOS I/O space and expanded I/O space 1 = Remaps DMA address into expanded I/O space
1	—	Reserved.
0	TR	0 = Makes the timer control unit accessible in both DOS I/O space and expanded I/O space 1 = Remaps timer control unit address into expanded I/O space

Figure 4-3. Address Configuration Register (REMAPCFG)

4.5.2 Enabling and Disabling the Expanded I/O Space

The Intel386 EX processor's expanded I/O space is enabled by a specific write sequence to I/O addresses 22H and 23H (Figure 4-4). Once the expanded I/O space is enabled, internal peripherals (timers, DMA, interrupt controllers and serial communication channels) can be mapped out of DOS I/O space (using the REMAPCFG register) and registers associated with other internal peripherals (such as the chip-select unit, power management unit, watchdog timer) can be accessed.

4.5.2.1 Programming REMAPCFG Example

The expanded I/O space enable (ESE) bit in the REMAPCFG register can be set only by three sequential write operations to I/O addresses 22H and 23H as described in Figure 4-4. Once ESE is set, REMAPCFG and all the on-chip registers in the expanded I/O address range 0F000H–0FFFFH can be accessed. The remap bits in REMAPCFG are still in effect even after the ESE bit is cleared.

```
;;disable interrupts
   CLI
; Enable expanded I/O space of Intel386(tm) EX processor
; for peripheral initialization.
   MOV AX, 08000H          ; Enable expanded I/O space
   OUT 23H, AL             ; and unlock the re-map bits
   XCHG AL, AH
   OUT 22H, AL
   OUT 22H, AX

;; at this point PC/AT peripherals can be mapped out
;; For example,
;; Map out the on-chip DMA channels from the DOS I/O space (slot 0)
   MOV AL, 04H
   OUT 22H, AL
; Disables expanded I/O space
   MOV AL, 00H
   OUT 23H, AL
;; Re-enable Interrupts
   STI
```

Figure 4-4. Setting the ESE Bit Code Example

The REMAPCFG register is write-protected until the expanded I/O space is enabled. When the enabling write sequence is executed, it sets the ESE bit. A program can check this bit to see whether it has access to the expanded I/O space registers. Clearing the ESE bit disables the expanded I/O space. This can be done by a byte write with a value of 0 to I/O address 23H. This again locks the REMAPCFG register and makes it read-only.

4.6 ADDRESSING MODES

Combinations of the value of ESE bit and the individual remap bits in the REMAPCFG register yield four different peripheral addressing modes for I/O address decoding.

4.6.1 DOS-compatible Mode

DOS-compatible mode is achieved by clearing ESE and all the peripheral remap bits. In this mode, all PC/AT-compatible peripherals are mapped into the DOS I/O space. Only address lines A9:0 are decoded for internal peripherals. Accesses to PC/AT-compatible peripherals are valid, while all other internal peripherals are inaccessible (see Figure 4-5).

This mode is useful for accessing the internal timer, interrupt controller, serial I/O ports, or DMA controller in a DOS-compatible environment.

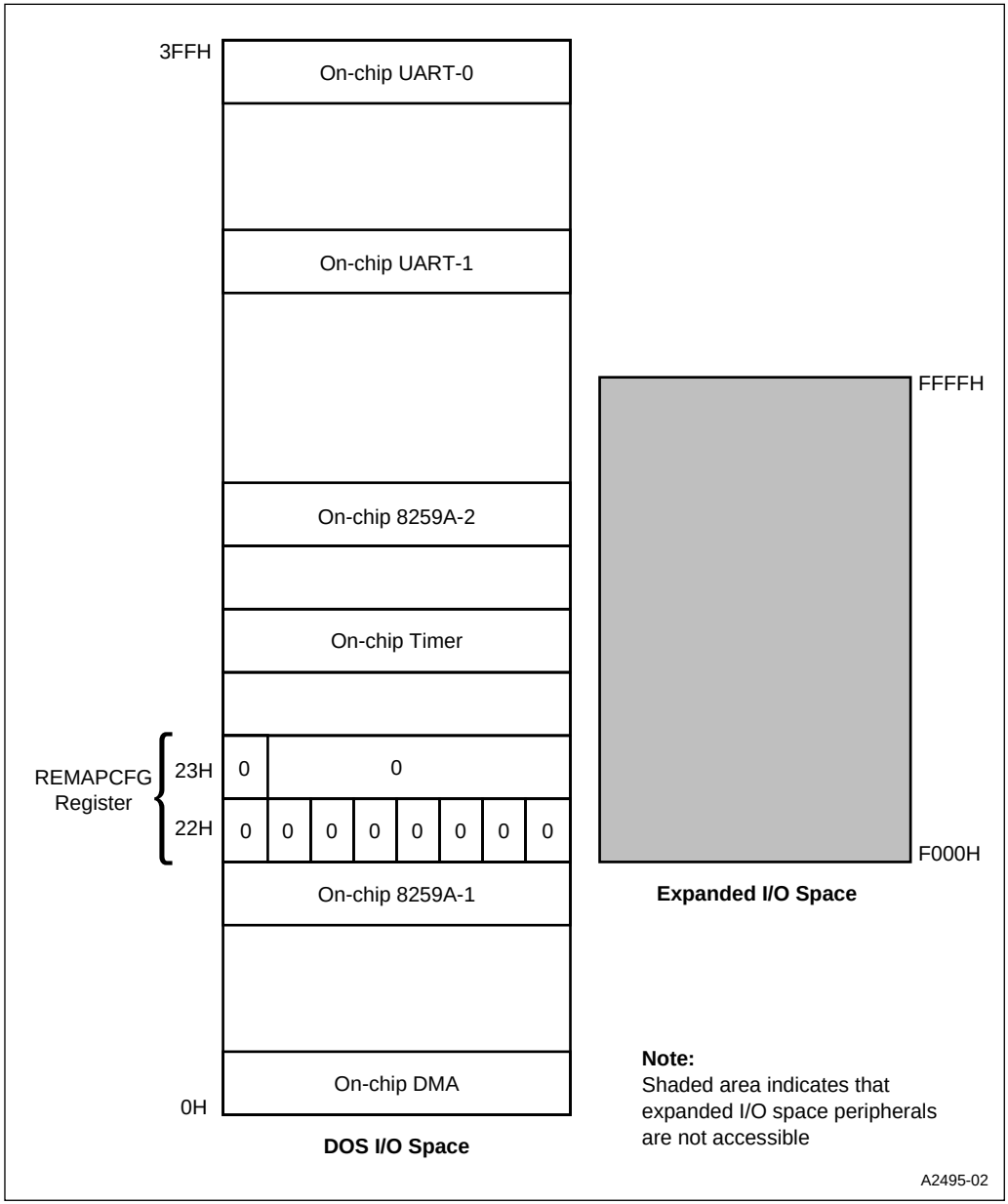


Figure 4-5. DOS-Compatible Mode

4.6.2 Nonintrusive DOS Mode

This mode is achieved by first setting the ESE bit (using the three sequential writes), setting the individual peripherals' remap bits, and then clearing the ESE bit. Peripherals whose remap bits are set are mapped out of DOS I/O space. Like DOS-compatible mode, only address lines A9:0 are decoded internally. This mode is useful for connecting an external peripheral instead of using the integrated peripheral. For example, a system might use an external 8237A DMA rather than using the internal DMA unit. For this configuration, set the ESE bit, set the remap bit associated with the DMA unit and then clear the ESE bit. In this case, the external 8237A is accessible in the DOS I/O space, while the internal DMA can be accessed only after the expanded I/O space is enabled. (See Figure 4-6.)

4.6.3 Enhanced DOS Mode

This mode is achieved by setting the ESE bit and clearing all PC/AT-compatible peripherals' remap bits. Address lines A15:0 are decoded internally. The expanded I/O space is enabled and the PC/AT-compatible internal peripherals are accessible in either DOS I/O space or expanded I/O space. (See Figure 4-7.) If an application frequently requires the additional peripherals, but at the same time wants to maintain DOS compatibility for ease of development, this is the most useful mode.

4.6.4 Non-DOS Mode

This mode is achieved by setting the ESE bit and setting all peripherals' remap bits. Address lines A15:0 are decoded internally. The expanded I/O space is enabled and all peripherals can be accessed only in expanded I/O space. This mode is useful for systems that don't require DOS compatibility and have other custom peripherals in slot 0 of the I/O space. (See Figure 4-8.)

For all DOS peripherals, the lower 10 bits in the DOS I/O space and in the expanded I/O space are identical (except the UARTs, whose lower 8 bits are identical). This makes correlation of their respective offsets in DOS and expanded I/O spaces easier. Also, the UARTs have fixed I/O addresses. This differs from standard PC/AT configurations, in which these address ranges are programmable.

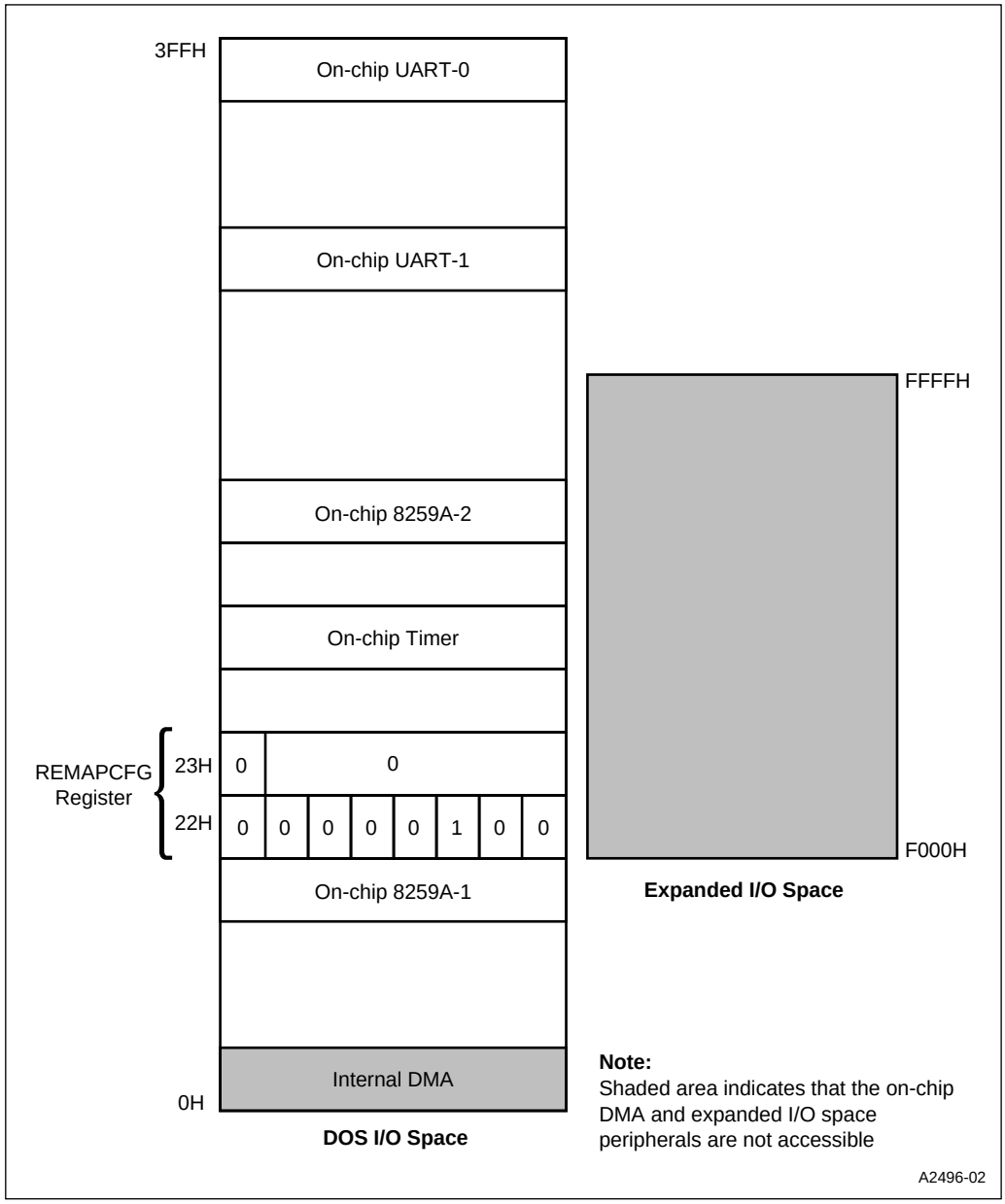


Figure 4-6. Example of Nonintrusive DOS-Compatible Mode

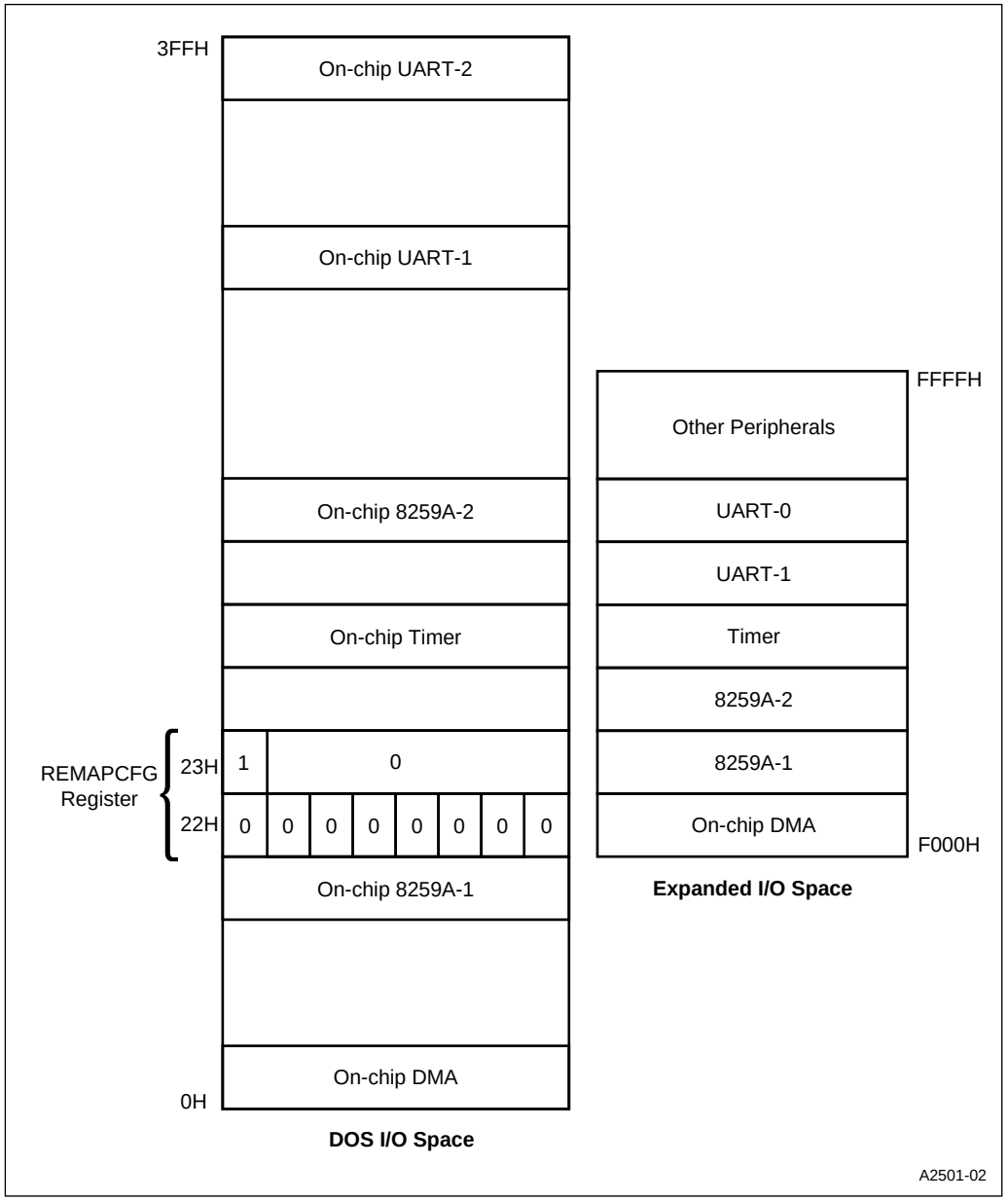


Figure 4-7. Enhanced DOS Mode

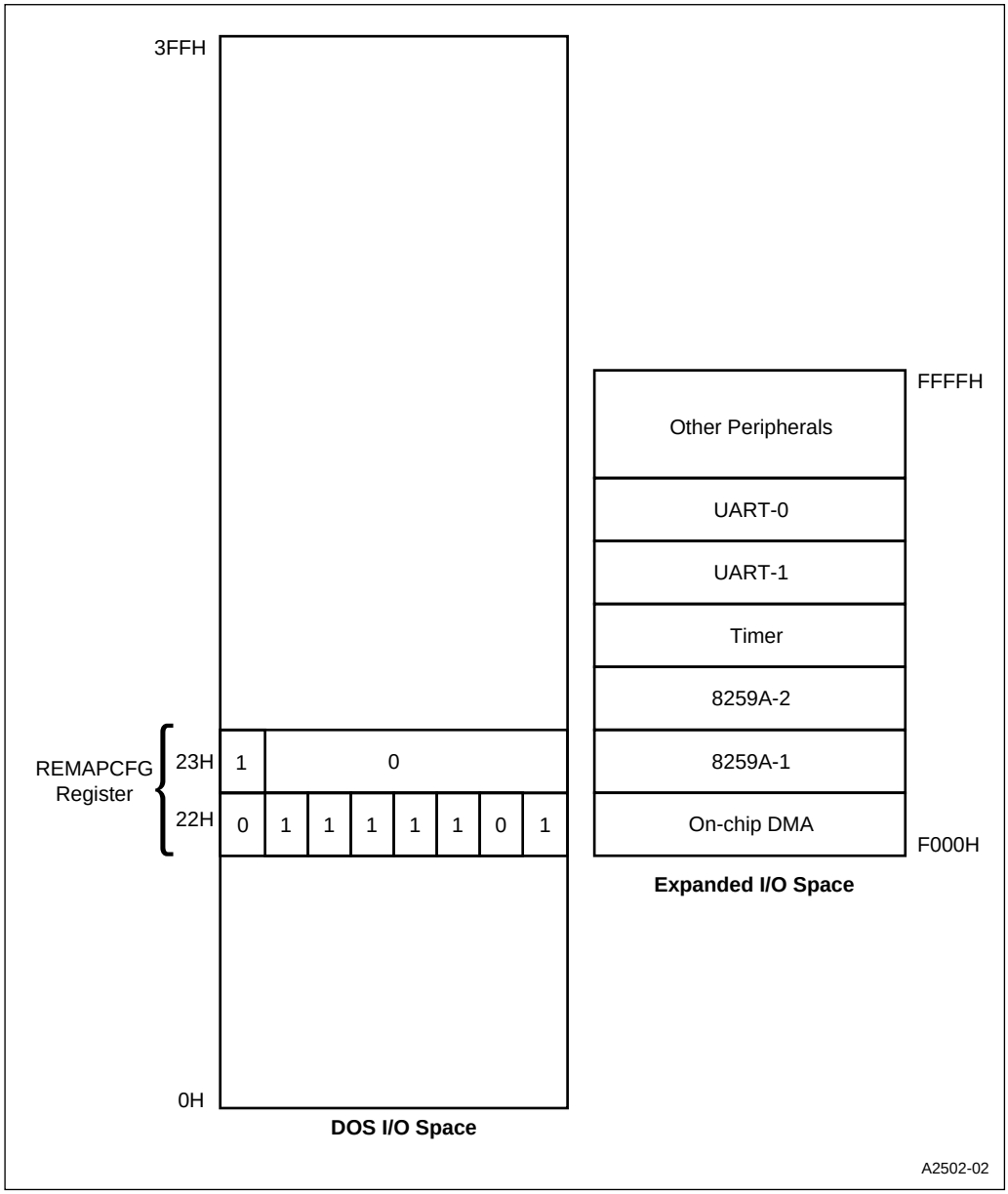


Figure 4-8. NonDOS Mode

4.7 PERIPHERAL REGISTER ADDRESSES

Table 4-2 lists the addresses and names of all user-accessible peripheral registers. I/O Registers can be accessed as bytes or words. Word accesses to byte registers result in two sequential 8-bit I/O transfers. The default (reset) value of each register is shown in the *Reset Value* column. An *X* in this column signifies that the register bits are undefined. Some address values do not access registers, but are decoded to provide a logic control signal. These addresses are listed as *Not a register* in the *Reset* column.

Table 4-2. Peripheral Register Addresses (Sheet 1 of 6)

Expanded Address	PC/AT Address	Access Type (Byte/Word)	Register Name	Reset Value
DMA Controller and Bus Arbiter				
F000H	0000H	Byte	DMA0TAR0/1 (Note 1)	XX
F001H	0001H	Byte	DMA0BYC0/1 (Note 1)	XX
F002H	0002H	Byte	DMA1TAR0/1 (Note 1)	XX
F003H	0003H	Byte	DMA1BYC0/1 (Note 1)	XX
F004H	0004H		Reserved	
F005H	0005H		Reserved	
F006H	0006H		Reserved	
F007H	0007H		Reserved	
F008H	0008H	Byte	DMACMD1/DMASTS	00H
F009H	0009H	Byte	DMASRR	00H
F00AH	000AH	Byte	DMAMSK	04H
F00BH	000BH	Byte	DMAMOD1	00H
F00CH	000CH	Byte	DMACLRBP	Not a register
F00DH	000DH	Byte	DMACLR	Not a register
F00EH	000EH	Byte	DMACLRMSK	Not a register
F00FH	000FH	Byte	DMAGRPMSK	03H
F010H		Byte	DMA0REQ0/1	XX
F011H		Byte	DMA0REQ2/3	XX
F012H		Byte	DMA1REQ0/1	XX
F013H		Byte	DMA1REQ2/3	XX
F014H			Reserved	
F015H			Reserved	
F016H			Reserved	
F017H			Reserved	

NOTES:
1. Byte pointer in flip-flop in DMA determines which register is accessed.
2. Shaded rows indicate reserved areas.

Table 4-2. Peripheral Register Addresses (Sheet 2 of 6)

Expanded Address	PC/AT Address	Access Type (Byte/Word)	Register Name	Reset Value
F018H		Byte	DMABSR	X1X10000B
F019H		Byte	DMACHR/DMAIS	00H
F01AH		Byte	DMACMD2	08H
F01BH		Byte	DMAMOD2	00H
F01CH		Byte	DMAIEN	00H
F01DH		Byte	DMAOVFE	0AH
F01EH		Byte	DMACLRTC	Not a register
Master Interrupt Controller				
F020H	0020H	Byte	ICW1m/IRRm/ISRm/ OCW2m/OCW3m	XX
F021H	0021H	Byte	ICW2m/ICW3m/ICW4m/ OCW1m/POLLm	XX
Address Configuration Register				
0022H	0022H	Word	REMAPCFG	0000H
Timer/counter Unit				
F040H	0040H	Byte	TMR0	XX
F041H	0041H	Byte	TMR1	XX
F042H	0042H	Byte	TMR2	XX
F043H	0043H	Byte	TMRCON	XX
DMA Page Registers				
F080H			Reserved	
F081H	0081H		Reserved	
F082H	0082H		Reserved	
F083H	0083H	Byte	DMA1TAR2	XX
F084H			Reserved	
F085H		Byte	DMA1TAR3	XX
F086H		Byte	DMA0TAR3	XX
F087H	0087H	Byte	DMA0TAR2	XX
F088H			Reserved	
F089H	0089H		Reserved	
F08AH	008AH		Reserved	
F08BH	008BH		Reserved	
F08CH			Reserved	

NOTES:
1. Byte pointer in flip-flop in DMA determines which register is accessed.
2. Shaded rows indicate reserved areas.

Table 4-2. Peripheral Register Addresses (Sheet 3 of 6)

Expanded Address	PC/AT Address	Access Type (Byte/Word)	Register Name	Reset Value
F08DH			Reserved	
F08EH			Reserved	
F08FH			Reserved	
F098H		Byte	DMA0BYC2	XX
F099H		Byte	DMA1BYC2	XX
F09AH			Reserved	
F09BH			Reserved	
A20GATE and Fast CPU Reset				
F092H	0092H	Byte	PORT92	XXXXXX10B
Slave Interrupt Controller				
F0A0H	00A0H	Byte	ICW1s/IRRs/ISRs/ OCW2s/OCW3s	XX
F0A1H	00A1H	Byte	ICW2s/ICW3s/ICW4s/ OCW1s/POLLs	XX
Chip-select Unit				
F400H		Word	CS0ADL	0000H
F402H		Word	CS0ADH	0000H
F404H		Word	CS0MSKL	0000H
F406H		Word	CS0MSKH	0000H
F408H		Word	CS1ADL	0000H
F40AH		Word	CS1ADH	0000H
F40CH		Word	CS1MSKL	0000H
F40EH		Word	CS1MSKH	0000H
F410H		Word	CS2ADL	0000H
F412H		Word	CS2ADH	0000H
F414H		Word	CS2MSKL	0000H
F416H		Word	CS2MSKH	0000H
F418H		Word	CS3ADL	0000H
F41AH		Word	CS3ADH	0000H
F41CH		Word	CS3MSKL	0000H
F41EH		Word	CS3MSKH	0000H
F420H		Word	CS4ADL	0000H
F422H		Word	CS4ADH	0000H

NOTES:
1. Byte pointer in flip-flop in DMA determines which register is accessed.
2. Shaded rows indicate reserved areas.

Table 4-2. Peripheral Register Addresses (Sheet 4 of 6)

Expanded Address	PC/AT Address	Access Type (Byte/Word)	Register Name	Reset Value
F424H		Word	CS4MSKL	0000H
F426H		Word	CS4MSKH	0000H
F428H		Word	CS5ADL	0000H
F42AH		Word	CS5ADH	0000H
F42CH		Word	CS5MSKL	0000H
F42EH		Word	CS5MSKH	0000H
F430H		Word	CS6ADL	0000H
F432H		Word	CS6ADH	0000H
F434H		Word	CS6MSKL	0000H
F436H		Word	CS6MSKH	0000H
F438H		Word	UCSADL	FF6FH
F43AH		Word	UCSADH	FFFFH
F43CH		Word	UCSMSKL	FFFFH
F43EH		Word	UCSMSKH	FFFFH
Synchronous Serial I/O Unit				
F480H		Word	SSIOTBUF	0000H
F482H		Word	SSIORBUF	0000H
F484H		Byte	SSIOBAUD	00H
F486H		Byte	SSIOCON1	C0H
F488H		Byte	SSIOCON2	00H
F48AH		Byte	SSIOCTR	00H
Refresh Control Unit				
F4A0H		Word	RFSBAD	0000H
F4A2H		Word	RFSCIR	0000H
F4A4H		Word	RFSCON	0000H
F4A6H		Word	RFSADD	00FFH
Watchdog Timer Unit				
F4C0H		Word	WDTRLDH	003FH
F4C2H		Word	WDTRLDL	FFFFH
F4C4H		Word	WDTCNTH	003FH
F4C6H		Word	WDTCNTL	FFFFH
F4C8H		Word	WDTCLR	Not a register

NOTES:
1. Byte pointer in flip-flop in DMA determines which register is accessed.
2. Shaded rows indicate reserved areas.

Table 4-2. Peripheral Register Addresses (Sheet 5 of 6)

Expanded Address	PC/AT Address	Access Type (Byte/Word)	Register Name	Reset Value
F4CAH		Byte	WDTSTATUS	00H
Asynchronous Serial I/O Channel 0 (COM1)				
F4F8H	03F8H	Byte	RBR0/TBR0/DLL0	XX/XX/02H
F4F9H	03F9H	Byte	IER0/DLH0	00H/00H
F4FAH	03FAH	Byte	IIR0	01H
F4FBH	03FBH	Byte	LCR0	00H
F4FCH	03FCH	Byte	MCR0	00H
F4FDH	03FDH	Byte	LSR0	60H
F4FEH	03FEH	Byte	MSR0	X0H
F4FFH	03FFH	Byte	SCR0	XX
Clock Generation and Power Management				
F800H		Byte	PWRCON	00H
F804H		Word	CLKPRS	0000H
Device Configuration Registers				
F820H		Byte	P1CFG	00H
F822H		Byte	P2CFG	00H
F824H		Byte	P3CFG	00H
F826H		Byte	PINCFG	00H
F830H		Byte	DMACFG	00H
F832H		Byte	INTCFG	00H
F834H		Byte	TMRCFG	00H
F836H		Byte	SIOCFG	00H
Parallel I/O Ports				
F860H		Byte	P1PIN	XX
F862H		Byte	P1LTC	FFH
F864H		Byte	P1DIR	FFH
F868H		Byte	P2PIN	XX
F86AH		Byte	P2LTC	FFH
F86CH		Byte	P2DIR	FFH
F870H		Byte	P3PIN	XX
F872H		Byte	P3LTC	FFH
F874H		Byte	P3DIR	FFH

NOTES:
1. Byte pointer in flip-flop in DMA determines which register is accessed.
2. Shaded rows indicate reserved areas.

Table 4-2. Peripheral Register Addresses (Sheet 6 of 6)

Expanded Address	PC/AT Address	Access Type (Byte/Word)	Register Name	Reset Value
Asynchronous Serial I/O Channel 1 (COM2)				
F8F8H	02F8H	Byte	RBR1/TBR1/DLL1	XX/XX/02H
F8F9H	02F9H	Byte	IER1/DLH1	00H/00H
F8FAH	02FAH	Byte	IIR1	01H
F8FBH	02FBH	Byte	LCR1	00H
F8FCH	02FCH	Byte	MCR1	00H
F8FDH	02FDH	Byte	LSR1	60H
F8FEH	02FEH	Byte	MSR1	X0H
F8FFH	02FFH	Byte	SCR1	XX

NOTES:
1. Byte pointer in flip-flop in DMA determines which register is accessed.
2. Shaded rows indicate reserved areas.

5

DEVICE CONFIGURATION

CHAPTER 5
DEVICE CONFIGURATION

The Intel386™ EX processor provides many possible signal to pin connections as well as peripheral to peripheral connections. This chapter describes the available configurations and how to configure them.

This chapter is organized as follows:

5.1 INTRODUCTION

Device configuration is the process of setting up the microprocessor's on-chip peripherals† for a particular system design. Specifically, device configuration consists of programming registers to connect peripheral signals to the package pins and interconnect the peripherals. The peripherals include the following:

- DMA Controller (DMA)
- Interrupt Control Unit (ICU)
- Timer/counter Unit (TCU)
- Asynchronous Serial I/O Units (SIO0, SIO1)
- Synchronous Serial I/O Unit (SSIO)
- Refresh Control Unit (RCU)
- Chip-select Unit (CSU)
- Watchdog Timer Unit (WDT)

In addition, the pin configuration registers control connections from the coprocessor to the core and pin connections to the bus arbiter.

† In this chapter, the terms "peripheral" and "on-chip peripheral" are used interchangeably. An "off-chip peripheral" is external to the Intel386 EX processor.

Figure 5-1 shows Peripheral A and its connections to other peripherals and the package pins. The "Internal Connection Logic" provides three kinds of connections:

- Connections between peripherals
- Connections to package pins via multiplexers
- Direct connections to package pins without multiplexers

The internal connection logic is controlled by the Peripheral A configuration register.

Each pin multiplexer ("Pin Mux") connects one of two internal signals to a pin. One is a peripheral signal. The second signal can be an I/O port signal or a signal from/to another peripheral. The pin multiplexers are controlled by the pin configuration registers. Some input-only pins without multiplexers ("Shared Pins w/o Muxes") are routed to two different peripherals. Your design should use only one of the inputs and disable or ignore the input going to the second peripheral.

Together, the peripheral configuration registers and the pin configuration registers allow you to select the peripherals to be used, to interconnect them as your design requires, and to bring selected signals to the package pins.

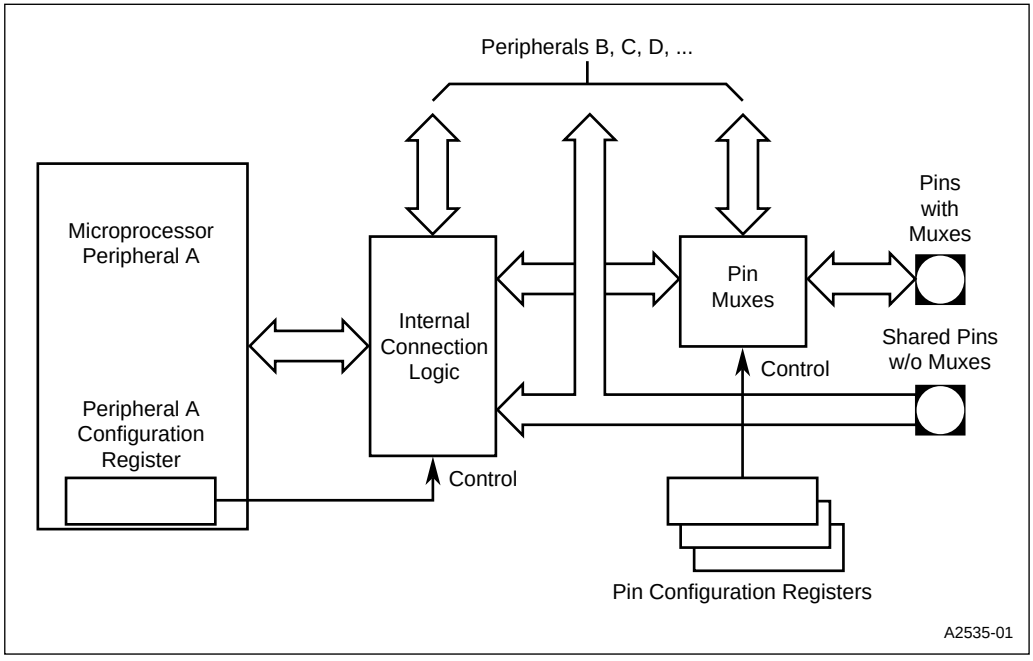


Figure 5-1. Peripheral and Pin Connections

5.2 PERIPHERAL CONFIGURATION

This section describes the configuration of each on-chip peripheral. For more detailed information on the peripheral itself, see the chapter describing that peripheral.

The symbology used for signals that share a device pin is shown in Figure 5-2. Of the two signal names by a pin, the upper signal is associated with the peripheral in the figure. The lower signal in parentheses is the alternate signal, which connects to a different peripheral or the core. When a pin has a multiplexer, it is shown as a switch, and the register bit that controls it is noted above the switch.

5.2.1 DMA Controller, Bus Arbiter, and Refresh Unit Configuration

Figure 5-2 shows the DMA controller, bus arbiter, and refresh unit configuration. Requests for a DMA data transfer are shown as inputs to the multiplexer:

- A serial I/O transmitter (TXEDMA0, TXEDMA1) or receiver (RBFDMA0, RBFDMA1)
- A synchronous serial I/O transmitter (SSTBE) or receiver (SSRBF)
- A timer (OUT1, OUT2)
- An external source (DRQ0, DRQ1)

The inputs are selected by the DMA configuration register (see Figure 5-3).

5.2.1.1 Using The DMA Unit with External Devices

For each DMA channel, three bits in the DMA configuration register (Figure 5-3) select the external request input or one of seven request inputs from the peripherals. Another bit enables or disables that channel's DMA acknowledge signal (DACK*n*#) at the device pin. Enable the DACK*n*# signal only when you are using the external request signal (DRQ*n*) and need DACK*n*#. The acknowledge signals are not routed to the on-chip peripherals, and therefore, these peripherals cannot initiate single-cycle (fly-by) DMA transfers.

An external bus master cannot talk directly to internal peripheral modules because the external address lines are outputs only. However, an external device could use a DMA channel to transfer data to or from an internal peripheral because the DMA generates the addresses. This transaction would be a two-cycle DMA bus transaction.

5.2.1.2 DMA Service to an SIO or SSIO Peripheral

A DMA unit is useful for servicing an SIO or SSIO peripheral operating at a high baud rate. At high baud rates, the interrupt response time of the core may be too long to allow the serial channels to use an interrupt to service the receive-buffer-full condition. By the time the interrupt service routine (ISR) is ready to transfer the receive-buffer data to memory, new data would have been loaded into the buffer. The issue is the interrupt latency which is the amount of time the processor takes from recognizing the interrupt to executing the first line of code in the ISR. This interrupt latency needs to be calculated to determine if an ISR can handle the high baud rate. If the Interrupt Latency is too high, data transfers to and from the serial channels can occur within a few bus cycles of the time that a serial unit is ready to move data by using an appropriately

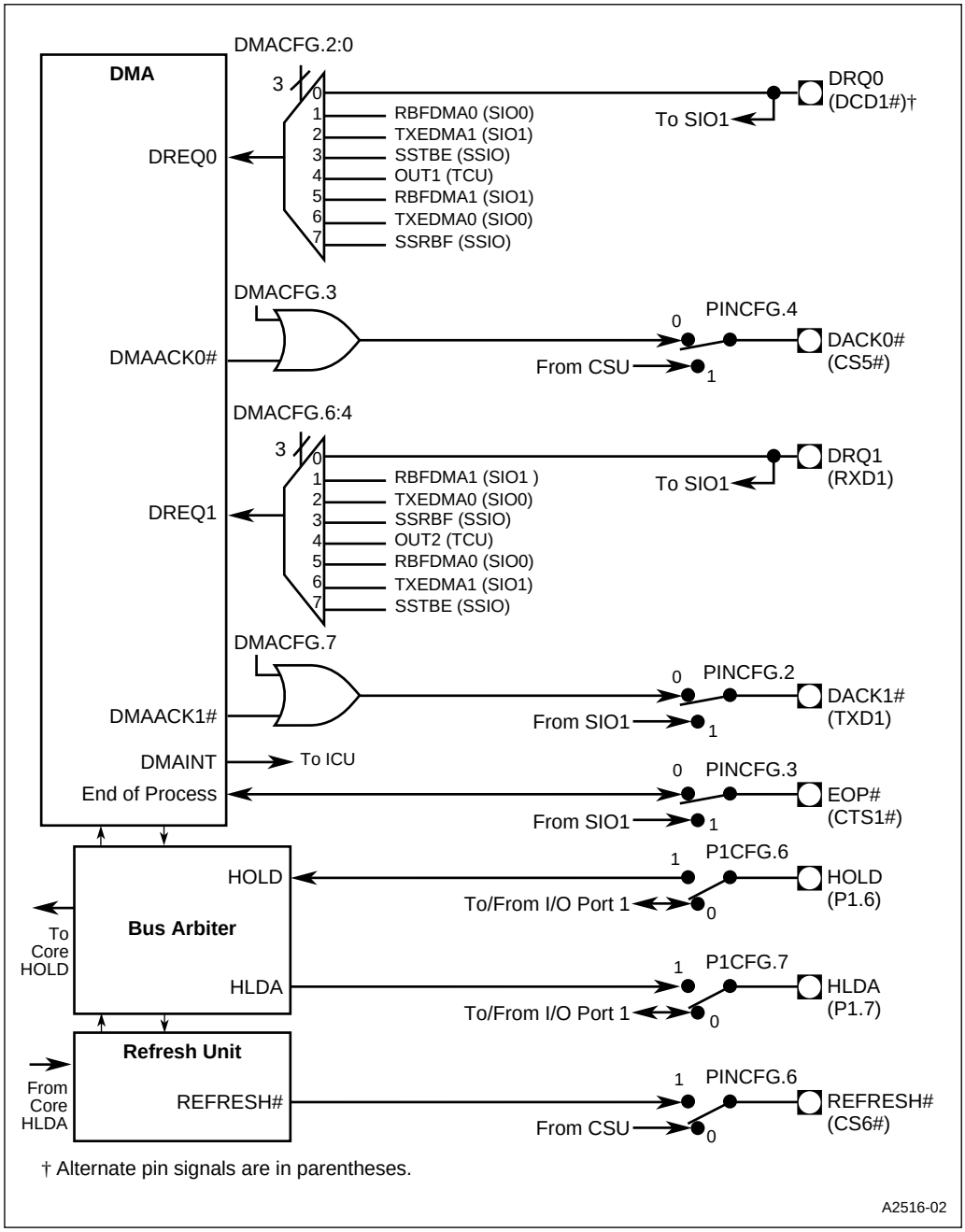
configured DMA channel. SIO and SSIO inputs to the DMA are selected by the DMA configuration register (Figure 5-3).

5.2.1.3 Using The Timer To Initiate DMA Transfers

A timer output (OUT1, OUT2) can initiate periodic data transfers by the DMA. A DMA channel is programmed for the transfer, then a timer output pulse triggers the transfer. The most useful DMA and timer combinations for this type of transfer are the periodic timer modes (mode 2 and mode 3) with the DMA block-transfer mode programmed. See Chapter 10, "TIMER/COUNTER UNIT," and Chapter 12, "DMA CONTROLLER," for more information on how to program the peripherals.

5.2.1.4 Limitations Due To Pin Signal Multiplexing

Pin signal multiplexing can preclude the simultaneous use of a DMA channel and another peripheral or specific peripheral signal (see Figure 5-2). For example, using DMA channel 1 with an external requester device precludes using SIO channel 1 due to the multiplexed signal pairs DRQ1/RXD1 and DACK1#/TXD1. Please refer to the *Intel386™ EX Microprocessor Pin Multiplexing Map* (Order Number 272587) for a complete diagram of multiplexed signals.



† Alternate pin signals are in parentheses.

A2516-02

Figure 5-2. Configuration of DMA, Bus Arbiter, and Refresh Unit

DMA Configuration
DMACFG
(read/write)

Expanded Addr: **F830H**
ISA Addr: **—**
Reset State: **00H**

7							0
D1MSK	D1REQ2	D1REQ1	D1REQ0	D0MSK	D0REQ2	D0REQ1	D0REQ0

Bit Number	Bit Mnemonic	Function
7	D1MSK	DMA Acknowledge 1 Mask: 0 = DMA channel 1's acknowledge (DMAACK1#) signal is not masked. 1 = Masks DMA channel 1's acknowledge (DMAACK1#) signal. Useful when channel 1's request (DREQ1) input is connected to an internal peripheral.
6–4	D1REQ2:0	DMA Channel 1 Request Connection: Connects one of the eight possible hardware sources to channel 1's request input (DREQ1). 000 = DRQ1 pin (external peripheral) 001 = SIO channel 1's receive buffer full signal (RBFDMA1) 010 = SIO channel 0's transmit buffer empty signal (TXEDMA0) 011 = SSIO receive holding buffer full signal (SSRBF) 100 = TCU counter 2's output signal (OUT2) 101 = SIO channel 0's receive buffer full signal (RBFDMA0) 110 = SIO channel 1's transmit buffer empty signal (TXEDMA1) 111 = SSIO transmit holding buffer empty signal (SSTBE)
3	D0MSK	DMA Acknowledge 0 Mask: 0 = DMA channel 0's acknowledge (DMAACK0#) signal is not masked. 1 = Masks DMA channel 0's acknowledge (DMAACK0#) signal. Useful when channel 0's request (DREQ0) input is connected to an internal peripheral.
2–0	D0REQ2:0	DMA Channel 0 Request Connection: Connects one of the eight possible hardware sources to channel 0's request input (DREQ0). 000 = DRQ0 pin (external peripheral) 001 = SIO channel 0's receive buffer full signal (RBFDMA0) 010 = SIO channel 1's transmit buffer empty signal (TXEDMA1) 011 = SSIO transmit holding buffer empty signal (SSTBE) 100 = TCU counter 1's output signal (OUT1) 101 = SIO channel 1's receive buffer full signal (RBFDMA1) 110 = SIO channel 0's transmit buffer empty signal (TXEDMA0) 111 = SSIO receive holding buffer full signal (SSRBF)

Figure 5-3. DMA Configuration Register (DMACFG)

5.2.2 Interrupt Control Unit Configuration

The interrupt control unit (ICU) comprises two 82C59A interrupt controllers connected in cascade, as shown in Figure 5-4. (See Chapter 9 for more information.) Figure 5-5 describes the interrupt configuration register (INTCFG).

The ICU receives requests from eight internal sources:

- Three outputs from the timer/counter unit (OUT2:0)
- An output from each of the serial I/O units (SIOINT1:0)
- An output from the synchronous serial I/O unit (SSIOINT)
- An output from the DMA unit (DMAINT)
- An output from the WDT unit (WDTOUT#)

In addition, the ICU controls the interrupt sources on ten external pins:

- INT3:0 (multiplexed with I/O port signals P3.5:2) are enabled or disabled by the P3CFG register (see Figure 5-18).
- INT7:4 share their package pins with four TCU inputs: TMRGATE1, TMRCLK1, TMRGATE0, and TMRCLK0. These signal pairs are not multiplexed; however, the pin inputs are enabled or disabled by the INTCFG register.
- INT9:8 share their pins with TMROUT1, TMROUT0, P3.1, P3.0

The three cascade outputs (CAS2:0) should be enabled when an external 82C59A module is connected to one of the INT9:8 or INT3:0 signals. The cascade outputs are ORed with address lines A18:16. See “Interrupt Acknowledge Cycle” on page 6-23 for details.

Use Tables 5-1 and 5-2 to configure the functionality of the master 82C59A’s IR3, IR4 inputs, and the associated external pins.

Table 5-1. Master's IR3 Connections

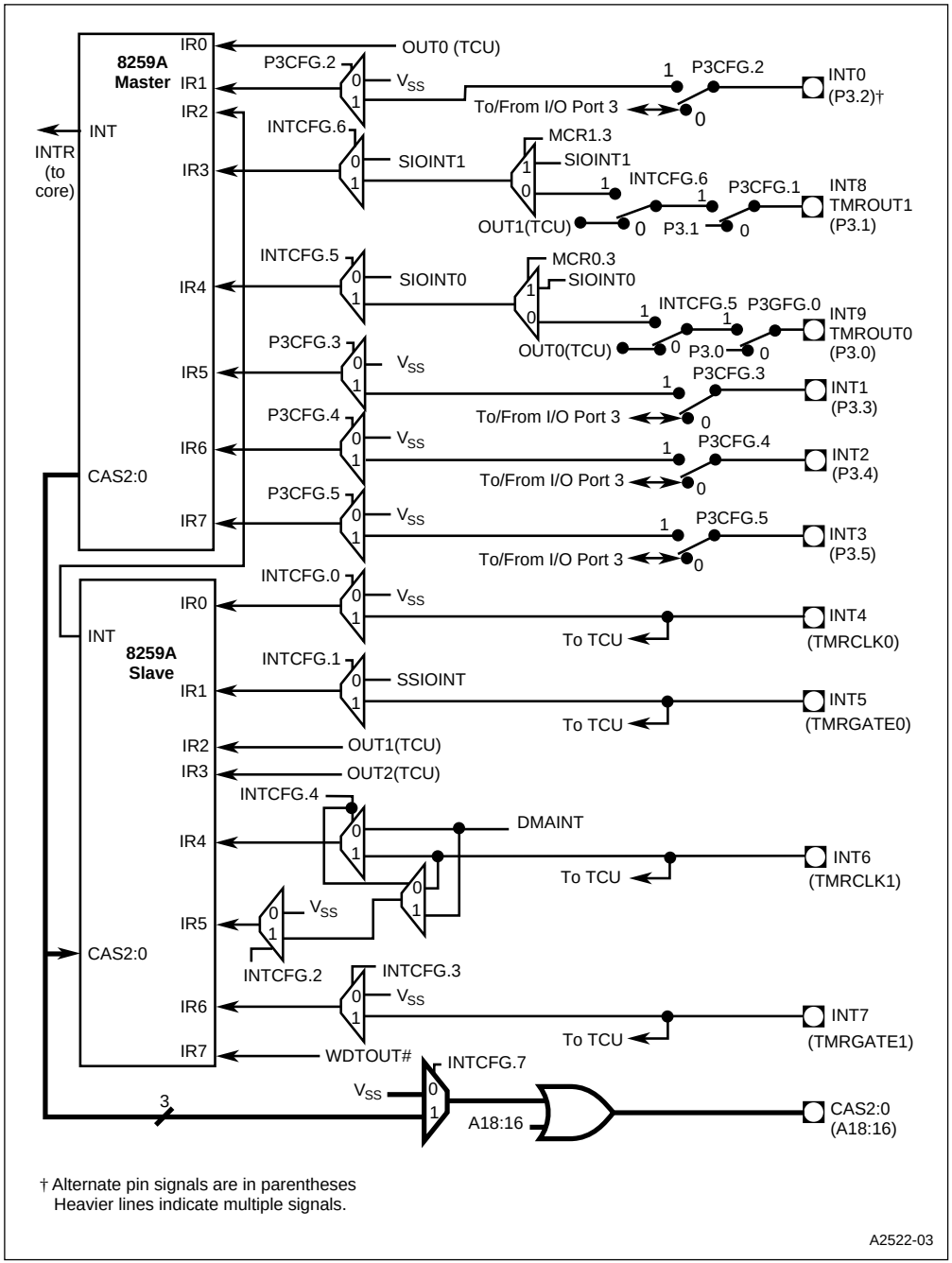
Function	INTCFG.6	MCR1.3	P3CFG.1
IR3 connected to SIOINT1 P3.1 selected at pin (P3.1)	0	X	0
IR3 connected to SIOINT1 OUT1 connected to pin (TMROUT1)	0	X	1
IR3 internally driven low P3.1 selected at pin (P3.1)	1	0	0
IR3 connected to pin (INT8)	1	0	1
IR3 connected to SIOINT1 P3.1 selected at pin (P3.1)	1	1	0
IR3 connected to SIOINT1 pin (INT8) must not be left floating	1	1	1

NOTE: X is a don't care

Table 5-2. Master's IR4 Connections

Function	INTCFG.5	MCR0.3	P3CFG.0
IR4 connected to SIOINT0 P3.0 selected at pin (P3.0)	0	X	0
IR4 connected to SIOINT0 OUT0 connected to pin (TMROUT0)	0	X	1
IR4 internally driven low P3.0 selected at pin (P3.0)	1	0	0
IR4 connected to pin (INT9)	1	0	1
IR4 connected to SIOINT0 P3.0 selected at pin (P3.0)	1	1	0
IR4 connected to SIOINT0 pin (INT9) must not be left floating	1	1	1

NOTE: X is a don't care



† Alternate pin signals are in parentheses
Heavier lines indicate multiple signals.

A2522-03

Figure 5-4. Interrupt Control Unit Configuration

Interrupt Configuration
INTCFG
(read/write)

Expanded Addr: **F832H**
ISA Addr: **—**
Reset State: **00H**

7							0
CE	IR3	IR4	SWAP	IR6	IR5/IR4	IR1	IR0

Bit Number	Bit Mnemonic	Function
7	CE	Cascade Enable: 0 = Disables the cascade signals CAS2:0 from appearing on the A18:16 address lines during interrupt acknowledge cycles. 1 = Enables the cascade signals CAS2:0, providing access to external slave 82C59A devices. The cascade signals are used to address specific slaves. If enabled, slave IDs appear on the A18:16 address lines during interrupt acknowledge cycles, but are high during idle cycles.
6	IR3	Internal Master IR3 Connection: See Table 5-1 on page 5-8 for all the IR3 configuration options.
5	IR4	Internal Master IR4 Connection: See Table 5-2 on page 5-8 for all the IR4 configuration options.
4	SWAP	INT6/DMAINT Connection: 0 = Connects DMAINT to the slave IR4. Connects INT6 to the slave IR5. 1 = Connects the INT6 pin to the slave IR4. Connects DMAINT to the slave IR5.
3	IR6	Internal Slave IR6 Connection: 0 = Connects V_{SS} to the slave IR6 signal. 1 = Connects the INT7 pin to the slave IR6 signal.
2	IR5/IR4	Internal Slave IR4 or IR5 Connection: These depend on whether INTCFG.4 is set or clear. 0 = Connects V_{SS} to the slave IR5 signal. 1 = Connects either the INT6 pin or DMAINT to the slave IR5 signal.
1	IR1	Internal Slave IR1 Connection: 0 = Connects the SSIO interrupt signal (SSIOINT) to the slave IR1 signal. 1 = Connects the INT5 pin to the slave IR1 signal.
0	IR0	Internal Slave IR0 Connection: 0 = Connects V_{SS} to the slave IR0 signal. 1 = Connects the INT4 pin to the slave IR0 signal.

Figure 5-5. Interrupt Configuration Register (INTCFG)

5.2.3 Timer/counter Unit Configuration

The three-channel Timer/counter Unit (TCU) and its configuration register (TMRCFG) are shown in Figure 5-6 and Figure 5-7. The clock inputs can be external signals (TMRCLK2:0) or the on-chip programmable clock (PSCLK). All clock inputs can be held low by programming bits in the TMRCFG register. The gate inputs can be controlled through software using TMRCFG.6 and the appropriate GT*n*CON bits in the TMRCFG register. Several of the timer signals go to the interrupt control unit (see Figure 5-4).

The Timer/counter0 and Timer/counter1 signals are selected individually. In contrast, the Timer/counter2 signals (TMRCLK2, TMRGATE2, TMROUT2) are selected as a group. Note that using the Timer/counter2 signals precludes use of the coprocessor signals (PEREQ, BUSY#, and ERROR#).

The CLKIN*n* and GATE*n* inputs of Timer/counter0 and Timer/counter1 are routed directly to shared input pins, TMRCLK0/INT4, TMRCLK1/INT6, TMRGATE0/INT5 and TMRGATE1/INT7. The OUT*n* inputs of these two counters can be connected to pins TMROUT0/INT9/P3.0 and TMROUT1/INT8/P3.1 respectively, using bits in registers P3CFG and INTCFG.

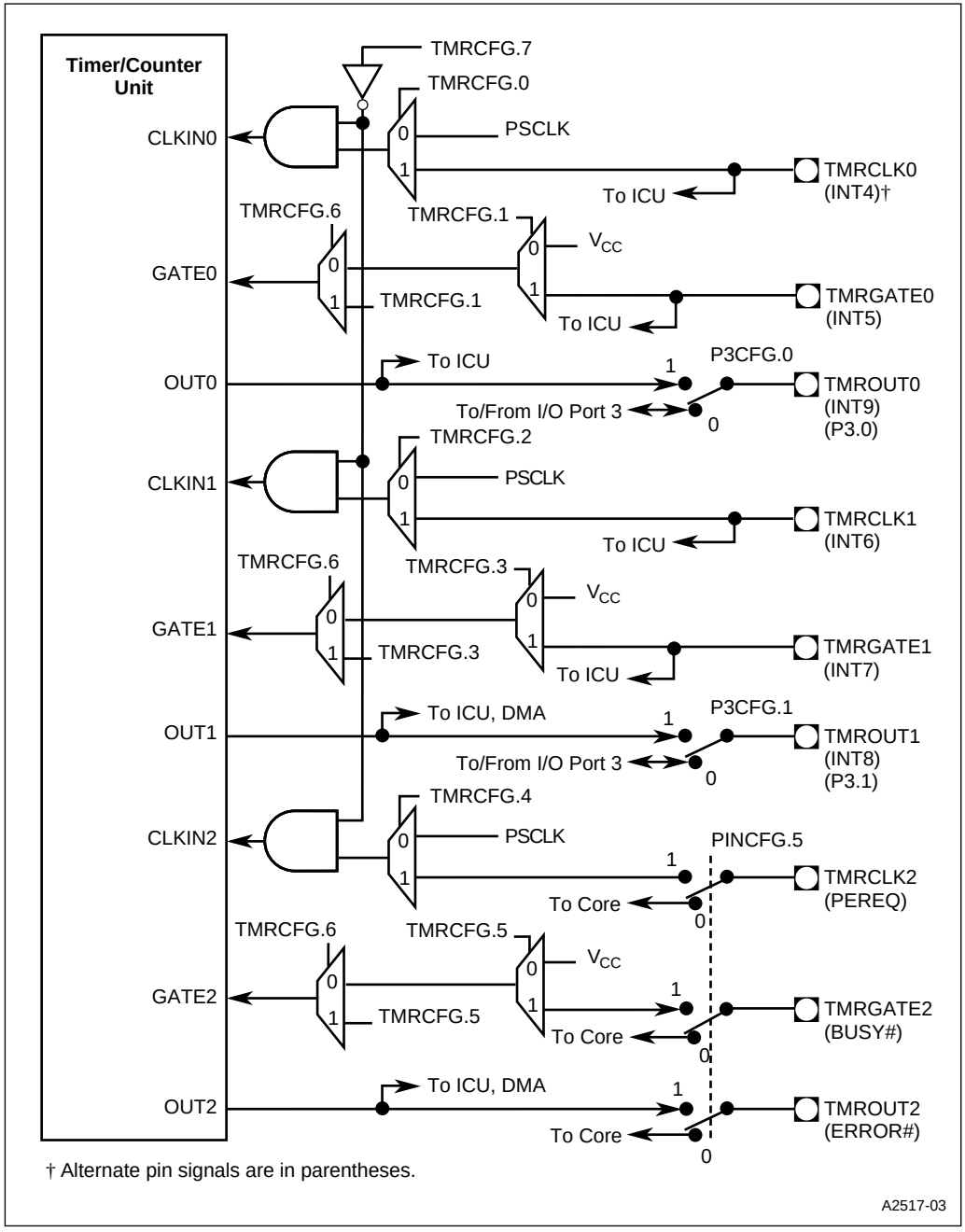


Figure 5-6. Timer/Counter Unit Configuration

Timer Configuration
TMRCFG
(read/write)

Expanded Addr: **F834H**
ISA Addr: **—**
Reset State: **00H**

7							0
TMRDIS	SWGTEN	GT2CON	CK2CON	GT1CON	CK1CON	GT0CON	CK0CON

Bit Number	Bit Mnemonic	Function
7	TMRDIS	Timer Disable: 0 = Enables the CLKIN*n* signals. 1 = Disables the CLKIN*n* signals.
6	SWGTEN	Software GATE*n* Enable 0 = Connects GATE*n* to either the V_{CC} pin or the TMRGATE*n* pin. 1 = Enables GT2CON, GT1CON, and GT0CON to control the connections to GATE2, GATE1 and GATE0 respectively.
5	GT2CON	Gate 2 Connection: **SWGTEN GT2CON** 0 0 Connects GATE2 to V_{CC}. 0 1 Connects GATE2 to the TMRGATE2 pin. 1 0 Turns GATE2 off. 1 1 Turns GATE2 on.
4	CK2CON	Clock 2 Connection: 0 = Connects CLKIN2 to the internal PSCLK signal. 1 = Connects CLKIN2 to the TMRCLK2 pin.
3	GT1CON	Gate 1 Connection: **SWGTEN GT1CON** 0 0 Connects GATE1 to V_{CC}. 0 1 Connects GATE1 to the TMRGATE1 pin. 1 0 Turns GATE1 off. 1 1 Turns GATE1 on.
2	CK1CON	Clock 1 Connection: 0 = Connects CLKIN1 to the internal PSCLK signal. 1 = Connects CLKIN1 to the TMRCLK1 pin.
1	GT0CON	Gate 0 Connection: **SWGTEN GT0CON** 0 0 Connects GATE0 to V_{CC}. 0 1 Connects GATE0 to the TMRGATE1 pin. 1 0 Turns GATE0 off. 1 1 Turns GATE0 on.
0	CK0CON	Clock 0 Connection: 0 = Connects CLKIN0 to the internal PSCLK signal. 1 = Connects CLKIN0 to the TMRCLK0 pin.

Figure 5-7. Timer Configuration Register (TMRCFG)

5.2.4 Asynchronous Serial I/O Configuration

Figures 5-8 and 5-9 show the asynchronous serial I/O unit configuration, consisting of channels SIO0 and SIO1. Each channel has one output (SIOINT0, SIOINT1) to the interrupt control unit (see Figure 5-4) and two outputs to the DMA unit. (These signals do not go to package pins.) SIOINT*n* is active when any one of the SIO status signals (receiver line status, receiver buffer full, transmit buffer empty, modem status) is set and enabled. All SIO0 pins are multiplexed with I/O port signals.

Using SIO1 precludes using DMA channel 1 for external DMA requests due to the multiplexing of the transmit and receive signals with DMA signals (RXD1/DRQ1, TXD1/DACK1#).

NOTE

Using SIO1 modem signals RTS1#, DSR1#, DTR1#, and RI1# precludes use of the SSIO unit.

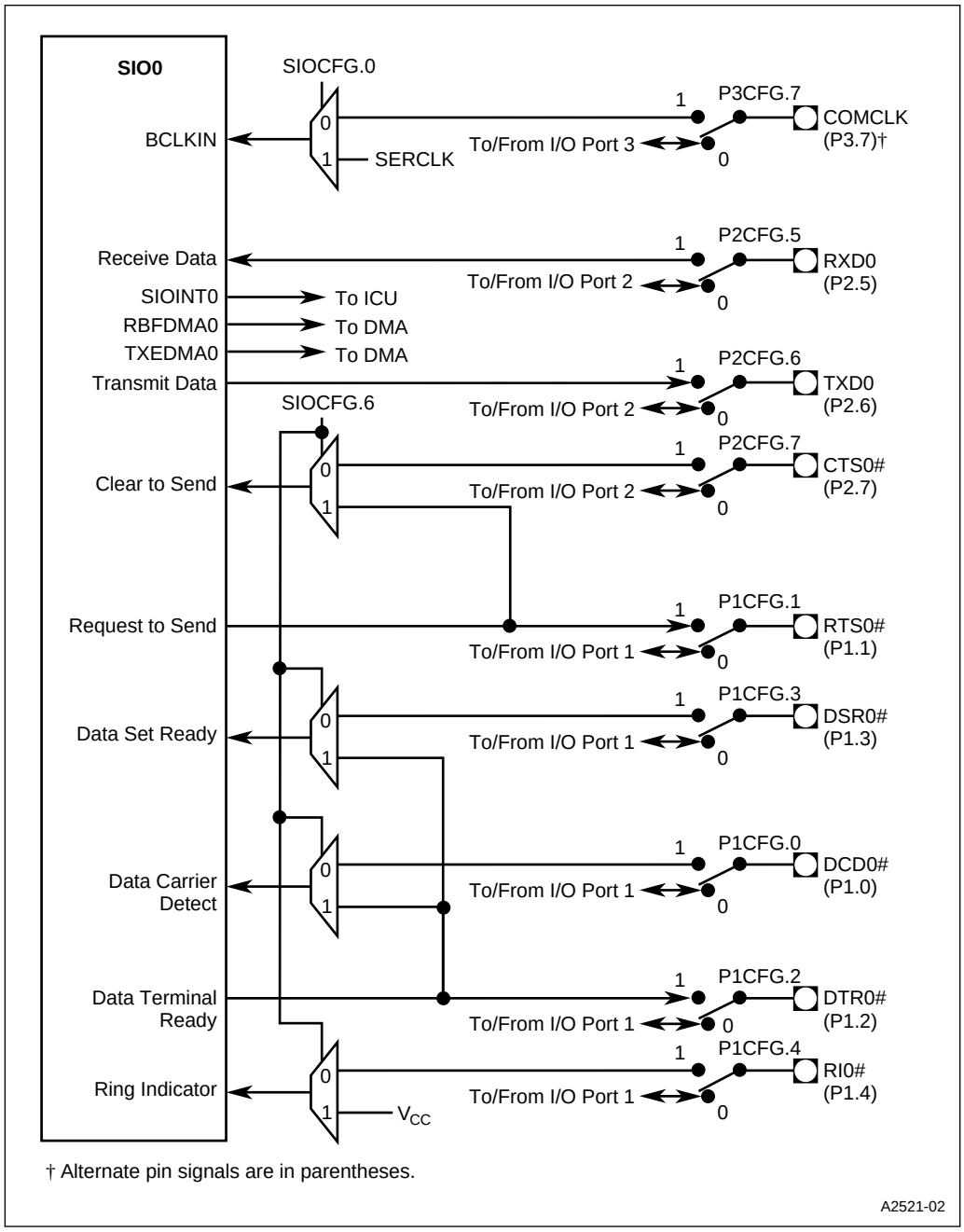


Figure 5-8. Serial I/O Unit 0 Configuration

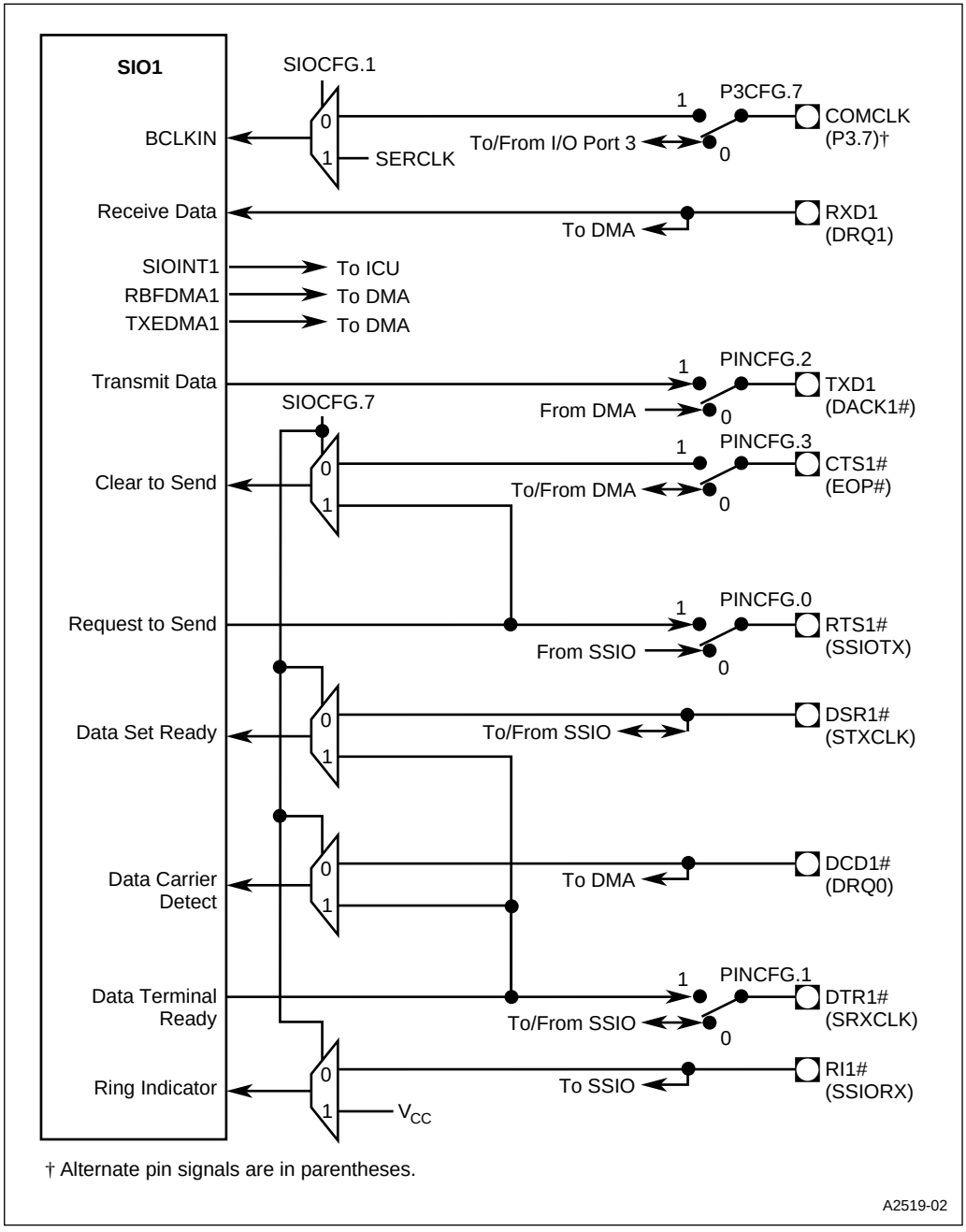


Figure 5-9. Serial I/O Unit 1 Configuration

SIO and SSIO Configuration
SIOCFG
(read/write)

Expanded Addr: **F836H**
ISA Addr: **—**
Reset State: **00H**

7							0
S1M	S0M	—	—	—	SSBSRC	S1BSRC	S0BSRC

Bit Number	Bit Mnemonic	Function
7	S1M	SIO1 Modem Signal Connections: 0 = Connects the SIO1 modem input signals to the package pins. 1 = Connects the SIO1 modem input signals internally.
6	S0M	SIO0 Modem Signal Connections: 0 = Connects the SIO0 modem input signals to the package pins. 1 = Connects the SIO0 modem input signals internally.
5–3	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
2	SSBSRC	SSIO Baud-rate Generator Clock Source: 0 = Connects the internal PSCLK signal to the SSIO baud-rate generator. 1 = Connects the internal SERCLK signal to the SSIO baud-rate generator.
1	S1BSRC	SIO1 Baud-rate Generator Clock Source: 0 = Connects the COMCLK pin to the SIO1 baud-rate generator. 1 = Connects the internal SERCLK signal to the SIO1 baud-rate generator.
0	S0BSRC	SIO0 Baud-rate Generator Clock Source: 0 = Connects the COMCLK pin to the SIO0 baud-rate generator. 1 = Connects the internal SERCLK signal to the SIO0 baud-rate generator.

Figure 5-10. SIO and SSIO Configuration Register (SIOCFG)

5.2.5 Synchronous Serial I/O Configuration

The synchronous serial I/O unit (SSIO) is shown in Figure 5-11. Its single configuration register bit is in the SIOCFG register (Figure 5-10). The transmit buffer empty and receive buffer full signals (SSTBE and SSRBF) go to the DMA unit (Figure 5-2), and an interrupt signal (SSIOINT) goes to the ICU (Figure 5-4). Depending on the settings in the SSIOCON1 register (see Chapter 13), SSIOINT is asserted for one of two conditions: the receive buffer is full or the transmit buffer is empty. Note that using the SSIO signals precludes the use of four of the SIO1 modem signals.

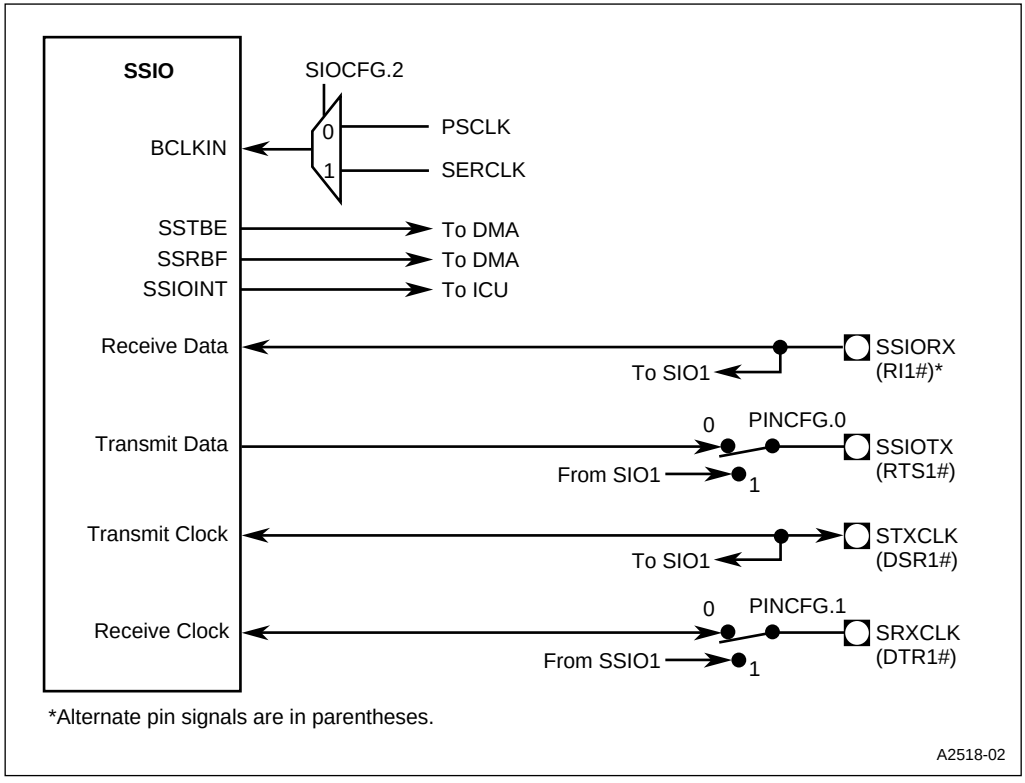


Figure 5-11. SSIO Unit Configuration

5.2.6 Chip-select Unit and Clock and Power Management Unit Configuration

Figure 5-12 shows the multiplexing of signals of the Chip-select Unit and the Clock and Power Management Unit.

The Chip-select signals, CS6# and CS5# are multiplexed with the REFRESH# signal from the Refresh Control Unit and the DACK0# signal from the DMA Unit, respectively. Bits 6 and 4 in the PINCFG register (see Figure 5-15) control these multiplexers. CS3#, CS2#, CS1# and CS0# are multiplexed with I/O Port 2 signals, P2.3, P2.2, P2.1 and P2.0, respectively. Bits 4:0 in the P2CFG register (see Figure 5-17) control these multiplexers.

The PWRDOWN output signal of the Clock and Power Management Unit is multiplexed with I/O Port 3 signal, P3.6. Bit 6 in the P3CFG register (see Figure 5-18) controls this multiplexer.

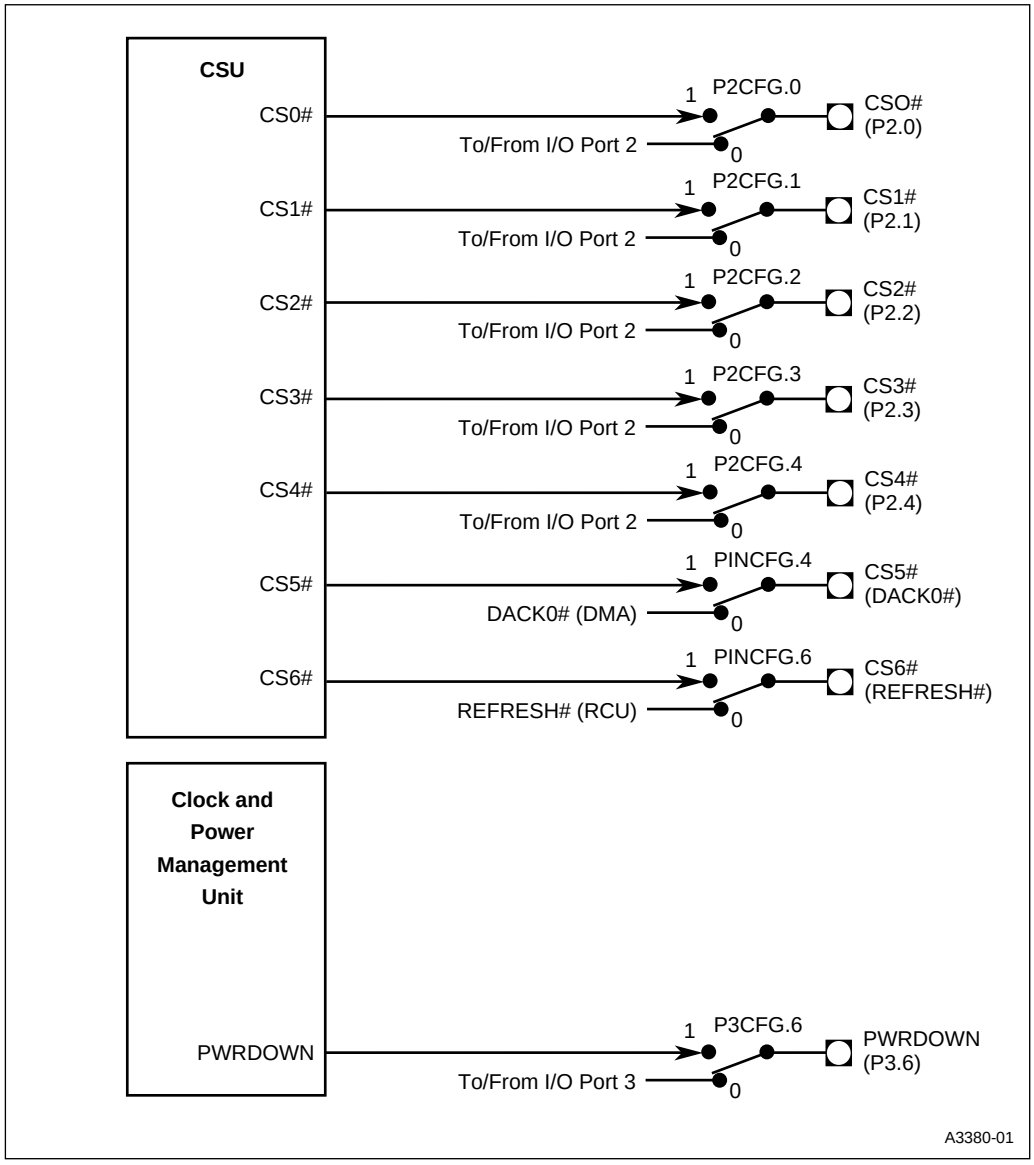


Figure 5-12. Configuration of Chip-select Unit and Clock and Power Management Unit

5.2.7 Core Configuration

Three coprocessor signals (ERROR#, PEREQ, and BUSY# in Figure 5-13) can be routed to the core, as determined by bit 5 of the PINCFG register (see Figure 5-15). Due to signal multiplexing at the pins, the coprocessor and Timer/counter2 cannot be used simultaneously.

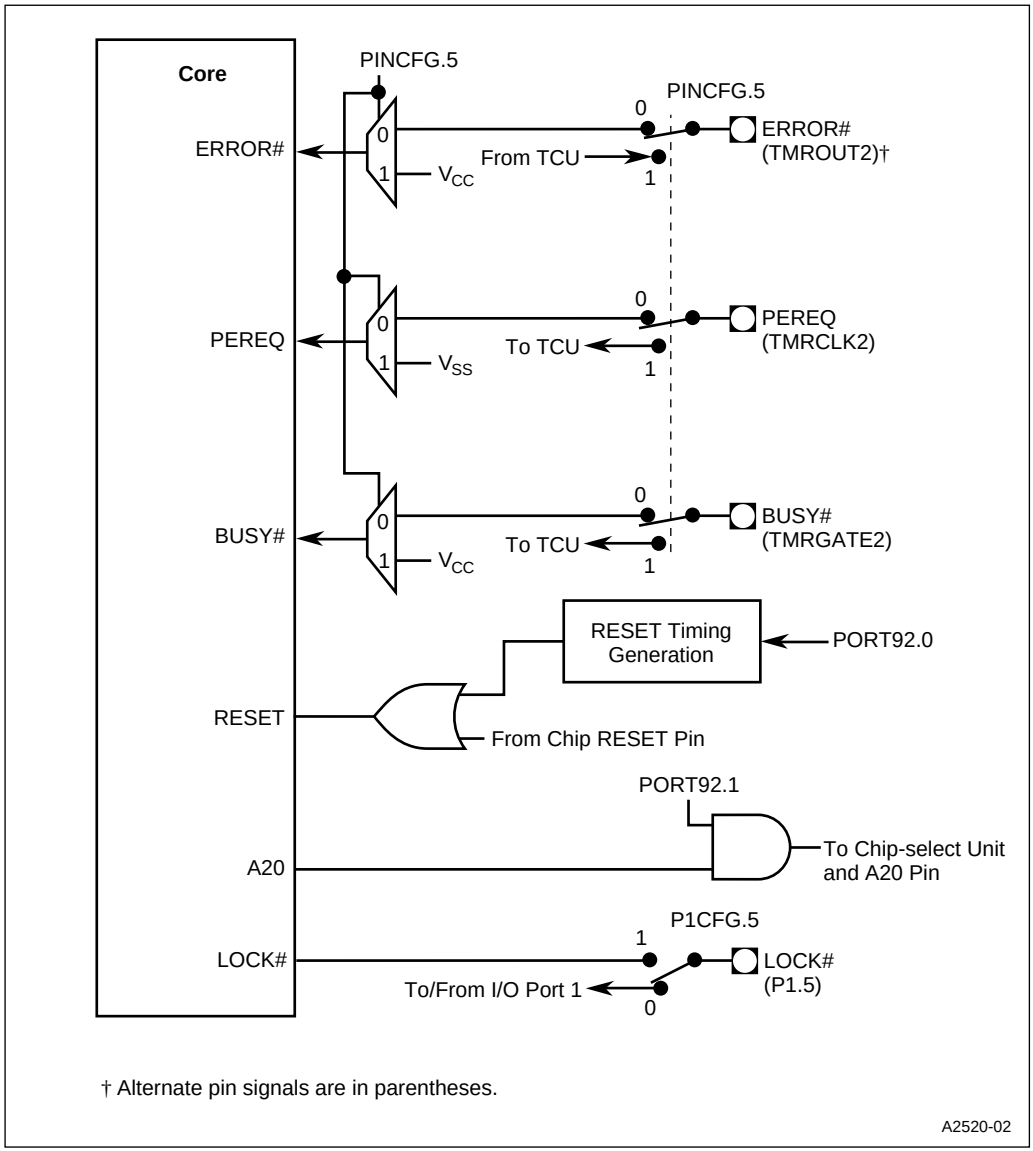


Figure 5-13. Core Configuration

Setting bit 0 in the PORT92 register (see Figure 5-14) resets the core without resetting the peripherals. Unlike the RESET pin, which is asynchronous and can be used to synchronize internal clocks to CLK2, this core-only reset is synchronized with the on-chip clocks and does not affect the on-chip clock synchronization. After the CPU-RESET this bit is still set to 1. It must be cleared and then set to cause another core-only reset.

Clearing bit 1 in the PORT92 register forces address line A20 to 0. This bit affects only addresses generated by the core; addresses generated by the DMA and the refresh control unit are not affected.

Port 92 Configuration
PORT92
(read/write)

Expanded Addr: **F092H**
ISA Addr: **0092H**
Reset State: **XXXXXX10B**

7							**0**
—	—	—	—	—	—	A20G	CPURST

Bit Number	Bit Mnemonic	Function
7–2	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
1	A20G	A20 Grounded: 0 = Clearing this bit forces address line A20 to 0. This bit affects addresses generated only by the core. Addresses generated by the DMA and the Refresh Unit are not affected by this bit. 1 = Setting this bit leaves core-generated addresses unmodified.
0	CPURST	CPU Reset: 0 = Clearing this bit performs no operation. 1 = Setting this bit resets the core without resetting the peripherals. This bit must be cleared before issuing another reset.

Figure 5-14. Port 92 Configuration Register (PORT92)

5.3 PIN CONFIGURATION

Most of the microprocessor's package pins support two peripheral functions. Some of these pins are routed to two peripheral inputs without the use of a multiplexer. These input-signal pairs are listed in Table 5-3. The pin is connected to both peripheral inputs.

The remaining pins supporting two signals have multiplexers. For each such pin, a bit in a pin configuration register enables one of the signals. Table 5-9 lists the bits in each of the four pin configuration registers. These abbreviated register tables are discussed in "Configuration Example" on page 5-28.

When configuring ports to use INT8 or INT9, first set the appropriate INTCFG bit, then the P3CFG bit. Setting the bits in this order avoids any potential contention on INT8 or INT9.

Table 5-3. Signal Pairs on Pins without a Multiplexer

Names	Signal Descriptions
DRQ0/ DCD1#	**DMA External Request 0** indicates that an off-chip peripheral requires DMA service.
	Data Carrier Detect SIO1 indicates that the modem or data set has detected the asynchronous serial channel's data carrier.
DRQ1/ RXD1	**DMA External Request1** indicates that an off-chip peripheral requires DMA service.
	Receive Data SIO1 accepts serial data from the modem or data set to the asynchronous serial channel SIO1.
DSR1#/ STXCLK	**Data Set Ready SIO1** indicates that the modem or data set is ready to establish a communication link with asynchronous serial channel SIO1.
	SSIO Transmit Clock synchronizes data being sent by the synchronous serial port.
RI1#/ SSIORX	**Ring Indicator SIO1** indicates that the modem or data set has received a telephone ringing signal.
	SSIO Receive Serial Data accepts serial data (most-significant bit first) being sent to the synchronous serial port.
TMRCLK0/ INT4	**Timer/Counter0 Clock Input** can serve as an external clock input for timer/counter0. (The timer/counters can also be clocked internally.)
	Interrupt 4 is an undedicated external interrupt.
TMRGATE0/ INT5	**Timer/Counter0 Gate Input** can control timer/counter0's counting (enable, disable, or trigger, depending on the programmed mode).
	Interrupt 5 is an undedicated external interrupt.
TMRCLK1/ INT6	**Timer/Counter1 Clock Input** can serve as an external clock input for timer/counter1. (The timer/counters can also be clocked internally.)
	Interrupt 6 is an undedicated external interrupt.
TMRGATE1/ INT7	**Timer/Counter1 Gate Input** can control timer/counter1's counting (enable, disable, or trigger, depending on the programmed mode).
	Interrupt 7 is an undedicated external interrupt.

Pin Configuration
PINCFG
(read/write)

Expanded Addr: **F826H**
ISA Addr: **—**
Reset State: **00H**

7							0
—	PM6	PM5	PM4	PM3	PM2	PM1	PM0

Bit Number	Bit Mnemonic	Function
7	—	Reserved. This bit is undefined; for compatibility with future devices, do not modify this bit.
6	PM6	Pin Mode: 0 = Selects CS6# at the package pin. 1 = Selects REFRESH# at the package pin.
5	PM5	Pin Mode: 0 = Selects the coprocessor signals, PEREQ, BUSY#, and ERROR#, at the package pins. 1 = Selects the timer control unit signals, TMROUT2, TMRCLK2, and TMRGATE2, at the package pins.
4	PM4	Pin Mode: 0 = Selects DACK0# at the package pin. 1 = Selects CS5# at the package pin.
3	PM3	Pin Mode: 0 = Selects EOP# at the package pin. 1 = Selects CTS1# at the package pin.
2	PM2	Pin Mode: 0 = Selects DACK1# at the package pin. 1 = Selects TXD1 at the package pin.
1	PM1	Pin Mode: 0 = Selects SRXCLK at the package pin. 1 = Selects DTR1# at the package pin.
0	PM0	Pin Mode: 0 = Selects SSIOTX at the package pin. 1 = Selects RTS1# at the package pin.

Figure 5-15. Pin Configuration Register (PINCFG)

Port 1 Configuration
P1CFG
(read/write)

Expanded Addr: **F820H**
ISA Addr: **—**
Reset State: **00H**

7							0
PM7	PM6	PM5	PM4	PM3	PM2	PM1	PM0

Bit Number	Bit Mnemonic	Function
7	PM7	Pin Mode: 0 = Selects P1.7 at the package pin. 1 = Selects HLDA at the package pin.
6	PM6	Pin Mode: 0 = Selects P1.6 at the package pin. 1 = Selects HOLD at the package pin.
5	PM5	Pin Mode: 0 = Selects P1.5 at the package pin. 1 = Selects LOCK# at the package pin.
4	PM4	Pin Mode: 0 = Selects P1.4 at the package pin. 1 = Selects RI0# at the package pin.
3	PM3	Pin Mode: 0 = Selects P1.3 at the package pin. 1 = Selects DSR0# at the package pin.
2	PM2	Pin Mode: 0 = Selects P1.2 at the package pin. 1 = Selects DTR0# at the package pin.
1	PM1	Pin Mode: 0 = Selects P1.1 at the package pin. 1 = Selects RTS0# at the package pin.
0	PM0	Pin Mode: 0 = Selects P1.0 at the package pin. 1 = Selects DCD0# at the package pin.

Figure 5-16. Port 1 Configuration Register (P1CFG)

Port 2 Configuration
P2CFG
(read/write)

Expanded Addr: **F822H**
ISA Addr: **—**
Reset State: **00H**

7							0
PM7	PM6	PM5	PM4	PM3	PM2	PM1	PM0

Bit Number	Bit Mnemonic	Function
7	PM7	Pin Mode: 0 = Selects P2.7 at the package pin. 1 = Selects CTS0# at the package pin.
6	PM6	Pin Mode: 0 = Selects P2.6 at the package pin. 1 = Selects TXD0 at the package pin.
5	PM5	Pin Mode: 0 = Selects P2.5 at the package pin. 1 = Selects RXD0 at the package pin.
4	PM4	Pin Mode: 0 = Selects P2.4 at the package pin. 1 = Selects CS4# at the package pin.
3	PM3	Pin Mode: 0 = Selects P2.3 at the package pin. 1 = Selects CS3# at the package pin.
2	PM2	Pin Mode: 0 = Selects P2.2 at the package pin. 1 = Selects CS2# at the package pin.
1	PM1	Pin Mode: 0 = Selects P2.1 at the package pin. 1 = Selects CS1# at the package pin.
0	PM0	Pin Mode: 0 = Selects P2.0 at the package pin. 1 = Selects CS0# at the package pin.

Figure 5-17. Port 2 Configuration Register (P2CFG)

Port 3 Configuration
P3CFG
(read/write)

Expanded Addr: **F824H**
ISA Addr: **—**
Reset State: **00H**

7							0
PM7	PM6	PM5	PM4	PM3	PM2	PM1	PM0

Bit Number	Bit Mnemonic	Function
7	PM7	Pin Mode: 0 = Selects P3.7 at the package pin. 1 = Selects COMCLK at the package pin.
6	PM6	Pin Mode: 0 = Selects P3.6 at the package pin. 1 = Selects PWRDOWN at the package pin.
5	PM5	Pin Mode: 0 = Selects P3.5 at the package pin. 1 = Connects master IR7 to the package pin (INT3).
4	PM4	Pin Mode: 0 = Selects P3.4 at the package pin. 1 = Connects master IR6 to the package pin (INT2).
3	PM3	Pin Mode: 0 = Selects P3.3 at the package pin. 1 = Connects master IR5 to the package pin (INT1).
2	PM2	Pin Mode: 0 = Selects P3.2 at the package pin. 1 = Connects master IR1 to the package pin (INT0).
1	PM1	Pin Mode: See Table 5-1 on page 5-8 for all the PM1 configuration options.
0	PM0	Pin Mode: See Table 5-1 on page 5-8 for all the PM0 configuration options.

Figure 5-18. Port 3 Configuration Register (P3CFG)

5.4 DEVICE CONFIGURATION PROCEDURE

Before configuring the microprocessor, make the following selections:

- The set of peripherals to be used
- The signals to be available at the package pins
- The desired peripheral-peripheral and peripheral-core connections

Although final decisions regarding these selections may be influenced by the possible configurations, we recommend that you initially make the selections without regard to limitations on the configurations.

We suggest the following procedure for configuring the device for your design. An aide for recording the steps in the procedure and an example configuration are given in "Configuration Example" on page 5-28.

1. **Pin Configuration.** For each desired pin signal, consult the peripheral configuration diagram to find the bit value in the pin configuration register that connects the signal to a device pin. When the signal shares a pin that has no multiplexer, make a note of its companion signal.
2. **Peripheral Configuration.** For each peripheral in your design, consult the peripheral configuration diagram and the peripheral configuration register to find the bit values for your desired internal connections.
3. **Configuration Review.** Review the results of steps 1 and 2 to see if the configuration registers have conflicting bit values. If conflicts exist, follow steps 3a and 3b.
   a. Attempt to resolve the pin configuration conflicts first. In some cases you may find that using a different peripheral channel resolves the conflict.
   b. Attempt to resolve peripheral configuration conflicts.

If conflicts remain, consider peripheral substitutions.

5.5 CONFIGURATION EXAMPLE

This section presents an example of a PC/AT*-compatible configuration. The last set of tables are blank; you can use them as worksheets as you follow the steps in the configuration process.

5.5.1 Example Design Requirements

The example is a PC/AT-compatible design with the following requirements:

- Interrupt Control Unit:
  - — External interrupt inputs available at package pins: INT1:0, INT7:4
- Timer Control Unit:
  - — Counters 0, 1: Clock input is on-chip programmable clock (PSCLK); no signals connected externally.

  - Counter 2: Clock input is on-chip programmable clock (PSCLK); no signals connected to package pins
- DMA Unit:
  - Not Used
- Asynchronous Serial I/O channel 0 (SIO0):
  - Clock input is the internal clock SERCLK
  - RXD0, TXD0 connected to package pins
  - Modem Signals connected internally.
- Asynchronous Serial I/O channel 1 (SIO1):
  - Clock input is the internal clock SERCLK
  - Modem signals externally connected
- Synchronous Serial I/O (SSIO):
  - Not Used
- Chip Select:
  - Chip select signals CS6#, CS5:1#, UCS# connected to package pins
- Core and Bus Arbiter:
  - Coprocessor signals connected to package pins
  - HOLD and HLDA not connected to package pins
  - LOCK# and PWRDOWN not connected to package pins

5.5.2 Example Design Solution

The configuration register bit values for the example design are recorded in the following abbreviated register tables. Blank worksheets are provided for you to use when designing your system.

Table 5-4 summarizes the bit selections you would need to make in the pin configuration registers to implement the example design. Tables 5-5 through 5-8 summarize the bit selections you would make in the peripheral configuration registers.

Bit #	P1CFG	Value
7	0 = P1.7	**0**
	1 = HLDA	
6	0 = P1.6	**0**
	1 = HOLD	
5	0 = P1.5	**0**
	1 = LOCK#	
4	0 = P1.4	**0**
	1 = RIO#	
3	0 = P1.3	**0**
	1 = DSR0#	
2	0 = P1.2	**0**
	1 = DTR0#	
1	0 = P1.1	**0**
	1 = RTS0#	
0	0 = P1.0	**0**
	1 = DCD0#	

Bit #	P2CFG	Value
7	0 = P2.7	**0**
	1 = CTS0#	
6	0 = P2.6	**1**
	1 = TXD0	
5	0 = P2.5	**1**
	1 = RXD0	
4	0 = P2.4	**1**
	1 = CS4#	
3	0 = P2.3	**1**
	1 = CS3#	
2	0 = P2.2	**1**
	1 = CS2#	
1	0 = P2.1	**1**
	1 = CS1#	
0	0 = P2.0	**0**
	1 = CS0#	

Bit #	P3CFG	Value
7	0 = P3.7	**0**
	1 = COMCLK	
6	0 = P3.6	**0**
	1 = PWRDOWN	
5	0 = P3.5	**0**
	1 = INT3	
4	0 = P3.4	**0**
	1 = INT2	
3	0 = P3.3	**1**
	1 = INT1	
2	0 = P3.2	**1**
	1 = INT0	
1	0 = P3.1	**0**
	1 = mux	
0	0 = P3.0	**0**
	1 = mux	

Bit #	PINCFG	Value
7	Reserved	**R**
6	0 = CS6#	**0**
	1 = REFRESH#	
5	0 = Coprocessor Sigs.[1]	**0**
	1 = TMR2 Signals[2]	
4	0 = DACK0#	**1**
	1 = CS5#	
3	0 = EOP#	**1**
	1 = CTS1#	
2	0 = DACK1#	**1**
	1 = TXD1	
1	0 = SRXCLK	**1**
	1 = DTR1#	
0	0 = SSIOTX	**1**
	1 = RTS1#	

Pins w/o Muxes	X
DRQ0	**X**
DCD1#	
DRQ1	
RXD1	**X**
DSR1#	
STXCLK	**X**
RI1#	
SSIORX	**X**

Pins w/o Muxes	X
TMRCLK0	
INT4	**X**
TMRGATE0	
INT5	**X**
TMRCLK1	
INT6	**X**
TMRGATE1	
INT7	**X**

NOTES:

[1] PEREQ, BUSY#, ERROR#

[2] TMROUT2, TMRCLK2, TMRGATE2

Table 5-4. Example Pin Configuration Registers

Bit #	DMACFG	Value
7	0 = Enables DACK1# at chip pin	**1**
	1 = Disables DACK1# at chip pin	
6–4	000 = DRQ1 pin (external peripheral) connected to DREQ1	**000**
	001 = SIO channel 1's receive buffer full signal (RBFDMA1) connected to DREQ1	
	010 = SIO channel 0's transmit buffer empty signal (TXEDMA0) to DREQ1	
	011 =SSIO receive holding buffer full signal (SSRBF) to DREQ1	
	100 = TCU counter 2's output signal (OUT2) to DREQ1	
	101 = SIO channel 0's receive buffer full signal (RBFDMA0) to DREQ1	
	110 = SIO channel 1's transmit buffer empty signal (TXEDMA1) to DREQ1	
	111 = SSIO transmit holding buffer empty signal (SSTBE) to DREQ1	
3	0 = Enables DACK0# at chip pin	**1**
	1 = Disables DACK0# at chip pin	
2–0	000 = DRQ0 pin (external peripheral) connected to DREQ0	**000**
	001 = SIO channel 0's receive buffer full signal (RBFDMA0) connected to DREQ0	
	010 = SIO channel 1's transmit buffer empty signal (TXEDMA1) connected to DREQ0	
	011 = SSIO transmit holding buffer empty signal (THBE) connected to DREQ0	
	100 = TCU counter 1's output signal (OUT1) connected to DREQ0	
	101 = SIO channel 1's receive buffer full signal (RBFDMA1) connected to DREQ0	
	110 = SIO channel 0's transmit buffer empty signal (TXEDMA0) connected to DREQ0	
	111 = SSIO receive holding buffer full signal (RHBF) connected to DREQ0	

Table 5-5. Example DMACFG Configuration Register

Bit #	TMRCFG	Value
7	0 = All clock inputs enabled	**0**
	1 = CLK2, CLK1, CLK0 forced to 0	
6	0 = Connects GATE*n* to either the V_{CC} pin or the TMRGATE*n* pin	**0**
	1 = Turns GATEn on or off, depending on whether bits 1, 3, and 5 are set or clear	
5	0 = With bit 6 clear: V_{CC} to GATE2; with bit 6 set: GATE2 off.	**0**
	1 = With bit 6 clear: TMRGATE2 pin conn. to GATE2; with bit 6 set: GATE2 on.	
4	0 = PSCLK connected to CLK2	**0**
	1 = TMRCLK2 connected to CLK2	
3	0 = With bit 6 clear: V_{CC} to GATE1; with bit 6 set: GATE1 turned off.	**0**
	1 = With bit 6 clear: TMRGATE1 pin conn. to GATE1; with bit 6 set: GATE1 on.	
2	0 = PSCLK connected to CLK1	**0**
	1 = TMRCLK1 connected to CLK1	
1	0 = With bit 6 clear: V_{CC} to GATE0; with bit 6 set: GATE0 turned off.	**0**
	1 = With bit 6 clear: TMRGATE0 pin conn. to GATE0; with bit 6 set: GATE0 on.	
0	0 = PSCLK connected to CLK0	**0**
	1 = TMRCLK0 connected to CLK0	

Table 5-6. Example TMRCFG Configuration Register

Bit #	INTCFG	Value
7	0 = CAS2:0 disabled to pins	**0**
	1 = CAS2:0 enabled from pins	
6	0 = SIOINT1 connected to master IR3	**0**
	1 = P3.1 connected to IR3	
5	0 = SIOINT0 connected to master IR4	**0**
	1 = P3.0 connected to IR4	
4	0 = DMAINT connected to slave IR4. INT6 connected to slave IR5.	**1**
	1 = INT6 connected to slave IR4. DMAINT connected to slave IR5.	
3	0 = VSS connected to slave IR6	**1**
	1 = INT7 connected to slave IR6	
2	0 = V_{SS} connected to slave IR5	**1**
	1 = INT6 connected to slave IR5	
1	0 = SSIO Interrupt to slave IR1	**1**
	1 = INT5 connected to slave IR1	
0	0 = VSS connected to slave IR0	**1**
	1 = INT4 connected to slave IR0	

Table 5-7. Example INTCFG Configuration Register

SIOCFG		
7	0 = SIO1 modem sigs. conn. to pin muxes	**1**
	1 = SIO1 modem signals internal	
6	0 = SIO0 modem sigs. conn. to pin muxes	**0**
	1 = SIO0 modem signals internal	
5–3	Reserved	**R**
2	0 = PSCLK connected to SSIO BLKIN	**1**
	1 = SERCLK connected to SSIO BCLKIN	
1	0 = COMCLK connected to SIO1 BCLKIN	**0**
	1 = SERCLK connected to SIO1 BCLKIN	
0	0 = COMCLK connected to SIO0 BCLKIN	**0**
	1 = SERCLK connected to SIO0 BCLKIN	

Table 5-8. Example SIOCFG Configuration Register

Bit #	P1CFG	Value
7	0 = P1.7	
	1 = HLDA	
6	0 = P1.6	
	1 = HOLD	
5	0 = P1.5	
	1 = LOCK#	
4	0 = P1.4	
	1 = RIO#	
3	0 = P1.3	
	1 = DSR0#	
2	0 = P1.2	
	1 = DTR0#	
1	0 = P1.1	
	1 = RTS0#	
0	0 = P1.0	
	1 = DCD0#	

Bit #	P2CFG	Value
7	0 = P2.7	
	1 = CTS0#	
6	0 = P2.6	
	1 = TXD0	
5	0 = P2.5	
	1 = RXD0	
4	0 = P2.4	
	1 = CS4#	
3	0 = P2.3	
	1 = CS3#	
2	0 = P2.2	
	1 = CS2#	
1	0 = P2.1	
	1 = CS1#	
0	0 = P2.0	
	1 = CS0#	

Bit #	P3CFG	Value
7	0 = P3.7	
	1 = COMCLK	
6	0 = P3.6	
	1 = PWRDOWN	
5	0 = P3.5	
	1 = INT3	
4	0 = P3.4	
	1 = INT2	
3	0 = P3.3	
	1 = INT1	
2	0 = P3.2	
	1 = INT0	
1	0 = P3.1	
	1 = mux	
0	0 = P3.0	
	1 = mux	

Bit #	PINCFG	Value
7	Reserved	
6	0 = CS6#	
	1 = REFRESH#	
5	0 = Coprocessor Sigs.[1]	
	1 = TMR2 Signals[2]	
4	0 = DACK0#	
	1 = CS5#	
3	0 = EOP#	
	1 = CTS1#	
2	0 = DACK1#	
	1 = TXD1	
1	0 = SRXCLK	
	1 = DTR1#	
0	0 = SSIOTX	
	1 = RTS1#	

Pins w/o Muxes	X
DRQ0	
DCD1#	
DRQ1	
RXD1	
DSR1#	
STXCLK	
RI1#	
SSIORX	

Pins w/o Muxes	X
TMRCLK0	
INT4	
TMRGATE0	
INT5	
TMRCLK1	
INT6	
TMRGATE1	
INT7	

NOTES:

[1] PEREQ, BUSY#, ERROR#

[2] TMROUT2, TMRCLK2, TMRGATE2

Table 5-9. Pin Configuration Register Design Woksheet

Bit #	DMACFG	Value
7	0 = Enables DACK1# at chip pin	
	1 = Disables DACK1# at chip pin	
6–4	000 = DRQ1 pin (external peripheral) connected to DREQ1	
	001 = SIO channel 1's receive buffer full signal (RBFDMA1) connected to DREQ1	
	010 = SIO channel 0's transmit buffer empty signal (TXEDMA0) to DREQ1	
	011 =SSIO receive holding buffer full signal (SSRBF) to DREQ1	
	100 = TCU counter 2's output signal (OUT2) to DREQ1	
	101 = SIO channel 0's receive buffer full signal (RBFDMA0) to DREQ1	
	110 = SIO channel 1's transmit buffer empty signal (TXEDMA1) to DREQ1	
	111 = SSIO transmit holding buffer empty signal (SSTBE) to DREQ1	
3	0 = Enables DACK0# at chip pin	
	1 = Disables DACK0# at chip pin	
2–0	000 = DRQ0 pin (external peripheral) connected to DREQ0	
	001 = SIO channel 0's receive buffer full signal (RBFDMA0) connected to DREQ0	
	010 = SIO channel 1's transmit buffer empty signal (TXEDMA1) connected to DREQ0	
	011 = SSIO transmit holding buffer empty signal (THBE) connected to DREQ0	
	100 = TCU counter 1's output signal (OUT1) connected to DREQ0	
	101 = SIO channel 1's receive buffer full signal (RBFDMA1) connected to DREQ0	
	110 = SIO channel 0's transmit buffer empty signal (TXEDMA0) connected to DREQ0	
	111 = SSIO receive holding buffer full signal (RHBF) connected to DREQ0	

Table 5-10. DMACFG Register Design Worksheet

Bit #	TMRCFG	Value
7	0 = All clock inputs enabled	
	1 = CLK2, CLK1, CLK0 forced to 0	
6	0 = Connects GATE*n* to either the V_{CC} pin or the TMRGATE*n* pin.	
	1 = Turns GATEn on or off, depending on whether bits 1, 3, and 5 are set or clear.	
5	0 = With bit 6 clear: V_{CC} to GATE2; with bit 6 set: GATE2 off.	
	1 = With bit 6 clear: TMRGATE2 pin conn. to GATE2; with bit 6 set: GATE2 on.	
4	0 = PSCLK connected to CLK2	
	1 = TMRCLK2 connected to CLK2	
3	0 = With bit 6 clear: V_{CC} to GATE1; with bit 6 set: GATE1 turned off.	
	1 = With bit 6 clear: TMRGATE1 pin conn. to GATE1; with bit 6 set: GATE1 on.	
2	0 = PSCLK connected to CLK1	
	1 = TMRCLK1 connected to CLK1	
1	0 = With bit 6 clear: V_{CC} to GATE0; with bit 6 set: GATE0 turned off.	
	1 = With bit 6 clear: TMRGATE0 pin conn. to GATE0; with bit 6 set: GATE0 on.	
0	0 = PSCLK connected to CLK0	
	1 = TMRCLK0 connected to CLK0	

Table 5-11. TMRCFG Register Design Worksheet

Bit #	INTCFG	Value
7	0 = CAS2:0 disabled to pins	
	1 = CAS2:0 enabled from pins	
6	0 = SIOINT1 connected to master IR3	
	1 = P3.1 connected to IR3	
5	0 = SIOINT0 connected to master IR4	
	1 = P3.0 connected to IR4	
4	0 = DMAINT connected to slave IR4. INT6 connected to slave IR5.	
	1 = INT6 connected to slave IR4. DMAINT connected to slave IR5.	
3	0 = VSS connected to slave IR6	
	1 = INT7 connected to slave IR6	
2	0 = V_{SS} connected to slave IR5	
	1 = INT6 connected to slave IR5	
1	0 = SSIO Interrupt to slave IR1	
	1 = INT5 connected to slave IR1	
0	0 = VSS connected to slave IR0	
	1 = INT4 connected to slave IR0	

Table 5-12. INTCFG Register Design Worksheet

SIOCFG		
7	0 = SIO1 modem sigs. conn. to pin muxes	
	1 = SIO1 modem signals internal	
6	0 = SIO0 modem sigs. conn. to pin muxes	
	1 = SIO0 modem signals internal	
5–3	Reserved	
2	0 = PSCLK connected to SSIO BLKIN	
	1 = SERCLK connected to SSIO BCLKIN	
1	0 = COMCLK connected to SIO1 BCLKIN	
	1 = SERCLK connected to SIO1 BCLKIN	
0	0 = COMCLK connected to SIO0 BCLKIN	
	1 = SERCLK connected to SIO0 BCLKIN	

Table 5-13. SIOCFG Register Design Worksheet

6

BUS INTERFACE UNIT

CHAPTER 6
BUS INTERFACE UNIT

The processor communicates with memory, I/O, and other devices through bus operations. Address, data, status, and control information define a bus cycle. The Bus Interface Unit supports read and write cycles to external memory and I/O devices. It also contains the signals that allow external bus masters to request and acquire control of the bus. The Bus Interface Unit (BIU) can execute memory read/write cycles, I/O read/write cycles, interrupt acknowledge cycles, refresh cycles and processor halt/shutdown cycles.

This chapter is organized as follows:

6.1 OVERVIEW

The Intel386™ EX processor's external bus is controlled by the bus interface unit (BIU). To communicate with memory and I/O, the external bus consists of a data bus, a separate address bus, seven bus status pins, two data status pins, and three control pins.

- Bidirectional data bus (D15:0) can transfer 8 or 16 bits of data per cycle.
- Address bus includes the address pins (A25:1), a high-byte-enable pin (BHE#), and a low-byte-enable pin (BLE#). Address pins select a word in memory, and byte-enable pins select the byte within the word to access.
- Bus status pins include:
  - — ADS# indicates the start of a bus cycle and valid address bus outputs.
  - — W/R# identifies the bus cycle as a write or a read.
  - — M/IO# identifies the bus cycle as a memory or I/O access.
  - — D/C# identifies the bus cycle as a data or control cycle.
  - — LOCK# identifies a locked bus cycle.
  - — LBA# indicates that the processor generates an internal READY# for the current bus cycle.
  - — REFRESH# identifies a refresh bus cycle.

- Data status pins indicate that data is available on the data bus for a write (WR#) or that the processor is ready to accept data for a read (RD#). These pins are available so that certain system configurations can easily connect the processor directly to memory or I/O without external logic.
- Bus control pins allow external logic to control the bus cycle on a cycle-by-cycle basis:
  - READY# indicates that internal logic has completed the current bus cycle or that external hardware has terminated it.
  - NA# requests the next address to be put on the bus during a pipelined bus cycle.
  - BS8# indicates that the current bus transaction is for an 8-bit data bus.

The remaining external bus pins interface to external bus masters and external logic for transferring control of the bus.

- An external bus master activates the HOLD pin to request the external bus.
  - The internal bus arbiter arbitrates between the HOLD input and other potential requests (DMA Units 0 and 1, Refresh Control Unit) based on their priorities.
  - When another unit has control of the bus, the bus is released to the external bus master based on the arbiter's arbitration scheme (refer to "Bus Control Arbitration" on page 12-9 for information on internal bus masters also controlled by the internal bus arbiter and the arbitration protocol used by the arbiter).
  - When the core has control of the bus, the arbiter passes the request on to the core by asserting the core HOLD signal.
  - The core finishes the current nonlocked bus transfer and releases the bus signals.
  - The core asserts the core HLDA signal to indicate that the bus has been released.
  - The arbiter then asserts the HLDA pin to indicate to the external bus master that the bus has been released.

6.1.1 Bus Signal Descriptions

Table 6-1 describes the signals associated with the BIU.

Table 6-1. Bus Interface Unit Signals (Sheet 1 of 2)

Signal	Device Pin or Internal Signal only	Description
A25:1	Device pins	Address Bus: Outputs physical memory or I/O addresses. These signals are valid when ADS# is active and remain valid until the next T1, T2P, or Ti.
ADS#	Device pin	Address Strobe: Indicates that the processor is driving a valid bus-cycle definition and address. (The processor is driving W/R#, D/C#, M/IO#, WR#, RD#, UCS#, CS6:0#, LOCK#, REFRESH#, A25:1, BHE#, and BLE# on its pins.)
BHE# BLE#	Device pins	Byte Enable Outputs: Indicates which byte of the 16-bit data bus of the processor is being transferred. **BHE# BLE# Output** 0 0 word transfer 0 1 upper byte (D15:8) transfer 1 0 lower byte (D7:0) transfer 1 1 refresh cycle
BS8#	Device pin	Bus Size: Indicates that the currently addressed device is an 8-bit device.
D15:0	Device pins	Data Bus: Inputs data during memory read, I/O read, and interrupt acknowledge cycles; outputs data during memory write and I/O write cycles. During reads, data is latched at the falling edge of phase 2 (coincides with rising edge of PH1) of T2, T2P, or T2i when READY# is sampled active. During writes, this bus is driven during phase 2 of T1 and T1P and remains active until phase 2 of the next T1, T1P, or Ti.
LBA#	Device pin	Local Bus Access: Indicates that the processor provides the READY# signal internally to terminate a bus transaction. This signal is active when the processor accesses an internal peripheral or when the chip-select unit generates the READY# signal for accesses to an external peripheral. LBA# is also active when internal READY# generation is enabled for Halt/Shutdown cycles and the Watchdog Timer Unit's Bus Monitor Mode timeouts. The LBA# signal goes active in the first T2 state and stays active until the first T2, T2i or T2P state of the next cycle that does not have internal READY# generation.
LOCK#	Device pin	Bus Lock: Prevents other bus masters from gaining control of the system bus.

Table 6-1. Bus Interface Unit Signals (Sheet 2 of 2)

Signal	Device Pin or Internal Signal only	Description
M/IO# D/C# W/R# REFRESH#	Device pins	Bus Cycle Definition Signals (Memory/IO, Data/Control, Write/Read, and Refresh): These four status outputs define the current bus cycle type, as shown in Table 6-2.
NA#	Device pin	Next Address: Requests address pipelining.
RD#	Device pin	Read Enable: Indicates that the current bus cycle is a read cycle and the data bus is able to accept data.
READY#	Device pin	Ready: This bidirectional pin is used to terminate the current bus cycle. The processor drives READY# when LBA# is active. The processor samples the READY# pin at the falling edge of Phase 2 of T2, T2P or T2i. The READY# signal is also used to deassert the WR# signal (Refer to "Write Cycle" on page 6-16).
WR#	Device pin	Write Enable: Indicates that the current bus cycle is a write cycle and valid data is present on the data bus.

6.2 BUS OPERATION

The processor generates eight different types of bus operations:

- Memory data read (data fetch)
- Memory data write
- Memory code read (instruction fetch)
- I/O data read (data fetch)
- I/O data write
- Halt or shutdown
- Refresh
- Interrupt acknowledge

These operations are defined by the states of four bus status pins: M/IO#, D/C#, W/R# and REFRESH#. Table 6-2 lists the various combinations and their definitions.

Table 6-2. Bus Status Definitions

M/IO#	D/C#	W/R#	REFRESH#	Bus Operation
0	0	0	1	interrupt acknowledge cycle
0	0	1	1	never occurs
0	1	0	1	I/O data read
0	1	1	1	I/O data write
1	0	0	1	memory code read
1	0	1	1	halt or shutdown cycle*
1	1	0	0	refresh cycle
1	1	0	1	memory data read
1	1	1	1	memory data write

*The byte address is 2 for a halt and 0 for a shutdown. For both conditions, BHE# is high and BLE# is low.

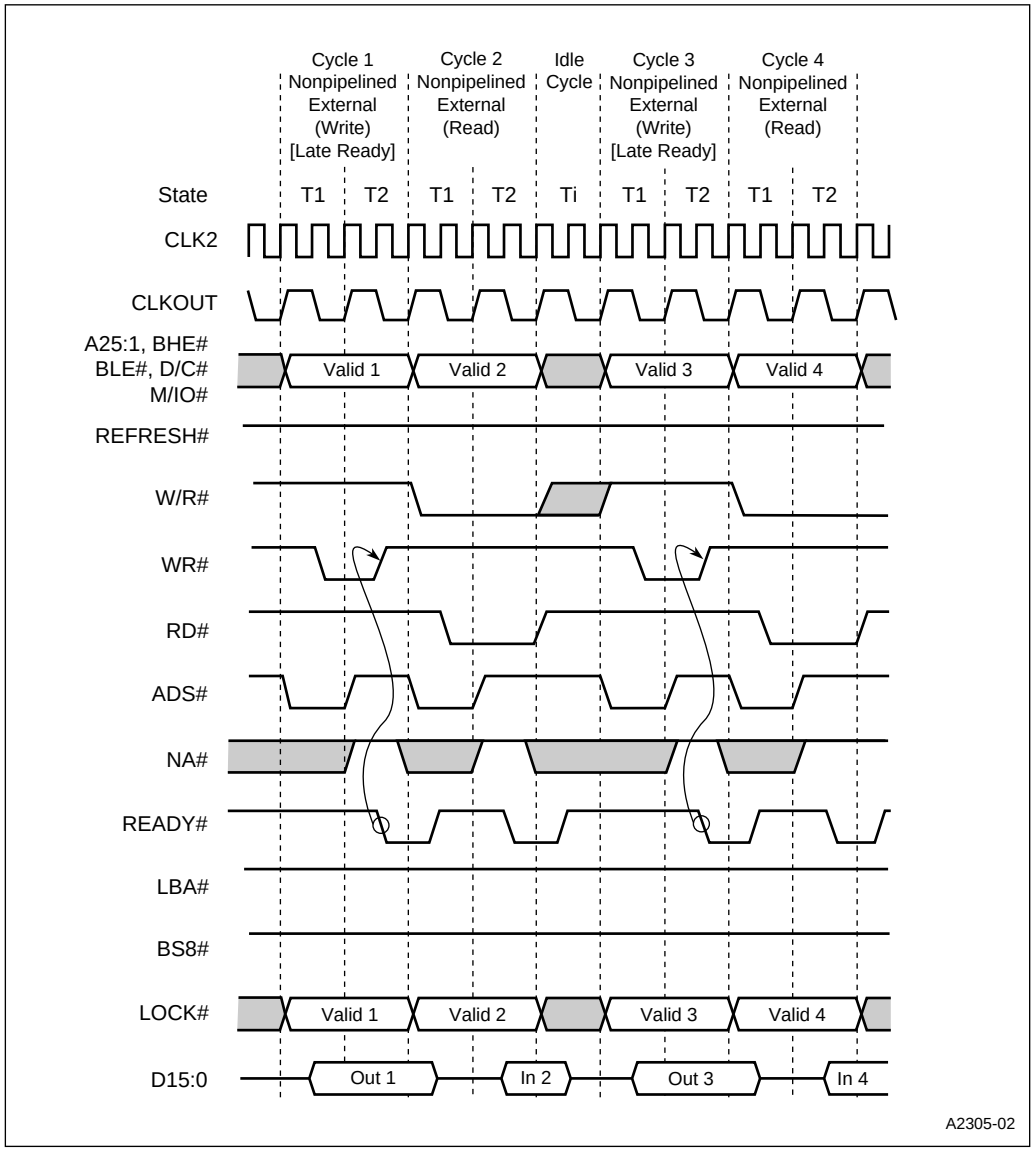


Figure 6-1. Basic External Bus Cycles

6.2.1 Bus States

The processor uses a double-frequency clock input (CLK2). This clock is internally divided by two and synchronized to the falling edge of RESET (see Figure 8-2 in Chapter 8) to generate the internal processor clock signal. Each processor clock cycle is two CLK2 cycles wide.

Each bus cycle is composed of at least two bus states: T1 and T2. Each bus state in turn consists of two CLK2 cycles, which can be thought of as Phase 1 (PH1) and Phase 2 (PH2) of the bus state.

External circuitry can use the CLKOUT signal (generated by the processor) to synchronize itself with the processor. This signal is a replica of the PH1P clock, which is the PH1 clock that is used by the internal peripherals. (For more information, refer to Chapter 8, "CLOCK AND POWER MANAGEMENT UNIT.") The CLKOUT signal is used as a phase status indicator for external circuitry. All device inputs are sampled and outputs are activated at CLK2 rising edges. This makes synchronous circuit design easy through the use of rising-edge-triggered, registered logic (such as PALs, PLDs and EPLDs).

Many signals are sampled by the processor on every other CLK2 rising edge: some are sampled on the CLK2 edge when CLKOUT is going high, while others are sampled on the CLK2 edge when PH1 is going low.

The maximum data transfer rate for a bus operation is 16 bits for every two processor clock cycles (two CLKOUT cycles).

During the first bus state (T1), address and bus status pins go active. During the second bus state (T2), external logic and devices respond.

- When the READY# input is sampled low at the falling edge of PH2 in T2, the bus cycle terminates.
- When READY# is high when sampled, the bus cycle continues for an additional T2 state, called a wait-state, and READY# is sampled again. This process continues until READY# is sampled active, at which point the bus cycle terminates.

Wait-states are added until READY# is sampled low. READY# is sampled externally when the LBA# signal is inactive. When the LBA# signal is active, the processor is generating the READY# signal internally. READY# can be generated internally by either an internal peripheral or the chip-select unit's wait-state generator. When no bus cycles are needed (no bus requests are pending), the processor remains in the idle bus state, Ti. The relationship between T1, T2, and Ti is shown in Figure 6-2.

From an idle bus, the processor begins a bus cycle by first driving a valid address and bus cycle status onto the address and status buses. Hardware can distinguish the difference between an idle cycle and an active bus cycle by the address strobe (ADS#) signal being driven active. The ADS# signal remains active for only the first T-state of the bus cycle, while the address signals and status signals remain active until the bus cycle is terminated by an active READY# signal or the bus cycle is pipelined. Pipelined bus cycles are discussed in "Pipelining" on page 6-8. Basic bus cycles are illustrated in Figure 6-1. The bus status signals indicate the type of bus cycle the processor is executing. Notice that the signal combinations marked as invalid states may occur when the bus is idle and ADS# is inactive.

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

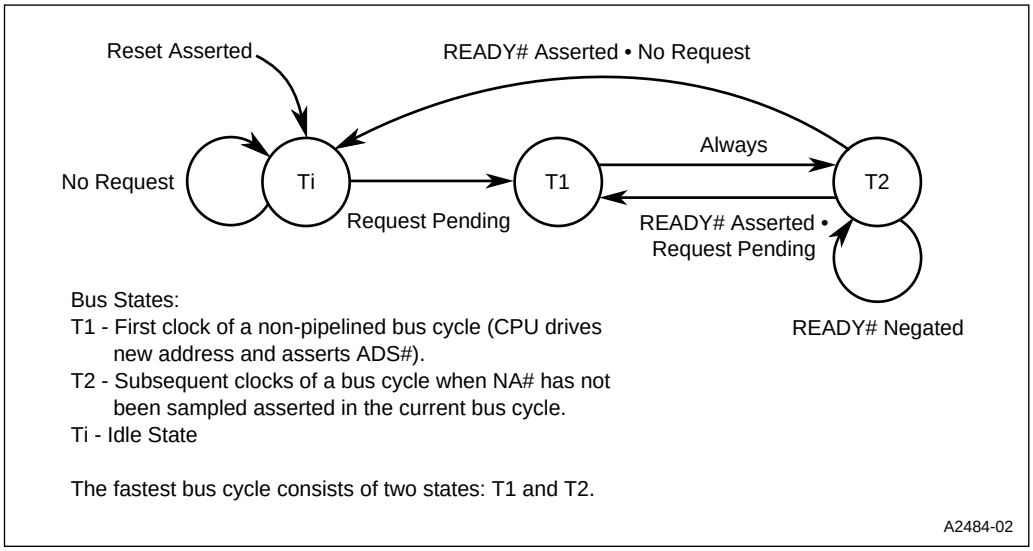


Figure 6-2. Simplified Bus State Diagram (Does Not Include Address Pipelining or Hold states)

6.2.2 Pipelining

The processor can control the address and status outputs so that the outputs for the next bus cycle become valid before the end of the present bus cycle. This technique, allowing bus cycles to overlap, is called *pipelining.*

Pipelining increases bus throughput without decreasing allowable memory or I/O access time, thus allowing high bandwidth with relatively slow, inexpensive components. In addition, using pipelining to address slower devices can yield the same throughput as addressing faster devices with no pipelining. With pipelining, a device operating at 33 MHz (CLK2 = 66 MHz) can transfer data at 33 Mbytes per second while requiring a device with access time of approximately 3 T-states (90 ns at 33 MHz, neglecting signal delays). Without address pipelining, the access time has to be approximately 2 T-states (60 ns at 25 MHz). Therefore, when pipelining is used, slower devices can be used in the system to achieve performance similar to a faster device in a non-pipelined system.

Pipelining is not supported during I/O bus cycles and BS8 cycles (16-bit accesses to 8-bit devices).

NOTE

During I/O cycles, NA# is ignored. NA# must be kept deasserted (blocked externally) during the T2 states of BS8 memory cycles.

NOTE

Pipelining is also supported during memory cycles initiated by the two integrated DMA units.

Refer to “Pipelined Cycle” on page 6-19 for a description of pipelined cycles.

6.2.3 Data Bus Transfers and Operand Alignment

The processor can address up to 64 Mbytes (2^{26} bytes, addresses 0000000H–3FFFFFFH) of physical memory and up to 64 Kbytes (2^{16} bytes, addresses 0000H–FFFFH) of I/O. The device maintains separate physical memory and I/O spaces.

A programmer views the address space (memory or I/O) as a sequence of bytes:

- Words consist of 2 consecutive bytes
- Doublewords consist of 4 consecutive bytes

However, in the system hardware, address space is implemented in 2-byte portions. When the processor reads a word, it accesses a byte from each portion of the 16-bit data bus. The processor automatically translates the programmer’s view of consecutive bytes into this hardware implementation.

Memory and I/O spaces are organized physically as sequences of 16-bit words (2^{25} 16-bit memory locations and 2^{15} 16-bit I/O ports maximum). Each word starts at a physical address that is a multiple of 2 and has 2 individually addressable bytes at consecutive addresses.

Pins A25:1 correspond to the most-significant bits of the physical address; these pins address words of memory. The least-significant bit of the physical address is used internally to activate the appropriate byte enable outputs (BHE# or BLE# or both).

Data can be transferred in quantities of either 8 or 16 bits for each bus cycle of a data transfer. When a data transfer can be completed in a single cycle, the transfer is said to be *aligned*. For example, a word transfer involving D15:0 and activating BHE# and BLE# is aligned.

Word transfers that cross a word boundary or doubleword transfers that cross two word boundaries are called *nonaligned* transfers. Nonaligned word transfers require two bus cycles, while nonaligned doubleword transfers require three. The processor automatically generates these cycles. For example:

- A word (16-bit) transfer at (byte) address 03H requires two byte transfers:
  - — The first activates word address 04H and uses D7:0 (to write or read the upper byte of the 16-bit word)
  - — The second activates word address 02H and uses D15:8 (to write or read the lower byte of the 16-bit word)

- A doubleword (32-bit) transfer at (byte) address 03H requires three transfers, one word transfer and two byte transfers:
  - — The first word transfer activates word address 04H and uses D15:0 (to write or read the middle 2 bytes of the 32-bit doubleword)
  - — The next transfer activates word address 06H and uses D7:0 (to write or read the upper byte of the 32-bit word)
  - — The last transfer activates word address 02H and uses D15:8 (to write or read the lower byte of the 32-bit word)

Table 6-3 shows the sequence of bus cycles for all possible alignments and operand length transfers. Even though nonaligned transfers are transparent to a program, they are slower than aligned transfers (due to the extra cycles needed) and should be avoided.

Table 6-3. Sequence of Nonaligned Bus Transfers

Transfer Type	Physical Address	First Cycle		Second Cycle		Third Cycle	
		Address Bus	Byte Enable	Address Bus	Byte Enable	Address Bus	Byte Enable
word	4N+1	4N	BHE#	4N+2	BLE#		
word	4N+3	4N+4	BLE#	4N+2	BHE#		
doubleword	4N+1	4N+4	BLE#	4N	BHE#	4N+2	both
doubleword	4N+2	4N+4	both	4N+2	both		
doubleword	4N+3	4N+4	both	4N+6	BLE#	4N+3	BHE#

6.2.4 Ready Logic

A bus cycle is terminated externally by asserting the READY# pin or internally by either an internal peripheral or the Chip-select Unit's wait-state logic. When an access is to an internal peripheral, the address also goes out to the external bus. When an external device incorrectly decodes a match to the address and drives the READY# pin, contention occurs on the signal. The LBA# pin should be used to alleviate the possibility of contention on the READY# pin. The READY# pin is an output of the processor whenever LBA# is asserted and an input to the processor whenever LBA# is deasserted.

The LBA# pin becomes active when the processor is generating the READY# internally. Figure 6-3 shows the implementation of the READY# signal using the LBA# signal. If you wish to simplify decoding of address space and overlap internal I/O registers, you need to provide external logic to monitor LBA# and end the bus cycle externally when the processor generates the READY# internally.

NOTE

Since LBA# may be used as an output-enable by both the internal and external READY# buffers, care must be taken in selecting the external READY# buffer to minimize contention on the READY# signal caused by differences in buffer characteristics.

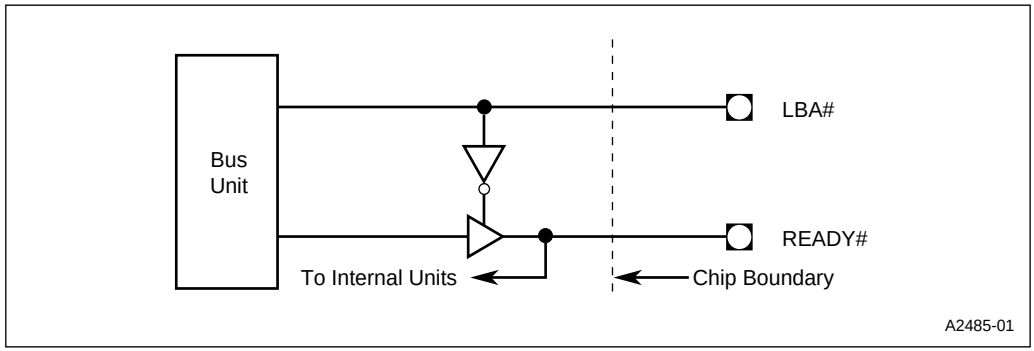


Figure 6-3. Ready Logic

When an internal cycle occurs, the LBA# signal becomes active in Phase 1 of the first T2 state. It then stays active until the rising edge of PH1 of the first T2, T2i or T2P state of the next bus cycle that requires external READY# to terminate the bus cycle. For example, the processor may start an internal bus cycle, go through a few idle states, perform another internal cycle, then a cycle in which the Chip-select Unit generates READY#, run through a few more idle states and then finally do a cycle in which READY# needs to be generated by external logic. LBA# goes active in the first T2 state of the first internal cycle, and stay active through the next two cycles (even during all the idle states in between) and go inactive at the rising edge of PH1 in the first T2, T2i or T2P state of the final cycle (the one that requires an external READY# to terminate).

NOTE

LBA# is deasserted during HOLD cycles.

Figure 6-4 shows internal and external bus cycles.

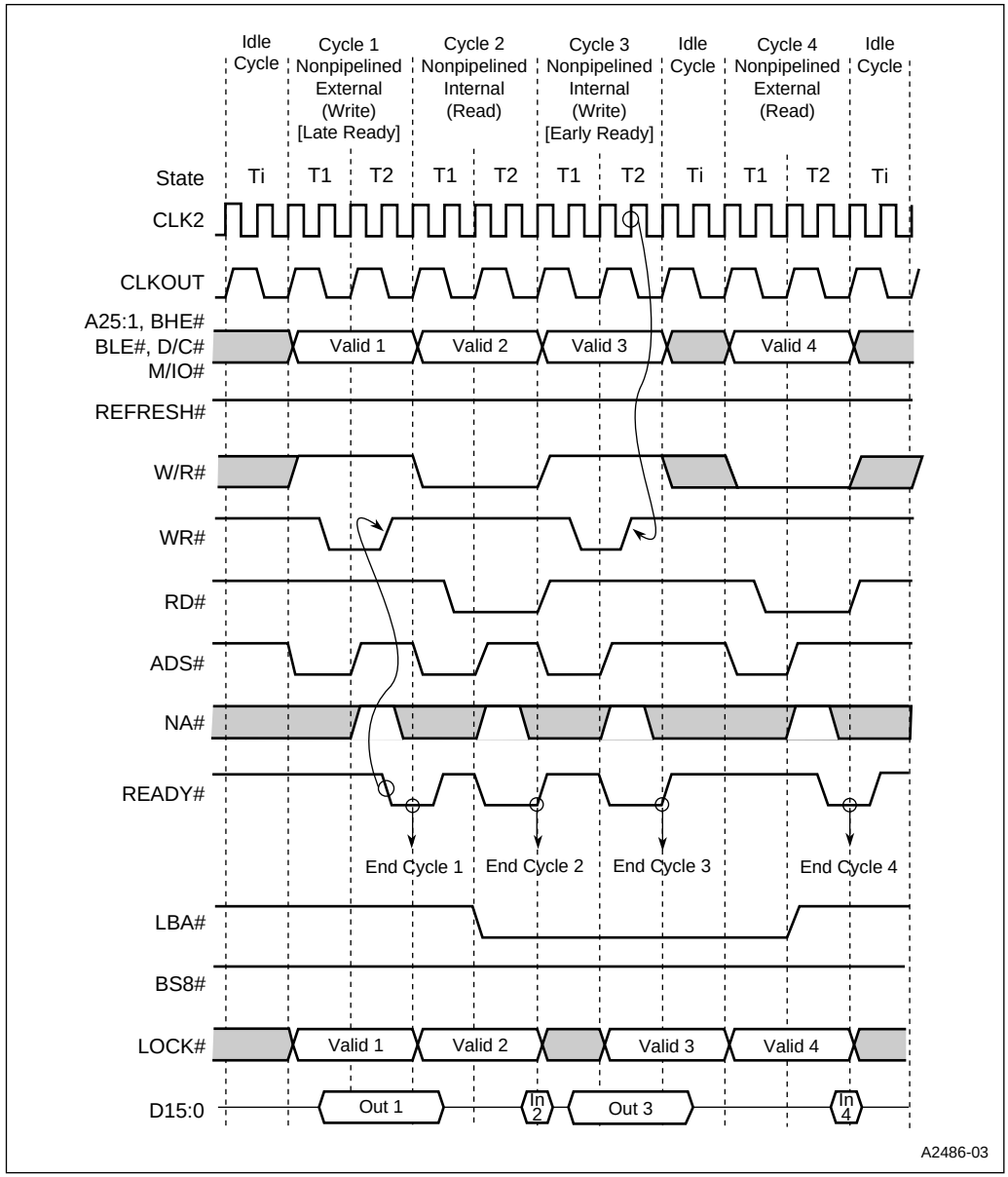


Figure 6-4. Basic Internal and External Bus Cycles

6.3 BUS CYCLES

The processor executes five types of bus cycles:

- Read
- Write
- Interrupt
- Halt/shutdown
- Refresh

6.3.1 Read Cycle

Read cycles are of two types:

- In a **pipelined** cycle, the address and status signals are output in the previous bus cycle, to allow longer memory access times. Pipelined cycles are described in “Pipelined Cycle” on page 6-19.
- In a **nonpipelined** cycle, the address and status signals become valid during the first T-state of the cycle (T1). Figure 6-5 shows the timing for two nonpipelined read cycles (one with and one without a wait-state).

The sequence of signals for the nonpipelined read cycle is as follows:

1. The processor initiates the cycle by driving the address bus and the status signals active and asserting ADS#. The type of bus cycle occurring is determined by the states of the address bus (A25:1), byte enable pins (BLE# and BHE#), and bus status outputs (W/R#, M/IO#, D/C#, REFRESH#, and LOCK#). Because of output delays, these signals should be sampled at the rising edge of the CLK2 signal that coincides with the falling edge of PH2, when ADS# is definitely active. For a read cycle, the bus status outputs have the following states:
   - W/R# is low
   - M/IO# is high for a memory read and low for an I/O read
   - D/C# is high for a memory or I/O data read and low for a memory code read
   - REFRESH# is deasserted
   - LOCK# is asserted for a locked cycle and deasserted for a nonlocked cycle. In a read-modify-write sequence, both the memory data read and memory data write cycles are locked. No other bus master should be permitted to control the bus between two locked bus cycles.

   The address bus, byte enable pins, and bus status pins (with the exception of ADS#) remain active through the end of the read cycle.
2. At the start of phase 2 of T1, RD# becomes active as the processor prepares the data bus for input. This indicates that the processor is ready to accept data.

3. When a chip-select region is enabled for the current read cycle but internal READY# generation is disabled for that region, and the Chip-select Unit is programmed to insert wait-states, the READY# signal is ignored (not sampled) by the processor until the programmed number of wait-states are inserted into the cycle.
4. At the falling edge of PH2 in every T2 state (after the wait-states, if any are programmed in the Chip-select Unit, have expired), READY# is sampled. If READY# is active, the processor reads the input data on the data bus and deactivates RD#.
5. If READY# is high, wait states are added (additional T2 states for nonpipelined cycles) until READY# is sampled low. READY# is sampled at the end of each T2 state (at the falling edge of PH2).
6. Once READY# is sampled low, the processor reads the input data, deactivates RD#, and terminates the read cycle. If a new bus cycle is pending, it begins on the next T-state.

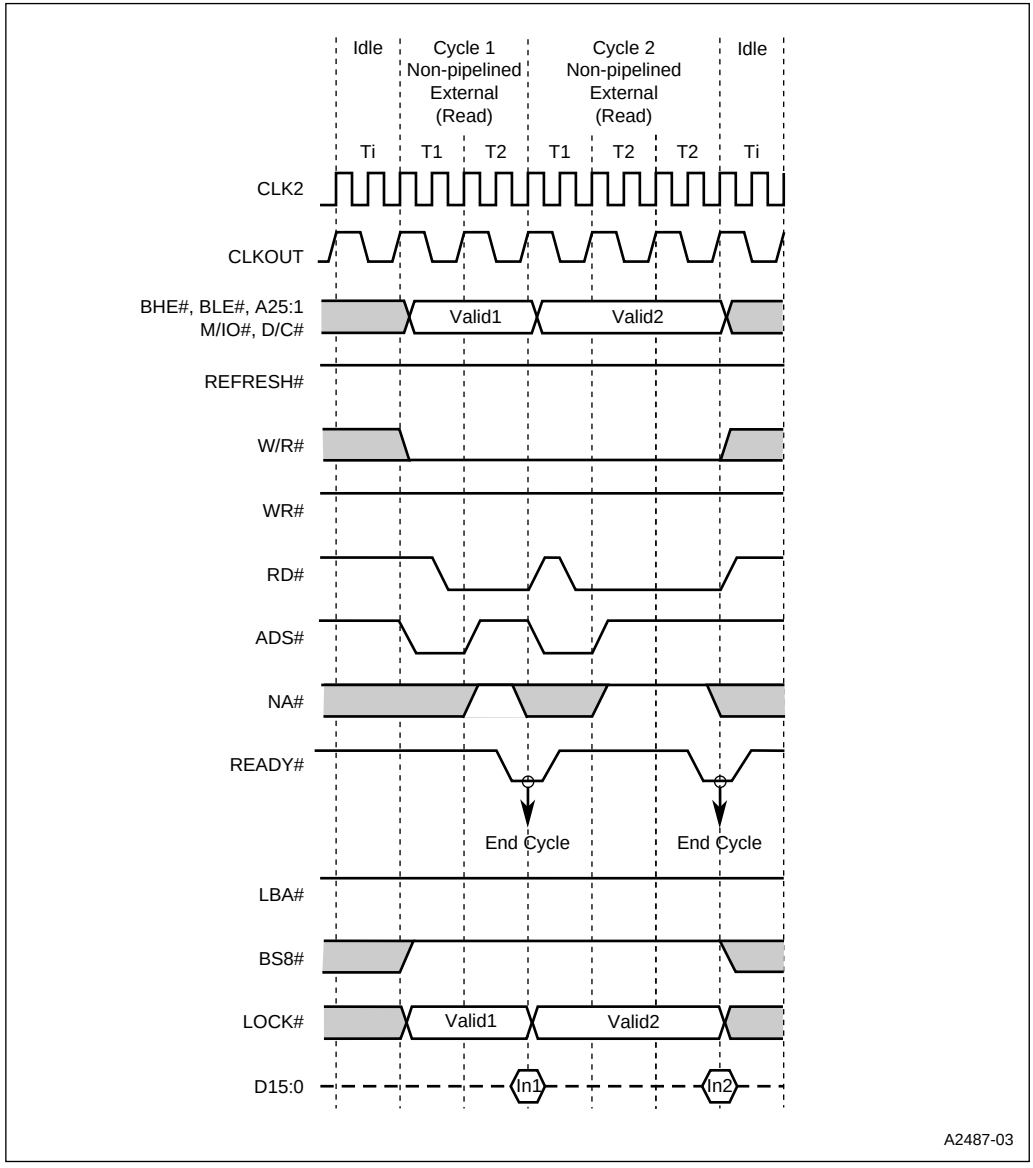


Figure 6-5. Nonpipelined Address Read Cycles

6.3.2 Write Cycle

Write cycles are of two types:

- **Pipelined.** Pipelined write cycles are described in "Pipelined Cycle" on page 6-19.
- **Nonpipelined**. Figure 6-6 shows two nonpipelined write cycles (one with and one without a wait state).

The sequence of signals for a nonpipelined write cycle is as follows:

1. The processor initiates the cycle by driving the address bus and the status signals active and asserting ADS#. The type of bus cycle occurring is determined by the states of the address bus (A25:1), byte enable pins (BLE# and BHE#), and bus status outputs (W/R#, M/IO#, D/C#, REFRESH#, and LOCK#). Because of output delays, these signals should be sampled at the rising edge of the CLK2 signal that coincides with the falling edge of PH2, when ADS# is definitely active. For a write cycle, the bus status outputs have the following states:
   - W/R# is high
   - M/IO# is high for a memory write and low for an I/O write
   - D/C# is high for a memory write or I/O write cycle. During halt and shutdown cycles, D/C# is low. Unless D/C# is decoded by external chip-select logic, the shutdown or halt cycle looks like a memory write cycle to byte address zero or two, respectively. Therefore, the signal D/C# needs to be decoded for memory device chip-selects in this address range (normally SRAM or DRAM devices) in order to recognize halt and shutdown cycles, thus preventing incorrect write cycles to memory
   - REFRESH# is deasserted
   - LOCK# is asserted for a locked cycle and deasserted for an unlocked cycle. In a read-modify-write sequence, both the memory data read and memory data write cycles are locked. No other bus master should be permitted to control the bus between two locked bus cycles.

   The address bus, byte enable pins, and bus status pins (with the exception of ADS# and WR#) remain active through the end of the write cycle.
2. At the start of Phase 2 in T1, the WR# signal is asserted and the CPU begins to drive output data on its data pins. The data remains valid until the start of phase 2 in the T-State after the present bus cycle has terminated.
3. If a chip-select region is enabled for the current read cycle but internal READY# generation is disabled for that region, and the Chip-select Unit is programmed to insert wait-states, then the READY# signal is ignored (not sampled) by the processor until the programmed number of wait-states are inserted into the cycle.

4. The WR# signal can be deasserted in two ways.

   - **Early Ready:** WR# is deasserted at the rising edge of CLK2 in the middle of the T2 state, after any wait states programmed in the Chip-select Unit have expired.

     At the rising edge of PH2, READY# is sampled. If it is found active, WR# is synchronously deasserted in the middle of T2, driven inactive by the rising edge of the PH2 clock. The write cycle is then terminated at the end of the T2 state.

   NOTE

   When READY# is generated by the processor (e.g., when the Chip-select Unit generates it), then the write cycle is always an Early Ready cycle.

   - **Late Ready:** When READY# goes low after the rising edge of PH2 of the T2 state (after the wait-states, if any are programmed in the Chip-select Unit, have expired), WR# is asynchronously deasserted as soon as READY# is asserted (after a small delay caused by the logic). The write cycle is then terminated at the end of the T2 state.

   The WR# signal operates in this manner to ensure sufficient address and chip-select hold time during write cycles (required by many memory and I/O devices). In the first case, the address and chip-select hold time is approximately one CLK2 cycle.

5. When READY# is high, wait-states are added (additional T2 states for nonpipelined cycles) until READY# is sampled low. READY# is sampled in each T2 state (starting at the rising edge of PH2) to deassert the WR# signal appropriately, and at the end of each T2 state (at the falling edge of PH2) to terminate the cycle.

6. Once READY# is sampled low, the write cycle terminates. If a new bus cycle is pending, it begins on the next T-state.

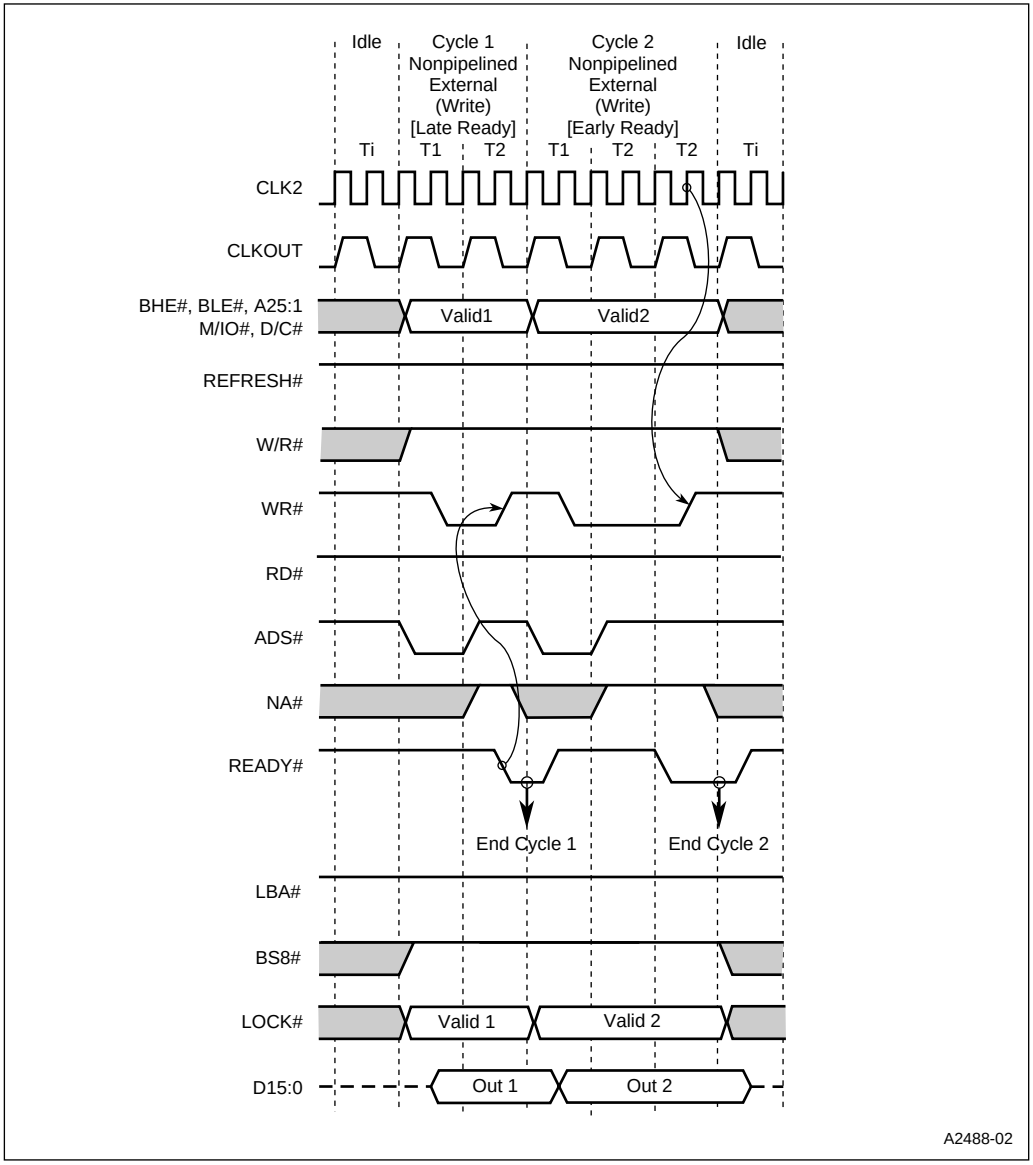


Figure 6-6. Nonpipelined Address Write Cycles

6.3.3 Pipelined Cycle

The pipelining feature of the processor is normally used to achieve zero-wait-state memory subsystems using devices that are slower than those in a zero-wait-state non-pipelined system. Pipelining allows bus cycles to be overlapped, increasing the amount of time available for the memory or I/O device to respond. The next address (NA#) input controls pipelining. NA# is generated by logic in the system to indicate that the address and status buses are no longer needed by the system. When pipelining is not desired in a system, the NA# input should be tied inactive.

During any particular bus cycle, NA# is sampled only after the address and status have been valid for one T-state (the T1P state of pipelined cycles or the first T2 state of nonpipelined cycles) and is continuously sampled in each subsequent T-state until it is found active or the bus cycle is terminated. In particular, NA# is sampled at the rising CLK2 edge in the middle of the T-state (rising edge of Phase 2).

When the system is designed to assert NA#, pipelining may be dynamically requested on a cycle-by-cycle basis by asserting NA#. Typically, only some devices in a system are pipelined.

NOTE

Asserting the NA# pin is a request for pipelining. Asserting NA# during a bus cycle does not guarantee that the next cycle is pipelined. NA# is ignored during I/O cycles and **must** be kept deasserted during the T2 states of BS8 memory cycles.

During the T2 state of a nonpipelined cycle, if NA# is sampled active, one of four states occur:

- If a bus cycle is internally pending in the processor and READY# is returned inactive to the processor and the HOLD input is inactive, then the address, byte enables, and bus status signals for the next bus cycle are driven and the processor bus unit enters a T2P state. T2P states are repeated until the bus cycle is terminated.
- If a bus cycle is internally pending in the processor and READY# is returned active to the processor and the HOLD input is inactive, then the address, byte enables, and bus status signals for the next bus cycle are driven and the processor bus unit enters a T1 (nonpipelined) state. In effect, the NA# input is ignored in this case.
- If READY# is returned inactive and either a bus cycle is not internally pending or the HOLD input is active, then the address and byte enables enter an unknown state, the bus status signals go inactive, and the processor bus unit enters a T2i state. If the bus cycle is not terminated, then the next state is either a T2P state or a T2i state depending on whether a bus cycle is pending.
- If HOLD is asserted to the processor and READY# is returned active, then the Th state is entered from a T2 state regardless of whether an internal bus cycle is pending.

Figure 6-8 illustrates the effect of NA# (Figure 6-7 shows the full bus state diagram including the states related to pipelining). During the second T-state (T2) of a nonpipelined read cycle (cycle 2), NA# is sampled low. A bus cycle was pending internally (cycle 3) and the address, byte enables, and bus status signals for this pending bus cycle (cycle 3) are driven during the next T2P state (the first wait state of the current bus cycle). The RD# and WR# signals do not change until READY# is sampled low.

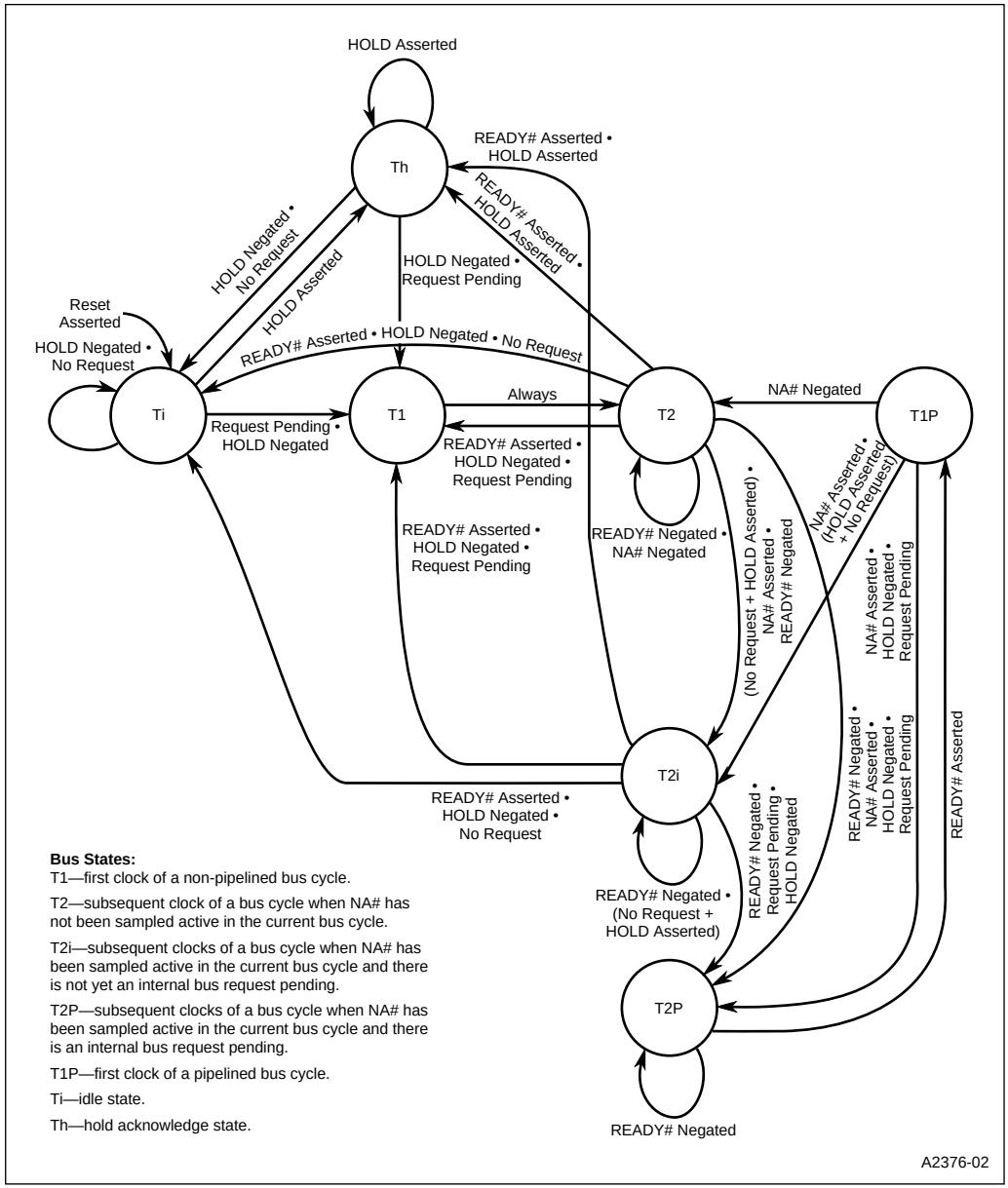


Figure 6-7. Complete Bus States (Including Pipelined Address)

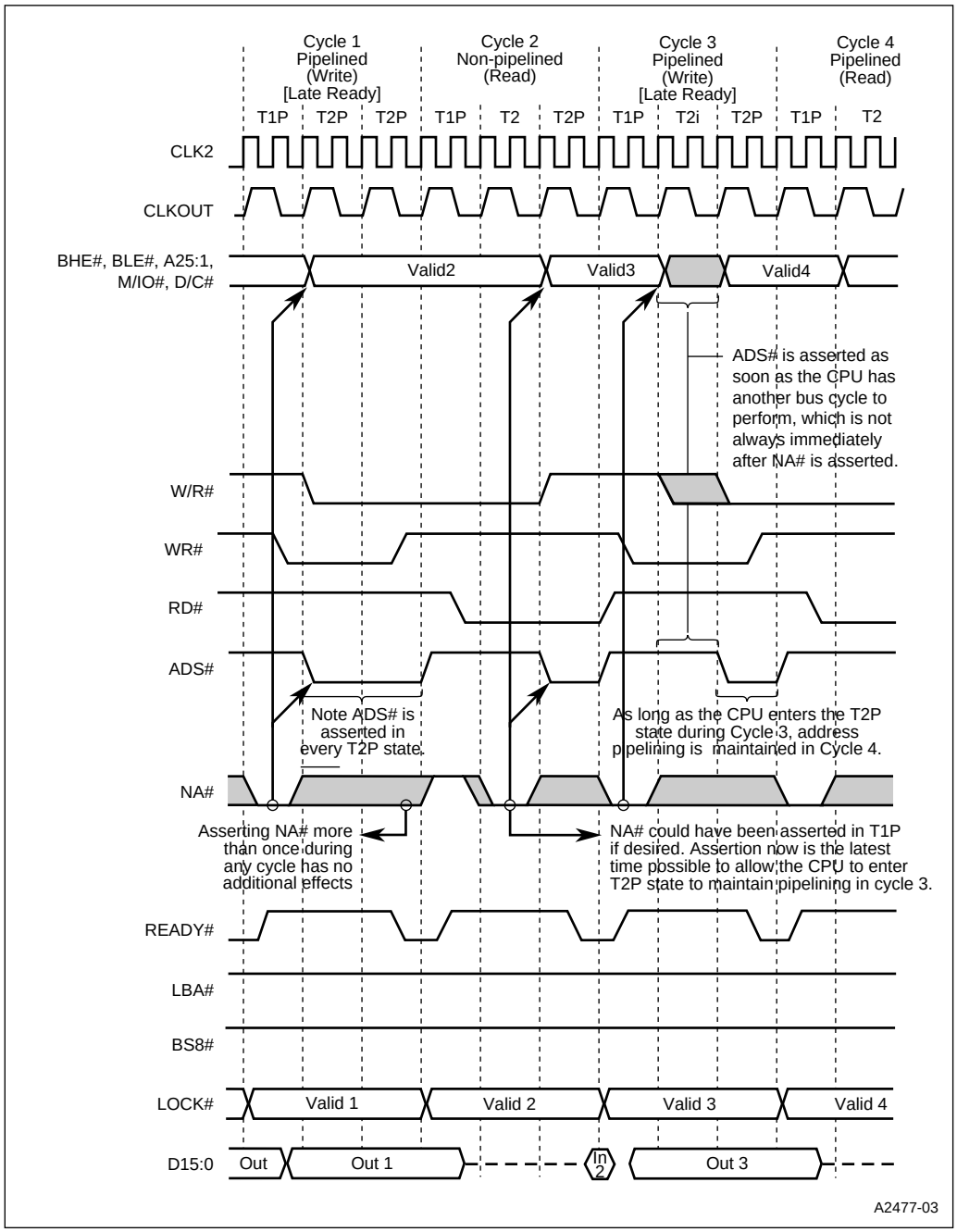


Figure 6-8. Pipelined Address Cycles

In cycle 3, NA# is sampled in the first T-state (T1P); the address and status have been valid for one previous T-state and this is a new bus cycle. NA# is sampled active and — because a bus cycle (cycle 4) is pending internally — the address, byte enables, and bus status signals for this pending bus cycle (cycle 4) are driven during the next T2P state.

In cycle 4, NA# is sampled in the first T-state (T1P); the address and status have been valid for one previous T-state, and this is a new bus cycle. NA# is sampled active and — because a bus cycle is **not** internally pending — the address and byte enables go to an unknown state and the bus status signals go inactive in the next T2i state. When this cycle is terminated by an active READY# signal, there is no bus cycle pending internally and the bus enters the idle state (Ti).

From an idle bus, an additional overhead of one clock cycle is required to start a pipelined bus cycle (this is true with all pipelined bus architectures). This additional clock is used to pipeline the address and status signals for the first bus cycle in a train of pipelined bus cycles. As long as back-to-back bus cycles are executed, the pipelined bus can maintain the same throughput as the nonpipelined bus. Only when the bus pipeline gets broken (by entering an idle or hold state) is the additional one-clock overhead required to start the pipe again for the next train of pipelined bus cycles.

The first bus cycle after an idle bus state is always nonpipelined. Systems that use pipelining typically assert NA# during this cycle to enter pipelining. To initiate pipelining, this nonpipelined cycle must be extended by at least one T-state so that the address and status can be pipelined before the end of the cycle. Subsequent cycles can be pipelined as long as no idle bus cycles occur.

Specifically, NA# is sampled at the start of phase 2 of any T-state in which the address and status signals have been active for one T-state and a new cycle has begun:

- The first T2 state of a nonpipelined cycle (the second T-state)
- The T1P state of a pipelined cycle (the first T-state)
- Any wait state of a nonpipelined or pipelined cycle unless NA# has already been sampled active

Once NA# is sampled active, it remains active internally throughout the current bus cycle. When NA# and READY# are active in the same T2 state, the state of NA# is irrelevant because READY# causes the start of a new bus cycle. Therefore, the new address and status signals are always driven, regardless of the state of NA#. NA# has no effect on a refresh cycle because the refresh cycle is entered from an idle bus state and exits to an idle bus state.

With this processor, address pipelining is optional so that bus cycle timing can be closely tailored to the access time of the memory device.

- Pipelining can be activated once the address is latched externally.
- Pipelining can be not activated if the address is not latched.

For systems that use address pipelining, the great majority of accesses are pipelined. Very few idle states occur in an Intel386 EX processor system. This means that once the processor has entered pipelining, another bus cycle request is almost always internally pending, resulting in a continuous train of pipelined cycles. In measured systems, about 85% of bus cycles are pipelined.

A complete discussion of the considerations for using pipelining can be found in the *Intel386™ SX Processor* datasheet (order number 240187) or the *Intel386™ SX Microprocessor Hardware Reference Manual* (order number 240332).

6.3.4 Interrupt Acknowledge Cycle

An interrupt causes the processor to suspend execution of the current program and execute instructions from another program called an *interrupt service routine*. Interrupts are described in Chapter 9.

The interrupt control unit coordinates the interrupts of several devices, internal and external. It contains two 82C59A programmable interrupt controllers (PICs) connected in cascade. The slave 82C59A module controls up to five internal interrupt sources and up to four external interrupt sources depending upon the configuration programmed. The master 82C59A module controls the slave 82C59A, three internal interrupt sources and up to six external interrupt sources depending upon the configuration programmed. When a device signals an interrupt request, the interrupt control unit activates the processor's INTR input.

Interrupt acknowledge cycles are special bus cycles that enable the interrupt control unit to output a service-routine vector onto the data bus. The processor performs two back-to-back interrupt acknowledge cycles in response to an active INTR input (as long as the interrupt flag is enabled). Interrupt acknowledge cycles are similar to regular bus cycles in that the processor initiates each bus cycle and an active READY# terminates each bus cycle. The cycles are shown in Figure 6-9. The sequence of signals for an interrupt acknowledge cycle is as follows:

1. The address and status signals are driven active and ADS# is driven low to start each bus cycle.
    - Status signals M/IO#, D/C#, and W/R# are low to indicate an interrupt acknowledge bus cycle. These signals must be decoded to generate the INTA input signal for an external 82C59A, if an external cascaded 82C59A is used. The REFRESH# signal is high.
    - LOCK# is active from the beginning of the first cycle to the end of the second. HOLD requests from other bus masters are not recognized until after the second interrupt acknowledge cycle is completed.
    - NA# is ignored.
    - The byte address driven during the first cycle is 4; during the second cycle the byte address is 0. BHE# is high, BLE# is low, and A25:3 and A1 are low for both cycles; A2 is high for the first cycle and low for the second. If the CAS enable bit in the interrupt control unit's configuration register is set (INTCFG.7=1), address bits A18:16 reflect the status of the CAS lines. The CAS lines go valid at the rising edge of PH2 of the T1 state of the first interrupt acknowledge cycle. They then go invalid at the rising edge of PH2 of the next Ti state. At the rising edge of PH2 of the T1 state of the second interrupt acknowledge cycle, the CAS lines go valid again. They then go invalid at the rising edge of PH2 of the next Ti state.

NOTE

Since the CAS lines are invalid in the Ti states between the two interrupt acknowledge cycles, cascading of external 82C59A devices requires latching the CAS lines. This ensures that the CAS lines remain valid during these Ti states to fulfill the requirements of the external 82C59A devices.

2. The processor floats D15:0 for both cycles; however, at the end of the second cycle, if the interrupt is from an external cascaded 82C59A, the service-routine vector number driven on the lower data bus by the 82C59A is read by the processor on data pins D7:0. Otherwise, the active internal 82C59A sends the vector to the processor.
3. The first cycle is always an internal cycle and the second may be internal or external. Therefore, READY# is generated internally for the first cycle and for the second cycle, if the interrupt request is from one of the internal 82C59A modules. If the interrupt is from a cascaded external 82C59A, external logic must assert READY# to terminate the second cycle. The internal Chip-select Unit can **not** generate READY# for the second interrupt acknowledge cycle.

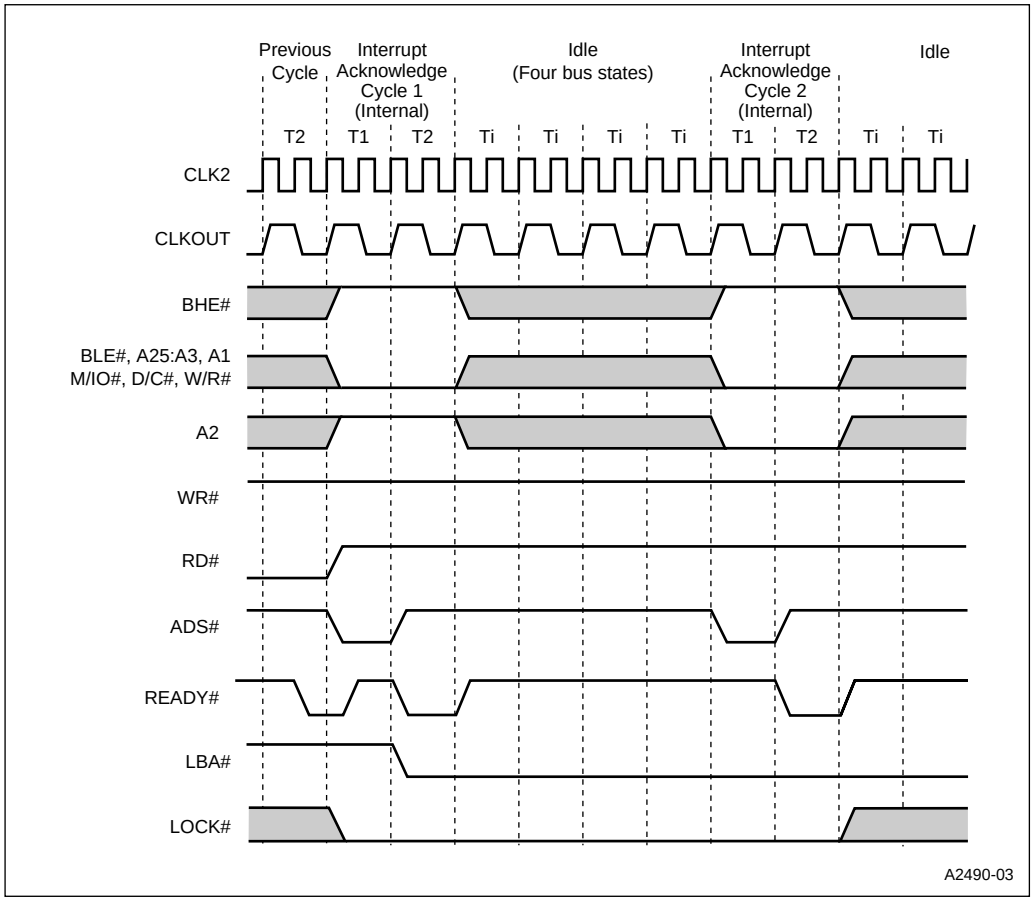


Figure 6-9. Interrupt Acknowledge Cycles

6.3.5 Halt/Shutdown Cycle

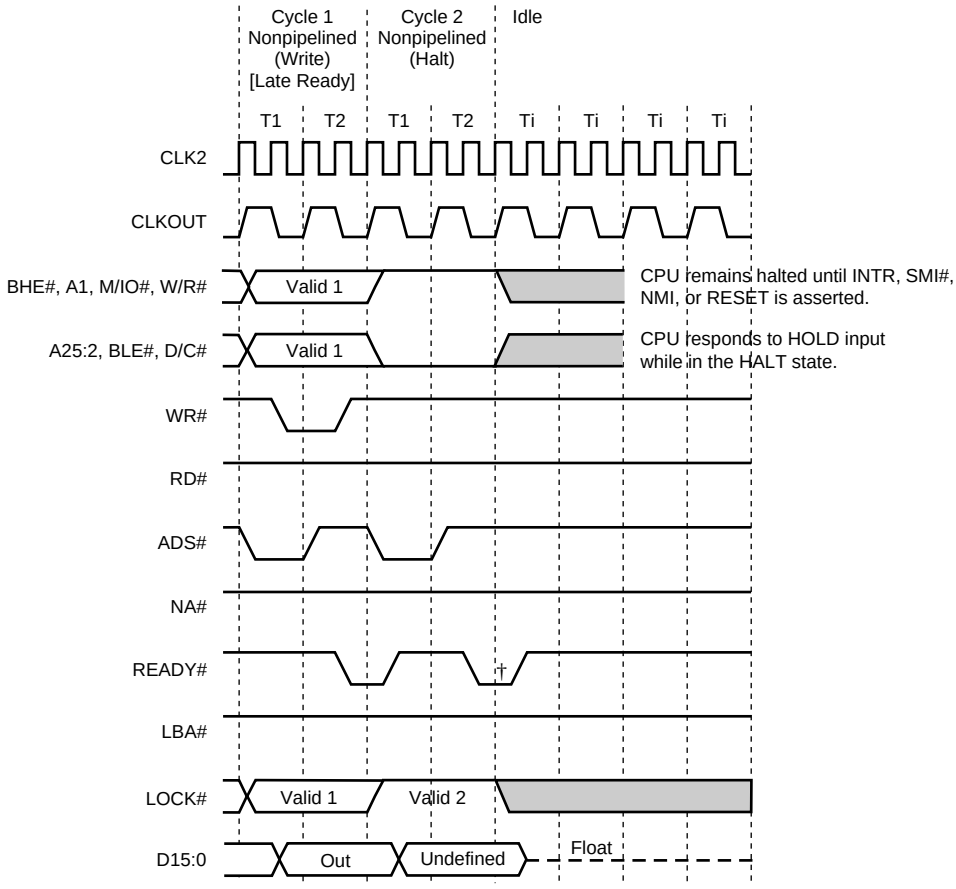
The halt condition occurs in response to a HALT instruction. The shutdown condition occurs when the processor is processing a double fault and encounters a protection fault; the processor cannot recover and therefore, shuts down. Externally, a shutdown cycle differs from a halt cycle only in the resulting address bus outputs. The sequence of signals for a halt cycle is as follows:

1. As with other bus cycles, a halt or shutdown cycle is initiated by driving the address and status signals active and asserting ADS#. Figure 6-10 shows a halt bus cycle. The address and status signals are driven to the following active states:
   - M/IO# and W/R# are driven high and D/C# is driven low to indicate a halt cycle or a shutdown cycle.
   - The address bus outputs a byte address of 2 for a halt condition and a byte address of 0 for a shutdown condition. These signals are used by external devices to respond to the halt or shutdown cycle.

   NOTE

   The halt or shutdown bus cycle appears as a memory write operation to byte address 0 or 2 (depending on whether a shutdown or halt cycle is being performed) if the D/C# signal is not decoded. External address decoders need to decode the D/C# signal to avoid erroneous writes to devices in this address region; otherwise, a halt or shutdown cycle corrupts the data at those addresses. RD#, WR# and the chip-select signals, UCS# and CS6:0#, are inactive during halt cycles.

2. READY# can be generated externally or internally to terminate a Halt/Shutdown cycle. The HSREADY bit in the Power Control Register (PWRCON, see Figure 8-5 in Chapter 8), can be set to generate an internal READY# for halt/shutdown cycles. If internal READY# generation is enabled, then the LBA# signal goes active and behaves as described in "Ready Logic" on page 6-10. Also, the cycle is always a zero-wait-state cycle. When external READY# is required to terminate the halt/shutdown cycle, then READY# may be delayed to add wait-states. The processor remains in the halt or shutdown condition until one of the following occurs:
   - NMI goes active; the processor then services the interrupt.
   - RESET goes active; the processor is reinitialized.
   - In the halt condition (but not in the shutdown condition), if maskable interrupts are enabled, an active INTR input causes the processor to end the halt cycle and service the interrupt. The processor can service processor extension (PEREQ) requests and hold (HOLD) requests while in the halt or shutdown condition.
   - The processor is in the halt condition and SMI# goes active; the processor then services the SMI#. When the processor is in the shutdown condition, SMI# has no effect.



† HALT cycle must be acknowledged by READY# asserted. This READY# could be generated internally or externally.

A2492-02

Figure 6-10. Halt Cycle

6.3.6 Refresh Cycle

The refresh control unit simplifies dynamic memory controller design by issuing dummy read cycles at specified intervals. (For more information, refer to Chapter 15, "REFRESH CONTROL UNIT.") Figure 6-11 shows a basic refresh cycle. The sequence of signals for a refresh cycle is as follows:

1. Like a read cycle, the refresh cycle is initiated by asserting ADS# and completed by asserting READY#. The address and status pins are driven to the following values:
   - M/IO# and D/C# are driven high and W/R# and REFRESH# are driven low to indicate a memory refresh.
   - Address lines are driven to the current refresh address (the value in the Refresh Address Counter in the Refresh Control Unit), while the BHE# and BLE# are driven high.
2. To complete the refresh cycle, either READY# must be asserted externally or the chip select unit must be programmed to generate READY# for the address region specified in the Refresh Address Base Register in the refresh control unit. The refresh control unit then relinquishes control to the current internal bus master until the next refresh cycle is needed.

During hold acknowledge cycles with the HLDA pin active, a refresh request causes the internal bus arbiter to deassert the HLDA pin. The processor then waits for the HOLD pin to be deasserted for at least one processor clock cycle. Once HOLD is deasserted, the processor begins the refresh cycle. Figure 6-12 shows a refresh cycle during a HOLD/HLDA condition.

NOTE

BS8# is ignored during refresh cycles. It has no effect on a refresh cycle.

CAUTION

External bus arbitration logic should monitor the HLDA signal when the refresh control unit is being used. If a refresh request is not serviced (by performing a refresh cycle) because an external master does not give up the bus, the DRAM devices may lose data.

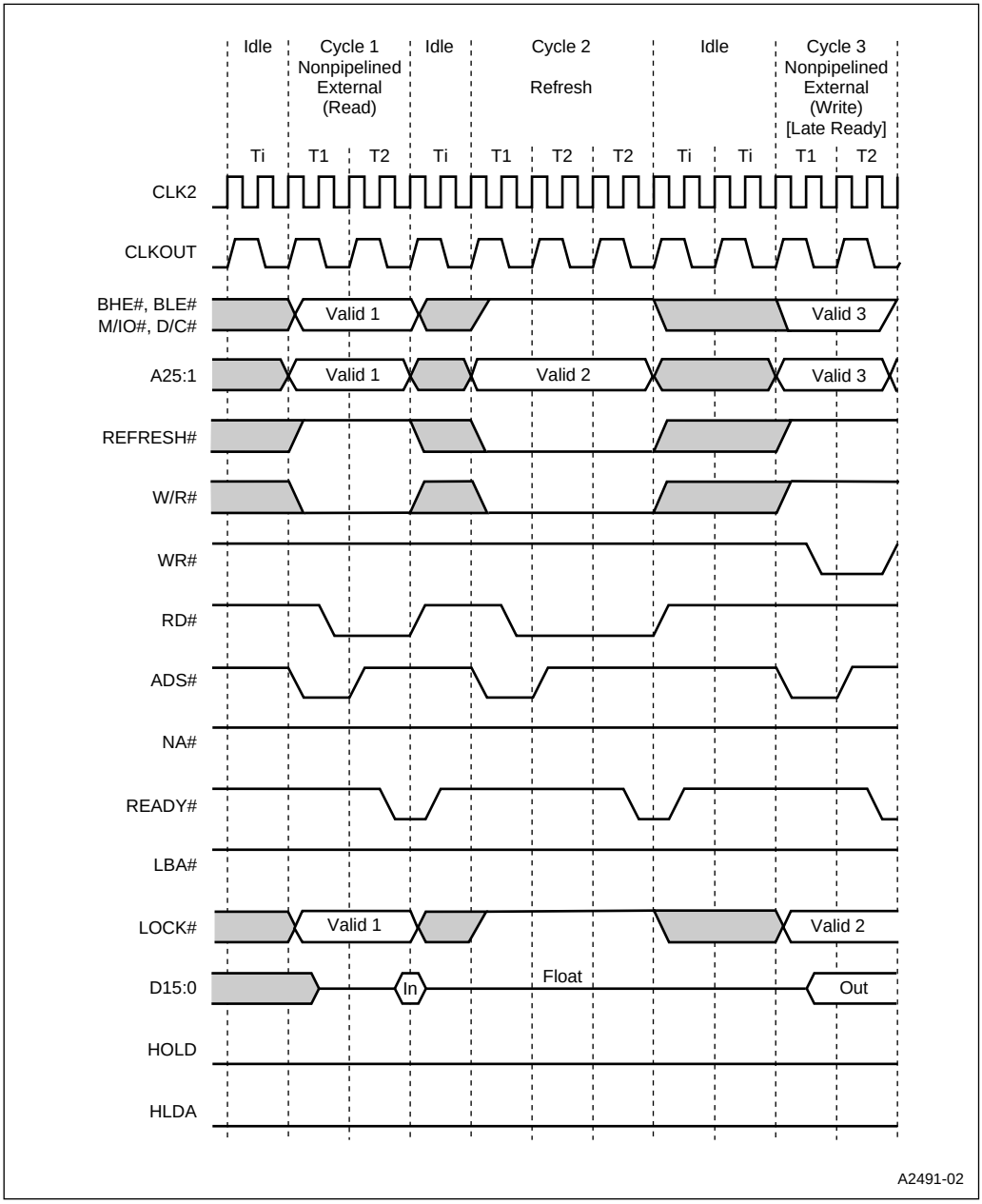


Figure 6-11. Basic Refresh Cycle

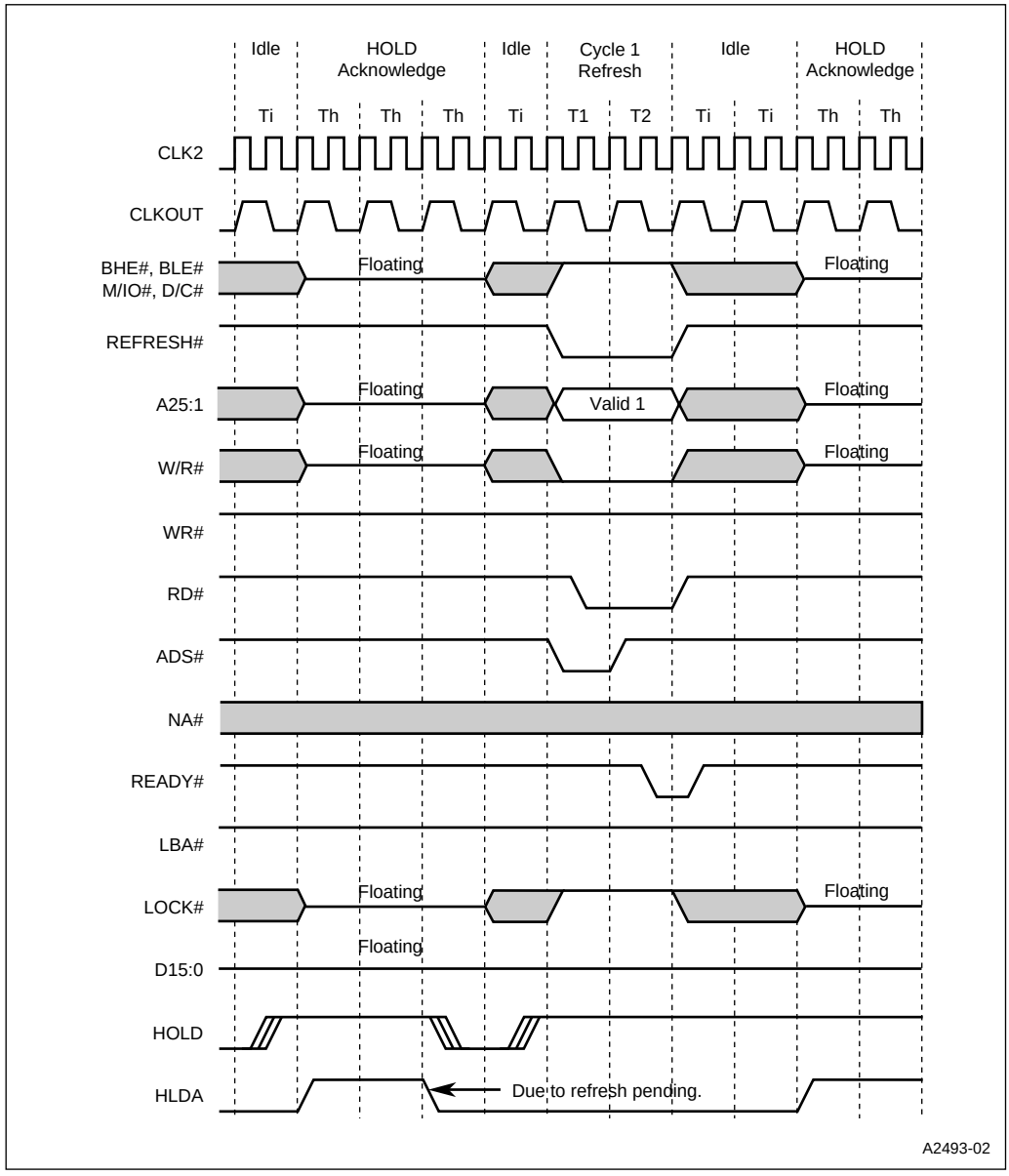


Figure 6-12. Refresh Cycle During HOLD/HLDA

6.3.7 BS8 Cycle

The BS8 cycle allows external logic to dynamically switch between an 8-bit data bus size and a 16-bit data bus size, by using the BS8# signal. Figure 6-13 shows a word access to an 8-bit peripheral.

To use the dynamic 8-bit bus sizing, an external memory or I/O should connect to the lower eight bits of the data bus (D7:0), use the BLE# as address bit 0, and assert BS8# (at the BS8# pin) in T2 of a memory or I/O access. A BS8 cycle can also be generated by the internal chip-select unit (Refer to Chapter 14, “CHIP-SELECT UNIT”). In this case, the Chip Select Unit generates the BS8# signal internally.

Depending upon the current bus access width and address and the state of the BS8# signal, the processor performs the actions described in the next two sections.

6.3.7.1 Write Cycles

- If the current bus cycle is a byte write with BHE# active and BLE# inactive, the processor copies the upper eight bits of the data bus (D15:8) to the lower eight bits of the data bus (D7:0), i.e. the byte appears on both the upper and lower data buses.
- If the current bus cycle is a byte write with BHE# inactive and BLE# active, the processor ignores the state of the BS8# signal.
- If the current bus cycle is a word write with both BHE# and BLE# active and the processor samples the BS8# signal active at the end of the last T2 (when READY# is sampled active), the processor waits for the current bus to complete and then executes another write cycle with the upper eight bits of the data bus (D15:8) copied to the lower eight bits of the data bus (D7:0). The processor deactivates BLE# on the second cycle (BLE# is used as address A0 to an 8-bit device; this translates to A0=0 for the first cycle and A0=1 for the second).

6.3.7.2 Read Cycles

- If the current bus cycle is a byte read with BHE# active and BLE# inactive, and the processor samples the BS8# signal active at the end of the last T2 (when READY# is sampled active), the processor latches the data on the lower eight bits of the data bus (D7:0) and internally routes this data to the upper data bus of the core.
- If the current bus cycle is a byte read with BHE# inactive and BLE# active, the processor ignores the state of the BS8# signal.
- If the current bus cycle is a word read with both BHE# and BLE# active and the processor samples the BS8# signal active at the end of the last T2 (when READY# is sampled active), the processor waits for the current bus cycle to complete and latches the data on the lower eight bits of the data bus (D7:0). It then executes another read cycle, with BLE# inactive (BLE# is used as address A0 to an 8-bit device; this translates to A0=0 for the first cycle and A0=1 for the second), latching the data on the lower eight bits of the data bus (D7:0) again and using it.

The BS8 cycle generates additional bus cycles for read and write cycles only. For interrupt and halt/shutdown cycles, the accesses are byte wide and the BS8# signal is ignored. For a refresh cycle, the byte enables are both disabled and the BS8# signal is ignored.

NOTE

If a BS8 cycle requires an additional bus cycle, the processor retains the current address for the second cycle. Address pipelining cannot be used with BS8 cycles because address pipelining requires that the next address be generated on the bus before the end of the current bus cycle. NA# must be kept deasserted during the T2 states of BS8 memory cycles. NA# is ignored in all I/O cycles.

NOTE

BS8# must be inactive at the falling edge of PH2 of the T1 state of a non-BS8 cycle; for example, if the current cycle is a BS8 cycle (BS8# asserted) and the next cycle is not a BS8 cycle, BS8# must be deasserted before the end of the T1 state of the next cycle, i.e. the non-BS8 cycle.

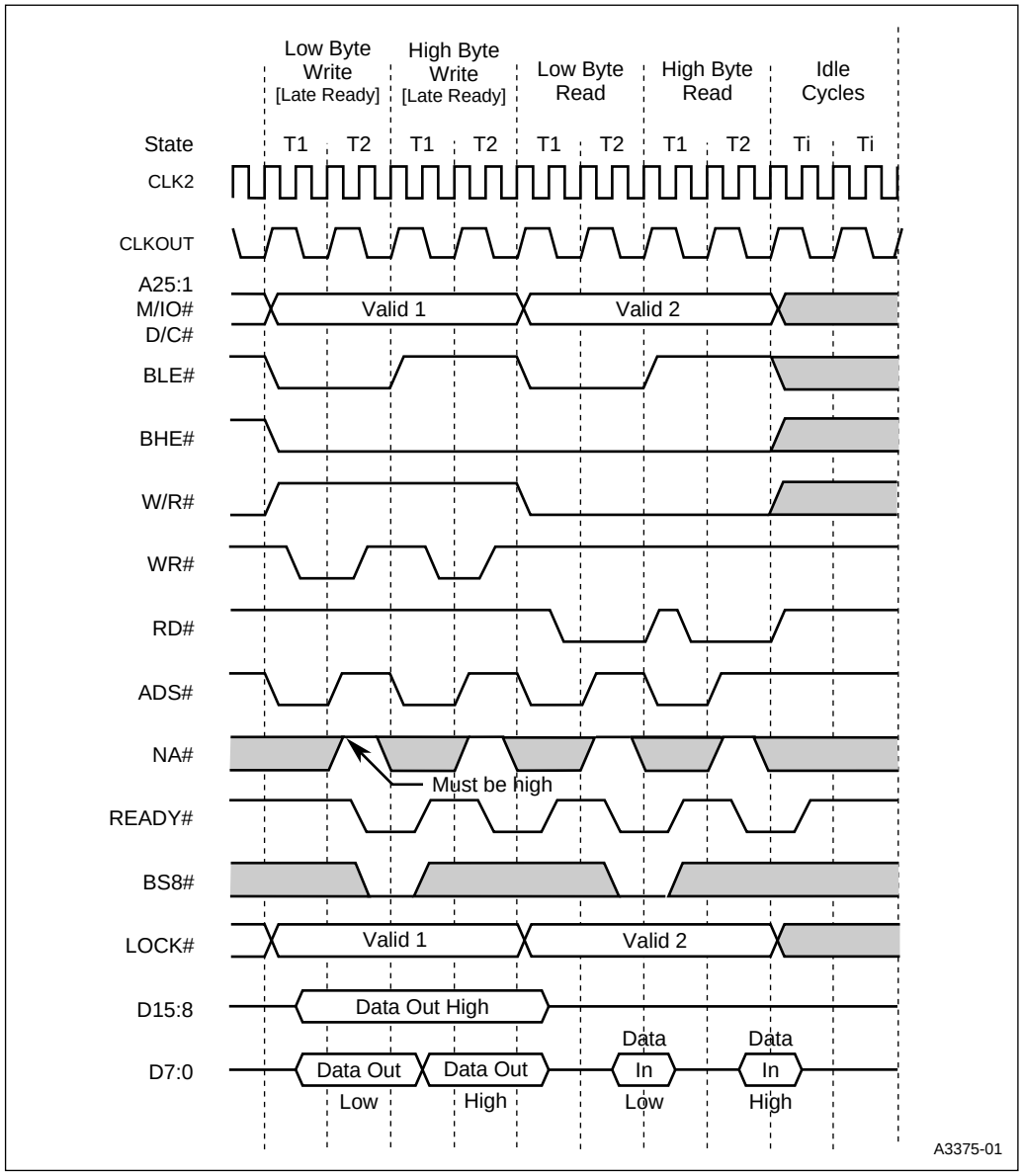


Figure 6-13. 16-bit Cycles to 8-bit Devices (Using BS8#)

6.4 BUS LOCK

In a system in which more than one device (a bus master) may control the local bus, locked cycles are used to make sequential bus cycles indivisible. Otherwise, the cycles may be separated by a cycle from another bus master.

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

The LOCK# output indicates, to the other bus masters, that they may not gain control of the bus. In addition, when LOCK# is asserted, the processor does not recognize a HOLD request from another bus master.

6.4.1 Locked Cycle Activators

The LOCK# signal is activated explicitly by the LOCK prefix on certain instructions. (The instructions are listed in the *Intel386™ SX Microprocessor Programmer's Reference Manual*, order number 240331). LOCK# is also asserted automatically for XCHG instructions, descriptor updates, and interrupt acknowledge cycles.

6.4.2 Locked Cycle Timing

LOCK# is activated on the CLK2 edge that begins the first locked bus cycle and deactivated when READY# is sampled active at the end of the last bus cycle to be locked. LOCK# is activated and deactivated on these CLK2 edges regardless of address pipelining. If address pipelining is used, LOCK# remains active until the current bus cycle is completed (READY# sampled active for the current bus cycle). Consequently, the LOCK# signal can extend into the next memory access cycle that does not need to be locked. (See Figure 6-14). The result is that the use of the bus by another bus master is delayed by one bus cycle.

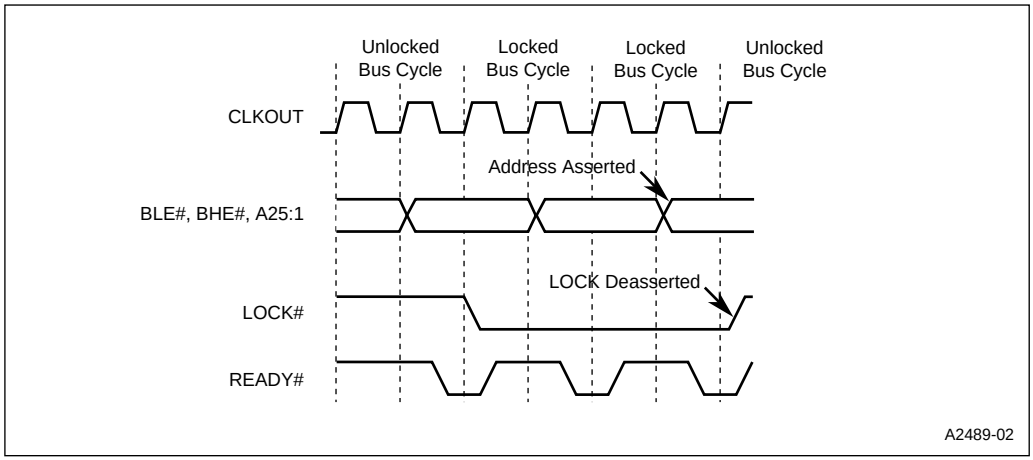


Figure 6-14. LOCK# Signal During Address Pipelining

6.4.3 LOCK# Signal Duration

The maximum duration of the LOCK# signal affects the maximum HOLD request latency because HOLD is recognized only after LOCK# goes inactive. The duration of LOCK# depends on the instruction being executed and the number of wait states per cycle. The longest duration of LOCK# is 9 bus cycles plus approximately 15 clocks. This occurs when an interrupt (hardware or software) occurs and the processor performs a Locked read of the gate in the interrupt descriptor table (8 bytes), a read of the target descriptor (8 bytes), and a write of the accessed bit in the target descriptor.

6.5 EXTERNAL BUS MASTER SUPPORT (USING HOLD, HLDA)

The processor provides internal arbitration logic that supports a protocol for transferring control of the processor bus to an external bus master. This protocol is implemented through the HOLD input and the HLDA output. The internal arbitration logic of the processor consists of a bus arbiter. This arbiter supports the core and four other bus masters, i.e. external bus master using HOLD, two internal DMA Units and the Refresh Control Unit. For a description of the protocol of the internal bus arbiter, refer to “Bus Control Arbitration” on page 12-9.

When the internal bus arbiter receives a request through one of its four possible request signals, it asserts the HOLD signal to the core. The core then completes its current nonlocked bus cycle and asserts its HLDA signal, thus informing the arbiter that control of the bus can now be turned over to the requester.The arbiter then asserts its appropriate acknowledge signal to the requester. For example, if an external bus master requests the bus using the HOLD input pin, then the arbiter asserts the HLDA output.

6.5.1 HOLD/HLDA Timing

To gain control of the local bus, the requesting bus master drives the HOLD input active. This signal can be asynchronous to the processor's CLK2 input. The processor responds by:

- completing its current bus cycle
- deasserting WR#, RD#, LBA#, SMIACT#, UCS#, CS6:0# and REFRESH# and three-stating all other bus outputs except HLDA (effectively removing itself from the bus)
- driving HLDA active to signal the requesting bus master that it may take control of the bus

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

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

If the internal refresh control unit is used, the HLDA signal may drop while an external master has control of the bus, in which case the external bus master may or may not drop HOLD to allow the processor to perform the refresh cycle. If the latter occurs, the memory device(s) may lose data because the refresh cycle could not execute.

When the processor receives an active HOLD input, it completes the current bus cycle before relinquishing control of the bus. Figure 6-7 shows the state diagram for the bus including the HOLD state.

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

HOLD has priority over most core bus cycles, but is not recognized under certain conditions:

- During locked cycles
- Between two interrupt acknowledge cycles (LOCK# asserted)
- During misaligned word transfers (LOCK# not asserted)
- During doubleword (32-bit) transfers (LOCK# not asserted)
- During misaligned doubleword transfers (LOCK# not asserted)
- During an active RESET signal (HOLD **is** recognized during the time between the falling edge of RESET and the first instruction fetch)

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

- HOLD pin - It is monitored to determine when the processor may regain control of the bus.
- RESET pin - It is of a higher priority than HOLD. An active RESET input reinitializes the device.

- NMI pin - The request is recognized and latched. It is serviced after HOLD is released.
- SMI# pin - The request is recognized and latched. It is serviced after HOLD is released.

6.5.2 HOLD Signal Latency

Because other bus masters may be used in time-critical applications, the amount of time the bus master must wait for bus access (HOLD latency) can be a critical design consideration. Because a bus cycle must be terminated before HLDA can go active, the maximum possible latency occurs when a bus-cycle instruction is being executed or a DMA block mode transfer is in progress. Wait states increase latency, and HOLD is not recognized between locked bus cycles and interrupt acknowledge cycles. The internal DMA may also contribute to the latency.

The HOLD latency is dependent on a number of parameters:

- The instruction being executed at the time the HOLD request occurs.
- The number of wait states during various access cycles, including the following:
  - — Memory wait states
  - — Code fetch wait states
  - — Interrupt acknowledge wait states
  - — Refresh wait states
- The priority of the requester.
- The mode of the DMA:
  - — Block mode
  - — Single cycle mode
  - — Demand transfer mode

6.6 DESIGN CONSIDERATIONS

- Upon reset, UCS# is configured as a 16-bit chip-select signal. If the Boot device is only an 8-bit device, then BS8# must be asserted whenever UCS# is active (until the UCS region can be reprogrammed to reflect an 8-bit region). One way of doing this is by connecting the UCS# pin directly to the BS8# pin, if there are no other devices that need to use the BS8# pin. If UCS# is tied directly to BS8#, then the UCS region need not be programmed to reflect an 8-bit region.
- Since LBA# may be used as an output-enable by both the internal and external READY# buffers, care must be taken in selecting the external READY# buffer to minimize contention on the READY# signal caused by differences in buffer characteristics.

6.6.1 Interface To Intel387™ SX Math Coprocessor

The Intel387 SX Math Coprocessor is an extension to the Intel386 EX embedded processor architecture. The combination of the Intel387 SX Math Coprocessor with the Intel386 EX embedded processor dramatically increases the processing speed of computer application software that uses high performance floating-point operations.

An internal Power Management Unit enables the Intel387 SX Math Coprocessor to perform floating-point operations while maintaining very low power consumption. The internal Power Management Unit effectively reduces power consumption by 95% when the coprocessor is idle.

This section describes special considerations for interfacing the Intel387 SX Math Coprocessor with the Intel386 EX embedded processor. For complete information, refer to the *Intel387™ SX Math Coprocessor* datasheet (Order number 240225).

6.6.1.1 System Configuration

The Intel387 SX Math Coprocessor can be interfaced to the Intel386 EX embedded processor as shown in Figure 6-15.

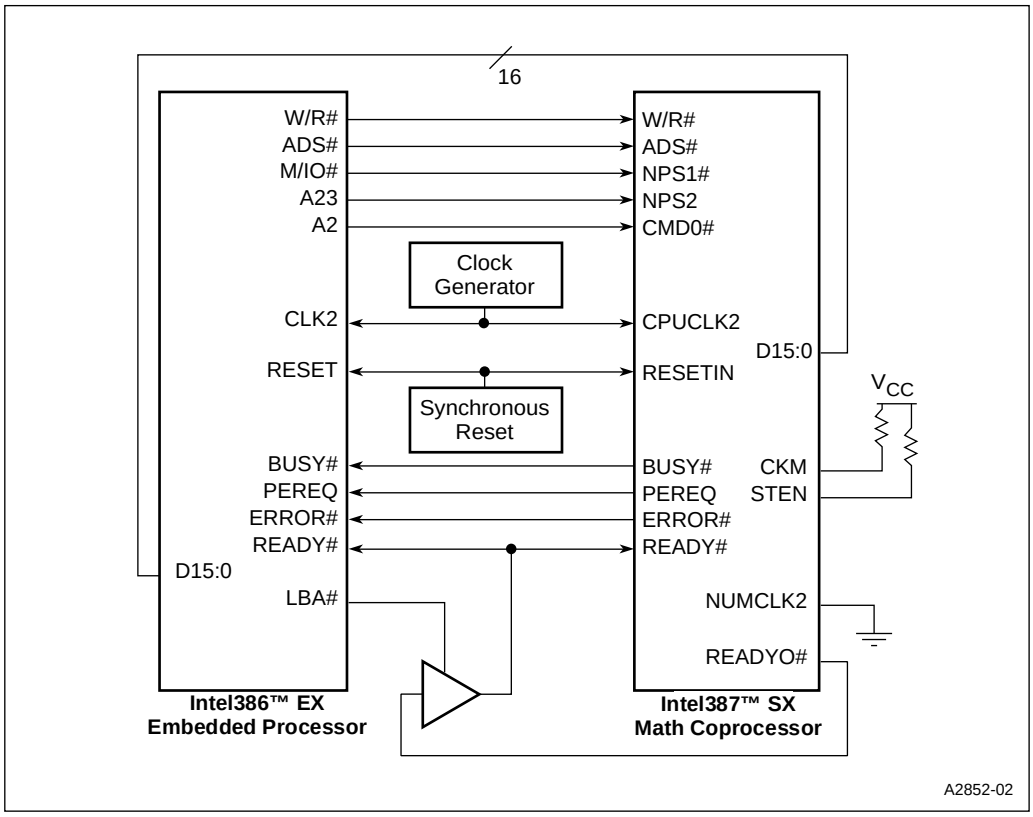


Figure 6-15. Intel386 EX Processor to Intel387 SX Math Coprocessor Interface

A dedicated communication protocol makes possible high-speed transfer of opcodes and operands between the Intel386 EX processor and the Intel387 SX math coprocessor. Most control pins of the Intel387 SX Math Coprocessor are connected directly to Intel386 EX processor pins.

The interface has these characteristics:

- The Intel387 SX Math Coprocessor shares the local bus of the Intel386 EX processor.
- The Intel386 EX processor and Intel387 SX Math Coprocessor share the same reset signals. They also share the same clock input.
- The corresponding BUSY#, ERROR#, and PEREQ pins are connected together.
- The Status Enable (STEN) selects the math coprocessor. It causes the chip to recognize other chip select inputs. STEN is tied high.
- CKM is tied high to select the synchronous mode of operation for the coprocessor.
- The math coprocessor NPS1# and NPS2 inputs are connected to the Intel386 EX processor M/IO# and A23 inputs respectively. For math coprocessor cycles, M/IO# is always LOW and A23 always HIGH.
- The math coprocessor input CMD0 is connected to the A2 output. The Intel386 EX embedded processor generates address 8000F8H when writing a command and address 8000FCH or 8000FEH (treated as 8000FCH by the Intel387 SX Math Coprocessor) when writing or reading data. It does not generate any other addresses during Intel387 SX Math Coprocessor bus cycles.

CAUTION

A chip-select signal could go active during coprocessor cycles if a match for the lower 16 bits of address is found in one of the chip-select regions of the Chip-select Unit. This can happen because only the lower 16 bits are decoded by the Chip-select Unit during I/O cycles.

- The READYO# pin of the coprocessor must be sent through a buffer to prevent the Intel386 EX processor and coprocessor from simultaneously driving the READY# pin. The buffer is enabled using the LBA# pin. During internal bus cycles, the LBA# pin is asserted and the Intel386 EX processor provides the READY# signal. In a coprocessor access, the LBA# is deasserted, the external buffer is enabled, and the coprocessor provides the READY# signal to the Intel386 EX processor.

6.6.1.2 Software Considerations

To enable math-coprocessor support in the Intel386 EX processor, you must set the MP (Math Present) bit and clear the EM (Coprocessor Emulation) bit in the Machine Status Word (lower half of the CR0 register in the core). This can be done using the following code:

```
smsw            ax              ;; Store Machine Status Word into AX
or              ax, 2           ;; Set MP bit
and             ax, 0fffbh      ;; Clear EM bit
lmsw            ax              ;; Load AX into Machine Status Word
```

Also, bit 5 in the PINCFG register (Figure 5-15 on page 5-24) **must** be cleared, to connect the coprocessor-related signals of the core to the package pins.

Below is an example of a simple routine that can be executed using the math-coprocessor:

```
fninit                              ;; Initialize Math Coprocessor
fldpi                               ;; Load (Push on to the 387 stack) “Pi”
fld1                                ;; Load (Push on to the 387 stack) “1”
fadd                                ;; Add the two values, i.e. Pi + 1
fist            word ptr [di]       ;; Convert to integer and Store at
                                    ;; location pointed to by DS:DI
```

6.6.2 SRAM/FLASH Interface

SRAM and FLASH devices can be connected directly to the Intel386 EX processor as shown in Figure 6-16. Separate CS*n*#, RD# and WR# strobes enable a “glueless” interface. The WR# signal, when used with an “EARLY READY#” (described in “Write Cycle” on page 6-16), guarantees the ‘WE#-Inactive-to-Address-Invalid’ time of most SRAM and FLASH devices.

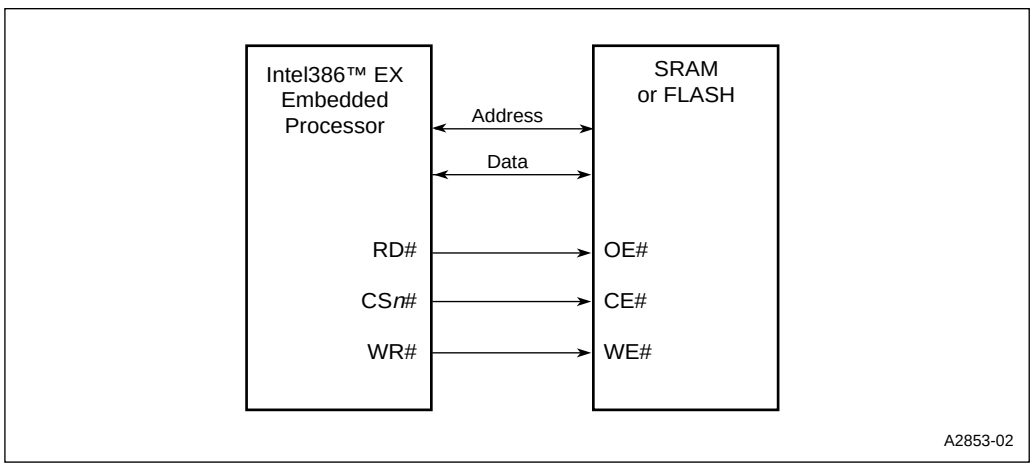


Figure 6-16. Intel386 EX Processor to SRAM/FLASH Interface

6.6.3 PSRAM Interface

Pseudo SRAM (PSRAM) devices can be easily interfaced (Figure 6-17) to the Intel386 EX processor. PSRAM devices have an interface that is similar to SRAM devices (They are also pin-compatible in many cases). The two major differences between PSRAM and SRAM devices are:

- PSRAM devices require a CE# precharge (inactive) time between access cycles. Since the Intel386 EX processor does not guarantee a minimum inactive time on it's CS*n*# signals, control logic is required to satisfy the PSRAM device's CE# precharge time.
- PSRAM devices have a RFSH# input pin. This signal activates an internal refresh cycle. The REFRESH# output of the Intel386 EX processor can be connected directly to the PSRAM device's RFSH# pin.

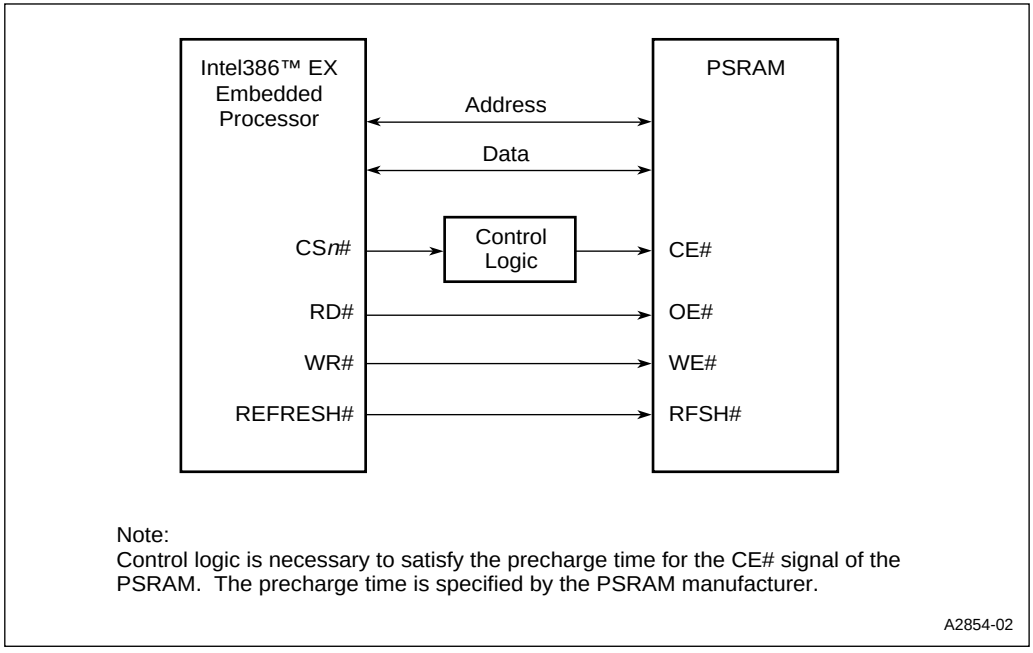


Figure 6-17. Intel386 EX Processor to PSRAM Interface

6.6.4 Paged DRAM Interface

External logic is required to interface the Intel386 EX processor to DRAM devices, as shown in Figure 6-18. The PLD generates the RAS# and CAS# signals.

If RAS#-Only Refresh is being performed (using the Refresh Control Unit of the processor), then during a Refresh Cycle, the PLD enables the Column Address Buffer and asserts the RAS# signal (shaded sections in the figure). Refer to Chapter 6, "BUS INTERFACE UNIT," for more information.

A single multiplexer can be used instead of the separate row and column address buffers.

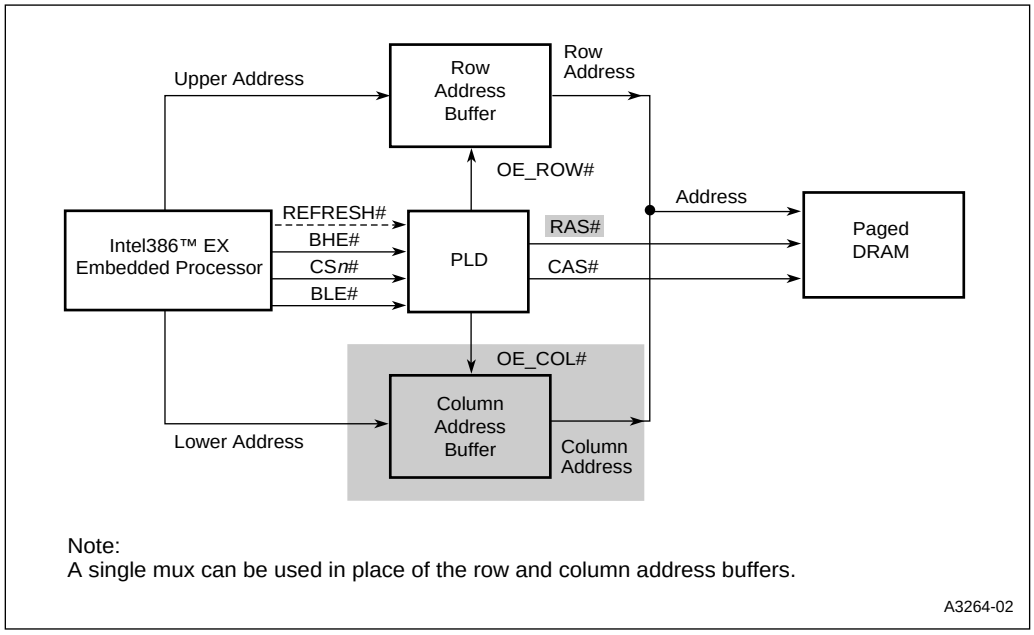


Figure 6-18. Intel386 EX Processor to Paged DRAM Interface

6.6.5 Non-Paged DRAM Interface

This interface is similar to the Paged DRAM Interface, except that in this case, the lower address bits are routed to the Row Address Buffer and the higher address bits to the Column Address Buffer. This is done to simplify the RAS#-Only Refresh logic. The PLD in this case enables the Row Address Buffer and asserts the RAS# signal (shaded sections in the figure) during a Refresh Cycle. Refer to Chapter 15, "REFRESH CONTROL UNIT," for more information.

A single multiplexer can be used instead of the separate row and column address buffers.

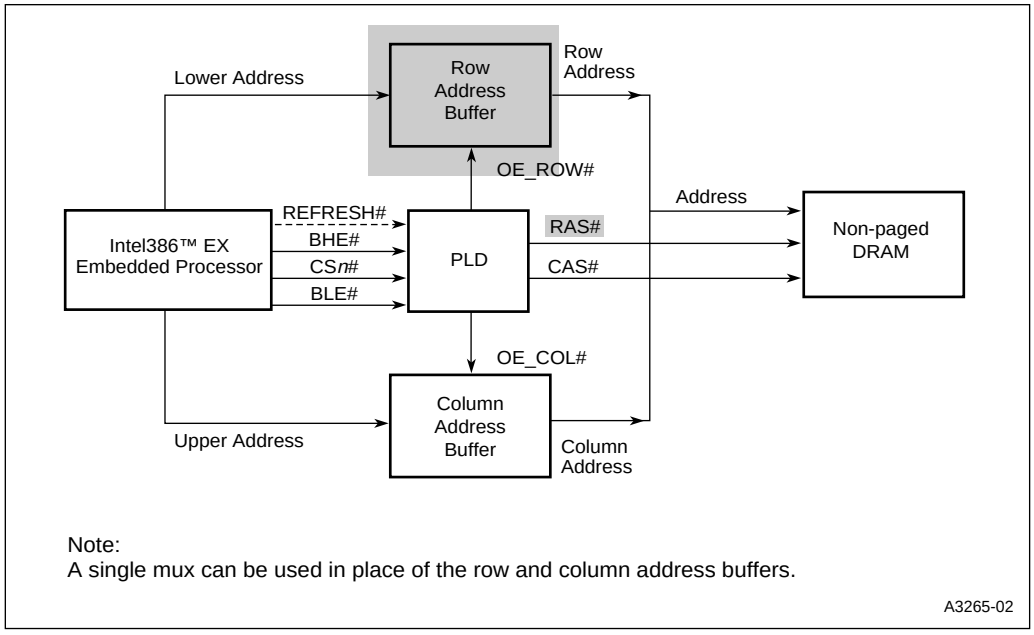


Figure 6-19. Intel386 EX Processor and Non-Paged DRAM Interface

7

SYSTEM MANAGEMENT MODE

CHAPTER 7
SYSTEM MANAGEMENT MODE

The Intel386™ EX processor provides a mechanism for system management with a combination of hardware and CPU microcode enhancements. For low power systems, the primary function of SMM is to provide a transparent means for power management. For systems where power management is not critical, SMM may be used for other functions such as alternate operating systems, debuggers, hard disk drive backup, or virtual I/O.

This chapter is organized as follows:

7.1 SYSTEM MANAGEMENT MODE OVERVIEW

An externally generated system management interrupt (SMI#) allows the execution of system-wide routines that are independent and transparent to the operating system. The system management mode (SMM) architectural extensions to the Intel386 CPU consist of the following elements:

- An interrupt input pin (SMI#) to invoke SMM
- An output pin (SMIACT#) to identify execution state
- A new instruction (RSM, executable only from SMM) to exit SMM

7.2 SMM HARDWARE INTERFACE

The Intel386 EX processor provides two pins for use in SMM systems: SMI# and SMIACT#.

7.2.1 System Management Interrupt Input (SMI#)

The SMI# input signal is used to invoke system management mode. SMI# is a falling edge triggered interrupt input signal and is the highest priority of all external interrupt sources. SMI# forces the core into SMM at the completion of the current instruction. SMI# has these characteristics:

- SMI# is not maskable.
- SMI# is recognized on an instruction boundary and at each iteration for repeat string instructions.
- SMI# does not break locked bus cycles.

- SMI# cannot interrupt currently executing SMM code. The processor latches the falling edge of a pending SMI# signal while the Intel386 EX processor is executing an existing SMI# (this allows one level of buffering). The nested SMI# is not recognized until after the execution of a resume instruction (RSM).
- SMI# brings the processor out of idle or powerdown mode.

7.2.2 SMM Active Output (SMIACT#)

This output indicates that the processor is operating in system management mode. It is asserted when the CPU initiates the SMM sequence and remains active (low) until the processor executes the RSM instruction (described in "Resume Instruction (RSM)" on page 7-15) to leave SMM. Before SMIACT# is asserted, the CPU waits until the end of the instruction boundary. SMIACT# is used to establish a new memory map for SMM operation. The processor supports this function by an extension to the internal chip-select unit. In addition, external logic can use this pin to qualify RESET and SMI#. SMIACT# never transitions during a pipelined bus cycle.

7.2.3 System Management RAM (SMRAM)

The SMM architecture requires that a partition of memory be set aside for the SMM driver. This is called the SMRAM. Several requirements must be met by the system:

- The address range of this partition must be, as a minimum, from 038000H to 03FFFFH (32 Kbytes).
- The address range from 03FE00H to 03FFFFH (512 bytes) is reserved for the CPU and must be RAM.
- The SMM handler must start execution at location 038000H. It is not relocatable.
- During normal operation the SMRAM is only accessible when the system is in SMM.
- During system initialization it must be possible to access the SMRAM in order to initialize it and possibly to install the SMM driver. Obviously, this must be done outside of SMM.
- When the SMRAM overlays other memory in the system, then address decoding and chip selects must allow the SMM driver to access the shadowed memory locations while in SMM.
- The SMRAM should not be accessible to alternate bus masters such as DMA.

These requirements are made to ensure that the SMM remains transparent to non-SMM code and to maintain uniformity across the various Intel processors that support this mode.

NOTE

It is possible for the designer of an embedded system to place the SMM driver code in read-only storage, as long as the address space between 03FE00H and 03FFFFH is writable.

The Intel386 EX processor does not support SMRAM relocation. Bit 17 of the SMM Revision Identifier (see "SMRAM State Dump Area" on page 7-14) indicates whether the processor sup-

ports the relocation of SMRAM. When this bit is set (1), the processor supports SMRAM relocation. When this bit is cleared (0), then the processor does not support SMRAM relocation. Since this device doesn't support SMRAM relocation, bit 17 of the SMM Revision Identifier is cleared. The SMRAM address space is fixed from 38000H to 3FFFFH.

7.3 SYSTEM MANAGEMENT MODE PROGRAMMING AND CONFIGURATION

7.3.1 Register Status During SMM

When the CPU recognizes SMI# on an instruction boundary, it waits for all write cycles to complete and asserts the SMIACT# pin. The processor then saves its register state to SMRAM space and begins to execute the SMM handler. The RSM instruction restores the registers, deasserts the SMIACT# pin, and returns to the user program.

Upon entering SMM, the processor's PE, MP, EM, TS and PG bits in CR0 are cleared, as shown in Table 7-1.

Table 7-1. CR0 Bits Cleared Upon Entering SMM

CR0 Bit	Mnemonic	Description	Function
0	PE	Protection Enable	0 = protection disabled 1 = protection enabled
1	MP	Math Coprocessor Present	0 = coprocessor not present 1 = coprocessor present
2	EM	Emulate Coprocessor	0 = coprocessor opcodes execute 1 = coprocessor opcodes generate a fault
3	TS	Task Switched	0 = coprocessor ESC opcode does not cause fault 1 = coprocessor ESC opcode causes fault
31	PG	Paging Enable	0 = paging disabled 1 = paging enabled

Debug register DR7 is also cleared, except for bits 11–15.

Internally, a descriptor register (invisible to the programmer) is associated with each programmer-visible segment register. Each descriptor register holds a 32-bit segment base address, a 32-bit segment limit, and other necessary segment attributes. When a selector value is loaded into a segment register, the associated descriptor register is automatically updated with the correct information. In Real mode, only the base address is updated directly (by shifting the selector value four bits to the left), since the segment maximum limit and attributes are fixed in Real mode. In Protected mode, the base address, the limit, and the attributes are all updated per the contents of the segment descriptor indexed by the selector. After saving the CPU state, the SMM State Save sequence sets the appropriate bits in the segment descriptor, placing the core in an environment similar to Real mode, without the 64 Kbyte limit checking.

In SMM, the CPU executes in a Real-like mode. In this mode, the CPU can access (read and write) any location within the 4 Gbyte logical address space. The physical address space is 64 Mbytes. The CPU can also perform a jump and a call anywhere within a 1 Mbyte boundary address space. In SMM, the processor generates addresses as it does in real mode; however, there

is no 64 Kbyte limit. The value loaded into the selector register is shifted to the left four bits and moved into its corresponding descriptor base, then added to the effective address. The effective address can be generated indirectly, using a 32-bit register. However, only 16 bits of the Extended Instruction Pointer (EIP) register are pushed onto the stack during calls, exceptions and INTR services. Therefore, when returning from calls, exceptions or INTRs, the upper 16 bits of the 32-bit EIP are zero. In an SMI# handler, the EIP should not be over the 64 Kbyte boundary. The 16-bit CS allows addressing within a 1 Mbyte boundary.

Instructions that explicitly access the stack, such as MOV instructions, can access the entire 4 Gbytes of logical address space by using a 32-bit address size prefix. However, instructions that implicitly access the stack, such as POP, PUSH, CALL, and RET, still have the 64 Kbytes limit.

After SMI# is recognized and the processor state is saved, the processor state is initialized to the default values shown in Table 7-2.

Table 7-2. SMM Processor State Initialization Values

Register	Content
General Purpose Register	Unpredictable
EFLAGS	00000002H
EIP	00008000H
CS Selector	3000H
DS,ES,FS,GS,SS Selectors	0000H
CS Descriptor Base	00030000H
DS,ES,FS,GS,SS Descriptor Base	00000000H
CS,DS,ES,FS,GS,SS Descriptor Limit	0FFFFFH
DS,ES,FS,GS,SS Attributes	16-bit
CR0	Bits 0, 1, 2, 3, 31 cleared
DR6	Unpredictable
DR7	Bits 0–10,16–31 cleared

When a valid SMI# is recognized on an instruction execution boundary, the CPU immediately begins execution of the SMM State Save sequence, asserting SMIACT# low (unless the CPU is in a shutdown condition). The CPU then starts SMI# handler execution. An SMI# cannot interrupt a CPU shutdown. The SMI# handler always starts at 38000H. When there are multiple causes of SMI#s, only one SMI# is generated, thereby ensuring that SMI#s are not nested.

7.3.2 System Management Interrupt

The Intel386 EX processor extends the standard Intel386 microprocessor architecture by adding a new feature called the system management interrupt (SMI#). This section describes in detail how the system designer uses SMI#.

The execution unit recognizes an SMI# (falling edge) on an instruction boundary (see instruction #3 in Figure 7-1). After all CPU bus cycles have completed, including pipelined cycles, the state

of the CPU is saved to the SMM State Dump Area. After executing a RSM instruction, the CPU proceeds to the next application code instruction (see instruction #4 in Figure 7-1). SMM latency is measured from the falling edge of SMI# to the first ADS# where SMIACT# is active (see Figure 7-2).

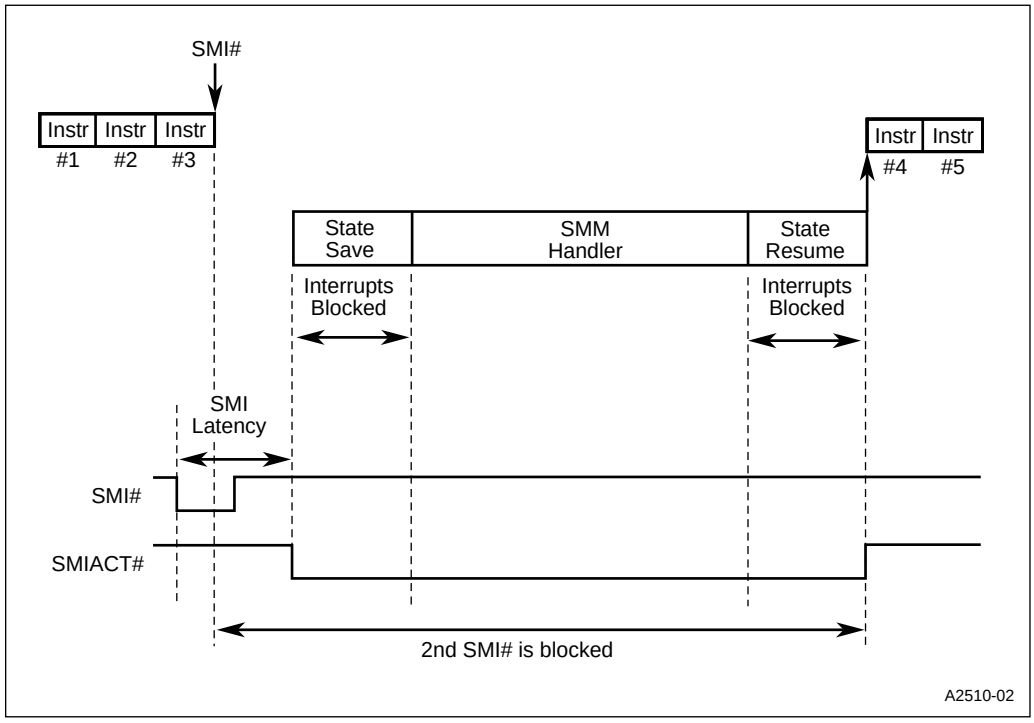
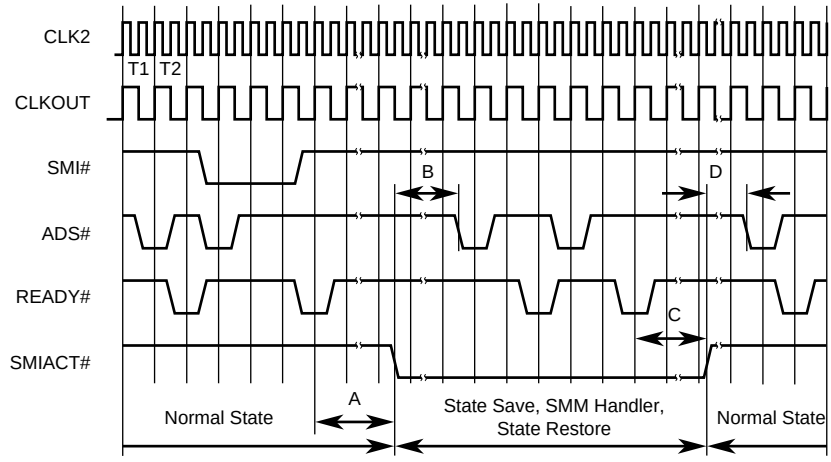


Figure 7-1. Standard SMI#

The SMM handler may optionally enable the NMI interrupt, but NMI is disabled when the SMM handler is entered. (Note that the CPU does not recognize NMI while executing the SMM State Save sequence or SMM State Resume sequence.) NMI is always enabled following the completion of the first interrupt service routine (ISR) or exception handler.

Even when the processor is in SMM, address pipelined bus cycles can be performed correctly by asserting NA#. Pipelined bus cycles can also be performed immediately before and after SMIACT# assertion. The numbers in Figure 7-2 also reflect a pipelined bus cycle.



A = 1 CLK min, B = 20 CLK min, C = 16 CLK min, D = 4 CLK min

A2512-02

Figure 7-2. SMIACT# Latency

NOTE

Even if bus cycles are pipelined, the minimum clock numbers are guaranteed.

7.3.2.1 SMI# Priority

When more than one exception or interrupt is pending at an instruction boundary, the processor services them in a predictable order. The priority among classes of exception and interrupt sources is shown in Table 7-3. The processor first services a pending exception or interrupt from the class that has the highest priority, transferring execution to the first instruction of the handler. Lower priority exceptions are discarded; lower priority interrupts are held pending. Discarded exceptions are reissued when the interrupt handler returns execution to the point of interruption. SMI# has the following relative priority, where 1 is highest and 11 is lowest:

Table 7-3. Relative Priority of Exceptions and Interrupts

1 (Highest priority)	Double Fault
2	Segmentation Violation
3	Page Fault
4	Divide-by-zero
5	SMI#
6	Single-step
7	Debug
8	ICE Break
9	NMI
10	INTR
11 (Lowest Priority)	I/O Lock

7.3.2.2 System Management Interrupt During HALT Cycle

Since SMI# is an asynchronous signal, it may be generated at any time. A condition of interest arises when an SMI# occurs while the CPU is in a HALT state. To give the system designer maximum flexibility, the processor allows an SMI# to optionally exit the HALT state. Figure 7-3 shows that the CPU normally re-executes the HALT instruction after RSM; however, by modifying the HALT restart slot in the SMM State Dump area, the SMM handler can redirect the instruction pointer past the HALT instruction.

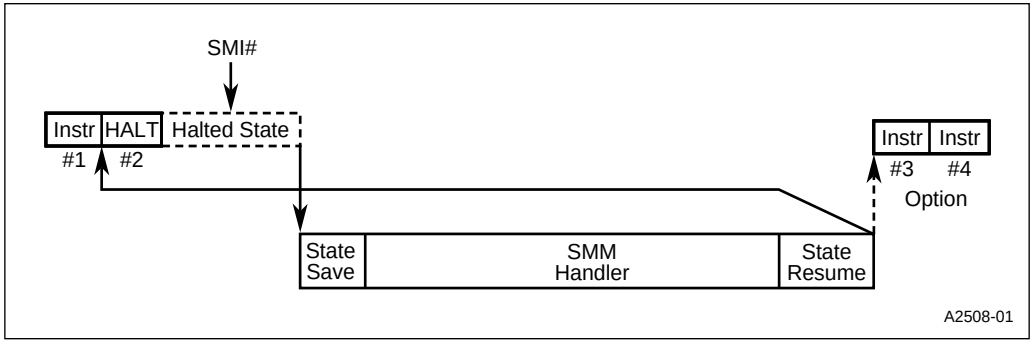


Figure 7-3. SMI# During HALT

7.3.2.3 HALT Restart

It is possible for SMI# to break into the HALT state. In some cases the application might want to return to the HALT state after RSM. The SMM architecture provides the option of restarting the HALT instruction after RSM.

The word at address 03FF02H is the HALT restart slot. The processor sets bit 0 of this location when the processor is in the HALT state while the SMI# occurred. If the SMM driver leaves this bit set, then the processor re-enters the HALT state when it exits from SMM. When the driver clears this bit, the processor continues execution with the instruction just after the interrupted HALT instruction.

7.3.2.4 System Management Interrupt During I/O Instruction

Like the HALT restart feature, the processor allows restarting I/O cycles which have been interrupted by an SMI#. This gives the system designer the option of performing a hardware I/O cycle restart without having to modify either application, operating system, or BIOS software. (See Figure 7-4.)

When a SMI# occurs during an I/O cycle, it then becomes the responsibility of the SMM handler to determine the source of the SMI#. If, for example, the source is the powered down I/O device, the SMM handler would power up the I/O device and reinitialize it. The SMM handler would then write 0FFH to the I/O restart slot in the SMM State Dump area and the RSM instruction would then restart the I/O instruction.

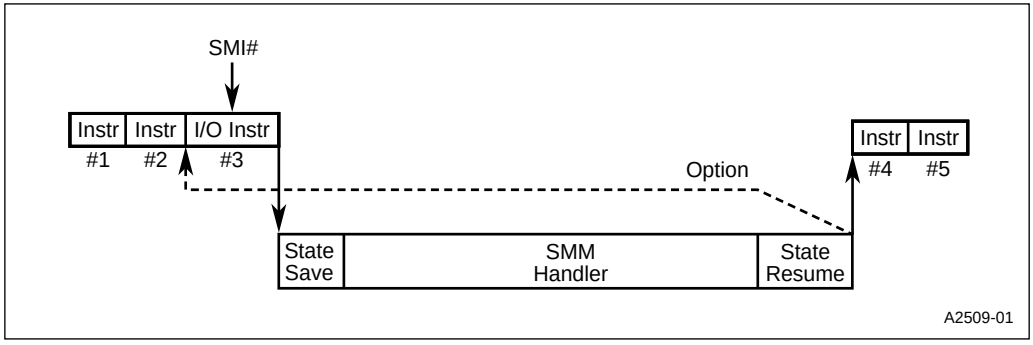


Figure 7-4. SMI# During I/O Instruction

The SMI# input signal can be asynchronous; as a result, SMI# must be valid at least three clock periods before READY# is asserted for it to be recognized right after the current bus cycle. SMI# must be sampled valid for at least two clocks, with the other clock used to internally arbitrate for control. See Figure 7-5 for details. (Note that this diagram is only for I/O cycles and memory data read cycles.)

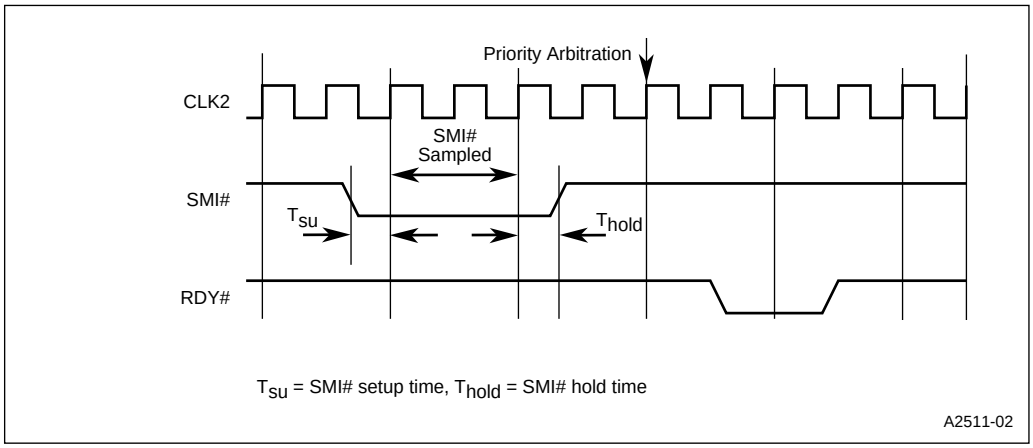


Figure 7-5. SMI# Timing

7.3.2.5 I/O Restart

Bit 16 of the SMM Revision Identifier is set (1) indicating that this device does support the I/O trap restart extension to the SMM base architecture.

The I/O trap restart slot provides the SMM handler the option of automatically re-executing an interrupted I/O instruction using the RSM instruction. When the RSM instruction is executed with the I/O trap restart slot set to a value of 0FFH, the CPU automatically re-executes the I/O instruction that the SMI# has trapped. If the slot contains 00H when the RSM instruction is executed, the CPU does not re-execute the I/O instruction. This slot is initialized to 00H during an SMI#. It is the SMM handler's responsibility to load the I/O trap restart slot with 0FFH when restart is desired.

NOTE

The SMM handler must **not** set the I/O trap restart slot to 0FFH when the SMI# is not asserted on an I/O instruction boundary, because this causes unpredictable results.

7.3.3 SMM Handler Interruption

7.3.3.1 Interrupt During SMM Handler

When the CPU enters SMM, both INTR and NMI are disabled (Figure 7-6). The SMM handler may enable INTR by executing the STI instruction. NMI is enabled after the completion of the first interrupt service routine (software or hardware initiated ISR) or exception handler within the SMM handler. Software interrupt and exception instructions are not blocked during the SMM handler.

The SMM feature can be used without any other interrupts. INTR and NMI are blocked by the system during SMI#, unless enabled by software. If INTR or NMI are not enabled during SMM,

then any pending INTR and NMI is serviced after completion of RSM instruction execution. Only one INTR and one NMI can be pending.

The SMM handler may choose to enable interrupts to take advantage of device drivers. Since interrupts were enabled while under control of the SMM handler, the signal SMIACT# continues to be asserted. If the system designer wants to take advantage of existing device drivers that leverage interrupts, the memory controller must take this into account.

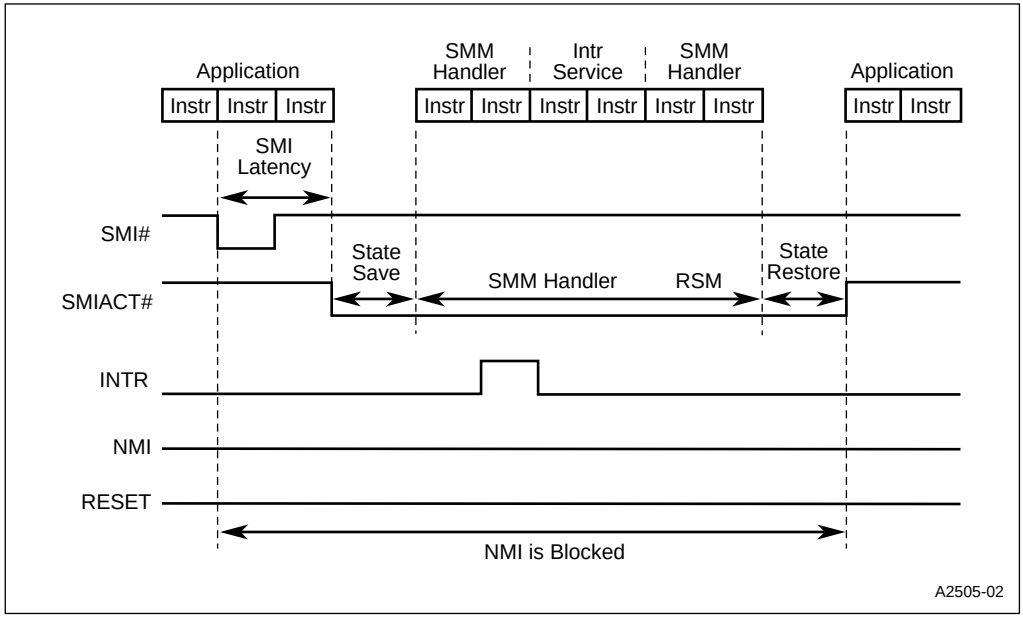


Figure 7-6. Interrupted SMI# Service

7.3.3.2 HALT During SMM Handler

The system designer may wish to place the system into a HALT condition while in SMM. The CPU allows this condition to occur; however, unlike a HALT while in normal mode, the CPU internally blocks INTR and NMI from being recognized until after the RSM instruction is executed. When a HALT needs to be breakable in SMM, the SMM handler must enable INTR and NMI before a HALT instruction execution. NMI is enabled after the completion of the first interrupt service routine within the SMM handler.

After the SMM handler has enabled INTR and NMI, the CPU exits the HALT state and returns to the SMM handler when INTR or NMI occurs. See Figure 7-7 for details.

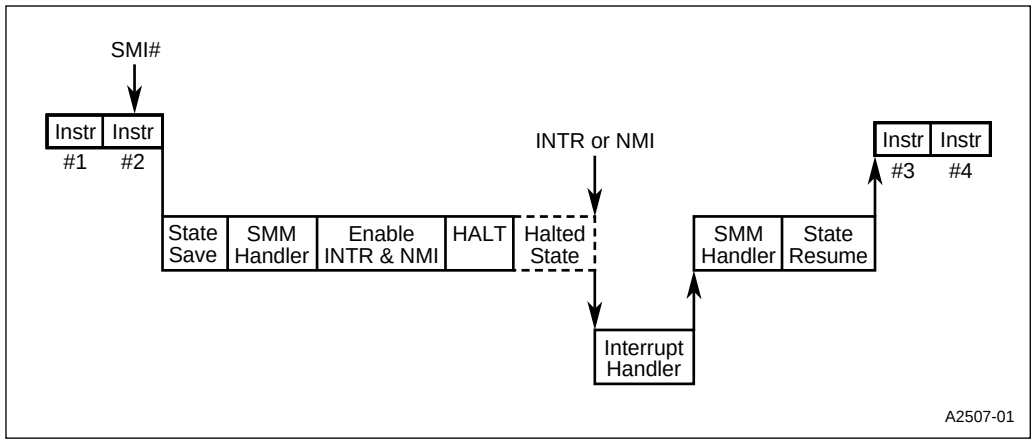


Figure 7-7. HALT During SMM Handler

7.3.3.3 Idle Mode and Powerdown Mode During SMM

Both Idle Mode and Powerdown Mode may be used while in SMM. Entering and exiting either of these power management modes from SMM is identical to entering or exiting from normal mode. The interaction between SMM and power management modes is described in Chapter 8.

7.3.3.4 SMI# During SMM Operation

If the SMI# request is asserted during SMM operation, the second SMI# cannot nest the currently executing SMM. The second SMI# request is latched, and held pending by the CPU. Only one SMI# request can be pending. After RSM execution is completed, the pending SMI# is serviced. At this time, SMIACT# is deasserted once at completion of RSM, then asserted again for the second SMI#.

When the SMM handler polls the various SMI# sources for one of the SMI# triggers, and two SMI# sources are found in the SMI# generation circuit, the SMM handler services both SMI# sources and executes a RSM instruction. In this SMM handler, if the SMI# generation circuit asserts the second SMI# during the first SMI# service routine, the second SMI# is pending. Next, the SMM handler finds and services two SMI# sources. After the CPU completes the RSM execution, the pending SMI# (second SMI#) is generated, but there is nothing to service because the second SMI# was serviced during the first SMM handler. This unnecessary SMI# transaction requires a few hundred clocks. There may be some performance degradation if this example occurs frequently. For good performance, it is the responsibility of the SMI# generation circuitry to manage multiple SMI# assertions.

7.3.4 SMRAM Programming

7.3.4.1 Chip-select Unit Support for SMRAM

The internal chip-select unit (CSU) has been extended to support the SMRAM by using bit 10 in each Low Address (CASMM) and Low Mask register (CMSMM). The CSU acts on these bits

exactly as if they represented another address line. The following options are supported by the chip select unit:

CASMM	CMSMM	Chip select active:
0	0	During normal mode only
1	0	During SMM only
X	1	During normal mode or SMM

To see how this extension of the CSU supports the SMRAM requirements, consider an embedded system which has 1 Mbyte of 16-bit wide EPROM in the region 03F00000H to 03FFFFFFH and 1 Mbyte of 16-bit wide RAM in the region 00000000H to 000FFFFFH. A single 32 Kbyte RAM in the region 00038000H to 0003FFFFH is added to support SMM. The chip selects for this system during normal operation would be programmed as follows:

REGION	CA25:11	CM25:11	CASMM	CMSMM	BS16
EPROM	11 1111 0000 0000 0	00 0000 1111 1111 1	0	0	1
RAM	00 0000 0000 0000 0	00 0000 1111 1111 1	0	0	1
SMRAM	00 0000 0011 1000 0	00 0000 0000 0111 1	1	0	0

Each row in the above table represents a region of memory and its associated chip select logic. During initialization, these same chip selects could be programmed as follows:

REGION	CA25:11	CM25:11	CASMM	CMSMM	BS16
EPROM	11 1111 0000 0000 0	00 0000 1111 1111 1	0	0	1
RAM	00 0000 0000 0000 0	00 0000 1111 1111 1	0	0	1
SMRAM	00 0001 0011 1000 0	00 0000 0000 0111 1	0	0	0

Only the SMRAM row has been changed; the SMRAM chip select has been redirected to the region 013F800H to 013FFFFH and the CASMM bit has been cleared. This allows the initialization software to set up the SMRAM without entering the SMM. Note that the external design of the system must guarantee that an SMI# cannot occur while the SMRAM is being initialized.

If the SMM driver needs to access the memory shadowed under the SMRAM, the chip selects can be reconfigured as follows:

REGION	CA25:11	CM25:11	CASMM	CMSMM	BS16
EPROM	11 1111 0000 0000 0	00 0000 1111 1111 1	0	0	1
RAM	00 0001 0000 0000 0	00 0000 1111 1111 1	0	1	1
SMRAM	00 0000 0011 1000 0	00 0000 0000 0111 1	1	0	0

This leaves the SMRAM in place but moves the normal RAM into the partition 0100000H to 01FFFFFH. The CASMM bit is masked so that the RAM is selected independent of SMM.

7.3.4.2 SMRAM State Dump Area

The SMM State Save sequence asserts SMIACT#. This mechanism indicates to internal modules that the CPU has entered and is currently executing SMM. The resume (RSM) instruction is only valid when in SMM. SMRAM space is an area located in the memory address range 38000H–3FFFFH. The SMRAM area cannot be relocated internally. SMRAM space is intended for access by the CPU only, and should be accessible only when SMM is enabled. This area is used by the SMM State Save sequence to save the CPU state in a stack-like fashion from the top of the SMRAM area downward.

The CPU state dump area always starts at 3FFFFH and ends at 3FE00H. The following is a map of the CPU state dump in the SMRAM.

Hex Address	Name	Description
03FFFC	CR0	Control flags that affect the processor state
03FFF8	CR3	Page directory base register
03FFF4	EFLGS	General condition and control flags
03FFF0	EIP	Instruction pointer
03FFEC	EDI	Destination index
03FFE8	ESI	Source index
03FFE4	EBP	Base pointer
03FFE0	ESP	Stack pointer
03FFDC	EBX	General register
03FFC8	EDX	General register
03FFD4	ECX	General register
03FFD0	EAX	General register
03FFCC	DR6	Debug register; contains status at exception
03FFC8	DR7	Debug register; controls breakpoints
03FFC4	TR	Task register; used to access current task descriptor
03FFC0	LDTR	Local descriptor table pointer
03FFBC	GS	General-purpose segment register
03FFB8	FS	General-purpose segment register
03FFB4	DS	Data segment register
03FFB0	SS	Stack segment register
03FFAC	CS	Code segment register
03FFA8	ES	General-purpose segment register
03FFA7–03FF04	—	Reserved
03FF02	—	Halt restart slot
03FF00	—	I/O trap restart slot
03FEFC	—	SMM revision identifier (10000H)
03FEFB–03FE00	—	Reserved

The programmer should not modify the contents of this area in SMRAM space directly. SMRAM space is reserved for CPU access only and is intended to be used only when the processor is in SMM.

7.3.5 Resume Instruction (RSM)

After an SMI# request is serviced, the RSM instruction must be executed to allow the CPU to return to an application transparently after servicing the SMI#. When the RSM instruction is executed, it restores the CPU state from SMRAM and passes control back to the operating system. The RSM instruction uses the special opcode of 0FAAH. The RSM instruction is reserved for the SMI# handler and should only be executed by the SMI# handler. Any attempt to execute the RSM outside of SMM mode results in an invalid opcode exception. At the end of the RSM instruction, the processor drives SMIACT# high, indicating the end of an SMM routine.

7.4 THE Intel386 EX PROCESSOR IDENTIFIER REGISTERS

The processor has two identifier registers: the Component and Revision ID register and the SMM Revision ID register. The component ID is 23H; the component revision ID is 09H. This register can be read as 2309H. The SMM revision identifier is 10000H.

7.5 PROGRAMMING CONSIDERATIONS

7.5.1 System Management Mode Code Example

The following code example contains these software routines.

SerialWriteStr2 Located in SMRAM upon program execution, this routine loops endlessly while writing a character "X" out the serial port on the EV386EX board.

SerialWriteStr Located in the main program in FLASH, this routine loops endlessly while writing a string out the serial port before entering SMM.

InitSIO Initializes the serial port including the mode, baud rate, and clock rate.

MAIN Executes the program once it is located in FLASH. It also configures chip selects, copies SMM handler to SMRAM, and loops endlessly until an SMI# is issued.

See Appendix C for the included header files.

```
#include "80386EX.h"
#include "EV386EX.h"
#include <string.h>
#include <conio.h>
#include <dos.h>

#if _DEBUG_ == 0           // _DEBUG_ must be defined on the command line
#define  SIO_PORT  SIO_1   // The debugger uses SIO_0 for host communications
#else                      // Under the debugger we must avoid using SIO_0
#define  SIO_PORT  SIO_0
#endif

#define BAUD_CLKIN  1843200L  // Clock rate of COMCLK, i.e., External clocking,
extern char far SMMString[];
extern void InitEXSystem(void);

int   DataSeg;             // For assembly data segment register init.
BYTE  Buf[20];

/********************** Function SerialWriteStr2 *************************
   Parameters:
      None
   Returns:
      None
   Assumptions:
        Not called from main. This function is used as a jump point and is
        relocated by the main to 38000H (SRAM) for SMM.
   Real/Protected Mode:
       No changes required
```

```
-------------------------------------------------------------------------*/

void SerialWriteStr2()
         /* Loops while writing a char out to the serial port */
{
   _asm
   {
      mov              ax,0x3900
      mov              ss,ax
      mov sp,0x100

   Forever:
      mov              dx,0xf4fd
   TstStatus:
      in               al,dx
      testal,0x20
      je               TstStatus
        // Code below is same as _SetEXRegByte(TransmitPortAddr,'X')
      mov              ax,'X'
      mov              dx,0xf4f8
      out              dx, al
      jmp              Forever
   }
}
/*********************** Function SerialWriteStr ************************
   Parameters:
      Unit        Unit number of the serial port.  0 for SIO port 0, 1 for SIO
                  port 1.
      *str        Character string to be written out the serial port.
   Returns:
      None
   Assumptions:
        None
   Real/Protected Mode
 -------------------------------------------------------------------------*/

void SerialWriteStr(int Unit, const char far *str)
{
   WORD TransmitPortAddr;
   WORD StatusPortAddr;

        // Set Port base, based on serial port used
   TransmitPortAddr = (Unit ? TBR1 : TBR0);
   StatusPortAddr =  (Unit ? LSR1 : LSR0);

   for( ; *str != '\0'; str++)
   {
        // Wait until buffer is empty
      while(!(_GetEXRegByte(StatusPortAddr) & SIO_TX_BUF_EMPTY)) ;
        // Write Character
      _SetEXRegByte(TransmitPortAddr,*str);
   }
```

```
}
/************************** Function InitSIO *****************************

   Parameters:
      Unit        Unit number of the serial port.  0 for SIO port 0, 1 for SIO
                  port 1.
      Mode        Defines parity, number of data bits, number of stop bits...
                  Reference Serial Line Control register for various options
      ModemCntrl  Defines the operation of the modem control lines
      BaudRate    Specifies baud rate.  The baud divisor value is calculated
                  based on clocking source and clock frequency.  The clocking
                  frequency is set by calling the InitializeLibrary function.
      ClockRate   Specifies the serial port clocking rate, for internal clocking
                  = CLK2 for external = COMCLK
   Returns:   Error Codes
      E_INVAILD_DEVICE     -- Unit number specifies a non-existing device
      E_OK                 -- Initialized OK, No error.

   Assumptions:
      SIOCFG  Has already been configured for Clocking source and Modem control
      source

      REMAPCFG register has Expanded I/O space access enabled (ESE bit set).
      The processor Port pin are initialized separately.

   Real/Protected Mode
      No changes required.
-------------------------------------------------------------------------*/

int InitSIO(int Unit,
            BYTE Mode,
            BYTE ModemCntrl,
            DWORD BaudRate,
            DWORD BaudClkIn)
{
   WORD SIOPortBase;
   WORD BaudDivisor;

      // Check for valid unit
   if(Unit > 1)
      return E_INVALID_DEVICE;

      // Set Port base based on serial port used
   SIOPortBase = (Unit ? SIO1_BASE : SIO0_BASE);

      // Initialized Serial Port registers
      // Calculate the baud divisor value, based on baud clocking
   BaudDivisor = (WORD)(BaudClkIn / (16*BaudRate));

      // Turn on access to baud divisor register
   _SetEXRegByte(SIOPortBase + LCR, 0x80);
      // Set the baud rate divisor register, High byte first
```

```
   _SetEXRegByte(SIOPortBase + DLH, HIBYTE(BaudDivisor) );
   _SetEXRegByte(SIOPortBase + DLL, LOBYTE(BaudDivisor) );

      // Set Serial Line control register
   _SetEXRegByte(SIOPortBase + LCR, Mode); // Sets Mode and resets the
                                           // Divisor latch

      // Set modem control bits
   _SetEXRegByte(SIOPortBase + MCR, ModemCntrl);

   return E_OK;
}
/****************************** MAIN *********************************/
   Parameters:
      None
   Returns:
      None
   Assumptions:
      None
   Real/Protected Mode
      No changes required.
-------------------------------------------------------------------------*/

#ifndef SetEXRegWordInline
#define SetEXRegWordInline(address, word) \
      _asm mov dx, address;      \
      _asm mov ax, word;         \
      _asm out dx, ax;
#endif

void main(void)
{

    InitSIO(SIO_PORT,                   // Which Serial Port
            SIO_8N1,                    // Mode, 8-data, no parity, 1-stop
            SIO_MCR_RTS+SIO_MCR_DTR,    // Modem line controls
            9600,                       // Baud Rate
            BAUD_CLKIN);                // Baud Clocking Rate


    _asm                               // Store registers to preserve values
     {
     push DI
     push SI
     push DS
     push ES
      }

   SetEXRegWordInline(CS4ADL, 0x702);     // Configure chip select 4
   SetEXRegWordInline(CS4ADH, 0x0);
   SetEXRegWordInline(CS4MSKL, 0xFC01);
   SetEXRegWordInline(CS4MSKH, 0x0);
```

```
   SetEXRegWordInline(CS2ADL,0x08700);    // Enables SRAM as memory
   SetEXRegWordInline(CS2ADH,0x3);
   SetEXRegWordInline(CS2MSKL,0x07C01);
   SetEXRegWordInline(CS2MSKH,0x00);


    _asm                                  // Copy SMM_EXAM.BIN code into SRAM
    {
    mov ax,0x3800                        // Starting address for SMM_EXAM file
    mov es,ax                            // to be placed

    mov ax,seg SerialWriteStr2           // Address where SMM_EXAM is located
    mov ds,ax

    mov cx,0x100                         // Length of SMM_EXAM file in bytes
    mov si,offset SerialWriteStr2
    mov di,0
    rep movsb
    }

    SetEXRegWordInline(CS2MSKL,0x7801);  // Resets SRAM to enabled in SMM only

    _asm                                 // Restore register values
    {
    pop DI
    pop SI
    pop DS
    pop ES
    }
           // Loop endlessly and display another serial message
   while(1)                              // Serial Write Loop
      {
        SerialWriteStr(SIO_PORT,SMMString);
      }
}
/*************************** END MAIN *********************************/
```

8

CLOCK AND POWER MANAGEMENT UNIT

CHAPTER 8
CLOCK AND POWER MANAGEMENT UNIT

The clock generation circuitry provides uniform, nonoverlapping clock signals to the core and integrated peripherals. The power management features control the clock signals to provide power conservation options.

This chapter is organized as follows:

8.1 OVERVIEW

The clock and power management unit (Figure 8-1) includes clock generation, power management, and system reset circuitry. It also provides a clock output signal (CLKOUT) for synchronizing external logic to the processor's system clock. CLKOUT is the PH1P clock.

8.1.1 Clock Generation Logic

An external oscillator must provide an input signal to CLK2, which provides the fundamental timing for the processor. As Figure 8-1 shows, the clock generation circuitry includes two divide-by-two counters and a programmable clock divider. The first divide-by-two counter divides the CLK2 frequency to generate two clocks (PH1 and PH2). For power management, independent clock signals are routed to the core (PH1C and PH2C) and to the internal peripherals (PH1P and PH2P).

The second divide-by-two counter divides the processor clock to generate a clock input (SERCLK) for the baud-rate generators of the asynchronous and synchronous serial I/O units. The SERCLK frequency is half the internal clock frequency, or CLK2/4.

The programmable divider generates a prescaled clock (PSCLK) input for the timer/counter and synchronous serial I/O units. The maximum PSCLK frequency is the internal clock frequency divided by 2 (CLK2/4) and the minimum is the internal clock frequency divided by 513 (CLK2/1026).

Three of the internal peripherals have selectable clock sources.

- The asynchronous serial I/O (SIO) unit can use either the SERCLK signal or an external clock (connected to the COMCLK pin) as its clock source.
- The synchronous serial I/O (SSIO) unit can use either the SERCLK signal or the PSCLK signal.
- The timer/counters can use either the PSCLK signal or an external clock connected to the TMRCLK*n* input pin.

The individual peripheral chapters explain how to select the clock inputs.

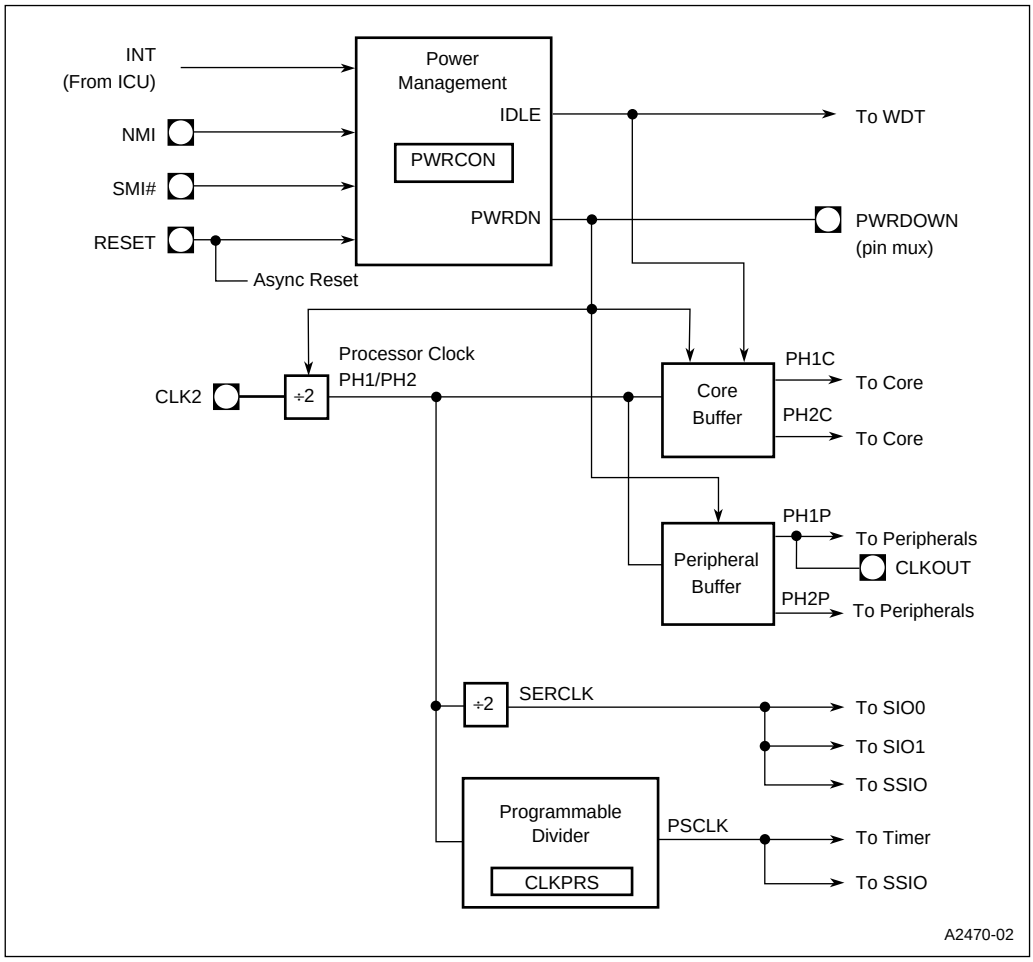


Figure 8-1. Clock and Power Management Unit Connections

The signal from the RESET pin is also routed to the clock generation unit, which synchronizes the processor clock with the falling edge of the RESET signal and provides a synchronous internal RESET signal to the rest of the device. The RESET falling edge can occur in either PH1 or PH2. If RESET falls during PH1, the clock generation circuitry inserts a PH2, so that the next phase is PH1 (Figure 8-2). If it falls during PH2, the next phase is automatically PH1.

NOTE

The RESET signal must be high for 16 CLK2 cycles to properly reset the processor.

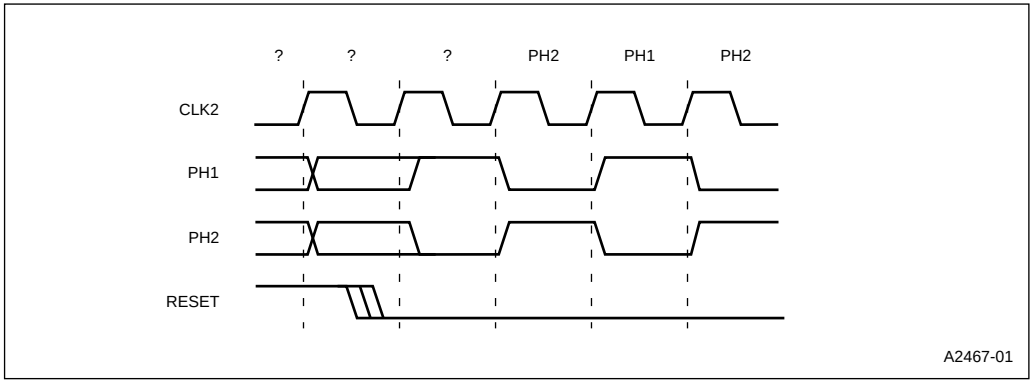


Figure 8-2. Clock Synchronization

In addition to internal synchronization, a CLKOUT (PH1P) clock output is provided to enable external circuitry to maintain synchronization with the Intel386 EX processor. Since it is one of the peripheral clock signals, it remains active during idle mode, but is driven low during powerdown mode.

8.1.2 Power Management Logic

The power management circuitry provides two power management modes:

Idle Mode — Idle mode freezes the core clocks, but leaves the peripheral clocks running. Idle mode can reduce power consumption by about half, depending on peripheral usage.

Powerdown mode — Powerdown mode freezes both the core and peripheral clocks, reducing current to leakage current (microamps). Peripherals that are clocked externally (SIO, Timers, SSIO) continue to run. If inputs are toggling, power consumption is higher.

To prepare for a power management mode, program the power control register as described in “Controlling Power Management Modes” on page 8-8, then execute a HALT instruction. The de-

vice enters the programmed mode when the HALT cycle is terminated by a valid READY#. This READY# may be generated either internally or externally.

A device reset, an NMI or SMI#, or any unmasked interrupt request from the interrupt control unit causes the device to exit the power management mode. After a reset, the CPU starts executing instructions at 3FFFFF0H and the device remains in normal operation. After an interrupt, the CPU executes the interrupt service routine, then returns to the instruction following the HALT that prompted the power management mode. Unless software modifies the power control register, the next HALT instruction returns the device to the programmed power management mode.

8.1.2.1 SMM Interaction with Power Management Modes

When the processor receives an SMI# interrupt while it is in idle or powerdown mode, it exits the power management mode and enters System Management Mode (SMM). Upon exiting SMM, software can check whether the processor was in a halt state before entering SMM. If it was, software can set a flag that returns the processor to the halt state when it exits SMM. Assuming the power control register bits were not altered in SMM, the processor re-enters idle or powerdown when it exits SMM. Figure 8-3 illustrates the relationships among these modes.

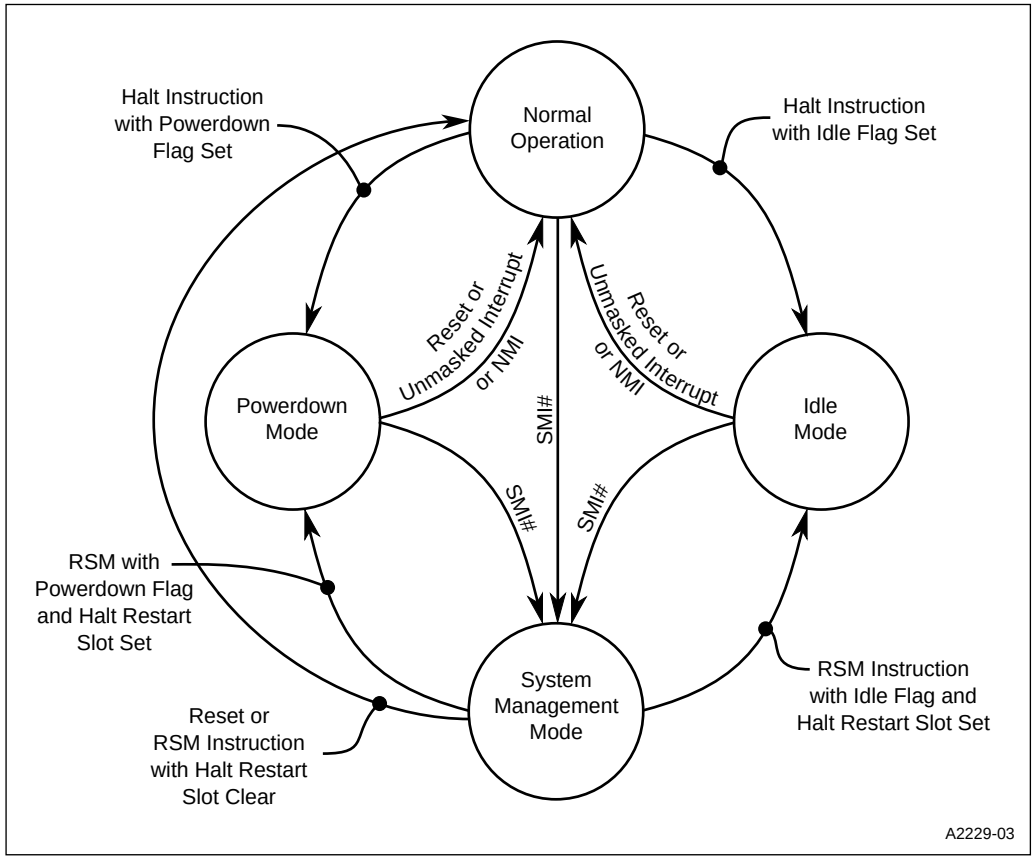


Figure 8-3. SMM Interaction with Idle and Powerdown Modes

8.1.2.2 Bus Interface Unit Operation During Idle Mode

The bus interface unit (BIU) can process DMA, DRAM refresh, and external hold requests during idle mode. When the first request occurs, the core wakes up long enough to relinquish bus control to the bus arbiter, then returns to idle mode. For the remaining time in idle mode, the bus arbiter controls the bus. DMA, DRAM refresh, and external hold requests are processed in the same way as during normal operation.

8.1.2.3 Watchdog Timer Unit Operation During Idle Mode

When the watchdog timer unit is in system watchdog mode, idle mode stops the down-counter. Since no software can run while the CPU is idle, a software watchdog is not needed. When it is in bus monitor or general-purpose timer mode, the watchdog timer unit continues to run while the device is in idle mode. (Chapter 17 describes the watchdog timer unit.)

8.1.3 Clock and Power Management Registers and Signals

Table 8-1 lists the registers and Table 8-2 list the signals associated with the clock and power management unit.

Table 8-1. Clock and Power Management Registers

Register	Expanded Address	Description
CLKPRS	0F804H	Clock Prescale: This register contains the programmed divisor value used to generate PSCLK from the internal clock.
PWRCON	0F800H	Power Control: This register selects the power management mode and internal ready options.

Table 8-2. Clock and Power Management Signals

Signal	Device Pin or Internal Signal	Description
CLK2	Device pin	Input Clock: Connect an external clock to this pin to provide the fundamental timing for the microprocessor.
CLKOUT	Device pin	Output Clock: CLKOUT is a Phase 1 output clock (PH1P)
IDLE	Internal signal	Idle Output (to the Watchdog Timer Unit): IDLE indicates that the device is in idle mode.
INTR	Internal signal	Interrupt Input (from the Interrupt Control Unit): INT causes the device to exit powerdown or idle mode.
NMI	Device pin	Nonmaskable Interrupt Input: NMI causes the device to exit powerdown or idle mode.
PSCLK	Internal signal	Prescaled Clock Output: PSCLK is one of two possible clock inputs for the SSIO baud-rate generator and the Timer/counter Unit. The PSCLK frequency is controlled by the CLKPRS register.
PWRDOWN	Device pin	Powerdown Output (multiplexed with P3.6): A high state on the PWRDOWN pin indicates that the device is in powerdown mode.
RESET	Device pin	System Reset Input: This signal resets the processor and causes the device to exit powerdown or idle mode.
SERCLK	Internal signal	Serial Clock Output: SERCLK is one of two possible clock inputs for the SIO or SSIO baud-rate generator. The SERCLK frequency is one-fourth the CLK2 frequency.
SMI#	Device pin	System Management Interrupt Input: SMI# causes the device to exit powerdown or idle mode and causes the processor to enter System Management Mode.

8.2 CONTROLLING THE PSCLK FREQUENCY

The PSCLK signal can provide a 50% duty cycle prescaled clock to the timer/counter and SSIO units. This feature is useful for providing various frequencies, including a 1.19318 MHz output for a PC-compatible system timer, or speaker tone generator. Determine the required prescale value using the following formula, then write this value to the CLKPRS register (Figure 8-4).

$$\text{Prescale value} = \frac{\text{internal clock frequency (CLK2/2)}}{\text{desired PSCLK frequency}} - 2$$

Clock Prescale Register
CLKPRS
(read/write)

Expanded Addr: **F804H**
ISA Addr: **—**
Reset State: **0000H**

15							8
—	—	—	—	—	—	—	PS8

7							0
PS7	PS6	PS5	PS4	PS3	PS2	PS1	PS0

Bit Number	Bit Mnemonic	Function
15–9	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
8–0	PS8:0	Prescale Value: These bits determine the divisor that is used to generate PSCLK. Legal values are from 0000H (divide by 2) to 01FFH (divide by 513). divisor = PS8:0 + 2

Figure 8-4. Clock Prescale Register (CLKPRS)

To change the frequency of PSCLK, write a new value to the CLKPRS register. The new frequency takes effect at the first high-to-low transition of PSCLK after CLKPRS has been changed.

8.3 CONTROLLING POWER MANAGEMENT MODES

Two power management modes are available: idle and powerdown. These modes are clock distribution functions controlled by the power control register (PWRCON), shown in Figure 8-5.

Power Control Register
PWRCON
(read/write)

Expanded Addr: **F800H**
ISA Addr: **—**
Reset State: **00H**

7							0
—	—	—	—	WDTRDY	HSREADY	PC1	PC0

Bit Number	Bit Mnemonic	Function
7–4	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
3	WDTRDY	Watch Dog Timer Ready: 0 = An external READY must be generated to terminate the cycle when the WDT times out in Bus Monitor Mode. 1 = Internal logic generates READY# to terminate the cycle when the WDT times out in Bus Monitor Mode.
2	HSREADY	Halt/Shutdown Ready: 0 = An external ready must be generated to terminate a HALT/Shutdown cycle. 1 = Internal logic generates READY# to terminate a HALT/Shutdown cycle.
1–0	PC1:0	Power Control: Program these bits, then execute a HALT instruction. The device enters the programmed mode when READY# (internal or external) terminates the halt bus cycle. When these bits have equal values, the HALT instruction causes a normal halt and the device remains in active mode. **PC1 PC0** 0 0 active mode 1 0 idle mode 0 1 powerdown mode 1 1 active mode

Figure 8-5. Power Control Register (PWRCON)

8.3.1 Idle Mode

Idle mode freezes the core clocks (PH1C low and PH2C) high, and leaves the peripheral clocks (PH1P and PH2P) toggling. To enter idle mode:

1. Program the PWRCON register (Figure 8-5).
2. Execute a HALT instruction.
3. The CPU enters idle mode when READY# terminates the halt bus cycle.

NOTE

CLKOUT continues to run while the CPU is in idle mode.

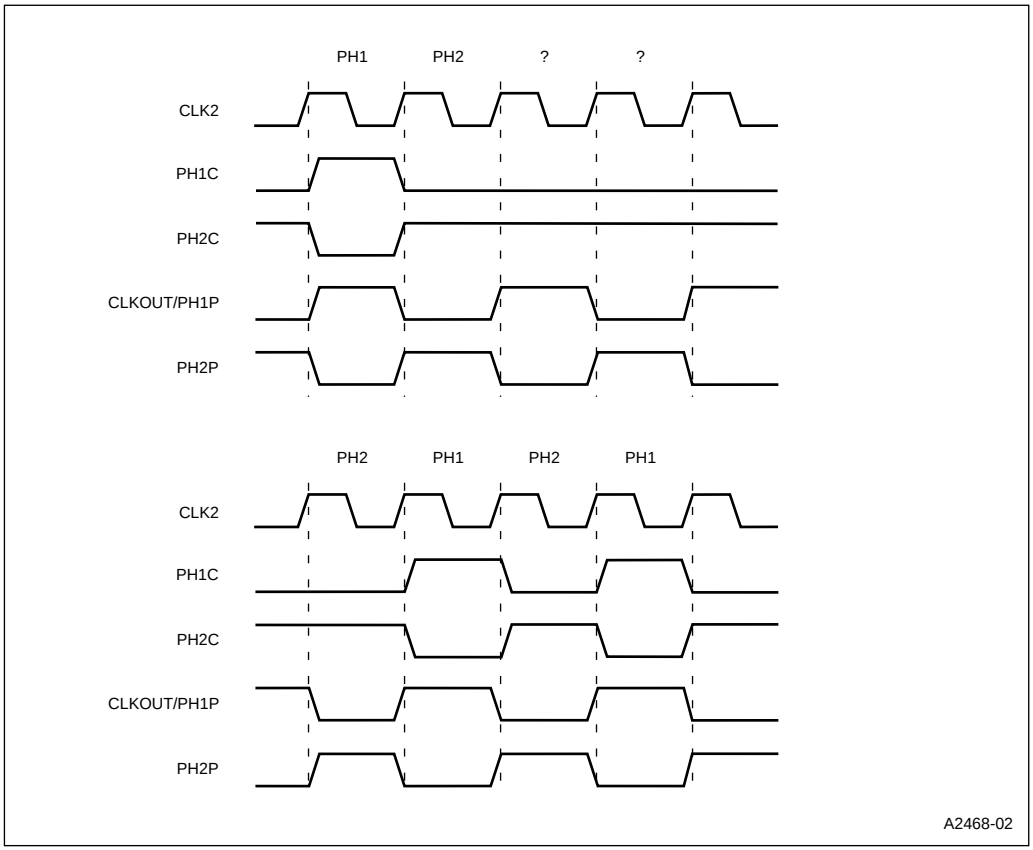


Figure 8-6. Timing Diagram, Entering and Leaving Idle Mode

8.3.2 Powerdown Mode

Powerdown mode freezes both the core clocks and the peripheral clocks (PH1C and PH1P low, PH2C and PH2P high). The BIU **cannot** acknowledge DMA, refresh, and external hold requests in powerdown mode, since all the clocks are frozen.

To enter powerdown mode, follow these steps:

1. Program the PWRCON register (Figure 8-5).
2. Execute a HALT instruction.
3. The CPU enters powerdown mode when READY# (internal or external) terminates the halt bus cycle.

When P3.6/PWRDOWN is configured as a peripheral pin, the pin goes high when the clocks stop, to indicate that the device is in powerdown mode. (Chapter 16 explains how to configure the pin as either a peripheral pin or a general-purpose I/O port pin.)

8.3.3 Ready Generation During HALT

A halt cycle, like all other CPU bus cycles, requires a valid READY# to complete. This ready can be generated by either external logic, or from the internal bus interface unit (BIU). Setting bit 2 of the PWRCON causes the READY# to be generated by the internal BIU, and clearing bit 2 requires it to be generated by external logic. When READY# is generated internally the LBA# signal is driven low.

External logic can use the PWRDOWN output to control other system components and prevent DMA and hold requests.

NOTE

When the processor exits Powerdown Mode, use the CLKOUT pin for external synchronization with the processor clock.

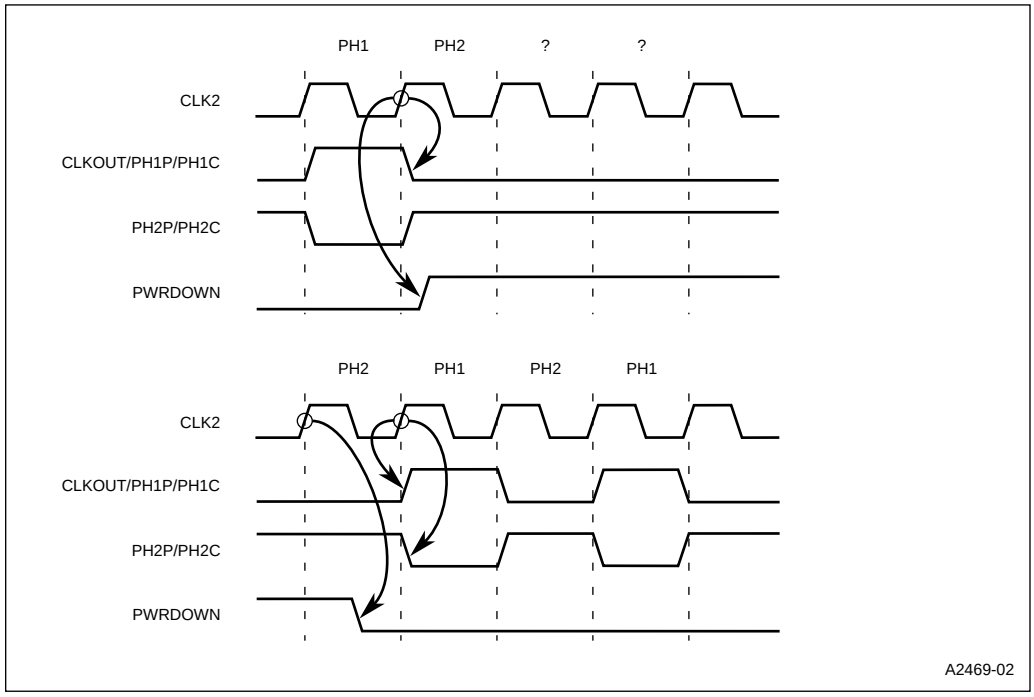


Figure 8-7. Timing Diagram, Entering and Leaving Powerdown Mode

8.4 DESIGN CONSIDERATIONS

This section outlines design considerations for the clock and power management unit.

8.4.1 Reset Considerations

External circuitry must provide an input to the RESET pin. The RESET input must remain high for at least 16 CLK2 cycles to reset the chip properly.

The RESET pin signal is routed directly to the device's bidirectional pins. Even in idle or powerdown, a device reset floats the bidirectional pins and turns on the weak pull-up or pull-down transistors.

The clock generation logic generates a synchronous internal RESET signal for the internal peripherals. If you need a synchronous RESET signal for other system components, you can use a simple circuit such as the one shown in Figure 8-8 to generate it. Otherwise, the CPU does not need a synchronous reset.

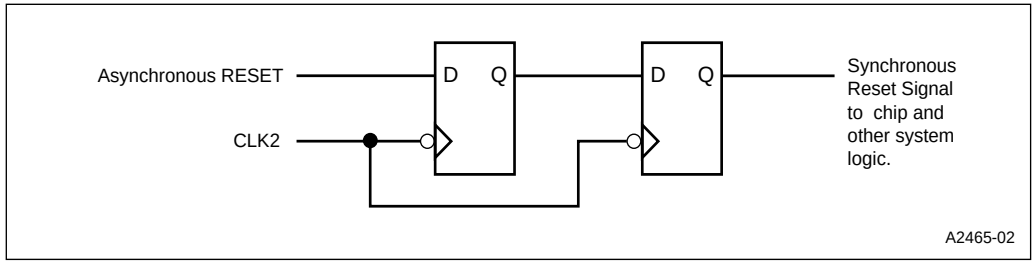


Figure 8-8. Reset Synchronization Circuit

8.4.2 Power-up Considerations

8.4.2.1 Built-in Self Test

The Intel386 EX processor supports the Intel386 SX processor built-in self-test (BIST) mode for testing core functions. To initiate the self test, follow these steps:

1. Hold the RESET pin high for a minimum of 80 CLK2 cycles.
2. Transition the RESET pin from high to low while keeping the BUSY# pin asserted. The BUSY# input should be asserted at least eight CLK2 cycles before the falling edge of RESET and must be kept asserted for at least eight CLK2 cycles after the falling edge of RESET.

Once BIST has been initiated, it takes approximately 2^{20} processor clock cycles to complete. At the completion of the BIST, the processor performs an internal reset and begins normal operation.

8.4.2.2 JTAG Reset

The processor supports an IEEE 1149.1 compliant JTAG boundary scan. The JTAG unit has its own clock and RESET signals, independent from the rest of the processor. The processor requires that the JTAG unit be reset before normal operation can begin. To reset the JTAG unit, invert the processor RESET signal and connect this inverted RESET signal to the TRST# pin.

8.4.3 Powerdown Mode and Idle Mode Considerations

- The “wake-up” signals (INT, NMI, and SMI#) are level-sensitive inputs to the wake-up circuitry. The active state of any of these inputs prevents the device from entering powerdown or idle mode.
- The refresh control unit cannot perform DRAM refreshes during powerdown.
- Powerdown mode freezes PSCLK and SERCLK.
- When the device exits powerdown mode, the PWRDOWN signal is synchronized with CLK2 (at the falling edge of PWRDOWN) so that other devices in the system exit powerdown at the same internal clock phase as the processor.
- The INTR output of the ICU cannot be masked off to the power management unit using the CLI instruction. If it is necessary to mask off INTR to the power management unit, all the interrupt inputs to the 82C59As must be masked. This applies to both powerdown and idle modes.

8.5 PROGRAMMING CONSIDERATIONS

8.5.1 Clock and Power Management Unit Code Example

This section contains these software routines:

Set_Prescale_Value	Sets the clock prescale value.
Enter_Idle_Mode	Programs the Intel386 EX processor for idle mode.
Enter_Powerdown_Mode	Programs the Intel386 EX processor for powerdown mode.
Mode_Setting_to_Active	Returns the Intel386 EX processor to active mode.

See Appendix C for the included header files.

```
#include <conio.h>
#include “80386ex.h”
#include “EV386EX.h”

/*****************************************************************************
   Set_Prescale_Value:

  Description:
     This function sets the clock prescale value.

  Parameters:
     Prescale            Prescale value

  Returns: Error Codes
     E_BAD_VECTOR        -- Specified Prescale is invalid
     E_OK                -- Initialized OK, No error.

  Assumptions:
```

```
      None

   Syntax:

      int error;
      WORD psclk = 0x02;

      error = Set_Prescale_Value(psclk);


   Real/Protected Mode:
      No changes required.
*****************************************************************************/

int Set_Prescale_Value(WORD Prescale)
{

  WORD clkprs = 0x0000;

  clkprs = _GetEXRegWord(CLKPRS);

  /* clear lowest nine bits of clkprs */
  clkprs = clkprs & 0xfe00;

  /* check that prescale value is only 9 bits in length */
  if (Prescale != (Prescale & 0x01ff))
     return(E_BADVECTOR);

  _SetEXRegWord(CLKPRS, (clkprs | Prescale));

  return(E_OK);

}/*Set_Prescale_Value*/



/*****************************************************************************

    Enter_Idle_Mode:

   Description:
      This function programs the 386EX for Idle mode.  This freezes the
      core clocks while leaving the peripheral clocks toggling.

   Parameters:
      None

   Returns:
      None

   Assumptions:
      None

    Syntax:

     Enter_Idle_Mode();

    Real/Protected Mode:
```

```
      No changes required.

*************************************************************************/

void Enter_Idle_Mode(void)
{
  BYTE pwrcon = 0x00;

  pwrcon = _GetEXRegByte(PWRCON);

  /* clear lowest two bits of pwrcon */
  pwrcon = pwrcon & 0xfc;

  /* Set mode to idle */
  _SetEXRegByte(PWRCON, (pwrcon | IDLE));

  /* call HALT instruction to execute IDLE mode */
  _asm {

     HLT
  }

}/* Enter_Idle_Mode */



/*************************************************************************

   Enter_Powerdown_Mode:

   Description:
      This function programs the 386EX for Powerdown mode.  This freezes
      both the core and peripheral clocks.

   Parameters:
      None

   Returns:
      None

   Assumptions:
      None

   Syntax:

      Enter_Powerdown_Mode();

    Real/Protected Mode:
     No changes required.

*************************************************************************/


void Enter_Powerdown_Mode(void)
{
  BYTE pwrcon = 0x00;

  pwrcon = _GetEXRegByte(PWRCON);
```

```
  /* clear lowest two bits of pwrcon */
  pwrcon = pwrcon & 0xfc;

  /* Set mode to powerdown */
  _SetEXRegByte(PWRCON, pwrcon | PWDWN);

  /* call HALT instruction to execute POWERDOWN mode */
  _asm {

     HLT
  }

}/* Enter_Powerdown_Mode */


/*****************************************************************************

   Mode_Setting_To_Active:

   Description:
      This function returns the 386EX to Active mode.  Thus, the next
      HALT instruction will not invoke the Idle or Powerdown Mode.

   Parameters:
      None

   Returns:
      None

   Assumptions:
      None

   Syntax:

      Mode_Setting_To_Active();

   Real/Protected Mode:
      No changes required.

*****************************************************************************/

void Mode_Setting_To_Active(void)
{
  BYTE pwrcon = 0x00;

  pwrcon = _GetEXRegByte(PWRCON);

  /* clear lowest two bits of pwrcon */
  pwrcon = pwrcon & 0xfc;

  /* Set mode to active */
  _SetEXRegByte(PWRCON, pwrcon | ACTIVE);

}/*Mode_Setting_To_Active*/
```

9

INTERRUPT CONTROL UNIT

CHAPTER 9
INTERRUPT CONTROL UNIT

The Interrupt Control Unit (ICU) consists of two cascaded interrupt controllers, a master and a slave, that allow internal peripherals and external devices (through interrupt pins) to interrupt the core through its interrupt input.

The interrupt control unit is functionally identical to two industry-standard 82C59As connected in cascade. The system supports a maximum of 15 simultaneous interrupt sources, which can be individually or globally disabled. The ICU passes the interrupts on to the core based on a programmable priority structure.

Though the ICU can only handle a maximum of 15 simultaneous sources, a total of 18 interrupt sources can be connected to the ICU. Eight of these interrupt sources come from internal peripherals and the other ten come from external pins. To increase the number of possible interrupts, you can cascade additional 82C59As to six of the external interrupt pins (the pins that connect to the master 82C59A only).

This chapter describes the interrupt control unit and is organized as follows:

9.1 OVERVIEW

The ICU consists of two 82C59As configured as master and slave. Each 82C59A has eight interrupt request (IR) signals. The master has seven interrupt sources and a slave 82C59A connected to its IR signals. The slave has nine interrupt sources connected to its IR signals (two sources are multiplexed into IR1). The interrupts can be globally or individually enabled or disabled.

The master can receive multiple interrupt requests at once. It can also receive a request while the core is already processing another interrupt. The master uses a programmable priority structure that determines:

- The order in which to process multiple interrupt requests
- Which requests can interrupt the processing of other requests

When the master receives an interrupt request, it checks to see that the interrupt is enabled and determines its priority. If the interrupt is enabled and has sufficient priority, the master sends the request to the core. This causes the core to initiate an internal interrupt acknowledge cycle.

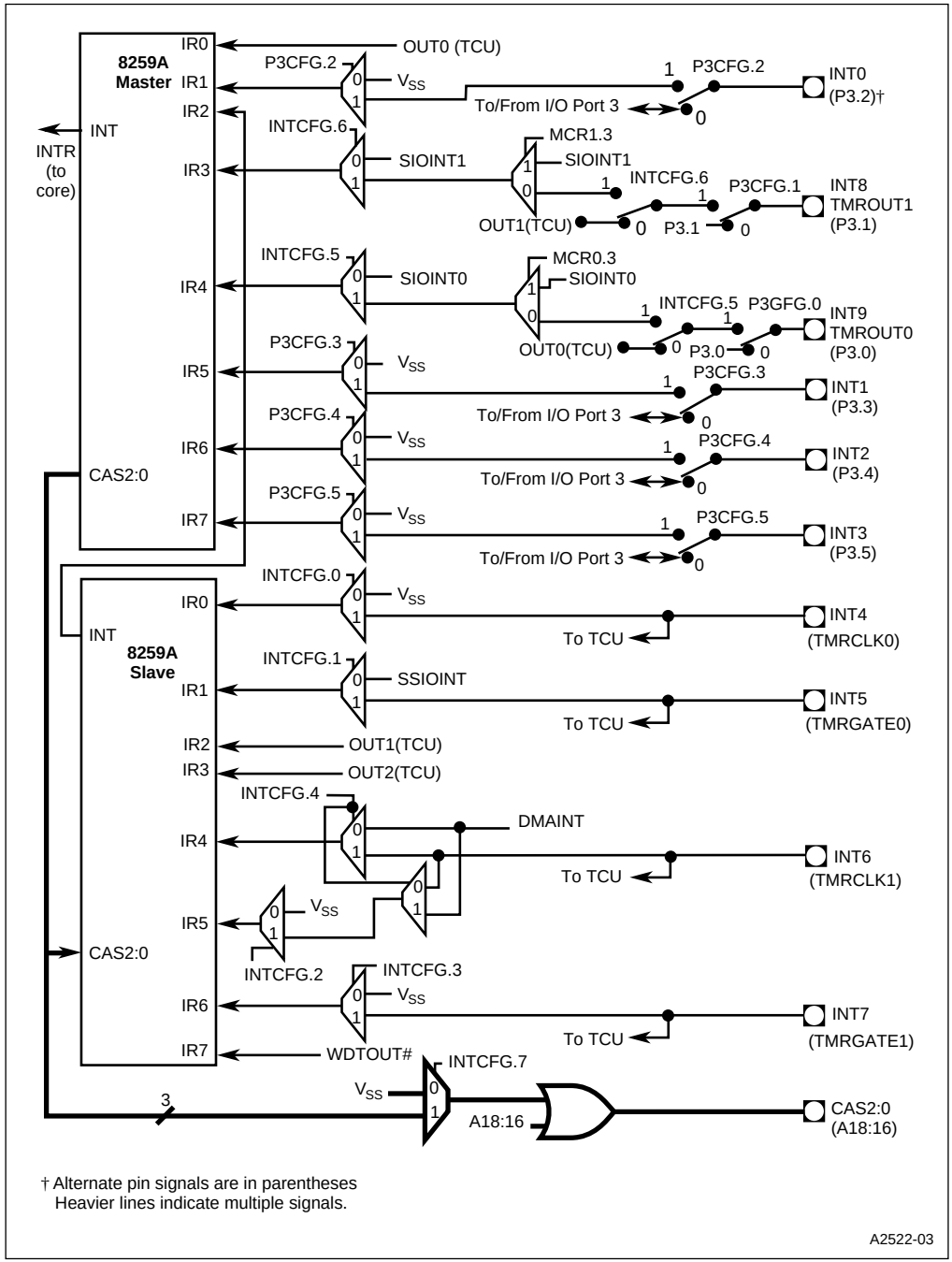
The slave 82C59A is cascaded from (or connected to) the master's IR2 signal. Like the master, the slave uses a programmable priority structure. When the slave receives an interrupt request, it sends the request to the master (assuming the request is enabled and has sufficient priority). The master sees the slave request as a request on its IR2 line. The master then sends the request to the core (assuming the request is enabled and has sufficient priority) and the core initiates an internal interrupt acknowledge cycle.

The internal interrupt acknowledge cycle consists of two pulses that are sent to the 82C59A INTA# inputs. This cycle causes the 82C59A that received the original interrupt request to put the request's vector number on the bus. The master's cascade signals (CAS2:0) determine which 82C59A is being acknowledged (i.e., which 82C59A needs to put the vector number on the bus). The core uses its processing mode (real or protected) and the vector number to find the address of the interrupt service routine.

The master 82C59A has six device pins (INT9:8, INT3:0) connected to it. You can cascade additional external 82C59A slaves to these pins to increase the number of possible interrupt sources. The external interrupt signals, INT9:8, are multiplexed with the internal asynchronous serial I/O interrupt signals, SIOINT0 and SIOINT1. On the slave 82C59A, the external interrupt signal, INT6, and the DMA Unit's DMAINT signal, can be swapped before connecting to the slave's IR4 and IR5 inputs (see Figure 9-1). The core initiates interrupt acknowledge cycles for the internal 82C59As. External logic must decode the bus signals (M/IO#, D/C#, W/R# and REFRESH#, see Table 6-2 on page 6-5) to generate external interrupt acknowledge signals. Since the cascade bus determines which 82C59A is being acknowledged, each external slave must monitor the master's cascade signals to determine whether it is the acknowledged slave. For external slaves, the master's cascade signals (CAS2:0) can be driven (using bit 7 of the INTCFG register) onto the A18:16 address pins.

NOTE

Since external 82C59As require the CAS2:0 signals to stay valid through the idle states that occur between the two interrupt acknowledge cycles, and since the processor drives these lines high during these idle states, the CAS2:0 lines must be latched externally to ensure validity during the idle states.



† Alternate pin signals are in parentheses
Heavier lines indicate multiple signals.

A2522-03

Figure 9-1. Interrupt Control Unit Configuration

9.2 ICU OPERATION

The following sections describe the ICU operation. The ICU's interrupt sources, interrupt priority structure, interrupt vectors, interrupt processing, and polling mode are discussed.

9.2.1 Interrupt Sources

The ICU support a total of 18 interrupt sources (see Table 9-1) but only a maximum of 15 simultaneous sources. Eight of these sources are internal peripherals and ten are external device pins (INT9:0). However, IR3 and IR4 of the master can be connected to either SIOINT1 and SIOINT0 (internal Asynchronous Serial I/O interrupts), or to external device pins INT8 and INT9, respectively. Similarly, IR1 of the slave can be connected to either SSIOINT (internal Synchronous Serial I/O interrupt), or to external device pin INT5. On the slave, the external interrupt signal, INT6, and the DMA Unit's DMAINT signal can be swapped before connecting to the slave's IR4 and IR5 inputs

The device pins (INT3:0) are multiplexed with port pins. When the port pin function (rather than the interrupt function) is enabled at the pin, V_{SS} is internally connected to the ICU's respective interrupt request input. The device pins, INT7, INT6, and INT4, must be enabled (using register bits) in order to be used. The port 3 configuration register (P3CFG) controls INT3:0 interrupt source connections, and the interrupt configuration register (INTCFG) controls the INT9:4 interrupt source connections. The modem control registers (MCR1 and MCR0) are also used to control the INT9:8 interrupt source connections.

Table 9-1. 82C59A Master and Slave Interrupt Sources

Master IR Line	Source	Connected by
IR0	TMROUT0 (timer control unit)	Hardwired
IR1	V_{SS}	P3CFG.2=0
	INT0 (device pin)	P3CFG.2=1
IR2	Slave 82C59A Cascade	Hardwired
IR3	SIOINT1 (SIO unit)	INTCFG.6=0 P3CFG.1=0
	INT8 (device pin)	INTCFG.6=1 P3CFG.1=1 MCR0.3=1
IR4	SIOINT0 (SIO unit)	INTCFG.5=0 P3CFG.0=0
	INT9 (device pin)	INTCFG.5=1 P3CFG.0=1 MCR1.3=1
IR5	V_{SS}	P3CFG.3=0
	INT1 (device pin)	P3CFG.3=1
IR6	V_{SS}	P3CFG.4=0
	INT2 (device pin)	P3CFG.4=1
IR7	V_{SS}	P3CFG.5=0
	INT3 (device pin)	P3CFG.5=1

Slave IR Line	Source	Connected by
IR0	V_{SS}	INTCFG.0=0
	INT4 (device pin)	INTCFG.0=1
IR1	SSIOINT (SSIO unit)	INTCFG.1=0
	INT5 (Device pin)	INTCFG.1=1
IR2	TMROUT1 (timer control unit)	Hardwired
IR3	TMROUT2 (timer control unit)	Hardwired
IR4	DMAINT (DMA unit)	INTCFG.4=0
	INT6 (device pin)	INTCFG.4=1
IR5	INT6 (device pin)	INTCFG.4=0 INTCFG.2=1
	DMAINT (DMA unit)	INTCFG.4=1 INTCFG.2=1
IR6	V_{SS}	INTCFG.3=0
	INT7 (device pin)	INTCFG.3=1
IR7	WDTOUT# (watchdog timer)	Hardwired

Interrupt processing begins with the assertion of an IR signal. During the ICU initialization process (described in "Register Definitions" on page 9-15), you can program the ICU to be either edge-triggered or level-triggered. See "Interrupt Detection" on page 9-29 for a description of the difference between level and edge triggered signals.

9.2.2 Interrupt Priority

Each 82C59A contains eight interrupt request signals. An 82C59A can receive several concurrent interrupt requests or can receive a request while the core is servicing another interrupt. When either condition occurs, the 82C59A uses a programmable priority structure to determine the order in which to process the interrupts. There are two parts to the priority structure:

- Assigning an interrupt level to each IR signal
- Determining their relative priorities

9.2.2.1 Assigning an Interrupt Level

By default, the interrupt structure for each 82C59A is configured so that IR0 has the highest level and IR7 has the lowest level. Two methods (shown in Figure 9-2) are available for changing this interrupt structure:

Specific Rotation This method assigns a specific IR signal as the lowest level. The other IR signals are automatically rearranged in a circular manner. For example, if you specify IR5 as the lowest level, IR6 becomes the highest level, IR7 becomes the second-highest, and so on, with IR4 the second-lowest.

Automatic Rotation This method assigns an IR signal to the lowest level after the core services its interrupt. As with specific rotation, the other signals are automatically rearranged in a circular manner. For example, the IR4 signal is assigned the lowest level after the core services its interrupt, IR5 becomes the highest level, IR6 becomes the second-highest, and so on, with IR3 the second-lowest.

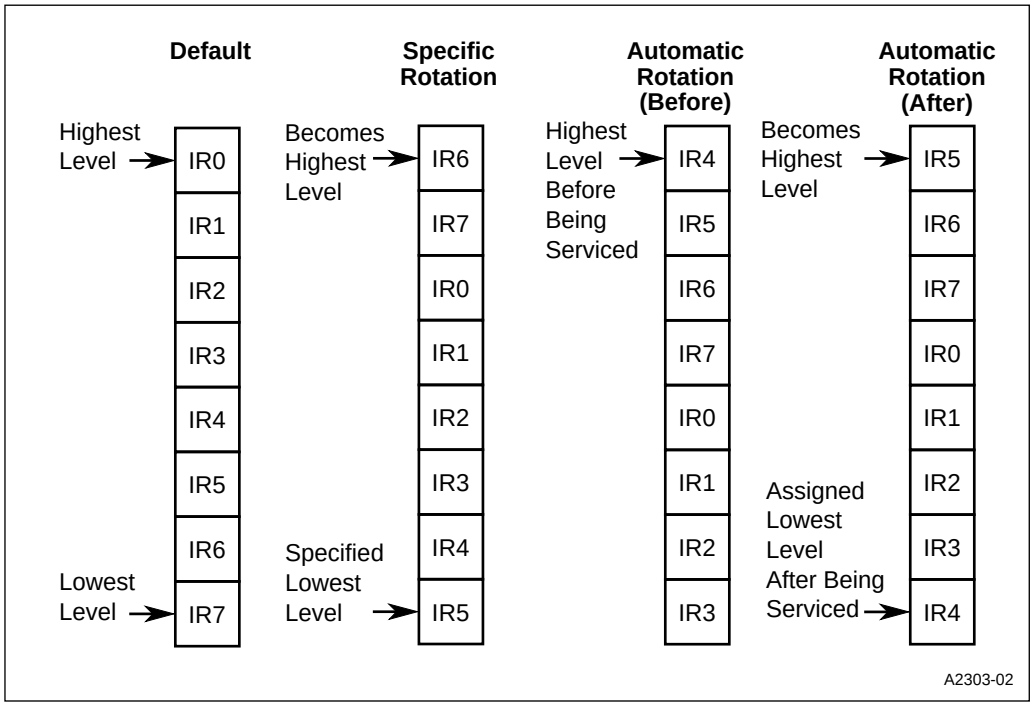


Figure 9-2. Methods for Changing the Default Interrupt Structure

9.2.2.2 Determining Priority

There are three modes that determine relative priorities, i.e., whether a level higher, lower, or equal to another level has higher or lower interrupt priority.

Fully nested — In the fully nested mode, higher level IR signals have higher interrupt priority. In this mode, when an 82C59A receives multiple interrupt requests, it passes the highest level request to the core (or to the master if the 82C59A is a slave). The core stops processing the lower level request, processes the higher level request, then returns to finish the lower level request.

Special fully nested — The special fully nested mode allows higher or equal level IR signals to have higher interrupt priority. In this mode, if the core is processing an interrupt, a higher or equal level interrupt request is passed through to the core. Also, since all interrupts from the slave are directed into a single IR line (IR2) on the master (the master does not know the priorities of the slave interrupts it receives), this mode enables a higher-level interrupt on the slave to interrupt the

processing of a lower-level slave interrupt. The special fully nested mode is generally used by the master in a cascaded system.

Special mask In some applications, you may want to allow lower-level requests interrupt the processing of higher-level interrupts. The special mask mode supports these applications. Unlike the special-fully nested and fully nested modes, which are selected during ICU initialization, the special mask mode can be enabled and disabled during program operation. When special mask mode is enabled, only interrupts from the source currently in service are inhibited. All other interrupt requests (of both higher or lower levels) are passed on.

When the internal slave receives an interrupt request, it passes that request to the master. The master receives all internal slave interrupt requests on its IR2 signal. This means that in fully nested mode, higher-level slave requests cannot interrupt lower-level slave interrupts. For example, suppose the slave gets an interrupt request on its IR7 signal. The slave sends the interrupt request to the master's IR2 signal (assuming the slave's IR7 interrupt is enabled and has sufficient priority). The master sends the interrupt request to the core (assume the master's IR2 interrupt is enabled and has sufficient priority). The core initiates an interrupt acknowledge cycle and begins processing the interrupt. Next, the slave gets an interrupt request on its IR0 signal (assume IR0 is assigned a higher level than IR7). It then sends another IR2 to the master.

When the master is in fully nested mode, it does not relay the request to the core because the core is in the process of servicing the previous IR2 interrupt and only a higher-level request can interrupt its process (IR2 is not higher than IR2).

When the master is in special fully nested mode, the request is passed through to the core (IR2 is equal to IR2).

9.2.3 Interrupt Vectors

Each interrupt request has a corresponding interrupt vector number. The interrupt vector number is a pointer to a location in memory where the address of the interrupt's service routine is stored. The relationship between the interrupt vector number and the location in memory of the interrupt's service routine address depends on the system's programmed operating mode (real, protected, or virtual86). Chapter 9 of the *Intel386™ SX Microprocessor Programmer's Reference Manual* explains this relationship.

During an interrupt acknowledge cycle, the ICU puts the interrupt's vector number on the bus. From the interrupt vector number and the system's operating mode, the core determines where to find the address of the interrupt's service routine.

You must initialize each 82C59A with an interrupt vector base number. The 82C59As determine the vector number for each interrupt request from this base number. The base vector number corresponds to the IR0 signal's vector number and must be on an 8-byte boundary.

Other vector numbers are determined by adding the line number of the IR signal to the base. For example, if the base vector number is 32, the IR5 vector number is 37. Valid vector numbers for maskable interrupts range from 32 to 255. Because the base vector number must reside on an 8-byte boundary, the valid base vector numbers are $32 + n \times 8$ where $0 \leq n \leq 27$.

9.2.4 Interrupt Process

Each IR signal has a mask, a pending, and an in-service bit associated with it.

- The mask bit disables the IR signal. The respective mask bits provide a way to individually disable the IR signals. You can globally disable all interrupts to the core using the CLI instruction. The mask bits reside in the OCW1.
- The pending bit indicates that the IR signal is requesting interrupt service. The pending bit resides in the IRR (Interrupt Request Register, which is accessed through OCW3).
- The in-service bit indicates that the processor is in the process of servicing the interrupt. The in-service bit resides in the ISR (Interrupt Service Register, which is accessed through OCW3).

When the master 82C59A receives an interrupt request, it sets the corresponding pending bit and sends the request to the core (assuming the request is enabled and has sufficient priority). The core then initiates an acknowledge cycle: the master clears its pending bit, sets its in-service bit, and puts the interrupt vector number on the bus.

When the slave 82C59A receives an interrupt request, it sets the corresponding pending bit and sends the request to the master (assuming the request is enabled and has sufficient priority). When the master receives the slave request, it sets its IR2 pending bit and sends the IR2 request to the core (assuming the request is enabled and has sufficient priority). The core initiates an interrupt acknowledge cycle: the master clears its IR2 pending bit and sets its IR2 in-service bit. The master's cascade bus activates the slave, which responds to the interrupt acknowledge cycle, clears its pending bit, sets its in-service bit, and puts the interrupt vector number on the bus.

An 82C59A uses its in-service bits and programmed priority structure to determine whether an interrupt has sufficient priority. The in-service bits indicate which interrupt requests are being serviced. The priority structure determines whether a new interrupt request's level has sufficient priority to interrupt the current process.

You can use one of three methods to clear an in-service bit: enable the automatic end-of-interrupt (AEOI) mode, issue a specific end-of-interrupt (EOI) command, or issue a nonspecific EOI command. The AEOI mode is available only on the master 82C59A.

AEOI mode	This mode is enabled during system initialization. In this mode, the 82C59A clears the in-service bit at the beginning of an interrupt's processing. This means that interrupts of any level can interrupt the processing of other interrupts.
Specific EOI command	This command instructs the 82C59A to clear a specific IR in-service bit.
Nonspecific EOI command	This command instructs the 82C59A to clear the in-service bit that corresponds to the highest level IR signal active at that time.

NOTE

Unlike the AEOI mode (this is a mode, and not a command like specific EOI or nonspecific EOI), which is enabled during initialization, the other methods are commands issued during interrupt processing, usually at the end of an interrupt's service routine.

Figure 9-3 illustrates the process that takes place when the master receives a non-slave interrupt request (which is a request on any IR signal to the master, that does not have a slave cascaded from it). Figure 9-4 illustrates the process that occurs when a slave receives an interrupt request. Figure 9-5 continues by showing what happens when the master receives a slave interrupt request (for example, an IR2 request).

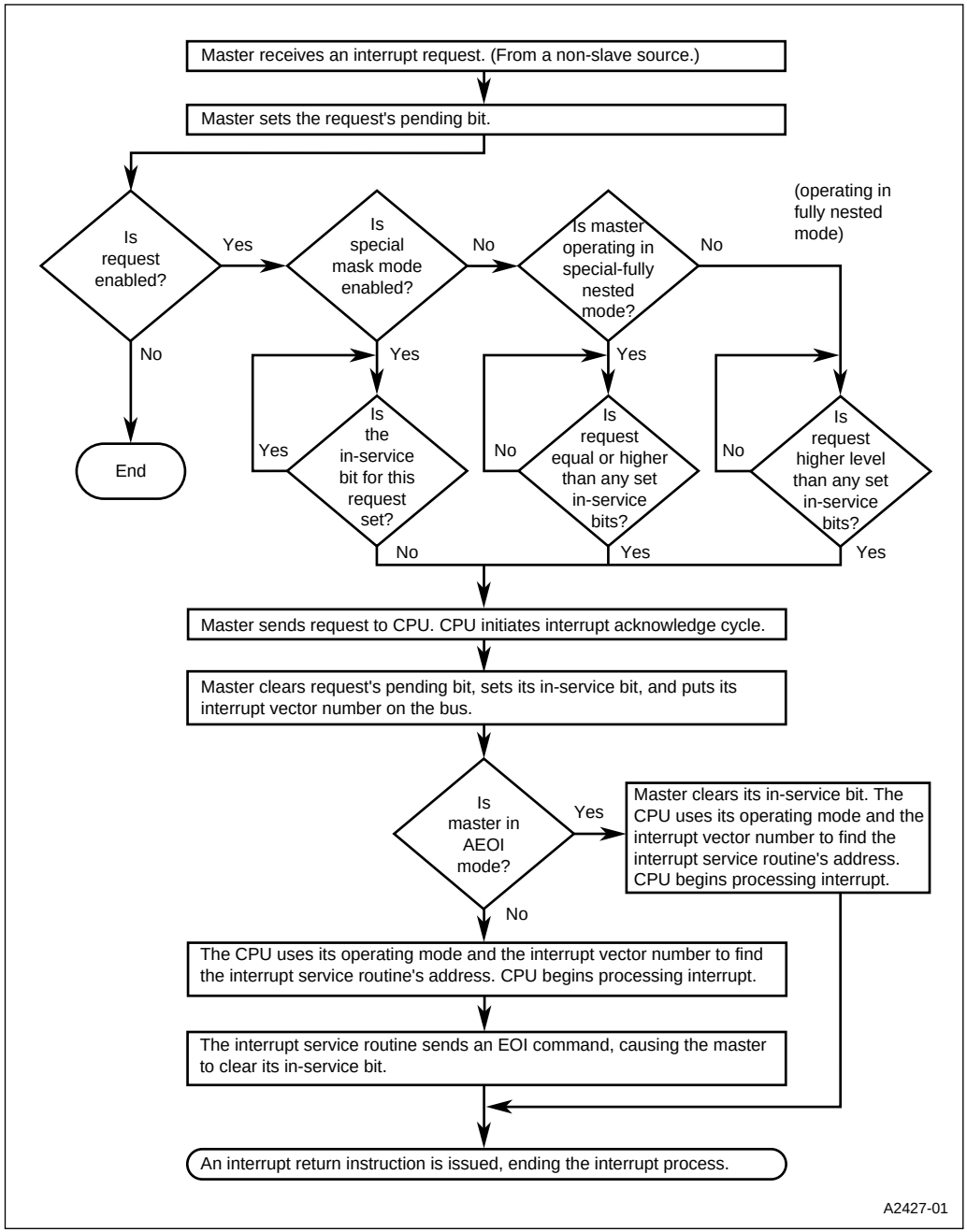
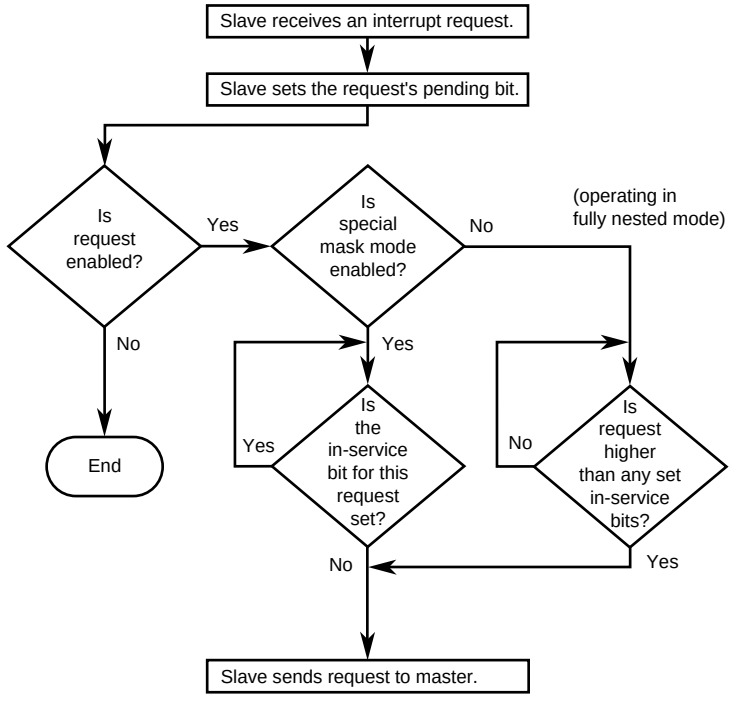


Figure 9-3. Interrupt Process – Master Request from Non-slave Source



Note:
See the "Interrupt Process - Master Request from Slave Source" figure for the continuation of this flow chart.

A2428-01

Figure 9-4. Interrupt Process – Slave Request

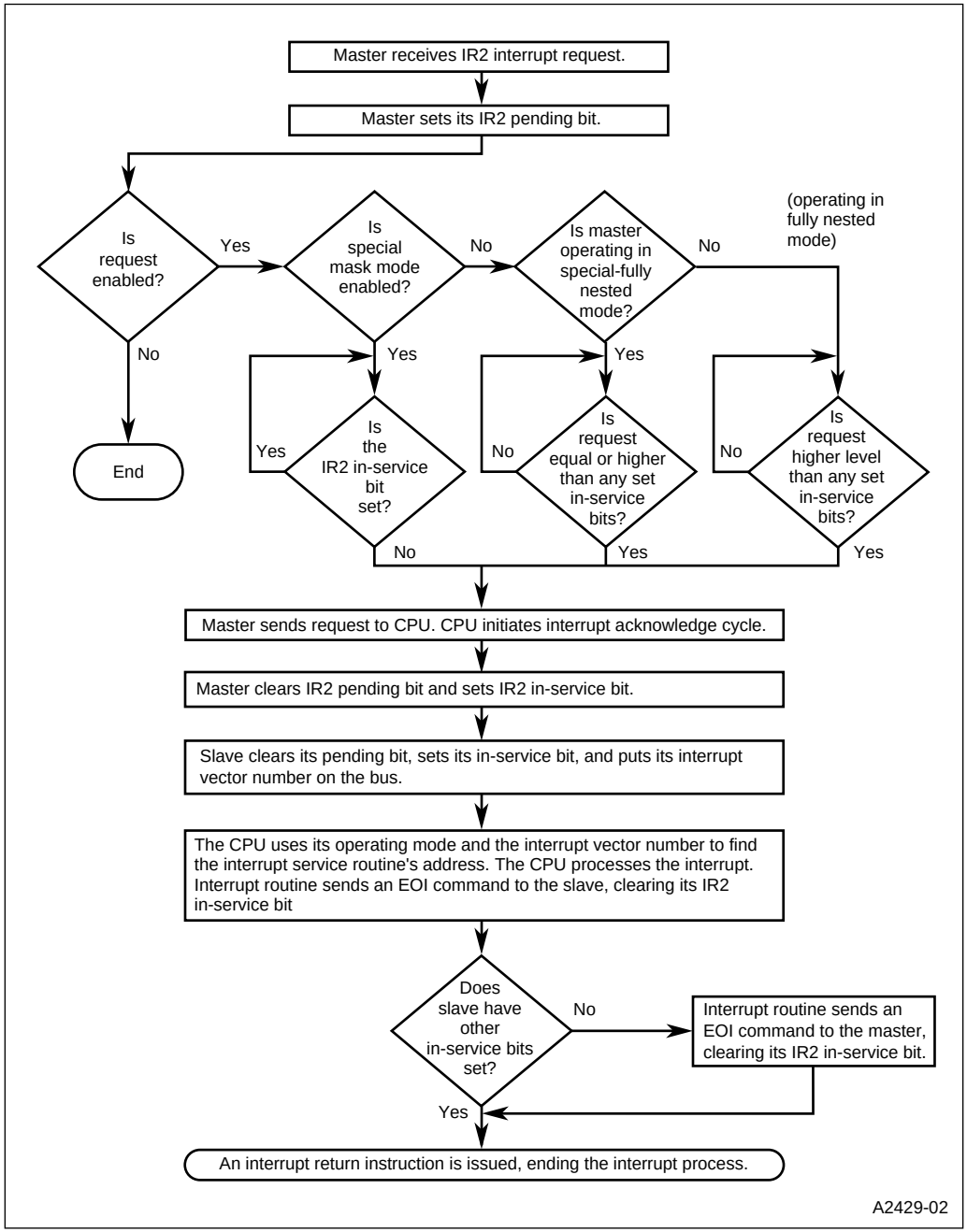


Figure 9-5. Interrupt Process – Master Request from Slave Source

The interrupt's priority structure determines which EOI command should be used. Use the specific EOI command for the special mask mode. In this mode, a lower-level interrupt can interrupt the processing of a higher-level interrupt. The specific EOI command is necessary because it allows you to specifically clear the lower level in-service bit.

The fully nested mode allows only interrupts of higher levels to interrupt the processing of a lower-level interrupt. In this mode, the nonspecific EOI command automatically clears the in-service bit for the current process (because it has the highest level).

Special-fully nested mode allows equal or higher level requests to interrupt the processing of other interrupts. For this mode, the nonspecific EOI command automatically clears the appropriate in-service bit. However, when processing master IR2 interrupts, you must make sure all the slave in-service bits are cleared before issuing the nonspecific EOI command to the master.

9.2.5 Poll Mode

The 82C59A modules can operate in a polling mode. Conventional polling requires the core to check each peripheral device in the system periodically to see whether it requires servicing. With the 82C59A's polling mode, the core, by initiating the polling process, can determine whether any of the devices attached to the 82C59A require servicing. This improves conventional polling efficiency by allowing the core to poll only the 82C59A, not each of the devices connected to it. The polling mode is enabled by setting the polling bit in the Operation Command Word 3 register (OCW3).

NOTE

After the polling procedure has been executed once, polling is disabled, i.e., it is a one-shot operation. To repeat the polling procedure, the polling bit must be set again.

The polling process takes the place of the standard interrupt process. In the standard interrupt process, the master sends interrupt requests to the core. In the polling mode, an interrupt request can be detected by reading the 82C59A's poll status byte. The poll status byte indicates whether the 82C59A requires servicing. If the 82C59A requires servicing, the poll status byte indicates the highest-priority pending interrupt request.

Polling is always a two-step process:

- A poll command is issued.
- The poll status byte is read.

When an 82C59A receives an interrupt request before it receives a poll command, it sets the request's in-service bit and configures the poll status byte to reflect the interrupt request. The poll status byte is used to determine which device connected to the 82C59A requires servicing. At the end of a request's servicing, you must issue a command to clear the request's in-service bit.

The polling mode allows expansion of the system's external interrupt capability. Without polling, the system can have a maximum of 52 external interrupt sources. This is accomplished by cascading six 82C59As to the master's six external interrupt pins and using the four external interrupt pins connected to the slave. The polling mode increases the system's interrupt capability by

configuring more than six external 82C59As. Since the polling mode doesn't require that the additional 82C59As be cascaded from the master, the number of interrupt request sources for a polled system is limited only by the number of 82C59As that the system can address.

Polling and standard interrupt processing can be used within the same program. Systems that use polling as the only method of device servicing must still fully initialize the 82C59A modules. Also, the interrupt requests to the core must be disabled using the mask bits or the CLI instruction.

9.3 REGISTER DEFINITIONS

The registers associated with the ICU consist of pin and signal configuration registers, initialization command words (ICWs), operation command words (OCWs), and status registers.

- The configuration registers enable the external interrupt sources.
- The ICWs initialize the 82C59As during system initialization.
- The OCWs modify an 82C59A's operation during program execution.
- The status registers reflect pending and in-service interrupts.

NOTE

ICW2, ICW3 and ICW4 of an 82C59A are all at the same address. Therefore a programming sequence must be followed to program these registers. The first access goes to ICW2, the second to ICW3 and the third to ICW4. When programming any of these registers, the above sequence must be followed and completed every time.

When initializing the ICU, write first to ICW1, then to ICW2, ICW3 and ICW4 in order.

Table 9-2 describes these registers and the following sections contain bit descriptions for each register.

Table 9-2. ICU Registers (Sheet 1 of 2)

Register	Expanded Address	PC/AT* Address	Function
P3CFG (read/write)	0F824H	—	Port 3 Configuration: The INT3:0 signals are multiplexed with P3.5:2. This register determines which signals are connected to the package pins. When a P3.*n* signal rather than an INT*n* signal is connected to a package pin, V_{SS} is connected to the master's IR*n* signal.
INTCFG (read/write)	0F832H	—	Interrupt Configuration: Determines the master's and the slave's IR signal connections: SIOINT1 or INT8; SIOINT0 or INT9; V_{SS} or INT7; V_{SS} or INT6; SSIOINT or INT5; V_{SS} or INT4. Swaps DMAINT and INT6. Also enables the master's cascade bus (CAS2:0). When enabled, the cascade signals appear on the A18:16 address lines during an interrupt acknowledge cycle.
ICW1 (master) ICW1 (slave) (write only)	0F020H 0F0A0H	0020H 00A0H	Initialization Command Word 1: Determines whether interrupt request signals are level sensitive or edge triggered.
ICW2 (master) ICW2 (slave) (write only)	0F021H 0F0A1H	0021H 00A1H	Initialization Command Word 2: Contains the base interrupt vector number for the 82C59A. The base interrupt vector is the IR0 vector number, the base plus one is the IR1 vector number, and so on.
ICW3 (master) (write only)	0F021H	0021H	Initialization Command Word 3: Identifies the master's IR signals that are connected to slave 82C59A devices. The internal slave is connected to the master's IR2 signal. You can connect external slaves to the master's IR1, IR3, IR4, IR5, IR6, and IR7 signals.
ICW3 (slave) (write only)	0F0A1H	00A1H	Initialization Command Word 3: Indicates that the internal slave is cascaded from the master's IR2 signal.
ICW4 (master) ICW4 (slave) (write only)	0F021H 0F0A1H	0021H 00A1H	Initialization Command Word 4: Selects either special-fully nested or fully nested mode and enables the automatic end-of-interrupt mode.
OCW1 (master) OCW1 (slave) (read/write)	0F021H 0F0A1H	0021H 00A1H	Operation Command Word 1: Masks (disables) individual interrupt request signals.
OCW2 (master) OCW2 (slave) (write only)	0F020H 0F0A0H	0020H 00A0H	Operation Command Word 2: Changes interrupt levels and sends end-of-interrupt commands.
OCW3 (master) OCW3 (slave) (write only)	0F020H 0F0A0H	0020H 00A0H	Operation Command Word 3: Enables special mask mode, issues the poll command, and allows access to the interrupt request and in-service registers.

NOTE: All master 82C59A registers are accessed through two expanded or PC/AT addresses; all the slave registers are accessed through two other expanded or PC/AT addresses. The order in which you write or read these addresses along with certain register bit settings determines which register is accessed.

Table 9-2. ICU Registers (Sheet 2 of 2)

Register	Expanded Address	PC/AT* Address	Function
IRR (master) IRR (slave) (read only)	0F020H 0F0A0H	0020H 00A0H	Interrupt Request: Indicates pending interrupt requests.
ISR (master) ISR (slave) (read only)	0F020H 0F0A0H	0020H 00A0H	In-service: Indicates the interrupt requests that are currently being serviced.
POLL (master) POLL (slave) (read only)	0F020H 0F021H 0F0A0H 0F0A1H	0020H 0021H 00A0H 00A1H	Poll Status Byte: Indicates whether any of the devices connected to the 82C59A require servicing. If the 82C59A requires servicing, this byte indicates the highest-priority pending interrupt. **NOTE**: Once the polling bit is set in OCW3, the Poll Status Byte of a particular 82C59A can be read by doing an access to any of the four addresses of that 82C59A.

NOTE: All master 82C59A registers are accessed through two expanded or PC/AT addresses; all the slave registers are accessed through two other expanded or PC/AT addresses. The order in which you write or read these addresses along with certain register bit settings determines which register is accessed.

To initialize the 82C59As:

1. Globally disable all maskable interrupts to the core using the CLI instruction.
2. Write to the initialization command words.

NOTE

You must initialize both the master and the slave (either can be initialized first).

The 8259A module has a state machine that controls access to the individual registers. Improper initialization occurs when the following sequences are not followed:

- To initialize the master, write to its initialization command words in order (ICW1, ICW2, ICW3, then ICW4).
- To initialize the slave, write to its initialization command words in order (ICW1, ICW2, ICW3, then ICW4).

9.3.1 Port 3 Configuration Register (P3CFG)

Use the P3CFG register to connect the interrupt request signals (INT3:0) to the package pins. These signals are multiplexed with port 3 signals, P3.5–2. Connecting a port 3 signal to the package pin also connects V_{SS} to the corresponding master's IR signal, disabling the signal.

Port 3 Configuration
P3CFG
(read/write)

Expanded Addr: **F824H**
ISA Addr: **—**
Reset State: **00H**

7							0
PM7	PM6	PM5	PM4	PM3	PM2	PM1	PM0

Bit Number	Bit Mnemonic	Function
7	PM7	Pin Mode: 0 = Selects P3.7 at the package pin. 1 = Selects COMCLK at the package pin.
6	PM6	Pin Mode: 0 = Selects P3.6 at the package pin. 1 = Selects PWRDOWN at the package pin.
5	PM5	Pin Mode: 0 = Selects P3.5 at the package pin. 1 = Connects master IR7 to the package pin (INT3).
4	PM4	Pin Mode: 0 = Selects P3.4 at the package pin. 1 = Connects master IR6 to the package pin (INT2).
3	PM3	Pin Mode: 0 = Selects P3.3 at the package pin. 1 = Connects master IR5 to the package pin (INT1).
2	PM2	Pin Mode: 0 = Selects P3.2 at the package pin. 1 = Connects master IR1 to the package pin (INT0).
1	PM1	Pin Mode: See Table 5-1 on page 5-8 for all the PM1 configuration options.
0	PM0	Pin Mode: See Table 5-1 on page 5-8 for all the PM0 configuration options.

Figure 9-6. Port 3 Configuration Register (P3CFG)

9.3.2 Interrupt Configuration Register (INTCFG)

Use the INTCFG register to connect the INT9:4 interrupt request pins to the master's and the slave's IR signals and to enable the master's external cascade signals. When enabled, the cascade signals appear on address lines A18:16 during interrupt acknowledge cycles. Every external slave monitors these lines to determine whether it is the slave being addressed.

Interrupt Configuration
INTCFG
(read/write)

Expanded Addr: **F832H**
ISA Addr: **—**
Reset State: **00H**

7							0
CE	IR3	IR4	SWAP	IR6	IR5/IR4	IR1	IR0

Bit Number	Bit Mnemonic	Function
7	CE	Cascade Enable: 0 = Disables the cascade signals CAS2:0 from appearing on the A18:16 address lines during interrupt acknowledge cycles. 1 = Enables the cascade signals CAS2:0, providing access to external slave 82C59A devices. The cascade signals are used to address specific slaves. If enabled, slave IDs appear on the A18:16 address lines during interrupt acknowledge cycles, but are high during idle cycles.
6	IR3	Internal Master IR3 Connection: See Table 5-1 on page 5-8 for all the IR3 configuration options.
5	IR4	Internal Master IR4 Connection: See Table 5-2 on page 5-8 for all the IR4 configuration options.
4	SWAP	INT6/DMAINT Connection: 0 = Connects DMAINT to the slave IR4. Connects INT6 to the slave IR5. 1 = Connects the INT6 pin to the slave IR4. Connects DMAINT to the slave IR5.
3	IR6	Internal Slave IR6 Connection: 0 = Connects V_{SS} to the slave IR6 signal. 1 = Connects the INT7 pin to the slave IR6 signal.
2	IR5/IR4	Internal Slave IR4 or IR5 Connection: These depend on whether INTCFG.4 is set or clear. 0 = Connects V_{SS} to the slave IR5 signal. 1 = Connects either the INT6 pin or DMAINT to the slave IR5 signal.
1	IR1	Internal Slave IR1 Connection: 0 = Connects the SSIO interrupt signal (SSIOINT) to the slave IR1 signal. 1 = Connects the INT5 pin to the slave IR1 signal.
0	IR0	Internal Slave IR0 Connection: 0 = Connects V_{SS} to the slave IR0 signal. 1 = Connects the INT4 pin to the slave IR0 signal.

Figure 9-7. Interrupt Configuration Register (INTCFG)

9.3.3 Initialization Command Word 1 (ICW1)

Initialization begins with writing ICW1. Use ICW1 to select the interrupt request triggering type (level or edge). The following actions occur within an 82C59A module when its ICW1 is written:

- The interrupt mask register is cleared, enabling all interrupt request signals.
- The IR7 signal is assigned the lowest interrupt level (default).
- Special mask mode is disabled.

Initialization Command Word 1
ICW1 (master and slave)
(write only)

	master	slave
Expanded Addr:	**F020H**	**F0A0H**
ISA Addr:	**0020H**	**00A0H**
Reset State:	**XXH**	**XXH**

7							0
0	0	0	RSEL1	LS	0	0	1

Bit Number	Bit Mnemonic	Function
7–5	—	Clear these bits to guarantee device operation.
4	RSEL1	Register Select 1 (Also see OCW2 and OCW3): ICW1, OCW2, and OCW3 are accessed through the same addresses. 0 = OCW2 or OCW3 is accessed (Figure 9-13 and Figure 9-15). 1 = ICW1 register is accessed.
3	LS	Level/Edge Sensitive: 0 = Selects edge-triggered IR input signals. 1 = Selects level-sensitive IR input signals. All internal peripherals interface with the 82C59As in edge-triggered mode only. This is compatible with the PC/AT bus specification. Each source signal initiates an interrupt request by making a low-to-high transition. External peripherals interface with the 8259As in edge-triggered or level-sensitive mode. The modes are selected for the device, not for individual interrupts. NOTE: If an internal peripheral interrupt is used, the 8259A that the interrupt is connected to must be programmed for edge-triggered interrupts.
2–1	—	Clear these bits to guarantee device operation.
0	—	Set this bit to guarantee device operation.

NOTE: The 82C59A must be initialized before it can be used. After reset, the 82C59A register states are undefined. The 82C59A modules must be initialized before the IF flag in the core FLAG register is set. All peripherals that use interrupts connected to the ICU must be initialized before initializing the ICU.

Figure 9-8. Initialization Command Word 1 Register (ICW1)

9.3.4 Initialization Command Word 2 (ICW2)

Use the ICW2 register to define the base interrupt vector for the 82C59A. Valid vector numbers for maskable interrupts range from 32 to 255. Because the base vector number must reside on an 8-byte boundary, the valid base vector numbers are $32 + n \times 8$ where $0 \leq n \leq 27$. Write the base interrupt vector's five most-significant bits to ICW2's five most-significant bits. The 82C59A determines specific IR signal vector numbers by adding the number of the IR signal to the base interrupt vector.

Initialization Command Word 2
ICW2 (master and slave)
(write only)

	master	slave
Expanded Addr:	**F021H**	**F0A1H**
ISA Addr:	**0021H**	**00A1H**
Reset State:	**XXH**	**XXH**

7							0
T7	T6	T5	T4	T3	0	0	0

Bit Number	Bit Mnemonic	Function
7–3	T7:3	Base Interrupt Type: Write the base interrupt vector's five most-significant bits to these bits.
2–0	T2:0	Clear these bits to guarantee device operation.

Figure 9-9. Initialization Command Word 2 Register (ICW2)

9.3.5 Initialization Command Word 3 (ICW3)

The ICW3 register contains information about the master/slave connections. For this reason, the functions of the master's ICW3 and the slave's ICW3 differ.

ICW3 (at 0F021H or 0021H) is the master's cascade configuration register (Figure 9-11). The master has an internal slave cascaded from its IR2 signal. You can cascade additional slaves from the master's IR7, IR6, IR5, IR4, IR3 and IR1 signals. Setting a bit indicates that a slave 82C59A is cascaded from the corresponding master's IR signal.

NOTE

Since the internal slave is cascaded from the master's IR2 signal, you must set the S2 bit.

Initialization Command Word 3
ICW3 (master)
(write only)

Expanded Addr: **F021H**
ISA Addr: **0021H**
Reset State: **XXH**

7							0
S7	S6	S5	S4	S3	S2	S1	0

Bit Number	Bit Mnemonic	Function
7–3	S7:3	Slave IRs 0 = No slave 8259A is attached to the corresponding IR signal of the master. 1 = A slave 82C59A is attached to the corresponding IR signal of the master.
2	S2	0 = Internal slave not used 1 = Internal slave is cascaded from the master's IR2 signal.
1	S1	Slave IRs 0 = No slave 8259A is attached to the master through the IR1 signal of the master. 1 = A slave 82C59A is attached to the IR1 signal of the master.
0	—	Clear this bit to guarantee device operation.

Figure 9-10. Initialization Command Word 3 Register (ICW3 – Master)

ICW3 (at 0F0A1H or 00A1H) is the internal slave ID register (Figure 9-11). Use this register to indicate that the slave is cascaded from the master's IR2 signal. This gives the internal slave an ID of 2. Each slave device uses the IDs to determine whether it is the slave being addressed. During a slave access, the slave's ID is driven on the master's CAS2:0 signals. If these signals are enabled (bit 7 of INTCFG is 1), they appear on the A18:16 address lines.

Initialization Command Word 3
ICW3 (slave)
(write only)

Expanded Addr: **F0A1H**
ISA Addr: **00A1H**
Reset State: **XXH**

7							0
0	0	0	0	0	0	1	0

Bit Number	Bit Mnemonic	Function
7–2	—	Clear these bits to guarantee device operation.
1	—	Set this bit to guarantee device operation.
0	—	Clear this bit to guarantee device operation.

Figure 9-11. Initialization Command Word 3 Register (ICW3 – Slave)

9.3.6 Initialization Command Word 4 (ICW4)

Use ICW4 to select the special-fully nested mode or the fully nested mode and to enable the automatic EOI mode.

Initialization Command Word 4
ICW4 (master and slave)
(write only)

	master	slave
Expanded Addr:	**F021H**	**F0A1H**
ISA Addr:	**0021H**	**00A1H**
Reset State:	**XXH**	**XXH**

7							0
0	0	0	SFNM	0	0	AEOI	1

Bit Number	Bit Mnemonic	Function
7–5	—	Write zero to these bits to guarantee device operation.
4	SFNM	Special-fully Nested Mode: 0 = Selects fully nested mode. 1 = Selects special-fully nested mode. Only the master 82C59A can operate in special-fully nested mode.
3–2	—	Write zero to these bits to guarantee device operation.
1	AEOI	Automatic EOI Mode: 0 = Disables automatic EOI mode. 1 = Enables automatic EOI mode. Only the master 82C59A can operate in automatic EOI mode.
0	—	Write one to this bit to guarantee device operation.

Figure 9-12. Initialization Command Word 4 Register (ICW4)

9.3.7 Operation Command Word 1 (OCW1)

OCW1 is the interrupt mask register. Setting a bit in the interrupt mask register disables (masks) interrupts from the corresponding IR signal. For example, setting the master's OCW1 M3 bit disables interrupts from the master IR3 signal. Clearing a bit in the interrupt mask register enables interrupts from the corresponding IR signal.

Operation Command Word 1
OCW1 (master and slave)
(read/write)

	master	slave
Expanded Addr:	**F021H**	**F0A1H**
ISA Addr:	**0021H**	**00A1H**
Reset State:	**XXH**	**XXH**

7							0
M7	M6	M5	M4	M3	M2	M1	M0

Bit Number	Bit Mnemonic	Function
7–0	M7:0	Mask IR: 0 = Enables interrupts on the corresponding IR signal. 1 = Disables interrupts on the corresponding IR signal. NOTE: Setting the mask bit does not clear the respective interrupt pending bit.

NOTE: The 8259A must be initialized before it can be used. After reset, the 8259A register states are undefined. The 8259A modules must be initialized before the IF flag in the core FLAG register is set.

Figure 9-13. Operation Command Word 1 (OCW1)

9.3.8 Operation Command Word 2 (OCW2)

Use OCW2 to change the priority structure and issue EOI commands.

Operation Command Word 2
OCW2 (master and slave)
(write only)

	master	slave
Expanded Addr:	**F020H**	**F0A0H**
ISA Addr:	**0020H**	**00A0H**
Reset State:	**XXH**	**XXH**

7							0
R	SL	EOI	RSEL1	RSEL0	L2	L1	L0

Bit Number	Bit Mnemonic	Function
7	R	The Rotate (R), Specific Level (SL), and End-of-Interrupt (EOI) Bits: These bits change the priority structure and/or send an EOI command. **R SL EOI Command** 0 0 0 Cancel automatic rotation* 0 0 1 Send a nonspecific EOI command 0 1 0 No operation 0 1 1 Send a specific EOI command** 1 0 0 Enable automatic rotation* 1 0 1 Enable automatic rotation and send a nonspecific EOI 1 1 0 Initiate specific rotation** 1 1 1 Initiate specific rotation and send a specific EOI** * These cases allow you to change the priority structure while the 82C59A is operating in the automatic EOI mode. ** The L2:0 bits (see below) specify the specific level for these cases.
6	SL	
5	EOI	
4–3	RSEL1:0	Register Select Bits: ICW1, OCW2 and OCW3 are accessed through the same addresses. The states of RSEL1:0 determine which register is accessed. **Write 00 to these bits to access OCW2.** **RSEL1 RSEL0** 0 0 OCW2 0 1 OCW3 1 X ICW1
2–0	L2:0	IR Level: When you program bits 7–5 to initiate specific rotation, these bits specify the IR signal that is assigned the lowest level. When you program bits 7–5 to send a specific EOI command, these bits specify the IR signal that receives the EOI command. If SL=0, then these bits have no effect.

Figure 9-14. Operation Command Word 2 (OCW2)

9.3.9 Operation Command Word 3 (OCW3)

Use OCW3 to enable the special mask mode, issue a poll command, and provide access to the interrupt in-service and request registers (ISR, IRR).

Operation Command Word 3
OCW3 (master and slave)
(write only)

	master	slave
Expanded Addr:	**F020H**	**F0A0H**
ISA Addr:	**0020H**	**00A0H**
Reset State:	**XXH**	**XXH**

7							0
0	ESMM	SMM	RSEL1	RSEL0	POLL	ENRR	RDSEL

Bit Number	Bit Mnemonic	Function
7	—	Clear this bit to guarantee device operation.
6	ESMM	Enable Special Mask Mode (ESMM) and Special Mask Mode (SMM): Use these bits to enable or disable special mask mode. **ESMM SMM** 0 0 No action 0 1 No action 1 0 Disable special mask mode 1 1 Enable special mask mode
5	SMM	
4–3	RSEL1:0	Register Select: ICW1, OCW2 and OCW3 are accessed through the same addresses. The states of RSEL1:0 determine which register is accessed. **Write 01 to these bits to access OCW3.** **RSEL1 RSEL0** 0 0 OCW2 0 1 OCW3 1 X ICW1
2	POLL	Poll Command: Set this bit to issue a poll command, thus initiating the polling process.
1	ENRR	Enable Register Read Select (ENRR) and Read Register Select (RDSEL): These bits select which register is read during the next F020H and F0A0H (or PC/AT address 0020H, 00A0H) read access. **ENRR RDSEL Register Read on Next Read Pulse** 0 0 No action 0 1 No action 1 0 Interrupt Request Register 1 1 In-service Register
0	RDSEL	

Figure 9-15. Operation Command Word 3 (OCW3)

9.3.10 Interrupt Request Register (IRR)

This 8-bit, read-only register contains the levels requesting an interrupt to be acknowledged. It is accessed using OCW3 (see Figure 9-15). The highest request level is reset from the IRR when an interrupt is acknowledged. Bits 7:0 of this register are the pending bits, respectively, of interrupt requests IR7:0.

9.3.11 In-Service Register (ISR)

This 8-bit, read-only register contains the priority levels that are being serviced. It is accessed using OCW3 (see Figure 9-15). The ISR is updated when an End-of-Interrupt command is issued. Bits 7:0 of this register are the in-service bits, respectively, of interrupt requests IR7:0.

9.3.12 Poll Status Byte (POLL)

Read the poll status byte after issuing a poll command to determine whether any of the devices connected to the 82C59A require servicing. Once the polling bit is set in OCW3, the Poll Status Byte of a particular 82C59A can be read by doing an access to any of the four addresses of that 82C59A.

Poll Status Byte
POLL (master and slave)
(read only)

	master	**slave**
Expanded Addr:	**F020H**	**F0A0H**
ISA Addr:	**0020H**	**00A0H**
Reset State:	**XXH**	**XXH**

7							**0**
INT	—	—	—	—	L2	L1	L0

Bit Number	Bit Mnemonic	Function
7	INT	Interrupt Pending: 0 = No request pending. 1 = Indicates that a device attached to the 82C59A requires servicing.
6–3	—	Reserved. These bits are undefined.
2–0	L2:0	Interrupt Request Level: When bit 7 is set, these bits indicate the highest-priority IR signal that requires servicing. When bit 7 is clear, i.e., no request is pending, these bits are indeterminate.

Figure 9-16. Poll Status Byte (POLL)

9.4 DESIGN CONSIDERATIONS

The following sections discuss some design considerations.

9.4.1 Interrupt Acknowledge Cycle

When the core receives an interrupt request from the master, it completes the instruction in progress and any succeeding locked instructions, then initiates an interrupt acknowledge cycle. The interrupt acknowledge cycle generates an internal interrupt acknowledge (INTA#) signal that consists of two locked pulses (Figure 9-17). This INTA# signal is connected to the internal 82C59A interrupt acknowledge inputs. On the falling edge of the second INTA#, the 82C59A sets its interrupt in-service bit. It then clears its interrupt pending bit on the rising edge of the second INTA#. On the second INTA# falling edge, the addressed 82C59A (determined by the master's cascade signals) also drives the interrupt vector number on the data bus.

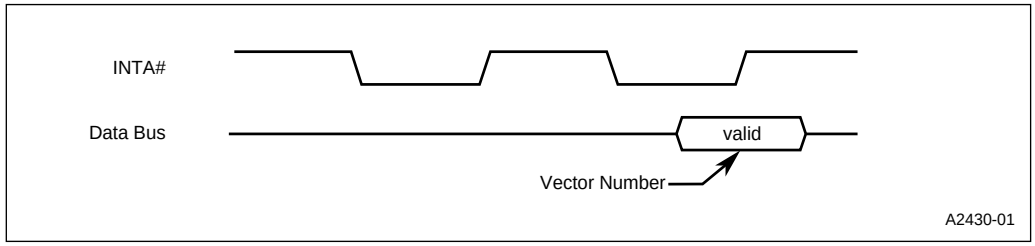


Figure 9-17. Interrupt Acknowledge Cycle

9.4.2 Interrupt Detection

The processing of an interrupt begins with the assertion of an interrupt request at one of the IR signals. During system initialization, you can program the IR signals, as a group, to be either edge or level triggered (using ICW1 described in Figure 9-8).

Edge triggered The 82C59A recognizes a rising edge transition on an IR signal as an interrupt request. A device requesting service must maintain a high state on an IR signal until after the falling edge of the first INTA# pulse. You can reset the edge-detection circuit during initialization of the 82C59A or by deasserting the IR signal. To reset the edge-detection circuit properly, the interrupt source must hold the IR line low for a minimum time of 10ns.

Level triggered The 82C59A recognizes a high level on an IR line as an interrupt request. A device must maintain the high level until after the falling edge of the first INTA# pulse. Unlike an edge-triggered IR signal, a level-triggered IR signal continues to generate interrupts as long as it is asserted. To avoid continuous interrupts from the same source, a device must deassert a level-sensitive IR signal before the interrupt handler issues an end-of-interrupt (EOI) command.

All internal peripherals interface with their respective 82C59As in edge-triggered mode. This is compatible with the PC/AT bus specification. Each source signal initiates an interrupt by making a low-to-high transition.

9.4.3 Spurious Interrupts

For both edge and level-triggered interrupts, a high level must be maintained on the IR line until after the falling edge of the first INTA# pulse (see Figure 9-18). A spurious interrupt request is generated if this stipulation is not met. A spurious interrupt on any IR line generates the same vector number as an IR7 request. The spurious interrupt, however, does not set the in-service bit for IR7. Therefore, an IR7 interrupt service routine must check the in-service register to determine whether the interrupt source was a valid IR7 (the in-service bit is set) or a spurious interrupt (the in-service bit is cleared).

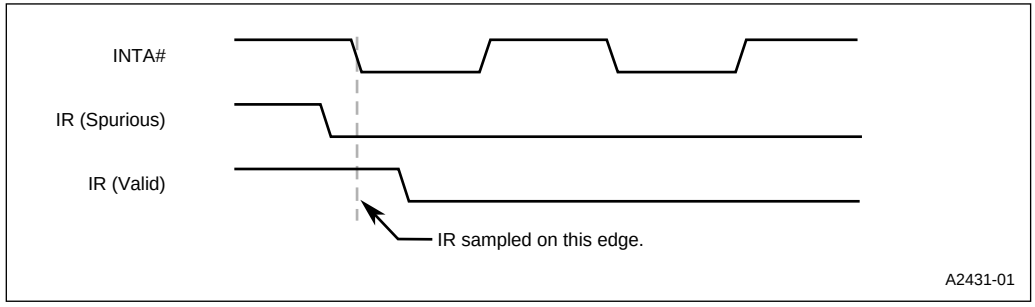


Figure 9-18. Spurious Interrupts

9.4.4 Cascading Interrupt Controllers

Figure 9-19 is a block diagram showing the connections for two cascaded 82C59As. The PLD generates READY# (for the second Interrupt Acknowledge Cycle) and INTA# to the external 82C59A devices. The PLD also generates appropriate timings for the INTA# signals to satisfy 82C59A specifications.

The RD# and WR# strobes are used to read and write to the 82C59A registers. These strobes are inactive during Interrupt Acknowledge Cycles.

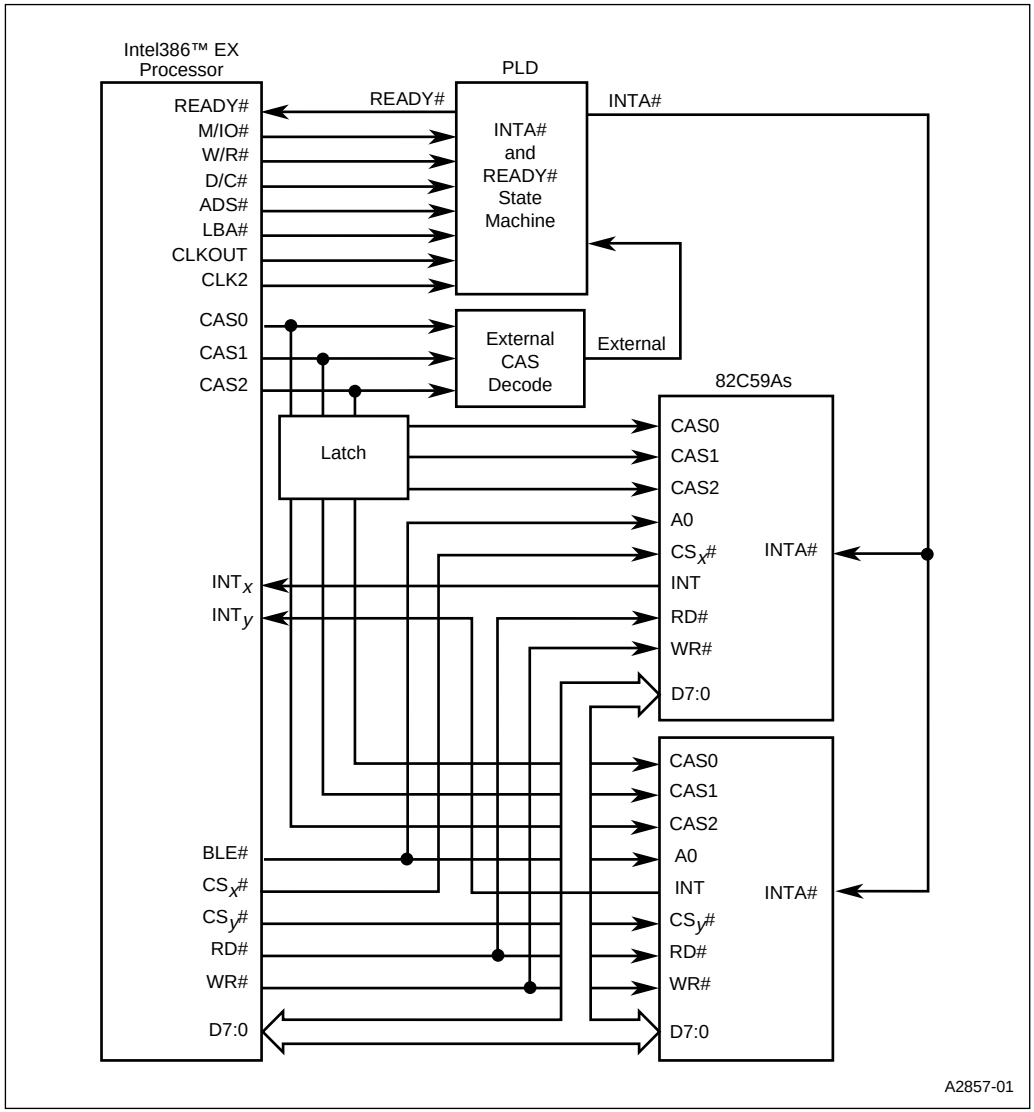


Figure 9-19. Cascading External 82C59A Interrupt Controllers

9.5 PROGRAMMING CONSIDERATIONS

Consider the following when programming the ICU.

- When an 82C59A receives an interrupt request, it sets the request's pending bit (regardless of whether the IR signal is masked or not). The pending bit remains set until the interrupt is serviced.
- When the LS bit in ICW1 is set to edge-triggered during initialization, all the interrupt pending bits will be cleared.
- In special-fully nested mode, care must be taken when processing interrupt requests from the master's internal cascade signal (IR2). At the end of the slave's interrupt service routine, first issue a nonspecific EOI to the slave. Before issuing a nonspecific EOI command to the master, make sure that the slave has no other in-service bits set.
- Systems that use polling as the only method of device servicing must still fully initialize the 82C59A modules. Also, the interrupt requests to the core must be disabled using the mask bits or the CLI instruction.
- The 8259A must be initialized before it can be used. After reset, the 82C59A register states are undefined. The 82C59A modules must be initialized before the IF flag in the core FLAG register is set. All peripherals that use interrupts connected to the ICU must be initialized before initializing the ICU.

9.5.1 Interrupt Control Unit Code Examples

The example code contains these software routines:

InitICU	Initializes the Master and Slave 82C59A Interrupt Controllers
InitICUSlave	Initializes the Slave 82C59A Interrupt Controllers
Disable8259Interrupt	Disables interrupts on the Master and internal Slave
Enable8259Interrupt	Enables interrupts on the Master and internal Slave
SetIRQVector	Loads the interrupt vector table with the address of the Interrupt Service Routine
SetInterruptVector	Called by SetIRQVector to load vector table
Poll_Command	Issues a poll command to read the poll status byte of the ICU

See Appendix C for included header files.

```
#include <conio.h>
#include “80386ex.h”
#include “EV386EX.h”

    /* Globals For information about the ICU */
BYTE    _IRQ_SlaveBase_= 0x30;
BYTE    _IRQ_MstrBase_ = 0x20;
```

```
BYTE    _CascadeBits_  = 0x4;

/*****************************************************************************
   InitICU

   Description:
      Initialization for both the master and slave Interrupt Control
      Units (ICU). tine only initializes the internal interrupt
      controllers, external ICUs must be initialized separately. These
      should be initialized before interrupts are enabled(i.e., enable()).

   Parameters:
      MstrMode         Mode of operation for Master ICU
      MstrBase         Specifies the base interrupt vector number for the
                       Master interrupts.
                       For example, if IR1 of the master goes active and the
                       MstrBase = 0x20, the processor uses interrupt
                       vector table entry 0x21.
      MstrCascade      Which Master IRQs are used for Slave ICUs.
      SlaveMode        Mode of operation for Slave ICU
      SlaveBase        Specifies the base interrupt vector number for the
                       Slave interrupts.
                       For example, if IR1 of the slave goes active and the
                       SlaveBase = 0x40, the processor uses interrupt
                       vector table entry 0x41.
      MstrPins         Defines what EX pins are available externally to the
                       chip for the Master.
      SlavePins        Defines what EX pins are available externally to the
                       chip for the Slave.
   Returns:Error Code
      E_OK             -- Initialized OK, No error.

   Assumptions:
      REMAPCFG register has Expanded I/O space access enabled (ESE bit set).

   Syntax:

      #define ICU_TRIGGER_EDGE   0x0
      #define MPIN_INT0          0x4
      #define MCAS_IR1           0x2
      #define SPIN_INT4          0x1

      int error_code;

      error_code = InitICU(ICU_TRIGGER_EDGE,
                           0x20,
                           MCAS_IR1,
                           ICU_TRIGGER_EDGE,
                           0x30,
                           MPIN_INT0,
                           SPIN_INT4);
```

```
   Real/Protected Mode
      No changes required.

*************************************************************************/

int InitICU(BYTE MstrMode, BYTE MstrBase, BYTE MstrCascade, BYTE SlaveMode,
            BYTE SlaveBase, BYTE MstrPins, BYTE SlavePins)
{
  BYTE  icw, cfg_pins;

/* Program Slave ICU */

  _IRQ_SlaveBase_ = SlaveBase & 0xf8;
  _SetEXRegByte(ICW1S, 0x11 | SlaveMode);// Set slave triggering
  _SetEXRegByte(ICW2S, _IRQ_SlaveBase_); // Set slave base interrupt type,
                                         // least 3-bit must be 0
  _SetEXRegByte(ICW3S, 0x2);             // Set slave ID
  _SetEXRegByte(ICW4S, 0x1);             // Set bit 0 to guarantee operation

/* Program Master ICU */

  _IRQ_MstrBase_ = MstrBase & 0xf8;
  _CascadeBits_  = MstrCascade | 0x4;
  icw = (MstrMode & ICU_TRIGGER_LEVEL) ? 0x19 : 0x11;
  _SetEXRegByte(ICW1M, icw);             // Set master triggering
  _SetEXRegByte(ICW2M, _IRQ_MstrBase_);  // Set master base interrupt
                                         // type, least 3-bit must be 0
  _SetEXRegByte(ICW3M, _CascadeBits_);   // Set master cascade pins,
                                         // Make sure IR2 set for Cascade
  icw = (MstrMode & ~ICU_TRIGGER_LEVEL) | 1;// Set bit 0 and remove
                                         // Trigger_level bit (in ICW1)
  _SetEXRegByte(ICW4M, icw);             // Set slave IDs in master

/* Program chip configuration registers */

  cfg_pins = _GetEXRegByte(INTCFG);
  if( (MstrCascade & 0xfb) != 0 )        // bit 2 (IR2) is internal,
                                         // external signals not required
                                         // for just IR2
   cfg_pins |= 0x80;                     // Using external slaves,
                                         // therefore enable Cascade signals
  cfg_pins |= SlavePins;
  _SetEXRegByte(INTCFG, SlavePins);      // Set Slave external interrupt pins
  cfg_pins = _GetEXRegByte(P3CFG);       // Preserve other set bits
  _SetEXRegByte(P3CFG, cfg_pins | MstrPins);// Set Master external
                                         // interrupt pins

  return E_OK;

}/* InitICU */
```

```
/*****************************************************************************
   InitICUSlave

   Description:
      Initialization only the internal slave Interrupt Control Units (ICU).
      This routine only initializes the internal interrupt controller,
      external ICUs must be initialized separately.

   Parameters:
      SlaveMode        Mode of operation for Slave ICU
      SlaveBase        Specifies the base interrupt vector number for the
                       Slave interrupts.
                       For example, if IR1 of the slave goes active and the
                       SlaveBase = 0x40 the processor uses interrupt
                       vector table entry 0x41.
       SlavePins       Defines what EX pins are available externally to the
                       chip for the Slave.

   Returns:Error Code
      E_OK                       -- Initialized OK, No error.

   Assumptions:
      REMAPCFG register has Expanded I/O space access enabled (ESE bit set).

   Syntax:

      /* ICU Modes */
      #define ICU_SFNM           0x10
      #define ICU_AUTOEOI        0x2
      #define ICU_TRIGGER_LEVEL  0x8
      #define ICU_TRIGGER_EDGE   0x0
      /* ICU Slave Pins */
      #define SPIN_INT4          0x1
      #define SPIN_INT5          0x2
      #define SPIN_INT6          0x4
      #define SPIN_INT7          0x8

      int error_code;

      error_code = InitICUSlave(ICU_TRIGGER_EDGE, 0x30, SPIN_INT4);

   Real/Protected Mode
      No changes required.

*****************************************************************************/

int InitICUSlave(BYTE SlaveMode, BYTE SlaveBase, BYTE SlavePins)
{
  BYTE  cfg_pins;

/* Program Slave ICU */
```

```
  _IRQ_SlaveBase_ = SlaveBase & 0xf8;
  _SetEXRegByte(ICW1S, 0x11 | SlaveMode);  // Set slave triggering
  _SetEXRegByte(ICW2S, _IRQ_SlaveBase_); // Set slave base interrupt
                                         // type, least 3-bit must be 0
  _SetEXRegByte(ICW3S, 0x2);             // Set slave ID
  _SetEXRegByte(ICW4S, 0x1);             // Set bit 0 to guarantee
                                         // operation
  cfg_pins = _GetEXRegByte(INTCFG);
  cfg_pins |= SlavePins;
  _SetEXRegByte(INTCFG, SlavePins);      // Set Slave external interrupt
                                         // pins

  return E_OK;

}/* InitICUSlave */




/*****************************************************************************
   Disable8259Interrupt:

   Description:
      Disables 8259a interrupts for the master and the slave.

   Parameters:
      MstrMask         Mask value for master ICU
      SlaveMask        Mask value for slave ICU

      Each bit location that is set disables the corresponding
      interrupt (by setting the bit in the interrupt control register).
      For example, to disable master IR3 and IR5 set MstrMask = 0x28
      (bits 3 and 5 are set).

   Returns:
      None

   Assumptions:
      REMAPCFG register has Expanded I/O space access enabled (ESE bit set).

   Syntax:

      /* ICU IRQ Mask Values*/
      #define IR0          0x1
      #define IR1          0x2
      #define IR2          0x4
      #define IR3          0x8
      #define IR4          0x10
      #define IR5          0x20
      #define IR6          0x40
```

```
      #define IR7          0x80

      Disable8259Interrupt(IR0 | IR1 | IR3 | IR4 | IR5 | IR6 | IR7,
                           IR1 | IR2 | IR3 | IR4 |IR5 | IR6);

   Real/Protected Mode
      No changes required.

*****************************************************************************/

void Disable8259Interrupt(BYTE MstrMask, BYTE SlaveMask)
{
   BYTE Mask;

   if(MstrMask != 0)
   {
      Mask = _GetEXRegByte(OCW1M);
      _SetEXRegByte(OCW1M, Mask | MstrMask);
   }

   if(SlaveMask != 0)
   {
      Mask = _GetEXRegByte(OCW1S);
      _SetEXRegByte(OCW1S, Mask | SlaveMask);
   }

}/* Disable8259Interrupt */


/*****************************************************************************
   Enable8259Interrupt:

   Description:
      Enables 8259a interrupts for the master and the slave.

   Parameters:
      MstrMask         Enable mask value for master ICU
      SlaveMask        Enable mask value for slave ICU

      Each bit location that is set enables the corresponding
      interrupt (by clearing the bit in the interrupt control register).
      For example, to enable master IR3 and IR5 set MstrMask = 0x28
      (bits 3 and 5 are set).

   Returns:
      None

   Assumptions:
      REMAPCFG register has Expanded I/O space access enabled (ESE bit set).

   Syntax:
```

```
      /* ICU IRQ Mask Values*/
      #define IR0          0x1
      #define IR1          0x2
      #define IR2          0x4
      #define IR3          0x8
      #define IR4          0x10
      #define IR5          0x20
      #define IR6          0x40
      #define IR7          0x80


      Enable8259Interrupts(IR2, IR0 | IR7); //Enable MasterIR2 for cascading
                                            //Enable INT4 and WDTOUT on Slave

   Real/Protected Mode
      No changes required.

*****************************************************************************/

void Enable8259Interrupt(BYTE MstrMask, BYTE SlaveMask)
{
   BYTE Mask;

   if(MstrMask != 0)
   {
      Mask = _GetEXRegByte(OCW1M);
      _SetEXRegByte(OCW1M, Mask & (~MstrMask));
   }

   if(SlaveMask != 0)
   {
      Mask = _GetEXRegByte(OCW1S);
      _SetEXRegByte(OCW1S, Mask & (~SlaveMask));
   }

}/* Enable8259Interrupt */


/*****************************************************************************
   SetIRQVector:

   Description:
      Loads the interrupt vector table with the address of the interrupt
      routine. The vector table entry number is determined by the vector
      number.

   Parameters:
      InterProc        Address of interrupt function, will be loaded into
                       the interrupt table.
      IRQ              Hardware Interrupt request number (0-15).
      ISR_Type         Specifies if the interrupt function should be treated
                       as a TRAP_ISR or an INTERRUPT_ISR. Real Mode only
```

```
                        supports INTERRUPT_ISR (parameter is ignored).
                        Protected mode supports both.

   Returns:Error Code
      E_INVALID_VECTOR    -- An IRQ of greater than 15 was passed
      E_BADVECTOR         -- IRQ is used for cascading to a slave interrupt
                             controller
      E_OK             -- Initialized OK, No error.

   Assumptions:
      Compiler supports far and interrupt keywords

      ICU must be configured before this function is call for it to
      operate properly

      _IRQ_SlaveBase_,_IRQ_MstrBase_,_CascadeBits_ are set before function
      is called. These are initialized in the InitICU functions supplied
      in this source.

   Syntax:

      int error_code;

      error_code = SetIRQVector(wdtISR,
                           15,   // Slave IR#'s are offset by 8 in
                                 // Vector Table
                           INTERRUPT_ISR);


   Real/Protected Mode
      No changes required.  Uses SetInterruptVector which is mode
      dependant (separate source)

*****************************************************************************/

int SetIRQVector( void (far interrupt *IntrProc)(void), int IRQ, int IntrType)
{
   int Vector;

   if(IRQ > 15) return E_INVALID_VECTOR;

   if(IRQ > 7)             // Get Vector from Slave
      Vector = _IRQ_SlaveBase_ + IRQ - 8;

   else                    // From Master
   {
      if((1 << IRQ) & _CascadeBits_) return E_BADVECTOR;
      Vector = _IRQ_MstrBase_ + IRQ;
   }

    SetInterruptVector(IntrProc, Vector, IntrType);
```

```
   return E_OK;

}/* SetIRQVector */



/*****************************************************************************
   SetInterruptVector:

   Description:
      Loads the interrupt vector table with the address of the interrupt
      routine. The vector table entry number is determined by the vector
      number.

   Parameters:
      InterProc        Address of interrupt function, will be loaded into
                       the interrupt table.
      ISR_Type         Specifies if the interrupt function. Real Mode only
                       supports INTERRUPT_ISR (the parameter is ignored). The
                       parameter is kept to maintain compatibility with the
                       protected mode version of this function.

   Returns:
      None

   Assumptions:
      Compiler supports far and interrupt keywords
      Compiler may issue a warning about IntrType not used.
      IntrType is kept for protected mode compatibility.

   Syntax:

      SetInterruptVector(wdtISR, INTERRUPT_ISR);

   Real/Protected Mode
      Real Mode only

*****************************************************************************/

void SetInterruptVector( void (far interrupt *IntrProc)(void),
                         int Vector, int IntrType)
{
   (void) IntrType;    // Reference to avoid compiler warning

   ((unsigned long far *)(0))[Vector] = (unsigned long)IntrProc;

}/* SetInterruptVector */



/*****************************************************************************
```

```
   Poll_Command:

   Description:
      This routine issues a poll command which reads the poll status byte
      of the ICU.

   Parameters:
      Master_or_Slave  Specifies which interrupt controller is polled

   Returns:
      Current value of poll status byte

   Assumptions:
      None

   Syntax:

      in poll_status;

      poll_status = Poll_Command();

   Real/Protected Mode:
      No changes required.

*****************************************************************************/

  int Poll_Command( int Master_or_Slave)
{

  int poll_status;

  if (Master_or_Slave == Master) {

      _SetEXRegByte(OCW3M, 0x0c);//Initiate polling sequence
      poll_status = _GetEXRegByte(ICW2M);

  }

  else {

      _SetEXRegByte(OCW3S, 0x0c); //Initiate polling sequence
      poll_status = _GetEXRegByte(ICW2S);
 }

 return(poll_status);

} /* Poll_Command */
```

10

TIMER/COUNTER UNIT

CHAPTER 10
TIMER/COUNTER UNIT

The Timer/counter Unit (TCU) has the same basic functionality as the industry-standard 82C54 counter/timer. It contains three independent 16-bit down counters, which can be driven by a prescaled value of the processor clock or an external clock. The counters contain two count formats (binary and BCD) and six different operating modes, two of which are periodic. Both hardware and software triggered modes exist, providing for internal or external control. The counter's output signals can appear at device pins, generate interrupt requests, and initiate DMA transactions.

This chapter is organized as follows:

10.1 OVERVIEW

The TCU contains control logic and three independent 16-bit down counters (Figure 10-1). Each counter has two input signals and one output signal:

CLKIN*n* You can independently connect each counter's clock input (CLKIN*n*) signal to either the internal prescaled clock (PSCLK) signal or the external timer clock (TMRCLK*n*) pin. This allows you to use either a prescaled value of the processor's internal clock or an external clock to drive each counter.

NOTE

The maximum CLKIN*n* frequency, whether connected internally or externally, is 8 MHz.

GATE*n* Each counter has a gate (GATE*n*) input signal. This signal provides counter operation control. In some of the counter operating modes, a high level on a counter's GATE*n* signal enables or resumes counting and a low level disables or suspends counting. In other modes, a rising edge on GATE*n* loads a new count value. You can independently connect each counter's GATE*n* signal to either V_{CC} or the external timer gate (TMRGATE*n*) pin, or you can drive each counter's GATE*n* signal high or low through register bits.

OUT*n* Each counter contains an output signal called OUT*n*. You can independently connect these signals to the external timer clock output (TMROUT*n*) pins. OUT0, OUT1, and OUT2 are routed to the interrupt control unit. OUT1 is also routed to DMA channel 0, and OUT2 is also routed to DMA channel 1.

Therefore, the OUT*n* signals can drive external devices, generate interrupt requests, initiate DMA transactions or combinations of the three.

Each counter operates independently. Six different counting modes are available and two count formats: binary (16 bits) or BCD (4 decades). Each operating mode allows you to program the counter with an initial count and to change this value "on the fly." You can determine the count and status of each counter without disturbing its current operation.

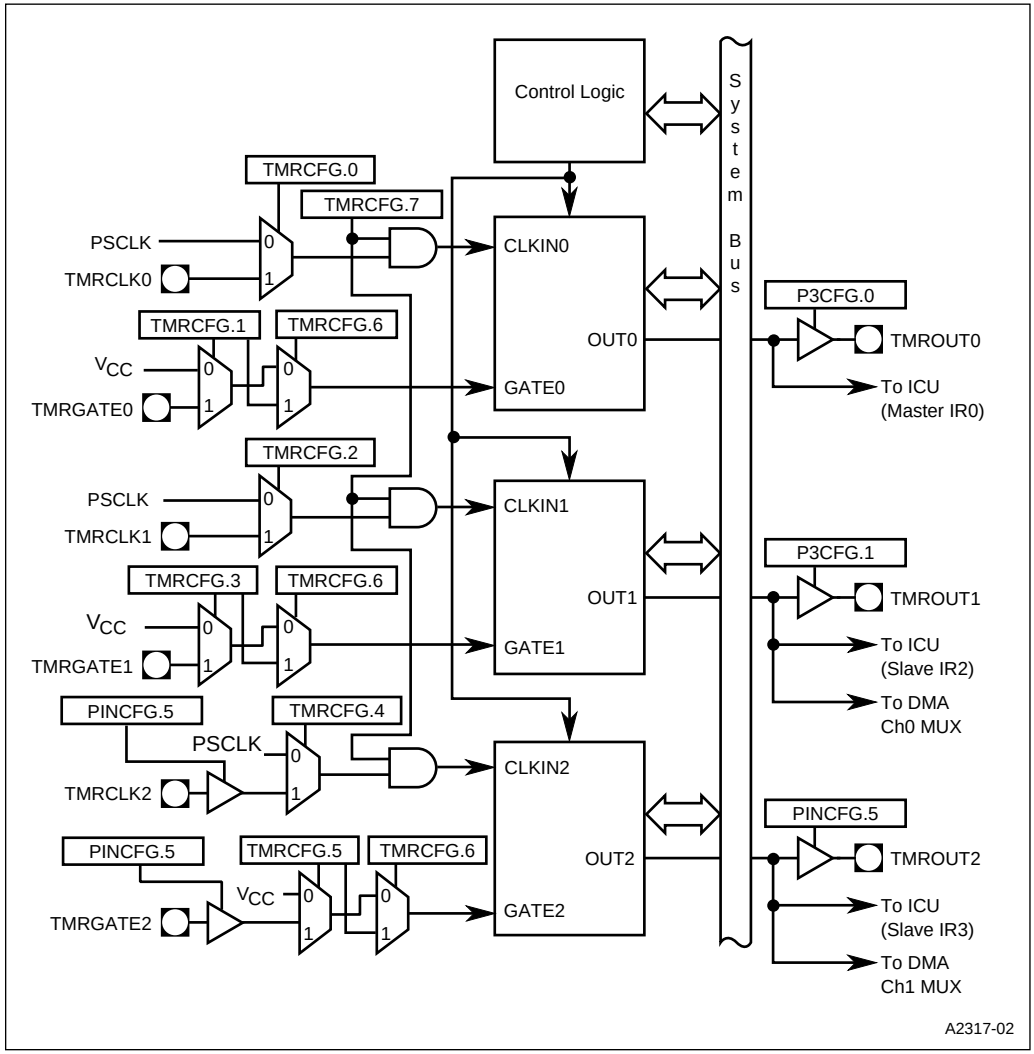


Figure 10-1. Timer/Counter Unit Signal Connections

10.1.1 TCU Signals and Registers

Table 10-1 and Table 10-2 lists the signals and registers associated with the TCU.

Table 10-1. TCU Signals

Signal	Device Pin or Internal Signal	Description
PSCLK	Internal signal	Prescaled Clock: This is one of the two possible connections for the counter's CLKIN*n* signal. PSCLK is an internal signal that is a prescaled value of the processor internal clock. The clock and power management unit contains a programmable divider that determines the PSCLK frequency. See "Controlling the PSCLK Frequency" on page 8-7, for information on how to program PSCLK's frequency.
TMRCLK0 TMRCLK1 TMRCLK2	Device pin	Timer Clock Input: This is one of the two possible connections for the counter's CLKIN*n* signal. You can drive a counter with an external clock source by connecting the clock source to the counter's TMRCLK*n* pin.
TMRGATE0 TMRGATE1 TMRGATE2	Device pin	Timer Gate Input: This input can be connected to the counter's GATE*n* input to control the counter's operation. In some of the counter's operating modes, a high level on GATE*n* enables or resumes counting, while a low level disables or suspends counting. In other modes, a rising edge on GATE*n* loads a new count value.
TMROUT0 TMROUT1 TMROUT2	Device pin	Timer Output: The counter's OUT*n* signal can be connected to this pin. The operation, and consequently the waveform, of the output depends on the counter's operating mode.

Table 10-2. TCU Associated Registers

Register	Expanded Address	PC/AT* Address	Function
P3CFG PINCFG (read/write)	0F824H 0F826H	—	Peripheral Pin Selections: These registers determine whether a counter's input and output signals are connected to package pins.
TMRCFG (read/write)	0F834H	—	Timer Configuration: Enables the counter's CLKIN*n* input signal, selects the CLKIN*n* connection (PSCLK or TMRCLK*n*) for each counter, and either connects TMRGATE*n* or V_{CC} to each counter's GATE*n* input signal, or sets GATE*n* high or low through register bits.
TMRCON	0F043H	0043H	TMRCON has three formats: control word, counter-latch, and read-back. When writing to TMRCON, certain bit settings determine which format is accessed. *Control Word Format*: Programs a specific counter. Selects a counter's operating mode and count format. After programming a counter, you can write a count value to the counter's TMR*n* register at any time. *Counter-latch Format*: Issues a counter-latch command to a specific counter. The counter-latch command allows you to latch the count of a specified counter. After issuing a counter-latch command, you can check the counter's count by reading the counter's TMR*n* register. *Read-back Format*: Issues a read-back command to one or more counters. The read-back command allows you to latch the count and status of one or more counters. After issuing the read-back command, you can check the counter's status by reading the counter's TMR*n* register. After checking a counter's status, you can read the counter's TMR*n* register again to check its count.
TMR0 TMR1 TMR2	0F040H 0F041H 0F042H	0040H 0041H 0042H	*Status Format:* Read this register after issuing a read-back command to check counter *n*'s status. Reading TMR*n* again accesses its read format. *Read Format*: Read this register to check counter *n*'s count value. *Write Format*: Write this register at any time after initializing counter *n* to change the counter's count value.

10.2 TCU OPERATION

Each counter can operate in any one of six operating modes. These modes are described in sections 10.2.1 through 10.2.6. In all modes, the counters decrement on the falling edge of CLKIN*n*. In modes 0, 1, 4, and 5, the counters roll over to the highest count, either 0FFFFH for binary counting or 9999 for BCD counting, and continue counting down. However, the state of the OUT*n* is only affected by the first run through the counter and does not change on subsequent runs. Modes 2 and 3 are periodic modes; in these modes, when the counter reaches terminal count it is reloaded with the currently programmed count value.

To specify a counter's operating mode, write to the TMRCON register's control word format. Writing to this register initiates counting. To specify a count, write to the counter's TMR*n* register's write format. In modes 0 and 4, the count is loaded on the falling edge of CLKIN*n*. Modes 1 and 5 require a rising edge on a counter's GATE*n* signal (or gate-trigger) to load the count. In modes 2 and 3, the count is loaded when the counter reaches terminal count or when the counter receives a gate-trigger, whichever is first.

The GATE*n* signal affects the counting operation for each mode differently (Table 10-3). For modes 0, 2, 3, and 4, GATE*n* is level sensitive, and the logic level is sampled on the rising edge of CLKIN*n*. The action then occurs on the falling edge of the next CLKIN*n*. In these modes, GATE*n* must be high for counting to begin. During a counting sequence, a low level at GATE*n* suspends counting, while a high level at GATE*n* resumes counting.

For modes 1, 2, 3, and 5, GATE*n* is rising-edge sensitive. In these modes, a rising edge at GATE*n* sets an edge-sensitive flip-flop in the counter. This flip-flop is then sampled on the next rising edge of CLKIN*n*; the flip-flop is reset immediately after it is sampled. In this way, a trigger is detected no matter when it occurs - a high level does not have to be maintained until the next rising edge of CLKIN*n*. Therefore, a rising edge on GATE*n* that occurs between two rising CLKIN*n* edges is recognized as a gate-trigger. The operation caused by a gate-trigger occurs on the falling CLKIN*n* edge following the trigger. Note that in modes 2 and 3, the GATE*n* input is both edge- and level-sensitive. In modes 1, 2, 3, and 5, a gate-trigger causes the counter to load new count values.

Table 10-3. Operations Caused by GATE*n*

Operating Modes	Low or Falling	Rising	High
0	Disables counting	—	Enables counting
1	—	1) Initiates counting 2) Resets OUT*n* after next CLKIN*n*	—
2	1) Disables counting 2) Sets OUT*n* immediately high	Initiates counting	Enables counting
3	1) Disables counting 2) Sets OUT*n* immediately high	Initiates counting	Enables counting
4	Disables counting	—	Enables counting
5	—	Initiates counting	—

10.2.1 Mode 0 – Interrupt on Terminal Count

This mode allows you to generate a rising edge on a counter's OUT*n* signal. Initializing a counter for mode 0 drives the counter's OUT*n* signal low and initiates counting. When the counter reaches terminal count, OUT*n* is driven high. At this point, the counter rolls over and continues counting with OUT*n* high. OUT*n* stays high and the counter keeps counting down and rolling over until a new count is written or you reprogram the counter. You can write a new count to the counter at any time to drive OUT*n* low and start a new counting sequence. Writing a new control word reprograms the counter.

Mode 0's basic operation is outlined below and shown in Figure 10-2.

1. After a control word write, OUT*n* is driven low.
2. On the CLKIN*n* pulse following a count write, the count is loaded.
3. On each succeeding CLKIN*n* pulse, the count is decremented.
4. When the count reaches terminal count, OUT*n* is driven high.

NOTE

Writing a count of N causes a rising edge on OUT*n* in N + 1 CLKIN*n* pulses (provided GATE*n* remains high and count was written before the rising edge of CLKIN*n*).

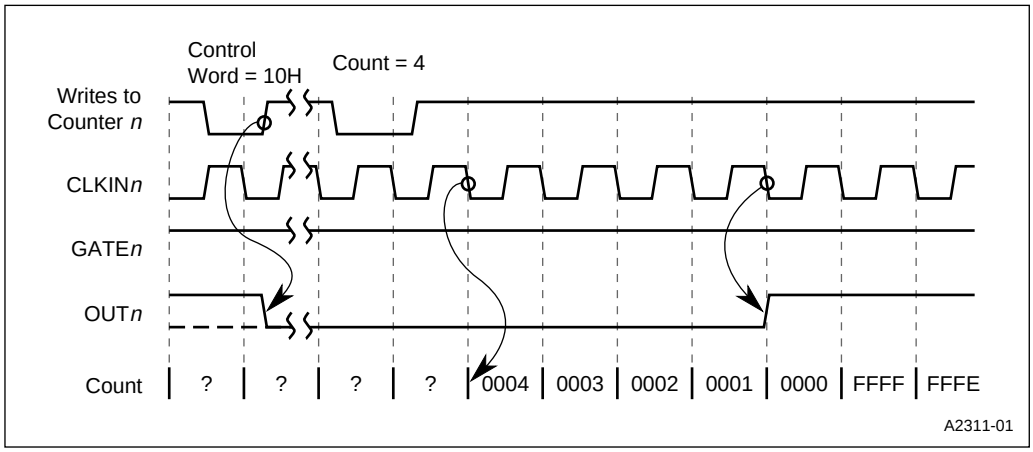


Figure 10-2. Mode 0 – Basic Operation

Figure 10-3 shows suspending the counting sequence. A low level on GATE*n* causes the counter to suspend counting (both the state of OUT*n* and the count remain unchanged). A high level on GATE*n* resumes counting.

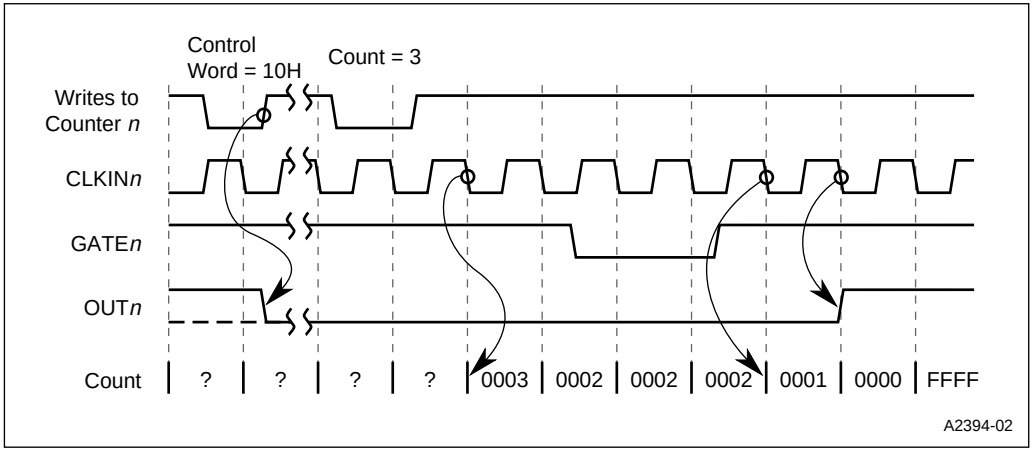


Figure 10-3. Mode 0 – Disabling the Count

Figure 10-4 shows writing a new count before the current count reaches zero. The counter loads the new count on the CLKIN*n* pulse after you write it, then decrements this new count on each succeeding CLKIN*n* pulse. OUT*n* remains low until the new count reaches zero.

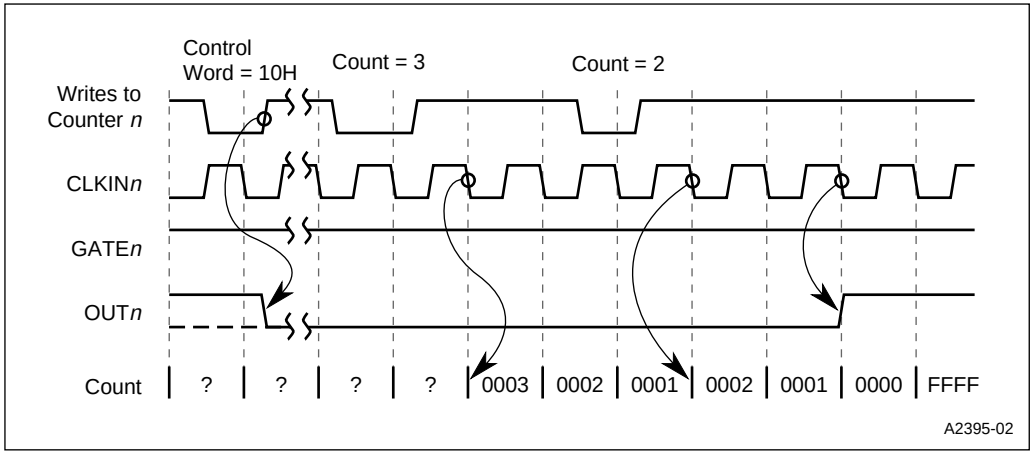


Figure 10-4. Mode 0 – Writing a New Count

10.2.2 Mode 1 – Hardware Retriggerable One-shot

This mode is similar to mode 0; it allows you to generate a rising edge on a counter's OUT*n* signal. Unlike mode 0, however, the counter waits for a gate-trigger before loading the count and driving its OUT*n* signal low. When the counter reaches zero, OUT*n* is driven high. At this point, the counter rolls over and continues counting with OUT*n* high. OUT*n* stays high and keeps counting down and rolling over until the counter receives another gate-trigger or you reprogram it. You can retrigger the one-shot at any time with a gate-trigger, causing the counter to reload the count and drive OUT*n* low. Writing a new control word to the counter reprograms it.

Mode 1's basic operation is outlined below and shown in Figure 10-5.

1. After a control word write, OUT*n* is driven high.
2. On the CLKIN*n* pulse following a gate-trigger, the count is loaded and OUT*n* is driven low.
3. On each succeeding CLKIN*n* pulse, the count is decremented.
4. When the count reaches zero, OUT*n* is driven high.

NOTE

Writing a count of N causes a rising edge on OUT*n* in N CLKIN*n* pulses after the count is loaded (using a gate-trigger).

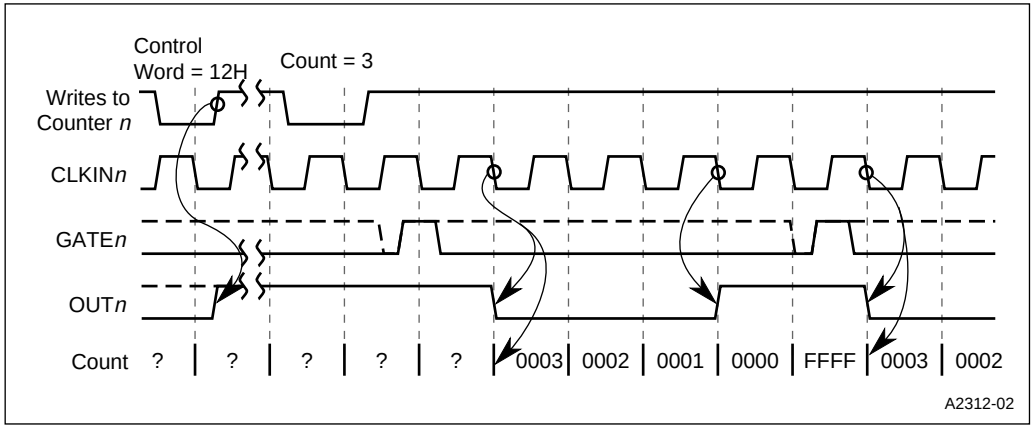


Figure 10-5. Mode 1 – Basic Operation

Figure 10-6 shows retriggering the one-shot. On the CLKIN*n* pulse following the retrigger, the counter reloads the count. The control logic then decrements the count on each succeeding CLKIN*n* pulse; OUT*n* remains low until the count reaches zero.

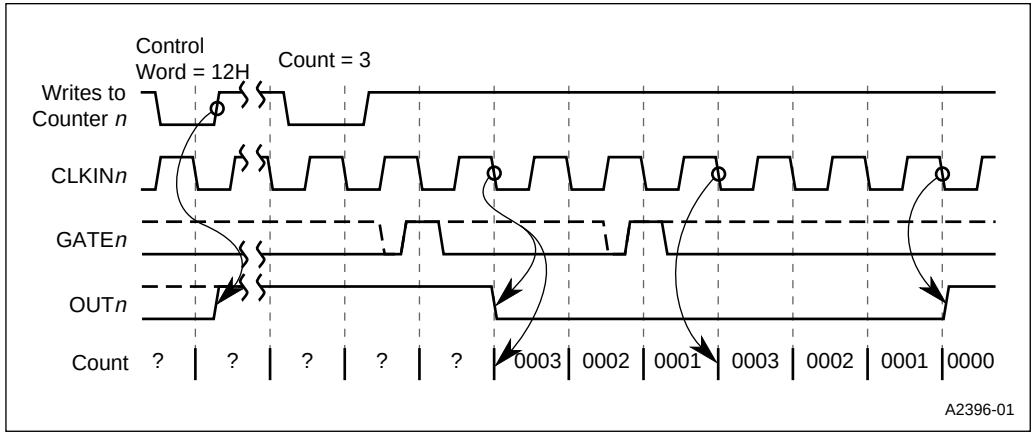


Figure 10-6. Mode 1 – Retriggering the One-shot

Figure 10-7 shows writing a new count. The counter waits for a gate-trigger to load the new count. The counter loads the new count on the CLKIN*n* pulse following the trigger, then decrements the count on each succeeding CLKIN*n* pulse. OUT*n* remains low until the count reaches zero.

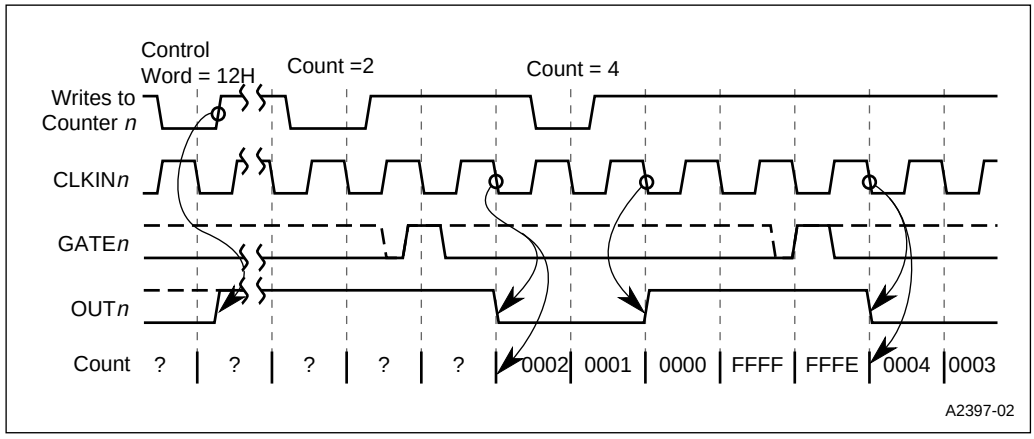


Figure 10-7. Mode 1 – Writing a New Count

10.2.3 Mode 2 – Rate Generator

In this periodic mode, a counter's OUT*n* signal remains high until the count reaches one, then goes low for one clock pulse (CLKIN*n*). After this single clock pulse, OUT*n* goes high and the count is reloaded. The cycle then repeats. You can use a gate-trigger to reload the count at any time. This provides a way to synchronize the counting cycle. A high level on a counter's GATE*n* signal enables counting; a low level on a counter's GATE*n* signal disables counting.

Mode 2's basic operation is outlined below and shown in Figure 10-8.

1. After a control word write, OUT*n* is driven high.
2. The count is loaded on the CLKIN*n* pulse following one of these events:
   - A write to a control word followed by a write to count
   - A gate trigger
   - The counter reaches one
3. On each succeeding CLKIN*n* pulse, the count is decremented.
4. When the count reaches one, OUT*n* is driven low.
5. On the following CLKIN*n* pulse, OUT*n* is driven high and the count is reloaded.
6. The process is repeated from step 3.

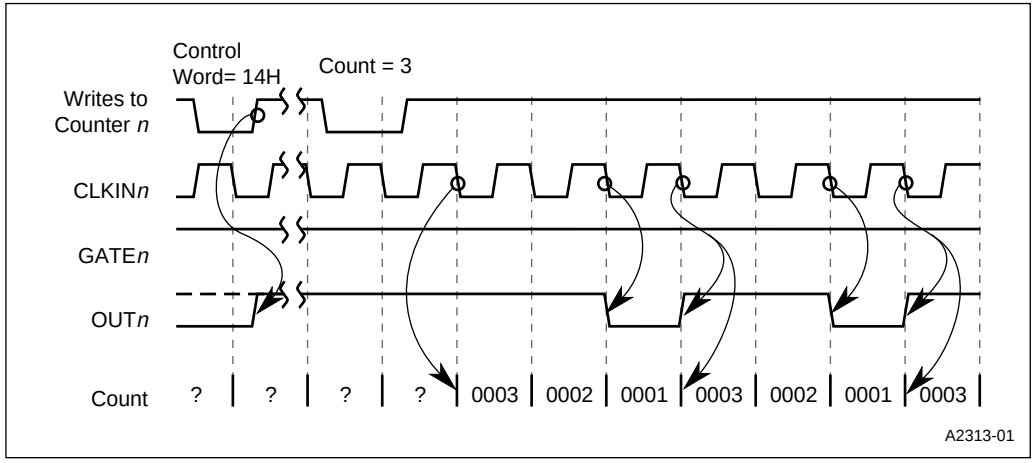


Figure 10-8. Mode 2 – Basic Operation

Figure 10-9 shows suspending the counting sequence. A low level on GATE*n* causes the counter to suspend counting. The count remains unchanged and OUT*n* is immediately driven (or stays) high (If the GATE*n* goes low when OUT*n* is low, then OUT*n* is immediately driven high). A rising edge on the GATE*n* causes the counter to be reloaded with the count. A high level on GATE*n* resumes counting.

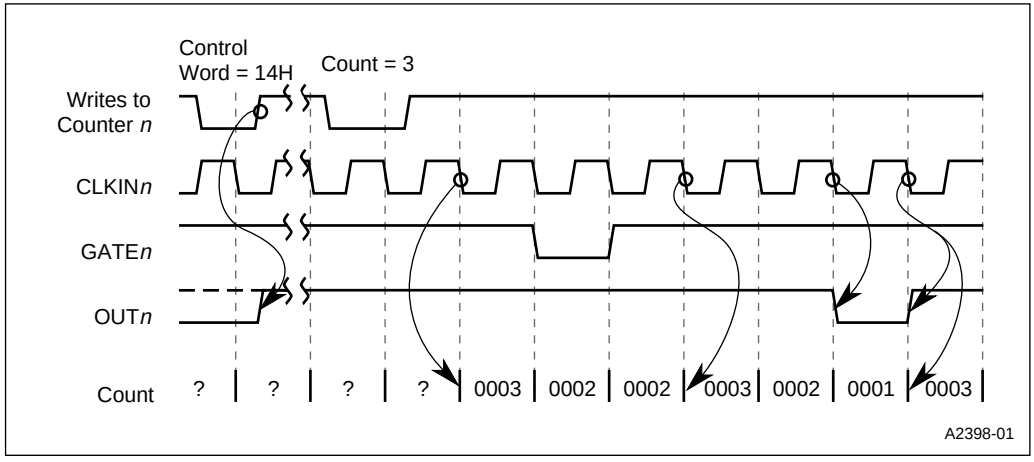


Figure 10-9. Mode 2 – Disabling the Count

Figure 10-10 shows writing a new count. The counter loads the new count after the counter reaches one. When the counter receives a gate-trigger after a new count was written to it, the counter loads the new count on the next CLKIN*n* pulse. This allows GATE*n* to synchronize the counters.

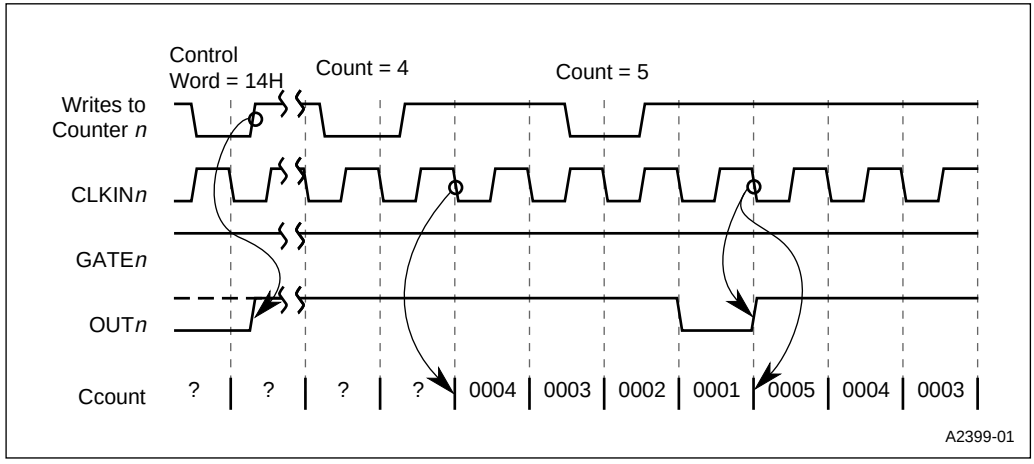


Figure 10-10. Mode 2 – Writing a New Count

10.2.4 Mode 3 – Square Wave

In this periodic mode, a counter's OUT*n* signal remains high for half a specified count, then goes low for the remainder of the count. A count of N results in a square wave with a period of N CLKIN*n* pulses. A high level on a counter's GATE*n* signal enables counting; a low level on a counter's GATE*n* signal disables counting. The output produced by a counter's OUT*n* signal depends on whether a count is odd or even. Mode 3's basic operation for even and odd counts is outlined below and shown in Figure 10-11 and Figure 10-12.

Even count basic operation:

1. After a control word write, OUT*n* is driven high.
2. The count is loaded on the CLKIN*n* pulse following one of these events:
   - A write to a control word followed by a write to count
   - A gate trigger
   - The counter reaches terminal count
3. On each succeeding CLKIN*n* pulse, the count is decremented by two.
4. After the count reaches terminal count, OUT*n* is driven low and the count is reloaded.
5. On each succeeding CLKIN*n* pulse, the count is decremented by two.
6. After the count reaches terminal count, OUT*n* is driven high and the count is reloaded.
7. The process is repeated from step 3.

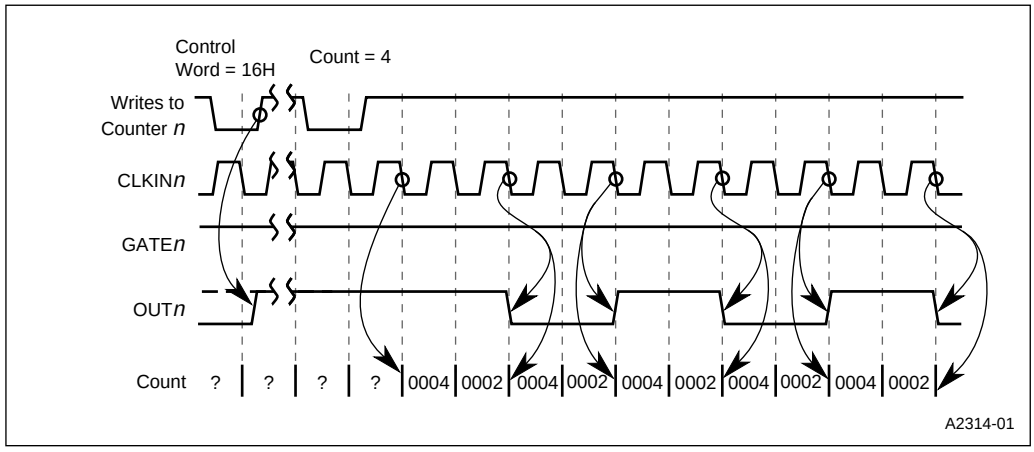


Figure 10-11. Mode 3 – Basic Operation (Even Count)

Odd count basic operation:

1. After a control word write, OUT*n* is driven high.
2. On the CLKIN*n* pulse following a gate-trigger or when the count rolls over, count minus one is loaded.
3. On each succeeding CLKIN*n* pulse, the count is decremented by two.
4. When the count rolls over, OUT*n* is driven low and the count minus one is loaded. (This causes OUT*n* to stay high for one more CLKIN*n* pulse than it stays low.)
5. On each succeeding CLKIN*n* pulse, the count is decremented by two.
6. After the count reaches terminal count, OUT*n* is driven high and the count minus one is loaded.
7. The process is repeated from step 3.

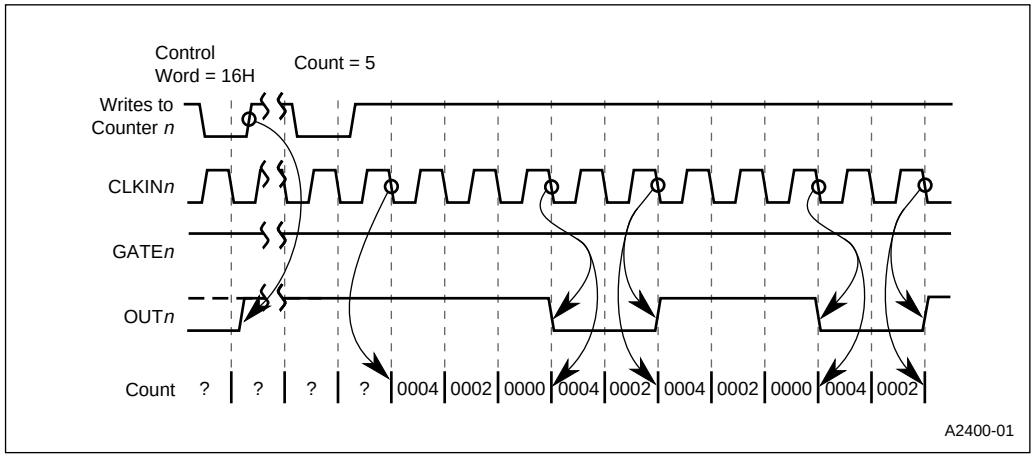


Figure 10-12. Mode 3 – Basic Operation (Odd Count)

NOTE

For an even count of N, OUT*n* remains high for N/2 counts and low for N/2 counts (provided GATE*n* remains high). For an odd count of N, OUT*n* remains high for (N + 1)/2 counts and low for (N – 1)/2 counts (provided GATE*n* remains high).

Figure 10-13 shows suspending the counting sequence. A low level on GATE*n* causes the counter to drive OUT*n* active (When OUT*n* is low, a falling edge on GATE*n* causes OUT*n* to be driven high immediately) and suspend counting. A rising edge on the GATE*n* causes the counter to be reloaded with the count. A high level on GATE*n* resumes counting.

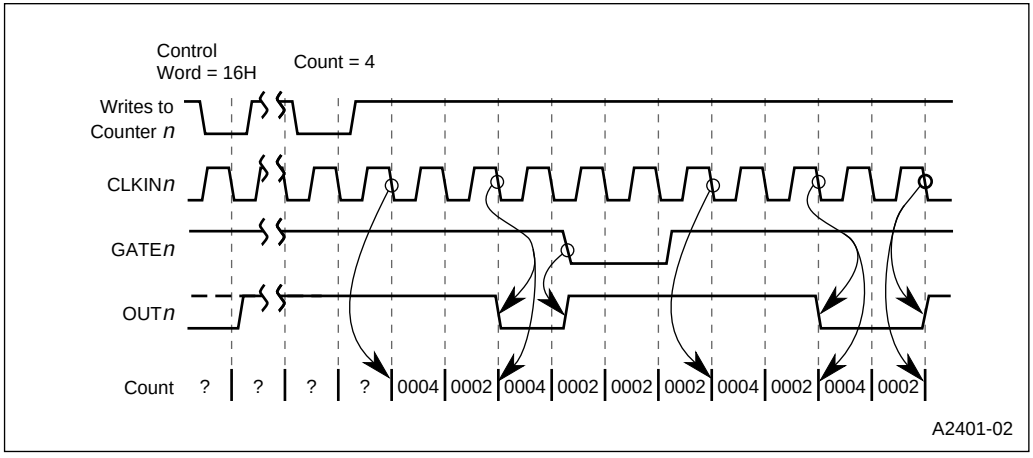


Figure 10-13. Mode 3 – Disabling the Count

Figure 10-14 and Figure 10-15 shows writing a new count. If the counter receives a gate-trigger after writing a new count but before the end of the current half-cycle, the count is loaded on the next CLKIN*n* pulse and counting continues from the new count (Figure 10-14). Otherwise, the new count is loaded at the end of the current half-cycle (Figure 10-15).

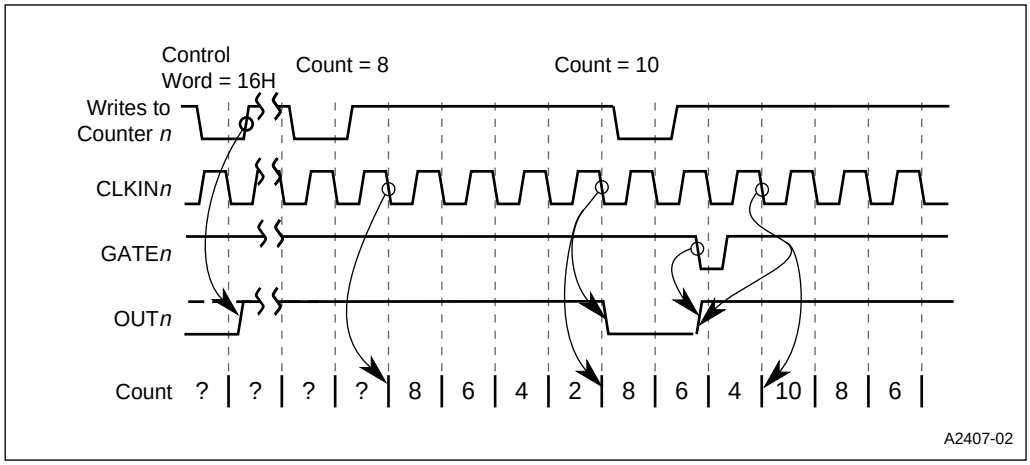


Figure 10-14. Mode 3 – Writing a New Count (With a Trigger)

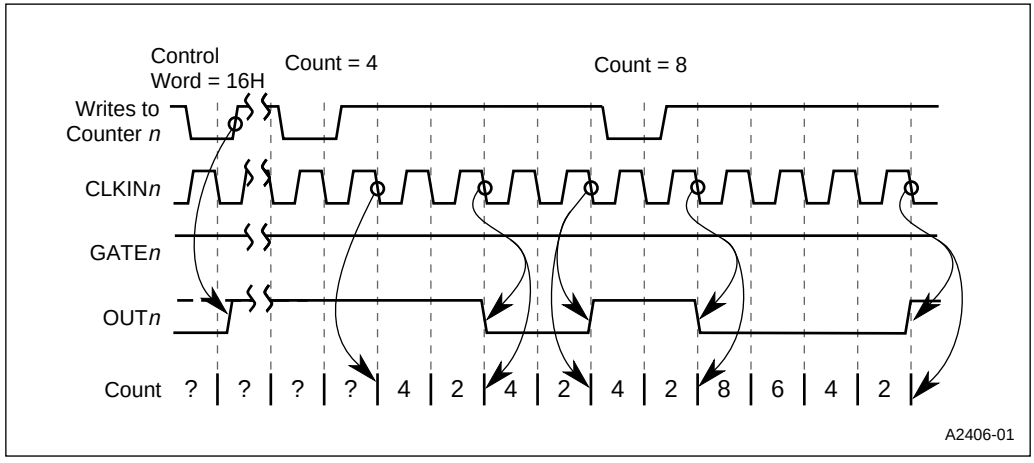


Figure 10-15. Mode 3 – Writing a New Count (Without a Trigger)

10.2.5 Mode 4 – Software-triggered Strobe

Initializing a counter for mode 4 drives the counter's OUT*n* signal high and initiates counting. A count is loaded on the CLKIN*n* pulse following a count write. When the counter reaches zero, OUT*n* strobes low for one clock pulse. The counter rolls over and continues counting, but does not strobe low when it reaches zero again. The counter strobes low only the first time it reaches zero after a count write. A high level on a counter's GATE*n* signal enables counting; a low level on a counter's GATE*n* signal disables counting.

Mode 4's basic operation is outlined below and shown in Figure 10-16.

1. After a control word write, OUT*n* is driven high.
2. On the CLKIN*n* pulse following the count write, the count is loaded.
3. On each succeeding CLKIN*n* pulse, the count is decremented.
4. When the count reaches zero, OUT*n* is driven low.
5. On the following CLKIN*n* pulse, OUT*n* is driven high.

NOTE

Writing a count of N causes OUT*n* to strobe low in N + 1 CLKIN*n* pulses, provided GATE*n* remains high. OUT*n* remains low for one CLKIN*n* pulse, then goes high.

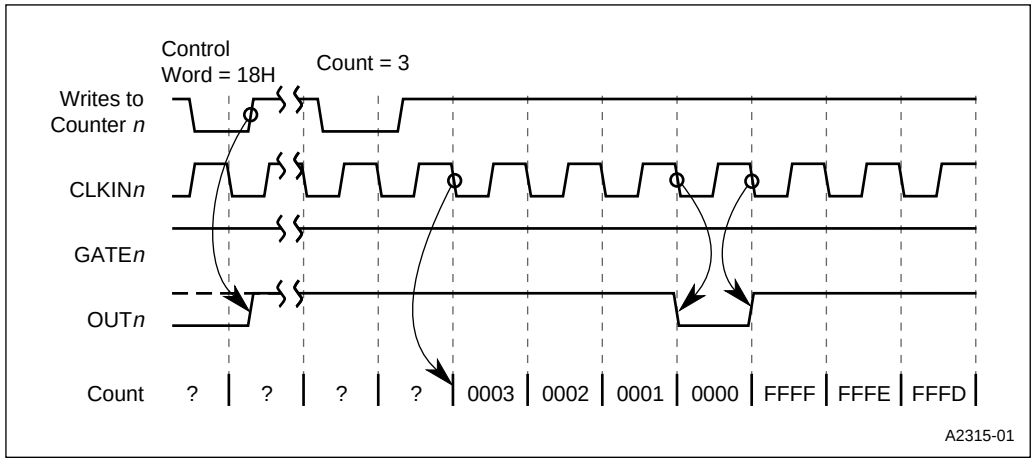


Figure 10-16. Mode 4 – Basic Operation

Figure 10-17 shows suspending the counting sequence. A low level on GATE*n* causes the counter to suspend counting (both the state of OUT*n* and the count remain unchanged). A high level on GATE*n* resumes counting.

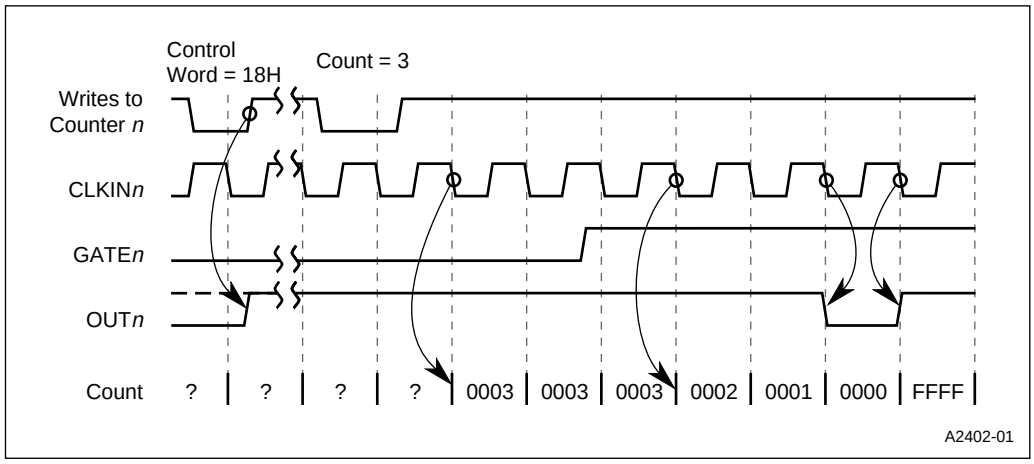


Figure 10-17. Mode 4 – Disabling the Count

Figure 10-18 shows writing a new count. On the CLKIN*n* pulse following the new count write, the counter loads the new count and counting continues from the new count.

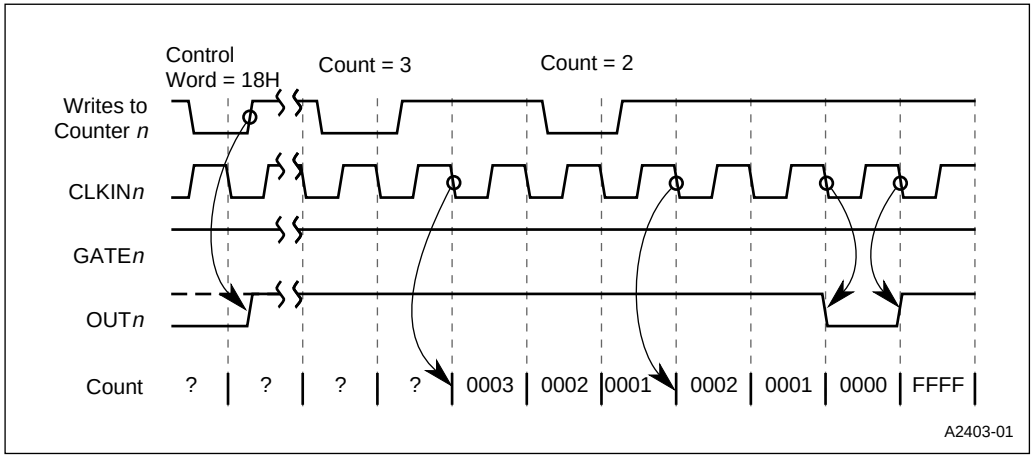


Figure 10-18. Mode 4 – Writing a New Count

10.2.6 Mode 5 – Hardware-triggered Strobe

Initializing a counter for mode 5 sets the counter's OUT*n* signal, starting the counting sequence. A gate-trigger loads the programmed count. When the counter reaches zero, OUT*n* strobes low for one clock pulse. The counter then rolls over and continues counting, but OUT*n* does not strobe low when the count reaches zero. The OUT*n* strobes low only the first time it reaches zero after a count is loaded.

Mode 5's basic operation is outlined below and shown in Figure 10-19.

1. After a control word write, OUT*n* is driven high.
2. On the CLKIN*n* pulse following a gate-trigger, the count is loaded.
3. On each succeeding CLKIN*n* pulse, the count is decremented.
4. When the count reaches zero, OUT*n* is driven low.
5. On the following CLKIN*n* pulse, OUT*n* is driven high.

NOTE

Writing a count of N causes OUT*n* to strobe low N + 1 CLKIN*n* pulses after the counter receives a gate-trigger. OUT*n* remains low for one CLKIN*n* pulse, then goes high.

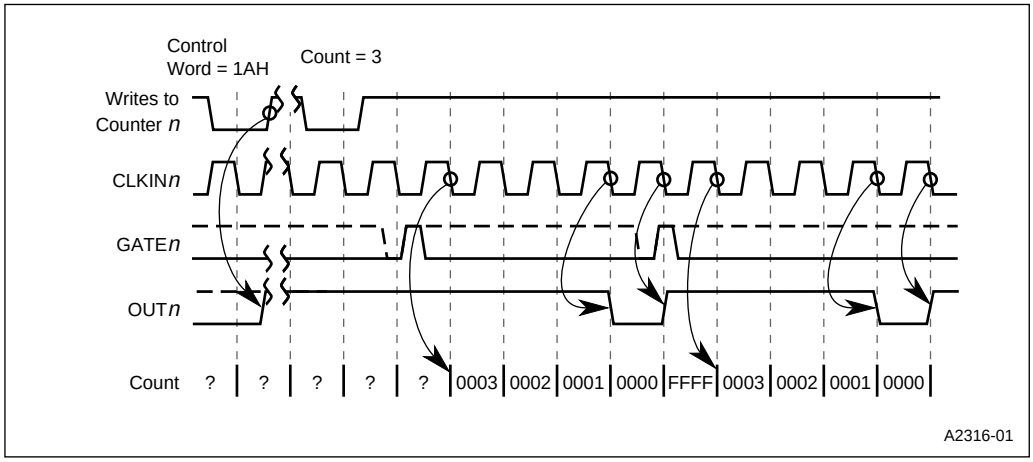


Figure 10-19. Mode 5 – Basic Operation

Figure 10-20 shows retriggering the strobe with a gate-trigger. On the CLKIN*n* pulse following the retrigger, the counter reloads the count. The control logic then decrements the count on each succeeding CLKIN*n* pulse. OUT*n* remains high until the count reaches zero, then strobes low for one CLKIN*n* pulse.

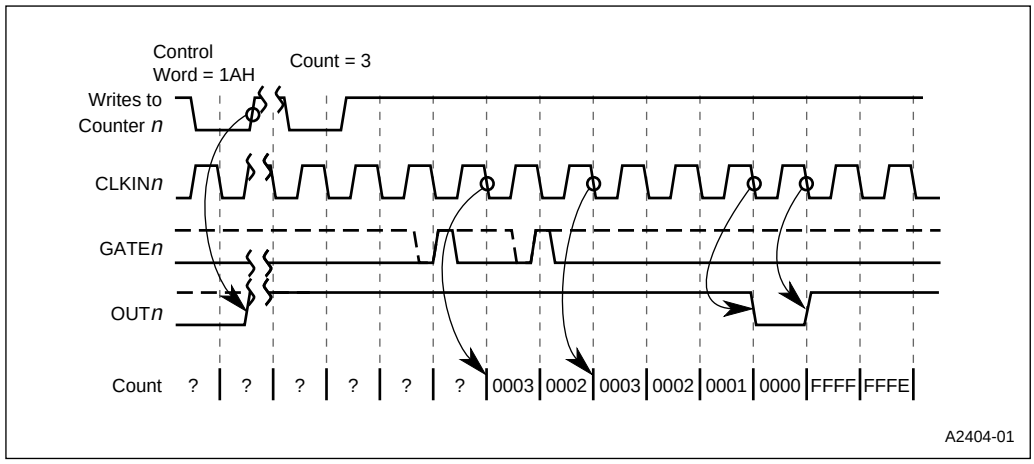


Figure 10-20. Mode 5 – Retriggering the Strobe

Figure 10-21 shows the writing of a new count value. The counter waits for a gate-trigger to load the new count; it does not affect the current sequence until the counter receives a trigger. On the CLKIN*n* pulse following the trigger, the control logic loads the new count. The control logic then decrements the count on each succeeding CLKIN*n* pulse. OUT*n* remains high until the count reaches zero, then strobes low for one CLKIN*n* pulse.

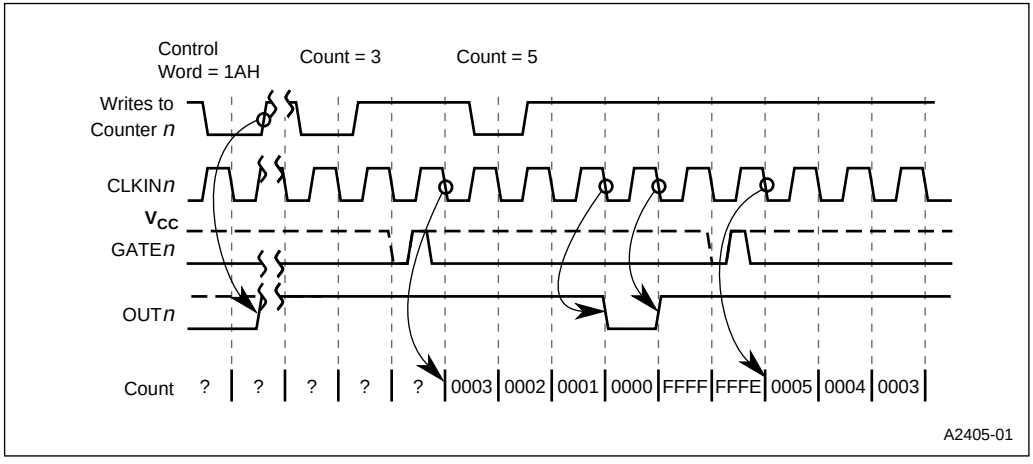


Figure 10-21. Mode 5 – Writing a New Count Value

10.3 REGISTER DEFINITIONS

The following sections describe how to configure a counter's input and output signals, initialize a counter for a specific operating mode and count format, write count values to a counter, and read a counter's status and count.

10.3.1 Configuring the Input and Output Signals

Each counter is driven by a clock pulse on its CLKIN*n* input. You can connect each counter's CLKIN*n* input to either its timer clock (TMRCLK*n*) pin or the prescaled clock (PSCLK) signal. The counters can handle up to 1/2 the processor clock (CLK2/4) input frequency but only up to a maximum of 8 MHz (CLKIN*n* frequency can never be more than 8 MHz). PSCLK is an internal signal that is a prescale value of the processor's internal clock. The frequency of PSCLK is programmable. See "Controlling the PSCLK Frequency" on page 8-7.

The GATEn signals of the counters can be controlled through hardware or software, as described in the next two sections.

10.3.1.1 Hardware Control of GATE*n*

You can connect each counter's GATEn signal to:

- Its timer gate (TMRGATEn) pin
- V_{CC}

Hardware (through a pin or V_{CC}) control of the GATE*n* requires that the SWGTEN bit in the TMRCFG register be reset.

10.3.1.2 Software Control of GATE*n*

You can also use the TMRCFG register to drive GATE*n* high or low through register bits. The SWGTEN and GT*n*CON bits are used to control the GATE*n* signal. If SWGTEN is set, then the value of the GT*n*CON bit causes the GATE*n* input of the counter to be driven to the corresponding voltage level.

Table 10-4. GATE*n* Connection Options

SWGTEN	GT*n*CON	GATE*n* connection
0	0	V_{CC}
0	1	TMRGATE*n*
1	0	0 (Gate*n* is OFF)
1	1	1 (Gate*n* is ON)

The timer configuration register (TMRCFG) enables the counter's CLKIN*n* signals and determines each counter's CLKIN*n* and GATE*n* signal connections or logical value (Figure 10-22).

Timer Configuration **Expanded Addr: F834H**
TMRCFG **ISA Addr: —**
(read/write) **Reset State: 00H**

7							0
TMRDIS	SWGTEN	GT2CON	CK2CON	GT1CON	CK1CON	GT0CON	CK0CON

Bit Number	Bit Mnemonic	Function
7	TMRDIS	Timer Disable: 0 = Enables the CLKIN*n* signals. 1 = Disables the CLKIN*n* signals.
6	SWGTEN	Software GATE*n* Enable 0 = Connects GATE*n* to either the V_{CC} pin or the TMRGATE*n* pin. 1 = Enables GT2CON, GT1CON, and GT0CON to control the connections to GATE2, GATE1 and GATE0 respectively.
5	GT2CON	Gate 2 Connection: **SWGTEN GT2CON** 0 0 Connects GATE2 to V_{CC}. 0 1 Connects GATE2 to the TMRGATE2 pin. 1 0 Turns GATE2 off. 1 1 Turns GATE2 on.
4	CK2CON	Clock 2 Connection: 0 = Connects CLKIN2 to the internal PSCLK signal. 1 = Connects CLKIN2 to the TMRCLK2 pin.
3	GT1CON	Gate 1 Connection: **SWGTEN GT1CON** 0 0 Connects GATE1 to V_{CC}. 0 1 Connects GATE1 to the TMRGATE1 pin. 1 0 Turns GATE1 off. 1 1 Turns GATE1 on.
2	CK1CON	Clock 1 Connection: 0 = Connects CLKIN1 to the internal PSCLK signal. 1 = Connects CLKIN1 to the TMRCLK1 pin.
1	GT0CON	Gate 0 Connection: **SWGTEN GT0CON** 0 0 Connects GATE0 to V_{CC}. 0 1 Connects GATE0 to the TMRGATE1 pin. 1 0 Turns GATE0 off. 1 1 Turns GATE0 on.
0	CK0CON	Clock 0 Connection: 0 = Connects CLKIN0 to the internal PSCLK signal. 1 = Connects CLKIN0 to the TMRCLK0 pin.

Figure 10-22. Timer Configuration Register (TMRCFG)

The peripheral pin selection registers (P3CFG and PINCFG) determine whether each counter's OUT*n* signal is connected to its TMROUT*n* pin. See Figure 10-1 for the TCU signal connections. For details on the P3CFG and PINCFG registers see Figure 10-23 and Figure 10-24. The counter output signals are automatically connected to the interrupt control unit. Counter 1's output signal (OUT1) is automatically connected to DMA channel 0, and counter 2's output signal (OUT2) is automatically connected to DMA channel 1.

Use P3CFG bits 0 and 1 to connect TMROUT0 and TMROUT1 to package pins.

Port 3 Configuration
P3CFG
(read/write)

Expanded Addr: **F824H**
ISA Addr: **—**
Reset State: **00H**

7							0
PM7	PM6	PM5	PM4	PM3	PM2	PM1	PM0

Bit Number	Bit Mnemonic	Function
7	PM7	Pin Mode: 0 = Selects P3.7 at the package pin. 1 = Selects COMCLK at the package pin.
6	PM6	Pin Mode: 0 = Selects P3.6 at the package pin. 1 = Selects PWRDOWN at the package pin.
5	PM5	Pin Mode: 0 = Selects P3.5 at the package pin. 1 = Connects master IR7 to the package pin (INT3).
4	PM4	Pin Mode: 0 = Selects P3.4 at the package pin. 1 = Connects master IR6 to the package pin (INT2).
3	PM3	Pin Mode: 0 = Selects P3.3 at the package pin. 1 = Connects master IR5 to the package pin (INT1).
2	PM2	Pin Mode: 0 = Selects P3.2 at the package pin. 1 = Connects master IR1 to the package pin (INT0).
1	PM1	Pin Mode: See Table 5-1 on page 5-8 for all the PM1 configuration options.
0	PM0	Pin Mode: See Table 5-1 on page 5-8 for all the PM0 configuration options.

Figure 10-23. Port 3 Configuration Register (P3CFG)

Use PINCFG bit 5 to connect TMROUT2, TMRCLK2, and TMRGATE2 to package pins.

Pin Configuration
PINCFG
(read/write)

Expanded Addr: **F826H**
ISA Addr: **—**
Reset State: **00H**

7							0
—	PM6	PM5	PM4	PM3	PM2	PM1	PM0

Bit Number	Bit Mnemonic	Function
7	—	Reserved. This bit is undefined; for compatibility with future devices, do not modify this bit.
6	PM6	Pin Mode: 0 = Selects CS6# at the package pin. 1 = Selects REFRESH# at the package pin.
5	PM5	Pin Mode: 0 = Selects the coprocessor signals, PEREQ, BUSY#, and ERROR#, at the package pins. 1 = Selects the timer control unit signals, TMROUT2, TMRCLK2, and TMRGATE2, at the package pins.
4	PM4	Pin Mode: 0 = Selects DACK0# at the package pin. 1 = Selects CS5# at the package pin.
3	PM3	Pin Mode: 0 = Selects EOP# at the package pin. 1 = Selects CTS1# at the package pin.
2	PM2	Pin Mode: 0 = Selects DACK1# at the package pin. 1 = Selects TXD1 at the package pin.
1	PM1	Pin Mode: 0 = Selects SRXCLK at the package pin. 1 = Selects DTR1# at the package pin.
0	PM0	Pin Mode: 0 = Selects SSIOTX at the package pin. 1 = Selects RTS1# at the package pin.

Figure 10-24. Pin Configuration Register (PINCFG)

10.3.2 Initializing the Counters

The timer control register (TMRCON) has three formats: *control word*, *counter-latch*, and *read-back*. When writing to TMRCON, certain bit settings determine which format is accessed.

Use the TMRCON's control word format (Figure 10-25) to specify a counter's count format and operating mode. Writing the control word forces OUT*n* to go to an initial state that depends on the selected operating mode.

Timer Control (Control Word Format)
TMRCON

Expanded Addr: **F043H**
ISA Addr: **0043H**
Reset State: **XXH**

7							0
SC1	SC0	RW1	RW0	M2	M1	M0	CNTFMT

Bit Number	Bit Mnemonic	Function
7–6	SC1:0	Select Counter: Use these bits to specify a particular counter. The selections you make for bits 5–0 define this counter's operation. 00 = counter 0 01 = counter 1 10 = counter 2 11 is not an option for TMRCON's control word format. Selecting 11 accesses TMRCON's read-back format, which is shown in Figure 10-29.
5–4	RW1:0	Read/Write Select: These bits select a read/write option for the counter specified by bits 7–6. 01 = read/write least-significant byte only 10 = read/write most-significant byte only 11 = read/write least-significant byte first, then most-significant byte 00 is not an option for TMRCON's control word format. Selecting 00 accesses TMRCON's counter-latch format, which is shown in Figure 10-27.
3–1	M2:0	Mode Select: These bits select an operating mode for the counter specified by bits 7–6. 000 = mode 0 001 = mode 1 X10 = mode 2 X11 = mode 3 100 = mode 4 101 = mode 5 X is a don't care.
0	CNTFMT	Count Format: This bit selects the count format for the counter specified by bits 7–6. 0 = binary (16 bits) 1 = binary coded decimal (4 decades)

Figure 10-25. Timer Control Register (TMRCON – Control Word Format)

10.3.3 Writing the Counters

Use the write format of a counter's Timer *n* register (TMR*n*) to specify a counter's count. The count must conform to the write selection specified in the control word (least-significant byte only, most-significant byte only, or least-significant byte followed by the most-significant byte). You can write a new count to a counter without affecting the counter's programmed operating mode. New counts must also conform to the specified write selection.

Timer *n* (Write Format)
TMR*n* (*n* = 0–2)

Expanded Addr: **F040H, F041H F042H**
ISA Addr: **0040H, 0041H 0042H**
Reset State: **XXH**

7							0
CV7	CV6	CV5	CV4	CV3	CV2	CV1	CV0

Bit Number	Bit Mnemonic	Function
7–0	CV7:0	Count Value: Write a count value for the counter to these bits. When writing the counter's count value, follow the write selection specified in the counter's control word.

Figure 10-26. Timer *n* Register (TMR*n* – Write Format)

Table 10-5 lists the minimum and maximum initial counts for each mode.

Table 10-5. Minimum and Maximum Initial Counts

Mode	Minimum Count	Maximum Count
0–1	1	0
2–3	2	0
4–5	1	0

NOTE: 0 is equivalent to 2^{16} for binary counting and 10^4 for BCD counting.

10.3.4 Reading the Counter

To read the counter you can perform a simple read operation or send a latch command to the counter. TMRCON contains two formats that allow you to send latch commands to individual counters: the counter-latch and read-back format. The counter-latch command latches the count of a specific counter. The read-back command latches the count and/or status of one or more specified counters.

10.3.4.1 Simple Read

To perform a simple read operation in modes 0, 2, 3 and 4, suspend the counter's operation (using the counter's GATE*n* signal), then read the counter's TMR*n* register. To read an accurate value, you must disable the counter so that the count is not in the process of changing when it is read. However, in modes 1 and 5, where the counter's operation can not be suspended, the counter can still be read. But since the counter is running, there is a minor inaccuracy in the read value.

10.3.4.2 Counter-latch Command

Use the counter-latch format of TMRCON (Figure 10-27) to latch the count of a specific counter. To issue a counter-latch command to a counter, write to the TMRCON register with bits 5-4 reset and SC1 and SC0 (bits 7-6) programmed appropriately. A counter continues to run even after the count is latched. The counter-latch command allows reading the count without disturbing the count in progress.

Timer Control (Counter-latch Format)
TMRCON

Expanded Addr: **F043H**
ISA Addr: **0043H**
Reset State: **XXH**

7							0
SC1	SC0	0	0	0	0	0	0

Bit Number	Bit Mnemonic	Function
7–6	SC1:0	Select Counter: These bits specify the counter that receives the counter-latch command. 00 = counter 0 01 = counter 1 10 = counter 2 11 is not an option for TMRCON's counter-latch format. Selecting 11 accesses TMRCON's read-back format, which is shown in Figure 10-29.
5–4	—	Write zeros to these bits to issue a counter-latch command to the counter specified by bits 7–6. 01, 10, and 11 are not valid options for TMRCON's counter-latch format.
3–0	—	Reserved; for compatibility with future devices, write zeros to these bits.

NOTE: Bits 5–0 serve another function when you select the read-back command (SC1:0 = 11). See Figure 10-29 for the read-back bit functions.

Figure 10-27. Timer Control Register (TMRCON – Counter-latch Format)

When a counter receives a counter-latch command, it latches the count. This count remains latched until you either read the count or reconfigure the counter. When you send multiple counter-latch commands without reading the counter, only the first counter-latch command latches the count value.

After issuing a counter-latch command, you can read the counter's TMR*n* register. When reading the counter's TMR*n* register you must follow the counter's programmed read selection (least-significant byte only, most-significant byte only, or least-significant byte followed by the most-significant byte). If the counter is programmed for two-byte counts, you must read two bytes. You need not read the two bytes consecutively; you may insert read, write, or programming operations between the byte reads.

You can interleave reads and writes of the same counter; for example, if the counter is programmed for the two-byte read/write selection, the following sequence is valid.

1. Read least-significant byte.
2. Write new least-significant byte.
3. Read most-significant byte.
4. Write new most-significant byte.

Timer *n* (Read Format)
TMR*n* (*n* = 0–2)

Expanded Addr: **F040H, F041H F042H**
ISA Addr: **0040H, 0041H 0042H**
Reset State: **XXH**

7							0
CV7	CV6	CV5	CV4	CV3	CV2	CV1	CV0

Bit Number	Bit Mnemonic	Function
7–0	CV7:0	Count Value: These bits contain the counter's count value. When reading the counter's count value, follow the read selection specified in the counter's control word.

Figure 10-28. Timer *n* Register (TMR*n* – Read Format)

10.3.4.3 Read-back Command

Use the read-back format of TMRCON (Figure 10-29) to latch the count and/or status of one or more counters. Latch a counter's status to check its programmed operating mode, count format, and read/write selection and to determine whether the latest count written to it has been loaded.

Timer Control (Read-back Format)
TMRCON

Expanded Addr: **F043H**
ISA Addr: **0043H**
Reset State: **XXH**

7							0
1	1	COUNT#	STAT#	CNT2	CNT1	CNT0	0

Bit Number	Bit Mnemonic	Function
7–6	—	Write ones to these bits to select the read-back command. 00, 01, and 10 are not valid options for TMRCON's read-back format.
5	COUNT#	Count Latch: 0 = Clearing this bit latches the count of each selected counter. Use bits 3–1 to select one or more of the counters. 1 = No effect
4	STAT#	Status Latch: 0 = Clearing this bit latches the status of each selected counter. Use bits 3–1 to select one or more of the counters. 1 = No effect
3	CNT2	Counter 2 Select: 0 = The actions specified by bits 5 and 4 do not affect counter 2. 1 = The actions specified by bits 5 and 4 affect counter 2.
2	CNT1	Counter 1 Select: 0 = The actions specified by bits 5 and 4 do not affect counter 1. 1 = The actions specified by bits 5 and 4 affect counter 1.
1	CNT0	Counter 0 Select: 0 = The actions specified by bits 5 and 4 do not affect counter 0. 1 = The actions specified by bits 5 and 4 affect counter 0.
0	—	Reserved; for compatibility with future devices, write zero to this bit.

Figure 10-29. Timer Control Register (TMRCON – Read-back Format)

The read-back command can latch the count and status of multiple counters. This single command is functionally equivalent to several counter-latch commands, one for each counter latched. Each counter's latched count and status is held until it is read or until you reconfigure the counter. A counter's latched count or status is automatically unlatched when read, but other counters' latched values remain latched until they are read.

After latching a counter's status and count with a read-back command, reading TMR*n* accesses its status format (Figure 10-30). Reading TMR*n* again accesses its read format. When both the count and status of a counter are latched, the first read of TMR*n* indicates the counter's status and the next one or two reads (depending on the counter's read selection) indicate the counter's count. Subsequent reads return unlatched count values. When only the count of a counter is latched, then the first one or two reads of TMR*n* return the counter's count. When the counter is programmed for the two-byte read selection, you must read two bytes.

Timer *n* (Status Format)
TMR*n* (*n* = 0–2)

Expanded Addr: **F040H, F041H F042H**
ISA Addr: **0040H, 0041H 0042H**
Reset State: **XXH**

7							0
OUTPUT	NULCNT	RW1	RW0	M2	M1	M0	CNTFMT

Bit Number	Bit Mnemonic	Function
7	OUTPUT	Output Status: This bit indicates the current state of the counter's output signal. 0 = OUT*n* is low 1 = OUT*n* is high
6	NULCNT	Count Status: This bit indicates whether the latest count written to the counter has been loaded. Some modes require a gate-trigger before the counter loads new count values. 0 = the latest count written to the counter has been loaded 1 = a count has been written to the counter but has not yet been loaded
5–4	RW1:0	Read/Write Select Status: These bits indicate the counter's programmed read/write selection. 00 = Never occurs 01 = read/write least-significant byte only 10 = read/write most-significant byte only 11 = read/write least-significant byte first, then most-significant byte
3–1	M2:0	Mode Status: These bits indicate the counter's programmed operating mode. 000 = mode 0 001 = mode 1 X10 = mode 2 X11 = mode 3 100 = mode 4 101 = mode 5 X is a don't care.
0	CNTFMT	Counter Format Status: This bit indicates the counter's programmed count format. 0 = binary (16 bits) 1 = binary coded decimal (4 decades)

Figure 10-30. Timer *n* Register (TMR*n* – Status Format)

When a counter receives multiple read-back commands, it ignores all but the first command; the count/status that the core reads is the count/status latched from the first read-back command (see Table 10-6).

Table 10-6. Results of Multiple Read-back Commands Without Reads

Command Sequence	Read-back Command	Command Result
1	Latch counter 0's count and status.	Counter 0's count and status latched.
2	Latch counter 1's status.	Counter 1's status latched.
3	Latch counter 2 and 1's status.	Counter 2's status latched; counter 1's status command ignored because command 2 already latched its status.
4	Latch counter 2's count.	Counter 2's count latched.
5	Latch counter 1's count and status.	Counter 1's count latched; counter 1's status command ignored because command 2 already latched its status.
6	Latch counter 0's count.	Counter 0's count command ignored because command 1 already latched its count.

10.4 PROGRAMMING CONSIDERATIONS

Consider the following when programming the TCU.

- The 16-bit counters are read and written a byte at a time. The control word format of TMRCON selects whether you read or write the least-significant byte only, most-significant byte only, or least-significant byte then most-significant byte (this is called the counter's read/write selection). You must read and write the counters according to their programmed read/write selections.
- When you program a counter for the two-byte read or write selection, you must read or write both bytes. If you're using more than one subroutine to read or write a counter, make sure that each subroutine reads or writes both bytes before transferring control.
- You can program the counters for either an internal or external clock source (to CLKIN*n*). The internal source is a prescaled value of the processor clock and therefore, is turned off in the processor's powerdown mode (processor clock is off). If an external clock source is used, it is not affected by the processor's powerdown mode, because the clock signal is provided by an off-chip source. "Controlling Power Management Modes" on page 8-8 describes the processor's powerdown and idle modes.

- With the readback command:
  - — If both the status and counter values are latched, the user can read the value of the Read/Write selection bits from the status register to know what bytes of the counter value are being latched in the TMR*n* register.
  - — If only the counter value is latched, you must know the Read/Write selection before the counter value can be read correctly.
- When a read-back command is issued to latch both the counter and the status of a timer, the TMR*n* register holds both of these values. The first read of TMR*n* accesses the status byte and the next one or two reads (depending on the R/W format) of TMR*n* access the timer's counter value.

10.4.1 Timer/Counter Unit Code Examples

The example code contains these software routines:

InitTimer	Initializes the specified timer's mode, counter value, inputs, and outputs
SetUp_ReadBack	Configures the specified timer(s) for a read-back command
CounterLatch	Latches the counter value of the specified timer
ReadCounter	Performs a simple read command on the specified timer's current counter value
TimerISR	Interrupt Service Routine for timer-generated interrupts

Code is also included that demonstrates how to change the timer's counter value and issue a read-back command. See Appendix C for included header files.

```
#include <conio.h>
#include "80386ex.h"
#include "ev386ex.h"


/*************************************************************************
   InitTimer:

   Description:
      This function initializes a timer's inputs, outputs, operating mode,
      and initial counter value.

    Parameters:
     Unit              Unit number of the timer. The processor supports 0, 1 or 2
      Mode             Defines Counter Mode
      Inputs           Specifies Input sources
      Output           Specifies Which Output to drive
      InitCount        Value to be loaded into count register
      Enable           Enable (1) or disable (0) Timer
```

```
   Returns:Error Codes
      E_INVALID_DEVICE     -- Unit number specifies a non-existing device
      E_OK                 -- Initialized OK, No error.

   Assumptions:
      REMAPCFG register has Expanded I/O space access enabled (ESE bit set);
      This function also initializes the Timer-Counter Unit to be in the
      Read/Write Format of least-significant byte first, then most-significant
      byte

   Syntax:
      int error;

      error = InitTimer    (TMR_2,
                            TMR_SQWAVE | TMR_CLK_BIN,
                            TMR_CLK_INTRN,
                            TMR_OUT_ENABLE,
                            0xFFFF,
                            TMR_ENABLE );

   Real/Protected Mode:
      No changes required.

*****************************************************************************/

int InitTimer(int Unit, WORD Mode, BYTE Inputs, BYTE Output, WORD InitCount,
              int Enable)
{
    BYTE               TmpByte;
    WORD               TmrCntPort;

    if(Unit > 2)
   return E_INVALID_DEVICE;

    TmrCntPort = 0xf040 + Unit;// Set depending on which timer

/* Set Pin configuration */
    if(Unit < 2)
    {
        TmpByte = _GetEXRegByte(P3CFG) | (Output << Unit);   // Bit 0 or 1
        _SetEXRegByte(P3CFG,TmpByte);
    }
    else
    {
    TmpByte = _GetEXRegByte(PINCFG) | (Output << 5);  // Bit 5
        _SetEXRegByte(PINCFG,TmpByte);
    }

/* Set Timer Config */
    TmpByte = _GetEXRegByte(TMRCFG);     // All Timers share this register,
                                         // Keep previous settings
```

```
    if(!Enable)
     TmpByte |= 0x80;                    // Set Timer Disable Bit

    TmpByte |= (Inputs << (Unit*2));     // Set CKnCON and GTnCON bits
    _SetEXRegByte(TMRCFG,TmpByte);

/* Set Timer Control Register */
    TmpByte = Unit << 6;          // Set counter select
    TmpByte |= (0x30 | Mode);     // Set R/W low then high byte and Mode bits
    _SetEXRegByte(TMRCON,TmpByte);

/* Set Initial Counter Value */
    TmpByte = HIBYTE(InitCount);
    _SetEXRegByte(TmrCntPort, LOBYTE(InitCount));
    _SetEXRegByte(TmrCntPort, TmpByte);

    return E_OK;

}/* InitTimer */


/*****************************************************************************

   SetUp_ReadBack:

   Description:
      This routine configures the Control Word for a Read Back Command.
      After calling this function, the latched status and counter values
      can be read from the TMRn registers. Example code of how to do
      this for Timer2 is included after this function.

   Parameters:
      Timer0          Cleared if Timer0's values are not to be latched
      Timer1          Cleared if Timer1's values are not to be latched
      Timer2          Cleared if Timer2's values are not to be latched
      GetStatus       Cleared if Status Byte is not to be latched
      GetCount        Cleared if Count Byte(s) is not to be latched


   Returns:
     None

   Assumptions:
      No assumptions have been made for this set-up function. However, if
      a user were to latch only the counter value, the configured R/W
      Format would have to be known. The setting of the R/W format can be
      read from the Status Byte if this value is latched. An example of
      this is included after the SetUp_ReadBack function.

   Syntax:

   #define ENABLE   1
```

```
   #define DISABLE  0

   SetUp_ReadBack(DISABLE, DISABLE, ENABLE, ENABLE, ENABLE);

   Real/Protected Mode:
      No changes required

*****************************************************************************/

void SetUp_ReadBack(  BYTE Timer0, BYTE Timer1, BYTE Timer2, BYTE GetStatus,
                      BYTE GetCount )
{
  BYTE rb_control = 0;

  rb_control |= 0xc0; // Set TMRCON to read-back command

  if (GetStatus != 0)
    rb_control &= 0xef;

  if (GetCount != 0)
    rb_control &= 0xdf;

  if (Timer0 != 0)
    rb_control |= 0x02;

  if (Timer1 != 0)
    rb_control |= 0x04;

  if (Timer2 != 0)
    rb_control |= 0x08;

  _SetEXRegByte(TMRCON, rb_control);

} /* SetUp_ReadBack */



/*****************************************************************************
   CounterLatch:

   Description:
      This function invokes a counter-latch command for the specified
      timer and returns the latched counter value.

   Parameters:
      Timer Unit number of timer whose counter value is to be latched
```

```
   Returns:
      Counter Value of specified timer

   Assumptions:
      This function assumes that the R/W format is configured to be LSB
      first, then MSB

    Syntax:

      WORD Counter_Value;

      Counter_Value = CounterLatch(TMR_1);

   Real/Protected Mode:
      No changes required

*************************************************************************/

WORD CounterLatch( BYTE Timer )
{

  BYTE control_word = 0;
  BYTE CounterL, CounterH;
  WORD Counter;

  control_word = Timer << 6;

  control_word &= 0xc0;

  _SetEXRegByte(TMRCON, control_word); //Select which counter

  switch (Timer) {

  case TMR_0:
    CounterL = _GetEXRegByte(TMR0);
    CounterH = _GetEXRegByte(TMR0);
    break;
  case TMR_1:
    CounterL = _GetEXRegByte(TMR1);
    CounterH = _GetEXRegByte(TMR1);
    break;
  case TMR_2:
    CounterL = _GetEXRegByte(TMR2);
    CounterH = _GetEXRegByte(TMR2);
    break;
  }

  Counter = (((WORD)CounterH << 8) + CounterL);

  return(Counter);

}/* CounterLatch */
```

```
/*****************************************************************************

   ReadCounter:

   Description:
      This function performs a simple read operation on the specified
      timer. However, because the counter value is not latched, the timer
      must be disabled, read, and then re-enabled.

   Parameters:
      Timer            Unit number of Timer whose count is being read

   Returns:
      Counter value that was read

   Assumptions:
      This function assumes that the R/W format is configured to be LSB
      first, then MSB

   Syntax:

   WORD Counter_Value;

   Counter_Value = ReadCounter(TMR0);

   Real/Protected Mode:
      No changes required

*****************************************************************************/

WORD ReadCounter(BYTE Timer)
{

  BYTE CountL, CountH;
  WORD Count = 0;

  DisableTimer();

  switch (Timer) {

  case TMR_0:
    CountL = _GetEXRegByte(TMR0);
    CountH = _GetEXRegByte(TMR0);
    break;
  case TMR_1:
    CountL = _GetEXRegByte(TMR1);
    CountH = _GetEXRegByte(TMR1);
    break;
```

```
  case TMR_2:
    CountL = _GetEXRegByte(TMR2);
    CountH = _GetEXRegByte(TMR2);
    break;
  }

  Count = (((WORD)CountH << 8) + CountL);

  EnableTimer();

  return(Count);

}/* ReadCounter */



/*****************************************************************************

   TimerISR:

   Description:
      Interrupt Service Routine for Timer-generated interrupts.

   Parameters:
      None

   Returns:
      None

   Assumptions:
      None

   Syntax:
      Not called by user.

   Real/Protected Mode:
      No changes required

*****************************************************************************/

void interrupt far TimerISR(void)
{
 /* Write message out to serial port as an example */

        SerialWriteStr(SIO_0, "In TimerISR\n");

   NonSpecificEOI();       // If this ISR services Timer1 or Timer2,
                           // an EOI is also needed for the Slave 8259

}/* TimerISR */
```

```
/*****************************************************************************

   Example of how to write a new initial counter value to a timer
   This value can be rewritten at any time without affecting the
   Counter's programmed mode.

   Before writing an initial count value, the Control Word must be
   configured for the proper R/W and Count formats.

   -->This example assumes that Timer1 is in the R/W format of LSB first,
   then MSB, and that the Count format is binary.

   _SetEXRegByte(TMR1, InitialCountL);
   _SetEXRegByte(TMR1, InitialCountH);

*****************************************************************************/



/*****************************************************************************/

  ***Example of how to issue a Read Back command for Timer2, latching
     both the status and the counter.

 BYTE Status, CountL, CountH, RWmode;
 WORD Count;

 SetUp_ReadBack(0, 0, 1, 1, 1);   //Configure Read Back command for timer2,
                                    latching both status and count
 Status = GetEXRegByte(TMR2);

 RWmode = Status & 0x30;  //Mask off bits that correspond to the Read/Write Mode

 switch (RWmode) {  //Read Counter Value according to configured R/W format

 case 0x10:         //Read/Write least significant byte only
   Count = _GetEXRegByte(TMR2);
   break;
 case 0x20:  //Read/Write most significant byte only
   CountH = _GetEXRegByte(TMR2);
   Count = (WORD)CountH << 8;
   break;
 case 0x30:  //Read/Write LSB first, then MSB
   CountL = _GetEXRegByte(TMR2);
   CountH = _GetEXRegByte(TMR2);
   Count = (((WORD)CountH << 8) + CountL);
   break;
 }

*****************************************************************************/
```

11

ASYNCHRONOUS SERIAL I/O UNIT

CHAPTER 11
ASYNCHRONOUS SERIAL I/O UNIT

The asynchronous serial I/O (SIO) unit provides a means for the system to communicate with external peripheral devices and modems. The SIO unit performs serial-to-parallel conversions on data characters received from a peripheral device or modem and parallel-to-serial conversions on data characters received from the CPU. The SIO unit consists of two independent SIO channels, each of which is compatible with National Semiconductor's NS16C450.

This chapter is organized as follows:

11.1 OVERVIEW

Each SIO channel contains a baud-rate generator, transmitter, receiver, and modem control unit. These are shown in Figure 11-1 for SIO Unit 1 (see Figure 5-8 on page 5-15 for the SIO Unit 0 configuration). The baud-rate generator can be clocked by either the internal serial clock (SERCLK) signal or the COMCLK pin. The transmitter and receiver contain shift registers and buffers. Data to be transmitted is written to the transmit buffer. The buffer's contents are transferred to the transmit shift register and shifted out via the transmit data pin (TXD*n*). Data received is shifted in via the receive data pin (RXD*n*). When a data byte is received, the contents of the receive shift register are transferred to the receive buffer. The modem control logic provides interfacing for the handshaking signals between an SIO channel and a modem or data set.

In addition to the transmit and receive channels, each SIO can generate an interrupt or a request to the DMA unit, or both. An interrupt can be generated when an error has occurred in the receive channel, when the transmit channel is ready to transmit another character, when the receive channel is full, or when a change in any of the modem control signals has occurred. A DMA request may be issued any time a channel's receive buffer is full or its transmit buffer is empty. This allows the SIO to run at higher speeds for more efficient processing of serial data.

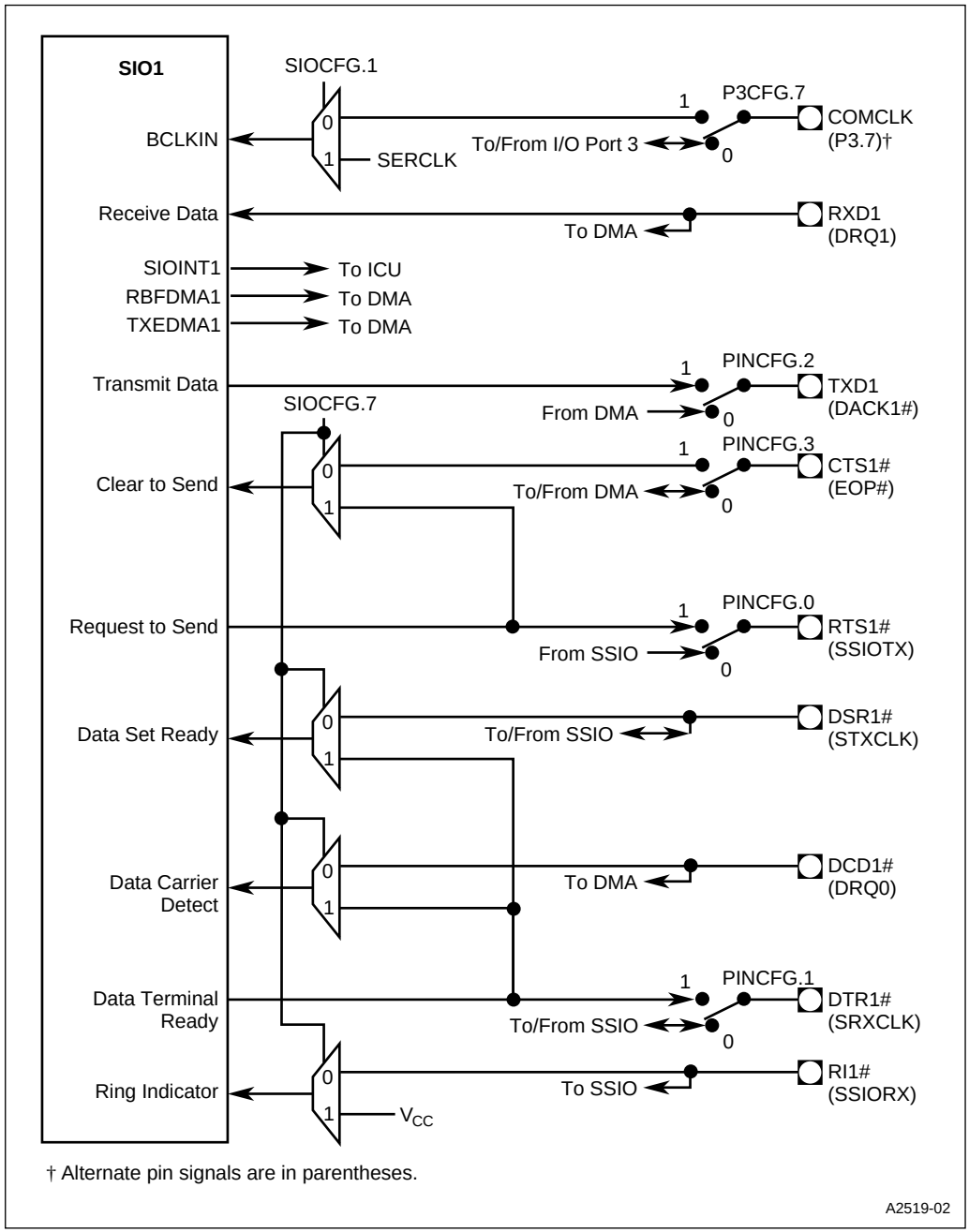


Figure 11-1. Serial I/O Unit 1 Configuration

11.1.1 SIO Signals

Table 11-1 lists the SIO*n* signals.

Table 11-1. SIO Signals

Signal	Device Pin or Internal Signal	Description
Baud-rate Generator Clock Source	Internal signal Device pin (input)	SERCLK: This internal signal is the processor's input clock, CLK2, divided by four. COMCLK: An external source connected to this pin can clock the SIO*n* baud-rate generator.
TXD*n*	Device pin (output)	Transmit Data: The transmitter uses this pin to shift serial data out. Data is transmitted least-significant bit first.
RXD*n*	Device pin (input)	Receive Data: The receiver uses this pin to shift serial data in. Data is received least-significant bit first.
CTS*n*#	Device pin (input)	Clear to Send: Indicates that the modem or data set is ready to exchange data with the SIO*n* channel.
DSR*n*#	Device pin (input)	Data Set Ready: Indicates that the modem or data set is ready to establish the communications link with the SIO*n* channel.
DCD*n*#	Device pin (input)	Data Carrier Detect: Indicates that the modem or data set has detected the data carrier.
RI*n*#	Device pin (input)	Ring Indicator: Indicates that the modem or data set has detected a telephone ringing signal.
RTS*n*#	Device pin (output)	Request to Send: Indicates to the modem or data set that the SIO*n* channel is ready to exchange data.
DTR*n*#	Device pin (output)	Data Terminal Ready: Indicates to the modem or data set that the SIO*n* channel is ready to establish a communications link.
SIOINT*n*	Internal Signal	SIOINT: This signal is connected to the interrupt control unit and is asserted (HIGH) when any one of the following interrupt types has an active condition and is enabled via the IER register: Receiver Error flag, Received Data Available, Transmitter Holding Register Empty, or Modem Status. The SIOINT signal is deasserted (LOW) upon the appropriate interrupt service or reset operation.

Table 11-1. SIO Signals

Signal	Device Pin or Internal Signal	Description
TXEDMA*n*	Internal Signal	Transmitter Empty: When this signal is high, the Transmitter Holding Register is empty (transmit data has been loaded into the Transmit Shift Register).
RBFDMA*n*	Internal Signal	Receiver Full: When high, this signal indicates that the Receive Buffer has been loaded with data from the Receive Shift Register.

11.2 SIO OPERATION

The following sections describe the operation of the baud-rate generator, transmitter, and receiver and discusses the modem control logic, SIO diagnostic mode, and SIO interrupt sources.

11.2.1 Baud-rate Generator

Each SIO channel's baud-rate generator provides the clocking source for the channel's transmitter and receiver. The baud-rate generator can divide its input (BCLKIN) by any divisor from 1 to (2^{16}–1). The output frequency is 16 times the desired bit time. The transmitter shifts data out on the rising edge of BCLKIN. The receiver samples input data in the middle of a bit time.

The internal serial clock (SERCLK) signal or the COMCLK pin can be connected to the baud-rate generator's BCLKIN signal (Figure 11-2). The SIO configuration register (SIOCFG) selects one of these sources.

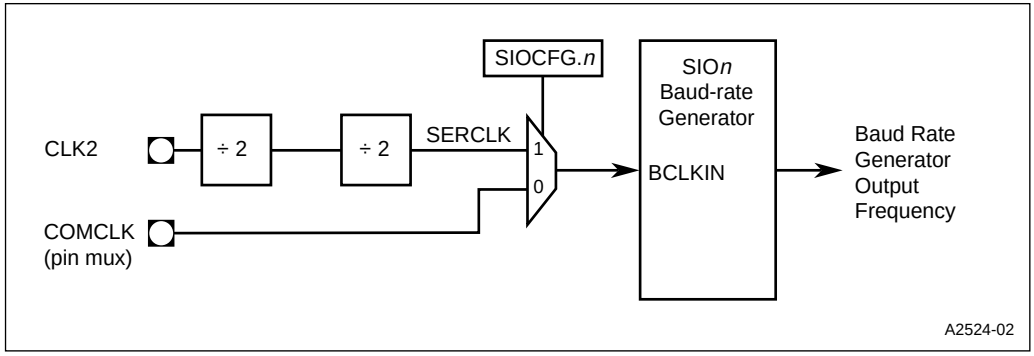


Figure 11-2. SIO*n* Baud-rate Generator Clock Sources

SERCLK provides a baud-rate input frequency (BCLKIN) of CLK2/4. The COMCLK pin allows an external source with a maximum frequency of CLK2/4 to provide the baud-rate generator input frequency.

The baud-rate generator's output frequency is determined by BCLKIN and a divisor as follows.

$$\text{baud-rate generator output frequency} = \frac{\text{BCLKIN frequency}}{\text{divisor}},$$

$$\text{bit rate} = \frac{\text{baud rate generator output frequency}}{16}$$

The minimum divisor value is 1, giving a maximum baud rate of BCLKIN. The maximum divisor value is 0FFFFH (65535), giving a minimum of BCLKIN/65535. For example, the maximum and minimum bit-rate frequencies using SERCLK with a 25 MHz device (CLK2 = 50 MHz) or COMCLK with a 12.5 MHz input are shown in Table 11-2. Table 11-3 shows the divisor values required for common baud rates.

Table 11-2. Maximum and Minimum Output Bit Rates

Input Clock (BCLKIN)	Divisor	Output Bit Rate
12.5 MHz	0001H	781.25 KHz (max)
12.5 MHz	0FFFFH	11.921 Hz (min)

Table 11-3. Divisor Values for Common Bit Rates

Divisor	Input Clock (BCLKIN)	Output Bit Rate	% Error
1AEH	16.5 MHz (processor clock = 33 MHz)	2400 b/s	– 0.07
6BH	16.5 MHz (processor clock = 33 MHz)	9600 b/s	+ 0.39
48H	16.5 MHz (processor clock = 33 MHz)	14.4 Kb/s	– 0.54
145H	12.5 MHz (processor clock = 25 MHz)	2400 b/s	+ 0.15
51H	12.5 MHz (processor clock = 25 MHz)	9600 b/s	+ 0.47
36H	12.5 MHz (processor clock = 25 MHz)	14.4 Kb/s	+ 0.46
104H	10 MHz (processor clock = 20 MHz)	2400 b/s	+ 0.15
41H	10 MHz (processor clock = 20 MHz)	9600 b/s	+ 0.16
2BH	10 MHz (processor clock = 20 MHz)	14.4 Kb/s	+ 0.94
0D0H	8 MHz (processor clock = 16 MHz)	2400 b/s	+ 0.16
34H	8 MHz (processor clock = 16 MHz)	9600 b/s	+ 0.16
23H	8 MHz (processor clock = 16 MHz)	14.4 Kb/s	– 0.79

11.2.2 SIO*n* Transmitter

The data frame for transmissions is programmable. It consists of a start bit, 5 to 8 data characters, an optional parity bit, and 1 to 2 stop bits. The transmitter can produce even, odd, forced, or no parity. The transmitter can also produce break conditions. A break condition forces the serial output (TXD*n*) to the spacing (logic 0) state for longer than a transmission time (the time of the start bit + data bits + parity bit + stop bits). On the receiving end, a break condition sets an error flag.

Forced parity ("sticky bit") allows the SIO to be used in multiprocessor communications. When using forced parity the serial port uses the parity bit to distinguish between address and data bytes.

Forced parity is enabled in the SIO by setting the PEN and SP bits in the serial line control register (Figure 11-15). When enabled for forced parity, the bit that is transmitted in the parity bit location is the complement of the EPS bit (also in the serial line control register). In the receiver, if PEN and SP are 1, the receiver compares the bit that is received in the parity bit location with the complement of the EPS bit. If the values being compared are not equal, the receiver sets the Parity Error bit in LSR and causes an error interrupt if line status interrupts are enabled.

For example, if forced parity is enabled and EPS is 0, the receiver expects the bit received at the parity bit location to be 1. If it is not, the parity error bit is set. By forcing the bit value at the parity bit location, rather than calculating a parity value, a system with a master transmitter and multiple receivers can identify some transmitted characters as receiver addresses and the rest of the characters as data. If PEN = 0, the SP bit is ignored.

Each SIO channel transmitter contains a transmit shift register, a transmit buffer, and a transmit data pin (TXD*n*). Data to be transmitted is written to the transmit buffer. The transmitter then transfers the data to the transmit shift register. The transmitter shifts the data along with asynchronous communication bits (start, stop, and parity) out via the TXD*n* pin. The TXD0 and TXD1 pins are multiplexed with other functions. The pin configuration registers (PINCFG and P2CFG) determine whether a TXD*n* signal or an alternate function is connected to the package pin.

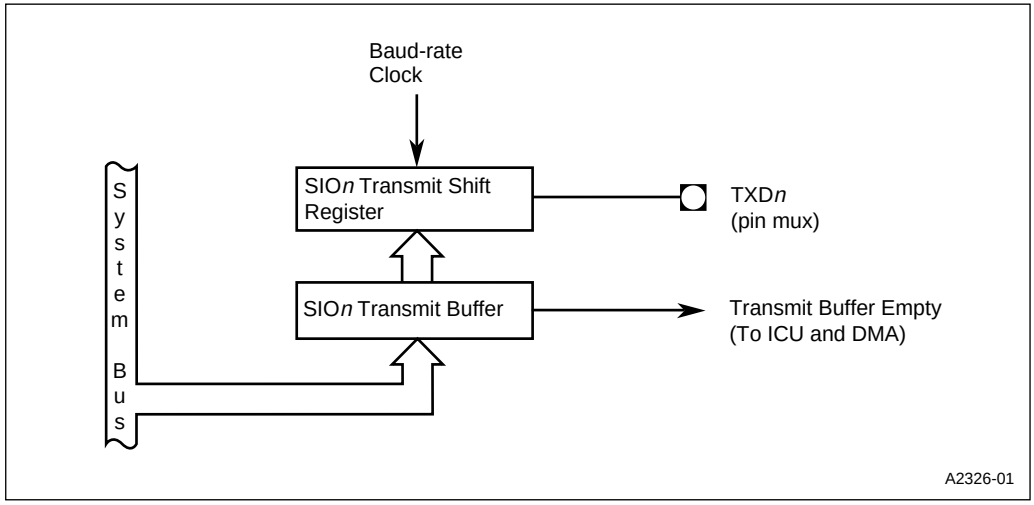


Figure 11-3. SIO*n* Transmitter

The transmitter contains a transmitter empty (TE) flag and a transmit buffer empty (TBE) flag. At reset, TBE and TE are set, indicating that the transmit buffer and shift register are empty. Writing data to the transmit buffer clears TBE and TE. When the transmitter transfers data from the buffer to the shift register, TBE is set. Unless new data is written to the transmit buffer, TE is set when the transmitter finishes shifting out the shift register's contents.

The transmitter's transmit buffer empty signal can be connected to the interrupt control and DMA units. Figure 11-4 shows the process for transmitting data.

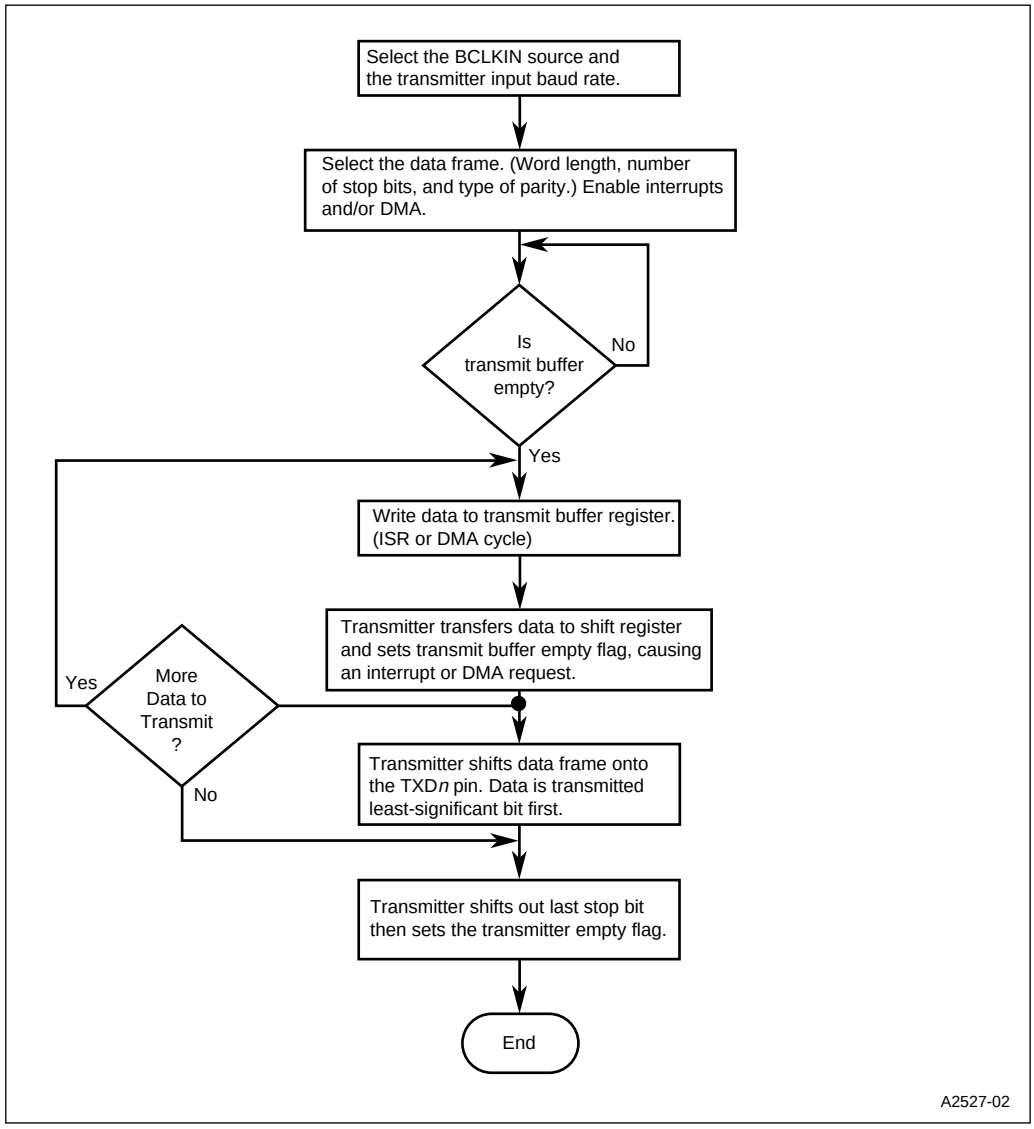


Figure 11-4. SIO*n* Data Transmission Process Flow

11.2.3 SIO*n* Receiver

The data frame for receptions is programmable, and is identical to the data frame for transmissions. It consists of a start bit, 5 to 8 data characters, an optional parity bit, and 1 to 2 stop bits. The receiver can be programmed for even, odd, forced, or no parity. When the receiver detects a parity condition other than what it was programmed for, it sets a parity error flag. In addition to detecting parity errors, the receiver can detect break conditions, framing errors, and overrun errors.

- A break condition indicates that the received data input is held in the spacing (logic 0) state for longer than a data transmission time (the time of the start bit + data bits + parity + stop bits).
- A framing error indicates that the received character did not have a valid stop bit.
- An overrun error indicates that new data overwrote old data before the old data was read.

Each SIO channel receiver contains a receive shift register, a receive buffer, and a receive data pin (RXD*n*). Data received is shifted into the receive shift register via the RXD*n* pin. Once a data byte has been received, the receiver strips off the asynchronous communication bits (start, stop, and parity) and transfers the contents of its shift register to the receive buffer.

The RXD0 pin is multiplexed with another function. The pin configuration register (P2CFG) determines whether the RXD0 signal or the alternate function is connected to the package pin.

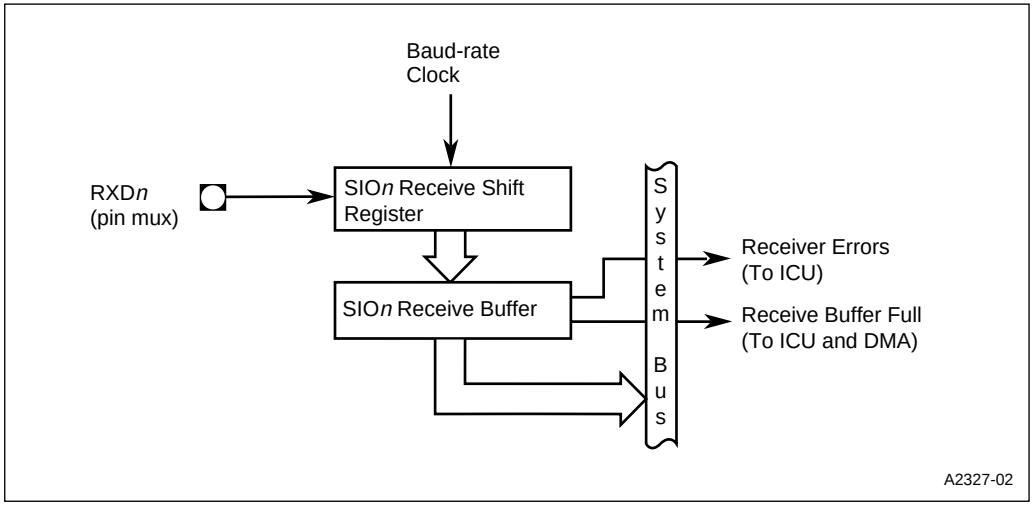


Figure 11-5. SIO*n* Receiver

The receiver contains a receive buffer full (RBF) flag and flags for each of the error conditions described above. At reset, RBF and each of the error flags (PE, FE, OE, and BI) are clear, indicating that the receive buffer is empty, and no error has occurred. When a character is received the receiver checks for parity, framing or break errors, and sets the appropriate bits, if necessary. It then shifts the data into the receive buffer, sets the OE bit if an overrun occurs, and then sets the RBF flag. Reading the receive buffer clears the RBF flag and any error flags in the LSR that may have been set, except for the OE bit. The OE bit is cleared by reading the LSR.

High speed serial transfers may require using the DMA to eliminate interrupt latency time in servicing the SIO. Since the SIO unit clears the error bits in the line status register each time the receive buffer register is read, it would be impossible to detect an error using DMA. Because of this, two RBF signals are used:

- One RBF signal (RBFDMA) goes directly to the DMA unit. This signal is blocked when an error (parity, overrun, break, or framing) occurs. This prevents a DMA request from being generated by the RBF.
- The other RBF signal (RBFINT) goes directly to the interrupt priority logic and out on SIOINT if enabled in the Interrupt Enable Register.

When the Interrupt Enable Register is programmed to generate an SIOINT on receiver errors, the error can be serviced as part of the interrupt handler.

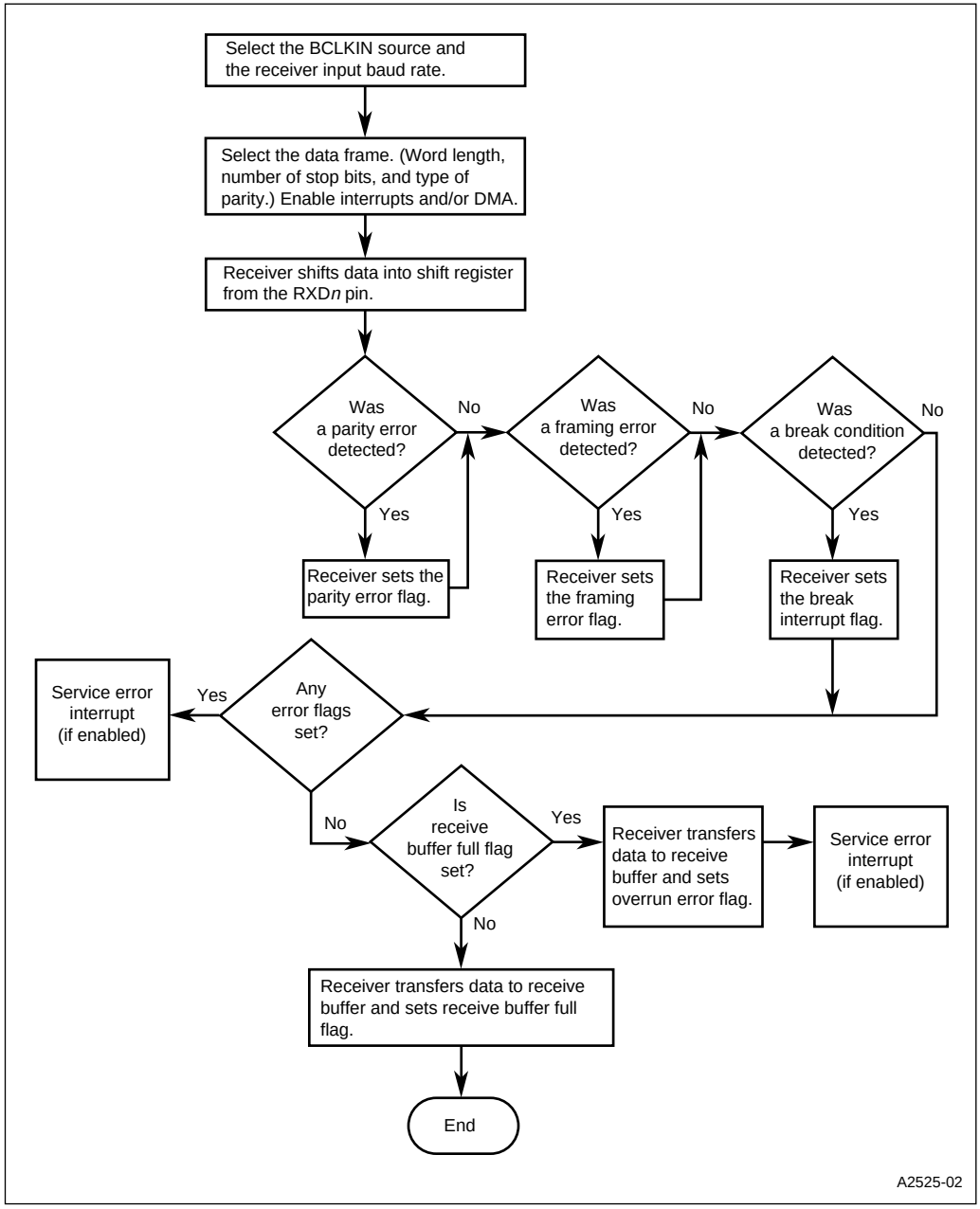


Figure 11-6. SIO*n* Data Reception Process Flow

11.2.4 Modem Control

The modem control logic provides interfacing for four input signals and two output signals used for handshaking and status indication between the SIO*n* and a modem or data set. An external modem or data set uses the input signals to inform the SIO*n* when:

- It is ready to establish a communications link (DSR*n*#)
- It has detected a data carrier signal (DCD*n*#)
- It has detected a telephone ringing signal (RI*n*#)
- It is ready to exchange data (CTS*n*#)

The SIO*n* uses its output signals to inform the modem or data set when it is ready to establish a communication link (DTR*n*#), and when it is ready to exchange data (RTS*n*#).

The modem output signals can be internally connected to the modem input signals using the SIO configuration register. In this case, the modem input signals are disconnected from the pins, RTS*n*# is connected to CTS*n*#, DTR*n*# is connected to both DSR*n*# and DCD*n*#, and V_{CC} is connected RI*n*#.

The SIO contains status flags that indicate the current state of the modem control input signals and status flags that indicate whether any of the modem control input signals have changed state.

11.2.5 Diagnostic Mode

The SIO channels provide a diagnostic mode to aid in isolating faults in the communications link. In this mode, data that is transmitted is immediately received. This feature allows the processor to verify the internal transmit and receive data paths of an SIO*n* channel.

The diagnostic mode connections are as follows:

- The transmitter serial output (TXD*n*) is set to a logic 1 state.
- The receiver serial input (RXD*n*) is disconnected from the pin.
- The transmit shift register output is "looped back" into the receive shift register.
- The four modem control inputs (CTS*n*#, DSR*n*#, DCD*n*#, and RI*n*#) are disconnected from the pins and controlled by modem control register bits.
- The modem control output pins (RTS*n*#, DTR*n*#) are forced to their inactive states.

11.2.6 SIO Interrupt and DMA Sources

11.2.6.1 SIO Interrupt Sources

Each SIO channel has four status signals: receiver line status, receiver buffer full, transmit buffer empty, and modem status. An overrun error, parity error, framing error, or break condition can activate the receiver line status signal. When the receiver transfers data from its shift register to its buffer, it activates the receive buffer full signal. When the transmitter transfers data from its transmit buffer to its transmit shift register, it activates the transmit buffer empty signal. A change on any of the modem control input signals activates the modem status signal. When the modem signals are connected internally either through the configuration register or the diagnostic mode, changes of state still activate the modem status signal. For these cases, however, the signal values are controlled by register bits rather than by external input signals.

Each of the four status signals can be used as an interrupt request source for the SIOINT*n* signal. The Interrupt Enable register (IER) is used to enable any or all of the status signals as interrupt sources. When an SIOINT*n* occurs the IP# bit (bit 0) of the Interrupt ID register (IIR*n*) is cleared and the interrupt handler must determine which of the status signals caused the interrupt by reading bits 1 and 2 of the IIR*n* register (Table 11-4). When more than one status signal is enabled as an interrupt source and two or more are active at the same time then the source of the interrupt is based on a fixed priority scheme (Table 11-4).

Table 11-4. Status Signal Priorities and Sources

Interrupt ID Register			Priority	Status Signal	Activated By
Bit 2	Bit 1	Bit 0			
1	1	0	1 (Highest)	Receiver Line Status	overrun error, parity error, framing error, or break condition
1	0	0	2	Receive Buffer Full	the receiver transferring data from its shift register to its buffer
0	1	0	3	Transmit Buffer Empty	the transmitter transmitting data from its transmit buffer to its transmit shift register
0	0	0	4 (Lowest)	Modem Status	a change on any of the modem control input signals (CTS*n*#, DCD*n*#, DSR*n*#, and RI*n*#

11.2.6.2 SIO DMA sources

The transmit and receive channel on each SIO is supported by both DMA channels. The receiver buffer full and transmit buffer empty signals of the line status register are brought out as RBFDMA*n*, and TXEDMA*n*. The TXEDMA*n* signal is connected directly to the multiplexers controlling the source of DREQ*n* for each of the two DMA channels. The RBFDMA*n* signal is also connected to the DREQ*n* muxes, but it is qualified by the LSR error conditions so that the DMA request is blocked if an error has occurred in the reception of a character. This prevents the DMA from transferring a character from the SIO with an error.

11.2.7 External UART Support

Many PC compatible applications may need to support COM3 and COM4 serial ports. Since the integrated serial ports are mapped to I/O addresses that support only COM1 and COM2, an interface to support an external serial I/O unit has been included. The master ICU interrupt inputs IR3 and IR4 may be brought out to package pins as INT8 (muxed with P3.1/TMROUT1) and INT9 (muxed with P3.0/TMROUT0), respectively.

In order to select between the internal SIO units and the external SIO units, use the OUT2 bit in the modem control register (MCR). In normal user mode (with no loopback) clear the OUT2 bit to enable external SIO support, and set OUT2 to enable internal SIO support. Doing this, combined with the correct settings of the P3CFG.1:0 and INTCFG.6:5 bits connect the INT8 and INT9 pins to IR3 and IR4 of the master ICU, respectively. Note that the reset state of P3CFG and INTCFG enables SIOINT and disconnects OUT2 gating. See Chapter 5, "DEVICE CONFIGURATION" (Tables 5-1 and 5-2) for more details on how to select this option.

11.3 REGISTER DEFINITIONS

Table 11-5 lists the registers associated with the SIO unit and the following sections contain bit descriptions for each register.

Table 11-5. SIO Registers (Sheet 1 of 2)

Register	Expanded Address	PC/AT* Address	Function
PINCFG (read/write)	0F826H	—	Pin Configuration: Connects the SIO1 transmit data (TXD1), data terminal ready (DTR1#), and request to send (RTS1#) signals to package pins.
P1CFG (read/write)	0F820H	—	Port 1 Configuration: Connects the SIO0 ring indicator (RI0#), data set ready (DSR0#), data terminal ready (DTR0#), request to send (RTS0#), and data carrier detected (DCD0#) signals to package pins.
P2CFG (read/write)	0F822H	—	Port 2 Configuration: Connects the SIO0 clear to send (CTS0#), transmit data (TXD0), and receive data (RXD0) signals to package pins.
P3CFG (read/write)	0F824H	—	Port 3 Configuration: Connects COMCLK to the package pin.
SIOCFG (read/write)	0F836H	—	SIO and SSIO Configuration: Connects the SIO*n* modem input signals internally or to package pins and connects either the internal SERCLK signal or the COMCLK pin to the SIO*n* baud-rate generator input.
DLL0 DLL1 (read/write)	0F4F8H 0F8F8H	03F8H 02F8H	Divisor Latch Low: Stores the lower 8 bits of the SIO*n* baud-rate generator divisor.
DLH0 DLH1 (read/write)	0F4F9H 0F8F9H	03F9H 02F9H	Divisor Latch High: Stores the upper 8 bits of the SIO*n* baud-rate generator divisor.
TBR0 TBR1 (write only)	0F4F8H 0F8F8H	03F8H 02F8H	Transmit Buffer: Holds the data byte to transmit.
RBR0 RBR1 (read only)	0F4F8H 0F8F8H	03F8H 02F8H	Receiver Buffer: Holds the data byte received.
LCR0 LCR1 (read/write)	0F4FBH 0F8FBH	03FBH 02FBH	Line Control: Specifies the data frame (word length, number of stop bits, and type of parity) for transmissions and receptions. Allows the transmitter to transmit a break condition.
LSR0 LSR1 (read only)	0F4FDH 0F8FDH	03FDH 02FDH	Line Status: Contains the transmitter empty, transmit buffer empty, receive buffer full, and receive error flags.
IER0 IER1 (read/write)	0F4F9H 0F8F9H	03F9H 02F9H	Interrupt Enable: Independently connects the four signals (modem status, receive line status, transmit buffer empty, and receive buffer full) to the interrupt request output (SIOINT*n*).

Table 11-5. SIO Registers (Sheet 2 of 2)

Register	Expanded Address	PC/AT* Address	Function
IIR0 IIR1 (read only)	0F4FAH 0F8FAH	03FAH 02FAH	Interrupt ID: Indicates whether the modem status, transmit buffer empty, receive buffer full, or receiver line status signal generated an interrupt request.
MCR0 MCR1 (read/write)	0F4FCH 0F8FCH	03FCH 02FCH	Modem Control: Controls the interface with the modem or data set. Allows use of external UARTs.
MSR0 MSR1 (read/write)	0F4FEH 0F8FEH	03FEH 02FEH	Modem Status: Provides the current state of the control lines for the modem or data set to the CPU.
SCR0 SCR1 (read/write)	0F4FFH 0F8FFH	03FFH 02FFH	Scratch Pad: An 8-bit read/write register available for use as a scratch pad; has no effect on SIO*n* operation.

For PC compatibility, the SIO unit accesses its 11 registers through 8 I/O addresses. The RBR*n*, TBR*n*, and DLL*n* registers share the same addresses and the IER*n* and DLH*n* registers share the same addresses. Bit 7 (DLAB) of the LCR*n* determines which register is accessed during a read or write operation (Table 11-6).

Table 11-6. Access to Multiplexed Registers

Expanded Address	PC/AT Address	Register Accessed	
		DLAB = 0	DLAB = 1
0F4F8H (read)	03F8H (read)	RBR0	DLL0
0F4F8H (write)	03F8H (write)	TBR0	DLL0
0F4F9H (read/write)	03F9H (read/write)	IER0	DLH0
0F8F8H (read)	02F8H (read)	RBR1	DLL1
0F8F8H (write)	02F8H (write)	TBR1	DLL1
0F8F9H (read/write)	02F9H (read/write)	IER1	DLH1

11.3.1 Pin and Port Configuration Registers (PINCFG and P*n*CFG [*n* = 1–3])

Use PINCFG bits 2:0 to connect the SIO1 signals to package pins.

Pin Configuration
PINCFG
(read/write)

Expanded Addr: **F826H**
ISA Addr: **—**
Reset State: **00H**

7							0
—	PM6	PM5	PM4	PM3	PM2	PM1	PM0

Bit Number	Bit Mnemonic	Function
7	—	Reserved. This bit is undefined; for compatibility with future devices, do not modify this bit.
6	PM6	Pin Mode: 0 = Selects CS6# at the package pin. 1 = Selects REFRESH# at the package pin.
5	PM5	Pin Mode: 0 = Selects the coprocessor signals, PEREQ, BUSY#, and ERROR#, at the package pins. 1 = Selects the timer control unit signals, TMROUT2, TMRCLK2, and TMRGATE2, at the package pins.
4	PM4	Pin Mode: 0 = Selects DACK0# at the package pin. 1 = Selects CS5# at the package pin.
3	PM3	Pin Mode: 0 = Selects EOP# at the package pin. 1 = Selects CTS1# at the package pin.
2	PM2	Pin Mode: 0 = Selects DACK1# at the package pin. 1 = Selects TXD1 at the package pin.
1	PM1	Pin Mode: 0 = Selects SRXCLK at the package pin. 1 = Selects DTR1# at the package pin.
0	PM0	Pin Mode: 0 = Selects SSIOTX at the package pin. 1 = Selects RTS1# at the package pin.

Figure 11-7. Pin Configuration Register (PINCFG)

Use P1CFG bits 4:0 to connect SIO0 signals to package pins.

Port 1 Configuration
P1CFG
(read/write)

Expanded Addr: **F820H**
ISA Addr: **—**
Reset State: **00H**

7							0
PM7	PM6	PM5	PM4	PM3	PM2	PM1	PM0

Bit Number	Bit Mnemonic	Function
7	PM7	Pin Mode: 0 = Selects P1.7 at the package pin. 1 = Selects HLDA at the package pin.
6	PM6	Pin Mode: 0 = Selects P1.6 at the package pin. 1 = Selects HOLD at the package pin.
5	PM5	Pin Mode: 0 = Selects P1.5 at the package pin. 1 = Selects LOCK# at the package pin.
4	PM4	Pin Mode: 0 = Selects P1.4 at the package pin. 1 = Selects RI0# at the package pin.
3	PM3	Pin Mode: 0 = Selects P1.3 at the package pin. 1 = Selects DSR0# at the package pin.
2	PM2	Pin Mode: 0 = Selects P1.2 at the package pin. 1 = Selects DTR0# at the package pin.
1	PM1	Pin Mode: 0 = Selects P1.1 at the package pin. 1 = Selects RTS0# at the package pin.
0	PM0	Pin Mode: 0 = Selects P1.0 at the package pin. 1 = Selects DCD0# at the package pin.

Figure 11-8. Port 1 Configuration Register (P1CFG)

Use P2CFG bits 7–5 to connect SIO0 signals to package pins.

Port 2 Configuration
P2CFG
(read/write)

Expanded Addr: **F822H**
ISA Addr: **—**
Reset State: **00H**

7							0
PM7	PM6	PM5	PM4	PM3	PM2	PM1	PM0

Bit Number	Bit Mnemonic	Function
7	PM7	Pin Mode: 0 = Selects P2.7 at the package pin. 1 = Selects CTS0# at the package pin.
6	PM6	Pin Mode: 0 = Selects P2.6 at the package pin. 1 = Selects TXD0 at the package pin.
5	PM5	Pin Mode: 0 = Selects P2.5 at the package pin. 1 = Selects RXD0 at the package pin.
4	PM4	Pin Mode: 0 = Selects P2.4 at the package pin. 1 = Selects CS4# at the package pin.
3	PM3	Pin Mode: 0 = Selects P2.3 at the package pin. 1 = Selects CS3# at the package pin.
2	PM2	Pin Mode: 0 = Selects P2.2 at the package pin. 1 = Selects CS2# at the package pin.
1	PM1	Pin Mode: 0 = Selects P2.1 at the package pin. 1 = Selects CS1# at the package pin.
0	PM0	Pin Mode: 0 = Selects P2.0 at the package pin. 1 = Selects CS0# at the package pin.

Figure 11-9. Port 2 Configuration Register (P2CFG)

Use P3CFG bit 7 to connect the COMCLK pin to the package pin.

Port 3 Configuration
P3CFG
(read/write)

Expanded Addr: **F824H**
ISA Addr: **—**
Reset State: **00H**

7							0
PM7	PM6	PM5	PM4	PM3	PM2	PM1	PM0

Bit Number	Bit Mnemonic	Function
7	PM7	Pin Mode: 0 = Selects P3.7 at the package pin. 1 = Selects COMCLK at the package pin.
6	PM6	Pin Mode: 0 = Selects P3.6 at the package pin. 1 = Selects PWRDOWN at the package pin.
5	PM5	Pin Mode: 0 = Selects P3.5 at the package pin. 1 = Connects master IR7 to the package pin (INT3).
4	PM4	Pin Mode: 0 = Selects P3.4 at the package pin. 1 = Connects master IR6 to the package pin (INT2).
3	PM3	Pin Mode: 0 = Selects P3.3 at the package pin. 1 = Connects master IR5 to the package pin (INT1).
2	PM2	Pin Mode: 0 = Selects P3.2 at the package pin. 1 = Connects master IR1 to the package pin (INT0).
1	PM1	Pin Mode: See Table 5-1 on page 5-8 for all the PM1 configuration options.
0	PM0	Pin Mode: See Table 5-1 on page 5-8 for all the PM0 configuration options.

Figure 11-10. Port 3 Configuration Register (P3CFG)

11.3.2 SIO and SSIO Configuration Register (SIOCFG)

Use SIOCFG to select the baud-rate generator clock source for the SIO channels and to have a channel's modem input signals connected internally rather than to package pins. Selecting the internal modem signal connection option connects RTS# to CTS#, DTR# to DSR# and DCD#, and V_{CC} to RI#. The modem signal connections for this internal option are shown in Figure 11-20.

SIO and SSIO Configuration
SIOCFG
(read/write)

Expanded Addr: **F836H**
ISA Addr: **—**
Reset State: **00H**

7							0
S1M	S0M	—	—	—	SSBSRC	S1BSRC	S0BSRC

Bit Number	Bit Mnemonic	Function
7	S1M	SIO1 Modem Signal Connections: 0 = Connects the SIO1 modem input signals to the package pins. 1 = Connects the SIO1 modem input signals internally.
6	S0M	SIO0 Modem Signal Connections: 0 = Connects the SIO0 modem input signals to the package pins. 1 = Connects the SIO0 modem input signals internally.
5–3	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
2	SSBSRC	SSIO Baud-rate Generator Clock Source: 0 = Connects the internal PSCLK signal to the SSIO baud-rate generator. 1 = Connects the internal SERCLK signal to the SSIO baud-rate generator.
1	S1BSRC	SIO1 Baud-rate Generator Clock Source: 0 = Connects the COMCLK pin to the SIO1 baud-rate generator. 1 = Connects the internal SERCLK signal to the SIO1 baud-rate generator.
0	S0BSRC	SIO0 Baud-rate Generator Clock Source: 0 = Connects the COMCLK pin to the SIO0 baud-rate generator. 1 = Connects the internal SERCLK signal to the SIO0 baud-rate generator.

Figure 11-11. SIO and SSIO Configuration Register (SIOCFG)

11.3.3 Divisor Latch Registers (DLL*n* and DLH*n*)

Use these registers to program the baud-rate generator's output frequency. The baud-rate generator's output determines the transmitter and receiver bit times.

Divisor Latch Low
DLL0, DLL1
(read/write)

	DLL0	DLL1
Expanded Addr:	**F4F8H**	**F8F8H**
ISA Addr:	**03F8H**	**02F8H**
Reset State:	**02H**	**02H**

7							0
LD7	LD6	LD5	LD4	LD3	LD2	LD1	LD0

Divisor Latch High
DLH0, DLH1
(read/write)

	DLH0	DLH1
Expanded Addr:	**F4F9H**	**F8F9H**
ISA Addr:	**03F9H**	**02F9H**
Reset State:	**00H**	**00H**

7							0
UD15	UD14	UD13	UD12	UD11	UD10	UD9	UD8

Bit Number	Bit Mnemonic	Function
DLL*n* (7–0)	LD7:0	Lower 8 Divisor and Upper 8 Divisor Bits: Write the lower 8 divisor bits to DLL*n* and the upper 8 divisor bits to DLH*n*. The baud-rate generator output is a function of the baud-rate generator input (BCLKIN) and the 16-bit divisor. $\text{baud-rate generator output frequency} = \frac{\text{BCLKIN frequency}}{\text{divisor}}$ bit rate (shifting rate) = baud-rate generator output frequency/16
DLH*n* (7–0)	UD15:8	

NOTE: The divisor latch registers share address ports with other SIO registers. Bit 7 (DLAB) of LCR*n* must be set in order to access the divisor latch registers.

If DLL = DLH = 00H, baud-rate generator ouput frequency = 0 (stops clock).

Figure 11-12. Divisor Latch Registers (DLL*n* and DLH*n*)

11.3.4 Transmit Buffer Register (TBR*n*)

Write the data words to be transmitted to TBR*n*. Use the interrupt control or DMA units or poll the serial line status register (LSR*n*) to determine whether the transmit buffer is empty.

Transmit Buffer
TBR0, TBR1
(write only)

	TBR0	TBR1
Expanded Addr:	**F4F8H**	**F8F8H**
ISA Addr:	**03F8H**	**02F8H**
Reset State:	**XXH**	**XXH**

7							0
TB7	TB6	TB5	TB4	TB3	TB2	TB1	TB0

Bit Number	Bit Mnemonic	Function
7–0	TB7:0	Transmit Buffer Bits: These bits make up the next data word to be transmitted. The transmitter loads this word into the transmit shift register. The transmit shift register then shifts the bits out, along with the asynchronous communication bits (start, stop, and parity). The data bits are shifted out least-significant bit (TB0) first.

NOTE: The transmit buffer register shares an address port with other SIO registers. You must clear bit 7 (DLAB) of LCR*n* before you can write to the transmit buffer register.

Figure 11-13. Transmit Buffer Register (TBR*n*)

11.3.5 Receive Buffer Register (RBR*n*)

Read RBR*n* to obtain the last data word received. Use the interrupt control or DMA units or poll the serial line status register (LSR*n*) to determine whether the receive buffer is full.

Receive Buffer
RBR0, RBR1
(read only)

	RBR0	RBR1
Expanded Addr:	**F4F8H**	**F8F8H**
ISA Addr:	**03F8H**	**02F8H**
Reset State:	**XXH**	**XXH**

7							0
RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0

Bit Number	Bit Mnemonic	Function
7–0	RB7:0	Receive Buffer Bits: These bits make up the last word received. The receiver shifts bits in, starting with the least-significant-bit. The receiver then strips off the asynchronous bits (start, parity, and stop) and transfers the received data bits from the receive shift register to the receive buffer.

NOTE: The receive buffer register shares an address port with other SIO registers. Bit 7 (DLAB) of the LCR*n* must be cleared in order to read the receive buffer register.

Figure 11-14. Receive Buffer Register (RBR*n*)

11.3.6 Serial Line Control Register (LCR*n*)

Use LCR*n* to provide access to the multiplexed registers, send a break condition, and determine the data frame for receptions and transmissions.

Serial Line Control
LCR0, LCR1
(read/write)

	LCR0	LCR1
Expanded Addr:	**F4FBH**	**F8FBH**
ISA Addr:	**03FBH**	**02FBH**
Reset State:	**00H**	**00H**

7							0
DLAB	SB	SP	EPS	PEN	STB	WLS1	WLS0

Bit Number	Bit Mnemonic	Function
7	DLAB	Divisor Latch Access Bit: This bit determines which of the multiplexed registers is accessed. 0 = Allows access to the receiver and transmit buffer registers (RBR*n* and TBR*n*) and the interrupt enable register (IER*n*). 1 = Allows access to the divisor latch registers (DLL*n* and DLH*n*).
6	SB	Set Break: 0 = No effect on TXD*n*. 1 = Forces the TXD*n* pin to the spacing (logic 0) state for as long as bit is set.
5	SP	Sticky Parity, Even Parity Select, and Parity Enable: These bits determine whether the control logic produces (during transmission) or checks for (during reception) even, odd, no, or forced parity. **SP EPS PEN Function** X X 0 parity disabled (no parity option) 0 0 1 produce or check for odd parity 0 1 1 produce or check for even parity 1 0 1 produce or check for forced parity (parity bit = 1) 1 1 1 produce or check for forced parity (parity bit = 0)
4	EPS	
3	PEN	
2	STB	Stop Bits: This bit specifies the number of stop bits transmitted and received in each serial character. 0 = 1 stop bit 1 = 2 stop bits (1.5 stop bits for 5-bit characters)
1–0	WLS1:0	Word Length Select: These bits specify the number of data bits in each transmitted or received serial character. 00 = 5-bit character 01 = 6-bit character 10 = 7-bit character 11 = 8-bit character

Figure 11-15. Serial Line Control Register (LCR*n*)

11.3.7 Serial Line Status Register (LSR*n*)

Use LSR*n* to check the status of the transmitter and receiver.

Serial Line Status
LSR0, LSR1
(read only)

	LSR0	LSR1
Expanded Addr:	**F4FDH**	**F8FDH**
ISA Addr:	**03FDH**	**02FDH**
Reset State:	**60H**	**60H**

7							0
—	TE	TBE	BI	FE	PE	OE	RBF

Bit Number	Bit Mnemonic	Function
7	—	Reserved. This bit is undefined.
6	TE	Transmitter Empty: The transmitter sets this bit to indicate that the transmit shift register and transmit buffer register are both empty. Writing to the transmit buffer register clears this bit.
5	TBE	Transmit Buffer Empty: The transmitter sets this bit after it transfers data from the transmit buffer to the transmit shift register. Writing to the transmit buffer register clears this bit.
4	BI	Break Interrupt: The receiver sets this bit whenever the received data input is held in the spacing (logic 0) state for longer than a full word transmission time. Reading the receive buffer register or the serial line status register clears this bit.
3	FE	Framing Error The receiver sets this bit to indicate that the received character did not have a valid stop bit. Reading the receive buffer register or the serial line status register clears this bit. If data frame is set for two stop bits the second stop bit is ignored.
2	PE	Parity Error: The receiver sets this bit to indicate that the received data character did not have the correct parity. Reading the receive buffer register or the serial line status register clears this bit.
1	OE	Overrun Error: The receiver sets this bit to indicate an overrun error. An overrun occurs when the receiver transfers a received character to the receive buffer register before the CPU reads the buffer's old character. Reading the serial line status register clears this bit.
0	RBF	Receive Buffer Full: The receiver sets this bit after it transfers a received character from the receive shift register to the receive buffer register. Reading the receive buffer register clears this bit.

Figure 11-16. Serial Line Status Register (LSR*n*)

11.3.8 Interrupt Enable Register (IER*n*)

Use IER*n* to connect the SIO*n* status signals to the interrupt control unit. All four status signals can be connected to the interrupt control unit.

Interrupt Enable
IER0, IER1
(read/write)

	IER0	IER1
Expanded Addr:	**F4F9H**	**F8F9H**
ISA Addr:	**03F9H**	**02F9H**
Reset State:	**00H**	**00H**

7							0
—	—	—	—	MS	RLS	TBE	RBF

Bit Number	Bit Mnemonic	Function
7–4	—	Reserved; for compatibility with future devices, write zeros to these bits.
3	MS	Modem Status Interrupt Enable: 0 = Modem input signal changes do not cause interrupts. 1 = Connects the modem status signal to the interrupt control unit's SIOINT*n* output. A change on one or more of the modem input signals activates the modem status signal.
2	RLS	Receiver Line Status Interrupt Enable: 0 = LSR error conditions do not cause interrupts. 1 = Connects the receiver line status signal to the interrupt control unit's SIOINT*n* output. Sources for this interrupt include overrun error, parity error, framing error, and break interrupt.
1	TBE	Transmit Buffer Empty Interrupt Enable: 0 = Transmit Buffer Empty signal does not cause interrupts. 1 = Connects the transmit buffer empty signal to the interrupt control unit's SIOINT*n* output.
0	RBF	Receive Buffer Full Interrupt Enable: 0 = Receive buffer full signal does not cause interrupts. 1 = Connects the receive buffer full signal to the interrupt control unit's SIOINT*n* output.

NOTE: The interrupt enable register is multiplexed with the divisor latch high register. You must clear bit 7 (DLAB) of the serial line control register (LCR*n*) before you can access the interrupt control register.

Figure 11-17. Interrupt Enable Register (IER*n*)

11.3.9 Interrupt ID Register (IIR*n*)

Use the IIR*n* to determine whether an interrupt is pending and, if so, which status signal generated the interrupt request.

Interrupt ID
IIR0, IIR1
(read only)

	IIR0	IIR1
Expanded Addr:	**F4FAH**	**F8FAH**
ISA Addr:	**03FAH**	**02FAH**
Reset State:	**01H**	**01H**

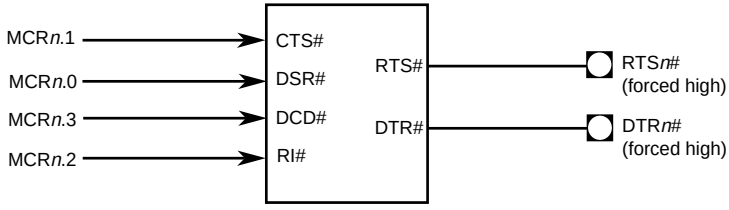
7							0
—	—	—	—	—	IS2	IS1	IP#

Bit Number	Bit Mnemonic	Function
7–3	—	Reserved. These bits are undefined.
2	IS2:1	Interrupt Source: If an interrupt is pending (bit 0 = 0), these bits specify which status signal caused the pending interrupt. **IS2 IS1 Interrupt Source** 0 0 modem status signal* 0 1 transmitter buffer empty signal 1 0 receive buffer full signal 1 1 receiver line status signal** * When one of the modem input signals (CTS*n*#, DSR*n*#, RI*n*#, and DCD*n*#) changes state, the modem status signal is activated. ** A framing error, overrun error, parity error, or break interrupt activates the receiver line status signal. Reading the modem status register clears the modem status signal. Reading the IIR*n* register or writing to the transmit buffer register clears the transmit buffer empty signal. Reading the receive buffer register clears the receive buffer full signal. Reading the receive buffer register or the serial line status register clears the LSR*n* error bits, which clears the receiver line status signal.
0	IP#	Interrupt Pending: This bit indicates whether an interrupt is pending. 0 = Interrupt is pending 1 = No interrupt is pending

Figure 11-18. Interrupt ID Register (IIR*n*)

11.3.10 Modem Control Register (MCR*n*)

Use MCR*n* to put the SIO*n* into a diagnostic test mode. In this mode, the modem input signals are disconnected from the package pins and controlled by the lower four MCR*n* bits and the modem output signals are forced to their inactive states (Figure 11-19). Additionally, the MCR*n* signals are also forced into the MSR register bits.

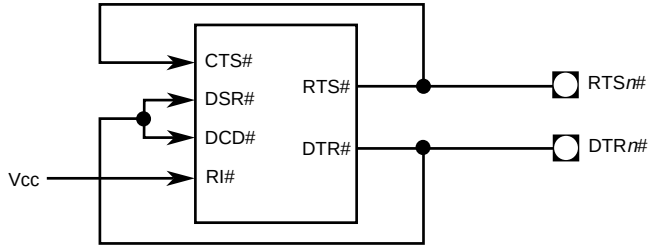


Note: MCR*n*.1 indicates that modem control register bit 1 controls the CTS input, and so on.

A2529-01

Figure 11-19. Modem Control Signals – Diagnostic Mode Connections

Besides the diagnostic mode, there are two other options for connecting the modem input signals. You can connect the signals internally using the SIO configuration (SIOCFG) register. The internal connection mode disconnects the modem input signals from the package pins and connects the modem output signals to the modem input signals (in this case, the modem output signals remain connected to package pins). See Figure 11-20. In this mode, the values you write to MCR*n* bits 0 and 1 control the state of the modem's internal input signals and output pins.



A2528-01

Figure 11-20. Modem Control Signals – Internal Connections

The other option is standard mode. In standard mode, the modem input and output signals are connected to the package pins. In this mode, the values you write to MCR*n* bits 0 and 1 control the state of the modem's output pins.

Modem Control
MCR0, MCR1
(read/write)

	MCR0	MCR1
Expanded Addr:	**F4FCH**	**F8FCH**
ISA Addr:	**03FCH**	**02FCH**
Reset State:	**00H**	**00H**

7							0
—	—	—	LOOP	OUT2	OUT1	RTS	DTR

Bit Number	Bit Mnemonic	Function
7–5	—	Reserved; for compatibility with future devices, write zeros to these bits.
4	LOOP	Loop Back Test Mode: 0 = Normal mode 1 = Setting this bit puts the SIO*n* into diagnostic (or loop back test) mode. This causes the SIO channel to: • set its transmit serial output (TXD*n*) • disconnect its receive serial input (RXD*n*) from the package pin • loop back the transmitter shift register's output to the receive shift register's input • disconnect the modem control inputs (CTS*n*#, DSR*n*#, RI*n*#, and DCD*n*#) from the package pins • force modem control outputs (RTS*n*# and DTR*n*#) to their inactive states • connects MCR*n* bits to MSR*n* bits
3–2	OUT2:1	Test Bits: In diagnostic mode (bit 4=1), these bits control the ring indicator (RI*n*) and data carrier detect (DCD*n*#) modem inputs. Setting OUT1 activates the internal RI*n* bit; clearing OUT1 deactivates the internal RI*n* bit. Setting OUT2 activates the internal DCD*n* bit; clear OUT2 deactivates the internal DCD*n* bit. In normal user mode (bit 4=0) OUT1 has no effect and OUT2 in conjunction with INTCFG.5/6 selects internal SIO interrupt or external interrupt. See Table 5-1 on page 5-8 for the configuration options.
1	RTS	Ready to Send: The function of this bit depends on whether the SIO*n* is in diagnostic mode (MCR*n*.4=1), internal connection mode, or standard mode. In diagnostic mode, setting this bit activates the internal CTS*n* bit; clearing this bit deactivates the internal CTS*n* bit. In internal connection mode, setting this bit activates the internal CTS*n*# signal and the RTS*n*# pin; clearing this bit deactivates the internal CTS*n*# signal and the RTS*n*# pin. In standard mode, setting this bit activates the RTS*n*# pin; clearing this bit deactivates the RTS*n*# pin. Note that pin is inverted from bit.
0	DTR	Data Terminal Ready: The function of this bit depends on whether the SIO*n* is in diagnostic mode (MCR*n*.4=1), internal connection mode, or standard mode. In diagnostic mode, setting this bit activates the internal DSR*n*# signal; clearing this bit deactivates the internal DSR*n*# signal. In internal connection mode, setting this bit activates the internal DSR*n*# and DCD*n*# signals and the DTR*n*# pin; clearing this bit deactivates the internal DSR*n*# and DCD*n*# signals and the DTR*n*# pin. Note that pin is inverted from bit. In standard mode, setting this bit activates the DTR*n*# pin; clearing this bit deactivates the DTR*n*# pin. Note that pin is inverted from bit.

Figure 11-21. Modem Control Register (MCR*n*)

11.3.11 Modem Status Register (MSR*n*)

Read MSR*n* to determine the status of the modem control input signals. The upper four bits reflect the current state of the modem input signals and the lower four bits indicate whether the inputs (except for RI#) have changed state since the last time this register was read. These lower four bits are reset to zero when the CPU reads the Modem Status register.

Modem Status
MSR0, MSR1
(read only)

	MSR0	MSR1
Expanded Addr:	**F4FEH**	**F8FEH**
ISA Addr:	**03FEH**	**02FEH**
Reset State:	**X0H**	**X0H**

7							0
DCD	RI	DSR	CTS	DDCD	TERI	DDSR	DCTS

Bit Number	Bit Mnemonic	Function
7	DCD	Data Carrier Detect: This bit is the complement of the data carrier detect (DCD*n*#) input. In diagnostic test mode, this bit is equivalent to MCR*n*.3 (OUT2).
6	RI	Ring Indicator: This bit is the complement of the ring indicator (RI*n*#) input. In diagnostic test mode, this bit is equivalent to MCR*n*.2 (OUT1).
5	DSR	Data Set Ready: This bit is the complement of the data set ready (DSR*n*#) input. In diagnostic test mode, this bit is equivalent to MCR*n*.0 (DTR).
4	CTS	Clear to Send: This bit is the complement of the clear to send (CTS*n*#) input. In diagnostic test mode, this bit is equivalent to MCR*n*.1 (RTS).
3	DDCD	Delta Data Carrier Detect: When set, this bit indicates that the DCD*n*# input has changed state since the last time this register was read. Reading this register clears this bit.
2	TERI	Trailing Edge Ring Indicator: When set, this bit indicates that the RI*n*# input has changed from a low to a high state since the last time this register was read. Reading this register clears this bit.
1	DDSR	Delta Data Set Ready: When set, this bit indicates that the DSR*n*# input has changed state since the last time this register was read. Reading this register clears this bit.
0	DCTS	Delta Clear to Send: When set, this bit indicates that the CTS*n*# input has changed state since the last time this register was read. Reading this register clears this bit.

Figure 11-22. Modem Status Register (MSR*n*)

11.3.12 Scratch Pad Register (SCR*n*)

SCR*n* is available for use as a scratch pad. Writing and reading this register has no effect on SIO*n* operation.

Scratch Pad
SCR0, SCR1
(read/write)

	SCR0	SCR1
Expanded Addr:	F4FFH	F8FFH
ISA Addr:	03FFH	02FFH
Reset State:	XXH	XXH

7							0
SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0

Bit Number	Bit Mnemonic	Function
7–0	SP7:0	Writing and reading this register has no effect on SIO*n* operation.

Figure 11-23. Scratch Pad Register (SCR*n*)

11.4 PROGRAMMING CONSIDERATIONS

Consider the following when programming the SIO.

- The divisor latch low register (DLL*n*) is multiplexed with the receive and transmit buffer registers (RBR*n* and TBR*n*) and the divisor latch high register (DLH*n*) is multiplexed with the interrupt enable register (IER*n*). Bit 7 of the serial line control register (LCR*n*) controls which register is accessed.
- The SIO contains four status signals: receiver line status, receive buffer full, transmit buffer empty, and modem status. You can connect (OR) these signals to the interrupt control unit's SIOINT*n* interrupt request signal using the interrupt enable register (IER*n*). If you receive an interrupt request on the SIOINT*n* signal, read the interrupt ID register (IIR*n*) to determine which status signal with the highest priority caused the request.

  Several sources can activate the receiver line status and the modem status signals. If IIR*n* indicates that the receiver line status signal caused an interrupt request, read the serial line status register (LSR*n*) to determine the receive error condition that activated the receiver line status signal. If IIR*n* indicates that the modem status signal caused an interrupt request, read the modem status register (MSR*n*) to determine which modem input signal activated the modem status signal.
- DMA can be used for servicing the SIO channels for higher baud rates. When doing this, remember that the isolated RBF and TBE (RBFDMA and TBEDMA) signals are connected to the DMA DREQ inputs. RBFDMA is blocked if any of the error bits in the LSR are set. Neither signal is gated by the IER*n* register.

11.4.1 Asynchronous Serial I/O Unit Code Examples

The code example contains these software routines:

InitSIO	Initializes the SIO for asynchronous transfers
SerialReadStr	Polled serial read function that reads a specified number of characters
SerialReadChar	Polled serial read function that reads a single character
SerialWriteChar	Polled serial write function that writes a single character
SerialWriteStr	Polled serial write function that writes out an entire string of characters
SerialWriteMem	Polled serial write function that writes out a specified number of characters stored in a buffer
Serial0_ISR	Template interrupt service routine for SIO_0 interrupts
Service_RBF	Service routine for interrupts generated by the Receive Buffer Full signal
SerialWriteStr_Int	Interrupt driven serial write function
Service_TBE	Service routine for interrupts generated by the Transmit Buffer Empty signal

The last software routine shows how to use these functions to enable RBF interrupts on the SIO. See Appendix C for the included header files.

```
#include <conio.h>
#include <stdio.h>
#include “80386ex.h”
#include “ev386ex.h”

/* Variable Declarations */
int Tbuffer_index = 0;
char trans_buffer[1024];
char rec_buffer;

/*****************************************************************************
   InitSIO:

   Description:
      Initialization routine for Asynchronous Serial I/O Port.

   Parameters:
      Unit             Unit number of the serial port. 0 for SIO port 0,
                       1 for SIO port 1.
      Mode             Defines Parity, number of data bits, number of stop bits--
                       Reference Serial Line Control register for various
```

```
options
      ModemCntrl       Defines the operation of the modem control lines
      BaudRate         Specifies baud rate. The baud divisor value is calculated
                       based on clocking source and clock frequency. The
                       clocking frequency is set by calling the
                       InitializeLibrary function.
      ClockRate        Specifies the serial port clocking rate, for internal
                       clocking = CLK2, for external = COMCLK

   Returns:Error Codes
      E_INVAILD_DEVICE -- Unit number specifies a non-existing device
      E_OK             -- Initialized OK, No error.

   Assumptions:
      SIOCFG  Has already been configured for Clocking source and Modem
      control source

      REMAPCFG register has Expanded I/O space access enabled (ESE bit set).
      The processor Port pin are initialized separately.

    Syntax:

      #define SIO_0        0x0
      #define SIO_8N1      (SIO_8DATA | SIO_1STOPBIT | SIO_NOPARITY)
      #define SIO_MCR_RTS  0x2
      #define SIO_MCR_DTR  0x1
      #define SIO_8DATA    0x3
      #define SIO_1STOPBIT 0x0
      //Clock rate of COMCLK,  i.e., External clocking
      #define BAUD_CLKIN   1843200L

      int error;

      error = InitSIO(SIO_0,             // Which Serial Port
                  SIO_8N1,               // Mode, 8-data, no parity, 1-stop
                  SIO_MCR_RTS+SIO_MCR_DTR, // Modem line controls
                  9600,                  // Baud Rate
                  BAUD_CLKIN);           // Baud Clocking Rate

   Real/Protected Mode:
      No changes required.

*************************************************************************/

int InitSIO(int Unit, BYTE Mode, BYTE ModemCntrl, DWORD BaudRate,
            DWORD BaudClkIn)
{
   WORD SIOPortBase;
   WORD BaudDivisor;

   /* Check for valid unit */
   if(Unit > 1)
```

```
      return E_INVALID_DEVICE;

   /* Set Port base based on serial port used */
   SIOPortBase = (Unit ? SIO1_BASE : SIO0_BASE);

   /* Initialized Serial Port registers */

   /* Calculate the baud divisor value, based on baud clocking */
   BaudDivisor = (WORD)(BaudClkIn / (16*BaudRate));

   /* Turn on access to baud divisor register */
   _SetEXRegByte(SIOPortBase + LCR, 0x80);

   /* Set the baud rate divisor register, High byte first */
   _SetEXRegByte(SIOPortBase + DLH, HIBYTE(BaudDivisor) );
   _SetEXRegByte(SIOPortBase + DLL, LOBYTE(BaudDivisor) );

   /*** Set Serial Line control register ***/
   _SetEXRegByte(SIOPortBase + LCR, Mode);// Sets Mode and
                                          //reset the Divisor latch

   /* Set modem control bits */
   _SetEXRegByte(SIOPortBase + MCR, ModemCntrl);

   return E_OK;
}/* InitSIO */




/*****************************************************************************
   SerialReadStr

   Description:
      Is a Polled serial port read function that waits forever or until
      count characters are read from the serial port.

   Parameters:
      Unit             Unit number of the serial port. 0 for SIO port 0,
                       1 for SIO port 1.
      str              Address of where to place the input data
      count            Number of characters to read before returning.

   Returns: Error Code
      E_OK or Error code status (value of Line Status Register (LSR)

   Assumptions:
      REMAPCFG register has Expanded I/O space access enabled (ESE bit set).
      The processor Port pin are initialized separately.

   Syntax:
```

```
      #define SIO_0  0
      #define LENGTH 32

      char String_Read[LENGTH];
      int error;

      error = SerialReadStr (SIO_0,
                          String_Read,
                          LENGTH);

   Real/Protected Mode
      No changes required.

*****************************************************************************/

int SerialReadStr(int Unit, char far *str, int count)
{
   WORD ReceivePortAddr;
   WORD StatusPortAddr;
   BYTE Status;
   int i;

   /* Set Port base, based on serial port used */
   ReceivePortAddr = (Unit ? RBR1 : RBR0);
   StatusPortAddr =  (Unit ? LSR1 : LSR0);

   for(i=0; i < count-1; i++)
   {
      // Status register is cleared after read, so we must save
      // it's value when read
      while(!((Status=_GetEXRegByte(StatusPortAddr)) & SIO_RX_BUF_FULL))
        if( Status & SIO_ERROR_BITS )  /* Error Bit set then return NULL */
        {
                      str[i+1] = '\0';
                      return Status & SIO_ERROR_BITS;
        }
        str[i] = _GetEXRegByte(ReceivePortAddr);
    }
    str[i] = '\0';
    return E_OK;
}/* SerialReadStr */




/*****************************************************************************
   SerialReadChar:

   Description:
      Is a Polled serial port read function that waits forever or
```

```
      until a character has been received from the serial port.

   Parameters:
      Unit  Unit number of the serial port. 0 for SIO port 0,
            1 for SIO port 1.

   Returns:
      BYTE Read from serial port, if zero an error occurred.

   Assumptions:
      REMAPCFG register has Expanded I/O space access enabled (ESE bit set).
      The processor Port pin are initialized separately.

   Syntax:

      #define SIO_0 0

      BYTE character;

      character = SerialReadChar (SIO_0);

   Real/Protected Mode
      No changes required.

*****************************************************************************/

BYTE SerialReadChar(int Unit)
{
   WORD ReceivePortAddr;
   WORD StatusPortAddr;
   WORD Status;

   /* Set Port base, based on serial port used */
   ReceivePortAddr = (Unit ? RBR1 : RBR0);
   StatusPortAddr =  (Unit ? LSR1 : LSR0);

   // Status register is cleared after read, so we must save
   // it's value when read
   while(!((Status=_GetEXRegByte(StatusPortAddr)) & SIO_RX_BUF_FULL))
      if( Status & SIO_ERROR_BITS )  // Error Bit set then return NULL
      {
        return 0;
      }

   return _GetEXRegByte(ReceivePortAddr);
}/* SerialReadChar */




/*****************************************************************************
```

```
   SerialWriteChar:

   Description:
      Is a Polled serial port write function that waits forever or
      until a character has been written to the serial port.

   Parameters:
      Unit   Unit number of the serial port. 0 for SIO port 0,
             1 for SIO port 1.
      ch     Character value to be written out

   Returns:
      None

   Assumptions:
      REMAPCFG register has Expanded I/O space access enabled (ESE bit set).
      The processor Port pin are initialized separately.

   Syntax:

       #define SIO_0 0

       char Char_Out = 'a';

      SerialWriteChar (SIO_0, Char_Out);

   Real/Protected Mode
      No changes required.

*****************************************************************************/

void SerialWriteChar(int Unit, BYTE ch)
{
   WORD TransmitPortAddr;
   WORD StatusPortAddr;

   /* Set Port base, based on serial port used */
   TransmitPortAddr = (Unit ? TBR1 : TBR0);
   StatusPortAddr =  (Unit ?  LSR1 : LSR0);

   /* Wait until buffer is empty */
   while(!(_GetEXRegByte(StatusPortAddr) & SIO_TX_BUF_EMPTY)) ;

   _SetEXRegByte(TransmitPortAddr,ch);
}/* SerialWriteChar */




/*****************************************************************************
   SerialWriteStr:
```

```
   Description:
      Is a Polled serial port write function that waits forever
      or until all characters have been written to the serial port.
      The NUL character ('\0') is used to indicate end of string.

   Parameters:
      Unit   Unit number of the serial port. 0 for SIO port 0,
             1 for SIO port 1.
      str    Address of a zero terminated string to be transmitted

   Returns:
      None

   Assumptions:
      REMAPCFG register has Expanded I/O space access enabled (ESE bit set).
      The processor Port pin are initialized separately.

   Syntax:

      #define SIO_0 0

      SerialWriteStr  (SIO_0,
                       HelloString);

   Real/Protected Mode
      No changes required.

*************************************************************************/

void SerialWriteStr(int Unit, const char far *str)
{
   WORD TransmitPortAddr;
   WORD StatusPortAddr;

   /* Set Port base, based on serial port used */
   TransmitPortAddr = (Unit ? TBR1 : TBR0);
   StatusPortAddr =  (Unit ? LSR1 : LSR0);

   for( ; *str != '\0'; str++)
   {
      /* Wait until buffer is empty */
      while(!(_GetEXRegByte(StatusPortAddr) & SIO_TX_BUF_EMPTY)) ;
      /* Write Character */
      _SetEXRegByte(TransmitPortAddr,*str);
   }
}/* SerialWriteStr */
```

```
/*****************************************************************************
   SerialWriteMem:

   Description:
      Is a Polled serial port write function that waits forever or
      until count characters have been written to the serial port.

   Parameters:
      Unit   Unit number of the serial port. 0 for SIO port 0,
             1 for SIO port 1.
      mem    Address of a buffer to be transmitted
      count  Number of characters in buffer to be transmitted

   Returns:
      None

   Assumptions:
      REMAPCFG register has Expanded I/O space access enabled (ESE bit set).
      The processor Port pin are initialized separately.

   Syntax:

      #define SIO_0 0
      #define COUNT 32

      char Buffer[COUNT];

      SerialWriteMem   (SIO_0,
                        Buffer,
                        COUNT);

   Real/Protected Mode
      No changes required.

*****************************************************************************/

void SerialWriteMem(int Unit, const char far *mem, int count)
{
   WORD TransmitPortAddr;
   WORD StatusPortAddr;
   int i;

   /* Set Port base, based on serial port used */
   TransmitPortAddr = (Unit ? TBR1 : TBR0);
   StatusPortAddr =  (Unit ? LSR1 : LSR0);

   for(i=0 ; i < count; i++)
   {
      /* Wait until buffer is empty */
      while(!(_GetEXRegByte(StatusPortAddr) & SIO_TX_BUF_EMPTY)) ;
      /* Write Character */
      _SetEXRegByte(TransmitPortAddr,mem[i]);
```

```
   }

} /* SerialWriteMem */



/*****************************************************************************

   Serial0_ISR:

   Description:
      Template Interrupt Service Routine for Serial Port0 interrupts.
      This function identifies the cause of the interrupt and branches
      to the corresponding action.

   Parameters:
      None (Not called by user)

   Returns:
      None

   Assumptions:
      None

   Syntax:
      Not a user function.

   Real/Protected Mode:
      No changes required.

*****************************************************************************/

void interrupt far Serial0_ISR (void)
{

 BYTE iir0, lsr0, msr0;

 iir0 = _GetEXRegByte(IIR0);

 switch ((iir0&0x06) >> 1) {

   case 0:
   /* modem status signal */

      msr0 = _GetEXRegByte(MSR0);

      if ((msr0&0x08) && (msr0&0x80)){

        /* data carrier detect has been set */
      }
```

```
      if ((msr0 & 0x04) && (msr0 & 0x40)) {

        /* ring indicator */
      }

      if ((msr0 & 0x02) && (msr0 & 0x20)) {

        /* data set ready bit has been set */
      }

      if ((msr0 & 0x01) && (msr0&0x10)) {

        /* clear to send signal has been set */
      }

      break;
   case 1:
   Service_TBE(); /* Routine for Interrupt driven Serial Writes */
      break;
   case 2:
   /* RBF signal */
   Service_RBF(); /* Routine specific to RBF generated interrupts */
   break;
   case 3:
   /* receive line status signal */

   lsr0 = _GetEXRegByte(LSR0);

   if (lsr0 & 0x10) {
     /* break interrupt */
   }

   if (lsr0 & 0x08) {
     /* framing error */
   }

   if (lsr0 & 0x04) {
     /* parity error */
   }

   if (lsr0 & 0x02) {
     /* overrun error */
   }

      break;

 } /* End of switch */

 NonSpecificEOI();  // Send End-Of-Interrupt Signal to Master

}/* Serial0_ISR */
```

```
/*****************************************************************************

   Service_RBF:

   Description:
      Service routine for interrupts generated by RBF signal. This
      routine is used for Interrupt-Driven Serial Reads. It echoes
      the typed character to the screen, stopping when it receives
      an ESC character.

   Parameters:
      None

   Returns:
      None

   Assumptions:
      None

   Syntax:
      Not called by user

   Real/Protected Mode:
      No changes required.

*****************************************************************************/

void Service_RBF (void)
{


   /*  Read in contents of RBR0 */
   rec_buffer = _GetEXRegByte(RBR0);

   SerialWriteChar(SIO_0, rec_buffer); // Echo to screen

   if ( rec_buffer == 0x1b ) {

   /* ESC character received, disable RBF interrupts*/
   _SetEXRegByte(IER0, 0x00);
   }

}/* Service_RBF */

/*****************************************************************************

   SerialWriteStr_Int:
```

```
   Description:
      Is an interrupt driven serial port write function.
      The NUL character ('\0') is used to indicate end of string.

   Parameters:
      Unit   Unit number of the serial port. 0 for SIO port 0,
             1 for SIO port 1.
      str    Address of a zero terminated string to be transmitted

   Returns:
      None

   Assumptions:
      REMAPCFG register has Expanded I/O space access enabled (ESE bit set).
      The processor Port pin are initialized separately.

   Syntax:

      #define SIO_0 0

      SerialWriteStr_Int (SIO_0, HelloString);

   Real/Protected Mode
      No changes required.

*****************************************************************************/


void SerialWriteStr_Int(int Unit, const char far *str)
{
   BYTE PortIntEnable;

   PortIntEnable = (Unit ? IER1 : IER0);

   strcpy (trans_buffer, str);  // Copy string into buffer

   /* Enable TBE interrupts */
   _SetEXRegByte(IER0,0x02);

}/* SerialWriteStr_Int */


/*****************************************************************************

   Service_TBE:

   Description:
      Service routine for TBE generated interrupts. This function is used
      for Interrupt-Driven Serial Transmits.

   Parameters:
```

```
      None

   Assumptions:
      None

   Syntax:
      Not called by user.

   Real/Protected Mode:
      No changes required.

*****************************************************************************/

void Service_TBE(void)
{
   if (trans_buffer[Tbuffer_index] != '\0') {
      _SetEXRegByte(TBR0, trans_buffer[Tbuffer_index]);
      Tbuffer_index++;
   }
   else {
   /* Disable TBE interrupts */
      _SetEXRegByte(IER0, 0x00);

   }

}/* Service_TBE */


/*****************************************************************************

 Example code to show how to set up for a Serial Port interrupt.
 This example is for an interrupt on SIO_0 sourced by the
 Receive Buffer Full Signal. The source code for the functions "SetIRQVector"
 and "Disable8259Interrupt" is included in the Interrupt Control Unit chapter.



   SetIRQVector(Serial0_ISR, 4, INTERRUPT_ISR);  // Set vector for Interrupt
                                                 // on Master line 4

   Disable8259Interrupt(IR1+IR5+IR6+IR7, IR0+IR1+IR2+IR3+IR4+IR5+IR6+IR7);
   Enable8259Interrupt(IR2+IR4,0);// Enable slave interrupt to master(IR2),
                                  // Enable SIO_0 (IR4)
   _enable(); // Enable Interrupts

   _SetEXRegByte(IER0, 0x01);    // Enable interrupt on RBF signal

/*****************************************************************************/
```

12

DMA CONTROLLER

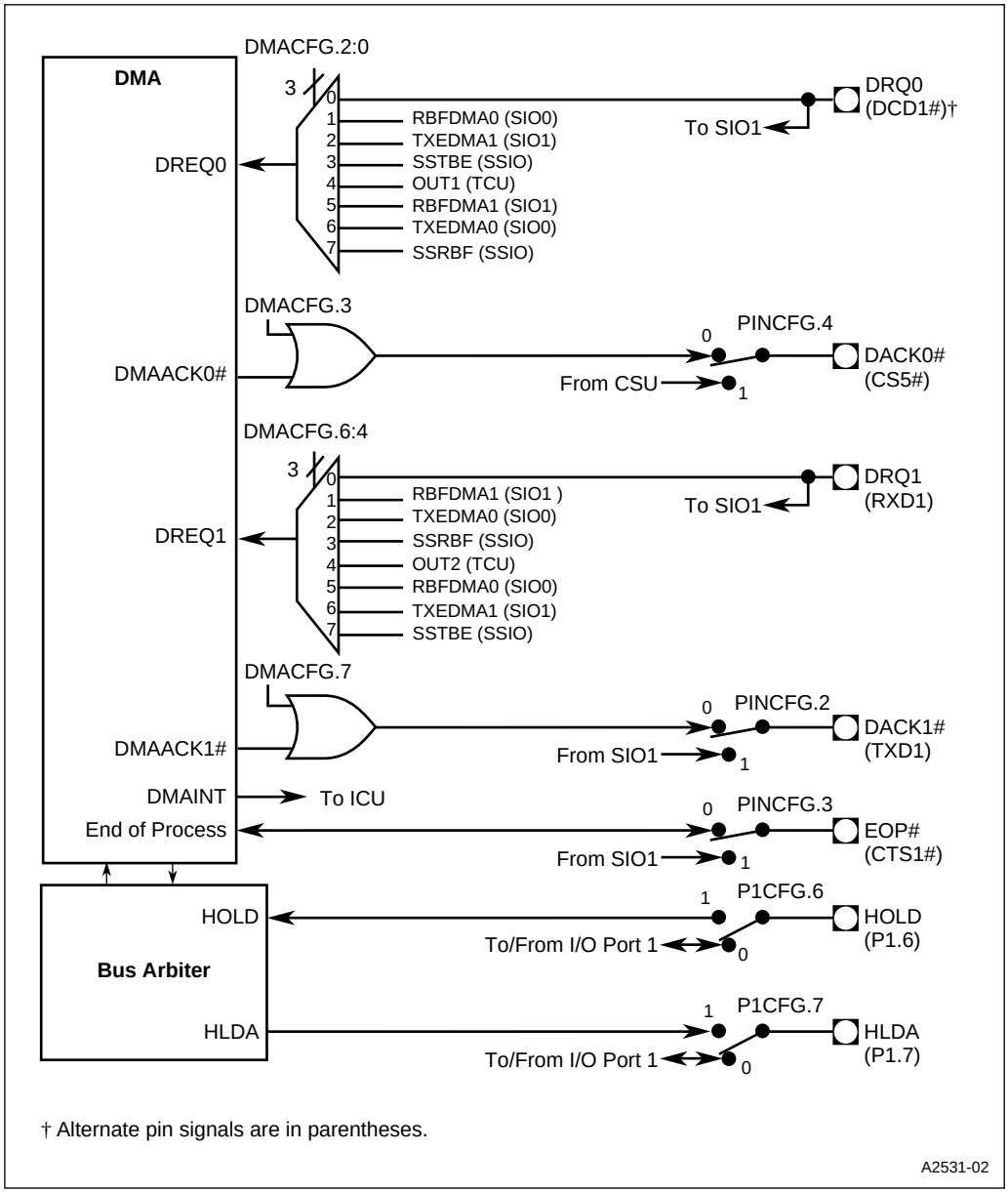
CHAPTER 12
DMA CONTROLLER

The DMA controller improves system performance by allowing external or internal peripherals to directly transfer information to or from the system. The DMA controller can transfer data between any combination of memory and I/O, with any combination of data path widths (8 or 16 bits). It contains two identical channels. The DMA controller has features that are unavailable on an 8237A, but it can be configured to operate in an 8237A-compatible mode.

This chapter is organized as follows:

12.1 OVERVIEW

Figure 12-1 shows a block diagram of the DMA unit. The DMA channels are independently configurable. Each channel contains a request input (DREQ*n*) and an acknowledge output (DMAACK*n*#). An external peripheral (connected to the DRQ*n* pin) or one of the internal peripherals (asynchronous serial I/O, synchronous serial I/O, or timer control unit) can request DMA service. The DMA configuration register is used to select one of the possible sources. In addition to these hardware request sources, each channel contains a software request register that can be used to initiate software requests. The channels share an end-of-process signal (EOP#). This signal functions as either an input or an open-drain output. EOP# either terminates a transfer (as an input) or signals that a transfer is completed (as an output).



† Alternate pin signals are in parentheses.

A2531-02

Figure 12-1. DMA Unit Block Diagram

12.1.1 DMA Terminology

This section provides a definition of some of the terms used in this chapter to describe the DMA controller.

DMA Process — A DMA process is the execution of a programmed DMA task from beginning to end. Each DMA process requires initial programming by the Intel386 EX processor.

Buffer — A contiguous block of data.

Buffer Transfer — The action required by the DMA to transfer an entire buffer.

Data Transfer — The DMA action in which a group of bytes or words are moved between devices by the DMA controller. A data transfer operation may involve movement of one or many bytes.

Bus Cycle — Access by the DMA to a single byte or word.

Requester — The Requester is the device which requests service by the DMA controller. All of the control signals which the DMA monitors or generates for specific channels are logically related to the requester. Only the requester is considered capable of initiating or terminating a DMA process. The requester may be either I/O or memory and may be the Source or the Destination of the transfer or neither.

Target — The Target is the device with which the Requester wishes to communicate. As far as the DMA process is concerned, the Target is a slave which is incapable of control over the process. The Target may be either I/O or memory, and may be either the Source or the Destination of the transfer.

Source — The Source is the memory or I/O from which data is being read.

Destination — The Destination is the memory or I/O to which data is being written.

12.1.2 DMA Signals

Table 12-1 describes the DMA signals.

Table 12-1. DMA Signals

Signal	Device Pin or Internal Signal	Description
DRQ0 SIO0 RBFDMA0/TXEDMA0 SIO1 TXEDMA1/RBFDMA1 SSIO Transmitter/Receiver TCU Counter 1	Device pin (input) Internal signals	DMA Channel 0 Requests: The SIO channel 0 receiver, SIO channel 0 transmitter, SIO channel 1 receiver, SIO channel 1 transmitter, SSIO transmitter, SSIO receiver, TCU counter 1 output, or an external device can request DMA channel 0 service. These sources are referred to as channel 0 hardware requests. You can also issue channel 0 software requests by writing to the DMA software request register.
DRQ1 SIO1 RBFDMA1/TXEDMA1 SIO0 TXEDMA0/RBFDMA0 SSIO Receiver/Transmitter TCU Counter 2	Device pin (input) Internal signals	DMA Channel 1 Requests: The SIO channel 1 receiver, SIO channel 1 transmitter, SIO channel 0 transmitter, SIO channel 0 receiver, SSIO receiver, SSIO transmitter, TCU counter 2 output, or an external device can request DMA channel 1 service. These sources are referred to as channel 1 hardware requests. You can also issue channel 1 software requests by writing to the DMA software request register.
DACK*n*#	Device pin (output)	DMA Channel *n* Acknowledge: Indicates that channel *n* is ready to service the requesting device. An external device uses the DRQ*n* pin to request DMA service; the DMA uses the DACK*n*# pin to indicate that the request is being serviced.
EOP#	Device pin (input/open-drain output)	End-of-process: *As an input:* Activating this signal terminates a DMA transfer. *As an output:* This signal is activated when a DMA transfer completes.

12.2 DMA OPERATION

The following sections describe the operation of the DMA. See “Register Definitions” on page 12-28 for details on implementing DMA Controller options.

12.2.1 DMA Transfers

The DMA transfers data between a requester and a target. The data can be transferred from the requester to target or vice versa. The target addresses and requester addresses can be located in either memory or I/O space, and data transfers can be on a byte or word basis. The requester can be in external device I/O space, in internal peripheral I/O space, or memory mapped I/O. (Very simply, the requester is the thing that activated DREQ*n*.) An external device or an internal peripheral requests service by activating a channel’s request input (DREQ*n*). A requester in memory requests service through the DMA software request register. The requester either deposits data to or fetches data from the target.

A channel is programmed by writing to a set of requester address, target address, byte count, and control registers. The address registers specify base addresses for the target and requester, and the byte count registers specify the number of bytes that need to be transferred to or from the target. Typically, a channel is programmed to transfer a block of data. Therefore, it is necessary to distinguish between the process of transferring one byte or word (data transfer) and the process of transferring the entire block of data (buffer transfer).

The byte count determines the number of data transfers that make up a buffer transfer. After each data transfer within a buffer transfer, the byte count is decremented (by 1 for byte transfers and by 2 for word transfers) and the requester and target addresses are either incremented, decremented, or left unchanged. When the byte count expires (reaches FFFFFFH), the buffer transfer is complete. If the channel’s end-of-process (EOP#) signal is activated before the byte count expires, the buffer transfer is terminated.

NOTE

Since the buffer transfer is complete when the byte count reaches FFFFFFH, the number of bytes transferred is the byte count + 1.

12.2.2 Bus Cycle Options for Data Transfers

There are two bus cycle options for transferring data, fly-by and two-cycle. Fly-by allows data to be transferred in one bus cycle. It requires that the requester be in external I/O and the target be in memory. The two-cycle option allows data to be transferred between any combination of memory and I/O through the use of a four-byte temporary buffer.

12.2.2.1 Fly-By Mode

The fly-by option performs either a memory write or a memory read bus cycle. A write cycle transfers data from the requester to the target (memory), and a read cycle transfers data from the target (memory) to the requester. When a data transfer is initiated, the DMA places the memory address of the target on the bus and selects the requester by asserting the DACK*n*# signal. The requester then either deposits the transfer data on the data bus or fetches the transfer data off the

data bus, depending on the transfer direction. Since the requester is selected via the DACK*n*# signal the requester address is not meaningful in a fly-by mode transfer.

Support logic (either external or built in to the I/O device) must be designed to monitor the DACK*n*# signal and accordingly generate the correct control signals to the I/O device, since all processor signals are used to access memory. This means that if it is an I/O to memory transfer, this logic generates an I/O read cycle and the processor generates the memory write cycle. If it is a memory to I/O transfer, the logic generates an I/O write cycle and the processor generates the memory read cycle. This way the data is driven by the I/O device and latched by the memory device during an I/O to memory transfer, and driven by the memory device and latched by the I/O device during a memory to I/O transfer.

12.2.2.2 Two-Cycle Mode

The two-cycle option first fills the four-byte temporary buffer with data from the source, then writes that data to the destination. This method allows transfers between any combination of memory and I/O with any combination of data path widths (8- or 16-bit). The amount of data and the data bus widths determine the number of bus cycles required to transfer data. For example, it takes six bus cycles to transfer four bytes of data from an 8-bit source to a 16-bit destination: four read cycles to fill the temporary buffer from the 8-bit source, and two write cycles to transfer the data to the 16-bit destination.

A buffer transfer can complete, be terminated, or be suspended before the temporary buffer is filled from the source. If the buffer transfer completes or is terminated before the temporary buffer is filled, the DMA writes the partial data to the destination. When a requester suspends a buffer transfer, the contents of the partially filled temporary buffer are stored until the transfer is restarted. At this point, the DMA performs read cycles until the buffer is full, then performs write cycles to transfer the data to the destination.

12.2.2.3 Programmable DMA Transfer Direction

The relationship between Requester, Target, Source, and Destination is determined by the programmable DMA transfer direction. The transfer directions are defined as Write, Read, or Verify. The following table describes which operations are being performed by the Requester and Target for each transfer direction. In this table, the device being read is the Source, and the device being written is the Destination. The Verify cycle is used to perform a data read only. No write cycle is indicated or assumed in a Verify cycle. The Verify cycle is useful for validating block fill operations. An external comparator must be provided to do any comparisons on the data read.

Table 12-2. Operations Performed During Transfer

	Read	Write	Verify
Requester	Read	Write	Read
Target	Write	Read	Read

A special case not indicated in this table is when the Requester is neither the Source nor Destination. One example of this case would be when the DMA is being used to transfer data from one memory or I/O location to another, and one of the timer outputs is being used to initiate that transfer. In this case, the timer output would be selected as the DMA request source (using the

DMACFG register), but the Requester address registers would be programmed with one of the memory addresses. It doesn't really matter which memory is the Requester and which is the Target, as long as the transfer direction is set to provide the correct Source and Destination.

12.2.2.4 Ready Generation For DMA Cycles

DMA cycles are identical to any other type of memory or I/O cycles in terms of how they are completed. A valid READY# must be sampled at the end of the last T2 state in order to complete a DMA Read or Write cycle. This READY# may be generated externally, or internally using the appropriate chip select unit (see Chapter 14, "CHIP-SELECT UNIT" for a description of generating READY# internally).

12.2.2.5 DMA Usage of the 4-Byte Temporary Register

Each DMA channel has a 4-byte temporary FIFO register used for temporary data storage during two cycle transfers. The way the DMA channel fills and empties this register depends on the data transfer mode, the bus sizes of the source and destination, and the data transfer direction. The following describes how the Temporary Register is filled and emptied for the Read and Write Transfer Directions.

Filling the Temporary Register:

Read Cycle In a Read Cycle data is transferred from the Requester to the Target. Each request (DREQ*n*) in a Read Cycle results in the DMA transferring a byte (if requester is an 8-bit device) or a word (if the requester is a 16-bit device) from the Source (Requester) to the temporary register. This continues until either the Temporary Register is full, or until the byte count or terminal count is reached.

Write Cycle In a Write Cycle data is transferred from the Target to the Requester. The first request (DREQ*n*) initiates a fill of the temporary register (four byte reads of the Target if the Target is 8-bit, or two word reads if it is 16-bit). The buffer is considered full if either four bytes have been stored, or if less than four bytes, the byte count or terminal count has been reached.

Emptying the Temporary Register:

Read Cycle Once the Temporary Register has been filled the DMA empties it by doing four byte write cycles (if Target is 8-bit), or two word write cycles (if Target is 16-bit). This is done in a burst-type fashion since all four requests have already occurred. The byte counter is decremented after each write has occurred.

Write Cycle Once the Temporary Register has been filled the DMA does a single write cycle transferring the first byte (if Requester is 8-bit), or the first word (if Requester is 16-bit). This first write cycle happens immediately after the buffer has been filled. Each subsequent request (DREQ*n*) results in another write cycle transferring another byte or word from the Temporary Register to the Requester. This continues until either the Temporary Register is empty, or byte count or terminal count has been reached.

Figures 12-2 and 12-3 are simple diagrams of how the Temporary Register is filled and emptied for a Read DMA cycle and a Write DMA cycle.

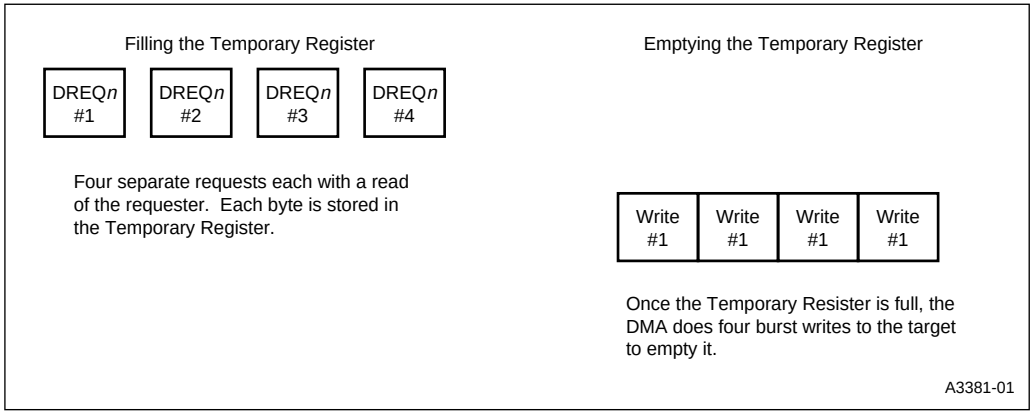


Figure 12-2. DMA Temporary Buffer Operation for a Read Transfer

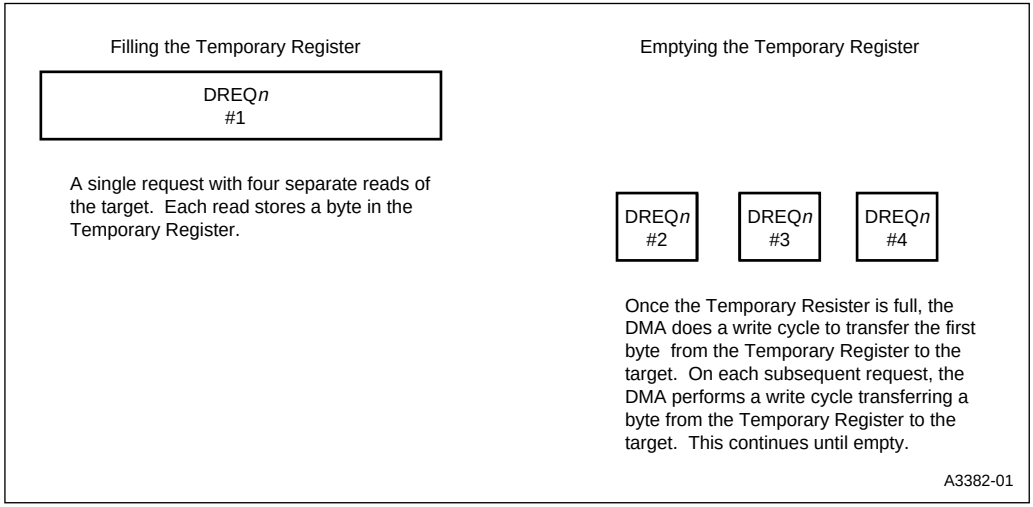


Figure 12-3. DMA Temporary Buffer Operation for A Write Transfer

12.2.3 Starting DMA Transfers

Internal I/O, external I/O, or memory can request DMA service. The internal I/O requesters (the asynchronous serial I/O, synchronous serial I/O, and timer control units) are internally connected to the DMA request inputs. You must connect an external I/O source to the DMA DRQ*n;* when you are using fly-by mode, you must also connect an external I/O source to the DACK*n*# signals. In addition, memory mapped I/O peripherals may use DRQ*n*/DACK*n*#. DACK*n*# is active during the entire fly-by mode transfer, but during a two-cycle mode transfer it is only active during the access to the requester. These sources make up the DMA hardware request sources. The DMA unit also contains a software request register that allows you to generate software DMA requests. This allows memory-to-memory transfers. Figure 12-4 shows the timing for the start of a DMA transfer.

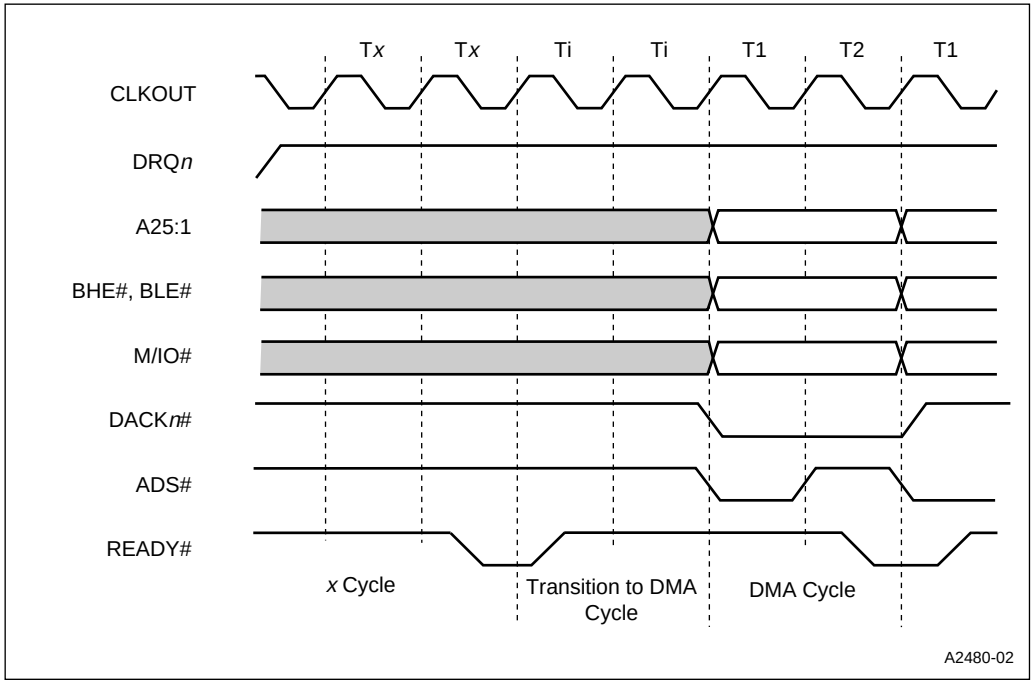


Figure 12-4. Start of a Two-cycle DMA Transfer Initiated by DRQ*n*

12.2.4 Bus Control Arbitration

The bus arbiter services bus control requests from the two DMA channels, an external device, and the refresh control unit. The DMA channels interface with the bus arbiter through its DMA channel request signals (DREQ*n*) and its DMA channel acknowledge signals (DMAACK*n*#). Other external bus masters interface with the bus arbiter through similar request and acknowledge signals, the HOLD and HOLDA signals respectively. The refresh control unit gains bus control through an internal Refresh request. The REFRESH# status pin indicates that the Refresh Control Unit has gained bus control and that a valid refresh cycle is being executed. After receiving a bus

control request, the bus arbiter services these requests by issuing an internal hold signal requesting control of the bus from the core. The core returns an internal hold acknowledge signal to the arbiter when bus ownership is granted. The arbiter then issues an acknowledge signal to the requesting device.

Refresh requests always have the highest priority, while the priority structure of the other three requests is configurable. By default, DMA channel 0 requests have the next highest priority, followed by DMA channel 1 requests, and external bus master requests There are two methods for changing the priority of the DMA and external bus requests, low-priority selection or rotation. The priority requests are programmed in the DMACMD2 register (see Figure 12-24). The low-priority selection method allows you to assign a particular request to the lowest priority level. With the rotation method, a request is automatically assigned to the lowest priority level after it gains bus control. The rotation method allows requesting devices to share the system bus more evenly. With both methods, the other request priority levels are adjusted in a circular manner (see Figure 12-5).

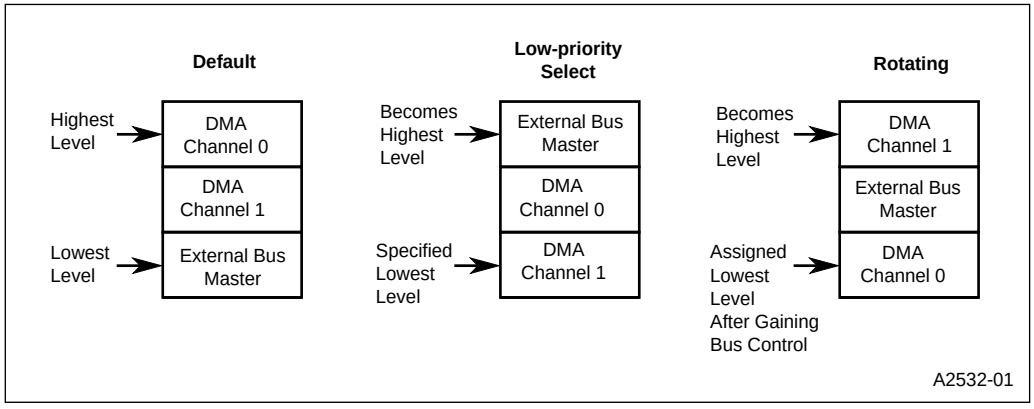


Figure 12-5. Changing the Priority of the DMA Channel and External Bus Requests

12.2.5 Ending DMA Transfers

When a channel's byte count expires, the buffer transfer is complete and the end-of-process (EOP#) output is activated (Figure 12-6). A buffer transfer can be terminated before the byte count expires by activating the EOP# input. The channel can sample the EOP# input synchronously or asynchronously. With synchronous sampling, the channel samples EOP# at the end of the last state of every data transfer. With asynchronous sampling, the DMA samples the inputs at the beginning of every state of requester access, then waits until the end of the state to act on the input. Figure 12-7 illustrates terminating a buffer transfer by activating the EOP# input; the figure shows both asynchronous and synchronous EOP# sampling. EOP# sampling is programmed in the DMACMD2 register (Figure 12-24).

Terminating a buffer transfer by deasserting DREQ*n* can also be done either synchronously or asynchronously. The effect is identical to that of synchronous or asynchronous sampling of EOP#. When DREQ*n* is used to terminate a DMA transfer in asynchronous mode, DREQ*n* must be sampled inactive one CLKOUT before READY#. In synchronous mode it must be sampled inactive at the same time as READY#. When DREQ*n* is sampled active in either of the above cases another DMA cycle is executed (depending on operating mode).

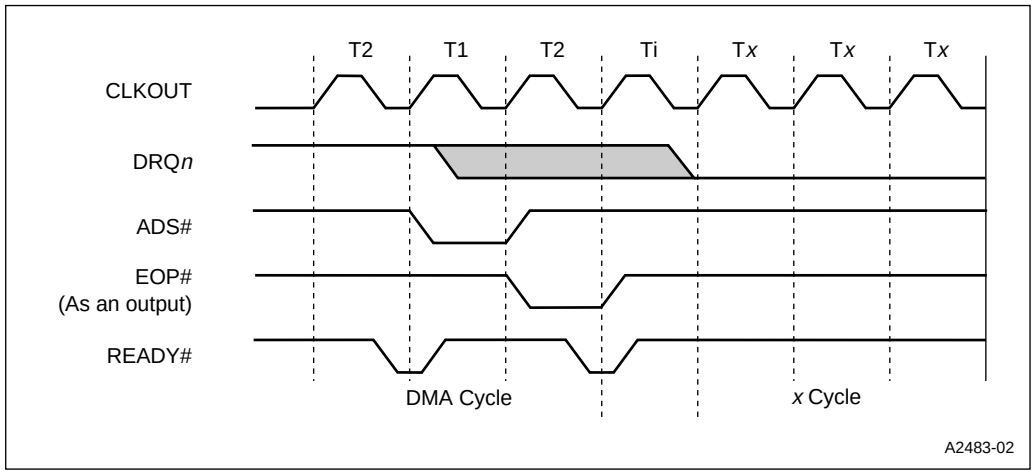


Figure 12-6. Buffer Transfer Ended by an Expired Byte Count

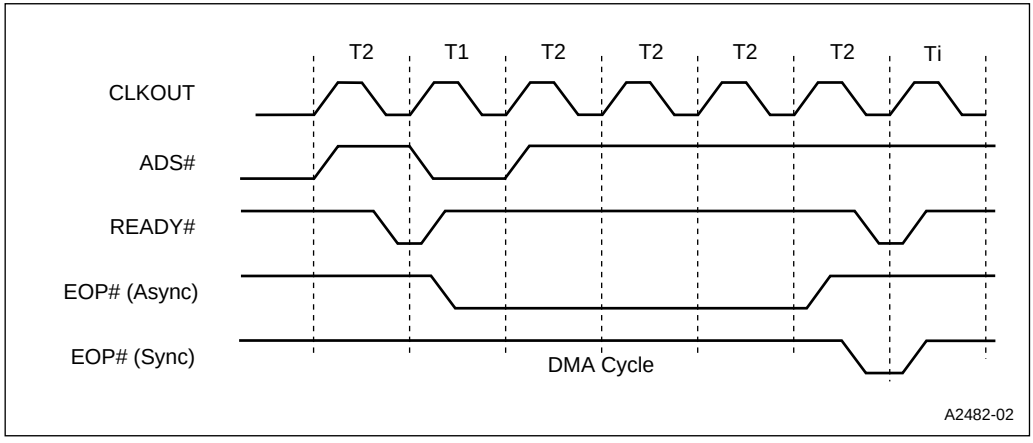


Figure 12-7. Buffer Transfer Ended by the EOP# Input

12.2.6 Buffer-transfer Modes

After a buffer transfer is completed or terminated, a channel can either become idle (require reprogramming) or reprogram itself and begin another buffer transfer after it is initiated by a hardware or software request. The DMA's three buffer-transfer modes (single, autoinitialize, and chaining) determine whether a channel becomes idle or is reprogrammed after it completes or terminates a buffer transfer.

12.2.6.1 Single Buffer-Transfer Mode

By default (single buffer-transfer mode), the DMA transfers a channel's buffer only once. When the entire buffer of data has been transferred, the channel becomes idle and must be reprogrammed before it can perform another buffer transfer. The single buffer-transfer mode is useful when you know the exact amount of data to be transferred and you know that there is time to reprogram the channel (requester and target addresses and byte count) before another buffer of data needs to be transferred.

12.2.6.2 Autoinitialize Buffer-Transfer Mode

When programmed for the autoinitialize buffer-transfer mode, the DMA automatically reloads the channel with the original transfer information (the requester and target addresses and the byte count) when the transfer completes. The channel then repeats the original buffer transfer. The autoinitialize buffer-transfer mode is useful when you need to transfer a fixed amount of data between the same locations multiple times.

12.2.6.3 Chaining Buffer-Transfer Mode

This mode is similar to the autoinitialize buffer-transfer mode, in that the DMA automatically reprograms the channel after the current buffer transfer is complete. The difference is that the autoinitialize buffer-transfer mode uses the original transfer information, while the chaining buffer-transfer mode uses new transfer information. While a channel is performing a chaining buffer transfer, you write new requester and target addresses and a new byte count to it. This prepares the channel for the next buffer transfer, without affecting the current buffer transfer. When the channel completes its current buffer transfer, the channel is automatically programmed with the new transfer information that you wrote to it. The chaining buffer-transfer mode is useful when you need to transfer data between multiple requesters and targets.

NOTE

If a channel does not contain new transfer information at the end of its buffer transfer, the channel becomes idle, ending the chaining process; it must be reprogrammed before it can perform another buffer transfer.

The Chaining Buffer Transfer Mode is entered from the Single Buffer Transfer Mode. The mode registers should be programmed first, with all of the transfer modes defined as if the channel were to operate in the Single Buffer Transfer Mode. The channel's base and current registers are then loaded. When the channel has been set up in this way and the chaining interrupt service routine is in place, the Chaining Buffer Transfer Mode can be entered by programming the Chaining register. "Chaining Register (DMACHR)" on page 12-47 describes this process.

The DMAINT signal is active immediately after the Chaining Process has been entered, as the channel then perceives the Base Registers to be empty and in need of reloading. It is important to have the interrupt service routine in place at the time the Chaining Process is entered. The interrupt request is removed when the most significant byte of the Base Target Address is loaded.

NOTE

Since the most significant byte of the Base Target Address only exists in 0FXXXH I/O address space, the Chaining Buffer Transfer Mode cannot be used in a DOS Compatible-only mode.

The interrupt occurs again when the first buffer transfer expires and the Current Registers are loaded from the Base Registers. The cycle continues until the Chaining Process is disabled, or the host fails to respond to DMAINT before the Current Buffer expires.

Exiting the Chaining Process can be done by resetting the Chaining Mode Register. If an interrupt is pending for the channel when the Chaining Register is reset, the interrupt request is removed. The Chaining Process can be temporarily disabled by setting the channel's mask bit in the Mask Register.

The interrupt service routine for DMAINT has the responsibility of reloading the Base Register as necessary. It should check the status of the channel to determine the cause of the channel expiration, etc. It should also have access to operating system information regarding the channel, if any exists. The DMAINT service routine should be capable of determining whether the chain should be continued or terminated and act on that information.

NOTE

The chaining buffer-transfer mode is not useful with block transfer mode since the CPU must be able to get control of the bus before the end of the "block" in order to reprogram the new values into the DMA registers. Since block transfer mode locks out any other bus requests (except refresh) the processor cannot regain control of the bus until the entire block has been transferred.

12.2.7 Data-transfer Modes

There are three data-transfer modes (single, block, and demand) that determine how the bytes or words that make up a buffer of data are transferred. The DMAMOD1 register is used to select a channel's data transfer mode.

Single Mode — A channel request causes one byte or word (depending on the selected bus widths) to be transferred. Single mode requires a channel request for every data transfer within a buffer transfer.

Block Mode — A channel request causes the entire buffer of data to be transferred.

Demand Mode — The amount of buffer data (bytes or words) that the channel transfers depends on how long the channel request input is held active. In this mode, the channel continues to transfer data while the channel request input is held active; when the signal goes inactive, the buffer

transfer is suspended and the channel waits for the request input to be reactivated before it continues.

12.2.7.1 Single Data-transfer Mode

In single data-transfer mode, a DMA request causes the channel to gain bus control. The channel transfers data (a byte or a word), decrements the buffer byte count (by 1 for byte transfers and 2 for word transfers), then relinquishes bus control. The channel will then Autoinitialize if it has been programmed to do so. The channel continues to operate in this manner until the buffer transfer is complete or terminated. In this mode, the channel gives up bus control after every data transfer and must regain bus control (through priority arbitration) before every data transfer. The channel's buffer-transfer mode determines whether the channel becomes idle or is reprogrammed after a buffer transfer completes or is terminated.

The single data-transfer mode is compatible with all of the buffer-transfer modes. The following flowcharts show the transfer process flow for a channel programmed for single data-transfer mode with each buffer-transfer mode: single (Figure 12-8), autoinitialize (Figure 12-9), and chaining (Figure 12-10).

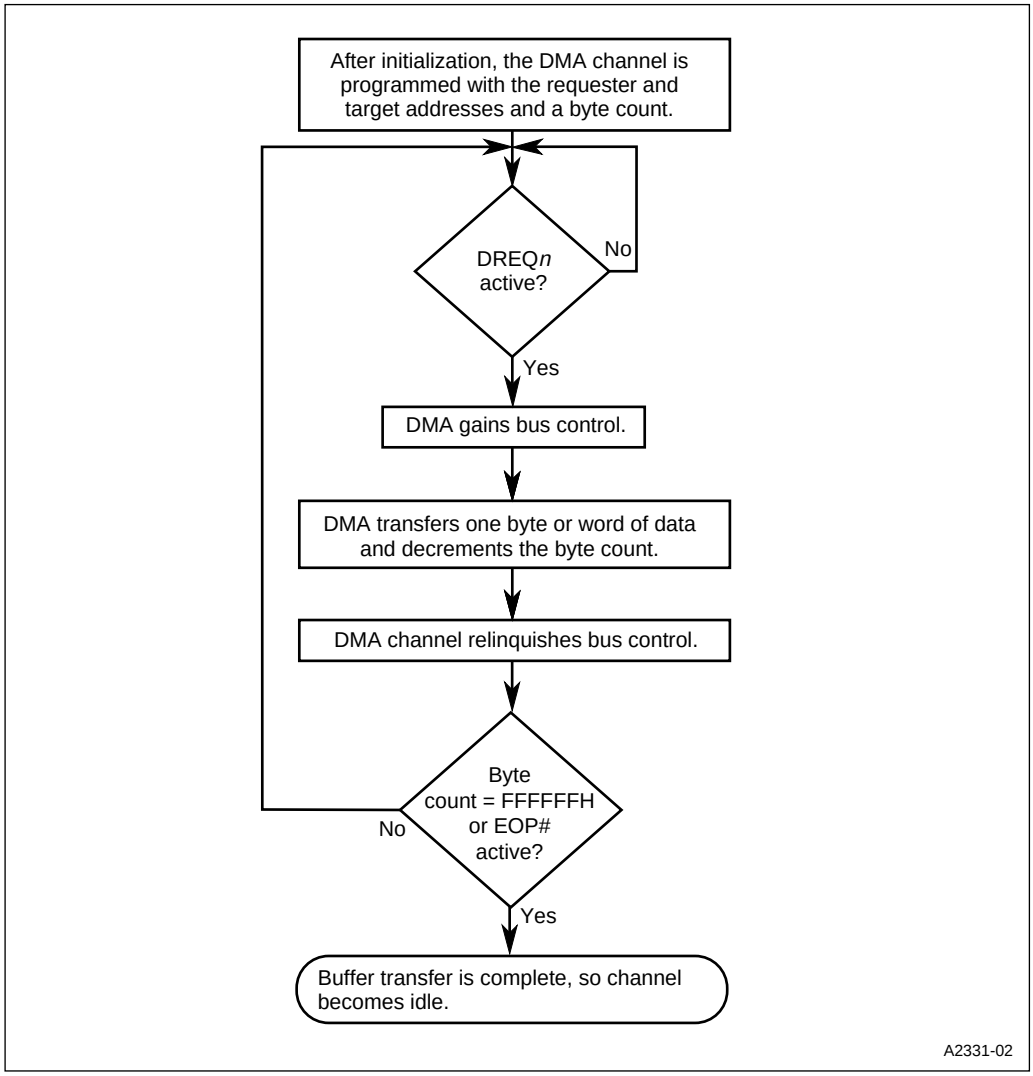


Figure 12-8. Single Data-transfer Mode with Single Buffer-transfer Mode

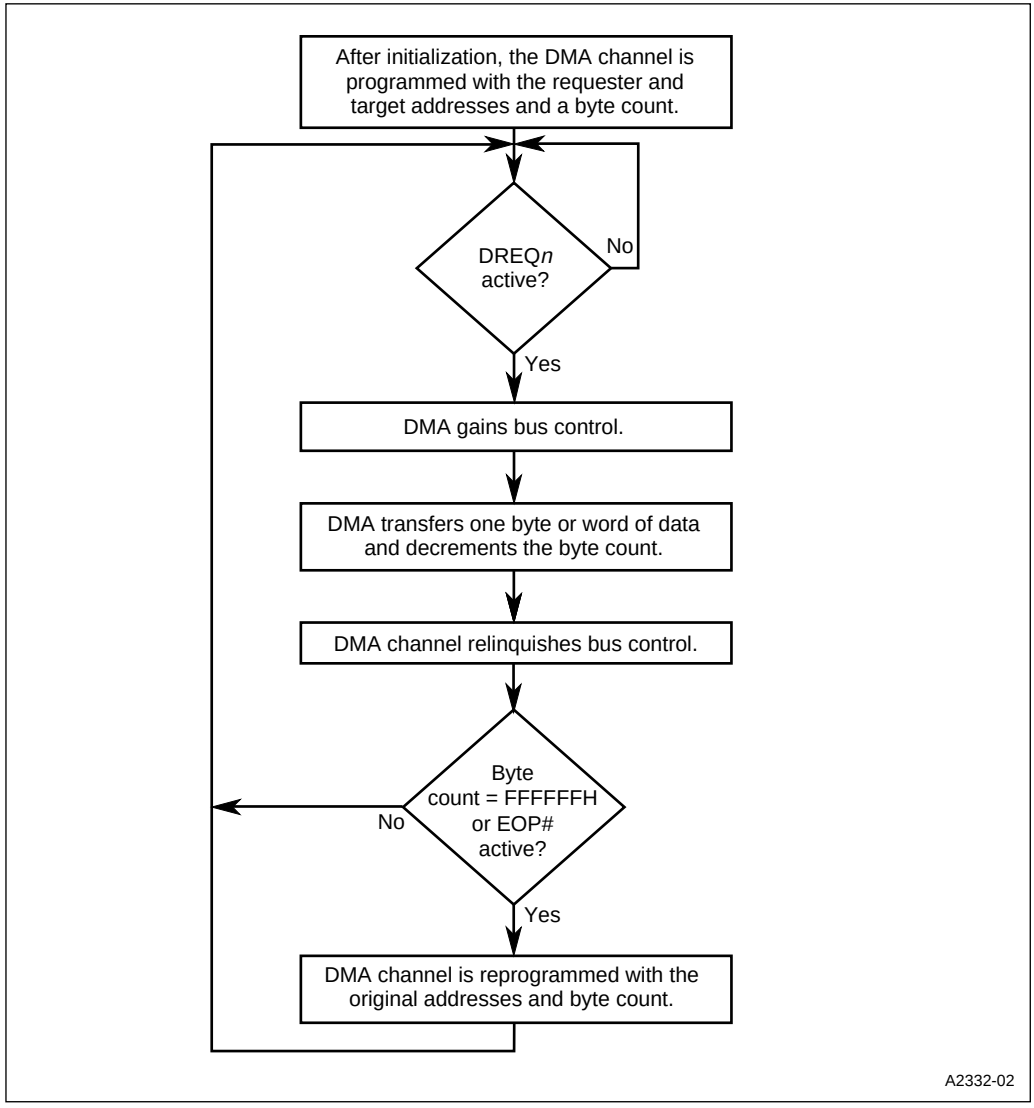


Figure 12-9. Single Data-transfer Mode with Autoinitialize Buffer-transfer Mode

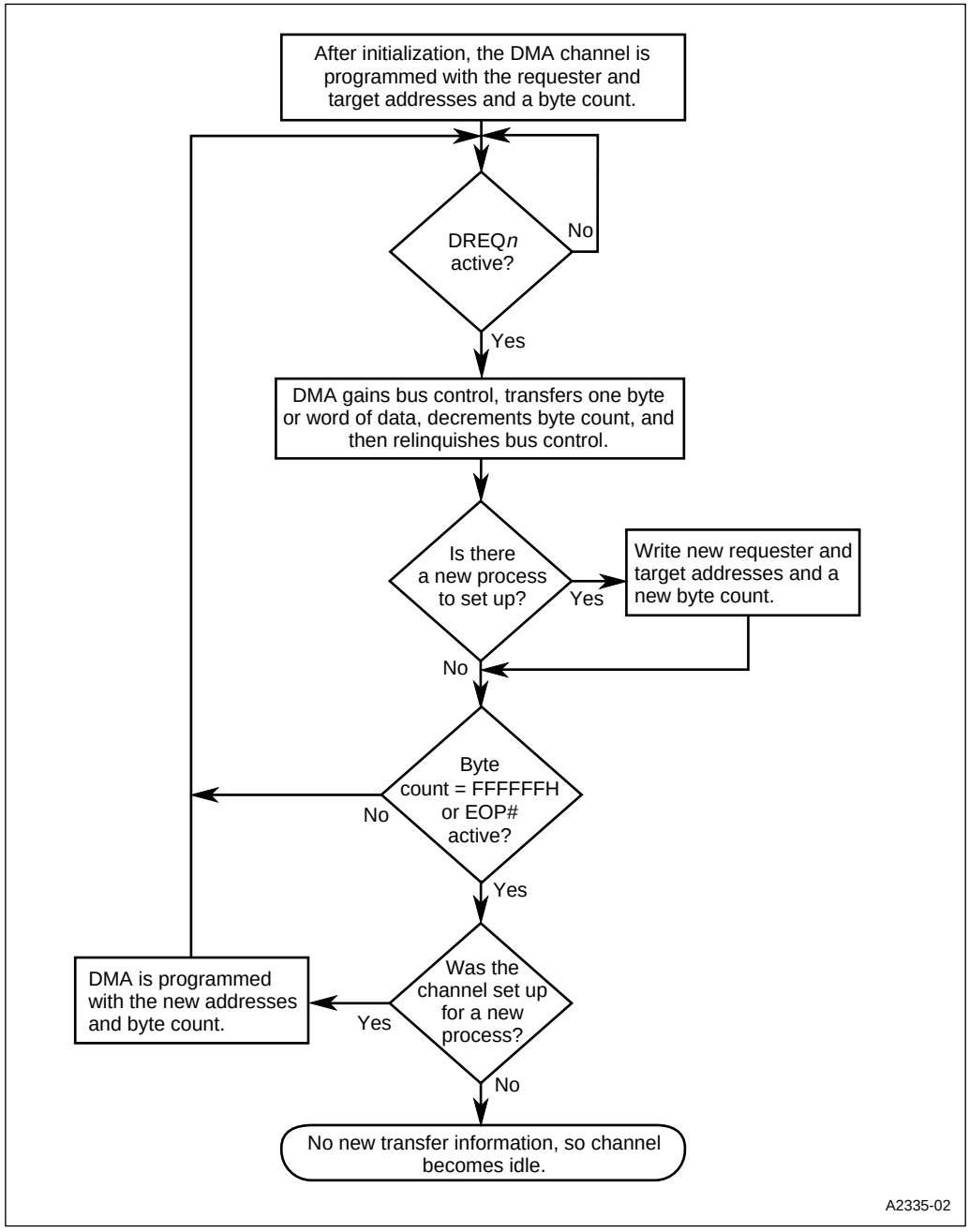


Figure 12-10. Single Data-transfer Mode with Chaining Buffer-transfer Mode

12.2.7.2 Block Data-transfer Mode

In block data-transfer mode, a channel request initiates a buffer transfer. The channel gains bus control, then transfers the entire buffer of data. The DRQ*n* signal only needs to be held active until DACK*n*# is active.

NOTE

Block mode, unlike the single mode, only gives up control of the bus for DRAM refresh cycles.

As with single mode, the channel's buffer-transfer mode determines whether the channel becomes idle or is reprogrammed after the buffer transfer completes or is terminated.

The block data-transfer mode is compatible with the single and autoinitialize buffer-transfer modes, but not with the chaining buffer-transfer mode. The chaining buffer-transfer mode requires that the transfer information for the next buffer transfer be written to the channel before the current buffer transfer completes. This is impossible with block data-transfer mode, because the channel only relinquishes control of the bus for DRAM refresh cycles during the buffer transfer. The following flowcharts show the transfer process flow for a channel programmed for the block data-transfer mode with the single (Figure 12-11) and autoinitialize (Figure 12-12) buffer-transfer modes.

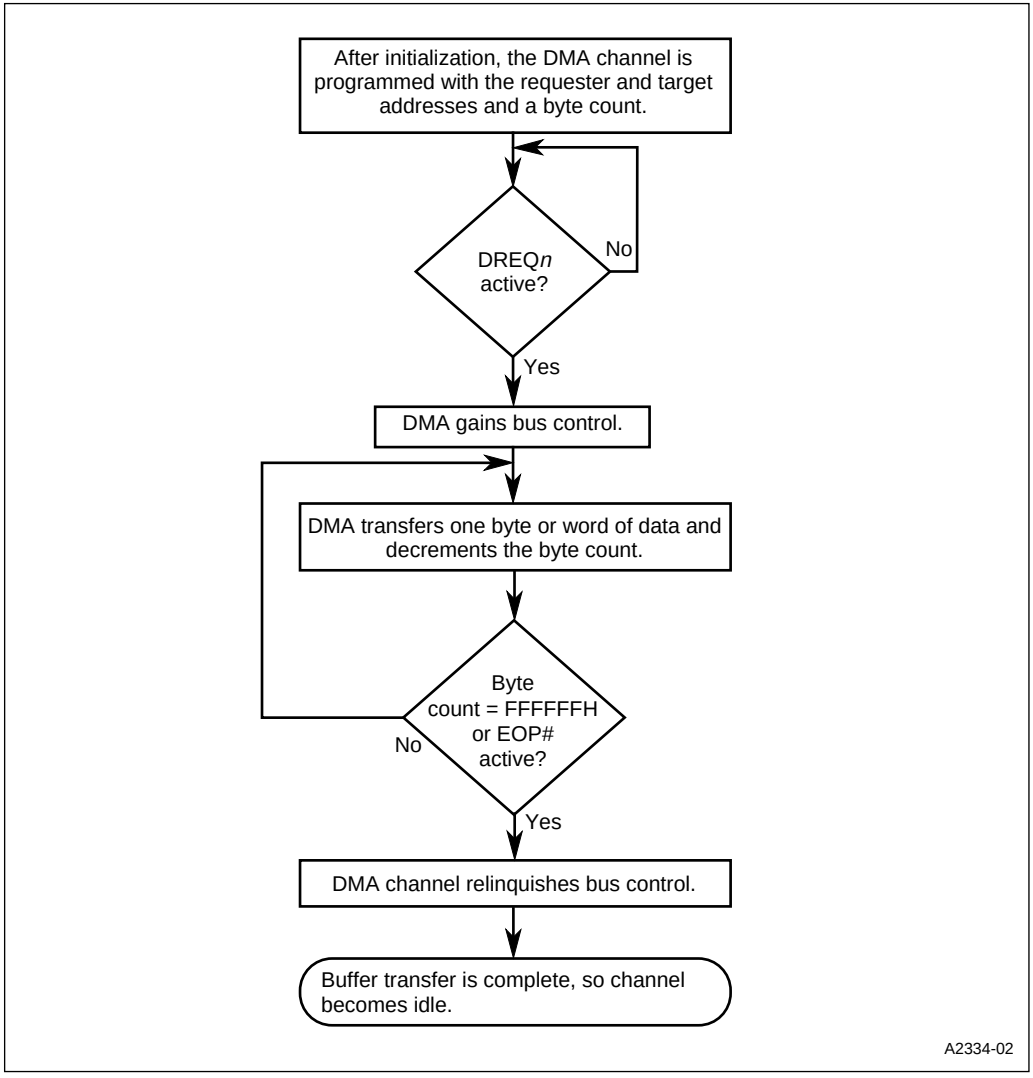


Figure 12-11. Block Data-transfer Mode with Single Buffer-transfer Mode

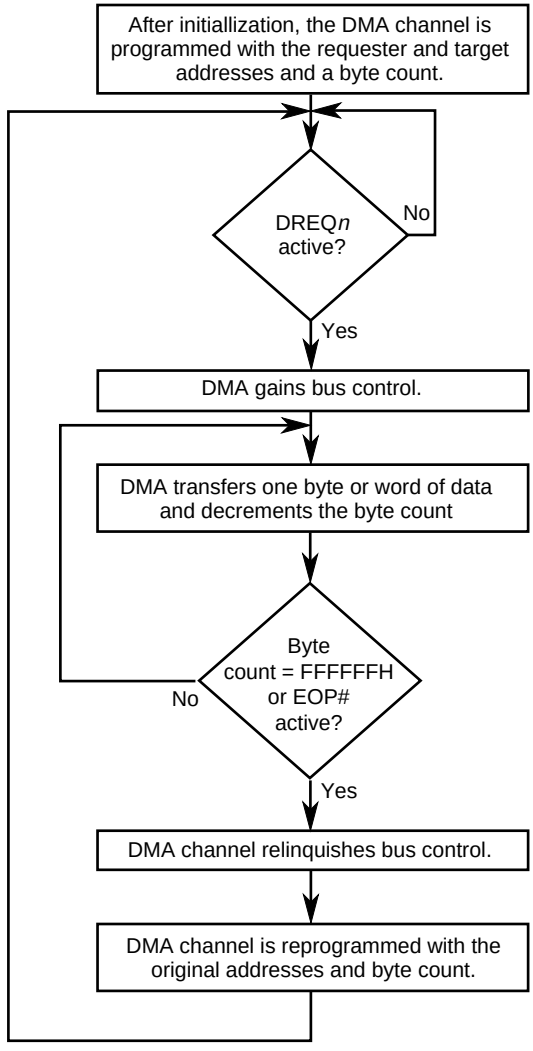


Figure 12-12. Block Data-transfer Mode with Autoinitialize Buffer-transfer Mode

12.2.7.3 Demand Data-transfer Mode

In demand data-transfer mode, a channel request initiates a buffer transfer. The channel gains bus control and begins the buffer transfer. As long as the request signal (DRQ*n)* remains active, the channel continues to perform data transfers. When the DRQ*n* signal goes inactive, the channel completes its current bus cycle and relinquishes bus control, suspending the buffer transfer. In this way, the demand mode allows peripherals to access memory in small, irregular bursts without wasting bus control time. As in other data-transfer modes, a buffer transfer is completed when the buffer's byte count expires or is terminated if the EOP# input is activated. At this point, the channel's buffer-transfer mode determines whether the channel becomes idle or is reprogrammed.

Since DRQ*n* going inactive suspends a buffer transfer, the channel continually samples DRQ*n* during a demand buffer transfer. During a buffer transfer**,** the channel can sample DRQ*n* synchronously or asynchronously (it always samples DRQ*n* asynchronously at the start of a buffer transfer). With synchronous sampling, the channel samples DRQ*n* at the end of the last state of every data transfer. With asynchronous sampling, the channel samples DRQ*n* at the beginning of every state, then waits until the end of the state to act on the input. See Figure 12-13. The DRQ*n* sampling is programmed in the DMACMD2 register (Figure 12-24).

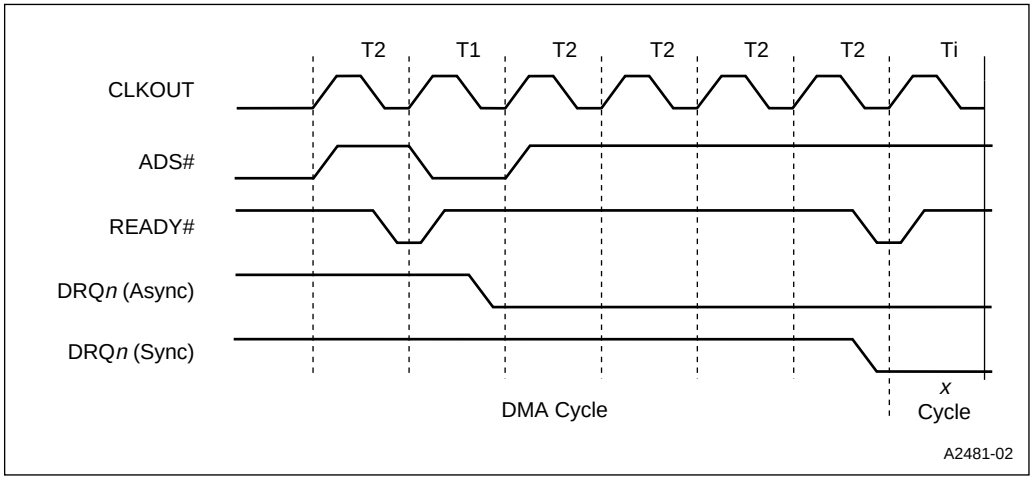


Figure 12-13. Buffer Transfer Suspended by the Deactivation of DRQ*n*

The demand data-transfer mode is compatible with all of the buffer-transfer modes. The following flowcharts show the transfer process flow for a channel programmed for the demand data-transfer mode with each buffer-transfer mode: single (Figure 12-14), autoinitialize (Figure 12-15), and chaining (Figure 12-16).

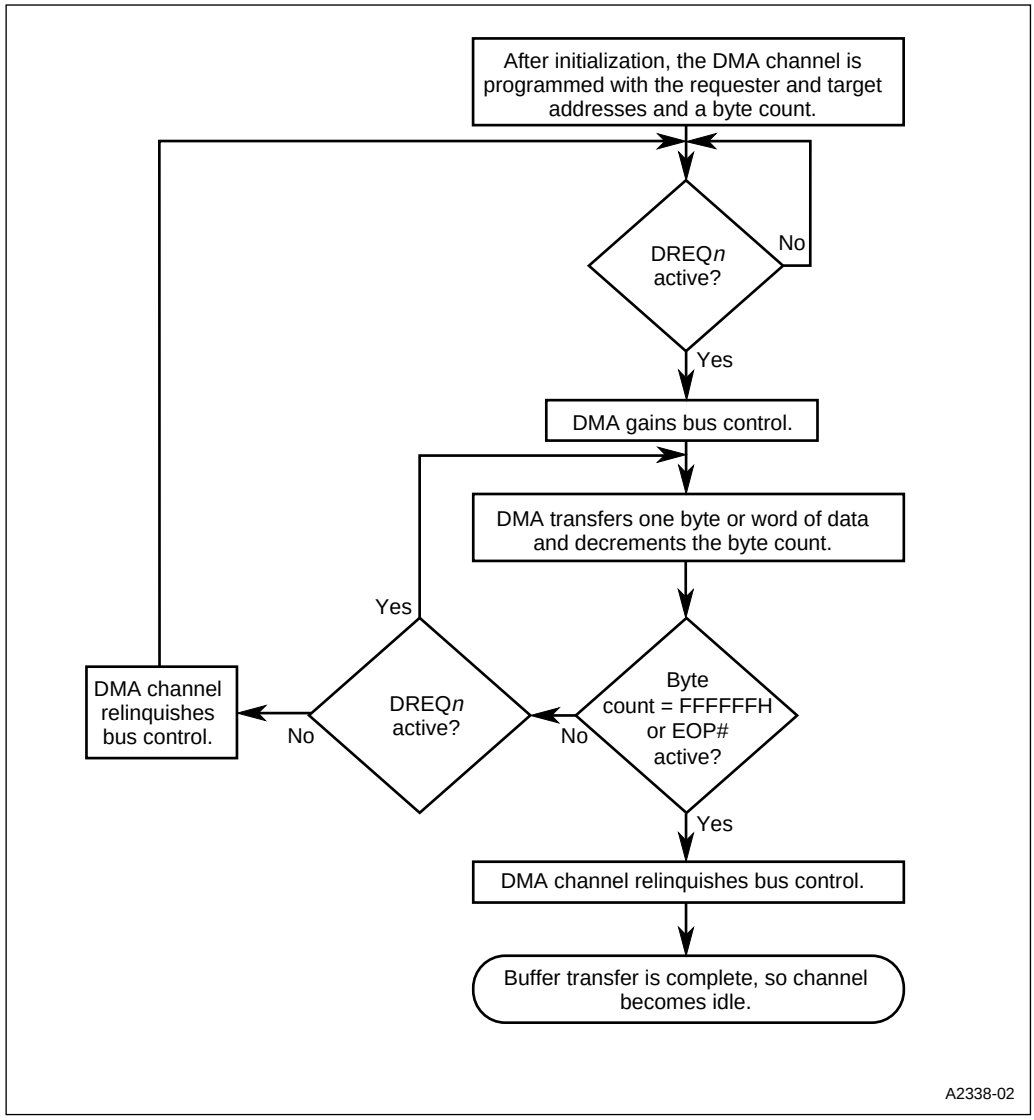


Figure 12-14. Demand Data-transfer Mode with Single Buffer-transfer Mode

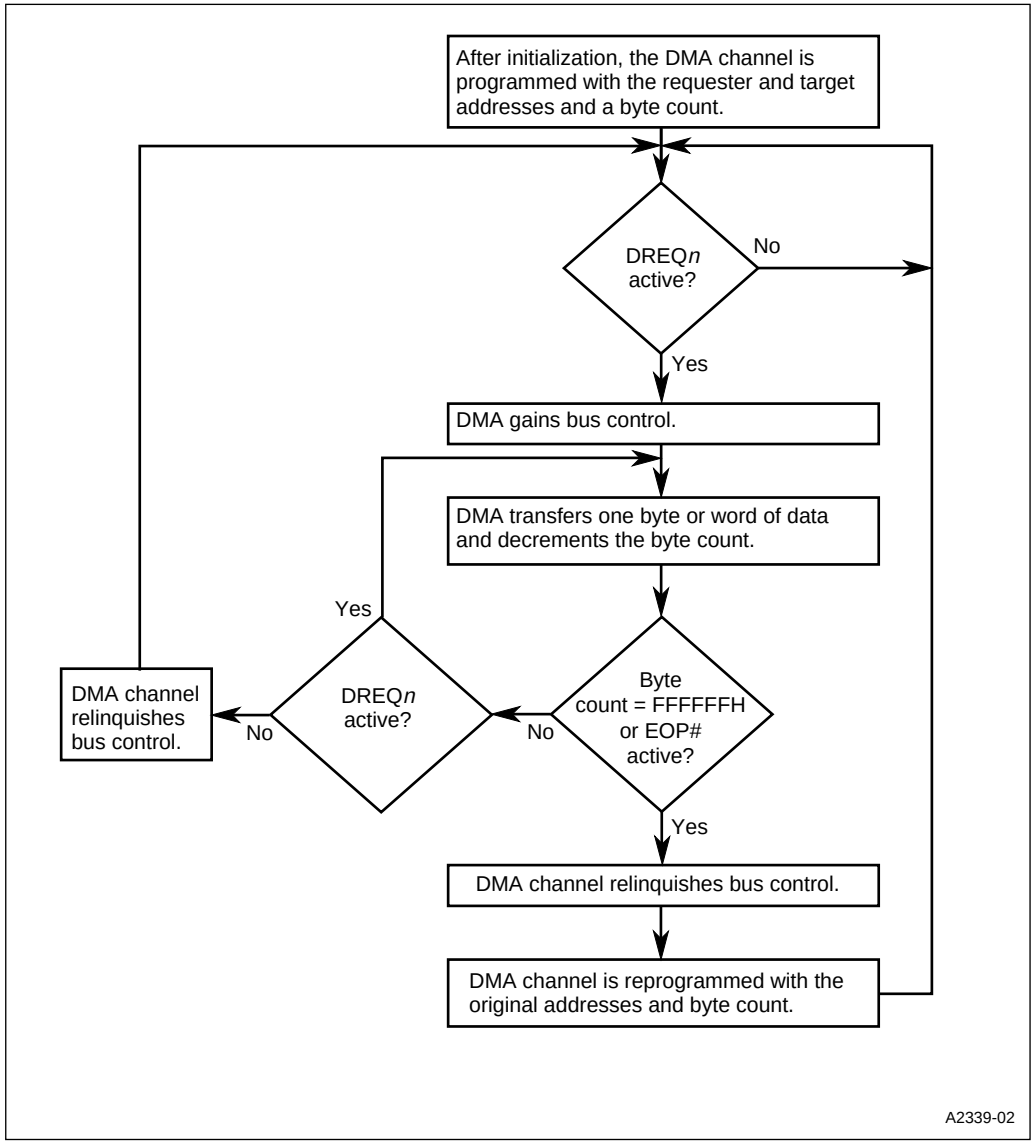


Figure 12-15. Demand Data-transfer Mode with Autoinitialize Buffer-transfer Mode

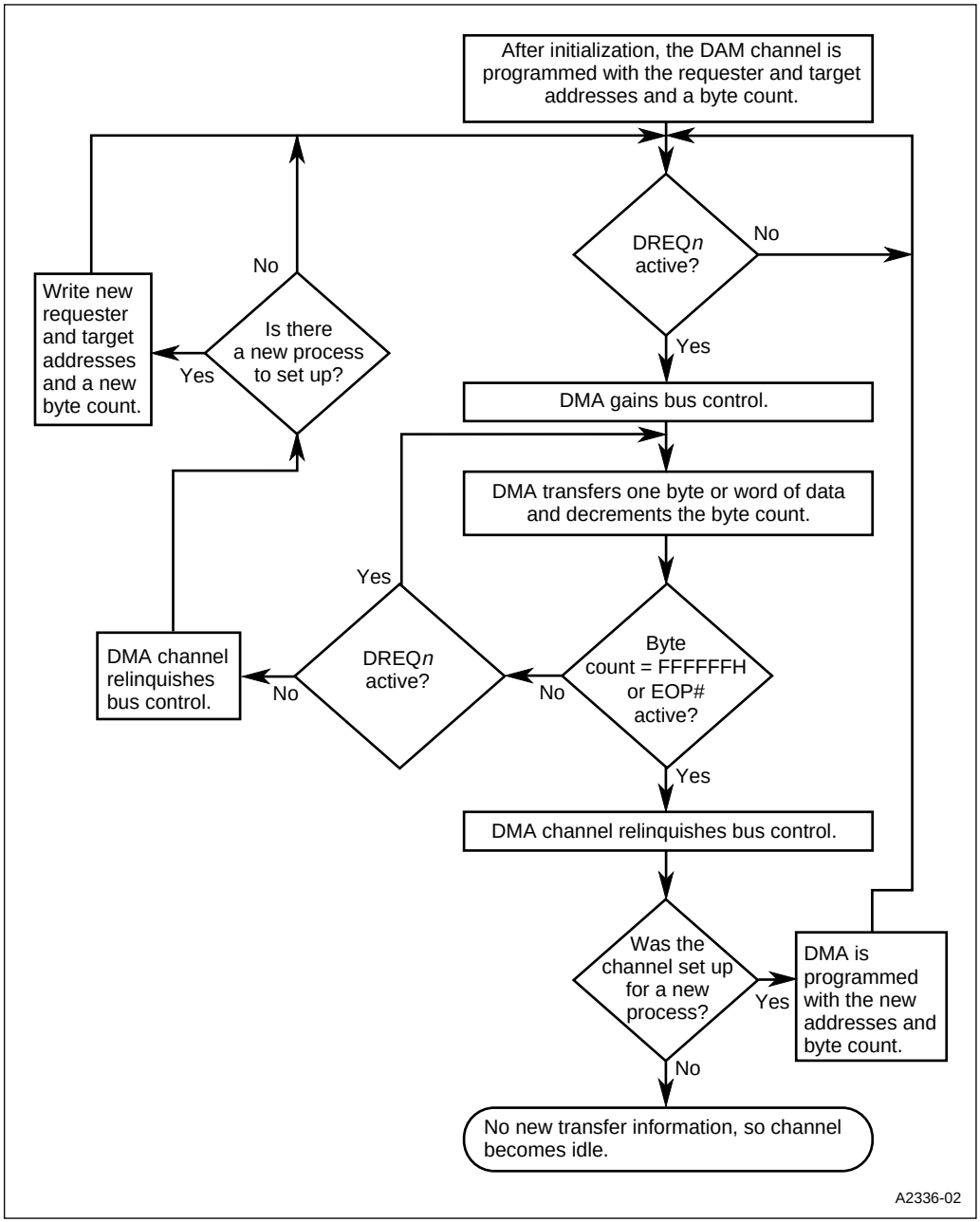


Figure 12-16. Demand Data-transfer Mode with Chaining Buffer-transfer Mode

12.2.8 Cascade Mode

Cascade mode allows an external 8237A or another DMA-type device to gain bus control. A cascaded device requests bus control by holding a channel's request input (DRQ*n)* active. Once granted bus control, the cascaded device remains bus master until it relinquishes bus control by deactivating DRQ*n*.

If a refresh request occurs while a cascaded device has bus control, the cascaded device must deassert its request or the refresh cycle will be missed. The following steps take place in response to a refresh request.

1. The channel deasserts its acknowledge signal (DACK*n*#) to the cascaded device.
   — At this point, the cascaded device should relinquish bus control by removing DRQ*n*.
2. As soon as DRQ*n* is removed, the refresh cycle is started.
   — At this point, if the cascaded device wants to regain bus control after the refresh cycle, it must reassert DRQ*n*.
3. If the cascaded device has reasserted DRQ*n* when the refresh cycle is complete, the channel reasserts DACK*n*#, giving bus control back to the cascaded device without bus priority arbitration.

The following flowchart (Figure 12-17) shows this process flow.

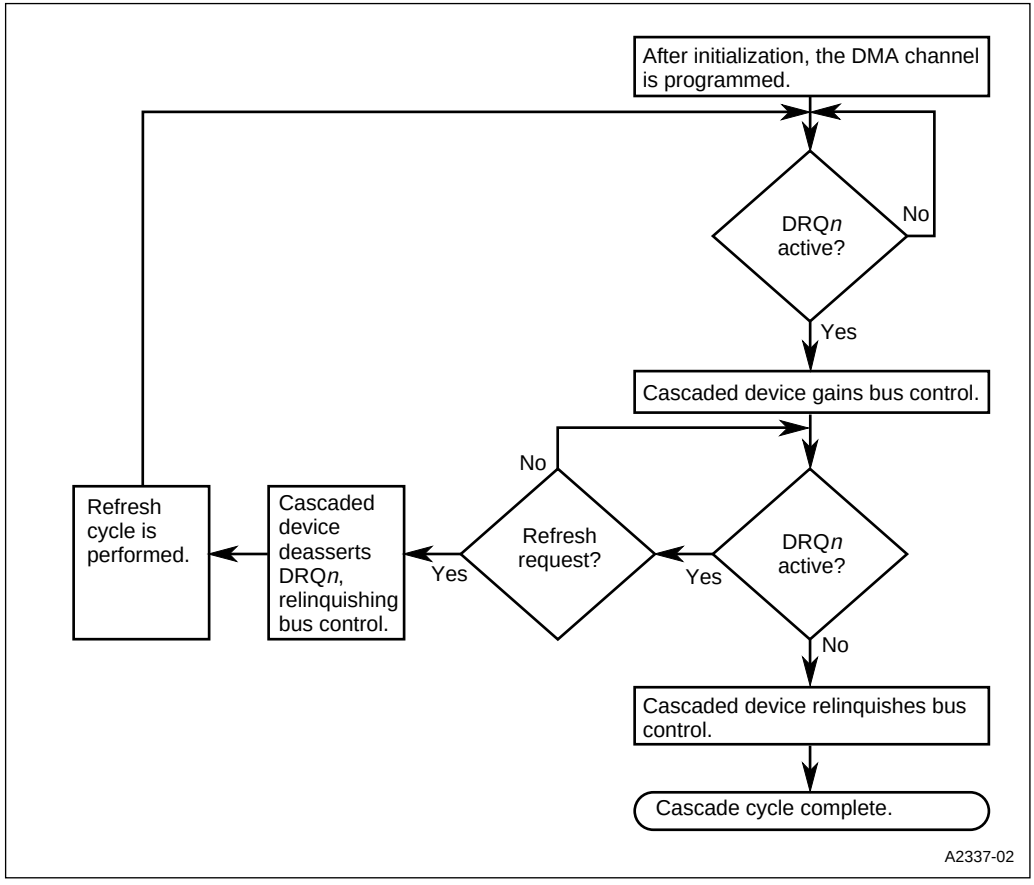


Figure 12-17. Cascade Mode

12.2.9 DMA Interrupts

Each channel contains two interrupt causing signals, chaining status and transfer complete. When a channel is configured for the chaining buffer-transfer mode, the chaining status signal indicates that the channel has started its buffer transfer and new transfer information can be written without affecting the current buffer transfer. Once activated, the chaining status signal remains active until the most significant byte of the base target address is written, or resetting the chaining enable bit.

The transfer complete status signal indicates that the channel has finished a buffer transfer — either the channel's byte count has expired or the buffer transfer was terminated by an EOP# input. DMACLRTC clears the DMAINT signal going to the Interrupt Control Unit. DMACLRTC is executed by writing to location F01EH; the data written to the location is immaterial — writing any data to the location causes the DMA to deactivate the transfer complete status signal.

The four interrupt source signals (two per channel) are internally connected (ORed) to the interrupt request output (DMAINT). When an interrupt from DMAINT is detected, you can determine which signal caused the request by reading the DMA interrupt status register.

12.2.10 8237A Compatibility

Although the DMA is an enhancement over the 8237A, you can configure it to operate in an 8237A-compatible mode. A list of the features common to the DMA and 8237A and a list of DMA enhancements follow.

Features common to the DMA and 8237A:

- Data-transfer modes (single, block, and demand)
- Buffer-transfer modes (single and autoinitialize)
- Fly-by data transfer bus cycle option
- Programmed via 8-bit registers
- Transfers between memory and I/O (target must be in memory and requester must be external)

DMA enhancements:

- Chaining buffer-transfer mode
- Two-cycle data transfer bus cycle option (provides byte assembly and allows memory-to-memory transfers using only one channel)
- Transfers between any combination of memory and I/O
- Address registers for both the target and the requester; addresses can be incremented, decremented, or left unchanged during a buffer transfer

A channel is configured for 8237A compatibility by enabling only the common features and limiting the byte count and the target address modification capability. The 8237A uses a 16-bit target address and a 16-bit byte count, while the DMA uses a 26-bit target address and a 24-bit byte count. Therefore, for compatibility, the DMA contains an overflow register that allows you to configure the target and byte count so that only the lower 16 bits are modified during buffer transfers. With this configuration, the upper byte count bits are ignored; the byte count expires when it is decremented from 0000H to FFFFH (16-bit versus 24-bit rollovers).

12.3 REGISTER DEFINITIONS

Table 12-3 lists the registers associated with the DMA unit, and the following sections contain bit descriptions for each register.

Table 12-3. DMA Registers (Sheet 1 of 3)

Register	Expanded Address	PC/AT* Address	Description
PINCFG (read/write)	F826H	—	Pin Configuration: Connects the DMA channel acknowledge (DMAACK0#, DMAACK1#) and end-of-process signals to package pins DACK0#, DACK1# and EOP#, respectively.
DMACFG (read/write)	F830H	—	DMA Configuration: Determines which signal is connected to the DMA channel request inputs (DREQ*n*). Masks the channel acknowledge signals (DMAACK0#, DMAACK1#), which is useful when using internal requesters.
DMACMD1 (write only)	F008H	0008H	DMA Command 1: Simultaneously enables or disables both DMA channels. Enables the rotating method for changing the bus control priority structure.
DMA0REQ0 DMA0REQ1 DMA0REQ2 DMA0REQ3 DMA1REQ0 DMA1REQ1 DMA1REQ2 DMA1REQ3 (read/write)	F010H F010H F011H F011H F012H F012H F013H F013H	— — — — — — — —	Channel 0 and 1 Requester Address: Contains channel *n*'s 26-bit requester address. During a buffer transfer, this address may be incremented, decremented, or left unchanged. Reading these registers returns the current address.
DMA0TAR0 DMA0TAR1 DMA0TAR2 DMA0TAR3 DMA1TAR0 DMA1TAR1 DMA1TAR2 DMA1TAR3 (read/write)	F000H F000H F087H F086H F002H F002H F083H F085H	0000H 0000H 0087H — 0002H 0002H 0083H —	Channel 0 and 1 Target Address: Contains channel *n*'s 26-bit target address. During a buffer transfer, this address may be incremented, decremented, or left unchanged. Reading these registers returns the current address.
DMA0BYC0 DMA0BYC1 DMA0BYC2 DMA1BYC0 DMA1BYC1 DMA1BYC2 (read/write)	F001H F001H F098H F003H F003H F099H	0001H 0001H — 0003H 0003H —	Channel 0 and 1 Byte Count: Contains channel *n*'s 24-bit byte count. During a buffer transfer, this byte count is decremented. Reading these registers returns the current byte count.

Table 12-3. DMA Registers (Sheet 2 of 3)

Register	Expanded Address	PC/AT* Address	Description
DMASTS (read only)	F008H	0008H	DMA Status: Indicates whether a hardware request is pending on channel 0 and 1. Indicates whether channel 0's or channel 1's byte count has expired.
DMACMD2 (write only)	F01AH	—	DMA Command 2: Assigns a bus control requester (DMA channel 0, DMA channel 1, or external bus master) to the lowest priority level. Selects the type of sampling for the end-of-process (EOP#) and the DMA request (DRQ*n*) inputs. The DMA can sample these signals asynchronously or synchronously.
DMAMOD1 (write only)	F00BH	000BH	DMA Mode 1: Determines the data-transfer mode. Enables the autoinitialize buffer-transfer mode. Determines the transfer direction (whether the target is the destination or source for a transfer). Determines whether the DMA increments or decrements the target address during a buffer transfer (only if the DMA is set up to modify the target address; see DMAMOD2).
DMAMOD2 (write only)	F01BH	—	DMA Mode 2: Selects the data transfer bus cycle option. Specifies whether the requester and target are in memory or I/O. Determines whether the DMA modifies the target and requester addresses. Determines whether the DMA increments or decrements the requester address during a buffer transfer (only if the DMA is set up to modify the requester address).
DMASRR (read/write)	F009H	0009H	DMA Software Request: *Write Format* Generates a channel 0 and/or a channel 1 software request. *Read Format* Indicates whether a software request is pending on DMA channel 0 or 1.
DMAMSK (write only)	F00AH	000AH	DMA Individual Channel Mask: Individually masks (disables) channel 0's and 1's hardware request input (DREQ0 and DREQ1). This does not mask software requests.
DMAGRPMSK (read/write)	F00FH	000FH	DMA Group Channel Mask: Simultaneously masks (disables) both channels' hardware request inputs (DREQ0 and DREQ1). This does not mask software requests.
DMABSR (write only)	F018H	—	DMA Bus Size: Determines the requester and target data bus widths (8 or 16 bits).

Table 12-3. DMA Registers (Sheet 3 of 3)

Register	Expanded Address	PC/AT* Address	Description
DMACHR (write only)	F019H	—	DMA Chaining: Enables chaining buffer-transfer mode for a specified channel.
DMAIEN (read/write)	F01CH	—	DMA Interrupt Enable: Connects the channel transfer complete status signals to the interrupt request output (DMAINT).
DMAIS (read only)	F019H	—	DMA Interrupt Status: Indicates which signal generated an interrupt request: channel 0 transfer complete, channel 1 transfer complete, channel 0 chaining, or channel 1 chaining status.
DMAOVFE (read/write)	F01DH	—	DMA Overflow Enable: Included for 8237A compatibility. Controls whether all 26 bits or only the lower 16 bits of the requester and target addresses are incremented or decremented during buffer transfers. Controls whether the byte count is 24 bits or 16 bits.

12.3.1 Pin Configuration Register (PINCFG)

Use PINCFG to connect DACK0#, EOP#, and DACK1# to package pins.

Pin Configuration
PINCFG
(read/write)

Expanded Addr: **F826H**
ISA Addr: **—**
Reset State: **00H**

7							0
—	PM6	PM5	PM4	PM3	PM2	PM1	PM0

Bit Number	Bit Mnemonic	Function
7	—	Reserved. This bit is undefined; for compatibility with future devices, do not modify this bit.
6	PM6	Pin Mode: 0 = Selects CS6# at the package pin. 1 = Selects REFRESH# at the package pin.
5	PM5	Pin Mode: 0 = Selects the coprocessor signals, PEREQ, BUSY#, and ERROR#, at the package pins. 1 = Selects the timer control unit signals, TMROUT2, TMRCLK2, and TMRGATE2, at the package pins.
4	PM4	Pin Mode: 0 = Selects DACK0# at the package pin. 1 = Selects CS5# at the package pin.
3	PM3	Pin Mode: 0 = Selects EOP# at the package pin. 1 = Selects CTS1# at the package pin.
2	PM2	Pin Mode: 0 = Selects DACK1# at the package pin. 1 = Selects TXD1 at the package pin.
1	PM1	Pin Mode: 0 = Selects SRXCLK at the package pin. 1 = Selects DTR1# at the package pin.
0	PM0	Pin Mode: 0 = Selects SSIOTX at the package pin. 1 = Selects RTS1# at the package pin.

Figure 12-18. Pin Configuration Register (PINCFG)

12.3.2 DMA Configuration Register (DMACFG)

Use DMACFG to select one of the hardware sources for each channel and to mask the DMA acknowledge (DMAACK*n*#) signals when using internal requesters.

DMA Configuration
DMACFG
(read/write)

Expanded Addr: **F830H**
ISA Addr: **—**
Reset State: **00H**

7							0
D1MSK	D1REQ2	D1REQ1	D1REQ0	D0MSK	D0REQ2	D0REQ1	D0REQ0

Bit Number	Bit Mnemonic	Function
7	D1MSK	DMA Acknowledge 1 Mask: 0 = DMA channel 1's acknowledge (DMAACK1#) signal is not masked. 1 = Masks DMA channel 1's acknowledge (DMAACK1#) signal. Useful when channel 1's request (DREQ1) input is connected to an internal peripheral.
6–4	D1REQ2:0	DMA Channel 1 Request Connection: Connects one of the eight possible hardware sources to channel 1's request input (DREQ1). 000 = DRQ1 pin (external peripheral) 001 = SIO channel 1's receive buffer full signal (RBFDMA1) 010 = SIO channel 0's transmit buffer empty signal (TXEDMA0) 011 = SSIO receive holding buffer full signal (SSRBF) 100 = TCU counter 2's output signal (OUT2) 101 = SIO channel 0's receive buffer full signal (RBFDMA0) 110 = SIO channel 1's transmit buffer empty signal (TXEDMA1) 111 = SSIO transmit holding buffer empty signal (SSTBE)
3	D0MSK	DMA Acknowledge 0 Mask: 0 = DMA channel 0's acknowledge (DMAACK0#) signal is not masked. 1 = Masks DMA channel 0's acknowledge (DMAACK0#) signal. Useful when channel 0's request (DREQ0) input is connected to an internal peripheral.
2–0	D0REQ2:0	DMA Channel 0 Request Connection: Connects one of the eight possible hardware sources to channel 0's request input (DREQ0). 000 = DRQ0 pin (external peripheral) 001 = SIO channel 0's receive buffer full signal (RBFDMA0) 010 = SIO channel 1's transmit buffer empty signal (TXEDMA1) 011 = SSIO transmit holding buffer empty signal (SSTBE) 100 = TCU counter 1's output signal (OUT1) 101 = SIO channel 1's receive buffer full signal (RBFDMA1) 110 = SIO channel 0's transmit buffer empty signal (TXEDMA0) 111 = SSIO receive holding buffer full signal (SSRBF)

Figure 12-19. DMA Configuration Register (DMACFG)

12.3.3 Channel Registers

To program a DMA channel's requester and target addresses and its byte count, write to the DMA channel registers. Some of the channel registers require the use of a byte pointer (BP) flip-flop to control the access to the upper and lower bytes. After you write or read a register that requires a byte pointer specification, the DMA toggles the byte pointer. For example, writing to DMA0TAR0 with BP=0 causes the DMA to set BP. The clear byte pointer software command (DMACLRBP) is available so that you can force BP to a known state (0) before writing to the channel registers. Issue DMACLRBP by writing to location F00CH or 000CH; the data written to the location doesn't matter —writing to the location is all that is necessary to cause the DMA to clear the byte pointer.

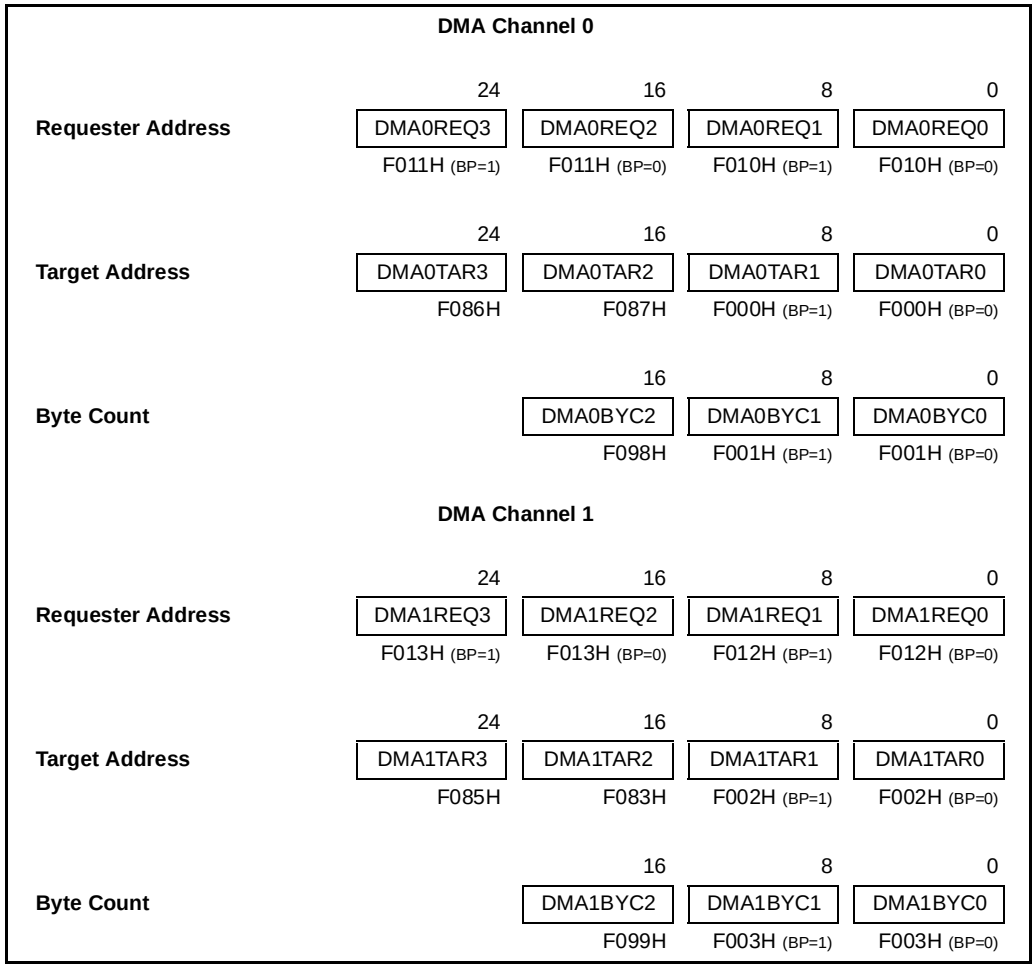


Figure 12-20. DMA Channel Address and Byte Count Registers (DMA*n*REQ*n*, DMA*n*TAR*n*, DMA*n*BYC*n*)

NOTE

The value you write to the byte count register must be one less than the number of bytes to be transferred. To transfer one byte, write zero to the byte count register (byte count = number of bytes – 1). To transfer one word, write one (byte) to the byte count register (byte count = [number of words X 2] – 1).

12.3.4 Overflow Enable Register (DMAOVFE)

Use DMAOVFE to specify whether all 26 bits or only the lower 16 bits of the target and requester addresses are incremented or decremented during buffer transfers and to determine whether all 24 bits of the byte count or only the lower 16 bits of the byte count are decremented during buffer transfers. A byte count configured for 16-bit decrementing expires when it is decremented from 0000H to 0FFFFH.

DMA Overflow Enable
DMAOVFE
(read/write)

Expanded Addr: **F01DH**
ISA Addr: **—**
Reset State: **0AH**

7							0
—	—	—	—	ROV1	TOV1	ROV0	TOV0

Bit Number	Bit Mnemonic	Function
7–4	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
3	ROV1	Channel 1 Requester Overflow Enable: 0 = lowest 16 bits of requester address increment/decrement 1 = all bits of requester address increment/decrement
2	TOV1	Channel 1 Target & Byte Counter Overflow Enable: 0 = lowest 16 bits of target address and byte count increment/decrement 1 = all bits of target address and byte count increment/decrement
1	ROV0	Channel 0 Requester Overflow Enable: 0 = lowest 16 bits of requester address increment/decrement 1 = all bits of requester address increment/decrement
0	TOV0	Channel 0 Target & Byte Counter Overflow Enable: 0 = lowest 16 bits of target address and byte count increment/decrement 1 = all bits of target address and byte count increment/decrement

Figure 12-21. DMA Overflow Enable Register (DMAOVFE)

12.3.5 Command 1 Register (DMACMD1)

Use DMACMD1 to enable both channels and to select the rotating method for changing the bus control priority structure.

DMA Command 1
DMACMD1
(write only)

Expanded Addr: **F008H**
ISA Addr: **0008H**
Reset State: **00H**

7							0
—	—	—	PRE	—	CE	—	—

Bit Number	Bit Mnemonic	Function
7–5	—	Reserved; for compatibility with future devices, write zeros to these bits.
4	PRE	Priority Rotation Enable: 0 = Priority is fixed based on value in DMACMD2. 1 = Enables the rotation method for changing the bus control priority structure. That is, after the external bus master or one of the DMA channels is given bus control, it is assigned to the lowest priority level.
3	—	Reserved; for compatibility with future devices, write zero to this bit.
2	CE	Channel Enable: 0 = Enables channel 0 and 1. 1 = Disables the channels.
1–0	—	Reserved; for compatibility with future devices, write zeros to these bits.

Figure 12-22. DMA Command 1 Register (DMACMD1)

12.3.6 Status Register (DMASTS)

Use DMASTS to check the status of the channels individually. The DMA sets bits in this register to indicate that a channel has a hardware request pending or that a channel's byte count has expired.

DMA Status
DMASTS
(read only)

Expanded Addr: **F008H**
ISA Addr: **0008H**
Reset State: **00H**

7							0
—	—	R1	R0	—	—	TC1	TC0

Bit Number	Bit Mnemonic	Function
7–6	—	Reserved. These bits are undefined.
5	R1	Request 1: When set, this bit indicates that channel 1 has a hardware request pending. When the request is removed, this bit is cleared.
4	R0	Request 0: When set, this bit indicates that channel 0 has a hardware request pending. When the request is removed, this bit is cleared.
3–2	—	Reserved. These bits are undefined.
1	TC1	Transfer Complete 1: When set, this bit indicates that channel 1 has completed a buffer transfer (either its byte count expired or it received an EOP# input). Reading this register clears this bit and clears TC1 in DMAIS.
0	TC0	Transfer Complete 0: When set, this bit indicates that channel 0 has completed a buffer transfer (either its byte count expired or it received an EOP# input). Reading this register clears this bit and clears TC0 in DMAIS.

Figure 12-23. DMA Status Register (DMASTS)

12.3.7 Command 2 Register (DMACMD2)

Use DMACMD2 to select the DREQ*n* and EOP# sampling: asynchronous or synchronous. Bus timing diagrams that show the differences between asynchronous and synchronous sampling are shown in Figure 12-5 on page 12-10 and Figure 12-13 on page 12-21. Also, use DMACMD2 to assign a particular bus request to the lowest priority level for fixed priority mode.

DMA Command 2
DMACMD2
(write only)

Expanded Addr: **F01AH**
ISA Addr: **—**
Reset State: **08H**

7							0
—	—	—	—	PL1	PL0	ES	DS

Bit Number	Bit Mnemonic	Function
7–4	—	Reserved; for compatibility with future devices, write zeros to these bits.
3–2	PL1:0	Low Priority Level Set: Use these bits to assign a particular bus request to the lowest priority level in fixed priority mode. 00 = Assigns channel 0's request (DREQ0) to the lowest priority level 01 = Assigns channel 1's request (DREQ1) to the lowest priority level 10 = Assigns the external bus master request (HOLD) to the lowest priority level 11 = Reserved
1	ES	EOP# Sampling: 0 = Causes the DMA to sample the EOP# input asynchronously. 1 = Causes the DMA to sample the end-of-process (EOP#) input synchronously.
0	DS	DREQ*n* Sampling: 0 = Causes the DMA to sample the DREQ*n* inputs asynchronously. 1 = Causes the DMA to sample the channel request (DREQ*n*) inputs synchronously.

Figure 12-24. DMA Command 2 Register (DMACMD2)

12.3.8 Mode 1 Register (DMAMOD1)

Use DMAMOD1 to select a particular channel's data-transfer mode and transfer direction and to enable the channel's auto-initialize buffer-transfer mode. You can configure the DMA to modify the target address during a buffer transfer by clearing DMAMOD2.2, then use DMAMOD1.5 to specify how the channel modifies the address.

DMA Mode 1
DMAMOD1
(write only)

Expanded Addr: **F00BH**
ISA Addr: **000BH**
Reset State: **00H**

7							0
DTM1	DTM0	TI	AI	TD1	TD0	0	CS

Bit Number	Bit Mnemonic	Function
7–6	DTM1:0	Data-transfer Mode: 00 = Demand 01 = Single 10 = Block 11 = Cascade
5	TI	Target Increment/Decrement: 0 = Causes the target address to be incremented after each data transfer in a buffer transfer. 1 = Causes the target address for the channel specified by bit 0 to be decremented after each data transfer in a buffer transfer. Note that it does not decrement words. When decrementing it will do two byte transfers for a word. Note: When the target address is programmed to remain constant (DMAMOD2.2 = 1), this bit is a don't care.
4	AI	Autoinitialize: 0 = Disables the autoinitialize buffer-transfer mode for the channel specified by bit 0. 1 = Enables the autoinitialize buffer-transfer mode for the channel specified by bit 0.
3–2	TD1:0	Transfer Direction: Determines the transfer direction for the channel specified by bit 0. 00 = Target is read; nothing is written (used for testing) 01 = Data is transferred from the requester to the target 10 = Data is transferred from the target to the requester 11 = Reserved Note: In cascade mode, these bits become don't cares.
1	0	Must be 0 for correct operation.
0	CS	Channel Select: 0 = The selections for bits 7–2 affect channel 0. 1 = The selections for bits 7–2 affect channel 1.

Figure 12-25. DMA Mode 1 Register (DMAMOD1)

12.3.9 Mode 2 Register (DMAMOD2)

Use DMAMOD2 to select the data transfer bus cycle option, specify whether the requester and target are in memory or I/O, and determine whether the DMA modifies the target and requester addresses. If you set up the DMA to modify the requester address, use DMAMOD2 to determine whether the DMA increments or decrements the requester address during a buffer transfer.

DMA Mode 2
DMAMOD2
(write only)

Expanded Addr: **F01BH**
ISA Addr: **—**
Reset State: **00H**

7							0
BCO	RD	TD	RH	RI	TH	0	CS

Bit Number	Bit Mnemonic	Function
7	BCO	Bus Cycle Option: 0 = Selects the fly-by data transfer bus cycle option for the channel specified by bit 0. 1 = Selects the two-cycle data transfer bus cycle option for the channel specified by bit 0.
6	RD	Requester Device Type: 0 = Clear this bit when the requester for the channel specified by bit 0 is in memory space. 1 = Set this bit when the requester for the channel specified by bit 0 is in I/O space. This bit is ignored if BCO is cleared.
5	TD	Target Device Type: 0 = Clear this bit when the target for the channel specified by bit 0 is in memory space. 1 = Set this bit when the target for the channel specified by bit 0 is in I/O space.
4	RH	Requester Address Hold: 0 = Causes the address to be modified (incremented or decremented, depending on DMAMOD2.3). 1 = Causes the requester's address for the channel specified by bit 0 to remain constant during a buffer transfer.
3	RI	Requester Address Increment/Decrement: 0 = Causes the requester address to be incremented after each data transfer in a buffer transfer. 1 = Causes the requester address for the channel specified by bit 0 to be decremented after each data transfer in a buffer transfer. Note that it does not decrement words. When decrementing it will do two byte transfers for a word. Note: When the target address is programmed to remain constant (DMAMOD2.4 = 1), this bit is a don't care.
2	TH	Target Address Hold: 0 = Causes the address to be modified (incremented or decremented, depending on DMAMOD1.5). 1 = Causes the target's address for the channel specified by bit 0 to remain constant during a buffer transfer.
1	0	Must be 0 for correct operation.
0	CS	Channel Select: 0 = The selections for bits 7–2 affect channel 0. 1 = The selections for bits 7–2 affect channel 1.

Figure 12-26. DMA Mode 2 Register (DMAMOD2)

12.3.10 Software Request Register (DMASRR)

Write DMASRR to issue software DMA service requests. Software requests are subject to bus control priority arbitration with all other software and hardware requests. A software request activates the internal channel request signal. This signal remains active until the channel completes its buffer transfer (either by an expired byte count or an EOP# input). In the demand data-transfer mode, a buffer transfer is suspended by deactivating the channel request signal. Because you cannot deactivate the internal channel request signal before the end of a buffer transfer, you cannot use software requests with demand data-transfer mode.

DMA Software Request (write format)
DMASRR

Expanded Addr: **F009H**
ISA Addr: **0009H**
Reset State: **00H**

7							0
—	—	—	—	—	SR	0	CS

Bit Number	Bit Mnemonic	Function
7–3	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
2	SR	Software Request: Setting this bit generates a software request for the channel specified by bit 0. When the channel's buffer transfer completes, this bit is cleared.
1	0	Must be 0 for correct operation.
0	CS	Channel Select: 0 = The selection for bit 2 affects channel 0. 1 = The selection for bit 2 affects channel 1.

Figure 12-27. DMA Software Request Register (DMASRR – write format)

Read DMASRR to see whether a software request for a particular channel is pending. Each request bit is cleared upon Terminal Count or external EOP#. When in auto-initialize mode, both bits are cleared when a Terminal Count or external EOP# occurs.

DMA Software Request (read format)
DMASRR

Expanded Addr: **F009H**
ISA Addr: **0009H**
Reset State: **00H**

7							0
—	—	—	—	—	—	SR1	SR0

Bit Number	Bit Mnemonic	Function
7–2	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
1	SR1	Software Request 1: When set, this bit indicates that channel 1 has a software request pending.
0	SR0	Software Request 0: When set, this bit indicates that channel 0 has a software request pending.

Figure 12-28. DMA Software Request Register (DMASRR – read format)

12.3.11 Channel Mask and Group Mask Registers (DMAMSK and DMAGRPMSK)

Use the DMAMSK and DMAGRPMSK registers to disable (mask) or enable channel hardware requests. DMAMSK allows you to disable or enable hardware requests for only one channel at a time, while DMAGRPMSK allows you to disable or enable hardware requests for both channels at once.

NOTE

Each mask bit is set when its associated channel produces an End-of-Process if the channel is not programmed for Autoinitialize. Software must then clear the appropriate mask bit to allow further DREQ*n* requests from initiating transfers.

DMA Individual Channel Mask
DMAMSK
(write only)

Expanded Addr: **F00AH**
ISA Addr: **000AH**
Reset State: **04H**

7							0
—	—	—	—	—	HRM	0	CS

Bit Number	Bit Mnemonic	Function
7–3	—	Reserved; for compatibility with future devices, write zeros to these bits.
2	HRM	Hardware Request Mask: 0 = Unmasks (enables) hardware requests for the channel specified by bit 0. 1 = Masks (disables) hardware requests for the channel specified by bit 0. NOTE: When this bit is set, the channel can still receive software requests.
1	0	Must be 0 for correct operation.
0	CS	Channel Select: 0 = The selection for bit 2 affects channel 0. 1 = The selection for bit 2 affects channel 1.

Figure 12-29. DMA Channel Mask Register (DMAMSK)

DMA Group Channel Mask
DMAGRPMSK
(read/write)

Expanded Addr: **F00FH**
ISA Addr: **000FH**
Reset State: **03H**

7							0
—	—	—	—	—	—	HRM1	HRM0

Bit Number	Bit Mnemonic	Function
7–2	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
1	HRM1	Hardware Request Mask 1: 0 = Channel 1's hardware requests are not masked. 1 = Masks (disables) channel 1's hardware requests. When this bit is set, channel 1 can still receive software requests.
0	HRM0	Hardware Request Mask 0: 0 = Channel 0's hardware requests are not masked. 1 = Masks (disables) channel 0's hardware requests. When this bit is set, channel 0 can still receive software requests.

Figure 12-30. DMA Group Channel Mask Register (DMAGRPMSK)

12.3.12 Bus Size Register (DMABSR)

Use DMABSR to determine the requester and target data bus widths (8 or 16 bits).

DMA Bus Size
DMABSR
(write only)

Expanded Addr: **F018H**
ISA Addr: **—**
Reset State: **X1X10000B**

7							0
—	RBS	—	TBS	—	—	0	CS

Bit Number	Bit Mnemonic	Function
7	—	Reserved; for compatibility with future devices, write zero to this bit.
6	RBS	Requester Bus Size: Specifies the requester's data bus width for the channel specified by bit 0. 0 = 16-bit bus 1 = 8-bit bus
5	—	Reserved; for compatibility with future devices, write zero to this bit.
4	TBS	Target Bus Size: Specifies the target's data bus width for the channel specified by bit 0. 0 = 16-bit bus 1 = 8-bit bus
3–1	0	Must be 0 for correct operation.
0	CS	Channel Select: 0 = The selections for bits 7–4 affect channel 0. 1 = The selections for bits 7–4 affect channel 1.

Figure 12-31. DMA Bus Size Register (DMABSR)

12.3.13 Chaining Register (DMACHR)

Use DMACHR to enable or disable the chaining buffer-transfer mode for a selected channel. The following steps describe how to set up a channel to perform chaining buffer transfers.

1. Set up the chaining interrupt (DMAINT) service routine.
2. Configure the channel for the single buffer-transfer mode.
3. Program the mode registers.
4. Program the target address, requester address, and byte count registers.
5. Enable the channel for the chaining buffer-transfer mode. (This enables the DMAINT output.)
6. Enable the DMAINT interrupt in the ICU and service it. (The service routine should load the transfer information for the next buffer transfer.)
7. Enable the channel by unmasking DREQ*n* and setting bit 2 in DMACMD1.

From this point, the chaining interrupt indicates each time the channel requires new transfer information. The cycle continues as long as the chaining buffer-transfer mode is enabled and new transfer information is written to the channel. New transfer information must be written to the channel before the channel's current buffer transfer completes.

DMA Chaining
DMACHR
(write only)

Expanded Addr: **F019H**
ISA Addr: **—**
Reset State: **00H**

7							0
—	—	—	—	—	CE	0	CS

Bit Number	Bit Mnemonic	Function
7–3	—	Reserved; for compatibility with future devices, write zeros to these bits.
2	CE	Chaining Enable: 0 = Disables the chaining buffer-transfer mode for the channel specified by bit 0. 1 = Enables the chaining buffer-transfer mode for the channel specified by bit 0.
1	0	Must be 0 for correct operation.
0	CS	Channel Select: 0 = The selection for bit 2 affects channel 0. 1 = The selection for bit 2 affects channel 1.

Figure 12-32. DMA Chaining Register (DMACHR)

12.3.14 Interrupt Enable Register (DMAIEN)

Use DMAIEN to individually connect channel 0's and 1's transfer complete signal to the DMAINT interrupt request output.

DMA Interrupt Enable
DMAIEN
(read/write)

Expanded Addr: **F01CH**
ISA Addr: **—**
Reset State: **00H**

7							0
—	—	—	—	—	—	TC1	TC0

Bit Number	Bit Mnemonic	Function
7–2	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
1	TC1	Transfer Complete 1: 0 = Disables Transfer Complete interrupts. 1 = Connects channel 1's transfer complete signal to the interrupt control unit's DMAINT input. Note: When channel 1 is in chaining mode (DMACHR.2=1 and DMACHR.0=1), this bit is a don't care.
0	TC0	Transfer Complete 0: 0 = Disables Transfer Complete interrupts. 1 = Connects channel 0's transfer complete signal to the interrupt control unit's DMAINT input. Note: When channel 0 is in chaining mode (DMACHR.2=1 and DMACHR.0=0), this bit is a don't care.

Figure 12-33. DMA Interrupt Enable Register (DMAIEN)

12.3.15 Interrupt Status Register (DMAIS)

DMAIS indicates which source activated the DMA interrupt request signal (channel 0 transfer complete, channel 1 transfer complete, channel 0 chaining, or channel 1 chaining).

DMA Interrupt Status
DMAIS
(read only)

Expanded Addr: **F019H**
ISA Addr: **—**
Reset State: **00H**

7							0
—	—	TC1	TC0	—	—	CI1	CI0

Bit Number	Bit Mnemonic	Function
7–6	—	Reserved. These bits are undefined.
5	TC1	Transfer Complete 1: When set, this bit indicates that channel 1 has completed a buffer transfer (either its byte count expired or it received an EOP# input). This bit is set only if bit 1 of the interrupt enable register is set. Reading the DMA status register (DMASTS) clears this bit. Note: In chaining mode, this bit becomes a don't care.
4	TC0	Transfer Complete 0: When set, this bit indicates that channel 0 has completed a buffer transfer (either its byte count expired or it received an EOP# input). This bit is set only if bit 0 of the interrupt enable register is set. Reading the DMA status register (DMASTS) clears this bit. Note: In chaining mode, this bit becomes a don't care.
3–2	—	Reserved. These bits are undefined.
1	CI1	Chaining Interrupt 1: When set, this bit indicates that new requester and target addresses and a new byte count should be written to channel 1. This bit is cleared when new transfer information is written to the channel. (Writing to the most-significant byte of the target address clears this bit.) Note: Outside chaining mode, this bit becomes a don't care.
0	CI0	Chaining Interrupt 0: When set, this bit indicates that new requester and target addresses and a new byte count should be written to channel 0. This bit is cleared when new transfer information is written to the channel. (Writing to the most-significant byte of the target address clears this bit.) Note: Outside chaining mode, this bit becomes a don't care.

Figure 12-34. DMA Interrupt Status Register (DMAIS)

12.3.16 Software Commands

The DMA contains four software commands: clear byte pointer, clear DMA, clear mask register, and clear transfer complete signal. Each software command has an I/O address associated with it (see Table 12-4). To issue a software command, write to its I/O address; the data written doesn't matter — writing to the location is all that is necessary.

Table 12-4. DMA Software Commands

Name (Address)	Command	Functions
DMACLRBP (0F00CH or 000CH)	Clear byte pointer	Resets the byte pointer flip-flop. Perform this command at the beginning of any access to the channel registers, to ensure a predictable place in the register programming sequence.
DMACLR (0F00DH or 000DH)	Clear DMA	Sets all DMA functions to their default states.
DMACLRMSK (0F00EH or 000EH)	Clear mask register	Simultaneously clears the mask bits of all channels (enabling all channels).
DMACLRTC (0F01EH)	Clear transfer complete signal	Resets the transfer complete signal (DMAINT). Allows the source of the DMA request (hardware or software) to acknowledge the completion of a transfer process.

12.4 DESIGN CONSIDERATIONS

EOP# requires an external pull-up resistor. To determine the maximum value, the rise time must be less than three bus cycles. To determine the minimum value, use the I_{OL} specification from the *Intel386™ EX Embedded Microprocessor* datasheet (order number 272420).

12.5 PROGRAMMING CONSIDERATIONS

Consider the following when programming the DMA.

- The DMA transfers data between a requester and a target. The transfer direction is programmable and determines whether the requester or the target is the source or destination of a transfer.
- The two-cycle data transfer bus cycle option uses a four-byte temporary buffer. During a buffer transfer, the channel fills the temporary buffer from the source before writing any data to the destination. Therefore, the number of bus cycles that it takes to transfer data from the source to the destination depends on the amount of data to transfer and the source and destination data bus widths.
- Each channel contains a 26-bit requester address, 26-bit target address, and 24-bit byte count. These values are programmed through the use of 8-bit registers, some of which are multiplexed at the same addresses. A byte pointer (BP) controls the access to these multiplexed registers. After you write or read a register that requires a byte pointer specification, the channel toggles the byte pointer. For example, writing to DMA0TAR0

with BP=0 causes the DMA to set BP. The clear byte pointer software command (DMACLRBP) allows you to force BP to a known state (0) before writing to the registers.

- The target and requester addresses are incremented, decremented, or left unchanged and the byte count is decremented after each data transfer within a buffer transfer. Reading a register returns the current (or modified) value rather than the original programmed values.
- The chaining buffer-transfer mode requires that you write new transfer information to the channel before the current buffer transfer completes. The channel determines whether new transfer information was written to it by checking the most-significant byte of the target address. Writing to this byte sets an internal flag that tells the channel that new transfer information was written to it. Therefore, it is only necessary to change the target address between chaining buffer transfers. If you want to change the requester address and byte count also, you should write these values before writing the most-significant byte of the target address.
- If a channel is configured to increment the requester address and the requester's bus size is selected as 16 bits, the channel increments the requester address by two after each data transfer. However, if the channel is configured to decrement the requester address, the channel only decrements the address by one. This is true for the target also. In other words, the channels cannot decrement by words. When a channel is configured to decrement the requester or target address and transfer words, the correct number of words is transferred; however, the transfers are on a byte basis.
- Enabling both the autoinitialize and chaining buffer-transfer modes will have unpredictable results.
- The DMA controller does not allow programming one channel while another channel is active. If both channels are being used, the programmer must mask an active channel before reprogramming the other channel. Failure to do this may result in incorrect DMA transfers.

12.5.1 DMA Controller Code Examples

This section contains these software routines:

EnableDMAHWRequests	Enables channel hardware requests for the given DMA channel
DisableDMAHWRequests	Disables channel hardware requests for the specified DMA channel
SetDMAReqIOAddr	Sets the requester to an I/O port address for the specified channel
SetDMATargMemAddr	Sets the target memory address for the specified DMA channel
SetDMAXferCount	Sets the target memory device for the specified DMA channel
InitDMA	Initializes the DMA

InitDMA1ForSSIXmitterToMem	Initializes DMA channel1 for transfers between the SIO transmitter port and memory
DMAInterrupt	Interrupt Service Routine for DMA generated interrupts

See Appendix C for included header files.

```
#include "80386ex.h"
#include "ev386ex.h"
#include "dma.h"
#include <DOS.h>

#pragma warning(disable:4704)          /*Disable optimization warning*/


/*****************************************************************************
   EnableDMAHWRequests:
   int EnableDMAHWRequests(int nChannel)

   Description:
      Enables channel hardware requests for the given DMA channel.

   Parameters:
      nChannel        --channel to enable hardware requests

   Returns:
   Error Code

   Assumptions:
    None

   Syntax:
   int error_code;

   error_code = EnableDMAHWRequests (DMA_Channel0);

   Real/Protected Mode:
   No changes required

*****************************************************************************/

int EnableDMAHWRequests(int nChannel)
{
   BYTE regDMAMSK = 0; /*Clear regDMAMSK[HRM]*/

   /*Check input*/
   if (  (nChannel != DMA_Channel0) && (nChannel != DMA_Channel1) )
      return ERR_BADINPUT;

   regDMAMSK = nChannel;  /*Set regDMAMSK[CS] to channel*/
   _SetEXRegByte(DMAMSK, regDMAMSK);  /*Clear hardware request mask for*/
```

```
                                        /*    given channel*/
}


/*****************************************************************************
   DisableDMAHWRequests:

   Description:
      Disables channel hardware requests for the given DMA channel.
      The channel, however, can still receive software requests.

   Parameters:
      nChannel       --channel to mask hardware requests

   Returns:
   Error Code

   Assumptions:
   None

   Syntax:
      int error_code;

      error_code = DisableDMAHWRequests(DMA_Channel0);

   Real/Protected Mode:
      No changes required

*****************************************************************************/

int DisableDMAHWRequests(int nChannel)
{
   WORD regDMAMSK = 0;

   //Check input
   if (  (nChannel != DMA_Channel0) && (nChannel != DMA_Channel1) )
      return ERR_BADINPUT;

   regDMAMSK = nChannel;                //Set regDMAMSK[CS] to channel
   regDMAMSK &= 0x04;                   //Set regDMAMSK[HRM]
   _SetEXRegByte(DMAMSK, regDMAMSK);    //Set hw request mask for given
                                        //channel
   return ERR_NONE;
}




/*****************************************************************************
   SetDMAReqIOAddr:
```

```
   Description:
      Sets the requester to an I/O port address, wIO, for the DMA
      channel specified by nChannel.

   Parameters:
      nChannel       --channel for which to set Requester I/O port address
      wIO            --I/O address

   Returns:
   None

   Assumptions:
   None

   Syntax:

      SetDMAReqIOAddr(DMA_Channel1, TBR0);    //Sets Req. to Serial Receiver

   Real/Protected Mode:
   No changes required

*************************************************************************/

int SetDMAReqIOAddr(int nChannel, WORD wIO)
{
   WORD addrDMAReq0_1;
   WORD addrDMAReq2_3;
                                          /*Check input*/
   if (  (nChannel != DMA_Channel0) && (nChannel != DMA_Channel1) )
      return ERR_BADINPUT;

   /*Set registers to correct channel*/
   addrDMAReq0_1 = ( nChannel == DMA_Channel0 ? DMA0REQ0_1 : DMA1REQ0_1);
   addrDMAReq2_3 = ( nChannel == DMA_Channel0 ? DMA0REQ2_3 : DMA1REQ2_3);

   _SetEXRegByte(DMACLRBP, 0x0);  /* Clear the byte pointer flip-flop */

   /* Write requester I/O address, bits 0-7 */
   _SetEXRegByte(addrDMAReq0_1, (BYTE) (wIO & 0xFF));

   /* Write requester I/O address, bits 8-15 */
   _SetEXRegByte(addrDMAReq0_1, (BYTE) ((wIO >> 8) & 0xFF));
   _SetEXRegByte(addrDMAReq2_3, 0x00); /* Zero requester address bits 16-23 */
   _SetEXRegByte(addrDMAReq2_3, 0x00); /* Zero requester address bits 24-25 */

   return ERR_NONE;
}


/*************************************************************************
   SetDMATargMemAddr:
```

```
   Description:
      Sets the target memory address for the DMA channel specified
      by nChannel.

   Parameters:
      nChannel       --channel for which to set target address
      ptMemory       --pointer to target memory location

   Returns:
   None

   Assumptions:
      Processor is in real mode.

   Syntax:
      static char lpsz[]="Hello World";

      SetDMATargMemAddr(DMA_Channel1, lpsz);

   Real/Protected Mode:
      The address calculation from ptMemory assumes the processor is in real
      mode.

*************************************************************************/

int SetDMATargMemAddr(int nChannel, void *ptMemory)
{
   WORD addrDMATar0_1;
   WORD addrDMATar2;
   WORD addrDMATar3;
   WORD wSegment;
   WORD wOffset;
   DWORD lAddress;

   /*Check input*/
   if (  (nChannel != DMA_Channel0) && (nChannel != DMA_Channel1) )
      return ERR_BADINPUT;

   /*Set registers to correct channel*/
   addrDMATar0_1 = ( nChannel == DMA_Channel0 ? DMA0TAR0_1 : DMA1TAR0_1);
   addrDMATar2 = ( nChannel == DMA_Channel0 ? DMA0TAR2 : DMA1TAR2);
   addrDMATar3 = ( nChannel == DMA_Channel0 ? DMA0TAR3 : DMA1TAR3);

   /*If in tiny, small, or medium model,*/
   #if defined(M_I86TM) || defined(M_I86SM) || defined(M_I86MM)
      _asm
      {                        /*...then grab our segment from DS*/
         mov ax, ds
         mov wSegment, ds
      }
      wOffset = (WORD) ptMemory; /*...and our offset from the pointer*/
```

```
   #else      /*Else in compact, large, or huge memory model*/
      wSegment = _FP_SEG(ptMemory);  /*...grab the segment from the pointer*/
      wOffset = _FP_OFF(ptMemory);   /*...and the offset from the pointer*/
   #endif                            /*Assuming real mode, compute our physical
address*/
      lAddress = ((DWORD) wSegment << 4) + wOffset;

   _SetEXRegByte(DMACLRBP, 0x0);   /*Clear the byte pointer flip-flop */

   /* Write target address, bits 0-7 */
   _SetEXRegByte(addrDMATar0_1, (BYTE) (lAddress & 0xFF));

   /* Write target address, bits 8-15 */
   _SetEXRegByte(addrDMATar0_1, (BYTE) ((lAddress >> 8) & 0xFF));

   /* Write target address, bits 16-23 */
   _SetEXRegByte(addrDMATar2, (BYTE) ((lAddress >> 16) & 0xFF));

   /* Write target address, bits 24-25 */
   _SetEXRegByte(addrDMATar3, (BYTE) ((lAddress >> 24) & 0x03));

   return ERR_NONE;
}



/*****************************************************************************
   SetDMAXferCount:

   Description:
      Sets the target memory device for the DMA channel specified
      by nChannel

   PARAMETERS:
      nChannel       --channel for which to set target address
      ptMemory       --pointer to target memory location

   Returns:
   None

   Assumptions:
      Processor is in real mode.

   Syntax:
      static char lpsz[]="Hello World";

      SetDMATargMemAddr(DMA_Channel0, lpsz);

   Real/Protected Mode:
      The address calculation from ptMemory assumes the processor is in real
      mode.
```

```
*****************************************************************************/

int SetDMAXferCount(int nChannel, DWORD lCount)
{
   WORD addrDMAByc0_1;
   WORD addrDMAByc2;
                                              /*Check input*/
   if (  (nChannel != DMA_Channel0) && (nChannel != DMA_Channel1) )
      return ERR_BADINPUT;

   /*Set registers to correct channel*/
   addrDMAByc0_1 = (nChannel == DMA_Channel0 ? DMA0BYC0_1 : DMA1BYC0_1);
   addrDMAByc2 = (nChannel == DMA_Channel0 ? DMA0BYC2 : DMA1BYC2);


   _SetEXRegByte(DMACLRBP, 0x0);  /* Clear the byte pointer flip-flop */

   /* Write count, bits 0-7 */
   _SetEXRegByte(addrDMAByc0_1, (BYTE) ( lCount & 0xFF));

   /* Write count, bits 8-15 */
   _SetEXRegByte(addrDMAByc0_1, (BYTE) ((lCount >> 8)  & 0xFF));

   /* Write count, bits 16-23 */
   _SetEXRegByte(addrDMAByc2, (BYTE) ((lCount >> 16) & 0xFF));

   return ERR_NONE;
}


/*****************************************************************************
   InitDMA:

   Description:
      Enables the DMA and initializes settings independent of the
      two channels:

         bus arbitration--set to no rotation, external bus master
                          request(HOLD) assigned to lowest priority level
         EOP# sampling--set to asynch. (no effect when DMA is used
                        with internal peripherals)
         DRQn sampling--set to synch. (no effect when DMA is used
                        with internal peripherals)

   Parameters:
   None

   Returns:
   None

   Assumptions:
      None
```

```
   Syntax:
      InitDMA();      //Initialize DMA peripheral

   Real/Protected Mode:
   No changes required

*****************************************************************************/

void InitDMA(void)
{
   _SetEXRegByte(DMACLR, 0x0);   /*Resets DMA peripheral*/
   _SetEXRegByte(DMACMD1, 0x0);  /*DMACMD1[7:5]=0: reserved*/
                                 /*DMACMD1[4]=0: disable priority rotation*/
                                 /*    enable*/
                                 /*DMACMD1[2]=0: enable channel's 0 and 1*/
                                 /*DMACMD1[1:0]=0: reserved*/

   _SetEXRegByte(DMACMD2, 0x8);  /*DMACMD2[7:4]=0: reserved*/
                                 /*DMACMD2[3:2]=2: assign HOLD to the lowest*/
                                 /*    priority level*/
                                 /*DMACMD2[1]=0:  EOP# samples input async.*/
                                 /*DMACMD2[0]=0:  DRQn samples input async.*/
}


/*****************************************************************************
   InitDMA1ForSSIXmitterToMem:

   Description:
      This function prepares DMA channel 1 for transfers between the async.
      serial port transmitter (channel 0) and memory.  After calling this
      function, a DMA transfer can be initiated by setting the Target address,
      setting the transfer count, and clearing the hardware request mask
      for this DMA channel.

   Parameters:
   None

   Returns:
   None

   Assumptions:
      InitDMA() has been called to enable the peripheral.

   Syntax:
      static char lpsz[]="Hello World";

      InitDMA();                      //Initialize DMA peripheral
      InitDMA1ForSerialXmitter();     //Initialize DMA channel 1
      .
      .
```

```
      .
      SetDMATargMemAddr(DMA_Channel1, lpsz); //Set target memory address
                                             //Set transfer count
      SetDMAXferCount(DMA_Channel1, strlen(lpsz) );
      EnableDMAHWRequests(DMA_Channel1);     //Begin transfer at SIO request

   Real/Protected Mode:
   No changes required

*****************************************************************************/

void InitDMA1ForSerialXmitter(void)
{
   BYTE regDMACfg;
   BYTE regDMAIE;
   BYTE regDMAOvfE;

   DisableDMAHWRequests(DMA_Channel1); /*Disable channel 1 Hardware requests*/

   regDMACfg = (_GetEXRegByte(DMACFG) & 0x0F) | 0xA0;
   _SetEXRegByte(DMACFG, regDMACfg); /*DMACFG[7]=1: mask DMA Acknowledge for*/
                               /*   channel 1*/
                                    /*DMACFG[6:4]=3: set channel request to*/
                                    /*   SIO's channel 0's transmit buffer*/
                                    /*   empty signal*/
                                    /*DMAMSK[3:0]=unmodified: channel 0*/
                                    /*   settings*/

   _SetEXRegByte(DMAMOD1, 0x9); /*DMAMOD1[7:6]=0: set to demand data-xfer*/
                             /* mode*/
                                 /*DMAMOD1[5]=0: increment target*/
                                 /*DMAMOD1[4]=0: disable autoinitialize*/
                                 /*    buffer-xfer mode*/
                                 /*DMAMOD1[3:2]=2: data is xfer'd from targ.*/
                                 /*    to req.*/
                                 /*DMAMOD1[1]=0: reserved*/
                                 /*DMAMOD1[0]=1: selections for bits 7-2*/
                                 /*    affect channel 1*/

   _SetEXRegByte(DMAMOD2, 0xD1); /*DMAMOD2[7]=1: Select 2-cycle data xfer*/
                                  /*DMAMOD2[6]=1: Requester is in I/O space*/
                                  /*DMAMOD2[5]=0: Target is in memory space*/
                                  /*DMAMOD2[4]=1: Requester is held constant*/
                                  /*   thru xfer*/
                                  /*DMAMOD2[3]=x: Req. Inc/Dec...see*/
                                  /*   DMAMOD2[4]*/
                                  /*DMAMOD2[2]=0: Target address is*/
                                  /*   modified...see DMAMOD1[5]*/
                                  /*DMAMOD2[1]=0: reserved*/
                                  /*DMAMOD2[0]=1: selections for bits 7-2*/
                                  /*    affect channel 1*/
```

```
   _SetEXRegByte(DMABSR, 0x51); /*DMABSR[7]=0: reserved*/
                                /*DMABSR[6]=1: sets req.'s bus size to 8-bit*/
                                /*DMABSR[5]=0: reserved*/
                                /*DMABSR[4]=1: sets tar.'s bus size to 8-bit*/
                                /*DMABSR[3:2]=0: reserved*/
                                /*DMABSR[0]=1: selections for bits 7-2*/
                                /*    affect channel 1*/

   _SetEXRegByte(DMACHR, 0x1); /*DMACHR[7:3]=0: reserved*/
                               /*DMACHR[2]=0: disable chaining buffer-xfer*/
                               /*    mode*/
                               /*DMACHR[1]=0: reserved*/
                               /*DMACHR[0]=1: selections for bits 7-2 affect*/
                               /*    channel 1*/

   regDMAIE = _GetEXRegByte(DMAIEN) & 0x1;
   _SetEXRegByte(DMAIEN, regDMAIE); /*DMAIE[7:2]=untouched: reserved*/
                                    /*DMAIE[1]=0: masks channel 1's transfer*/
                                    /*   complete signal from interrupt*/
                                    /*   controller*/
                                    /*DMAIE[0]=untouched:  channel 0 setting*/

   regDMAOvfE = _GetEXRegByte(DMAOVFE) | 0xC;
   _SetEXRegByte(DMAOVFE, regDMAOvfE); /*DMAOVFE[7:4]=untouched: reserved*/
                                       /*DMAOVFE[3]=1: all bits of channel 1
                                       /*   req. address are inc/dec*/
                                       /*   (see DMAMOD[4])*/
                                       /*DMAOVFE[2]=1: all bits of channel 1*/
                                       /*   target addr. are inc/dec*/
                                       /*DMAOVFE[1:0]=untouched: channel 0*/
                                /*   settings*/

   SetDMAReqIOAddr(DMA_Channel1, TBR0); /*Sets Req. I/O address to Serial*/
                                        /*Receiver*/
}


/*****************************************************************************
   DMAInterrupt:

   Description:
      This function is called by the DMA unit when it either completes a
      transfer or (in chaining xfer mode) when a new requester, target, and
      byte count should be written to the device.

   Parameters:
      None

   Returns:
      None

   Assumptions:
```

```
      None

   Syntax:
      regDMAIE = _GetEXRegByte(DMAIEN) | 0x2; //Enable tc interrupt for
                                              // channel 0
      _SetEXRegByte(DMAIEN, regDMAIE);

      //Set interrupt routine
      SetIRQVector(DMAInterrupt, 12, INTERRUPT_ISR);
      Enable8259Interrupt(0, IR4);  //Enable slave IR4, DMA interrupt

      NonSpecificEOI();  //Clear all interrupts

   Real/Protected Mode:
   No changes required

*************************************************************************/

void interrupt far DMAInterrupt(void)
{
   WORD regDMAIS;

   regDMAIS = _GetEXRegByte(DMAIS); /*Get interrupt status register*/

   if (regDMAIS & 0x10)
   {                    /*Transfer Complete, channel 0*/

      _SetEXRegByte(DMACLRTC, 0x00); /*Clear transfer complete signal*/
   }


   if (regDMAIS & 0x20)
   {                    /*Transfer Complete, channel 1*/
      _SetEXRegByte(DMACLRTC, 0x00); /*Clear transfer complete signal*/
   }

   if (regDMAIS & 0x1)
   {                   /*Chaining Interrupt, channel 0*/

   }

   if (regDMAIS & 0x2)
   {                   /*Chaining Interrupt, channel 1*/

   }

   NonSpecificEOI();  /*Send End-Of-Interrupt Signal to Master/Slave*/
}
```

13

SYNCHRONOUS SERIAL I/O UNIT

CHAPTER 13
SYNCHRONOUS SERIAL I/O UNIT

The synchronous serial I/O (SSIO) unit provides 16-bit bidirectional serial communications. The transmit and receive channels can operate independently (that is, with different clocks) to provide full-duplex communications. Either channel can originate the clocking signal or receive an externally generated clocking signal.

This chapter is organized as follows:

13.1 OVERVIEW

The SSIO unit contains a baud-rate generator, transmitter, and receiver. The baud-rate generator has two possible internal clock sources (PSCLK or SERCLK). The transmitter and receiver are double buffered. They contain 16-bit holding buffers and 16-bit shift registers. Data to be transmitted is written to the transmit holding buffer. The buffer's contents are transferred to the transmit shift register and shifted out via the serial data transmit pin (SSIOTX). Data received is shifted in via the serial data receive pin (SSIORX). Once 16 bits have been received, the contents of the receive shift register are transferred to the receive buffer.

Both the transmitter and receiver can operate in either master or slave mode. In master mode, the internal baud-rate generator controls the serial communications by clocking the internal transmitter or receiver. If the transmitter or receiver is enabled in master mode, the baud-rate generator's signal appears on the transmit or receive clock pin, and is available for clocking an external slave transmitter or receiver. In slave mode, an external master device controls the serial communications. An input on the external transmit or receive clock pin clocks the transmitter or receiver. The transmitter and receiver need not operate in the same mode. This allows the transmitter and receiver to operate at different frequencies (an internal and an external clock source or two different external clock sources can be used). Figures 13-1 through 13-4 illustrate the various transmitter/receiver master/slave combinations.

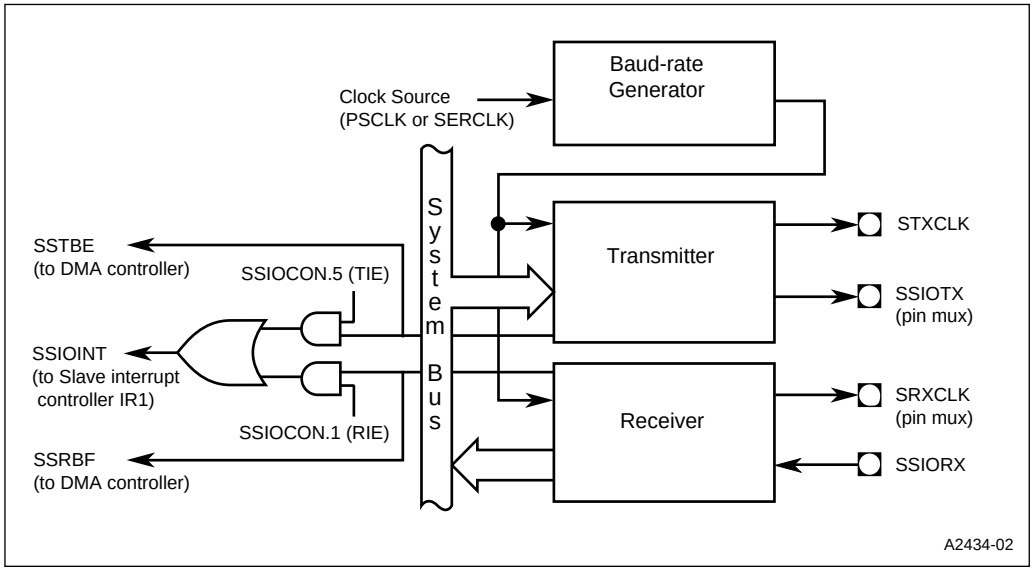


Figure 13-1. Transmitter and Receiver in Master Mode

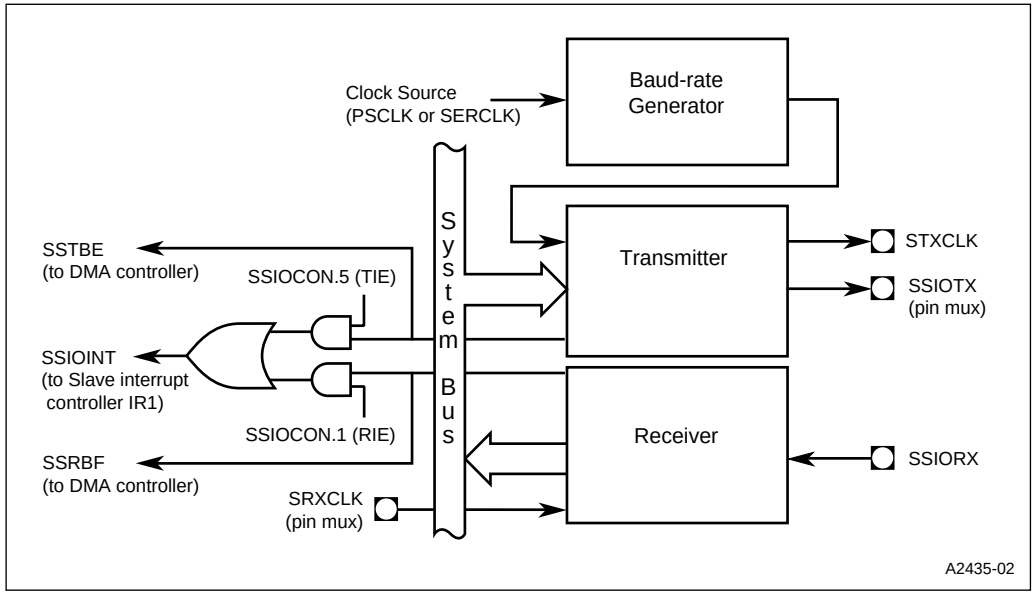


Figure 13-2. Transmitter in Master Mode, Receiver in Slave Mode

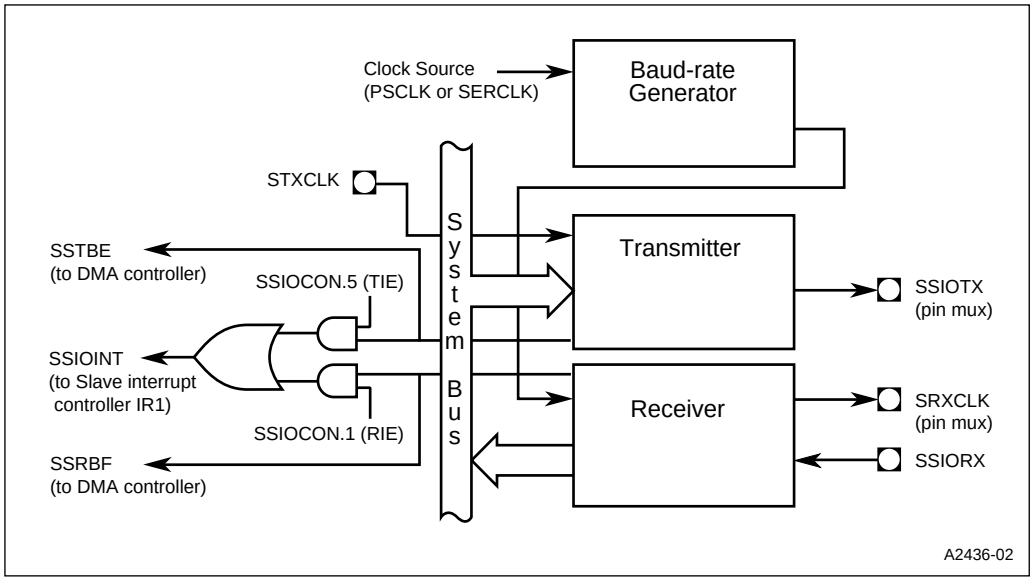


Figure 13-3. Transmitter in Slave Mode, Receiver in Master Mode

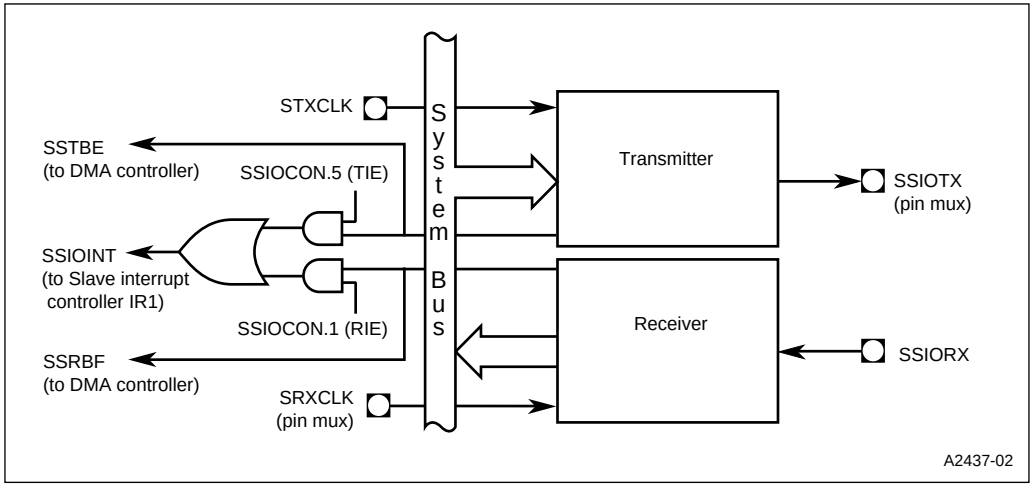


Figure 13-4. Transmitter and Receiver in Slave Mode

13.1.1 SSIO Signals

Table 13-1 lists the SSIO signals.

Table 13-1. SSIO Signals

Signal	Device Pin or Internal Signal	Description
STXCLK	Device pin (input or output)	Serial Transmit Clock: This pin functions as either an output or an input, depending on whether the transmitter is operating in master or slave mode. In master mode, STXCLK functions as an output. The baud-rate generator's output appears on this pin through the transmitter and can be used to clock a slave receiver. In slave mode, STXCLK functions as an input clock for the transmitter.
SRXCLK	Device pin (input or output)	Serial Receive Clock: This pin functions as either an output or an input, depending on whether the receiver is operating in master or slave mode. In master mode, SRXCLK functions as an output. The baud-rate generator's output appears on this pin through the receiver and can be used to clock a slave transmitter. In slave mode, SRXCLK functions as an input clock for the receiver.
SSIOTX	Device pin (output)	Transmit Serial Data: The transmitter uses this pin to shift serial data out of the device. Data is transmitted most-significant bit first.
SSIORX	Device pin (input)	Receive Serial Data: The receiver uses this pin to shift serial data into the device. Data is received most-significant bit first.
SSRBF	Internal signal (output)	Receive Buffer Full: This internal signal is used to indicate that received serial data has been transferred from the receive shift register to the receive holding buffer.
SSTBE	Internal signal (output)	Transmit Buffer Empty: This internal signal is used to indicate that serial data has been shifted from the transmit holding register to the transmit shift register.
SSIOINT	Internal signal (output)	SSIO Interrupt: This internal signal goes active when either the transmit holding register is empty or the receive holding register is full.
BCLKIN	Internal signal (input)	*Prescaled Clock (PSCLK)*: This internal signal is a prescaled value of the internal clock frequency (CLK2/2). PSCLK is programmable for a range of divide-by values. *Serial Clock (SERCLK)*: This internal signal is half the internal clock frequency (CLK2/4).

13.2 SSIO OPERATION

The following sections describe the operation of the baud-rate generator, transmitter, and receiver.

13.2.1 Baud-rate Generator

Either the prescaled clock or the serial clock (PSCLK or SERCLK) can drive the baud-rate generator (Figure 13-5). The SIO and SSIO configuration register (SIOCFG) selects one of these sources.

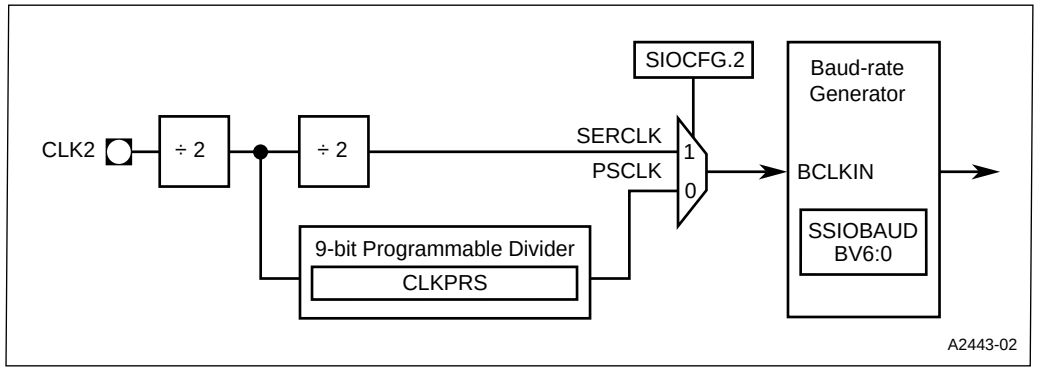


Figure 13-5. Clock Sources for the Baud-rate Generator

$$BCLKIN = SERCLK = \frac{CLK2}{4}$$

OR

$$BCLKIN = PSCLK = \frac{CLK2/2}{\text{prescale value} + 2}$$

SERCLK provides a baud-rate input frequency (BCLKIN) of CLK2/4. The PSCLK frequency depends on the 9-bit programmable divider. The input to the programmable divider is divided by a 9-bit prescale value + 2.

A prescale value of 0 gives the maximum PSCLK frequency (CLK2/4) and a prescale value of 1FFH (511) gives the minimum PSCLK frequency (CLK2/1026).

The baud-rate generator contains a seven-bit down counter. A programmable baud-rate value (BV) is the reload value for the counter. The counter counts down from BV to zero, toggles the baud-rate generator output, then reloads the BV and counts down again. The baud-rate generator's output is a function of BV and BCLKIN as follows.

$$\text{baud-rate output frequency} = \frac{\text{BCLKIN}}{2\text{BV} + 2}$$

A BV of 0 gives the maximum output frequency (BCLKIN/2) and a BV of 7FH (127) gives the minimum output frequency (BCLKIN/256).

If you know the desired baud-rate output frequency, you can determine BV as follows.

$$\text{BV} = \left(\frac{\text{BCLKIN}}{2 \times \text{baud-rate output frequency}}\right) - 1$$

The maximum and minimum baud-rate output frequencies with a 33MHz (CLK2 = 66MHz) device are shown in Table 13-2.

Table 13-2. Maximum and Minimum Baud-rate Output Frequencies

Baud-rate Value (BV)	Input Frequency (BCLKIN)	Output Frequency
0	16.5 MHz (using either SERCLK or PSCLK with a prescale value of 0)	8.25 MHz
7FH	64.327 KHz (using PSCLK with a prescale value of 1FFH)	251.277 Hz

13.2.2 Transmitter

The transmitter contains a 16-bit buffer and a 16-bit shift register. When the transmitter is enabled, the contents of the buffer are immediately transferred to the shift register. The shift register shifts data out via SSIOTX. Either the internal baud-rate generator (master mode) or an input signal on the STXCLK pin (slave mode) drives the transmitter. The maximum transmitter input frequency is 8.25 MHz with a 33MHz processor clock (CLK2 = 66MHz). In master mode, the baud-rate generator must be programmed and enabled prior to enabling the transmitter. In slave mode, the transmitter must be enabled prior to the application of an external clock.

13.2.2.1 Transmit Mode using Enable Bit

The transmitter contains a transmit holding buffer empty (THBE) flag and a transmit underrun error (TUE) flag. At reset, THBE is set, indicating that the buffer is empty. Writing data to the buffer clears THBE. When the transmitter transfers data from the buffer to the shift register, THBE is set. If the transmitter is enabled (TEN bit is set, AUTOTXM is clear), it transfers the new contents of the transmit buffer to the shift register each time the shift register finishes shifting its current contents.

If the shift register finishes shifting out its current contents before a new value is written to the transmit buffer, it reloads the old value and shifts it out again. This condition is known as a transmitter underrun error. TUE is set to indicate an underrun error. For high speed transfers this can be a problem, since the Baud-rate generator clock may be too fast; it may not allow enough time to control the TEN bit for each word transfer. This could cause the same word to be transmitted more than once. See “Autotransmit Mode” on page 13-12 for a description of how to avoid this problem.

The transmitter also has a transmit holding buffer empty signal (SSTBE). This signal can be connected to the interrupt control and DMA units. This allows you to use either an interrupt service routine or a DMA transfer to load new data in the transmit holding buffer.

Figures 13-6 and 13-7 are simple descriptions of the SSIO transmitter state-machine when Autotransmit mode is enabled or disabled.

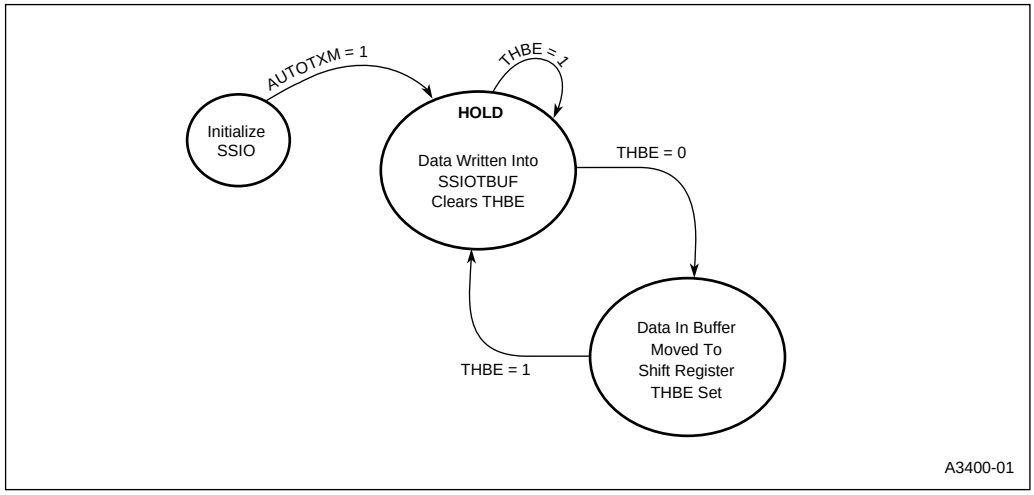


Figure 13-6. SSIO Transmitter with Autotransmit Mode Enabled

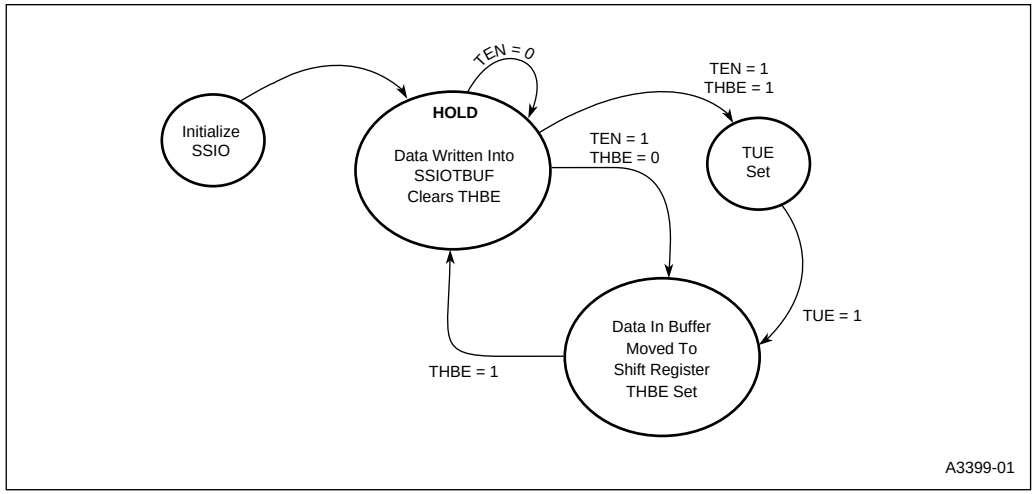


Figure 13-7. SSIO Transmitter with Autotransmit Mode Disabled

The SSIO Unit can be operated either by using a polling method or through interrupts.

- Figure 13-8 shows a basic flowchart for using the polling method to transmit data through the SSIO.
- Figure 13-9 shows a basic flowchart for the Interrupt Service Routine necessary when using interrupts to transmit data through the SSIO. If interrupts are used, follow the below sequence for initialization:

1. Initialize the SSIO.
2. Initialize the interrupts - ICU initialization, Interrupt Service Routine, etc.
3. Unmask the interrupts on the ICU.

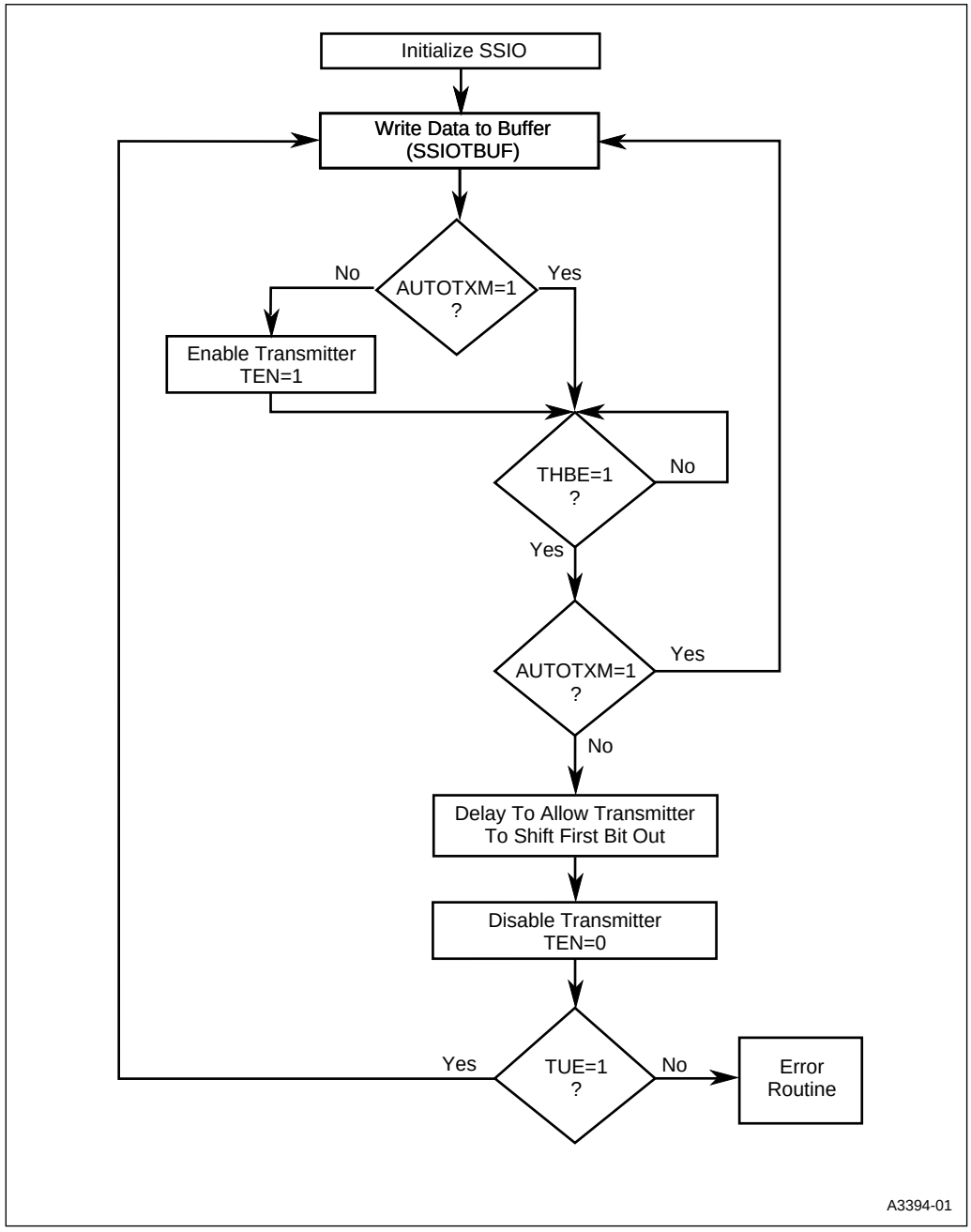


Figure 13-8. Transmit Data by Polling

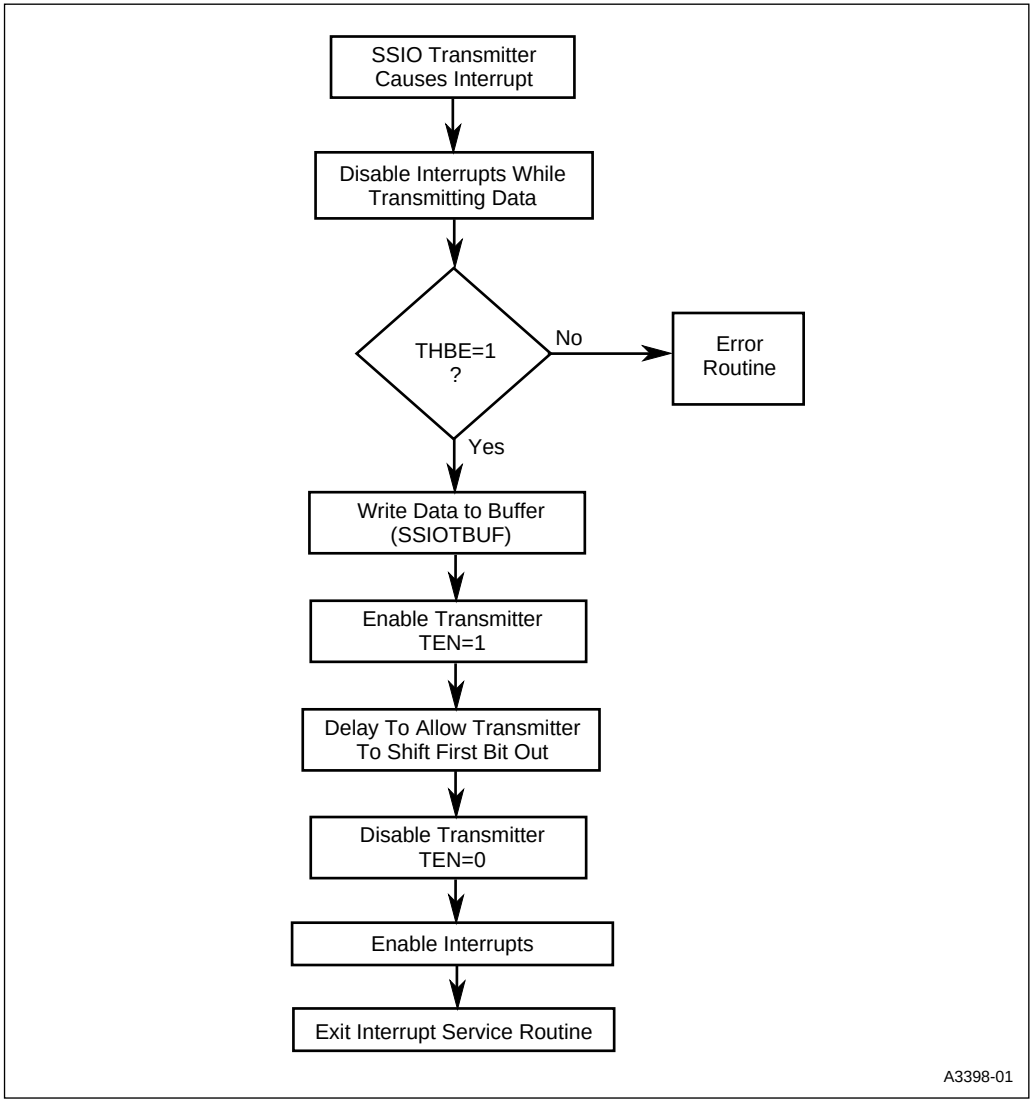


Figure 13-9. Interrupt Service Routine for Transmitting Data Using Interrupts

If the transmitter is disabled while a data value in the shift register is being shifted out, it continues running until the last bit is shifted out. Then the shift register stops and the data and clock pins (SSIOTX and STXCLK) are three-stated; the contents of the buffer register are **not** loaded into the shift register.

If the transmitter is disabled then re-enabled before the current value has been shifted out, it continues as if it were never disabled.

If you enable the transmitter while the baud-rate generator clock is high, the data and clock pin states are as shown in Figure 13-10. If you enable the transmitter while the baud-rate generator clock is low, the data and clock pin states are as shown in Figure 13-11. These figures show master mode, single word transfers. At the end of transmission, STXCLK and SSIOTX are three-stated and require external pull-up resistors. For single word transfers, you must enable the transmitter, which starts the shifting process, then disable the transmitter before 16 bits are shifted out. For high baud rates use the Autotransmit mode.

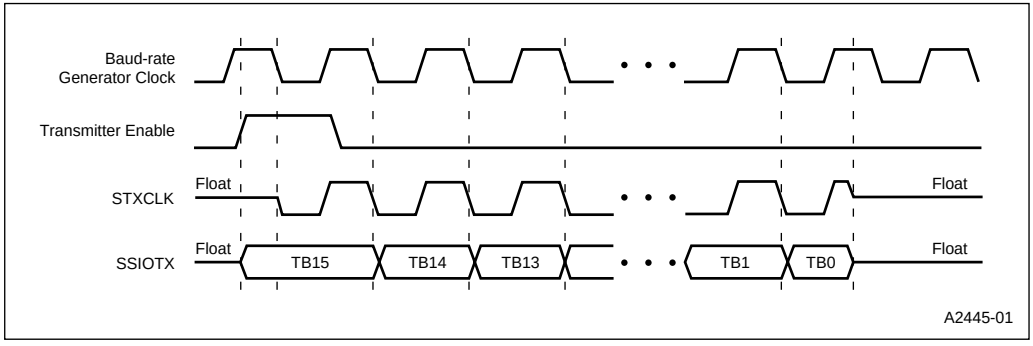


Figure 13-10. Transmitter Master Mode, Single Word Transfer (Enabled when Clock is High)

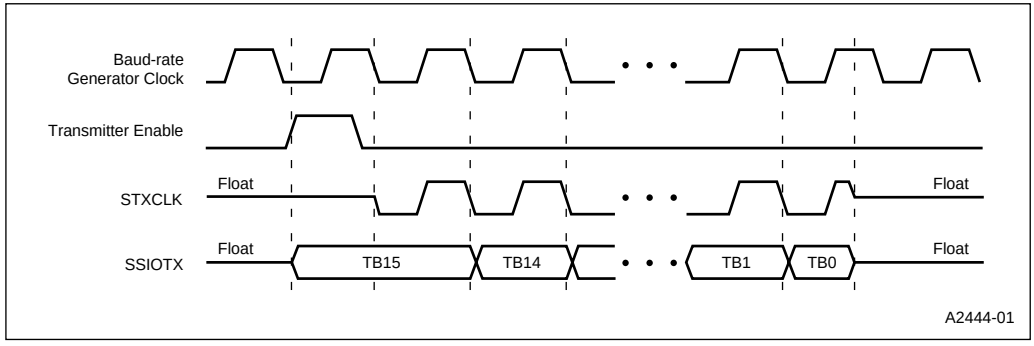


Figure 13-11. Transmitter Master Mode, Single Word Transfer (Enabled when Clock is Low)

13.2.2.2 Autotransmit Mode

Set the AUTOTXM bit (SSIOCON2.2) and the TXMM bit (SSIOCON2.1) to enable Autotransmit mode. When the AUTOTXM bit is set, the word is automatically transferred to the shift register and the THBE bit is set. In this mode the TEN bit is ignored. Once the data is transferred to the shift register, the word is shifted out. If no new data has been written into the buffer, the transmitter stops.

The Transmit Underrun Error (TUE) bit is not used in Autotransmit mode.

The Autotransmit mode eliminates the problem of controlling the TEN bit during high-speed data transfers using polling or interrupts to move new data to the transmit buffer (SSIOTBUF).

13.2.2.3 Slave Mode

Operation in transmitter slave mode is similar to master mode, except the transmitter is clocked from the STXCLK pin. When the transmitter is enabled any time during the STXCLK clock cycle, TB15 appears on the SSIOTX pin and remains on the pin until the second falling edge of STXCLK.

13.2.3 Receiver

The receiver contains a 16-bit holding buffer and a 16-bit shift register. When enabled, the shift register shifts data in via the SSIORX pin. After the receiver shifts in 16 bits of data, the contents of the shift register are transferred to the buffer. Either the internal baud-rate generator (master mode) or an input signal on the SRXCLK pin (slave mode) can clock the receiver.

The receiver contains a receive holding buffer full flag (RHBF) and a receive overflow error flag (ROE). At reset, RHBF is clear, indicating that the buffer is empty. When the receiver transfers data from the shift register to the buffer, RHBF is set. Reading the buffer clears RHBF. When the receiver is enabled, it transfers the contents of the shift register to the receive buffer each time the shift register finishes shifting its current contents. If the shift register finishes shifting in its current contents before the old value is read from the receive buffer, the receiver transfers the new value into the buffer, overwriting the old value and sets the ROE flag. This condition is known as a receive overflow error.

The receiver also has an internal receive holding buffer full signal (SSRBF). This signal can be connected to the DMA unit for DMA initiated transfers. The SSRBF signal is also ORed with the SSTBE signal to generate the SSIOINT signal which is sent to the interrupt controller. Before the SSRBF signal is ORed it is masked with the Receive Interrupt Bit (RIE) in the SSIOCON1 register. These options allow you to use either an interrupt service routine or a DMA transfer to read data from the receive holding buffer.

The SSIO Unit can be operated either by using a polling method or through interrupts.

- Figure 13-12 shows a basic flowchart for using the polling method to receive data through the SSIO.
- Figure 13-13 shows a basic flowchart for the Interrupt Service Routine necessary when using interrupts to receive data through the SSIO. If interrupts are used, follow the below sequence for initialization:

1. Initialize the SSIO.
2. Initialize the interrupts - ICU initialization, Interrupt Service Routine, etc.
3. Unmask the interrupts on the ICU.

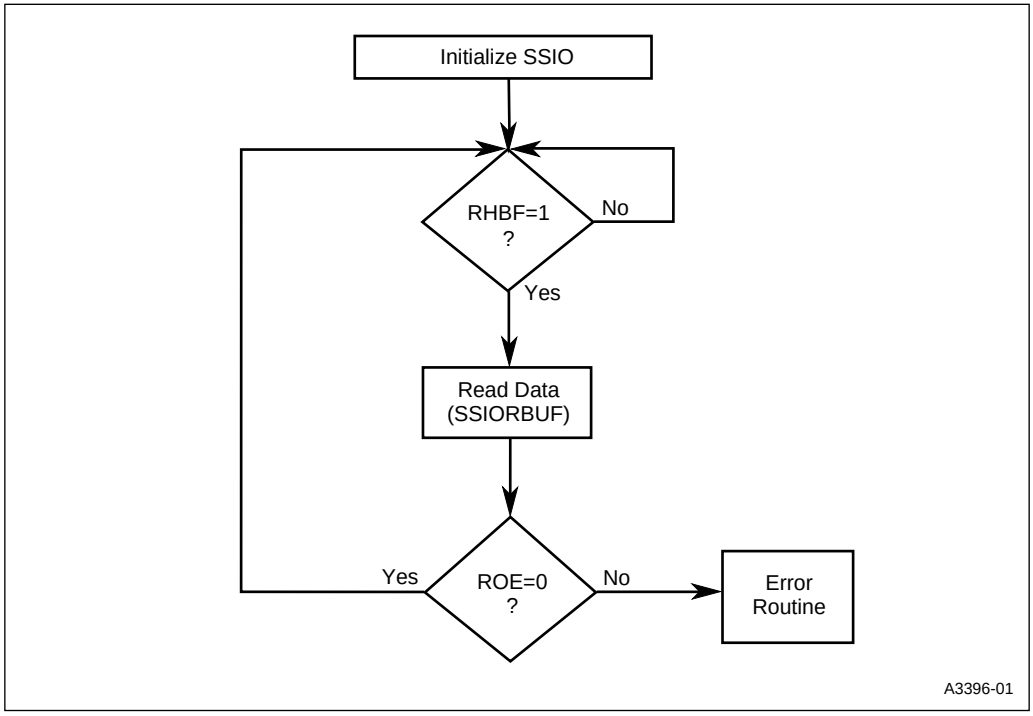


Figure 13-12. Receive Data by Polling

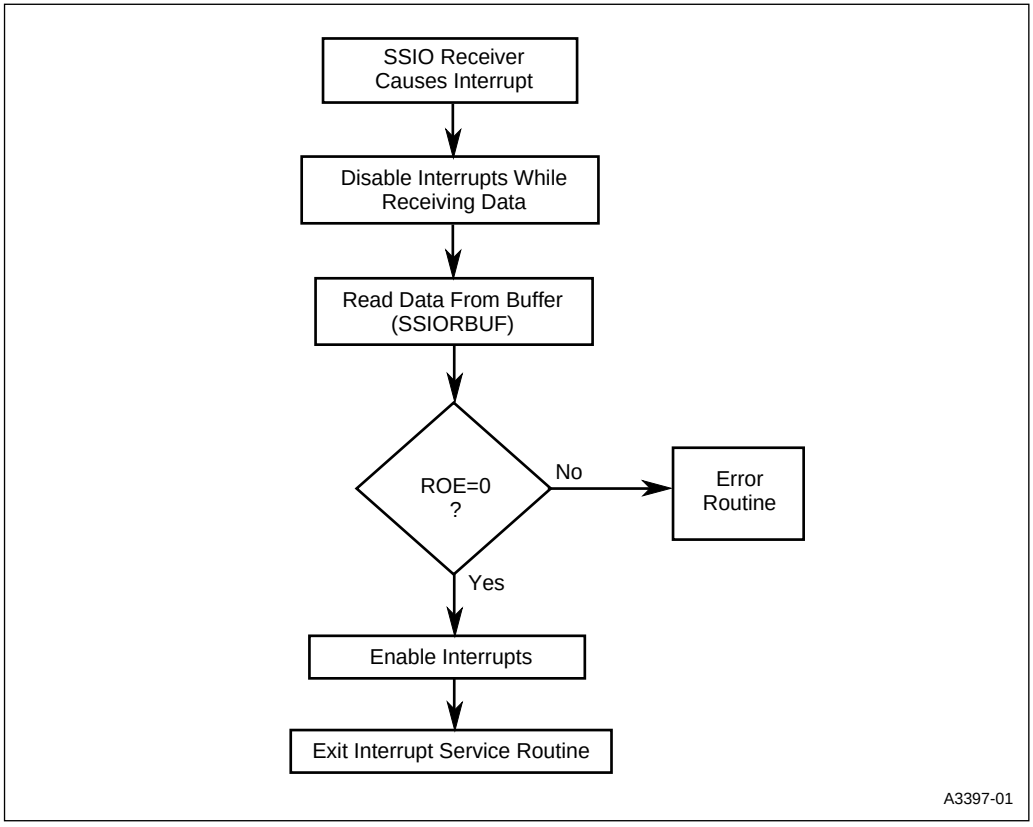


Figure 13-13. Interrupt Service Routine for Receiving Data Using Interrupts

If the receiver is disabled while a data value is being shifted into the shift register, it continues running until the last bit is shifted in. Then the shift register is loaded into the buffer register, the shift register stops and the clock pin (SRXCLK) is three-stated if in the master mode.

If the receiver is disabled then enabled before the current word has been shifted in, it continues as if it were never disabled.

Figure 13-14 shows the serial receive data (SSIORX) pin values for a master mode, single word transfer. For single word transfers, it is necessary to enable the receiver thus starting the shifting process, then disable the receiver before 16 bits are shifted in.

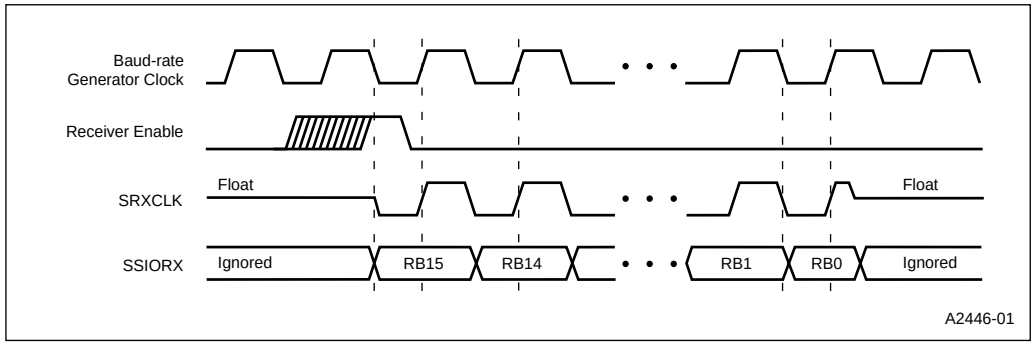


Figure 13-14. Receiver Master Mode, Single Word Transfer

Operation in receiver slave mode is similar to master mode, except the receiver is clocked from the SRXCLK pin. When the receiver is enabled any time during the SRXCLK clock cycle, data on the SSIORX pin is latched into the shift register at the next rising edge of SRXCLK. The SRX-CLK and SSIORX pins are three-stated.

13.3 REGISTER DEFINITIONS

Table 13-3 list the registers associated with the SSIO and the following sections contain bit descriptions for each register.

Table 13-3. SSIO Registers

Register	Expanded Address	Function
PINCFG (read/write)	F826H	Pin Configuration: Connects the serial receive clock signal (SRXCLK) and the transmit serial data signal (SSIOTX) to the package pin.
SIOCFG (read/write)	F836H	SIO and SSIO Configuration: Selects the baud-rate generator's clock source, SERCLK or PSCLK.
CLKPRS (read/write)	F804H	Clock Prescale: Controls the frequency of PSCLK.
SSIOBAUD (read/write)	F484H	SSIO Baud-rate Control: Enables the baud-rate generator and determines its baud rate. In master mode, the transmitter and receiver are clocked by the baud-rate generator.
SSIOCTR (read only)	F48AH	SSIO Baud-rate Count Down: Indicates whether the baud-rate generator is enabled and reflects the current value of the baud-rate down-counter.
SSIOCON1 (read/write)	F486H	SSIO Control 1: Enables the transmitter and receiver, indicates when the transmit buffer is empty and the receive buffer is full. Enables or disables the transmitter or receiver interrupts. SSIOCON1 also indicates two error conditions: the transmit underrun and receiver overflow.
SSIOCON2 (read/write)	F488H	SSIO Control 2: Selects whether the transmitter and receiver are in master or slave mode. In master mode, the baud-rate generator clocks the transmitter or receiver. In slave mode, an external master clocks the transmitter or receiver. Also controls the enabling of the Automatic Transmit mode.
SSIOTBUF (read/write)	F480H	SSIO Transmit Buffer: Holds the 16-bit data word to transmit. Data is transmitted most-significant bit first.
SSIORBUF (read only)	F482H	SSIO Receive Buffer: Holds the 16-bit data word received. Data is received most-significant bit first.

13.3.1 Pin Configuration Register (PINCFG)

The serial receive clock (SRXCLK) and transmit serial data (SSIOTX) pins are multiplexed with other functions. Use PINCFG bits 0 and 1 to select the pin functions.

Pin Configuration
PINCFG
(read/write)

Expanded Addr: **F826H**
ISA Addr: **—**
Reset State: **00H**

7							0
—	PM6	PM5	PM4	PM3	PM2	PM1	PM0

Bit Number	Bit Mnemonic	Function
7	—	Reserved. This bit is undefined; for compatibility with future devices, do not modify this bit.
6	PM6	Pin Mode: 0 = Selects CS6# at the package pin. 1 = Selects REFRESH# at the package pin.
5	PM5	Pin Mode: 0 = Selects the coprocessor signals, PEREQ, BUSY#, and ERROR#, at the package pins. 1 = Selects the timer control unit signals, TMROUT2, TMRCLK2, and TMRGATE2, at the package pins.
4	PM4	Pin Mode: 0 = Selects DACK0# at the package pin. 1 = Selects CS5# at the package pin.
3	PM3	Pin Mode: 0 = Selects EOP# at the package pin. 1 = Selects CTS1# at the package pin.
2	PM2	Pin Mode: 0 = Selects DACK1# at the package pin. 1 = Selects TXD1 at the package pin.
1	PM1	Pin Mode: 0 = Selects SRXCLK at the package pin. 1 = Selects DTR1# at the package pin.
0	PM0	Pin Mode: 0 = Selects SSIOTX at the package pin. 1 = Selects RTS1# at the package pin.

Figure 13-15. Pin Configuration Register (PINCFG)

13.3.2 SIO and SSIO Configuration Register (SIOCFG)

Use SIOCFG bit 2 to connect either PSCLK or SERCLK to the baud-rate generator's input (BCLKIN).

SIO and SSIO Configuration
SIOCFG
(read/write)

Expanded Addr: **F836H**
ISA Addr: **—**
Reset State: **00H**

7							0
S1M	S0M	—	—	—	SSBSRC	S1BSRC	S0BSRC

Bit Number	Bit Mnemonic	Function
7	S1M	SIO1 Modem Signal Connections: 0 = Connects the SIO1 modem input signals to the package pins. 1 = Connects the SIO1 modem input signals internally.
6	S0M	SIO0 Modem Signal Connections: 0 = Connects the SIO0 modem input signals to the package pins. 1 = Connects the SIO0 modem input signals internally.
5–3	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
2	SSBSRC	SSIO Baud-rate Generator Clock Source: 0 = Connects the internal PSCLK signal to the SSIO baud-rate generator. 1 = Connects the internal SERCLK signal to the SSIO baud-rate generator.
1	S1BSRC	SIO1 Baud-rate Generator Clock Source: 0 = Connects the COMCLK pin to the SIO1 baud-rate generator. 1 = Connects the internal SERCLK signal to the SIO1 baud-rate generator.
0	S0BSRC	SIO0 Baud-rate Generator Clock Source: 0 = Connects the COMCLK pin to the SIO0 baud-rate generator. 1 = Connects the internal SERCLK signal to the SIO0 baud-rate generator.

Figure 13-16. SIO and SSIO Configuration Register (SIOCFG)

13.3.3 Prescale Clock Register (CLKPRS)

Use CLKPRS to program the PSCLK frequency.

Clock Prescale Register
CLKPRS
(read/write)

Expanded Addr: **F804H**
ISA Addr: **—**
Reset State: **0000H**

15							**8**
—	—	—	—	—	—	—	PS8
7							**0**
PS7	PS6	PS5	PS4	PS3	PS2	PS1	PS0

Bit Number	Bit Mnemonic	Function
15–9	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
8–0	PS8:0	Prescale Value: These bits determine the divisor that is used to generate PSCLK. Legal values are from 0000H (divide by 2) to 01FFH (divide by 513). divisor = PS8:0 + 2

Figure 13-17. Clock Prescale Register (CLKPRS)

13.3.4 SSIO Baud-rate Control Register (SSIOBAUD)

Use SSIOBAUD to enable the baud-rate generator and determine the baud-rate generator's seven-bit down counter's reload value (BV).

SSIO Baud-rate Control
SSIOBAUD
(read/write)

Expanded Addr: **F484H**
ISA Addr: **—**
Reset State: **00H**

7							0
BEN	BV6	BV5	BV4	BV3	BV2	BV1	BV0

Bit Number	Bit Mnemonic	Function
7	BEN	Baud-rate Generator Enable: Setting this bit enables the baud-rate generator. Clearing this bit disables the baud-rate generator, clears the baud-rate count value, and forces the baud rate clock to zero.
6–0	BV6:0	Baud-rate Value: The baud-rate value (BV) is the reload value for the baud-rate generator's seven-bit down counter. The baud-rate generator's output is a function of BV and the baud-rate generator's input (BCLKIN), as follows. $\text{baud-rate output frequency (Hz)} = \frac{\text{BCLKIN}}{2\text{BV} + 2}$ (Hz) If you know the desired output baud-rate frequency, you can determine BV as follows. $\text{BV} = \left(\frac{\text{BCLKIN}}{2 \times \text{baud-rate output frequency}}\right) - 1$

Figure 13-18. SSIO Baud-rate Control Register (SSIOBAUD)

13.3.5 SSIO Baud-rate Count Down Register (SSIOCTR)

Read SSIOCTR to determine the status of the baud-rate generator. The down counter is reloaded when CV6:0 reaches zero or when a new value is written to SSIOBAUD.

Baud-rate Count Down
SSIOCTR
(read only)

Expanded Addr: **F48AH**
ISA Addr: **—**
Reset State: **00H**

7							0
BSTAT	CV6	CV5	CV4	CV3	CV2	CV1	CV0

Bit Number	Bit Mnemonic	Function
7	BSTAT	Baud-rate Generator Status: 0 = The baud-rate generator is disabled. 1 = The baud-rate generator is enabled.
6–0	CV6:0	Current Value: These bits indicate the current value of the baud-rate down counter.

Figure 13-19. SSIO Baud-rate Count Down Register (SSIOCTR)

13.3.6 SSIO Control 1 Register (SSIOCON1)

SSIOCON1 contains both transmit and receive control and status bits. Use the control bits to enable the receiver and transmitter and to connect the transmit buffer empty and receive buffer full signals to the interrupt control unit. The status bits indicate that the transmit buffer is empty, a transmit underrun error occurred, the receive buffer is full, or a receive overflow error occurred.

Both the transmit buffer empty and the receive buffer full signals can be connected (ORed) to the interrupt request source (SSIOINT). When an interrupt request from this source is detected, you can determine which signal caused the request by reading the SSIOCON1 receive buffer full and transmit buffer empty status bits.

SSIO Control 1 **Expanded Addr: F486H**
SSIOCON1 **ISA Addr: —**
(read/write) **Reset State: C0H**

7							0
TUE	THBE	TIE	TEN	ROE	RHBF	RIE	REN

Bit Number	Bit Mnemonic	Function
7	TUE	Transmit Underrun Error: The transmitter sets this bit to indicate a transmit underrun error in the TEN transfer mode. Clear this bit to clear the error flag. If a one is written to TUE, it is ignored and TUE retains its previous value.
6	THBE (read only bit)	Transmit Holding Buffer Empty: The transmitter sets this bit when the transmit buffer contents have been transferred to the transmit shift register, indicating that the buffer is now ready to accept new data. Writing data to the transmit buffer clears THBE. When this bit is clear, the buffer is not ready to accept any new data.
5	TIE	Transmitter Interrupt Enable: 0 = Clearing this bit prevents the Interrupt Control Unit from sensing when the transmit buffer is empty. 1 = Setting this bit connects the transmit buffer empty internal signal to the Interrupt Control Unit.
4	TEN	Transmitter Enable: 0 = Disables the transmitter. 1 = Enables the transmitter.
3	ROE	Receive Overflow Error: The receiver sets this bit to indicate a receiver overflow error. Write zero to this bit to clear the flag. If a one is written to ROE, the one is ignored and ROE retains its previous value.
2	RHBF (read only bit)	Receive Holding Buffer Full: The receiver sets this bit when the receive shift register contents have been transferred to the receive buffer. Reading the buffer clears this bit.
1	RIE	Receive Interrupt Enable: 0 = Clearing this bit prevents the Interrupt Control Unit from sensing when the receive buffer is full. 1 = Setting this bit connects the receiver buffer full internal signal to the Interrupt Control Unit.
0	REN	Receiver Enable: 0 = Clearing this bit disables the receiver. 1 = Setting this bit enables the receiver.

Figure 13-20. SSIO Control 1 Register (SSIOCON1)

13.3.7 SSIO Control 2 Register (SSIOCON2)

Use the control bits TXMM and RXMM in SSIOCON2 to put the transmitter or receiver in master or slave mode. The AUTOTXM bit is used to determine if the TEN bit controls the transmitting of the data.

SSIO Control 2
SSIOCON2
(read/write)

Expanded Addr: **F488H**
ISA Addr: **—**
Reset State: **00H**

7							0
—	—	—	—	—	AUTOTXM	TXMM	RXMM

Bit Number	Bit Mnemonic	Function
7–3	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
2	AUTOTXM	Automatic Transmit off mode for master mode 0 = Clearing this bit puts the TEN bit into normal operation 1 = Setting this bit and the TXMM bit causes TEN to be ignored. Every time a word is loaded into the transmit shift register from the transmit holding buffer it is transmitted out and then stops.
1	TXMM	Transmit Master Mode: 0 = Clearing this bit puts the transmitter in slave mode. In slave mode, an external device controls the transmit serial communications. An input on the STXCLK pin clocks the transmitter. 1 = Setting this bit puts the transmitter in master mode. In master mode, the internal baud-rate generator controls the transmit serial communications. The baud-rate generator's output clocks the internal transmitter and appears on the STXCLK pin.
0	RXMM	Receive Master Mode: 0 = Clearing this bit puts the receiver in slave mode. In slave mode, an external device controls the receive serial communications. An input on the SRXCLK pin clocks the receiver. 1 = Setting this bit puts the receiver in master mode. In master mode, the internal baud-rate generator controls the receive serial communications. The baud-rate generator's output clocks the internal receiver and appears on the SRXCLK pin.

Figure 13-21. SSIO Control 2 Register (SSIOCON2)

13.3.8 SSIO Transmit Holding Buffer (SSIOTBUF)

Write the data words to be transmitted to SSIOTBUF. Use the interrupt controller, DMA unit or polling (read SSIOCON1) to determine when to write to the transmit buffer.

Transmit Holding Buffer
SSIOTBUF
(read/write)

Expanded Addr: **F480H**
ISA Addr: **—**
Reset State: **0000H**

15							**8**
TB15	TB14	TB13	TB12	TB11	TB10	TB9	TB8
7							**0**
TB7	TB6	TB5	TB4	TB3	TB2	TB1	TB0

Bit Number	Bit Mnemonic	Function
15–0	TB15:0	Transmit Buffer Bits: These bits make up the next data word to be transmitted. The control logic loads this word into the transmit shift register. The transmit shift register shifts the bits out on the falling edge of the tranmitter clock pin. The word is transmitted out starting with the most-significant bit (TB15).

Figure 13-22. SSIO Transmit Holding Buffer (SSIOTBUF)

13.3.9 SSIO Receive Holding Buffer (SSIORBUF)

Read SSIORBUF to obtain the last data word received. Use the interrupt controller, DMA unit or polling (read SSIOCON1) to determine when to read the receive buffer.

Receive Holding Buffer
SSIORBUF
(read only)

Expanded Addr: **F482H**
ISA Addr: **—**
Reset State: **0000H**

15							8
RB15	RB14	RB13	RB12	RB11	RB10	RB9	RB8
7							**0**
RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0

Bit Number	Bit Mnemonic	Function
15–0	RB15:0	Receive Buffer Bits: This register contains the last word received. The receive shift register shifts bits in with the rising edge of the receiver clock pin. Data is shifted in starting with the most-significant bit. The control logic then transfers the received word from the receive shift register to SSIORBUF.

Figure 13-23. SSIO Receive Holding Buffer (SSIORBUF)

13.4 DESIGN CONSIDERATIONS

The transmit buffer empty signal can be connected to the interrupt control and DMA units. However, at high baud-rates interrupt latency is too long to prevent a transmit underrun error. For these cases, use the DMA to load the data to be transmitted into the transmit buffer.

To illustrate this point, assume the maximum input transmit baud-rate of 8.25 MHz. To prevent a transmit underrun error, a new 16-bit data word must be written to the transmit buffer before the transmit shift register shifts out 16 bits.

$$16 \text{ bits} \times \frac{1}{8.25 \text{ MHz}} = 16 \times 121 \text{ ns} = 1939 \text{ ns}$$

At 33 MHz, one clock is 30 ns. The transmit buffer must be reloaded within 64 clocks (1939/30), but interrupt latency is longer than 64 clocks. Therefore, the DMA unit is required to load the transmit buffer.

13.5 PROGRAMMING CONSIDERATIONS

- When operating the transmitter in Master mode, and not in Autotransmit mode, you must ensure that the last character to be transmitted is in the process of being shifted out before disabling the transmitter. If the transmitter is disabled before the character has begun shifting, the character remains in the shift register and is shifted out when the transmitter is re-enabled. At high baud rates this can be a problem and using the Autotransmit mode is recommended.
- The SSIO interrupt line is multiplexed with INT5. When configuring your system for SSIO-generated interrupts, you must clear INTCFG.1 to connect the SSIO interrupt signal to the ICU.
- The serial receive clock (SRXCLK) and transmit serial data (SSIOTX) pins are multiplexed with other functions. Use PINCFG bits 0 and 1 to select the pin functions.
- No register programming is required for the shared signal pairs RI1#/SSIORX and DSR1#/STXCLK. Both do not have multiplexers since one of the shared signals is a dedicated input.

13.5.1 SSIO Example Code

This section includes these software routines:

InitSSIO	Initializes the SSIO for synchronous transfers
SSerialReadWord	Polled serial read function that receives a single character
SSerialWriteWord	Polled serial write function that transmits a single character
SSIO_ISR	Interrupt Service Routine for interrupts generated by the SSIO
Service_RHBF	Service routine for interrupts generated by the RHBF signal
Service_THBE	Service routine for interrupts generated by the THBE signal

The final code example shows an SSIO transfer in which the transmitter is interrupt-driven and the receiver is polled. See Appendix C for included header files.

```
#include <conio.h>
#include “80386EX.h”
#include “EV386EX.h”


WORD value = ‘1’;
BYTE Control;
BYTE poll;


/*************************************************************************
   InitSSIO:

   Description:
```

```
      Initialization routine for Synchronous Serial I/O Port.

   Parameters:
   Mode                Enables receiver and transmitter; Enables TBE and RHBF
                       interrupts
   MasterTxRx          Defines whether Tx and/or Rx are in Master Mode
      BaudValue        Enables Baud-rate generator and sets Baud-rate Value
      PreScale         9-bit Clock prescale value

   Returns:
      None

   Assumptions:
      PINCFG & SIOCFG should be configured before this is called.
      Prescale is only used if SIOCFG.2 is clear.

   Syntax:

      #define SSIO_TX_MASTR      0x2            // Transmit Master Mode
      #define SSIO_RX_MASTR      0x1            // Receive Master Mode
      #define SSIO_TX_SLAVE      0              // Transmit Slave Mode
      #define SSIO_RX_SLAVE      0              // Receive Slave Mode
      #define SSIO_TX_IE         0x20           // Transmit Interrupt Enable
      #define SSIO_TX_ENAB       0x10           // Transmitter Enable
      #define SSIO_RX_IE         0x2            // Receive Interrupt Enable
      #define SSIO_RX_ENAB       0x1            // Receiver Enable
      #define SSIO_BAUD_ENAB     0x80           // Enable Baud Rate Generator
   #define SSIO_CLK_SERCLK       0x1            // Baud Rate Clocking Source:
                                                // SERCLK = CLK2/4
      #define SSIO_CLK_PSCLK     0x0            // Baud Rate Clocking Source:
                                                // PSCLK  = (CLK2/2) /
(CLKPRS+2)

    InitSSIO (SSIO_TX_IE| SSIO_TX_ENAB | SSIO_RX_ENAB,
            SSIO_RX_MASTR | SSIO_TX_SLAVE,
            SSIO_BAUD_ENAB,
            SSIO_CLK_PSCLK);

   Real/Protected Mode:
   No changes required.

*************************************************************************/

void InitSSIO(BYTE Mode, BYTE MasterTxRx, BYTE BaudValue, BYTE PreScale)
{
  /*** Set clocking iff either TX or RX is a master ***/
    if(MasterTxRx != 0)
    {
      /* If 0 using PSCLK, therefore set PreScale */
    if((_GetEXRegByte(SIOCFG) & BIT2MSK) == 0)
      _SetEXRegByte(CLKPRS, PreScale);
```

```
      /* Init Baud Rate Generator */
     _SetEXRegByte(SSIOBAUD,BaudValue);
    }
    _SetEXRegByte(SSIOCON1,Mode);
    _SetEXRegByte(SSIOCON2,MasterTxRx);

}/* InitSSIO */


/*****************************************************************************
    SSerialReadWord:

    Description:
       Is a Polled serial port read function that will wait forever
       or until a character has been received from the serial port.

    Parameters:
       MasterSlave      Defines if receiver is in Master or Slave mode

    Returns:
       Word read from serial port

    Assumptions:
       In Slave Mode, receiver must be enabled prior to this function call.

    Syntax:

    #define SSIO_RX_MASTR         0x1
    #define SSIO_RX_SLAVE         0x0

    WORD character;

    character = SSerialReadWord( SSIO_RX_MASTR );

    Real/Protected Mode:
     No changes required.

*****************************************************************************/

WORD SSerialReadWord(BYTE MasterSlave)
{
    register BYTE SSControl;
    if(MasterSlave == SSIO_RX_MASTR)
    {
       /* Save Control Register */
       SSControl = _GetEXRegByte(SSIOCON1);
       /* Get Control Register Ready to disable */
       SSControl &= (~SSIO_RX_ENAB);// Clear the bit
       /* Enable Receiver */
       _SetEXRegByte(SSIOCON1, SSControl | SSIO_RX_ENAB);
       /* Wait until Receive Holding Buffer is Full */
       while( !(_GetEXRegByte(SSIOCON1) & SSIO_RHBF) );
```

```
      /* Disable Receiver */
      _SetEXRegByte(SSIOCON1, SSControl);
   }
   else  {  // Slave Receiver, Receiver MUST already be Enabled
         /* Wait until Receive Holding Buffer is Full */
         while(!(_GetEXRegByte(SSIOCON1) & SSIO_RHBF) );
   }
   return (WORD)_GetEXRegWord(SSIORBUF);

}/* SSerialReadWord */


/*****************************************************************************
   SSerialWriteWord:

   Description:
      Is a Polled serial port write function that will wait forever
      or until a character has been written to the serial port.

   Parameters:
      Ch                   Word to be written out to serial port
      MasterSlave          Defines whether transmitter is Master or Slave

   Returns:
    None

   Assumptions:
   If transmitter is in Slave mode, it must already be enabled.

   Syntax:

   #define SSIO_TX_MASTR         0x2
   #define SSIO_TX_SLAVE         0x0

      char Ch = 'a';

      SSerialWriteWord( (WORD)Ch, SSIO_TX_MASTR );

   Real/Protected Mode:
      No changes required.

*****************************************************************************/

void SSerialWriteWord(WORD Ch,BYTE MasterSlave)
{
   register BYTE SSControl;
   unsigned int i;

   if(MasterSlave == SSIO_TX_MASTR)
   {
      /* Save Control Register */
      SSControl = _GetEXRegByte(SSIOCON1);
```

```
      /* Get Control Register Ready to disable */
      SSControl &= (~SSIO_TX_ENAB);      // Clear the bit

      /* Set Buffer to Character */
      _SetEXRegWord(SSIOTBUF,Ch);

      /* Enable Transmitter */
      _SetEXRegByte(SSIOCON1, SSControl | SSIO_TX_ENAB);

      /* Wait until Transmit Holding Buffer is empty */
      while( !(_GetEXRegByte(SSIOCON1) & SSIO_THBE) );
      for(i=0;i < 4000; i++) {  // Delay so transmit begins before disable
        _asm {
                      nop
        }
      }

      /* Disable Transmitter */
      _SetEXRegByte(SSIOCON1, SSControl);
   }
   else               // Slave, Transmitter MUST already be Enabled
   {
      /* Wait until Transmit Holding Buffer is empty */
      while( !(_GetEXRegByte(SSIOCON1) & SSIO_THBE) );
      _SetEXRegWord(SSIOTBUF,Ch);// Write to Buffer
   }

}/* SSerialWriteWord */




/*****************************************************************************

   SSIO_ISR:

   Description:
      Interrupt Service Routine for SSIO generated interrupts. This ISR
      identifies the cause of the interrupt and calls the appropriate
      routine.

   Parameters:
   None

   Returns:
    None

   Assumptions:
   It is assumed that the Slave 8259 is operating in Fully Nested Mode.
   If the Slave were in SMM, a Specific EOI would have to be sent to the
```

```
   Slave to clear the in-service bit.
   It is also assumed that the Master is not operating in AEOI, SFNM, or
   SMM. If the Master were in SMM or SFNM, a Specific EOI would have to
   be used. On the other hand, if the Master were operating in AEOI mode,
   no EOI signal would have to be sent.

   Syntax:
   Not called by user

   Real/Protected Mode:
   No changes required.

*****************************************************************************/

void interrupt far SSIO_ISR (void)
{

  Control = _GetEXRegByte(SSIOCON1);

  /* If THBE is set and Transmitter Interrupts are enabled */
  if ((Control & SSIO_THBE) && (Control & SSIO_TX_IE)) {
    Service_THBE(); // Service routine specific to THBE interrupts
  }

  /* Else if RHBF is set and Receiver Interrupts are enabled */
  else if ((Control & SSIO_RHBF) && (Control & SSIO_RX_IE)) {
    Service_RHBF(); // Service routine specific to RHBF interrupts
  }


  NonSpecificEOI(); // For Slave
  NonSpecificEOI(); // For Master


}/* SSIO_ISR */



/*****************************************************************************

   Service_RHBF:

   Description:
      Service Routine for SSIO interrupts generated by the RHBF signal.

   Parameters:
      None

   Returns:
      None

   Assumptions:
```

```
      None

   Syntax:
      Not called by user

   Real/Protected Mode:
      No changes required.

*************************************************************************/

void Service_RHBF(void)
{

  WORD buffer;

  buffer = _GetEXRegWord(SSIORBUF);

  /* Display received character on the screen */
  SerialWriteChar(SIO_0, (BYTE)buffer);

}/* Service_RHBF */


/*************************************************************************

   Service_THBE:

   Description:
      Service routine for SSIO interrupts generated by THBE signal.

   Parameters:
      None

   Returns:
      None

   Assumptions:
      None

   Syntax:
      Not called by user

   Real/Protected Mode:
      No changes required.

*************************************************************************/

void Service_THBE(void)
{

  int i;
  if ( value <= '9') {
```

```
    _SetEXRegWord(SSIOTBUF, value);
    value++;
  }

  else {
    /* Disable Transmitter and Transmitter interrupts */

    for(i=0;i < 4000; i++) {  // Delay so transmit begins before disable
      _asm {
        nop
      }
    }
    _SetEXRegByte(SSIOCON1,_GetEXRegByte(SSIOCON1) & 0xcf); // Clear TEN, TIE
  }

} /* Service_THBE */

/*****************************************************************************

  Example Code showing SSIO transfer in which the transmitter is
    interrupt-driven and the receiver is polled:


  InitSSIO(SSIO_RX_ENAB | SSIO_TX_ENAB | SSIO_TX_IE,
          SSIO_TX_MASTR | SSIO_RX_SLAVE, 0xF0, 0);

  // Setup  SSIO interrupts
  _SetEXRegByte(INTCFG, _GetEXRegByte(INTCFG) & 0xfd); // Slave IR1 is
                                                       // multiplexed
  SetIRQVector(SSIO_ISR, 9, INTERRUPT_ISR); // SSIO IR will be generated
                                            // on Slave IR1

  Disable8259Interrupt(IR1+IR3+IR4+IR5+IR6+IR7, IR0+IR2+IR3+IR4+IR5+IR6+IR7);
  Enable8259Interrupt(IR2,IR1); // Enable slave interrupt to master(IR2),
                                // Enable slave IR1
  _enable(); // Enable Interrupts

  // Initialize SSIO Ports
  _SetEXRegByte(PINCFG, _GetEXRegByte(PINCFG) & 0xfc);
  _SetEXRegByte(SIOCFG, _GetEXRegByte(SIOCFG) & 0xfb);


  // Fill up transmit buffer with first character
  _SetEXRegWord(SSIOTBUF, 'a');

  // Use Polled SSIO receiver function to receive character
  while ( input < 'z' ) {
    input = SSerialReadWord(SSIO_RX_SLAVE);
    SerialWriteChar(SIO_0, (BYTE)input); // Print to screen
  }
*****************************************************************************/
```

14

CHIP-SELECT UNIT

CHAPTER 14
CHIP-SELECT UNIT

The Chip-select Unit (CSU) of the processor can be used to eliminate external address and bus-cycle decoders in your system. The chip-selects generated by this unit can simplify external "glue logic" by providing signals that can be connected directly to the chip-enable inputs of external memory and I/O devices. If a particular device or address region does not require a chip-enable signal, a chip-select region can be programmed only to enable termination of accesses to that region. A chip-select region can also be programmed to generate a chip-enable signal and terminate accesses to that region.

The chip-select unit provides eight signals, or *channels*, allowing direct access to up to eight devices or address regions. You can individually configure the channels for compatibility with a variety of devices. Each channel can operate in either 16-bit or 8-bit bus mode, generate up to 31 wait states, and either terminate a bus cycle automatically or wait for an external ready signal.

This chapter is organized as follows:

14.1 OVERVIEW

Each chip-select channel consists of address and mask registers and an output signal. The address and mask registers allow you to define memory or I/O address blocks for each channel. You also specify whether or not the chip-select is activated when the processor is operating in system management mode. When the processor accesses a channel's address block, the CSU activates the channel's output signal. Connecting a channel's output to a memory or I/O device simplifies memory and I/O interfacing by removing the need for and delay of decoding addresses externally.

NOTE

Chip-select channels are not activated during interrupt acknowledge cycles and halt and shutdown cycles.

14.2 CSU UPON RESET

Upon reset of the processor, only the UCS channel is enabled and all other chip-selects are disabled. UCS is enabled for the entire memory space of the processor.

The UCS region is initialized upon reset with the following settings:

- Mask set to 7FFFH (CM25:11 in UCSMSKH and UCSMSKL registers)
- CMSMM set
- 16-bit bus size
- Memory access
- External READY# ignored
- 15 wait states

With all the UCS mask bits set to 1, the UCS# pin is active for the entire 64 MBytes of the processor's memory address space. The UCS region can be programmed for a smaller size during initialization. Normally, UCS# is used to select non-volatile memory devices, such as ROM and FLASH, at the top of the memory address space so that the processor can fetch the first instruction from address 3FFFFF0H after RESET. If the Port92 CPU-only RESET is used (described in Chapter 5), the UCS channel must remain enabled for the top of the memory address space (a CPU-only RESET does not affect the chip-select registers) and therefore, the UCS channel does not re-initialize to its reset state.

14.3 CSU OPERATION

Each chip-select channel functions independently. The following sections describe chip-select channel address blocks, system management mode support, and bus cycle length and bus size control.

14.3.1 Defining a Channel's Address Block

A 15-bit channel address and mask are used to specify a channel's active address block. When the processor accesses an address in memory or I/O, the upper 15 bits of the address are compared to the chip-select channel address and OR'd with the channel mask. This means that the CSU compares the channel address and ORs the channel mask to A25:11 for memory addresses and A15:1 for I/O addresses. Ones in the channel's mask exclude the corresponding bits from address comparisons. Figure 14-1 shows the logic for determining address equality.

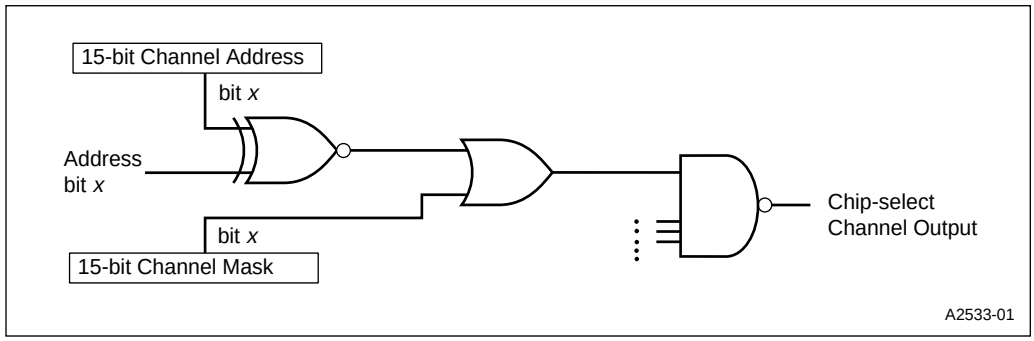


Figure 14-1. Channel Address Comparison Logic

The lower address bits are excluded from address comparisons (only 15 bits are compared). For memory addresses which have 26-bit addresses, the minimum channel address block size is 2 Kbytes; for I/O addresses with 16-bit addresses, the minimum channel address block size is 2 bytes.

NOTE

The starting address of any channel address block must be a multiple of the block size. For example, a 256 Kbyte block can only start at an address that is a multiple of 256 Kbytes (0H, 4000H, 8000H, etc.).

Because you can set ones in the channel mask to exclude certain address bits from comparisons, you can increase the size of a channel's address block (by powers of 2 in Kbytes for memory addresses and by powers of 2 in bytes for I/O addresses). Figure 14-2 illustrates how memory address block sizes are determined from the channel's mask; the concept is the same for I/O address block sizes (replace Kbyte with byte). As shown in Figure 14-2, the bit location of the right-most zero in the channel mask determines the channel's active address block size.

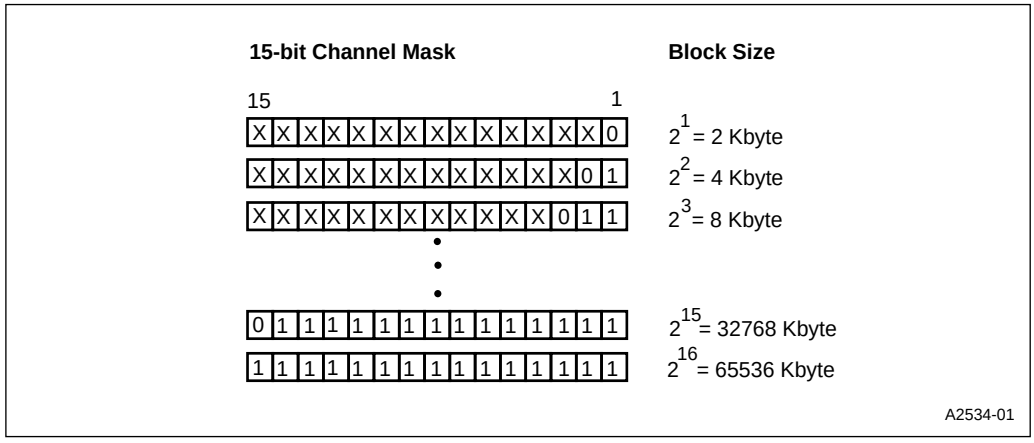
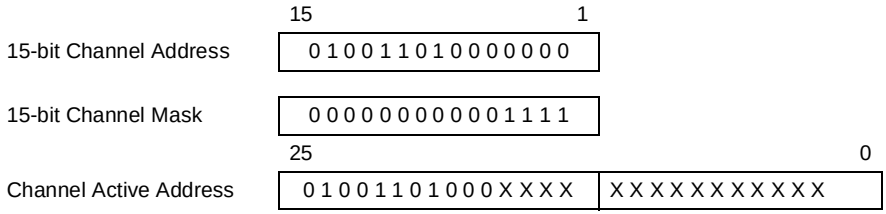


Figure 14-2. Determining a Channel's Address Block Size

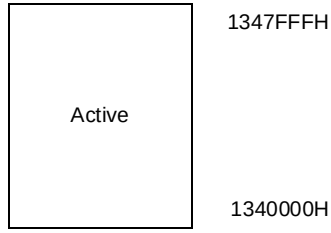
Any ones that are to the left of the right-most zero determine the number of blocks and the locations where the blocks are repeated. This is best illustrated by the following four examples. The examples assume the channel is configured for memory addresses; however, the concepts discussed also apply to I/O-configured channels.

Example 1

This example establishes a single 32-Kbyte address block starting at 1340000H (a 32-Kbyte boundary). In this example, the 15-bit channel address is the starting address of the channel's active address block (because there are no 1's in the channel mask where there are 1's in the channel address)

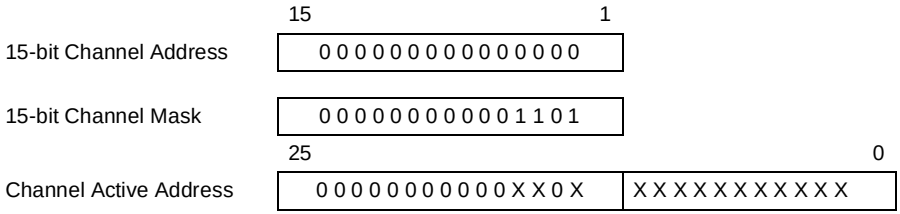


Because the least-significant 0 in the channel's mask is in bit position 5, this channel's active address block size is 2^5 = 32 Kbytes. Because there are no 1's to the left of the right-most 0 in the channel's mask, the block is not repeated.

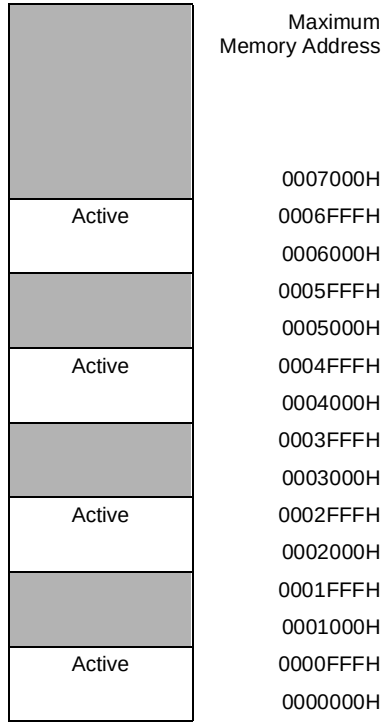


Example 2

This example establishes four 4-Kbyte address blocks starting at 0000000H, 0002000H, 0004000H, and 0006000H (4-Kbyte boundaries).

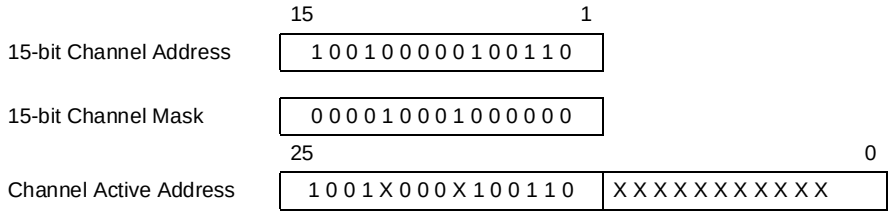


Because the least-significant 0 in the channel's mask is in bit position 2, this channel's active address block size is $2^2 = 4$ Kbytes. Because there are two 1's to the left of the right-most 0 in the channel's mask, the block is repeated $2^2 = 4$ times. Also, because there are no 1's in the channel mask where there are 1's in the channel address, the channel address is the starting address of the lowest active address block. In this example, each active 4-Kbyte address block in memory is followed by an inactive 4-Kbyte address block and each active address block starts on a 4-Kbyte address boundary.

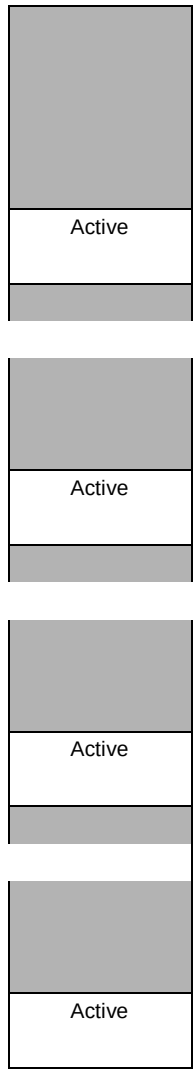


Example 3

This example establishes four 2-Kbyte address blocks starting at 2413000H, 2433000H, 2613000H, and 2633000H.

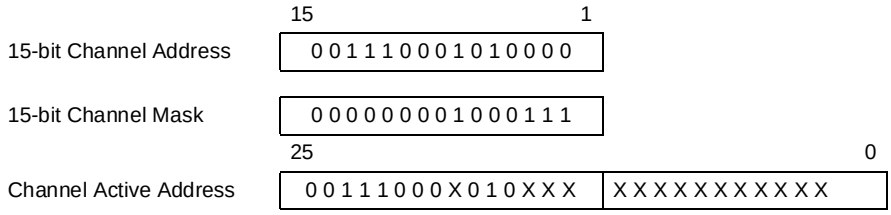


Because the least-significant 0 in the channel's mask is in bit position 1, this channel's active address block size is $2^1 = 2$ Kbytes. Because there are two 1's to the left of the right-most 0 in the channel's mask, the address block is repeated $2^2 = 4$ times. Also, because there are no 1's in the channel mask where there are 1's in the channel address, the channel address is the starting address of the lowest active address block. In this example, each active 2-Kbyte address block in memory is followed by an inactive 2-Kbyte address block and each active address block starts at a 2-Kbyte boundary.

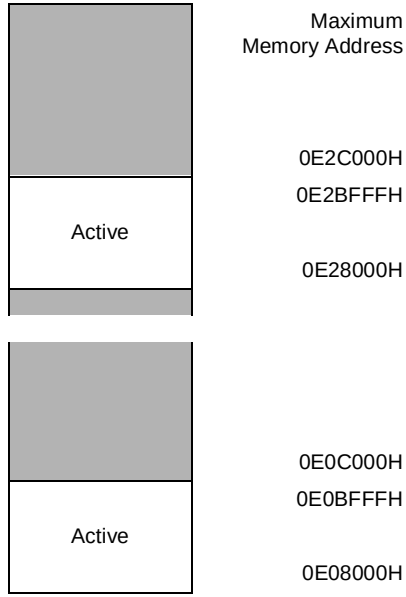

Maximum Memory Address
2633800H
Active
26337FFH
2633000H
2613FFFH
2613800H
Active
26137FFH
2613000H
2433FFFH
2433800H
Active
24337FFH
2433000H
2432FFFH
2413800H
Active
24137FFH
2413000H

Example 4

This example establishes two 16-Kbyte address blocks starting at 0E08000H and 0E28000H (16-Kbyte boundaries).



Because the least-significant 0 in the channel mask is in bit position 4, this channel's active address block size is 2^4 = 16 Kbytes. Because there is one 1 to the left of the right-most 0 in the channel mask, the address block is repeated 2^1 = 2 times. Unlike the other examples, there is a 1 in the channel mask where there is a 1 in the channel address. For this reason, the channel address is **not** the starting address of the lowest active address block. In this example, each active 16-Kbyte address block is followed by an inactive 16-Kbyte address block and each block starts at a 16-Kbyte address boundary.



14.3.2 System Management Mode Support

The processor supports four operating modes: system management mode (SMM), protected, real and virtual-86 mode. In order for a system to operate correctly in SMM, it must meet several requirements. The CSU provides support for some of these requirements. To use SMM, you must set aside a partition of memory, called SMRAM, for the SMM driver. SMRAM must meet the following conditions:

- Located at 38000H–3FFFFH (32 Kbytes)
- Accessible only when the processor is in SMM during normal operation
- Accessible during system initialization when the processor is not in SMM

The CSU allows you to specify an address block and control whether or not the chip-select is activated while the processor is in SMM. While in SMM (with CASMM=1), the chip-select is active only when the processor is the bus master, such as when the processor is not in a hold state.

Refer to Chapter 7 ("Programming Considerations" on page 7-16) for a code example of programming chip-selects to support SMM.

14.3.3 Bus Cycle Length Control

Each chip-select channel controls how bus cycles to its address block terminate. Each channel can generate up to 31 wait states and then unconditionally terminate or wait for an external bus ready signal to terminate. If the channel is programmed for wait states and to sample external READY#, the external READY# is ignored until the programmed number of wait states has been inserted into the cycle. If greater than 31 wait states are required, ready must be generated externally, and the external READY# option must be selected.

NOTE

When a chip-select region overlaps on-chip peripheral addresses, the on-chip peripheral always generates READY# and overrides the channel's configuration.

14.3.4 Bus Size Control

The processor assumes that the currently addressed device requires a 16-bit data bus unless the bus size control pin (BS8#) is asserted. When asserted, BS8# tells the processor that the addressed device requires an 8-bit data bus. You can program a chip-select channel specifically for 8-bit devices. This causes the CSU to assert BS8# automatically each time it activates the channel.

14.3.5 Overlapping Regions

You can configure CSU channels to have overlapping address blocks. When channels with overlapping address blocks have different bus cycle length and bus size configurations, the CSU must adjust these parameters. Figure 14-3 shows how the CSU adjusts the bus cycle length. In the case of different bus sizes, the CSU defaults to an 8-bit bus size.

If one overlapping chip-select region has the RDY bit set and the other overlapping region does not, the CSU defaults to the 'RDY Bit Set' operation; in this case an external READY# is necessary to terminate accesses to the address locations in which the two chip-selects overlap.

NOTE

If a bus cycle address activates multiple overlapping CSU channels, all the enabled chip-select signals of those channels go active. To avoid contention on the data bus, care must be taken when using these chip-select signals externally.

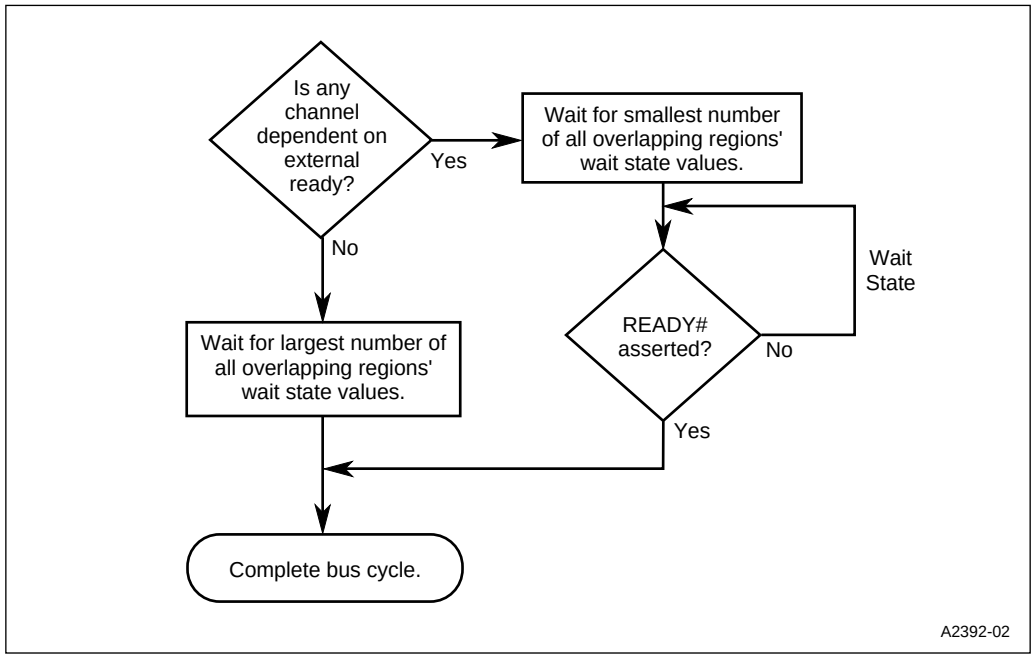


Figure 14-3. Bus Cycle Length Adjustments for Overlapping Regions

14.4 REGISTER DEFINITIONS

Table 14-1 and Table 14-2 list the signals and registers associated with the chip-select unit. There are seven general-purpose chip-select channels (CS*n*) and one upper chip-select channel (UCS). Upon reset, the UCS is enabled with the entire 64 Mbyte memory address space as its address block. The UCS can be used to select a memory device at the top of the memory address space so that the processor can fetch the first instruction from address 3FFFFF0H after reset.

Table 14-1. CSU Signals

Signal	Device Pin or Internal Signal	Description
CS6:0# UCS#	Device pins (output)	Chip-select Signals: Indicates that the memory or I/O address that the processor is accessing is in channel *n*'s active address region.

Table 14-2. CSU Registers

Register	Expanded Address	Description
PINCFG (read/write)	0F826H	Pin Configuration: Connects the CS6:5# signals to package pins.
P2CFG (read/write)	0F822H	Port 2 Configuration: Connects the CS4:0# signals to package pins.
CS0ADH CS1ADH CS2ADH CS3ADH CS4ADH CS5ADH CS6ADH UCSADH (read/write)	0F402H 0F40AH 0F412H 0F41AH 0F422H 0F42AH 0F432H 0F43AH	Chip-select High Address: Defines the upper 10 bits of the chip-select channel address. The processor uses a chip-select's channel address to determine the starting location of the channel's active address block.
CS0ADL CS1ADL CS2ADL CS3ADL CS4ADL CS5ADL CS6ADL UCSADL (read/write)	0F400H 0F408H 0F410H 0F418H 0F420H 0F428H 0F430H 0F438H	Chip-select Low Address: Defines the lower 5 bits of the chip-select channel address. Configures the channel for memory or I/O addresses, determines whether or not the channel is activated when the processor is operating in system management mode, configures the channel's bus size, defines the minimum number of wait states inserted into the bus cycle, and defines whether an external READY# is required to terminate the bus cycle.
CS0MSKH CS1MSKH CS2MSKH CS3MSKH CS4MSKH CS5MSKH CS6MSKH UCSMSKH (read/write)	0F406H 0F40EH 0F416H 0F41EH 0F426H 0F42EH 0F436H 0F43EH	Chip-select High Mask: Defines the upper 10 bits of the chip-select channel mask. The processor uses a chip-select's channel mask to determine the size of the channel's active address block and if the address block is repeated.
CS0MSKL CS1MSKL CS2MSKL CS3MSKL CS4MSKL CS5MSKL CS6MSKL UCSMSKL (read/write)	0F404H 0F40CH 0F414H 0F41CH 0F424H 0F42CH 0F434H 0F43CH	Chip-select Low Mask: Defines the lower 5 bits of the chip-select channel mask and enables the channel's output pin.

14.4.1 Pin Configuration Register (PINCFG)

Use PINCFG bits 6 and 4 to connect the CS6# and CS5# signals to package pins.

Pin Configuration
PINCFG
(read/write)

Expanded Addr: **F826H**
ISA Addr: **—**
Reset State: **00H**

7							0
—	PM6	PM5	PM4	PM3	PM2	PM1	PM0

Bit Number	Bit Mnemonic	Function
7	—	Reserved. This bit is undefined; for compatibility with future devices, do not modify this bit.
6	PM6	Pin Mode: 0 = Selects CS6# at the package pin. 1 = Selects REFRESH# at the package pin.
5	PM5	Pin Mode: 0 = Selects the coprocessor signals, PEREQ, BUSY#, and ERROR#, at the package pins. 1 = Selects the timer control unit signals, TMROUT2, TMRCLK2, and TMRGATE2, at the package pins.
4	PM4	Pin Mode: 0 = Selects DACK0# at the package pin. 1 = Selects CS5# at the package pin.
3	PM3	Pin Mode: 0 = Selects EOP# at the package pin. 1 = Selects CTS1# at the package pin.
2	PM2	Pin Mode: 0 = Selects DACK1# at the package pin. 1 = Selects TXD1 at the package pin.
1	PM1	Pin Mode: 0 = Selects SRXCLK at the package pin. 1 = Selects DTR1# at the package pin.
0	PM0	Pin Mode: 0 = Selects SSIOTX at the package pin. 1 = Selects RTS1# at the package pin.

Figure 14-4. Pin Configuration Register (PINCFG)

14.4.2 Port 2 Configuration Register (P2CFG)

Use P2CFG bits 4–0 to connect the CS4:0# signals to package pins.

Port 2 Configuration
P2CFG
(read/write)

Expanded Addr: **F822H**
ISA Addr: **—**
Reset State: **00H**

7							0
PM7	PM6	PM5	PM4	PM3	PM2	PM1	PM0

Bit Number	Bit Mnemonic	Function
7	PM7	Pin Mode: 0 = Selects P2.7 at the package pin. 1 = Selects CTS0# at the package pin.
6	PM6	Pin Mode: 0 = Selects P2.6 at the package pin. 1 = Selects TXD0 at the package pin.
5	PM5	Pin Mode: 0 = Selects P2.5 at the package pin. 1 = Selects RXD0 at the package pin.
4	PM4	Pin Mode: 0 = Selects P2.4 at the package pin. 1 = Selects CS4# at the package pin.
3	PM3	Pin Mode: 0 = Selects P2.3 at the package pin. 1 = Selects CS3# at the package pin.
2	PM2	Pin Mode: 0 = Selects P2.2 at the package pin. 1 = Selects CS2# at the package pin.
1	PM1	Pin Mode: 0 = Selects P2.1 at the package pin. 1 = Selects CS1# at the package pin.
0	PM0	Pin Mode: 0 = Selects P2.0 at the package pin. 1 = Selects CS0# at the package pin.

Figure 14-5. Port 2 Configuration Register (P2CFG)

14.4.3 Chip-select Address Registers

The Address Register of each chip-select channel defines the address block that the channel responds to during an access. The value in this register is compared to A25:11 of the processor bus during a memory access and to A15:1 during an I/O access. A bus cycle whose address matches the non-masked (see "Chip-select Mask Registers" on page 14-19) bits of the Address Register causes the respective chip-select channel to have an address match. Even if there is an address match, whether or not the CSU activates the channel depends on the values of the channel's SMM address and mask bits (CASMM and CMSMM) and the chip-select channel enable bit (CSEN). The CASMM and CMSMM bits determine whether or not the channel is activated when the processor is operating in SMM.

Write a channel's 15-bit address to the chip-select address registers. These bits are masked by the channel's 15-bit mask.

NOTE

When a chip-select channel is activated, it either asserts a chip-select signal, controls wait states and READY# generation, or both.

Chip-select High Address CS*n*ADH (n = 0–6), UCSADH (read/write)

Expanded Addr: **F402H, F40AH F412H, F41AH F422H, F42AH F432H, F43AH**
ISA Addr: **—**
Reset State: **0000H (CS*n*ADH) FFFFH (UCSADH)**

15							8
—	—	—	—	—	—	CA15	CA14
7							**0**
CA13	CA12	CA11	CA10	CA9	CA8	CA7	CA6

Bit Number	Bit Mnemonic	Function
15–10	—	Reserved; for compatibility with future devices, write zeros to these bits.
9–0	CA15:6	Chip-select Channel Address Upper Bits: Defines the upper 10 bits of the channel's 15-bit address. The address bits CA15:6 and the mask bits CM15:6 form a masked address that is compared to memory address bits A25:16 or I/O address bits A15:6.

Figure 14-6. Chip-select High Address Register (CS*n*ADH, UCSADH)

Chip-select Low Address
CS*n*ADL (*n* = 0–6), UCSADL
(read/write)

Expanded Addr: **F400H, F408H**
F410H, F418H
F420H, F428H
F430H, F438H
ISA Addr: **—**
Reset State: **0000H (CS*n*ADL)**
FF6FH (UCSADL)

15							**8**
CA5	CA4	CA3	CA2	CA1	CASMM	BS16	MEM
7							**0**
RDY	—	—	WS4	WS3	WS2	WS1	WS0

Bit Number	Bit Mnemonic	Function
15–11	CA5:1	Chip-select Address Value Lower Bits: Defines the lower 5 bits of the channel's 15-bit address. The address bits CA5:1 and the mask bits CM5:1 form a masked address that is compared to memory address bits A15:11 or I/O address bits A5:1.
10	CASMM	SMM Address Bit: If this bit is set (and unmasked), the CSU activates the chip-select channel only while the processor is in SMM (and not in a hold state). Otherwise, the CSU activates the channel only when processor is operating in a mode other than SMM. Setting the SMM mask bit in the channel's mask low register masks this bit. When this bit is masked, an address match activates the chip-select, regardless of whether the processor is in SMM or not.
9	BS16	Bus Size 16-bit: 0 = All bus cycles to addresses in the channel's address block are byte-wide. 1 = Bus cycles are 16 bits unless the bus size control pin (BS8#) is asserted.
8	MEM	Bus Cycle Type: 0 = Configures the channel for an I/O addresses 1 = Configures the channel for memory addresses
7	RDY	Bus Ready Enable: 0 = External READY# is ignored. READY# generated by CSU to terminate the bus cycle. 1 = Requires that external READY# be active to complete a bus cycle. This bit must be set to extend wait states beyond the number determined by WS4:0 (see "Bus Cycle Length Control" on page 14-11).
6–5	—	Reserved; for compatibility with future devices, write zeros to these bits.
4–0	WS4:0	Wait State Value: WS4:0 defines the minimum number of wait states inserted into the bus cycle. A zero value means no wait states.

Figure 14-7. Chip-select Low Address Register (CS*n*ADL, UCSADL)

14.4.4 Chip-select Mask Registers

The Mask Register of each chip-select region is used to prevent bits from being compared with the starting address, thus masking them from the comparison. This masking allows you to specify the size of the region being defined. The mask should be set such that it masks the lower address bits being compared, up to the size that you would like the block to be.

Write a channel's 15-bit mask to the chip-select mask registers. Also, use the chip-select low mask register to enable the channel and to mask the channel's SMM address bit. When the channel's SMM address bit is masked, the CSU activates the channel even if the channel is operating in SMM.

Chip-select High Mask
CS*n*MSKH (*n* = 0–6), UCSMSKH
(read/write)

Expanded Addr: **F406H, F40EH**
F416H, F41EH
F426H, F42EH
F436H, F43EH
ISA Addr: **—**
Reset State: **0000H (CS*n*MSKH)**
FFFFH (UCSMSKH)

15							**8**
—	—	—	—	—	—	CM15	CM14
7							**0**
CM13	CM12	CM11	CM10	CM9	CM8	CM7	CM6

Bit Number	Bit Mnemonic	Function
15–10	—	Reserved; for compatibility with future devices, write zeros to these bits.
9–0	CM15:6	Mask Value Upper Bits: Defines the upper 10 bits of the channel's 15-bit mask. The mask bits CM15:6 and the address bits CA15:6 form a masked address that is compared to memory address bits A25:16 or I/O address bits A15:6.

Figure 14-8. Chip-select High Mask Registers (CS*n*MSKH, UCSMSKH)

Chip-select Low Mask
CS*n*MSKL (*n* = 0–6), UCSMSKL
(read/write)

Expanded Addr: **F404H, F40CH**
F414H, F41CH
F424H, F42CH
F434H, F43CH
ISA Addr: **—**
Reset State: **0000H (CS*n*MSKL)**
FFFFH (UCSMSKL)

15							**8**
CM5	CM4	CM3	CM2	CM1	CMSMM	—	—
7							**0**
—	—	—	—	—	—	—	CSEN

Bit Number	Bit Mnemonic	Function
15–11	CM5:1	Chip-select Mask Value Lower Bits: Defines the lower 5 bits of the channel's 15-bit mask. The mask bits CM5:1 and the address bits CA5:1 form a masked address that is compared to memory address bits A15:11 or I/O address bits A5:1.
10	CMSMM	SMM Mask Bit: 0 = The SMM address bit is not masked. 1 = Masks the SMM address bit in the channel's Chip-Select Low Address register. When the SMM address bit is masked, an address match activates the chip-select, regardless of whether the processor is in SMM.
9–1	—	Reserved; for compatibility with future devices, write zeros to these bits.
0	CSEN	Chip-select Enable: 0 = Disables the chip-select channel. 1 = Enables the chip-select channel.

Figure 14-9. Chip-select Low Mask Registers (CS*n*MSKL, UCSMSKL)

14.5 DESIGN CONSIDERATIONS

When designing with the CSU, consider the following:

- Upon reset, UCS# is configured as a 16-bit chip-select signal. If the Boot device is only an 8-bit device, then BS8# must be asserted whenever UCS# is active (until the UCS channel can be reprogrammed to reflect an 8-bit region). One way of doing this is by connecting the UCS# pin directly to the BS8# pin, if there are no other devices that need to use the BS8# pin. If UCS# is tied directly to BS8#, then the UCS channel need not be programmed to reflect an 8-bit region.
- If the Port92 CPU-only RESET is used (described in Chapter 5), the UCS channel must remain enabled for the top of the memory address space (a CPU-only RESET does not affect the chip-select registers) and therefore, the UCS channel does not re-initialize to its reset state.
- If arbitrary chip-select regions are required to access external memory and I/O devices and a single channel can not be programmed to accommodate the address space of these regions, multiple chip-select signals can be “ORed” to create a single chip-enable to a device. For example a 512 Kbyte region chip-select signal starting on a 256 Kbyte boundary can be created by “ORing” two 256 Kbyte chip-select signals.
- Refer to Chapter 6 (“Design Considerations” on page 6-38) for examples of using chip-select signals to access external devices.

14.6 PROGRAMMING CONSIDERATIONS

When programming the CSU, consider the following:

- When programming a chip-select channel, always program the Low Mask Register last. This ensures that all other bits are properly programmed before the region is enabled. When reprogramming the channel, always disable the channel before changing anything else.
- A chip-select channel is enabled by setting bit 0 of its Chip-Select Low Mask register and its output signal is connected to the package pin by setting or clearing the appropriate PINCFG or P2CFG register bit. The PINCFG and P2CFG registers are shown in Figures 14-4 and 14-5.
- The minimum address block for memory address-configured channels is 2 Kbytes and for I/O address-configured channels is 2 bytes. The size of these address blocks can be increased by powers of 2 Kbytes for memory addresses and by powers of 2 bytes for I/O addresses.
- A channel's address block of size *n* always starts on an *n* address boundary.

14.6.1 Chip-Select Unit Code Example

This following code example initializes the UCS and CS4 channels of the CSU. See Appendix C for the included header files.

```
#include <conio.h>
#include "80386ex.h"
#include "EV386EX.h"


/*
Description:

Initialize Chip Select Unit for:
  UCS:   Start address is 00H.
         Region size is 512 Kbytes.
         0 wait states.
         Upper chip select is Enabled.
         16 bit data bus size in memory space.
         External bus ready is Disabled.
         SMM region is accessible during SMI access and memory access.

  CS4:   Start address is 080000H.
         Region size is 512 Kbytes.
         0 wait states.
         Chip select 4 is Enabled.
         16 bit data bus size in memory space.
         External bus ready is Disabled.
         SMM region is accessible during SMI access only.

Parameters:
    None
```

```
Assumptions:
      REMAPCFG register has Expanded I/O space access enabled (ESE bit set).

*/


void Init_CSU(void)
{

  _SetEXRegWord(UCSADL, 0x700);  /* Configure the upper chip select */
  _SetEXRegWord(UCSADH, 0x0);
  _SetEXRegWord(UCSMSKL, 0xFC01);
  _SetEXRegWord(UCSMSKH, 0x7);

  _SetEXRegWord(CS4ADL, 0x300);  /* Configure chip select 4 */
  _SetEXRegWord(CS4ADH, 0x8);
  _SetEXRegWord(CS4MSKL, 0xF801);
  _SetEXRegWord(CS4MSKH, 0x7);

}
```

15

REFRESH CONTROL UNIT

CHAPTER 15
REFRESH CONTROL UNIT

The Refresh Control Unit (RCU) simplifies the interface between the processor and a dynamic random access memory (DRAM) device by providing a way to generate periodic refresh requests and refresh addresses. These refresh requests and addresses can then be used by an external DRAM controller to generate the appropriate DRAM signals and addresses needed to perform refresh operations. The RCU can be used in conjunction with the Chip-select Unit to generate chip select signals for DRAM regions; these signals can be used by the external DRAM controller to initiate refresh cycles.

The RCU can also be used when interfacing to pseudo-static random access memory (PSRAM). This type of memory has an interface similar to a static random access memory (SRAM), but requires a periodic refresh similar to DRAM.

This chapter is organized as follows:

15.1 DYNAMIC MEMORY CONTROL

Typical DRAM devices require control logic to enable read, write, and refresh operations. The RCU simplifies control logic design requirements by providing the necessary cell access requirements for refresh operations.

DRAM devices are built as matrices of memory cells. Therefore, each memory cell has a row and column address associated with it. A typical controller design strobes addresses into a DRAM device through the use of two control lines: a row address strobe (RAS#) and a column address strobe (CAS#). The controller presents lower (or row) address bits during RAS# and upper (or column) address bits during CAS#. Activating RAS# accesses all cells within the specified row. Accessing a cell refreshes it; therefore, cycling through the row addresses refreshes a DRAM device.

15.1.1 Refresh Methods

There are two common methods for refreshing a DRAM device: RAS#-only and CAS#-before-RAS#. The DRAM controller design requirements are simpler for RAS#-only than for CAS#-before-RAS#.

The RAS#-only method requires that the DRAM controller activate its RAS# signal when the RCU activates its REFRESH# signal. This causes the controller to drive the refresh address generated by the RCU onto the DRAM address inputs, refreshing the specified DRAM row. With this method, the controller need not assert the CAS# signal whenever the REFRESH# signal is active.

The CAS#-before-RAS# method requires that the DRAM device contain an internal counter to determine the DRAM row addresses. To perform a refresh cycle using the CAS#-before-RAS# method, the controller must generate a CAS# signal followed by a RAS# signal when the RCU activates its REFRESH# signal. With this method, the DRAM device generates its own refresh addresses and the RCU provides the REFRESH# signal.

If the CS6#/REFRESH# pin is being used for its CS6# function, another way of identifying a refresh cycle is to look at the states of the bus status signals, M/IO#, D/C# and W/R#, (shown in Table 6-2 on page 6-5) and the byte-enable signals (BHE# and BLE#). M/IO# and D/C# are high, W/R# is low, and both BHE# and BLE# are inactive during a refresh cycle. These signals can be used by the DRAM controller to initiate a DRAM refresh cycle.

15.2 REFRESH CONTROL UNIT OVERVIEW

The RCU includes an interval timer unit, a control unit, and an address generation unit (Figure 15-1). The interval timer unit uses a refresh clock interval register and a 10-bit interval counter to create a periodic signal (**timeout**). The control unit uses this signal to initiate periodic refresh requests. The address generation unit uses a refresh base address register and a 13-bit address counter to generate DRAM refresh addresses. The DRAM device can use these addresses as row addresses during RAS-only refresh cycles. Each time the interval timer unit times out, a new refresh address is generated.

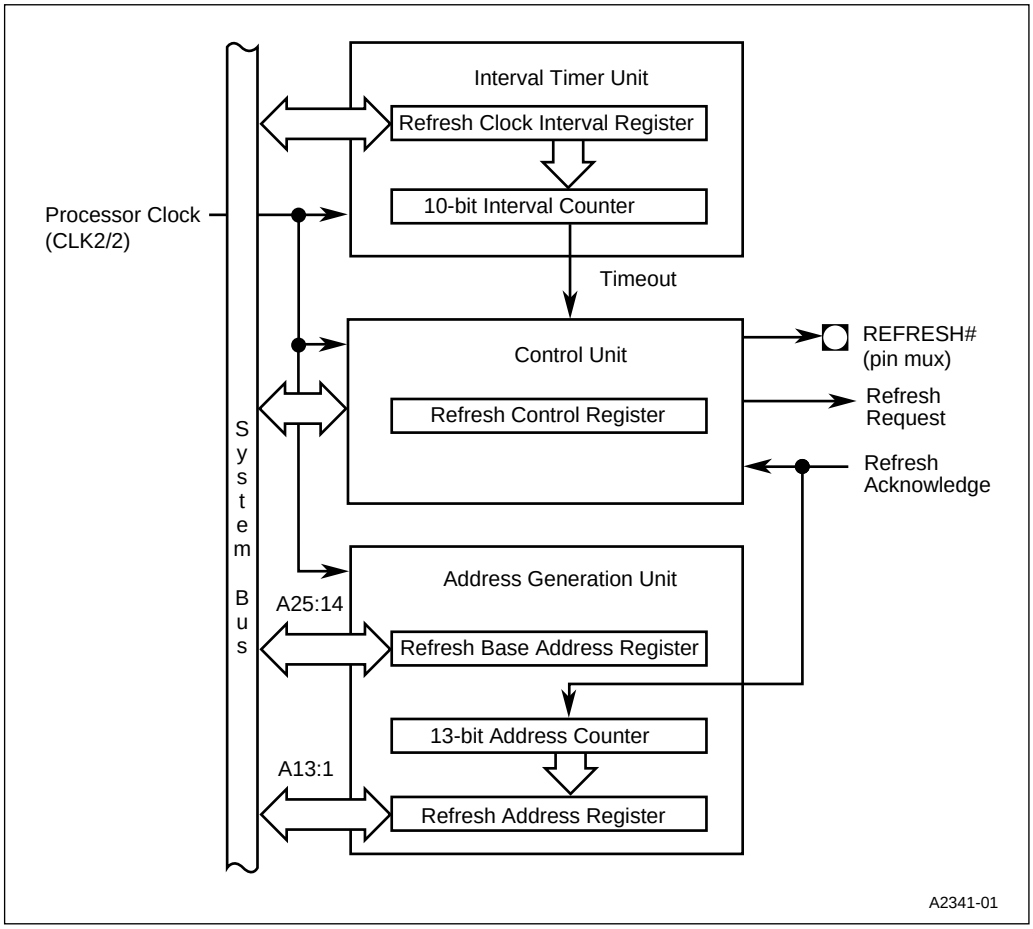


Figure 15-1. Refresh Control Unit Connections

15.2.1 RCU Signals

Table 15-1 describes the signals associated with the RCU.

Table 15-1. RCU Signals

Signal	Device Pin or Internal Signal	Description
CLKOUT	Device Pin (from Clock and Power Management Unit)	Processor Clock: Provides the clocking signal for the interval counter. The interval timer unit loads and decrements the counter on the falling edges of the processor clock.
Timeout	Internal signal (from the interval counter to the control unit)	Timeout: Indicates that the interval counter has reached one. The control unit initiates a refresh request when it detects this signal, unless a refresh request is pending, in which case it ignores this signal.
REFRESH#	Device pin (output)	External Refresh: Indicates that a refresh bus cycle is in progress and that the refresh address is on the bus.
Refresh Request	Internal signal	Refresh Request: Indicates that the control unit is requesting bus ownership.
Refresh Acknowledge	Internal signal	Refresh Acknowledge: Indicates that the refresh control unit is being granted bus ownership.
A25:1	Device pins (output)	Address Bus: Contains the refresh address during refresh cycles. This address can be used by the DRAM device to refresh a single row.

15.2.2 Refresh Intervals

The interval timer unit controls the rate at which the control unit generates refresh requests. Refresh intervals are programmable through the use of a refresh control interval register (RFSCIR) and a 10-bit down counter. The counter is loaded from RFSCIR, then decremented on each CLKOUT falling edge. When the counter reaches one, the interval timer unit reloads the counter from the RFSCIR and asserts its timeout signal. The timeout signal causes the control unit to initiate a refresh request, provided there is not one already pending. (The RCU must complete the present refresh cycle before the control logic can generate a new refresh request). The control unit ignores the timeout signal if it already has a refresh request pending.

15.2.3 Refresh Addresses

The physical address generated during a refresh bus cycle has two components: address bits A25:14 (from the refresh base address register) and address bits A13:1 (from the 13-bit address counter).

The 13-bit address counter is a combination of a binary counter and a 7-bit linear-feedback shift register. The binary counter produces address bits A13:8 and the linear-feedback shift register produces address bits A7:1. The shift register nonsequentially produces all 128 (2^7) possible combinations. Each time the lower seven bits cycle through all 128 combinations, the binary counter increments the upper 6 bits. This continues until the 13-bit address counter cycles through 8192 (2^{13}) address combinations. The counter then rolls over to its original value and the process repeats.

15.2.4 Bus Arbitration

Because the two DMA channels, an external device (via the HOLD pin), and the refresh control unit can all request bus control, bus control priority must be arbitrated. Refresh requests always have the highest priority. “Bus Control Arbitration” on page 12-9 discusses the priority structure of the other bus control requests.)

When a refresh occurs while a DMA channel is performing a transfer, the RCU “steals” a bus cycle to perform a refresh. An external device can gain bus control through either the HOLD signal or the DMA cascade mode. In this case, a refresh request causes the HLDA or DMACKn# signal to be deasserted. When this happens, the external device should deassert its request line (HOLD or DRQ*n*) to allow the RCU to perform a refresh cycle. The refresh cycle is not executed until the external device deasserts its request. If the external device reasserts its request signal before the RCU completes the refresh cycle, bus control is given back to the external device after the refresh cycle completes, without further arbitration.

15.3 RCU OPERATION

The following steps describe the basic refresh cycle, which is initiated every time the interval counter reaches one.

1. The interval timer unit asserts the timeout signal and reloads the interval counter with the refresh clock interval register value. The interval counter decrements on each succeeding processor clock falling edge.
2. The RCU requests bus ownership.
3. Bus ownership is given to the control unit.
4. The control unit asserts the REFRESH# signal and a bus memory read cycle (with neither Byte-enable signal active) is executed with the address supplied by the RCU.
5. The DRAM controller asserts RAS#, latching the row address inside the DRAM device. This refreshes the row.
6. The control unit deasserts REFRESH#, and the process repeats from step 1 when the interval counter reaches one again.

Once enabled, the DRAM refresh process continues until you reprogram the RCU, a reset occurs, or the processor enters powerdown mode.

15.4 REGISTER DEFINITIONS

Table 15-2 provides an overview of the registers associated with the RCU. The following sections provide specific programming information for each register.

Table 15-2. RCU Registers

Register	Expanded Address	Description
RFSCIR (read/write)	0F4A2H	Refresh Clock Interval: Determines the processor clock (CLK2/2) count between refresh requests.
RFSCON (read/write)	0F4A4H	Refresh Control: Enables the refresh control unit. Reading this register also provides the current value of the interval counter.
RFSBAD (read/write)	0F4A0H	Refresh Base Address: Contains the A25:14 address bits of the refresh address. This establishes a memory region for refreshing.
RFSADD (read/write)	0F4A6H	Refresh Address: Contains the A13:1 address bits of the refresh address. The 13-bit address counter generates these values.

15.4.1 Refresh Clock Interval Register (RFSCIR)

Use RFSCIR to program the interval timer unit's 10-bit down counter. The refresh counter value is a function of DRAM specifications and processor frequency as follows:

$$\text{counter value} = \frac{\text{DRAM refresh period } (\mu s) \times \text{processor clock (MHz)}}{X},$$

where X = 128 or the # of DRAM rows, whichever is greater.

The DRAM refresh period is the time required to refresh all rows in the DRAM device.

NOTE

Because the lower seven address bits come from a linear-feedback shift register, which generates all address bit combinations in a nonsequential order, X in the equation above must never be less than 128 to ensure proper refresh of all the rows in a DRAM device that has less than 128 rows.

Refresh Clock Interval
RFSCIR
(read/write)

Expanded Addr: **F4A2H**
ISA Addr: **—**
Reset State: **0000H**

15							8
—	—	—	—	—	—	RC9	RC8
7							**0**
RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0

Bit Number	Bit Mnemonic	Function
15–10	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
9–0	RC9:0	Refresh Counter Value: Write the counter value to these ten bits. The interval counter counts down from this value. When the interval counter reaches one, the control unit initiates a refresh request (provided it does not have a request pending). The counter value is a function of DRAM specifications and processor frequency (see the equation above).

Figure 15-2. Refresh Clock Interval Register (RFSCIR)

15.4.2 Refresh Control Register (RFSCON)

Use RFSCON to enable and disable the refresh control unit and to check the current interval counter value.

Refresh Control
RFSCON
(read/write)

Expanded Addr: **F4A4H**
ISA Addr: **—**
Reset State: **0000H**

15							8
REN	—	—	—	—	—	CV9	CV8
7							**0**
CV7	CV6	CV5	CV4	CV3	CV2	CV1	CV0

Bit Number	Bit Mnemonic	Function
15	REN	Refresh Control Unit Enable: This bit enables or disables the refresh control unit. 0 = Disables refresh control unit 1 = Enables refresh control unit
14–10	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
9–0	CV9:0	Counter Value: These read-only bits represent the current value of the interval counter. Write operations to these bits have no effect.

Figure 15-3. Refresh Control Register (RFSCON)

15.4.3 Refresh Base Address Register (RFSBAD)

Use RFSBAD to set up the memory region that needs refreshing. The value written to this register forms the upper bits (A25:14) of the refresh address. The RFSBAD register can be used in conjunction with the Chip Select Unit (CSU) to generate a chip-select for the DRAM region during refresh cycles. If the address in the RFSBAD matches the region programmed in the CSU for DRAM, then the DRAM chip-select is generated for both access and refresh cycles.

By programming two separate regions in the CSU, one for DRAM access cycles and the other for DRAM refresh cycles, separate chip-selects can be generated for the two types of cycles. In this case, the RFSBAD needs to be programmed with an address that matches the CSU region that is programmed for the refresh cycle chip-select.

Refresh Base Address
RFSBAD
(read/write)

Expanded Addr: **F4A0H**
ISA Addr: **—**
Reset State: **0000H**

15							8
—	—	—	—	RA25	RA24	RA23	RA22
7							**0**
RA21	RA20	RA19	RA18	RA17	RA16	RA15	RA14

Bit Number	Bit Mnemonic	Function
15–12	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
11–0	RA25:14	Refresh Base: These bits make up the A25:14 address bits of the refresh address. This establishes a memory region for refreshing.

Figure 15-4. Refresh Base Address Register (RFSBAD)

15.4.4 Refresh Address Register (RFSADD)

RFSADD contains the bits A13:1 of the refresh address. The lowest address bit is not used because most DRAM devices contain word-wide memory arrays; for all refresh operations, the lowest address bit remains set.

Refresh Address
RFSADD
(read/write)

Expanded Addr: **F4A6H**
ISA Addr: **—**
Reset State: **00FFH**

15							**8**
—	—	RA13	RA12	RA11	RA10	RA9	RA8
7							**0**
RA7	RA6	RA5	RA4	RA3	RA2	RA1	1

Bit Number	Bit Mnemonic	Function
15–14	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
13–1	RA13:1	Refresh Address Bits: These bits comprise A13:1 of the refresh address.
0	—	Refresh Bit 0: A0 of the refresh address. This bit is always 1 and is read-only.

Figure 15-5. Refresh Address Register (RFSADD)

15.5 DESIGN CONSIDERATIONS

Consider the following when programming the RCU.

- The system address bus does not contain an address A0 signal; instead, it uses the BLE# and the BHE# pins to generate the lowest address bit. During all refresh operations, BLE# and BHE# are driven high.

  This needs to be noted especially when interfacing to an 8-bit wide Pseudo Static RAM (PSRAM) device. The lowest address bit generated by the refresh address counter is A1. A circuit like the one shown in Figure 15-6 can be used to ensure the refresh of all rows. Here BLE# is connected to an address line of the PSRAM that is not used during refresh. Address A1 of the processor is connected to A0 of the PSRAM and so forth. For example, when using a 128Kx8-bit PSRAM device (refresh cycles only use the address present on inputs A8:0), connect A1 of the processor to A0 of the PSRAM, A2 to A1 and so on, until A9 to A8. Then connect BLE# of the processor to any one of the A16:9 address lines of the PSRAM. Since PSRAM is random access memory, this scheme works. During access cycles, sequential accesses by the processor go to non-contiguous addresses in the PSRAM, but since the processor does both the read and write cycles, this does not pose a problem.

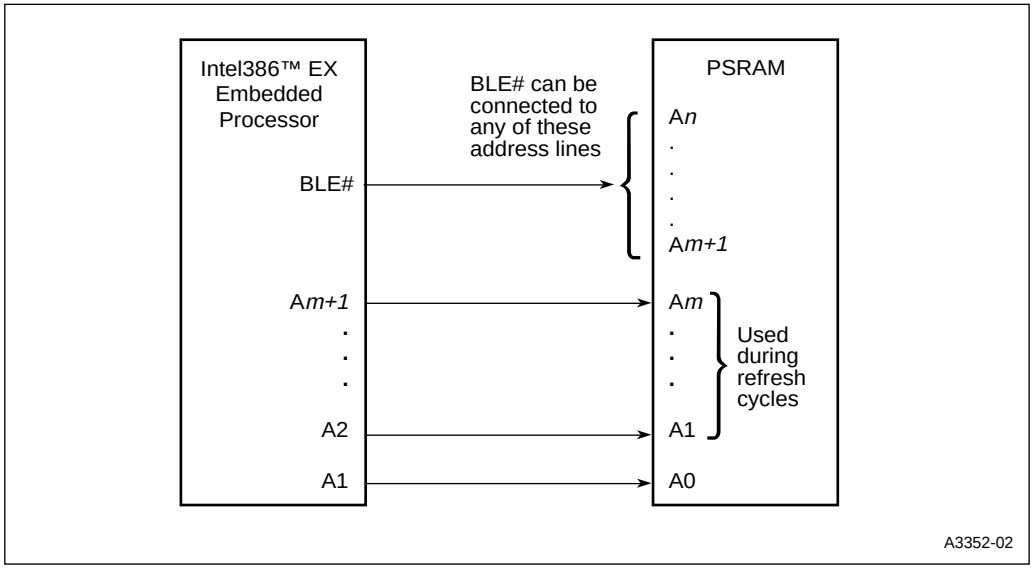


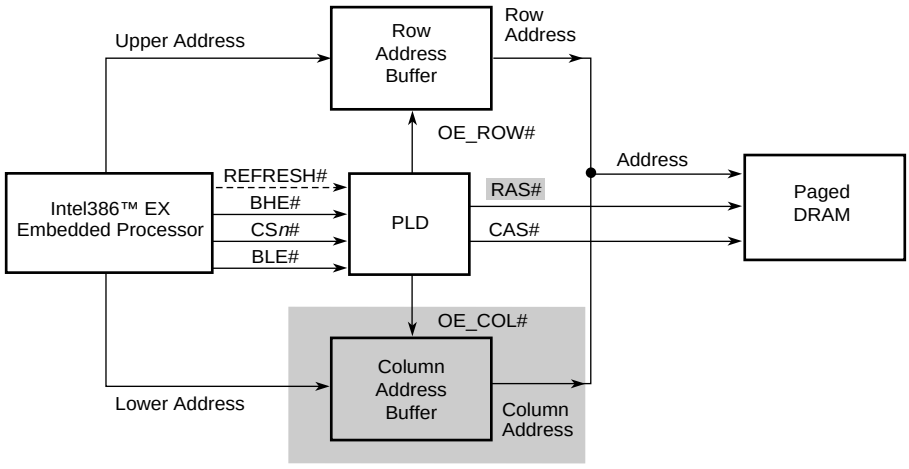
Figure 15-6. Connections to Ensure Refresh of All Rows in an 8-Bit Wide PSRAM Device

- An external device can gain bus control through either the HOLD signal or the DMA cascade mode. In this case, a refresh request causes the HLDA or DACKn# signal to be deasserted. When this happens, the external device must drop its request line (HOLD or DRQn) to allow the RCU to perform a refresh cycle. The refresh request remains pending until the RCU gets control of the bus.

- If the counter value stored in the Refresh Clock Interval Register (RFSCIR) is <8 and the RCU is enabled, the RCU always has bus control and other devices will never gain access to the bus. This is because refresh requests have the highest priority in the bus arbitration scheme and you are requesting the bus too often.
- There are two common methods of refreshing DRAM: RAS#-only and CAS#-before-RAS#.
  - — RAS#-only refresh takes advantage of the Intel386 EX Embedded Processor's built in refresh address counter (RFSADD).
  - — In a CAS#-before-RAS# refresh, the DRAM provides the row address for the refresh cycle. The RCU counter still generates the row addresses, but they are disregarded by the DRAM. The only external logic required is a PLD to recognize a refresh cycle and provide the CAS# and RAS# signals to the DRAM.

Page Mode

A paged DRAM access uses the upper address lines for the row addresses and the lower lines for the column addresses. On the Intel386 EX embedded processor, the lower address lines are connected to the Refresh Address Counter Register (RFSADD). The RFSADD increments through a set sequence at each refresh request. Because the lower address bits (wired to the Column Address Buffer) change with each refresh request, the PLD must enable this buffer when RAS# is asserted during a refresh cycle. Figure 15-7 shows the external logic needed for paged RAS#-only refresh cycles. The PLD can determine a refresh cycle by monitoring BHE# and BLE# (they are both inactive during a refresh cycle), or by an active signal on the REFRESH# pin. The buffer and lines that are active during this type of refresh have a shaded background in Figure 15-7.



Note:
A single mux can be used in place of the row and column address buffers.

Figure 15-7. RAS# Only Refresh Logic: Paged Mode

Non-page Mode

In non-paged mode, the row address buffer can be connected to the lower address lines and the column address buffer to the upper lines. Figure 15-8 illustrates the hardware configuration for non-paged DRAM accesses. The lines and buffer that are enabled in this type of refresh are highlighted in the figure. The lower address bits are connected to the Row Address Buffer and the upper address bits are connected to the Column Address Buffer. As in Page Mode, the PLD recognizes a refresh request by sampling both BHE# and BLE# (they are both inactive during a refresh cycle), or by detecting an active signal on the REFRESH# pin. The buffer and lines that are active during this type of refresh have a shaded background in Figure 15-8.

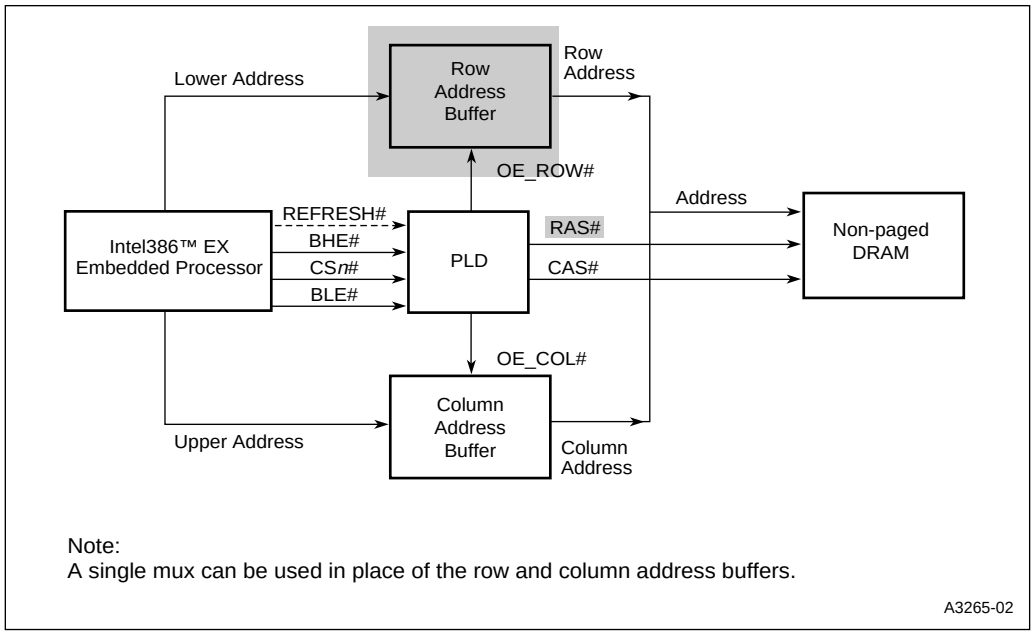


Figure 15-8. RAS# Only Refresh Logic: Non-Paged Mode

15.6 PROGRAMMING CONSIDERATIONS

REFRESH# and CS6# share a package pin. To select the REFRESH# signal at this pin, set bit 6 in the PINCFG register:

```
_SetEXRegByte(PINCFG, (_GetEXRegByte(PINCFG) | 0x40));
```

15.6.1 Refresh Control Unit Example Code

The following code example contains software routines that initialize the refresh control unit and retrieve the current value of the refresh interval timer. See Appendix C for the included header files.

```
#include <conio.h>
#include “80386ex.h”
#include “ev386ex.h”

/*****************************************************************************
   InitRCU:

   Description:
      Initializes the Refresh Control Unit
```

```
   Parameters:
      Counter_Value Value of the refresh interval

   Returns:
      Error Codes:
      E_BADVECTOR      User input an invalid parameter
      E_OK             Executed correctly

   Assumptions:
      None

   Syntax:

      #define REFRESH_INTERVAL  0x186    //Counter value for DRAM with
                                         // 1024 rows and a refresh period
                                         // of 16 msec (25 MHz Processor Clock)
      int error_code;

      error_code = InitRCU(REFRESH_INTERVAL);

   Real/Protected Mode:
      No changes required

*************************************************************************/

extern int InitRCU(WORD Counter_Value)
{

  /* Check that Counter_Value is 10 bits in length */
  if (Counter_Value != (Counter_Value & 0x03ff) )
     return(E_BADVECTOR);

  /* Clear lower 10 bits of RFSCIR */
  _SetEXRegWord(RFSCIR, 0xfc00);

  /* Set lower 10 bits of RFSCIR to Counter_Value */
  _SetEXRegWord(RFSCIR, _GetEXRegWord(RFSCIR) | Counter_Value);

  /* Enable Refresh Unit */
  _SetEXRegWord(RFSCON, _GetEXRegWord(RFSCON) | 0x8000);

  return(E_OK);

}/* InitRCU */


/*************************************************************************
   Get_RCUCounterValue:

   Description:
      This function returns the current value of the refresh interval timer.
```

```
   Parameters:
      None

   Returns:
      Refresh Interval Counter Value

   Assumptions:
      NONE

   Syntax:

      WORD CounterValue;

      CounterValue = Get_RCUCounterValue();

   Real/Protected Mode:
      No changes required
*************************************************************************/

extern WORD Get_RCUCounterValue(void)
{
  WORD Counter_Value;

  Counter_Value = _GetEXRegWord(RFSCON) & 0x3ff; // Counter value contained
                                                 // in bits RFSCON9:0

  return(Counter_Value);

}/* Get_RCUCounterValue */
```

16

INPUT/OUTPUT PORTS

CHAPTER 16
INPUT/OUTPUT PORTS

Input/Output (I/O) ports allow you to transfer information between the processor and the surrounding system circuitry. I/O ports are typically used to read system status, monitor system operation, output device status, configure system options, and generate control signals.

The Intel386™ EX processor's I/O port pins are multiplexed with peripheral pin functions. With this multiplexed arrangement, you can use just those peripheral functions required for your design and use any remaining pins for general-purpose I/O. For example, this device offers eight chip-select lines, five of which (CS0#–CS4#) are multiplexed with I/O port pins. If your design does not need all eight chip-selects, you can use up to five pins (P2.0–P2.4) for I/O.

This chapter describes the I/O ports and explains how to configure them. The information is arranged as follows:

16.1 OVERVIEW

The Intel386 EX processor has three 8-bit bidirectional I/O ports, all of which are functionally identical (Figure 16-1). Each port has three control registers and a status register.

All three ports share pins with internal peripherals (see Table 16-1). If your design does not require a pin's peripheral function, you can configure that pin for use as an I/O port. For example, if you don't need serial channel 0, you can use P1.4–P1.0 and P2.7–P2.5 as I/O ports and still allow the bus interface unit to use P1.7–P1.5 and the chip-select unit to use P2.4–P2.0.

Each pin can operate either in I/O mode or in peripheral mode. In I/O mode, a pin has three possible configurations:

- high-impedance input
- open-drain output (requires an external pull-up resistor)
- complementary output

In I/O mode, register bits control the direction (input or output) of each pin and the value of each output pin. In peripheral mode, the internal peripheral controls the operation (input or output) of the pin. Table 16-1 lists the port pins with their reset status, multiplexed peripheral functions, direction (input or output), and associated internal peripheral.

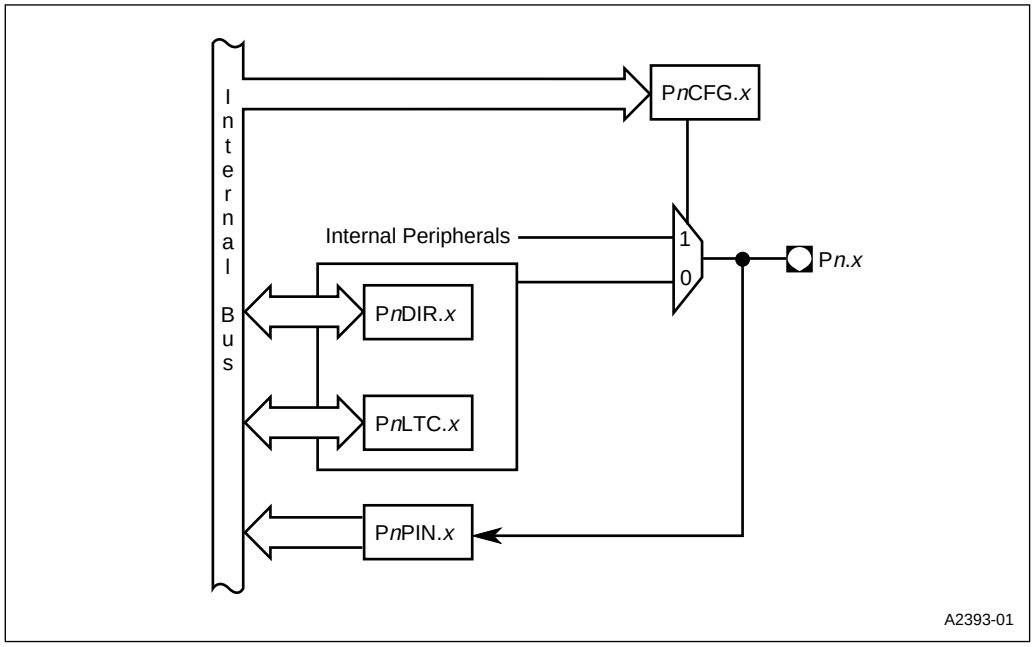


Figure 16-1. I/O Port Block Diagram

16.1.1 Port Functionality

The function of a bi-directional port pin is controlled by the state of the Port Control Latch (P*n*-LTC). This is shown in Figure 16-2.

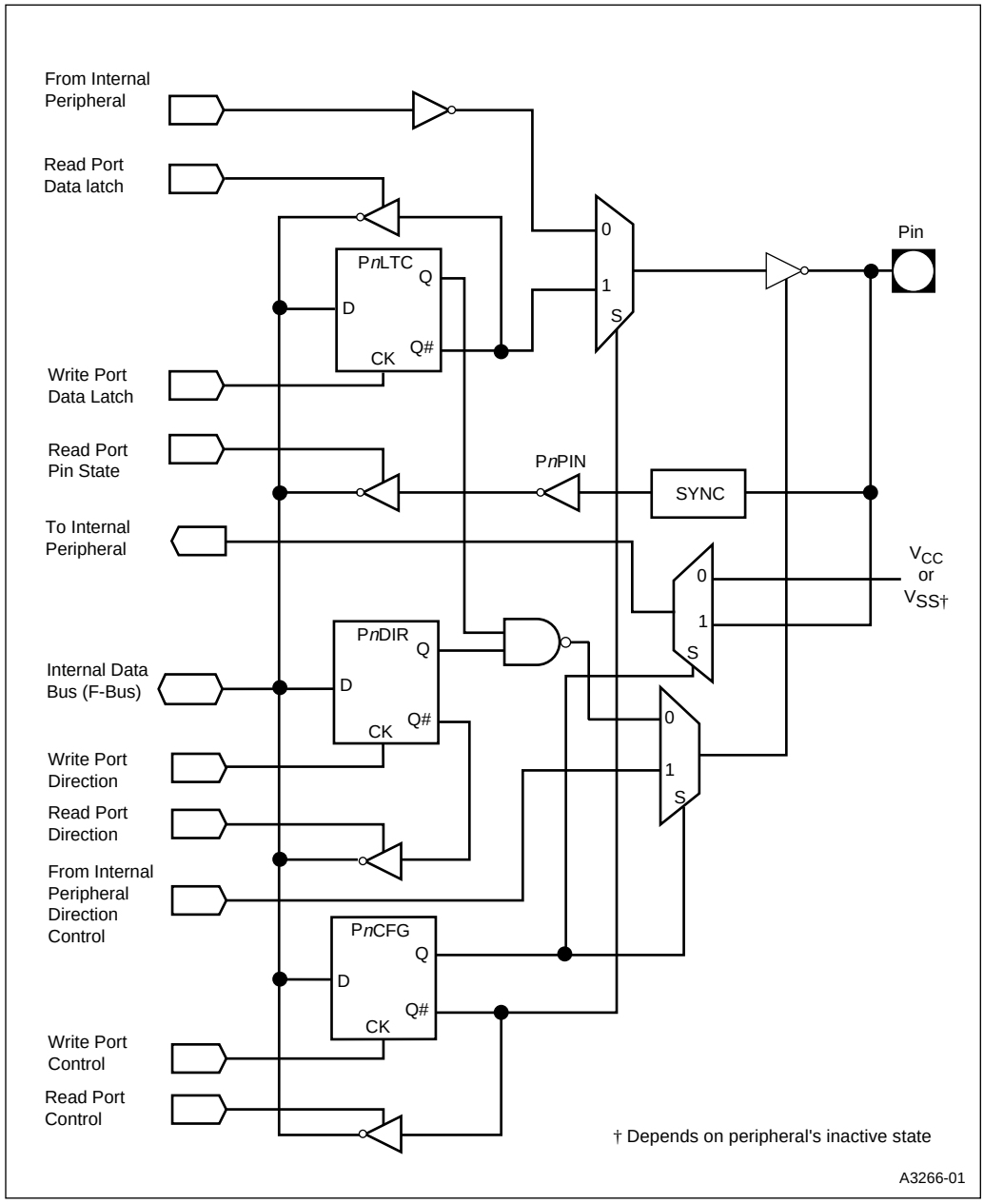


Figure 16-2. Logic Diagram of a Bi-directional Port

The output of the Pin Configuration latch (P*n*CFG) selects whether the I/O port or peripheral is connected to the pin. When the port is programmed to act as a peripheral pin, both the data for the pin and the directional control signal for the pin come from the associated integrated peripheral. When a bi-directional port pin is programmed as an I/O port, all port parameters are under software control.

The output of the Port Direction latch (P*n*DIR) enables or disables the three-state output driver when the pin is programmed as an I/O port. The three-state output driver is enabled by clearing the Port Direction latch. The data driven to an output port pin is held in the Port Data latch. Setting the Port Direction latch disables the three-state output driver making the pin an input.

The signal present on the pin is routed through a synchronizer to a three-state buffer that connects the I/O port path to the internal data bus. Not all peripheral input functions are synchronous. For example, the interrupt pins (INT9-INT0) are asynchronous so that they can wake up the chip from Powerdown mode when the clocks are stopped.

The state of the pin can be read at any time regardless of whether the pin is used as an I/O port or for a peripheral function.

Table 16-1. Pin Multiplexing

Port Pin		Peripheral Function		
Pin	**Reset Status(1)**	**Signal**	**Direction(2)**	**Internal Peripheral**
P1.0	wk 1	DCD0#	I	SIO0
P1.1	wk 1	RTS0#	O	SIO0
P1.2	wk 1	DTR0#	O	SIO0
P1.3	wk 1	DSR0#	I	SIO0
P1.4	wk 1	RI0#	I	SIO0
P1.5	wk 1	LOCK#	O	BIU
P1.6	wk 0	HOLD	I	BIU
P1.7	wk 0	HLDA	O	BIU
P2.0	wk 1	CS0#	O	CSU
P2.1	wk 1	CS1#	O	CSU
P2.2	wk 1	CS2#	O	CSU
P2.3	wk 1	CS3#	O	CSU
P2.4	wk 1	CS4#	O	CSU
P2.5	wk 0	RXD0	I	SIO0
P2.6	wk 0	TXD0	O	SIO0
P2.7	wk 1	CTS0#	I	SIO0
P3.0	wk 0	TMROUT0	O	Timer 0
P3.1	wk 0	TMROUT1	O	Timer 1
P3.2	wk 0	INT0	I	ICU
P3.3	wk 0	INT1	I	ICU
P3.4	wk 0	INT2	I	ICU
P3.5	wk 0	INT3	I	ICU
P3.6	wk 0	PWRDOWN	O	CLK & PM
P3.7	wk 0	COMCLK	I	SIO0, SIO1

NOTES:
1. wk 0 = weakly pulled down; wk 1 = weakly pulled up.
2. I = input; O = output.

16.2 REGISTER DEFINITIONS

Each port has three control registers and a status register associated with it (Table 16-2). The control registers (P*n*CFG, P*n*DIR, and P*n*LTC) can be both read and written. The status register (P*n*-PIN) can only be read. All four registers reside in I/O address space.

Table 16-2. I/O Port Registers

Register	Address	Description
P1CFG P2CFG P3CFG (read/write)	0F820H 0F822H 0F824H	Port *n* Mode Configuration: Each bit controls the mode of the associated pin. 0 = Selects I/O mode. 1 = Selects peripheral mode.
P1DIR P2DIR P3DIR (read/write)	0F864H 0F86CH 0F874H	Port *n* Direction: Each bit controls the direction of a pin that is in I/O mode. 0 = Configures a pin as a complementary output. If a pin is in peripheral mode, this value is ignored. 1 = Configures a pin as either an input or an open-drain output.
P1LTC P2LTC P3LTC (read/write)	0F862H 0F86AH 0F872H	Port *n* Data Latch: Each bit contains data to be driven onto an output pin that is in I/O mode. Write the desired pin state value to this register. If a pin is in peripheral mode, this value is ignored. Reading this register returns the value in the register—not the actual pin state.
P1PIN P2PIN P3PIN (read only)	0F860H 0F868H 0F870H	Port *n* Pin State: Each bit of this read-only register reflects the state of the associated pin. Reading this register returns the current pin state value, regardless of the pin's mode and direction.

16.2.1 Pin Configuration

You select the operating mode of each pin by writing to the associated bit in the P*n*CFG registers (Figure 16-3 gives an abbreviated version of these registers; for the complete register descriptions, see Appendix D). Setting a bit selects peripheral mode; clearing a bit selects I/O mode. Internal peripherals control pins configured for peripheral mode, while the P*n*DIR (Figure 16-4) and P*n*LTC (Figure 16-5) registers control pins configured for I/O mode. Table 16-3 shows the P*n*DIR and P*n*LTC register values that determine the pin direction and state.

NOTE

You must program both registers to correctly configure the pins.

Table 16-3. Control Register Values for I/O Port Pin Configurations

Desired Pin Configuration	Desired Pin State	P*n*DIR	P*n*LTC
High-impedance input	high impedance	1	1
Open-drain output	high impedance	1	1
	0	1	0
Complementary output	1	0	1
	0	0	0

Regardless of the pin's configuration, you can read the P*n*PIN registers (Figure 16-6) to determine the current pin state.

Port *n* Configuration
P*n*CFG (*n*=1–3)
(read/write)

Expanded Addr: **F820H, F822H, F824H**
ISA Addr: **—**
Reset State: **00H**

7							0
PM7	PM6	PM5	PM4	PM3	PM2	PM1	PM0

Bit Number	Bit Mnemonic	Function
7–0	PM7:0	Pin Mode: 0 = Places pin in I/O mode, controlled by P*n*DIR and P*n*LTC registers. 1 = Places pin in peripheral mode, controlled by the internal peripheral

Figure 16-3. Port *n* Configuration Register (P*n*CFG)

Port DIrection
P*n*DIR (*n*=1–3)
(read/write)

Expanded Addr: **F864H, F86CH, F874H**
ISA Addr: **—**
Reset State: **FFH**

7							0
PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0

Bit Number	Bit Mnemonic	Function
7–0	PD7:0	Pin Direction: 0 = Configures the pin as a complementary output. 1 = Configures the pin as an open-drain output or high-impedance input.

Figure 16-4. Port Direction Register (P*n*DIR)

Port Data Latch
P*n*LTC (*n*=1–3)
(read/write)

Expanded Addr: **F862H, F86AH, F872H**
ISA Addr: **—**
Reset State: **FFH**

7							0
PL7	PL6	PL5	PL4	PL3	PL2	PL1	PL0

Bit Number	Bit Mnemonic	Function
7–0	PL7:0	Port Data Latch: Writing a value to a PL bit causes that value to be driven onto the corresponding pin. For a complementary output, write the desired pin value to its PL bit. This value is strongly driven onto the pin. For an open-drain output, a one results in a high-impedance (input) state at the pin. For a high-impedance input, write a one to the corresponding PL bit. A one results in a high-impedance state at the pin, allowing external hardware to drive it.

Figure 16-5. Port Data Latch Register (P*n*LTC)

Port Pin State
P*n*PIN (*n*=1–3)
(read only)

Expanded Addr: **F860H, F868H, F870H**
ISA Addr: **—**
Reset State: **XXH**

7							0
PS7	PS6	PS5	PS4	PS3	PS2	PS1	PS0

Bit Number	Bit Mnemonic	Function
7–0	PS7:0	Pin State: Reading a PS bit returns the logic state present on the associated port pin.

Figure 16-6. Port Pin State Register (P*n*PIN)

16.2.2 Initialization Sequence

After a device reset, a weak pull-up or pull-down resistor holds each pin high or low until user software writes to the P*n*CFG register. The pins are configured as inputs in I/O port mode. To ensure that the pins are initialized correctly and that the weak resistors are turned off, follow this suggested initialization sequence.

NOTE

Even if you want to use the entire port as I/O (its default configuration after reset), you must write to P*n*CFG to turn off the weak pull-up and pull-down resistors.

1. Write to P*n*LTC to specify the pin value. Writing to P*n*LTC before P*n*DIR ensures that output pins initialize to known values.
   - For an output pin, write the data that is to be driven by the pin to its P*n*LTC bit.
   - For an input pin, set its P*n*LTC bit.
2. Write to P*n*DIR to specify the pin direction.
   - To configure a pin as a complementary output, clear its P*n*DIR bit.
   - To configure a pin as an input or open-drain output, set its P*n*DIR bit.
3. Write to P*n*CFG to turn off the weak resistors and select either I/O or peripheral mode.
   - To configure a pin for I/O mode, clear its P*n*CFG bit.
   - To configure a pin for peripheral mode, set its P*n*CFG bit.

16.3 DESIGN CONSIDERATIONS

This section outlines design considerations for the I/O ports.

- Source and sink current are different between the three ports. Consult the latest *Intel386™ EX Embedded Microprocessor* datasheet (order number 272420) for exact specifications.
- Use read/modify/write operations to set and clear bits.

16.3.1 Pin Status During and After Reset

A device reset applies an asynchronous reset signal to the port pins. To avoid contention with external drivers, the pins are configured as inputs in I/O port mode. To prevent pins from floating, a weak pull-up or pull-down resistor holds each pin high or low (Table 16-1). Writing to the P*n*CFG register (regardless of the value written) turns off these resistors. For example, writing any value to P1CFG after a reset turns off the weak pull-down resistors on P1.7–P1.6 and the weak pull-up resistors on P1.5–P1.0. The resistors remain off until the next reset.

16.4 PROGRAMMING CONSIDERATIONS

16.4.1 I/O Ports Code Example

The following code example contains a software routine that initializes the I/O port pins. See Appendix C for the included header files.

```
#include <conio.h>
#include “80386ex.h”
#include “ev386ex.h”

/*****************************************************************************

   Init_IOPorts:

   Description:
      This function initializes the direction and mode of the I/O port pins.
      Although the pins are default configured to the peripheral state after
      RESET, they must still be initialized to turn off the weak resistors.

   Parameters:
      Port1               Port1 Mode Configuration
      Port2            Port2 Mode Configuration
      Port3            Port3 Mode Configuration
      PortDir1         Port1 Direction
      PortDir2         Port2 Direction
      PortDir3         Port3 Direction
      PortLtc1         Port1 Data Latch Value
      PortLtc2         Port2 Data Latch Value
      PortLtc3         Port3 Data Latch Value

   Returns:
      None

   Assumptions:
      None

   Syntax:

      // Port 1 configuration defines
      #define DCD0         0x1
      #define RTS0         0x2
      #define DTR0         0x4
      #define DSR0         0x8
      #define RI0          0X10
      #define LOCK         0x20
      #define HOLD         0X40
      #define HOLDACK      0X80

      // Port 2 configuration defines
      #define CS0          0x1
      #define CS1          0x2
```

```
      #define CS2          0x4
      #define CS3          0x8
      #define CS4          0X10
      #define RXD0         0x20
      #define TXD0         0X40
      #define CTS0         0X80

      // Port 3 configuration defines
      #define TMROUT0      0x1
      #define TMROUT1      0x2
      #define INT0         0x4
      #define INT1         0x8
      #define INT2         0x10
      #define INT3         0x20
      #define PWRDWN       0x40
      #define COMCLK       0x80

      // Port Direction defines
      #define P0_IN        0x1
      #define P1_IN        0x2
      #define P2_IN        0x4
      #define P3_IN        0x8
      #define P4_IN        0x10
      #define P5_IN        0x20
      #define P6_IN        0x40
      #define P7_IN        0x80
      #define Px_OUT       0

      //Initialize SIO0 pins, DRAM and SRAM Chip Selects, Interrupt Signals,
      //and TimerOut Signals to be in peripheral mode

      Init_IOPorts( DCD0|RTS0|DTR0|DSR0|RI0,
                    CS2|CS4|RXD0|TXD0|CTS0,
                    TMROUT0|TMROUT1|INT0|INT1|INT2|INT3|PWRDWN|COMCLK,
                    Px_OUT,
                    Px_OUT,
                    Px_OUT,
                    0xff, // This example shows all output pins being
                          // initially 1
                    0xff,    // Note: Input pins must be given an initial
                    0xff );  // value of 1 whereas peripheral pins initially
                             // can be set or cleared

   Real/Protected Mode:
      No changes required.

*****************************************************************************
/

extern void Init_IOPorts(BYTE Port1, BYTE Port2, BYTE Port3, BYTE PortDir1,
                         BYTE PortDir2, BYTE PortDir3, BYTE PortLtc1,
                         BYTE PortLtc2, BYTE PortLtc3)
```

```
{
  /* Select pin values */
  _SetEXRegByte(P1LTC, PortLtc1);
  _SetEXRegByte(P2LTC, PortLtc2);
  _SetEXRegByte(P3LTC, PortLtc3);

  /* Select pin directions */
  _SetEXRegByte(P1DIR, PortDir1);
  _SetEXRegByte(P2DIR, PortDir2);
  _SetEXRegByte(P3DIR, PortDir3);

  /* Turn off weak resistors and select either I/O or peripheral mode */
  _SetEXRegByte(P1CFG, Port1);
  _SetEXRegByte(P2CFG, Port2);
  _SetEXRegByte(P3CFG, Port3);

} /* Init_IOPorts */
```

17

WATCHDOG TIMER UNIT

CHAPTER 17
WATCHDOG TIMER UNIT

The watchdog timer (WDT) unit can function as a general-purpose timer, a software watchdog timer, or a bus monitor, or it can be disabled.

This chapter is organized as follows:

17.1 OVERVIEW

The watchdog timer unit (Figure 17-1) includes a 32-bit reload register, a 32-bit down-counter, an 8-state binary counter, a readable counter value register, and a status register.

The watchdog timer can operate in three modes:

- General-purpose 32-bit timer/counter mode (default mode)
- Watchdog mode
- Bus-monitor mode

Only a single mode can be active at one time. If you have no need for any of its functions, you can disable the unit entirely.

Watchdog mode protects systems from software upsets. In watchdog mode, system software must reload the down-counter at regular intervals. If it fails to do so, the timer expires and asserts WDTOUT. For example, the watchdog times out if the software goes into an endless loop.

Some possible uses of this feature include:

- Connecting WDTOUT to the NMI pin to generate a non-maskable interrupt
- Connecting the WDTOUT signal to the RESET pin to reset the processor (and possibly the entire system)

In watchdog mode **only**, idle mode stops the down-counter. Since no software can execute while the CPU is idle, a software watchdog is unnecessary. (Chapter 8, “CLOCK AND POWER MANAGEMENT UNIT,” discusses idle mode.)

Bus monitor mode protects *normally not-ready* systems from ready-hang conditions. (A *normally not-ready* system is one in which a bus cycle continues until the accessed device asserts

READY#). In bus monitor mode, the ADS# signal from the bus interface unit (BIU) reloads the down-counter and the READY# signal stops it. The READY# signal can be generated either externally or internally, using the WDTRDY bit in the PWRCON register (Figure 17-5). If this bit is deasserted, then an external READY# is required to terminate the cycle when the WDT times out (WDTOUT is asserted) in Bus Monitor mode. In this case, if a READY# is never generated by external logic, the processor hangs (since the bus cycle never terminates). If the WDTRDY bit is set, the processor generates an internal READY# to terminate the cycle upon time-out (WDTOUT is asserted) in Bus Monitor mode.

The WDT circuitry correctly matches each READY# with a corresponding ADS# (even in pipelined mode when two ADS# pulses occur before the first READY# pulse).

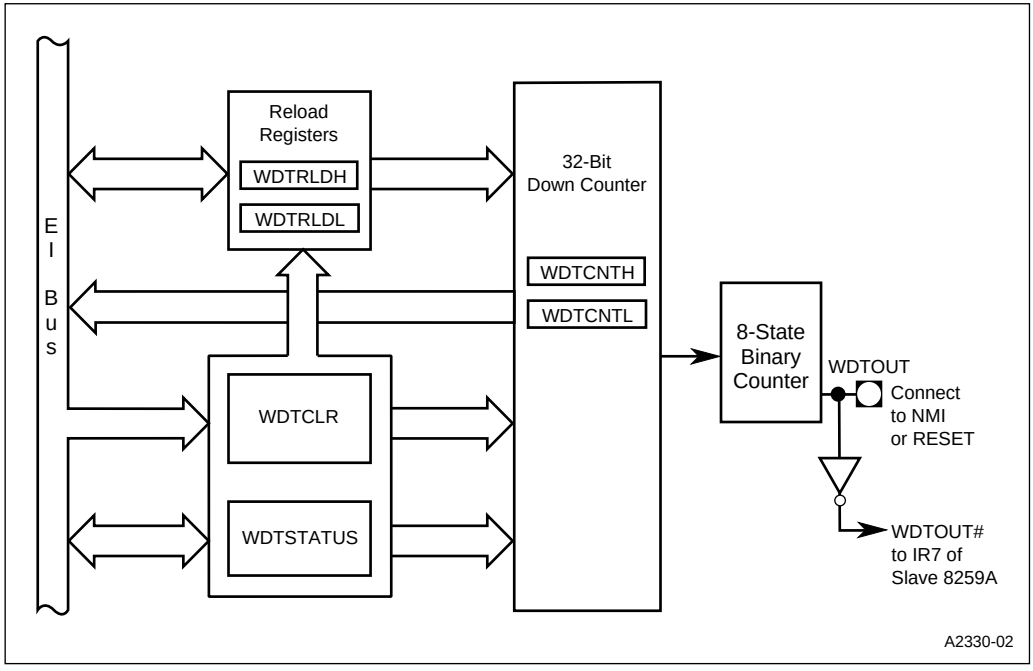


Figure 17-1. Watchdog Timer Unit Connections

17.1.1 WDT Signals

Table 17-1 describes the signals associated with the WDT.

Table 17-1. WDT Signals

Signal	Device Pin or Internal Signal	Description
ADS#	Device pin	Address Status (from the bus interface unit): Indicates that the processor is driving a valid bus-cycle definition and address onto its pins. Bus monitor mode reloads and starts the down-counter each time ADS# is asserted.
IDLE	Internal signal	Idle (from the clock and power management unit): Indicates that the device is in idle mode (core clocks stopped and peripheral clocks running). In watchdog mode, the down-counter stops when the core is idle. In bus monitor or general-purpose timer mode, the WDT continues to run while the core is idle.
READY#	Device pin	Ready (from the bus interface unit): Indicates that the current bus cycle has completed. Bus monitor mode stops the down-counter when READY# is asserted.
WDTOUT	Device pin	Watchdog Timer Output: Indicates that the down-counter has timed out. If you want a WDT timeout to reset the device, connect WDTOUT to the RESET input. If you want a WDT timeout to generate a nonmaskable interrupt, connect WDTOUT to the NMI input. An internal signal carries the inverted value of WDTOUT to the interrupt control unit (the slave's IR7 line). If you want a WDT timeout to cause a maskable interrupt, enable the interrupt. (Chapter 8, "Interrupt Control Unit," explains how to do this.)

17.2 WATCHDOG TIMER UNIT OPERATION

After a device reset, the WDT begins counting down in general-purpose timer mode. Unless you change the mode, change the reload value, or disable it, the WDT times out and asserts WDTOUT after 4 million (2^{22}) processor clock cycles (PH1 or CLKOUT cycles).

The 32-bit down-counter decrements on every processor clock cycle. When the down-counter reaches zero, the 8-state binary counter drives the WDTOUT pin high for eight processor clock cycles (16 CLK2 cycles) to signal the timeout. An internal signal carries the inverted value of the WDTOUT pin to the interrupt control unit (the slave's IR7 line). A WDT timeout can reset the system or generate an interrupt request, depending on how WDTOUT is used in your system.

The reload registers hold a user-defined value that reloads the down-counter when one of the following *reload events* occurs:

- In watchdog mode, when system software executes a specific instruction sequence (called a *lockout* sequence) to the WDTCLR location
- In bus monitor mode, when the bus interface unit asserts ADS#
- In all modes, when the down-counter reaches zero

Software can read the status register to determine the mode of the WDT, and can read the count registers to determine the current value of the down-counter.

17.2.1 Idle and Powerdown modes

In CPU-idle mode, the WDT is disabled only if it is in watchdog mode. Since no software can execute while the CPU is in idle mode, the software watchdog is unnecessary. The WDT operates normally in general-purpose timer and bus-monitor modes if the CPU is in idle mode.

In CPU-powerdown mode, the WDT unit is disabled, like all other peripherals.

17.2.2 General-purpose Timer Mode

The WDT defaults to general-purpose timer mode after reset. If your system has no requirement for a software watchdog or a bus monitor, you can use the WDT in this mode. At reset, the down-counter begins decrementing once every clock cycle, beginning at 3FFFFFH (the initial values of the reload and count registers). Unless you intervene, the WDT times out after 4 million (2^{22}) processor clock cycles.

Software can read the count registers (WDTCNTH and WDTCNTL) at any time to determine the current value of the down-counter. You might, for example, read the count when one event occurs, read it again when a second event occurs, then calculate the elapsed time between the two events.

When the down-counter reaches zero, the 8-state binary counter drives the WDTOUT pin high for eight processor clock cycles (16 CLK2 cycles). During the clock cycle immediately after the down-counter reaches zero, the down-counter is reloaded with the contents of the reload registers.

If you want fewer than 4 million (2^{22}) processor clock cycles between WDT timeouts, write a 32-bit reload value to the reload registers (Figure 17-4):

1. Write the upper 16 bits of the reload value to WDTRLDH.
2. Write the lower 16 bits of the reload value to WDTRLDL.

In the general-purpose timer mode, you cannot reload the counter except on a WDT timeout. However, you can force a reload by entering bus monitor mode, allowing an ADS# to reload the counter, then switching back to general-purpose timer mode.

17.2.3 Software Watchdog Mode

In software watchdog mode, system software must periodically reload the down-counter with a reload value or the timer expires and asserts WDTOUT. The reload value depends on the design of the system software. In general, determining the proper reload value requires software analysis and some experimentation.

After reset, the WDT defaults to general-purpose timer mode. Unless you intervene, the WDT times out after 4 million (2^{22}) processor clock cycles. If you want to use the WDT as a system watchdog, use this sequence to enable watchdog mode:

1. Write the upper 16 bits of the reload value to WDTRLDH (Figure 17-4).
2. Write the lower 16 bits of the reload value to WDTRLDL (Figure 17-4).
3. Write two sequential words, 0F01EH followed by 0FE1H, to the WDTCLR location (0F4C8H). This sequence (called a *lockout sequence*) sets the WDTEN bit in the watchdog status register and loads the contents of the reload value register into the down-counter.

Regardless of the values of the two control bits (BUSMON and CLKDIS) in the WDTSTATUS register (Figure 17-3), the lockout sequence sets the WDTEN bit and clears the remaining bits. The lockout sequence prohibits writes to the WDTSTATUS and reload registers; only a system reset can change them. This reduces the possibility for errant software to duplicate the instructions and illegally reload the timer.

The same lockout sequence that enables the watchdog reloads the down-counter. Write two sequential words, 0F01EH followed immediately by 0FE1H, to the WDTCLR location (0F4C8H).

17.2.4 Bus Monitor Mode

In bus monitor mode, ADS# reloads and starts the down-counter and READY# stops it. The initial values of the reload register and down-counter are 3FFFFFH.

CAUTION

For correct operation in Bus Monitor mode (see “Overview” on page 17-1), you must have a minimum reload value = (Maximum number of wait-states in your system + 12). For example, if the slowest device in your system requires 8 wait-states during an access, the reload value must be greater than or equal to 20.

Use this sequence to enable bus monitor mode:

1. Write the upper word of the reload value to WDTRLDH (Figure 17-4).
2. Write the lower word of the reload value to WDTRLDL (Figure 17-4).
3. Set the bus monitor bit (BUSMON) in WDTSTATUS (Figure 17-3).

Because you never execute the lockout sequence in bus monitor mode, you can change the reload value and enable or disable the mode at any time.

- To change the reload value, write the new values to the WDTRLDH and WDTRLDL registers, as described in steps 1 and 2 above.
- To disable or enable bus monitor mode, write to the bus monitor bit (BUSMON):
  - 0 = disabled
  - 1 = enabled

17.3 DISABLING THE WDT

If your system has no need for the WDT, when the unit is in bus monitor or general-purpose timer mode, you can disable the unit by setting the CLKDIS bit in the WDTSTATUS register (Figure 17-3), which stops the clock to the WDT. In this configuration, the WDT consumes minimal power, but you can re-enable the unit at any time.

If the WDT is in watchdog mode, you cannot write to the WDTSTATUS register to stop the clock and therefore cannot disable the unit.

17.4 REGISTER DEFINITIONS

This section describes the registers associated with the WDT, and explains how these registers can be used to enable and use each WDT mode.

Table 17-2 describes the registers associated with the WDT.

Table 17-2. WDT Registers

Register	Address	Description
WDTCLR	0F4C8H	Watchdog Timer Clear: Write the lockout sequence to this location. Circuitry at this address decodes the lockout sequence to enable watchdog mode, reload the counter, or both. This location is used only for watchdog mode.
WDTCNTH WDTCNTL (read only)	0F4C4H 0F4C6H	WDT Counter: These registers hold the current value of the WDT down-counter. Software can read them to determine the current count value. Any reload event reloads these registers with the contents of WDTRLDH and WDTRLDL.
WDTRLDH WDTRLDL (read/write)	0F4C0H 0F4C2H	WDT Reload Value: Write the reload value to these registers, using two word writes. After a lockout sequence is issued, these registers cannot be written again until after a device reset. A reload event (each WDT mode has its own; refer to Sections 17.2.2 through 17.2.4) reloads WDTCNTH and WDTCNTL with the contents of these registers.
WDTSTATUS (read/write)	0F4CAH	WDT Status: This register contains one **read-only** bit (WDTEN) that indicates whether watchdog mode is enabled and two read/write bits that control bus monitor mode and the WDT clock. A lockout sequence sets the WDTEN bit and clears the two read/write bits, disabling bus monitor mode and enabling the WDT clock. After a lockout sequence is issued, a write to this register has no effect unless the device is reset. Software can read this register to determine the current status of the WDT and (unless a lockout sequence has been issued) can set the BUSMON bit to enable bus monitor mode or set the CLKDIS bit to disable the WDT.
PWRCON (read/write)	0F800H	Power Control register: This register holds the WDTRDY bit that is used to enable/disable internal READY# generation for the WDT Bus Monitor mode.

WDT Counter Value (High)
WDTCNTH
(read only)

Expanded Addr: **F4C4H**
ISA Addr: **—**
Reset State: **003FH**

15							8
WC31	WC30	WC29	WC28	WC27	WC26	WC25	WC24
7							**0**
WC23	WC22	WC21	WC20	WC19	WC18	WC17	WC16

WDT Counter Value (Low)
WDTCNTL
(read only)

Expanded Addr: **F4C6H**
ISA Addr: **—**
Reset State: **FFFFH**

15							8
WC15	WC14	WC13	WC12	WC11	WC10	WC9	WC8
7							**0**
WC7	WC6	WC5	WC4	WC3	WC2	WC1	WC0

Bit Number	Bit Mnemonic	Function
High 15–0	WC31:16	WDT Counter Value High Word and Low Word:
Low 15–0	WC15:0	Read the high word of the counter value from WDTCNTH and the low word from WDTCNTL.

Figure 17-2. WDT Counter Value Registers (WDTCNTH and WDTCNTL)

WDT Status
WDTSTATUS
(read/write)

Expanded Addr: **F4CAH**
ISA Addr: **—**
Reset State: **00H**

7							0
WDTEN	—	—	—	—	—	BUSMON	CLKDIS

Bit Number	Bit Mnemonic	Function
7	WDTEN	Watchdog Mode Enabled: This **read-only** bit indicates whether watchdog mode is enabled. Only a lockout sequence can set this bit and only a device reset can clear it. 0 = Watchdog mode disabled 1 = Watchdog mode enabled
6–2	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
1	BUSMON	Bus Monitor Enable: 0 = Disables bus monitor mode 1 = Enables bus monitor mode Read this bit to determine the current status. A lockout sequence clears BUSMON and prevents writes to the WDTSTATUS register.
0	CLKDIS	Clock Disable: Write to this bit to stop or restart the clock to the WDT; read it to determine the current clock status. A lockout sequence clears CLKDIS and prevents writing to this register. 0 = Clock enabled 1 = Processor clock (frequency=CLK2/2) disabled (stopped)

Figure 17-3. WDT Status Register (WDTSTATUS)

WDT Reload Value (High)
WDTRLDH
(read/write)

Expanded Addr: **F4C0H**
ISA Addr: **—**
Reset State: **003FH**

15							8
WR31	WR30	WR29	WR28	WR27	WR26	WR25	WR24
7							**0**
WR23	WR22	WR21	WR20	WR19	WR18	WR17	WR16

WDT Reload Value (Low)
WDTRLDL
(read/write)

Expanded Addr: **F4C2H**
ISA Addr: **—**
Reset State: **FFFFH**

15							8
WR15	WR14	WR13	WR12	WR11	WR10	WR9	WR8
7							**0**
WR7	WR6	WR5	WR4	WR3	WR2	WR1	WR0

Bit Number	Bit Mnemonic	Function
High 15–0	WR31:16	WDT Reload Value (High Word and Low Word):
Low 15–0	WR15:0	Write the high word of the reload value to WDTRLDH and the low word to the WDTRLDL.

Figure 17-4. WDT Reload Value Registers (WDTRLDH and WDTRLDL)

Power Control Register
PWRCON
(read/write)

Expanded Addr: **F800H**
ISA Addr: **—**
Reset State: **00H**

7							0
—	—	—	—	WDTRDY	HSREADY	PC1	PC0

Bit Number	Bit Mnemonic	Function
7–4	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
3	WDTRDY	Watch Dog Timer Ready: 0 = An external READY must be generated to terminate the cycle when the WDT times out in Bus Monitor Mode. 1 = Internal logic generates READY# to terminate the cycle when the WDT times out in Bus Monitor Mode.
2	HSREADY	Halt/Shutdown Ready: 0 = An external ready must be generated to terminate a HALT/Shutdown cycle. 1 = Internal logic generates READY# to terminate a HALT/Shutdown cycle.
1–0	PC1:0	Power Control: Program these bits, then execute a HALT instruction. The device enters the programmed mode when READY# (internal or external) terminates the halt bus cycle. When these bits have equal values, the HALT instruction causes a normal halt and the device remains in active mode. **PC1 PC0** 0 0 active mode 1 0 idle mode 0 1 powerdown mode 1 1 active mode

Figure 17-5. Power Control Register (PWRCON)

17.5 DESIGN CONSIDERATIONS

This section outlines design considerations for the watchdog timer unit.

Depending on the system configuration, a WDT timeout can cause a maskable interrupt, a non-maskable interrupt, or a system reset.

Maskable interrupt The WDT timeout signal is internally inverted and connected to the interrupt control unit's slave IR7 line. If you want a WDT timeout to generate a slave IR7 interrupt (maskable interrupt), you need only enable (unmask) the interrupt (Refer to Chapter 9, for details). Ensure that the slave 8259A is configured for edge-triggered interrupts (refer to Chapter 9, Interrupt Control Unit) if IR7 is unmasked. Otherwise, the WDT generates continuous interrupts.

Nonmaskable interrupt If you want a WDT timeout to cause a nonmaskable interrupt, connect the WDTOUT pin to the NMI input pin.

Reset If you want a WDT timeout to reset the system, connect the WDTOUT pin to the RESET input pin.

17.6 PROGRAMMING CONSIDERATIONS

This section outlines programming considerations for the watchdog timer unit.

17.6.1 Writing to the WDT Reload Registers (WDTRLDH and WDTRLDL)

WDTRLDH and WDTRLDL are 16 bit registers at addresses 0F4C0H and 0F4C2H respectively. Therefore, when using a 32-bit write to load the two registers, the lower 16 bits should contain the data for WDTRLDH and the higher 16 bits should contain the data for WDTRLDL.

For example, 4321H can be written to WDTRLDH and 0CCCCH to WDTRLDL using a 32-bit write of the number 0CCCC4321H to I/O address 0F4C0H.

17.6.2 Minimum Counter Reload Value

To ensure correct operation of the Watchdog Timer, the WDT's counter should never be reloaded with a value less than 8.

17.6.3 Watchdog Timer Unit Code Examples

This section includes these software routines:

ReLoadDownCounter Initiates a lockout sequence

GetWDT_Count Reads the value of the counter

WDT_BusMonitor Places the WDT in Bus Monitor Mode

EnableWDTInterrupt Enables WDT interrupts

See Appendix C for included header files.

```
#include <dos.h>
#include <conio.h>
#include “80386ex.h”
#include “ev386ex.h”

/*****************************************************************************
   ReLoadDownCounter:

   Description:
      This function initiates a lockout sequence which results in the
      setting of the WDTEN bit in the status register. By setting
      WDTEN, the software watchdog mode is enabled.

   Parameters:
      None

   Returns:
      None

   Assumptions:
      None

   Syntax:

      ReloadDownCounter();

   Real/Protected Mode:
      No changes required

*****************************************************************************/

void ReLoadDownCounter(void)
{
   _disable(); /* Disable interrupts */

   _SetEXRegWord(WDTCLR,0xf01e);
   _SetEXRegWord(WDTCLR,0xfe1);

   _enable(); /* Enable interrupts */

}/* ReLoadDownCounter */

/*****************************************************************************

   GetWDT_Count:

   Description:
      Returns current value of watch dog counter.
```

```
   Parameters:
      None

   Returns:
      16-bit down-counter value

   Assumptions:
      None

   Syntax:

      WORD counter_value;

      counter_value = GetWDT_Count();

   Real/Protected Mode:
      No changes required.

*****************************************************************************/

DWORD GetWDT_Count(void)
{
        WORD LowWord, HiWord;
        LowWord = _GetEXRegWord(WDTCNTL);
        HiWord  = _GetEXRegWord(WDTCNTH);
        return (((DWORD)HiWord << 16) + LowWord);

}/* GetWDT_Count */


/*****************************************************************************

   WDT_BusMonitor:

   Description:
      Enables the bus monitor mode of the Watch Dog Timer.

   Parameters:
      EnableDisable    Nonzero if bus monitor mode is to be enabled,
                       Zero if it is to be disabled

   Returns:
      None

   Assumptions:
      None

   Syntax:

      #define Enable 0x01
      #define Disable 0x00
```

```
      WDT_BusMonitor(Enable);

   Real/Protected Mode:
      No changes required.

*****************************************************************************/

void WDT_BusMonitor(BYTE EnableDisable)
{
         BYTE Status;

         Status = _GetEXRegByte(WDTSTATUS);

         if(EnableDisable)  /* If true, Enable */
           _SetEXRegByte(WDTSTATUS, Status | BIT1MSK);     /* Set Bit */

        else  /* else,   Disable */
           _SetEXRegByte(WDTSTATUS, Status & ~BIT1MSK);  /* Clear Bit */

}/* WDT_BusMonitor */


/*****************************************************************************
   EnableWDTInterrupt:

   Description:
      Enables a maskable interrupt on the assertion of WDTOUT

   Parameters:
      None

   Returns:
      None

   Assumptions:
      None

   Syntax:

      EnableWDTInterrupt();

   Real/Protected Mode:
      No changes required

*****************************************************************************/

extern void EnableWDTInterrupt(void)
{
   InitICUSlave(ICU_TRIGGER_EDGE, 0x30, 0);  /* Initialize Slave ICU */

   SetIRQVector(wdtISR, 15, INTERRUPT_ISR);  /* Puts address of interrupt
service
```

```
                                        routine in Interrupt Vector Table */

   Enable8259Interrupt(IR2,IR7);   /* Enable slave interrupt to master(IR2),
                                Enable slave IR2 */

   _enable();  /* Enable Interrupts */

} /* EnableWDTInterrupt */


/*****************************************************************************
   wdtISR:

   Description:
      Interrupt Service Routine for Watchdog Timer

   Parameters:
      None

   Returns:
      None

   Assumptions:
      None

   Syntax:
      Not called by user; Interrupt Control Unit executes this
      routine upon acknowledgment of a WDT interrupt

   Real/Protected Mode:
      No changes required

*****************************************************************************/

void interrupt far wdtISR(void)
{
   SerialWriteStr(SIO_PORT,"Executing in WDT_ISR");  /* Prints out to Serial
                                                        Port as a demonstration */

   NonSpecificEOI();

} /* wdtISR */
```

18

JTAG TEST-LOGIC UNIT

CHAPTER 18
JTAG TEST-LOGIC UNIT

The JTAG test-logic unit enables you to test both the device logic and the interconnections between the device and the board (system) it is plugged into. The term *JTAG* refers to the Joint Test Action Group, the IEEE technical subcommittee that developed the testability standard published as Standard 1149.1-1990, *IEEE Standard Test Access Port and Boundary-Scan Architecture*† *and its supplement, Standard 1149.1a-1993*. The Intel386™ EX Embedded Processor JTAG test-logic unit is fully compliant with this standard.

You can use the JTAG unit for other purposes. For example you can perform in-system programming of flash memory; refer to AP-720, *Programming Flash Memory through the Intel386™ EX Embedded Processor JTAG Port* (order number 272753).

This chapter is organized as follows:

18.1 OVERVIEW

As the title of the IEEE standard suggests, two major components of the test-logic unit are the te*st access port* and the *boundary-scan* register. The term *test access port* (TAP) refers to the dedicated input and output pins through which a tester communicates with the test-logic unit. The term *boundary-scan* refers to the ability to *scan* (observe) the signals at the *boundary* (the pins) of a device. A boundary-scan cell resides at each pin. These cells are connected serially to form the boundary-scan register, which allows you to control or observe every device pin except the clock pin, the power and ground pins, and the test access port pins.

The test-logic unit allows a tester to perform these tasks:

- Identify a component on a board (manufacturer, part number, and version)
- Bypass one or more components on a board while testing others
- Preload a pin state for a test or read the current pin state
- Perform static (slow-speed) testing of this device
- Test off-chip circuitry and board-level interconnections

† Some of the figures and tables in this chapter were reproduced from Standard 1149.1-1990, *IEEE Standard Test Access Port and Boundary-Scan Architecture*, Copyright 1993 by the Institute of Electrical and Electronics Engineers, Inc., with the permission of the IEEE.

- Place all device output pins into their inactive drive (high-impedance) state, allowing external hardware to drive connections that the processor normally drives

The test-logic unit (Figure 18-1) is fully compliant with IEEE Standard 1149.1. It consists of the test access port (TAP), the test access port controller, the instruction register (IR), and three data registers (IDCODE, BYPASS, and BOUND). It also includes logic for generating necessary clock and control signals.

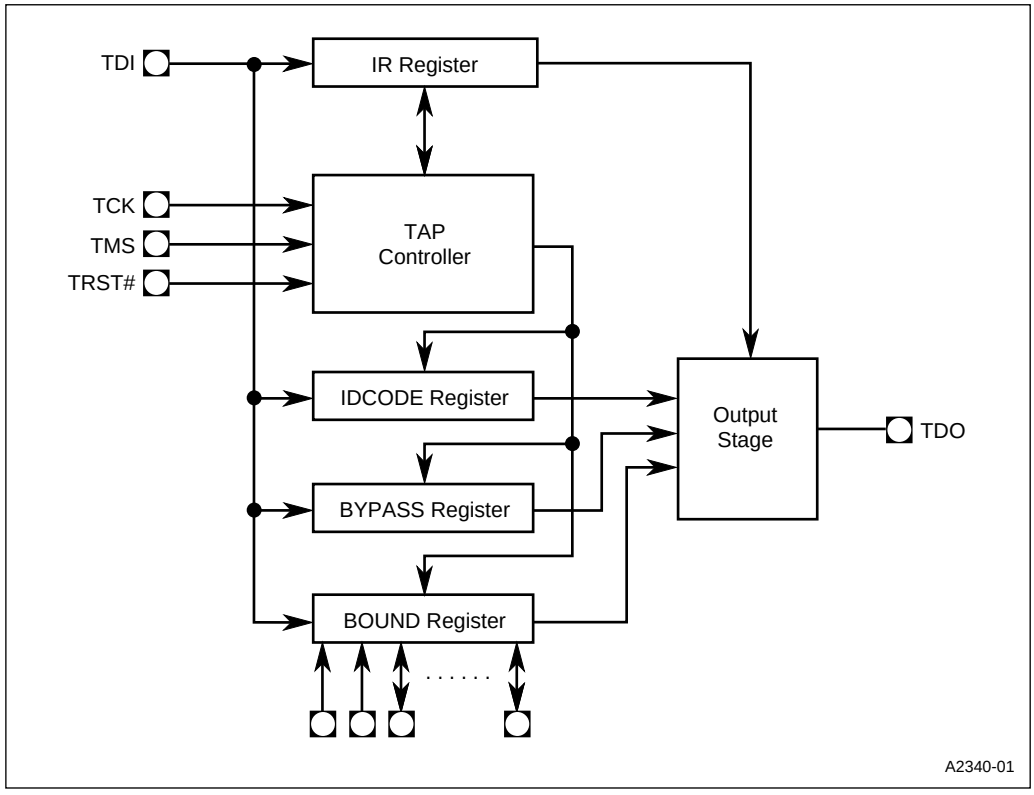


Figure 18-1. Test Logic Unit Connections

18.2 TEST-LOGIC UNIT OPERATION

18.2.1 Test Access Port (TAP)

The test access port consists of five dedicated pins (four inputs and one output). It is through these pins that all communication with the test-logic unit takes place. This unit has its own clock (TCK) and reset (TRST#) pins, so it is independent of the rest of the device. The test-logic unit can read or write its registers even if the rest of the device is in reset or powerdown.

CAUTION

The JTAG Test-Logic Unit must be reset upon power-up using the TRST# pin. (To do this, invert the RESET signal and send this inverted RESET to the TRST# pin). If this is not done, the processor may power-up with the JTAG test-logic unit in control of the device pins, and the system does not initialize properly.

The test-logic unit allows you to shift test instructions and test data into the device and to read the results of the test. A tester (that is, an external bus master such as automatic test equipment or a component that interfaces to a higher-level test bus) controls the TAP controller's operation by applying signals to the clock (TCK) and test-mode-select (TMS) inputs. Instructions and data are shifted serially from the test-data input (TDI) to the test-data output (TDO). Table 18-1 describes the test access port pins.

Table 18-1. Test Access Port Dedicated Pins

Pin Name	Description
TCK	Test Clock Input: Provides the clock input for the test-logic unit. An external signal must provide a maximum input frequency of one-half the CLK2 input frequency. TCK is driven by the test-logic unit's control circuitry.
TDI	Test Data Input: Serial input for test instructions and data. Sampled on the rising edge of TCK; valid only when either the instruction register or a data register is being serially loaded (SHIFT-IR, SHIFT-DR).
TDO	Test Data Output: Serial output for test instructions and data. TDO shifts out the contents of the instruction register or the selected data register (LSB first) on the falling edge of TCK. If serial shifting is not taking place, TDO floats.
TMS	Test Mode Select Input: Controls the sequence of the TAP controller's states. Sampled on the rising edge of TCK.
TRST#	Test Reset Input: Resets the TAP controller. Asynchronously clears the data registers and initializes the instruction register to 0010 (the IDCODE instruction opcode).

NOTE: The JTAG Test-Logic Unit must be reset upon power-up using the TRST# pin. If this is not done the processor may power-up with the JTAG test-logic unit in control of the device pins, and the system does not initialize properly.

18.2.2 Test Access Port (TAP) Controller

The TAP controller is a finite-state machine that is capable of 16 states (Figure 18-2). Three of its states provide the basic actions required for testing:

- Applying stimulus (update-data-register)
- Executing a test (run-test/idle)
- Capturing the response (capture-data-register)

Its remaining states support loading instructions, shifting information toward TDO, scanning pins, and pausing to allow time for the tester to perform other operations.

The TAP controller changes state only in response to the assertion of the test-reset input (TRST#) or the state of the mode-select pin (TMS) on the rising edge of TCK. TRST# causes the TAP controller to enter its test-logic-reset state, and the state of TMS on the rising edge of TCK controls the subsequent states.Table 18-2 describes the states and Figure 18-2 illustrates how the TAP state machine moves from one state to another.

Table 18-2. TAP Controller State Descriptions (Sheet 1 of 2)

State	Description	Next State (on TCK Rising Edge)	
		TMS = 0	TMS = 1
Test-Logic-Reset	Resets the test-logic unit and forces the IDCODE instruction into the instruction register. (In components that have no IDCODE instruction, the BYPASS instruction is loaded instead.) Test logic is disabled; the device is in normal operating mode.	Run-Test/Idle	Test-Logic-Reset
Run-Test/Idle	Executes a test or disables the test logic.	Run-Test/Idle	Select-DR-Scan
Select-DR-Scan	Selects the data register to be placed in the serial path between TDI and TDO.	Capture-DR	Select-IR-Scan
Capture-DR	Parallel loads data into the active data register, if necessary. Otherwise, the active register retains its previous state.	Shift-DR	Exit1-DR
Shift-DR	The active register shifts data one stage toward TDO on each TCK rising edge.	Shift-DR	Exit1-DR
Exit1-DR	The active register retains its previous state.	Pause-DR	Update-DR
Pause-DR	The active register temporarily stops shifting data and retains its previous state.	Pause-DR	Exit2-DR
Exit2-DR	The active register retains its previous state.	Shift-DR	Update-DR
Update-DR	Applies stimulus to the device. Data is latched onto the active register's parallel output on the falling edge of TCK. If the register has no parallel output, it retains its previous state.	Run-Test/Idle	Select-DR-Scan

NOTE: By convention, the abbreviation *DR* stands for *data register*, and *IR* stands for *instruction register*. The *active register* is the register that the current instruction has placed in the serial path between TDI and TDO.

Table 18-2. TAP Controller State Descriptions (Sheet 2 of 2)

State	Description	Next State (on TCK Rising Edge)	
		TMS = 0	TMS = 1
Select-IR-Scan	Test-logic is idle and the instruction register retains its previous state.	Capture-IR	Test-Logic-Reset
Capture-IR	Loads the SAMPLE/PRELOAD instruction instruction (0001) into the instruction register.	Shift-IR	Exit1-IR
Shift-IR	Shifts the SAMPLE/PRELOAD instruction one stage toward TDO while shifting the new instruction in from TDI on each rising edge of TCK.	Shift-IR	Exit1-IR
Exit1-IR	The instruction register retains its previous state.	Pause-IR	Update-IR
Pause-IR	The instruction register temporarily stops shifting and retains its previous state.	Pause-IR	Exit2-IR
Exit2-IR	The instruction register retains its previous state.	Shift-IR	Update-IR
Update-IR	Latches the current instruction onto the instruction register's parallel output on the falling edge of TCK.	Run-Test/Idle	Select-DR-Scan

NOTE: By convention, the abbreviation *DR* stands for *data register*, and *IR* stands for *instruction register*. The *active register* is the register that the current instruction has placed in the serial path between TDI and TDO.

For example, assume that the TAP controller is in its test-logic-reset state and you want it to start shifting the contents of the instruction register from TDI toward TDO (Shift-IR state). This state change requires a zero, two ones, then two zeros on TMS at the next five rising edges of TCK (see Table 18-3). By supplying the proper values in the correct sequence, you can move the TAP controller from any state to any other state.

Table 18-3. Example TAP Controller State Selections

Initial State	TMS Value at TCK Rising Edge	Resulting State
Test-Logic-Reset	0	Run-Test/Idle
Run-Test/Idle	1	Select-DR-Scan
Select-DR-Scan	1	Select-IR-Scan
Select-IR-Scan	0	Capture-IR
Capture-IR	0	Shift-IR

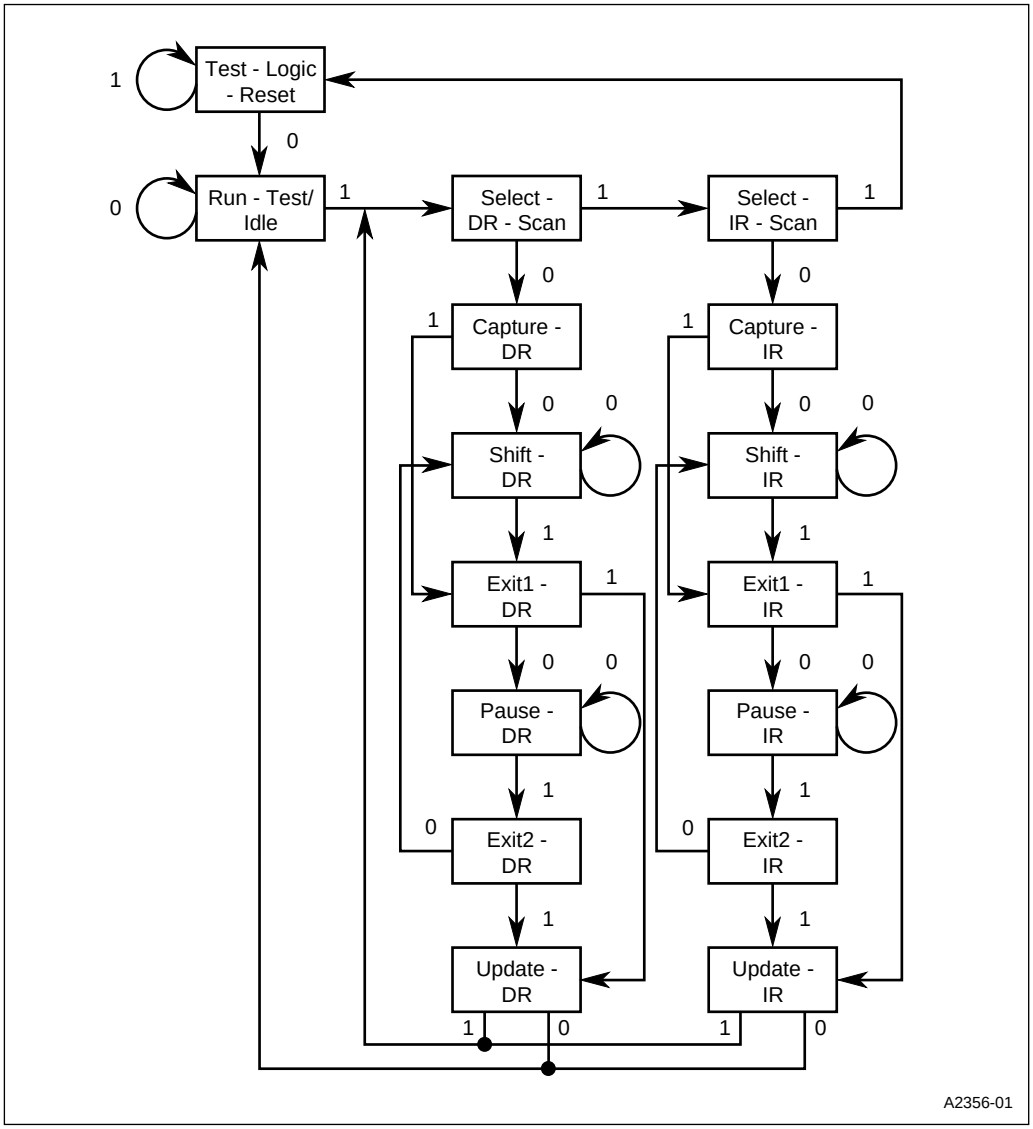


Figure 18-2. TAP Controller (Finite-State Machine)

18.2.3 Instruction Register (IR)

An instruction opcode is clocked serially through the TDI pin into the four-bit instruction register (Figure 18-3). The instruction determines which data register is affected. Table 18-4 lists the instructions with their binary opcodes, descriptions, and associated registers.

Instruction Register
IR

Reset State
(Using TRST#): 02H

3			**0**
INST3	INST2	INST1	INST0

Bit Number	Bit Mnemonic	Function
3–0	INST3:0	Instruction opcode. At reset (using TRST#, or after 5 TCK cycles with TMS held low), this field is loaded with 0010, the opcode for the IDCODE instruction. Instructions are shifted into this field serially through the TDI pin. (Table 18-4 lists the valid instruction opcodes.)

Figure 18-3. Instruction Register (IR)

Table 18-4. Test-logic Unit Instructions

Mnemonic	Opcode†	Description	Affected Register
BYPASS	1111	Bypass on-chip system logic (mandatory instruction). Used for those components that are not being tested.	BYPASS
EXTEST	0000	Off-chip circuitry test (mandatory instruction). Used for testing device interconnections on a board.	BOUND
SAMPRE	0001	Sample pins/preload data (mandatory instruction). Used for controlling (preload) or observing (sample) the signals at device pins. This test has no effect on system operation.	BOUND
IDCODE	0010	ID code test (optional instruction). Used to identify devices on a board.	IDCODE
HIGHZ	1000	High-impedance/On-Circuit Emulation (ONCE) mode test (optional instruction). Used to place device pins into their inactive drive states. Allows external components to drive signals onto connections that the processor normally drives.	BYPASS

† The opcode is the sequence of data bits shifted serially into the instruction register (IR) from the TDI input. The opcodes for EXTEST and BYPASS are mandated by IEEE 1149.1, so they should be the same for all JTAG-compliant devices. The remaining opcodes are designer-defined, so they may vary among devices.

All unlisted opcodes are reserved. Use of reserved opcodes could cause the device to enter reserved factory-test modes.

18.2.4 Data Registers

The test-logic unit uses three data registers: bypass, identification code, and boundary-scan. The instruction determines which data register is used.

The single-bit bypass register (BYPASS) provides a minimal-length serial path between TDI and TDO. During board-level testing, you can use this path for any devices that are **not** currently under test. This speeds access to the data registers for the devices that **are** being tested.

The 32-bit identification code register (IDCODE) identifies a device by manufacturer, part number, and version number. Figure 18-4 describes the register and shows the values for the Intel386 EX processor.

Identification Code Register
IDCODE

Reset State: 2027 0013H (3V)
2827 0013H (5V)

31							24
0	0	1	0	0 (3V) 1 (5V)	0	0	0
23							**16**
0	0	1	0	0	1	1	1
15							**8**
0	0	0	0	0	0	0	0
7							**0**
0	0	0	1	0	0	1	1

Bit Number	Bit Mnemonic	Function
31–28	V3:0	Device version number.
27–12	PN15:0	Device part number.
11–1	MFR10:0	Manufacturer identification (compressed JEDEC106-A code).
0	IDP	Identification Present. Always true for this device. This is the first data bit shifted out of the device during a data scan immediately following an exit from the test-logic-reset state. A one indicates that an IDCODE register is present. (A zero originates from the BYPASS register and indicates that the device being interrogated has **no** IDCODE register.)

Figure 18-4. Identification Code Register (IDCODE)

The boundary-scan register (BOUND) holds data to be applied to the pins or data observed at the pins. Each bit corresponds to a specific pin (Table 18-5).

Table 18-5. Boundary-scan Register Bit Assignments

Bit	Pin	Bit	Pin	Bit	Pin	Bit	Pin
0	M/IO#	25	A15	50	TMROUT2	75	P2.2
1	D/C#	26	A16/CAS0	51	TMRGATE2	76	P2.3
2	W/R#	27	A17/CAS1	52	INT4/TMRCLK0	77	P2.4
3	READY#	28	A18/CAS2	53	INT5/TMRGATE0	78	DACK0#
4	BS8#	29	A19	54	INT6/TMRCLK1	79	P2.5/RXD0
5	RD#	30	A20	55	INT7/TMRGATE1	80	P2.6/TXD0
6	WR#	31	A21	56	STXCLK	81	P2.7
7	BLE#	32	A22	57	FLT#	82	UCS#
8	BHE#	33	A23	58	P1.0	83	CS6#/REFRESH#
9	ADS#	34	A24	59	P1.1	84	LBA#
10	NA#	35	A25	60	P1.2	85	D0
11	A1	36	SMI#	61	P1.3	86	D1
12	A2	37	P3.0/TMROUT0/ INT9	62	P1.4	87	D2
13	A3	38	P3.1/TMROUT1/ INT8	63	P1.5	88	D3
14	A4	39	SRXCLK	64	P1.6/HOLD	89	D4
15	A5	40	SSIORX	65	RESET	90	D5
16	A6	41	SSIOTX	66	P1.7/HLDA	91	D6
17	A7	42	P3.2/INT0	67	DACK1#/TXD1	92	D7
18	A8	43	P3.3/INT1	68	EOP#	93	D8
19	A9	44	P3.4/INT2	69	WDTOUT	94	D9
20	A10	45	P3.5/INT3	70	DRQ0	95	D10
21	A11	46	P3.6/PWRDOWN	71	DRQ1/RXD1	96	D11
22	A12	47	P3.7/SERCLK	72	SMIACT#	97	D12
23	A13	48	PEREQ/TMRCLK2	73	P2.0	98	D13
24	A14	49	NMI	74	P2.1	99	D14
						100	D15

NOTES:
1. Bit 0 is closest to TDI; bit 100 is closest to TDO.
2. The boundary-scan chain consists of 101 bits; however, each bit has both a control cell and a data cell, so an EXTEST instruction requires 202 shifts (101 bits × 2 cells).

18.3 TESTING

This section explains how to use the test-logic unit to test the device and the board interconnections. For any test, you must load an instruction and perform an instruction-scan cycle, then supply the correct sequence of ones and zeros to move the TAP controller through the required states to perform the test.

18.3.1 Identifying the Device

The IDCODE instruction allows you to determine the contents of a device's IDCODE register. When TRST# is asserted, the test-logic-reset state forces the IDCODE instruction into the instruction register's parallel output latches. You can also load this instruction like any other, by manipulating the TDI input to supply the binary opcode (0010). The Capture-DR state loads the identification code into the IDCODE register, and the Shift-DR state shifts the value out.

18.3.2 Bypassing Devices on a Board

The BYPASS instruction allows you to bypass one or more devices on a board while testing others. This significantly reduces the time required for a test. For example, assume that a board has 100 devices, each of which has 101 bits in its boundary-scan register. If the boundary-scan cells are all connected in series, the boundary-scan path is 10,100 stages long. Bypassing devices allows you to shorten the path considerably. If you set 99 of the devices to shift through their bypass registers and only a single chip to shift through its boundary-scan register (101 bits in this case), the serial path is only 200 stages long.

You load the BYPASS instruction by manipulating TDI to supply the binary opcode (1111). The Capture-DR state loads a logic 0 into the bypass register and the Shift-DR state shifts the value out.

18.3.3 Sampling Device Operation and Preloading Data

The SAMPLE/PRELOAD instruction has two functions: SAMPLE takes a snapshot of data flowing from (or to) the system pins to (or from) on-chip system logic, while PRELOAD places an initial data pattern at the latched parallel outputs of the boundary-scan register cells in preparation for another boundary-scan test operation.

You load the SAMPLE/PRELOAD instruction by manipulating TDI to supply the binary opcode (0001). The Shift-DR state places the boundary-scan register in the serial path between TDI and TDO, the Capture-DR state loads the pin states into the boundary-scan register, and the Update-DR state loads the shift-register contents into the boundary-scan register's parallel outputs.

18.3.4 Testing the Interconnections (EXTEST)

The EXTEST instruction allows testing of off-chip circuitry and board-level interconnections. Boundary-scan cells at the system outputs are used to apply test stimuli, while cells at system inputs capture the results. The Capture-DR state captures input pins into the chain; the Update-DR state drives the new values of the parallel output onto the output pins.

Typically, you would use the SAMPLE/PRELOAD instruction to load data onto the boundary-scan register's latched parallel outputs before loading the EXTEST instruction. You load the EXTEST instruction by manipulating TDI to supply the binary opcode (0000). The Update-DR state drives the preloaded data onto the pins for the first test. Stimuli for the remaining tests are shifted in while the results for the completed tests are shifted out.

18.3.5 Disabling the Output Drivers

The HIGHZ instruction places all system logic outputs into an inactive drive (high impedance) state. This state allows an in-circuit emulator to drive signals onto connections that processor outputs normally drive, without risk of damaging the processor. It also allows you to connect a data source (such as a test chip) to board-level signals (such as an array of memory devices) that the processor outputs normally drive. During normal operation, the processor outputs would be active, while the test chip outputs would be inactive. During testing, you would use the HIGHZ instruction to place the processor outputs into an inactive drive state, then enable the test chip to drive the connections.

You load the HIGHZ instruction by manipulating the TDI input to supply the binary opcode (1000). The Capture-DR state loads a logic 0 into the bypass register, and the Shift-DR state shifts the value out.

18.4 TIMING INFORMATION

The test-logic unit's input/output timing is as specified in IEEE 1149.1. Figure 18-5 shows the pin timing associated with loading the instruction register and Figure 18-6 shows the timing for loading a given data register.

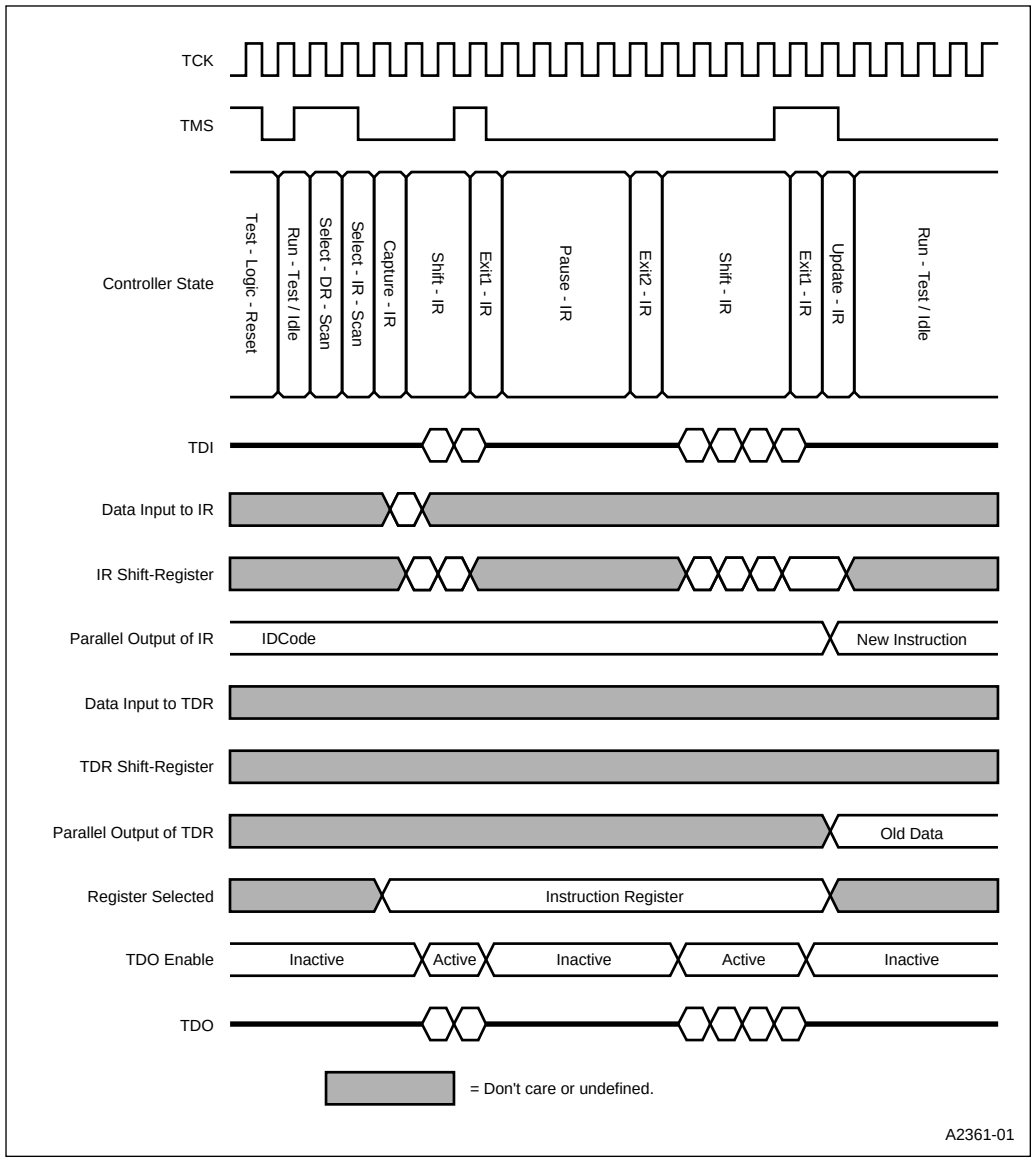


Figure 18-5. Internal and External Timing for Loading the Instruction Register

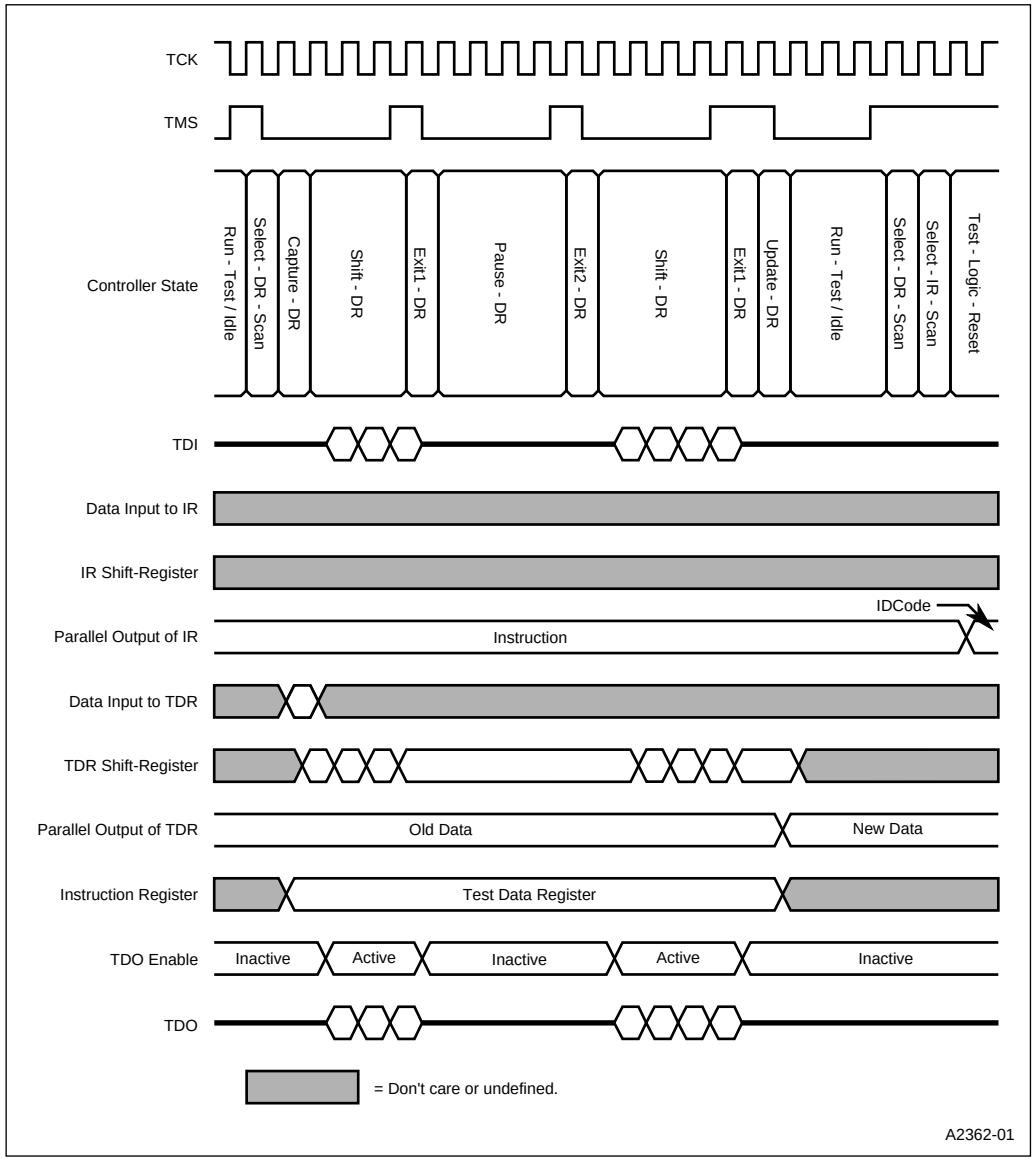


Figure 18-6. Internal and External Timing for Loading a Data Register

18.5 DESIGN CONSIDERATIONS

This section outlines considerations for the test-logic unit.

- The JTAG Test-Logic Unit must be reset upon power-up using the TRST# pin. (To do this, invert the RESET signal and send this inverted RESET to the TRST# pin). If this is not done, the processor may power-up with the JTAG test-logic unit in control of the device pins, and the system does not initialize properly.
- For system-level in-circuit emulation, use the HIGHZ instruction to enter ONCE mode. For device-level in-circuit emulation, you assert the FLT# pin to enter ONCE mode. This method can interfere with the test-logic unit's parallel functions, although it does not affect the shifting functions or the TDO output.

A

SIGNAL DESCRIPTIONS

APPENDIX A
SIGNAL DESCRIPTIONS

This appendix provides reference information for the pins and signals of the device, including the states of certain pins during reset, idle, powerdown, and hold. The information is presented in four tables:

- Table A-1 defines the abbreviations used in Table A-2 to describe the signals.
- Table A-2 describes each signal.
- Table A-3 defines the abbreviations used in Table A-4 to describe the pin states.
- Table A-4 lists the states of output and bidirectional pins after reset and during idle mode, powerdown, and hold.

Table A-1. Signal Description Abbreviations

Abbreviation	Definition
#	signal is active low
—	not applicable or none
I	standard TTL input
O	standard CMOS output
OD	open-drain output
I/O	bidirectional (input and output)
ST	Schmitt-trigger input
P	power pin
G	ground pin

Table A-2 is an alphabetical list of the signals available at the device pins. The *Multiplexed With* column lists other signals that share a pin with the signal listed in the *Signal* column.

Table A-2. Description of Signals Available at the Device Pins (Sheet 1 of 6)

Signal	Type	Name and Description	Multiplexed With (Alternate Function)
A25:19 A18:16 A15:1	O	Address Bus: Outputs physical memory or port I/O addresses. These signals are valid when ADS# is active and remain valid until the next T1, T2P, or Ti.	— CAS2:0 —
ADS#	O	Address Status: Indicates that the processor is driving a valid bus-cycle definition and address (W/R#, D/C#, M/IO#, A25:1, BHE#, BLE#) onto its pins.	—
BHE#	O	Byte High Enable: Indicates that the processor is transferring a high data byte.	—
BLE#	O	Byte Low Enable: Indicates that the processor is transferring a low data byte.	—
BS8#	I	Bus Size: Indicates that an 8-bit device is currently being addressed.	—
BUSY#	I	Busy: Indicates that the math coprocessor is busy. If BUSY# is sampled low at the falling edge of RESET, the processor performs an internal self test.	TMRGATE2
CAS2:0	O	Cascade Address: Carries the slave address information from the master 8259A interrupt module during interrupt acknowledge bus cycles.	A18:16
CLK2	ST	Input Clock: Is connected to an external clock that provides the fundamental timing for the microprocessor. The internal processor clock frequency is half the CLK2 frequency.	—
CLKOUT	O	Clock Output: Use this output to synchronize external devices with the processor.	—
COMCLK	I	SIO Baud Clock: An external source connected to this pin can clock the SIO*n* baud-rate generator.	P3.7
CS6# CS5# CS4# CS3# CS2# CS1# CS0#	O	Chip-selects: Activated when the address of a memory or I/O bus cycle is within the address region programmed by the user.	REFRESH# DACK0# P2.4 P2.3 P2.2 P2.1 P2.0

Table A-2. Description of Signals Available at the Device Pins (Sheet 2 of 6)

Signal	Type	Name and Description	Multiplexed With (Alternate Function)
CTS1# CTS0#	I	Clear to Send: Indicates that the modem or data set is ready to exchange data with the SIO channel.	EOP# P2.7
D15:0	I/O	Data Bus: Inputs data during memory read, I/O read, and interrupt acknowledge cycles; outputs data during memory write and I/O write cycles. During reads, data is latched during the falling edge of phase 2 of T2, T2P, or T2i. During writes, this bus is driven during phase 2 of T1 and remains active until phase 2 of the next T1, T1P, or Ti.	—
DACK1# DACK0#	O	DMA Channel Acknowledge: Indicates that the DMA channel is ready to service the requesting device. An external device uses the DRQ*n* pin to request DMA service; the DMA uses the DACK*n* pin to indicate that the request is being serviced.	TXD1 CS5#
D/C#	O	Data/Control: Indicates whether the current bus cycle is a data cycle (memory or I/O read or write) or a control cycle (interrupt acknowledge, halt/shutdown, or code fetch).	—
DCD1# DCD0#	I	Data Carrier Detect: Indicates that the modem or data set has detected the SIO channel's data carrier.	DRQ0 P1.0
DRQ1 DRQ0	I	DMA External Request: Indicates that an external device requires DMA service.	RXD1 DCD1#
DSR1# DSR0#	I	Data Set Ready: Indicates that the modem or data set is ready to establish the communications link with the SIO channel.	STXCLK P1.3
DTR1# DTR0#	O	Data Terminal Ready: Indicates that the SIO channel is ready to establish a communications link with the modem or data set.	SRXCLK P1.2
EOP#	I/OD	End-of-process: As an input, this signal terminates a DMA transfer. As an ouput, it indicates that a DMA transfer has completed.	CTS1#
ERROR#	I	Error: Indicates the the math coprocessor has an error condition.	TMROUT2
FLT#	I	Float: Forces all bidirectional and output signals except TDO to a high-impedance state.	—
HLDA	O	Hold Acknowledge: Indicates that the processor has relinquished local bus control to another bus master in response to a HOLD request.	P1.7

Table A-2. Description of Signals Available at the Device Pins (Sheet 3 of 6)

Signal	Type	Name and Description	Multiplexed With (Alternate Function)
HOLD	I	Hold Request: An external bus master asserts HOLD to request control of the local bus. The processor finishes the current nonlocked bus transfer, releases the bus signals, and asserts HLDA.	P1.6
INT9 INT8 INT7 INT6 INT5 INT4 INT3 INT2 INT1 INT0	I	Interrupt Requests: These maskable inputs cause the processor to suspend execution of the current program and execute an interrupt acknowledge cycle.	P3.0/TMROUT0 P3.1/TMROUT1 TMRGATE1 TMRCLK1 TMRGATE0 TMRCLK0 P3.5 P3.4 P3.3 P3.2
LBA#	O	Local Bus Access: Indicates that the processor provides the READY# signal internally to terminate a bus transaction. This signal is active when the processor accesses an internal peripheral or when the chip-select unit provides the READY# signal for an external peripheral.	—
LOCK#	O	Bus Lock: Prevents other bus masters from gaining control of the bus.	P1.5
M/IO#	O	Memory/IO: Indicates whether the current bus cycle is a memory cycle or an I/O cycle.	—
NA#	I	Next Address: Requests address pipelining.	—
NMI	ST	Nonmaskable Interrupt Request: Causes the processor to suspend execution of the current program and execute an interrupt acknowledge cycle.	—
PEREQ	I	Processor Extension Request: Indicates that the math coprocessor has data to transfer to the processor.	TMRCLK2
P1.7 P1.6 P1.5 P1.4 P1.3 P1.2 P1.1 P1.0	I/O	Port 1: General-purpose, bidirectional I/O port.	HLDA HOLD LOCK# RI0# DSR0# DTR0# RTS0# DCD0#

Table A-2. Description of Signals Available at the Device Pins (Sheet 4 of 6)

Signal	Type	Name and Description	Multiplexed With (Alternate Function)
P2.7 P2.6 P2.5 P2.4 P2.3 P2.2 P2.1 P2.0	I/O	Port 2: General-purpose, bidirectional I/O port.	CTS0# TXD0 RXD0 CS4# CS3# CS2# CS1# CS0#
P3.7 P3.6 P3.5 P3.4 P3.3 P3.2 P3.1 P3.0	I/O	Port 3: General-purpose, bidirectional I/O port.	COMCLK PWRDOWN INT3 INT2 INT1 INT0 TMROUT1/INT8 TMROUT0/INT9
PWRDOWN	O	Powerdown Output: Indicates that the device is in powerdown mode.	P3.6
RD#	O	Read Enable: Indicates that the current bus cycle is a read cycle and the data bus is able to accept data.	
READY#	I/O	Ready: Terminates the current bus cycle. The processor drives READY# when LBA# is active; otherwise, the processor samples READY# on the falling edge of phase 2 of T2, T2P or T2i.	—
REFRESH#	O	Refresh: Indicates that a refresh bus cycle is in progress and that the refresh address is on the bus for the DRAM controller.	CS6#
RESET	ST	System Reset Input: Suspends any operation in progress and places the processor into a known reset state.	—
RI1# RI0#	I	Ring Indicator: Indicates that the modem or data set has received a telephone ringing signal.	SSIORX P1.4
RTS1# RTS0#	O	Request to Send: Indicates that the SIO channel is ready to exchange data with the modem or data set.	SSIOTX P1.1
RXD1 RXD0	I	Receive Data: Accepts data from the modem or data set to the SIO channel.	DRQ1 P2.5

Table A-2. Description of Signals Available at the Device Pins (Sheet 5 of 6)

Signal	Type	Name and Description	Multiplexed With (Alternate Function)
SMI#	ST	System Management Interrupt: Causes the device to enter System Management Mode. SMI# is the highest priority external interrupt.	—
SMIACT#	O	System Management Interrupt Active: Indicates that the processor is in System Management Mode.	—
SRXCLK	I/O	SSIO Receive Clock: In master mode, the baud-rate generator's output appears on SRXCLK and can be used to clock a slave transmitter. In slave mode, SRXCLK functions as an input clock for the receiver.	DTR1#
SSIORX	I	SSIO Receive Serial Data: Accepts serial data (most-significant bit first) into the SSIO.	RI1#
SSIOTX	O	SSIO Transmit Serial Data: Sends serial data (most-significant bit first) from the SSIO.	RTS1#
STXCLK	I/O	SSIO Transmit Clock: In master mode, the baud-rate generator's output appears on STXCLK and can be used to clock a slave receiver. In slave mode, STXCLK functions as an input clock for the transmitter.	DSR1
TCK	I	Test Clock Input: Provides the clock input for the test-logic unit.	—
TDI	I	Test Data Input: Serial input for test instructions and data. Sampled on the rising edge of TCK; valid only when either the instruction register or a data register is being serially loaded.	—
TDO	O	Test Data Output: Serial output for test instructions and data. TDO shifts out the contents of the instruction register or the selected data register (LSB first) on the falling edge of TCK. If serial shifting is not taking place, TDO floats.	—
TMRCLK2 TMRCLK1 TMRCLK0	I	Timer/Counter Clock Input: An external clock source connected to the TMRCLK*n* pin can drive the corresponding timer/counter.	PEREQ INT6 INT4
TMRGATE2 TMRGATE1 TMRGATE0	I	Timer/Counter Gate Input: Can control the counter's operation (enable, disable, or trigger, depending on the programmed mode).	BUSY# INT7 INT5
TMROUT2 TMROUT1 TMROUT0	O	Timer/Counter Output: Can provide the timer/counter's output. The form of the output depends on the programmed mode.	ERROR# P3.1/INT8 P3.0/INT9

Table A-2. Description of Signals Available at the Device Pins (Sheet 6 of 6)

Signal	Type	Name and Description	Multiplexed With (Alternate Function)
TMS	I	Test Mode Select: Controls the sequence of the test-logic unit's TAP controller states. Sampled on the rising edge of TCK.	—
TRST#	ST	Test Reset: Resets the test-logic unit's TAP controller. Asynchronously clears the data registers and initializes the instruction register to 0010 (the IDCODE instruction opcode).	—
TXD1 TXD0	O	Transmit Data: Transmits serial data from the corresponding SIO channel.	DACK1# P2.6
UCS#	O	Upper Chip-select: Activated when the address of a memory or I/O bus cycle is within the address region programmed by the user.	—
V_{CC}	P	System Power: Provides the nominal DC supply input. Connected externally to a V_{CC} board plane.	—
V_{SS}	G	System Ground: Provides the 0 volt connection from which all inputs and outputs are measured. Connected externally to a ground board plane.	—
WDTOUT	O	Watchdog Timer Output: Indicates that the watchdog timer has expired.	—
W/R#	O	Write/Read: Indicates whether the current bus cycle is a write cycle or a read cycle.	—
WR#	O	Write Enable: Indicates that the current bus cycle is a write cycle.	—

Table A-3 defines the abbreviations used in Table A-4 to describe the pin states.

Table A-3. Pin State Abbreviations

Abbreviation	Description
1	Output driven to V_{CC}
0	Output driven to V_{SS}
Z	Output floats
Q	Output remains active
X	Output retains current state
WH	Pin floats and has a temporary weak pull-up
WL	Pin floats and has a temporary weak pull-down

Table A-4 lists the states of output and bidirectional pins after reset and during idle mode, powerdown, and hold.

Table A-4. Pin States After Reset and During Idle, Powerdown, and Hold (Sheet 1 of 2)

Symbol	Type	Pin State			
		Reset	Idle	Powerdown	Hold
A25:1	O	1	1	1	Z
ADS#	O	1	1	1	Z
BHE#	O	0	X	0	Z
BLE#	O	0	X	1	Z
CAS2:0	O	1	1	1	Z
CLKOUT	O	Q	Q	0	Q
CS4:0#	O	WH	Q	X	1
CS6:5#	O	1	Q	X	1
D15:0	I/O	Z	Z	Z	Z
DACK1:0#	O	1	Q	X	1
D/C#	O	1	0	0	Z
DTR1:0	O	WH	X	X	X
EOP#	I/OD	WH	Z	Z	Z
HLDA	O	WL	Q	X	1
LBA#	O	1	Q	X	1
LOCK#	O	WH	X	X	Z
M/IO#	O	0	1	1	Z
P1.5:0	I/O	WH	X	X	X
P1.7:6	I/O	WL	X	X	X
P2.4:0	I/O	WH	X	X	X
P2.6:5	I/O	WL	X	X	X
P2.7	I/O	WH	X	X	X
P3.7:0	I/O	WL	X	X	X
PWRDWN	O	WL	X	1	Q
RD#	O	1	1	1	1
READY#	I/O	Z	Z	Z	Z
REFRESH#	O	1	Q	X	1
RTS1#	O	WL	X	X	X
RTS0#	O	WH	X	X	X
SMIACT#	O	1	X	X	1
SRXCLK	I/O	WH	Q	X or Q(1)	Q
SSIOTX	O	WL	Q	X or Q(1)	Q

NOTES:
1. X if clock source is internal; Q if clock source is external.
2. Q when shifting data out through the JTAG port, otherwise Z.

Table A-4. Pin States After Reset and During Idle, Powerdown, and Hold (Sheet 2 of 2)

STXCLK	I/O	WH	Q	X or Q(1)	Q
TDO	O	Z or Q(2)	Z or Q(2)	Z or Q(2)	Z or Q(2)
TMROUT2	O	WH	Q	X or Q(1)	Q
TMROUT1:0	O	WL	Q	X or Q(1)	Q
TXD1	O	1	Q	X or Q(1)	Q
TXD0	O	WL	Q	X or Q(1)	Q
UCS#	O	0	Q	X	1
WDTOUT	O	0	Q	X	Q
W/R#	O	0	1	1	Z
WR#	O	1	1	1	1

NOTES:
1. X if clock source is internal; Q if clock source is external.
2. Q when shifting data out through the JTAG port, otherwise Z.

The following input pins have permanent weak pull-up resistors: TCK, TDI, TMS, TRST#, SMI#, PEREQ/TMRCLK2, and FLT#.

B

COMPATIBILITY WITH THE PC/AT* ARCHITECTURE

APPENDIX B
COMPATIBILITY WITH THE PC/AT* ARCHITECTURE

The Intel386™ EX embedded processor is **NOT** 100% PC/AT* compatible. Due to compatibility issues, not all PC software executes on the Intel386 EX processor. In addition, not all ISA/PC-104 cards operate in an Intel386 EX processor system.

It is the responsibility of the designer to determine if a specific PC/AT software or hardware package operates on an Intel386 EX processor system. Typically an embedded PC can be very different from a traditional desktop PC's system. The embedded PC may have more or less functionality than a desktop PC. It is important for the designer to evaluate the requirements of the PC software or hardware that is expected to operate on the Intel386 EX processor system.

This appendix is organized as follows:

B.1 HARDWARE DEPARTURES FROM PC/AT SYSTEM ARCHITECTURE

This appendix describes the areas in which the Intel386 EX processor departs from a standard PC/AT system architecture and explains how to work around those departures if necessary. Chapter 5, "DEVICE CONFIGURATION", shows an example configuration for a PC/AT-compatible system

B.1.1 DMA Unit

The PC/AT architecture uses two 8237A DMA controllers, connected in cascade, for a total of seven channels. One DMA controller allows byte transfers and the other allows word transfers. However, the 8237A has two major restrictions:

- It has only 16-bit addressing capability. This requires a page register to allow address extension for a system based on a processor like the Intel386 EX processor, with 26-bit (64 Mbyte) physical memory addressing capability. A page register implementation is cumbersome and degrades the system performance.
- The 8237A has no natural two-cycle data transfer mode to allow memory-to-memory transfers. Instead, two DMA channels have to be used in a very specific manner. Transferring data between memory and memory-mapped I/O devices, common in embedded applications, would not be easy.

To eliminate these problems with an 8237A DMA controller, the Intel386 EX processor integrates a DMA controller unit that differs from the 8237A DMA in these ways:

- It provides two channels, each capable of either byte or word transfers.
- Each channel can transfer data between any combination of memory and I/O. The Bus Interface Unit supports both external fly-by and two-cycle operation.
- For programming compatibility, the internal DMA unit preserves all of the 8-bit registers of the 8237A. The 8237A's command register bits that affect two-channel memory-to-memory transfers, compressed timing, and DREQ/DACK signal polarity selection are not supported by the internal DMA.
- The internal DMA uses 26-bit address registers to support the 26-bit address bus and uses 24-bit byte count registers to support larger data blocks than are possible with the 8237A. However, each channel can be configured to look like an 8237A with page registers (i.e., 16-bit address and byte count registers).

Chapter 12, "DMA CONTROLLER," describes the DMA unit's features in detail.

While the internal DMA offers a comprehensive set of features to meet the needs of most embedded applications, strict DOS compatibility may be critical to some. A PC/AT compatible system's Basic Input Output System (BIOS) only uses the DMA for floppy disk access. Since both MS-DOS* and Microsoft* Windows* make calls to the BIOS for disk access, it is possible to modify the BIOS. The floppy disk controller allows data transfers to occur using DMA, Polling, or Interrupt based. A few BIOS vendors have implemented the transfers using polling for disk transfers. Some programs bypass the BIOS and go directly to the hardware; typically, these are disk intensive programs like hard disk backup software or disk management software.

If more DMA channels are required for compatibility, external controllers could be added. The Intel386 EX processor's flexible address remapping scheme enables you to map the internal DMA out of the DOS I/O space and then connect an external 8237A to achieve PC/AT compatibility. The internal DMA can still be used for other non-DOS related functions.

B.1.2 Industry Standard Bus (ISA) Signals

The address, data, and control signals, along with the interrupt and DMA control signals, **do not** directly conform to the PC/AT ISA bus. (They more closely match the Intel386™ SX processor local bus signals.) However, you can easily construct a subset PC/AT ISA bus from these signals or a combination of these signals. For example, the AEN signal is typically generated as shown in Figure B-1 in a PC/AT-compatible system.

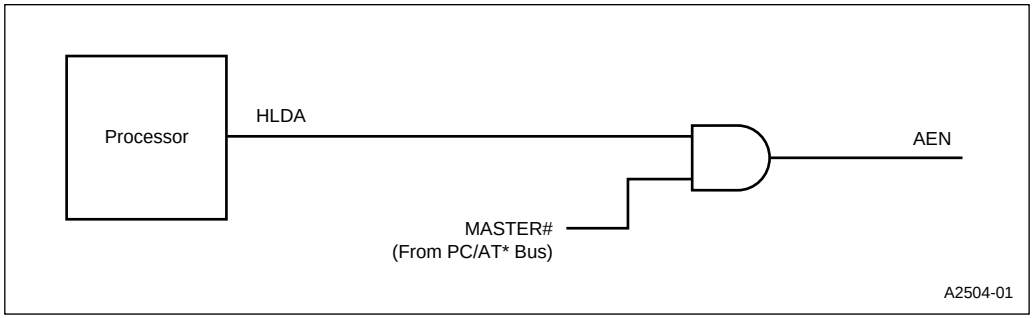


Figure B-1. Derivation of AEN Signal in a Typical PC/AT System

For systems based on Intel386 EX processor, the AEN signal could be derived as shown in Figure B-2. Notice that since the DMA acknowledge signals are used instead of a generic HLDA, there is no need to incorporate the REFRESH# signal in the logic.

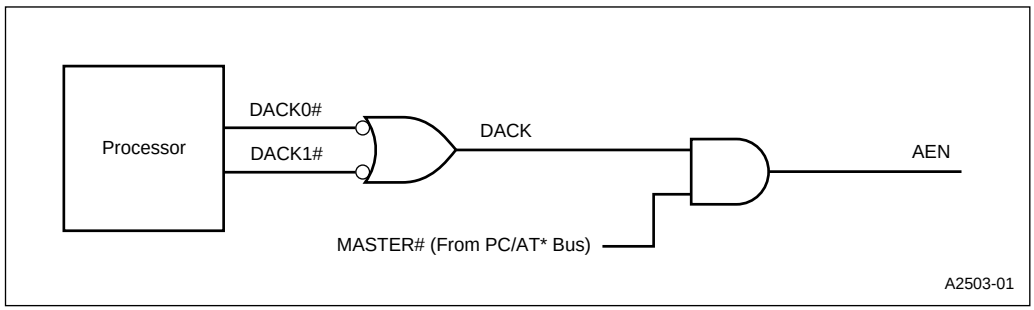


Figure B-2. Derivation of AEN Signal for Intel386™ EX processor-based Systems

In a PC/AT system using the 8237A DMA controller in fly-by mode, the 8237A generates appropriate control signals for memory (MEMR# or MEMW#) and for I/O (IOW# and IOR#). The Intel386 EX processor's internal DMA, during fly-by transfers, generates control signals (M/IO# and W/R#) that apply to the memory device. There needs to be some external logic that can detect the DMA operation (through the AEN signal) and generate a complementary I/O cycle. For example, if the DMA is generating a memory read cycle and AEN is active, then the logic should drive the IOW# signal on the PC/AT bus. Actually, the internal DMA could be programmed in a two-cycle mode eliminating the need for external logic. This will not have a significant impact on the performance — of the two cycles required to complete the transfer, the I/O cycle is the long one (meeting PC/AT timings) while the memory cycle is relatively very quick.

The drive capability and the operating frequency of the Intel386 EX processor signals are different from the standard PC/AT bus, which requires 24 mA drive capacity at 200 pF capacitive load.

Most PC/AT systems presently operate in a "quiet bus" mode so that non-ISA cycles are not reflected on the ISA bus. In a typical implementation, the address/data buses may change states, but the control signals are not strobed if a non-ISA cycle is detected. External three-state buffers and some decoding logic are needed to implement this scheme. The EV386EX (evaluation board for

the Intel386 EX embedded processor) demonstrates the design of a Synchronous Expansion Bus that is very similar to the ISA bus. The Intel386 EX processor is not capable of providing a 100% compatible ISA bus due to its lack of DMA channels and interrupt inputs.

B.1.3 Interrupt Control Unit

Interrupt signals IRQ10, IRQ11, and IRQ15 found on an ISA bus are not directly available for external interrupt connections in systems based on an Intel386 EX processor. If an application intends to use these IRQ*n* signals, then they can be rerouted to other IRQ signals available in an Intel386 EX processor architecture, and the respective interrupt handler routines assigned accordingly.

B.1.4 SIO Units

In the modem control register (MCR), the OUT1 register bit is used only in loopback tests. The OUT2 bit in the MCR is used as an SIO interrupt enable control signal. This allows two additional UARTs to be added externally as COM3 and COM4.

The SIO units (COM1 and COM2) are connected to the equivalent of a PC's local bus, not the ISA bus. However, this does not affect the compatibility with DOS application software in any form.

B.1.5 CPU-only Reset

The RESET pin on the Intel386 EX processor can be considered to function as a system reset function because all of the on-chip peripheral units, as well as the CPU core, are initialized to a known start-up state. There is no separate reset pin that goes only to the CPU. Some CPU-only reset modes, such as a keyboard controller generated CPU-only reset, will not function as expected.

A CPU-only reset can be implemented by routing the reset signal to either the NMI or SMI# signal, and the appropriate handler code could then generate a corresponding CPU-Only-Reset function by setting bit 0 of the PORT92H register.

B.1.6 HOLD, HLDA Pins

These pins do not connect directly to the CPU. Instead they go to the Bus Arbiter which controls the internal HOLD and HLDA signals connected to the CPU core. However the presence of the bus arbiter is transparent as far as functionality of the external HOLD and HLDA pins of Intel386 EX processor are concerned.

In a PC/AT system, if an external bus master gains the bus by raising HOLD to the CPU or raising DREQ in DMA cascade mode, the corresponding HLDA or DACK signal stays active until the bus master drops HOLD or DREQ. In the Intel386 EX processor, when the refresh control unit requests the bus, the bus arbiter deactivates the signals on the HLDA or DACK# pins while the external bus master still has the bus (HOLD or DREQ is high). At this point, the external bus master or DMA must deassert its HOLD or DREQ signal for a minimum of one CPU clock cycle and it can then assert the signal again.

B.1.7 Port B

The Port B register found on the PC/AT is not supported on the Intel386 EX processor. It can be implemented externally with a PLD. The EXPLR1 (Explorer Evaluation board) supports this Port B.

B.2 SOFTWARE CONSIDERATIONS FOR A PC/AT SYSTEM ARCHITECTURE

B.2.1 Embedded Basic Input Output System (BIOS)

The BIOS provides low-level drivers to interface to the hardware. The BIOS is hardware dependent and typically requires changes for the embedded design. There are several third party BIOS vendors that support the Intel386 EX processor.

Embedded PC features supported include PCMCIA, Flash, Advanced Power Management (APM), Source, Remote Floppy, OEM Configurable, and Video/Keyboard rerouted through the serial port. For a complete list of vendors and their features, call the Intel BBS as described in "Electronic Support Systems" on page 1-6. A good evaluation vehicle is the EV386EX Evaluation Board which comes with five different third party BIOS demonstrations. New and updated demonstrations are also available on the Intel BBS.

B.2.2 Embedded Disk Operating System (DOS)

The DOS operating system offers functions for I/O communication, floppy/hard disk, video, keyboard, program handling, memory management, and network support. All these are available to the Intel386 EX embedded processor user.

Embedded PC DOS features include Advanced Power Management (APM) support, ROMable, Source, Disk Compression, and XIP. A variety of third party DOS vendors support the Intel386 EX processor. For a complete list of vendors and their features, call the Intel BBS. The EV386EX Evaluation Board also provides a variety of DOS demonstrations. New and updated DOS demonstrations are also available on the Intel BBS.

B.2.3 Microsoft* Windows*

The Intel386 EX processor can run both Microsoft Windows 3.1 and Microsoft ROM Windows. Both require RAM and disk space to execute. Other hardware such as a keyboard controller, real time clock and video controller may be required. For more information on implementing ROM Windows, refer to *Mobile Intel486™ SX CPU PC Designs Using FlashFile™ Components* (Order Number 292149).

C

EXAMPLE CODE HEADER FILES

APPENDIX C
EXAMPLE CODE HEADER FILES

This appendix contains the header files called by the code examples that are included in several chapters of this manual. Section C.1 contains the register definitions for each code routine. Section C.2 contains the variable definitions.

C.1 REGISTER DEFINITIONS FOR CODE EXAMPLES

```
/*  80386EX REGISTER DEFINITIONS */
#define _SetEXRegWord(reg,val)    (outpw(reg,val))
#define _SetEXRegByte(reg,val)    (outp(reg,val))
#define _ReadEXRegWord(val,reg)   (val=inpw(reg))
#define _GetEXRegByte(reg)        inp(reg)
#define _GetEXRegWord(reg)        inpw(reg)

/* REMAP ADDRESSING Registers */
#define REMAPCFGH   0x0023
#define REMAPCFGL   0x0022
#define REMAPCFG    0x0022
/* INTERRUPT CONTROL REGISTERS -- SLOT 15 ADDRESSES */
#define ICW1M       0xF020
#define ICW1S       0xF0A0
#define ICW2M       0xF021
#define ICW2S       0xF0A1
#define ICW3M       0xF021
#define ICW3S       0xF0A1
#define ICW4M       0xF021
#define ICW4S       0xF0A1
#define OCW1M       0xF021
#define OCW1S       0xF0A1
#define OCW2M       0xF020
#define OCW2S       0xF0A0
#define OCW3M       0xF020
#define OCW3S       0xF0A0
/* INTERRUPT CONTROL REGISTERS -- SLOT 0 ADDRESSES */
#define ICW1MDOS    0x0020
#define ICW1SDOS    0x00A0
#define ICW2MDOS    0x0021
#define ICW2SDOS    0x00A1
#define ICW3MDOS    0x0021
#define ICW3SDOS    0x00A1
#define ICW4MDOS    0x0021
#define ICW4SDOS    0x00A1
#define OCW1MDOS    0x0021
#define OCW1SDOS    0x00A1
#define OCW2MDOS    0x0020
```

```
#define OCW2SDOS    0x00A0
#define OCW3MDOS    0x0020
#define OCW3SDOS    0x00A0


/* CONFIGURATION Registers */
#define DMACFG      0xF830
#define INTCFG      0xF832
#define TMRCFG      0xF834
#define SIOCFG      0xF836
#define P1CFG       0xF820
#define P2CFG       0xF822
#define P3CFG       0xF824
#define PINCFG      0xF826

/* WATCHDOG TIMER Registers */
#define WDTRLDH     0xF4C0
#define WDTRLDL     0xF4C2
#define WDTCNTH     0xF4C4
#define WDTCNTL     0xF4C6
#define WDTCLR      0xF4C8
#define WDTSTATUS   0xF4CA

/* TIMER CONTROL REGISTERS -- SLOT 15 ADDRESSES */
#define TMR0        0xF040
#define TMR1        0xF041
#define TMR2        0xF042
#define TMRCON      0xF043
/* TIMER CONTROL REGISTERS -- SLOT 0 ADDRESSES */
#define TMR0DOS     0x0040
#define TMR1DOS     0x0041
#define TMR2DOS     0x0042
#define TMRCONDOS   0x0043

/* INPUT/OUTPUT PORT UNIT Registers */
#define P1PIN       0xF860
#define P1LTC       0xF862
#define P1DIR       0xF864
#define P2PIN       0xF868
#define P2LTC       0xF86A
#define P2DIR       0xF86C
#define P3PIN       0xF870
#define P3LTC       0xF872
#define P3DIR       0xF874

/* ASYNCHRONOUS SERIAL CHANNEL 0 -- SLOT 15 ADDRESSES */
#define RBR0        0xF4F8
#define THR0        0xF4F8
#define TBR0        0xF4F8
#define DLL0        0xF4F8
#define IER0        0xF4F9
#define DLH0        0xF4F9
```

```
#define IIR0        0xF4FA
#define LCR0        0xF4FB
#define MCR0        0xF4FC
#define LSR0        0xF4FD
#define MSR0        0xF4FE
#define SCR0        0xF4FF
/* ASYNCHRONOUS SERIAL CHANNEL 0 -- SLOT 0 ADDRESSES */
#define RBR0DOS     0x03F8
#define THR0DOS     0x03F8
#define TBR0DOS     0x03F8
#define DLL0DOS     0x03F8
#define IER0DOS     0x03F9
#define DLH0DOS     0x03F9
#define IIR0DOS     0x03FA
#define LCR0DOS     0x03FB
#define MCR0DOS     0x03FC
#define LSR0DOS     0x03FD
#define MSR0DOS     0x03FE
#define SCR0DOS     0x03FF

/* ASYNCHRONOUS SERIAL CHANNEL 1 -- SLOT 15 ADDRESSES */
#define RBR1        0xF8F8
#define THR1        0xF8F8
#define TBR1        0XF8F8
#define DLL1        0xF8F8
#define IER1        0xF8F9
#define DLH1        0xF8F9
#define IIR1        0xF8FA
#define LCR1        0xF8FB
#define MCR1        0xF8FC
#define LSR1        0xF8FD
#define MSR1        0xF8FE
#define SCR1        0xF8FF
/* ASYNCHRONOUS SERIAL CHANNEL 1 -- SLOT 0 ADDRESSES */
#define RBR1DOS     0x02F8
#define THR1DOS     0x02F8
#define TBR1DOS     0x02F8
#define DLL1DOS     0x02F8
#define IER1DOS     0x02F9
#define DLH1DOS     0x02F9
#define IIR1DOS     0x02FA
#define LCR1DOS     0x02FB
#define MCR1DOS     0x02FC
#define LSR1DOS     0x02FD
#define MSR1DOS     0x02FE
#define SCR1DOS     0x02FF

/* SYNCHRONOUS SERIAL CHANNEL REGISTERS */
#define SSIOTBUF    0xF480
#define SSIORBUF    0xF482
#define SSIOBAUD    0xF484
#define SSIOCON1    0xF486
```

```
#define SSIOCON2    0xF488
#define SSIOCTR     0xF48A

/* CHIP SELECT UNIT Registers */
#define CS0ADL      0xF400
#define CS0ADH      0xF402
#define CS0MSKL     0xF404
#define CS0MSKH     0xF406
#define CS1ADL      0xF408
#define CS1ADH      0xF40A
#define CS1MSKL     0xF40C
#define CS1MSKH     0xF40E
#define CS2ADL      0xF410
#define CS2ADH      0xF412
#define CS2MSKL     0xF414
#define CS2MSKH     0xF416
#define CS3ADL      0xF418
#define CS3ADH      0xF41A
#define CS3MSKL     0xF41C
#define CS3MSKH     0xF41E
#define CS4ADL      0xF420
#define CS4ADH      0xF422
#define CS4MSKL     0xF424
#define CS4MSKH     0xF426
#define CS5ADL      0xF428
#define CS5ADH      0xF42A
#define CS5MSKL     0xF42C
#define CS5MSKH     0xF42E
#define CS6ADL      0xF430
#define CS6ADH      0xF432
#define CS6MSKL     0xF434
#define CS6MSKH     0xF436
#define UCSADL      0xF438
#define UCSADH      0xF43A
#define UCSMSKL     0xF43C
#define UCSMSKH     0xF43E

/* REFRESH CONTROL UNIT Registers */
#define RFSBAD      0xF4A0
#define RFSCIR      0xF4A2
#define RFSCON      0xF4A4
#define RFSADD      0xF4A6

/* POWER MANAGEMENT CONTROL Registers */
#define PWRCON      0xF800
#define CLKPRS      0xF804

/* DMA UNIT REGISTERS -- SLOT 15 ADDRESSES */
#define DMA0TAR     0xF000
#define DMA0BYC     0xF001
#define DMA1TAR     0xF002
#define DMA1BYC     0xF003
```

```
#define DMACMD1     0xF008
#define DMASTS      0xF008
#define DMASRR      0xF009
#define DMAMSK      0xF00A
#define DMAMOD1     0xF00B
#define DMACLRBP    0xF00C
#define DMACLR      0xF00D
#define DMACLRMSK   0xF00E
#define DMAGRPMSK   0xF00F
#define DMA0REQL    0xF010
#define DMA0REQH    0xF011
#define DMA1REQL    0xF012
#define DMA1REQH    0xF013
#define DMABSR      0xF018
#define DMACHR      0xF019
#define DMAIS       0xF019
#define DMACMD2     0xF01A
#define DMAMOD2     0xF01B
#define DMAIEN      0xF01C
#define DMAOVFE     0xF01D
#define DMACLRTC    0xF01E
#define DMA1TARPL   0xF083
#define DMA1TARPH   0xF085
#define DMA0TARPH   0xF086
#define DMA0TARPL   0xF087
#define DMA0BYCH    0xF098
#define DMA1BYCH    0xF099

/* DMA UNIT REGISTERS -- SLOT 0 ADDRESSES */
#define DMA0TARDOS    0x0000
#define DMA0BYCDOS    0x0001
#define DMA1TARDOS    0x0002
#define DMA1BYCDOS    0x0003
#define DMACMD1DOS    0x0008
#define DMASTSDOS     0x0008
#define DMASRRDOS     0x0009
#define DMAMSKDOS     0x000A
#define DMAMOD1DOS    0x000B
#define DMACLRBPDOS   0x000C
#define DMACLRDOS     0x000D
#define DMACLRMSKDOS  0x000E
#define DMAGRPMSKDOS  0x000F
#define DMA1TARPLDOS  0x0083
#define DMA0TARPLDOS  0x0087

/* A20GATE AND FAST CPU RESET -- SLOT 15 ADDRESS */
#define PORT92        0xF092
/* A20GATE AND FAST CPU RESET -- SLOT 0 ADDRESS */
#define PORT92DOS     0x0092
```

C.2 EXAMPLE CODE DEFINES

```
/********************** Global typedef *********************/
typedef unsigned char   BYTE;   /* 8-bit value */
typedef unsigned short  WORD;   /* 16-bit value */
typedef unsigned long   DWORD;  /* 32-bit value */

/******************* Global Used defines *******************/

/* Error Flags */
#define E_OK               0
#define E_INVALID_DEVICE   1
#define E_INVALID_VECTOR   2
#define E_BADVECTOR        3


#define INTERRUPT_ISR      1
#define TRAP_ISR           2
#define IDT_ALIAS          2            /* Only valid for protected mode */
#define TRAP_TYPE          0x8f00       /* Only valid for protected mode */
#define INTR_TYPE          0x8e00       /* Only valid for protected mode */


#define LOBYTE(w)      ((BYTE)(w))
#define HIBYTE(w)      ((BYTE)(((WORD)(w) >> 8) & 0xFF))

#define LOWORD(l)      ((WORD)(DWORD)(l))
#define HIWORD(l)      ((WORD)((((DWORD)(l)) >> 16) & 0xFFFF))

   /*** Bit Masks ***/
#define BIT0MSK            0x1
#define BIT1MSK            0x2
#define BIT2MSK            0x4
#define BIT3MSK            0x8
#define BIT4MSK            0x10
#define BIT5MSK            0x20
#define BIT6MSK            0x40
#define BIT7MSK            0x80

   /*** Global Function ***/
extern void _EnableExtIOMem(void);



/******* Interrupt Control Unit configuration defines ********/
   /* ICU Modes */
#define ICU_SFNM           0x10
#define ICU_AUTOEOI              0x2
#define ICU_TRIGGER_LEVEL        0x8
#define ICU_TRIGGER_EDGE         0x0
   /* ICU Master Pins */
```

```
#define MPIN_INT0               0x4
#define MPIN_INT1               0x8
#define MPIN_INT2               0x10
#define MPIN_INT3               0x20
   /* ICU Master External Cascade IRs */
#define MCAS_IR1                0x2
#define MCAS_IR2                0x4
#define MCAS_IR5                0x20
#define MCAS_IR6                0x40
#define MCAS_IR7                0x80
   /* ICU Slave Pins */
#define SPIN_INT4               0x1
#define SPIN_INT5               0x2
#define SPIN_INT6               0x4
#define SPIN_INT7               0x8
   /* ICU IRQ Mask Values*/
#define IR0                     0x1
#define IR1                     0x2
#define IR2                     0x4
#define IR3                     0x8
#define IR4                     0x10
#define IR5                     0x20
#define IR6                     0x40
#define IR7                     0x80
   /* ICU EOI Types */
#define NONSPECIFIC_EOI         0x20
#define SPECIFIC_EOI            0x60
#define NonSpecificEOI()       _SetEXRegByte(OCW2S,NONSPECIFIC_EOI);
                               _SetEXRegByte(OCW2M,NONSPECIFIC_EOI)
#define MstrSpecificEOI(irq)   _SetEXRegByte(OCW2M, 0x60 | ((BYTE)((irq) & 0x7))
)
#define SlaveSpecificEOI(irq)  _SetEXRegByte(OCW2S, 0x60 | ((BYTE)((irq) & 0x7))
)


#define Master                  1
#define Slave                   0

   /* ICU Function Definitions */
extern int  InitICU  (BYTE MstrMode, BYTE MstrBase, BYTE MstrCascade,
                      BYTE SlaveMode, BYTE SlaveBase,BYTE MstrPins,
                      BYTE SlavePins);
extern int  InitICUSlave(BYTE SlaveMode, BYTE SlaveBase, BYTE SlavePins);
extern void SetInterruptVector(void (far interrupt *IntrProc)(void),
                               int Vector, int IntrType);
extern int  SetIRQVector(void (far interrupt *IntrProc)(void), int IRQ,
                         int IntrType);
extern void Enable8259Interrupt(BYTE MstrMask, BYTE SlaveMask);
extern void Disable8259Interrupt(BYTE MstrMask, BYTE SlaveMask);
extern int  Poll_Command(int Master_or_Slave);
```

```
/************ Asynchronous Serial I/O Port defines ***********/
#define SIO_0              0
#define SIO_1              1

#define SIO0_IRQ           4       /* IRQ # Master IRQ4 */
#define SIO1_IRQ           3       /* IRQ # Master IRQ3 */

#define SIO_5DATA          0x0
#define SIO_6DATA          0x1
#define SIO_7DATA          0x2
#define SIO_8DATA          0x3

#define SIO_1STOPBIT       0x0
#define SIO_2STOPBIT       0x4

#define SIO_NOPARITY       0x0
#define SIO_ODDPARITY      0x8
#define SIO_EVNPARITY      0x18
#define SIO_FRC0PARITY     0x28
#define SIO_FRC1PARITY     0x38

#define SIO_SETBREAK       0x40

#define SIO_INTERNAL_SRC   0x1
#define SIO_EXTERNAL_SRC   0x0
#define SIO_CLKSRC_CLK2    0x1
#define SIO_CLKSRC_COMCLK  0x0

#define SIO_INTR_NONE      0
#define SIO_INTR_RBF       0x1
#define SIO_INTR_TBE       0x2
#define SIO_INTR_RLS       0x4
#define SIO_INTR_MS        0x8

#define SIO_MCR_LOOP_BACK  0x10
#define SIO_MCR_OUT2             0x8
#define SIO_MCR_OUT1             0x4
#define SIO_MCR_RTS              0x2
#define SIO_MCR_DTR              0x1

#define SIO_8N1                  (SIO_8DATA | SIO_1STOPBIT | SIO_NOPARITY)
#define SIO_7N1                  (SIO_7DATA | SIO_1STOPBIT | SIO_NOPARITY)

   /* Status Bits */
#define SIO_ERROR_BITS           0x1e
#define SIO_RX_BUF_FULL          0x1
#define SIO_OVERRUN              0x2
#define SIO_PARITY_ERR           0x4
#define SIO_FRAMING_ERR          0x8
#define SIO_BREAK_INTR           0x10
#define SIO_TX_BUF_EMPTY         0x20
```

```
#define SIO_TX_EMPTY            0x40


   /* Offsets from beginning of SIO port addresses */
#define RBR                     0
#define TBR                     0
#define DLL                     0
#define IER                     1
#define DLH                     1
#define IIR                     2
#define LCR                     3
#define MCR                     4
#define LSR                     5
#define MSR                     6
#define SCR                     7
#define SIO0_BASE               0xF4F8
#define SIO1_BASE               0xF8F8

   /* Define Function Macros */
#define GetSIO0Status()                 _GetEXRegByte(LSR0)
#define GetSIO1Status()                 _GetEXRegByte(LSR1)
#define GetSIO0InterruptID()            _GetEXRegByte(IIR0)
#define GetSIO1InterruptID()            _GetEXRegByte(IIR1)
#define GetSIO0ModemStatus()            _GetEXRegByte(MSR0)
#define GetSIO1ModemStatus()            _GetEXRegByte(MSR1)
#define GetSIO0Char()                   _GetEXRegByte(RBR0)
#define GetSIO1Char()                   _GetEXRegByte(RBR1)
#define ChangeSIO0IntrSrc(src)          _SetEXRegByte(IER0,src)
#define ChangeSIO1IntrSrc(src)          _SetEXRegByte(IER1,src)
#define ChangeSIO0Mode(Mode)            _SetEXRegByte(LCR0,Mode)
#define ChangeSIO1Mode(Mode)      _SetEXRegByte(LCR1,Mode)
#define DisableSIO0Interrupt(src) _SetEXRegByte(IER0,_GetEXRegByte(IER0) &
!(src))
#define DisableSIO1Interrupt(src) _SetEXRegByte(IER1,_GetEXRegByte(IER1) &
!(src))

   /* SIO Function Definitions */
extern int  InitSIO    (int Unit, BYTE Mode, BYTE ModemCntrl, DWORD BaudRate,
                        DWORD BaudClkIn);
extern BYTE SerialReadChar(int Unit);
extern int  SerialReadStr(int Unit, char far *str, int count);
extern void SerialWriteChar(int Unit, BYTE ch);
extern void SerialWriteStr(int Unit, const char far *str);
extern void SerialWriteMem(int Unit, const char far *mem, int count);
void interrupt far Serial0_ISR(void);
extern void Service_RBF (void);
extern void SerialWriteStr_Int(int Unit, const char far *str);
extern void Service_TBE(void);



/***************** DMA configuration defines *****************/
```

```
typedef enum
{
   DMA_Channel0 = 0,
   DMA_Channel1 = 1
} DMAChannelEnum;


typedef enum
{
   ERR_NONE = 0,
   ERR_BADINPUT = -1
} ERREnum;

   /* DMA Function Definitions */
int SetDMAReqIOAddr(int nChannel, WORD wIO);
int SetDMATargMemAddr(int nChannel, void *ptMemory);
int SetDMAXferCount(int nChannel, DWORD lCount);
int EnableDMAHWRequests(int nChannel);
int DisableDMAHWRequests(int nChannel);
void InitDMA(void);
void InitDMA1ForSerialXmitter(void);

/*************** Port I/O configuration defines **************/
   /* Port 1 configuration defines */
#define DCD0               0x1
#define RTS0               0x2
#define DTR0               0x4
#define DSR0               0x8
#define RI0                0X10
#define LOCK               0x20
#define HOLD               0X40
#define HOLDACK            0X80
   /* Port 2 configuration defines */
#define CS0                0x1
#define CS1                0x2
#define CS2                0x4
#define CS3                0x8
#define CS4                0X10
#define RXD0               0x20
#define TXD0               0X40
#define CTS0               0X80
   /* Port 3 configuration defines */
#define TMROUT0            0x1
#define TMROUT1            0x2
#define INT0               0x4
#define INT1               0x8
#define INT2               0x10
#define INT3               0x20
#define PWRDWN             0x40
#define COMCLK             0x80
   /* Port Direction defines */
#define P0_IN              0x1
```

```
#define P1_IN              0x2
#define P2_IN              0x4
#define P3_IN              0x8
#define P4_IN              0x10
#define P5_IN              0x20
#define P6_IN              0x40
#define P7_IN              0x80
#define Px_OUT             0

   /* Pin configuration defines */
#define RTS1               0x1
#define SSIOTX             0
#define DTR1               0x2
#define SRXCLK             0
#define TXD1               0x4
#define DACK1              0
#define CTS1               0x8
#define EOP                0
#define CS5                0x10
#define DACK0              0
#define TIMER2             0x20
#define COPROC             0
#define REFRESH            0x40
#define CS6                0

   /* Port I/O Function Definitions */
extern void Init_IOPorts   (BYTE Port1, BYTE Port2, BYTE Port3, BYTE PortDir1,
                            BYTE PortDir2, BYTE PortDir3, BYTE PortLtc1,
                            BYTE PortLtc2, BYTE PortLtc3);


/**************** Timer configuration defines ***************/
#define TMR_0              0
#define TMR_1              1
#define TMR_2              2
#define TMR0_IRQ           0       /* IRQ # Master IRQ0 */
#define TMR1_IRQ           10      /* IRQ # Slave IRQ2 */
#define TMR2_IRQ           11      /* IRQ # Slave IRQ3 */
   /* Timer Modes */
#define TMR_TERMCNT        0
#define TMR_1SHOT          (1<<1)
#define TMR_RATEGEN        (2<<1)
#define TMR_SQWAVE         (3<<1)
#define TMR_SW_TRIGGER     (4<<1)
#define TMR_HW_TRIGGER     (5<<1)
   /* Count Type */
#define TMR_CLK_BCD        1
#define TMR_CLK_BIN        0
   /* Timer Pin Configuration */
#define TMR_CLK_INTRN      0
#define TMR_CLK_EXTRN      0x1
#define TMR_GATE_VCC       0
```

```
#define TMR_GATE_EXTRN     0x2
#define TMR_OUT_ENABLE     0x1
#define TMR_OUT_DISABLE    0

#define TMR_ENABLE         1
#define TMR_DISABLE        0

   /* Timer Macros Definitions */
#define DisableTimer() \
    _SetEXRegByte( TMRCFG, (_GetEXRegByte(TMRCFG) | 0x80))

#define EnableTimer() \
    _SetEXRegByte( TMRCFG, (_GetEXRegByte(TMRCFG) & 0x7f))

   /* Timer Function Definitions */
extern int InitTimer   (int Unit, WORD Mode, BYTE Inputs, BYTE Output,
                        WORD InitCount, int Enable);

extern void SetUp_ReadBack (BYTE Timer0, BYTE Timer1, BYTE Timer2,
                            BYTE GetStatus, BYTE GetCount);

extern WORD CounterLatch(BYTE Timer);

extern WORD ReadCounter(BYTE Timer);

void interrupt far TimerISR(void);


/**************** SSIO configuration defines *****************/

#define SSIO_TUE                0x80    /* Transmit Underflow Error */
#define SSIO_THBE               0x40    /* Transmit Holding Buffer Empty */
#define SSIO_TX_IE              0x20    /* Transmit Interrupt Enable */
#define SSIO_TX_ENAB            0x10    /* Transmitter Enable */
#define SSIO_ROE                0x08    /* Receive Overflow Error */
#define SSIO_RHBF               0x04    /* Receive Holding Buffer Full */
#define SSIO_RX_IE              0x02    /* Receive Interrupt Enable */
#define SSIO_RX_ENAB            0x01    /* Receiver Enable */

#define SSIO_TX_MASTR           0x02    /* Transmit Master Mode */
#define SSIO_RX_MASTR           0x01    /* Receive Master Mode */
#define SSIO_TX_SLAVE           0
#define SSIO_RX_SLAVE           0

#define SSIO_CLK_SERCLK         0x01    /* Baud Rate Clocking Source:
                                           SERCLK = CLK2/4 */
#define SSIO_CLK_PSCLK          0x00    /* Baud Rate Clocking Source:
                                           PSCLK  = (CLK2/2) / (CLKPRS+2) */
#define SSIO_BAUD_ENAB          0x80    /* Enable Baud Rate Generator */

   /* SSIO Function Definitions */
extern void InitSSIO   (BYTE Mode, BYTE MasterTxRx, BYTE BaudValue,
```

```
                        BYTE PreScale);
extern WORD SSerialReadWord(BYTE MasterSlave);
extern void SSerialWriteWord(WORD Ch,BYTE MasterSlave);
void interrupt far SSIO_ISR(void);
extern void Service_RHBF(void);
extern void Service_THBE(void);


/******************** Watch Dog Timer **********************/
#define SetWatchDogReload(ReloadHi,ReloadLow) \
   _SetEXRegWord(WDTRLDL,ReloadLow);_SetEXRegWord(WDTRLDH,ReloadHi);

#define WatchDogClockDisable()\
   _SetEXRegByte(WDTSTATUS, _GetEXRegByte(WDTSTATUS) | BIT0MSK)

#define WatchDogClockEnable()\
   _SetEXRegByte(WDTSTATUS, _GetEXRegByte(WDTSTATUS) & ~BIT0MSK)

   /* Watch Dog Timer Function Definitions */
extern void  ReLoadDownCounter(void);
extern DWORD GetWDT_Count(void);
extern void  WDT_BusMonitor(BYTE EnableDisable);
extern void  EnableWDTInterrupt(void);
void interrupt far wdtISR(void);



/********************Refresh Control Unit*********************/

#define EnableRCU() \
    _SetEXRegWord(RDFSCON, _GetEXRegWord(RDFSCON) | 0x8000)

#define DisableRCU() \
    _SetEXRegWord(RDFSCON, _GetEXRegWord(RDFSCON) & 0x7fff)

   /* Refresh Control Unit Function Definitions */
extern int InitRCU(WORD counter_value);

extern WORD Get_RCUCounterValue(void);


/****************Clock and Power Management Unit**************/

#define IDLE               0x02
#define PWDWN              0x01
#define ACTIVE             0x00

   /* Clock and Power Management Function Definitions */
extern int  Set_Prescale_Value(WORD prescale);
extern void Enter_Idle_Mode(void);
extern void Enter_Powerdown_Mode(void);
extern void Mode_Setting_To_Active(void);
```

D

SYSTEM REGISTER QUICK REFERENCE

APPENDIX D
SYSTEM REGISTER QUICK REFERENCE

D.1 PERIPHERAL REGISTER ADDRESSES

Table D-1. Peripheral Register Addresses (Sheet 1 of 6)

Expanded Address	PC/AT Address	Access Type (Byte/Word)	Register Name	Reset Value
DMA Controller and Bus Arbiter				
F000H	0000H	Byte	DMA0TAR0/1 (Note 1)	XX
F001H	0001H	Byte	DMA0BYC0/1 (Note 1)	XX
F002H	0002H	Byte	DMA1TAR0/1 (Note 1)	XX
F003H	0003H	Byte	DMA1BYC0/1 (Note 1)	XX
F004H	0004H		Reserved	
F005H	0005H		Reserved	
F006H	0006H		Reserved	
F007H	0007H		Reserved	
F008H	0008H	Byte	DMACMD1/DMASTS	00H
F009H	0009H	Byte	DMASRR	00H
F00AH	000AH	Byte	DMAMSK	04H
F00BH	000BH	Byte	DMAMOD1	00H
F00CH	000CH	Byte	DMACLRBP	Not a register
F00DH	000DH	Byte	DMACLR	Not a register
F00EH	000EH	Byte	DMACLRMSK	Not a register
F00FH	000FH	Byte	DMAGRPMSK	03H
F010H		Byte	DMA0REQ0/1	XX
F011H		Byte	DMA0REQ2/3	XX
F012H		Byte	DMA1REQ0/1	XX
F013H		Byte	DMA1REQ2/3	XX
F014H			Reserved	
F015H			Reserved	
F016H			Reserved	
F017H			Reserved	
F018H		Byte	DMABSR	X1X10000B

NOTES:
1. Byte pointer in flip-flop in DMA determines which register is accessed.
2. Shaded rows indicate reserved areas.

Table D-1. Peripheral Register Addresses (Sheet 2 of 6)

Expanded Address	PC/AT Address	Access Type (Byte/Word)	Register Name	Reset Value
F019H		Byte	DMACHR/DMAIS	00H
F01AH		Byte	DMACMD2	08H
F01BH		Byte	DMAMOD2	00H
F01CH		Byte	DMAIEN	00H
F01DH		Byte	DMAOVFE	0AH
F01EH		Byte	DMACLRTC	Not a register
Master Interrupt Controller				
F020H	0020H	Byte	ICW1m/IRRm/ISRm/ OCW2m/OCW3m	XX
F021H	0021H	Byte	ICW2m/ICW3m/ICW4m/ OCW1m/POLLm	XX
Address Configuration Register				
0022H	0022H	Word	REMAPCFG	0000H
Timer/counter Unit				
F040H	0040H	Byte	TMR0	XX
F041H	0041H	Byte	TMR1	XX
F042H	0042H	Byte	TMR2	XX
F043H	0043H	Byte	TMRCON	XX
DMA Page Registers				
F080H			Reserved	
F081H	0081H		Reserved	
F082H	0082H		Reserved	
F083H	0083H	Byte	DMA1TAR2	XX
F084H			Reserved	
F085H		Byte	DMA1TAR3	XX
F086H		Byte	DMA0TAR3	XX
F087H	0087H	Byte	DMA0TAR2	XX
F088H			Reserved	
F089H	0089H		Reserved	
F08AH	008AH		Reserved	
F08BH	008BH		Reserved	
F08CH			Reserved	
F08DH			Reserved	

NOTES:
1. Byte pointer in flip-flop in DMA determines which register is accessed.
2. Shaded rows indicate reserved areas.

Table D-1. Peripheral Register Addresses (Sheet 3 of 6)

Expanded Address	PC/AT Address	Access Type (Byte/Word)	Register Name	Reset Value
F08EH			Reserved	
F08FH			Reserved	
F098H		Byte	DMA0BYC2	XX
F099H		Byte	DMA1BYC2	XX
F09AH			Reserved	
F09BH			Reserved	
A20GATE and Fast CPU Reset				
F092H	0092H	Byte	PORT92	XXXXXX10B
Slave Interrupt Controller				
F0A0H	00A0H	Byte	ICW1s/IRRs/ISRs/ OCW2s/OCW3s	XX
F0A1H	00A1H	Byte	ICW2s/ICW3s/ICW4s/ OCW1s/POLLs	XX
Chip-select Unit				
F400H		Word	CS0ADL	0000H
F402H		Word	CS0ADH	0000H
F404H		Word	CS0MSKL	0000H
F406H		Word	CS0MSKH	0000H
F408H		Word	CS1ADL	0000H
F40AH		Word	CS1ADH	0000H
F40CH		Word	CS1MSKL	0000H
F40EH		Word	CS1MSKH	0000H
F410H		Word	CS2ADL	0000H
F412H		Word	CS2ADH	0000H
F414H		Word	CS2MSKL	0000H
F416H		Word	CS2MSKH	0000H
F418H		Word	CS3ADL	0000H
F41AH		Word	CS3ADH	0000H
F41CH		Word	CS3MSKL	0000H
F41EH		Word	CS3MSKH	0000H
F420H		Word	CS4ADL	0000H
F422H		Word	CS4ADH	0000H
F424H		Word	CS4MSKL	0000H

NOTES:
1. Byte pointer in flip-flop in DMA determines which register is accessed.
2. Shaded rows indicate reserved areas.

Table D-1. Peripheral Register Addresses (Sheet 4 of 6)

Expanded Address	PC/AT Address	Access Type (Byte/Word)	Register Name	Reset Value
F426H		Word	CS4MSKH	0000H
F428H		Word	CS5ADL	0000H
F42AH		Word	CS5ADH	0000H
F42CH		Word	CS5MSKL	0000H
F42EH		Word	CS5MSKH	0000H
F430H		Word	CS6ADL	0000H
F432H		Word	CS6ADH	0000H
F434H		Word	CS6MSKL	0000H
F436H		Word	CS6MSKH	0000H
F438H		Word	UCSADL	FF6FH
F43AH		Word	UCSADH	FFFFH
F43CH		Word	UCSMSKL	FFFFH
F43EH		Word	UCSMSKH	FFFFH
Synchronous Serial I/O Unit				
F480H		Word	SSIOTBUF	0000H
F482H		Word	SSIORBUF	0000H
F484H		Byte	SSIOBAUD	00H
F486H		Byte	SSIOCON1	C0H
F488H		Byte	SSIOCON2	00H
F48AH		Byte	SSIOCTR	00H
Refresh Control Unit				
F4A0H		Word	RFSBAD	0000H
F4A2H		Word	RFSCIR	0000H
F4A4H		Word	RFSCON	0000H
F4A6H		Word	RFSADD	00FFH
Watchdog Timer Unit				
F4C0H		Word	WDTRLDH	003FH
F4C2H		Word	WDTRLDL	FFFFH
F4C4H		Word	WDTCNTH	003FH
F4C6H		Word	WDTCNTL	FFFFH
F4C8H		Word	WDTCLR	Not a register
F4CAH		Byte	WDTSTATUS	00H

NOTES:
1. Byte pointer in flip-flop in DMA determines which register is accessed.
2. Shaded rows indicate reserved areas.

Table D-1. Peripheral Register Addresses (Sheet 5 of 6)

Expanded Address	PC/AT Address	Access Type (Byte/Word)	Register Name	Reset Value
Asynchronous Serial I/O Channel 0 (COM1)				
F4F8H	03F8H	Byte	RBR0/TBR0/DLL0	XX/XX/02H
F4F9H	03F9H	Byte	IER0/DLH0	00H/00H
F4FAH	03FAH	Byte	IIR0	01H
F4FBH	03FBH	Byte	LCR0	00H
F4FCH	03FCH	Byte	MCR0	00H
F4FDH	03FDH	Byte	LSR0	60H
F4FEH	03FEH	Byte	MSR0	X0H
F4FFH	03FFH	Byte	SCR0	XX
Clock Generation and Power Management				
F800H		Byte	PWRCON	00H
F804H		Word	CLKPRS	0000H
Device Configuration Registers				
F820H		Byte	P1CFG	00H
F822H		Byte	P2CFG	00H
F824H		Byte	P3CFG	00H
F826H		Byte	PINCFG	00H
F830H		Byte	DMACFG	00H
F832H		Byte	INTCFG	00H
F834H		Byte	TMRCFG	00H
F836H		Byte	SIOCFG	00H
Parallel I/O Ports				
F860H		Byte	P1PIN	XX
F862H		Byte	P1LTC	FFH
F864H		Byte	P1DIR	FFH
F868H		Byte	P2PIN	XX
F86AH		Byte	P2LTC	FFH
F86CH		Byte	P2DIR	FFH
F870H		Byte	P3PIN	XX
F872H		Byte	P3LTC	FFH
F874H		Byte	P3DIR	FFH

NOTES:
1. Byte pointer in flip-flop in DMA determines which register is accessed.
2. Shaded rows indicate reserved areas.

Table D-1. Peripheral Register Addresses (Sheet 6 of 6)

Expanded Address	PC/AT Address	Access Type (Byte/Word)	Register Name	Reset Value
Asynchronous Serial I/O Channel 1 (COM2)				
F8F8H	02F8H	Byte	RBR1/TBR1/DLL1	XX/XX/02H
F8F9H	02F9H	Byte	IER1/DLH1	00H/00H
F8FAH	02FAH	Byte	IIR1	01H
F8FBH	02FBH	Byte	LCR1	00H
F8FCH	02FCH	Byte	MCR1	00H
F8FDH	02FDH	Byte	LSR1	60H
F8FEH	02FEH	Byte	MSR1	X0H
F8FFH	02FFH	Byte	SCR1	XX

NOTES:
1. Byte pointer in flip-flop in DMA determines which register is accessed.
2. Shaded rows indicate reserved areas.

D.2 CLKPRS

Clock Prescale Register
CLKPRS
(read/write)

Expanded Addr: **F804H**
ISA Addr: **—**
Reset State: **0000H**

15							**8**
—	—	—	—	—	—	—	PS8
7							**0**
PS7	PS6	PS5	PS4	PS3	PS2	PS1	PS0

Bit Number	Bit Mnemonic	Function
15–9	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
8–0	PS8:0	Prescale Value: These bits determine the divisor that is used to generate PSCLK. Legal values are from 0000H (divide by 2) to 01FFH (divide by 513). divisor = PS8:0 + 2

D.3 CS*n*ADH (UCSADH)

Chip-select High Address CS*n*ADH (n = 0–6), UCSADH (read/write)

Expanded Addr: **F402H, F40AH F412H, F41AH F422H, F42AH F432H, F43AH**
ISA Addr: **—**
Reset State: **0000H (CS*n*ADH) FFFFH (UCSADH)**

15							**8**
—	—	—	—	—	—	CA15	CA14
7							**0**
CA13	CA12	CA11	CA10	CA9	CA8	CA7	CA6

Bit Number	Bit Mnemonic	Function
15–10	—	Reserved; for compatibility with future devices, write zeros to these bits.
9–0	CA15:6	Chip-select Channel Address Upper Bits: Defines the upper 10 bits of the channel's 15-bit address. The address bits CA15:6 and the mask bits CM15:6 form a masked address that is compared to memory address bits A25:16 or I/O address bits A15:6.

D.4 CS*n*ADL (UCSADL)

Chip-select Low Address
CS*n*ADL (*n* = 0–6), UCSADL
(read/write)

Expanded Addr: **F400H, F408H**
F410H, F418H
F420H, F428H
F430H, F438H
ISA Addr: **—**
Reset State: **0000H (CS*n*ADL)**
FF6FH (UCSADL)

15							**8**
CA5	CA4	CA3	CA2	CA1	CASMM	BS16	MEM
7							**0**
RDY	—	—	WS4	WS3	WS2	WS1	WS0

Bit Number	Bit Mnemonic	Function
15–11	CA5:1	Chip-select Address Value Lower Bits: Defines the lower 5 bits of the channel's 15-bit address. The address bits CA5:1 and the mask bits CM5:1 form a masked address that is compared to memory address bits A15:11 or I/O address bits A5:1.
10	CASMM	SMM Address Bit: If this bit is set (and unmasked), the CSU activates the chip-select channel only while the processor is in SMM (and not in a hold state). Otherwise, the CSU activates the channel only when processor is operating in a mode other than SMM. Setting the SMM mask bit in the channel's mask low register masks this bit. When this bit is masked, an address match activates the chip-select, regardless of whether the processor is in SMM or not.
9	BS16	Bus Size 16-bit: 0 = All bus cycles to addresses in the channel's address block are byte-wide. 1 = Bus cycles are 16 bits unless the bus size control pin (BS8#) is asserted.
8	MEM	Bus Cycle Type: 0 = Configures the channel for an I/O addresses 1 = Configures the channel for memory addresses
7	RDY	Bus Ready Enable: 0 = External READY# is ignored. READY# generated by CSU to terminate the bus cycle. 1 = Requires that external READY# be active to complete a bus cycle. This bit must be set to extend wait states beyond the number determined by WS4:0 (see "Bus Cycle Length Control" on page 14-11).
6–5	—	Reserved; for compatibility with future devices, write zeros to these bits.
4–0	WS4:0	Wait State Value: WS4:0 defines the minimum number of wait states inserted into the bus cycle. A zero value means no wait states.

D.5 CS*n*MSKH (UCSMSKH)

Chip-select High Mask
CS*n*MSKH (*n* = 0–6), UCSMSKH
(read/write)

Expanded Addr: **F406H, F40EH**
F416H, F41EH
F426H, F42EH
F436H, F43EH
ISA Addr: **—**
Reset State: **0000H (CS*n*MSKH)**
FFFFH (UCSMSKH)

15							**8**
—	—	—	—	—	—	CM15	CM14
7							**0**
CM13	CM12	CM11	CM10	CM9	CM8	CM7	CM6

Bit Number	Bit Mnemonic	Function
15–10	—	Reserved; for compatibility with future devices, write zeros to these bits.
9–0	CM15:6	Mask Value Upper Bits: Defines the upper 10 bits of the channel's 15-bit mask. The mask bits CM15:6 and the address bits CA15:6 form a masked address that is compared to memory address bits A25:16 or I/O address bits A15:6.

D.6 CS*n*MSKL (UCSMSKL)

Chip-select Low Mask
CS*n*MSKL (*n* = 0–6), UCSMSKL
(read/write)

Expanded Addr: **F404H, F40CH**
F414H, F41CH
F424H, F42CH
F434H, F43CH
ISA Addr: **—**
Reset State: **0000H (CS*n*MSKL)**
FFFFH (UCSMSKL)

15 **8**

CM5	CM4	CM3	CM2	CM1	CMSMM	—	—

7 **0**

—	—	—	—	—	—	—	CSEN

Bit Number	Bit Mnemonic	Function
15–11	CM5:1	Chip-select Mask Value Lower Bits: Defines the lower 5 bits of the channel's 15-bit mask. The mask bits CM5:1 and the address bits CA5:1 form a masked address that is compared to memory address bits A15:11 or I/O address bits A5:1.
10	CMSMM	SMM Mask Bit: 0 = The SMM address bit is not masked. 1 = Masks the SMM address bit in the channel's Chip-Select Low Address register. When the SMM address bit is masked, an address match activates the chip-select, regardless of whether the processor is in SMM.
9–1	—	Reserved; for compatibility with future devices, write zeros to these bits.
0	CSEN	Chip-select Enable: 0 = Disables the chip-select channel. 1 = Enables the chip-select channel.

D.7 DLL*n* AND DLH*n*

Divisor Latch Low
DLL0, DLL1
(read/write)

	DLL0	DLL1
Expanded Addr:	**F4F8H**	**F8F8H**
ISA Addr:	**03F8H**	**02F8H**
Reset State:	**02H**	**02H**

7							0
LD7	LD6	LD5	LD4	LD3	LD2	LD1	LD0

Divisor Latch High
DLH0, DLH1
(read/write)

	DLH0	DLH1
Expanded Addr:	**F4F9H**	**F8F9H**
ISA Addr:	**03F9H**	**02F9H**
Reset State:	**00H**	**00H**

7							0
UD15	UD14	UD13	UD12	UD11	UD10	UD9	UD8

Bit Number	Bit Mnemonic	Function
DLL*n* (7–0)	LD7:0	Lower 8 Divisor and Upper 8 Divisor Bits: Write the lower 8 divisor bits to DLL*n* and the upper 8 divisor bits to DLH*n*. The baud-rate generator output is a function of the baud-rate generator input (BCLKIN) and the 16-bit divisor. $\text{baud-rate generator output frequency} = \frac{\text{BCLKIN frequency}}{\text{divisor}}$ bit rate (shifting rate) = baud-rate generator output frequency/16
DLH*n* (7–0)	UD15:8	

NOTE: The divisor latch registers share address ports with other SIO registers. Bit 7 (DLAB) of LCR*n* must be set in order to access the divisor latch registers.

If DLL = DLH = 00H, baud-rate generator ouput frequency = 0 (stops clock).

D.8 DMABSR

DMA Bus Size
DMABSR
(write only)

Expanded Addr: **F018H**
ISA Addr: **—**
Reset State: **X1X10000B**

7							0
—	RBS	—	TBS	—	—	0	CS

Bit Number	Bit Mnemonic	Function
7	—	Reserved; for compatibility with future devices, write zero to this bit.
6	RBS	Requester Bus Size: Specifies the requester's data bus width for the channel specified by bit 0. 0 = 16-bit bus 1 = 8-bit bus
5	—	Reserved; for compatibility with future devices, write zero to this bit.
4	TBS	Target Bus Size: Specifies the target's data bus width for the channel specified by bit 0. 0 = 16-bit bus 1 = 8-bit bus
3–1	0	Must be 0 for correct operation.
0	CS	Channel Select: 0 = The selections for bits 7–4 affect channel 0. 1 = The selections for bits 7–4 affect channel 1.

D.9 DMACFG

DMA Configuration
DMACFG
(read/write)

Expanded Addr: **F830H**
ISA Addr: **—**
Reset State: **00H**

7							0
D1MSK	D1REQ2	D1REQ1	D1REQ0	D0MSK	D0REQ2	D0REQ1	D0REQ0

Bit Number	Bit Mnemonic	Function
7	D1MSK	DMA Acknowledge 1 Mask: 0 = DMA channel 1's acknowledge (DMAACK1#) signal is not masked. 1 = Masks DMA channel 1's acknowledge (DMAACK1#) signal. Useful when channel 1's request (DREQ1) input is connected to an internal peripheral.
6–4	D1REQ2:0	DMA Channel 1 Request Connection: Connects one of the eight possible hardware sources to channel 1's request input (DREQ1). 000 = DRQ1 pin (external peripheral) 001 = SIO channel 1's receive buffer full signal (RBFDMA1) 010 = SIO channel 0's transmit buffer empty signal (TXEDMA0) 011 = SSIO receive holding buffer full signal (SSRBF) 100 = TCU counter 2's output signal (OUT2) 101 = SIO channel 0's receive buffer full signal (RBFDMA0) 110 = SIO channel 1's transmit buffer empty signal (TXEDMA1) 111 = SSIO transmit holding buffer empty signal (SSTBE)
3	D0MSK	DMA Acknowledge 0 Mask: 0 = DMA channel 0's acknowledge (DMAACK0#) signal is not masked. 1 = Masks DMA channel 0's acknowledge (DMAACK0#) signal. Useful when channel 0's request (DREQ0) input is connected to an internal peripheral.
2–0	D0REQ2:0	DMA Channel 0 Request Connection: Connects one of the eight possible hardware sources to channel 0's request input (DREQ0). 000 = DRQ0 pin (external peripheral) 001 = SIO channel 0's receive buffer full signal (RBFDMA0) 010 = SIO channel 1's transmit buffer empty signal (TXEDMA1) 011 = SSIO transmit holding buffer empty signal (SSTBE) 100 = TCU counter 1's output signal (OUT1) 101 = SIO channel 1's receive buffer full signal (RBFDMA1) 110 = SIO channel 0's transmit buffer empty signal (TXEDMA0) 111 = SSIO receive holding buffer full signal (SSRBF)

D.10 DMACHR

DMA Chaining
DMACHR
(write only)

Expanded Addr: **F019H**
ISA Addr: **—**
Reset State: **00H**

7							0
—	—	—	—	—	CE	0	CS

Bit Number	Bit Mnemonic	Function
7–3	—	Reserved; for compatibility with future devices, write zeros to these bits.
2	CE	Chaining Enable: 0 = Disables the chaining buffer-transfer mode for the channel specified by bit 0. 1 = Enables the chaining buffer-transfer mode for the channel specified by bit 0.
1	0	Must be 0 for correct operation.
0	CS	Channel Select: 0 = The selection for bit 2 affects channel 0. 1 = The selection for bit 2 affects channel 1.

D.11 DMACMD1

DMA Command 1
DMACMD1
(write only)

Expanded Addr: **F008H**
ISA Addr: **0008H**
Reset State: **00H**

7							0
—	—	—	PRE	—	CE	—	—

Bit Number	Bit Mnemonic	Function
7–5	—	Reserved; for compatibility with future devices, write zeros to these bits.
4	PRE	Priority Rotation Enable: 0 = Priority is fixed based on value in DMACMD2. 1 = Enables the rotation method for changing the bus control priority structure. That is, after the external bus master or one of the DMA channels is given bus control, it is assigned to the lowest priority level.
3	—	Reserved; for compatibility with future devices, write zero to this bit.
2	CE	Channel Enable: 0 = Enables channel 0 and 1. 1 = Disables the channels.
1–0	—	Reserved; for compatibility with future devices, write zeros to these bits.

D.12 DMACMD2

DMA Command 2
DMACMD2
(write only)

Expanded Addr: **F01AH**
ISA Addr: **—**
Reset State: **08H**

7							0
—	—	—	—	PL1	PL0	ES	DS

Bit Number	Bit Mnemonic	Function
7–4	—	Reserved; for compatibility with future devices, write zeros to these bits.
3–2	PL1:0	Low Priority Level Set: Use these bits to assign a particular bus request to the lowest priority level in fixed priority mode. 00 = Assigns channel 0's request (DREQ0) to the lowest priority level 01 = Assigns channel 1's request (DREQ1) to the lowest priority level 10 = Assigns the external bus master request (HOLD) to the lowest priority level 11 = Reserved
1	ES	EOP# Sampling: 0 = Causes the DMA to sample the EOP# input asynchronously. 1 = Causes the DMA to sample the end-of-process (EOP#) input synchronously.
0	DS	DREQ*n* Sampling: 0 = Causes the DMA to sample the DREQ*n* inputs asynchronously. 1 = Causes the DMA to sample the channel request (DREQ*n*) inputs synchronously.

D.13 DMAGRPMSK

DMA Group Channel Mask
DMAGRPMSK
(read/write)

Expanded Addr: **F00FH**
ISA Addr: **000FH**
Reset State: **03H**

7							0
—	—	—	—	—	—	HRM1	HRM0

Bit Number	Bit Mnemonic	Function
7–2	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
1	HRM1	Hardware Request Mask 1: 0 = Channel 1's hardware requests are not masked. 1 = Masks (disables) channel 1's hardware requests. When this bit is set, channel 1 can still receive software requests.
0	HRM0	Hardware Request Mask 0: 0 = Channel 0's hardware requests are not masked. 1 = Masks (disables) channel 0's hardware requests. When this bit is set, channel 0 can still receive software requests.

D.14 DMAIEN

DMA Interrupt Enable
DMAIEN
(read/write)

Expanded Addr: **F01CH**
ISA Addr: **—**
Reset State: **00H**

7							0
—	—	—	—	—	—	TC1	TC0

Bit Number	Bit Mnemonic	Function
7–2	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
1	TC1	Transfer Complete 1: 0 = Disables Transfer Complete interrupts. 1 = Connects channel 1's transfer complete signal to the interrupt control unit's DMAINT input. Note: When channel 1 is in chaining mode (DMACHR.2=1 and DMACHR.0=1), this bit is a don't care.
0	TC0	Transfer Complete 0: 0 = Disables Transfer Complete interrupts. 1 = Connects channel 0's transfer complete signal to the interrupt control unit's DMAINT input. Note: When channel 0 is in chaining mode (DMACHR.2=1 and DMACHR.0=0), this bit is a don't care.

D.15 DMAIS

DMA Interrupt Status
DMAIS
(read only)

Expanded Addr: **F019H**
ISA Addr: **—**
Reset State: **00H**

7							0
—	—	TC1	TC0	—	—	CI1	CI0

Bit Number	Bit Mnemonic	Function
7–6	—	Reserved. These bits are undefined.
5	TC1	Transfer Complete 1: When set, this bit indicates that channel 1 has completed a buffer transfer (either its byte count expired or it received an EOP# input). This bit is set only if bit 1 of the interrupt enable register is set. Reading the DMA status register (DMASTS) clears this bit. Note: In chaining mode, this bit becomes a don't care.
4	TC0	Transfer Complete 0: When set, this bit indicates that channel 0 has completed a buffer transfer (either its byte count expired or it received an EOP# input). This bit is set only if bit 0 of the interrupt enable register is set. Reading the DMA status register (DMASTS) clears this bit. Note: In chaining mode, this bit becomes a don't care.
3–2	—	Reserved. These bits are undefined.
1	CI1	Chaining Interrupt 1: When set, this bit indicates that new requester and target addresses and a new byte count should be written to channel 1. This bit is cleared when new transfer information is written to the channel. (Writing to the most-significant byte of the target address clears this bit.) Note: Outside chaining mode, this bit becomes a don't care.
0	CI0	Chaining Interrupt 0: When set, this bit indicates that new requester and target addresses and a new byte count should be written to channel 0. This bit is cleared when new transfer information is written to the channel. (Writing to the most-significant byte of the target address clears this bit.) Note: Outside chaining mode, this bit becomes a don't care.

D.16 DMAMOD1

DMA Mode 1
DMAMOD1
(write only)

Expanded Addr: **F00BH**
ISA Addr: **000BH**
Reset State: **00H**

7							0
DTM1	DTM0	TI	AI	TD1	TD0	0	CS

Bit Number	Bit Mnemonic	Function
7–6	DTM1:0	Data-transfer Mode: 00 = Demand 01 = Single 10 = Block 11 = Cascade
5	TI	Target Increment/Decrement: 0 = Causes the target address to be incremented after each data transfer in a buffer transfer. 1 = Causes the target address for the channel specified by bit 0 to be decremented after each data transfer in a buffer transfer. Note that it does not decrement words. When decrementing it will do two byte transfers for a word. Note: When the target address is programmed to remain constant (DMAMOD2.2 = 1), this bit is a don't care.
4	AI	Autoinitialize: 0 = Disables the autoinitialize buffer-transfer mode for the channel specified by bit 0. 1 = Enables the autoinitialize buffer-transfer mode for the channel specified by bit 0.
3–2	TD1:0	Transfer Direction: Determines the transfer direction for the channel specified by bit 0. 00 = Target is read; nothing is written (used for testing) 01 = Data is transferred from the requester to the target 10 = Data is transferred from the target to the requester 11 = Reserved Note: In cascade mode, these bits become don't cares.
1	0	Must be 0 for correct operation.
0	CS	Channel Select: 0 = The selections for bits 7–2 affect channel 0. 1 = The selections for bits 7–2 affect channel 1.

D.17 DMAMOD2

DMA Mode 2
DMAMOD2
(write only)

Expanded Addr: **F01BH**
ISA Addr: **—**
Reset State: **00H**

7							0
BCO	RD	TD	RH	RI	TH	0	CS

Bit Number	Bit Mnemonic	Function
7	BCO	Bus Cycle Option: 0 = Selects the fly-by data transfer bus cycle option for the channel specified by bit 0. 1 = Selects the two-cycle data transfer bus cycle option for the channel specified by bit 0.
6	RD	Requester Device Type: 0 = Clear this bit when the requester for the channel specified by bit 0 is in memory space. 1 = Set this bit when the requester for the channel specified by bit 0 is in I/O space. This bit is ignored if BCO is cleared.
5	TD	Target Device Type: 0 = Clear this bit when the target for the channel specified by bit 0 is in memory space. 1 = Set this bit when the target for the channel specified by bit 0 is in I/O space.
4	RH	Requester Address Hold: 0 = Causes the address to be modified (incremented or decremented, depending on DMAMOD2.3). 1 = Causes the requester's address for the channel specified by bit 0 to remain constant during a buffer transfer.
3	RI	Requester Address Increment/Decrement: 0 = Causes the requester address to be incremented after each data transfer in a buffer transfer. 1 = Causes the requester address for the channel specified by bit 0 to be decremented after each data transfer in a buffer transfer. Note that it does not decrement words. When decrementing it will do two byte transfers for a word. Note: When the target address is programmed to remain constant (DMAMOD2.4 = 1), this bit is a don't care.
2	TH	Target Address Hold: 0 = Causes the address to be modified (incremented or decremented, depending on DMAMOD1.5). 1 = Causes the target's address for the channel specified by bit 0 to remain constant during a buffer transfer.
1	0	Must be 0 for correct operation.
0	CS	Channel Select: 0 = The selections for bits 7–2 affect channel 0. 1 = The selections for bits 7–2 affect channel 1.

D.18 DMAMSK

DMA Individual Channel Mask
DMAMSK
(write only)

Expanded Addr: **F00AH**
ISA Addr: **000AH**
Reset State: **04H**

7							0
—	—	—	—	—	HRM	0	CS

Bit Number	Bit Mnemonic	Function
7–3	—	Reserved; for compatibility with future devices, write zeros to these bits.
2	HRM	Hardware Request Mask: 0 = Unmasks (enables) hardware requests for the channel specified by bit 0. 1 = Masks (disables) hardware requests for the channel specified by bit 0. NOTE: When this bit is set, the channel can still receive software requests.
1	0	Must be 0 for correct operation.
0	CS	Channel Select: 0 = The selection for bit 2 affects channel 0. 1 = The selection for bit 2 affects channel 1.

D.19 DMA*n*BYC*n*, DMA*n*REQ*n* AND DMA*n*TAR*n*

DMA Channel 0

	24	16	8	0
Requester Address	DMA0REQ3	DMA0REQ2	DMA0REQ1	DMA0REQ0
	F011H (BP=1)	F011H (BP=0)	F010H (BP=1)	F010H (BP=0)
	24	16	8	0
Target Address	DMA0TAR3	DMA0TAR2	DMA0TAR1	DMA0TAR0
	F086H	F087H	F000H (BP=1)	F000H (BP=0)
		16	8	0
Byte Count		DMA0BYC2	DMA0BYC1	DMA0BYC0
		F098H	F001H (BP=1)	F001H (BP=0)

DMA Channel 1

	24	16	8	0
Requester Address	DMA1REQ3	DMA1REQ2	DMA1REQ1	DMA1REQ0
	F013H (BP=1)	F013H (BP=0)	F012H (BP=1)	F012H (BP=0)
	24	16	8	0
Target Address	DMA1TAR3	DMA1TAR2	DMA1TAR1	DMA1TAR0
	F085H	F083H	F002H (BP=1)	F002H (BP=0)
		16	8	0
Byte Count		DMA1BYC2	DMA1BYC1	DMA1BYC0
		F099H	F003H (BP=1)	F003H (BP=0)

D.20 DMAOVFE

DMA Overflow Enable
DMAOVFE
(read/write)

Expanded Addr: **F01DH**
ISA Addr: **—**
Reset State: **0AH**

7							0
—	—	—	—	ROV1	TOV1	ROV0	TOV0

Bit Number	Bit Mnemonic	Function
7–4	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
3	ROV1	Channel 1 Requester Overflow Enable: 0 = lowest 16 bits of requester address increment/decrement 1 = all bits of requester address increment/decrement
2	TOV1	Channel 1 Target & Byte Counter Overflow Enable: 0 = lowest 16 bits of target address and byte count increment/decrement 1 = all bits of target address and byte count increment/decrement
1	ROV0	Channel 0 Requester Overflow Enable: 0 = lowest 16 bits of requester address increment/decrement 1 = all bits of requester address increment/decrement
0	TOV0	Channel 0 Target & Byte Counter Overflow Enable: 0 = lowest 16 bits of target address and byte count increment/decrement 1 = all bits of target address and byte count increment/decrement

D.21 DMASRR

DMA Software Request (read format)
DMASRR

Expanded Addr: F009H
ISA Addr: 0009H
Reset State: 00H

7							0
—	—	—	—	—	—	SR1	SR0

Bit Number	Bit Mnemonic	Function
7–2	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
1	SR1	Software Request 1: When set, this bit indicates that channel 1 has a software request pending.
0	SR0	Software Request 0: When set, this bit indicates that channel 0 has a software request pending.

DMA Software Request (write format)
DMASRR

Expanded Addr: F009H
ISA Addr: 0009H
Reset State: 00H

7							0
—	—	—	—	—	SR	0	CS

Bit Number	Bit Mnemonic	Function
7–3	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
2	SR	Software Request: Setting this bit generates a software request for the channel specified by bit 0. When the channel's buffer transfer completes, this bit is cleared.
1	0	Must be 0 for correct operation.
0	CS	Channel Select: 0 = The selection for bit 2 affects channel 0. 1 = The selection for bit 2 affects channel 1.

D.22 DMASTS

DMA Status
DMASTS
(read only)

Expanded Addr: **F008H**
ISA Addr: **0008H**
Reset State: **00H**

7							0
—	—	R1	R0	—	—	TC1	TC0

Bit Number	Bit Mnemonic	Function
7–6	—	Reserved. These bits are undefined.
5	R1	Request 1: When set, this bit indicates that channel 1 has a hardware request pending. When the request is removed, this bit is cleared.
4	R0	Request 0: When set, this bit indicates that channel 0 has a hardware request pending. When the request is removed, this bit is cleared.
3–2	—	Reserved. These bits are undefined.
1	TC1	Transfer Complete 1: When set, this bit indicates that channel 1 has completed a buffer transfer (either its byte count expired or it received an EOP# input). Reading this register clears this bit and clears TC1 in DMAIS.
0	TC0	Transfer Complete 0: When set, this bit indicates that channel 0 has completed a buffer transfer (either its byte count expired or it received an EOP# input). Reading this register clears this bit and clears TC0 in DMAIS.

D.23 ICW1 (MASTER AND SLAVE)

Initialization Command Word 1
ICW1 (master and slave)
(write only)

	master	slave
Expanded Addr:	**F020H**	**F0A0H**
ISA Addr:	**0020H**	**00A0H**
Reset State:	**XXH**	**XXH**

7							0
0	0	0	RSEL1	LS	0	0	1

Bit Number	Bit Mnemonic	Function
7–5	—	Clear these bits to guarantee device operation.
4	RSEL1	Register Select 1 (Also see OCW2 and OCW3): ICW1, OCW2, and OCW3 are accessed through the same addresses. 0 = OCW2 or OCW3 is accessed (Figure 9-13 and Figure 9-15). 1 = ICW1 register is accessed.
3	LS	Level/Edge Sensitive: 0 = Selects edge-triggered IR input signals. 1 = Selects level-sensitive IR input signals. All internal peripherals interface with the 82C59As in edge-triggered mode only. This is compatible with the PC/AT bus specification. Each source signal initiates an interrupt request by making a low-to-high transition. External peripherals interface with the 8259As in edge-triggered or level-sensitive mode. The modes are selected for the device, not for individual interrupts. NOTE: If an internal peripheral interrupt is used, the 8259A that the interrupt is connected to must be programmed for edge-triggered interrupts.
2–1	—	Clear these bits to guarantee device operation.
0	—	Set this bit to guarantee device operation.

NOTE: The 82C59A must be initialized before it can be used. After reset, the 82C59A register states are undefined. The 82C59A modules must be initialized before the IF flag in the core FLAG register is set. All peripherals that use interrupts connected to the ICU must be initialized before initializing the ICU.

D.24 ICW2 (MASTER AND SLAVE)

Initialization Command Word 2
ICW2 (master and slave)
(write only)

	master	slave
Expanded Addr:	**F021H**	**F0A1H**
ISA Addr:	**0021H**	**00A1H**
Reset State:	**XXH**	**XXH**

7							0
T7	T6	T5	T4	T3	0	0	0

Bit Number	Bit Mnemonic	Function
7–3	T7:3	Base Interrupt Type: Write the base interrupt vector's five most-significant bits to these bits.
2–0	T2:0	Clear these bits to guarantee device operation.

D.25 ICW3 (MASTER)

Initialization Command Word 3
ICW3 (master)
(write only)

Expanded Addr: **F021H**
ISA Addr: **0021H**
Reset State: **XXH**

7							0
S7	S6	S5	S4	S3	S2	S1	0

Bit Number	Bit Mnemonic	Function
7–3	S7:3	Slave IRs 0 = No slave 8259A is attached to the corresponding IR signal of the master. 1 = A slave 82C59A is attached to the corresponding IR signal of the master.
2	S2	0 = Internal slave not used 1 = Internal slave is cascaded from the master's IR2 signal.
1	S1	Slave IRs 0 = No slave 8259A is attached to the master through the IR1 signal of the master. 1 = A slave 82C59A is attached to the IR1 signal of the master.
0	—	Clear this bit to guarantee device operation.

D.26 ICW3 (SLAVE)

Initialization Command Word 3
ICW3 (slave)
(write only)

Expanded Addr: **F0A1H**
ISA Addr: **00A1H**
Reset State: **XXH**

7							0
0	0	0	0	0	0	1	0

Bit Number	Bit Mnemonic	Function
7–2	—	Clear these bits to guarantee device operation.
1	—	Set this bit to guarantee device operation.
0	—	Clear this bit to guarantee device operation.

D.27 ICW4 (MASTER AND SLAVE)

Initialization Command Word 4
ICW4 (master and slave)
(write only)

	master	**slave**
Expanded Addr:	**F021H**	**F0A1H**
ISA Addr:	**0021H**	**00A1H**
Reset State:	**XXH**	**XXH**

7							0
0	0	0	SFNM	0	0	AEOI	1

Bit Number	Bit Mnemonic	Function
7–5	—	Write zero to these bits to guarantee device operation.
4	SFNM	Special-fully Nested Mode: 0 = Selects fully nested mode. 1 = Selects special-fully nested mode. Only the master 82C59A can operate in special-fully nested mode.
3–2	—	Write zero to these bits to guarantee device operation.
1	AEOI	Automatic EOI Mode: 0 = Disables automatic EOI mode. 1 = Enables automatic EOI mode. Only the master 82C59A can operate in automatic EOI mode.
0	—	Write one to this bit to guarantee device operation.

D.28 IDCODE

Identification Code Register
IDCODE

Reset State: **2027 0013H (3V)**
2827 0013H (5V)

31							24
0	0	1	0	0 (3V) 1 (5V)	0	0	0
23							**16**
0	0	1	0	0	1	1	1
15							**8**
0	0	0	0	0	0	0	0
7							**0**
0	0	0	1	0	0	1	1

Bit Number	Bit Mnemonic	Function
31–28	V3:0	Device version number.
27–12	PN15:0	Device part number.
11–1	MFR10:0	Manufacturer identification (compressed JEDEC106-A code).
0	IDP	Identification Present. Always true for this device. This is the first data bit shifted out of the device during a data scan immediately following an exit from the test-logic-reset state. A one indicates that an IDCODE register is present. (A zero originates from the BYPASS register and indicates that the device being interrogated has **no** IDCODE register.)

D.29 IER*n*

Interrupt Enable
IER0, IER1
(read/write)

	IER0	IER1
Expanded Addr:	**F4F9H**	**F8F9H**
ISA Addr:	**03F9H**	**02F9H**
Reset State:	**00H**	**00H**

7							0
—	—	—	—	MS	RLS	TBE	RBF

Bit Number	Bit Mnemonic	Function
7–4	—	Reserved; for compatibility with future devices, write zeros to these bits.
3	MS	Modem Status Interrupt Enable: 0 = Modem input signal changes do not cause interrupts. 1 = Connects the modem status signal to the interrupt control unit's SIOINT*n* output. A change on one or more of the modem input signals activates the modem status signal.
2	RLS	Receiver Line Status Interrupt Enable: 0 = LSR error conditions do not cause interrupts. 1 = Connects the receiver line status signal to the interrupt control unit's SIOINT*n* output. Sources for this interrupt include overrun error, parity error, framing error, and break interrupt.
1	TBE	Transmit Buffer Empty Interrupt Enable: 0 = Transmit Buffer Empty signal does not cause interrupts. 1 = Connects the transmit buffer empty signal to the interrupt control unit's SIOINT*n* output.
0	RBF	Receive Buffer Full Interrupt Enable: 0 = Receive buffer full signal does not cause interrupts. 1 = Connects the receive buffer full signal to the interrupt control unit's SIOINT*n* output.

NOTE: The interrupt enable register is multiplexed with the divisor latch high register. You must clear bit 7 (DLAB) of the serial line control register (LCR*n*) before you can access the interrupt control register.

D.30 IIR*n*

Interrupt ID
IIR0, IIR1
(read only)

	IIR0	IIR1
Expanded Addr:	**F4FAH**	**F8FAH**
ISA Addr:	**03FAH**	**02FAH**
Reset State:	**01H**	**01H**

7							0
—	—	—	—	—	IS2	IS1	IP#

Bit Number	Bit Mnemonic	Function
7–3	—	Reserved. These bits are undefined.
2	IS2:1	Interrupt Source: If an interrupt is pending (bit 0 = 0), these bits specify which status signal caused the pending interrupt. **IS2 IS1 Interrupt Source** 0 0 modem status signal* 0 1 transmitter buffer empty signal 1 0 receive buffer full signal 1 1 receiver line status signal** * When one of the modem input signals (CTS*n*#, DSR*n*#, RI*n*#, and DCD*n*#) changes state, the modem status signal is activated. ** A framing error, overrun error, parity error, or break interrupt activates the receiver line status signal. Reading the modem status register clears the modem status signal. Reading the IIR*n* register or writing to the transmit buffer register clears the transmit buffer empty signal. Reading the receive buffer register clears the receive buffer full signal. Reading the receive buffer register or the serial line status register clears the LSR*n* error bits, which clears the receiver line status signal.
0	IP#	Interrupt Pending: This bit indicates whether an interrupt is pending. 0 = Interrupt is pending 1 = No interrupt is pending

D.31 INTCFG

Interrupt Configuration
INTCFG
(read/write)

Expanded Addr: **F832H**
ISA Addr: **—**
Reset State: **00H**

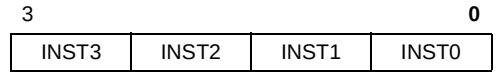
7							0
CE	IR3	IR4	SWAP	IR6	IR5/IR4	IR1	IR0

Bit Number	Bit Mnemonic	Function
7	CE	Cascade Enable: 0 = Disables the cascade signals CAS2:0 from appearing on the A18:16 address lines during interrupt acknowledge cycles. 1 = Enables the cascade signals CAS2:0, providing access to external slave 82C59A devices. The cascade signals are used to address specific slaves. If enabled, slave IDs appear on the A18:16 address lines during interrupt acknowledge cycles, but are high during idle cycles.
6	IR3	Internal Master IR3 Connection: See Table 5-1 on page 5-8 for all the IR3 configuration options.
5	IR4	Internal Master IR4 Connection: See Table 5-2 on page 5-8 for all the IR4 configuration options.
4	SWAP	INT6/DMAINT Connection: 0 = Connects DMAINT to the slave IR4. Connects INT6 to the slave IR5. 1 = Connects the INT6 pin to the slave IR4. Connects DMAINT to the slave IR5.
3	IR6	Internal Slave IR6 Connection: 0 = Connects V_{SS} to the slave IR6 signal. 1 = Connects the INT7 pin to the slave IR6 signal.
2	IR5/IR4	Internal Slave IR4 or IR5 Connection: These depend on whether INTCFG.4 is set or clear. 0 = Connects V_{SS} to the slave IR5 signal. 1 = Connects either the INT6 pin or DMAINT to the slave IR5 signal.
1	IR1	Internal Slave IR1 Connection: 0 = Connects the SSIO interrupt signal (SSIOINT) to the slave IR1 signal. 1 = Connects the INT5 pin to the slave IR1 signal.
0	IR0	Internal Slave IR0 Connection: 0 = Connects V_{SS} to the slave IR0 signal. 1 = Connects the INT4 pin to the slave IR0 signal.

D.32 IR

Instruction Register
IR

Reset State
(Using TRST#): 02H



Bit Number	Bit Mnemonic	Function
3–0	INST3:0	Instruction opcode. At reset (using TRST#, or after 5 TCK cycles with TMS held low), this field is loaded with 0010, the opcode for the IDCODE instruction. Instructions are shifted into this field serially through the TDI pin. (Table 18-4 lists the valid instruction opcodes.)

D.33 LCR*n*

Serial Line Control
LCR0, LCR1
(read/write)

	LCR0	LCR1
Expanded Addr:	**F4FBH**	**F8FBH**
ISA Addr:	**03FBH**	**02FBH**
Reset State:	**00H**	**00H**

7							**0**
DLAB	SB	SP	EPS	PEN	STB	WLS1	WLS0

Bit Number	Bit Mnemonic	Function
7	DLAB	Divisor Latch Access Bit: This bit determines which of the multiplexed registers is accessed. 0 = Allows access to the receiver and transmit buffer registers (RBR*n* and TBR*n*) and the interrupt enable register (IER*n*). 1 = Allows access to the divisor latch registers (DLL*n* and DLH*n*).
6	SB	Set Break: 0 = No effect on TXD*n*. 1 = Forces the TXD*n* pin to the spacing (logic 0) state for as long as bit is set.
5	SP	Sticky Parity, Even Parity Select, and Parity Enable: These bits determine whether the control logic produces (during transmission) or checks for (during reception) even, odd, no, or forced parity. **SP EPS PEN Function** X X 0 parity disabled (no parity option) 0 0 1 produce or check for odd parity 0 1 1 produce or check for even parity 1 0 1 produce or check for forced parity (parity bit = 1) 1 1 1 produce or check for forced parity (parity bit = 0)
4	EPS	
3	PEN	
2	STB	Stop Bits: This bit specifies the number of stop bits transmitted and received in each serial character. 0 = 1 stop bit 1 = 2 stop bits (1.5 stop bits for 5-bit characters)
1–0	WLS1:0	Word Length Select: These bits specify the number of data bits in each transmitted or received serial character. 00 = 5-bit character 01 = 6-bit character 10 = 7-bit character 11 = 8-bit character

D.34 LSR*n*

Serial Line Status
LSR0, LSR1
(read only)

	LSR0	LSR1
Expanded Addr:	**F4FDH**	**F8FDH**
ISA Addr:	**03FDH**	**02FDH**
Reset State:	**60H**	**60H**

7							**0**
—	TE	TBE	BI	FE	PE	OE	RBF

Bit Number	Bit Mnemonic	Function
7	—	Reserved. This bit is undefined.
6	TE	Transmitter Empty: The transmitter sets this bit to indicate that the transmit shift register and transmit buffer register are both empty. Writing to the transmit buffer register clears this bit.
5	TBE	Transmit Buffer Empty: The transmitter sets this bit after it transfers data from the transmit buffer to the transmit shift register. Writing to the transmit buffer register clears this bit.
4	BI	Break Interrupt: The receiver sets this bit whenever the received data input is held in the spacing (logic 0) state for longer than a full word transmission time. Reading the receive buffer register or the serial line status register clears this bit.
3	FE	Framing Error The receiver sets this bit to indicate that the received character did not have a valid stop bit. Reading the receive buffer register or the serial line status register clears this bit. If data frame is set for two stop bits the second stop bit is ignored.
2	PE	Parity Error: The receiver sets this bit to indicate that the received data character did not have the correct parity. Reading the receive buffer register or the serial line status register clears this bit.
1	OE	Overrun Error: The receiver sets this bit to indicate an overrun error. An overrun occurs when the receiver transfers a received character to the receive buffer register before the CPU reads the buffer's old character. Reading the serial line status register clears this bit.
0	RBF	Receive Buffer Full: The receiver sets this bit after it transfers a received character from the receive shift register to the receive buffer register. Reading the receive buffer register clears this bit.

D.35 MCR*n*

Modem Control
MCR0, MCR1
(read/write)

	MCR0	MCR1
Expanded Addr:	**F4FCH**	**F8FCH**
ISA Addr:	**03FCH**	**02FCH**
Reset State:	**00H**	**00H**

7 **0**

—	—	—	LOOP	OUT2	OUT1	RTS	DTR

Bit Number	Bit Mnemonic	Function
7–5	—	Reserved; for compatibility with future devices, write zeros to these bits.
4	LOOP	Loop Back Test Mode: 0 = Normal mode 1 = Setting this bit puts the SIO*n* into diagnostic (or loop back test) mode. This causes the SIO channel to: • set its transmit serial output (TXD*n*) • disconnect its receive serial input (RXD*n*) from the package pin • loop back the transmitter shift register's output to the receive shift register's input • disconnect the modem control inputs (CTS*n*#, DSR*n*#, RI*n*#, and DCD*n*#) from the package pins • force modem control outputs (RTS*n*# and DTR*n*#) to their inactive states • connects MCR*n* bits to MSR*n* bits
3–2	OUT2:1	Test Bits: In diagnostic mode (bit 4=1), these bits control the ring indicator (RI*n*) and data carrier detect (DCD*n*#) modem inputs. Setting OUT1 activates the internal RI*n* bit; clearing OUT1 deactivates the internal RI*n* bit. Setting OUT2 activates the internal DCD*n* bit; clear OUT2 deactivates the internal DCD*n* bit. In normal user mode (bit 4=0) OUT1 has no effect and OUT2 in conjunction with INTCFG.5/6 selects internal SIO interrupt or external interrupt. See Table 5-1 on page 5-8 for the configuration options.
1	RTS	Ready to Send: The function of this bit depends on whether the SIO*n* is in diagnostic mode (MCR*n*.4=1), internal connection mode, or standard mode. In diagnostic mode, setting this bit activates the internal CTS*n* bit; clearing this bit deactivates the internal CTS*n* bit. In internal connection mode, setting this bit activates the internal CTS*n*# signal and the RTS*n*# pin; clearing this bit deactivates the internal CTS*n*# signal and the RTS*n*# pin. In standard mode, setting this bit activates the RTS*n*# pin; clearing this bit deactivates the RTS*n*# pin. Note that pin is inverted from bit.
0	DTR	Data Terminal Ready: The function of this bit depends on whether the SIO*n* is in diagnostic mode (MCR*n*.4=1), internal connection mode, or standard mode. In diagnostic mode, setting this bit activates the internal DSR*n*# signal; clearing this bit deactivates the internal DSR*n*# signal. In internal connection mode, setting this bit activates the internal DSR*n*# and DCD*n*# signals and the DTR*n*# pin; clearing this bit deactivates the internal DSR*n*# and DCD*n*# signals and the DTR*n*# pin. Note that pin is inverted from bit. In standard mode, setting this bit activates the DTR*n*# pin; clearing this bit deactivates the DTR*n*# pin. Note that pin is inverted from bit.

D.36 MSR*n*

Modem Status
MSR0, MSR1
(read only)

	MSR0	MSR1
Expanded Addr:	**F4FEH**	**F8FEH**
ISA Addr:	**03FEH**	**02FEH**
Reset State:	**X0H**	**X0H**

7							0
DCD	RI	DSR	CTS	DDCD	TERI	DDSR	DCTS

Bit Number	Bit Mnemonic	Function
7	DCD	Data Carrier Detect: This bit is the complement of the data carrier detect (DCD*n*#) input. In diagnostic test mode, this bit is equivalent to MCR*n*.3 (OUT2).
6	RI	Ring Indicator: This bit is the complement of the ring indicator (RI*n*#) input. In diagnostic test mode, this bit is equivalent to MCR*n*.2 (OUT1).
5	DSR	Data Set Ready: This bit is the complement of the data set ready (DSR*n*#) input. In diagnostic test mode, this bit is equivalent to MCR*n*.0 (DTR).
4	CTS	Clear to Send: This bit is the complement of the clear to send (CTS*n*#) input. In diagnostic test mode, this bit is equivalent to MCR*n*.1 (RTS).
3	DDCD	Delta Data Carrier Detect: When set, this bit indicates that the DCD*n*# input has changed state since the last time this register was read. Reading this register clears this bit.
2	TERI	Trailing Edge Ring Indicator: When set, this bit indicates that the RI*n*# input has changed from a low to a high state since the last time this register was read. Reading this register clears this bit.
1	DDSR	Delta Data Set Ready: When set, this bit indicates that the DSR*n*# input has changed state since the last time this register was read. Reading this register clears this bit.
0	DCTS	Delta Clear to Send: When set, this bit indicates that the CTS*n*# input has changed state since the last time this register was read. Reading this register clears this bit.

D.37 OCW1 (MASTER AND SLAVE)

Operation Command Word 1
OCW1 (master and slave)
(read/write)

	master	slave
Expanded Addr:	**F021H**	**F0A1H**
ISA Addr:	**0021H**	**00A1H**
Reset State:	**XXH**	**XXH**

7 **0**

M7	M6	M5	M4	M3	M2	M1	M0

Bit Number	Bit Mnemonic	Function
7–0	M7:0	Mask IR: 0 = Enables interrupts on the corresponding IR signal. 1 = Disables interrupts on the corresponding IR signal. NOTE: Setting the mask bit does not clear the respective interrupt pending bit.

NOTE: The 8259A must be initialized before it can be used. After reset, the 8259A register states are undefined. The 8259A modules must be initialized before the IF flag in the core FLAG register is set.

D.38 OCW2 (MASTER AND SLAVE)

Operation Command Word 2
OCW2 (master and slave)
(write only)

	master	slave
Expanded Addr:	**F020H**	**F0A0H**
ISA Addr:	**0020H**	**00A0H**
Reset State:	**XXH**	**XXH**

7							0
R	SL	EOI	RSEL1	RSEL0	L2	L1	L0

Bit Number	Bit Mnemonic	Function
7	R	The Rotate (R), Specific Level (SL), and End-of-Interrupt (EOI) Bits: These bits change the priority structure and/or send an EOI command. **R SL EOI Command** 0 0 0 Cancel automatic rotation* 0 0 1 Send a nonspecific EOI command 0 1 0 No operation 0 1 1 Send a specific EOI command** 1 0 0 Enable automatic rotation* 1 0 1 Enable automatic rotation and send a nonspecific EOI 1 1 0 Initiate specific rotation** 1 1 1 Initiate specific rotation and send a specific EOI** * These cases allow you to change the priority structure while the 82C59A is operating in the automatic EOI mode. ** The L2:0 bits (see below) specify the specific level for these cases.
6	SL	
5	EOI	
4–3	RSEL1:0	Register Select Bits: ICW1, OCW2 and OCW3 are accessed through the same addresses. The states of RSEL1:0 determine which register is accessed. **Write 00 to these bits to access OCW2.** **RSEL1 RSEL0** 0 0 OCW2 0 1 OCW3 1 X ICW1
2–0	L2:0	IR Level: When you program bits 7–5 to initiate specific rotation, these bits specify the IR signal that is assigned the lowest level. When you program bits 7–5 to send a specific EOI command, these bits specify the IR signal that receives the EOI command. If SL=0, then these bits have no effect.

D.39 OCW3 (MASTER AND SLAVE)

Operation Command Word 3
OCW3 (master and slave)
(write only)

	master	slave
Expanded Addr:	**F020H**	**F0A0H**
ISA Addr:	**0020H**	**00A0H**
Reset State:	**XXH**	**XXH**

7 **0**

0	ESMM	SMM	RSEL1	RSEL0	POLL	ENRR	RDSEL

Bit Number	Bit Mnemonic	Function
7	—	Clear this bit to guarantee device operation.
6	ESMM	Enable Special Mask Mode (ESMM) and Special Mask Mode (SMM): Use these bits to enable or disable special mask mode. **ESMM SMM** 0 0 No action 0 1 No action 1 0 Disable special mask mode 1 1 Enable special mask mode
5	SMM	
4–3	RSEL1:0	Register Select: ICW1, OCW2 and OCW3 are accessed through the same addresses. The states of RSEL1:0 determine which register is accessed. **Write 01 to these bits to access OCW3.** **RSEL1 RSEL0** 0 0 OCW2 0 1 OCW3 1 X ICW1
2	POLL	Poll Command: Set this bit to issue a poll command, thus initiating the polling process.
1	ENRR	Enable Register Read Select (ENRR) and Read Register Select (RDSEL): These bits select which register is read during the next F020H and F0A0H (or PC/AT address 0020H, 00A0H) read access. **ENRR RDSEL Register Read on Next Read Pulse** 0 0 No action 0 1 No action 1 0 Interrupt Request Register 1 1 In-service Register
0	RDSEL	

D.40 P1CFG

Port 1 Configuration
P1CFG
(read/write)

Expanded Addr: **F820H**
ISA Addr: **—**
Reset State: **00H**

7							0
PM7	PM6	PM5	PM4	PM3	PM2	PM1	PM0

Bit Number	Bit Mnemonic	Function
7	PM7	Pin Mode: 0 = Selects P1.7 at the package pin. 1 = Selects HLDA at the package pin.
6	PM6	Pin Mode: 0 = Selects P1.6 at the package pin. 1 = Selects HOLD at the package pin.
5	PM5	Pin Mode: 0 = Selects P1.5 at the package pin. 1 = Selects LOCK# at the package pin.
4	PM4	Pin Mode: 0 = Selects P1.4 at the package pin. 1 = Selects RI0# at the package pin.
3	PM3	Pin Mode: 0 = Selects P1.3 at the package pin. 1 = Selects DSR0# at the package pin.
2	PM2	Pin Mode: 0 = Selects P1.2 at the package pin. 1 = Selects DTR0# at the package pin.
1	PM1	Pin Mode: 0 = Selects P1.1 at the package pin. 1 = Selects RTS0# at the package pin.
0	PM0	Pin Mode: 0 = Selects P1.0 at the package pin. 1 = Selects DCD0# at the package pin.

D.41 P2CFG

Port 2 Configuration
P2CFG
(read/write)

Expanded Addr: **F822H**
ISA Addr: **—**
Reset State: **00H**

7							0
PM7	PM6	PM5	PM4	PM3	PM2	PM1	PM0

Bit Number	Bit Mnemonic	Function
7	PM7	Pin Mode: 0 = Selects P2.7 at the package pin. 1 = Selects CTS0# at the package pin.
6	PM6	Pin Mode: 0 = Selects P2.6 at the package pin. 1 = Selects TXD0 at the package pin.
5	PM5	Pin Mode: 0 = Selects P2.5 at the package pin. 1 = Selects RXD0 at the package pin.
4	PM4	Pin Mode: 0 = Selects P2.4 at the package pin. 1 = Selects CS4# at the package pin.
3	PM3	Pin Mode: 0 = Selects P2.3 at the package pin. 1 = Selects CS3# at the package pin.
2	PM2	Pin Mode: 0 = Selects P2.2 at the package pin. 1 = Selects CS2# at the package pin.
1	PM1	Pin Mode: 0 = Selects P2.1 at the package pin. 1 = Selects CS1# at the package pin.
0	PM0	Pin Mode: 0 = Selects P2.0 at the package pin. 1 = Selects CS0# at the package pin.

D.42 P3CFG

Port 3 Configuration
P3CFG
(read/write)

Expanded Addr: **F824H**
ISA Addr: **—**
Reset State: **00H**

7							0
PM7	PM6	PM5	PM4	PM3	PM2	PM1	PM0

Bit Number	Bit Mnemonic	Function
7	PM7	Pin Mode: 0 = Selects P3.7 at the package pin. 1 = Selects COMCLK at the package pin.
6	PM6	Pin Mode: 0 = Selects P3.6 at the package pin. 1 = Selects PWRDOWN at the package pin.
5	PM5	Pin Mode: 0 = Selects P3.5 at the package pin. 1 = Connects master IR7 to the package pin (INT3).
4	PM4	Pin Mode: 0 = Selects P3.4 at the package pin. 1 = Connects master IR6 to the package pin (INT2).
3	PM3	Pin Mode: 0 = Selects P3.3 at the package pin. 1 = Connects master IR5 to the package pin (INT1).
2	PM2	Pin Mode: 0 = Selects P3.2 at the package pin. 1 = Connects master IR1 to the package pin (INT0).
1	PM1	Pin Mode: See Table 5-1 on page 5-8 for all the PM1 configuration options.
0	PM0	Pin Mode: See Table 5-1 on page 5-8 for all the PM0 configuration options.

D.43 PINCFG

Pin Configuration
PINCFG
(read/write)

Expanded Addr: **F826H**
ISA Addr: **—**
Reset State: **00H**

7							0
—	PM6	PM5	PM4	PM3	PM2	PM1	PM0

Bit Number	Bit Mnemonic	Function
7	—	Reserved. This bit is undefined; for compatibility with future devices, do not modify this bit.
6	PM6	Pin Mode: 0 = Selects CS6# at the package pin. 1 = Selects REFRESH# at the package pin.
5	PM5	Pin Mode: 0 = Selects the coprocessor signals, PEREQ, BUSY#, and ERROR#, at the package pins. 1 = Selects the timer control unit signals, TMROUT2, TMRCLK2, and TMRGATE2, at the package pins.
4	PM4	Pin Mode: 0 = Selects DACK0# at the package pin. 1 = Selects CS5# at the package pin.
3	PM3	Pin Mode: 0 = Selects EOP# at the package pin. 1 = Selects CTS1# at the package pin.
2	PM2	Pin Mode: 0 = Selects DACK1# at the package pin. 1 = Selects TXD1 at the package pin.
1	PM1	Pin Mode: 0 = Selects SRXCLK at the package pin. 1 = Selects DTR1# at the package pin.
0	PM0	Pin Mode: 0 = Selects SSIOTX at the package pin. 1 = Selects RTS1# at the package pin.

D.44 P*n*DIR

Port DIrection
P*n*DIR (*n*=1–3)
(read/write)

Expanded Addr: **F864H, F86CH, F874H**
ISA Addr: **—**
Reset State: **FFH**

7							0
PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0

Bit Number	Bit Mnemonic	Function
7–0	PD7:0	Pin Direction: 0 = Configures the pin as a complementary output. 1 = Configures the pin as an open-drain output or high-impedance input.

D.45 P*n*LTC

Port Data Latch
P*n*LTC (*n*=1–3)
(read/write)

Expanded Addr: **F862H, F86AH, F872H**
ISA Addr: **—**
Reset State: **FFH**

7							0
PL7	PL6	PL5	PL4	PL3	PL2	PL1	PL0

Bit Number	Bit Mnemonic	Function
7–0	PL7:0	Port Data Latch: Writing a value to a PL bit causes that value to be driven onto the corresponding pin. For a complementary output, write the desired pin value to its PL bit. This value is strongly driven onto the pin. For an open-drain output, a one results in a high-impedance (input) state at the pin. For a high-impedance input, write a one to the corresponding PL bit. A one results in a high-impedance state at the pin, allowing external hardware to drive it.

D.46 P*n*PIN

Port Pin State
P*n*PIN (*n*=1–3)
(read only)

Expanded Addr: **F860H, F868H, F870H**
ISA Addr: **—**
Reset State: **XXH**

7							0
PS7	PS6	PS5	PS4	PS3	PS2	PS1	PS0

Bit Number	Bit Mnemonic	Function
7–0	PS7:0	Pin State: Reading a PS bit returns the logic state present on the associated port pin.

D.47 POLL (MASTER AND SLAVE)

Poll Status Byte
POLL (master and slave)
(read only)

	master	slave
Expanded Addr:	**F020H**	**F0A0H**
ISA Addr:	**0020H**	**00A0H**
Reset State:	**XXH**	**XXH**

7							0
INT	—	—	—	—	L2	L1	L0

Bit Number	Bit Mnemonic	Function
7	INT	Interrupt Pending: 0 = No request pending. 1 = Indicates that a device attached to the 82C59A requires servicing.
6–3	—	Reserved. These bits are undefined.
2–0	L2:0	Interrupt Request Level: When bit 7 is set, these bits indicate the highest-priority IR signal that requires servicing. When bit 7 is clear, i.e., no request is pending, these bits are indeterminate.

D.48 PORT92

Port 92 Configuration
PORT92
(read/write)

Expanded Addr: **F092H**
ISA Addr: **0092H**
Reset State: **XXXXXX10B**

7							0
—	—	—	—	—	—	A20G	CPURST

Bit Number	Bit Mnemonic	Function
7–2	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
1	A20G	A20 Grounded: 0 = Clearing this bit forces address line A20 to 0. This bit affects addresses generated only by the core. Addresses generated by the DMA and the Refresh Unit are not affected by this bit. 1 = Setting this bit leaves core-generated addresses unmodified.
0	CPURST	CPU Reset: 0 = Clearing this bit performs no operation. 1 = Setting this bit resets the core without resetting the peripherals. This bit must be cleared before issuing another reset.

D.49 PWRCON

Power Control Register
PWRCON
(read/write)

Expanded Addr: **F800H**
ISA Addr: **—**
Reset State: **00H**

7							0
—	—	—	—	WDTRDY	HSREADY	PC1	PC0

Bit Number	Bit Mnemonic	Function
7–4	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
3	WDTRDY	Watch Dog Timer Ready: 0 = An external READY must be generated to terminate the cycle when the WDT times out in Bus Monitor Mode. 1 = Internal logic generates READY# to terminate the cycle when the WDT times out in Bus Monitor Mode.
2	HSREADY	Halt/Shutdown Ready: 0 = An external ready must be generated to terminate a HALT/Shutdown cycle. 1 = Internal logic generates READY# to terminate a HALT/Shutdown cycle.
1–0	PC1:0	Power Control: Program these bits, then execute a HALT instruction. The device enters the programmed mode when READY# (internal or external) terminates the halt bus cycle. When these bits have equal values, the HALT instruction causes a normal halt and the device remains in active mode. **PC1 PC0** 0 0 active mode 1 0 idle mode 0 1 powerdown mode 1 1 active mode

D.50 RBR*n*

Receive Buffer
RBR0, RBR1
(read only)

	RBR0	RBR1
Expanded Addr:	**F4F8H**	**F8F8H**
ISA Addr:	**03F8H**	**02F8H**
Reset State:	**XXH**	**XXH**

7							0
RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0

Bit Number	Bit Mnemonic	Function
7–0	RB7:0	Receive Buffer Bits: These bits make up the last word received. The receiver shifts bits in, starting with the least-significant-bit. The receiver then strips off the asynchronous bits (start, parity, and stop) and transfers the received data bits from the receive shift register to the receive buffer.

NOTE: The receive buffer register shares an address port with other SIO registers. Bit 7 (DLAB) of the LCR*n* must be cleared in order to read the receive buffer register.

D.51 REMAPCFG

Address Configuration Register
REMAPCFG

Expanded Addr: 0022H
PC/AT Address: 0022H
Reset State: 0000H

15							8
ESE	—	—	—	—	—	—	—

7							0
—	S1R	S0R	ISR	IMR	DR	—	TR

Bit Number	Bit Mnemonic	Function
15	ESE	0 = Disables expanded I/O space 1 = Enables expanded I/O space
14–7	—	Reserved.
6	S1R	0 = Makes serial channel 1 (COM2) accessible in both DOS I/O space and expanded I/O space 1 = Remaps serial channel 1 (COM2) address into expanded I/O space
5	S0R	0 = Makes serial channel 0 (COM1) accessible in both DOS I/O space and expanded I/O space 1 = Remaps serial channel 0 (COM1) address into expanded I/O space
4	ISR	0 = Makes the slave 82C59A interrupt controller accessible in both DOS I/O space and expanded I/O space 1 = Remaps slave 82C59A interrupt controller address into expanded I/O space
3	IMR	0 = Makes the master 82C59A interrupt controller accessible in both DOS I/O space and expanded I/O space 1 = Remaps master 82C59A interrupt controller address into expanded I/O space
2	DR	0 = Makes the DMA address accessible in both DOS I/O space and expanded I/O space 1 = Remaps DMA address into expanded I/O space
1	—	Reserved.
0	TR	0 = Makes the timer control unit accessible in both DOS I/O space and expanded I/O space 1 = Remaps timer control unit address into expanded I/O space

D.52 RFSADD

Refresh Address
RFSADD
(read/write)

Expanded Addr: **F4A6H**
ISA Addr: **—**
Reset State: **00FFH**

15							**8**
—	—	RA13	RA12	RA11	RA10	RA9	RA8
7							**0**
RA7	RA6	RA5	RA4	RA3	RA2	RA1	1

Bit Number	Bit Mnemonic	Function
15–14	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
13–1	RA13:1	Refresh Address Bits: These bits comprise A13:1 of the refresh address.
0	—	Refresh Bit 0: A0 of the refresh address. This bit is always 1 and is read-only.

D.53 RFSBAD

Refresh Base Address
RFSBAD
(read/write)

Expanded Addr: **F4A0H**
ISA Addr: **—**
Reset State: **0000H**

15							**8**
—	—	—	—	RA25	RA24	RA23	RA22
7							**0**
RA21	RA20	RA19	RA18	RA17	RA16	RA15	RA14

Bit Number	Bit Mnemonic	Function
15–12	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
11–0	RA25:14	Refresh Base: These bits make up the A25:14 address bits of the refresh address. This establishes a memory region for refreshing.

D.54 RFSCIR

Refresh Clock Interval
RFSCIR
(read/write)

Expanded Addr: **F4A2H**
ISA Addr: **—**
Reset State: **0000H**

15							8
—	—	—	—	—	—	RC9	RC8
7							**0**
RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0

Bit Number	Bit Mnemonic	Function
15–10	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
9–0	RC9:0	Refresh Counter Value: Write the counter value to these ten bits. The interval counter counts down from this value. When the interval counter reaches one, the control unit initiates a refresh request (provided it does not have a request pending). The counter value is a function of DRAM specifications and processor frequency (see the equation above).

D.55 RFSCON

Refresh Control
RFSCON
(read/write)

Expanded Addr: **F4A4H**
ISA Addr: **—**
Reset State: **0000H**

15							8
REN	—	—	—	—	—	CV9	CV8
7							**0**
CV7	CV6	CV5	CV4	CV3	CV2	CV1	CV0

Bit Number	Bit Mnemonic	Function
15	REN	Refresh Control Unit Enable: This bit enables or disables the refresh control unit. 0 = Disables refresh control unit 1 = Enables refresh control unit
14–10	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
9–0	CV9:0	Counter Value: These read-only bits represent the current value of the interval counter. Write operations to these bits have no effect.

D.56 SCR*n*

Scratch Pad
SCR0, SCR1
(read/write)

	SCR0	SCR1
Expanded Addr:	**F4FFH**	**F8FFH**
ISA Addr:	**03FFH**	**02FFH**
Reset State:	**XXH**	**XXH**

7							0
SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0

Bit Number	Bit Mnemonic	Function
7–0	SP7:0	Writing and reading this register has no effect on SIO*n* operation.

D.57 SIOCFG

SIO and SSIO Configuration
SIOCFG
(read/write)

Expanded Addr: **F836H**
ISA Addr: **—**
Reset State: **00H**

7							0
S1M	S0M	—	—	—	SSBSRC	S1BSRC	S0BSRC

Bit Number	Bit Mnemonic	Function
7	S1M	SIO1 Modem Signal Connections: 0 = Connects the SIO1 modem input signals to the package pins. 1 = Connects the SIO1 modem input signals internally.
6	S0M	SIO0 Modem Signal Connections: 0 = Connects the SIO0 modem input signals to the package pins. 1 = Connects the SIO0 modem input signals internally.
5–3	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
2	SSBSRC	SSIO Baud-rate Generator Clock Source: 0 = Connects the internal PSCLK signal to the SSIO baud-rate generator. 1 = Connects the internal SERCLK signal to the SSIO baud-rate generator.
1	S1BSRC	SIO1 Baud-rate Generator Clock Source: 0 = Connects the COMCLK pin to the SIO1 baud-rate generator. 1 = Connects the internal SERCLK signal to the SIO1 baud-rate generator.
0	S0BSRC	SIO0 Baud-rate Generator Clock Source: 0 = Connects the COMCLK pin to the SIO0 baud-rate generator. 1 = Connects the internal SERCLK signal to the SIO0 baud-rate generator.

D.58 SSIOBAUD

SSIO Baud-rate Control
SSIOBAUD
(read/write)

Expanded Addr: **F484H**
ISA Addr: **—**
Reset State: **00H**

7							0
BEN	BV6	BV5	BV4	BV3	BV2	BV1	BV0

Bit Number	Bit Mnemonic	Function
7	BEN	Baud-rate Generator Enable: Setting this bit enables the baud-rate generator. Clearing this bit disables the baud-rate generator, clears the baud-rate count value, and forces the baud rate clock to zero.
6–0	BV6:0	Baud-rate Value: The baud-rate value (BV) is the reload value for the baud-rate generator's seven-bit down counter. The baud-rate generator's output is a function of BV and the baud-rate generator's input (BCLKIN), as follows. $\text{baud-rate output frequency (Hz)} = \frac{\text{BCLKIN}}{2\text{BV} + 2}$ (Hz) If you know the desired output baud-rate frequency, you can determine BV as follows. $\text{BV} = \left(\frac{\text{BCLKIN}}{2 \times \text{baud-rate output frequency}}\right) - 1$

D.59 SSIOCON1

SSIO Control 1
SSIOCON1
(read/write)

Expanded Addr: **F486H**
ISA Addr: **—**
Reset State: **C0H**

7							0
TUE	THBE	TIE	TEN	ROE	RHBF	RIE	REN

Bit Number	Bit Mnemonic	Function
7	TUE	Transmit Underrun Error: The transmitter sets this bit to indicate a transmit underrun error in the TEN transfer mode. Clear this bit to clear the error flag. If a one is written to TUE, it is ignored and TUE retains its previous value.
6	THBE (read only bit)	Transmit Holding Buffer Empty: The transmitter sets this bit when the transmit buffer contents have been transferred to the transmit shift register, indicating that the buffer is now ready to accept new data. Writing data to the transmit buffer clears THBE. When this bit is clear, the buffer is not ready to accept any new data.
5	TIE	Transmitter Interrupt Enable: 0 = Clearing this bit prevents the Interrupt Control Unit from sensing when the transmit buffer is empty. 1 = Setting this bit connects the transmit buffer empty internal signal to the Interrupt Control Unit.
4	TEN	Transmitter Enable: 0 = Disables the transmitter. 1 = Enables the transmitter.
3	ROE	Receive Overflow Error: The receiver sets this bit to indicate a receiver overflow error. Write zero to this bit to clear the flag. If a one is written to ROE, the one is ignored and ROE retains its previous value.
2	RHBF (read only bit)	Receive Holding Buffer Full: The receiver sets this bit when the receive shift register contents have been transferred to the receive buffer. Reading the buffer clears this bit.
1	RIE	Receive Interrupt Enable: 0 = Clearing this bit prevents the Interrupt Control Unit from sensing when the receive buffer is full. 1 = Setting this bit connects the receiver buffer full internal signal to the Interrupt Control Unit.
0	REN	Receiver Enable: 0 = Clearing this bit disables the receiver. 1 = Setting this bit enables the receiver.

D.60 SSIOCON2

SSIO Control 2
SSIOCON2
(read/write)

Expanded Addr: **F488H**
ISA Addr: **—**
Reset State: **00H**

7							0
—	—	—	—	—	AUTOTXM	TXMM	RXMM

Bit Number	Bit Mnemonic	Function
7–3	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
2	AUTOTXM	Automatic Transmit off mode for master mode 0 = Clearing this bit puts the TEN bit into normal operation 1 = Setting this bit and the TXMM bit causes TEN to be ignored. Every time a word is loaded into the transmit shift register from the transmit holding buffer it is transmitted out and then stops.
1	TXMM	Transmit Master Mode: 0 = Clearing this bit puts the transmitter in slave mode. In slave mode, an external device controls the transmit serial communications. An input on the STXCLK pin clocks the transmitter. 1 = Setting this bit puts the transmitter in master mode. In master mode, the internal baud-rate generator controls the transmit serial communications. The baud-rate generator's output clocks the internal transmitter and appears on the STXCLK pin.
0	RXMM	Receive Master Mode: 0 = Clearing this bit puts the receiver in slave mode. In slave mode, an external device controls the receive serial communications. An input on the SRXCLK pin clocks the receiver. 1 = Setting this bit puts the receiver in master mode. In master mode, the internal baud-rate generator controls the receive serial communications. The baud-rate generator's output clocks the internal receiver and appears on the SRXCLK pin.

D.61 SSIOCTR

Baud-rate Count Down
SSIOCTR
(read only)

Expanded Addr: **F48AH**
ISA Addr: **—**
Reset State: **00H**

7							0
BSTAT	CV6	CV5	CV4	CV3	CV2	CV1	CV0

Bit Number	Bit Mnemonic	Function
7	BSTAT	Baud-rate Generator Status: 0 = The baud-rate generator is disabled. 1 = The baud-rate generator is enabled.
6–0	CV6:0	Current Value: These bits indicate the current value of the baud-rate down counter.

D.62 SSIORBUF

Receive Holding Buffer
SSIORBUF
(read only)

Expanded Addr: **F482H**
ISA Addr: **—**
Reset State: **0000H**

15							8
RB15	RB14	RB13	RB12	RB11	RB10	RB9	RB8
7							**0**
RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0

Bit Number	Bit Mnemonic	Function
15–0	RB15:0	Receive Buffer Bits: This register contains the last word received. The receive shift register shifts bits in with the rising edge of the receiver clock pin. Data is shifted in starting with the most-significant bit. The control logic then transfers the received word from the receive shift register to SSIORBUF.

D.63 SSIOTBUF

Transmit Holding Buffer
SSIOTBUF
(read/write)

Expanded Addr: **F480H**
ISA Addr: **—**
Reset State: **0000H**

15							**8**
TB15	TB14	TB13	TB12	TB11	TB10	TB9	TB8
7							**0**
TB7	TB6	TB5	TB4	TB3	TB2	TB1	TB0

Bit Number	Bit Mnemonic	Function
15–0	TB15:0	Transmit Buffer Bits: These bits make up the next data word to be transmitted. The control logic loads this word into the transmit shift register. The transmit shift register shifts the bits out on the falling edge of the tranmitter clock pin. The word is transmitted out starting with the most-significant bit (TB15).

D.64 TBR*n*

Transmit Buffer
TBR0, TBR1
(write only)

	TBR0	**TBR1**
Expanded Addr:	**F4F8H**	**F8F8H**
ISA Addr:	**03F8H**	**02F8H**
Reset State:	**XXH**	**XXH**

7							**0**
TB7	TB6	TB5	TB4	TB3	TB2	TB1	TB0

Bit Number	Bit Mnemonic	Function
7–0	TB7:0	Transmit Buffer Bits: These bits make up the next data word to be transmitted. The transmitter loads this word into the transmit shift register. The transmit shift register then shifts the bits out, along with the asynchronous communication bits (start, stop, and parity). The data bits are shifted out least-significant bit (TB0) first.

NOTE: The transmit buffer register shares an address port with other SIO registers. You must clear bit 7 (DLAB) of LCR*n* before you can write to the transmit buffer register.

D.65 TMRCFG

Timer Configuration
TMRCFG
(read/write)

Expanded Addr: **F834H**
ISA Addr: **—**
Reset State: **00H**

7							0
TMRDIS	SWGTEN	GT2CON	CK2CON	GT1CON	CK1CON	GT0CON	CK0CON

Bit Number	Bit Mnemonic	Function
7	TMRDIS	Timer Disable: 0 = Enables the CLKIN*n* signals. 1 = Disables the CLKIN*n* signals.
6	SWGTEN	Software GATE*n* Enable 0 = Connects GATE*n* to either the V_{CC} pin or the TMRGATE*n* pin. 1 = Enables GT2CON, GT1CON, and GT0CON to control the connections to GATE2, GATE1 and GATE0 respectively.
5	GT2CON	Gate 2 Connection: **SWGTEN GT2CON** 0 0 Connects GATE2 to V_{CC}. 0 1 Connects GATE2 to the TMRGATE2 pin. 1 0 Turns GATE2 off. 1 1 Turns GATE2 on.
4	CK2CON	Clock 2 Connection: 0 = Connects CLKIN2 to the internal PSCLK signal. 1 = Connects CLKIN2 to the TMRCLK2 pin.
3	GT1CON	Gate 1 Connection: **SWGTEN GT1CON** 0 0 Connects GATE1 to V_{CC}. 0 1 Connects GATE1 to the TMRGATE1 pin. 1 0 Turns GATE1 off. 1 1 Turns GATE1 on.
2	CK1CON	Clock 1 Connection: 0 = Connects CLKIN1 to the internal PSCLK signal. 1 = Connects CLKIN1 to the TMRCLK1 pin.
1	GT0CON	Gate 0 Connection: **SWGTEN GT0CON** 0 0 Connects GATE0 to V_{CC}. 0 1 Connects GATE0 to the TMRGATE1 pin. 1 0 Turns GATE0 off. 1 1 Turns GATE0 on.
0	CK0CON	Clock 0 Connection: 0 = Connects CLKIN0 to the internal PSCLK signal. 1 = Connects CLKIN0 to the TMRCLK0 pin.

D.66 TMRCON

Timer Control (Control Word Format)
TMRCON

Expanded Addr: **F043H**
ISA Addr: **0043H**
Reset State: **XXH**

7							0
SC1	SC0	RW1	RW0	M2	M1	M0	CNTFMT

Bit Number	Bit Mnemonic	Function
7–6	SC1:0	Select Counter: Use these bits to specify a particular counter. The selections you make for bits 5–0 define this counter's operation. 00 = counter 0 01 = counter 1 10 = counter 2 11 is not an option for TMRCON's control word format. Selecting 11 accesses TMRCON's read-back format, which is shown in Figure 10-29.
5–4	RW1:0	Read/Write Select: These bits select a read/write option for the counter specified by bits 7–6. 01 = read/write least-significant byte only 10 = read/write most-significant byte only 11 = read/write least-significant byte first, then most-significant byte 00 is not an option for TMRCON's control word format. Selecting 00 accesses TMRCON's counter-latch format, which is shown in Figure 10-27.
3–1	M2:0	Mode Select: These bits select an operating mode for the counter specified by bits 7–6. 000 = mode 0 001 = mode 1 X10 = mode 2 X11 = mode 3 100 = mode 4 101 = mode 5 X is a don't care.
0	CNTFMT	Count Format: This bit selects the count format for the counter specified by bits 7–6. 0 = binary (16 bits) 1 = binary coded decimal (4 decades)

D.67 TMR*n*

Timer *n* (Read Format)
TMR*n* (*n* = 0–2)

Expanded Addr: **F040H, F041H F042H**
ISA Addr: **0040H, 0041H 0042H**
Reset State: **XXH**

7							0
CV7	CV6	CV5	CV4	CV3	CV2	CV1	CV0

Bit Number	Bit Mnemonic	Function
7–0	CV7:0	Count Value: These bits contain the counter's count value. When reading the counter's count value, follow the read selection specified in the counter's control word.

Timer *n* (Status Format)
TMR*n* (*n* = 0–2)

Expanded Addr: **F040H, F041H F042H**
ISA Addr: **0040H, 0041H 0042H**
Reset State: **XXH**

7 **0**

OUTPUT	NULCNT	RW1	RW0	M2	M1	M0	CNTFMT

Bit Number	Bit Mnemonic	Function
7	OUTPUT	Output Status: This bit indicates the current state of the counter's output signal. 0 = OUT*n* is low 1 = OUT*n* is high
6	NULCNT	Count Status: This bit indicates whether the latest count written to the counter has been loaded. Some modes require a gate-trigger before the counter loads new count values. 0 = the latest count written to the counter has been loaded 1 = a count has been written to the counter but has not yet been loaded
5–4	RW1:0	Read/Write Select Status: These bits indicate the counter's programmed read/write selection. 00 = Never occurs 01 = read/write least-significant byte only 10 = read/write most-significant byte only 11 = read/write least-significant byte first, then most-significant byte
3–1	M2:0	Mode Status: These bits indicate the counter's programmed operating mode. 000 = mode 0 001 = mode 1 X10 = mode 2 X11 = mode 3 100 = mode 4 101 = mode 5 X is a don't care.
0	CNTFMT	Counter Format Status: This bit indicates the counter's programmed count format. 0 = binary (16 bits) 1 = binary coded decimal (4 decades)

D.68 UCSADH

See “CSnADH (UCSADH)” on page D-8.

D.69 UCSADL

See “CSnADL (UCSADL)” on page D-9.

D.70 UCSMSKH

See “CSnMSKH (UCSMSKH)” on page D-10.

D.71 UCSMSKL

See “CSnMSKL (UCSMSKL)” on page D-11.

D.72 WDTCNTH AND WDTCNTL

WDT Counter Value (High)
WDTCNTH
(read only)

Expanded Addr: **F4C4H**
ISA Addr: **—**
Reset State: **003FH**

15							8
WC31	WC30	WC29	WC28	WC27	WC26	WC25	WC24
7							**0**
WC23	WC22	WC21	WC20	WC19	WC18	WC17	WC16

WDT Counter Value (Low)
WDTCNTL
(read only)

Expanded Addr: **F4C6H**
ISA Addr: **—**
Reset State: **FFFFH**

15							8
WC15	WC14	WC13	WC12	WC11	WC10	WC9	WC8
7							**0**
WC7	WC6	WC5	WC4	WC3	WC2	WC1	WC0

Bit Number	Bit Mnemonic	Function
High 15–0	WC31:16	WDT Counter Value High Word and Low Word:
Low 15–0	WC15:0	Read the high word of the counter value from WDTCNTH and the low word from WDTCNTL.

D.73 WDTRLDH AND WDTRLDL

WDT Reload Value (High)
WDTRLDH
(read/write)

Expanded Addr: **F4C0H**
ISA Addr: **—**
Reset State: **003FH**

15							8
WR31	WR30	WR29	WR28	WR27	WR26	WR25	WR24
7							**0**
WR23	WR22	WR21	WR20	WR19	WR18	WR17	WR16

WDT Reload Value (Low)
WDTRLDL
(read/write)

Expanded Addr: **F4C2H**
ISA Addr: **—**
Reset State: **FFFFH**

15							8
WR15	WR14	WR13	WR12	WR11	WR10	WR9	WR8
7							**0**
WR7	WR6	WR5	WR4	WR3	WR2	WR1	WR0

Bit Number	Bit Mnemonic	Function
High 15–0	WR31:16	WDT Reload Value (High Word and Low Word):
Low 15–0	WR15:0	Write the high word of the reload value to WDTRLDH and the low word to the WDTRLDL.

D.74 WDTSTATUS

WDT Status
WDTSTATUS
(read/write)

Expanded Addr: **F4CAH**
ISA Addr: **—**
Reset State: **00H**

7							0
WDTEN	—	—	—	—	—	BUSMON	CLKDIS

Bit Number	Bit Mnemonic	Function
7	WDTEN	Watchdog Mode Enabled: This **read-only** bit indicates whether watchdog mode is enabled. Only a lockout sequence can set this bit and only a device reset can clear it. 0 = Watchdog mode disabled 1 = Watchdog mode enabled
6–2	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.
1	BUSMON	Bus Monitor Enable: 0 = Disables bus monitor mode 1 = Enables bus monitor mode Read this bit to determine the current status. A lockout sequence clears BUSMON and prevents writes to the WDTSTATUS register.
0	CLKDIS	Clock Disable: Write to this bit to stop or restart the clock to the WDT; read it to determine the current clock status. A lockout sequence clears CLKDIS and prevents writing to this register. 0 = Clock enabled 1 = Processor clock (frequency=CLK2/2) disabled (stopped)

E

INSTRUCTION SET SUMMARY

APPENDIX E
INSTRUCTION SET SUMMARY

This appendix provides reference information for the Intel386™ processor family instruction set.

The appendix is organized as follows:

- Instruction Encoding and Clock Count Summary (see below)
- Instruction Encoding (page E-22)

E.1 INSTRUCTION ENCODING AND CLOCK COUNT SUMMARY

To calculate elapsed time for an instruction, multiply the instruction clock count, as listed in Table E-1, by the processor clock period (e.g., 62.5 ns for 16 MHz).

Instruction clock count assumptions:

- The instruction has been prefetched, decoded, and is ready for execution.
- Bus cycles do not require wait states.
- There are no local bus HOLD requests delaying processor access to the bus.
- No exceptions are detected during instruction execution.
- When an effective address is calculated, it does not use two general register components. One register, scaling and displacement can be used within the clock counts shown. However, when the effective address calculation uses two general register components, add 1 clock to the clock count shown.

Instruction clock count notation:

- When two clock counts are given, the smaller refers to a register operand and the larger refers to a memory operand.
- n = number of times repeated.
- m = number of components in the next instruction executed, where the entire displacement (if any) counts as one component, the entire immediate data (if any) counts as one component, and all other bytes of the instruction and prefix(es) of each count as one component.

Misaligned or 32-bit operand accesses:

- When instructions access a misaligned 16-bit operand or 32-bit operand on even address:
  - — add 2* clocks for read or write
  - — add 4** clocks for read and write
- When instructions access a 32-bit operand on odd address:
  - — add 4* clocks for read or write
  - — add 8** clocks for read and write

Wait states:

Wait states add 1 clock per wait state to instruction execution for each data access.

Table E-1 lists the instructions with their formats and execution times. The description of the notes for Table E-1 begins on page E-20. See "Instruction Encoding" on page E-22 for the definition of the terms used in this table.

Table E-1. Instruction Set Summary (Sheet 1 of 19)

Instruction	Format	Clock Count		Notes	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
GENERAL DATA TRANSFER					
MOV = Move					
register to register/memory	1 0 0 0 1 0 0 w | mod reg r/m	2/2	2/2*	b	h
register/memory to register	1 0 0 0 1 0 1 w | mod reg r/m	2/4	2/4*	b	h
immediate to register/memory	1 1 0 0 0 1 1 w | mod 0 0 0 r/m | immediate data	2/2	2/2*	b	h
immediate to register (short form)	1 0 1 1 w reg | immediate data	2	2		
memory to accumulator (short form)	1 0 1 0 0 0 0 w | full displacement	4*	4*	b	h
accumulator to memory (short form)	1 0 1 0 0 0 1 w | full displacement	2*	2*	b	h
register memory to segment register	1 0 0 0 1 1 1 0 | mod sreg3 r/m	2/5	22/23	b	h, i, j
segment register to register/memory	1 0 0 0 1 1 0 0 | mod sreg3 r/m	2/2	2/2	b	h
MOVSX = Move with sign extension					
register from register/memory	0 0 0 0 1 1 1 1 | 1 0 1 1 1 1 1 w | mod reg r/m	3/6*	3/6*	b	h
MOVZX = Move with zero extension					
register from register/memory	0 0 0 0 1 1 1 1 | 1 0 1 1 0 1 1 w | mod reg r/m	3/6*	3/6*	b	h
PUSH = Push					
register/memory	1 1 1 1 1 1 1 1 | mod 1 1 0 r/m	5/7*	7/9*	b	h
register (short form)	0 1 0 1 0 reg	2	4	b	h
segment register (ES, CS, SS, or DS) (short form)	000 sreg2 110	2	4	b	h
segment register (ES, CS, SS, or DS, FS or GS)	0 0 0 0 1 1 1 1 | 10 sreg3 000	2	4	b	h

Table E-1. Instruction Set Summary (Sheet 2 of 19)

Instruction	Format			Clock Count		Notes	
				Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
immediate	0 1 1 0 1 0 s 0	immediate data		2	4	b	h
PUSHA = Push All	0 1 1 0 0 0 0 0			18	34	b	h
POP = Pop							
register/memory	1 0 0 0 1 1 1 1	mod 0 0 0 r/m		5/7	7/9	b	h
register (short form)	0 1 0 1 1 reg			6	6	b	h
segment register (ES, CS, SS, or DS) (short form)	000 sreg2 111			7	25	b	h, i, j
segment register (ES, CS, SS, or DS) FS or GS	0 0 0 0 1 1 1 1	10 sreg3 001		7	25	b	h, i, j
POPA = Pop all	0 1 1 0 0 0 0 1			29	35	b	h
XCHG = Exchange							
register/memory with register	1 0 0 0 0 1 1 w	mod reg r/m		3/5**	3/5**	b, f	f, h
register with accumulator (short form)	1 0 0 1 0 reg			3	3		
IN = Input from			Clk Count Virtual 8086 Mode				
fixed port	1 1 1 0 0 1 0 w	port number	†27	14*	8*/29*		s/t, m
variable port	1 1 1 0 1 1 0 w		†28	15*	9*/29*		s/t, m
OUT = Output to							
fixed port	1 1 1 0 0 1 1 w	port number	†27	14*	8*/29*		s/t, m
variable port	1 1 1 0 1 1 1 w		†28	15*	9*/29*		s/t, m
LEA = Load EA to register	1 0 0 0 1 1 0 1	mod reg r/m		2	2		
SEGMENT CONTROL							
LDS = Load pointer to DS	1 1 0 0 0 1 0 1	mod reg r/m		7*	26*/28*	b	h, i, j
LES = Load pointer to ES	1 1 0 0 0 1 0 0	mod reg r/m		7*	26*/28*	b	h, i, j
LFS = Load pointer to FS	0 0 0 0 1 1 1 1	1 0 1 1 0 1 0 0	mod reg r/m	7*	29*/31*	b	h, i, j
LGS = Load pointer to GS	0 0 0 0 1 1 1 1	1 0 1 1 0 1 0 1	mod reg r/m	7*	26*/28*	b	h, i, j
LSS = Load pointer to SS	0 0 0 0 1 1 1 1	1 0 1 1 0 0 1 0	mod reg r/m	7*	26*/28*	b	h, i, j
FLAG CONTROL							
CLC = Clear carry flag	1 1 1 1 1 0 0 0			2	2		

Table E-1. Instruction Set Summary (Sheet 3 of 19)

Instruction	Format	Clock Count: Real Address Mode or Virtual 8086 Mode	Clock Count: Protected Virtual Address Mode	Notes: Real Address Mode or Virtual 8086 Mode	Notes: Protected Virtual Address Mode
CLD = Clear direction flag	1 1 1 1 1 1 0 0	2	2		
CLI = Clear interrupt enable flag	1 1 1 1 1 0 1 0	8	8		m
CLTS = Clear task switched flag	0 0 0 0 1 1 1 1 | 0 0 0 0 0 1 1 0	5	5	c	l
CMC = Complement carry flag	1 1 1 1 0 1 0 1	2	2		
LAHF = Load AH into flag	1 0 0 1 1 1 1 1	2	2		
POPF = Pop flags	1 0 0 1 1 1 0 1	5	5	b	h, n
PUSHF = Push flags	1 0 0 1 1 1 0 0	4	4	b	h
SAHF = Store AH into flags	1 0 0 1 1 1 1 0	3	3		
STC = Set carry flag	1 1 1 1 1 0 0 1	2	2		
STD = Set direction flag	1 1 1 1 1 1 0 1				
STI = Set interrupt enable flag	1 1 1 1 1 0 1 1	8	8		m
ARITHMETIC INSTRUCTIONS					
ADD = Add					
register to register	0 0 0 0 0 0 d w | mod reg r/m	2	2		
register to memory	0 0 0 0 0 0 0 w | mod reg r/m	7**	7**	b	h
memory to register	0 0 0 0 0 0 1 w | mod reg r/m	6*	6*	b	h
immediate to register/memory	1 0 0 0 0 0 s w | mod 0 0 0 r/m | immediate data	2/7**	2/7**	b	h
immediate to accumulator (short form)	0 0 0 0 0 1 0 w | immediate data	2	2		
ADC = Add with carry					
register to register	0 0 0 1 0 0 d w | mod reg r/m	2	2		
register to memory	0 0 0 1 0 0 0 w | mod reg r/m	7**	7**	b	h
memory to register	0 0 0 1 0 0 1 w | mod reg r/m	6*	6*	b	h
immediate to register/memory	1 0 0 0 0 0 s w | mod 0 1 0 r/m | immediate data	2/7**	2/7**	b	h
immediate to accumulator (short form)	0 0 0 1 0 1 0 w | immediate data	2	2		
INC = Increment					
register/memory	1 1 1 1 1 1 1 w | mod 0 0 0 r/m	2/6**	2/6**	b	h
register (short form)	0 1 0 0 0 reg	2	2		

Table E-1. Instruction Set Summary (Sheet 4 of 19)

Instruction	Format	Clock Count		Notes	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
SUB = Subtract					
register from register	0 0 1 0 1 0 d w | mod reg r/m	2	2		
register from memory	0 0 1 0 1 0 0 w | mod reg r/m	7**	7**	b	h
memory from register	0 0 1 0 1 0 1 w | mod reg r/m	6*	6*	b	h
immediate from register/memory	1 0 0 0 0 0 s w | mod 101 r/m | immediate data	2/7**	2/7**	b	h
immediate from accumulator (short form)	0 0 1 0 1 1 0 w | immediate data	2	2		
SBB = Subtract with borrow					
register from register	0 0 0 1 1 0 d w | mod reg r/m	2	2		
register from memory	0 0 0 1 1 0 0 w | mod reg r/m	7**	7**	b	h
memory from register	0 0 0 1 1 0 1 w | mod reg r/m	6*	6*	b	h
immediate from register/memory	1 0 0 0 0 0 s w | mod 0 1 1 r/m | immediate data	2/7**	2/7**	b	h
immediate from accumulator (short form)	0 0 0 1 1 1 0 w | immediate data	2	2		
DEC = Decrement					
register/memory	1 1 1 1 1 1 1 w | reg 0 0 1 r/m	2/6	2/6	b	h
register (short form)	0 1 0 0 1 reg	2	2		
CMP = Compare					
register with register	0 0 1 1 1 0 d w | mod reg r/m	2	2		
memory with register	0 0 1 1 1 0 0 w | mod reg r/m	5*	5*	b	h
register with memory	0 0 1 1 1 0 1 w | mod reg r/m	6*	6*	b	h
immediate with register/memory	1 0 0 0 0 0 s w | mod 1 1 1 r/m | immediate data	2/5*	2/5*	b	h
immediate with accumulator (short form)	0 0 1 1 1 1 0 w | immediate data	2	2		
NEG = Change sign	1 1 1 1 0 1 1 w | mod 0 1 1 r/m	2/6*	2/6*	b	h
AAA = ASCII adjust for addition	0 0 1 1 0 1 1 1	4	4		
AAS = ASCII adjust for subtraction	0 0 1 1 1 1 1 1	4	4		
DAA = Decimal adjust for addition	0 0 1 0 0 1 1 1	4	4		
DAS = Decimal adjust for subtraction	0 0 1 0 1 1 1 1	4	4		

Table E-1. Instruction Set Summary (Sheet 5 of 19)

Instruction	Format	Clock Count: Real Address Mode or Virtual 8086 Mode	Clock Count: Protected Virtual Address Mode	Notes: Real Address Mode or Virtual 8086 Mode	Notes: Protected Virtual Address Mode
MUL = multiply (unsigned)					
accumulator with register/memory	1 1 1 1 0 1 1 w | mod 1 0 0 r/m				
multiplier					
— byte		12-17/ 15-20*	12-17/ 15-20*	b, d	d, h
— word		12-25/ 15-28*	12-25/ 15-28*	b, d	d, h
— doubleword		12-41/ 17-46*	12-41/ 17-46*	b, d	d, h
IMUL = Integer multiply (signed)					
accumulator with register/memory	1 1 1 1 0 1 1 w | mod 1 0 1 r/m				
multiplier					
— byte		12-17/ 15-20*	12-17/ 15-20*	b, d	d, h
— word		12-25/ 15-28*	12-25/ 15-28*	b, d	d, h
— doubleword		12-41/ 17-46*	12-41/ 17-46*	b, d	d, h
register with register/memory	0 0 0 0 1 1 1 1 | 1 0 1 0 1 1 1 1 | mod reg r/m				
multiplier					
— byte		12-17/ 15-20*	12-17/ 15-20*	b, d	d, h
— word		12-25/ 15-28*	12-25/ 15-28*	b, d	d, h
— doubleword		12-41/ 17-46*	12-41/ 17-46*	b, d	d, h
register/memory with immediate to register	0 1 1 0 1 0 s 1 | mod reg r/m | immediate data				
— word		13-26	13-26/ 14-27	b, d	d, h
— doubleword		13-42	13-42/ 16-45	b, d	d, h
DIV = Divide (unsigned)					
Accumulator by register/memory	1 1 1 1 0 1 1 w | mod 1 1 0 r/m				
divisor					
— byte		14/17	14/17	b, e	e, h
— word		22/25	22/25	b, e	e, h
— doubleword		38/43	38/43	b, e	e, h

Table E-1. Instruction Set Summary (Sheet 6 of 19)

Instruction	Format	Clock Count: Real Address Mode or Virtual 8086 Mode	Clock Count: Protected Virtual Address Mode	Notes: Real Address Mode or Virtual 8086 Mode	Notes: Protected Virtual Address Mode
IDIV = Integer divide (signed)					
Accumulator by register/memory	1 1 1 1 0 1 1 w | mod 111 r/m				
divisor					
— byte		19/22	19/22	b, e	e, h
— word		27/30	27/30	b, e	e, h
— doubleword		43/48	43/48	b, e	e, h
AAD = ASCII adjust for divide	1 1 0 1 0 1 0 1 | 0 0 0 0 1 0 1 0	19	19		
AAM = ASCII adjust for multiply	1 1 0 1 0 1 0 0 | 0 0 0 0 1 0 1 0	17	17		
CBW = Convert byte to word	1 0 0 1 1 0 0 0	3	3		
CWD = Convert word to double-word	1 0 0 1 1 0 0 1	2	2		
LOGIC					
shift rotate instructions not through carry (**ROL**, **ROR**, **SAL**, **SAR**, **SHL**, and **SHR**)					
register/memory by 1	1 1 0 1 0 0 0 w | mod TTT r/m	3/7**	3/7**	b	h
register/memory by CL	1 1 0 1 0 0 1 w | mod TTT r/m	3/7*	3/7*	b	h
register/memory by immediate count	1 1 0 0 0 0 0 w | mod TTT r/m | immed 8-bit data	3/7*	3/7*	b	h
through carry (**RCL** and **RCR**)					
register/memory by 1	1 1 0 1 0 0 0 w | mod TTT r/m	9/10*	9/10*	b	h
register/memory by CL	1 1 0 1 0 0 1 w | mod TTT r/m	9/10*	9/10*	b	h
register/memory by immediate count	1 1 0 0 0 0 0 w | mod TTT r/m | immed 8-bit data	9/10*	9/10*	b	h
	TTT / **Instruction**: 000 ROL; 001 ROR; 010 RCL; 011 RCR; 100 SHL/SAL; 101 SHR; 111 SAR				
SHLD = Shift left double					
register/memory by immediate	0 0 0 0 1 1 1 1 | 1 0 1 0 0 1 0 0 | mod reg r/m | immed 8-bit data	3/7**	3/7**		
register/memory by CL	0 0 0 0 1 1 1 1 | 1 0 1 0 0 1 0 1 | mod reg r/m	3/7**	3/7**		

Table E-1. Instruction Set Summary (Sheet 7 of 19)

Instruction	Format	Clock Count: Real Address Mode or Virtual 8086 Mode	Clock Count: Protected Virtual Address Mode	Notes: Real Address Mode or Virtual 8086 Mode	Notes: Protected Virtual Address Mode
SHRD = Shift right double					
register/memory by immediate	0 0 0 0 1 1 1 1 | 1 0 1 0 1 1 0 0 | mod reg r/m | immed 8-bit data	3/7**	3/7**		
register/memory by CL	0 0 0 0 1 1 1 1 | 1 0 1 0 1 1 0 1 | mod reg r/m	3/7**	3/7**		
AND = And					
register to register	0 0 1 0 0 0 d w | mod reg r/m	2	2		
register to memory	0 0 1 0 0 0 0 w | mod reg r/m	7**	7**	b	h
memory to register	0 0 1 0 0 0 1 w | mod reg r/m	6*	6*	b	h
immediate to register/memory	1 0 0 0 0 0 0 w | mod 1 0 0 r/m | immediate data	2/7*	2/7*	b	h
immediate to accumulator (short form)	0 0 1 0 0 1 0 w | immediate data	2	2		
TEST= And function to flags, no result					
register/memory and register	1 0 0 0 0 1 0 w | mod reg r/m	2/5*	2/5*	b	h
immediate data and register/memory	1 1 1 1 0 1 1 w | mod 0 0 0 r/m | immediate data	2/5*	2/5*	b	h
immediate data and accumulator (short form)	1 0 1 0 1 0 0 w | immediate data	2	2		
OR = Or					
register to register	0 0 0 0 1 0 d w | mod reg r/m	2	2		
register to memory	0 0 0 0 1 0 0 w | mod reg r/m	7**	7**	b	h
memory to register	0 0 0 0 1 0 1 w | mod reg r/m	6*	6*	b	h
immediate to register/memory	1 0 0 0 0 0 0 w | mod 0 0 1 r/m | immediate data	2/7**	2/7**	b	h
immediate to accumulator (short form)	0 0 0 0 1 1 0 w | immediate data	2	2		
XOR = Exclusive or					
register to register	0 0 1 1 0 0 d w | mod reg r/m	2	2		
register to memory	0 0 1 1 0 0 0 w | mod reg r/m	7**	7**	b	h
memory to register	0 0 1 1 0 0 1 w | mod reg r/m	6*	6*	b	h
immediate to register/memory	1 0 0 0 0 0 0 w | mod 1 1 0 r/m | immediate data	2/7**	2/7**	b	h
immediate to accumulator (short form)	0 0 1 1 0 1 0 w | immediate data	2	2		
NOT= Invert register/memory	1 1 1 1 0 1 1 w | mod 0 1 0 r/m	2/6**	2/6**	b	h

Table E-1. Instruction Set Summary (Sheet 8 of 19)

Instruction	Format				Clock Count		Notes	
					Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
STRING MANIPULATION INSTRUCTIONS				Clk Count Virtual 8086 Mode				
CMPS = Compare byte word	1 0 1 0 0 1 1 w				10*	10*	b	h
INS = Input byte/word from DX port	0 1 1 0 1 1 0 w			†30	17	10*/32**	b	s/t, h, m
LODS = Load byte/word to AL/AX/EAX	1 0 1 0 1 1 0 w				5	5*	b	h
MOVS = Move byte word	1 0 1 0 0 1 0 w				7	7**	b	h
OUTS = Output byte/word to DX port	0 1 1 0 1 1 1 w			†31	18	11*/33*	b	s/t, h, m
SCAS = Scan byte word	1 0 1 0 1 1 1 w				7*	7*	b	h
STOS = Store byte/word from AL/AX/EX	1 0 1 0 1 0 1 w				4*	4*	b	h
XLAT = Translate String	1 1 0 1 0 1 1 1				5*	5*		h
REPEATED STRING MANIPULATION Repeated by count in CX or ECX:								
REPE CMPS = Compare string								
find non-match	1 1 1 1 0 0 1 1	1 0 1 0 0 1 1 w		Clk Count Virtual 8086 Mode	5 + 9n**	5 + 9n**	b	h
find match	1 1 1 1 0 0 1 0	1 0 1 0 0 1 1 w			5 + 9n**	5 + 9n**	b	h
REP INS = Input string	1 1 1 1 0 0 1 0	0 1 1 0 1 1 0 w		†31+6n	17 + 7n*	11 + 7n*/ 32+ 6n*	b	s/t, h, m
REP LODS = Load string	1 1 1 1 0 0 1 0	1 0 1 0 1 1 0 w			5 + 6n*	5 + 6n*	b	h
REP MOVS = Move string	1 1 1 1 0 0 1 0	1 0 1 0 0 1 0 w			7 + 4n*	7 + 4n**	b	h
REP OUTS = Output string	1 1 1 1 0 0 1 0	0 1 1 0 1 1 1 w		†30+8n	16 + 8n*	10 + 8n*/ 31+ 8n*	b	s/t, h, m
REPE SCAS = Scan string Find non-AL/AX/EAX	1 1 1 1 0 0 1 1	1 0 1 0 1 1 1 w			5 + 8n*	5 + 8n*	b	h
REPNE SCAS = Scan string Find AL/AX/EAX	1 1 1 1 0 0 1 0	1 0 1 0 1 1 1 w			5 + 8n*	5 + 8n*	b	h
REP STOS = Store string	1 1 1 1 0 0 1 0	1 0 1 0 1 0 1 w			5 + 5n*	5 + 5n*	b	h
BIT MANIPULATION								
BSF = scan bit forward	0 0 0 0 1 1 1 1	1 0 1 1 1 1 0 0	mod reg r/m		10 + 3n*	10 + 3n*	b	h
BSR = scan bit reverse	0 0 0 0 1 1 1 1	1 0 1 1 1 1 0 1	mod reg r/m		10 + 3n*	10 + 3n*	b	h
BT = test bit								
register/memory, immediate	0 0 0 0 1 1 1 1	1 0 1 1 1 0 1 0	mod 1 0 0 r/m	immed 8-bit data	3/6*	3/6*	b	h

Table E-1. Instruction Set Summary (Sheet 9 of 19)

Instruction	Format	Clock Count: Real Address Mode or Virtual 8086 Mode	Clock Count: Protected Virtual Address Mode	Notes: Real Address Mode or Virtual 8086 Mode	Notes: Protected Virtual Address Mode
register/memory, register	0 0 0 0 1 1 1 1 | 1 0 1 0 0 0 1 1 | mod reg r/m	3/12*	3/12*	b	h
BTC = test bit and complement					
register/memory, immediate	0 0 0 0 1 1 1 1 | 1 0 1 1 1 0 1 0 | mod 1 1 1 r/m | immed 8-bit data	6/8*	6/8*	b	h
register/memory, register	0 0 0 0 1 1 1 1 | 1 0 1 1 1 0 1 1 | mod reg r/m	6/13*	6/13*	b	h
BTR = test bit and reset					
register/memory, immediate	0 0 0 0 1 1 1 1 | 1 0 1 1 1 0 1 0 | mod 1 1 0 r/m | immed 8-bit data	6/8*	6/8*	b	h
register/memory, register	0 0 0 0 1 1 1 1 | 1 0 1 1 0 0 1 1 | mod reg r/m	6/13*	6/13*	b	h
BTS = test bit and set					
register/memory, immediate	0 0 0 0 1 1 1 1 | 1 0 1 1 1 0 1 0 | mod 1 0 1r/m | immed 8-bit data	6/8*	6/8*	b	h
register/memory, register	0 0 0 0 1 1 1 1 | 1 0 1 0 1 0 1 1 | mod reg r/m	6/13*	6/13*	b	h
CONTROL TRANSFER					
CALL = Call					
direct within segment	1 1 1 0 1 0 0 0 | full displacement	7 + m*	9 + m*	b	r
reg/memory indirect within segment	1 1 1 1 1 1 1 1 | mod 0 1 0 r/m	7 + m*/ 10 + m*	9 + m*/ 12 + m*	b	h, r
direct intersegment	1 0 0 1 1 0 1 0 | unsigned full offset, selector	17 + m*	42 + m*	b	j, k, r
Protected mode only (direct intersegment)					
Via call gate to same privilege level			64 + m		h, j, k, r
Via call gate to different privilege level (no parameters)			98 + m		h, j, k, r
Via call gate to different privilege level (x parameters)			106+8x+ m		h, j, k, r
From 286 task to 286 TSS			285		h, j, k, r
From 286 task to Intel386 SX CPU TSS			310		h, j, k, r
From 286 task to virtual 8086 task (Intel386 SX CPU TSS)			229		h, j, k, r
From Intel386 SX CPU task to 286 TSS			285		h, j, k, r
From Intel386 SX CPU task to Intel386 SX CPU TSS			392		h, j, k, r
From Intel386 SX CPU task to Virtual 8086 task (Intel386 SX CPU TSS)			309		h, j, k, r
indirect intersegment	1 1 1 1 1 1 1 1 | mod 0 1 1 r/m	30 + m	46 + m	b	h, j, k, r
Protected mode only (indirect intersegment)					
Via call gate to same privilege level			68 + m		h, j, k, r

Table E-1. Instruction Set Summary (Sheet 10 of 19)

Instruction	Format	Clock Count		Notes	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
Via call gate to different privilege level (no parameters)			102 + m		h, j, k, r
Via call gate to different privilege level (x parameters)			110+8x+m		h, j, k, r
From 286 task to 286 TSS					h, j, k, r
From 286 task to Intel386 SX CPU TSS					h, j, k, r
From 286 task to virtual 8086 task (Intel386 SX CPU TSS)					h, j, k, r
From Intel386 SX CPU task to 286 TSS					h, j, k, r
From Intel386 SX CPU task to Intel386 SX CPU TSS			399		h, j, k, r
From Intel386 SX CPU task to Virtual 8086 task (Intel386 SX CPU TSS)					h, j, k, r
JMP = Unconditional jump					
short	1 1 1 0 1 0 1 1 8-bit displacement	7 + m	7 + m		r
direct within segment	1 1 1 0 1 0 0 1 full displacement	7 + m	7 + m		r
reg/memory indirect within segment	1 1 1 1 1 1 1 1 mod 1 0 0 r/m	9 + m/ 14 + m	9 + m/ 14 + m	b	h, r
direct intersegment	1 1 1 0 1 0 1 0 unsigned full offset, selector	16 + m	31 + m		j, k, r
Protected mode only (direct intersegment)					
Via call gate to same privilege level			53 + m		h, j, k, r
From 286 task to 286 TSS					h, j, k, r
From 286 task to Intel386 SX CPU TSS					h, j, k, r
From 286 task to virtual 8086 task (Intel386 SX CPU TSS)					h, j, k, r
From Intel386 SX CPU task to 286 TSS					h, j, k, r
From Intel386 SX CPU task to Intel386 SX CPU TSS					h, j, k, r
From Intel386 SX CPU task to Virtual 8086 task (Intel386 SX CPU TSS)			395		h, j, k, r
indirect intersegment	1 1 1 1 1 1 1 1 mod 1 0 1 r/m	17 + m	31 + m	b	h, j, k, r
Protected mode only (indirect intersegment)					
Via call gate to same privilege level			49 + m		h, j, k, r
From 286 task to 286 TSS					h, j, k, r
From 286 task to Intel386 SX CPU TSS					h, j, k, r
From 286 task to virtual 8086 task (Intel386 SX CPU TSS)					h, j, k, r
From Intel386 SX CPU task to 286 TSS					h, j, k, r
From Intel386 SX CPU task to Intel386 SX CPU TSS			328		h, j, k, r
From Intel386 SX CPU task to Virtual 8086 task (Intel386 SX CPU TSS)					h, j, k, r
RET = Return from CALL					
within segment	1 1 0 0 0 0 1 1		12 + m	b	g, h, r

Table E-1. Instruction Set Summary (Sheet 11 of 19)

Instruction	Format	Clock Count: Real Address Mode or Virtual 8086 Mode	Clock Count: Protected Virtual Address Mode	Notes: Real Address Mode or Virtual 8086 Mode	Notes: Protected Virtual Address Mode
within segment adding immed to SP	1 1 0 0 0 0 1 0 | 16-bit displacement		12 + m	b	g, h, r
intersegment	1 1 0 0 1 0 1 1		36 + m	b	g, h, j, k, r
intersegment adding immed to SP	1 1 0 0 1 0 1 0 | 16-bit displacement		36 + m	b	g, h, j, k, r
Protected mode only (RET):					
to different privilege level					
Intersegment			72		h, j, k, r
Intersegment adding immed to SP			72		h, j, k, r
CONDITIONAL JUMPS (times are jump "Taken or not Taken")					
JO = jump on overflow					
8-bit displacement	0 1 1 1 0 0 0 0 | 8-bit displacement	7 + m or 3	7 + m or 3		r
Full displacement	0 0 0 0 1 1 1 1 | 1 0 0 0 0 0 0 0 | Full displacement	7 + m or 3	7 + m or 3		r
JNO = Jump on not overflow					
8-bit displacement	0 1 1 1 0 0 0 1 | 8-bit displacement	7 + m or 3	7 + m or 3		r
Full displacement	0 0 0 0 1 1 1 1 | 1 0 0 0 0 0 0 1 | Full displacement	7 + m or 3	7 + m or 3		r
JB/JNAE = jump on below/not above or equal					
8-bit displacement	0 1 1 1 0 0 1 0 | 8-bit displacement	7 + m or 3	7 + m or 3		r
Full displacement	0 0 0 0 1 1 1 1 | 1 0 0 0 0 0 1 0 | Full displacement	7 + m or 3	7 + m or 3		r
JNB/JAE = jump on not below/above or equal					
8-bit displacement	0 1 1 1 0 0 1 1 | 8-bit displacement	7 + m or 3	7 + m or 3		r
Full displacement	0 0 0 0 1 1 1 1 | 1 0 0 0 0 0 1 1 | Full displacement	7 + m or 3	7 + m or 3		r
JE/JZ= jump on equal/zero					
8-bit displacement	0 1 1 1 0 1 0 0 | 8-bit displacement	7 + m or 3	7 + m or 3		r
Full displacement	0 0 0 0 1 1 1 1 | 1 0 0 0 0 1 0 0 | Full displacement	7 + m or 3	7 + m or 3		r
JNE/JNZ = jump on not equal/not zero					
8-bit displacement	0 1 1 1 0 1 0 1 | 8-bit displacement	7 + m or 3	7 + m or 3		r

Table E-1. Instruction Set Summary (Sheet 12 of 19)

Instruction	Format	Clock Count		Notes	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
Full displacement	00001111 | 10000101 | Full displacement	7 + m or 3	7 + m or 3		r
JBE/JNA = jump on below or equal/not above					
8-bit displacement	01110110 | 8-bit displacement	7 + m or 3	7 + m or 3		r
Full displacement	00001111 | 10000110 | Full displacement	7 + m or 3	7 + m or 3		r
JNBE/JA = jump on not below or equal/above					
8-bit displacement	01110111 | 8-bit displacement	7 + m or 3	7 + m or 3		r
Full displacement	00001111 | 10000111 | Full displacement	7 + m or 3	7 + m or 3		r
JS = jump on sign					
8-bit displacement	01111000 | 8-bit displacement	7 + m or 3	7 + m or 3		r
Full displacement	00001111 | 10001000 | Full displacement	7 + m or 3	7 + m or 3		r
JNS = jump on not sign					
8-bit displacement	01111001 | 8-bit displacement	7 + m or 3	7 + m or 3		r
Full displacement	00001111 | 10001001 | Full displacement	7 + m or 3	7 + m or 3		r
JP/JPE = jump on parity/parity even					
8-bit displacement	01111010 | 8-bit displacement	7 + m or 3	7 + m or 3		r
Full displacement	00001111 | 10001010 | Full displacement	7 + m or 3	7 + m or 3		r
JNP/JPO = jump on not parity/parity odd					
8-bit displacement	01111011 | 8-bit displacement	7 + m or 3	7 + m or 3		r
Full displacement	00001111 | 10001011 | Full displacement	7 + m or 3	7 + m or 3		r
JL/JNGE = jump on less/not greater or equal					
8-bit displacement	01111100 | 8-bit displacement	7 + m or 3	7 + m or 3		r
Full displacement	00001111 | 10001100 | Full displacement	7 + m or 3	7 + m or 3		r
JNL/JGE = jump on not less/greater or equal					
8-bit displacement	01111101 | 8-bit displacement	7 + m or 3	7 + m or 3		r

Table E-1. Instruction Set Summary (Sheet 13 of 19)

Instruction	Format	Clock Count		Notes	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
Full displacement	00001111 | 10001101 | Full displacement	7 + m or 3	7 + m or 3		r
JLE/JNG = jump on less or equal/not greater					
8-bit displacement	01111110 | 8-bit displacement	7 + m or 3	7 + m or 3		r
Full displacement	00001111 | 10001110 | Full displacement	7 + m or 3	7 + m or 3		r
JNLE/JG = jump on not less or equal/greater					
8-bit displacement	01111111 | 8-bit displacement	7 + m or 3	7 + m or 3		r
Full displacement	00001111 | 10001111 | Full displacement	7 + m or 3	7 + m or 3		r
JCXZ = jump on CX zero	11100011 | 8-bit displacement	9 + m or 5	9 + m or 5		r
JECXZ = jump on ECX zero	11100011 | 8-bit displacement	9 + m or 5	9 + m or 5		r
(Address size prefix differentiates JCXZ from JECXZ)					
LOOP = loop CX times	11100010 | 8-bit displacement	11 + m	11 + m		r
LOOPZ/LOOPE = loop with zero/equal	11100001 | 8-bit displacement	11 + m	11 + m		r
LOOPNZ/LOOPNE = loop while not zero	11100000 | 8-bit displacement	11 + m	11 + m		r
CONDITIONAL BYTE SET					
(Note: Times are register/memory)					
SETO = set byte on overflow					
to register/memory	00001111 | 10010000 | mod 000 r/m	4/5*	4/5*		h
SETNO = set byte on not overflow					
to register/memory	00001111 | 10010001 | mod 000 r/m	4/5*	4/5*		h
SETB/SETNAE = set byte on below/not above or equal					
to register/memory	00001111 | 10010010 | mod 000 r/m	4/5*	4/5*		h
SETNB = set byte on below/above or equal					
to register/memory	00001111 | 10010011 | mod 000 r/m	4/5*	4/5*		h
SETE/SETZ = set byte on equal/zero					
to register/memory	00001111 | 10010100 | mod 000 r/m	4/5*	4/5*		h
SETNE/SETNZ = set byte on not equal/not zero					
to register/memory	00001111 | 10010101 | mod 000 r/m	4/5*	4/5*		h

Table E-1. Instruction Set Summary (Sheet 14 of 19)

Instruction	Format	Clock Count: Real Address Mode or Virtual 8086 Mode	Clock Count: Protected Virtual Address Mode	Notes: Real Address Mode or Virtual 8086 Mode	Notes: Protected Virtual Address Mode
SETBE/SETNA = set byte on below or equal/not above					
to register/memory	0 0 0 0 1 1 1 1 | 1 0 0 1 0 1 1 0 | mod 0 0 0 r/m	4/5*	4/5*		h
SETNBE/SETA = set byte on not below or equal/above					
to register/memory	0 0 0 0 1 1 1 1 | 1 0 0 1 0 1 1 1 | mod 0 0 0 r/m	4/5*	4/5*		h
SETS = set byte on sign					
to register/memory	0 0 0 0 1 1 1 1 | 1 0 0 1 1 0 0 0 | mod 0 0 0 r/m	4/5*	4/5*		h
SETNS = set byte on not sign					
to register/memory	0 0 0 0 1 1 1 1 | 1 0 0 1 1 0 0 1 | mod 0 0 0 r/m	4/5*	4/5*		h
SETP/SETPE = set byte on parity/parity even					
to register/memory	0 0 0 0 1 1 1 1 | 1 0 0 1 1 0 1 0 | mod 0 0 0 r/m	4/5*	4/5*		h
SETNP/SETPO = set byte on not parity/parity odd					
to register/memory	0 0 0 0 1 1 1 1 | 1 0 0 1 1 0 1 1 | mod 0 0 0 r/m	4/5*	4/5*		h
SETL/SETNGE = set byte on less/not greater or equal					
to register/memory	0 0 0 0 1 1 1 1 | 1 0 0 1 1 1 0 0 | mod 0 0 0 r/m	4/5*	4/5*		h
SETNL/SETGE = set byte on not less/greater or equal					
to register/memory	0 0 0 0 1 1 1 1 | 0 1 1 1 1 1 0 1 | mod 0 0 0 r/m	4/5*	4/5*		h
SETLE/SETNG = set byte on less or equal/not greater					
to register/memory	0 0 0 0 1 1 1 1 | 1 0 0 1 1 1 1 0 | mod 0 0 0 r/m	4/5*	4/5*		h
SETNLE/SETG = set byte on not less or equal/greater					
to register/memory	0 0 0 0 1 1 1 1 | 1 0 0 1 1 1 1 1 | mod 0 0 0 r/m	4/5*	4/5*		h
ENTER = enter procedure	1 1 0 0 1 0 0 0 | 16-bit displacement, 8-bit level				
L = 0		10	10	b	h
L = 1		14	14	b	h
L > 1		17+8(n - 1)	17+8(n - 1)		
LEAVE = leave procedure	1 1 0 0 1 0 0 1	4	4	b	h
INTERRUPT INSTRUCTIONS					
INT = Interrupt:					
Type specified	1 1 0 0 1 1 0 1 | type	37		b	
Type 3	1 1 0 0 1 1 0 0	33		b	
INTO = Interrupt 4 if overflow flag set	1 1 0 0 1 1 1 0				
If OF = 1		35		b, e	
If OF = 0		3	3	b, e	

Table E-1. Instruction Set Summary (Sheet 15 of 19)

Instruction	Format	Clock Count		Notes	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
BOUND = Interrupt 5 if Detect value out of range	0 1 1 0 0 0 1 0 | mod reg r/m				
If out of range		44		b, e	e, g, h, j, k, r
If in range		10	10	b, e	e, g, h, j, k, r
Protected Mode Only (INT) INT: Type Specified					
Via interrupt or Trap Gate To same privilege level			71		g, j, k, r
Via Interrupt or Trap Gate To different privilege level			111		g, j, k, r
From 286 task to 286 TSS via Task Gate			438		g, j, k, r
From 286 task to Intel 386 SX CPU TSS via Task Gate			465		g, j, k, r
From 286 task to virtual 8086 mode via Task Gate			382		g, j, k, r
From Intel386 SX CPU task to 286 TSS via Task Gate			440		g, j, k, r
From Intel386 SX CPU task to Intel 386 SX CPU TSS via task gate			467		g, j, k, r
From Intel386 SX CPU task to virtual 8086 mode via Task Gate			384		g, j, k, r
From virtual 8086 mode to 286 TSS via Task Gate			445		g, j, k, r
From virtual 8086 mode to Intel 386 SX CPU TSS via task gate			472		g, j, k, r
From virtual 8086 mode to privilege level 0 via trap gate or interrupt gate			275		g, j, k, r
INT: TYPE 3					
Via interrupt or Trap Gate To same privilege level			71		g, j, k, r
Via Interrupt or Trap Gate To different privilege level			111		g, j, k, r
From 286 task to 286 TSS via Task Gate			382		g, j, k, r
From 286 task to Intel 386 SX CPU TSS via Task Gate			409		g, j, k, r
From 286 task to virtual 8086 mode via Task Gate			326		g, j, k, r
From Intel386 SX CPU task to 286 TSS via Task Gate			384		g, j, k, r
From Intel386 SX CPU task to Intel 386 SX CPU TSS via task gate			411		g, j, k, r
From Intel386 SX CPU task to virtual 8086 mode via Task Gate			328		g, j, k, r
From virtual 8086 mode to 286 TSS via Task Gate			389		g, j, k, r
From virtual 8086 mode to Intel 386 SX CPU TSS via task gate			416		g, j, k, r
From virtual 8086 mode to privilege level 0 via trap gate or interrupt gate			223		

Table E-1. Instruction Set Summary (Sheet 16 of 19)

Instruction	Format	Clock Count		Notes	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
INTO:					
Via interrupt or Trap Gate To same privilege level			71		g, j, k, r
Via Interrupt or Trap Gate To different privilege level			111		g, j, k, r
From 286 task to 286 TSS via Task Gate			384		g, j, k, r
From 286 task to Intel 386 SX CPU TSS via Task Gate			411		g, j, k, r
From 286 task to virtual 8086 mode via Task Gate			328		g, j, k, r
From Intel386 SX CPU task to 286 TSS via Task Gate			Intel386 DX		g, j, k, r
From Intel386 SX CPU task to Intel 386 SX CPU TSS via task gate			413		g, j, k, r
From Intel386 SX CPU task to virtual 8086 mode via Task Gate			329		g, j, k, r
From virtual 8086 mode to 286 TSS via Task Gate			391		g, j, k, r
From virtual 8086 mode to Intel 386 SX CPU TSS via task gate			418		g, j, k, r
From virtual 8086 mode to privilege level 0 via trap gate or interrupt gate			223		
BOUND:					
Via interrupt or Trap Gate To same privilege level			71		g, j, k, r
Via Interrupt or Trap Gate To different privilege level			111		g, j, k, r
From 286 task to 286 TSS via Task Gate			358		g, j, k, r
From 286 task to Intel 386 SX CPU TSS via Task Gate			388		g, j, k, r
From 286 task to virtual 8086 mode via Task Gate			335		g, j, k, r
From Intel386 SX CPU task to 286 TSS via Task Gate			368		g, j, k, r
From Intel386 SX CPU task to Intel 386 SX CPU TSS via task gate			398		g, j, k, r
From Intel386 SX CPU task to virtual 8086 mode via Task Gate			347		g, j, k, r
From virtual 8086 mode to 286 TSS via Task Gate			368		g, j, k, r
From virtual 8086 mode to Intel 386 SX CPU TSS via task gate			398		g, j, k, r
From virtual 8086 mode to privilege level 0 via trap gate or interrupt gate			223		
INTERRUPT RETURN					
IRET = Interrupt return	1 1 0 0 1 1 1 1	24			g, h, j, k, r
Protected Mode Only (IRET)					
To same privilege level (within task)			42		g, h, j, k, r

Table E-1. Instruction Set Summary (Sheet 17 of 19)

Instruction	Format	Clock Count		Notes	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
To different privilege level (within task)			86		g, h, j, k, r
From 286 task to 286 TSS			285		g, h, j, k, r
From 286 task to Intel386 SX CPU TSS			318		g, h, j, k, r
From 286 task to virtual 8086 task			267		g, h, j, k, r
From 286 task to virtual 8086 mode (within task)			113		
From Intel386 SX CPU task to 286 TSS			324		g, h, j, k, r
From Intel386 SX CPU task to Intel386 SX CPU TSS			328		g, h, j, k, r
From Intel386 SX CPU task to virtual 8086 task			377		g, h, j, k, r
From Intel386 SX CPU task to virtual 8086 mode (within task)			113		
PROCESSOR CONTROL INSTRUCTIONS					
HLT = Halt	1 1 1 1 0 1 0 0	7	7		l
MOV = Move to and from control/debug/test registers					
CR0/CR2/CR3 from register	0 0 0 0 1 1 1 1 | 0 0 1 0 0 0 1 0 | 1 1 e e e reg	10/4/5	10/4/5		l
register from CR0-3	0 0 0 0 1 1 1 1 | 0 0 1 0 0 0 0 0 | 1 1 e e e reg	6	6		l
DR0-3 from register	0 0 0 0 1 1 1 1 | 0 0 1 0 0 0 1 1 | 1 1 e e e reg	22	22		l
DR6-7 from register	0 0 0 0 1 1 1 1 | 0 0 1 0 0 0 1 1 | 1 1 e e e reg	16	16		l
register from DR6-7	0 0 0 0 1 1 1 1 | 0 0 1 0 0 0 0 1 | 1 1 e e e reg	14	14		l
register from DR0-3	0 0 0 0 1 1 1 1 | 0 0 1 0 0 0 0 1 | 1 1 e e e reg	22	22		l
TR6-7 from register	0 0 0 0 1 1 1 1 | 0 0 1 0 0 1 1 0 | 1 1 e e e reg	12	12		l
register from TR6-7	0 0 0 0 1 1 1 1 | 0 0 1 0 0 1 0 0 | 1 1 e e e reg	12	12		l
NOP = No operation	1 0 0 1 0 0 0 0	3	3		
WAIT = Wait until BUSY# pin is negated	1 0 0 1 1 0 1 1	6	6		
PROCESSOR EXTENSION INSTRUCTIONS					
Processor Extension Escape	1 1 0 1 1 T T T | mod L L L r/m TTT and LLL bits are opcode information for coprocessor	See Intel387 SX datasheet for clock counts			h

Table E-1. Instruction Set Summary (Sheet 18 of 19)

Instruction	Format	Clock Count		Notes	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
PREFIX BYTES					
Address size prefix	01100111	0	0		
LOCK = Bus lock prefix	11110000	0	0		m
Operand size prefix	01100110	0	0		
Segment override prefix					
CS:	00101110	0	0		
DS:	00111110	0	0		
ES:	00100110	0	0		
FS:	01100100	0	0		
GS:	01100101	0	0		
SS:	00110110	0	0		
PROTECTION CONTROL					
ARPL = adjust requested privilege level					
from register/memory	01100011 | mod reg r/m	N/A	20/21**	a	h
LAR = load access rights					
from register/memory	00001111 | 00000010 | mod reg r/m	N/A	15/16*	a	g, h, j, p
LGDT = load global descriptor					
table register	00001111 | 00000001 | mod 010 r/m	11*	11*	b, c	h, l
LIDT = load interrupt descriptor					
table register	00001111 | 00000001 | mod 011 r/m	11*	11*	b, c	h, l
LLDT = load local descriptor					
table register to register/memory	00001111 | 00000000 | mod 010 r/m	N/A	20/24*	a	g, h, j, l
LMSW = load machine status word					
from register/memory	00001111 | 00000001 | mod 110 r/m	10/13	10/13*	b, c	h, l
LSL = load segment limit					
from register memory	00001111 | 00000011 | mod reg r/m				
Byte-Granular limit		N/A	20/21*	a	g, h, j, p
Page-Granular limit		N/A	25/26*	a	g, h, j, p
LTR = load task register					
from register/memory	00001111 | 00000000 | mod 001 r/m	N/A	23/27*	a	g, h, j, l
SGDT = store global descriptor					
table register	00001111 | 00000001 | mod 000 r/m	9*	9*	b, c	h
SIDT = store interrupt descriptor					

Table E-1. Instruction Set Summary (Sheet 19 of 19)

Instruction	Format			Clock Count		Notes	
				Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
table register	00001111	00000001	mod 001 r/m	9*	9*	b, c	h
SLDT = store local descriptor table							
to register/memory	00001111	00000000	mod 000 r/m	N/A	2/2*	a	h
SMSW = store machine status word	00001111	00000001	mod 100 r/m	2/2*	2/2*	b, c	h, l
STR = store task register							
to register/memory	00001111	00000000	mod 001 r/m	N/A	2/2*	a	h
VERR = verify read access							
register/memory	00001111	00000000	mod 100 r/m	N/A	10/11*	a	g, h, j, p
VERW = verify write access	00001111	00000000	mod 101 r/m	N/A	15/16*	a	g, h, j, p

NOTES:

Notes a through c apply to Real Address Mode only:

a. This is a Protected Mode instruction. Attempted execution in Real Mode will result in exception 6 (invalid opcode).

b. Exception 13 fault (general protection) will occur in Real Mode if an operand reference is made that partially or fully extends beyond the maximum CS, DS, ES, FS, or GS limit, FFFFH. Exception 12 fault (stack segment limit violation or not present) will occur in Real Mode if an operand reference is made that partially or fully extends beyond the maximum SS limit.

c. This instruction may be executed in Real Mode, its purpose is primarily to initialize the CPU for Protected Mode.

Notes d through g apply to Real Address Mode and Protected Virtual Address Mode:

d. The Intel386 SX CPU uses an early-out multiply algorithm. The actual number of clocks depends on the position of the most significant bit in the operand (multiplier).

Clock counts given are minimum to maximum. To calculate actual clocks use this formula:

Actual Clock = if m < > 0 then max ([$\log_2$ |m|], 3) + b clocks:
if m = 0 then 3 + b clocks

In this formula, m is the multiplier, and

b = 9 for register to register
b = 12 for memory to register

b = 10 for register with immediate to register
b = 11 for memory with immediate to register.

e. An exception may occur, depending on the value of the operand.

f. LOCK# is automatically asserted, regardless of the presence or absence of the LOCK# prefix.

g. LOCK# is asserted during descriptor table accesses.

Notes h through r apply to Protected Virtual Address Mode only:

h. Exception 13 fault (general protection violation) will occur if the memory operand in CS, DS, ES, FS, or GS cannot be used due to either a segment limit violation or access rights violation. If a stack limit is violated, an exception 12 (stack segment limit violation or not present) occurs.

i. For segment load operations, the CPL, RPL, and DPL must agree with the privilege rules to avoid an exception 13 fault (general protection violation). The segment's descriptor must indicate "present" or exception 11 (CS, DS, ES, FS, GS not present.) If the SS register is loaded and a stack segment not present is detected, an exception 12 (stack segment limit violation or not present) occurs.

j. All segment descriptor accesses in the GDT or LDT made by this instruction will automatically assert LOCK# to maintain descriptor integrity in multiprocessor systems.

k. JMP, CALL, INT, RET, and IRET instructions referring to another code segment will cause an exception 13 (general protection violation) if an applicable privilege rule is violated.

l. An exception 13 fault occurs if CPL is greater than 0 (0 is the most privileged level)

m. An exception 13 fault occurs if CPL is greater than IOPL.

n. The IF bit of the flag register is not updated if CPL is greater than IOPL. The IOPL and VM fields of the flag register are updated only if CPL = 0.

o. The PE bit of the MSW (CR0) cannot be reset by this instruction. Use MOV into CR0 when resetting the PE bit.

p. Any violation of privilege rules as applied to the selector operand does not cause a protection exception; rather the zero flag is cleared.

q. If the coprocessor's memory operand violates a segment limit or segment access rights, an exception 13 fault (general protection exception) will occur before the ESC instruction is executed. An exception 12 fault (stack segment limit violation or not present) will occur if the stack limit is violated by the operand's starting address.

r. The destination of a JMP, CALL, INT, RET, or IRET must be in the defined limit of a code segment or an exception 13 fault (general protection violation) will occur.

s. The instruction will execute in s clocks if CPL $\leq$ IOPL. If CPL $>$ IOPL, the instruction will take t clocks.

† Clock count shown applies if I/O permission allows I/O to the port in virtual 8086 mode. If I/O bit map denies permission, exception 13 (general protection fault occurs; refer to clock counts for INT 3 instruction.

E.2 INSTRUCTION ENCODING

All instruction encodings are subsets of the general instruction format shown in Figure E-1. Instructions consist of one or two primary opcode bytes, possibly an address specifier consisting of the "mod r/m" byte and "scaled index" byte, a displacement if required, and an immediate data field if required.

Within the primary opcode or opcodes, smaller encoding fields may be defined. These fields vary according to the class of operation. The fields define such information as direction of the operation, size of the displacements, register encoding, or sign extension.

Almost all instructions referring to an operand in memory have an addressing mode byte following the primary opcode byte(s). This byte, the mod r/m byte, specifies the address mode to be used. Certain encodings of the mod r/m byte indicate a second addressing byte, the scale-index-base byte, which follows the mod r/m byte to fully specify the addressing mode.

Addressing modes can include a displacement immediately following the mod r/m byte, or scaled index byte. If a displacement is present, the possible sizes are 8, 16, or 32 bits.

If the instruction specifies an immediate operand, the immediate operand follows any displacement bytes. The immediate operand, if specified, is always the last field of the instruction.

Figure E-1 illustrates several of the fields that can appear in an instruction, such as the mod field and the r/m field, but the figure does not show all fields. Several smaller fields also appear in certain instructions, sometimes within the opcode bytes themselves. Table E-2 is a complete list of all fields appearing in the instruction set. Following Table E-2 are detailed tables for each field.

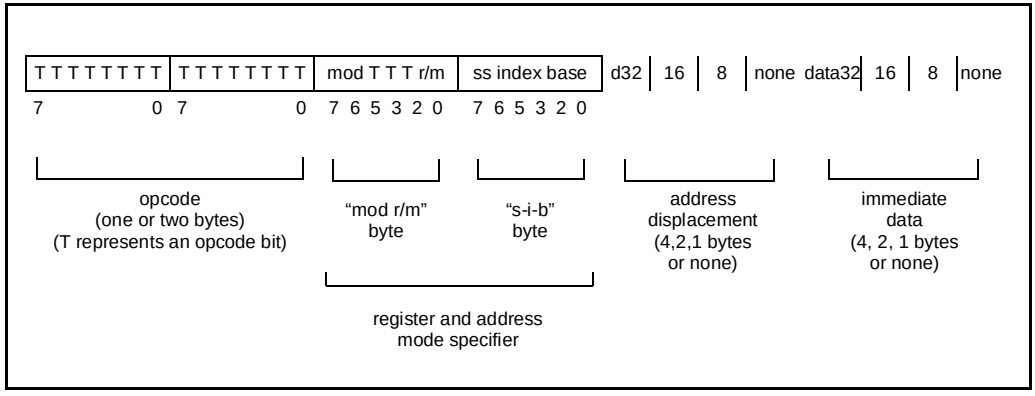


Figure E-1. General Instruction Format

Table E-2. Fields Within Instructions

Field Name	Description	Number of Bits
w	Specifies if data is byte of full size (full size is either 16 or 32 bits)	1
d	Specifies Direction of Data Operation	1
s	Specifies if an Immediate Data Field must be Sign-Extended	1
reg	General Register Specifier	3
mod r/m	Address Mode Specifier (Effective Address can be a General Register)	2 for mod: 3 for r/m
ss	Scale Factor for Scaled Index Address Mode	2
index	General Register to be used as Index Register	3
base	General Register to be used as Base Register	3
sreg2	Segment Register Specifier for CS, SS, DS, ES	2
sreg3	Segment Register Specifier for CS, SS, DS, ES, FS, GS	3
tttn	For Conditional Instructions, specifies a condition asserted or a condition negated	4

NOTE: Figure E-1 shows encoding of individual instructions.

E.2.1 32-bit Extensions of the Instruction Set

With the Intel386 EX processor the 8086/80186/80286 instruction set is extended in two orthogonal directions: 32-bit forms of all 16-bit instructions are added to support the 32-bit data types, and 32-bit addressing modes are made available for all instructions referencing memory. This orthogonal instruction set extension is accomplished having a Default (D) bit in the code segment descriptor, and by having 2 prefixes to the instruction set.

The instruction defaults to operations of 16 bits or 32 bits, depending on the setting of the D bit in the code segment descriptor, which gives the default length (either 32 bites or 16 bits) for both operands and effective addresses when executing that code segment. In the Real Address Mode or Virtual 8086 Mode, no code segment descriptors are used, but a D value of 0 is assumed internally by the Intel386 EX processor when operating in those modes (for 16-bit default sizes compatible with the 8086/80186/80286).

Two prefixes, the Operand Size Prefix and the Effective Address Size Prefix, allow overriding individually the Default selection of operand size and effective address size. These prefixes may precede any opcode bytes and affect only the instruction they precede. If necessary, one or both prefixes may be placed before the opcode bytes. The presence of the Operand Size Prefix and the Effective Address Prefix toggles the operand size or the effective address size, respectively, to the value “opposite” from the default setting. For example, if the default operand size is for 32-bit data operations, then presence of the Operand Size Prefix toggles the instruction to use 32-bit effective address computations.

These 32-bit extensions are available in all modes, including the Real Address Mode or the Virtual 8086 Mode. In these modes the default is always 16 bits, so prefixes are needed to specify 32-bit operands or addresses. For instructions with more than one prefix the order of prefixes is unimportant.

Unless specified otherwise, instructions with 8-bit and 16-bit operands do not affect the contents of the high-order bits of the extended registers.

E.2.2 Encoding of Instruction Fields

Within the instruction are several fields indicating register selection, addressing mode, and so on. The exact encodings of these fields are defined in the next several section.

E.2.2.1 Encoding of Operand Length (w) Field

For any given instruction performing a data operation, the instruction is executing as a 32-bit operation or a 16-bit operation. Within the constraints of the operation size, the w field encodes the operand size as either one byte or the full operation size, as shown in Table E-3.

Table E-3. Encoding of Operand Length (w) Field

w Field	Operand Size During 16-bit Data Operations	Operand Size During 32-bit Data Operations
0	8 bits	8 bits
1	16 bits	32 bits

E.2.2.2 Encoding of the General Register (reg) Field

The general register is specified by the reg field, which may appear in the primary opcode bytes, or as the reg field of the "mod r/m" byte, or as the r/m field of the "mod r/m" byte.

Table E-4. Encoding of reg Field When w Field is not Present in Instruction

reg Field	Register Selected During 16-bit Data Operations	Register Selected During 32-bit Data Operations
000	AX	EAX
001	CX	ECX
010	DX	EDX
011	BX	EBX
100	SP	ESP
101	BP	EBP
110	SI	ESI
111	DI	EDI

Table E-5. Encoding of reg Field When w Field is Present in Instruction

Register Specified by reg Field During 16-bit Data Operations		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Register Specified by reg Field During 32-bit Data Operations		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	EAX
001	CL	ECX
010	DL	EDX
011	BL	EBX
100	AH	ESP
101	CH	EBP
110	DH	ESI
111	BH	EDI

E.2.2.3 Encoding of the Segment Register (sreg) Field

The sreg field in certain instructions is a 2-bit field allowing one of the four 80286 segment registers to be specified. The sreg field in other instructions is a 3-bit field, allowing the FS and GS segment registers to be specified.

Table E-6. Encoding of the Segment Register (sreg) Field

2-Bit sreg2 Field	Segment Register Selected	3-Bit sreg3 Field	Segment Register Selected
00	ES	000	ES
01	CS	001	CS
10	SS	010	SS
11	DS	011	DS
		100	FS
		101	GS
		110	do not use
		111	do not use

E.2.2.4 Encoding of Address Mode

Except for special instructions, such as PUSH or POP, where the addressing mode is pre-determined, the addressing mode for the current instruction is specified by addressing bytes following the primary opcode. The primary addressing byte is the "mod r/m" byte, and a second byte of addressing information, the "s-i-b" (scale-index-base) byte can be specified.

The s-i-b byte is specified when using 32-bit addressing mode and the "mod r/m" byte has r/m = 100 and mod = 00, 01, or 10. When the s-i-b byte is present, the 32-bit addressing mode is a function of the mod, ss, index, and base fields.

The primary addressing byte, the "mod r/m" byte, also contains three bits (shown as TTT in Figure E-1) sometimes used as an extension of the primary opcode. The three bits, however, may also be used as a register field (reg).

When calculating an effective address, either 16-bit addressing or 32-bit addressing is used. 16-bit addressing uses 16-bit address components to calculate the effective address while 32-bit addressing uses 32-bit address components to calculate the effective address. When 16-bit addressing is used, the "mod r/m" byte is interpreted as a 32-bit addressing mode specifier.

The following tables define all encodings of all 16-bit addressing modes and 32-bit addressing modes.

Table E-7. Encoding of 16-bit Address Mode with "mod r/m" Byte

mod r/m	Effective Address
00 000	DS:[BX + SI]
00 001	DS:[BX + DI]
00 010	SS:[BP + SI]
00 011	SS:[BP + DI]
00 100	DS:[SI]
00 101	DS:[DI]
00 110	DS:d16
00 111	DS:[BX]
01 000	DS:[BX + SI + d8]
01 001	DS:[BX + DI + d8]
01 010	SS:[BP + SI + d8]
01 011	SS:[BP + DI + d8]
01 100	DS:[SI + d8]
01 101	DS:[DI + d8]
01 110	SS:[BP + d8]
01 111	DS:[BX + d8]

mod r/m	Effective Address
10 000	DS:[BX + SI + d16]
10 001	DS:[BX + DI + d16]
10 010	SS:[BP + SI + d16]
10 011	SS:[BX + DI + d16]
10 100	DS:[SI + d16]
10 101	DS:[DI + d16]
10 110	SS:[BP + d16]
10 111	DS:[BX + d16]
11 000	register - see tables below
11 001	register - see tables below
11 010	register - see tables below
11 011	register - see tables below
11 100	register - see tables below
11 101	register - see tables below
11 110	register - see tables below
11 111	register - see tables below

Register Specified by r/m During 16-bit Data Operations		
mod r/m	Function of w Field	
	(when w = 0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

Register Specified by r/m During 32-bit Data Operations		
mod r/m	Function of w Field	
	(when w = 0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI

Table E-8. Encoding of 32-bit Address Mode with "mod r/m" Byte (No s-i-b Byte Present)

mod r/m	Effective Address
00 000	DS:[EAX]
00 001	DS:[ECX]
00 010	SS:[EDX]
00 011	SS:[EBX]
00 100	s-i-b is present
00 101	DS:[d32]
00 110	DS:[ESI]
00 111	DS:[EDI]
01 000	DS:[EAX + d8]
01 001	DS:[ECX + d8]
01 010	SS:[EDX + d8]
01 011	SS:[EBX + d8]
01 100	s-i-b is present
01 101	SS:[EBP + d8]
01 110	DS:[ESI + d8]
01 111	DS:[EDI + d8]

mod r/m	Effective Address
10 000	DS:[EAX + d32]
10 001	DS:[ECX + d32]
10 010	SS:[EDX + d32]
10 011	SS:[EBX + d32]
10 100	s-i-b is present
10 101	SS:[EBP + d32]
10 110	SS:[ESI + d32]
10 111	DS:[EDI + d32]
11 000	register - see tables below
11 001	register - see tables below
11 010	register - see tables below
11 011	register - see tables below
11 100	register - see tables below
11 101	register - see tables below
11 110	register - see tables below
11 111	register - see tables below

Register Specified by r/m During 16-bit Data Operations		
mod r/m	Function of w Field	
	(when w = 0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

Register Specified by r/m During 32-bit Data Operations		
mod r/m	Function of w Field	
	(when w = 0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI

Table E-9. Encoding of 32-bit Address Mode (“mod r/m” Byte and s-i-b Byte Present)

mod r/m	Effective Address
00 000	DS:[EAX + (scaled index)]
00 001	DS:[ECX + (scaled index)]
00 010	DS:[EDX + (scaled index)]
00 011	DS:[EBX + (scaled index)]
00 100	SS:[ESP + (scaled index)]
00 101	DS:[d32 + (scaled index)]
00 110	DS:[ESI + (scaled index)]
00 111	DS:[EDI + (scaled index)]
01 000	DS:[EAX + (scaled index) + d8]
01 001	DS:[ECX + (scaled index) + d8]
01 010	DS:[EDX + (scaled index) + d8]
01 011	DS:[EBX + (scaled index) + d8]
01 100	SS:[ESP + (scaled index) + d8]
01 101	SS:[EBP + (scaled index) + d8]
01 110	DS:[ESI + (scaled index) + d8]
01 111	DS:[EDI + (scaled index) + d8]
10 000	DS:[EAX + (scaled index) + d32]
10 001	DS:[ECX + (scaled index) + d32]
10 010	DS:[EDX + (scaled index) + d32]
10 011	DS:[EBX + (scaled index) + d32]
10 100	SS:[ESP + (scaled index) + d32]
10 101	SS:[EBP + (scaled index) + d32]
10 110	DS:[ESI + (scaled index) + d32]
10 111	DS:[EDI + (scaled index) + d32]

NOTE: Mod field in “mod r/m” byte; ss, index, base fields in “s-i-b” byte.

ss	Scale Factor
00	x1
01	x2
10	x4
11	x8

Index	Index Register
000	EAX
001	ECX
010	EDX
011	EBX
100	no index reg†
101	EBP
110	ESI
111	EDI

† When index field is 100, indicating “no index register,” the ss field must equal 00. If this is not true, the effective address is undefined

E.2.2.5 Encoding of Operation Direction (d) Field

In many two-operand instructions the d field is present to indicate which operand is considered the source and which is the destination.

Table E-10. Encoding of Operation Direction (d) Field

d	Direction of Operation
0	Register/Memory<--Register "reg" field indicates source operand; "mod r/m" or "mod ss index base" indicates destination operand.
1	Register<--Register/Memory "reg" field indicates destination operand; "mod r/m" or "mod ss index base" indicates source operand.

E.2.2.6 Encoding of Sign-Extend (s) Field

The s field occurs primarily to instructions with immediate data fields. The s field has an effect only if the size of the immediate data is 8 bits and is being placed in a 16-bit or 32-bit destination.

Table E-11. Encoding of Sign-Extend (s) Field

s	Effect on Immediate Data8	Effect on Immediate Data 16|32
0	None	None
1	Sign-Extend Data8 to fill 16-bit or 32-bit destination	None

E.2.2.7 Encoding of Conditional Test (tttn) Field

For the conditional instructions (conditional jumps and set on condition), tttn is encoded with n indicating to use the condition (n=0) or its negation (n=1), and ttt giving the condition to test.

Table E-12. Encoding of Conditional Test (tttn) Field

Mnemonic	Condition	tttn
O	Overflow	0000
NO	No Overflow	0001
B/NAE	Below/Not Above or Equal	0010
NB/AE	Not Below/Above or Equal	0011
E/Z	Equal/Zero	0100
NE/NZ	Not Equal/Not Zero	0101
BE/NA	Below or Equal/Not Above	0110
NBE/A	Not Below or Equal/Above	0111
S	Sign	1000
NS	Not Sign	1001
P/PE	Parity/Parity Even	1010
NP/PO	Not Parity/Parity Odd	1011
L/NGE	Less Than/Not Greater or Equal	1100
NL/GE	Not Less Than/Greater or Equal	1101
LE/NG	Less Than or Equal/Greater Than	1110
NLE/G	Not Less or Equal/Greater Than	1111

E.2.2.8 Encoding of Control or Debug or Test Register (eee) Field

For the loading and storing of the Control, Debug and Test registers.

Table E-13. When Interpreted as Control Register Field

eee Code	Reg Name
000	CR0
010	CR2
011	CR3

NOTE: Do not use any other encoding

Table E-14. When Interpreted as Debug Register Field

eee Code	Reg Name
000	DR0
001	DR1
010	DR2
011	DR3
110	DR6
111	DR7

NOTE: Do not use any other encoding

Table E-15. When Interpreted as Test Register Field

eee Code	Reg Name
110	TR6
111	TR7

NOTE: Do not use any other encoding

GLOSSARY

GLOSSARY

This glossary defines acronyms, abbreviations, and terms that have special meaning in this manual. (Chapter 1, GUIDE TO THIS MANUAL, discusses notational conventions.)

Assert — The act of making a signal active (enabled). The polarity (high/low) is defined by the signal name. Active-low signals are designated by a pound symbol (#) suffix; active-high signals have no suffix. To *assert* RD# is to drive it low; to assert HLDA is to drive it high.

BIOS — Basic input/output system. The interface between the hardware and the operating system.

BIU — Bus interface unit. The internal peripheral that controls the external bus.

Boundary-scan — The term *boundary-scan* refers to the ability to scan (observe) the signals at the boundary (the pins) of a device. A major component of the *JTAG* standard.

CSU — Chip-select unit. The internal peripheral that selects an external memory device during an external bus cycle.

Clear — The term *clear* refers to the value of a bit or the act of giving it a value. If a bit is clear, its value is “0”; clearing a bit gives it a “0” value.

Deassert — The act of making a signal inactive (disabled). The polarity (high/low) is defined by the signal name. Active-low signals are designated by a pound symbol (#) suffix; active-high signals have no suffix. To deassert RD# is to drive it high; to deassert HLDA is to drive it low.

DMA — Direct memory access controller. The internal peripheral that allows external or internal peripherals to transfer information directly to or from the system. The two-channel DMA controller is an enhanced version of the industry-standard 8237A DMA peripheral.

DOS Address Space — Addresses 0H–03FFH. The internal timers, interrupt controller, serial I/O ports, and DMA controller can

	be mapped into this space. In this manual, the terms *DOS address* and *PC/AT address* are synonymous.
DOS-compatible Mode	The addressing mode in which the internal timer, interrupt controller, serial I/O ports, and DMA controller are mapped into the DOS address space. This mode decodes only the lower 10 address bits, so the *expanded address space* is inaccessible.
Edge-triggered	The mode in which the interrupt controller recognizes a rising edge (low-to-high transition) on an interrupt request signal as an interrupt request. The internal peripherals use edge-triggered interrupt requests; this is compatible with the PC/AT bus specification. External peripherals can use either edge-triggered or *level-sensitive* interrupt requests.
Enhanced DOS Mode	The addressing mode in which the internal timer, interrupt controller, serial I/O ports, and DMA controller are mapped into both the *DOS address space* and the *expanded address space*. This mode decodes all 16 address bits. All internal peripherals can be accessed in the expanded address space; the internal timer, interrupt controller, serial I/O ports, and DMA controller can also be accessed in the DOS address space.
Expanded Address Space	Addresses 0F000H–0F8FFH. All internal peripheral registers reside in this space. The internal timer, interrupt controller, serial I/O ports, and DMA controller can also be mapped into DOS (or PC/AT) address space.
ICU	Interrupt control unit. The internal peripheral that receives interrupt requests from internal peripherals and external pins, resolves priority, and presents the requests to the CPU. The ICU is functionally identical to two industry-standard 82C59A programmable interrupt controllers connected in cascade.
Idle Mode	The power conservation mode that freezes the core clocks but leaves the peripheral clocks running.
Interrupt Latency	The delay between the time that the master 82C59A presents an interrupt request to the CPU and the time that the interrupt acknowledge cycle begins.

Interrupt Response Time — The amount of time required to complete an interrupt acknowledge cycle and transfer program control to the interrupt service routine.

Interrupt Resolution — The delay between the time that the interrupt controller receives an interrupt request and the time that the master 82C59A presents the request to the CPU.

ISR — Interrupt service routine. A user-supplied software routine designed to service specific interrupt requests.

JTAG — Joint Test Action Group. The IEEE technical subcommittee that developed the testability standard published as Standard 1149.1-1990, *IEEE Standard Test Access Port and Boundary-Scan Architecture*, and its supplement, Standard 1149.1a-1993. The test-logic unit is fully compliant with this standard.

Level-sensitive — The mode in which the interrupt controller recognizes a high level (logic one) on an interrupt request signal as an interrupt request. Unlike an *edge-triggered* interrupt request, a level-sensitive interrupt request will continue to generate interrupts as long as it is asserted.

LSB — Least-significant bit of a byte or least-significant byte of a word.

NonDOS Mode — The addressing mode in which the internal timer, interrupt controller, serial I/O ports, and DMA controller are mapped into the *expanded address space*. This mode decodes all 16 address bits. All internal peripherals can be accessed only in the expanded address space.

Nonintrusive DOS Mode — The addressing mode in which the internal timer, interrupt controller, serial I/O ports, and DMA controller can be individually mapped out of the *DOS address space* and replaced by the corresponding external peripherals. This mode decodes only the lower 10 address bits, so the *expanded address space* is inaccessible.

Normally not-ready — The term *normally not-ready* refers to a system in which a bus cycle continues until the accessed device asserts READY#.

PC/AT Address Space — Addresses 0H–03FFH. The internal timers, interrupt controller, serial I/O ports, and DMA controller can be mapped into this space. In this manual, the terms *DOS address* and *PC/AT address* are synonymous.

Pipelining — A bus interface technique that controls the address and status outputs so the outputs for the next bus cycle become valid before the end of the current bus cycle, allowing external bus cycles to overlap. By increasing the amount of time available for external memory or I/O devices to respond, pipelining allows systems to achieve high bandwidth with relatively slow, inexpensive components.

Powerdown Mode — The power conservation mode that freezes both the core clocks and the peripheral clocks.

RCU — Refresh control unit. The module that simplifies the interface between the processor and DRAM components by providing the necessary bus control and timing for refresh operations.

Reserved Bits — Register bits that are not used in this device but may be used in future implementations. Avoid any software dependence on these bits.

Set — The term *set* refers to the value of a bit or the act of giving it a value. If a bit is set, its value is "1"; setting a bit gives it a "1" value.

SIO Unit — Serial input/output unit. The internal peripheral that allows the system to communicate with external peripheral devices and modems.

SMM — System management mode. The hardware and software enhancement that reduces system power consumption by allowing the device to execute specific routines for power management.

SMRAM — A 32-Kbyte memory partition (38000H–3FFFFH) used for *SMM*. The upper 512 bytes (3FE00H–3FFFFH) are reserved for the CPU and must reside in RAM; the remainder of the partition is used for user-supplied driver code and may reside in read-only storage.

SSIO Unit — Synchronous serial input/output unit. The internal peripheral that provides 16-bit bidirectional serial

	communications. The transmitter and receiver can operate independently (with different clocks) to provide full-duplex communication.
State Time (or State)	The basic time unit of the device; the combined period of the two internal timing signals, PH1 and PH2. With a 50 MHz external clock, one state time equals 80 ns. Because the device can operate at many frequencies, this manual defines time requirements in terms of *state times* rather than in specific units of time.
TAP	Test access port. The dedicated input and output pins through which a tester communicates with the *test-logic unit*. A major component of the *JTAG* standard.
TCU	Timer/counter unit. The internal peripheral that provides three independent 16-bit down-counters.
Test-logic Unit	The module that facilitates testing of the device logic and interconnections between the device and the board. This module is fully compliant with IEEE Standard 1149.1, commonly called the *JTAG* standard.
UART	Universal asynchronous receiver and transmitter. A part of the *SIO unit*.
WDT	Watchdog timer. An internal, 32-bit down-counter that can operate as a general-purpose timer, a software watchdog timer, or a bus monitor.

Index

INDEX

C

D

E

F

H

I

R

S

T