8XC196EA Microcontroller User's Manual

December 1998

CONTENTS

CHAPTER 3
PROGRAMMING CONSIDERATIONS

CHAPTER 4
MEMORY PARTITIONS

CHAPTER 15
INTERFACING WITH EXTERNAL MEMORY

CHAPTER 16
SERIAL DEBUG UNIT

FIGURES

FIGURES

FIGURES

FIGURES

FIGURES

FIGURES

TABLES

TABLES

TABLES

1

Guide to This Manual

CHAPTER 1
GUIDE TO THIS MANUAL

This manual describes the 8XC196EA embedded microcontroller. It is intended for use by both software and hardware designers familiar with the principles of microcontrollers. This chapter describes what you'll find in this manual, lists other documents that may be useful, and explains how to access the support services we provide to help you complete your design.

1.1 MANUAL CONTENTS

This manual contains several chapters and appendixes, a glossary, and an index. This chapter, Chapter 1, provides an overview of the manual. This section summarizes the contents of the remaining chapters and appendixes. The remainder of this chapter describes notational conventions and terminology used throughout the manual, provides references to related documentation, describes customer support services, and explains how to access information and assistance.

Chapter 2 — Architectural Overview provides an overview of the device hardware. It describes the core, internal timing, internal peripherals, and special operating modes. It also describes the chip configuration bytes (CCBs) and the chip configuration registers (CCRs), which control many aspects of the microcontroller's operation.

Chapter 3 — Programming Considerations — provides an overview of the instruction set, describes general standards and conventions, and defines the operand types and addressing modes supported by the MCS® 96 microcontroller family. (For additional information about the instruction set, see Appendix A.)

Chapter 4 — Memory Partitions — describes the address space of the device. It describes the address partitions, explains how to use windows to increase the amount of memory that can be accessed with direct addressing, and lists all special-function registers (SFRs) with their addresses, and describes options for ROM protection and addressing.

Chapter 5 — Stack Overflow Module — describes the stack overflow module, the 8XC196EA peripheral that monitors the value of the stack pointer (SP) and generates a nonmaskable interrupt request if the value is outside specified boundaries.

Chapter 6 — Standard and PTS Interrupts — describes the interrupt control circuitry, priority scheme, and timing for standard and peripheral transaction server (PTS) interrupts. It also explains interrupt programming and control.

Chapter 7 — I/O Ports — describes the input/output ports and explains how to configure the pins for general-purpose input/output or for special functions.

Chapter 8 — Serial I/O (SIO) Port — describes the asynchronous/synchronous serial I/O (SIO) port and explains how to program it.

Chapter 9 — Synchronous Serial I/O (SSIO) Port — describes the synchronous serial I/O (SSIO) port and explains how to program it.

Chapter 10 — Pulse-width Modulator — provides a functional overview of the pulse width modulator (PWM) modules, describes how to program them, and provides sample circuitry for converting the PWM outputs to analog signals.

Chapter 11 — Event Processor Array (EPA) — describes the event processor array, a timer/counter-based, high-speed input/output unit. It describes the timer/counters and explains how to program the EPA and how to use the EPA to produce pulse-width modulated (PWM) outputs.

Chapter 12 — Analog-to-digital (A/D) Converter — provides an overview of the analog-to-digital (A/D) converter and describes how to program the converter, read the conversion results, and interface with external circuitry.

Chapter 13 — Minimum Hardware Considerations — describes options for providing the basic requirements for device operation within a system, discusses other hardware considerations, and describes device reset options.

Chapter 14 — Special Operating Modes — provides an overview of the idle, powerdown, and on-circuit emulation (ONCE) modes and describes how to enter and exit each mode.

Chapter 15 — Interfacing with External Memory — lists the signals and registers used for interfacing to external devices. It discusses the bus width and memory configurations, the bus-hold protocol, write-control modes, and internal wait states and ready control. Finally, it provides timing information for the system bus.

Chapter 16 — Serial Debug Unit — describes the module that allows read and write access to the 3-Kbyte code/data RAM to aid in code development and debugging.

Chapter 17 — Using the Test-ROM Routines — describes the three test-ROM routines. It provides recommended circuits and the corresponding memory maps, and it defines the reduced instruction set monitor (RISM) commands.

Appendix A — Instruction Set Reference — provides reference information for the instruction set. It describes each instruction; defines the processor status word (PSW) flags; shows the relationships between instructions and PSW flags; and lists hexadecimal opcodes, instruction lengths, and execution times. (For additional information about the instruction set, see Chapter 3, "Programming Considerations.")

Appendix B — Signal Descriptions — provides reference information for the device pins, including descriptions of the pin functions, reset status of the I/O and control pins, and package pin assignments.

Appendix C — Registers — provides a compilation of all special-function registers (SFRs), arranged alphabetically by register mnemonic. It also includes tables that list the windowed direct addresses for all SFRs in each possible window.

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

Index — lists key topics with page number references.

1.2 NOTATIONAL CONVENTIONS AND TERMINOLOGY

The following notations and terminology are used throughout this manual. The Glossary defines other terms with special meanings.

— The pound symbol (#) has either of two meanings, depending on the context. When used with a signal name, the symbol means that the signal is active low. When used in an instruction, the symbol prefixes an immediate value in immediate addressing mode.

addresses — In this manual, both internal and external addresses use the number of hexadecimal digits that correspond with the number of available address bits.

When writing code, use the appropriate address conventions for the software tool you are using. (In general, assemblers require a zero preceding an alphabetic hexadecimal character and an "H" following any hexadecimal value, so FFFFFFH must be written as 0FFFFFFH. ANSI 'C' compilers require a zero plus an "x" preceding a hexadecimal value, so FFFFFFH must be written as 0xFFFFFF.) Consult the manual for your assembler or compiler to determine its specific requirements.

assert and deassert The terms *assert* and *deassert* refer to the act of making a signal active (enabled) and inactive (disabled), respectively. The active polarity (low or high) 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 ALE is to drive it high; to deassert RD# is to drive it high; to deassert ALE is to drive it low.

clear and set The terms *clear* and *set* refer 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. If a bit is set, its value is "1"; setting a bit gives it a "1" value.

f Lowercase "f" represents the internal operating frequency. See "Internal Timing" on page 2-9 for details.

instructions Instruction mnemonics are shown in upper case to avoid confusion. In general, you may use either upper case or lower case when programming. Consult the manual for your assembler or compiler to determine its specific requirements.

italics Italics identify variables and introduce new terminology. The context in which italics are used distinguishes between the two possible meanings.

Variables in registers and signal names are commonly represented by *x* and *y*, where *x* represents the first variable and *y* represents the second variable. For example, in register P*x*_MODE.*y*, *x* represents the variable that identifies the specific port associated with the register, and *y* represents the register bit variable (7:0 or 15:0). Variables must be replaced with the correct values when configuring or programming registers or identifying signals.

numbers Hexadecimal numbers are represented by a string of hexadecimal digits followed by the character *H*. 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 appended to binary numbers for clarity.)

register bits Bit locations are indexed by 7:0 (or 15:0), where bit 0 is the least-significant bit and bit 7 (or 15) is the most-significant bit. An individual bit is represented by the register name, followed by a period and the bit number. For example, WSR.7 is bit 7 of the window selection register. In some discussions, bit names are used.

register names Register mnemonics are shown in upper case. For example, TIMER2 is the timer 2 register; timer 2 is the timer. A register name containing a lowercase italic character represents more than one register. For example, the *x* in P*x*_REG indicates that the register name refers to any of the port data registers.

reserved bits Certain bits are described as *reserved* bits. In illustrations, reserved bits are indicated with a dash (—). These bits are not used in this device, but they may be used in future implementations. To help ensure that a current software design is compatible with future implementations, reserved bits should be cleared (given a value of "0") or left in their default states, unless otherwise noted. Do not rely on the values of reserved bits; consider them undefined.

reserved registers Certain special-function register (SFR) locations are described as *reserved*. These locations are not used in this microcontroller, but they may be used in future implementations. To help ensure that a current software design is compatible with future implementations, do not use these locations.

Some special-function register (SFR) locations are described as *reserved for Intel*. These locations are reserved for Intel's testing. Using these locations can cause unpredictable or undesirable results.

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. For example, the EPA signals are named EPA0, EPA1, EPA2, etc. Port pins are represented by the port abbreviation, a period, and the pin number (e.g., P2.0, P2.1); a range of pins is represented by P*x.y*:*z* (e.g., P2.4:0 represents five port pins: P2.4, P2.3, P2.2, P2.1, P2.0). A pound symbol (#) appended to a signal name identifies an active-low signal.

t Lowercase "t" represents the internal operating period. See "Internal Timing" on page 2-9 for details.

units of measure The following abbreviations are used to represent units of measure:

A	amps, amperes
DCV	direct current volts
Kbytes	kilobytes
kHz	kilohertz
kΩ	kilo-ohms
mA	milliamps, milliamperes
Mbytes	megabytes
MHz	megahertz
ms	milliseconds
mW	milliwatts
ns	nanoseconds
pF	picofarads
W	watts
V	volts
µA	microamps, microamperes
µF	microfarads
µs	microseconds
µW	microwatts

x Lowercase *x* (italic) represents a variable. Variables must be replaced with the correct values when configuring or programming registers or identifying signals.

X Uppercase X (no italics) represents an unknown value or an irrelevant ("don't care") state or condition. The value may be either binary or hexadecimal, depending on the context. For example, 2XAFH (hex) indicates that bits 11:8 are unknown; 10XXB (binary) indicates that the two least-significant bits are unknown.

y Lowercase *y* (italic) represents a variable. Variables must be replaced with the correct values when configuring or programming registers or identifying signals.

1.3 RELATED DOCUMENTS

The tables in this section list additional documents that you may find useful in designing systems incorporating MCS 96 microcontrollers. These are not comprehensive lists, but are a representative sample of relevant documents. For a complete list of available printed documents, please order the literature catalog (order number 210621). To order documents, please call the Intel literature center for your area (see Table 1-4 on page 1-8).

Table 1-1. Handbooks and Product Information

Title and Description	Order Number
Flash Memory Databook (2 volume set) A collection of datasheets and application notes devoted to techniques and information to help design semiconductor memory into an application or system.	210830
Packaging Databook Detailed information on the manufacturing, applications, and attributes of a variety of semiconductor packages.	240800
Embedded Microcontrollers Handbook Datasheets and architecture descriptions for Intel's MCS® 48, MCS 51, and MCS 96 microcontrollers. Documents in this handbook discuss hardware and software implementations and present helpful design techniques.	270646
MCS® Tools Handbook Information on third-party hardware and software tools that support Intel's embedded microcontrollers.	272326
Platform Components 1998 Comprehensive information on Intel's peripheral components, including datasheets, application notes, and technical briefs.	296467
MCS® 96 Microcontroller Product Selection Guide A table containing bus speed, internal memory, and other functional and physical information for all MCS microcontrollers.	270839

Table 1-2. Application Notes

Title	Order Number
AP-125, *Designing Microcontroller Systems for Electrically Noisy Environments*	210313
AP-155, *Oscillators for Microcontrollers*	230659
AP-406, *MCS® 96 Analog Acquisition Primer*	270365
AP-445, *8XC196KR Peripherals: A User's Point of View*	270873
AP-449, *A Comparison of the Event Processor Array (EPA) and High Speed Input/Output (HSIO) Unit*	270968
AP-700, *Intel Fuzzy Logic Tool Simplifies ABS Design*	272595
AP-711, *EMI Design Techniques for Microcontrollers in Automotive Applications*	272637
AP-715, *Interfacing an I^2C Serial EEPROM to an MCS® 96 Microcontroller*	272680
Interfacing a 20 MHz 8xC196 to an 82527 Serial Communications Controller	272732
Using the 4th Entry Interrupt in ABS Designs	270692

Table 1-3. MCS® 96 Microcontroller Datasheets (Automotive)

Title and Description	Order Number
83C196EA Automotive CHMOS 16-Bit Microcontroller	272788
80C196EA Commercial CHMOS 16-Bit Microcontroller	273153
83C196LD CHMOS 16-BIT Microcontroller	272805
87C196CB 20MHz Advanced 16-Bit CHMOS Microcontroller with Integrated CAN 2.0	272405
87C196JT 20 MHz Advanced 16-Bit CHMOS Microcontroller	272529
87C196JV 20 MHz Advanced 16-Bit CHMOS Microcontroller	272580
87C196KC 16-Bit High-Performance CHMOS Microcontroller	270846
87C196KD 16-Bit High-Performance CHMOS Microcontroller	272168
87C196KR, 87C196JV, 87C196JT, 87C196JR, and 87C196CA Advanced 16-Bit CHMOS Microcontrollers	270827
87C196KT Advanced 16-Bit CHMOS Microcontroller	273178
87C196LA CHMOS 16-Bit Microcontroller	272806
87C196KB Advanced 16-Bit CHMOS Microcontroller ROMless or ROM	270679

1.4 APPLICATION SUPPORT SERVICES

Table 1-4. Intel Application Support Services

Service	U.S. and Canada	Asia-Pacific and Japan	Europe
World Wide Web	http://www.intel.com/	http://www.intel.com/	http://www.intel.com/
Help Desk	800-628-8686 916-356-7999	Please contact your local distributor.	Please contact your local distributor.
Literature	800-548-4725	708-296-9333 +81(0)120 47 88 32	+44(0)1793-431155 England +44(0)1793-421777 France +44(0)1793-421333 Germany

2

Architectural Overview

CHAPTER 2
ARCHITECTURAL OVERVIEW

The 16-bit 8XC196EA CHMOS microcontroller is designed to handle high-speed calculations and fast input/output (I/O) operations. In addition to its 16-bit external bus, the 8XC196EA has an extended addressing port that provides 5 external address pins, for a total of 21 address pins. With 21 address pins, this microcontroller can access up to 2 Mbytes of linear address space. The 8XC196EA also has a chip-select unit that provides a glueless interface to external memory devices. The extended addressing port and chip-select unit enable the 8XC196EA microcontroller to handle larger, more complex programs and to access more external memory at a faster rate than could earlier MCS® 96 microcontrollers.

2.1 TYPICAL APPLICATIONS

MCS 96 microcontrollers are typically used for high-speed event control systems. Commercial applications include modems, motor-control systems, printers, photocopiers, air conditioner control systems, disk drives, and medical instruments. Automotive customers use MCS 96 microcontrollers in engine-control systems, airbags, suspension systems, and antilock braking systems (ABS). The 8XC196EA is especially well suited to applications that benefit from its processing speed and enhanced peripheral set, such as powertrain control.

2.2 MICROCONTROLLER FEATURES

The 8XC196EA is the first member of a new family of MCS 96 microcontrollers incorporating new features for automotive applications such as powertrain control. It has 2 Mbytes of linear address space, providing more space for high-level language compilation than did earlier MCS 96 microcontrollers. It also has three individually programmable chip-select signals and an external bus that can dynamically switch between multiplexed and demultiplexed operation, making it easier to design low-cost memory solutions.

The 8XC196EA has an optional clock multiplier, allowing it to operate at 40 MHz with a less expensive, 20 MHz external clock source. It also has a programmable clock output, allowing you to select one of five output frequencies. The 8XC196EA incorporates a serial debug unit (SDU) that allows read and write access to the internal code/data RAM, aiding in code development and debugging.

In addition to the interrupt controller and peripheral transaction server (PTS), the 8XC196EA incorporates two peripheral interrupt handlers (PIHs). Each PIH handles up to 16 interrupt requests from the event processor array (EPA). With the addition of the PIHs to the interrupt structure, the 8XC196EA can support 44 interrupt sources.

The 8XC196EA has both new and enhanced peripherals. The peripherals that are new to automotive MCS 96 products are a stack overflow module (SOM) and four pulse-width modulators (PWMs) with two channels each. The stack overflow module monitors the stack pointer and causes a nonmaskable interrupt if the stack pointer crosses upper or lower boundaries you define, assisting in code development. Each of the four pulse-width modulators (PWMs) consists of an

adjacent pair of PWM channels that can generate two output signals with a fixed, programmable frequency and a variable duty cycle. These outputs can be used to drive motors that require an unfiltered PWM waveform for optimal efficiency, or they can be filtered to produce a smooth analog signal.

The synchronous serial input/output (SSIO) port, event processor array (EPA), and analog-to-digital (A/D) converter have been enhanced for the 8XC196EA.

- The SSIO port provides one bidirectional communication channel or two unidirectional channels. The SSIO is compatible with most protocols because the serial clock is completely configurable. Paired, the SSIO channels can operate in a channel-select mode, allowing for communication in multiple-master systems without additional external hardware.
- The EPA has 4 timer/counters, 17 high-speed capture/compare channels, and 8 *output/simulcapture* channels. The output/simulcapture channels are output-only channels that simultaneously capture the value of any other timer upon a compare, providing easy conversion between angle and time domains. A pair of timer/counters (timer 1–2 or timer 3–4) can be concatenated to provide a 32-bit timer/counter.
- The A/D converter has sixteen 10-bit channels, a dedicated result register for each channel, and an automatic scan mode that operates without CPU intervention.

Table 2-1 lists the features of the 83C196EA, and Figure 2-1 shows a detailed block diagram.

Table 2-1. Features of the 83C196EA

Pins	ROM	Register RAM (Note 1)	Code/Data RAM	I/O Pins (Note 2)	EPA Pins	SIO/ SSIO Ports	A/D Chnl	PWM Pins	Chip-select Pins	External Interrupt Pins
160	8 Kbytes	1 Kbyte	3 Kbytes	132	25	4	16	8	3	1

NOTES:
1. Register RAM amount includes the 24 bytes allocated to core special-function registers (SFRs) and the stack pointer.
2. I/O pins include address, data, and bus control pins and 83 I/O port pins.

Table 2-2 lists the features of the 80C196EA, and Figure 2-2 shows a detailed block diagram.

Table 2-2. Features of the 80C196EA

Pins	ROM	Register RAM (Note 1)	Code/Data RAM	I/O Pins (Note 2)	EPA Pins	SIO/ SSIO Ports	A/D Chnl	PWM Pins	Chip-select Pins	External Interrupt Pins
160	None	1 Kbyte	3 Kbytes	132	25	4	16	8	3	1

NOTES:
1. Register RAM amount includes the 24 bytes allocated to core special-function registers (SFRs) and the stack pointer.
2. I/O pins include address, data, and bus control pins and 83 I/O port pins.

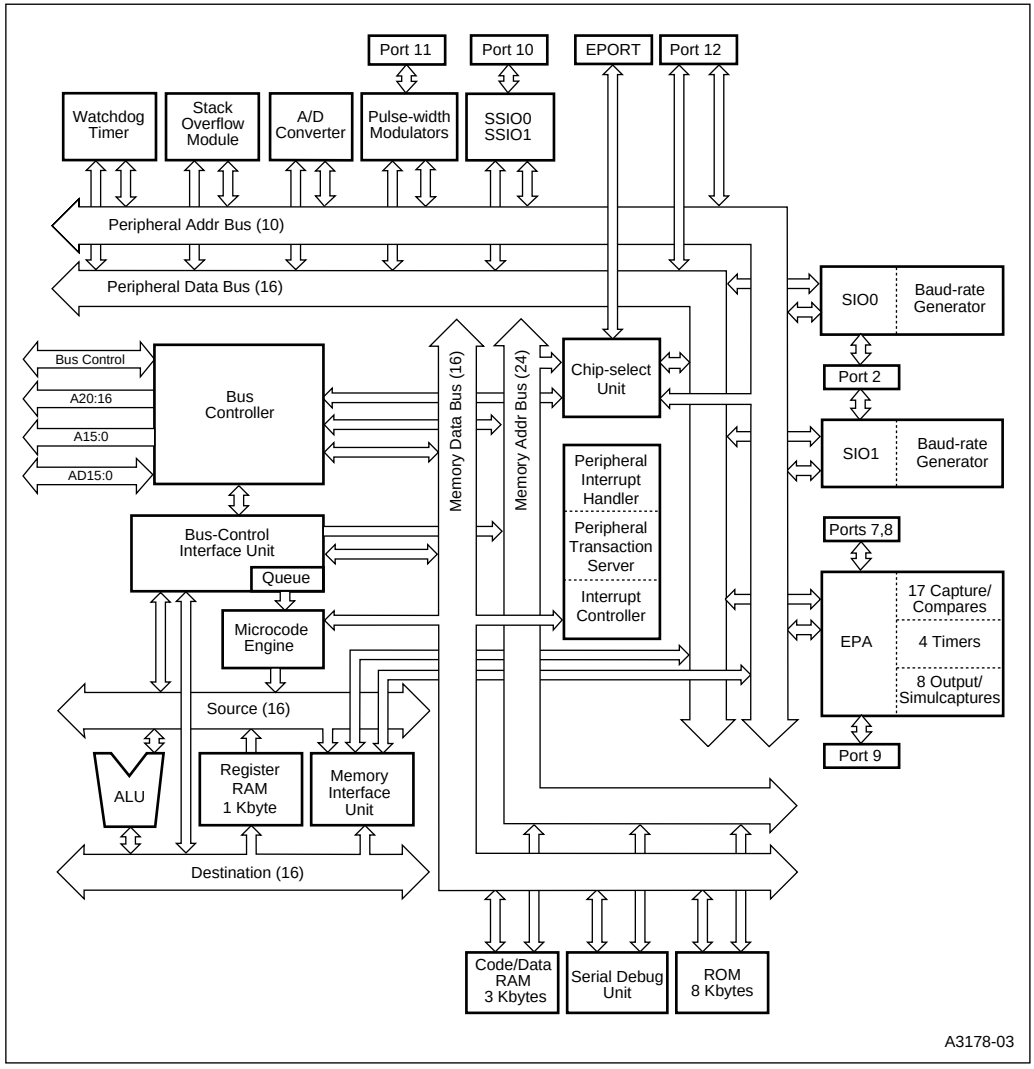


Figure 2-1. 83C196EA Detailed Block Diagram

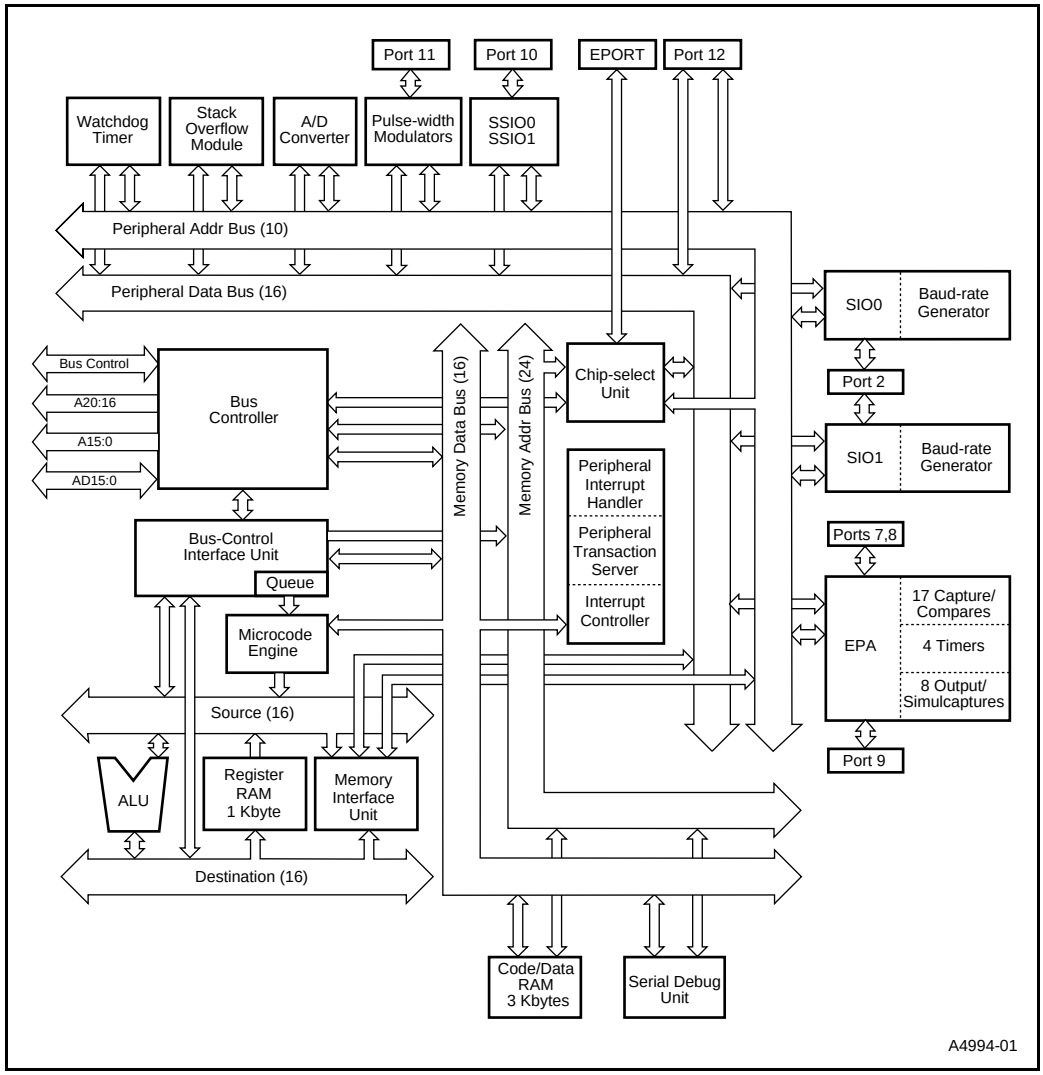


Figure 2-2. 80C196EA Detailed Block Diagram

2.3 FUNCTIONAL OVERVIEW

Figure 2-3 is a simplified block diagram that shows the major blocks within the microcontroller. The remainder of this section describes those blocks.

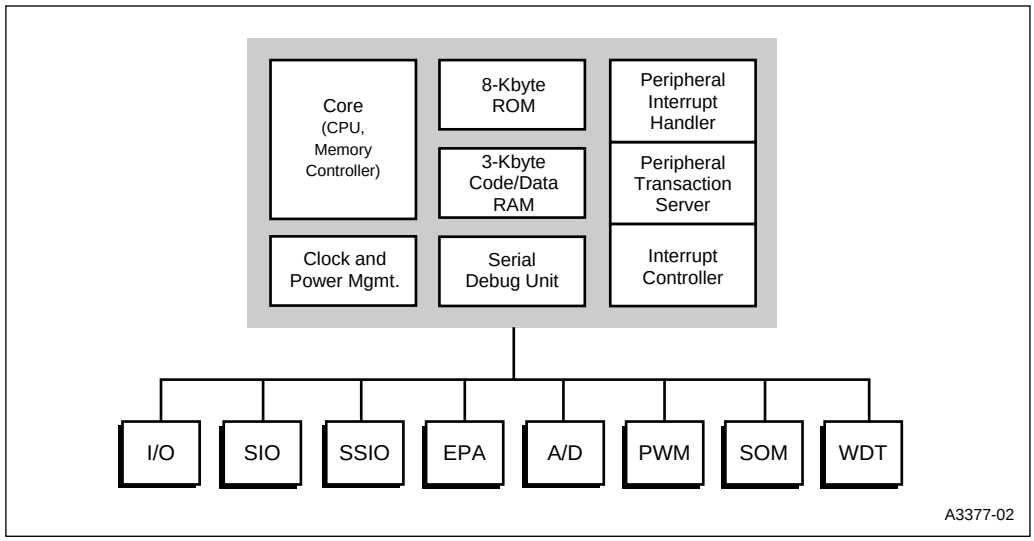


Figure 2-3. 83C196EA Simplified Block Diagram

2.3.1 Core

The core of the microcontroller (Figure 2-4) consists of the central processing unit (CPU) and memory controller. A 16-bit internal bus connects the CPU to both the memory controller and the interrupt controller. An extension of this bus connects the CPU to the internal peripheral modules. In addition, an 8-bit internal bus transfers instruction bytes from the memory controller to the instruction register in the RALU.

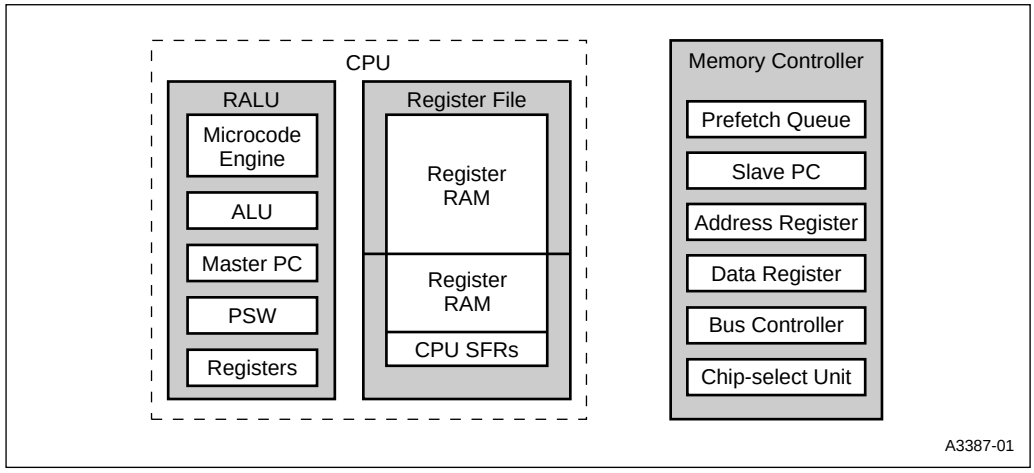


Figure 2-4. Block Diagram of the Core

2.3.1.1 CPU

The CPU contains the register arithmetic-logic unit (RALU) and the register file. CPU instructions move from the 8-byte prefetch queue in the memory controller into the RALU's instruction register. The microcode engine decodes the instructions and then generates the sequence of events that cause desired functions to occur.

2.3.1.2 Register Arithmetic-logic Unit (RALU)

The RALU contains the microcode engine, the 16-bit arithmetic logic unit (ALU), the master program counter (PC), the processor status word (PSW), and several registers. The microcode engine instructs the RALU to perform operations using bytes, words, or double-words from the register file.

The 24-bit master program counter (PC) provides a linear, nonsegmented 16-Mbyte memory space. (For compatibility with earlier microcontrollers, the PC can be configured as 16 bits wide.) Only 21 of the internal address bits are implemented with external pins, so you can physically address only 2 Mbytes. The master PC contains the address of the next instruction and has a built-in incrementer that automatically loads the next sequential address. However, if a jump, interrupt, call, or return changes the address sequence, the ALU loads the correct address into the master PC.

The processor status word (PSW) contains one bit (PSW.1) that globally enables or disables servicing of all maskable interrupts, one bit (PSW.2) that enables or disables the peripheral transaction server (PTS), and six Boolean flags that reflect the state of your program. See Table A-2 on page A-4 for a description of the status flags.)

The registers in the RALU are the instruction register, a constants register, a bit-select register, a loop counter, and three temporary registers (the upper-word, lower-word, and second-operand registers). The bit-select register is 3 bits, and the loop counter is 6 bits. All other registers in the

RALU are either 16 or 17 bits (16 bits plus a sign extension). Some of these registers can reduce the ALU's workload by performing simple operations.

CPU instructions move into the instruction register from the 8-byte prefetch queue in the memory controller. The microcode engine decodes the instructions and instructs the RALU to perform the required operations.

The RALU speeds up calculations by storing constants (e.g., 0, 1, and 2) in the constants register so that they are readily available when complementing, incrementing, or decrementing bytes or words. Based on the bit-select register, the constants register generates single-bit masks for bit-test instructions. The six-bit loop counter counts repetitive shifts.

The RALU uses the upper- and lower-word registers together for the 32-bit instructions and as temporary registers for many instructions. These registers have their own shift logic and are used for operations that require logical shifts, including normalize, multiply, and divide operations. The second-operand register stores the second operand for two-operand instructions, including the multiplier during multiply operations and the divisor during divide operations. During subtraction operations, the output of this register is complemented before it is moved into the ALU.

The RALU performs most calculations for the microcontroller, but it does not use an *accumulator*. Instead it operates directly on the lower register file, which essentially provides 256 accumulators. Because data does not flow through a single accumulator, the microcontroller's code executes faster and more efficiently than, for example, an 80C186 microprocessor.

MCS 96 microcontrollers combine a large set of general-purpose registers with a three-operand instruction format. This format allows a single instruction to specify two source registers and a separate destination register. For example, the following instruction multiplies two 16-bit variables and stores the 32-bit result in a third variable.

```
MUL  RESULT, FACTOR_1, FACTOR_2    ;multiply FACTOR_1 and FACTOR_2
                                   ;and store answer in RESULT
                                   ;(RESULT)←(FACTOR_1 × FACTOR_2)
```

An 80C186 microprocessor requires four instructions to accomplish the same operation. The following example shows the equivalent code for an 80C186 microprocessor:

```
MOV  AX, FACTOR_1                  ;move FACTOR_1 into accumulator (AX)
                                   ;(AX)←FACTOR1
MUL  FACTOR_2                      ;multiply FACTOR_2 and AX
                                   ;(DX:AX)←(AX)×(FACTOR_2)
MOV  RESULT, AX                    ;move lower byte into RESULT
                                   ;(RESULT)←(AX)
MOV  RESULT+2, DX                  ;move upper byte into RESULT+2
                                   ;(RESULT+2)←(DX)
```

2.3.1.3 Register File

The register file is divided into an upper and a lower file. In the lower register file, the lowest 24-byte section is allocated to the CPU's special-function registers (SFRs) and the stack pointer, while the remainder is available as general-purpose register RAM. The upper register file contains only general-purpose register RAM. The register RAM can be accessed as bytes, words, or double-words.

The RALU accesses the upper and lower register files differently. The lower register file is always directly accessible with direct addressing (see "Programming Considerations" on page 3-1). The upper register file is accessible with direct addressing only when *windowing* is enabled. Otherwise, the upper register file is accessed indirectly, through the memory controller. Windowing is a technique that maps blocks of the upper register file into a *window* in the lower register file. See Chapter 4, "Memory Partitions,"for more information about the register file and windowing.

2.3.2 Memory Controller

The RALU communicates with all memory, except the register file and peripheral SFRs, through the memory controller. (It communicates with the upper register file through the memory controller except when *windowing* is used; see Chapter 4, "Memory Partitions,") The memory controller contains the prefetch queue, the slave program counter (slave PC), address and data registers, the bus controller, and the chip-select unit.

The bus controller drives the memory bus, which consists of an internal memory bus and the external bus. The bus controller receives memory-access requests from either the RALU or the prefetch queue; queue requests always have priority. This queue is transparent to the RALU and your software.

NOTE

When using a logic analyzer to debug code, remember that instructions are preloaded into the prefetch queue and are not necessarily executed immediately after they are fetched.

When the bus controller receives a request from the queue, it fetches the code from the address contained in the slave PC. The slave PC increases execution speed because the next instruction byte is available immediately and the processor need not wait for the master PC to send the address to the memory controller. If a jump, interrupt, call, or return changes the address sequence, the master PC loads the new address into the slave PC, then the CPU flushes the queue and continues processing.

The memory controller includes a chip-select unit with three chip-select outputs for selecting an external device during an external bus cycle. During an external memory access, a chip-select output is asserted if the address falls within the address range assigned to that chip-select. The bus width, the number of wait states, and multiplexed or demultiplexed address/data lines are programmed independently for the chip-selects. The address range of the chip-selects can be programmed for various granularities: 256 bytes, 512 bytes, … 512 Kbytes, or 2 Mbytes. The base address can be any address that is evenly divisible by the selected address range. See Chapter 15, "Interfacing with External Memory," for more information.

2.4 INTERNAL TIMING

This section describes the internal clock circuitry and power management logic.

2.4.1 Clock and Power Management Logic

The 8XC196EA's clock circuitry (Figure 2-6) implements a phase-locked loop and clock multiplier, which can substantially increase the CPU clock rate while using a lower-frequency input clock.

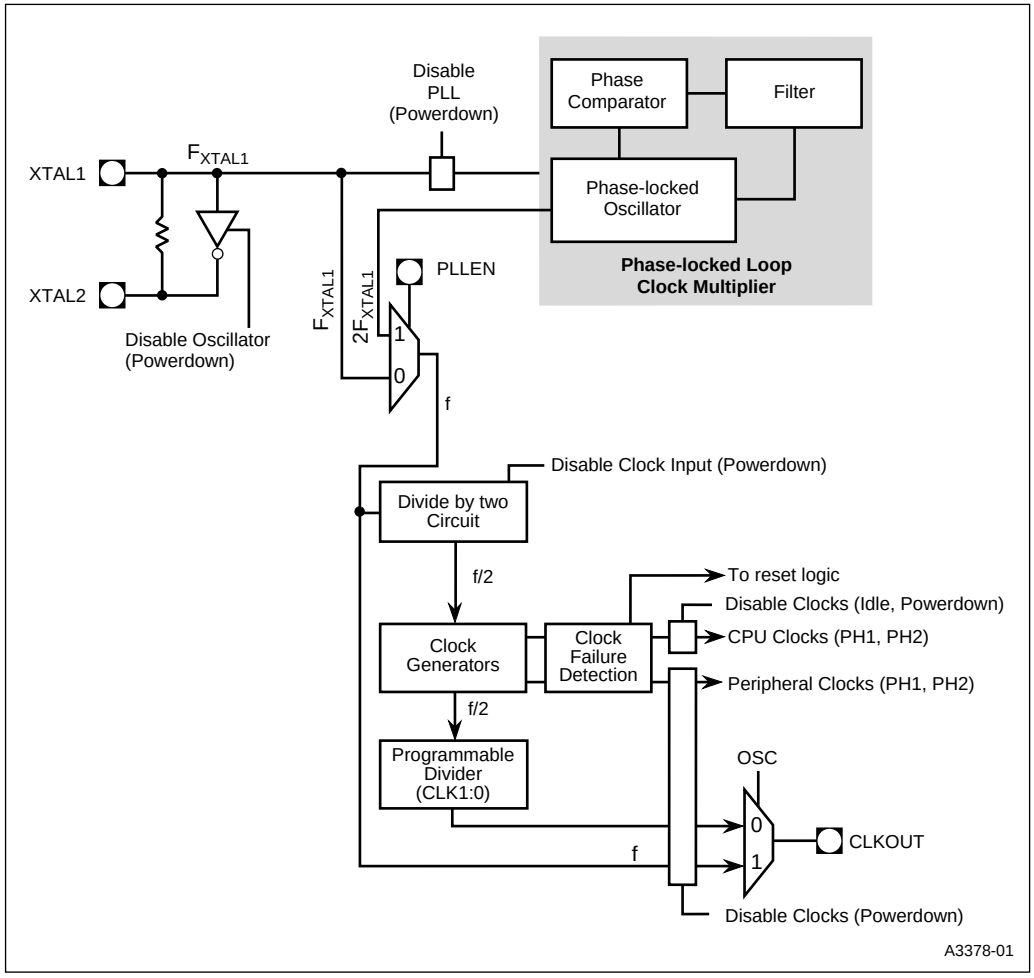


Figure 2-5. Clock Circuitry

NOTE

This manual uses lowercase "f" to represent the internal clock frequency. For the 8XC196EA, "f" is equal to either F_{XTAL1} or $2F_{XTAL1}$, depending on the clock multiplier mode, which is controlled by the PLLEN input pin. The "f" frequency is routed through the divide-by-two circuit to the clock generators, which produce the two nonoverlapping internal timing signals, PH1 and PH2.

2.4.2 Internal Timing

The clock circuitry accepts an input clock signal on XTAL1 provided by an external crystal or oscillator. This frequency is routed either through the phase-locked loop and multiplier circuitry or directly to the divide-by-two circuit. The multiplier circuitry can double the input frequency (F_{XTAL1}). The frequency (f) input to the divide-by-two circuit is either F_{XTAL1} or $2F_{XTAL1}$, depending on the PLLEN pin. The clock generators accept the frequency (f/2) from the divide-by-two circuit and produce two active-high, nonoverlapping internal timing signals, PH1 and PH2. The clock circuitry routes separate internal clock signals to the CPU and the peripherals for flexibility in power management.

The rising edges of PH1 and PH2 generate the internal CLKOUT signal (Figure 2-6).

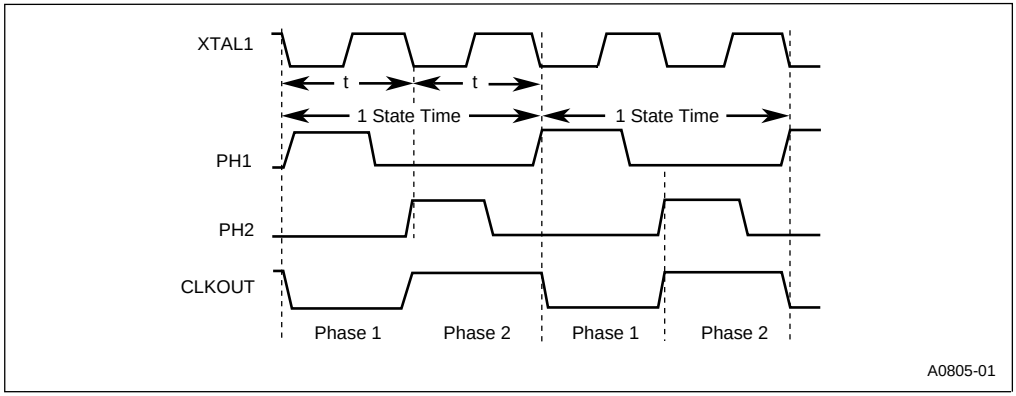


Figure 2-6. Internal Clock Phases (Assuming PLL is Bypassed)

The combined period of phase 1 and phase 2 of the internal CLKOUT signal defines the basic time unit known as a *state time* or *state*. Table 2-3 lists state time durations at various frequencies.

Table 2-3. State Times at Various Frequencies

f (Frequency Input to the Divide-by-two Circuit)	State Time
10 MHz	200 ns
20 MHz	100 ns
40 MHz	50 ns

The following formulas calculate the frequency of PH1 and PH2, the duration of a state time, and the duration of a clock period (t).

$$PH1 = \frac{f}{2} = PH2 \qquad \text{State Time} = \frac{2}{f} \qquad t = \frac{1}{f}$$

Because the microcontroller can operate at many frequencies, this manual defines time requirements (such as instruction execution times) in terms of state times rather than specific measurements. Datasheets list AC characteristics in terms of clock periods (t).

Table 2-4 details the relationships between the input frequency (F_{XTAL1}), the configuration of PLLEN, the operating frequency (f), the clock period (t), and state times.

Table 2-4. Relationships Between Input Frequency, Clock Multiplier, and State Times

F_{XTAL1} (Frequency on XTAL1)	PLLEN	Multiplier	f (Input Frequency to the Divide-by-two Circuit)	t (Clock Period)	State Time
40 MHz †	0	1	40 MHz	25 ns	50 ns
20 MHz	0	1	20 MHz	50 ns	100 ns
	1	2	40 MHz	25 ns	50 ns
10 MHz	0	1	10 MHz	100 ns	200 ns
	1	2	20 MHz	50 ns	100 ns

† Assumes an external clock. The maximum frequency for an external crystal oscillator is 20 MHz.

Figure 2-7 illustrates the timing relationships between the input frequency (F_{XTAL1}), the operating frequency (f), and the internal CLKOUT signal with each PLLEN pin configuration.

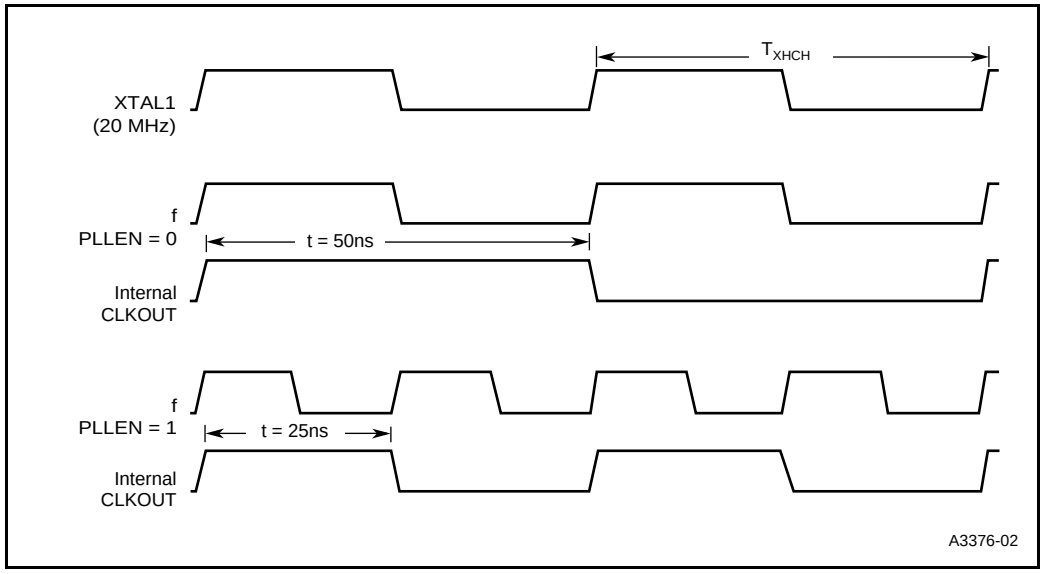


Figure 2-7. Effect of Clock Mode on Internal CLKOUT Frequency

2.4.2.1 Clock Failure Detection Logic

The clock failure detection logic monitors the output frequency from the clock generators and triggers a device reset if the frequency falls below approximately 100 kHz. Program the CFD bit in the chip configuration 1 (CCR1) register (see Figure 2-10 on page 2-20) to enable and disable this logic.

2.4.2.2 External Timing

You can control the output frequency on the CLKOUT pin by programming three bits in the CLKOUT_CON register (Figure 2-8). You can select one of five frequencies: f, f/2, f/4, f/8, or f/16. As Figure 2-5 on page 2-9 shows, the programmable divider accepts the output of the clock generators, f/2, and further divides that frequency to produce the desired output frequency. The CLK1:0 bits control the divisor (divide f/2 by either 1, 2, 4, or 8). The OSC bit selects either the output of the programmable divider or the undivided frequency, f (which bypasses the divide-by-two circuit and clock generators). Table 2-5 lists the CLKOUT output frequency for various configurations. Because of the complex logic in the clock circuitry, the signal on the CLKOUT pin is a delayed version of the internal CLKOUT signal. This delay varies with temperature and voltage.

CLKOUT_CON Address: 1F80H
Reset State: 00H

The CLKOUT control (CLKOUT_CON) register controls the output frequency on the CLKOUT pin.

7							0
—	—	—	—	—	OSC	CLK1	CLK0

Bit Number	Bit Mnemonic	Function
7:3	—	Reserved; for compatibility with future devices, write zeros to these bits.
2	OSC	This bit controls whether the CLKOUT output carries the undivided frequency, f, or the output of the programmable divider: 0 = use the divided frequency (divisor defined by CLK1:0) 1 = use the undivided frequency (CLKOUT = f)
1:0	CLK1:0	These bits specify the divisor to be used by the programmable divider: **CLK1 CLK0** 0 0 divide by 8 (CLKOUT = f/16) 0 1 divide by 4 (CLKOUT = f/8) 1 0 divide by 2 (CLKOUT = f/4) 1 1 divide by 1 (CLKOUT = f/2)

Figure 2-8. CLKOUT Control (CLKOUT_CON) Register

Table 2-5. Controlling the CLKOUT Output Frequency

Input Frequency (F_{XTAL1})	PLLEN	Operating Frequency (f)	Input to Programmable Divider (f/2)	OSC Bit	CLK1:0 Bits	CLKOUT Output Frequency
40 MHz	0	40 MHz	20 MHz	1	XX	40 MHz
20 MHz	0	20 MHz	10 MHz	1	XX	20 MHz
20 MHz	1	40 MHz	20 MHz	1	XX	40 MHz
40 MHz	0	40 MHz	20 MHz	0	11	20 MHz
					10	10 MHz
					10	5 MHz
					00	2.5 MHz
20 MHz	0	20 MHz	10 MHz	0	11	10 MHz
					10	5 MHz
					10	2.5 MHz
					00	1.25 MHz
20 MHz	1	40 MHz	20 MHz	0	11	20 MHz
					10	10 MHz
					10	5 MHz
					00	2.5 MHz

2.4.2.3 Power Management Options

The power saving modes selectively disable internal clocks to conserve power when the microcontroller is inactive. This microcontroller has two power-saving modes: idle and powerdown. If the power-saving modes are enabled in CCB0, the microcontroller enters a power-saving mode after executing the IDLPD instruction with a valid key (an invalid key causes a device reset). Figure 2-5 on page 2-9 illustrates the clock circuitry of the 8XC196EA.

In idle mode, the CPU stops executing instructions, but the peripheral clocks remain active. Power consumption drops to about 60% of normal execution mode consumption. Either a hardware reset or any enabled interrupt source will bring the microcontroller out of idle mode.

In powerdown mode, all internal clocks are frozen at logic state zero and the internal oscillator is shut off. The register file, internal code and data RAM, and most peripherals retain their data if V_{CC} is maintained. Power consumption drops into the µW range.

2.4.3 Internal Memory

In addition to its 1 Kbyte of register RAM, the 83C196EA has 8 Kbytes of ROM and 3 Kbytes of code/data RAM. Chapter 4, “Memory Partitions,” describes the memory contents and locations.

2.4.4 Serial Debug Unit

The serial debug unit allows you to read and write the contents of the code RAM using a high-speed, dedicated serial link. This debugging capability is new to the MCS 96 microcontroller family. The serial debug unit has four main features:

- It provides a real-time method for developing and debugging code with no CPU overhead.
- It provides a simple interface to the microcontroller without extensive external hardware.
- It supports reading and writing of all internal and external memory in interrogation mode.
- It supports breakpoints in both internal and external memory development systems.

Chapter 16, "Serial Debug Unit," explains how to transfer data to and from the code RAM and provides examples using the SDU command instruction set.

2.4.5 Interrupt Service

The microcontroller's flexible interrupt-handling system has two main components: the programmable interrupt controller and the peripheral transaction server (PTS). The programmable interrupt controller has a hardware priority scheme that can be modified by your software. Interrupts that go through the interrupt controller are serviced by interrupt service routines that you provide. The peripheral transaction server (PTS), a microcoded hardware interrupt processor, provides high-speed, low-overhead interrupt handling. You can configure most interrupts (except NMI, stack overflow, trap, and unimplemented opcode) to be serviced by the PTS instead of the interrupt controller.

The PTS provides four special microcoded routines that enable it to complete specific tasks in much less time than an equivalent interrupt service routine can. It can transfer bytes or words, either individually or in blocks, between any memory locations in page 00H; abort PTS service if a dummy PTS request occurs; and test for a missing event in a series of regular events. PTS interrupts have a higher priority than standard interrupts and may temporarily suspend interrupt service routines.

To provide additional support for the event processor array (EPA), which can generate several interrupt requests, the 8XC196EA incorporates two peripheral interrupt handlers (PIHs). A PIH provides the interrupt vector for the specific interrupt request to the interrupt controller or the PTS. See Chapter 6, "Standard and PTS Interrupts," for more information on interrupt service options.

2.5 INTERNAL PERIPHERALS

The internal peripheral modules provide special functions for a variety of applications. This section provides a brief description of the peripherals; subsequent chapters describe them in detail.

2.5.1 I/O Ports

The 8XC196EA has 11 input/output ports, ports 2–5, ports 7–12, and the EPORT. In general, you can configure individual port pins to serve as standard I/O or to carry special-function signals associated with on-chip peripherals or off-chip components. Ports 3 and 4 are exceptions; they are controlled at the port level, not at the pin level. When the bus controller needs to use the external bus, it takes control of the ports. When the external bus is idle, you can use the ports for I/O.

Ports 2, 7, 8, 9, and 11 are eight-bit, bidirectional, standard ports. Port 10 is a six-bit, bidirectional standard port. The standard ports share pins with individually selectable special-function signals. The standard ports can be windowed and accessed with direct addressing.

Ports 3 and 4 are eight-bit, bidirectional, memory-mapped ports that share pins with the multiplexed address/data bus. The EPORT is an eight-bit, bidirectional, memory-mapped port; it shares five of its pins with the five upper address lines, A20:16, to support extended addressing. Port 5 is an eight-bit, bidirectional, memory-mapped port that shares pins with individually selectable control signals. Port 12 is a five-bit, bidirectional, memory-mapped port; it shares three of its pins with signals that enable test-ROM execution. Memory-mapped ports must be accessed using indirect, indexed, or extended addressing; they cannot be windowed. See Chapter 7, “I/O Ports,” for more information.

2.5.2 Serial I/O (SIO) Port

The 8XC196EA has a two-channel serial I/O port. The serial I/O (SIO) port is an asynchronous/synchronous port that includes a universal asynchronous receiver and transmitter (UART). The UART has one synchronous mode (mode 0) and three asynchronous modes (modes 1, 2, and 3) for both transmission and reception. The asynchronous modes are full duplex, meaning that they have dedicated receive and transmit data signals. The receiver is buffered, so the reception of a second byte can begin before the first byte is read. The transmitter is also buffered, allowing continuous transmissions. The SIO port has two channels (channels 0 and 1) with identical signals and registers. See Chapter 8, “Serial I/O (SIO) Port,” for details.

2.5.3 Synchronous Serial I/O (SSIO) Port

The synchronous serial I/O (SSIO) port provides for simultaneous, bidirectional communications between two MCS 96 family microcontrollers or between an MCS 96 microcontroller and another synchronous serial I/O device. The SSIO port consists of two identical transceiver channels with a dedicated baud-rate generator.

The SSIO port provides one bidirectional communication channel or two unidirectional channels. The SSIO is compatible with most protocols because the serial clock is completely configurable. Paired, the SSIO channels can operate in a channel-select mode, allowing for communication in multiple-master systems without additional external hardware. See Chapter 9, “Synchronous Serial I/O (SSIO) Port,” for more information.

2.5.4 Event Processor Array (EPA) and Timer/Counters

The event processor array (EPA) performs high-speed input and output functions associated with its timer/counters. In the input mode, the EPA monitors an input for signal transitions. When an event occurs, the EPA records the timer value associated with it. This is a *capture* event. In the output mode, the EPA monitors a timer until its value matches that of a stored time value. When a match occurs, the EPA triggers an output event, which can set, clear, or toggle an output pin. This is a *compare* event. Both capture and compare events can initiate interrupts, which can be serviced by either the interrupt controller or the PTS.

The 8XC196EA has an enhanced EPA with 4 timer/counters, 17 capture/compare channels, and 8 specialized compare-only channels. A compare-only channel can capture the value of any timer other than its reference timer at the time of a compare event. Capturing the timer value while simultaneously triggering the compare event output is called *output/simulcapture*. This feature simplifies conversion between angle and time domains.

Timers 1–4 are 16-bit up/down timer/counters that can be clocked internally or externally. Each timer/counter is called a *timer* if it is clocked internally and a *counter* if it is clocked externally. Each pair of timer/counters (timers 1–2 and timers 3–4) can be concatenated to provide two 32-bit timer/counters. See Chapter 11, "Event Processor Array (EPA),"for additional information on the EPA and timer/counters.

2.5.5 Analog-to-digital Converter

The analog-to-digital (A/D) converter converts an analog input voltage to a digital equivalent. Resolution is either 8 or 10 bits; sample and convert times are programmable. Conversions can be performed on the analog ground and reference voltage, and the results can be used to calculate gain and zero-offset errors. The internal zero-offset compensation circuit enables automatic zero-offset adjustment. The A/D also has a threshold-detection mode that can generate an interrupt when a programmable threshold voltage is crossed in either direction. The 8XC196EA's enhanced A/D converter has an automatic scan mode that operates without CPU intervention. In addition, it has a dedicated result register for each analog input channel. See Chapter 12, "Analog-to-digital (A/D) Converter," for more information.

2.5.6 Pulse-width Modulator (PWM)

The output waveform from each PWM channel is a variable duty-cycle pulse with a programmable frequency. Several types of motors require a PWM waveform for most efficient operation. When filtered, the PWM waveform produces a DC level that can change in 256 steps by varying the duty cycle. The number of steps per PWM period is also programmable (8 bits). See Chapter 10, "Pulse-width Modulator," for more information.

2.5.7 Stack Overflow Module

The stack overflow module monitors the stack pointer (SP) and causes a monmaskable interrupt if the stack pointer crosses upper or lower boundaries you define, assisting in code development. See Chapter 5, "Stack Overflow Module," for details.

2.5.8 Watchdog Timer

The watchdog timer is a 16-bit internal timer that resets the microcontroller if the software fails to operate properly. See Chapter 13, “Minimum Hardware Considerations,” for more information.

2.6 SPECIAL OPERATING MODES

In addition to the normal execution and power-saving modes, the microcontroller operates in special-purpose mode. On-circuit emulation (ONCE) mode electrically isolates the microcontroller from the system. By invoking the ONCE mode, you can test the printed circuit board while the microcontroller is soldered onto the board. See Chapter 14, “Special Operating Modes,” for more information about power-saving and ONCE modes.

2.7 CHIP CONFIGURATION REGISTERS

Two chip configuration bytes (CCBs) located in ROM control the basic configuration of the microcontroller. These bytes are loaded into the chip configuration registers (CCRs) as part of the reset sequence. Once they are loaded, the CCRs control many aspects of the microcontroller’s operation. Figures 2-9 and 2-10 illustrate the CCRs and describe their functions.

CCR0 no direct access†

The chip configuration 0 (CCR0) register controls ROM access, enables or disables the IDLPD #1 and IDLPD #2 instructions and selects the write-control mode. It also contains the bus-control parameters for fetching chip configuration byte 1.

7							0
LOC	1	WS1	WS0	DEMUX	BHE#	BW16	PD

Bit Number	Bit Mnemonic	Function
7	LOC	Lock Bit This bit controls read access to the ROM during normal operation. 0 = read protect 1 = no protection Refer to “Controlling Read Access to the Internal ROM” on page 4-26 for details.
6	1	To guarantee proper operation, write one to this bit.

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after reset. The CCBs reside at addresses FF2018H (CCB0) and FF201AH (CCB1).

Figure 2-9. Chip Configuration 0 (CCR0) Register

CCR0 (Continued) no direct access†

The chip configuration 0 (CCR0) register controls ROM access, enables or disables the IDLPD #1 and IDLPD #2 instructions and selects the write-control mode. It also contains the bus-control parameters for fetching chip configuration byte 1.

7							0
LOC	1	WS1	WS0	DEMUX	BHE#	BW16	PD

Bit Number	Bit Mnemonic	Function
5:4	WS1:0	Wait States These bits, along with the READY pin, control the number of wait states that are used for an external fetch of chip configuration byte 1 (CCB1). **WS1 WS0** 0 0 zero wait states 0 1 one wait state 1 0 two wait states 1 1 three wait states If READY is low when this number is reached, additional wait states are added until READY is pulled high.
3	DEMUX	Select Demultiplexed Bus Selects the demultiplexed bus mode for an external fetch of CCB1: 0 = multiplexed — address and data are multiplexed on AD15:0. 1 = demultiplexed — data only on AD15:0.
2	BHE#	Write-control Mode Selects the write-control mode, which determines the functions of the BHE#/WRH# and WR#/WRL# pins for external bus cycles: 0 = write strobe mode: the BHE#/WRH# pin operates as WRH#, and the WR#/WRL# pin operates as WRL#. 1 = standard write-control mode: the BHE#/WRH# pin operates as BHE#, and the WR#/WRL# pin operates as WR#.

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after reset. The CCBs reside at addresses FF2018H (CCB0) and FF201AH (CCB1).

Figure 2-9. Chip Configuration 0 (CCR0) Register (Continued)

CCR0 (Continued) no direct access†

The chip configuration 0 (CCR0) register controls ROM access, enables or disables the IDLPD #1 and IDLPD #2 instructions and selects the write-control mode. It also contains the bus-control parameters for fetching chip configuration byte 1.

7							0
LOC	1	WS1	WS0	DEMUX	BHE#	BW16	PD

Bit Number	Bit Mnemonic	Function
1	BW16	Buswidth Control Selects the bus width for an external fetch of CCB1: 0 = 8-bit bus 1 = 16-bit bus
0	PD	Powerdown Enable Enables or disables the IDLPD #1 and IDLPD #2 instructions. When enabled, the IDLPD #1 instruction causes the microcontroller to enter idle mode and the IDLPD #2 instruction causes the microcontroller to enter powerdown mode. 0 = disable idle and powerdown modes 1 = enable idle and powerdown modes If your design uses idle or powerdown mode, set this bit when you program the CCBs. If it does not, clearing this bit when you program the CCBs will prevent accidental entry into idle or powerdown mode. (Chapter 14, “Special Operating Modes,” discusses idle and powerdown modes.)

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after reset. The CCBs reside at addresses FF2018H (CCB0) and FF201AH (CCB1).

Figure 2-9. Chip Configuration 0 (CCR0) Register (Continued)

CCR1 no direct access†

The chip configuration 1 (CCR1) register controls the clock failure detection circuitry, deferred mode, and the watchdog timer, selects the 64-Kbyte or 2-Mbyte addressing mode, and controls whether the upper 7-Kbyte section of internal ROM (FF2400–FF3FFFH) is mapped into page FFH only or into both pages FFH and 00H.

7							0
1	CFD	DM	0	WDD	REMAP	MODE64	0

Bit Number	Bit Mnemonic	Function
7	1	To guarantee proper operation, write one to this bit.
6	CFD	Clock-failure Detection This bit enables or disables the clock failure detection circuitry. 0 = disables clock-failure detection circuitry 1 = enables clock-failure detection circuitry (See "Clock Failure Detection Logic" on page 2-12.)
5	DM	Deferred Mode Enables the deferred bus-cycle mode. If the microcontroller is using a demultiplexed bus and deferred mode is enabled, a delay of 2t occurs in the first bus cycle following a chip-select output change, the first write cycle following a read cycle, and the first read cycle following a write cycle. (See "Deferred Bus-cycle Mode" on page 15-41.) 0 = deferred bus-cycle mode disabled 1 = deferred bus-cycle mode enabled
4		To guarantee proper operation, write zero to this bit.
3	WDD	Watchdog Timer Disable Selects whether the watchdog timer is always enabled or disabled until the first time it is cleared. If this bit is clear, the watchdog is enabled at reset, so software must clear the watchdog within 64K state times to prevent another reset. If this bit is set, the watchdog is disabled until the first time you clear it. (See "Enabling the Watchdog Timer" on page 13-12.) 0 = always enabled 1 = disabled at reset; enabled the first time it is cleared

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after reset. Locate your CCBs at addresses FF2018H (CCB0) and FF201AH (CCB1).

Figure 2-10. Chip Configuration 1 (CCR1) Register

CCR1 (Continued) no direct access†

The chip configuration 1 (CCR1) register controls the clock failure detection circuitry, deferred mode, and the watchdog timer, selects the 64-Kbyte or 2-Mbyte addressing mode, and controls whether the upper 7-Kbyte section of internal ROM (FF2400–FF3FFFH) is mapped into page FFH only or into both pages FFH and 00H.

7 **0**

1	CFD	DM	0	WDD	REMAP	MODE64	0

Bit Number	Bit Mnemonic	Function
2	REMAP	Internal ROM Mapping The EA# pin controls whether accesses to FF2000–FF3FFFH are directed to internal ROM or to external memory. When EA# is low (external execution), REMAP is ignored. When EA# is high (internal execution), REMAP controls whether the upper 7-Kbyte area (FF2400–FF3FFFH) of internal ROM is mapped only into page FFH or into both pages FFH and 00H. 0 = ROM maps to page FFH only 1 = ROM maps to page FFH and page 00H (See “Remapping Internal ROM” on page 4-29.)
1	MODE64	Addressing Mode Selects 64-Kbyte or 2-Mbyte addressing. 0 = selects 2-Mbyte addressing 1 = selects 64-Kbyte addressing In 2-Mbyte mode, code can execute from almost anywhere in the address space. In 64-Kbyte mode, code can execute only from page FFH. (See “Fetching Code and Data in the 2-Mbyte and 64-Kbyte Modes” on page 4-31 for more information.)
0	0	Reserved; for compatibility with future devices, write zero to this bit.

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after reset. Locate your CCBs at addresses FF2018H (CCB0) and FF201AH (CCB1).

Figure 2-10. Chip Configuration 1 (CCR1) Register (Continued)

3

Programming Considerations

CHAPTER 3
PROGRAMMING CONSIDERATIONS

This section provides an overview of the instruction set of the MCS® 96 microcontrollers and offers guidelines for program development. For detailed information about specific instructions, see Appendix A.

3.1 OVERVIEW OF THE INSTRUCTION SET

The instruction set supports a variety of data types likely to be useful in control applications (see Table 3-1).

NOTE

The data-type variables are shown in all capitals to avoid confusion. For example, a *BYTE* is an unsigned 8-bit variable in an instruction, while a *byte* is any 8-bit unit of data (either signed or unsigned).

Table 3-1. Data Type Definitions

Data Type	No. of Bits	Signed	Possible Values	Addressing Restrictions
BIT	1	No	True (1) or False (0)	As components of bytes
BYTE	8	No	0 through 2^8–1 (0 through 255)	None
SHORT-INTEGER	8	Yes	-2^7 through $+2^7$–1 (–128 through +127)	None
WORD	16	No	0 through 2^{16}–1 (0 through 65,535)	Even byte address
INTEGER	16	Yes	-2^{15} through $+2^{15}$–1 (–32,768 through +32,767)	Even byte address
DOUBLE-WORD (Note 1)	32	No	0 through 2^{32}–1 (0 through 4,294,967,295)	An address in the lower register file that is evenly divisible by four
LONG-INTEGER (Note 1)	32	Yes	-2^{31} through $+2^{31}$–1 (–2,147,483,648 through +2,147,483,647)	An address in the lower register file that is evenly divisible by four
QUAD-WORD (Note 2)	64	No	0 through 2^{64}–1	An address in the lower register file that is evenly divisible by eight

NOTES:
1. The 32-bit operands are supported only in shift operations, as the dividend in 32-by-16 division, and as the product of 16-by-16 multiplication.
2. QUAD-WORD variables are supported only as the operand for the EBMOVI instruction.

Table 3-2 lists the equivalent data-type names for both C programming and assembly language.

Table 3-2. Equivalent Data Types for Assembly and C Programming Languages

Data Type	Assembly Language Equivalent	C Programming Language Equivalent
BYTE	BYTE	unsigned char
SHORT-INTEGER	BYTE	char
WORD	WORD	unsigned int
INTEGER	WORD	int
DOUBLE-WORD	LONG	unsigned long
LONG-INTEGER	LONG	long
QUAD-WORD	—	—

3.1.1 BIT Operands

A BIT is a single-bit variable that can have the Boolean values, "true" and "false." The architecture requires that BITs be addressed as components of BYTEs or WORDs. The architecture does not support the direct addressing of BITs. (You can, however, test the state of a single bit. For example, the JBC and JBS instructions are conditional jump instructions that test a specified bit.)

3.1.2 BYTE Operands

A BYTE is an unsigned, 8-bit variable that can take on values from 0 through 255 (2^8–1). Arithmetic and relational operators can be applied to BYTE operands, but the result must be interpreted in modulo 256 arithmetic. Logical operations on BYTEs are applied bitwise. Bits within BYTEs are labeled from 0 to 7; bit 0 is the least-significant bit. Alignment restrictions do not apply to BYTEs, so they may be placed anywhere in the address space.

3.1.3 SHORT-INTEGER Operands

A SHORT-INTEGER is an 8-bit, signed variable that can take on values from –128 ($–2^7$) through +127 ($+2^7$–1). Arithmetic operations that generate results outside the range of a SHORT-INTEGER set the overflow flags in the processor status word (PSW). The numeric result is the same as the result of the equivalent operation on BYTE variables. There are no alignment restrictions on SHORT-INTEGERs, so they may be placed anywhere in the address space.

3.1.4 WORD Operands

A WORD is an unsigned, 16-bit variable that can take on values from 0 through 65,535 (2^{16}–1). Arithmetic and relational operators can be applied to WORD operands, but the result must be interpreted in modulo 65536 arithmetic. Logical operations on WORDs are applied bitwise. Bits within WORDs are labeled from 0 to 15; bit 0 is the least-significant bit.

WORDs must be aligned at even byte boundaries in the address space. The least-significant byte of the WORD is in the even byte address, and the most-significant byte is in the next higher (odd) address. The address of a WORD is that of its least-significant byte (the even byte address). WORD operations to odd addresses are not guaranteed to operate in a consistent manner.

3.1.5 INTEGER Operands

An INTEGER is a 16-bit, signed variable that can take on values from –32,768 (-2^{15}) through +32,767 ($+2^{15}$–1). Arithmetic operations that generate results outside the range of an INTEGER set the overflow flags in the processor status word (PSW). The numeric result is the same as the result of the equivalent operation on WORD variables.

INTEGERs must be aligned at even byte boundaries in the address space. The least-significant byte of the INTEGER is in the even byte address, and the most-significant byte is in the next higher (odd) address. The address of an INTEGER is that of its least-significant byte (the even byte address). INTEGER operations to odd addresses are not guaranteed to operate in a consistent manner.

3.1.6 DOUBLE-WORD Operands

A DOUBLE-WORD is an unsigned, 32-bit variable that can take on values from 0 through 4,294,967,295 (2^{32}–1). The architecture directly supports DOUBLE-WORD operands only as the operand in shift operations, as the dividend in 32-by-16 divide operations, and as the product of 16-by-16 multiply operations. For these operations, a DOUBLE-WORD variable must reside in the lower register file and must be aligned at an address that is evenly divisible by four. The address of a DOUBLE-WORD is that of its least-significant byte (the even byte address). The least-significant word of the DOUBLE-WORD is always in the lower address, even when the data is in the stack. This means that the most-significant word must be pushed onto the stack first.

DOUBLE-WORD operations that are not directly supported can be easily implemented with two WORD operations. For example, the following sequences of 16-bit operations perform a 32-bit addition and a 32-bit subtraction, respectively.

```
ADD  REG1,REG3                    ; (2-operand addition)
ADDC REG2,REG4

SUB  REG1,REG3                    ; (2-operand subtraction)
SUBC REG2,REG4
```

3.1.7 LONG-INTEGER Operands

A LONG-INTEGER is a 32-bit, signed variable that can take on values from –2,147,483,648 (-2^{31}) through +2,147,483,647 ($+2^{31}-1$). The architecture directly supports LONG-INTEGER operands only as the operand in shift operations, as the dividend in 32-by-16 divide operations, and as the product of 16-by-16 multiply operations. For these operations, a LONG-INTEGER variable must reside in the lower register file and must be aligned at an address that is evenly divisible by four. The address of a LONG-INTEGER is that of its least-significant byte (the even byte address).

LONG-INTEGER operations that are not directly supported can be easily implemented with two INTEGER operations. See the example in "DOUBLE-WORD Operands" on page 3-3.

3.1.8 QUAD-WORD Operands

A QUAD-WORD is a 64-bit, unsigned variable that can take on values from 0 through $2^{64}-1$. The architecture directly supports the QUAD-WORD operand only as the operand of the EBMOVI instruction. For this operation, the QUAD-WORD variable must reside in the lower register file and must be aligned at an address that is evenly divisible by eight.

3.1.9 Converting Operands

The instruction set supports conversions between the data types (Table 3-3). The LDBZE (load byte, zero extended) instruction converts a BYTE to a WORD. CLR (clear) converts a WORD to a DOUBLE-WORD by clearing (writing zeros to) the upper WORD of the DOUBLE-WORD. LDBSE (load byte, sign extended) converts a SHORT-INTEGER into an INTEGER. EXT (sign extend) converts an INTEGER to a LONG-INTEGER.

Table 3-3. Converting Data Types

To convert from …	to…	Use this instruction…	Which performs this function
BYTE	WORD	LDBZE	Writes zeros to the upper byte.
WORD	DOUBLE-WORD	CLR	Writes zeros to the upper word.
SHORT-INTEGER	INTEGER	LDBSE	Writes the sign bit to the upper byte.
INTEGER	LONG-INTEGER	EXT	Writes the sign bit to the upper word.

3.1.10 Conditional Jumps

The instructions for addition, subtraction, and comparison do not distinguish between unsigned (BYTE, WORD) and signed (SHORT-INTEGER, INTEGER) data types. However, the conditional jump instructions allow you to treat the results of these operations as signed or unsigned quantities. For example, the CMP (compare) instruction is used to compare both signed and unsigned 16-bit quantities. Following a compare operation, you can use the JH (jump if higher) instruction for unsigned operands or the JGT (jump if greater than) instruction for signed operands.

3.1.11 Floating-Point Operations

The hardware does not directly support operations on REAL (floating point) variables. Those operations are supported by floating-point libraries from third-party tool vendors. (See the *Development Tools Handbook*.) The performance of these operations is significantly improved by the NORML instruction and by the sticky bit (ST) flag in the processor status word (PSW). The NORML instruction normalizes a 32-bit variable; the sticky bit (ST) flag can be used in conjunction with the carry (C) flag to achieve finer resolution in rounding.

3.1.12 Extended Instructions

This section briefly describes the instructions that enable code execution and data access anywhere in the address space. These instructions are implemented for all MCS 96 microcontrollers that have extended addressing ports (currently the 8XC196NT, 8XC196NP, 80C196NU, and 8XC196EA microcontrollers). They function only in extended addressing modes.

In general, you should avoid creating tables or arrays that cross page boundaries. For example, if you are building a large array, start it at a base address that will accommodate the entire array within the same page. If you cannot avoid crossing a page boundary, keep in mind that you must use extended instructions to access data outside page 00H.

NOTE

In 2-Mbyte mode, ECALL, LCALL, and SCALL always push two words onto the stack; therefore, a RET must always pop two words from the stack. Because of the extra push and pop operations, interrupt routines and subroutines take slightly longer to execute in 2-Mbyte mode than in 64-Kbyte mode.

EBMOVI	**Extended interruptible block move**. Moves a block of word data from one memory location to another. This instruction allows you to move blocks of up to 64K words between any two locations in the address space. It uses two 24-bit autoincrementing pointers and a 16-bit counter.
EBR	**Extended branch**. This instruction is an unconditional indirect jump to anywhere in the address space.
ECALL	**Extended call**. This instruction is an unconditional relative call to anywhere in the address space.
EJMP	**Extended jump**. This instruction is an unconditional relative jump to anywhere in the address space.
ELD, ELDB	**Extended load word, extended load byte**. Loads the value of the source operand into the destination operand. This instruction allows you to move data from anywhere in the address space into the lower register file.
EST, ESTB	**Extended store word, extended store byte**. Stores the value of the source **(left-most)** operand into the destination **(right-most)** operand. This instruction allows you to move data from the lower register file to anywhere in the address space.

3.2 ADDRESSING MODES

This section describes the addressing modes as they are handled by the hardware. An understanding of these details will help programmers to take full advantage of the architecture. Most software tools have features that simplify the choice of addressing modes. Please consult the documentation for your assembler or compiler for details.

The instruction set uses four basic addressing modes:

- direct
- immediate
- indirect (with or without autoincrement)
- indexed (short-, long-, or zero-indexed)

The stack pointer can be used with indirect addressing to access the top of the stack, and it can also be used with short-indexed addressing to access data within the stack. The zero register can be used with long-indexed addressing to access any memory location.

Extended variations of the indirect and indexed modes support the extended load and store instructions. An extended load instruction moves a word (ELD) or a byte (ELDB) from any location in the address space into the lower register file. An extended store instruction moves a word (EST) or a byte (ESTB) from the lower register file into any location in the address space. An instruction can contain only one immediate, indirect, or indexed reference; any remaining operands must be direct references.

The examples in this section assume that temporary registers are defined as shown in Table 3-4.

Table 3-4. Definition of Temporary Registers

Temporary Register	Description
AX	word-aligned 16-bit register; AH is the high byte of AX and AL is the low byte
BX	word-aligned 16-bit register; BL is the low byte of BX
CX	word-aligned 16-bit register; CH is the high byte of CX and CL is the low byte
DX	word-aligned 16-bit register; DH is the high byte of DX and DL is the low byte
EX	double-word-aligned 24-bit register

3.2.1 Direct Addressing

Direct addressing directly accesses a location in the 256-byte lower register file, without involving the memory controller. Windowing allows you to remap other sections of memory into the lower register file for direct access (see Chapter 4, “Memory Partitions,” for details). You specify the registers as operands within the instruction. The register addresses must conform to the alignment rules for the operand type. Depending on the instruction, up to three registers can take part in a calculation. The following instructions use direct addressing:

```
ADD  AX,BX,CX          ; AX ← BX + CX
ADDB AL,BL,CL          ; AL ← BL + CL
MULB AX,BL             ; AX ← AX × BL
INCB CL                ; CL ← CL + 1
```

3.2.2 Immediate Addressing

Immediate addressing mode accepts one immediate value as an operand in the instruction. You specify an immediate value by preceding it with a number symbol (#). An instruction can contain only one immediate value; the remaining operands must be direct references. The following instructions use immediate addressing:

```
ADD  AX,#340       ; AX ← AX + 340
PUSH #1234H        ; SP ← SP - 2
                   ; MEM_WORD(SP) ← 1234H
DIVB AX,#10        ; AL ← AX/10
                   ; AH ← AX MOD 10
```

3.2.3 Indirect Addressing

The indirect addressing mode accesses an operand by obtaining its address from a WORD register in the lower register file. You specify the register containing the indirect address by enclosing it in square brackets ([ ]). The indirect address can refer to any location within the address space, including the register file. The register that contains the indirect address must be word-aligned, and the indirect address must conform to the rules for the operand type. An instruction can contain only one indirect reference; any remaining operands must be direct references. The following instructions use indirect addressing:

```
LD   AX,[BX]              ; AX ← MEM_WORD(BX)
ADDB AL,BL,[CX]           ; AL ← BL + MEM_BYTE(CX)
POP  [AX]                 ; MEM_WORD(AX) ← MEM_WORD(SP)
                          ; SP ← SP + 2
```

3.2.3.1 Extended Indirect Addressing

Extended load and store instructions can use indirect addressing. The only difference is that the register containing the indirect address must be a word-aligned 24-bit register to allow access to the entire address space. The following instructions use extended indirect addressing:

```
ELD  AX,[EX]         ; AX ← MEM_WORD(EX)
ELDB AL,[EX]         ; AL ← MEM_BYTE(EX)
EST  AX,[EX]         ; MEM_WORD(EX) ← AX
ESTB AL,[EX]         ; MEM_BYTE(EX) ← AL
```

3.2.3.2 Indirect Addressing with Autoincrement

You can choose to automatically increment the indirect address after the current access. You specify autoincrementing by adding a plus sign (+) to the end of the indirect reference. In this case, the instruction automatically increments the indirect address (by one if the destination is an 8-bit register or by two if it is a 16-bit register). When your code is assembled, the assembler automatically sets the least-significant bit of the indirect address register. The following instructions use indirect addressing with autoincrement:

```
LD   AX,[BX]+        ; AX ← MEM_WORD(BX)
                     ; BX ← BX + 2
ADDB AL,BL,[CX]+     ; AL ← BL + MEM_BYTE(CX)
                     ; CX ← CX + 1
PUSH [AX]+           ; SP ← SP - 2
                     ; MEM_WORD(SP) ← MEM_WORD(AX)
                     ; AX ← AX + 2
```

3.2.3.3 Extended Indirect Addressing with Autoincrement

The extended load and store instructions can also use indirect addressing with autoincrement. The only difference is that the register containing the indirect address must be a word-aligned 24-bit register to allow access to the entire address space. The following instructions use extended indirect addressing with autoincrement:

```
ELD  AX,[EX]+       ; AX ← MEM_WORD(EX)
                    ; EX ← EX + 2
ELDB AL,[EX]+       ; AL ← MEM_BYTE(EX)
                    ; EX ← EX + 1
EST  AX,[EX]+       ; MEM_WORD(EX) ← AX
                    ; EX ← EX + 2
ESTB AL,[EX]+       ; MEM_BYTE(EX) ← AL
                    ; EX ← EX + 1
```

3.2.3.4 Indirect Addressing with the Stack Pointer

You can also use indirect addressing to access the top of the stack by using the stack pointer as the WORD register in an indirect reference. The following instruction uses indirect addressing with the stack pointer:

```
PUSH [SP]      ; duplicate top of stack
               ; SP ← SP - 2
```

3.2.4 Indexed Addressing

Indexed addressing calculates an address by adding an offset to a base address. There are three variations of indexed addressing: short-indexed, long-indexed, and zero-indexed. Both short- and long-indexed addressing are used to access a specific element within a structure. Short-indexed addressing can access up to 256 byte locations, long-indexed addressing can access up to 65,536 byte locations, and zero-indexed addressing can access a single location. An instruction can contain only one indexed reference; any remaining operands must be direct references.

3.2.4.1 Short-indexed Addressing

In a short-indexed instruction, you specify the offset as an 8-bit constant and the base address as an indirect address register (a WORD). The following instructions use short-indexed addressing.

```
LD   AX,12H[BX]     ; AX ← MEM_WORD(BX + 12H)
MULB AX,BL,3[CX]    ; AX ← BL ×MEM_BYTE(CX + 3)
```

The instruction LD AX,12H[BX] loads AX with the contents of the memory location that resides at address BX+12H. That is, the instruction adds the constant 12H (the offset) to the contents of BX (the base address), then loads AX with the contents of the resulting address. For example, if BX contains 1000H, then AX is loaded with the contents of location 1012H. Short-indexed addressing is typically used to access elements in a structure, where BX contains the base address of the structure and the constant (12H in this example) is the offset of a specific element in a structure.

You can also use the stack pointer in a short-indexed instruction to access a particular location within the stack, as shown in the following instruction.

```
LD   AX,2[SP]
```

3.2.4.2 Long-indexed Addressing

In a long-indexed instruction, you specify the base address as a 16-bit variable and the offset as an indirect address register (a WORD). The following instructions use long-indexed addressing.

```
LD   AX,TABLE[BX]        ; AX ← MEM_WORD(TABLE + BX)
AND  AX,BX,TABLE[CX]     ; AX ← BX AND MEM_WORD(TABLE + CX)
ST   AX,TABLE[BX]        ; MEM_WORD(TABLE + BX) ← AX
ADDB AL,BL,LOOKUP[CX]    ; AL ← BL + MEM_BYTE(LOOKUP + CX)
```

The instruction LD AX, TABLE[BX] loads AX with the contents of the memory location that resides at address TABLE+BX. That is, the instruction adds the contents of BX (the offset) to the constant TABLE (the base address), then loads AX with the contents of the resulting address. For example, if TABLE equals 4000H and BX contains 12H, then AX is loaded with the contents of location 4012H. Long-indexed addressing is typically used to access elements in a table, where TABLE is a constant that is the base address of the structure and BX is the scaled offset ($n \times$ element size, in bytes) into the structure.

3.2.4.3 Extended Indexed Addressing

The extended load and store instructions can use extended indexed addressing. The only difference from long-indexed addressing is that both the base address and the offset must be 24 bits to support access to the entire address space. The following instructions use extended indexed addressing. (In these instructions, OFFSET is a 24-bit variable containing the offset, and EX is a double-word aligned 24-bit register containing the base address.)

```
ELD  AX,OFFSET[EX]       ; AX ← MEM_WORD(EX+OFFSET)
ELDB AL,OFFSET[EX]       ; AL ← MEM_BYTE(EX+OFFSET)
EST  AX,OFFSET[EX]       ; MEM_WORD(EX+OFFSET) ← AX
```

3.2.4.4 Zero-indexed Addressing

In a zero-indexed instruction, you specify the address as a 16-bit variable; the offset is zero, and you can express it in one of three ways: [0], [ZERO_REG], or nothing. Each of the following load instructions loads AX with the contents of the variable THISVAR.

```
LD   AX,THISVAR[0]
LD   AX,THISVAR[ZERO_REG]
LD   AX,THISVAR
```

The following instructions also use zero-indexed addressing:

```
ADD  AX,1234H[ZERO_REG]       ; AX ← AX + MEM_WORD(1234H)
POP  5678H[ZERO_REG]          ; MEM_WORD(5678H) ← MEM_WORD(SP)
                              ; SP ← SP + 2
```

3.2.4.5 Extended Zero-indexed Addressing

The extended instructions can also use zero-indexed addressing. The only difference is that you specify the address as a 24-bit constant or variable. The following extended instruction uses zero-indexed addressing. ZERO_REG acts as a 32-bit fixed source of the constant zero for an extended indexed reference.

```
ELD  AX,23456H[ZERO_REG]      ; AX ← MEM_WORD(23456H)
```

3.3 CONSIDERATIONS FOR CROSSING PAGE BOUNDARIES

In general, you should avoid creating tables or arrays that cross page boundaries. For example, if you are building a large array, start it at a base address that will accommodate the entire array within the same page. If you cannot avoid crossing a page boundary, keep in mind that you must use extended instructions to access data outside page 00H.

3.4 SOFTWARE PROTECTION FEATURES AND GUIDELINES

The microcontroller has several features to assist in recovering from hardware and software errors. The unimplemented opcode interrupt provides protection from executing unimplemented opcodes. The hardware reset instruction (RST) can cause a reset if the program counter goes out of bounds. The RST instruction opcode is FFH, so the processor will reset itself if it tries to fetch an instruction from unprogrammed locations in nonvolatile memory or from bus lines that have been pulled high. The watchdog timer (WDT) can also reset the microcontroller in the event of a hardware or software error.

Fill unused areas of code with NOPs and periodic jumps to an error routine or RST instruction. This is particularly important in the code surrounding lookup tables, since accidentally executing from lookup tables will cause undesired results. Wherever space allows, surround each table with seven NOPs (because the longest device instruction has seven bytes) and a RST or a jump to an error routine. Since RST is a one-byte instruction, the NOPs are unnecessary if RSTs are used instead of jumps to an error routine. This will help to ensure a speedy recovery from a software error.

When using the watchdog timer (WDT) for software protection, we recommend that you reset the WDT from only one place in code, reducing the chance of an undesired WDT reset. The section of code that resets the WDT should monitor the other code sections for proper operation. This can be done by checking variables to make sure they are within reasonable values. Simply using a software timer to reset the WDT every 10 milliseconds, for example, will provide protection only for catastrophic failures.

4

Memory Partitions

CHAPTER 4
MEMORY PARTITIONS

This chapter describes the organization of the address space, its major partitions, and the 2-Mbyte and 64-Kbyte operating modes. *2-Mbyte* refers to the address space defined by the 21 external address pins. In 2-Mbyte mode, code can execute from almost anywhere in the 2-Mbyte space. In 64-Kbyte mode, code can execute only from the 64-Kbyte area FF0000–FFFFFFH. The 64-Kbyte mode provides compatibility with software written for previous 16-bit MCS® 96 microcontrollers. In either mode, nearly all of the 2-Mbyte address space is available for data storage.

Other topics covered in this chapter include the following:

- the relationship between the 2-Mbyte address space defined by the 21 external address pins and the 16-Mbyte address space defined by the 24 internal address bits
- extended and nonextended data accesses
- a *windowing* technique for accessing the upper register file and peripheral special-function registers (SFRs) with direct addressing
- a method for remapping the upper 7 Kbytes of internal ROM (FF2400–FF3FFFH)

4.1 MEMORY MAP OVERVIEW

With 24 internal address bits, the microcontroller can address 16 Mbytes of memory. However, only 21 of the 24 address bits are implemented by external pins: A20:0 in demultiplexed mode, or A20:16 and AD15:0 in multiplexed mode. If, for example, an internal 24-bit address is FF2080H, the 21 external-address pins output 1F2080H. Further, the address seen by an external device depends on how many of the extended address pins are connected to the device. (See “Internal and External Addresses” on page 15-1.)

The 21 external-address pins can address 2 Mbytes of external memory. For purposes of discussion only, it is convenient to view this 2-Mbyte address space as thirty-two 64-Kbyte pages, numbered 00H–1FH (see Figure 4-1 on page 4-2). The lower 16 address pins enable the microcontroller to address external page 00H. The five extended address pins enable the microcontroller to address the remaining external address space, pages 01H–1FH.

Because the 3 most-significant bits (MSBs) of the internal address can take any values without changing the external address, these 3 bits effectively produce 8 copies of the 2-Mbyte address space, for a total of 16 Mbytes in 256 pages, 00H–FFH (Figure 4-1). For example, page 01H has 7 duplicates: 21H, 41H, …, E1H, and page 11H has 7 duplicates: 31H, 51H, …, F1H. The shaded areas in Figure 4-1 represent the overlaid areas.

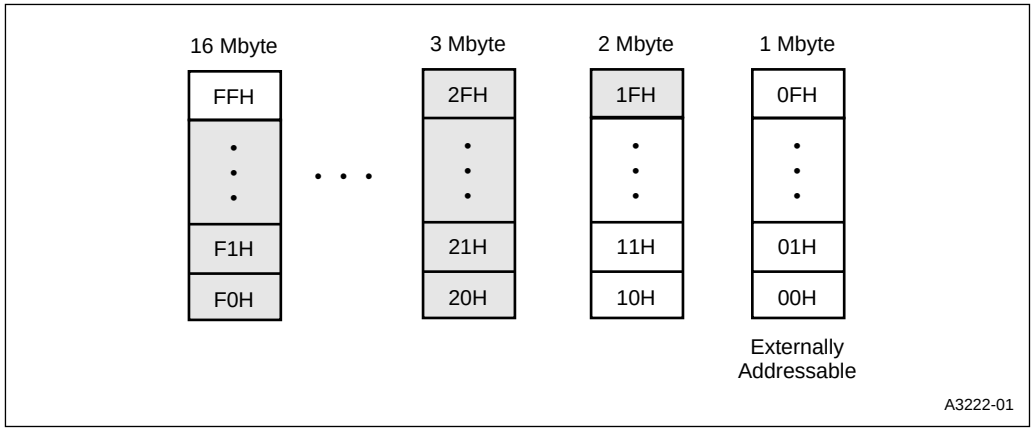


Figure 4-1. 16-Mbyte Address Space

The memory pages of interest are 00H–1EH and FFH. Pages 01H–1EH are external memory with unspecified contents; they can store either code or data. Pages 00H and FFH, shown in Figure 4-2, have special significance. Page 00H contains the register file and the special-function registers (SFRs), while page FFH contains special-purpose memory (chip configuration bytes and interrupt vectors) and program memory. The microcontroller fetches its first instruction from location FF2080H (or 1F2080H in external memory).

NOTE

Because the microcontroller has 24 bits of address internally, all programs must be written as though it uses all 24 bits. The microcontroller resets from page FFH, so all code must originate from this page. (Use the assembler directive, "cseg at 0FF*xxxx*H.") This is true even if your code is actually stored in external memory.

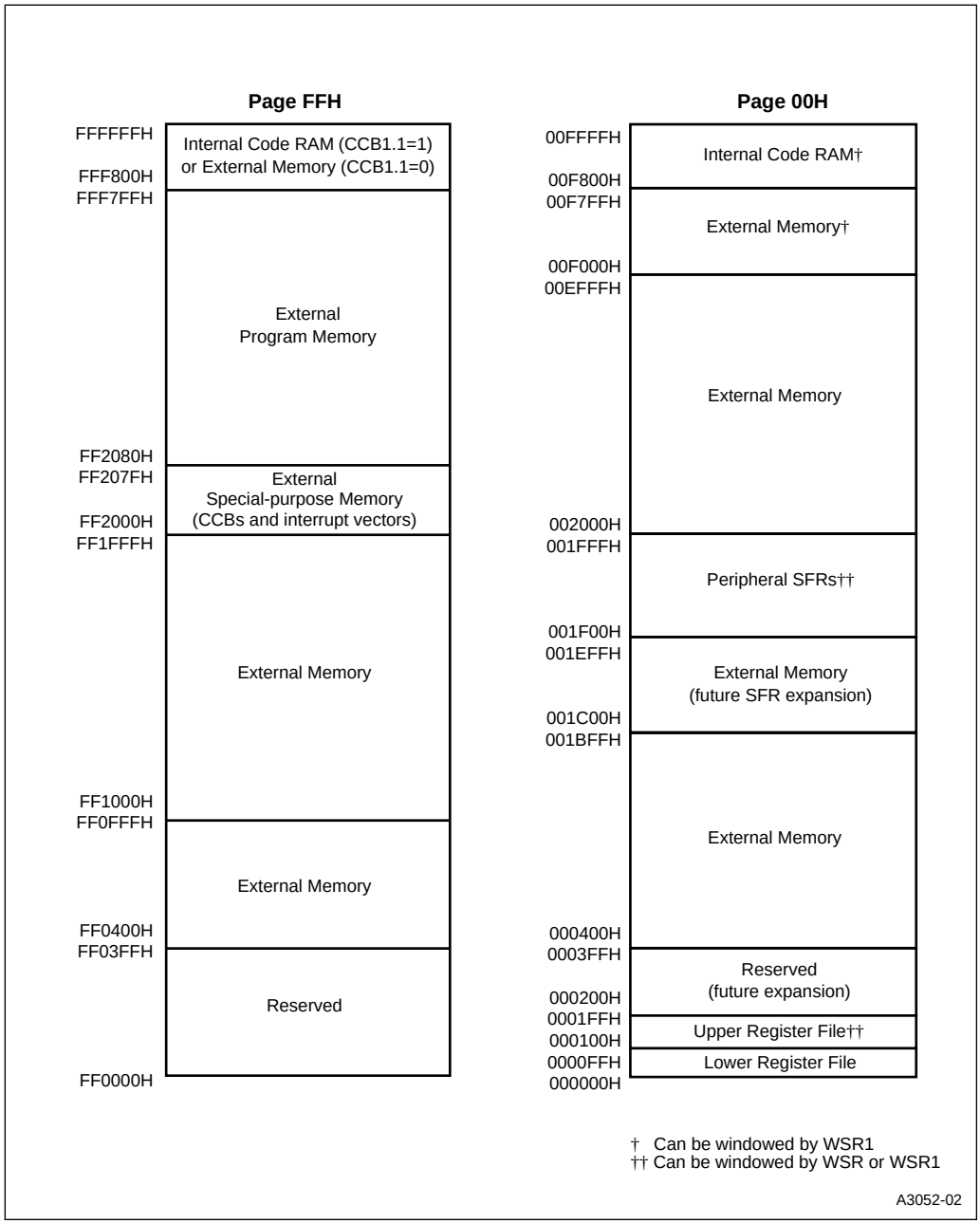


Figure 4-2. Pages FFH and 00H

4.2 MEMORY PARTITIONS

Table 4-1 is a memory map of the 8XC196EA. This section describes the partitions.

Table 4-1. 8XC196EA Memory Map

Hex Address	Description (Note 1, Note 2)	Addressing Modes
FFFFFF FF4000	External device (memory or I/O) connected to address/data bus	Indirect, indexed, extended
FF3FFF FF2400	Program memory (Note 3); can be mapped into page 00H (CCB1.2 = 1)	Indirect, indexed, extended
FF23FF FF2140	Program memory (Note 3)	Indirect, indexed, extended
FF213F FF20C0	Special-purpose memory (PIH vectors; Note 3)	Indirect, indexed, extended
FF20BF FF2080	Program memory (Note 3); (After reset, the first instruction is fetched from FF2080H.)	Indirect, indexed, extended
FF207F FF2000	Special-purpose memory (CCBs, interrupt vectors, PTS vectors); (Note 3)	Indirect, indexed, extended
FF1FFF FF1000	External device (memory or I/O) connected to address/data bus	Indirect, indexed, extended
FF0FFF FF0400	Internal code/data RAM (identically mapped from page 00H)	Indirect, indexed, extended
FF03FF FF0000	Reserved for in-circuit emulators	—
FEFFFF 1F0000	Overlaid memory (reserved for future microcontrollers); locations *x*F0000–*x*F03FFH are reserved for in-circuit emulators	Indirect, indexed, extended
1EFFFF 004000	External device (memory or I/O) connected to address/data bus	Indirect, indexed, extended
003FFF 002400	A copy of internal ROM (FF2400–FF3FFFH) if CCB1.2=0 External memory if CCB1.2=1 (Note 4)	Indirect, indexed, extended
0023FF 002000	External device (memory or I/O) connected to address/data bus	Indirect, indexed, extended
001FFF 001FFC	Memory-mapped special-function registers (SFRs); (Note 3)	Indirect, indexed, extended
001FFB 001FE0	Memory-mapped special-function registers (SFRs)	Indirect, indexed, extended
001FDF 001C00	Peripheral special-function registers (SFRs)	Indirect, indexed, extended, windowed direct
001BFF 001000	External device (memory or I/O) connected to address/data bus	Indirect, indexed, extended

NOTES:
1. Unless otherwise noted, write 0FFH to reserved memory locations and write 0 to reserved SFR bits.
2. The contents or functions of reserved locations may change in future microcontroller revisions, in which case a program that relies on one or more of these locations might not function properly.
3. External memory if EA# is low; internal ROM if EA# is high. (Only applies to the 83C196EA.)
4. When EA# is low, CCB1.2 is ignored. When EA# is high, CCB1.2 controls whether the upper 7-Kbyte section of ROM is mapped into page FFH only or into both pages FFH and 00H. (Only applies to the 83C196EA.)

Table 4-1. 8XC196EA Memory Map (Continued)

Hex Address	Description (Note 1, Note 2)	Addressing Modes
000FFF 000400	Internal code/data RAM (identically mapped into page FFH)	Indirect, indexed, extended
0003FF 000100	Upper register file (general-purpose register RAM)	Indirect, indexed, windowed direct
0000FF 00001A	Lower register file (general-purpose register RAM)	Direct, indirect, indexed
000019 000000	Lower register file (stack pointer and CPU SFRs)	Direct, indirect, indexed

NOTES:
1. Unless otherwise noted, write 0FFH to reserved memory locations and write 0 to reserved SFR bits.
2. The contents or functions of reserved locations may change in future microcontroller revisions, in which case a program that relies on one or more of these locations might not function properly.
3. External memory if EA# is low; internal ROM if EA# is high. (Only applies to the 83C196EA.)
4. When EA# is low, CCB1.2 is ignored. When EA# is high, CCB1.2 controls whether the upper 7-Kbyte section of ROM is mapped into page FFH only or into both pages FFH and 00H. (Only applies to the 83C196EA.)

4.2.1 External Memory

Several partitions in pages 00H and FFH and all of pages 01H–1EH are assigned to external memory (see Table 4-1). Data can be stored in any part of this memory. Instructions can be stored in any part of this memory in 2-Mbyte mode, but can be stored only in page FFH in 64-Kbyte mode. Chapter 15, “Interfacing with External Memory,” describes the external memory interface and shows examples of external memory configurations.

4.2.2 Internal ROM

The 83C196EA has 8 Kbytes of internal ROM located at FF2000–FF3FFFH. The upper 7-Kbyte section (FF2400–FF3FFFH) is used for storing your application code and data. This 7-Kbyte partition can be identically mapped into page 00H (002400–003FFFH).

The lower 1-Kbyte section (FF2000–FF23FFH) is used for storing boot code, chip configuration bytes (CCBs), interrupt vectors, peripheral interrupt handler (PIH) vectors, and peripheral transaction server (PTS) vectors. This 1-Kbyte partition **cannot** be mapped into page 00H.

During application development, you may need to use an external memory device to store code and data that will later reside in the internal ROM. To do so, hold the EA# pin low during reset, forcing all accesses to FF2000–FF3FFFH to the external memory device.

To optimize your application, you may want to make the data in FF2400–FF3FFFH accessible as *near data* in page 00H. To accomplish this, set the REMAP bit (CCB1.2) when you program the CCBs, and hold EA# high during reset. This combination forces all accesses to FF2000–FF3FFFH to the internal ROM and "shadows" the contents of FF2400–FF3FFFH into page 00H. Mapping the ROM contents into both pages 00H and FFH allows you to access data and constants as *near data* and *near constants* in page 00H and execute code in page FFH.

NOTE

The EA# input is effective only for accesses to the internal ROM (FF2000–FF3FFFH). For accesses to any other location, the state of EA# is irrelevant. The REMAP bit is effective only for accesses to FF2400–FF3FFFH and only when EA# is inactive. When EA# is active, execution is external and the REMAP bit is ignored. The REMAP bit is loaded from the CCB and the state of EA# is latched upon leaving reset; neither can be changed until the next reset.

4.2.2.1 Program Memory in Page FFH

The address space in page FFH is implemented by a combination of internal ROM, internal code RAM, and external memory devices. Nearly all of page FFH is available for storing executable code. A 1-Kbyte section (FF0000–FF03FFH) is reserved for in-circuit emulators, and 256 bytes (FF2000–FF207FH and FF20C0–FF23FFH) are used for special-purpose memory (chip configuration bytes and interrupt vectors). The 3-Kbyte partition FF0400–FF0FFFH is occupied by an identical copy of the internal code RAM from page 00H. Mapping the internal code RAM into both pages allows you to access data and constants as *near data* and *near constants* in page 00H and execute code in page FFH. The remainder of page FFH is available for storing code.

NOTE

We recommend that you write FFH (the opcode for the RST instruction) to unused program memory locations. This causes a reset if a program begins to execute in unused memory.

4.2.2.2 Special-purpose Memory in Page FFH

Special-purpose memory resides in locations FF2000–FF207FH and FF20C0–FF21FFH (Table 4-2). It contains several reserved memory locations, the chip configuration bytes (CCBs), and vectors for peripheral transaction server (PTS), peripheral interrupt handler (PIH), and standard interrupts.

Table 4-2. 8XC196EA Special-purpose Memory Addresses

Hex Address	Description
FF213F FF2100	PIH 1 vectors †
FF20FF FF20C0	PIH 0 vectors †
FF20BF FF2080	Program memory †
FF207F FF205E	Reserved (each byte must contain FFH)
FF205D FF2040	PTS vectors
FF203F FF2030	Upper interrupt vectors
FF202F FF2020	Security key
FF2019 FF201C	Reserved (each byte must contain FFH)
FF201B	Reserved (must contain 20H)
FF201A	CCB1
FF2019	Reserved (must contain 20H)
FF2018	CCB0
FF2017 FF2016	PIH dummy PTS vector (point to ZERO_REG; 0000H)
FF2015 FF2014	PIH dummy interrupt vector (point to a RET instruction)
FF2013 FF2010	Reserved (each byte must contain FFH)
FF200F FF2000	Lower interrupt vectors

† The PIH interrupt vectors are located above the program startup address (FF2080H). Your code must branch around the vector table (FF20C0–FF213FH). The jump instruction must occur before address FF20C0H and branch to or past address FF2140H.

4.2.2.3 Reserved Memory Locations

Several memory locations are reserved for testing or for use in future products. Do not read or write these locations except to initialize them to the values shown in Table 4-2. The function or contents of these locations may change in future revisions; software that uses reserved locations may not function properly.

4.2.2.4 Interrupt, PIH, and PTS Vectors

The peripheral transaction server (PTS) vectors must contain the addresses of the PTS control blocks. The peripheral interrupt handler (PIH) vectors must contain the addresses of interrupt service routines that are to service PIH interrupt requests. The upper and lower interrupt vectors must contain the addresses of interrupt service routines for the interrupt controller. See Table 6.3, "Interrupt Sources, Priorities, and Vector Addresses," on page 6-5 for more information.

4.2.2.5 Chip Configuration Bytes

The chip configuration bytes (CCB0 and CCB1) specify the operating environment. They specify the bus width, bus mode (multiplexed or demultiplexed), write-control mode, wait states, power-down enabling, and the operating mode (2-Mbyte or 64-Kbyte mode). CCB1 also controls ROM remapping.

The chip configuration bytes are the first bytes fetched from memory when the microcontroller leaves the reset state. The post-reset sequence loads the CCBs into the chip configuration registers (CCRs). Once they are loaded, the CCRs cannot be changed until the next reset. Typically, the CCBs are programmed once when your program is compiled and are not redefined during normal operation. (See Figure 4-8 on page 4-27 and Figure 4-9 on page 4-30 for descriptions of the CCBs and CCRs.)

4.2.3 Internal RAM (Code RAM)

The 8XC196EA has 3 Kbytes of internal RAM at 000400–000FFFH. Although it is called *code RAM* to distinguish it from *register RAM*, this internal RAM can store both executable code and data. This memory is typically used for the stack or for time-critical code and data.

This partition is mapped identically into page FFH. Mapping this partition into both pages allows you to access *near constants* and *near data* in page 00H and execute code in page FFH. The code RAM is accessed through the memory controller, so code executes as it would from external memory with zero wait states. Data stored in this area must be accessed with indirect or indexed addressing, so data accesses to this area take longer than data accesses to the register RAM. The code RAM cannot be windowed.

During application development, you may need to use external memory to store code and data that will later reside in the internal code RAM. The IRAM_CON register (Figure 4-3) provides a simple method for handling this situation.

IRAM_CON Address: 1FE0H
Reset State: 00H

The internal RAM control (IRAM_CON) register has two functions related to memory accesses. The IRAM bit allows you to control access to locations 000400–000FFH and FF0400–FF0FFH. The EA_STAT bit allows you to determine the status of the EA# pin, which controls access to locations FF2000–FFFFFH. (IRAM_CON is a memory-mapped SFR. It must be accessed using indirect or indexed addressing; it cannot be windowed.)

7							0
EA_STAT	IRAM	—	—	—	—	—	—

Bit Number	Bit Mnemonic	Function
7	EA_STAT	EA# Status: This **read-only** bit contains the complement of the EA# pin, which controls whether accesses to locations FF2000–FFFFFH are directed to the internal ROM or to external memory. 0 = the EA# pin is inactive (accesses are directed to the ROM) 1 = the EA# pin is active (accesses are directed to external memory) (describes additional options for ROM access.)
6	IRAM	Internal RAM Control: This bit controls whether accesses to locations 000400–000FFH and FF0400–FF0FFH are directed to internal code RAM or to external memory. 0 = use the internal code RAM 1 = use external memory
5:0	—	Reserved; for compatibility with future devices, write zeros to these bits.

Figure 4-3. Internal RAM Control (IRAM_CON) Register

4.2.4 Special-function Registers (SFRs)

The 8XC196EA has both peripheral SFRs and memory-mapped SFRs. The peripheral SFRs are physically located in the on-chip peripherals, and they can be windowed (see "Windowing" on page 4-17). The memory-mapped SFRs must be accessed using indirect or indexed addressing modes and **cannot** be windowed.

Do not use reserved SFRs; write zeros to them or leave them in their default state. When read, reserved bits and reserved SFRs return undefined values.

NOTE

Using any SFR as a base or index register for indirect or indexed operations can cause unpredictable results. External events can change the contents of SFRs, and some SFRs are cleared when read. For this reason, consider the implications of using an SFR as an operand in a read-modify-write instruction (e.g., XORB).

4.2.4.1 Memory-mapped SFRs

Locations 1FE0–1FFFH contain memory-mapped SFRs (Table 4-3). The memory-mapped SFRs must be accessed from page 00H with indirect or indexed addressing modes, and they cannot be windowed. If you read a location in this range through a window, the SFR **appears** to contain FFH (all ones). If you write a location in this range through a window, the write operation has **no effect** on the SFR.

Table 4-3. Memory-mapped SFRs

Ports 3, 4, 5 SFRs		
Address	**High (Odd) Byte**	**Low (Even) Byte**
1FFEH	P4_PIN	P3_PIN
1FFCH	P4_REG	P3_REG
1FFAH	Reserved	Reserved
1FF8H	Reserved	Reserved
1FF6H	P5_PIN	Reserved
1FF4H	P5_REG	P34_DRV
1FF2H	P5_DIR	Reserved
1FF0H	P5_MODE	Reserved

EPORT, Port 12, and Internal RAM SFRs		
Address	**High (Odd) Byte**	**Low (Even) Byte**
1FEEH	Reserved	P12_PIN
1FECH	Reserved	P12_REG
1FEAH	Reserved	P12_DIR
1FE8H	Reserved	P12_MODE
1FE6H	EP_PIN	Reserved
1FE4H	EP_REG	Reserved
1FE2H	EP_DIR	Reserved
1FE0H	EP_MODE	IRAM_CON

4.2.4.2 Peripheral SFRs

Locations 1C00–1FDFH provide access to the peripheral SFRs (see Table 4-4). Locations in this range marked “Reserved” and locations that are omitted from the table are reserved for future expansion. Locations marked “Reserved for Intel” are used for various test functions; your code should neither write to nor read from these locations. The peripheral SFRs are I/O control registers; they are physically located in the on-chip peripherals. Peripheral SFRs can be windowed and they can be addressed as bytes, except as noted in the table.

Table 4-4. 8XC196EA Peripheral SFRs

Ports 2 and 7–11 SFRs		
Address	**High (Odd) Byte**	**Low (Even) Byte**
1FDEH	Reserved	Reserved
…	…	…
1FDAH	Reserved	Reserved
1FD6H	Reserved	P2_PIN
1FD4H	Reserved	P2_REG
1FD2H	Reserved	P2_DIR
1FD0H	Reserved	P2_MODE
1FCEH	P8_PIN	P7_PIN
1FCCH	P8_REG	P7_REG
1FCAH	P8_DIR	P7_DIR
1FC8H	P8_MODE	P7_MODE
1FC6H	P10_PIN	P9_PIN
1FC4H	P10_REG	P9_REG
1FC2H	P10_DIR	P9_DIR
1FC0H	P10_MODE	P9_MODE
1FBEH	Reserved	P11_PIN
1FBCH	Reserved	P11_REG
1FBAH	Reserved	P11_DIR
1FB8H	Reserved	P11_MODE
1FB6H	Reserved	Reserved
…	…	…
1FA6H	Reserved	Reserved for Intel
Reset Source SFR		
Address	**High (Odd) Byte**	**Low (Even) Byte**
1FA4H	Reserved	RSTSRC
Stack Overflow Module SFRs		
Address	**High (Odd) Byte**	**Low (Even) Byte**
1FA2H	STACK_TOP (H)	STACK_TOP (L)
1FA0H	STACK_BOTTOM (H)	STACK_BOTTOM (L)
Serial I/O Port 1 (SIO1) SFRs		
Address	**High (Odd) Byte**	**Low (Even) Byte**
1F9EH	Reserved for Intel	Reserved for Intel
1F9CH	SP1_BAUD (H)	SP1_BAUD (L)
1F9AH	SP1_CON	SBUF1_TX
1F98H	SP1_STATUS	SBUF1_RX
Synchronous Serial I/O Port (SSIO) SFRs		
Address	**High (Odd) Byte**	**Low (Even) Byte**
1F96H	SSIO1_CLK	Reserved
1F94H	SSIO0_CLK	SSIO_BAUD
1F92H	SSIO1_CON	SSIO1_BUF
1F90H	SSIO0_CON	SSIO0_BUF

Serial I/O Port 0 (SIO0) SFRs		
Address	**High (Odd) Byte**	**Low (Even) Byte**
1F8EH	Reserved for Intel	Reserved for Intel
1F8CH	SP0_BAUD (H)	SP0_BAUD (L)
1F8AH	SP0_CON	SBUF0_TX
1F88H	SP0_STATUS	SBUF0_RX
1F86H	Reserved	Reserved
…	…	…
1F82H	Reserved	Reserved
CLKOUT Control SFR		
Address	**High (Odd) Byte**	**Low (Even) Byte**
1F80H	Reserved	CLKOUT_CON
Event Processor Array (EPA) SFRs		
Address	**High (Odd) Byte**	**Low (Even) Byte**
†1F7EH	TIMER1 (H)	TIMER1 (L)
†1F7CH	T1CONTROL (H)	T1CONTROL (L)
†1F7AH	TIMER2 (H)	TIMER2 (L)
†1F78H	T2CONTROL (H)	T2CONTROL (L)
†1F76H	TIMER3 (H)	TIMER3 (L)
†1F74H	T3CONTROL (H)	T3CONTROL (L)
†1F72H	TIMER4 (H)	TIMER4 (L)
†1F70H	T4CONTROL (H)	T4CONTROL (L)
1F6EH	Reserved	TIMER_MUX
1F6CH	Reserved	Reserved
…	…	…
1F6AH	Reserved	Reserved
1F68H	Reserved	Reserved
…	…	…
1F60H	Reserved	Reserved
†1F5EH	EPA0_TIME (H)	EPA0_TIME (L)
†1F5CH	EPA0_CON (H)	EPA0_CON (L)
†1F5AH	EPA1_TIME (H)	EPA1_TIME (L)
†1F58H	EPA1_CON (H)	EPA1_CON (L)
†1F56H	EPA2_TIME (H)	EPA2_TIME (L)
†1F54H	EPA2_CON (H)	EPA2_CON (L)
†1F52H	EPA3_TIME (H)	EPA3_TIME (L)
†1F50H	EPA3_CON (H)	EPA3_CON (L)
†1F4EH	EPA4_TIME (H)	EPA4_TIME (L)
†1F4CH	EPA4_CON (H)	EPA4_CON (L)
†1F4AH	EPA5_TIME (H)	EPA5_TIME (L)
†1F48H	EPA5_CON (H)	EPA5_CON (L)
†1F46H	EPA6_TIME (H)	EPA6_TIME (L)
†1F44H	EPA6_CON (H)	EPA6_CON (L)

† Must be addressed as a word.

Table 4-4. 8XC196EA Peripheral SFRs (Continued)

Event Processor Array (EPA) SFRs (Cont'd)		
Address	**High (Odd) Byte**	**Low (Even) Byte**
†1F42H	EPA7_TIME (H)	EPA7_TIME (L)
†1F40H	EPA7_CON (H)	EPA7_CON (L)
†1F3EH	EPA8_TIME (H)	EPA8_TIME (L)
†1F3CH	EPA8_CON (H)	EPA8_CON (L)
†1F3AH	EPA9_TIME (H)	EPA9_TIME (L)
†1F38H	EPA9_CON (H)	EPA9_CON (L)
†1F36H	EPA10_TIME (H)	EPA10_TIME (L)
†1F34H	EPA10_CON (H)	EPA10_CON (L)
†1F32H	EPA11_TIME (H)	EPA11_TIME (L)
†1F30H	EPA11_CON (H)	EPA11_CON (L)
†1F2EH	EPA12_TIME (H)	EPA12_TIME (L)
†1F2CH	EPA12_CON (H)	EPA12_CON (L)
†1F2AH	EPA13_TIME (H)	EPA13_TIME (L)
†1F28H	EPA13_CON (H)	EPA13_CON (L)
†1F26H	EPA14_TIME (H)	EPA14_TIME (L)
†1F24H	EPA14_CON (H)	EPA14_CON (L)
†1F22H	EPA15_TIME (H)	EPA15_TIME (L)
†1F20H	EPA15_CON (H)	EPA15_CON (L)
†1F1EH	EPA16_TIME (H)	EPA16_TIME (L)
†1F1CH	EPA16_CON (H)	EPA16_CON (L)
1F1AH	Reserved	Reserved
…	…	…
1F08H	Reserved	Reserved
1F06H	Reserved	Reserved
1F04H	Reserved	Reserved
1F02H	Reserved	Reserved
1F00H	Reserved	Reserved
†1EFEH	OS0_TIME (H)	OS0_TIME (L)
†1EFCH	OS0_CON (H)	OS0_CON (L)
†1EFAH	OS1_TIME (H)	OS1_TIME (L)
†1EF8H	OS1_CON (H)	OS1_CON (L)
†1EF6H	OS2_TIME (H)	OS2_TIME (L)
†1EF4H	OS2_CON (H)	OS2_CON (L)
†1EF2H	OS3_TIME (H)	OS3_TIME (L)
†1EF0H	OS3_CON (H)	OS3_CON (L)
†1EEEH	OS4_TIME (H)	OS4_TIME (L)
†1EECH	OS4_CON (H)	OS4_CON (L)
†1EEAH	OS5_TIME (H)	OS5_TIME (L)
†1EE8H	OS5_CON (H)	OS5_CON (L)
†1EE6H	OS6_TIME (H)	OS6_TIME (L)
†1EE4H	OS6_CON (H)	OS6_CON (L)
†1EE2H	OS7_TIME (H)	OS7_TIME (L)
†1EE0H	OS7_CON (H)	OS7_CON (L)

Pulse-width Modulator (PWM) SFRs		
Address	**High (Odd) Byte**	**Low (Even) Byte**
1EDEH	PWM0_1_PERIOD	PWM0_CONTROL
1EDCH	PWM0_1_COUNT	PWM1_CONTROL
1EDAH	PWM2_3_PERIOD	PWM2_CONTROL
1ED8H	PWM2_3_COUNT	PWM3_CONTROL
1ED6H	PWM4_5_PERIOD	PWM4_CONTROL
1ED4H	PWM4_5_COUNT	PWM5_CONTROL
1ED2H	PWM6_7_PERIOD	PWM6_CONTROL
1ED0H	PWM6_7_COUNT	PWM7_CONTROL
1ECEH	Reserved	Reserved
…	…	…
1ECAH	Reserved	Reserved
1EC8H	Reserved for Intel	Reserved for Intel
…	…	…
1EC0H	Reserved for Intel	Reserved for Intel
1EBEH	Reserved	Reserved
…	…	…
1EACH	Reserved	Reserved
Peripheral Interrupt Handler (PIH) SFRs		
Address	**High (Odd) Byte**	**Low (Even) Byte**
1EAAH	PIH1_INT_PEND (H)	PIH1_INT_PEND (L)
1EA8H	PIH1_INT_MASK (H)	PIH1_INT_MASK (L)
1EA6H	PIH1_PTSSEL (H)	PIH1_PTSSEL (L)
1EA4H	PIH1_PTSSRV (H)	PIH1_PTSSRV (L)
1EA2H	PIH1_VEC_BASE (H)	PIH1_VEC_BASE (L)
1EA0H	Reserved	PIH1_VEC_IDX
1E9EH	Reserved	Reserved
1E9CH	Reserved	Reserved
1E9AH	PIH0_INT_PEND (H)	PIH0_INT_PEND (L)
1E98H	PIH0_INT_MASK (H)	PIH0_INT_MASK (L)
1E96H	PIH0_PTSSEL (H)	PIH0_PTSSEL (L)
1E94H	PIH0_PTSSRV (H)	PIH0_PTSSRV (L)
1E92H	PIH0_VEC_BASE (H)	PIH0_VEC_BASE (L)
1E90H	Reserved	PIH0_VEC_IDX
Chip-select 1 and 2 SFRs		
Address	**High (Odd) Byte**	**Low (Even) Byte**
1E8EH	Reserved	Reserved
1E8CH	Reserved	BUSCON2
1E8AH	ADDRMSK2 (H)	ADDRMSK2 (L)
1E88H	ADDRCOM2 (H)	ADDRCOM2 (L)
1E86H	Reserved	Reserved
1E84H	Reserved	BUSCON1
1E82H	ADDRMSK1 (H)	ADDRMSK1 (L)
1E80H	ADDRCOM1 (H)	ADDRCOM1 (L)

† Must be addressed as a word.

Table 4-4. 8XC196EA Peripheral SFRs (Continued)

Chip-select 0 and A/D Converter SFRs			Chip-select 0 and A/D Converter SFRs (Cont'd)		
Address	**High (Odd) Byte**	**Low (Even) Byte**	**Address**	**High (Odd) Byte**	**Low (Even) Byte**
1E7EH	Reserved for Intel	Reserved for Intel	1E66H	AD_RESULT11 (H)	AD_RESULT11 (L)
1E7CH	Reserved	BUSCON0	1E64H	AD_RESULT10 (H)	AD_RESULT10 (L)
1E7AH	ADDRMSK0 (H)	ADDRMSK0 (L)	1E62H	AD_RESULT9 (H)	AD_RESULT9 (L)
1E78H	ADDRCOM0 (H)	ADDRCOM0 (L)	1E60H	AD_RESULT8 (H)	AD_RESULT8 (L)
1E76H	AD_TIME	AD_TEST0	1E5EH	AD_RESULT7 (H)	AD_RESULT7 (L)
1E74H	Reserved for Intel	AD_COMMAND	1E5CH	AD_RESULT6 (H)	AD_RESULT6 (L)
1E72H	AD_RESULT (H)	AD_RESULT (L)	1E5AH	AD_RESULT5 (H)	AD_RESULT5 (L)
1E70H	AD_SCAN (H)	AD_SCAN (L)	1E58H	AD_RESULT4 (H)	AD_RESULT4 (L)
1E6EH	AD_RESULT15 (H)	AD_RESULT15 (L)	1E56H	AD_RESULT3 (H)	AD_RESULT3 (L)
1E6CH	AD_RESULT14 (H)	AD_RESULT 14 (L)	1E54H	AD_RESULT2 (H)	AD_RESULT2 (L)
1E6AH	AD_RESULT13 (H)	AD_RESULT13 (L)	1E52H	AD_RESULT1 (H)	AD_RESULT1 (L)
1E68H	AD_RESULT12 (H)	AD_RESULT12 (L)	1E50H	AD_RESULT0 (H)	AD_RESULT0 (L)

† Must be addressed as a word.

4.2.5 Register File

The register file is divided into an upper register file and a lower register file (Figure 4-4). The upper register file consists of general-purpose register RAM. The lower register file contains additional general-purpose register RAM along with the stack pointer (SP) and the CPU special-function registers (SFRs).

Table 4-5 on page 4-15 lists the register file memory addresses. The RALU accesses the lower register file directly, without the use of the memory controller. It also accesses a *windowed* location directly (see "Windowing" on page 4-17). Only the upper register file and the peripheral SFRs can be windowed. Registers in the lower register file and registers being windowed can be accessed with direct addressing.

NOTE

The register file must not contain code. An attempt to execute an instruction from a location in the register file causes the memory controller to fetch the instruction from external memory.

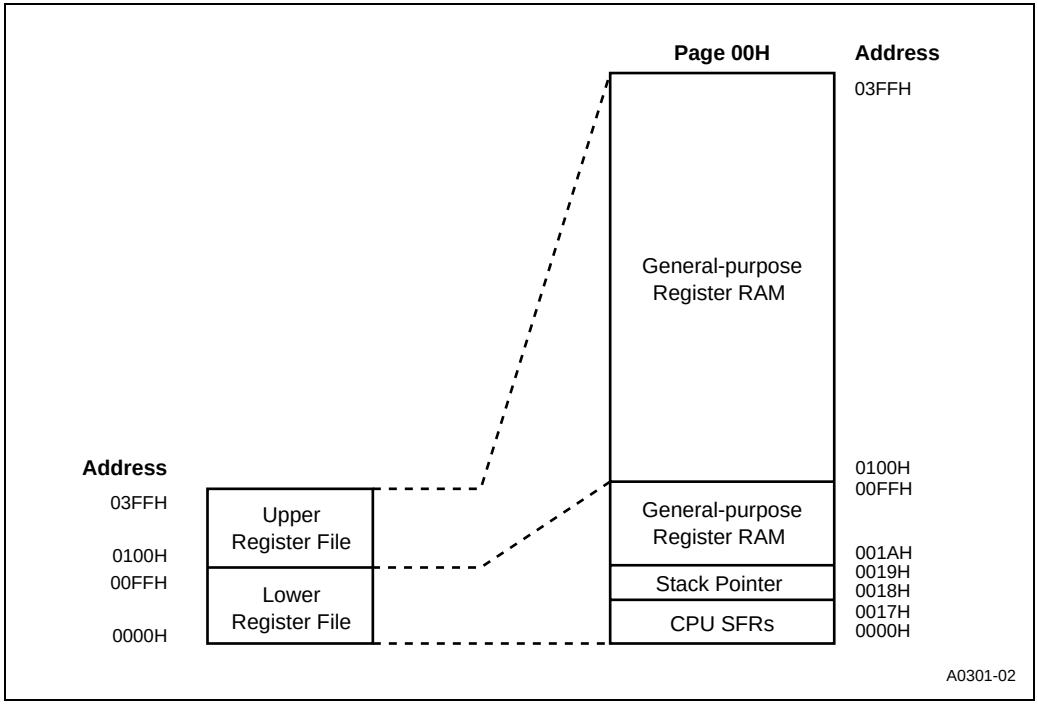


Figure 4-4. Register File Memory Map

Table 4-5. Register File Memory Addresses

Address Range	Description	Addressing Modes
03FFH 0100H	General-purpose register RAM; upper register file	Indirect, indexed, extended, windowed direct
00FFH 001AH	General-purpose register RAM; lower register file	Direct, indirect, indexed, extended
0019H 0018H	Stack pointer (SP); lower register file	Direct
0017H 0000H	CPU special-function registers (SFRs); lower register file	Direct

4.2.5.1 General-purpose Register RAM

The lower register file contains general-purpose register RAM. The stack pointer locations can also be used as general-purpose register RAM when stack operations are not being performed. The RALU can access this memory directly, using direct addressing.

The upper register file also contains general-purpose register RAM. The RALU normally uses indirect or indexed addressing to access the RAM in the upper register file. Windowing enables the RALU to use direct addressing to access this memory, providing fast context switching of interrupt tasks and faster program execution. (Refer to Chapter 3, "Programming Considerations," for a discussion of addressing modes, and see "Windowing" on page 4-17 for details on windowing.) PTS control blocks and the stack are most efficient when located in the internal code RAM or in the upper register file.

4.2.5.2 Stack Pointer (SP)

Memory locations 0018H and 0019H contain the stack pointer (SP). The SP contains the address of the stack. The SP must point to a word (even) address that is two bytes (for 64-Kbyte mode) or four bytes (for 2-Mbyte mode) greater than the desired starting address. Before the CPU executes a subroutine call or interrupt service routine, it decrements the SP (by two in 64-Kbyte mode; by four in 2-Mbyte mode). Next, it copies (PUSHes) the address of the next instruction from the program counter onto the stack. It then loads the address of the subroutine or interrupt service routine into the program counter. When it executes the return-from-subroutine (RET) instruction at the end of the subroutine or interrupt service routine, the CPU loads (POPs) the contents of the top of the stack (that is, the return address) into the program counter. Finally, it increments the SP (by two in 64-Kbyte mode; by four in 2-Mbyte mode).

Subroutines may be nested. That is, each subroutine may call others. The CPU PUSHes the contents of the program counter onto the stack each time it executes a subroutine call. The stack grows downward as entries are added. The only limit to the nesting depth is the amount of available memory. As the CPU returns from each nested subroutine, it POPs the address off the top of the stack, and the next return address moves to the top of the stack.

Your program must load a word-aligned (even) address into the stack pointer. Select an address that is two bytes (for 64-Kbyte mode) or four bytes (for 2-Mbyte mode) greater than the desired starting address because the CPU automatically decrements the stack pointer before it pushes the first byte of the return address onto the stack. Remember that the stack grows downward, so allow sufficient room for the maximum number of stack entries. The stack must be located in page 00H, in either the internal register file, the internal code RAM, or external RAM. The stack can be used most efficiently when it is located in the upper register file or internal code RAM.

The following example initializes the top of the upper register file as the stack.

```
LD   SP, #400H          ;Stack begins at 03FEH and grows downward
```

The 8XC196EA features a stack overflow module, which monitors the stack pointer and generates a nonmaskable interrupt if it crosses the upper or lower boundary you have specified. Refer to Chapter 5, “Stack Overflow Module,” for details.

4.2.5.3 CPU Special-function Registers (SFRs)

Locations 0000–0017H in the lower register file are the CPU SFRs (Table 4-6). Appendix C describes the CPU SFRs.

Table 4-6. CPU SFRs

Address	High (Odd) Byte	Low (Even) Byte
0016H	Reserved	Reserved
0014H		WSR
0012H	INT_MASK1	INT_PEND1
0010H	Reserved	Reserved
000EH	Reserved	Reserved
000CH	Reserved	Reserved
000AH	Reserved	WATCHDOG
0008H	INT_PEND	INT_MASK
0006H	PTSSRV (H)	PTSSRV (L)
0004H	PTSSEL (H)	PTSSEL (L)
0002H	ONES_REG (H)	ONES_REG (L)
0000H	ZERO_REG (H)	ZERO_REG (L)

4.3 WINDOWING

Windowing expands the amount of memory that is accessible with direct addressing. Direct addressing can access the lower register file with short, fast-executing instructions. With windowing, direct addressing can also access the upper register file and peripheral SFRs.

NOTE

Memory-mapped SFRs must be accessed using indirect or indexed addressing modes; they cannot be windowed. Reading a memory-mapped SFR through a window returns FFH (all ones). Writing to a memory-mapped SFR through a window has no effect.

Windowing maps a segment of higher memory (the upper register file or peripheral SFRs) into the lower register file. The 8XC196EA has two window selection registers, WSR and WSR1. WSR selects a 32-, 64-, or 128-byte segment of higher memory to be windowed into the top of the lower register file space. WSR1 selects a 32- or 64-byte segment of higher memory to be mapped into the middle of the lower register file.

Because the areas in the lower register file do not overlap, two windows can be in effect at the same time. This allows you to directly address a block of peripheral SFRs in one window and a block of register RAM in another. For example, you can activate a 128-byte window using WSR and a 64-byte window using WSR1 (Figure 4-5). These two windows occupy locations 0040–00FFH in the lower register file, leaving locations 001A–003FH for use as general-purpose register RAM, locations 0018–0019H for the stack pointer or general-purpose register RAM, and locations 0000–0017H for the CPU SFRs.

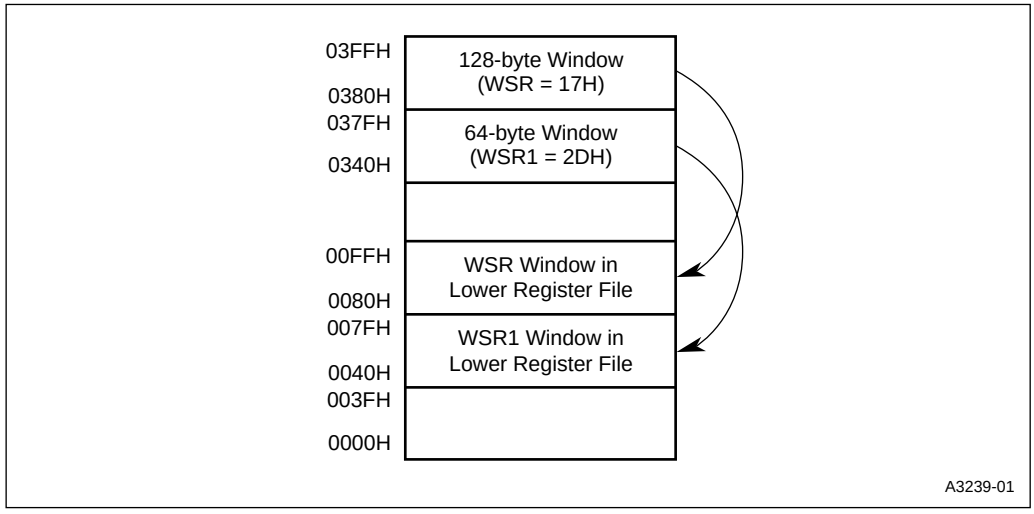


Figure 4-5. Windowing

4.3.1 Selecting a Window

The window selection register (Figure 4-6) has two functions. The HLDEN bit (WSR.7) enables and disables the bus-hold protocol (see Chapter 15, "Interfacing with External Memory"); it is unrelated to windowing. The remaining bits select a window to be mapped into the top of the lower register file. Table 4-7 on page 4-20 provides a quick reference of WSR values for windowing the peripheral SFRs. Table 4-8 on page 4-21 lists the WSR values for windowing the upper register file.

WSR

Address: 0014H
Reset State: 00H

The window selection register (WSR) has two functions. One bit enables and disables the bus-hold protocol. The remaining bits select windows. Windows map sections of RAM into the top of the lower register file, in 32-, 64-, or 128-byte increments. PUSHA saves this register on the stack and POPA restores it.

							0
HLDEN	W6	W5	W4	W3	W2	W1	W0

Bit Number	Bit Mnemonic	Function
7	HLDEN	HOLD#, HLDA# Protocol Enable This bit enables and disables the bus-hold protocol (see Chapter 15, “Interfacing with External Memory”). It has no effect on windowing. 0 = disable 1 = enable
6:0	W6:0	Window Selection These bits specify the window size and number. See Table 4-7 on page 4-20 for peripheral SFR windows or Table 4-8 on page 4-21 for upper register file windows.

Figure 4-6. Window Selection (WSR) Register

WSR1

Address: 0015H
Reset State: 00H

Window selection 1 (WSR1) register selects a 32- or 64-byte segment of the upper register file or peripheral SFRs to be windowed into the middle of the lower register file.

NOTE: The PUSHA and POPA instructions do **not** save and restore WSR1.

							0
	W6	W5	W4	W3	W2	W1	W0

Bit Number	Bit Mnemonic	Function
7	—	Reserved; for compatibility with future devices, write zero to this bit.
6:0	W6:0	Window Selection These bits specify the window size and number. See Table 4-7 on page 4-20 for peripheral SFR windows or Table 4-8 on page 4-21 for upper register file windows.

Figure 4-7. Window Selection 1 (WSR1) Register

Table 4-7. Selecting a Window of Peripheral SFRs

Peripherals	SFR Locations (Hex)	WSR or WSR1 Value for 32-byte Window (00E0–00FFH or 0060–007FH)	WSR or WSR1 Value for 64-byte Window (00C0–00FFH or 0040–007FH)	WSR Value for 128-byte Window (0080–00FFH)
Ports 3–5, 12, EPORT, memory access control†	1FE0–1FFF†	†		
Ports 2, 7, 8, 9, 10	1FC0–1FDF	7EH	3FH†	
Port 11, reset source	1FA0–1FBF	7DH		
SIO, SSIO	1F80–1F9F	7CH	3EH	1FH†
Timers, clockout control	1F60–1F7F	7BH		
EPA0–7	1F40–1F5F	7AH	3DH	
EPA8–15	1F20–1F3F	79H	3CH	
EPA16	1F00–1F1F	78H		1EH
OS0–7	1EE0–1EFF	77H		
PWM	1EC0–1EDF	76H	3BH	
PIH1	1EA0–1EBF	75H		
Chip-select 1 & 2, PIH0	1E80–1E9F	74H	3AH	1DH
Chip-select 0, A/D control, A/D 8–15 result	1E60–1E7F	73H		
A/D 0–7 result	1E40–1E5F	72H	39H	1CH

† Locations 1FE0–1FFFH contain memory-mapped SFRs that cannot be accessed through windows. Reading these locations through a window returns FFH; writing these locations through a window has no effect. Memory-mapped SFRs must be accessed with indirect or indexed addressing.

Table 4-8. Selecting a Window of the Upper Register File

Register RAM Locations (Hex)	WSR or WSR1 Value for 32-byte Window (00E0–00FFH or 0060–007FH)	WSR or WSR1 Value for 64-byte Window (00C0–00FFH or 0040–007FH)	WSR Value for 128-byte Window (0080–00FFH)
03E0–03FF	5FH		
03C0–03DF	5EH	2FH	
03A0–03BF	5DH		
0380–039F	5CH	2EH	17H
0360–037F	5BH		
0340–035F	5AH	2DH	
0320–033F	59H		
0300–031F	58H	2CH	16H
02E0–02FF	57H		
02C0–02DF	56H	2BH	
02A0–02BF	55H		
0280–029F	54H	2AH	15H
0260–027F	53H		
0240–025F	52H	29H	
0220–023F	51H		
0200–021F	50H	28H	14H
01E0–01FF	4FH		
01C0–01DF	4EH	27H	
01A0–01BF	4DH		
0180–019F	4CH	26H	13H
0160–017F	4BH		
0140–015F	4AH	25H	
0120–013F	49H		
0100–011F	48H	24H	12H

4.3.2 Addressing a Location Through a Window

After you have selected the desired window, you need to know the direct address of the memory location (the address in the lower register file). For SFRs, refer to the WSR tables in Appendix C. For register file locations, calculate the direct address as follows:

1. Subtract the base address of the area to be remapped (from Table 4-8) from the address of the desired location. This gives you the offset of that particular location.
2. Add the offset to the base address of the window (from Table 4-9). The result is the direct address.

Table 4-9. Windowed Base Addresses

Window Size	WSR Windowed Base Address (Base Address in Lower Register File)	WSR1 Windowed Base Address (Base Address in Lower Register File)
32-byte	00E0H	0060H
64-byte	00C0H	0040H
128-byte	0080H	—

Appendix C includes a table of the windowable SFRs with the window selection register values and direct addresses for each window size. The following examples explain how to determine the WSR value and direct address for any windowable location. An additional example shows how to set up a window by using the linker locator.

4.3.2.1 32-byte Windowing Example

Assume that you wish to access location 014BH (a location in the upper register file used for general-purpose register RAM) with direct addressing through a 32-byte window. Table 4-8 on page 4-21 shows that you need to write 4AH to the window selection register. It also shows that the base address of the 32-byte memory area is 0140H. To determine the offset, subtract that base address from the address to be accessed (014BH – 0140H = 000BH). Add the offset to the base address of the window in the lower register file (from Table 4-9). The direct address is 00EBH (000BH + 00E0H) for a WSR window or 006BH (000BH + 0060H) for a WSR1 window.

4.3.2.2 64-byte Windowing Example

Assume that you wish to access the SFR at location 1F8CH with direct addressing through a 64-byte window. Table 4-8 on page 4-21 shows that you need to write 3EH to the window selection register. It also shows that the base address of the 64-byte memory area is 1F80H. To determine the offset, subtract that base address from the address to be accessed (1F8CH – 1F80H = 000CH). Add the offset to the base address of the window in the lower register file (from Table 4-9). The direct address is 00CCH (000CH + 00C0H) for a WSR window or 004CH (000CH + 0040H) for a WSR1 window.

4.3.2.3 128-byte Windowing Example

Assume that you wish to access the SFR at location 1F82H with direct addressing through a 128-byte window. Table 4-8 on page 4-21 shows that you need to write 1FH to the window selection register. It also shows that the base address of the 128-byte memory area is 1F80H. To determine the offset, subtract that base address from the address to be accessed (1F82H – 1F80H = 0002H). Add the offset to the base address of the window in the lower register file (from Table 4-9). The direct address is 0082H (0002H + 0080H).

4.3.2.4 Unsupported Locations Windowing Example

Assume that you wish to access location 1FE7H (the EP_PIN register, a memory-mapped SFR) with direct addressing through a 128-byte window. This location is in the range of addresses (1FE0–1FFFH) that cannot be accessed through windows. Although you could set up the window by writing 1FH to the WSR, reading this location through the window would return FFH (all ones) and writing to it would not change the contents. However, you could directly address the remaining SFRs in the range of 1F80–1FDFH.

4.3.2.5 Using the Linker Locator to Set Up a Window

In this example, the linker locator is used to set up a window. The linker locator locates the window in the upper register file and determines the value to load in the WSR for access to that window. (Please consult the manual provided with the linker locator for details.)

```
*********  mod1  **************
mod1 module main              ;Main module for linker
public function1
extrn  ?WSR                   ;Must declare ?WSR as external

wsr  equ   14h:byte
sp   equ   18h:word

oseg
     var1:   dsw  1           ;Allocate variables in an overlayable segment
     var2:   dsw  1
     var3:   dsw  1

cseg

function1:
     push  wsr                ;Prolog code for wsr
     ldb   wsr, #?WSR         ;Prolog code for wsr

     add var1, var2, var3     ;Use the variables as registers
     ;
     ;
     ;

     ldb  wsr, [sp]           ;Epilog code for wsr
     add sp, #2               ;Epilog code for wsr
     ret
end

********  mod2  **************
```

```
public function2
extrn  ?WSR

wsr  equ   14h:byte
sp   equ   18h:word

oseg
     var1:   dsw  1
     var2:   dsw  1
     var3:   dsw  1

cseg

function2:
     push  wsr                ;Prolog code for wsr
     ldb   wsr, #?WSR         ;Prolog code for wsr

     add var1, var2, var3
     ;
     ;
     ;

     ldb  wsr, [sp]           ;Epilog code for wsr
     add sp, #2               ;Epilog code for wsr
     ret
end
*****************************
```

The following is an example of a linker invocation to link and locate the modules and to determine the proper windowing.

```
RL196 MOD1.OBJ, MOD2.OBJ registers(100h-03ffh) windowsize(32)
```

The above linker controls tell the linker to use registers 0100–03FFH for windowing and to use a window size of 32 bytes. (These two controls enable windowing.)

The following is the map listing for the resultant output module (MOD1 by default):

```
SEGMENT MAP FOR mod1(MOD1):

             TYPE          BASE    LENGTH   ALIGNMENT    MODULE NAME
             ----          ----    ------   ---------    -----------
**RESERVED*               0000H     001AH
             STACK        001AH     0006H   WORD
*** GAP ***               0020H     00E0H
             OVRLY        0100H     0006H   WORD         MOD2
             OVRLY        0106H     0006H   WORD         MOD1
*** GAP ***               010CH     1F74H
             CODE         2080H     0011H   BYTE         MOD2
             CODE         2091H     0011H   BYTE         MOD1
*** GAP ***               20A2H     DF5EH
```

This listing shows the disassembled code:

```
2080H         ;C814               | PUSH  WSR
2082H         ;B14814             | LDB   WSR,#48H
2085H         ;44E4E2E0           | ADD   E0H,E2H,E4H
2089H         ;B21814             | LDB   WSR,[SP]
208CH         ;65020018           | ADD   SP,#02H
2090H         ;F0                 | RET
2091H         ;C814               | PUSH  WSR
2093H         ;B14814             | LDB   WSR,#48H
2096H         ;44EAE8E6           | ADD   E6H,E8H,EAH
209AH         ;B21814             | LDB   WSR,[SP]
209DH         ;65020018           | ADD   SP,#02H
20A1H         ;F0                 | RET
```

The C compiler can also take advantage of this feature if the “windows” switch is enabled. For details, see the MCS 96 microcontroller architecture software products in the *Development Tools Handbook.*

4.3.3 Windowing and Addressing Modes

Once windowing is enabled, the windowed locations can be accessed both through the window using direct addressing and through the actual addresses using indirect or indexed addressing. The lower register file locations that are covered by the window are always accessible by indirect or indexed operations. To re-enable direct access to the entire lower register file, clear bits 6:0 of the WSR and all bits of WSR1. To enable direct access to a particular location in the lower register file, you may select a smaller window that does not cover that location.

When windowing is enabled:

- a direct instruction that uses an address within the lower register file actually accesses the window in the upper register file;
- an indirect or indexed instruction that uses an address within either the lower register file or the upper register file accesses the actual location in memory.

The following sample code illustrates the difference between direct and indexed addressing when using windowing.

```
PUSHA                   ; Pushes the contents of WSR onto the stack
LDB  WSR, #17H          ; Selects window 17H, a 128-byte block
                        ; (windows 0380-03FFH into 0080-00FFH)
                        ; The next instruction uses direct addr
ADD  40H, 80H           ; mem_word(40H)←mem_word(40H) + mem_word(380H)
                        ; The next two instructions use indirect addr
ADD  40H, 80H[0]        ; mem_word(40H)←mem_word(40H) + mem_word(80H +0)
ADD  40H, 380H[0]       ; mem_word(40H)←mem_word(40H) + mem_word(380H +0)
POPA                    ; reloads the previous contents into WSR
```

4.4 CONTROLLING READ ACCESS TO THE INTERNAL ROM

The LOC bit in CCR0 (Figure 4-8) controls read access to the internal ROM. Clear the LOC bit in CCB0 to enable read protection. To allow **authorized** access for program verification, program a security key into special-purpose memory (locations FF2020–FF202FH). Once a security key is programmed, you must supply a matching key to gain access to the internal ROM. (See Chapter 17, "Using the Test-ROM Routines," for additional information.)

CCR0 no direct access†

The chip configuration 0 (CCR0) register controls ROM access, enables or disables the IDLPD #1 and IDLPD #2 instructions and selects the write-control mode. It also contains the bus-control parameters for fetching chip configuration byte 1.

7							0
LOC	1	WS1	WS0	DEMUX	BHE#	BW16	PD

Bit Number	Bit Mnemonic	Function
7	LOC	Lock Bit This bit controls read access to the ROM during normal operation. 0 = read protect 1 = no protection Refer to “Controlling Read Access to the Internal ROM” on page 4-26 for details.
6	1	To guarantee proper operation, write one to this bit.
5:4	WS1:0	Wait States These bits, along with the READY pin, control the number of wait states that are used for an external fetch of chip configuration byte 1 (CCB1). **WS1 WS0** 0 0 zero wait states 0 1 one wait state 1 0 two wait states 1 1 three wait states If READY is low when this number is reached, additional wait states are added until READY is pulled high.
3	DEMUX	Select Demultiplexed Bus Selects the demultiplexed bus mode for an external fetch of CCB1: 0 = multiplexed — address and data are multiplexed on AD15:0. 1 = demultiplexed — data only on AD15:0.
2	BHE#	Write-control Mode Selects the write-control mode, which determines the functions of the BHE#/WRH# and WR#/WRL# pins for external bus cycles: 0 = write strobe mode: the BHE#/WRH# pin operates as WRH#, and the WR#/WRL# pin operates as WRL#. 1 = standard write-control mode: the BHE#/WRH# pin operates as BHE#, and the WR#/WRL# pin operates as WR#.

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after reset. The CCBs reside at addresses FF2018H (CCB0) and FF201AH (CCB1).

Figure 4-8. Chip Configuration 0 (CCR0) Register

CCR0 (Continued) no direct access†

The chip configuration 0 (CCR0) register controls ROM access, enables or disables the IDLPD #1 and IDLPD #2 instructions and selects the write-control mode. It also contains the bus-control parameters for fetching chip configuration byte 1.

7							0
LOC	1	WS1	WS0	DEMUX	BHE#	BW16	PD

Bit Number	Bit Mnemonic	Function
1	BW16	Buswidth Control Selects the bus width for an external fetch of CCB1: 0 = 8-bit bus 1 = 16-bit bus
0	PD	Powerdown Enable Enables or disables the IDLPD #1 and IDLPD #2 instructions. When enabled, the IDLPD #1 instruction causes the microcontroller to enter idle mode and the IDLPD #2 instruction causes the microcontroller to enter powerdown mode. 0 = disable idle and powerdown modes 1 = enable idle and powerdown modes If your design uses idle or powerdown mode, set this bit when you program the CCBs. If it does not, clearing this bit when you program the CCBs will prevent accidental entry into idle or powerdown mode. (Chapter 14, "Special Operating Modes," discusses idle and powerdown modes.)

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after reset. The CCBs reside at addresses FF2018H (CCB0) and FF201AH (CCB1).

Figure 4-8. Chip Configuration 0 (CCR0) Register (Continued)

4.5 REMAPPING INTERNAL ROM

The 83C196EA has 8 Kbytes of ROM (FF2000–FF3FFFH). The upper 7-Kbyte section (FF2400–FF3FFFH) is used for storing application code and data; the lower 1-Kbyte section (FF2000–FF23FFH) stores boot code, chip configuration bytes (CCBs), and interrupt vectors.

To optimize your application, you may want to make the application data in FF2400–FF3FFFH accessible as *near data* in page 00H, while still allowing the application code to execute in page FFH. To accomplish this, set the REMAP bit (CCB1.2) when you program the CCBs, and hold EA# high during reset. This combination forces all accesses to FF2000–FF3FFFH to the internal ROM and "shadows" the contents of FF2400–FF3FFFH into page 00H. Mapping the ROM contents into both pages 00H and FFH allows you to access data and constants as *near data* and *near constants* in page 00H and execute code in page FFH.

The EA# input controls whether accesses to FF2000–FF3FFFH are directed to internal ROM or an external memory device. When accesses are directed to internal ROM, the REMAP bit (CCB1.2) controls whether the upper 7-Kbyte section (FF2400–FF3FFFH) of internal ROM is mapped into page FFH only or into both pages FFH and 00H. The REMAP bit is loaded from the CCB and the state of EA# is latched upon leaving reset; neither can be changed until the next reset. You can read IRAM_CON (Figure 4-3 on page 4-10) to determine the state of the EA# pin (bit 7 contains the **complement** of EA#) and read CCR1 (Figure 4-9) to determine the state of the REMAP bit.

NOTE

The EA# input is effective only for accesses to the internal ROM (FF2000–FF3FFFH). For accesses to any other location, the state of EA# is irrelevant. The REMAP bit is effective only for accesses to FF2400–FF3FFFH and only when EA# is inactive. When EA# is active, execution is external and the REMAP bit is ignored.

With EA# active, the REMAP bit (CCB1.2) is ignored. Accesses to FF2400–FF3FFFH are directed to external memory (1F2400–1F3FFFH). Data in this area must be accessed with extended instructions.

With EA# inactive and remapping disabled (CCB1.2 = 0), accesses to FF2400–FF3FFFH are directed to internal ROM (FF2400–FF3FFFH). Data in this area must be accessed with extended instructions.

With EA# inactive and remapping enabled (CCB1.2 = 1), you can access the contents of FF2400–FF3FFFH in two ways:

- in internal ROM (FF2400–FF3FFFH) using extended instructions, and
- in internal ROM (002400–003FFFH) using nonextended instructions. This makes the far data in FF2000–FF3FFFH accessible as near data.

An advantage of remapping ROM is that it makes the data in ROM accessible as near data in internal memory page 00H. The data can then be accessed more quickly with nonextended instructions. An advantage of not remapping ROM is that the corresponding area in memory page 00H is available for storing additional near data.

CCR1 no direct access†

The chip configuration 1 (CCR1) register controls the clock failure detection circuitry, deferred mode, and the watchdog timer, selects the 64-Kbyte or 2-Mbyte addressing mode, and controls whether the upper 7-Kbyte section of internal ROM (FF2400–FF3FFFH) is mapped into page FFH only or into both pages FFH and 00H.

7							0
1	CFD	DM	0	WDD	REMAP	MODE64	0

Bit Number	Bit Mnemonic	Function
7	1	To guarantee proper operation, write one to this bit.
6	CFD	Clock-failure Detection This bit enables or disables the clock failure detection circuitry. 1 = enables clock-failure detection circuitry 0 = disables clock-failure detection circuitry (See "Clock Failure Detection Logic" on page 2-12.)
5	DM	Deferred Mode Enables the deferred bus-cycle mode. If the microcontroller is using a demultiplexed bus and deferred mode is enabled, a delay of 2t occurs in the first bus cycle following a chip-select output change, the first write cycle following a read cycle, and the first read cycle following a write cycle. (See "Deferred Bus-cycle Mode" on page 15-41.) 0 = deferred bus-cycle mode disabled 1 = deferred bus-cycle mode enabled
4	0	To guarantee proper operation, write zero to this bit.
3	WDD	Watchdog Timer Disable Selects whether the watchdog timer is always enabled or disabled until the first time it is cleared. If this bit is clear, the watchdog is enabled at reset, so software must clear the watchdog within 64K state times to prevent another reset. If this bit is set, the watchdog is disabled until the first time you clear it. (See "Enabling the Watchdog Timer" on page 13-12.) 0 = always enabled 1 = disabled at reset; enabled the first time it is cleared

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after reset. Locate your CCBs at addresses FF2018H (CCB0) and FF201AH (CCB1).

Figure 4-9. Chip Configuration 1 (CCR1) Register

CCR1 (Continued) no direct access†

The chip configuration 1 (CCR1) register controls the clock failure detection circuitry, deferred mode, and the watchdog timer, selects the 64-Kbyte or 2-Mbyte addressing mode, and controls whether the upper 7-Kbyte section of internal ROM (FF2400–FF3FFFH) is mapped into page FFH only or into both pages FFH and 00H.

7							0
1	CFD	DM	0	WDD	REMAP	MODE64	0

Bit Number	Bit Mnemonic	Function
2	REMAP	Internal ROM Mapping The EA# pin controls whether accesses to FF2000–FF3FFFH are directed to internal ROM or to external memory. When EA# is low (external execution), REMAP is ignored. When EA# is high (internal execution), REMAP controls whether the upper 7-Kbyte area (FF2400–FF3FFFH) of internal ROM is mapped only into page FFH or into both pages FFH and 00H. 0 = ROM maps to page FFH only 1 = ROM maps to page FFH and page 00H (See “Remapping Internal ROM” on page 4-29.)
1	MODE64	Addressing Mode Selects 64-Kbyte or 2-Mbyte addressing. 0 = selects 2-Mbyte addressing 1 = selects 64-Kbyte addressing In 2-Mbyte mode, code can execute from almost anywhere in the address space. In 64-Kbyte mode, code can execute only from page FFH. (See “Fetching Code and Data in the 2-Mbyte and 64-Kbyte Modes” on page 4-31 for more information.)
0	0	Reserved; for compatibility with future devices, write zero to this bit.

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after reset. Locate your CCBs at addresses FF2018H (CCB0) and FF201AH (CCB1).

Figure 4-9. Chip Configuration 1 (CCR1) Register (Continued)

4.6 FETCHING CODE AND DATA IN THE 2-MBYTE AND 64-KBYTE MODES

When the microcontroller leaves reset, the MODE64 bit (CCB1.1) selects the 2-Mbyte or 64-Kbyte mode. The mode cannot be changed until the next reset. (Typically, the CCBs are programmed once when our program is compiled and are not changed during normal operation.)

In 64-Kbyte mode, code must execute from page FFH and data must reside in page 00H. In 2-Mbyte mode, code can execute from any page. Data must reside in page 00H for nonextended instructions, but it can reside in any page for extended instructions. In either mode, data and constants that reside in page 00H are called *near data* and *near constants*. Data outside page 00H are called *far data*.

5

Stack Overflow Module

CHAPTER 5
STACK OVERFLOW MODULE

The stack overflow module monitors the value of the stack pointer (SP) and generates a nonmaskable interrupt request if the value is outside the boundaries you specify. This module helps to ensure data integrity. Without the stack overflow module, the stack could grow too large and corrupt other data, causing a fatal system error.

5.1 FUNCTIONAL OVERVIEW

The stack overflow module (Figure 5-1) decodes the stack pointer addresses (0018–0019H) from the peripheral address bus. The comparator compares that value to the boundaries you specify. A stack pointer value that is equal to or outside the boundaries sets the R/S flip-flop, which in turn sets the nonmaskable interrupt pending bit.

The R/S flip-flop can be set only once, and only a reset or another write operation to the STACK_TOP register can clear it. Your initialization code and your interrupt service routine for the stack overflow error must write to the STACK_TOP register to enable the stack overflow module.

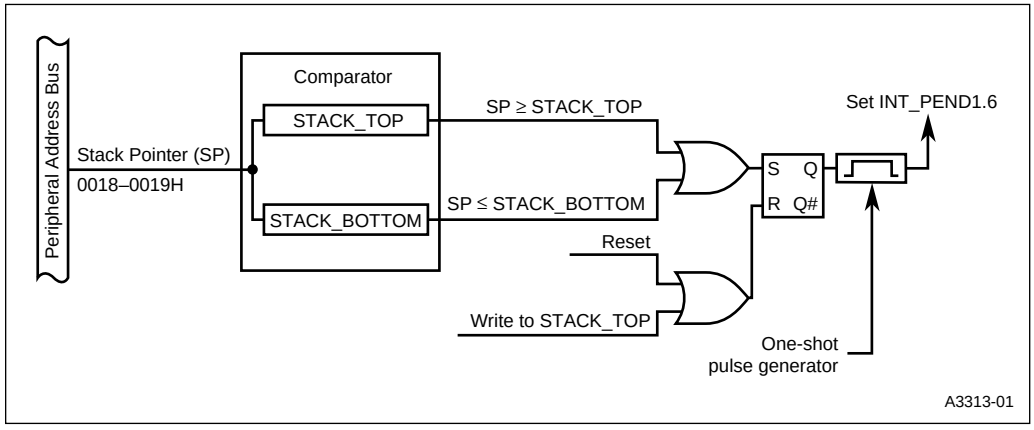


Figure 5-1. Stack Overflow Module Block Diagram

5.2 STACK OPERATIONS

The stack is a last-in-first-out (LIFO) register used to keep track of program addresses and the processor status word (PSW), as well as to pass parameters from one routine to another. You control the location of the stack by initializing the stack pointer (SP) with an address that is two bytes (for 64-Kbyte mode) or four bytes (for 2-Mbyte mode) greater than the starting address (top) of the stack. The stack grows downward as entries are added (PUSHed onto the stack) and shrinks upward as entries are unloaded (POPped off the stack). The stack must be located in page 00H, and is most efficient when located in the upper register file.

When the CPU encounters a subroutine call or interrupt service routine, it manipulates the stack pointer (SP), the stack, and the program counter (PC) as shown in Table 5-1.

Table 5-1. Effect of Subroutine Execution on the Stack, SP, and PC

Instruction	Operation
BR, CALL, JUMP	SP ← (SP–2) in 64-Kbyte mode or (SP–4) in 2-Mbyte mode) Stack ← PC (next instruction address) PC ← address of subroutine or interrupt service routine
…	Execute subroutine or interrupt service routine
RET	PC ← next instruction address (from top of stack) SP ← (SP+2) in 64-Kbyte mode or (SP+4) in 2-Mbyte mode)

5.3 STACK OVERFLOW MODULE REGISTERS

Table 5-2 describes the registers associated with the stack overflow module.

Table 5-2. Stack Overflow Module Control and Status Registers

Mnemonic	Address	Description
INT_MASK1	0013H	Interrupt Mask 1 The stack overflow interrupt is always enabled. The nonfunctional STACK mask bit in INT_MASK1 exists for design symmetry with the INT_PEND1 register. Write zero to the STACK bit or leave it in its default state.
INT_PEND1	0012H	Interrupt Pending 1 The STACK bit in this register, when set, indicates that a stack overflow interrupt is pending.
SP	0018H	Stack Pointer Initialize the stack pointer to an address that is two bytes (in 64-Kbyte mode) or four bytes (in 2-Mbyte mode) greater than the desired starting address (top) of the stack. Ensure that this value is within the boundaries you specify in STACK_BOTTOM and STACK_TOP.
STACK_BOTTOM	1FA0H	Lower Stack Limit Write the boundary address for the bottom of the stack to this register. A stack pointer value that is equal to or less than this address will set the stack overflow interrupt pending bit in INT_PEND1.
STACK_TOP	1FA2H	Upper Stack Limit You must write to this register to enable the stack overflow module after a reset and after a stack overflow. Write the boundary address for the top of the stack to this register. A stack pointer value that is equal to or greater than this address will set the stack overflow interrupt pending bit in INT_PEND1.

5.4 PROGRAMMING THE STACK OVERFLOW MODULE

To use the stack overflow module, you must initialize the stack pointer and specify the stack boundaries. The stack overflow interrupt is nonmaskable, so you do not need to enable the interrupt in the INT_MASK1 register or enable interrupt servicing by executing the EI instruction.

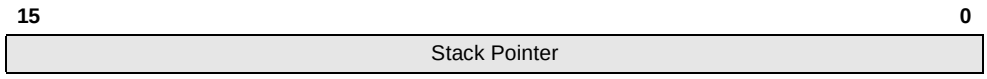
5.4.1 Initializing the Stack Pointer

Initialize the stack pointer (Figure 5-2) to an address that is two bytes (in 64-Kbyte mode) or four bytes (in 2-Mbyte mode) greater than the desired starting address (top) of the stack. Ensure that this value is within the boundaries you specify in STACK_BOTTOM and STACK_TOP.

SP

Address: 18H
Reset State: XXXXH

The system's stack pointer (SP) can point anywhere in page 00H; it must be word aligned and must always be initialized before use. The stack pointer is decremented before a PUSH and incremented after a POP, so you should initialize the stack pointer to two bytes (in 64-Kbyte mode) or four bytes (in 2-Mbyte mode) above the highest stack location. If stack operations are not being performed, locations 18H and 19H may be used as standard registers.



Bit Number	Function
15:0	Stack Pointer This register makes up the system's stack pointer.

Figure 5-2. Stack Pointer (SP)

5.4.2 Enabling the Stack Overflow Module and Specifying Stack Boundaries

Two registers, STACK_BOTTOM and STACK_TOP, control the stack overflow detection circuitry. Write to STACK_BOTTOM (Figure 5-3) to establish the lower boundary for the stack, then write to STACK_TOP (Figure 5-4) to establish the upper boundary and to enable the overflow detection circuitry.

If an overflow occurs, the stack overflow module sets the stack overflow interrupt pending bit in INT_PEND1. Your interrupt service routine must write to the STACK_TOP register to re-enable the overflow detection circuitry. Otherwise, the R/S flip-flop (Figure 5-1 on page 5-1) remains set, and the stack overflow module is unable to signal another overflow.

STACK_BOTTOM Address: 1FA0H
Reset State: 0000H

The value you write to this register specifies the lower address limit for the stack. The overflow detection circuitry compares the stack pointer (SP) value on the peripheral address bus to the STACK_TOP and STACK_BOTTOM values.

15	14	13	12	11	10	9	8
Lower Stack Limit (high byte)							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Lower Stack Limit (low byte)							

Bit Number	Function
15:0	Lower Stack Limit This register specifies the lower address limit for the stack. If the stack pointer value is equal to or less than this value, the stack overflow module sets the nonmaskable stack error interrupt (INT14) pending bit in INT_PEND1.

Figure 5-3. Lower Stack Limit (STACK_BOTTOM) Register

STACK_TOP Address: 1FA2H
Reset State: FFFFH

The value you write to this register specifies the upper address limit for the stack. You must write to this register to enable the stack overflow module. The overflow detection circuitry compares the stack pointer (SP) value on the peripheral address bus to the STACK_TOP and STACK_BOTTOM values.

15	14	13	12	11	10	9	8
Upper Stack Limit (high byte)							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Upper Stack Limit (low byte)							

Bit Number	Function
15:0	Upper Stack Limit This register specifies the upper address limit for the stack. If the stack pointer value is equal to or greater than this value, the stack overflow module sets the nonmaskable stack error interrupt (INT14) pending bit in INT_PEND1.

Figure 5-4. Upper Stack Limit (STACK_TOP) Register

6

Standard and PTS Interrupts

CHAPTER 6
STANDARD AND PTS INTERRUPTS

This microcontroller's flexible interrupt-handling system has three main components: the programmable interrupt controller, the peripheral transaction server (PTS), and the peripheral interrupt handlers (PIHs). This chapter describes these components and explains how to program them.

6.1 OVERVIEW OF THE INTERRUPT CONTROL CIRCUITRY

The interrupt control circuitry within a microcontroller permits real-time events to control program flow. When an event generates an interrupt, the microcontroller suspends the execution of current instructions while it performs some service in response to the interrupt. When the interrupt is serviced, program execution resumes at the point where the interrupt occurred. An internal peripheral, an external signal, or an instruction can generate an interrupt request. In the simplest case, the microcontroller receives the request, performs the service, and returns to the task that was interrupted.

The interrupt sources fall into two categories. The unimplemented opcode, software trap, and NMI interrupt sources are always enabled. All other sources can be individually enabled.

Interrupts that go through the interrupt controller are serviced by interrupt service routines that you provide (Figure 6-1). The lower 16 bits of the addresses of these interrupt service routines are stored in the upper and lower interrupt vectors in special-purpose memory (Table 4-2 on page 4-8). The CPU automatically adds FF0000H to the 16-bit vector in special-purpose memory to calculate the address of the interrupt service routine, and then executes the routine.

The peripheral transaction server (PTS), a microcoded hardware interrupt processor, provides high-speed, low-overhead interrupt handling; it does not modify the stack or the PSW. You can configure most interrupts (except NMI, software trap (TRAP), unimplemented opcode, stack overflow, PIH0_INT, and PIH1_INT) to be serviced by the PTS instead of the interrupt controller.

The PTS provides four special microcoded routines that enable it to complete specific tasks faster than an equivalent interrupt service routine. It can transfer bytes or words, either individually or in blocks, between any memory locations in page 00H; abort PTS service if a dummy PTS request occurs; and test for a missing event in a series of regular events. PTS interrupts have a higher priority than standard interrupts and may temporarily suspend interrupt service routines.

A block of data called the PTS control block (PTSCB) contains the specific details for each PTS routine (see "Initializing the PTS Control Blocks" on page 6-30). When a PTS interrupt occurs, the priority resolver selects the appropriate vector from special-purpose memory and fetches the PTS control block (PTSCB).

To provide support for the large number of event processor array (EPA) channels, the 8XC196EA incorporates two peripheral interrupt handlers (PIHs). Each PIH services 16 different interrupt sources. You can select either interrupt controller or PTS service for each PIH interrupt source. When a PIH receives an interrupt request from an enabled source, it generates either a standard

interrupt request or PTS service request to the CPU. Although the PIH interrupt vectors are stored in special-purpose memory, the PIH must supply the interrupt vector address to the CPU.

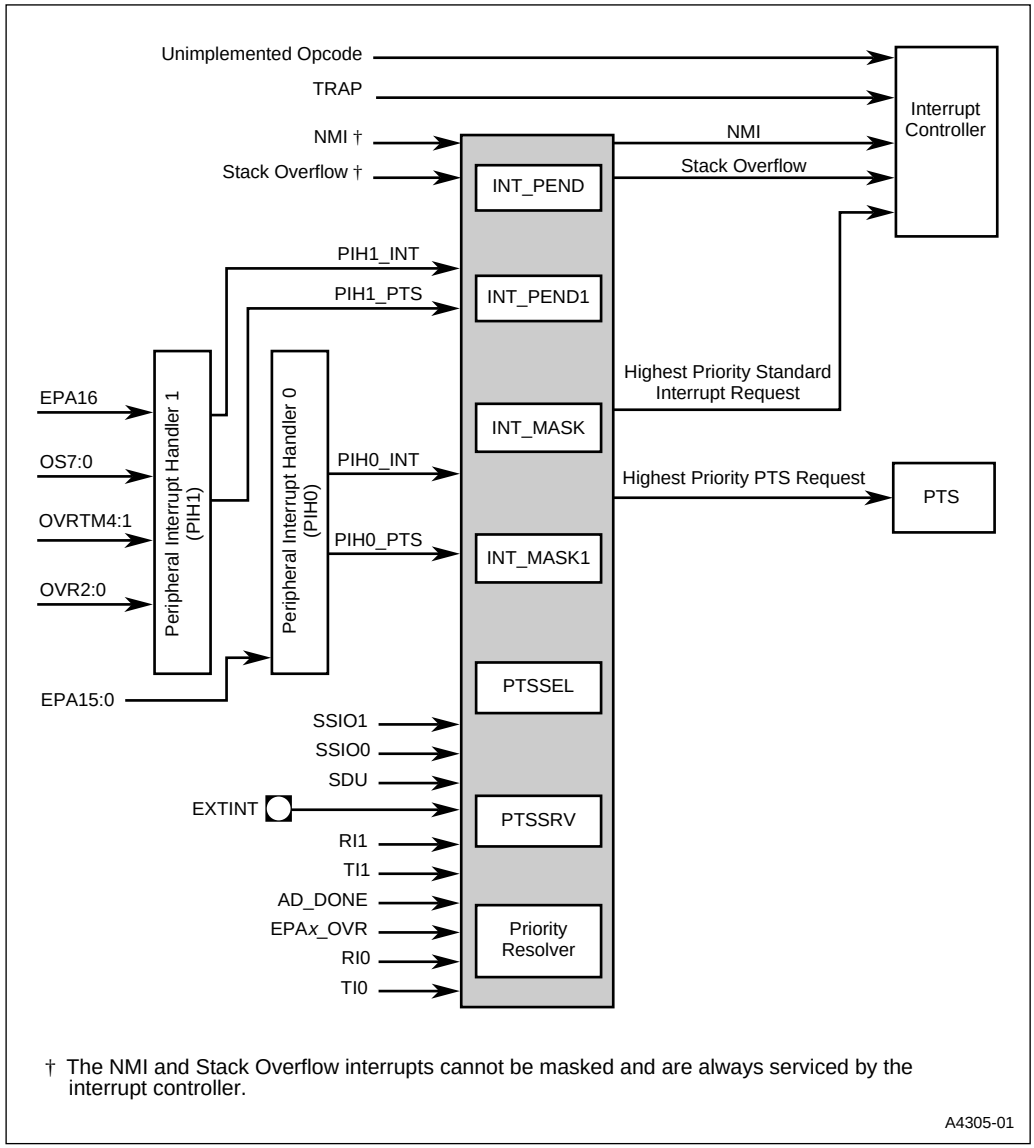


Figure 6-1. Interrupt Structure Block Diagram

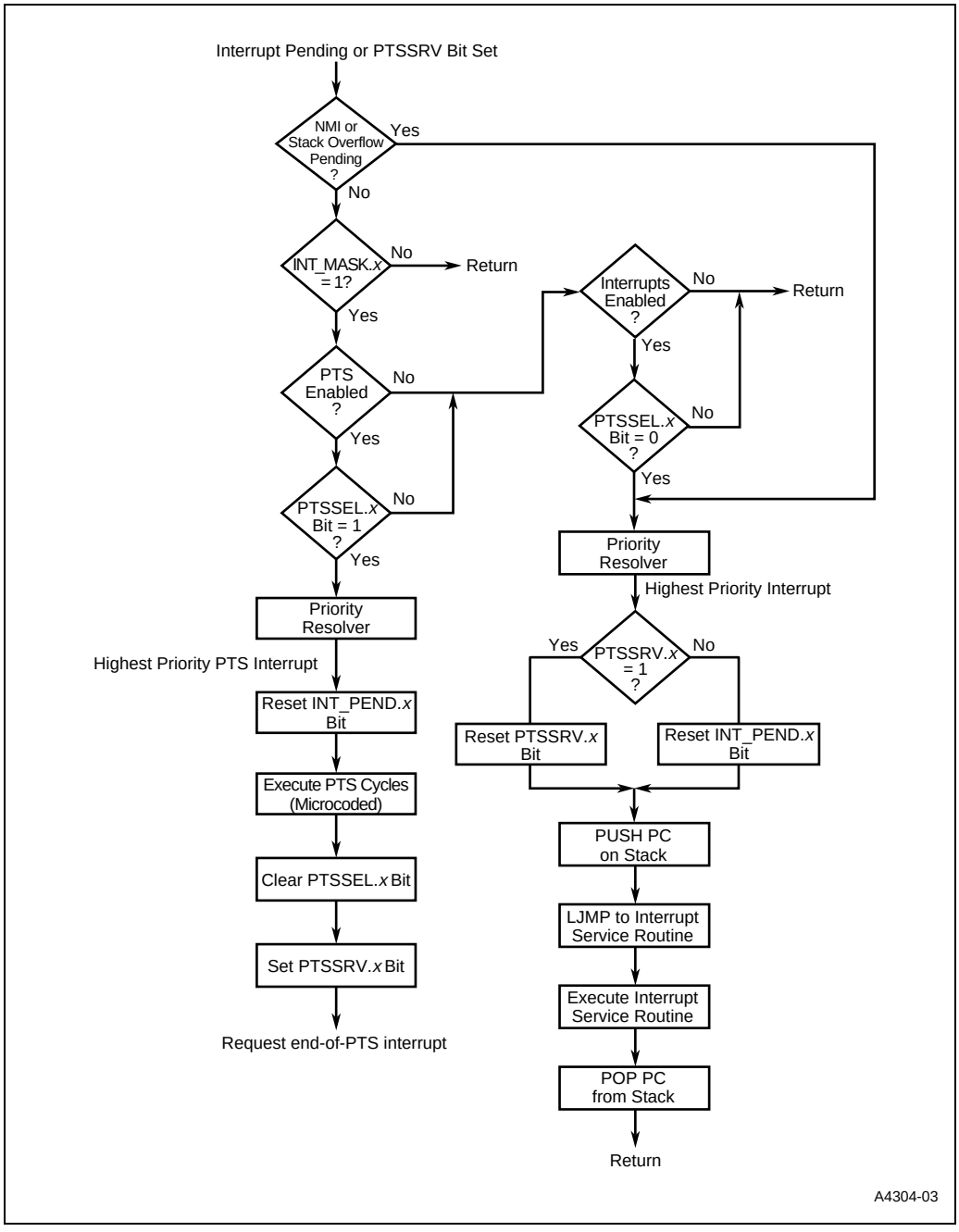


Figure 6-2. Interrupt Service Flow Diagram

6.2 INTERRUPT SIGNALS AND REGISTERS

Table 6-1 describes the external interrupt signals and Table 6-2 describes the interrupt control and status registers.

Table 6-1. Interrupt Signals

Interrupt Signal	Type	Description
EXTINT	I	External Interrupt In normal operating mode, a rising edge on EXTINT sets the EXTINT interrupt pending bit. EXTINT is sampled during phase 2 (CLKOUT high). The minimum high time is one state time. In powerdown mode, asserting the EXTINT signal for at least 50 ns causes the device to resume normal operation. The interrupt does not need to be enabled, but the pin must be configured as a special-function input. If the EXTINT interrupt is enabled, the CPU executes the interrupt service routine. Otherwise, the CPU executes the instruction that immediately follows the command that invoked the power-saving mode. In idle mode, asserting any enabled interrupt causes the device to resume normal operation. EXTINT shares a package pin with P2.2.
NMI	I	Nonmaskable Interrupt In normal operating mode, a rising edge on NMI generates a nonmaskable interrupt. NMI has the highest priority of all prioritized interrupts. Assert NMI for greater than one state time to guarantee that it is recognized.

Table 6-2. Interrupt and PTS Control and Status Registers

Mnemonic	Address	Description
INT_MASK INT_MASK1	0008H 0013H	Interrupt Mask Registers The bits in these registers enable or disable each maskable interrupt (that is, each interrupt except unimplemented opcode, software trap, NMI, and stack overflow).
INT_PEND INT_PEND1	0009H 0012H	Interrupt Pending Registers The bits in these registers are set by hardware to indicate that an interrupt is pending.
PIH0_INT_MASK PIH1_INT_MASK	1E98H 1EA8H	Peripheral Interrupt Handler (PIH) Interrupt Mask Registers The bits in these registers enable or disable each interrupt that is routed through the PIH.
PIH0_INT_PEND PIH1_INT_PEND	1E9AH 1EAAH	Peripheral Interrupt Handler (PIH) Interrupt Pending Registers The bits in these registers are set by hardware to indicate that a PIH interrupt source is pending.
PIH0_PTSSEL PIH1_PTSSEL	1E96H 1EA6H	Peripheral Interrupt Handler (PIH) PTS Select Registers These registers select either a PTS interrupt service request or a standard interrupt service request for each interrupt that is routed through the PIH.

Table 6-2. Interrupt and PTS Control and Status Registers (Continued)

Mnemonic	Address	Description
PIH0_PTSSRV PIH1_PTSSRV	1E94H 1EA4H	Peripheral Interrupt Handler (PIH) PTS Service Registers The bits in these registers are set by hardware to request an end-of-PTS interrupt.
PIH0_VEC_BASE PIH1_VEC_BASE	1E92H 1EA2H	Peripheral Interrupt Handler (PIH) Vector Base Address Registers These registers contain bits 6–15 of the PIH interrupt vector address. Always initialize PIH0_VEC_BASE to 20C0H and PIH1_VEC_BASE to 2100H.
PIH0_VEC_IDX PIH1_VEC_IDX	1E90H 1EA0H	Peripheral Interrupt Handler Vector Index Address Registers These registers contain the number of the highest priority, active, standard PIH interrupt request.
PSW	No direct access	Processor Status Word This register contains one bit that globally enables or disables servicing of all maskable interrupts and another that enables or disables the PTS. These bits are set or cleared by executing the enable interrupts (EI), disable interrupts (DI), enable PTS (EPTS), and disable PTS (DPTS) instructions.
PTSSEL	0004H, 0005H	PTS Select Register This register selects either a PTS interrupt service request or a standard interrupt service request for each of the maskable interrupt requests. Never use this register to disable PTS service for individual PIH interrupts. Instead, either use the PIH*x*_PTSSEL register to disable PTS service for individual PIH interrupt sources or use the INT_MASK registers to mask the PIH*x*_PTS or PIH*x*_INT interrupts.
PTSSRV	0006H, 0007H	PTS Service Register The bits in this register are set by hardware to request an end-of-PTS interrupt.

6.3 INTERRUPT SOURCES, PRIORITIES, AND VECTOR ADDRESSES

Table 6-3 lists the interrupt sources, their default priorities (30 is highest and 0 is lowest), and their vector addresses. Higher priority interrupts are serviced before lower priority interrupts. A low-priority interrupt is always interrupted by a higher priority interrupt, but not by another interrupt of equal or lower priority. The absolute highest priority interrupt is not interrupted by any other interrupt source.

The unimplemented opcode and software trap interrupts are not prioritized; they go directly to the interrupt controller for servicing. The priority resolver determines the priority of all other pending interrupt requests. NMI and the stack overflow error have higher priority than any other prioritized interrupts. PTS interrupts always have higher priority than standard interrupts. The priority resolver selects the highest priority pending request and the interrupt controller selects the corresponding vector location in special-purpose memory. This vector contains the starting (base) address of the corresponding PTS control block (PTSCB) or interrupt service routine. PTSCBs

must be located on a quad-word boundary in the internal register file. Interrupt service routines must begin execution in page FFH, but can jump anywhere after the initial vector is taken.

Table 6-3. Interrupt Sources, Vectors, and Priorities

Interrupt Source	Mnemonic	Interrupt Controller Service			PTS Service		
		Name	Vector	Priority†	Name	Vector	Priority
Nonmaskable Interrupt	NMI	INT15	FF203EH	30	—	—	—
Stack Overflow Error	Stack	INT14	FF203CH	14	—	—	—
PIH0 PTS Interrupt	PIH0_PTS	INT13	Table 6-4	13	PTS13	Table 6-4	28
PIH0 Standard Interrupt	PIH0_INT	INT12	Table 6-4	12	PTS12	Table 6-4	27
PIH1 PTS Interrupt	PIH1_PTS	INT11	Table 6-5	11	PTS11	Table 6-5	26
PIH1 Standard Interrupt	PIH1_INT	INT10	Table 6-5	10	PTS10	Table 6-5	25
SSIO Channel 1 Transfer	SSIO1	INT09	FF2032H	9	PTS09	FF2052H	24
SSIO Channel 0 Transfer	SSIO0	INT08	FF2030H	8	PTS08	FF2050H	23
Dummy PTS Cycle	—	—	—	—	—	FF2016H	—
Dummy Standard Interrupt	—	—	FF2014H	—	—	—	—
Unimplemented Opcode	—	—	FF2012H	—	—	—	—
Software TRAP Instruction	—	—	FF2010H	—	—	—	—
Serial Debug Unit Interrupt	SDU	INT07	FF200EH	7	PTS07	FF204EH	22
EXTINT Pin	EXTINT	INT06	FF200CH	6	PTS06	FF204CH	21
SIO1 Receive	RI1	INT05	FF200AH	5	PTS05	FF204AH	20
SIO1 Transmit	TI1	INT04	FF2008H	4	PTS04	FF2048H	19
A/D Conversion Complete	AD_DONE	INT03	FF2006H	3	PTS03	FF2046H	18
EPA Channel 3–16 Overrun	EPA*x*_OVR	INT02	FF2004H	2	PTS02	FF2044H	17
SIO0 Receive	RI0	INT01	FF2002H	1	PTS01	FF2042H	16
SIO0 Transmit	TI0	INT00	FF2000H	0	PTS00	FF2040H	15

† The higher the priority number, the higher the priority.

6.3.1 PIH Interrupt Sources, Priorities, and Vector Addresses

Tables 6-4 and 6-5 list the interrupt sources for the peripheral interrupt handler (PIH) interrupts with relative priority within the PIH and vector addresses. Each PIH provides the capability to handle 16 additional interrupt sources.

A major difference between PIH interrupts and interrupts that go directly to the CPU is that the PIH must provide the address of the individual PIH interrupt vector to the CPU. It can provide the address in one of two ways. Either your code can read the vector index register (PIH*x*_VEC_IDX, Figure 6-3 on page 6-8) directly, or the CPU can request the address as part of an interrupt acknowledge cycle.

Table 6-4. PIH0 Interrupt Sources, Vectors, and Priorities

Interrupt Source	Mnemonic	Interrupt Controller Service (PIH0_INT) Vector	PTS Service (PIH0_PTS) Vector	PIH0 Priority†
EPA Capture/Compare 15	EPA15	FF20FCH	FF20FEH	15
EPA Capture/Compare 14	EPA14	FF20F8H	FF20FAH	14
EPA Capture/Compare 13	EPA13	FF20F4H	FF20F6H	13
EPA Capture/Compare 12	EPA12	FF20F0H	FF20F2H	12
EPA Capture/Compare 11	EPA11	FF20ECH	FF20EEH	11
EPA Capture/Compare 10	EPA10	FF20E8H	FF20EAH	10
EPA Capture/Compare 9	EPA9	FF20E4H	FF20E6H	9
EPA Capture/Compare 8	EPA8	FF20E0H	FF20E2H	8
EPA Capture/Compare 7	EPA7	FF20DCH	FF20DEH	7
EPA Capture/Compare 6	EPA6	FF20D8H	FF20DAH	6
EPA Capture/Compare 5	EPA5	FF20D4H	FF20D6H	5
EPA Capture/Compare 4	EPA4	FF20D0H	FF20D2H	4
EPA Capture/Compare 3	EPA3	FF20CCH	FF20CEH	3
EPA Capture/Compare 2	EPA2	FF20C8H	FF20CAH	2
EPA Capture/Compare 1	EPA1	FF20C4H	FF20C6H	1
EPA Capture/Compare 0	EPA0	FF20C0H	FF20C2H	0

† The higher the number, the higher the priority.

Table 6-5. PIH1 Interrupt Sources, Vectors, and Priorities

Interrupt Source	Mnemonic	Interrupt Controller Service (PIH1_INT) Vector	PTS Service (PIH1_PTS) Vector	PIH1† Priority
EPA Capture/Compare 16	EPA16	FF213CH	FF213EH	15
Output Simulcapture 7	OS7	FF2138H	FF213AH	14
Output Simulcapture 6	OS6	FF2134H	FF2136H	13
Output Simulcapture 5	OS5	FF2130H	FF2132H	12
Output Simulcapture 4	OS4	FF212CH	FF212EH	11
Output Simulcapture 3	OS3	FF2128H	FF212AH	10
Output Simulcapture 2	OS2	FF2124H	FF2126H	9
Output Simulcapture 1	OS1	FF2120H	FF2122H	8
Output Simulcapture 0	OS0	FF211CH	FF211EH	7
Timer 1 Overflow/Underflow	OVRTM1	FF2118H	FF211AH	6
Timer 2 Overflow/Underflow	OVRTM2	FF2114H	FF2116H	5
Timer 3 Overflow/Underflow	OVRTM3	FF2110H	FF2112H	4
Timer 4 Overflow/Underflow	OVRTM4	FF210CH	FF210EH	3
EPA0 Capture Overrun	OVR0	FF2108H	FF210AH	2
EPA1 Capture Overrun	OVR1	FF2104H	FF2106H	1
EPA2 Capture Overrun	OVR2	FF2100H	FF2102H	0

† The higher the number, the higher the priority.

PIH*x*_VEC_IDX
***x* = 0–1**

Address: 1E90H, 1EA0H
Reset State: 10H

The peripheral interrupt handler vector index (PIH*x*_VEC_IDX) registers indicate which standard PIH input is requesting interrupt service.

15	14	13	12	11	10	9	8
—	—	—	—	—	—	—	—

7	6	5	4	3	2	1	0
—	—	—	NI	IS3	IS2	IS1	IS0

Bit Number	Bit Mnemonic	Function
15:5	—	Reserved. These bits are undefined.

Figure 6-3. Peripheral Interrupt Handler *x* Vector Index (PIH*x*_VEC_IDX) Registers

PIH*x*_VEC_IDX (Continued)
***x* = 0–1**

Address: 1E90H, 1EA0H
Reset State: 10H

The peripheral interrupt handler vector index (PIH*x*_VEC_IDX) registers indicate which standard PIH input is requesting interrupt service.

15							8
—	—	—	—	—	—	—	—
7							**0**
—	—	—	NI	IS3	IS2	IS1	IS0

Bit Number	Bit Mnemonic	Function
4	NI	No Interrupt Pending This bit is set when the last PIH*x* interrupt request is serviced.
3:0	IS3:0	Interrupt Source These bits indicate the hex value of the highest-priority pending, standard PIH*x* interrupt request. PTS interrupt requests are not recorded in this register.

Figure 6-3. Peripheral Interrupt Handler *x* Vector Index (PIH*x*_VEC_IDX) Registers

6.3.1.1 Using Software to Provide the Vector Address

This method works only when the PIH generates a standard interrupt request. If software reads the PIH*x*_VEC_IDX register directly, the highest-priority unmasked, standard interrupt is read and hardware clears the appropriate PIH*x*_INT_PEND and PIH*x*_PTSSRV bits. The value seen by software will be in one of two formats.

If no interrupt is active, the PIH*x*_VEC_IDX always contains 0010H:

0000 0000 0001 0000

If an interrupt is active, the PIH*x*_VEC_IDX register contains the following information:

0000 0000 0000 *xxxx*

where:

xxxx is the hex value of the highest priority interrupt pending

For example, if both the EPA3 (priority 3) and EPA10 (priority 10) interrupts are pending in PIH0, the PIH0_VEC_IDX register contains 000AH, the value of the highest-priority pending interrupt:

0000 0000 0000 1010

Hardware clears the interrupt pending bit for EPA10 (PIH0_INT_PEND.10) and software services the EPA10 interrupt. When software reads the PIH0_VEC_IDX register a second time, it now contains 0003H, the value of the current highest-priority pending interrupt:

0000 0000 0000 0011

Hardware clears the interrupt pending bit for EPA3 (PIH0_INT_PEND.3) and software services the EPA3 interrupt. If software reads the register after the EPA3 interrupt is serviced, it now contains 0010H:

0000 0000 0001 0000

which indicates that there are no pending PIH0 interrupt requests.

The TIJMP instruction can be used to supply the interrupt vector addresses to the CPU. (See Appendix A for additional information about TIJMP.)

The format for the TIJMP instruction is:

TIJMP *tbase,[index],#index_mask*

where:

tbase	is a word register containing the 16-bit starting address of the jump table, which must be located in page FFH.
[index]	is a word register containing a 16-bit address that points to a register that contains a 7-bit value used to calculate the offset into the jump table.
#index_mask	is 7-bit immediate data to mask the index. This value is ANDed with the 7-bit value pointed to by *[index]* and the instruction multiplies the result by two to determine the offset into the jump table.

TIJMP calculates the destination address as follows:

$$(\textit{[index]}\ \text{AND}\ \textit{\#index\_mask}) \times 2 + \textit{tbase}$$

To use the TIJMP instruction in this application, you would create a jump table with 17 destination addresses: one for each of the 16 standard PIH*x* interrupt sources and one for the return. The table must contain the lower 16 bits of each destination address. The TIJMP instruction will automatically add FF0000H to the destination address.

The following code is a simplified example of an interrupt service routine that uses the TIJMP instruction to service the PIH0_INT interrupt. This routine services all active PIH0_INT interrupt sources in order of their priority. The TIJMP instruction calculates an offset to fetch a word from a jump table (JTBASE in this example) which contains the start addresses of the interrupt service routines.

```
INIT_INTERRUPTS:
   LD JTBASE_PTR,#LSW JTBASE;store jump table base address

PIH0_INT_ISR:
   LD PIH0_VEC_IDX_PTR,#PIH0_VEC_IDX;read PIH0_VEC_IDX

   PUSHA               ;save INT_MASK/INT_MASK1/WSR/PSW
   TIJMP JTBASE_PTR,[PIH0_VEC_IDX_PTR],#1FH;initiate jump to correct ISR

EPA0_ISR:              ;EPA0 interrupts service routine
   .                   ;
   .                   ;
   TIJMP JTBASE_PTR,[PIH0_VEC_IDX_PTR],#1FH
                       ;check for pending
                       ;interrupts, exit
PIH0_DONE:
     POPA
     RET                                       ;exit, all PIHx
                                               ;interrupts serviced
JTBASE:
     DCW  LSW EPA0_ISR                         ;0 (EPA0)
     DCW  LSW EPA1_ISR                         ;1 (EPA1)
     DCW  LSW EPA2_ISR                         ;2 (EPA2)
     DCW  .
     DCW  .
     DCW  .
     DCW  LSW PIH0_DONE                        ;0010H (no pending interrupts)
```

Notice that the interrupt service routine ends with another TIJMP instruction instead of a RET instruction. The TIJMP instruction checks the PIH0_VEC_IDX register for any other pending PIH0 interrupts. When PIH0_VEC_IDX contains a 0010H (no pending interrupts) the routine vectors to PIH0_DONE and executes a RET instruction.

6.3.1.2 Providing the Vector Address in Response to a CPU Request

If the CPU requests the interrupt vector address as part of an interrupt request acknowledge cycle, the PIH provides the address in the following format:

*zzzz zzzz zzxxxxy*0

where:

zzzz zzzz zz	is the ten address bits from the base-address register (Figure 6-4).
xxxx	is the hex value of the highest priority interrupt request pending
y	is zero if the request is for standard interrupt service and one if the request is for PTS service.

The last bit is always zero. The CPU automatically adds FF0000H to the 16-bit vector because all interrupt vectors are located in page FFH.

NOTE

Always initialize the base-address registers with the starting address of the appropriate PIH vectors in special-purpose memory (i.e., PIH0_VEC_BASE = 20C0H and PIH1_VEC_BASE = 2100H).

If no PIH interrupt request is active when the CPU requests the interrupt vector address, the PIH responds with 2014H for a standard interrupt cycle or 2016H for a PTS interrupt cycle as shown in the following example.

0010 0000 0001 01*y*0

where:

y is zero if the request is for standard interrupt service and one if the request is for PTS service.

These vectors are reserved for dummy interrupt or PTS cycles. Initialize address FF2014H to point to a simple routine that contains only a RET instruction and address FF2016H to point to the zero register (0000H).

PIH*x*_VEC_BASE
***x* = 0–1**

Address: 1E92H, 1EA2H
Reset State: XXXXH

The peripheral interrupt handler *x* vector base-address (PIH*x*_VEC_BASE) registers contain address bits 6–15 of the interrupt vector. Your code must initialize this register before enabling any PIH interrupt sources.

15							8
VA15	VA14	VA13	VA12	VA11	VA10	VA9	VA8
7							**0**
VA7	VA6	—	—	—	—	—	—

Bit Number	Bit Mnemonic	Function
15:6	VA15:6	Peripheral Interrupt Handler Vector Base-address Bits 6 through 15 This register contains address bits 6 through 15 for the PIH*x* interrupt vectors. Always initialize these registers to the following values: PIH0_VEC_BASE = 20C0H PIH1_VEC_BASE = 2100H
5:0	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.

Figure 6-4. Peripheral Interrupt Handler *x* Vector Base (PIH*x*_VEC_BASE) Registers

6.3.2 Special Interrupts

Four special interrupt sources are always enabled: unimplemented opcode, software trap, and NMI, and stack overflow. These interrupts are not affected by the EI (enable interrupts) and DI (disable interrupts) instructions, and they cannot be masked. All of these interrupts are serviced by the interrupt controller; they cannot be assigned to the PTS. Only NMI and stack overflow go through the transition detector and priority resolver. The other two special interrupts go directly to the interrupt controller for servicing. Be aware that these interrupts are often assigned to special functions in development tools.

6.3.2.1 Unimplemented Opcode

If the CPU attempts to execute an unimplemented opcode, an indirect vector through location FF2012H occurs. This prevents random software execution during hardware and software failures. The interrupt vector should contain the starting address of an error routine that will not further corrupt an already erroneous situation. When an unimplemented opcode interrupt occurs, no other interrupt request can be acknowledged until after the next instruction executes.

6.3.2.2 Software Trap

The TRAP instruction (opcode F7H) causes an interrupt call that is vectored through location FF2010H (default). This interrupt is useful when debugging software or generating software interrupts. When the TRAP instruction generates the interrupt, no other interrupt request can be acknowledged until after the next instruction executes.

6.3.2.3 NMI

The external NMI pin generates a nonmaskable interrupt for implementation of critical interrupt routines. NMI has a higher priority than all the prioritized interrupts. (Only the unimplemented opcode and software trap interrupts have higher priority.) It is passed directly from the transition detector to the priority resolver, and it vectors indirectly through location FF203EH.The NMI pin is sampled during phase 2 (CLKOUT high) and is latched internally. Because interrupts are edge-triggered, only one interrupt is generated, even if the pin is held high.

If your system does not use the NMI interrupt, connect the NMI pin to V_{SS} to prevent spurious interrupts.

6.3.2.4 Stack Overflow

The stack overflow module monitors the value of the stack pointer (SP) and generates a nonmaskable interrupt request if the value is outside the boundaries you specify. This module helps to ensure data integrity. Without the stack overflow module, the stack could grow too large and corrupt other data, causing a fatal system error.

If an overflow occurs, the stack overflow module sets the stack overflow interrupt pending bit in INT_PEND1. Your interrupt service routine must write to the STACK_TOP register to re-enable the overflow detection circuitry. Otherwise, the stack overflow module is unable to signal another overflow. Chapter 5, "Stack Overflow Module," describes the stack overflow module in detail.

6.3.3 External Interrupt Signal

A momentary spike on the EXTINT pin may be interpreted as an interrupt request. To prevent this type of erroneous interrupt, hold EXTINT low when it is inactive.

6.3.4 Shared Interrupt Requests

The event processor array can generate many individual interrupt requests. To accommodate all EPA interrupt sources, two peripheral interrupt handlers (PIHs) were incorporated into the 8XC196EA. Each PIH handles 16 individual interrupt requests; each can be programmed to request service from either the interrupt controller or the PTS.

In addition, a capture overrun on EPA capture/compare channels 3–16 sets the interrupt pending bit for the EPA*x*_OVR interrupt. When the CPU receives an EPA*x*_OVR interrupt request, it cannot determine which channel had the overrun.

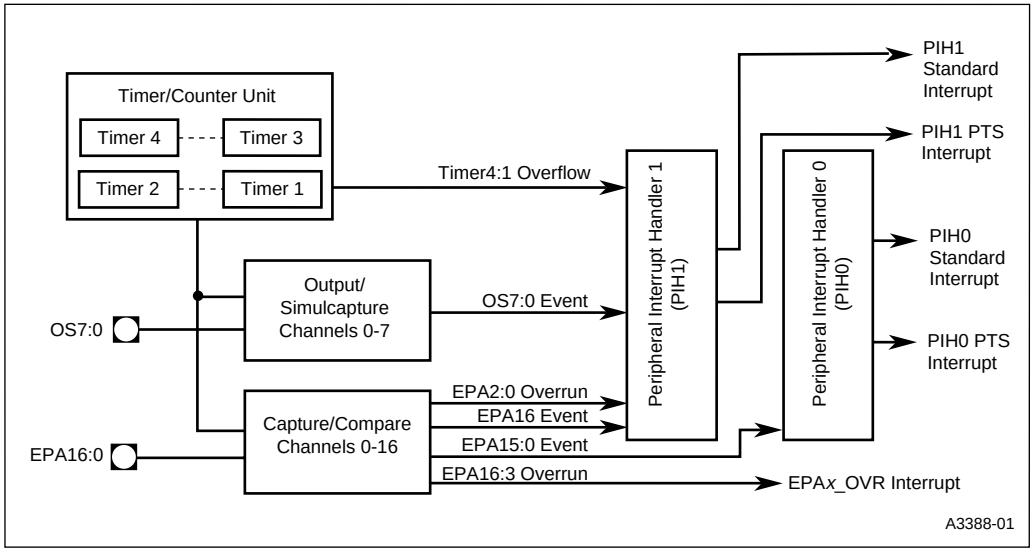


Figure 6-5. Peripheral Interrupt Handler (PIH) Interrupt Sources

6.3.5 End-of-PTS Interrupts

When the PTSCOUNT register decrements to zero at the end of a single transfer, block transfer, or missed-event routine, hardware clears the corresponding bit in the PTSSEL register (Figure 6-12 on page 6-23), which disables PTS service for that interrupt. It also sets the corresponding PTSSRV bit, requesting an end-of-PTS interrupt. An end-of-PTS interrupt has the same priority as a corresponding standard interrupt. The interrupt controller processes it with an interrupt service routine that is stored in the memory location pointed to by the standard interrupt vector. For example, the PTS services the SIO1 transmit (TI1) interrupt if PTSSEL.4 is set. The PTS interrupt vectors through FF2048H, but the corresponding end-of-PTS interrupt vectors through FF2008H, the standard TI1 interrupt vector. When the end-of-PTS interrupt vectors to the interrupt service routine, hardware clears the PTSSRV bit. The end-of-PTS interrupt service routine should reinitialize the PTSCB, if required, and set the appropriate PTSSEL bit to re-enable PTS interrupt service.

6.4 INTERRUPT LATENCY

Interrupt latency is the total delay between the time that the interrupt request is generated (not acknowledged) and the time that the microcontroller begins executing either the interrupt service routine or the PTS interrupt service routine. A delay occurs between the time that the interrupt request is detected and the time that it is acknowledged. An interrupt request is acknowledged when the current instruction or uninterruptable instruction sequence completes execution. If the interrupt request occurs during one of the last four state times of the instruction, it may not be acknowledged until after the next instruction finishes. This additional delay occurs because instructions are prefetched from external memory and assembled a minimum of four state times before they are executed. Thus, the maximum delay between interrupt request and acknowledgment

is four state times plus remaining instruction fetch time and the execution time of the next instruction.

When a standard interrupt request is acknowledged, the hardware clears the interrupt pending bit and forces a call to the address contained in the corresponding interrupt vector. When a PTS interrupt request is acknowledged, the hardware immediately vectors to the PTSCB and begins executing the PTS routine.

6.4.1 Situations that Increase Interrupt Latency

If an interrupt request occurs while any of the following instructions are executing, the interrupt will not be acknowledged until after the **next** instruction is executed:

- the signed prefix opcode (FE) for the two-byte, signed multiply and divide instructions
- a *protected instruction*: DI, EI, DPTS, EPTS, POPA, POPF, PUSHA, PUSHF (see Appendix A for descriptions of these instructions)
- a read-modify-write instruction: AND, ANDB, OR, ORB, XOR, XORB
- the unimplemented opcode interrupt and the software trap interrupt

Each PTS cycle within a PTS routine cannot be interrupted. A PTS cycle is the entire PTS response to a single interrupt request. In block transfer mode, a PTS cycle consists of the transfer of an entire block of bytes or words. This means a worst-case latency of 500 states if you assume a block transfer of 32 words from one external memory location to another. See Table 6-6 on page 6-18 for PTS cycle execution times.

6.4.2 Calculating Latency

The maximum latency occurs when the interrupt request occurs too late (four states before the current instruction finishes executing) for acknowledgment following the current instruction. The following worst-case calculation assumes that the current instruction is not a protected instruction To calculate latency, add the following terms:

- Time for the interrupt request to be detected (4 state times).
  - — One state each to clock edge, synchronize the interrupt, prioritize the interrupt request, and request interrupt service.
- Time for the current instruction to finish execution (4 state times).
  - — If this is a protected instruction, the instruction that follows it must also execute before the interrupt can be acknowledged. Add the execution time of the instruction that follows a protected instruction.
- Time for the next instruction to execute. (See Appendix A for instruction execution times.)
  - — The longest instruction, NORML, takes 39 state times. However, the BMOV instruction could actually take longer if it is transferring a large block of data. If your code contains routines that transfer large blocks of data, you may get a more accurate worst-case value if you use the BMOV execution time in your calculation instead of NORML.

- For standard interrupts only, the response time to get the vector, force the call, and fetch the first instruction of the service routine.
  - — in 64-Kbyte mode, 11 state times for an internal stack or 13 for an external (outside register RAM) stack, assuming a zero-wait-state bus.
  - — in 2-Mbyte mode, 15 state times for an internal stack or 18 for an external (outside register RAM) stack, assuming a zero-wait-state bus.

6.4.2.1 Worst-case Interrupt Latency

Figure 6-6 illustrates worst-case interrupt latency. In 64-Kbyte mode, the worst-case delay for an interrupt is 56 state times (4 + 39 + 11 + 2) if the stack is in external memory. In 2-Mbyte mode, the worst-case delay increases to 61 state times (4 + 39 + 15 + 3). This delay time does **not** include the time needed to execute the first instruction in the interrupt service routine or to execute the instruction following a protected instruction.

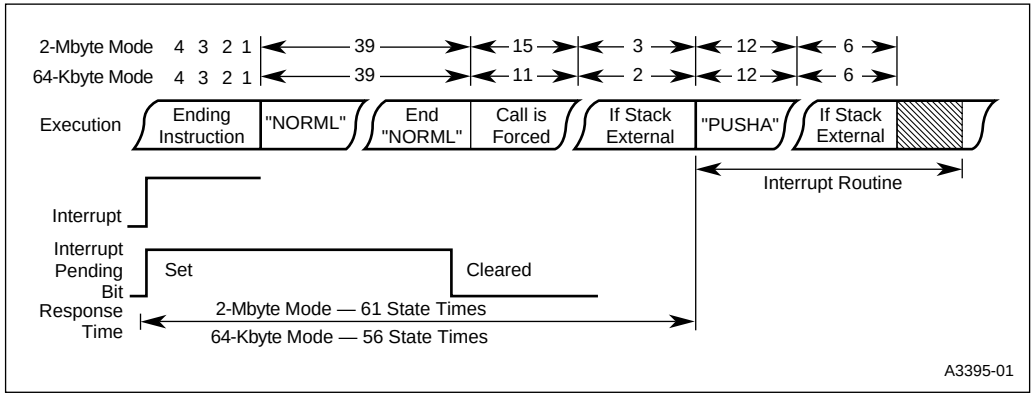


Figure 6-6. Worst-case Interrupt Response Time

6.4.2.2 PTS Interrupt Latency

In both 64-Kbyte and 2-Mbyte modes, the maximum delay for a PTS interrupt is 43 state times (4 + 39) as shown in Figure 6-7. This delay time does not include the added delay if a protected instruction is being executed or if a PTS request is already in progress. See Table 6-6 for execution times for PTS cycles.

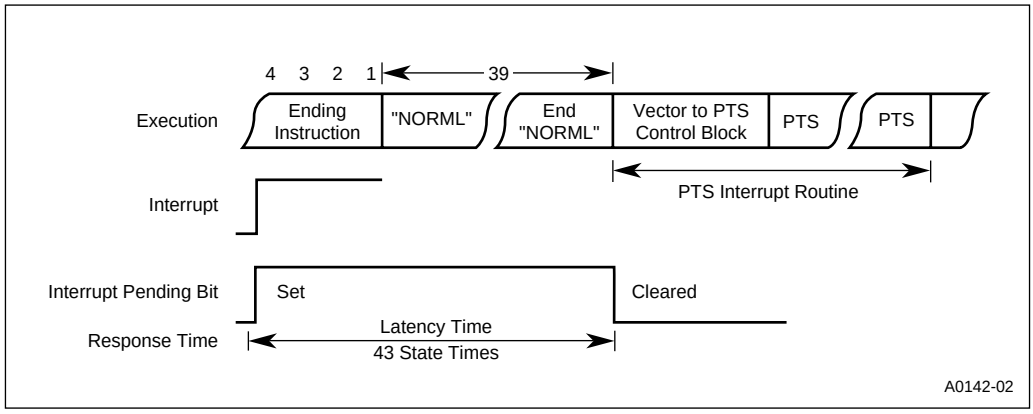


Figure 6-7. PTS Interrupt Response Time

Table 6-6. Execution Times for PTS Cycles

PTS Mode	Execution Time (in State Times)
Single transfer mode register/register† memory/register† memory/memory†	 22 per byte or word transfer + 1 25 per byte or word transfer + 1 28 per byte or word transfer + 1
Block transfer mode register/register† memory/register† memory/memory†	 14 + 8 per byte or word transfer (1 minimum) 17 + 8 per byte or word transfer (1 minimum) 20 + 8 per byte or word transfer (1 minimum)
Dummy mode	13
Missed-event mode	27

† *Register* indicates an access to the register file or a peripheral SFR. *Memory* indicates an access to a memory-mapped register, I/O, or memory. See Table 6-4 on page 6-7 for address information.

6.5 PROGRAMMING THE INTERRUPTS

Table 6-7 describes how to program each maskable interrupt.

Table 6-7. Programming the Interrupts

To perform this function:	Your code must:
Enable interrupt controller service for the maskable interrupts	Execute the EI instruction.
Disable interrupt controller service for the maskable interrupts, including the end-of-PTS interrupts	Execute the DI instruction.
Enable PTS service for the maskable interrupts (Notes 1, 2)	Execute the EPTS instruction.
Disable PTS service for the maskable interrupts	Execute the DPTS instruction.
Disable an individual maskable interrupt	**Clear** the interrupt's mask bit in the INT_MASK or INT_MASK1 register (Figure 6-8 or 6-9).
Disable a PIH interrupt source	**Clear** the interrupt's mask bit in the PIH*x*_INT_MASK register (Figure 6-10 or 6-11).
Enable a maskable interrupt for standard interrupt service (Note 3)	**Set** the interrupt's mask bit in the INT_MASK or INT_MASK1 register (Figure 6-8 or 6-9) and **clear** its PTS select bit (Figure 6-12).
Enable a maskable interrupt for PTS interrupt service	**Set** the interrupt's mask bit in the INT_MASK or INT_MASK1 register (Figure 6-8 or 6-9) and **set** its PTS select bit (Figure 6-12).
Enable a PIH interrupt for standard interrupt service	First enable the PIH interrupt source and select standard interrupt service: **Set** the interrupt's mask bit in the PIH*x*_INT_MASK register (Figure 6-10 or 6-11) and **clear** its PTS select bit in the PIH*x*_PTSSEL register (Figure 6-12). Then enable the PIH*x*_INT interrupt request: **Set** its mask bit in the INT_MASK or INT_MASK1 register (Figure 6-8 or 6-9) and **clear** its PTS select bit (Figure 6-12).
Enable a PIH interrupt for PTS interrupt service	First enable the PIH interrupt source and select PTS interrupt service: **Set** the interrupt's mask bit in the PIH*x*_INT_MASK register (Figure 6-10 or 6-11) and **set** its PTS select bit in the PIH*x*_PTSSEL register (Figure 6-12). Then enable the PIH*x*_PTS interrupt: **Set** its mask bit in the INT_MASK or INT_MASK1 register (Figure 6-8 or 6-9) and **set** its PTS select bit (Figure 6-12).

NOTES:
1. When you assign an interrupt to the PTS, you must set up a PTS control block (PTSCB) for each interrupt source (see "Initializing the PTS Control Blocks" on page 6-30).
2. PTS service is not useful for shared interrupts because the PTS cannot readily determine the source of these interrupts.
3. Never select standard interrupt service for the PIH*x*_PTS interrupt. If you do, the interrupt controller will try to service the PIH*x*_PTS interrupt. If no normal interrupts are pending in the PIH, the PIH responds with a dummy interrupt and does not clear the PTS interrupt pending bit. The PIH will continue to request service, causing an infinite loop.

INT_MASK

Address: 0008H
Reset State: 00H

The interrupt mask (INT_MASK) register enables or disables (masks) individual interrupt requests. (The EI and DI instructions enable and disable servicing of all maskable interrupts.) INT_MASK is the low byte of the processor status word (PSW). PUSHF or PUSHA saves the contents of this register onto the stack and then clears this register. Interrupt calls cannot occur immediately following a push instruction. POPF or POPA restores it.

7							0
SDU	EXTINT	RI1	TI1	AD	EPA*x*_OVR	RI0	TI0

Bit Number	Function
7:0	Setting a bit enables the corresponding interrupt.

Bit Mnemonic	Interrupt Description
SDU	Serial Debug Unit Interrupt
EXTINT	External Interrupt Pin
RI1	SIO1 Receive
TI1	SIO1 Transmit
AD	A/D Conversion Complete
EPA*x*_OVR	EPA Channel 3–16 Overrun
RI0	SIO0 Receive
TI0	SIO0 Transmit

Figure 6-8. Interrupt Mask (INT_MASK) Register

INT_MASK1

Address: 0013H
Reset State: 00H

The interrupt mask 1 (INT_MASK1) register enables or disables (masks) individual interrupt requests. (The EI and DI instructions enable and disable servicing of all maskable interrupts.) INT_MASK1 can be read from or written to as a byte register. PUSHA saves this register on the stack and POPA restores it.

7							0
NMI	STACK	PIH0_PTS	PIH0_INT	PIH1_PTS	PIH1_INT	SSIO1	SSIO0

Bit Number	Function
7:0	Setting a bit enables the corresponding interrupt. **Bit Mnemonic** — **Interrupt Description** NMI† — Nonmaskable Interrupt STACK† — Stack Overflow Error PIH0_PTS — PIH0 PTS Service Request PIH0_INT — PIH0 Interrupt Request PIH1_PTS — PIH1 PTS Service Request PIH1_INT — PIH1 Interrupt Request SSIO1 — SSIO1 Transfer SSIO0 — SSIO0 Transfer † The NMI and stack overflow interrupts are always enabled. These nonfunctional mask bits exist for design symmetry with the INT_PEND1 register. Always write zeros to these bits.

Figure 6-9. Interrupt Mask 1 (INT_MASK1) Register

PIH0_INT_MASK

Address: 1E98H
Reset State: 0000H

The PIH0 interrupt mask (PIH0_INT_MASK) register enables or disables (masks) individual interrupt requests to peripheral interrupt handler 0. (The EI and DI instructions enable and disable servicing of all maskable interrupts.)

15							8
EPA15	EPA14	EPA13	EPA12	EPA11	EPA10	EPA9	EPA8
7							**0**
EPA7	EPA6	EPA5	EPA4	EPA3	EPA2	EPA1	EPA0

Bit Number	Function
15:0	Setting a bit enables the corresponding interrupt request to peripheral interrupt handler 0. **Bit Mnemonic** — **Interrupt Description** EPA15:0 — EPA Capture/Compare Channels 0–15

Figure 6-10. PIH0 Interrupt Mask (PIH0_INT_MASK) Register

PIH1_INT_MASK

Address: 1EA8H
Reset State: 0000H

The PIH1 interrupt mask (PIH1_INT_MASK) register enables or disables (masks) individual interrupt requests to the peripheral interrupt handler 1. (The EI and DI instructions enable and disable servicing of all maskable interrupts.)

15							8
EPA16	OS7	OS6	OS5	OS4	OS3	OS2	OS1
7							**0**
OS0	OVRTM1	OVRTM2	OVRTM3	OVRTM4	OVR0	OVR1	OVR2

Bit Number	Function
15:0	Setting a bit enables the corresponding interrupt request to peripheral interrupt handler 1. **Bit Mnemonic** — **Interrupt Description** EPA16 — EPA Capture/Compare Channel 16 OS7:0 — Output Simulcapture Channel 0–7 OVRTM1:4 — Timer 1–4 Overflow/Underflow OVR0:2 — EPA Channel 0–2 Capture Overrun

Figure 6-11. PIH1 Interrupt Mask (PIH1_INT_MASK) Register

PTSSEL

Address: 0004H
Reset State: 0000H

The PTS select (PTSSEL) register selects either a PTS microcode routine or a standard interrupt service routine for each interrupt request. Setting a bit selects a PTS microcode routine; clearing a bit selects a standard interrupt service routine. In PTS modes that use the PTSCOUNT register, hardware clears the corresponding PTSSEL bit when PTSCOUNT reaches zero. The end-of-PTS interrupt service routine must reset the PTSSEL bit to re-enable the PTS channel.

15							8
—	—	PIH0_PTS	0	PIH1_PTS	0	SSIO1	SSIO0

7							0
SDU	EXTINT	RI1	TI1	AD	EPA*x*_OVR	RI0	TI0

Bit Number	Function
15, 14	Reserved; for compatibility with future devices, write zeros to these bits.
12, 10	To guarantee proper device operation, write zeros to these bits.
13, 11, 9:0	Setting a bit causes the corresponding interrupt to be handled by a PTS microcode routine. The PTS interrupt vector locations are as follows:

Bit Mnemonic	Interrupt	PTS Vector
PIH0_PTS†	PIH0 PTS Service Request	see PIH0_PTSSEL
PIH1_PTS†	PIH1 PTS Service Request	see PIH1_PTSSEL
SSIO1	SSIO 1 Transfer	FF2052H
SSIO0	SSIO 0 Transfer	FF2050H
SDU	Serial Debug Unit Interrupt	FF204EH
EXTINT	External Interrupt Pin	FF204CH
RI1	SIO1 Receive	FF204AH
TI1	SIO1 Transmit	FF2048H
AD	A/D Conversion Complete	FF2046H
EPA*x*_OVR	EPA Channel 3–16 Overrun	FF2044H
RI0	SIO0 Receive	FF2042H
TI0	SIO0 Transmit	FF2040H

† Unless these interrupts are masked in the INT_MASK1 register, always write ones to these bits.

Figure 6-12. PTS Select (PTSSEL) Register

PIH0_PTSSEL Address: 1E96H
Reset State: 0000H

The PTS select (PIH0_PTSSEL) register for the peripheral interrupt handler 0 selects either a PTS service request or a standard interrupt service request for the corresponding interrupt. Setting a bit selects a PTS service request; clearing a bit selects a standard interrupt service request. In PTS modes that use the PTSCOUNT register, hardware clears the corresponding bit in both the PIH0_PTSSEL and PTSSEL registers when PTSCOUNT reaches zero. The end-of-PTS interrupt service routine must reset the PIH0_PTSSEL bit to re-enable the PTS channel.

15							8
EPA15	EPA14	EPA13	EPA12	EPA11	EPA10	EPA9	EPA8
7							**0**
EPA7	EPA6	EPA5	EPA4	EPA3	EPA2	EPA1	EPA0

Bit Number	Function
15:0	Setting a bit causes the peripheral interrupt handler 0 (PIH0) to generate the PIH0_PTS interrupt request when the corresponding interrupt is generated and enabled. The PTS interrupt vector locations are as follows:

Bit Mnemonic	Interrupt	PTS Vector
EPA15	EPA Capture/Compare Channel 15	FF20FEH
EPA14	EPA Capture/Compare Channel 14	FF20FAH
EPA13	EPA Capture/Compare Channel 13	FF20F6H
EPA12	EPA Capture/Compare Channel 12	FF20F2H
EPA11	EPA Capture/Compare Channel 11	FF20EEH
EPA10	EPA Capture/Compare Channel 10	FF20EAH
EPA9	EPA Capture/Compare Channel 9	FF20E6H
EPA8	EPA Capture/Compare Channel 8	FF20E2H
EPA7	EPA Capture/Compare Channel 7	FF20DEH
EPA6	EPA Capture/Compare Channel 6	FF20DAH
EPA5	EPA Capture/Compare Channel 5	FF20D6H
EPA4	EPA Capture/Compare Channel 4	FF20D2H
EPA3	EPA Capture/Compare Channel 3	FF20CEH
EPA2	EPA Capture/Compare Channel 2	FF20CAH
EPA1	EPA Capture/Compare Channel 1	FF20C6H
EPA0	EPA Capture/Compare Channel 0	FF20C2H

Figure 6-13. PIH0 PTS Select (PIH0_PTSSEL) Register

PIH1_PTSSEL

Address: 1EA6H
Reset State: 0000H

The PTS select register for the peripheral interrupt handler 1 (PIH1_PTSSEL) selects either a PTS service request or a standard interrupt service request for the corresponding interrupt. Setting a bit selects a PTS service request; clearing a bit selects a standard interrupt service request. In PTS modes that use the PTSCOUNT register, hardware clears the corresponding bit in both the PIH1_PTSSEL and PTSSEL registers when PTSCOUNT reaches zero. The end-of-PTS interrupt service routine must reset the PIH1_PTSSEL bit to re-enable the PTS channel.

15							8
EPA16	OS7	OS6	OS5	OS4	OS3	OS2	OS1
7							**0**
OS0	OVRTM1	OVRTM2	OVRTM3	OVRTM4	OVR0	OVR1	OVR2

Bit Number	Function
15:0	Setting a bit causes the peripheral interrupt handler 1(PIH1) to generate the PIH1_PTS interrupt request when the corresponding interrupt is generated and enabled. The PTS interrupt vector locations are as follows:

Bit Mnemonic	Interrupt	PTS Vector
EPA16	EPA Capture/Compare Channel 16	FF213EH
OS7	Output Simulcapture Channel 7	FF213AH
OS6	Output Simulcapture Channel 6	FF2136H
OS5	Output Simulcapture Channel 5	FF2132H
OS4	Output Simulcapture Channel 4	FF212EH
OS3	Output Simulcapture Channel 3	FF212AH
OS2	Output Simulcapture Channel 2	FF2126H
OS1	Output Simulcapture Channel 1	FF2122H
OS0	Output Simulcapture Channel 0	FF211EH
OVRTM1	Timer 1 Overflow/Underflow	FF211AH
OVRTM2	Timer 2 Overflow/Underflow	FF2116H
OVRTM3	Timer 3 Overflow/Underflow	FF2112H
OVRTM4	Timer 4 Overflow/Underflow	FF210EH
OVR0	EPA Channel 0 Capture Overrun	FF210AH
OVR1	EPA Channel 1 Capture Overrun	FF2106H
OVR2	EPA Channel 2 Capture Overrun	FF2102H

Figure 6-14. PIH1 PTS Select (PIH1_PTSSEL) Register

6.5.1 Modifying Interrupt Priorities

Your software can modify the default priorities of maskable interrupts by controlling the interrupt mask registers (INT_MASK and INT_MASK1). For example, you can specify which interrupts, if any, can interrupt an interrupt service routine. The following code shows one way to prevent all interrupts, except EXTINT (priority 6), from interrupting an SIO1 receive interrupt (RI1) service routine (priority 5).

```
SERIAL_RI1_ISR:
     PUSHA                     ; Save PSW, INT_MASK, INT_MASK1, & WSR.
                               ; No interrupts acknowledged until after next
                               ; instruction
     LDB INT_MASK, #01000000B  ; Enable EXTINT only
     EI                        ; Enable interrupt servicing

                               ; Service the RI1 interrupt

     POPA                      ; Restore PSW, INT_MASK, INT_MASK1, & WSR

     RET
CSEG AT 0FF200AH               ; fill in interrupt table
   DCW LSW SERIAL_RI1_IS       ; LSW is a compiler directive that means
                               ; least-significant word of vector address
   END
```

Note that location FF200AH in the interrupt vector table must be loaded with the value of the label SERIAL_RI1_ISR before the interrupt request occurs and that the interrupt must be enabled for this routine to execute.

This routine, like all interrupt service routines, is handled in the following manner:

1. After the hardware detects and prioritizes an interrupt request, it generates and executes an interrupt call. This pushes the program counter onto the stack and then loads it with the contents of the vector corresponding to the highest priority, pending, unmasked interrupt. The hardware will not allow another interrupt call until after the first instruction of the interrupt service routine is executed.
2. The PUSHA instruction saves the contents of the PSW, INT_MASK, INT_MASK1, and window selection register (WSR) onto the stack and then clears the PSW, INT_MASK, and INT_MASK1 registers. In addition to the arithmetic flags, the PSW contains the global interrupt enable bit (I) and the PTS enable bit (PSE). By clearing the PSW and the interrupt mask registers, PUSHA effectively masks all maskable interrupts, disables standard interrupt servicing, and disables the PTS. Because PUSHA is a protected instruction, it also inhibits interrupt calls until after the next instruction executes.
3. The LDB INT_MASK1 instruction enables those interrupts that you choose to allow to interrupt the service routine. In this example, only EXTINT can interrupt the receive interrupt service routine. By enabling or disabling interrupts, the software establishes its own interrupt servicing priorities.
4. The EI instruction re-enables interrupt processing and inhibits interrupt calls until after the next instruction executes.
5. The actual interrupt service routine executes within the priority structure established by the software.
6. At the end of the service routine, the POPA instruction restores the original contents of the PSW, INT_MASK, INT_MASK1, and WSR registers; any changes made to these registers during the interrupt service routine are overwritten. Because interrupt calls cannot occur immediately following a POPA instruction, the last instruction (RET) will execute before another interrupt call can occur.

Notice that the “preamble” and exit code for this routine do not save or restore register RAM. The interrupt service routine is assumed to allocate its own private set of registers from the lower register file. The general-purpose register RAM in the lower register file makes this quite practical. In addition, the RAM in the upper register file is available via *windowing* (see “Windowing” on page 4-17).

6.5.2 Determining the Source of an Interrupt

When hardware detects an interrupt, it sets the corresponding bit in the INT_PEND or INT_PEND1 register (Figures 6-15 or 6-16). It sets the bit even if the individual interrupt is disabled (masked). Software can read INT_PEND and INT_PEND1 to determine which interrupts are pending. If a PIH interrupt request (PIH*x*_INT or PIH*x*_PTS) is pending, software can read the PIH*x*_INT_PEND register (Figure 6-17 or 6-18) to determine the source of the interrupt request.

When the CPU acknowledges an interrupt request, hardware clears the bit in the INT_PEND or INT_PEND1 register.

Software can generate an interrupt by setting a bit in the INT_PEND or INT_PEND1 register. Use the read-modify-write instructions, such as AND and OR, to modify these registers.

```
ANDB INT_PEND, #11111110B; Clears the TI0 pending bit
ORB  INT_PEND, #00000001B; Sets the TI0 pending bit
```

Other methods could result in a partial interrupt cycle. For example, an interrupt could occur during an instruction sequence that loads the contents of the interrupt pending register into a temporary register, modifies the contents of the temporary register, and then writes the contents of the temporary register back into the interrupt pending register. If the interrupt occurs during one of the last four states of the second instruction, it will not be acknowledged until after the completion of the third instruction. Because the third instruction overwrites the contents of the interrupt pending register, the interrupt will not be acknowledged.

If software clears the pending bit in the PIH*x*_INT_PEND register while the shared interrupt pending bit is still set, the CPU will try to service the interrupt. When this occurs, the PIH responds with a dummy interrupt vector address (2014H for standard interrupts and 2016H for PTS interrupts). Initialize FF2014H to point to a simple routine that contains only a RET instruction, and initialize FF2016H to point to address 0000H (see “Dummy Mode” on page 6-40).

INT_PEND Address: 0009H
Reset State: 00H

When hardware detects an interrupt request, it sets the corresponding bit in the interrupt pending (INT_PEND or INT_PEND1) registers. When the vector is taken, the hardware clears the pending bit. Software can generate an interrupt by setting the corresponding interrupt pending bit.

7							0
SDU	EXTINT	RI1	TI1	AD	EPA*x*_OVR	RI0	TI0

Bit Number	Function
7:0	Any set bit indicates that the corresponding interrupt is pending. The interrupt bit is cleared when processing transfers to the corresponding interrupt vector.

Bit Mnemonic	Interrupt Description
SDU	Serial Debug Unit Interrupt
EXTINT	External Interrupt Pin
RI1	SIO1 Receive
TI1	SIO1 Transmit
AD	A/D Conversion Complete
EPA*x*_OVR	EPA Channel 3–16 Overrun†
RI0	SIO0 Receive
TI0	SIO0 Transmit

† It is not possible to determine which EPA channel caused the EPA*x*_OVR interrupt request.

Figure 6-15. Interrupt Pending (INT_PEND) Register

INT_PEND1 Address: 0012H
Reset State: 00H

When hardware detects an interrupt request, it sets the corresponding bit in the interrupt pending (INT_PEND or INT_PEND1) registers. When the vector is taken, the hardware clears the pending bit. Software can generate an interrupt by setting the corresponding interrupt pending bit.

7							0
NMI	STACK	PIH0_PTS	PIH0_INT	PIH1_PTS	PIH1_INT	SSIO1	SSIO0

Bit Number	Function
7:0	Any set bit indicates that the corresponding interrupt is pending. The interrupt bit is cleared when processing transfers to the corresponding interrupt vector.

Bit Mnemonic	Interrupt Description
NMI	Nonmaskable Interrupt
STACK	Stack Overflow Error
PIH0_PTS	PIH0 PTS Service Request
PIH0_INT	PIH0 Interrupt Request
PIH1_PTS	PIH1 PTS Service Request
PIH1_INT	PIH1 Interrupt Request
SSIO1	SSIO1 Transfer
SSIO0	SSIO0 Transfer

Figure 6-16. Interrupt Pending 1 (INT_PEND1) Register

PIH0_INT_PEND Address: 1E9AH
Reset State: 0000H

When the peripheral interrupt handler 0 detects an interrupt request, it sets the corresponding bit in the interrupt pending (PIH0_INT_PEND) register. When the vector is taken, the hardware clears the pending bit. Software can generate an interrupt by setting the corresponding interrupt pending bit.

15							8
EPA15	EPA14	EPA13	EPA12	EPA11	EPA10	EPA9	EPA8
7							**0**
EPA7	EPA6	EPA5	EPA4	EPA3	EPA2	EPA1	EPA0

Bit Number	Function
15:0	Any set bit indicates that the corresponding PIH0 interrupt is pending. The interrupt bit is cleared when processing transfers to the corresponding interrupt vector.

Bit Mnemonic	Interrupt Description
EPA15:0	EPA Capture/Compare Channels 0–15

Figure 6-17. PIH0 Interrupt Pending (PIH0_INT_PEND) Register

PIH1_INT_PEND Address: 1EAAH
Reset State: 0000H

When the peripheral interrupt handler 1 detects an interrupt request, it sets the corresponding bit in the interrupt pending (PIH1_INT_PEND) register. When the vector is taken, the hardware clears the pending bit. Software can generate an interrupt by setting the corresponding interrupt pending bit.

15							8
EPA16	OS7	OS6	OS5	OS4	OS3	OS2	OS1
7							**0**
OS0	OVRTM1	OVRTM2	OVRTM3	OVRTM4	OVR0	OVR1	OVR2

Bit Number	Function
15:0	Any set bit indicates that the corresponding PIH1 interrupt is pending. The interrupt bit is cleared when processing transfers to the corresponding interrupt vector. **Bit Mnemonic** **Interrupt Description** EPA16 EPA Capture/Compare Channel 16 OS7:0 Output Simulcapture Channel 0–7 OVRTM1:4 Timer 1–4 Overflow/Underflow OVR0:2 EPA Channel 0–2 Capture Overrun

Figure 6-18. PIH1 Interrupt Pending (PIH1_INT_PEND) Register

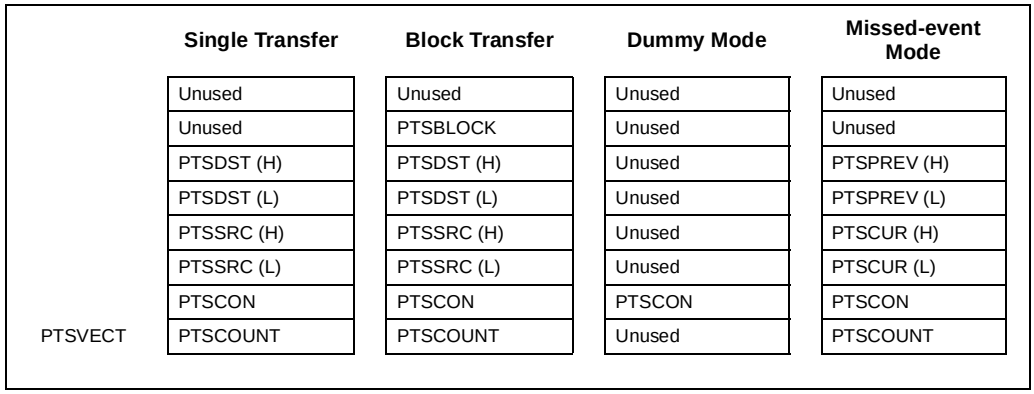
6.6 INITIALIZING THE PTS CONTROL BLOCKS

Each PTS interrupt requires a block of data, in register RAM, called the PTS control block (PTSCB). The PTSCB identifies which PTS microcode routine will be invoked and sets up the specific parameters for the routine. You must set up the PTSCB for each interrupt source **before** enabling the corresponding PTS interrupts.

Store the address of the first (lowest) PTSCB byte in the PTS vector table in special-purpose memory (see "Special-purpose Memory in Page FFH" on page 4-7). Figure 6-19 shows the PTSCB for each PTS mode. Unused PTSCB bytes can be used as extra RAM.

NOTE

The PTSCB must be located in the internal register file. The location of the first byte of the PTSCB must be aligned on a quad-word boundary (an address evenly divisible by 8).



	Single Transfer	Block Transfer	Dummy Mode	Missed-event Mode
	Unused	Unused	Unused	Unused
	Unused	PTSBLOCK	Unused	Unused
	PTSDST (H)	PTSDST (H)	Unused	PTSPREV (H)
	PTSDST (L)	PTSDST (L)	Unused	PTSPREV (L)
	PTSSRC (H)	PTSSRC (H)	Unused	PTSCUR (H)
	PTSSRC (L)	PTSSRC (L)	Unused	PTSCUR (L)
	PTSCON	PTSCON	PTSCON	PTSCON
PTSVECT	PTSCOUNT	PTSCOUNT	Unused	PTSCOUNT

Figure 6-19. PTS Control Blocks

6.6.1 Specifying the PTS Count

For single transfer, block transfer, and missed-event routines, the first location of the PTSCB contains an 8-bit value called PTSCOUNT. This value defines the number of interrupts that will be serviced by the PTS routine. The PTS decrements PTSCOUNT after each PTS cycle. When PTSCOUNT reaches zero, hardware clears the corresponding PTSSEL bit and sets the PTSSRV bit (Figure 6-20), which requests an end-of-PTS interrupt. If PIH*x*_PTS is the interrupt source, then hardware also clears the corresponding PIH*x*_PTSSEL bit and sets the PIH*x*_PTSSRV bit (Figure 6-12 and “Windowing” on page 4-17).

The end-of-PTS interrupt service routine should reinitialize the PTSCB, if required, and set the appropriate PTSSEL and PIH*x*_PTSSEL bits to re-enable PTS interrupt service.

PTSSRV Address: 0006H
Reset State: 0000H

The PTS service (PTSSRV) register is used by the hardware to indicate that the final PTS interrupt has been serviced by the PTS routine. When PTSCOUNT reaches zero, hardware clears the corresponding PTSSEL bit and sets the PTSSRV bit, which requests the end-of-PTS interrupt. When the end-of-PTS interrupt is called, hardware clears the PTSSRV bit. The end-of-PTS interrupt service routine must set the PTSSEL bit to re-enable the PTS channel.

15							8
—	—	PIH0_PTS	—	PIH1_PTS	—	SSIO1	SSIO0
7							**0**
SDU	EXTINT	RI1	TI1	AD	EPA*x*_OVR	RI0	TI0

Bit Number	Function
15, 14, 12, 10	Reserved; always write as zeros.
13, 11, 9:0	A bit is set by hardware to request an end-of-PTS interrupt for the corresponding interrupt through its standard interrupt vector. The standard interrupt vector locations are as follows:

Bit Mnemonic	Interrupt	Standard Vector
PIH0_PTS	PIH0 PTS Service Request	see PIH0_PTSSRV
PIH1_PTS	PIH1 PTS Service Request	see PIH1_PTSSRV
SSIO0	SSIO 0 Transfer	FF2032H
SSIO0	SSIO 0 Transfer	FF2030H
SDU	Serial Debug Unit Interrupt	FF200EH
EXTINT	External Interrupt Pin	FF200CH
RI1	SIO1 Receive	FF200AH
TI1	SIO1 Transmit	FF2008H
AD	A/D Conversion Complete	FF2006H
EPA*x*_OVR	EPA Channel 3–16 Overrun	FF2004H
RI0	SIO0 Receive	FF2002H
TI0	SIO0 Transmit	FF2000H

Figure 6-20. PTS Service (PTSSRV) Register

PIH0_PTSSRV

Address: 1E94H
Reset State: 0000H

The PTS service (PIH0_PTSSRV) register for peripheral interrupt handler 0 is used by the hardware to indicate that the final PTS interrupt has been serviced by the PTS routine. When PTSCOUNT reaches zero, hardware clears the corresponding PIH0_PTSSEL bit and sets the PIH0_PTSSRV bit, which requests the end-of-PTS interrupt. When the end-of-PTS interrupt is called, hardware clears the PIH0_PTSSRV bit. The end-of-PTS interrupt service routine must set the PIH0_PTSSEL bit to re-enable the PTS channel.

15							8
EPA15	EPA14	EPA13	EPA12	EPA11	EPA10	EPA9	EPA8
7							**0**
EPA7	EPA6	EPA5	EPA4	EPA3	EPA2	EPA1	EPA0

Bit Number	Function
15:0	A bit is set by hardware to request an end-of-PTS interrupt for the corresponding interrupt through its standard interrupt vector. The standard interrupt vector locations are as follows:

Bit Mnemonic	Interrupt	Standard Vector
EPA15	EPA Capture/Compare Channel 15	FF20FCH
EPA14	EPA Capture/Compare Channel 14	FF20F8H
EPA13	EPA Capture/Compare Channel 13	FF20F4H
EPA12	EPA Capture/Compare Channel 12	FF20F0H
EPA11	EPA Capture/Compare Channel 11	FF20ECH
EPA10	EPA Capture/Compare Channel 10	FF20E8H
EPA9	EPA Capture/Compare Channel 9	FF20E4H
EPA8	EPA Capture/Compare Channel 8	FF20E0H
EPA7	EPA Capture/Compare Channel 7	FF20DCH
EPA6	EPA Capture/Compare Channel 6	FF20D8H
EPA5	EPA Capture/Compare Channel 5	FF20D4H
EPA4	EPA Capture/Compare Channel 4	FF20D0H
EPA3	EPA Capture/Compare Channel 3	FF20CCH
EPA2	EPA Capture/Compare Channel 2	FF20C8H
EPA1	EPA Capture/Compare Channel 1	FF20C4H
EPA0	EPA Capture/Compare Channel 0	FF20C0H

Figure 6-21. PIH0 PTS Service (PIH0_PTSSRV) Register

PIH1_PTSSRV Address: 1EA4H
Reset State: 0000H

The PTS service (PIH1_PTSSRV) register for peripheral interrupt handler 1 is used by the hardware to indicate that the final PTS interrupt has been serviced by the PTS routine. When PTSCOUNT reaches zero, hardware clears the corresponding PIH1_PTSSEL bit and sets the PIH1_PTSSRV bit, which requests the end-of-PTS interrupt. When the end-of-PTS interrupt is called, hardware clears the PIH1_PTSSRV bit. The end-of-PTS interrupt service routine must set the PIH1_PTSSEL bit to re-enable the PTS channel.

15							8
EPA16	OS7	OS6	OS5	OS4	OS3	OS2	OS1
7							**0**
OS0	OVRTM1	OVRTM2	OVRTM3	OVRTM4	OVR0	OVR1	OVR2

Bit Number	Function
15:0	A bit is set by hardware to request an end-of-PTS interrupt for the corresponding interrupt through its standard interrupt vector. The standard interrupt vector locations are as follows:

Bit Mnemonic	Interrupt	Standard Vector
EPA16	EPA Capture/Compare Channel 16	FF213CH
OS7	Output Simulcapture Channel 7	FF2138H
OS6	Output Simulcapture Channel 6	FF2134H
OS5	Output Simulcapture Channel 5	FF2130H
OS4	Output Simulcapture Channel 4	FF212CH
OS3	Output Simulcapture Channel 3	FF2128H
OS2	Output Simulcapture Channel 2	FF2124H
OS1	Output Simulcapture Channel 1	FF2120H
OS0	Output Simulcapture Channel 0	FF211CH
OVRTM1	Timer 1 Overflow/Underflow	FF2118H
OVRTM2	Timer 2 Overflow/Underflow	FF2114H
OVRTM3	Timer 3 Overflow/Underflow	FF2110H
OVRTM4	Timer 4 Overflow/Underflow	FF210CH
OVR0	EPA Channel 0 Capture Overrun	FF2108H
OVR1	EPA Channel 1 Capture Overrun	FF2104H
OVR2	EPA Channel 2 Capture Overrun	FF2100H

Figure 6-22. PIH1 PTS Service (PIH1_PTSSRV) Register

6.6.2 Selecting the PTS Mode

The second byte of each PTSCB is always an 8-bit value called PTSCON. Bits 5–7 select the PTS mode (Figure 6-23). The function of bits 0–4 differ for each PTS mode. Refer to the sections that describe each mode in detail to see the function of these bits. Table 6-6 on page 6-18 lists the cycle execution times for each PTS mode.

PTSCON Address: PTSCB + 1

The PTS control (PTSCON) register selects the PTS mode and sets up control functions for that mode.

7							0
M2	M1	M0	†	†	†	†	†

Bit Number	Bit Mnemonic	Function
7:5	M2:0	PTS Mode These bits select the PTS mode: **M2 M1 M0** 0 0 0 dummy mode 0 0 1 block transfer 0 1 0 reserved 0 1 1 reserved 1 0 0 reserved 1 0 1 single transfer 1 1 0 missed-event 1 1 1 reserved

† The function of this bit depends upon which mode is selected. See the PTS control block description in each PTS mode section.

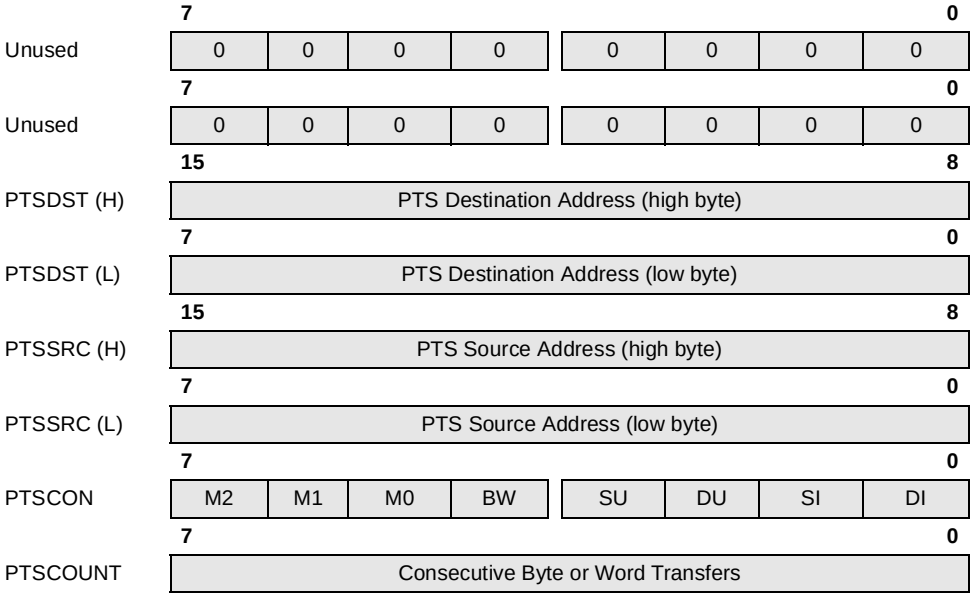
Figure 6-23. PTS Mode Selection Bits (PTSCON Bits 7:5)

6.6.3 Single Transfer Mode

In single transfer mode, an interrupt causes the PTS to transfer a single byte or word (selected by the BW bit in PTSCON) from one memory location to another. This mode is typically used with serial I/O or synchronous serial I/O interrupts. It can also be used with the EPA to move captured time values from the event-time register to internal RAM for further processing. See AP-445, *8XC196KR Peripherals: A User's Point of View*, for application examples with code. Figure 6-24 shows the PTS control block for single transfer mode.

PTS Single Transfer Mode Control Block

In single transfer mode, the PTS control block contains both a source and a destination address (PTSSRC and PTSDST), a control register (PTSCON), and a transfer count (PTSCOUNT).



Register	Location	Function
PTSDST	PTSCB + 4	PTS Destination Address Write the destination memory location to this register. A valid address is any unreserved memory location within page 00H; however, it must point to an even address if word transfers are selected.
PTSSRC	PTSCB + 2	PTS Source Address Write the source memory location to this register. A valid address is any unreserved memory location within page 00H; however, it must point to an even address if word transfers are selected.

Figure 6-24. PTS Control Block — Single Transfer Mode

PTS Single Transfer Mode Control Block (Continued)

Register	Location	Function	
PTSCON	PTSCB + 1	PTS Control Bits	
		M2:0	PTS Mode **M2 M1 M0** 1 0 1 single transfer mode
		BW	Byte/Word Transfer 0 = word transfer 1 = byte transfer
		SU†	Update PTSSRC 0 = reload original PTS source address after each byte or word transfer 1 = retain current PTS source address after each byte or word transfer
		DU†	Update PTSDST 0 = reload original PTS destination address after each byte or word transfer 1 = retain current PTS destination address after each byte or word transfer
		SI†	PTSSRC Autoincrement 0 = the contents of PTSSRC are not incremented after each byte or word transfer 1 = the contents of PTSSRC are incremented after each byte or word transfer
		DI†	PTSDST Autoincrement 0 = the contents of PTSDST are not incremented after each byte or word transfer 1 = the contents of PTSDST are incremented after each byte or word transfer
PTSCOUNT	PTSCB + 0	Consecutive Word or Byte Transfers Defines the number of words or bytes that will be transferred during the single transfer routine. Each word or byte transfer is one PTS cycle. Maximum value is 255.	

† The DU/DI bits and SU/SI bits are paired in single transfer mode. Each pair must be set or cleared together. However, the two pairs, DU/DI and SU/SI, do not need to be equal.

Figure 6-24. PTS Control Block — Single Transfer Mode (Continued)

The PTSCB in Table 6-8 defines nine PTS cycles. Each cycle moves a single word from location 20H to an external memory location. The PTS transfers the first word to location 6000H. Then it increments and updates the destination address and decrements the PTSCOUNT register; it does not increment the source address. When the second cycle begins, the PTS moves a second word from location 20H to location 6002H. When PTSCOUNT equals zero, the PTS will have filled locations 6000–600FH, and an end-of-PTS interrupt is generated.

Table 6-8. Single Transfer Mode PTSCB

Unused
Unused
PTSDST (H) = 60H
PTSDST (L) = 00H
PTSSRC (H) = 00H
PTSSRC (L) = 20H
PTSCON = A5H (Mode = 101, BW = 0, SI/SU = 0, DI/DU = 1)
PTSCOUNT = 09H

6.6.4 Block Transfer Mode

In block transfer mode, an interrupt causes the PTS to move a block of bytes or words from one memory location to another. See AP-445, *8XC196KR Peripherals: A User's Point of View*, for application examples with code. Figure 6-25 shows the PTS control block for block transfer mode.

In this mode, each PTS cycle consists of the transfer of an entire block of bytes or words. Because a PTS cycle cannot be interrupted, the block transfer mode can create long interrupt latency. The worst-case latency could be as high as 500 states if you assume a block transfer of 32 words from one external memory location to another, using an 8-bit bus with no wait states. See Table 6-6 on page 6-18 for execution times of PTS cycles.

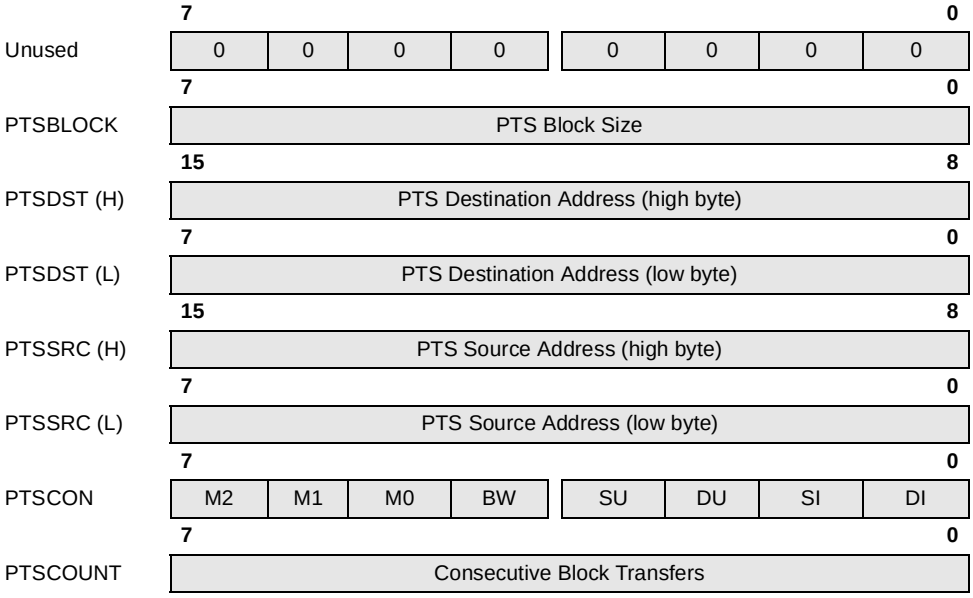
The PTSCB in Table 6-9 sets up three PTS cycles that will transfer five bytes from memory locations 20–24H to 6000–6004H (cycle 1), 6005–6009H (cycle 2), and 600A–600EH (cycle 3). The source and destination are incremented after each byte transfer, but the original source address is reloaded into PTSSRC at the end of each block-transfer cycle. In this routine, the PTS always gets the first byte from location 20H.

Table 6-9. Block Transfer Mode PTSCB

Unused
PTSBLOCK = 05H
PTSDST (H) = 60H
PTSDST (L) = 00H
PTSSRC (H) = 00H
PTSSRC (L) = 20H
PTSCON = 37H (Mode = 001; DI, SI, DU, BW = 1; SU = 0)
PTSCOUNT = 03H

PTS Block Transfer Mode Control Block

In block transfer mode, the PTS control block contains a block size (PTSBLOCK), a source and destination address (PTSSRC and PTSDST), a control register (PTSCON), and a transfer count (PTSCOUNT).



Register	Location	Function
PTSBLOCK	PTSCB + 6	PTS Block Size Specifies the number of bytes or words in each block. Valid values are 1–32, inclusive.
PTSDST	PTSCB + 4	PTS Destination Address Write the destination memory location to this register. A valid address is any unreserved memory location within page 00H; however, it must point to an even address if word transfers are selected.
PTSSRC	PTSCB + 2	PTS Source Address Write the source memory location to this register. A valid address is any unreserved memory location within page 00H; however, it must point to an even address if word transfers are selected.

Figure 6-25. PTS Control Block — Block Transfer Mode

PTS Block Transfer Mode Control Block (Continued)

Register	Location	Function	
PTSCON	PTSCB + 1	PTS Control Bits	
		M2:0	PTS Mode These bits select the PTS mode: **M2 M1 M0** 0 0 1 block transfer mode
		BW	Byte/Word Transfer 0 = word transfer 1 = byte transfer
		SU	Update PTSSRC 0 = reload original PTS source address after each block transfer is complete 1 = retain current PTS source address after each block transfer is complete
		DU	Update PTSDST 0 = reload original PTS destination address after each block transfer is complete 1 = retain current PTS destination address after each block transfer is complete
		SI	PTSSRC Autoincrement 0 = the contents of PTSSRC are not incremented after each byte or word transfer 1 = the contents of PTSSRC are incremented after each byte or word transfer
		DI	PTSDST Autoincrement 0 = the contents of PTSDST are not incremented after each byte or word transfer 1 = the contents of PTSDST are incremented after each byte or word transfer
PTSCOUNT	PTSCB + 0	Consecutive Block Transfers Defines the number of blocks that will be transferred during the block transfer routine. Each block transfer is one PTS cycle. Maximum number is 255.	

Figure 6-25. PTS Control Block — Block Transfer Mode (Continued)

6.6.5 Dummy Mode

The PTS dummy mode protects against situations where software manually clears the pending bits in the PIH*x*_PEND register before clearing the PIH*x*_PTS interrupt pending bit in the INT_PEND1 register. In this situation, the CPU may attempt to service the interrupt request after it is no longer valid. If this occurs, the PIH responds with the dummy PTS request vector address (2016H). The PTS aborts service when it reads the PTSCB and determines that the PTS dummy mode is selected. Write 0000H to location FF2016H so that the dummy PTS routine vectors to the zero-register and no register RAM locations are wasted. Figure 6-26 shows the PTS control block for dummy mode.

PTS Dummy Mode Control Block

In dummy mode, the PTS control block contains only a control register (PTSCON). All other PTSCB bytes can be used as extra RAM.

	7							0
PTSCON	M2	M1	M0	0	0	0	0	0
	7							0
Unused	0	0	0	0	0	0	0	0

Register	Location	Function	
Register	**Location**	**Function**	
PTSCON	PTSCB + 1	PTS Control Bits	
		M2:0	PTS Mode These bits select the PTS mode: **M2 M1 M0** 0 0 0 dummy mode

Figure 6-26. PTS Control Block — Dummy Mode

6.6.6 Missed-Event Mode

In missed-event mode, the PTS uses a pair of event processor array (EPA) capture/compare channels to generate an interrupt if an input pulse fails to occur within a predictable period of time (Figure 6-27). The PTS monitors a series of regular input pulses on the EPA channel and calculates when the next pulse should occur based upon the time of the last two pulses. It stores the calculated value into the event-time register (EPA*x*_TIME) of a second EPA channel. The second EPA channel compares the timer value to the contents of EPA*x*_TIME. If the next pulse does not occur before the two values are equal, an interrupt occurs. Figure 6-28 shows the PTS control block for missed-event mode.

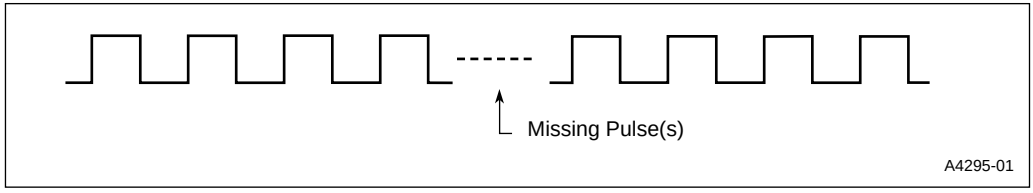
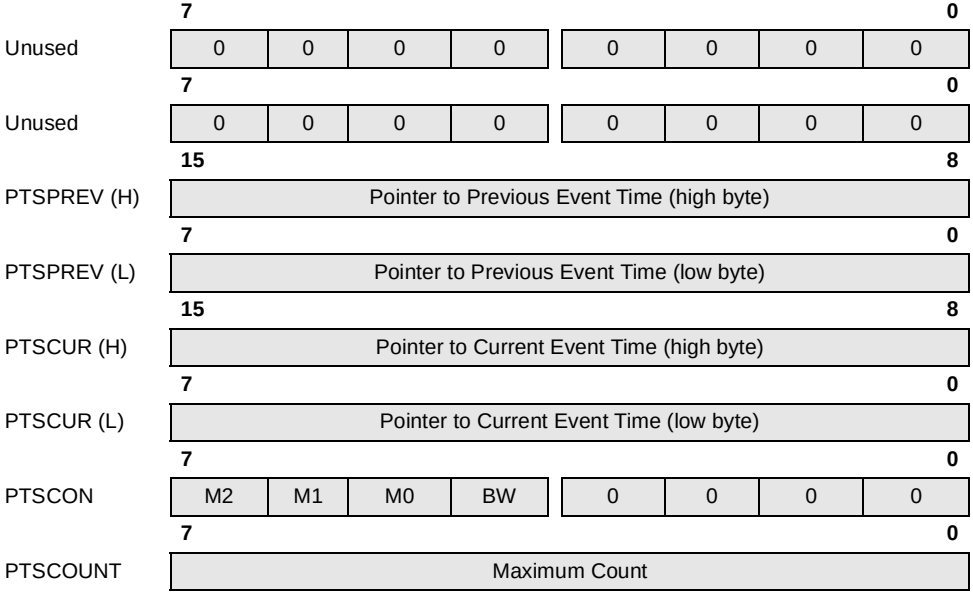


Figure 6-27. An Example of a Missed Event

PTS Missed-event Mode Control Block

In missed-event mode, the PTS control block contains a pointer to the current event-time (PTSCUR), a pointer to the previous event-time (PTSPREV), a control register (PTSCON), and a maximum count register (PTSCOUNT).



Register	Location	Function
PTSPREV	PTSCB + 4	Pointer to Previous Event Time This register points to the memory location where the PTS stores the time of the previous event. A valid address is any unreserved memory location within page 00H; however, it must point to an even address if word event times are selected.
PTSCUR	PTSCB + 2	Pointer to Current Event Time This register points to the event-time register (EPA*x*_TIME) for the EPA channel that is capturing the input pulse signal.

Figure 6-28. PTS Control Block — Missed-event Mode

PTS Missed-event Mode Control Block (Continued)

Register	Location	Function	
PTSCON	PTSCB + 1	PTS Control Bits	
		M2:0	PTS Mode These bits select the PTS mode: **M2 M1 M0** 1 1 0 missed-event mode
		BW	Byte/Word Event-time Value 0 = word 1 = byte
		4:0	To guarantee proper device operation, write zeros to these bits.
PTSCOUNT	PTSCB + 0	Maximum Count Defines the number of times the PTS will check for the occurrence of an event. Each comparison is one PTS cycle. Maximum number is 255.	

Figure 6-28. PTS Control Block — Missed-event Mode (Continued)

To use the missed-event mode, configure one event processor array (EPA) channel to capture events and a second EPA channel to monitor a timer/counter and compare its value with the expected next event time. For example, configure EPA1 as the input capture channel and EPA0 as the compare-only channel.

Configure EPA1 to select timer 1, capture on rising edge, select associated pin for input, reset timer 1, old data lost on overrun, no module concantenation (EPA1_CON = 0000 0010 X0X0 0010).

Configure EPA0 to select timer 1, compare mode, interrupt only, no action (EPA0_CON = 0000 0100 0000 0000).

Initialize the PTSCB as shown in Table 6-10.

Table 6-10. Missed-event Mode PTSCBs

0000H (Unused)
0000H (Unused)
PTSPREV (H) = *xx*H (any unreserved memory location within page 00H)
PTSPREV (L) = *xx*H (any unreserved memory location within page 00H)
PTSCUR (H) = 1FH (EPA1_TIME)
PTSCUR (L) = 5AH (EPA1_TIME)
PTSCON = C0H (missed-event mode)
PTSCOUNT = FFH (check for 255 events)

Whenever an event occurs on EPA1, the EPA loads the value of the reference timer into the EPA1_TIME register. The PTS uses two temporary registers to calculate the expected next event time. The following steps outline a missed-event PTS cycle.

1. EPA1 captures an event and the value of the reference timer is loaded into EPA1_TIME.
2. The PTS subtracts the time of the previous event, which is stored in a memory location pointed to by the PTSPREV register, from the current event-time stored in EPA1_TIME. It then loads the result into a temporary register.

   TEMP1 ← (EPA1_TIME) – (PTSPREV)

   For example, assume that the previous event occurred at timer value 04H and the current event occurred at timer value 08H. The PTS would load 04H into TEMP1.
3. To calculate the expected next event time, the PTS divides the contents of the temporary register (TEMP1) by two and loads the result into a second temporary register (TEMP2). It then adds the contents of TEMP1 to the contents of TEMP2 and loads the result into TEMP1. Then the PTS adds TEMP1 to the current event-time value to calculate the next event time. It stores this value in TEMP1.

   TEMP2 ← TEMP1/2

   TEMP1 ← TEMP1 + TEMP2

   TEMP1 ← TEMP1 + EPA1_TIME

   In the case of our example, TEMP1 would now contain 0EH.
4. The PTS loads the expected next event time into the location pointed to by EPA1_TIME + 4, which in this example, is the address of the EPA0_TIME register.

   (EPA0_TIME) ← TEMP1
5. The PTS then loads the current event-time value into the location pointed to by the PTSPREV register.

   (PTSPREV) ← (EPA1_TIME)
6. Finally, the PTS decrements the PTSCOUNT register.

   PTSCOUNT ← PTSCOUNT – 1

If a match occurs between the reference timer and the contents of EPA0_TIME, EPA0 causes an interrupt which indicates a missed-event. As long as an event occurs on EPA1 before a match occurs between the reference timer and the contents of EPA0_TIME, the PTS cycle repeats until PTSCOUNT decrements to zero. When PTSCOUNT reaches zero, the PTS generates an end-of-PTS interrupt.

7

I/O Ports

CHAPTER 7
I/O PORTS

The microcontroller contains one 5-bit I/O port, one 6-bit I/O port, and nine 8-bit I/O ports. Each port pin can function as a general-purpose I/O signal or as a special-function signal. General-purpose I/O signals provide a mechanism to transfer information between the microcontroller and the surrounding system circuitry. They can read system status, monitor system operation, output microcontroller status, configure system options, generate control signals, provide serial communication, and so on. Special-function signals are associated with on-chip peripherals or system functions.

7.1 I/O PORTS OVERVIEW

Most port pins can serve as low-speed input/output signals (I/O mode) or as signals for peripheral and/or system functions (special-function mode). Each port pin can function as a complementary or open-drain signal. For complementary signals, the microcontroller drives a one or a zero on the pin. For open-drain signals, the microcontroller either floats the pin, making it available as a high impedance input, or pulls the pin low. Each port contains dedicated special-function registers (SFRs) that allow you to select a pin's mode, configuration, and output value, and read a pin's input value.

The microcontroller contains two port types: standard and memory mapped. The SFRs for standard I/O ports are located in the special-function address space. Using windowing, these registers are accessible with direct addressing. (See "Windowing" on page 4-17.) The SFRs for memory-mapped I/O ports are located in the memory-mapped address space. Unlike standard I/O ports, memory-mapped I/O port registers are accessible only with indirect or indexed addressing; windowing is not available for memory-mapped I/O port registers.

For each port, Table 7-1 lists the number of pins, the port type, and the associated peripheral or system function.

Table 7-1. Microcontroller I/O Ports

Port	Pins	Type	Associated Peripheral or System Function
Extended Port (EPORT)	8	Memory mapped	Extended address lines, chip-select unit
Port 2	8	Standard	SIO, interrupts, bus control, clock generation
Port 3	8	Memory mapped	Address/data bus
Port 4	8	Memory mapped	Address/data bus
Port 5	8	Memory mapped	Bus control
Port 7	8	Standard	EPA, SIO, timers
Port 8	8	Standard	EPA
Port 9	8	Standard	EPA
Port 10	6	Standard	EPA, SIO
Port 11	8	Standard	PWM
Port 12	5	Memory mapped	—

For each port pin, Table 7-2 lists the I/O and special-function signal names, the special-function signal type, and the special-function signal's associated peripheral or system function. For a description of a pin's special-function signal, see "Using the Special-function Signals" on page 7-11.

Table 7-2. Microcontroller Port Signals

Port	I/O Signal	Special-function Signal	Special-function Signal Type	Special-function Signal Peripheral or System Function
Extended Port	EPORT.0	A16	O	Extended address bus
	EPORT.1	A17	O	Extended address bus
	EPORT.2	A18	O	Extended address bus
	EPORT.3	A19	O	Extended address bus
	EPORT.4	A20	O	Extended address bus
	EPORT.5	CS0#	O	Chip-select unit
	EPORT.6	CS1#	O	Chip-select unit
	EPORT.7	CS2#	O	Chip-select unit
Port 2	P2.0	TXD0	O	Serial I/O unit
	P2.1	RXD0	I/O	Serial I/O unit
	P2.2	EXTINT	I	Interrupts
	P2.3	TXD1	O	Bus controller
	P2.4	RXD1	/O	Interrupts
	P2.5	HOLD#	I	Bus controller
	P2.6	HLDA#/ONCE#	O	Bus controller
	P2.7	CLKOUT	O	Clock generator
Port 3	P3.7:0	AD7:0	I/O	Address/data bus, low byte
Port 4	P4.7:0	AD15:8	I/O	Address/data bus, high byte
Port 5	P5.0	ALE	O	Bus controller
	P5.1	INST	O	Bus controller
	P5.2	WR#/WRL#	O	Bus controller
	P5.3	RD#	O	Bus controller
	P5.4	BREQ#	O	Bus controller
	P5.5	BHE#/WRH#	O	Bus controller
	P5.6	READY	I	Bus controller
	P5.7	RPD	I	Powerdown
Port 7	P7.0	EPA0/T1CLK	I/O	EPA, timer 1
	P7.1	EPA1/T1RST	I/O	EPA, timer 1
	P7.2	EPA2/T2CLK	I/O	EPA, timer 2
	P7.3	EPA3/T2RST	I/O	EPA, timer 2
	P7.4	EPA4/T3CLK	I/O	EPA, timer 3
	P7.5	EPA5/T3RST	I/O	EPA, timer 3
	P7.6	EPA6/T4CLK	I/O	EPA, timer 4
	P7.7	EPA7/T4RST	I/O	EPA, timer 4

Table 7-2. Microcontroller Port Signals (Continued)

Port	I/O Signal	Special-function Signal	Special-function Signal Type	Special-function Signal Peripheral or System Function
Port 8	P8.0	EPA8	I/O	EPA
	P8.1	EPA9	I/O	EPA
	P8.2	EPA10	I/O	EPA
	P8.3	EPA11	I/O	EPA
	P8.4	EPA12	I/O	EPA
	P8.5	EPA13	I/O	EPA
	P8.6	EPA14	I/O	EPA
	P8.7	EPA15	I/O	EPA
Port 9	P9.0	OS0	O	EPA
	P9.1	OS1	O	EPA
	P9.2	OS2	O	EPA
	P9.3	OS3	O	EPA
	P9.4	OS4	O	EPA
	P9.5	OS5	O	EPA
	P9.6	OS6	O	EPA
	P9.7	OS7	O	EPA
Port 10	P10.0	SC0	I/O	SSIO
	P10.1	SD0	I/O	SSIO
	P10.2	SC1	I/O	SSIO
	P10.3	SD1	I/O	SSIO
	P10.4	EPA16	I/O	EPA
	P10.5	—	—	—
Port 11	P11.0	PWM0	O	PWM
	P11.1	PWM1	O	PWM
	P11.2	PWM2	O	PWM
	P11.3	PWM3	O	PWM
	P11.4	PWM4	O	PWM
	P11.5	PWM5	O	PWM
	P11.6	PWM6	O	PWM
	P11.7	PWM7	O	PWM
Port 12	P12.0	—	—	—
	P12.1	—	—	—
	P12.2	—	—	—
	P12.3	—	—	—
	P12.4	—	—	—

Table 7-3 lists the registers associated with the ports.

Table 7-3. Port Control and Status Registers

Mnemonic	Address	Description
EP_DIR P2_DIR P5_DIR P7_DIR P8_DIR P9_DIR P10_DIR P11_DIR P12_DIR	1FE3H 1FD2H 1FF3H 1FCAH 1FCBH 1FC2H 1FC3H 1FBAH 1FEAH	Port Direction Register Each bit controls the configuration of the corresponding pin. Clearing a bit configures a pin as a complementary output; setting a bit configures a pin as a high-impedance input or an open-drain output.
EP_MODE P2_MODE P5_MODE P7_MODE P8_MODE P9_MODE P10_MODE P11_MODE P12_MODE	1FE1H 1FD0H 1FF1H 1FC8H 1FC9H 1FC0H 1FC1H 1FB8H 1FE8H	Port Mode Register Each bit controls the mode of the corresponding pin. Setting a bit configures a pin as a special-function signal; clearing a bit configures a pin as a general-purpose I/O signal.
EP_PIN P2_PIN P3_PIN P4_PIN P5_PIN P7_PIN P8_PIN P9_PIN P10_PIN P11_PIN P12_PIN	1FE7H 1FD6H 1FFEH 1FFFH 1FF7H 1FCEH 1FCFH 1FC6H 1FC7H 1FBEH 1FEEH	Port Pin Register Each bit reflects the current state of the corresponding pin, regardless of the pin's mode and configuration.
P34_DRV	1FF4H	Ports 3/4 Driver Enable Register Bits 7 and 6 of the P34_DRV register control whether ports 3 and 4, respectively, are configured as complementary or open-drain. Setting a bit configures a port as complementary; clearing a bit configures a port as open-drain.

Table 7-3. Port Control and Status Registers (Continued)

Mnemonic	Address	Description
EP_REG P2_REG P5_REG P7_REG P8_REG P9_REG P10_REG P11_REG P12_REG	1FE5H 1FD4H 1FF5H 1FCCH 1FCDH 1FC4H 1FC5H 1FBCH 1FECH	Port Data Output Register **For I/O Mode (P*x*_MODE.*x* = 0)** When a port pin is configured as a complementary output (P*x*_DIR.*x* = 0), setting the corresponding port data bit drives a one on the pin, and clearing the corresponding port data bit drives a zero on the pin. When a port pin is configured as a high-impedance input or an open-drain output (P*x*_DIR.*x* = 1), clearing the corresponding port data bit drives a zero on the pin, and setting the corresponding port data bit floats the pin, making it available as a high-impedance input. **For Special-function Mode (P*x*_MODE.*x* = 1)** When a port pin is configured as an output (either complementary or open-drain), the corresponding port data bit value is immaterial because the corresponding on-chip peripheral or system function controls the pin. To configure a pin as a high-impedance input, set both the P*x*_DIR and P*x*_REG bits.
P3_REG P4_REG	1FFCH 1FFDH	Port *x* Data Output Register When the port 3 or 4 pins are configured as complementary signals, setting a P*x*_REG bit drives a one on the corresponding pin and clearing a P*x*_REG bit drives a zero on the corresponding pin. When the port 3 or 4 pins are configured as open-drain signals, setting a P*x*_REG bit floats the corresponding pin and clearing a P*x*_REG bit drives a zero on the corresponding pin.

The remainder of this chapter explains how to configure the ports, discusses using the port special-function signals, and describes the internal port structures.

7.2 CONFIGURING THE PORT PINS

Each port 2, 5, 7–12 and EPORT pin can be configured independently to operate as a special-function signal or an I/O signal. In addition, these signals can be independently configured to operate as complementary outputs, high-impedance inputs, or open-drain outputs. Unlike the other ports, the pins of ports 3 and 4 are not individually configurable. Ports 3 and 4 can be configured as complementary or open-drain ports. The remainder of this section details how to configure the port pins.

7.2.1 Configuring Ports 2, 5, 7–12, and EPORT

Using the port mode register, you can individually configure each pin for port 2, 5, 7–12, and EPORT to operate either as a general-purpose I/O signal (I/O mode) or as a special-function signal (special-function mode). In either mode, three configurations are possible: complementary output, high-impedance input, or open-drain output. The port direction and data output registers select the configuration for each pin. Complementary output means that the microcontroller drives the signal high or low. High-impedance input means that the microcontroller floats the signal. Open-drain output means that the microcontroller drives the signal low or floats it. For I/O mode, the port data output register determines whether the microcontroller drives the signal high, drives it low, or floats it. For special-function mode, the on-chip peripheral or system function determines whether the microcontroller drives the signal high or low for complementary outputs.

The pins for ports 2, 5, 7–12, and EPORT are weakly pulled high during and after reset. Initializing the pins by writing to the port mode register turns off the weak pull-ups. To ensure that the ports are initialized correctly, follow this suggested initialization sequence:

1. Write to P*x*_DIR (or EP_DIR) to configure the individual pins. Clearing a bit configures a pin as a complementary output. Setting a bit configures a pin as a high-impedance input or open-drain output.
2. Write to P*x*_MODE (or EP_MODE) to select either I/O or special-function mode. Writing to P*x*_MODE (regardless of the value written) turns off the weak pull-ups. Even if the entire port is to be used as I/O (its default configuration after reset), **you must write to P*x*_MODE (or EP_MODE) to ensure that the weak pull-ups are turned off**.

3. Write to P*x*_REG (or EP_REG).

   For complementary output configurations:
   In I/O mode, write the data that is to be driven by the pins to the corresponding P*x*_REG (or EP_REG) bits. In special-function mode, the value is immaterial because the on-chip peripheral or system function controls the pin. However, you must still write to P*x*_REG (or EP_REG) to initialize the pin.

   For high-impedance input or open-drain output configurations:
   In I/O mode, write to P*x*_REG (or EP_REG) to either float the pin, making it available as a high impedance input, or pull it low. Setting the corresponding P*x*_REG (or EP_REG) bit floats the pin; clearing the corresponding P*x*_REG (or EP_REG) bit pulls the pin low. In special-function mode, if the on-chip peripheral uses the pin as an input signal, you must set the corresponding P*x*_REG (or EP_REG) bit so that the pin can be driven externally. If the on-chip peripheral uses the pin as an output signal, the value of the corresponding P*x*_REG (or EP_REG) bit is immaterial because the on-chip peripheral or system function controls the pin. However, you must still write to P*x*_REG (or EP_REG) to initialize the pin.

Table 7-4 lists the control register values for each possible configuration.

Table 7-4. Control Register Values for Each Configuration

Desired Pin Configuration	Configuration Register Settings		
General-purpose I/O Signal	**P*x*_DIR**	**P*x*_MODE**	**P*x*_REG**
Complementary, driving 0	0	0	0
Complementary, driving 1	0	0	1
Open drain, strongly driving 0	1	0	0
Input (high impedance)	1	0	1
Special-function Signal	**P*x*_DIR**	**P*x*_MODE**	**P*x*_REG**
Complementary, output value controlled by peripheral	0	1	X
Open drain, output value controlled by peripheral	1	1	X
Input (high impedance)	1	1	1

7.2.2 Configuring Ports 3 and 4 (Address/Data Bus)

The microcontroller has two bus modes available: multiplexed and demultiplexed. For multiplexed mode, during external memory bus cycles, ports 3 and 4 serve as the lower 16 address bits and the data bus. For demultiplexed mode, during external memory bus cycles, ports 3 and 4 serve only as the data bus. When boot code resides in internal memory (EA# inactive), port 3 and 4 pins can function as general-purpose I/O pins when an external memory cycle is not required. When boot code resides in external memory (EA# active), port 3 and 4 pins should not be used for general-purpose I/O unless external logic can differentiate bus cycles from general-purpose I/O.

Ports 3 and 4 are eight-bit, memory-mapped I/O ports. These ports contain three control registers (P34_DRV, P3_REG, and P4_REG) and two status registers (P3_PIN and P4_PIN). The port registers are addressable only with indirect or indexed addressing — they cannot be windowed.

During external memory bus cycles, the processor takes control of ports 3 and 4 and automatically configures them as complementary outputs for driving address and/or data or as inputs for reading data. For general-purpose I/O, two configurations are possible: complementary and open drain. The P34_DRV register selects the configuration for each port. With a complementary configuration, the microcontroller drives the signal high or low. With an open-drain configuration, the microcontroller drives the signal low or floats it.

To configure a port for general-purpose I/O, first select a complementary or open-drain configuration by writing to P34_DRV. Set P34_DRV.7 to configure port 3 as complementary; clear P34_DRV.7 to configure port 3 as open-drain. Likewise, set P34_DRV.6 to configure port 4 as complementary; clear P34_DRV.6 to configure port 4 as open-drain. In complementary mode, write the output data to the corresponding P*x*_REG bit. In open-drain mode, set the corresponding P*x*_REG bit to float a pin or clear the corresponding bit to pull it low. When the device requires access to external memory, it takes control of the port and drives the address or data onto the pins. The address or data bits replace your output during this time. When the external access is completed, the device restores your data onto the pins.

To use the port pins as inputs, first clear the corresponding P34_DRV bit to configure the port as open-drain. Next, set the corresponding P*x*_REG bit to drive the pin to a high-impedance state. You may then read the pin's input value in the P*x*_PIN register. When the device requires access to external memory, it automatically takes control of the port; therefore, you must configure the input source to avoid contention on the bus.

7.2.3 Port Configuration Example

Assume that you wish to configure the pins of port 7 as shown in Table 7-5.

Table 7-5. Port 7 Configuration Example

Port Pin(s)	Configuration	Data
P7.0, P7.1	high-impedance input	high impedance
P7.2, P7.3	open-drain, driving 0	0
P7.4	open-drain, output with external pull-up	1 (because of external pull-up)
P7.5, P7.6	complementary, driving 0	0
P7.7	complementary, driving 1	1

To do so, you could use the following example code segment. shows the state of each pin after reset and after execution of each line of the example code.

```
LDB   P7_DIR,#00011111B
LDB   P7_MODE,#00000000B
LDB   P7_REG,#10010011B
```

Table 7-6. Port 7 Pin States After Reset and After Example Code Execution

Action or Code	Resulting Pin States†							
	P7.7	P7.6	P7.5	P7.4	P7.3	P7.2	P7.1	P7.0
Reset	WK	WK	WK	WK	WK	WK	WK	WK
LDB P7_DIR, #00011111B	1	1	1	WK	WK	WK	WK	WK
LDB P7_MODE, #00000000B	1	1	1	HZ	HZ	HZ	HZ	HZ
LDB P7_REG, #10010011B	1	0	0	1††	0	0	HZ	HZ

† WK = weakly pulled high, HZ = high impedance.

†† Pulled high by external pull-up.

7.3 USING THE SPECIAL-FUNCTION SIGNALS

Most port pins can function as either general-purpose I/O signals or as special-function signals. The following sections describe the special-function signals and outline special considerations for using these signals.

7.3.1 Address and Data Signals (Ports 3, 4, and EPORT)

During external memory bus cycles, the processor takes control of ports 3 and 4 and automatically configures them as complementary output ports for driving the lower 16 address signals and/or the data bus or as inputs for reading the data bus. When EA# is inactive, these port pins are available for general-purpose I/O when the microcontroller is not performing external bus cycles.

Some of the extended port pins can function as address signals or general-purpose I/O signals (Table 7-7). To use an extended port pin as an address signal, set the corresponding EP_MODE bit, selecting special-function mode. When an extended port pin is configured as an address signal, the microcontroller automatically configures the pin as a complementary output.

When EA# is inactive, ports 3 and 4 output the contents of the P3_REG and P4_REG registers. Because these registers reset to FFH and the P34_DRV register resets to 00H (open-drain mode), ports 3 and 4 will float unless you connect external pull-up resistors to the pins, write zeros to the P3_REG and P4_REG registers, or write ones to the P34_DRV register.

Table 7-7. Address and Data Signals

Address Signal	I/O Signal	Address Signal Description and Considerations
AD15:8 AD7:0	P4.7:0 P3.7:0	**Description**: Address/Data Lines. The function of these pins depend on the bus size and mode. When a bus access is not occurring, these pins revert to their I/O port function. 16-bit Multiplexed Bus Mode: AD15:0 drive address bits 0–15 during the first half of the bus cycle and drive or receive data during the second half of the bus cycle. 8-bit Multiplexed Bus Mode: AD15:8 drive address bits 8–15 during the entire bus cycle. AD7:0 drive address bits 0–7 during the first half of the bus cycle and drive or receive data during the second half of the bus cycle. 16-bit Demultiplexed Mode: AD15:0 drive or receive data during the entire bus cycle. 8-bit Demultiplexed Mode: AD7:0 drive or receive data during the entire bus cycle. AD15:8 drive the data that is currently on the high byte of the internal bus.

Table 7-7. Address and Data Signals (Continued)

Address Signal	I/O Signal	Address Signal Description and Considerations
A20:16	EPORT.4:0	**Description**: Address Lines 16–20. These address lines provide address bits 16–20 during the entire external memory cycle, supporting extended addressing of the 2-Mbyte address space. **Considerations**: During the CCB fetch, all EPORT pins are strongly driven high. Designers should ensure that this does not conflict with external systems that are outputting signals to the EPORT. When EPORT pins are floated during idle, powerdown, or hold, the external system must provide circuitry to prevent CMOS inputs on external devices from floating. During powerdown, the EPORT input buffers on pins configured for their extended-address function are disconnected from the pins, so a floating pin will not cause increased power consumption. Open-drain outputs require an external pull-up resistor. Inputs must be driven or pulled high or low; they must not be allowed to float.

7.3.1.1 EPORT Status During Reset, CCB Fetch, Idle, Powerdown, and Hold

During reset, the EPORT pins are forced to their extended-address functions and are weakly pulled high. During the CCB fetch, FFH is strongly driven onto the pins. This value remains strongly driven until either the pin is configured for I/O or a different extended address is accessed. If the pins remain configured as extended-address functions, they are placed in a high-impedance state during idle, powerdown, and hold. If they are configured as I/O, they retain their I/O function during those modes. See Figure 13-7 on page 13-9 and Table B-5 on page B-15 for additional information.

7.3.2 Bus-control Signals (Ports 2 and 5)

Some port 2 and 5 pins function as either general-purpose I/O signals or as bus-control signals (Table 7-8). To use a port 2 pin as a bus-control signal, set the corresponding P2_MODE bit, selecting special-function mode. To configure a port 2 pin as a complementary output signal, clear the corresponding P2_DIR bit. To configure a port 2 pin as an input signal, set the corresponding P2_DIR and P2_REG bits. To configure a port 2 pin as an open-drain output, set the corresponding P2_DIR bit. To use a port 5 pin as a bus-control signal, set the corresponding P5_MODE bit; this selects special-function mode. For port 5, when a pin is configured as a bus-control signal, the microcontroller automatically configures the pin as a complementary output or high-impedance input signal, depending on the bus-control signal function.

Table 7-8. Bus-control Signals

Bus-control Signal	I/O Signal	Bus-control Signal Description and Considerations
ALE	P5.0	**Description**: Address Latch Enable. This active-high output signal is asserted only during external memory cycles. ALE signals the start of an external bus cycle and indicates that valid address information is available on the system address/data bus. **Considerations**: When the boot code is located in external memory (EA# active), the microcontroller automatically configures P5.0 at reset as a special-function complementary output signal. When the boot code is located in internal memory (EA# inactive), P5.0 is weakly held high. Set P5_MODE.0 to turn off the internal pull-up and configure P5.0 as a special-function complementary output signal.
BHE#	P5.5	**Description**: Byte High Enable. During 16-bit bus cycles, this active-low output signal is asserted for word and high-byte reads and writes to external memory. BHE# indicates that valid data is being transferred over the upper half of the system data bus. **Considerations**: If EA# is high on reset (internal access), P5.5 is weakly held high until your software writes to P5_MODE.5. If EA# is low on reset (external access), P5.5 is weakly held high until the CCB0 fetch is completed. If CCR0.2 is clear, the microcontroller automatically configures this pin as WRH#. If CCR0.2 is set, the microcontroller automatically configures this pin as BHE#.
BREQ#	P5.4	**Description**: Bus Request. This active-low output signal is asserted during a hold cycle when the bus controller has a pending external memory cycle. **Considerations**: When the bus-hold protocol is enabled (WSR.7 is set), the P5.4/BREQ# pin can function only as BREQ#, regardless of the configuration selected through the port configuration registers (P5_MODE, P5_DIR, and P5_REG). An attempt to change the pin configuration is ignored until the bus-hold protocol is disabled (WSR.7 is cleared). The microcontroller can assert BREQ# at the same time as or after it asserts HLDA#. Once it is asserted, BREQ# remains asserted until HOLD# is deasserted.
CLKOUT	P2.7	**Description**: Clock Output. Output of the internal clock generator. The CLKOUT frequency can be programmed to one of five frequencies: the internal operating frequency (f) divided by a factor of two, four, eight, or sixteen, or the same frequency as the oscillator input (F_{XTAL1}). **Considerations**: Following reset, the microcontroller automatically configures P2.7 as CLKOUT. It is **not** held high. When P2.7 is configured as CLKOUT (P2_MODE.7 = 1), it is always a complementary output.

Table 7-8. Bus-control Signals (Continued)

Bus-control Signal	I/O Signal	Bus-control Signal Description and Considerations
HLDA#	P2.6	**Description**: Bus Hold Acknowledge. The HLDA# pin is used in systems with more than one processor using the system bus. The microcontroller asserts HLDA# to indicate that it has released the bus in response to HOLD# and another processor can take control. (This signal is active low to avoid misinterpretation by external hardware immediately after reset.) **Considerations**: When the bus-hold protocol is enabled (WSR.7 is set), the P2.6/HLDA# pin can function only as HLDA#, regardless of the configuration selected through the port configuration registers (P2_MODE, P2_DIR, and P2_REG). An attempt to change the pin configuration is ignored until the bus-hold protocol is disabled (WSR.7 is cleared).
HOLD#	P2.5	**Description**: Bus Hold Request. An external device uses this active-low input signal to request control of the bus. **Considerations**: When the bus-hold protocol is enabled (WSR.7 is set), the P2.5/HOLD# pin can function only as HOLD#, regardless of the configuration selected through the port configuration registers (P2_MODE, P2_DIR, and P2_REG). An attempt to change the pin configuration is ignored until the bus-hold protocol is disabled (WSR.7 is cleared). If P2.5 is configured as a general-purpose I/O signal, the device does not recognize signals on this pin as HOLD#. Instead, the bus controller receives an internal HOLD signal. This enables the device to access the external bus while it is performing I/O at P2.5.
INST	P5.1	**Description**: Instruction Fetch. When high, INST indicates that an instruction is being fetched from external memory. The signal remains high during the entire bus cycle of an external instruction fetch. INST is low for data accesses, including interrupt vector fetches and chip configuration byte reads. INST is low during internal memory fetches. **Considerations**: Following reset, pin P5.1 is weakly held high until your software writes configuration data into P5_MODE.
READY	P5.6	**Description**: Ready Input. This active-high input can be used to insert wait states in addition to those programmed in the chip configuration byte 0 (CCB0) and the bus control *x* register (BUSCON*x*). CCB0 is programmed with the minimum number of wait states (0–3) for an external fetch of CCB1 and BUSCON*x* is programmed with the minimum number of wait states (0–3) for all external accesses to the address range assigned to the chip-select *x* channel. If READY is low when the programmed number of wait states is reached, additional wait states are added until READY is pulled high. **Considerations**: Following reset, pin P5.6 is weakly held high until your software writes configuration data into P5_MODE.

Table 7-8. Bus-control Signals (Continued)

Bus-control Signal	I/O Signal	Bus-control Signal Description and Considerations
RD#	P5.3	**Description**: Read. Read-signal output to external memory. RD# is asserted only during external memory reads. **Considerations**: If EA# is high on reset (internal access), P5.3 is weakly held high until your software writes to P5_MODE. If EA# is low on reset (external access), the microcontroller automatically configures P5.3 as a special-function complementary output signal.
WR#	P5.2	**Description**: Write. This active-low output indicates that an external write is occurring. This signal is asserted only during external memory writes. **Considerations**: The chip configuration register 0 (CCR0) determines whether this pin functions as WR# or WRL#. CCR0.2 = 1 selects WR#; CCR0.2 = 0 selects WRL#. Following reset, pin P5.2 is weakly held high until your software writes configuration data into P5_MODE.
WRL#	P5.2	**Description**: Write Low. During 16-bit bus cycles, this active-low output signal is asserted for low-byte writes and word writes to external memory. During 8-bit bus cycles, WRL# is asserted for all write operations. **Considerations**: The chip configuration register 0 (CCR0) determines whether this pin functions as WR# or WRL#. CCR0.2 = 1 selects WR#; CCR0.2 = 0 selects WRL#. Following reset, pin P5.2 is weakly held high until your software writes configuration data into P5_MODE.
WRH#	P5.5	**Description**: Write High. During 16-bit bus cycles, this active-low output signal is asserted for high-byte writes and word writes to external memory. During 8-bit bus cycles, WRH# is asserted for all write operations. **Considerations**: If EA# is high on reset (internal access), P5.5 is weakly held high until your software writes to P5_MODE.5. If EA# is low on reset (external access), P5.5 is weakly held high until the CCB0 fetch is completed. If CCR0.2 is clear, the microcontroller automatically configures this pin as WRH#. If CCR0.2 is set, the microcontroller automatically configures this pin as BHE#.

7.3.3 Chip-select Signals (EPORT)

Some EPORT pins function as chip-select signals or general-purpose I/O (Table 7-9). To use an EPORT pin as a chip-select signal, set the corresponding EP_MODE bit, selecting special-function mode, and clear the corresponding EP_DIR bit, selecting a complementary output configuration.

Table 7-9. Chip-select Signals

Chip-select Signal	I/O Signal	Chip-select Signal Descriptions and Considerations
CS2:0#	EPORT.7:5	**Description**: Chip-select Lines 0–2. The active-low output CS*x*# is asserted during an external memory cycle when the address to be accessed is in the range programmed for chip select *x*. **Considerations**: Pins EPORT.7:5 are weakly pulled high during reset. After reset, EPORT.5 defaults to the CS0# function. This chip-select signal detects address ranges that contain the CCBs and FF2080H (program start-up address). The bus configuration programmed in BUSCON*x* applies to address range *x*, regardless of the port configuration.

7.3.4 EPA and Timer Signals (Ports 7–10)

The ports 7–10 pins can function as EPA and timer signals or general-purpose I/O signals (). To use the ports 7–10 pins as EPA and timer signals, set the corresponding P*x*_MODE bits, selecting special-function mode. To configure an EPA or timer signal as a complementary output, clear the corresponding P*x*_DIR bit. To configure an EPA or timer signal as an input, set the corresponding P*x*_DIR and P*x*_REG bits. To configure an EPA or timer signal as an open-drain output, set the corresponding P*x*_DIR bit.

Table 7-10. EPA and Timer Signals

EPA or Timer Signal	I/O Signal	EPA, Enhanced EPA, or Timer Signal Descriptions and Considerations
EPA16 EPA15:8 EPA7:0	P10.4 P8.7:0 P7.7:0	**Description**: Event Processor Array (EPA) Capture/Compare Channels. High-speed input/output signals for the EPA capture/compare channels. **Considerations**: Following reset, these pins are weakly pulled high until your software writes configuration data into P*x*_MODE.
OS7:0	P9.7:0	**Description**: Event Processor Array (EPA) Compare-only Channels with Simulcapture. Outputs of the EPA's compare-only channels. These pins are multiplexed with port 9 and may be configured as standard I/O. **Considerations**: Following reset, pins P9.7:0 are weakly pulled high until your software writes configuration data into P9_MODE.
T1CLK T2CLK T3CLK T4CLK	P7.0 P7.2 P7.4 P7.6	**Description**: Timer *x* External Clock. External clock for timer *x*. Timer *x* is programmable to increment or decrement on the rising edge, the falling edge, or both rising and falling edges of T*x*CLK. **Considerations**: Following reset, pins P7.0, P7.2, P7.4, and P7.6 are weakly pulled high until your software writes configuration data into P7_MODE.
T1RST T2RST T3RST T4RST	P7.1 P7.3 P7.5 P7.6	**Description**: Timer *x* External Reset. External reset for timer *x*. Timer *x* is programmable to reset on the rising edge, the falling edge, or both rising and falling edges of T*x*RST. **Considerations**: Following reset, pins P7.1, P7.3, P7.5, and P7.6 are weakly pulled high until your software writes configuration data into P7_MODE.

7.3.5 External Interrupt Signal (Port 2)

Port 2, pin 2 can function as the external interrupt (EXTINT) signal or as a general-purpose I/O signal (). To configure port 2, pin 2 as the external interrupt, set P2_DIR.2, P2_MODE.2, and P2_REG.2. Setting the P2_MODE.2 bit could cause the device to set the corresponding interrupt pending bit, indicating an interrupt request; therefore, follow this sequence to prevent a false interrupt request:

1. Disable interrupts by executing the DI instruction.
2. Set P2_DIR.2.
3. Set P2_MODE.2.
4. Set P2_REG.2.
5. Clear the external interrupt pending bit (INT_PEND1.6).
6. Enable interrupts (optional) by executing the EI instruction.

Table 7-11. External Interrupt Signal

External Interrupt Signal	I/O Signal	External Interrupt Signal Description and Considerations
EXTINT	P2.2	**Description**: External Interrupt. In normal operating mode, a rising edge on EXTINT sets the EXTINT interrupt pending bit. EXTINT is sampled during phase 2 (CLKOUT high). The minimum high time is one state time. In powerdown mode, asserting the EXTINT signal for at least 50 ns causes the device to resume normal operation. The interrupt does not need to be enabled, but the pin must be configured as a special-function input. If the EXTINT interrupt is enabled, the CPU executes the interrupt service routine. Otherwise, the CPU executes the instruction that immediately follows the command that invoked the power-saving mode. In idle mode, asserting any enabled interrupt causes the device to resume normal operation. **Considerations**: Setting P2_MODE.2 could cause the microcontroller to set the external interrupt pending bit; therefore, to prevent a false interrupt request, clear the interrupt pending bit before globally enabling interrupts.

7.3.6 PWM Signals (Port 11)

The port 11 pins can function as PWM signals or general-purpose I/O signals (Table 7-12). To use a port 11 pin as a PWM signal, set the corresponding P11_MODE bit, selecting special-function mode, and clear the corresponding P11_DIR bit, configuring the pin as a complementary output.

Table 7-12. PWM Signals

PWM Signal	I/O Signal	PWM Signal Description and Considerations
PWM7:0	P11.7:0	**Description**: Pulse Width Modulator Outputs. These are PWM output pins with high-current drive capability. **Considerations**: Following reset, pins P11.7:0 are weakly pulled high until your software writes configuration data into P11_MODE.

7.3.7 Serial I/O Port Signals (Ports 2 and 7)

Some port 2 and 7 pins can function as SIO signals or general-purpose I/O signals (). To use a port 2 or 7 pin as an SIO signal, set the corresponding P*x*_MODE bit, selecting special-function mode. To configure an SIO signal as a complementary output, clear the corresponding P*x*_DIR bit. To configure an SIO signal as an input, set the corresponding P*x*_DIR and P*x*_REG bits. To configure an SIO signal as an open-drain output, set the corresponding P*x*_DIR bit.

Table 7-13. SIO Signals

SIO Signal	I/O Signal	SIO Signal Description and Considerations
RXD0 RXD1	P2.1 P2.4	**Description**: Receive Serial Data 0 and 1. In modes 1, 2, and 3, RXD0 and 1 receive serial port input data. In mode 0, they function as either inputs or open-drain outputs for data. **Considerations**: Following reset, pins P2.1 and P2.4 are weakly pulled high until your software writes configuration data into P2_MODE.
T2CLK	P7.2	**Description**: Timer 2 External Clock. External clock for the serial I/O baud-rate generator input (program selectable). **Considerations**: Following reset, pin P7.2 is weakly pulled high until your software writes configuration data into P7_MODE.
TXD0 TXD1	P2.0 P2.3	**Description**: Transmit Serial Data 0 and 1. In serial I/O modes 1, 2, and 3, TXD0 and 1 transmit serial port output data. In mode 0, they are the serial clock output. **Considerations**: Following reset, pins P2.0 and P2.3 are weakly pulled high until your software writes configuration data into P2_MODE.

7.3.8 Special Operating Mode Signal (Port 5 Pin 7)

Port 5, pin 7 can function as either the return-from-powerdown signal or as a general-purpose I/O signal (). To use port 5, pin 7 as the return-from-powerdown signal, set P5_MODE.7. When port 5, pin 7 is configured as its special-function signal RPD, the microcontroller automatically configures the pin as an input signal. (It is not necessary to program P5_DIR and P5_REG.)

Table 7-14. Special Operating Mode Signal

Special Operating Mode Signal	I/O Signal	Description
RPD	P5.7	Return from Powerdown. Timing pin for the return-from-powerdown circuit.

7.3.9 Synchronous Serial I/O Port Signals (Port 10)

Some port 10 pins can function as SSIO signals or general-purpose I/O signals (Table 7-15). To use a port 10 pin as an SSIO signal, set the corresponding P10_MODE bit, selecting special-function mode. To configure an SSIO signal as a complementary output, clear the corresponding P10_DIR bit. To configure an SSIO signal as an input, set the corresponding P10_DIR and P10_REG bits. To configure an SSIO signal as an open-drain output, set the corresponding P10_DIR bit.

Table 7-15. SSIO Signals

SSIO Signal	I/O Signal	SSIO Signal Description and Considerations
CHS#	P9.2	**Description**: Channel Select. This signal is available only when the SSIO is configured for channel-select operation. The function of the signal depends on whether the SSIO is configured as master or slave. When the SSIO is configured as a slave, an external master activates CHS# to communicate with the SSIO. When the SSIO is configured as a master, an external master activates CHS# when it wants the SSIO to give up the bus.
SC0 SC1	P10.0 P10.2	**Description**: Clock Pins for SSIO0 and 1. These pins carry a signal only during receptions and transmissions. When the SSIO port is idle, the pin remains either high (with handshaking) or low (without handshaking). **Considerations**: Following reset, pins P10.0 and P10.2 are weakly held high until your software writes configuration data into P10_MODE. For handshaking mode, configure SC1:0 as open-drain outputs.
SD0 SD1	P10.1 P10.3	**Description**: Data Pins for SSIO0 and 1. These pins are the data I/O pins for SSIO0 and 1. **Considerations**: Following reset, pins P10.1 and P10.3 are weakly held high until your software writes configuration data into P10_MODE.

7.4 I/O PORT INTERNAL STRUCTURES

The following sections describe the internal structure of the ports.

7.4.1 Internal Structure for the Extended I/O Port (EPORT Pins 0–4)

Figure 7-1 shows the internal structure for the EPORT pins 0–4. Consult the datasheet for specifications on the amount of current that the EPORT pins 0–4 can source and sink.

During reset, the falling edge of RESET# generates a short pulse that turns on the medium pull-up transistor Q3, which remains on for about 300 ns, causing the pin to change rapidly to its reset state. The active-low level of RESET# turns on transistor Q4, which weakly holds the pin high. When RESET# is inactive, both Q3 and Q4 are off; Q1 and Q2 determine output drive.

If RESET#, HOLD#, idle, or powerdown is asserted, the gates that control Q1 and Q2 are disabled and Q1 and Q2 remain off. Otherwise, the gates are enabled and complementary or open-drain operation is possible.

For complementary output mode, the gates that control Q1 and Q2 must be enabled. The Q2 gate is always enabled (except when RESET#, HOLD#, idle, or powerdown is asserted). Either clearing EP_DIR (selecting complementary mode) **or** setting EP_MODE (selecting address mode) enables the logic gate preceding Q1. The value of DATA determines which transistor is turned on. If DATA is equal to one, Q1 is turned on and the pin is pulled high. If DATA is equal to zero, Q2 is turned on and the pin is pulled low.

For open-drain output mode, the gate that controls Q1 must be disabled. Setting EP_DIR (selecting open-drain mode) **and** clearing EP_MODE (selecting I/O mode) disables the logic gate preceding Q1. The value of DATA determines whether Q2 is turned on. If DATA is equal to one, both Q1 and Q2 remain off and the pin is left in high-impedance state (floating). If DATA is equal to zero, Q2 is turned on and the pin is pulled low.

Input mode is obtained by configuring the pin as an open-drain output (EP_DIR set and EP_MODE clear) and writing a one to EP_REG.*x*. In this configuration, Q1 and Q2 are both off, allowing an external device to drive the pin. To determine the value of the I/O pin, read EP_PIN.*x*.

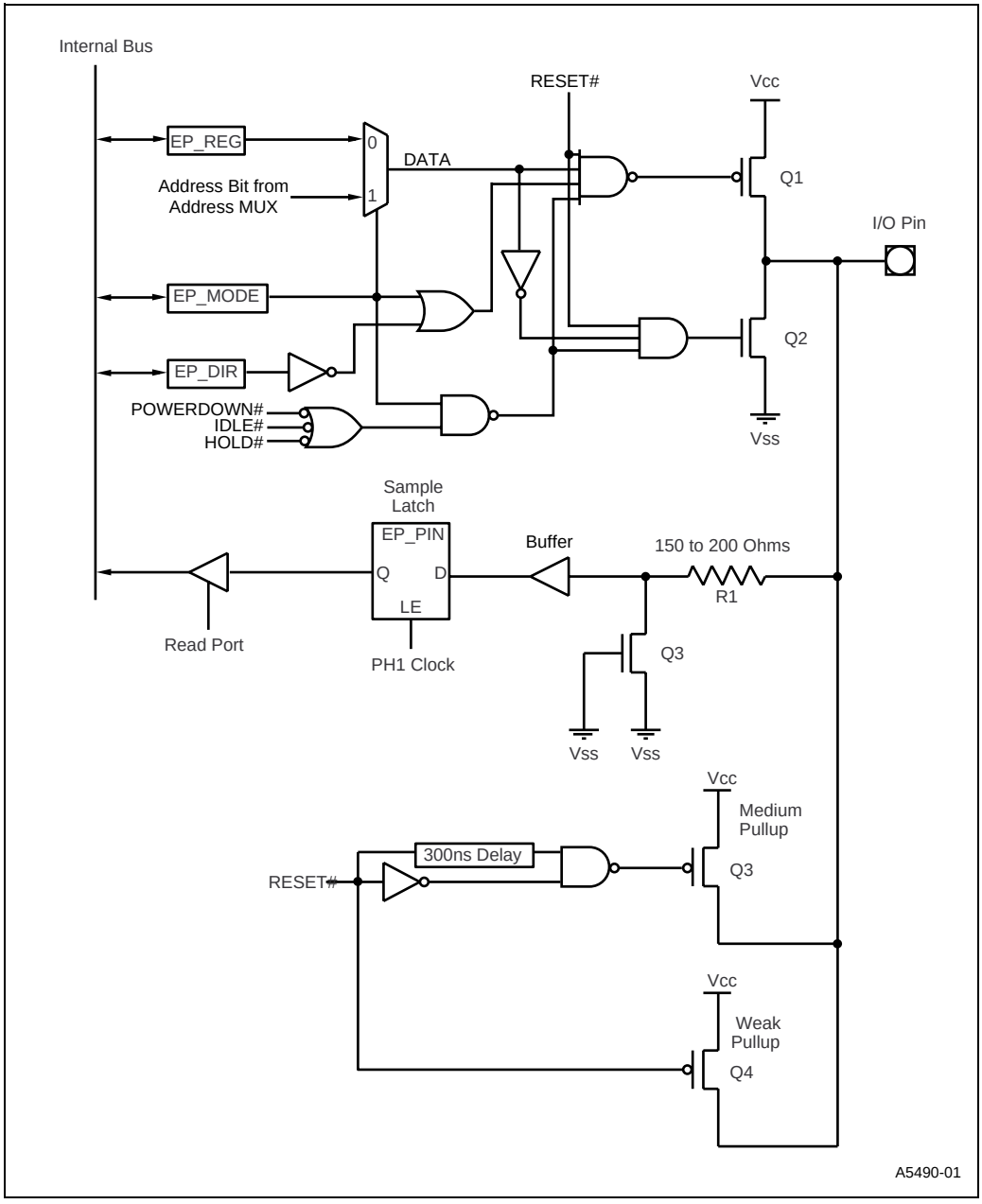


Figure 7-1. EPORT Pins 0–4 Internal Structure

7.4.2 Internal Structure for Ports 2, 5, 7–12, and EPORT Pins 5–7

Figure 7-2 shows the logic for driving the output transistors, Q1 and Q2. Consult the datasheet for specifications on the amount of current that each port can source or sink.

In I/O mode (selected by clearing a port mode register bit), the port data output and the port direction registers are input to the multiplexers. These signals combine to drive the gates of Q1 and Q2 so that the output is high, low, or high impedance.

In special-function mode (selected by setting a port mode register bit), SFDIR and SFDATA are input to the multiplexers. These signals combine to drive the gates of Q1 and Q2 so that the output is high, low, or high impedance. Special-function output signals clear SFDIR; special-function input signals set SFDIR. Even if a pin is to be used in special-function mode, you must still initialize the pin as an input or output by writing to the port direction register.

Resistor R1 provides ESD protection for the pin. Input signals are buffered. The standard ports use Schmitt-triggered buffers for improved noise immunity. Port 5 uses a standard input buffer because of the high speeds required for bus control functions. The signals are latched into the port pin register sample latch and output onto the internal bus when the port pin register is read.

The falling edge of RESET# turns on transistor Q3, which remains on for about 300 ns, causing the pin to change rapidly to its reset state. The active-low level of RESET# turns on transistor Q4, which weakly holds the pin high. Q4 remains on, weakly holding the pin high, until your software writes to the port mode register.

NOTE

P2.7 is an exception. After reset, P2.7 carries the CLKOUT signal rather than being held high. When CLKOUT is selected, it is always a complementary output.

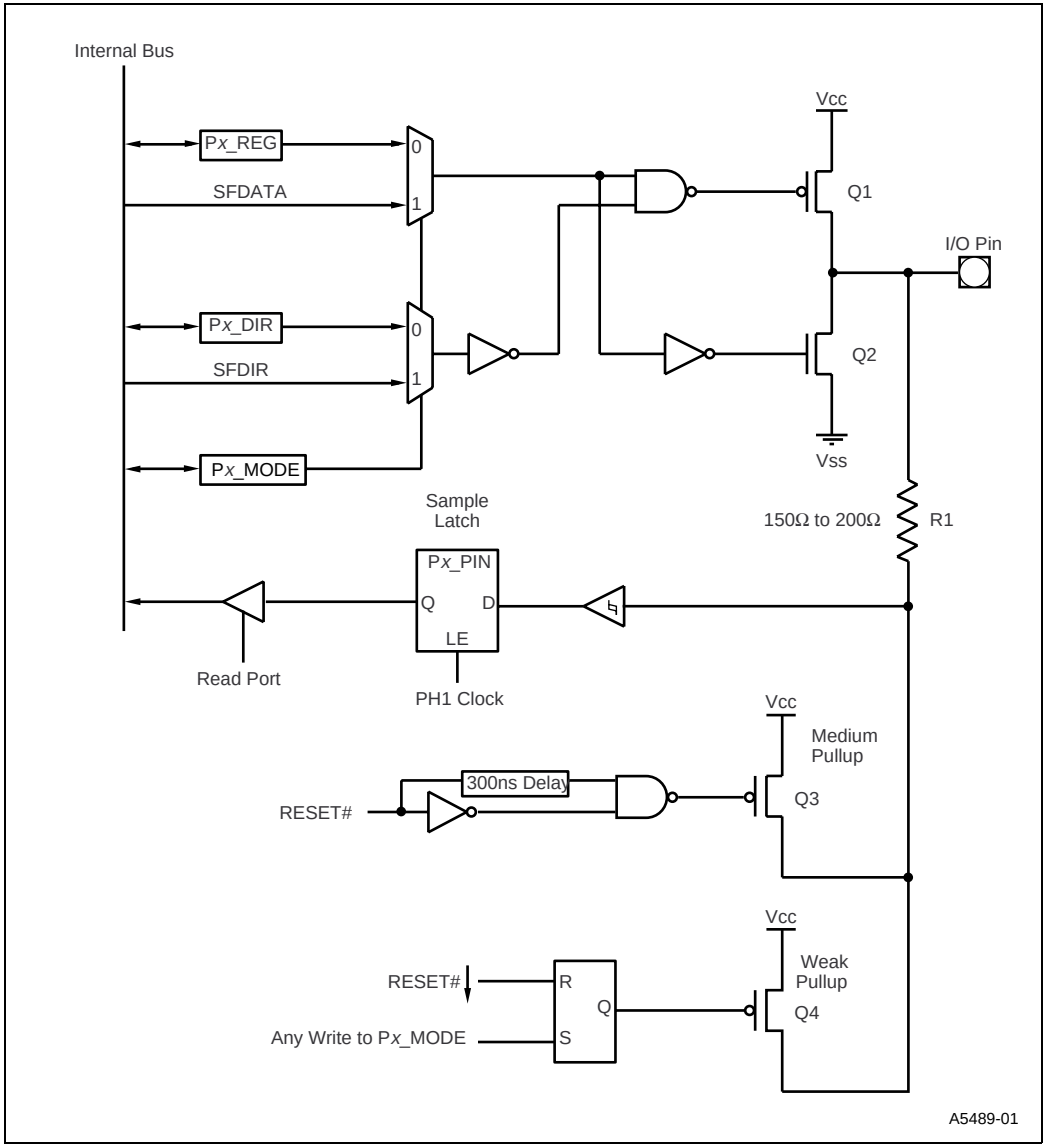


Figure 7-2. Ports 2, 5, 7–12, and EPORT Pins 5–7 Internal Structure

7.4.3 Internal Structure for Ports 3 and 4 (Address/Data Bus)

Figure 7-3 shows the logic of ports 3 and 4. Consult the datasheet for specifications on the amount of current ports 3 and 4 can source and sink.

During reset, the active-low level of RESET# turns off Q1 and Q2 and turns on transistor Q4, which weakly holds the pin high. Resistor R1 provides ESD protection for the pin. During normal operation, the device controls the port through BUS CONTROL SELECT, an internal control signal.

When the device needs to access external memory, it clears BUS CONTROL SELECT, selecting ADDRESS/DATA as the input to the multiplexer. ADDRESS/DATA then drives Q1 and Q2 as complementary outputs.

When external memory access is **not** required, the device sets BUS CONTROL SELECT, selecting P*x*_REG as the input to the multiplexer. P*x*_REG then drives Q1 and Q2. If P34_DRV is set, Q1 and Q2 are driven as complementary outputs. If P34_DRV is cleared, Q1 is disabled and Q2 is driven as an open-drain output requiring an external pull-up resistor. With the open-drain configuration (BUS CONTROL SELECT set and P34_DRV cleared) and P*x*_REG set, the pin can be used as an input. The signal on the pin is latched in the P*x*_PIN register. The pins can be read, making it easy to see which pins are driven low by the device and which are driven high by external drivers while in open-drain mode.

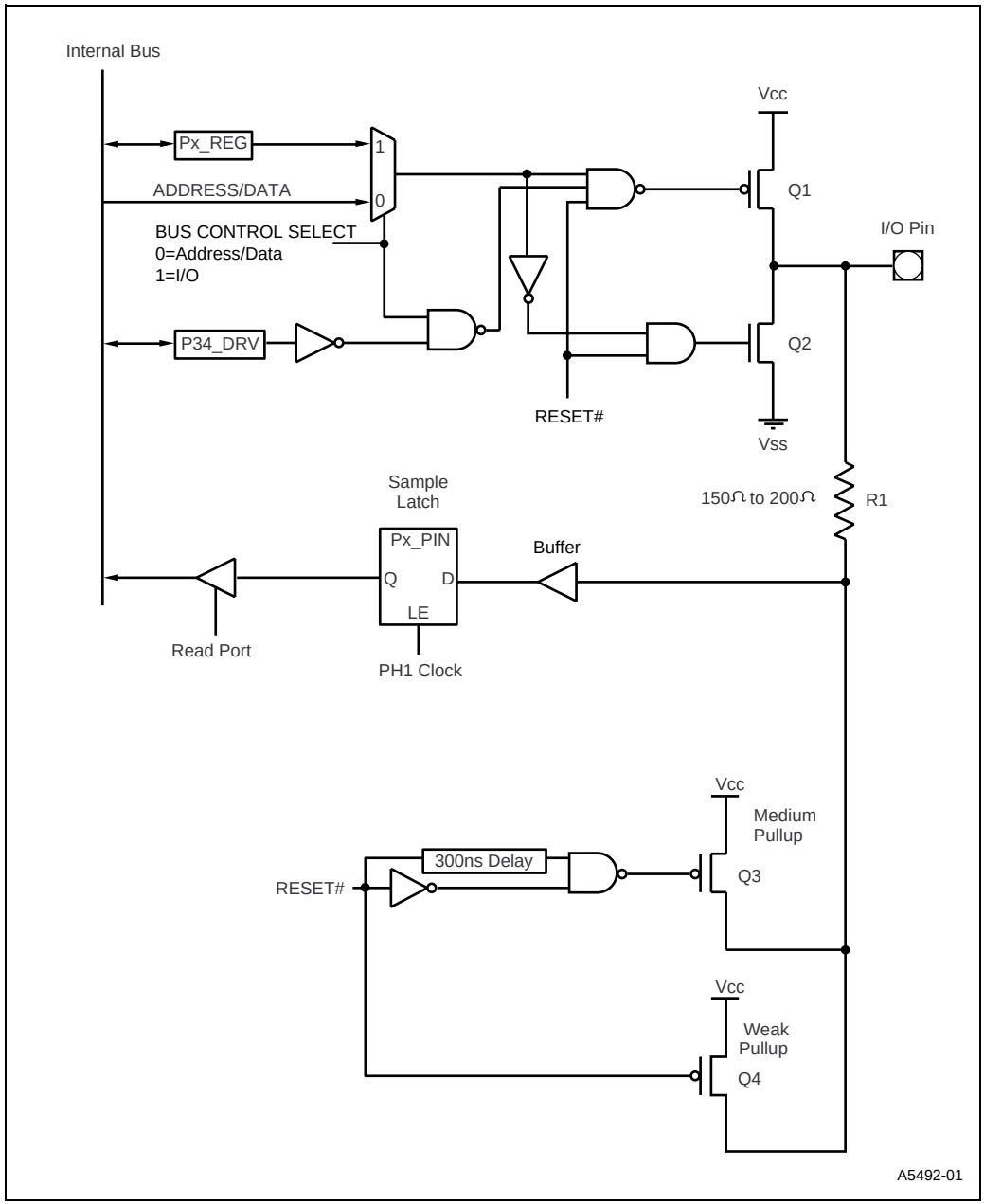


Figure 7-3. Ports 3 and 4 Internal Structure

8

Serial I/O (SIO) Port

CHAPTER 8
SERIAL I/O (SIO) PORT

A serial input/output (SIO) port provides a means for the system to communicate with external devices. This microcontroller has a two-channel serial I/O (SIO) port that shares pins with port 2. This chapter describes the SIO port and explains how to configure it.

8.1 SERIAL I/O (SIO) PORT FUNCTIONAL OVERVIEW

The serial I/O port is an asynchronous/synchronous port that has two identical channels. Each channel includes a universal asynchronous receiver and transmitter (UART) and has four modes of operation: one synchronous mode (mode 0) and three asynchronous modes (modes 1, 2, and 3). Each channel consists of a dedicated receiver, transmitter, control logic, two interrupt signals, and a baud-rate generator.

The transmitter and receiver contain buffers and shift registers. The buffers are accessible as special-function registers (SFRs). Write transmit data to the transmit buffer (SBUF*x*_TX) and read received data from the receive buffer (SBUF*x*_RX). Unlike the buffers, the shift registers are internal registers and are not accessible as SFRs. For receptions, data is shifted into the receive shift register, least-significant bit first, via the receive data pin (RXD*x*). After the last bit (eighth bit for mode 0 or stop bit for modes 1, 2, and 3) is shifted in, the receiver transfers the data from the receive shift register to SBUF*x*_RX where it can be accessed. For transmissions, data in SBUF*x*_TX is transferred to the transmit shift register then shifted out through the serial transmit pin (RXD*x* for mode 0 or TXD*x* for modes 1, 2, and 3).

Each channel contains a serial port control (SP*x*_CON) register and a serial port status (SP*x*_STATUS) register. SP*x*_CON configures the SIO channel for one of the operating modes and for receptions or transmissions. SP*x*_STATUS contains status and error flags. These registers are discussed in detail in “Programming the Control Register” on page 8-11 and “Determining Serial Port Status” on page 8-16.

Each SIO channel has two interrupt signals, allowing for interrupt-driven transmit and receive service routines. The receive interrupt (RI*x*) indicates the receive buffer (SBUF*x*_RX) contains received data, available for reading. The transmit interrupt (TI*x*) indicates the transmit buffer (SBUF*x*_TX) is empty and available for writing.

Each channel contains a 15-bit baud-rate generator. Either the internal peripheral clock or a signal input on the T2CLK pin can provide the clock signal. The baud-rate register (SP*x*_BAUD) selects the clock source and the baud rate. For synchronous mode 0, the baud-rate generator controls the baud rate output on the serial clock pin (TXD*x*). For asynchronous modes 1, 2, and 3, the baud-rate generator controls the transmit and receive shift clocks.

The SIO channel signals, registers, and interrupts are shown in Figures 8-1 and 8-2. The signals and registers are described in the following section.

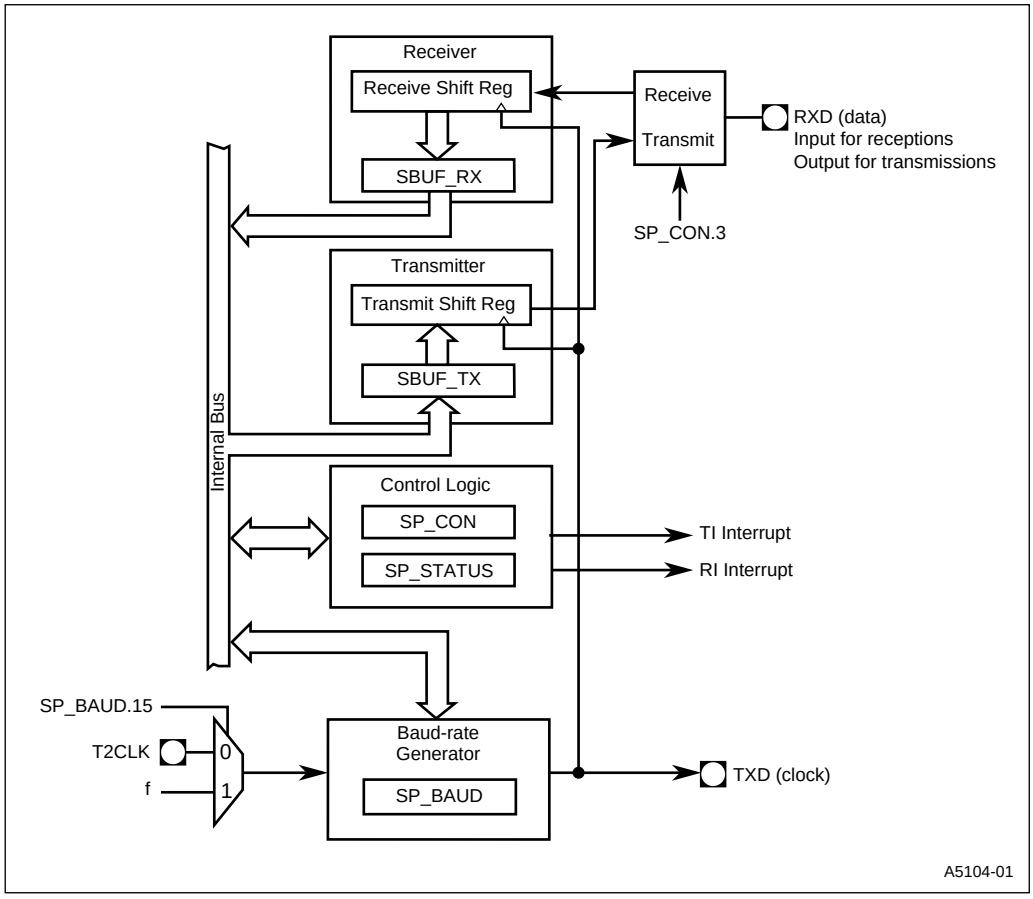


Figure 8-1. SIO Block Diagram (Mode 0)

As shown in Figure 8-1, the RXD*x* pin is the data pin and the TXD*x* pin is the clock pin for synchronous mode 0 operation. In this mode, the baud-rate generator drives eight pulses out the TXD*x* pin and the UART shifts data, least-significant bit first, into or out of the microcontroller via the RXD*x* pin. The UART samples data when the TXD*x* pulse is low. "Synchronous Mode (Mode 0)" on page 8-6 describes mode 0 in detail.

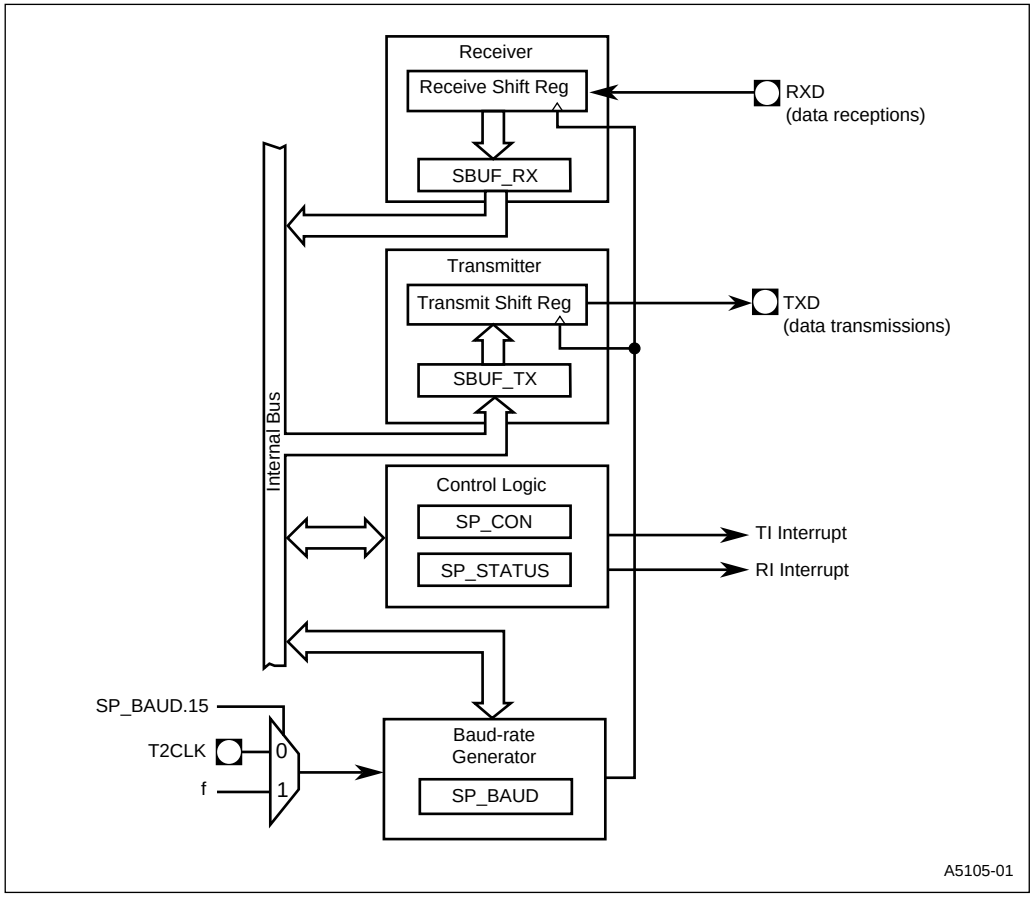


Figure 8-2. SIO Block Diagram (Modes 1, 2, and 3)

As shown in Figure 8-2, the RXD*x* pin is the receive data pin and the TXD*x* pin is the transmit data pin for asynchronous modes 1, 2, and 3. Either the internal operating frequency (f) or an input signal on the T2CLK pin provides the clock input to the baud-rate generator. “Asynchronous Modes (Modes 1, 2, and 3)” on page 8-7 describes modes 1, 2, and 3 in detail.

8.2 SERIAL I/O PORT SIGNALS AND REGISTERS

Table 8-1 describes the SIO signals and Table 8-2 describes the control and status registers.

Table 8-1. Serial Port Signals

Serial Port Signal	Serial Port Signal Type	Description
RXD	I/O	Receive Serial Data 0 and 1 In modes 1, 2, and 3, RXD0 and 1 receive serial port input data. In mode 0, they function as either inputs or open-drain outputs for data. RXD0 shares a package pin with P2.1, and RXD1 shares a package pin with P2.4.
T2CLK	I	Timer 2 Clock The internal operating freakiness (f) or an input signal on T2CLK provides the clock source for the baud-rate generator. Clearing SP*x*_BAUD.15 selects T2CLK as the clock source. T2CLK shares a package pin with P7.2.
TXD	O	Transmit Serial Data 0 and 1 In serial I/O modes 1, 2, and 3, TXD0 and 1 transmit serial port output data. In mode 0, they are the serial clock output. TXD0 shares a package pin with P2.0 and TXD1 shares a package pin with P2.3.

Table 8-2. Serial Port Control and Status Registers

Mnemonic	Address	Description
INT_MASK	0008H	Interrupt Mask Setting the TI*x* bit enables the transmit interrupt; clearing the bit disables (masks) the interrupt. Setting the RI*x* bit enables the receive interrupt; clearing the bit disables (masks) the interrupt.
INT_PEND	0009H	Interrupt Pending When set, the TI*x* bit indicates a pending transmit interrupt. When set, the RI*x* bit indicates a pending receive interrupt.
P2_DIR P7_DIR	1FD2H 1FCAH	Port Direction Register Each bit controls the configuration of the corresponding pin. Clearing a bit configures the corresponding pin as a complementary output; setting a bit configures the corresponding pin as an open-drain output or a high-impedance input. Write to P2_DIR.1, P2_DIR.4, P7_DIR.2, P2_DIR.0 and P7_DIR.2 to configure RXD0, RXD1, T2CLK, TXD0, and TXD1. (See "Configuring the Serial Port Pins" on page 8-10.)

Table 8-2. Serial Port Control and Status Registers (Continued)

Mnemonic	Address	Description
P2_MODE P7_MODE	1FD0H 1FC8H	Port Mode Register Each bit controls the mode of the corresponding pin. Setting a bit configures a pin as a special-function signal; clearing a bit configures a pin as a general-purpose I/O signal. Set P2_MODE.1, P2_MODE.4, P7_MODE.2, P2_MODE.0 and P7_MODE.2 to configure pins P2.1, P2.4, P7.2, and P2.0 as RXD0, RXD1, T2CLK, TXD0, and TXD1. (See "Configuring the Serial Port Pins" on page 8-10.)
P2_PIN P7_PIN	1FD6H 1FCEH	Port Pin Register Each bit reflects the current state of the corresponding pin, regardless of the pin's mode and configuration.
P2_REG P7_REG	1FD4H 1FCCH	Port Data Output Register **For I/O Mode (P*x*_MODE.*x* = 0)** When a port pin is configured as a complementary output (P*x*_DIR.*x* = 0), setting the corresponding port data bit drives a one on the pin, and clearing the corresponding port data bit drives a zero on the pin. When a port pin is configured as a high-impedance input or an open-drain output (P*x*_DIR.*x* = 1), clearing the corresponding port data bit drives a zero on the pin, and setting the corresponding port data bit floats the pin, making it available as a high-impedance input. **For Special-function Mode (P*x*_MODE.*x* = 1)** When a port pin is configured as an output (either complementary or open-drain), the corresponding port data bit value is immaterial because the corresponding on-chip peripheral or system function controls the pin. To configure a pin as a high-impedance input, set both the P*x*_DIR and P*x*_REG bits. Write to P2_REG.1, P2_REG.4, P7_REG.2, P2_REG.0 and P7_REG.2 to configure RXD0, RXD1, T2CLK, TXD0, and TXD1. (See "Configuring the Serial Port Pins" on page 8-10.)
SBUF0_RX SBUF1_RX	1F88H 1F98H	Serial Port *x* Receive Buffer This register contains data received from the RXD*x* pin.
SBUF0_TX SBUF1_TX	1F8AH 1F9AH	Serial Port *x* Transmit Buffer This register contains data that is ready for transmission. In modes 1, 2, and 3, writing to SBUF*x*_TX starts a transmission. In mode 0, writing to SBUF*x*_TX starts a transmission only if the receiver is disabled (SP*x*_CON.3 = 0).
SP0_BAUD SP1_BAUD	1F8CH, 1F8DH 1F9CH, 1F8DH	Serial Port *x* Baud Rate This register selects the serial port baud rate and clock source. The most-significant bit selects the clock source. The lower 15 bits represent the baud value, an unsigned integer that determines the baud rate.

Table 8-2. Serial Port Control and Status Registers (Continued)

Mnemonic	Address	Description
SP0_CON SP1_CON	1F8BH 1F9BH	Serial Port *x* Control This register selects the serial mode and enables or disables the receiver for all modes. For modes 1 and 3, it enables parity. For mode 2, and for mode 3 with parity disabled, it contains the ninth bit to be transmitted.
SP0_STATUS SP1_STATUS	1F89H 1F99H	Serial Port *x* Status This register contains the serial port status bits. It has status bits for receive overrun error (OE), transmit buffer empty (TXE), framing error (FE), transmit interrupt (TI), receive interrupt (RI), and received parity error (RPE) or received bit 8 (RB8). Reading SP*x*_STATUS clears all bits except TXE; writing a byte to SBUF*x*_TX clears the TXE bit.

8.3 SERIAL PORT MODES

This section describes the serial port operating modes. Mode 0 is a synchronous mode. Mode 1 is an eight-bit asynchronous mode with optional parity. Modes 2 and 3 are nine-bit asynchronous modes. Like mode 1, mode 3 has optional parity. For mode 2, the SIO flags receptions (by setting the RI status bit and RI*x* pending bit) only when the ninth data bit received is a one. This is useful for multiprocessor communication, which is described in detail in "Multiprocessor Communications" on page 8-10.

8.3.1 Synchronous Mode (Mode 0)

In mode 0, the TXD*x* pin outputs a set of eight clock pulses, while the RXD*x* pin either transmits or receives data. Data is transferred eight bits at a time, with the least-significant bit first. Figure 8-3 shows a diagram of the relative timing of these signals.

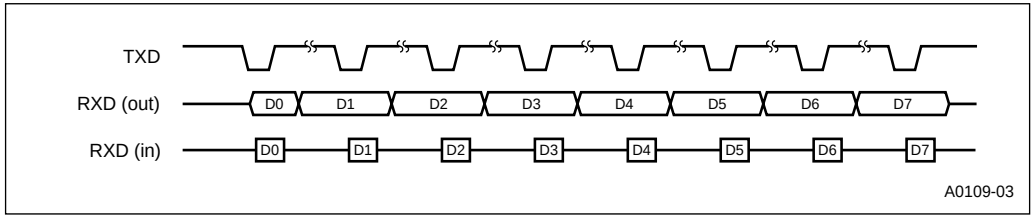


Figure 8-3. Mode 0 Timing

Each channel has a receive interrupt (RI*x*) and a transmit interrupt (TI*x*) that indicate when the receive buffer is full or the transmit buffer is empty. Additionally, the serial port status (SP*x*_STATUS) register contains RI and TI flags. During a reception, the SIO sets the RI flag in SP*x*_STATUS after it samples the eighth data bit. The RI*x* pending bit in the interrupt pending register is set immediately before the RI flag is set. During a transmission, the SIO sets the TI flag immediately after it transmits the eighth data bit unless a delay is programmed. The SIO can delay the setting of the TI flag by one, three, or seven bit times. (See "Programming the Control Register" on page 8-11.) The TI*x* pending bit in the interrupt pending register is set when the TI flag in SP*x*_STATUS is set.

In mode 0, the receiver must be enabled for receptions and disabled for transmissions. (The SP*x*_CON register contains a bit that enables or disables the receiver. See "Programming the Control Register" on page 8-11.) When the receiver is enabled, clearing the receive interrupt (RI) flag in SP*x*_STATUS starts a reception. When the receiver is disabled, writing to SBUF*x*_TX starts a transmission.

Disabling the receiver stops a reception in progress and inhibits further receptions. When the receiver is enabled, clearing the RI flag in SP*x*_STATUS starts a reception; therefore, to avoid a corrupted reception, disable the receiver before clearing the RI flag. This can be handled in an interrupt environment by using software flags or in straight-line code by polling the interrupt pending register to signal the completion of a reception.

8.3.2 Asynchronous Modes (Modes 1, 2, and 3)

Modes 1, 2, and 3 are full-duplex serial modes, meaning that they have dedicated receive and transmit data signals. Mode 1 is the standard eight-bit, asynchronous mode used for normal serial communications. With parity disabled, mode 1 transmits or receives eight data bits; with parity enabled, mode 1 transmits or receives seven data bits and a parity bit. Modes 2 and 3 are nine-bit asynchronous modes typically used for interprocessor communications (see "Multiprocessor Communications" on page 8-10). Like mode 1, mode 3 has optional parity. With parity disabled, mode 3 transmits or receives nine data bits; with parity enabled, mode 3 transmits or receives eight data bits and a parity bit.

When the serial port is configured for mode 1, 2, or 3, writing to SBUF*x*_TX causes the serial port to start transmitting data. (The transmitter transfers the data to the transmit shift register and starts shifting the data out through TXD*x*.) New data placed in SBUF*x*_TX is transferred to the shift register only after the stop bit of the previous data has been sent. If the receiver is enabled, a falling edge on the RXD*x* input causes the serial port to begin receiving data. Disabling the receiver stops a reception in progress and inhibits further receptions. (See "Programming the Control Register" on page 8-11.)

To minimize noise-related errors, the SIO samples the data line three times and uses majority logic to identify a valid start bit. That is, if two of the three samples are low, the bit is a valid start bit.

8.3.2.1 Mode 1

Mode 1 is the standard asynchronous communications mode with optional parity. If parity is enabled, the receiver checks for even or odd parity, and the transmitter sends data with even or odd parity. When parity is disabled, the data frame used in this mode () consists of ten bits: a start bit (0), eight data bits (LSB first), and a stop bit (1). When parity is enabled, the eighth data bit becomes the parity bit; therefore, the data frame consists of a start bit (0), seven data bits (LSB first), a parity bit, and a stop bit (1).

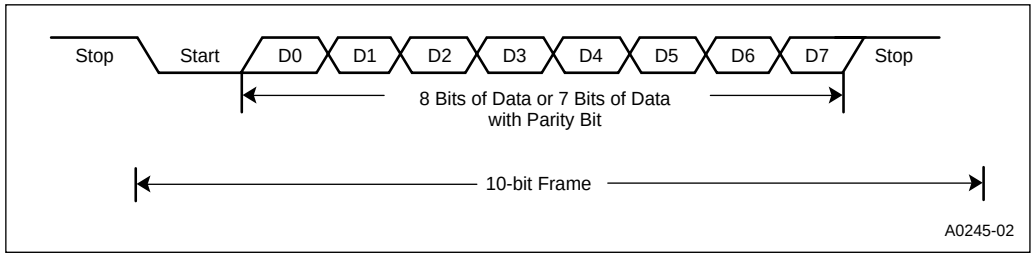


Figure 8-4. Serial Port Frames for Mode 1

The transmit and receive functions are controlled by separate shift clocks. The baud-rate generator controls both the transmit and receive shift clocks. The transmit shift clock starts when the baud-rate generator is initialized. The receive shift clock is reset when a start bit (falling edge) is received. Therefore, the transmit clock may not be synchronized with the receive clock, although both will be at the same frequency.

The SIO sets the transmit interrupt (TI) and receive interrupt (RI) flags in SP*x*_STATUS to indicate completed operations. During a reception, the SIO sets both the RI flag and the RI*x* interrupt pending bit just before it receives the end of the stop bit. During a transmission, the SIO sets the TI flag immediately after it starts to transmit the stop bit unless a delay is programmed. The SIO can delay the setting of the TI flag by one, three, or seven bit times. (See "Programming the Control Register" on page 8-11.)

When connecting more than two microcontrollers with the serial port in half-duplex (that is, using a single data signal for both transmit and receive operations), it is important to wait for a reception to complete before starting to transmit. The receiving processor must wait for one bit time after the RI flag is set before starting to transmit. Otherwise, the transmission could corrupt the stop bit, causing a problem for other microcontrollers listening on the link. In addition to the receive interrupt pending bit (RI*x*), which indicates that the reception will be complete in one bit time, the serial port status (SP*x*_STATUS) register contains a reception in progress (RIP) bit. The SIO sets RIP at the start of a reception and clears it after the completion of a reception. (See "Determining Serial Port Status" on page 8-16.)

8.3.2.2 Mode 2

Mode 2 is the asynchronous, ninth-bit recognition mode. shows the data frame used in this mode. It consists of a start bit (0), nine data bits (LSB first), and a stop bit (1). During transmissions, write data bits 0–7 to the transmit buffer (SBUF*x*_TX) and write data bit 8 (the ninth data bit) to the transmit bit 8 (TB8) bit in the serial port control (SP*x*_CON) register. The SIO clears the TB8 bit after every transmission, so you must set it (if desired) before each write to SBUF*x*_TX. During receptions, the receive buffer (SBUF*x*_RX) contains data bits 0–7, and bit 7 in the serial port status (SP*x*_STATUS) register contains data bit 8 (the ninth data bit received).

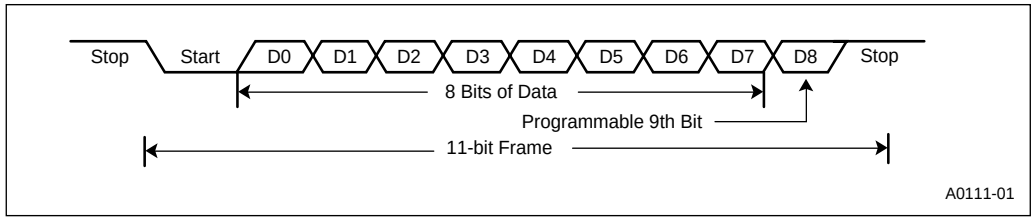


Figure 8-5. Serial Port Frames in Mode 2 and 3

As in mode 1, in mode 2, the SIO sets the transmit interrupt (TI) flag in SP*x*_STATUS to indicate completed transmissions. During a transmission, the SIO sets the TI flag immediately after it starts to transmit the stop bit unless a delay is programmed. The SIO can delay the setting of the TI flag by one, three, or seven bit times. (See “Programming the Control Register” on page 8-11.) Unlike mode 1, in mode 2, the SIO sets the receive interrupt (RI) flag in SP*x*_STATUS only when the ninth data bit received is set. During a reception, when the ninth data bit is set, both the RI flag and the RI*x* interrupt pending bit are set just before the end of the stop bit. This feature provides an easy way to have selective reception on a data link. (See “Multiprocessor Communications” on page 8-10.) Parity is not available in mode 2.

8.3.2.3 Mode 3

Mode 3 is the asynchronous, ninth-bit mode with optional parity. The data frame for this mode is identical to that of mode 2 (Figure 8-5 on page 8-9). Mode 3 differs from mode 2 during transmissions in that parity can be enabled, in which case the ninth bit becomes the parity bit. When parity is disabled, data bits 0–7 are written to the serial port transmit buffer (SBUF*x*_TX), and the ninth data bit is written to SP*x*_CON.4 (TB8). In mode 3, a reception always sets the RI*x* interrupt pending bit, regardless of the state of the ninth bit. If parity is disabled, the SP*x*_STATUS register bit 7 (RB8) contains the ninth data bit received. If parity is enabled, then the SP*x*_STATUS register bit 7 becomes the received parity error (RPE) flag.

8.3.2.4 Multiprocessor Communications

Modes 2 and 3 are provided for multiprocessor communications. In mode 2, during receptions, the serial port sets the RI flag in SP*x*_STATUS and the RI*x* interrupt pending bit only when the ninth data bit received (SP*x*_STATUS.7, the RB8 bit) is a one. In mode 3, the serial port sets the RI flag and the RI*x* interrupt pending bit regardless of the value of the ninth data bit received.

One way to use these modes for multiprocessor communication is to set the master processor to mode 3 and the slave processors to mode 2. When the master processor wants to transmit a block of data to one of several slaves, it sends out an address frame that identifies the target slave. The ninth bit is always set in the address frame, so an address frame interrupts all slaves. Each slave examines the address byte to check whether it is being addressed. The addressed slave switches to mode 3 to receive the data frames, which are sent with the ninth bit cleared. The slaves that are not addressed continue to operate in mode 2, and therefore are not interrupted by the data frames, which are sent with the ninth data bit cleared.

8.4 PROGRAMMING THE SERIAL PORT

To use the SIO port, you must configure the port pins to serve as special-function signals and set up the SIO channels.

8.4.1 Configuring the Serial Port Pins

Before you can use the serial port, you must configure the associated port pins to serve as special-function signals. Table 8-1 on page 8-4 describes the pins associated with the serial port, Table 8-2 on page 8-4 describes the port configuration registers, and Table 8-3 explains how to configure the pins.

Table 8-3. Port Register Settings for the SIO Signals

Signal	Configuration	Port Register Settings
RXD0 (mode 0)	Input for receptions Open-drain output for transmissions	P2_DIR.1 = 1 P2_MODE.1 = 1 (external pull-up required)
RXD0 (modes 1, 2, and 3)	Input	P2_DIR.1 = 1 P2_MODE.1 = 1 P2_REG.1 = 1
RXD1 (mode 0)	Input for receptions Open-drain output for transmissions	P2_DIR.4 = 1 P2_MODE.4 = 1 (external pull-up required)
RXD1 (modes 1, 2, and 3)	Input	P2_DIR4 = 1 P2_MODE.4 = 1 P2_REG.4 = 1

Table 8-3. Port Register Settings for the SIO Signals (Continued)

Signal	Configuration	Port Register Settings
T2CLK	Input	P7_DIR.2 = 1 P7_MODE.2 = 1 P7_REG.2 = 1
TXD0	Complementary output	P2_DIR.0 = 0 P2_MODE.0 = 1
TXD1	Complementary output	P2_DIR.3 = 0 P2_MODE.3 = 1

8.4.2 Programming the Control Register

The SP*x*_CON register (Figure 8-6) selects the communication mode and enables or disables the receiver for all modes. For modes 1 and 3, SP*x*_CON enables or disables even or odd parity. For modes 2 and 3, SP*x*_CON contains the ninth data bit to be transmitted. Use this register to delay the setting of the TI and TXE flags in the serial port status register and the TI*x* interrupt pending bit by one, three, or seven bit times. Selecting a new mode stops any transmission or reception in progress on the channel.

SP*x*_CON
***x* = 0–1**

Address: 1F8BH, 1F9BH
Reset State: 00H

The serial port control (SP*x*_CON) register selects the communications mode and enables or disables the receiver for all modes. For modes 1 and 3, it enables or disables even or odd parity. For modes 2 and 3, it contains the ninth data bit to be transmitted. Use this register to delay the setting of the TI and TXE flags in the serial port status register and the TI*x* interrupt pending bit by one, three, or seven bit times.

7							0
ID1	ID0	PAR	TB8	REN	PEN	M1	M0

Bit Number	Bit Mnemonic	Function
7:6	ID1:0	Interrupt Delay These bits delay the setting of the TI and TXE flags in the SP*x*_STATUS register and the TI*x* interrupt pending bit after transmissions. For mode 0, the SIO sets the TI and TXE flags and the TI*x* pending bit immediately after it shifts out the eighth data bit unless a delay is selected. For modes 1, 2, and 3, the SIO sets TI and TXE flags and the TI*x* pending bit when it starts to shift out the stop bit unless a delay is selected. **ID1 ID0** 0 0 set immediately (no delay) 0 1 delay by one bit time 1 0 delay by three bit times 1 1 delay by seven bit times
5	PAR	Parity Selection Bit In modes 1 and 3, this bit selects even or odd parity. 0 = even parity 1 = odd parity For modes 0 and 2, this bit is ignored.
4	TB8	Transmit Ninth Data Bit This is the ninth data bit that will be transmitted in mode 2 or 3. This bit is cleared after each transmission, so you must write to this bit before writing to SBUF*x*_TX. For mode 3, when parity is enabled (SP*x*_CON.2 = 1), the transmitter sets or clears this bit so that the byte being transmitted contains the correct parity.
3	REN	Receive Enable In mode 1, 2, or 3, setting this bit enables receptions. When this bit is set, a falling edge on the RXD*x* pin starts a reception. In these modes, this bit has no effect on transmissions. In mode 0, clearing this bit enables transmissions and setting it enables receptions. Clearing this bit stops a reception in progress and inhibits further receptions. In mode 0, clearing the RI flag in the SP*x*_STATUS register starts a reception; therefore, to avoid corrupting your reception, clear this bit before clearing the RI bit.

Figure 8-6. Serial Port Control (SP*x*_CON) Register

SP*x*_CON (Continued)
***x* = 0–1**

Address: 1F8BH, 1F9BH
Reset State: 00H

The serial port control (SP*x*_CON) register selects the communications mode and enables or disables the receiver for all modes. For modes 1 and 3, it enables or disables even or odd parity. For modes 2 and 3, it contains the ninth data bit to be transmitted. Use this register to delay the setting of the TI and TXE flags in the serial port status register and the TI*x* interrupt pending bit by one, three, or seven bit times.

7							0
ID1	ID0	PAR	TB8	REN	PEN	M1	M0

Bit Number	Bit Mnemonic	Function
2	PEN	Parity Enable In modes 1 and 3, setting this bit enables parity. For mode 1, when this bit is set, the seventh data bit takes the parity value on transmissions and SP*x*_STATUS.7 becomes the receiver parity error bit. For mode 3, when this bit is set, SP*x*_CON.4 (TB8) takes the parity value on transmissions and SP*x*_STATUS.7 becomes the receive parity error bit. Clear this bit for mode 2. For mode 0, this bit is ignored.
1:0	M1:0	Mode Selection These bits select the communications mode. **M1 M0** 0 0 mode 0, synchronous 0 1 mode 1, 8-bit asynchronous with optional parity 1 0 mode 2, 9-bit asynchronous with optional receive interrupt 1 1 mode 3, 9-bit asynchronous with optional parity

Figure 8-6. Serial Port Control (SP*x*_CON) Register (Continued)

8.4.3 Programming the Baud Rate and Clock Source

The SP*x*_BAUD register (Figure 8-7) selects the clock input for the baud-rate generator and defines the baud rate for all serial I/O modes. For mode 0, this register determines the baud rate output on the serial clock pin (TXD*x*). For modes 1, 2, and 3, this register controls the transmit and receive shift clocks.

SP*x*_BAUD
***x* = 0–1**

Address: 1F8CH, 1F9CH
Reset State: 0000H

The serial port baud rate *x* (SP*x*_BAUD) register selects the clock source and serial port baud rate. The most-significant bit selects the clock source. The lower 15 bits represent baud value, an unsigned integer that determines the baud rate.

The maximum baud value is 32,767 (7FFFH). In asynchronous modes 1, 2, and 3, the minimum baud value is 0000H when using the internal clock source (f) and 0001H when using T2CLK. In synchronous mode 0, the minimum baud value is 0001H for transmissions and 0002H for receptions.

WARNING: Writing to the SP*x*_BAUD register during a reception or transmission can corrupt the received or transmitted data. Before writing to SP*x*_BAUD, check SP*x*_STATUS or the interrupt pending register to ensure that the reception or transmission is complete.

15							**8**
CLKSRC	BV14	BV13	BV12	BV11	BV10	BV9	BV8
7							**0**
BV7	BV6	BV5	BV4	BV3	BV2	BV1	BV0

Bit Number	Bit Mnemonic	Function
15	CLKSRC	Serial Port Clock Source This bit determines whether the baud-rate generator is clocked from an internal or an external source. 0 = signal on the T2CLK pin (external source) 1 = internal operating frequency (f)
14:0	BV14:0	These bits constitute the baud value. Use the following equations to determine the baud value for a given baud rate. Synchronous mode 0:† $\text{Baud Value} = \frac{f}{\text{Baud Rate} \times 2} - 1$ or $\frac{\text{T2CLK}}{\text{Baud Rate}}$ Asynchronous modes 1, 2, and 3: $\text{Baud Value} = \frac{f}{\text{Baud Rate} \times 16} - 1$ or $\frac{\text{T2CLK}}{\text{Baud Rate} \times 8}$ † For mode 0 receptions, the baud value must be 0002H or greater. For mode 0 transmissions, the baud value must be 0001H or greater.

Figure 8-7. Serial Port *x* Baud Rate (SP*x*_BAUD) Register

NOTE

For mode 0 receptions, the baud value must be 0002H or greater. Otherwise, the resulting data in the receive shift register will be incorrect.

The reason for this restriction is that the receive shift register is clocked from an internal signal rather than the signal on TXD*x*. Although these two signals are normally synchronized, the internal signal generates one clock before the first pulse transmitted by TXD*x* and this first clock signal is not synchronized with TXD*x*. This clock signal causes the receive shift register to shift in whatever data is present on the RXD*x* pin. This data is treated as the least-significant bit (LSB) of the reception. The reception then continues in the normal synchronous manner, but the data received is shifted left by one bit because of the false LSB. The seventh data bit transmitted is received as the most-significant bit (MSB), and the transmitted MSB is never shifted into the receive shift register.

Using the internal peripheral clock at 40 MHz, the maximum baud rate for mode 0 is 5.52 Mbaud. The maximum baud rate for modes 1, 2, and 3 is 2.0 Mbaud.

Table 8-4 shows the SP*x*_BAUD values for common baud rates when using a 40 MHz, internal peripheral clock. Because of rounding, the baud value formula is not exact and the resulting baud rate is slightly different than desired. The tables show the percentage of error when using the sample SP*x*_BAUD values. In most cases, a serial link will work with up to a 5.0% difference in the receiving and transmitting baud rates.

Table 8-4. SP*x*_BAUD Values When Using the Internal Clock at 40 MHz

Baud Rate	SP*x*_BAUD Register Value†		% Error	
	Mode 0	Mode 1, 2, 3	Mode 0	Mode 1, 2, 3
19200	8411H	8081H	–0.04	+0.16
9600	8822H	8103H	+0.02	+0.16
4800	9046H	8208H	–0.01	–0.04
2400	A080H	8411H	+0.01	–0.04
1200	C11AH	8822H	–0.01	+0.02

† Bit 15 is always set when the internal peripheral clock is selected as the clock source for the baud-rate generator.

8.4.4 Enabling the Serial Port Interrupts

Each serial port channel has both a transmit interrupt (TI*x*) and a receive interrupt (RI*x*). These interrupts indicate completed operations. For mode 0 receptions, the SIO sets the RI*x* interrupt pending bit after it samples the eighth data bit. For mode 1 and 3 receptions, the SIO sets the RI*x* interrupt pending bit just before it receives the end of the stop bit. For mode 2 receptions, the SIO sets the RI*x* interrupt pending bit just before it receives the end of the stop bit only if the ninth data bit received was set. For mode 0 transmissions, the SIO sets the TI*x* interrupt pending bit immediately after it transmits the eighth data bit, unless a delay is programmed. For mode 1, 2, and 3 transmissions, the SIO sets the TI flag immediately after it starts to transmit the stop bit, unless a delay is programmed. The SIO can delay the setting of the TI flag by one, three, or seven bit times. (See "Programming the Control Register" on page 8-11.)

To enable an interrupt, set the corresponding mask bit in the interrupt mask register (see INT_MASK on page C-33) and execute the EI instruction to globally enable servicing of interrupts. See **Chapter 8, "Programming the Control Register,"** for more information about interrupts.

8.4.5 Determining Serial Port Status

The SP*x*_STATUS register (Figure 8-8) contains several bits that reflect the status of the serial port. Reading SP*x*_STATUS **clears all bits** except TXE and RIP. To check the status of the serial port, copy the contents of the SP*x*_STATUS register into a shadow register and then execute bit-test instructions such as JBC and JBS on the shadow register. Otherwise, the first bit-test instruction will clear the SP*x*_STATUS register, losing all status information. Since the shadow register is not cleared when read, this method allows you to execute more than one bit-test instruction on the serial port status information. You can also read the interrupt pending register (see INT_PEND on page C-35) to determine the status of the serial port interrupts.

SP*x*_STATUS
***x* = 0–1**

Address: 1F89H, 1F99H
Reset State: 00H

The serial port status (SP*x*_STATUS) register contains bits that indicate the status of the serial port *x*.

7							0
RPE/RB8	RI	TI	FE	TXE	OE	RIP	—

Bit Number	Bit Mnemonic	Function
7	RPE/RB8	Received Parity Error/Received Bit 8 For modes 1 and 3, RPE is set if parity is enabled (SP*x*_CON.2 = 1) and the data received does not contain the correct parity, as programmed in SP*x*_CON. For mode 2, and for mode 3 with parity disabled, this bit is the ninth data bit received. (The serial port receive buffer contains the received data bits 0–7. The received data bit 8 is written to this bit.) Reading SP*x*_STATUS clears this bit.
6	RI	Receive Interrupt This bit indicates whether an incoming data byte has been received. For modes 0, 1, and 3, this bit is set when the last bit (eighth bit for mode 0, or stop bit for modes 1 and 3) is sampled. For mode 2, this bit is set when the stop bit is detected only if the ninth bit received (SP*x*_STATUS, RB8) is a one. Reading SP*x*_STATUS clears this bit.
5	TI	Transmit Interrupt This bit indicates whether a data byte has finished transmitting. For mode 0 transmissions, the SIO sets this bit immediately after it transmits the eighth data bit, unless a delay is selected in the SP*x*_CON register. For mode 1, 2, and 3 transmissions, the SIO sets this bit immediately after it starts to transmit the stop bit, unless a delay is selected in the SP*x*_CON register. You can delay setting this bit by one, three, or seven bit times. Reading SP*x*_STATUS clears this bit.
4	FE	Framing Error For modes 1, 2, and 3, this bit is set if the receiver does not detect a valid stop bit within the appropriate period of time. Reading SP*x*_STATUS clears this bit. For mode 0, this bit has no function.
3	TXE	SBUF*x*_TX Empty The SIO sets this bit, along with the TI flag, if the transmit buffer and the transmit shift register are both empty. Using the SP*x*_CON register, you can delay the setting of this bit by one, three, or seven bit times. When set, this bit indicates that two bytes can be written to the transmit buffer. Writing to the transmit buffer clears this bit.

Figure 8-8. Serial Port Status (SP*x*_STATUS) Register

SP*x*_STATUS (Continued)
***x* = 0–1**

Address: 1F89H, 1F99H
Reset State: 00H

The serial port status (SP*x*_STATUS) register contains bits that indicate the status of the serial port *x*.

7							0
RPE/RB8	RI	TI	FE	TXE	OE	RIP	—

Bit Number	Bit Mnemonic	Function
2	OE	Overrun Error The SIO sets this bit if data in the receive shift register is loaded into SBUF*x*_RX before the previous byte in SBUF*x*_RX is read. Reading SP*x*_STATUS clears this bit.
1	RIP	Reception In Progress For modes 1, 2, and 3, this bit simplifies detecting an idle data line for asynchronous, half-duplex operations. A falling edge on RXD (indicating the possibility of a valid start bit) sets this bit. Either of three conditions clears this bit: • the start bit is found to be invalid, • the stop bit is received, or • a framing error occurs (there is no valid stop bit within the proper time). 0 = receiver is idle 1 = data is being received For mode 0, this bit is not useful. A falling edge on RXD sets this bit and a rising edge clears it.
0	—	Reserved; for compatibility with future devices, write zero to this bit.

Figure 8-8. Serial Port Status (SP*x*_STATUS) Register (Continued)

9

Synchronous Serial I/O (SSIO) Port

CHAPTER 9
SYNCHRONOUS SERIAL I/O (SSIO) PORT

This device has a synchronous serial I/O (SSIO) port that shares pins with port 10. The SSIO provides two unidirectional serial communication channels or one synchronous bidirectional communication channel. The SSIO is compatible with most protocols because the serial clock is completely configurable. Paired, the SSIO channels can operate in a channel-select mode, allowing for communication in multiple-master systems without additional external hardware. Each SSIO channel can also perform handshaking transfers, which along with the PTS, provides unidirectional, multi-byte transfers with no CPU overhead. This chapter describes the SSIO port and explains how to program it.

9.1 SSIO PORT OVERVIEW

The SSIO port has three modes of operation: standard, duplex, and channel-select. For single-slave, single-master systems requiring only one SSIO channel or two independent SSIO channels, use the standard mode of operation. In standard mode, each channel has a unique serial clock and data signal. For single-slave, single-master systems requiring two SSIO channels, use the duplex mode of operation. In duplex mode, the channels have unique serial data signals, but share a common clock signal. For systems with multiple masters or multiple slaves, use the channel-select mode of operation. Like duplex mode, in channel-select mode, the channels have unique serial data signals, but share a common clock signal. Channel-select mode provides an additional channel-select signal designed for communicating in systems with multiple masters or multiple slaves.

In addition to three operating modes, the SSIO port can perform two types of data transfers: normal and handshaking. All modes support normal transfers, while only standard mode supports handshaking transfers. Normal transfers are compatible with most protocols because the serial clock is completely configurable. Handshaking transfers allow a slave device to notify the master that it is not ready for a transfer. “Normal Transfers (All Modes)” on page 9-9 and “Handshaking Transfers (Standard Mode Only)” on page 9-10 discuss these transfer types in detail.

9.1.1 Standard Mode

In standard mode, the SSIO channels operate as two independent serial channels. Each channel contains one clock signal and one data signal. This mode is useful for systems in which unidirectional or half-duplex communication is acceptable. Half-duplex communication uses the same data signal for both transmissions and receptions. In standard mode, the channels can act as either master or slave and as either transmitter or receiver, resulting in the four configuration options shown in Figure 9-1.

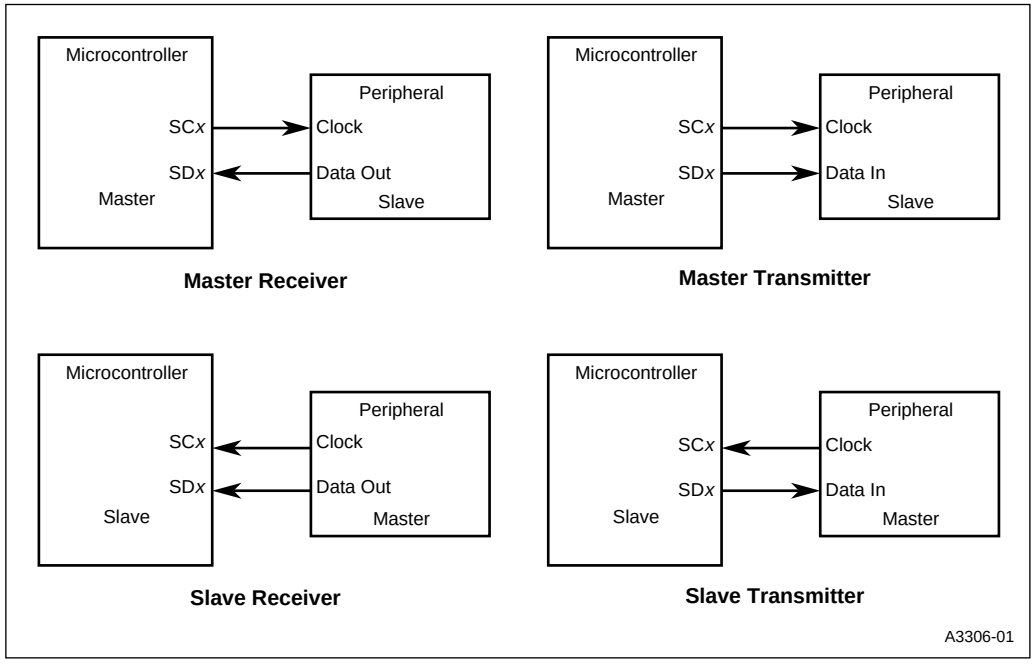


Figure 9-1. Standard Mode Configuration Options

A master device generates a clocking signal and a slave device receives a clock signal. As shown in Figure 9-1, with a master configuration, the SSIO channel outputs a clocking signal on its serial clock (SC*x*) pin. With a slave configuration, the clocking signal is input on SC*x*. When the channel acts as a receiver, the SSIO port shifts an 8-bit data stream (MSB first) into the microcontroller through its serial data (SD*x*) pin. When the channel acts as a transmitter, the SSIO port shifts data out of the microcontroller through SD*x*.

9.1.2 Duplex Mode

In duplex mode, the SSIO channels operate as a pair to provide full-duplex communication. Full-duplex communication uses separate data signals for receptions and transmissions. In this mode, the SSIO can act as either master or slave, and it transmits data through its serial data 0 (SD0) signal and receives data through its serial data 1 (SD1) signal. As master, the SSIO outputs a clocking signal on its serial clock (SC0) pin. As slave, the SSIO receives an externally generated clocking signal on SC0. Figure 9-2 shows the duplex-mode configuration options.

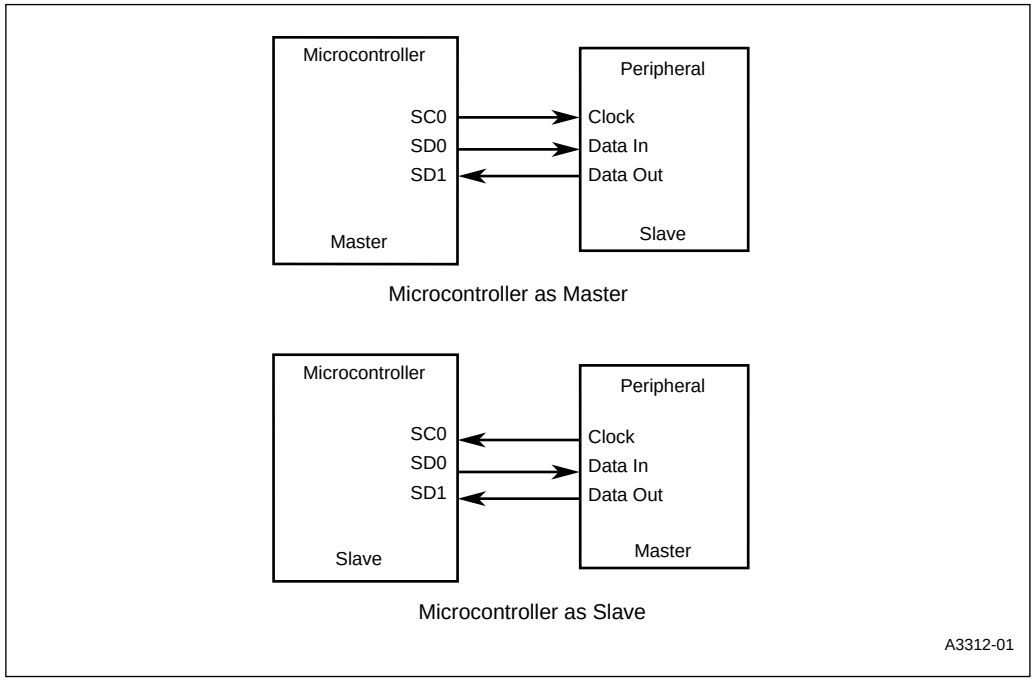


Figure 9-2. Duplex Mode Configuration Options

9.1.3 Channel-select Mode

In channel-select mode, the SSIO channels operate as one serial channel for communications in systems with multiple slaves or multiple masters. With this configuration, the SSIO can act as either master or slave. Channel-select mode uses four signals: two serial data signals (SD0 and SD1), a clock signal (SC0), and a channel-select signal (CHS#). In channel-select mode, the SSIO can provide full-duplex communication, using both SD0 and SD1 data signals, or it can provide half-duplex communication, using only the SD1 data signal. When the SSIO is configured as a slave, an external master activates CHS# to communicate with the SSIO. When the SSIO is configured as a master, an external master activates CHS# when it wants the SSIO to give up the bus. When the microcontroller is the only master in a system, tie the CHS# signal high because the SSIO does not need to relinquish the bus to another master. Figure 9-3 shows channel-select configurations.

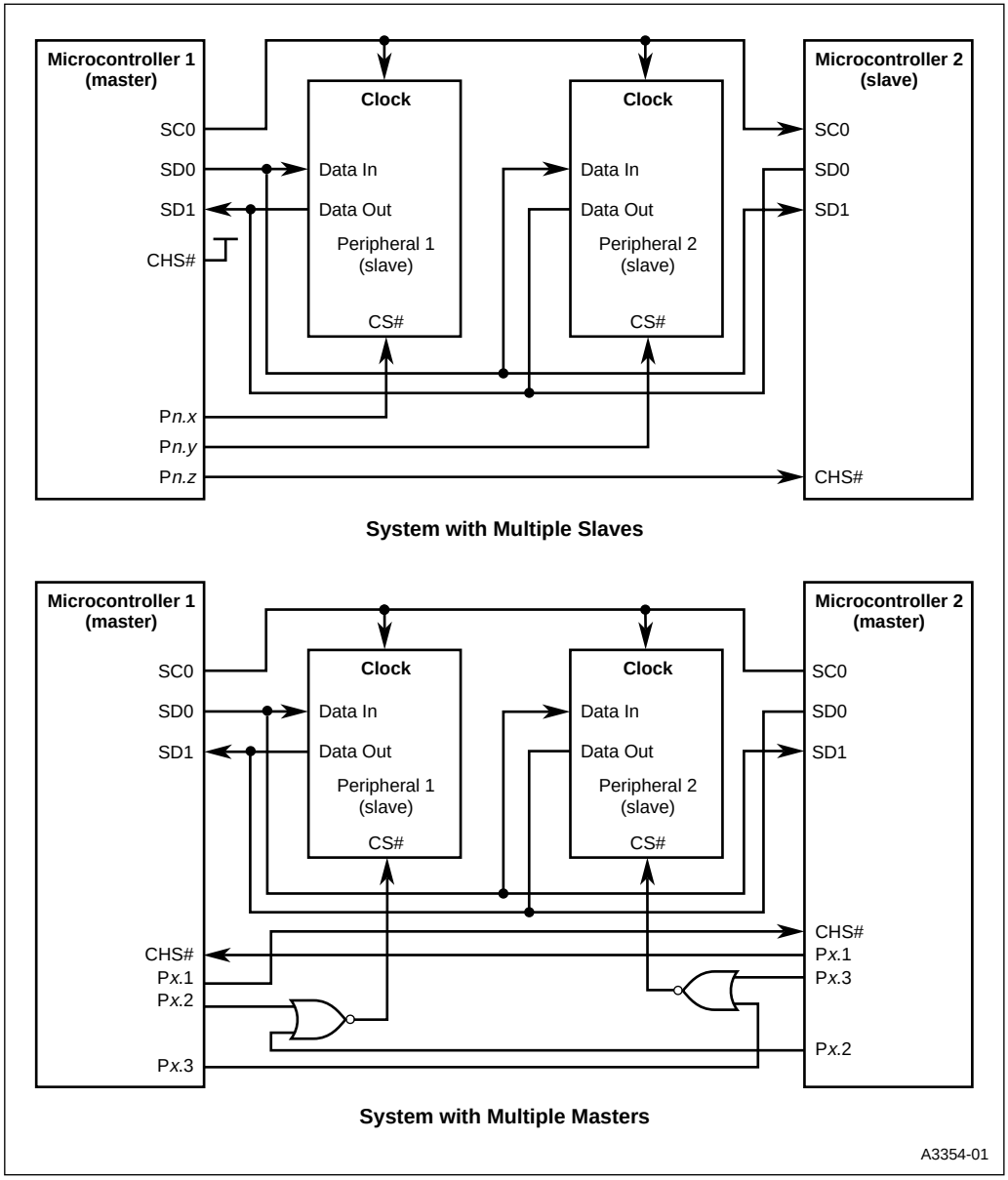


Figure 9-3. Channel-select Configuration Options

9.2 SSIO PORT SIGNALS AND REGISTERS

Table 9-1 and Table 9-2 describe the signals and registers associated with the SSIO port.

Table 9-1. SSIO Port Signals

Signal	Type	Description
CHS#	I	Channel Select This signal is available only when the SSIO is configured for channel-select operation. The function of the signal depends on whether the SSIO is configured as master or slave. When the SSIO is configured as a slave, an external master activates CHS# to communicate with the SSIO. When the SSIO is configured as a master, an external master activates CHS# when it wants the SSIO to give up the bus. CHS# shares a package pin with P10.2.
SC0 SC1	I/O	Serial Clock *x* In standard mode, SC0 is the serial clock pin for channel 0 and SC1 is the serial clock pin for channel 1. In duplex and channel-select modes, SC0 is the serial clock pin for both channels 0 and 1 and SC1 is not available. For normal transfers, configure SC*x* as either a complementary output signal (for master) or a high-impedance input signal (for slave). When channel *x* is configured as a master, the serial clock is output on SC*x*. During transfers, the SC*x* output signal is synchronized with the baud clock; between transfers, the SSIO drives SC*x* to its idle state. When channel *x* is configured as a slave, the serial clock is input on SC*x*. For handshaking transfers, configure SC*x* as an open-drain signal and use an external pull-up resistor. Transfers are initiated only when SC*x* is high; therefore, a slave device can pull SC*x* low when it is not ready. When channel *x* is configured as a master, the serial clock is output on SC*x*. During transfers, the SC*x* output signal is synchronized with the baud clock; between transfers, the SSIO floats SC*x*. When channel *x* is configured as a slave, the SSIO pulls SC*x* low when channel *x*'s buffer is empty for transmissions or full for receptions; it floats SC*x* when channel *x*'s buffer is full for transmissions and empty for receptions. SC0 shares a package pin with P10.0, and SC1 shares a package pin with P10.2.
SD0 SD1	I/O	Serial Data *x* This pin is the serial data pin for channel *x*. For transmissions, configure SD*x* as a complementary output signal. For receptions, configure SD*x* as a high-impedance input signal. In standard mode, SD0 and SD1 can be configured for either transmissions or receptions. In duplex mode, configure SD0 for transmissions and SD1 for receptions. In channel-select half-duplex mode, configure SD1 for transmissions or receptions. In channel-select full-duplex mode, configure SD0 for transmissions and SD1 for receptions. SD0 shares a package pin with P10.1, and SD1 shares a package pin with P10.3.

Table 9-2. SSIO Port Registers

Mnemonic	Address	Description
INT_MASK1	0013H	Interrupt Mask 1 Use this register to enable or disable interrupts from the SSIO channels. As a transmitter, an SSIO channel generates an interrupt request when its transmit buffer is empty. As a receiver, an SSIO channel generates an interrupt request when its receive buffer is full. As a master, in channel-select mode, the SSIO can generate an interrupt request when another device takes control of the bus. In this case, the interrupt request corresponds to the SSIO0 interrupt mask bit. Setting the SSIO0 (INT_MASK1.0) bit of this register enables the SSIO channel 0 transfer interrupt or the SSIO channel-select master contention interrupt; clearing the bit disables (masks) the interrupt. Setting the SSIO1 (INT_MASK1.1) bit of this register enables the SSIO channel 1 transfer interrupt; clearing the bit disables (masks) the interrupt.
INT_PEND1	0012H	Interrupt Pending 1 When set, SSIO0 (INT_PEND1.0) indicates a pending SSIO channel 0 transfer interrupt or SSIO channel-select master contention interrupt. When set, SSIO1 (INT_PEND1.1) indicates a pending channel 1 transfer interrupt.
P10_DIR	1FC3H	Port 10 Direction Each bit controls the configuration of the corresponding pin. Clearing a bit configures the corresponding pin as a complementary output; setting a bit configures the corresponding pin as an open-drain output or high-impedance input. Write to P10_DIR.0, P10_DIR.1, P10_DIR.2, and P10_DIR.3 to configure SC0, SD0, SC1, and SD1. (See "Configuring the SSIO Registers" on page 9-12.)
P10_MODE	1FC1H	Port 10 Mode Each bit controls the mode of the corresponding pin. Setting a bit configures a pin as a special-function signal; clearing a bit configures a pin as a general-purpose I/O signal. Set P10_MODE.0, P10_MODE.1, P10_MODE.2, and P10_MODE.3 to configure pins P10.0, P10.1, P10.2, and P10.3 as SC0, SD0, SC1, and SD1. (See "Configuring the SSIO Registers" on page 9-12.)
P10_PIN	1FC7H	Port 10 Pin Each bit reflects the current state of the corresponding pin, regardless of the pin's mode and configuration.

Table 9-2. SSIO Port Registers (Continued)

Mnemonic	Address	Description
P10_REG	1FC5H	Port 10 Data Output **For I/O Mode** When a port pin is configured as a complementary signal, setting the corresponding port data bit drives a one on the pin and clearing the corresponding port data bit drives a zero on the pin. When a port pin is configured as an open-drain signal, clearing the corresponding port data bit drives a zero on the pin and setting the corresponding port data bit floats the pin, making it available as a high-impedance input. **For Special-function Mode** When a port pin is configured as an output (either complementary or open-drain) signal, the corresponding port data bit value is immaterial because the corresponding on-chip peripheral or system function controls the pin. When a port pin is configured as an open-drain signal, setting the corresponding port data bit floats the pin, making it available as a high-impedance input. Write to P10_REG.0, P10_REG.1, P10_REG.2, and P10_REG.3 to configure SC0, SD0, SC1, and SD1. (See “Configuring the SSIO Registers” on page 9-12.)
PTSSEL	0004H	PTS Select Selects either a PTS microcode routine or a standard interrupt service routine for the SSIO*x* interrupts. Setting the SSIO0 or SSIO1 bit (PTSSEL.8 or PTSSEL.9) of this register selects a PTS microcode routine; clearing the SSIO*x* bit selects a standard interrupt service routine.
PTSSRV	0006H	PTS Service Used by the hardware to indicate that the SSIO*x* interrupt has been serviced by the PTS routine.
SSIO_BAUD	1F94H	SSIO Baud Rate Enables the baud-rate generator and defines the baud rate for the baud clock.
SSIO0_BUF SSIO1_BUF	1F90H 1F92H	SSIO *x* Receive and Transmit Buffers Contains either received data or data for transmission, depending on the channel configuration. Data is shifted into this register from the SD*x* pin or from this register to the SD*x* pin, with the most-significant bit first.
SSIO0_CON SSIO1_CON	1F91H 1F93H	SSIO *x* Control Configures the SSIO channel as either master or slave, transmitter or receiver, selects the channel’s transfer type (normal or handshaking), determines whether the SSIO enables or disables the channel at the completion of a transfer, and indicates the buffer status and whether a buffer error occurred in the last transfer.

Table 9-2. SSIO Port Registers (Continued)

Mnemonic	Address	Description
SSIO0_CLK	1F95H	SSIO 0 Clock Configures the serial clock for channel 0. It determines the idle state of the serial clock. For transmissions, SSIO0_CLK determines whether the SSIO shifts out data bits on rising or falling clock edges. For receptions, SSIO0_CLK determines whether the SSIO samples data bits on rising or falling clock edges. The serial clock is configurable only for normal transfers; therefore, this register is ignored for handshaking transfers.
SSIO1_CLK	1F97H	SSIO 1 Clock Selects the operating mode (standard, duplex, or channel select) and enables the channel-select mode's master contention interrupt request. For normal transfers, this register configures the serial clock for channel 1. It determines the idle state of the serial clock. For transmissions, SSIO1_CLK determines whether the SSIO shifts out data bits on rising or falling clock edges. For receptions, SSIO1_CLK determines whether the SSIO samples data bits on the rising or falling clock edges.

9.3 SSIO PORT OPERATION

The SSIO port contains two identical transceiver channels and a baud-rate generator. Each transceiver channel contains an 8-bit buffer register (SSIO*x*_BUF), a control register (SSIO*x*_CON), a data signal (SD*x*), and an interrupt signal (SSIO*x*). In standard mode, each channel contains a clock signal (SC*x*). In duplex and channel-select modes, the channels share a clock signal (SC0). Additionally, the SSIO port contains two registers (SSIO0_CLK and SSIO1_CLK) that select the operating mode and configure the serial clock signals.

9.3.1 Transmitting and Receiving Data

The SSIO port can perform two types of data transfers: normal and handshaking. For normal transfers, the idle state of the serial clock and the serial clock edge on which the SSIO shifts out or samples data bits is programmable. For handshaking transfers, the slave device can pull the serial clock signal low to indicate that it is not ready.

All modes support normal transfers, while only standard mode supports handshaking transfers. For both transfer types, the serial clock controls the rate at which the SSIO shifts data bits out for transmissions and samples data bits for receptions. During transfers, eight pulses on the serial clock cause the SSIO to shift out or sample data bits; between transfers, the serial clock is held at its idle state. When the SSIO channel is configured as a master, the serial clock (SC*x*) signal is internally derived from the SSIO baud-rate generator. During transfers, the serial clock is synchronized with the baud clock and output on the serial clock (SC*x*) pin; between transfers, the SSIO channel drives the serial clock to its idle state. When the SSIO channel is configured as a slave, the serial clock signal is externally derived and input on the serial clock (SC*x*) pin.

9.3.1.1 Normal Transfers (All Modes)

For normal transfers, two conditions must be true for a transfer to begin: the transfer must be enabled and the SSIO buffer must be full for transmissions or empty for receptions. Once a transfer is initiated, the SSIO monitors SC*x*. The SSIO shifts data out during transmissions on falling SC*x* edges and samples data bits during receptions on rising SC*x* edges. The edge on which the SSIO shifts data out during transmissions and the edge on which the SSIO samples data bits during receptions is programmable. When the SSIO channel is configured as a master, after it shifts out or samples the least-significant data bit, it drives the serial clock to its programmed idle state. Figure 9-4 shows the four clock options for transmissions and Figure 9-5 shows the four clock options for receptions.

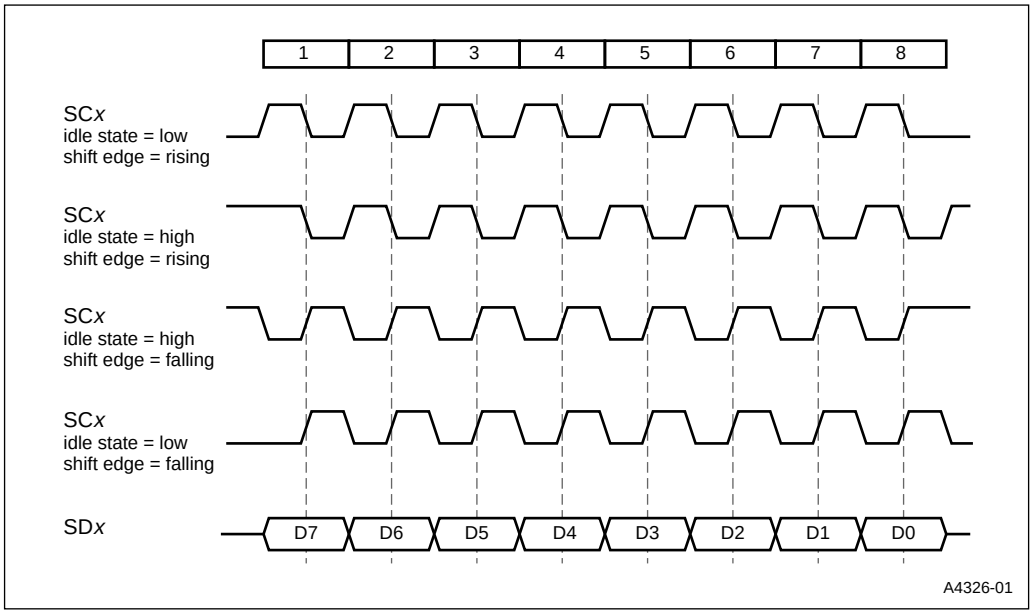


Figure 9-4. Serial Clock Options for Transmissions

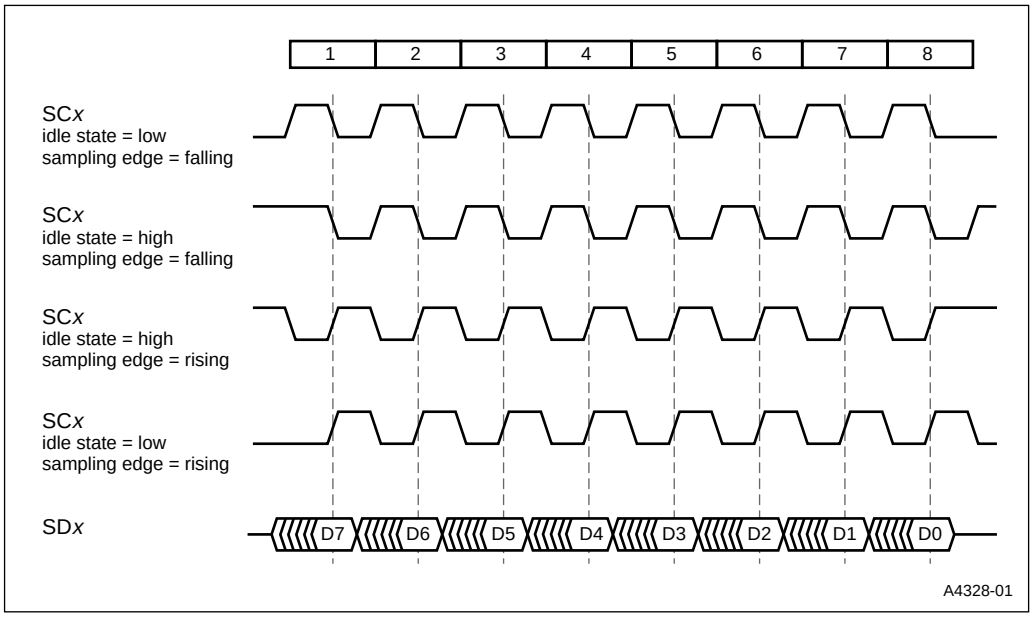


Figure 9-5. Serial Clock Options for Receptions

After the SSIO shifts out or samples its last data bit, it sets a flag in the SSIO control (SSIO*x*_CON) register, indicating the SSIO buffer status, and generates an SSIO interrupt request. If the channel was configured as transmitter, at the completion of the transmission, the SSIO can re-enable the channel as a transmitter, enable the channel as a receiver, or disable the channel. Similarly, if the channel was configured as a receiver, at the completion of the reception, the SSIO can re-enable the channel as a receiver, enable the channel as a transmitter, or disable the channel.

9.3.1.2 Handshaking Transfers (Standard Mode Only)

For handshaking transfers, the clock signal is used with an open-drain configuration. Three conditions must be true for a transfer to begin: the transfer must be enabled, the SSIO buffer must be full for transmissions or empty for receptions, and the serial clock signal must be high. Once a transfer is initiated, the SSIO monitors SC*x*. For transmissions, the SSIO shifts data bits out on rising clock edges. For receptions, the SSIO samples data bits on falling clock edges. When the SSIO channel is configured as a master, after it shifts out or samples the least-significant data bit, it floats the serial clock signal. At this point, the serial clock signal is either pulled high by an external resistor or pulled low by an external slave. When the SSIO channel is configured as a

slave transmitter, it pulls the clock low when the buffer is empty and floats the clock signal when the buffer is full. When the SSIO channel is configured as a slave receiver, it pulls the clock low when the buffer is full and floats the clock signal when the buffer is empty. Figure 9-6 shows the relationship between the serial clock signal (SC*x*) and the serial data signal (SD*x*) for handshaking transfers.

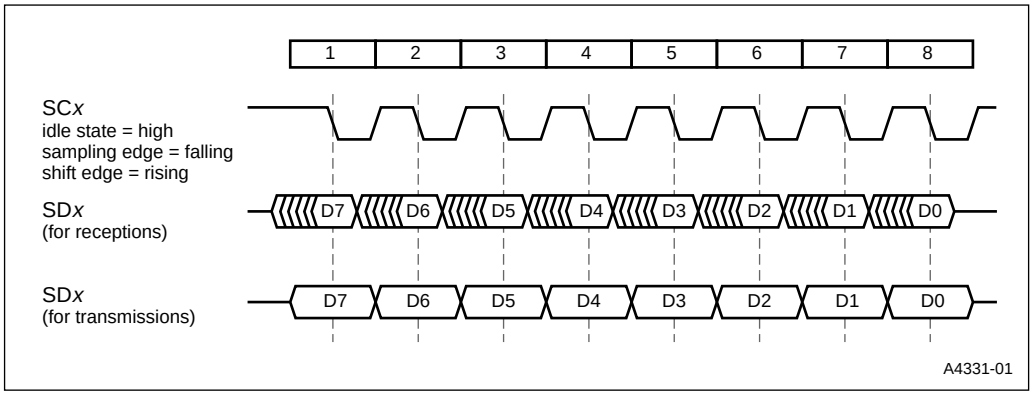


Figure 9-6. Relationship Between Clock and Data Signals (Handshaking Transfers)

9.4 PROGRAMMING THE SSIO PORT

In order to program the SSIO port, you must configure the SSIO port pins and registers. At the end of a transfer, the SSIO generates an interrupt request. Enabling the interrupt allows you to use interrupt service or PTS microcode routines to configure and initiate subsequent transfers. The following sections discuss configuring the SSIO port pins and registers and enabling the SSIO interrupts.

9.4.1 Configuring the SSIO Port Pins

Before you can use the SSIO port, you must configure the necessary port 10 pins to serve as SSIO signals. For the different SSIO configurations, Table 9-3 lists the SSIO signal types and the corresponding port register settings. See Table 9-1 on page 9-5 and Table 9-2 on page 9-6 for descriptions of the SSIO signals and port registers.

Table 9-3. Port Register Settings for the SSIO Signals

SSIO Signal	Signal Type	Port Register Settings
SC0 (master, normal transfers)	Complementary output	P10_DIR.0 = 0 P10_MODE.0 = 1
SC0 (slave, normal transfers)	High-impedance input	P10_DIR.0 = 1 P10_MODE.0 = 1 P10_REG.0 = 1
SC0 (master or slave, handshaking transfers)	Open drain with external pull-up resistor	P10_DIR.0 = 1 P10_MODE.0 = 1
SC1 (master or slave, handshaking transfers)	Open drain with external pull-up resistor	P10_DIR.2 = 1 P10_MODE.2 = 1
SD0 (receiver)	High-impedance input	P10_DIR.1 = 1 P10_MODE.1 = 1 P10_REG.1 = 1
SD0 (transmitter)	Complementary output	P10_DIR.1 = 0 P10_MODE.1 = 1
SD1 (receiver)	High-impedance input	P10_DIR.3 = 1 P10_MODE.3 = 1 P10_REG.3 = 1
SD1 (transmitter)	Complementary output	P10_DIR.3 = 0 P10_MODE.3 = 1
CHS# (master or slave)	High-impedance input	P10_DIR.2 = 1 P10_MODE.2 = 1 P10_REG.2 = 1

9.4.2 Configuring the SSIO Registers

Use the serial port registers to configure the SSIO port. The SSIO baud register enables the baud-rate generator and defines the baud rate for the SSIO baud clock. The SSIO *x* control register selects the configuration (master or slave, transmitter or receiver) and the transfer type (normal or handshaking) for channel *x*. The SSIO 0 clock register configures the serial clock for channel 0. The SSIO 1 clock register selects the SSIO mode of operation (standard, duplex, or channel-select), enables the channel-select master contention interrupt request, and configures the serial clock for channel 1. The following sections describe each SSIO register in detail.

9.4.2.1 The SSIO Baud (SSIO_BAUD) Register

When an SSIO channel is configured as a master, its serial clock is synchronized with the baud clock during transfers. The SSIO_BAUD register (Figure 9-7) enables the baud-rate generator and defines the baud rate of the baud clock. This register acts as a control register during write operations and as a down-counter monitor during read operations. The frequency ranges from f/8 to f/1024. With a 40-MHz oscillator frequency, this corresponds to a range from a maximum of 5 MHz to a minimum of 39. kHz. Table 9-4 lists SSIO_BAUD values for common baud rates.

SSIO_BAUD

Address: 1F94H
Reset State: 00H

The synchronous serial port baud (SSIO_BAUD) register enables and disables the baud-rate generator and selects the baud rate for the internal baud clock. During read operations, SSIO_BAUD serves as the down-counter monitor. The down-counter is decremented once every four state times when the baud-rate generator is enabled.

7							0
BE	BV6	BV5	BV4	BV3	BV2	BV1	BV0

Bit Number	Bit Mnemonic	Function
7	BE	Baud-rate Generator Enable This bit enables and disables the baud-rate generator. **For write operations:** 0 = disable the baud-rate generator and clear BV6:0 1 = enable the baud-rate generator and start the down-counter **For read operations:** 0 = baud-rate generator is disabled 1 = baud-rate generator is enabled and down-counter is running
6:0	BV6:0	Baud Value **For write operations:** These bits represent baud value, an unsigned integer that determines the baud rate. The maximum value of baud value is 7FH; the minimum value is 0. Use the following equation to determine baud value for a given baud rate. $\text{Baud Value} = \frac{f}{\text{Baud Rate} \times 8} - 1$ **For read operations:** These bits contain the current value of the down-counter.

Figure 9-7. Synchronous Serial Port Baud (SSIO_BAUD) Register

Table 9-4. Common SSIO_BAUD Values at 40 MHz Operating Frequency

Baud Rate		SSIO_BAUD Value†
(Maximum)	5.0 MHz	80H
	100.0 KHz	B1H
	50.0 KHz	D3H
	40.0 KHz	FCH
(Minimum)	39.1 KHz	FFH

† Bit 7 must be set to enable the baud-rate generator.

9.4.2.2 The SSIO Control (SSIO*x*_CON) Register

The SSIO*x*_CON register (Figure 9-8) selects the channel configuration (master or slave, transmitter or receiver), data transfer type (normal or handshaking), and indicates buffer status. SSIO*x*_CON also determines whether the SSIO re-enables the channel at the completion of a transfer.

SSIO*x*_CON
***x* = 0–1**

Address: 1F91H, 1F93H
Reset State: 00H

The synchronous serial control *x* (SSIO*x*_CON) register selects the channel configuration (master or slave, transmitter or receiver), the data transfer type (normal or handshaking), and indicates buffer status. SSIO*x*_CON also determines whether the SSIO re-enables the channel at the completion of a transfer.

7							0
M/S#	T/R#	TRT	THS	STE	ATR	OUF	TBS

Bit Number	Bit Mnemonic	Function
7	M/S#	Master/Slave Select Configures the channel as either master or slave. 0 = slave (externally generated clocking signal is input on SC*x*) 1 = master (internally generated clocking signal is output on SC*x*) In standard mode, either channel can act as master or slave. In channel-select and duplex modes, configure channel 1 as a slave (SSIO1_CON.7 = 0) and configure channel 0 as desired.
6	T/R#	Transmit/Receive Select Configures the channel as either transmitter or receiver. 0 = receiver (data is input on SD*x*) 1 = transmitter (data is output on SD*x*) In standard mode, either channel can act as transmitter or receiver. In channel-select and duplex modes, configure channel 0 as the transmitter (SSIO0_CON.6 = 1) and channel 1 as the receiver (SSIO0_CON.6 = 0).
5	TRT	Transmitter/Receiver Toggle Setting this bit causes the SSIO to toggle the channel configuration (transmitter or receiver) at the completion of a transfer, thus avoiding possible contention on the data line. The SSIO clears this bit at the completion of each transfer.
4	THS	Transceiver Handshake Select Selects the type of data transfer. 0 = normal transfers (all modes) 1 = handshaking transfers (standard mode) In standard mode, the channels can perform either normal or handshaking transfers. In channel-select and duplex modes, always configure the channels for normal transfers by clearing this bit. Set STE and ATR, along with THS, for handshaking transfers.

Figure 9-8. Synchronous Serial Control *x* (SSIO*x*_CON) Registers

SSIO*x*_CON (Continued) Address: 1F91H, 1F93H
***x* = 0–1** Reset State: 00H

The synchronous serial control *x* (SSIO*x*_CON) register selects the channel configuration (master or slave, transmitter or receiver), the data transfer type (normal or handshaking), and indicates buffer status. SSIO*x*_CON also determines whether the SSIO re-enables the channel at the completion of a transfer.

7 **0**

M/S#	T/R#	TRT	THS	STE	ATR	OUF	TBS

Bit Number	Bit Mnemonic	Function
3	STE	Single Transfer Enable Enables and disables transfer of a single byte. Unless ATR is set, STE is automatically cleared at the end of a transfer. 0 = disable transfer 1 = enable transfer Set THS, STE, and ATR for handshaking transfers.
2	ATR	Automatic Transfer Re-enable Enables and disables subsequent transfers. 0 = disable subsequent transfers (allow automatic clearing of STE) 1 = enable subsequent transfers (prevent automatic clearing of STE) Set THS, STE, and ATR for handshaking transfers.
1	OUF	Overflow/Underflow Flag Indicates whether an overflow or underflow has occurred. An attempt to access SSIO*x*_BUF during a byte transfer sets this bit. **For the master (M/S# = 1)** 0 = no overflow or underflow has occurred 1 = the core attempted to access SSIO*x*_BUF during the current transfer **For the slave (M/S# = 0)** 0 = no overflow or underflow has occurred 1 = the core attempted to access SSIO*x*_BUF during the current transfer **or** the master attempted to clock data into or out of the slave's SSIO*x*_BUF before the buffer was available
0	TBS	Transceiver Buffer Status Indicates the status of the channel's SSIO*x*_BUF. **For transmissions (T/R# =1)** 0 = SSIO*x*_BUF is full; waiting to transmit 1 = SSIO*x*_BUF is empty; buffer available **For receptions (T/R# = 0)** 0 = SSIO*x*_BUF is empty; waiting to receive 1 = SSIO*x*_BUF is full; data available

Figure 9-8. Synchronous Serial Control *x* (SSIO*x*_CON) Registers (Continued)

9.4.2.3 The SSIO 0 Clock (SSIO0_CLK) Register

The SSIO0_CLK register (Figure 9-9) selects the phase and polarity for the SC0 clock signal. In standard mode, SC0 is channel 0's clock signal. In duplex and channel-select modes, SC0 is the common clock signal for both SSIO channels.

SSIO0_CLK

Address: 1F95H
Reset State: 00H

The SSIO 0 clock (SSIO0_CLK) register configures the serial clock for channel 0. In standard mode, the SC0 is channel 0's clock signal. In duplex and channel-select modes, SC0 is the common clock signal for both SSIO channels.

7							0
—	—	—	—	—	—	PHAS	POLS

Bit Number	Bit Mnemonic	Function
7:2	—	Reserved; for compatibility with future devices, write zeros to these bits.
1	PHAS	Phase and Polarity Select
0	POLS	For normal transfers, these bits determine the idle state of the serial clock and select the serial clock signal edge on which the SSIO samples incoming data bits or shifts out outgoing data bits. These bits are ignored for handshaking transfers. Use SSIO0_ CON to select the type of data transfer (normal or handshaking) for channel 0. **For transmissions** PHAS POLS 0 0 low idle state; shift on falling edges 0 1 high idle state; shift on rising edges 1 0 low idle state; shift on rising edges 1 1 high idle state; shift on falling edges **For receptions** PHAS POLS 0 0 low idle state; sample on rising edges 0 1 high idle state; sample on falling edges 1 0 low idle state; sample on falling edges 1 1 high idle state; sample on rising edges

Figure 9-9. SSIO 0 Clock (SSIO0_CLK) Register

9.4.2.4 The SSIO 1 Clock (SSIO1_CLK) Register

The SSIO1_CLK register (Figure 9-10) selects the SSIO mode of operation (standard, duplex, or channel-select), enables the channel-select master contention interrupt request, and selects the phase and polarity for the serial clock (SC1) for channel 1.

SSIO1_CLK

Address: 1F97H
Reset State: 00H

The SSIO 1 clock (SSIO1_CLK) register selects the SSIO mode of operation (standard, duplex, or channel-select), enables the channel-select master contention interrupt request, and selects the phase and polarity for the serial clock (SC1) for channel 1.

7							0
—	—	CHS	DUP	CONINT	CONPND	PHAS	POLS

Bit Number	Bit Mnemonic	Function
7:6	—	Reserved; for compatibility with future devices, write zeros to these bits.
5	CHS	These bits determine the SSIO operating mode. **CHS DUP** 0 0 standard mode 0 1 duplex mode 1 0 channel-select half-duplex mode (uses SD1 only) 1 1 channel-select full-duplex mode (uses both SD0 and SD1)
4	DUP	
3	CONINT	Master Contention Interrupt For channel-select master operations, the SSIO sets the master contention interrupt pending bit (CONPND) when the CHS# pin is externally activated. In a system with multiple masters, an external master activates the CHS# signal to request control of the serial clock. CONINT determines whether the SSIO sets both CONPND and the SSIO0 interrupt pending bit or only CONPND when the CHS# pin is externally activated. 0 = SSIO sets only CONPND 1 = SSIO sets both CONPND and the SSIO0 interrupt pending bit This bit is valid for channel-select master operations and ignored for all other operations.
2	CONPND	Master Contention Interrupt Pending For channel-select master operations, the SSIO sets this bit when the CHS# pin is externally activated. In a system with multiple masters, an external master activates the CHS# signal to request control of the serial clock. This bit is valid for channel-select master operations and ignored for all other operations.

Figure 9-10. SSIO 1 Clock (SSIO1_CLK) Register

SSIO1_CLK (Continued)

Address: 1F97H
Reset State: 00H

The SSIO 1 clock (SSIO1_CLK) register selects the SSIO mode of operation (standard, duplex, or channel-select), enables the channel-select master contention interrupt request, and selects the phase and polarity for the serial clock (SC1) for channel 1.

7							0
—	—	CHS	DUP	CONINT	CONPND	PHAS	POLS

Bit Number	Bit Mnemonic	Function
1	PHAS	Phase and Polarity Select
0	POLS	For normal transfers, these bits determine the idle state of the serial clock and select the serial clock signal edge that the SSIO samples incoming data bits or shifts out outgoing data bits. **For transmissions** PHAS POLS 0 0 low idle state; shift on falling edges 0 1 high idle state; shift on rising edges 1 0 low idle state; shift on rising edges 1 1 high idle state; shift on falling edges **For receptions** PHAS POLS 0 0 low idle state; sample on rising edges 0 1 high idle state; sample on falling edges 1 0 low idle state; sample on falling edges 1 1 high idle state; sample on rising edges These bits are ignored for duplex and channel-select modes; these modes use SC0 as the common clock signal. These bits are ignored for handshaking transfers. Use SSIO1_ CON to select the type of data transfer (normal or handshaking) for channel 1.

Figure 9-10. SSIO 1 Clock (SSIO1_CLK) Register (Continued)

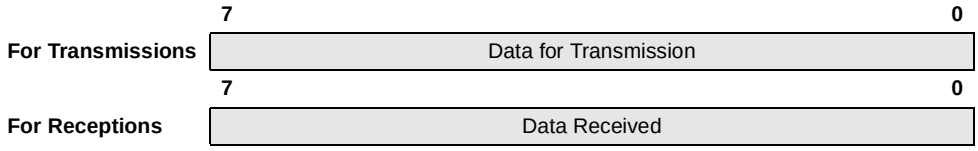
9.4.2.5 The SSIO Buffer (SSIO*x*_BUF) Register

The SSIO*x*_BUF register (Figure 9-11) contains either data for transmission or data received, depending on channel *x*'s configuration. For transmissions, data is shifted from this register to the SD*x* pin; for receptions, data is sampled into this register from the SD*x* pin. For both transmissions and receptions, the data is transferred with most-significant bit first.

SSIO*x*_BUF
***x* = 0–1**

Address: 1F90H, 1F92H
Reset State: 00H

The SSIO *x* buffer (SSIO*x*_BUF) register contains either data for transmission or data received, depending on channel *x*'s configuration. For transmissions, data is shifted from this register to the SD*x* pin; for receptions, data is sampled into this register from the SD*x* pin. For both transmissions and receptions, the data is transferred with most-significant bit first.



Bit Number	Function
7:0	Data for Transmission During transmissions, this register contains a byte of data to be transmitted by the synchronous serial port.
	Data Received During receptions, this register contains the last byte of data received from the synchronous serial port.

Figure 9-11. Synchronous Serial *x* Buffer (SSIO*x*_BUF) Register

9.4.3 Enabling the SSIO Interrupts

Each SSIO channel can generate an interrupt if you enable the individual interrupt as well as globally enabling servicing of all maskable interrupts. The INT_MASK1 register enables and disables individual interrupts. To enable the SSIO0 or SSIO1 interrupts, set the corresponding bits in INT_MASK1 (INT_MASK1.0 or INT_MASK1.1) and execute the EI instruction to globally enable interrupt servicing. For handshaking transfers, you can use the single-transfer PTS microcode interrupt routine to complete multi-byte transfers. In this case, set the SSIO0 or SSIO1 PTS interrupt bit (PTSSEL.8 or PTSSEL.9). See **Chapter 6, "Standard and PTS Interrupts,"** for more information about interrupts.

As a transmitter, SSIO channel *x* generates an SSIO*x* interrupt request when its transmit buffer is empty. As a receiver, SSIO channel *x* generates an SSIO*x* interrupt request when its receive buffer is full. As a master, in channel-select mode, the SSIO can generate a master contention interrupt request when the CHS# pin is externally activated. In this case, the interrupt request corresponds to the SSIO0 interrupt. SSIO1_CLK contains two bits associated with the master contention interrupt: an interrupt enable bit (CONINT) and an interrupt pending bit (CONPND). As a master, in channel-select mode, the SSIO always sets CONPND when the CHS# pin is externally activated. If CONINT is set, the SSIO sets the SSIO0 interrupt pending bit (INT_PEND1.0) along with CONPND when the CHS# pin is externally activated.

9.5 PROGRAMMING CONSIDERATIONS

The following sections discuss some considerations for programming the SSIO port.

9.5.1 Variable-width MSB

For transmissions, the time that you write to SSIO*x*_BUF determines the data setup time (the length of time between data being placed on the data pin and the first clock edge on the clock pin). The reason for this anomaly is that the baud-rate down-counter starts when you write to SSIO_BAUD, but the transmission does not start until you write to SSIO*x*_BUF. The write to SSIO*x*_BUF can occur at any point during the count. Since the most-significant bit (MSB) does not change until the selected shift edge of SC*x*, the width of the MSB appears to vary (Figure 9-12). If you write to SSIO*x*_BUF early in the count, the MSB seems relatively long. If you write to SSIO*x*_BUF late in the count, the MSB seems relatively short.

For example, assume that you write 93H to SSIO_BAUD (the MSB enables the baud-rate generator, and the lower seven bits define the initial count value) and program the SSIO to shift out data on falling SC*x* edges. Also, assume an internal operating frequency of 32 MHz. As soon as you write SSIO_BAUD, the down-counter starts decrementing from 13H. (The baud-rate down-counter decrements every four state times and a state time is equal to 2/f, where f is the internal operating frequency.) If the counter is at 11H when you write to SSIO*x*_BUF, the MSB remains on the data pin for approximately 4.25 µs ($17 \times 4 \times 2/(32 \times 10^6)$). If the counter is at 03H when you write to SSIO*x*_BUF, the MSB remains on the data pin for only approximately 0.75 µs ($3 \times 4 \times 2/(32 \times 10^6)$).

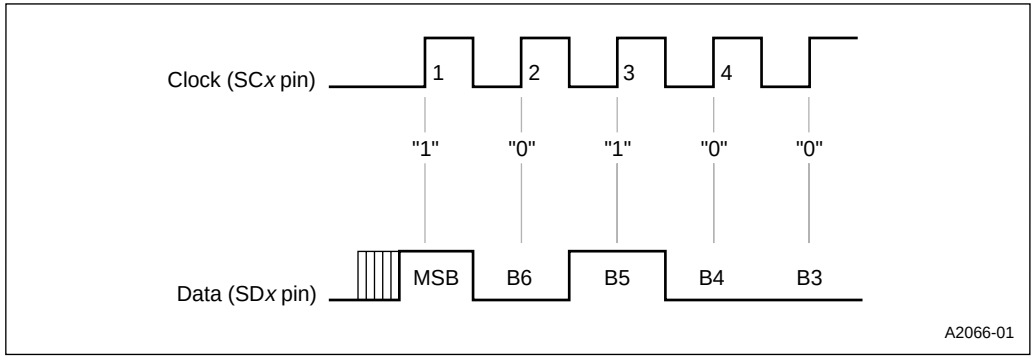


Figure 9-12. Variable-width MSB in SSIO Transmissions

NOTE

This condition exists only for the MSB. Once the MSB is clocked out, the remaining bits are clocked out consistently at the programmed frequency.

One way to achieve a consistent MSB bit length is to start the down-count at a fixed time, using these steps: (Assume ATR bit is set.)

1. Clear SSIO_BAUD bit 7. This disables the baud-rate generator and clears the remaining bits (BV6:0).
2. Disable interrupts.
3. Write the byte to be transmitted to SSIO*x*_BUF.
4. Set the MSB of SSIO_BAUD and write the desired baud value to the remaining bits. This enables the baud-rate generator and starts the down count.
5. Enable interrupts.

Using this procedure starts the clock at a known point before each transmission, establishing a predictable MSB bit time.

9.5.2 Standard Mode Considerations

For standard operations, the serial channels function as two independent channels. SSIO_BAUD enables the baud-rate generator and defines the baud rate for the baud clock. When either channel is configured as master, its serial clock is synchronized with the baud clock during transfers. Use SSIO1_CLK to select standard mode. Use SSIO0_CON, SSIO0_CLK, and SSIO0_BUF for configuring channel 0. Likewise, use SSIO1_CON, SSIO1_CLK, and SSIO1_BUF for configuring channel 1.

9.5.3 Duplex Mode Considerations

For duplex operations, the serial channels function as a pair with SC0 as the common clock signal. However, PHAS and POLS bits in the SSIO0_CLK and SSIO1_CLK registers still affect the internal clocking of channel 0 and channel 1, respectively. Therefore, these bits should be set accordingly for each channel. Use SSIO1_CLK to select duplex mode. In duplex mode, the paired channels can function as master or slave. Use SSIO0_CON to configure the channel pair as master or slave. For master operations, use SSIO_BAUD to enable the baud-rate generator and define the baud clock. In this mode, channel 0's clock signal is internally connected to channel 1's serial clock signal; therefore, you must configure channel 1 as a slave, using SSIO1_CON. This allows serial clock 0 to be input on serial clock 1.

Additionally, for duplex operations, configure channel 0 as a transmitter and channel 1 as a receiver, using SSIO0_CON and SSIO1_CON. At the completion of a transfer, always read SSIO1_BUF before writing SSIO0_BUF to avoid losing the received data in SSIO1_BUF. When the channels are enabled, writing SSIO0_BUF starts a transmission; therefore, writing SSIO0_BUF before reading SSIO1_BUF would cause SSIO1_BUF to be overwritten with the new reception.

9.5.4 Channel-select Mode Considerations

For channel-select operations, the serial channels function as a pair with SC0 as the common clock signal. Use SSIO0_CON to configure the channel pair as master or slave. For master operations, use SSIO_BAUD to enable the baud-rate generator and define the baud clock. In this mode, channel 0's clock signal is internally connected to channel 1's serial clock signal; therefore, you must configure channel 1 as a slave, using SSIO1_CON. This allows serial clock 0 to be input on serial clock 1. Since SC0 is the common clock, use SSIO0_CLK to configure the clock signal.

Use SSIO1_CON to select either full-duplex or half-duplex channel-select operations. Full-duplex operations use both serial data signals (SD0 and SD1), while half-duplex operations use only one serial data signal (SD1). As for duplex operations, for channel-select full-duplex operations, configure channel 0 as a transmitter and channel 1 as a receiver, using SSIO0_CON and SSIO1_CON. At the completion of a transfer, always read SSIO1_BUF before writing SSIO0_BUF to avoid losing the data in SSIO1_BUF. When the channels are enabled, writing SSIO0_BUF starts a transmission; therefore, writing SSIO0_BUF before reading SSIO1_BUF would cause SSIO1_BUF to be overwritten with the new reception.

Since channel-select half-duplex operations use serial clock 0 (SC0) and serial data 1 (SD1), you must write both control registers and both clock registers for these operations. SSIO0_CON configures SC0 for master or slave operations. SSIO1_CON configures SD1 for transmissions or receptions. SSIO0_CLK configures SC0; it selects the clock's idle state and the clock edges on which the SSIO shifts out or samples data bits. Use SSIO1_CLK to configure the internal SC0 clock to channel 1, select channel-select half-duplex operations and enable or disable the master contention interrupt request.

10

Pulse-width Modulator

CHAPTER 10
PULSE-WIDTH MODULATOR

The 8XC196EA has four pulse-width modulator (PWM) modules. Each module consists of an adjacent pair of PWM channels that can generate two PWM output signals with a fixed, programmable frequency and a variable duty cycle. These outputs can be used to drive motors that require an unfiltered PWM waveform for optimal efficiency, or they can be filtered to produce a smooth analog signal.

This chapter provides a functional overview of the pulse-width modulator module, describes how to program it, and provides sample circuitry for converting the PWM outputs to analog signals.

10.1 PWM FUNCTIONAL OVERVIEW

Each PWM module has two channels, each of which consists of a control register (PWM_CONTROL), a buffer, a comparator, an RS flip-flop, and an output pin. Three other components, an up counter, an eight-bit counter (PWM*x_y*_COUNT) and a period register (PWM*x_y*_PERIOD), are shared across the PWM module's two channels, completing the circuitry (see Figure 10-1). Variables *x* and *y* represent the even- and odd-numbered members of an adjacent PWM channel pair, respectively.

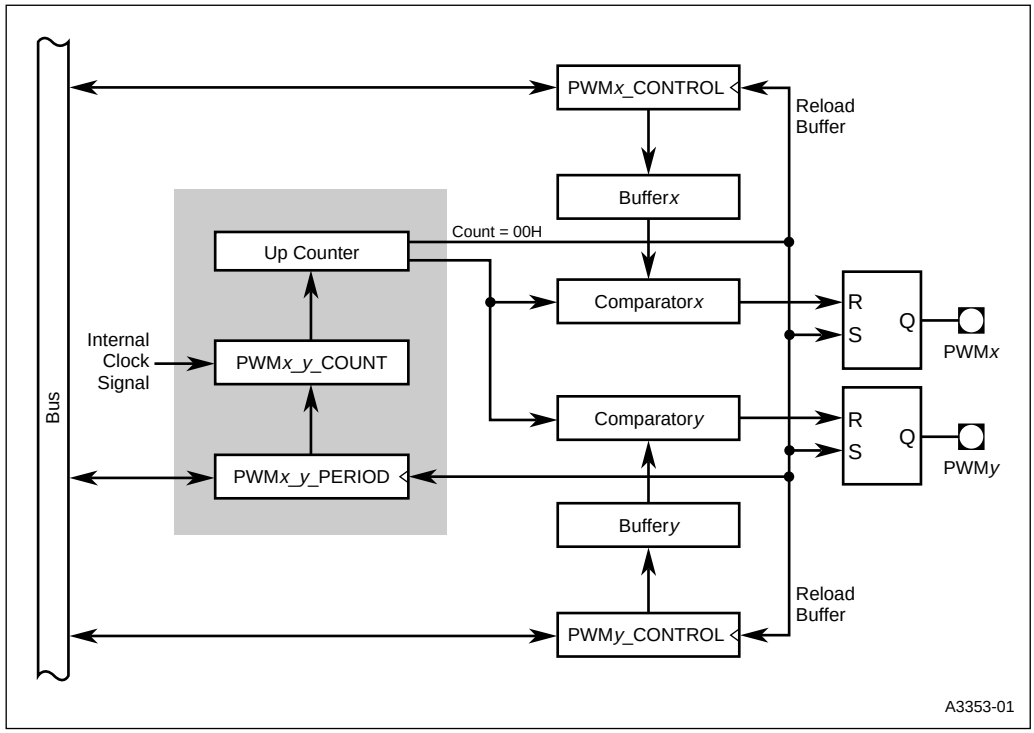


Figure 10-1. PWM Block Diagram

10.2 PWM SIGNALS AND REGISTERS

Table 10-1 describes the PWM's signals and Table 10-2 briefly describes the control and status registers.

Table 10-1. PWM Signals

Port Pin	PWM Signal	PWM Signal Type	Description
P11.0	PWM0	O	Pulse-width modulator 0 output with high-drive capability.
P11.1	PWM1	O	Pulse-width modulator 1 output with high-drive capability.
P11.2	PWM2	O	Pulse-width modulator 2 output with high-drive capability.
P11.3	PWM3	O	Pulse-width modulator 3 output with high-drive capability.
P11.4	PWM4	O	Pulse-width modulator 4 output with high-drive capability.
P11.5	PWM5	O	Pulse-width modulator 5 output with high-drive capability.
P11.6	PWM6	O	Pulse-width modulator 6 output with high-drive capability.

Table 10-1. PWM Signals (Continued)

Port Pin	PWM Signal	PWM Signal Type	Description
P11.7	PWM7	O	Pulse-width modulator 7 output with high-drive capability.

Table 10-2. PWM Control and Status Registers

Mnemonic	Address	Description
P11_DIR	1FBAH	Port Direction Register Each bit controls the configuration of the corresponding pin. Clearing a bit configures a pin as a complementary output; setting a bit configures a pin as a high-impedance input or an open-drain output.
P11_MODE	1FB8H	Port Mode Register Each bit controls the mode of the corresponding pin. Setting a bit configures a pin as a special-function signal; clearing a bit configures a pin as a general-purpose I/O signal.
P11_PIN	1FBEH	Port Pin Register Each bit reflects the current state of the corresponding pin, regardless of the pin's mode and configuration.
P11_REG	1FBCH	Port Data Output Register **For I/O Mode (P*x*_MODE.*x* = 0)** When a port pin is configured as a complementary output (P*x*_DIR.*x* = 0), setting the corresponding port data bit drives a one on the pin, and clearing the corresponding port data bit drives a zero on the pin. When a port pin is configured as a high-impedance input or an open-drain output (P*x*_DIR.*x* = 1), clearing the corresponding port data bit drives a zero on the pin, and setting the corresponding port data bit floats the pin, making it available as a high-impedance input. **For Special-function Mode (P*x*_MODE.*x* = 1)** When a port pin is configured as an output (either complementary or open-drain), the corresponding port data bit value is immaterial because the corresponding on-chip peripheral or system function controls the pin. To configure a pin as a high-impedance input, set both the P*x*_DIR and P*x*_REG bits.
PWM0_CONTROL PWM1_CONTROL PWM2_CONTROL PWM3_CONTROL PWM4_CONTROL PWM5_CONTROL PWM6_CONTROL PWM7_CONTROL	1EDEH 1EDCH 1EDAH 1ED8H 1ED6H 1ED4H 1ED2H 1ED0H	PWM Duty Cycle This register controls the PWM duty cycle. A zero loaded into this register will cause the PWM to output a low continuously (0% duty cycle). An FFH in this register will cause the PWM to have its maximum duty cycle (99.6% duty cycle).

Table 10-2. PWM Control and Status Registers (Continued)

Mnemonic	Address	Description
PWM0_1_PERIOD PWM2_3_PERIOD PWM4_5_PERIOD PWM6_7_PERIOD	1EDFH 1EDBH 1ED7H 1ED3H	PWM Period This register holds a programmed value that determines the output period of a PWM output pair. The value is reloaded into the counter each time the counter resets to 00H.
PWM0_1_COUNT PWM2_3_COUNT PWM4_5_COUNT PWM6_7_COUNT	1EDDH 1ED9H 1ED5H 1ED1H	PWM Counter This read-only register contains the current value of the period counter.

10.3 PWM OPERATION

The period register (PWM*x_y*_PERIOD) of each module controls the output frequency of both PWM outputs. Each control register (PWM_CONTROL) controls the duty cycle (the pulse width stated as a percentage of the period) of the corresponding PWM output. Each control register contains an eight-bit value that is loaded into a buffer when the eight-bit counter rolls over from FFH to 00H. The comparators compare the contents of the buffers to the counter value. Since the value written to the control register is buffered, you can write a new eight-bit value to either PWM_CONTROL register at any time. However, the comparators recognize the new value only after the current eight-bit count expires. The new value is used during the next PWM output period.

The counter continually increments until it rolls over to 00H, at which time the PWM output is driven high and the contents of the control registers are loaded into the buffers. The PWM output remains high until the counter value matches the value in the buffer, at which time the output is pulled low. You can read the count register (PWM*x_y*_COUNT) to see the current value of the counter. When the counter resets again (i.e., when an overflow occurs) the output is switched high. (Loading PWM_CONTROL with 00H forces the output to remain low.) Figure 10-2 shows typical PWM output waveforms.

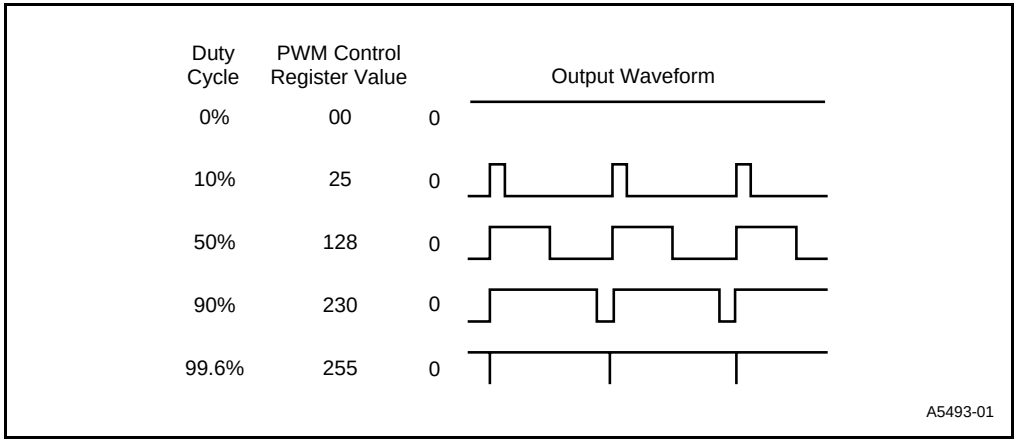


Figure 10-2. PWM Output Waveforms

10.4 PROGRAMMING THE FREQUENCY AND PERIOD

The input frequency on XTAL1 (F_{XTAL1}) and the contents of the PWM*x_y*_PERIOD register determine the PWM output frequency (F_{PWM}) and period (T_{PWM}). Table 10-3 shows the PWM output frequencies for common values of F_{XTAL1} with a variety of PWM*x_y*_PERIOD values. Use the following formulas to calculate the PWM period value for the desired output frequency and write the corresponding value to the PWM*x_y*_PERIOD register (Figure 10-3 on page 10-7).

$$T_{PWM}\ (\text{in } \mu s) = \frac{P \times (PWMx\_y\_PERIOD + 1)}{F_{XTAL1}}$$

$$F_{PWM}\ (\text{in MHz}) = \frac{F_{XTAL1}}{P \times (PWMx\_y\_PERIOD + 1)}$$

where:

T_{PWM} = output period on the PWM output pins, in µs

F_{PWM} = output frequency on the PWM output pins, in MHz

P = 256 if the phase-locked loop (PLL) circuitry is enabled
512 if the phase-locked loop (PLL) circuitry is disabled

PWM*x_y*_PERIOD = eight-bit value to load into the PWM_PERIOD register

F_{XTAL1} = input frequency on XTAL1 pin, in MHz

Table 10-3. PWM Output Frequencies (F_{PWM})

PWM*x_y*_PERIOD	F_{XTAL1} with PLL Enabled ($f^{\dagger} = 2F_{XTAL1}$)		F_{XTAL1} with PLL Disabled ($f = F_{XTAL1}$)		
	10 MHz	**20 MHz**	**10 MHz**	**20 MHz**	**40 MHz**
00H	39.1 KHz	78.1 KHz	19.5 KHz	39.1 KHz	78.1 KHz
0FH	2441.4 Hz	4882.8 Hz	1220.7 Hz	2441.4 Hz	4882.8 Hz
1FH	1220.7 Hz	2441.4 Hz	610.4 Hz	1220.7 Hz	2441.4 Hz
2FH	813.8 Hz	1627.6 Hz	406.9 Hz	813.8 Hz	1627.6 Hz
3FH	610.4 Hz	1220.7 Hz	305.2 Hz	610.4 Hz	1220.7 Hz
4FH	488.3 Hz	976.6 Hz	244.1 Hz	488.3 Hz	976.6 Hz
5FH	406.9 Hz	813.8 Hz	203.5 Hz	406.9 Hz	813.8 Hz
6FH	348.8 Hz	697.5 Hz	174.4 Hz	348.8 Hz	697.5 Hz
7FH	305.2 Hz	610.4 Hz	152.6 Hz	305.2 Hz	610.4 Hz
8FH	271.3 Hz	542.5 Hz	135.6 Hz	271.3 Hz	542.5 Hz
9FH	244.1 Hz	488.3 Hz	122.1 Hz	244.1 Hz	488.3 Hz
AFH	221.9 Hz	443.9 Hz	111.0 Hz	221.9 Hz	443.9 Hz
BFH	203.5 Hz	406.9 Hz	101.7 Hz	203.5 Hz	406.9 Hz
CFH	187.8 Hz	375.6 Hz	93.9 Hz	187.8 Hz	375.6 Hz
DFH	174.4 Hz	348.8 Hz	87.2 Hz	174.4 Hz	348.8 Hz
EFH	162.8 Hz	325.5 Hz	81.4 Hz	162.8 Hz	325.5 Hz
FFH	152.6 Hz	305.2 Hz	76.3 Hz	152.6 Hz	305.2 Hz

† f = internal operating frequency

PWM*x_y*_PERIOD
***x* = 0, 2, 4, 6**
***y* = 1, 3, 5, 7**

Address: See Table 10-2 on page 10-5
Reset State: 00H

The PWM period (PWM*x_y*_PERIOD) register of each module controls the period of the two PWM outputs. Write a value to this register to specify the number of state counts necessary for decrementing the PWM counter. The value of PWM*x_y*_PERIOD is loaded into the PWM period count register whenever the count equals zero.

7	0
PWM Period	

Bit Number	Function
7:0	PWM Period This register controls the period of the PWM outputs. The value of PWM*x_y*_PERIOD is loaded into the PWM*x_y*_COUNT register whenever the count equals zero.

Figure 10-3. PWM Period (PWM*x_y*_PERIOD) Register

10.5 PROGRAMMING THE DUTY CYCLE

The values written to the PWM_CONTROL and PWM*x_y*_PERIOD registers control the width of the high pulse, effectively controlling the duty cycle. The eight-bit value written to the control register is loaded into a buffer, and this value is used during the next period. Use the following duty cycle formula to calculate a desired duty cycle for a given value of PWM_CONTROL, and then write the value to the appropriate register.

$$\text{Duty Cycle (in \%)} = \frac{\text{PWM\_CONTROL}}{256} \times 100$$

$$\text{Pulse width (in } \mu\text{s)} = \frac{\text{Duty Cycle} \times T_{PWM}}{100}$$

where:

PWM_CONTROL = eight-bit decimal value to load into the PWM_CONTROL register

T_{PWM} = output period on the PWM pin, in µs

PWM*x*_CONTROL, PWM*y*_CONTROL
***x* = 0, 2, 4, 6**
***y* = 1, 3, 5, 7**

Address: See Figure 10-2 on page 10-5
Reset: 00H

The PWM control (PWM*x*_CONTROL, PWM*y*_CONTROL) registers determine the duty cycle of the PWM channel pair. A zero loaded into this register causes the PWM to output a low continuously (0% duty cycle). An FFH in this register causes the PWM to have its maximum duty cycle (99.6% duty cycle). Variables *x* and *y* represent the even- and odd-numbered members of an adjacent PWM channel pair, respectively.

7							0
PWM Duty Cycle							

Bit Number	Function
7:0	PWM Duty Cycle This register controls the PWM duty cycle. A zero loaded into this register causes the PWM to output a low continuously (0% duty cycle). An FFH in this register causes the PWM to have its maximum duty cycle (99.6% duty cycle).

Figure 10-4. PWM Control (PWM*x*_CONTROL, PWM*y*_CONTROL) Registers

10.5.1 Sample Calculations

For the purpose of our example, assume the following initial conditions:

- F_{XTAL1} = 16 MHz
- PWM_CONTROL = 8AH (138 decimal)
- PWM*x_y*_PERIOD = 4FH (79 decimal)

Using the equation on page 10-5 to calculate the desired period of the PWM output waveform, we find that:

- If the PLL circuitry is enabled, the pulsewidth is held high for 0.69 ms of the total 1.28 ms period, resulting in a duty cycle of approximately 54%.
- If the PLL circuitry is disabled, the same initial values produce a period of 2.56 ms and a pulsewidth of 1.38 ms. This also results in a duty cycle of approximately 54%.

10.5.2 Reading the Current Value of the Down-counter

You can read the PWM*x_y*_COUNT register (Figure 10-5) to find the current value of the period counter.

PWM*x_y*_COUNT
***x* = 0, 2, 4, 6**
***y* = 1, 3, 5, 7**
(read only)

Address: See Table 10-2 on page 10-3
Reset State: 00H

The PWM count (PWM*x_y*_COUNT) register provides the current value of the period counter.

7							0
PWM Count Value							

Bit Number	Function
7:0	PWM Count Value This register contains the current value of the period counter.

Figure 10-5. PWM Count (PWM*x_y*_COUNT) Register

10.5.3 Enabling the PWM Outputs

Each PWM output shares a pin with a port, so you must configure it as a special-function output signal before using the PWM function. To do so, follow this sequence:

1. Clear the corresponding bit of P11_DIR (see Table 10-4).
2. Set the corresponding bit of P11_MODE (see Table 10-4).
3. Set or clear the corresponding bit of P11_REG (see Table 10-4).

Table 10-4 shows the alternate port function along with the register setting that selects the PWM output instead of the port function.

Table 10-4. PWM Output Alternate Functions

PWM Output	Alternate Port Function	PWM Output Enabled When
PWM0	P11.0	P11_DIR.0 = 0, P11_MODE.0 = 1, P11_REG = X
PWM1	P11.1	P11_DIR.1 = 0, P11_MODE.1 = 1, P11_REG = X
PWM2	P11.2	P11_DIR.2 = 0, P11_MODE.2 = 1, P11_REG = X
PWM3	P11.3	P11_DIR.3 = 0, P11_MODE.3 = 1, P11_REG = X
PWM4	P11.4	P11_DIR.4 = 0, P11_MODE.4 = 1, P11_REG = X
PWM5	P11.5	P11_DIR.5 = 0, P11_MODE.5 = 1, P11_REG = X
PWM6	P11.6	P11_DIR.6 = 0, P11_MODE.6 = 1, P11_REG = X
PWM7	P11.7	P11_DIR.7 = 0, P11_MODE.7 = 1, P11_REG = X

10.5.4 Generating Analog Outputs

PWM modules can generate a rectangular pulse train that varies in duty cycle and period. Filtering this output will create a smooth analog signal. To make a signal swing over the desired analog range, first buffer the signal and then filter it with either a simple RC network or an active filter. Figure 10-6 is a block diagram of the type of circuit needed to create the smooth analog signal.

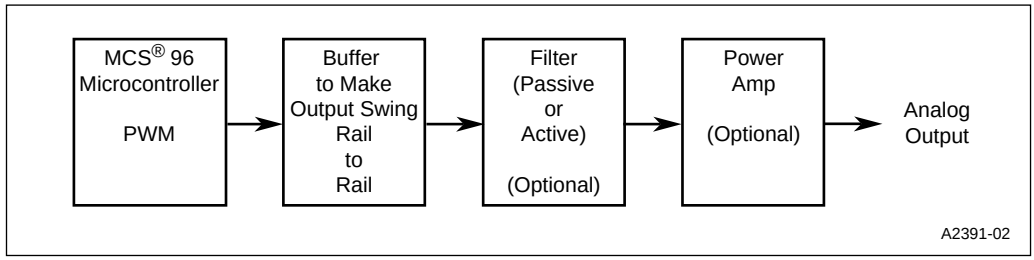


Figure 10-6. D/A Buffer Block Diagram

Figure 10-7 shows a sample circuit used for low output currents (less than 100 µA). Consider temperature and power-supply drift when selecting components for the external D/A circuitry. With proper components, a highly accurate 8-bit D/A converter can be made using the PWM.

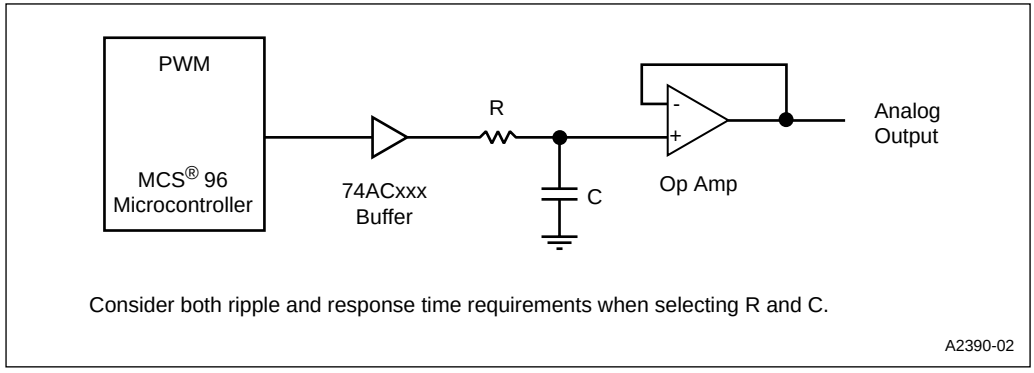


Figure 10-7. PWM to Analog Conversion Circuitry

11

Event Processor Array (EPA)

CHAPTER 11
EVENT PROCESSOR ARRAY (EPA)

Control applications often require high-speed event control. For example, the controller may need to periodically generate pulse-width modulated outputs, an analog-to-digital conversion, or an interrupt. In another application, the controller may monitor an input signal to determine the status of an external device. The event processor array (EPA) was designed to reduce the processor overhead associated with these types of event control. This chapter describes the EPA and its timers and explains how to configure and program them.

11.1 EPA FUNCTIONAL OVERVIEW

The EPA performs input and output functions associated with four timer/counters, timer 1 through timer 4 (Figure 11-1). In the input mode, the EPA monitors an input pin for an event: a rising edge, a falling edge, or an edge in either direction. When the event occurs, the EPA records the value of the timer/counter so that the event is tagged with a time. This is called an *input capture*. Input captures are buffered to allow two captures before an overrun occurs.

In the output mode, the EPA monitors a timer/counter and compares its value with a value stored in a register. When the timer/counter value matches the stored value, the EPA can trigger an event: a timer reset, an A/D conversion, or an output event (set a pin, clear a pin, toggle a pin, or take no action). This is called an *output compare*.

Each input capture or output compare sets an interrupt pending bit. This bit can optionally cause an interrupt. The EPA has 17 capture/compare channels (EPA16:0) and 8 specialized compare-only channels (OS7:0). These compare-only channels can capture the value of any other timer at the time of a compare event. Capturing the timer value while simultaneously triggering the compare event output is called *output/simulcapture*. This feature simplifies conversion between angle and time domains or between any two time bases.

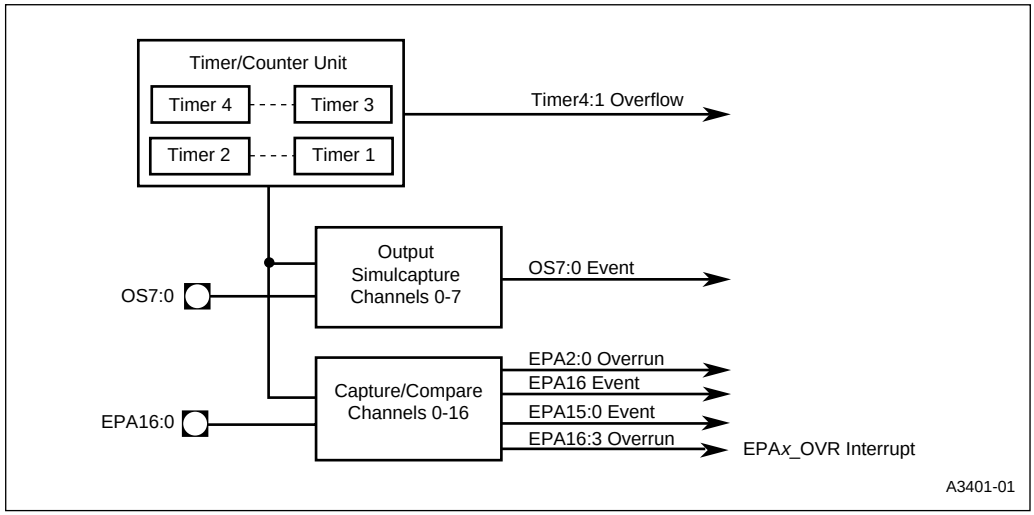


Figure 11-1. EPA Block Diagram

11.2 EPA AND TIMER/COUNTER SIGNALS AND REGISTERS

Table 11-1 describes the EPA and timer/counter input and output signals. Each signal shares a pin with a general-purpose I/O signal, as shown in the first column. Table 11-2 briefly describes the registers for EPA capture/compare channels, EPA compare-only channels, and timer/counters.

Table 11-1. EPA and Timer/Counter Signals

Port Pin	EPA Signals	EPA Signal Type	Description
P7.0	EPA0	I/O	High-speed input/output for capture/compare channel 0.
	T1CLK	I	External clock source for timer 1.
P7.1	EPA1	I/O	High-speed input/output for capture/compare channel 1.
	T1RST	I	External reset source for timer 1.
P7.2	EPA2	I/O	High-speed input/output for capture/compare channel 2.
	T2CLK	I	External clock source for timer 2.
P7.3	EPA3	I/O	High-speed input/output for capture/compare channel 3.
	T2RST	I	External reset source for timer 2.
P7.4	EPA4	I/O	High-speed input/output for capture/compare channel 4.
	T3CLK	I	External clock source for timer 3.

Table 11-1. EPA and Timer/Counter Signals (Continued)

Port Pin	EPA Signals	EPA Signal Type	Description
P7.5	EPA5	I/O	High-speed input/output for capture/compare channel 5.
	T3RST	I	External reset source for timer 3.
P7.6	EPA6	I/O	High-speed input/output for capture/compare channel 6.
	T4CLK	I	External clock source for timer 4.
P7.7	EPA7	I/O	High-speed input/output for capture/compare channel 7.
	T4RST	I	External reset source for timer 4.
P8.7:0	EPA15:8	I/O	High-speed input/output for capture/compare channels 8–15.
P10.4	EPA16	I/O	High-speed input/output for capture/compare channel 16.
P9.7:0	OS7:0	O	Outputs of compare-only (output/simulcapture) channels 0–7.

Table 11-2. EPA Control and Status Registers

Mnemonic	Address	Description
EPA0_CON EPA1_CON EPA2_CON EPA3_CON EPA4_CON EPA5_CON EPA6_CON EPA7_CON EPA8_CON EPA9_CON EPA10_CON EPA11_CON EPA12_CON EPA13_CON EPA14_CON EPA15_CON EPA16_CON	1F5CH 1F58H 1F54H 1F50H 1F4CH 1F48H 1F44H 1F40H 1F3CH 1F38H 1F34H 1F30H 1F2CH 1F28H 1F24H 1F20H 1F1CH	EPA*x* Capture/Compare Control These registers control the functions of the capture/compare channels. These registers must be addressed as words.
EPA0_TIME EPA1_TIME EPA2_TIME EPA3_TIME EPA4_TIME EPA5_TIME EPA6_TIME EPA7_TIME EPA8_TIME EPA9_TIME EPA10_TIME EPA11_TIME EPA12_TIME EPA13_TIME EPA14_TIME EPA15_TIME EPA16_TIME	1F5EH 1F5AH 1F56H 1F52H 1F4EH 1F4AH 1F46H 1F42H 1F3EH 1F3AH 1F36H 1F32H 1F2EH 1F2AH 1F26H 1F22H 1F1EH	EPA*x* Capture/Compare Time In capture mode, these registers contain the captured timer value. In compare mode, these registers contain the time at which an event is to occur. In capture mode, these registers are buffered to allow two captures before an overrun occurs. In compare mode, however, they are not buffered.
INT_MASK	0008H	Interrupt Mask Bit 2 in this register enables and disables the shared EPA*x*_OVR interrupt.
INT_PEND	0009H	Interrupt Pending Bit 2 in this register is set to indicate that the shared EPA*x*_OVR interrupt is pending.
OS0_CON OS1_CON OS2_CON OS3_CON OS4_CON OS5_CON OS6_CON OS7_CON	1EFCH 1EF8H 1EF4H 1EF0H 1EECG 1EE8H 1EE4H 1EE0H	OS*x* Compare-only Control These registers control the functions of the specialized compare-only (output/simulcapture) channels. They must be addressed as words.

Table 11-2. EPA Control and Status Registers (Continued)

Mnemonic	Address	Description
OS0_TIME OS1_TIME OS2_TIME OS3_TIME OS4_TIME OS5_TIME OS6_TIME OS7_TIME	1EFEH 1EFAH 1EF6H 1EF2H 1EEEH 1EEAH 1EE6H 1EE2H	OS*x* Compare-only Time These registers contain the time at which an event is to occur. They must be addressed as words.
P7_DIR P8_DIR P9_DIR P10_DIR	1FCAH 1FCBH 1FC2H 1FC3H	Port Direction Register Each bit controls the configuration of the corresponding pin. Clearing a bit configures a pin as a complementary output; setting a bit configures a pin as a high-impedance input or an open-drain output.
P7_MODE P8_MODE P9_MODE P10_MODE	1FC8H 1FC9H 1FC0H 1FC1H	Port Mode Register Each bit controls the mode of the corresponding pin. Setting a bit configures a pin as a special-function signal; clearing a bit configures a pin as a general-purpose I/O signal.
P7_PIN P8_PIN P9_PIN P10_PIN	1FCEH 1FCFH 1FC6H 1FC7H	Port Pin Register Each bit reflects the current state of the corresponding pin, regardless of the pin's mode and configuration.
P7_REG P8_REG P9_REG P10_REG	1FCCH 1FCDH 1FC4H 1FC5H	Port Data Output Register **For I/O Mode (P*x*_MODE.*x* = 0)** When a port pin is configured as a complementary output (P*x*_DIR.*x* = 0), setting the corresponding port data bit drives a one on the pin, and clearing the corresponding port data bit drives a zero on the pin. When a port pin is configured as a high-impedance input or an open-drain output (P*x*_DIR.*x* = 1), clearing the corresponding port data bit drives a zero on the pin, and setting the corresponding port data bit floats the pin, making it available as a high-impedance input. **For Special-function Mode (P*x*_MODE.*x* = 1)** When a port pin is configured as an output (either complementary or open-drain), the corresponding port data bit value is immaterial because the corresponding on-chip peripheral or system function controls the pin. To configure a pin as a high-impedance input, set both the P*x*_DIR and P*x*_REG bits.
PIH0_INT_MASK PIH1_INT_MASK	1E98H 1EA8H	Peripheral Interrupt Handler (PIH) Interrupt Mask The bits in these registers enable or disable each interrupt that is routed through the PIH.
PIH0_INT_PEND PIH1_INT_PEND	1E9AH 1EAAH	Peripheral Interrupt Handler (PIH) Interrupt Pending The bits in these registers are set by hardware to indicate that a PIH interrupt source is pending.

Table 11-2. EPA Control and Status Registers (Continued)

Mnemonic	Address	Description
PIH0_PTSSEL PIH1_PTSSEL	1E96H 1EA6H	Peripheral Interrupt Handler (PIH) PTS Select These registers select either a PTS interrupt service request or a standard interrupt service request for each interrupt that is routed through the PIH.
PIH0_PTSSRV PIH1_PTSSRV	1E94H 1EA4H	Peripheral Interrupt Handler (PIH) PTS Service The bits in these registers are set by hardware to request an end-of-PTS interrupt.
PIH0_VEC_BASE PIH1_VEC_BASE	1E92H 1EA2H	Peripheral Interrupt Handler (PIH) Vector Base Address These registers contain the upper ten bits of the PIH interrupt vector address.
PIH0_VEC_IDX PIH1_VEC_IDX	1E90H 1EA0H	Peripheral Interrupt Handler Vector Index Address These registers contain the number of the highest priority active PIH interrupt request.
T1CONTROL T2CONTROL T3CONTROL T4CONTROL	1F7CH 1F78H 1F74H 1F70H	Timer *x* Control This register enables and disables timer *x*, controls whether it counts up or down, optionally concatenates two timers to create a 32-bit time base (timer 2 with timer 1 or timer 4 with timer 3 only), selects the clock source and direction, and determines the clock prescaler setting.
TIMER1 TIMER2 TIMER3 TIMER4	1F7EH 1F7AH 1F76H 1F72H	Timer *x* Value This register contains the current value of timer *x*.
TIMER_MUX	1F6EH	Timer Multiplexing Control This register controls the number of timer values that can be time-multiplexed on the bus and available to the EPA channels for capture/compare.

11.3 TIMER/COUNTER FUNCTIONAL OVERVIEW

The EPA has four up/down timer/counters, timer 1 through timer 4, which can be clocked internally or externally. Each is called a *timer* if it is clocked internally and a *counter* if it is clocked externally. Figure 11-2 illustrates the timer/counter structure for timers 1 and 2. Timers 3 and 4 have an identical structure.

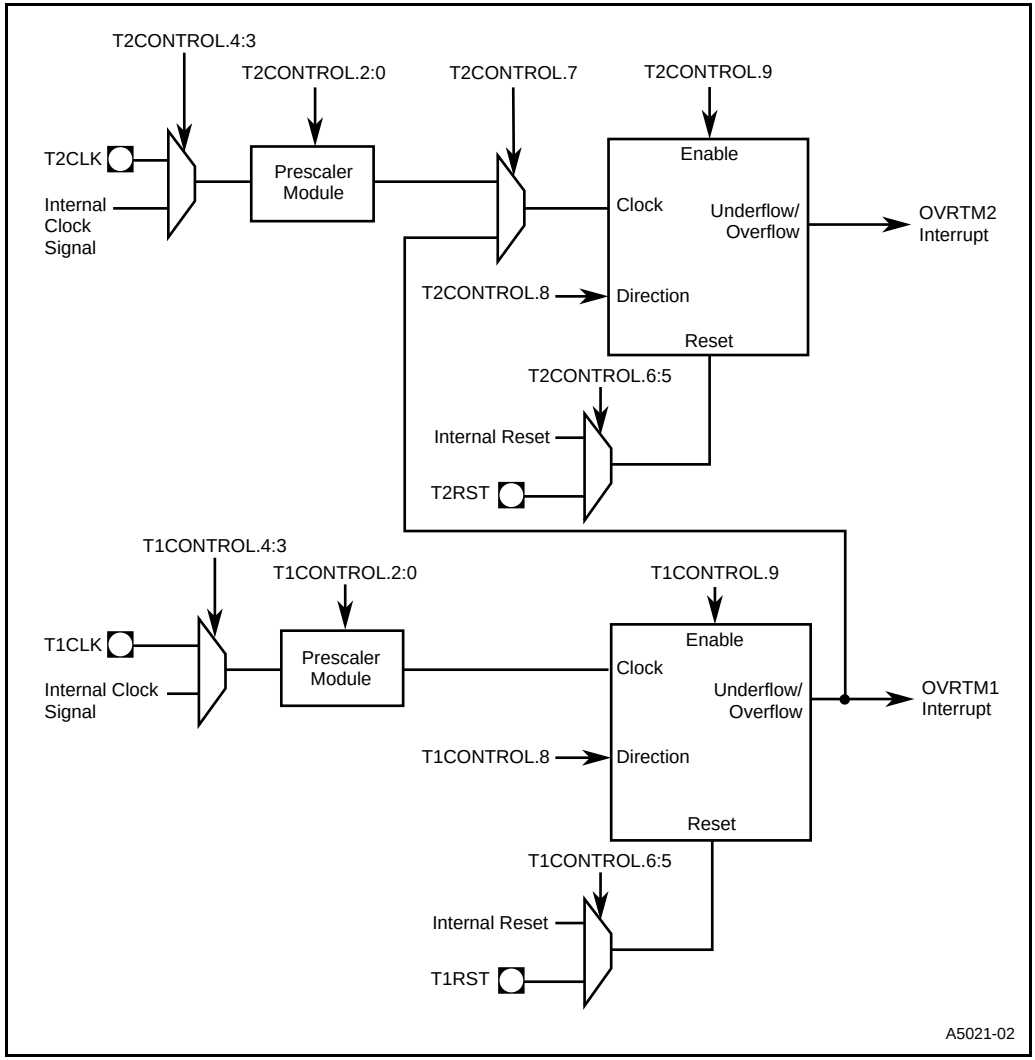


Figure 11-2. EPA Timer/Counters

The timer/counters can be used as time bases for input captures, output compares, and programmed interrupts (software timers). When a counter increments from FFFEH to FFFFH or decrements from 0001H to 0000h, the counter-overflow/underflow interrupt pending bit is set. This bit can optionally cause an interrupt. The clock source, count direction, and resolution of the input capture or output compare are all programmable (see "Programming the Timers" on page 11-15). The maximum count rate is determined by the TIMER_MUX register. With a two-state time field, the maximum count rate is one-fourth the internal clock frequency (f/4); with a four-state time field, it is one-fourth the internal clock frequency (f/8). This provides a minimum resolution for an input capture or output compare of 100 ns (at f = 40 MHz).

$$\text{resolution} = \frac{4 \times \text{prescaler\_divisor}}{f} \quad \text{or} \quad \frac{8 \times \text{prescaler\_divisor}}{f}$$

where:

4	is the multiplier for TIMER_MUX.0 = 0 (timers 1 and 2 count every two state times).
8	is the multiplier for TIMER_MUX.0 = 1 (timers 1–4 count every four state times).
prescaler_divisor	is the clock prescaler divisor from the T*x*CONTROL registers (see "Programming the Timers" on page 11-15).
f	is the internal clock frequency. See "Internal Timing" on page 2-9 for details.

11.3.1 Timer Multiplexing on the Time Bus

You can use all four timers, in which case each timer can change value every four state times, or you can use just two timers, in which case timers 1 and 2 can change value every two state times and timers 3 and 4 cannot be used. The TIMER_MUX register (Figure 11-12 on page 11-18) controls the number of timers on the time bus. Figure 11-3 illustrates the two options.

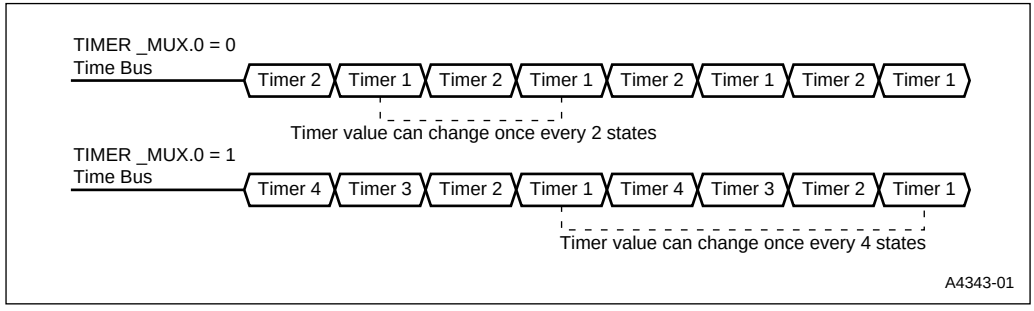


Figure 11-3. Sharing the Time Bus

11.4 EPA CHANNEL FUNCTIONAL OVERVIEW

The EPA has both programmable capture/compare and specialized compare-only channels. Each capture/compare channel can perform the following tasks. (The compare-only channels have the same functionality except that they cannot capture an external event. In addition, the compare-only channels can capture the value of a timer when they generate an output event.)

- capture the current timer value when a specified transition occurs on the EPA pin
- start an A/D conversion when an event is captured or when the timer value matches the programmed value in the event-time register
- clear, set, or toggle the EPA pin when the timer value matches the programmed value in the event-time register
- generate an interrupt when a capture or compare event occurs
- generate an interrupt when a capture overrun occurs

Each EPA channel has a control register, EPA*x*_CON; an event-time register, EPA*x*_TIME; and a timer/counter input (Figure 11-4). The control register selects the timer, the mode, and either the event that causes a timer/counter value to be captured or the event that is to occur at a given timer/counter value. The event-time register holds the captured timer value in capture mode or the event time in compare mode. See “Programming the Timers” on page 11-15 for configuration information.

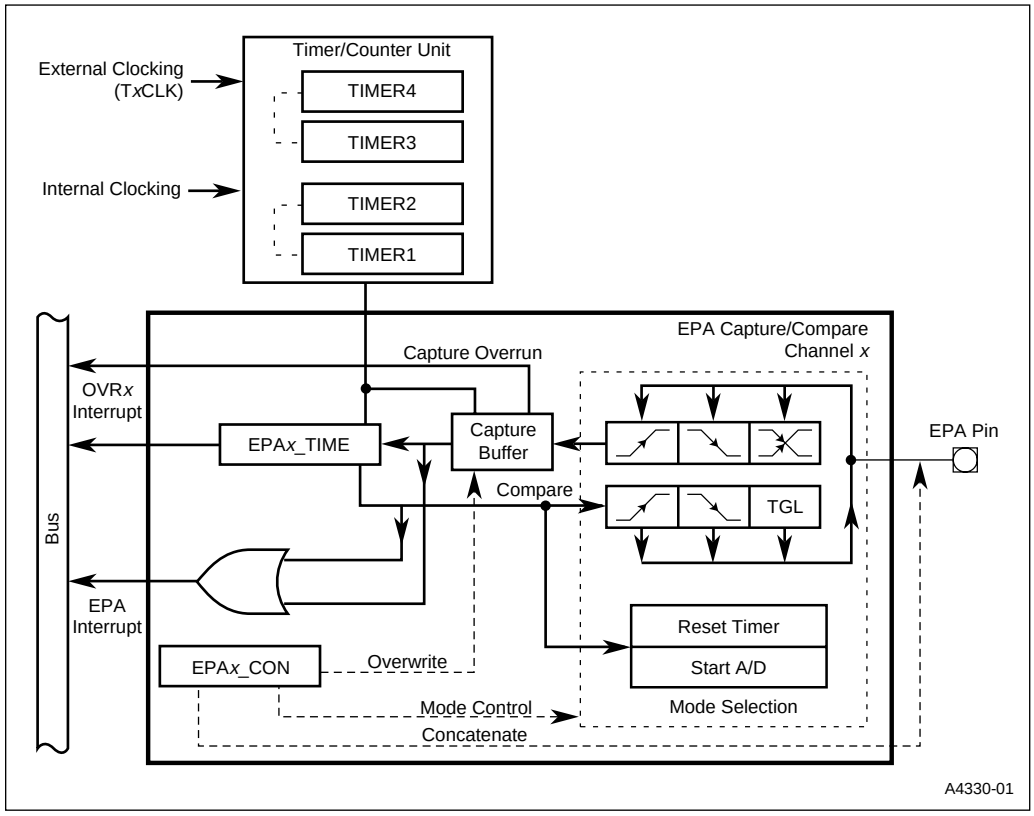


Figure 11-4. A Single EPA Capture/Compare Channel

11.4.1 Operating in Input Capture Mode

In capture mode, when a valid event occurs on the pin, the value of the selected timer is captured into a buffer. The timer value is then transferred from the buffer to the EPA*x*_TIME register, which sets the EPA interrupt pending bit as shown in Figure 11-5. If enabled, an interrupt is generated. If a second event occurs before the CPU reads the first timer value in EPA*x*_TIME, the second timer value is loaded into the buffer and held there. After the CPU reads the EPA*x*_TIME register, the value from the capture buffer is automatically transferred into EPA*x*_TIME and the EPA interrupt pending bit is set again.

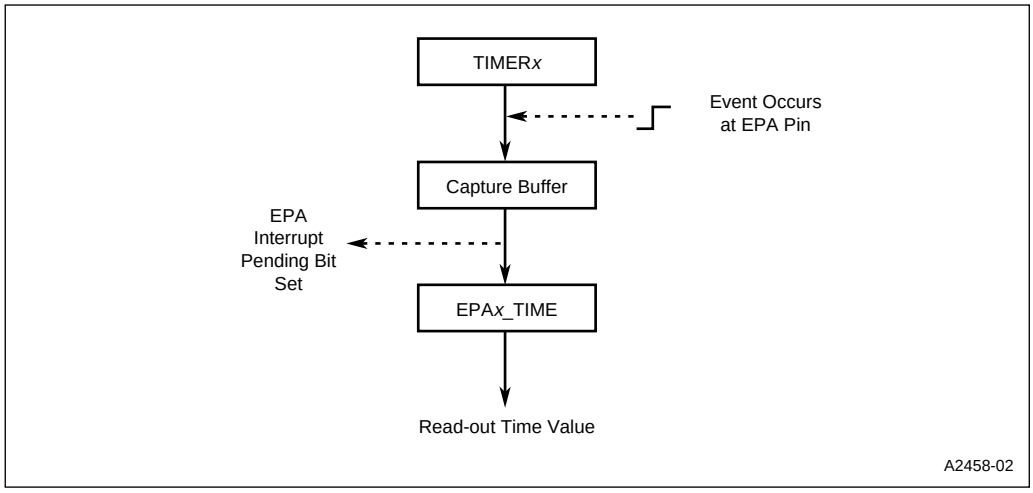


Figure 11-5. EPA Simplified Input Capture Structure

If a third event occurs before the CPU reads the event-time register, the overwrite bit (EPA*x*_CON.1) determines how the EPA will handle the event. If the bit is clear, the EPA ignores the third event. If the bit is set, the third event time overwrites the second event time in the capture buffer. Table 11-3 summarizes the possible actions when a valid event occurs.

NOTE

For an event to be captured, the signal must be stable for at least two state times both before and after the transition occurs (Figure 11-6).

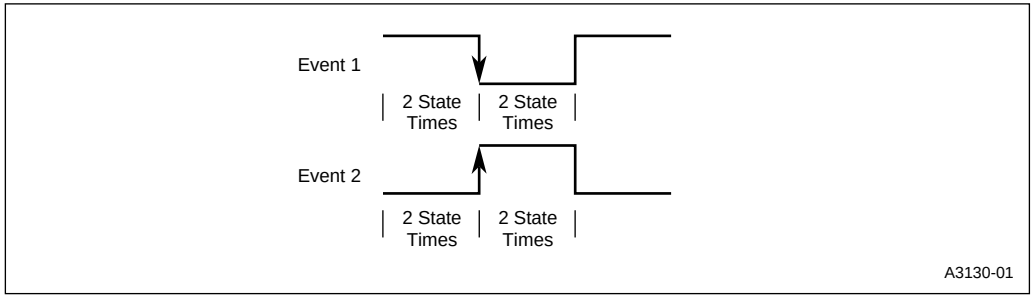


Figure 11-6. Valid EPA Input Events

Table 11-3. Action Taken When a Valid Edge Occurs

Overwrite Bit (EPA*x*_CON.1)	Status of Capture Buffer & EPA*x*_TIME	Action Taken When a Valid Edge Occurs
0	empty	Edge is captured and event time is loaded into the capture buffer and EPA*x*_TIME register.
0	full	New data is ignored — no capture, EPA interrupt, or transfer occurs.
1	empty	Edge is captured and event time is loaded into the capture buffer and EPA*x*_TIME register.
1	full	Old data is overwritten in the capture buffer.

An input capture event does not set the interrupt pending bit until the captured time value actually moves from the capture buffer into the EPA*x*_TIME register. If the buffer contains data and the PTS is used to service the interrupts, then two PTS interrupt requests will occur almost back-to-back (that is, with one instruction executed between the interrupts).

11.4.2 Operating in Output Compare Mode

When the selected timer value matches the event-time value, the action specified in the control register occurs (i.e., no output occurs or the pin is set, cleared, or toggled, or an A/D conversion is initiated). If the re-enable bit (EPA*x*_CON.5) is set, the action reoccurs on every timer match. If the re-enable bit is cleared, the action does not reoccur until a new value is written to the event-time register. See "Programming the Capture/Compare Channels" on page 11-19 for configuration information.

11.4.3 Operating in Compare Mode with the Output/Simulcapture Channels

When the selected timer value matches the event-time value, the action specified in the control register occurs (i.e., the pin is set, cleared, or toggled, or an A/D conversion is initiated, or the waveform generator is reloaded), and the value of the selected timer is captured. If the re-enable bit (OS*x*_CON.5) is set, the action reoccurs on every timer match. If the re-enable bit is cleared, the action does not reoccur until a new value is written to the event-time register. See "Programming the Timers" on page 11-15 and "Programming the Capture/Compare Channels" on page 11-19 for configuration information.

11.4.4 Generating a 32-bit Time Value

The 83C196EA's EPA module enables you to concatenate a pair of timers and a pair of EPA channels to generate or capture a 32-bit time value. Figure 11-7 illustrates this configuration using timers 1 and 2 and EPA channels 1 and 0. You can use timers 3 and 4 and any pair of adjacent channels in this same manner.

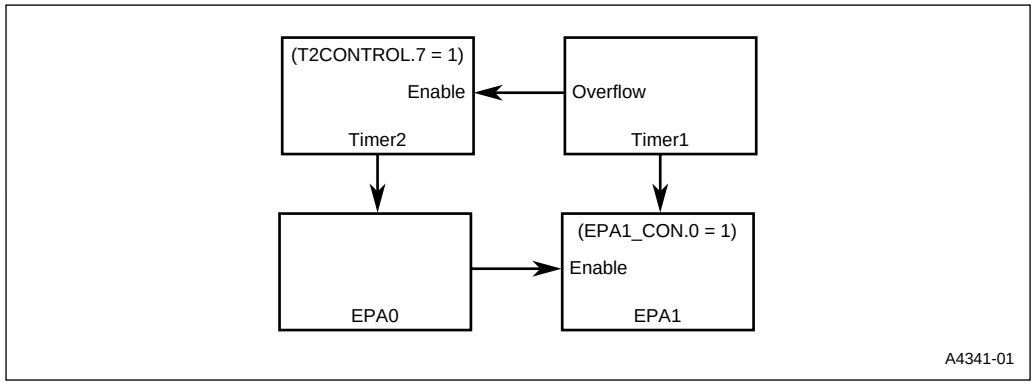


Figure 11-7. Generating a 32-bit Time Value

To set up the configuration shown in Figure 11-7, you must program six registers: T1CONTROL, T2CONTROL, TIMER1, TIMER2, EPA0_CON, and EPA1_CON. Bit 0 of EPA1_CON concatenates the EPA channels, and bit 7 of T2CONTROL concatenates the timers. (See "Programming the EPA and Timer/Counters" on page 11-14 for details.)

Once you enable timer 1 by programming the TIMER1 register, timer 1 begins counting and continues until it overflows or underflows. The overflow or underflow triggers timer 2 to count once. Figure 11-8 illustrates how timers 1 and 2 respond to a timer 1 overflow.

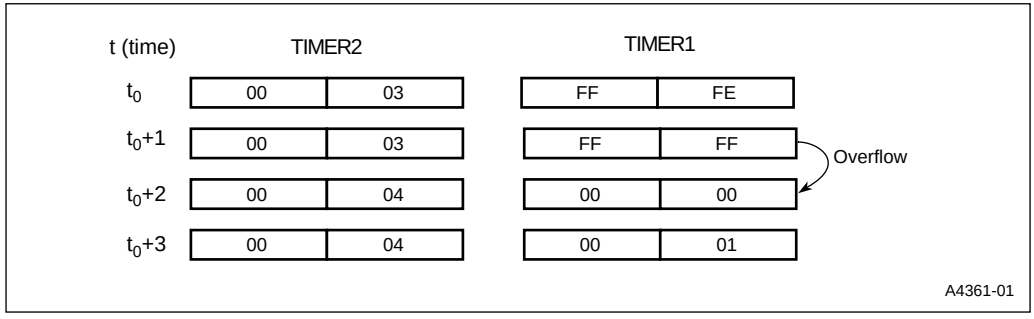


Figure 11-8. Timer 1 Overflow Occurrence

As timer 1 rolls over from FFFFH to 0000H, the overflow causes timer 2 to increment from 0003H to 0004H. Timer 2 retains the 0004H value until the next timer 1 overflow.

11.4.5 Controlling a Pair of Adjacent Pins

The EPA module enables you to configure two capture/compare or output/simulcapture channels to control both their own assigned pin and the adjacent pin. This configuration is called *cross-coupling*. Figure 11-9 illustrates this configuration using a single pair of capture/compare chan-

nels or a single pair of output/simulcapture channels. You can use any number of pairs of adjacent capture/compare or adjacent output/simulcapture channels.

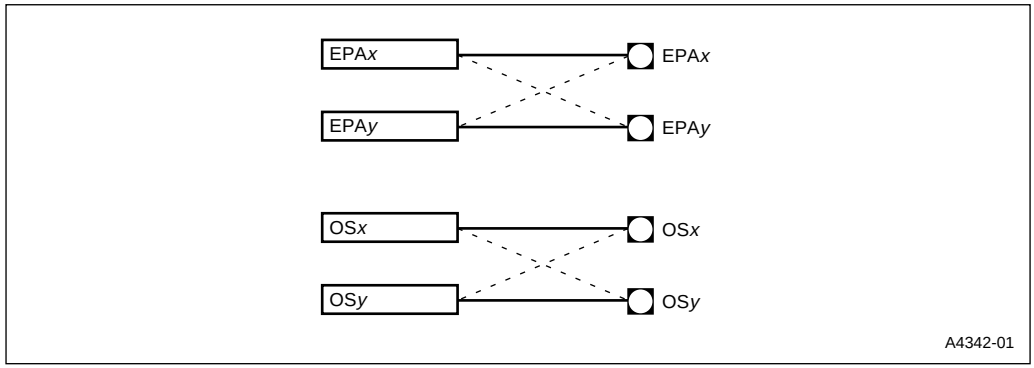


Figure 11-9. Controlling a Pair of Adjacent Pins

You might need to generate two edges that occur too close together to be generated by a single EPA channel (see Figure 11-10). To set up this configuration, you must program two EPA*x*_CON or OS*x*_CON registers for adjacent channels.

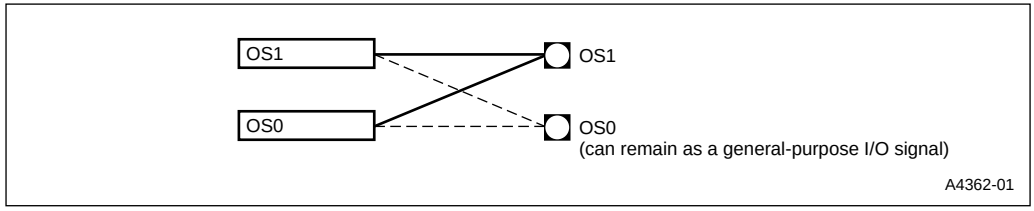


Figure 11-10. Generating Two Edges on One Pin

For example, you could configure OS0 to generate a falling edge and OS1 to generate a rising edge on the same pin. You can accomplish this by programming OS0_CON to 0600H and OS1_CON to 0440H, and using timer 1 as the time base.

11.5 PROGRAMMING THE EPA AND TIMER/COUNTERS

This section discusses configuring the port pins for the EPA and the timer/counters; describes how to program the timers, the capture/compare channels, and the compare-only (output/simulcapture) channels; and explains how to enable the EPA interrupts.

11.5.1 Configuring the EPA and Timer/Counter Signals

Before you can use the EPA, you must configure the appropriate port signals to serve as the special-function signals for the EPA and, optionally, for the timer/counter clock source and direction control signals. See “Configuring the Port Pins” on page 7-7 for information about configuring the ports.

Table 11-1 on page 11-2 lists the signals associated with the EPA and the timer/counters. Signals that are not being used for an EPA channel or timer/counter can be configured as general-purpose I/O signals.

11.5.2 Programming the Timers

The control registers for the timers are T*x*CONTROL (Figure 11-11) and TIMER_MUX (Figure 11-12). Write to these registers to configure the timers. Write to the TIMER*x* registers (Figure 11-13) to load a specific timer value.

T*x*CONTROL
***x* = 1–4**

Address: Table 11-2
Reset State: 0000H

The timer *x* control (T*x*CONTROL) register enables the associated timer/counter and specifies the counting direction, concatenation, reset source, clock source, and count rate for timer *x*.

15							8
—	—	—	—	—	—	CE	UD

7							0
ETC†	RM1	RM0	CM1	CM0	P2	P1	P0

15:10	—	Reserved; always write as zeros.
9	CE	Counter Enable This bit enables or disables the timer. From reset, the timers are disabled and not free running. 0 = disable timer/counter 1 = enable timer/counter
8	UD	Up/Down This bit specifies the timer’s counting direction. 0 = count down 1 = count up If T2CONTROL.7 is set, this bit in T1CONTROL controls the direction of both timers 2 and 1. If T4CONTROL.7 is set, this bit in T3CONTROL controls the direction of both timers 4 and 3.

† For T1CONTROL and T3CONTROL, this bit is reserved. Write zero to this bit for proper operation.

Figure 11-11. Timer *x* Control (T*x*CONTROL) Register

T*x*CONTROL (Continued)
***x* = 1–4**

Address: Table 11-2
Reset State: 0000H

The timer *x* control (T*x*CONTROL) register enables the associated timer/counter and specifies the counting direction, concatenation, reset source, clock source, and count rate for timer *x*.

15							**8**
—	—	—	—	—	—	CE	UD
7							**0**
ETC†	RM1	RM0	CM1	CM0	P2	P1	P0

7	ETC†	Enable Timer Concatenation This bit allows you to concatenate timers 2 and 1 and timers 4 and 3 to provide two 32-bit time bases. 0 = no concatenation 1 = concatenate timers Set this bit in T2CONTROL to concatenate timers 1 and 2. Set this bit in T4CONTROL to concatenate timers 3 and 4.
6:5	RM1:0	Reset Mode This bit specifies whether an output event causes a reset and, if so, the edge (falling, rising, or either) that causes the reset. **RM1 RM0 Reset** 0 0 internal reset only 0 1 reset reference timer *x* on falling edge 1 0 reset reference timer *x* on rising edge 1 1 reset reference timer *x* on falling or rising edge

† For T1CONTROL and T3CONTROL, this bit is reserved. Write zero to this bit for proper operation.

Figure 11-11. Timer *x* Control (T*x*CONTROL) Register (Continued)

T*x*CONTROL (Continued)
***x* = 1–4**

Address: Table 11-2
Reset State: 0000H

The timer *x* control (T*x*CONTROL) register enables the associated timer/counter and specifies the counting direction, concatenation, reset source, clock source, and count rate for timer *x*.

15							8
—	—	—	—	—	—	CE	UD
7							**0**
ETC†	RM1	RM0	CM1	CM0	P2	P1	P0

4:3	CM1:0	Clock Mode Select These bits specify whether the clock signal is provided by the internal frequency (f) or an external pin (T*x*CLK) **CM1 CM0 Clocking** 0 0 internal clocking (f) 0 1 external clocking on falling edge of T*x*CLK 1 0 external clocking on rising edge of T*x*CLK 1 1 external clocking on falling and rising edges of T*x*CLK
2:0	P2:0	EPA Clock Prescaler Bits These bits determine the clock prescaler value. **P2 P1 P0 Prescaler Divisor Resolution††** 0 0 0 divide by 1 (disabled) 100 ns 0 0 1 divide by 2 200 ns 0 1 0 divide by 4 400 ns 0 1 1 divide by 8 800 ns 1 0 0 divide by 16 1.6 µs 1 0 1 divide by 32 3.2 µs 1 1 0 divide by 64 6.4 µs 1 1 1 — reserved †† At f = 40 MHz and TIMER_MUX.0 = 0, use the formula on page 11-8 to calculate the resolution at other frequencies or with TIMER_MUX.0 =1.

† For T1CONTROL and T3CONTROL, this bit is reserved. Write zero to this bit for proper operation.

Figure 11-11. Timer *x* Control (T*x*CONTROL) Register (Continued)

TIMER_MUX Address: 1F6EH
Reset State: 00H

The timer multiplexing (TIMER_MUX) register controls the number of timer values that can be time-multiplexed on the bus and available to the EPA for capture/compare or. With a two-state time field, only timers 1 and 2 can be used, and each timer's maximum count rate is f/4. With a four-state time field, all four timers can be used, and each timer's maximum count rate is f/8.

7							0
—	—	—	—	—	—	—	MXS

Bit Number	Bit Mnemonic	Function
7:1	—	Reserved; always write as zeros.
0	MXS	MUX Select Selects a two-state or a four-state time field. 0 = two-state time field; timers 1 and 2 share the time bus, so each of them can count every two state times (timers 3 and 4 cannot be used) 1 = four-state time field; timers 1 through 4 share the time bus, so each timer can count every four state times

Figure 11-12. Timer/Counter Multiplexer (TIMER_MUX) Register

TIMER*x*
***x* = 1–4** Address: Table 11-2
Reset State: 0000H

This register contains the value of timer *x*. This register can be written, allowing timer *x* to be initialized to a value other than zero.

15	0
Timer Value	

Bit Number	Function
15:0	Timer Value Read the current timer *x* value from this register or write a new timer *x* value to this register.

Figure 11-13. Timer *x* Time (TIMER*x*) Registers

11.5.3 Programming the Capture/Compare Channels

The EPA*x*_CON register controls the function of its assigned capture/compare channel. To program a compare event, always write to EPA*x*_CON (Figure 11-14) first to configure the EPA capture/compare channel, and then load the event time into EPA*x*_TIME (Figure 11-15). To program a capture event, you need only write to EPA*x*_CON.

EPA*x*_CON
C*x* = 0–16

Address: See Table 11-2 on page 11-4
Reset: 0000H

The EPA control (EPA*x*_CON) registers control the functions of their assigned capture/compare channels.

15							8
—	—	—	TB1	TB0	CE	M1	M0
7							**0**
MX1	MX0	RE	AD	RTS1	RTS0	ON/RT	EMC

Bit Number	Bit Mnemonic	Function
15:13	—	Reserved; always write as zeros.
12:11	TB1:0	Time Base Select Specifies the reference timer. **TB1 TB0** 0 0 timer 1 0 1 timer 2 1 0 timer 3 1 1 timer 4
10	CE	Compare Enable Determines whether the EPA channel operates in capture or compare mode. 0 = capture mode 1 = compare mode

Figure 11-14. EPA Control (EPA*x*_CON) Registers

EPA*x*_CON (Continued)
C*x* = 0–16

Address: See Table 11-2 on page 11-4
Reset: 0000H

The EPA control (EPA*x*_CON) registers control the functions of their assigned capture/compare channels.

15							8
—	—	—	TB1	TB0	CE	M1	M0
7							**0**
MX1	MX0	RE	AD	RTS1	RTS0	ON/RT	EMC

Bit Number	Bit Mnemonic	Function
9:8	M1:0	EPA Mode Select In capture mode, specifies the type of event that triggers an input capture. In compare mode, specifies the action that the EPA executes when the reference timer matches the event time. **M1 M0 Capture Mode Event** 0 0 no capture 0 1 capture on falling edge 1 0 capture on rising edge 1 1 capture on either edge **M1 M0 Compare Mode Action** 0 0 no output 0 1 clear output pin 1 0 set output pin 1 1 toggle output pin
7:6	MX1:0	Cross-coupled Mode Select If module cross-coupling is enabled, these bits specify the action that the EPA executes on the cross-coupled pin when the reference timer matches the event-time value. **MX1 MX0 Capture Mode Event** X 0 select associated pin for input X 1 select cross-coupled pin for input **MX1 MX0 Compare Mode Action** 0 0 no output 0 1 clear cross-coupled pin 1 0 set cross-coupled pin 1 1 toggle cross-coupled pin
5	RE	Re-enable Re-enable applies to the compare mode only. It allows a compare event to continue to execute each time the event-time register (EPA*x*_TIME) matches the reference timer rather than only upon the first time match. 0 = compare function is disabled after a single event 1 = compare function always enabled

Figure 11-14. EPA Control (EPA*x*_CON) Registers (Continued)

EPA*x*_CON (Continued)
C*x* = 0–16

Address: See Table 11-2 on page 11-4
Reset: 0000H

The EPA control (EPA*x*_CON) registers control the functions of their assigned capture/compare channels.

15							8
—	—	—	TB1	TB0	CE	M1	M0
7							**0**
MX1	MX0	RE	AD	RTS1	RTS0	ON/RT	EMC

Bit Number	Bit Mnemonic	Function
4	AD	A/D Conversion Allows the EPA to start an A/D conversion that has been previously set up in the A/D command registers. To use this feature, you must select the EPA as the conversion source in the AD_COMMAND register. 0 = causes no A/D action 1 = EPA capture or compare event triggers an A/D conversion
3:2	RTS1:0	Reset Timer Select Selects the timer that is to be reset if the RT bit (EPA*x*_CON.1) is set. **RTS1 RTS0** 0 0 select timer 1 for reset 0 1 select timer 2 for reset 1 0 select timer 3 for reset 1 1 select timer 4 for reset
1	ON/RT	Overwrite New/Reset Timer The ON/RT bit functions as overwrite new in capture mode and reset timer in compare mode. **In Capture Mode (ON):** An overrun error is generated when an input capture occurs while the event-time register (EPA*x*_TIME) and its buffer are both full. When an overrun occurs, the ON bit determines whether old data is overwritten or new data is ignored: 0 = ignores new data 1 = overwrites old data in the buffer **In Compare Mode (RT):** 0 = disables the reset function 1 = resets the RTS-selected timer
0	EMC	Enable Module Concatenation Used in conjunction with TxCONTROL.8, this bit allows an EPA channel to enable an adjacent channel, providing a 32-bit time base. 0 = disable 1 = enable concatenation

Figure 11-14. EPA Control (EPA*x*_CON) Registers (Continued)

EPA*x*_TIME
***x* = 0–16**

Address: Table 11-2
Reset State: XXXXH

The EPA time (EPA*x*_TIME) registers are the event-time registers for the EPA channels. In capture mode, the value of the reference timer is captured in EPA*x*_TIME when an input transition occurs. Each event-time register is buffered, allowing the storage of two capture events at once. In compare mode, the EPA triggers a compare event when the reference timer matches the value in EPA*x*_TIME. EPA*x*_TIME is not buffered for compare mode.

15	0
EPA Time Value	

Bit Number	Function
15:0	EPA Time Value When an EPA channel is configured for capture mode, this register contains the value of the reference timer when the specified event occurred. When an EPA channel is configured for compare mode, write the compare event time to this register.

Figure 11-15. EPA Time (EPA*x*_TIME) Registers

11.5.4 Programming the Compare-only (Output/Simulcapture) Channels

To program a compare event, you must first write to the OS*x*_CON register (Figure 11-16) to configure the compare-only channel and then load the event time into OS*x*_TIME (Figure 11-17).

OS*x*_CON
***x* = 0–7**

Address: See Table 11-2 on page 11-4
Reset: 0000H

The output/simulcapture control (OSx_CON) registers control the functions of their assigned output/simulcapture channels.

15							8
—	—	—	TB1	TB0	CE	M1	M0
7							**0**
MX1	MX0	RE	AD	CS1	CS0	SE	EMC

Bit Number	Bit Mnemonic	Function
15:13	—	Reserved; always write as zeros.
12:11	TB1:0	Time Base Select Specifies the reference timer. **TB1 TB0** 0 0 timer 1 0 1 timer 2 1 0 timer 3 1 1 timer 4
10	CE	Compare Enable Enables the compare function of the output/simulcapture channel. 0 = compare function disabled 1 = compare function enabled
9:8	M1:0	Mode Select Specifies the action that the EPA executes when the reference timer matches the event-time value. **M1 M0** 0 0 no output 0 1 clear output pin 1 0 set output pin 1 1 toggle output pin
7:6	MX1:0	Cross-coupled Mode Select If module cross-coupling is enabled, this bit specifies the action that the EPA executes on the cross-coupled pin when the reference timer matches the event-time value. **MX1 MX0** 0 0 no output 0 1 clear cross-coupled pin 1 0 set cross-coupled pin 1 1 toggle cross-coupled pin

Figure 11-16. Output/Simulcapture *x* Control (OS*x*_CON) Register

OS*x*_CON (Continued)
***x* = 0–7**

Address: See Table 11-2 on page 11-4
Reset: 0000H

The output/simulcapture control (OSx_CON) registers control the functions of their assigned output/simulcapture channels.

15							8
—	—	—	TB1	TB0	CE	M1	M0
7							**0**
MX1	MX0	RE	AD	CS1	CS0	SE	EMC

Bit Number	Bit Mnemonic	Function
5	RE	Re-enable Allows a compare event to continue to execute each time the reference timer matches the event-time register value, rather than only upon the first match. 0 = compare function will drive the output only once 1 = compare function always enabled
4	AD	A/D Conversion Allows the EPA to start an A/D conversion that has been previously set up in the A/D command registers. To use this feature, you must select the EPA as the conversion source in the AD_COMMAND register. 0 = causes no A/D action 1 = OS compare event triggers an A/D conversion
3:2	CS1:0	Simulcapture Timer Select Selects the timer whose value is to be captured when the output event is generated. **CS1 CS0 Reference Timer** 0 0 timer 1 0 1 timer 2 1 0 timer 3 1 1 timer 4
1	SE	Simulcapture Enable Enables the simulcapture function of the output/simulcapture channel. 0 = simulcapture function disabled 1 = simulcapture function enabled (if compare occurs)
0	EMC	Enable Module Concatenation Used in conjunction with TxCONTROL.8, this bit allows an output/simulcapture channel to enable an adjacent channel, providing a 32-bit time base. 0 =disable 1 =enable concatenation

Figure 11-16. Output/Simulcapture *x* Control (OS*x*_CON) Register (Continued)

OS*x*_TIME
***x* = 0–7**

Address: See Table 11-2 on page 11-4
Reset: XXXXH

The output/simulcapture time registers (OS*x*_TIME) are the event-time registers for the output/simulcapture channels. The EPA triggers a compare event when the reference timer matches the value in OS*x*_TIME. If simulcapture is enabled, the EPA simultaneously captures the value of the specified timer into this register, overwriting the previous value.

15	0
OS Timer Value	

Bit Number	Function
15:0	OS Timer Value After programming the OS*x*_CON register, write the compare event time to this register. The EPA generates the programmed event when the value of the reference timer matches the event-time value in this register. If simulcapture is enabled, the EPA simultaneously captures the value of the specified timer into this register, overwriting the event-time value.

Figure 11-17. Output Simulcapture *x* Time (OS*x*_TIME) Register

11.6 ENABLING THE EPA INTERRUPTS

The EPA generates 32 individual event interrupts (EPA16:0, OS7:0, OVRTM4:1, OVR2:0) and the shared overrun event interrupt (EPA*x*_OVR) from EPA16:3. The individual interrupts are directly mapped into the two peripheral interrupt handler (PIH) interrupt pending registers (PIH0_INT_PEND and PIH1_INT_PEND). The timer overrun events from EPA16:3 share the EPA*x*_OVR interrupt, which maps into the INT_PEND register. To enable the individual interrupts, set the corresponding bits in the PIH*x*_INT_MASK registers, then set the appropriate bits in INT_MASK1 to enable either PTS or interrupt controller service for the PIH vectors. To enable the shared overrun interrupt, set the corresponding bit in the INT_MASK register. Chapter 6, “Standard and PTS Interrupts,” discusses the interrupts in greater detail.

PIH0_INT_MASK Address: 1E98H
Reset State: 0000H

The PIH0 interrupt mask (PIH0_INT_MASK) register enables or disables (masks) individual interrupt requests to peripheral interrupt handler 0. (The EI and DI instructions enable and disable servicing of all maskable interrupts.)

EPA15	EPA14	EPA13	EPA12	EPA11	EPA10	EPA9	EPA8
EPA7	EPA6	EPA5	EPA4	EPA3	EPA2	EPA1	EPA0

15:0 Setting a bit enables the corresponding interrupt request to peripheral interrupt handler 0.

Bit Mnemonic	Interrupt Description
EPA15:0	EPA Capture/Compare Channels 0–15

Figure 11-18. PIH0 Interrupt Mask (PIH0_INT_MASK) Register

PIH1_INT_MASK Address: 1EA8H
Reset State: 0000H

The PIH1 interrupt mask (PIH1_INT_MASK) register enables or disables (masks) individual interrupt requests to the peripheral interrupt handler 1. (The EI and DI instructions enable and disable servicing of all maskable interrupts.)

EPA16	OS7	OS6	OS5	OS4	OS3	OS2	OS1
OS0	OVRTM1	OVRTM2	OVRTM3	OVRTM4	OVR0	OVR1	OVR2

15:0 Setting a bit enables the corresponding interrupt request to peripheral interrupt handler 1.

Bit Mnemonic	Interrupt Description
EPA16	EPA Capture/Compare Channel 16
OS7:0	Output Simulcapture Channel 0–7
OVRTM1:4	Timer 1–4 Overflow/Underflow
OVR0:2	EPA Channel 0–2 Capture Overrun

Figure 11-19. PIH1 Interrupt Mask (PIH1_INT_MASK) Register

INT_MASK

Address: 0008H
Reset State: 00H

The interrupt mask (INT_MASK) register enables or disables (masks) individual interrupt requests. (The EI and DI instructions enable and disable servicing of all maskable interrupts.) INT_MASK is the low byte of the processor status word (PSW). PUSHF or PUSHA saves the contents of this register onto the stack and then clears this register. Interrupt calls cannot occur immediately following a push instruction. POPF or POPA restores it.

7							0
SDU	EXTINT	RI1	TI1	AD	EPA*x*_OVR	RI0	TI0

7:0	Setting a bit enables the corresponding interrupt.

Bit Mnemonic	Interrupt Description
SDU	Serial Debug Unit Interrupt
EXTINT	External Interrupt Pin
RI1	SIO1 Receive
TI1	SIO1 Transmit
AD	A/D Conversion Complete
EPA*x*_OVR	EPA Channel 3–16 Overrun
RI0	SIO0 Receive
TI0	SIO0 Transmit

Figure 11-20. Interrupt Mask (INT_MASK) Register

INT_MASK1 Address: 0013H
Reset State: 00H

The interrupt mask 1 (INT_MASK1) register enables or disables (masks) individual interrupt requests. (The EI and DI instructions enable and disable servicing of all maskable interrupts.) INT_MASK1 can be read from or written to as a byte register. PUSHA saves this register on the stack and POPA restores it.

7							0
NMI	STACK	PIH0_PTS	PIH0_INT	PIH1_PTS	PIH1_INT	SSIO1	SSIO0

7:0	Setting a bit enables the corresponding interrupt.

Bit Mnemonic	Interrupt Description
NMI†	Nonmaskable Interrupt
STACK†	Stack Overflow Error
PIH0_PTS	PIH0 PTS Service Request
PIH0_INT	PIH0 Interrupt Request
PIH1_PTS	PIH1 PTS Service Request
PIH1_INT	PIH1 Interrupt Request
SSIO1	SSIO1 Transfer
SSIO0	SSIO0 Transfer

† The NMI and stack overflow interrupts are always enabled. These nonfunctional mask bits exist for design symmetry with the INT_PEND1 register. Always write zeros to these bits.

Figure 11-21. Interrupt Mask 1 (INT_MASK1) Register

11.7 DETERMINING EVENT STATUS

In compare mode, an interrupt pending bit is set each time a match occurs on an enabled event (even if the interrupt is specifically masked in the mask register). In capture mode, an interrupt pending bit is set each time a programmed event is captured and the event time moves from the capture buffer to the EPA*x*_TIME register. If the capture buffer is full when an event occurs, an overrun interrupt pending bit is set.

Even if an interrupt is masked, software can poll the interrupt pending registers to determine whether an event has occurred.

12

Analog-to-digital (A/D) Converter

CHAPTER 12
ANALOG-TO-DIGITAL (A/D) CONVERTER

The analog-to-digital (A/D) converter can convert an analog input voltage to a digital value and set the A/D interrupt pending bit when it stores the result. It can also monitor a pin and set the A/D interrupt pending bit when the input voltage crosses over or under a programmed threshold voltage. This chapter describes the A/D converter and explains how to program it.

12.1 A/D CONVERTER FUNCTIONAL OVERVIEW

The A/D converter (Figure 12-1) can convert an analog input voltage to an 8- or 10-bit digital result and set the A/D interrupt pending bit when it stores the result. In addition, the 8XC196EA A/D features separate conversion result registers for each channel and a 16-channel auto-scanning mode that reduces the need for external multiplexing.

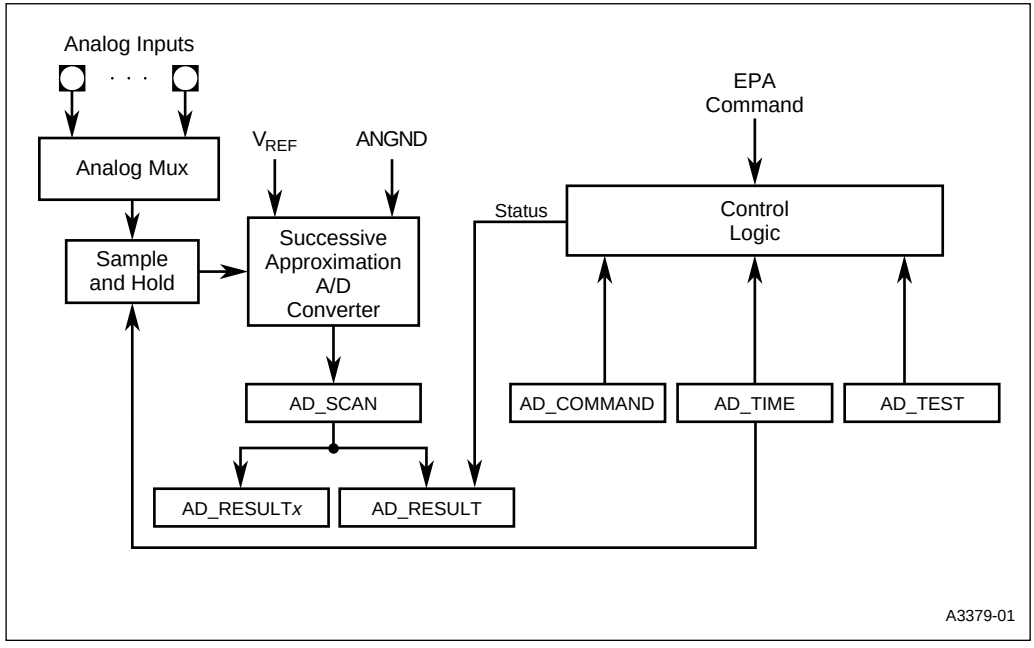


Figure 12-1. A/D Converter Block Diagram

12.2 A/D CONVERTER SIGNALS AND REGISTERS

Table 12-1 lists the A/D signals and Table 12-2 describes the control and status registers.

Table 12-1. A/D Converter Signals

Signal Name	Type	Description
ACH15:0	I	Analog Inputs Input channels to the A/D converter. See the "Voltage on Analog Input Pin" specification in the datasheet for acceptable voltage ranges.
ANGND	GND	Reference Ground Must be connected for A/D converter and port operation.
V_{REF}	PWR	Reference Voltage Must be connected for A/D converter and port operation.

Table 12-2. A/D Control and Status Registers

Mnemonic	Address	Description
AD_COMMAND	1E74H	A/D Command This register selects the A/D channel, controls whether the A/D conversion starts immediately or is triggered by the EPA, initiates automatic scan on selected channel inputs, and selects the operating mode.
AD_RESULT	1E72H	A/D Result For an A/D conversion, the high byte contains the eight MSBs from the conversion, while the low byte contains the two LSBs from a 10-bit conversion (undefined for an 8-bit conversion), indicates which A/D channel was used, and indicates whether the A/D is idle. For a threshold-detection, calculate the value for the successive approximation register and write that value to the high byte of AD_RESULT. Clear the low byte or leave it in its default state.
AD_RESULT0 AD_RESULT1 AD_RESULT2 AD_RESULT3 AD_RESULT4 AD_RESULT5 AD_RESULT6 AD_RESULT7 AD_RESULT8 AD_RESULT9 AD_RESULT10 AD_RESULT11 AD_RESULT12 AD_RESULT13 AD_RESULT14 AD_RESULT15	1E50H, 1E51H 1E52H, 1E53H 1E54H, 1E55H 1E56H, 1E57H 1E58H, 1E59H 1E5AH, 1E5BH 1E5CH, 1E5DH 1E5EH, 1E5FH 1E60H, 1E61H 1E62H, 1E63H 1E64H, 1E65H 1E66H, 1E67H 1E68H, 1E69H 1E6AH, 1E6BH 1E6CH, 1E6DH 1E6EH, 1E6FH	A/D Result on Channel *x* These registers contain the data result from an autoscan sequence. The low byte of each register contains the eight LSBs from the conversion, while the high byte contains the two MSBs from a 10-bit conversion (undefined for an 8-bit conversion).

Table 12-2. A/D Control and Status Registers (Continued)

Mnemonic	Address	Description
AD_SCAN	1E70H	A/D Scan This register selects A/D channels to be included in an autoscan sequence.
AD_TEST	1E76H	A/D Conversion Test This register enables conversions on ANGND and V_{REF} and specifies adjustments for zero-offset errors.
AD_TIME	1E77H	A/D Conversion Time This register defines the sample window time and the conversion time for each bit.
INT_MASK	0008H	Interrupt Mask The AD bit in this register enables or disables the A/D interrupt. Set the AD bit to enable the interrupt request.
INT_PEND	0009H	Interrupt Pending The AD bit in this register, when set, indicates that an A/D interrupt request is pending.

12.3 A/D CONVERTER OPERATION

An A/D conversion converts an analog input voltage to a digital value, stores the result in the AD_RESULT and AD_RESULT*x* registers, and sets the A/D interrupt pending bit. An 8-bit conversion provides 20 mV resolution, while a 10-bit conversion provides 5 mV resolution. An 8-bit conversion takes less time than a 10-bit conversion because it has two fewer bits to resolve and the comparator requires less settling time for 20 mV resolution than for 5 mV resolution.

You can convert either the voltage on an analog input channel or a test voltage. Converting the test inputs allows you to calculate the zero-offset error, and the zero-offset adjustment allows you to compensate for it. This feature can reduce or eliminate off-chip compensation hardware. Typically, you would convert the test voltages and adjust for the zero-offset error before performing conversions on an input channel. The AD_TEST register allows you to select a test voltage and program a zero-offset adjustment.

A threshold-detection compares an input voltage to a programmed reference voltage and sets the A/D interrupt pending bit when the input voltage crosses over or under the reference voltage.

A conversion can be started by a write to the AD_COMMAND register or it can be initiated by the EPA, which can provide equally spaced samples or synchronization with external events. (See “Configuring the EPA and Timer/Counter Signals” on page 11-15) The A/D scan mode allows you to perform multiple conversions and store their results in the corresponding AD_RESULT*x* registers.

Once the A/D converter receives the command to start a conversion, a delay time elapses before sampling begins. (EPA-initiated conversions begin after the capture/compare event. Immediate conversions, those initiated directly by a write to AD_COMMAND, begin within three state times after the instruction is completed.) During this *sample delay*, the hardware clears the successive approximation register and selects the designated multiplexer channel. After the sample delay, the device connects the multiplexer output to the sample capacitor for the specified sample time. After this *sample window* closes, the microcontroller disconnects the multiplexer output from the sample capacitor so that changes on the input pin will not alter the stored charge while the conversion is in progress. The device then zeros the comparator and begins the conversion.

The A/D converter uses a successive approximation algorithm to perform the analog-to-digital conversion. The converter hardware consists of a 256-resistor ladder, a comparator, coupling capacitors, and a 10-bit successive approximation register (SAR) with logic that guides the process. The resistive ladder provides 20 mV steps (V_{REF} = 5.12 volts), while capacitive coupling creates 5 mV steps within the 20 mV ladder voltages. Therefore, 1024 internal reference voltage levels are available for comparison against the analog input to generate a 10-bit conversion result. In 8-bit conversion mode, only the resistive ladder is used, providing 256 internal reference voltage levels.

The successive approximation conversion compares a sequence of reference voltages to the analog input, performing a binary search for the reference voltage that most closely matches the input. The ½ full scale reference voltage is the first tested. This corresponds to a 10-bit result where the most-significant bit is zero and all other bits are ones (0111111111B). If the analog input was less than the test voltage, bit 9 of the SAR is left at zero, and a new test voltage of ¼ full scale (0011111111B) is tried. If the analog input was greater than the test voltage, bit 8 of SAR is set. Bit 7 is then cleared for the next test (0101111111B). This binary search continues until 10 (or 8) tests have occurred, at which time the valid conversion result resides in the AD_RESULT and AD_RESULT*x* registers, where it can be read by software. The result is equal to the ratio of the input voltage divided by the analog supply voltage. If the ratio is 1.00, the result will be all ones.

12.4 PROGRAMMING THE A/D CONVERTER

The following A/D converter parameters are programmable:

- conversion input — input channel
- conversion times — sample window time and conversion time for each bit
- operating mode — 8- or 10-bit conversion or 8-bit high or low threshold detection
- conversion trigger — immediate or EPA starts

This section describes the A/D converter's registers and explains how to program them.

12.4.1 Programming the A/D Test Register

The AD_TEST register (Figure 12-2) selects either the analog input specified in AD_COMMAND or a test voltage (ANGND or V_{REF}) for conversion and specifies an offset voltage to be applied to the resistor ladder. To use the zero-offset adjustment, first perform two conversions, one on ANGND and one on V_{REF}. With the results of these conversions, use a software routine to calculate the zero-offset error. Specify the zero-offset adjustment by writing the appropriate value to AD_TEST. This offset voltage is added to the resistor ladder and applies to all input channels. “Understanding A/D Conversion Errors” on page 12-17 describes zero-offset and other errors inherent in A/D conversions.

AD_TEST

Address: 1E76H
Reset State: 00H

The A/D test (AD_TEST) register enables conversions on ANGND and V_{REF} and specifies adjustments for DC offset errors. Its functions allow you to perform two conversions, one on ANGND and one on V_{REF}. With these results, a software routine can calculate the offset and gain errors.

7							0
—	—	—	OFF2	OFF1	OFF0	TV	TE

Bit Number	Bit Mnemonic	Function
7:5	—	Reserved; for compatibility with future devices, write zeros to these bits.
4:2	OFF2:0	Offset These bits allow you to set the zero-offset point. **OFF2 OFF1 OFF0** 0 0 0 no adjustment 0 0 1 add 2.5 mV 0 1 0 subtract 2.5 mV 0 1 1 subtract 5.0 mV 1 0 0 no adjustment 1 0 1 add 5.0 mV 1 1 0 subtract 7.5 mV 1 1 1 subtract 10 mV

Figure 12-2. A/D Test (AD_TEST) Register

AD_TEST(Continued) Address: 1E76H
Reset State: 00H

The A/D test (AD_TEST) register enables conversions on ANGND and V_{REF} and specifies adjustments for DC offset errors. Its functions allow you to perform two conversions, one on ANGND and one on V_{REF}. With these results, a software routine can calculate the offset and gain errors.

7							0
—	—	—	OFF2	OFF1	OFF0	TV	TE

Bit Number	Bit Mnemonic	Function
1	TV	Test Voltage This bit selects the test voltage for a test mode conversion. (The TE bit must be set to enable test mode.) 0 = ANGND 1 = V_{REF}
0	TE	Test Enable This bit determines whether normal or test mode conversions will be performed. A normal conversion converts the analog signal input on one of the analog input channels. A test conversion allows you to perform a conversion on ANGND or V_{REF}, selected by the TV bit. 0 = normal 1 = test

Figure 12-2. A/D Test (AD_TEST) Register (Continued)

12.4.2 Programming the A/D Result Register (for Threshold Detection Only)

To use the threshold-detection modes, you must first write a value to the high byte of AD_RESULT (Figure 12-3) to set the desired reference (threshold) voltage.

AD_RESULT (Write)

Address: 1E72H
Reset State: 0000H

The high byte of the A/D result (AD_RESULT) register can be written to set the reference voltage for the A/D threshold-detection modes.

15							8
REFV7	REFV6	REFV5	REFV4	REFV3	REFV2	REFV1	REFV0
7							**0**
—	—	—	—	—	—	—	—

Bit Number	Bit Mnemonic	Function
15:8	REFV7:0	Reference Voltage These bits specify the threshold value. This reference voltage is compared with an analog input pin. When the voltage on the analog input pin crosses over (detect high) or under (detect low) the threshold value, the A/D conversion complete interrupt pending bit is set. Use the following formula to determine the value to write to this register for a given threshold voltage. $\text{reference voltage} = \frac{\text{desired threshold voltage} \times 256}{V_{REF} - ANGND}$
7:0	—	Reserved; for compatibility with future devices, write zeros to these bits.

Figure 12-3. A/D Result (AD_RESULT) Register — Write Format

12.4.3 Programming the A/D Time Register

Two parameters, sample time and conversion time, control the time required for an A/D conversion. The sample time is the length of time that the analog input voltage is actually connected to the sample capacitor. If this time is too short, the sample capacitor will not charge completely. If the sample time is too long, the input voltage may change and cause conversion errors. The conversion time is the length of time required to convert the analog input voltage stored on the sample capacitor to a digital value. The conversion time must be long enough for the comparator and circuitry to settle and resolve the voltage. Excessively long conversion times allow the sample capacitor to discharge, degrading accuracy.

The AD_TIME register (Figure 12-4) specifies the A/D sample and conversion times. To avoid erroneous conversion results, use the T_{SAM} and T_{CONV} specifications in the datasheet to determine appropriate values.

AD_TIME

Address: 1E77H
Reset State: FFH

The A/D time (AD_TIME) register programs the sample window time and the conversion time for each bit. This register programs the speed at which the A/D can run — not the speed at which it can convert correctly. Consult the data sheet for recommended values. Initialize the AD_TIME register before initializing the AD_COMMAND register. Do not write to this register while a conversion is in progress; the results are unpredictable.

7							0
SAM2	SAM1	SAM0	CONV4	CONV3	CONV2	CONV1	CONV0

Bit Number	Bit Mnemonic	Function
7:5	SAM2:0	A/D Sample Time These bits specify the sample time. Use the following formula to compute the sample time. $SAM = \frac{T_{SAM} \times f - 2}{8}$ where: SAM = 1 to 7 T_{SAM} = the sample time, in µs, from the data sheet f = the internal operating frequency, in MHz
4:0	CONV4:0	A/D Bit Conversion Time These bits specify the conversion time for each bit. Use the following formula to compute the conversion time. $CONV = \left[\frac{T_{CONV} \times f - 3}{2 \times B}\right] - 1$ where: CONV = 2 to 31 T_{CONV} = the conversion time, in µs, from the data sheet f = the internal operating frequency, in MHz B = the number of bits to be converted (8 or 10)

Figure 12-4. A/D Time (AD_TIME) Register

12.4.4 Programming the A/D Command Register

The A/D command register controls the operating mode, the analog input channel, and the conversion trigger.

AD_COMMAND Address: 1E74H
Reset State: 00H

The A/D command (AD_COMMAND) register selects the A/D channel number to be converted, controls whether the A/D converter starts immediately or with an EPA command, initiates automatic scan on selected A/D channel inputs, and selects the conversion mode.

7							0
SCAN	M1	M0	GO	ACH3	ACH2	ACH1	ACH0

Bit Number	Bit Mnemonic	Function
7	SCAN	A/D Scan This bit enables and disables automatic scanning. 0 = disable scan 1 = enable scan
6:5	M1:0	A/D Mode† These bits determine the A/D mode. **M1 M0 Mode** 0 0 10-bit conversion 0 1 8-bit conversion 1 0 threshold detect low 1 1 threshold detect high
4	GO	A/D Conversion Trigger†† Writing this bit arms the A/D converter. The value that you write to it determines at what point a conversion is to start. 0 = EPA initiates conversion 1 = start immediately
3:0	ACH3:0	A/D Channel Selection Write the A/D conversion channel number to these bits. The 83C196EA has 16 A/D channel inputs, numbered 0–15.

† While a threshold-detection mode is selected for an analog input pin, no other conversion can be started. If another value is loaded into AD_COMMAND, the threshold-detection mode is disabled and the new command is executed.

†† It is the act of writing to the GO bit, rather than its value, that starts a conversion. Even if the GO bit has the desired value, you must set it again to start a conversion immediately or clear it again to arm it for an EPA-initiated conversion. The GO bit must be set to 0 for A/D SCAN to work.

Figure 12-5. A/D Command (AD_COMMAND) Register

12.4.5 Programming the A/D Scan Register

The A/D can operate in a number of conversion modes. On the 8XC196EA, the A/D can operate in an automatic scan mode. In automatic scan mode, each channel is sampled automatically and the conversion result is loaded into the corresponding result register, AD_RESULT*x*, and AD_RESULT. The AD_SCAN register (Figure 12-6) allows you to include or exclude individual channels from the scan sequence. The scan sequence begins from the least-significant channel number and continues to the most-significant channel number.

AD_SCAN

Address: 1E70H
Reset State: 0000H

The A/D scan (AD_SCAN) register allows you to select individual channels to include in the scan sequence and those to exclude. The automatic scan begins with the least-significant channel and ends with the most-significant channel.

15							8
ACH15	ACH14	ACH13	ACH12	ACH11	ACH10	ACH9	ACH8
7							**0**
ACH7	ACH6	ACH5	ACH4	ACH3	ACH2	ACH1	ACH0

Bit Number	Bit Mnemonic	Function
15:0	ACH15:0	A/D Channel *x* These bits select the channels to be included in the automatic scan. 0 = exclude A/D channel *x* from scan sequence 1 = include A/D channel *x* in scan sequence

Figure 12-6. A/D Scan (AD_SCAN) Register

12.4.6 Enabling the A/D Interrupt

The A/D converter can set the A/D interrupt pending bit when it completes a conversion or when the input voltage crosses the threshold value in the selected direction. To enable the interrupt, set the corresponding mask bit in the interrupt mask register (see INT_MASK on page C-33) and execute the EI instruction to globally enable servicing of interrupts. See Chapter 6, "Standard and PTS Interrupts," for details about interrupts.

12.5 DETERMINING A/D STATUS AND CONVERSION RESULTS

You can read the status bit of the AD_RESULT register (Figure 12-7) to determine the status of the A/D converter. The AD_RESULT register is cleared when a new conversion is started; therefore, to prevent losing data, you must read both bytes before a new conversion starts. However, the AD_RESULT*x* register retains the individual A/D conversion result until the next sampling of A/D channel *x*. If you read AD_RESULT register before the conversion is complete, the result is not guaranteed to be accurate. The AD_RESULTx register can be read at any time.

The conversion result is the ratio of the input voltage to the reference voltage:

$$\text{RESULT (8-bit)} = 255 \times \frac{V_{IN} - \text{ANGND}}{V_{REF} - \text{ANGND}} \qquad \text{RESULT (10-bit)} = 1023 \times \frac{V_{IN} - \text{ANGND}}{V_{REF} - \text{ANGND}}$$

You can also read the interrupt pending register (see INT_PEND on page C-35) to determine the status of the A/D interrupt.

AD_RESULT (Read)

Address: 1E72H
Reset State: 0000H

The A/D result (AD_RESULT) register consists of two bytes. The high byte contains the eight most-significant bits from the A/D converter. The low byte contains the two least-significant bits from a 10-bit A/D conversion, indicates the A/D channel number that was used for the conversion, and indicates whether a conversion is currently in progress.

15							8
ADRLT9	ADRLT8	ADRLT7	ADRLT6	ADRLT5	ADRLT4	ADRLT3	ADRLT2
7							**0**
ADRLT1	ADRLT0	—	STATUS	ACH3	ACH2	ACH1	ACH0

Bit Number	Bit Mnemonic	Function
15:6	ADRLT9:0	A/D Result These bits contain the A/D conversion result.
5	—	Reserved. This bit is undefined.
4	STATUS	A/D Status Indicates the status of the A/D converter. Up to 8 state times are required to set this bit following a start command. When testing this bit, wait at least the 8 state times. 0 = A/D is idle 1 = A/D conversion is in progress
3:0	ACH3:0	A/D Channel Number These bits indicate the A/D channel number that was used for the conversion. The 8XC196EA has 16 A/D channel inputs, numbered 0–15.

Figure 12-7. A/D Result (AD_RESULT) Register — Read Format

AD_RESULT*x* (Read)

Address: Table 12-2
Reset State: 0000H

The individual result (AD_RESULT*x*) registers contain the result from the A/D channel *x* conversion.The low byte contains the eight LSBs from the conversion, while the high byte contains the two MSBs from a 10-bit conversion (undefined for an 8-bit conversion).

15							8
—	—	—	—	—	—	ADRLT9	ADRLT8
7							**0**
ADRLT7	ADRLT6	ADRLT5	ADRLT4	ADRLT3	ADRLT2	ADRLT1	ADRLT0

Bit Number	Bit Mnemonic	Function
15:10	—	Reserved. These bits are undefined.
9:0	ADRLT9:0	A/D Result on Channel *x*. These bits contain the A/D conversion result.

Figure 12-8. A/D Result *x* (AD_RESULT*x*) Register — Read Format

12.6 DESIGN CONSIDERATIONS

This section describes considerations for the external interface circuitry and describes the errors that can occur in any A/D converter. The datasheet lists the *absolute error* specification, which includes all deviations between the actual conversion process and an ideal converter. However, because the various components of error are important in many applications, the datasheet also lists the specific error components. This section describes those components. For additional information and design techniques, consult AP-406, *MCS® 96 Analog Acquisition Primer* (order number 270365). Application note AP-406 is also included in the *Embedded Microcontrollers* handbook.

12.6.1 Designing External Interface Circuitry

The external interface circuitry to an analog input is highly dependent upon the application and can affect the converter characteristics. Factors such as input pin leakage, sample capacitor size, and multiplexer series resistance from the input pin to the sample capacitor must be considered in the external circuit's design. These factors are idealized in Figure 12-9.

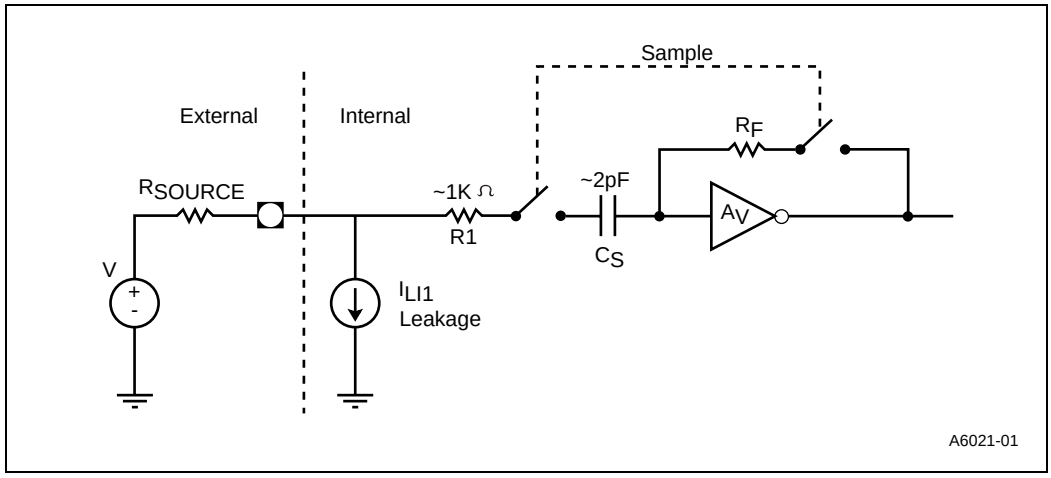


Figure 12-9. Idealized A/D Sampling Circuitry

During the sample window, the external input circuit must be able to charge the sample capacitor (C_S) through the series combination of the input source resistance (R_{SOURCE}), the input series resistance (R_1), and the comparator feedback resistance (R_F). The total effective series resistance (R_T) is calculated using the following formula, where A_V is the gain of the comparator circuit.

$$R_T = R_{SOURCE} + R_1 + \frac{R_F}{A_V + 1}$$

Typically, the (R_F / A_V + 1) term is the major contributor to the total resistance and the factor that determines the minimum sample time specified in the datasheet.

12.6.1.1 Minimizing the Effect of High Input Source Resistance

Under some conditions, the input source resistance (R_{SOURCE}) can be great enough to affect the measurement. You can minimize this effect by increasing the sample time or by connecting an external capacitor (C_{EXT}) from the input pin to ANGND. The external signal will charge C_{EXT} to the source voltage level. When the channel is sampled, C_{EXT} acts as a low-impedance source to charge the sample capacitor (C_S). A small portion of the charge in C_{EXT} is transferred to C_S, resulting in a drop of the sampled voltage. The voltage drop is calculated using the following formula.

$$\text{Sampled Voltage Drop, \%} = \frac{C_S}{C_{EXT} + C_S} \times 100\%$$

If C_{EXT} is 0.005 µF or greater, the error will be less than –0.4 LSB in 10-bit conversion mode. The use of C_{EXT} in conjunction with R_{SOURCE} forms a low-pass filter that reduces noise input to the A/D converter.

High R_{SOURCE} resistance can also cause errors due to the input leakage (I_{LI1}). I_{LI1} is typically much lower than its specified maximum (consult the datasheet for specifications). The combined effect of I_{LI1} leakage and high R_{SOURCE} resistance is calculated using the following formula.

$$\text{error (LSBs)} = \frac{R_{SOURCE} \times I_{LI1} \times 1024}{V_{REF}}$$

where:

R_{SOURCE}	is the input source resistance, in ohms
I_{LI1}	is the input leakage, in amperes
V_{REF}	is the reference voltage, in volts

External circuits with R_{SOURCE} resistance of 1 kΩ or lower and V_{REF} equal to 5.0 volts will have a resultant error due to source impedance of 0.6 LSB or less.

12.6.1.2 Suggested A/D Input Circuit

The suggested A/D input circuit shown in Figure 12-10 provides limited protection against overvoltage conditions on the analog input. Should the input voltage be driven significantly below ANGND or above V_{REF}, diode D2 or D1 will forward bias at about 0.8 volts. The device's input protection begins to turn on at approximately 0.5 volts beyond ANGND or V_{REF}. The 270Ω resistor limits the current input to the analog input pin to a safe value, less than 1 mA.

NOTE

Driving any analog input more than 0.5 volts beyond ANGND or V_{REF} begins to activate the input protection devices. This drives current into the internal reference circuitry and substantially degrades the accuracy of A/D conversions on all channels.

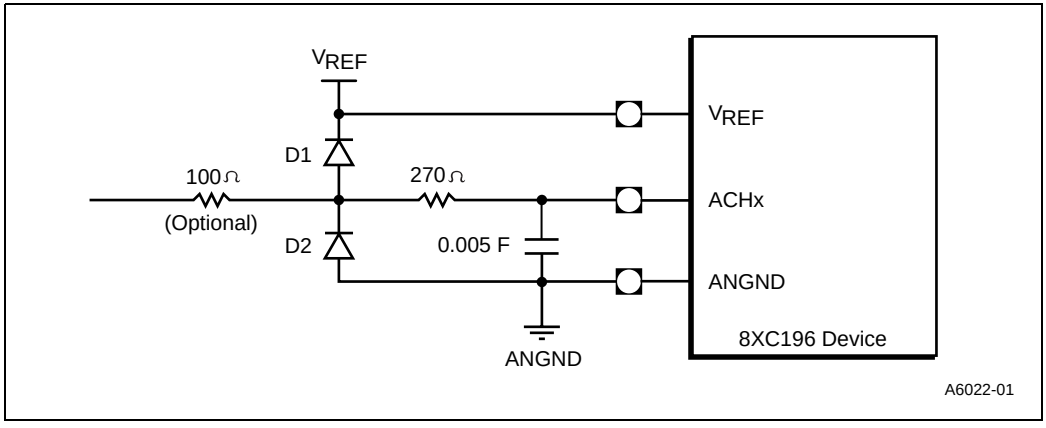


Figure 12-10. Suggested A/D Input Circuit

12.6.1.3 Analog Ground and Reference Voltages

Reference supply levels strongly influence the absolute accuracy of the conversion. For this reason, we recommend that you tie the ANGND pin to the V_{SS} pin as close to the device as possible, using a minimum trace length. In a noisy environment, we highly recommend the use of a separate analog ground plane that connects to V_{SS} at a single point as close to the device as possible. I_{REF} may vary between 2 mA and 5 mA during a conversion. To minimize the effect of this fluctuation, mount a 1.0 µF ceramic or tantalum bypass capacitor between V_{REF} and ANGND, as close to the device as possible.

ANGND should be within about ± 50 mV of V_{SS}. V_{REF} should be well regulated and used only for the A/D converter. The V_{REF} supply can be between 4.5 and 5.5 volts and must be able to source approximately 5 mA (see the datasheet for actual specifications). V_{REF} should be approximately the same voltage as V_{CC}. V_{REF} and V_{CC} should power up at the same time, to avoid potential latch-up conditions on V_{REF}. Large negative current spikes on the ANGND pin relative to V_{SS} may cause the analog circuitry to latch up. This is an additional reason to follow careful grounding practice.

The analog reference voltage (V_{REF}) is the positive supply to which all A/D conversions are compared. If high accuracy is not required, V_{REF} can be tied to V_{CC}. If accuracy is important, V_{REF} must be very stable. One way to accomplish this is through the use of a precision power supply or a separate voltage regulator (usually an IC). These devices must be referenced to ANGND, **not** to V_{SS}, to ensure that V_{REF} tracks ANGND and not V_{SS}.

12.6.2 Understanding A/D Conversion Errors

The conversion result is the ratio of the input voltage to the reference voltage.

$$\text{RESULT (8-bit)} = 255 \times \frac{V_{IN} - \text{ANGND}}{V_{REF} - \text{ANGND}} \qquad \text{RESULT (10-bit)} = 1023 \times \frac{V_{IN} - \text{ANGND}}{V_{REF} - \text{ANGND}}$$

This ratio produces a stair-stepped *transfer function* when the output code is plotted versus input voltage. The resulting digital codes can be taken as simple ratiometric information, or they provide information about absolute voltages or relative voltage changes on the inputs.

The more demanding the application, the more important it is to fully understand the converter's operation. For simple applications, knowing the *absolute error* of the converter is sufficient. However, closing a servo-loop with analog inputs requires a detailed understanding of an A/D converter's operation and errors.

In many applications, it is less critical to record the absolute accuracy of an input than it is to detect that a change has occurred. This approach is acceptable as long as the converter is *monotonic* and has *no missing codes*. That is, increasing input voltages produce adjacent, unique output codes that are also increasing. Decreasing input voltages produce adjacent, unique output codes that are also decreasing. In other words, there exists a unique input voltage range for each 10-bit output code that produces that code only, with a repeatability of typically ± 0.25 LSBs (1.5 mV).

The inherent errors in an analog-to-digital conversion process are quantizing error, zero-offset error, full-scale error, differential nonlinearity, and nonlinearity. All of these are *transfer function* errors related to the A/D converter. In addition, temperature coefficients, V_{CC} rejection, sample-hold feedthrough, multiplexer off-isolation, channel-to-channel matching, and random noise should be considered. Fortunately, one *absolute error* specification (listed in datasheets) describes the total of all deviations between the actual conversion process and an ideal converter. However, the various components of error are important in many applications.

An unavoidable error results from the conversion of a continuous voltage to an integer digital representation. This error, called *quantizing error,* is always ± 0.5 LSB. Quantizing error is the only error seen in a perfect A/D converter, and it is obviously present in actual converters. Figure 12-11 shows the transfer function for an ideal 3-bit A/D converter.

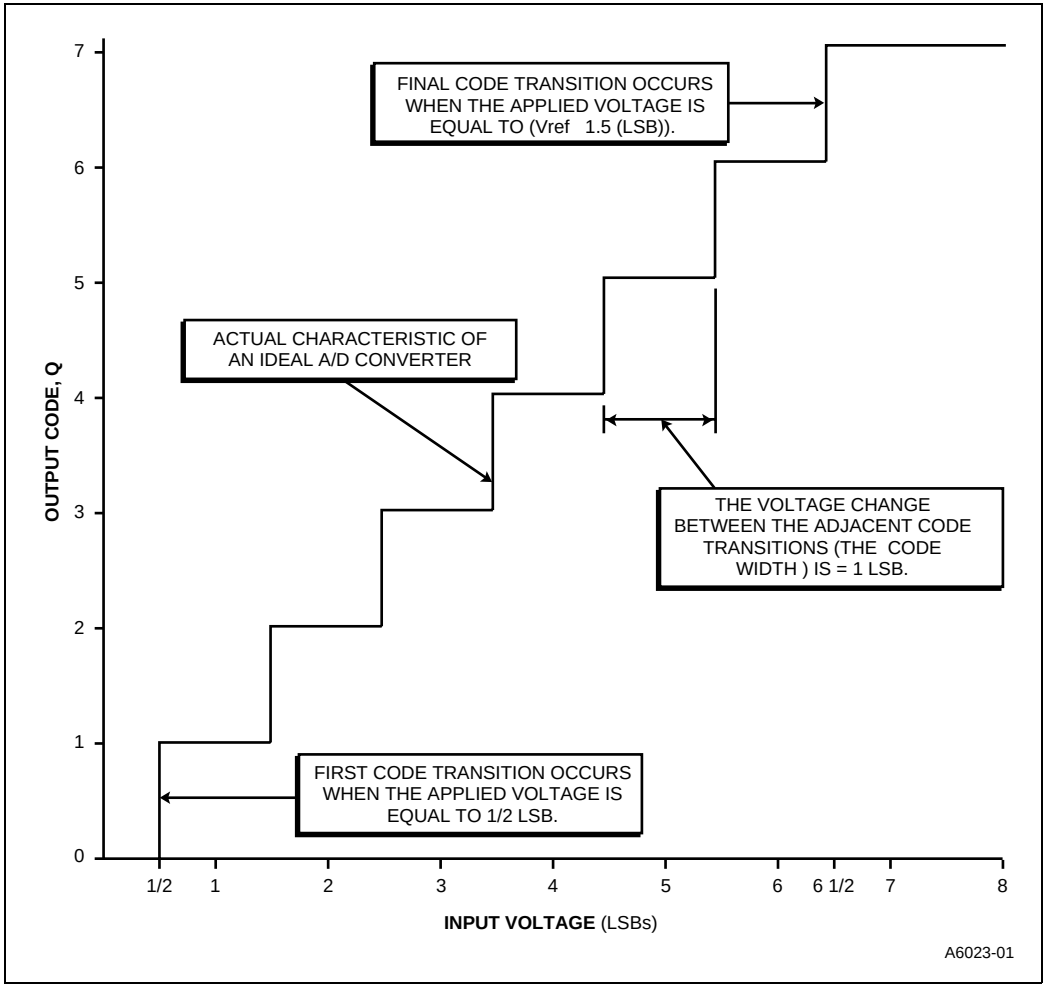


Figure 12-11. Ideal A/D Conversion Characteristic

Note that the ideal characteristic possesses unique qualities:

- its first code transition occurs when the input voltage is 0.5 LSB;
- its full-scale code transition occurs when the input voltage equals the full-scale reference voltage minus 1.5 LSB (V_{REF} – 1.5LSB); and
- its code widths are all exactly one LSB.

These qualities result in a digitization without zero-offset, full-scale, or linearity errors; in other words, a perfect conversion.

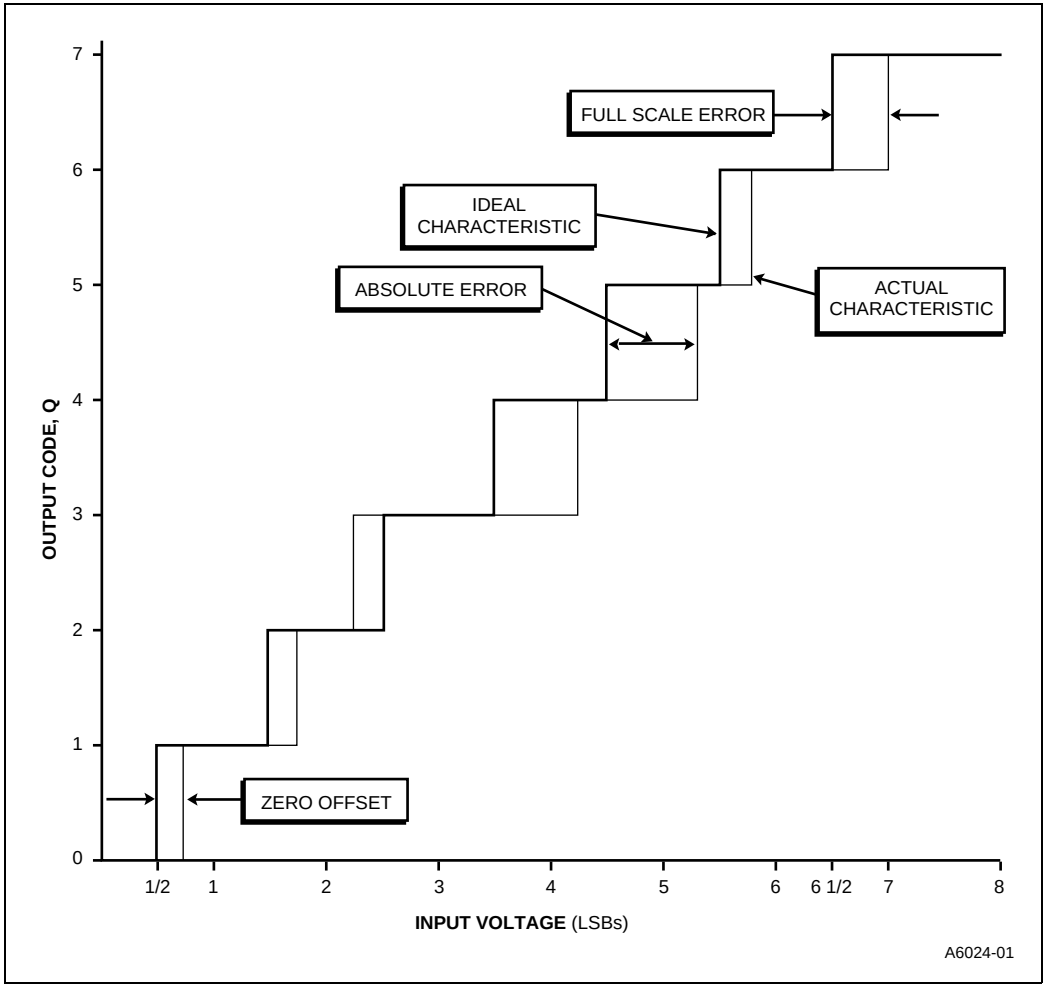


Figure 12-12. Actual and Ideal A/D Conversion Characteristics

The actual characteristic of a hypothetical 3-bit converter is not perfect. When the ideal characteristic is overlaid with the actual characteristic, the actual converter is seen to exhibit errors in the locations of the first and final code transitions and in code widths, as shown in Figure 12-12. The deviation of the first code transition from ideal is called *zero-offset* error, and the deviation of the final code transition from ideal is *full-scale* error. The deviation of a code width from ideal causes two types of errors: differential nonlinearity and nonlinearity. *Differential nonlinearity* is a measure of local code-width error, whereas *nonlinearity* is a measure of overall code-transition error.

Differential nonlinearity is the degree to which actual *code widths* differ from the ideal one-LSB width. It provides a measure of how much the input voltage may have changed in order to produce a one-count change in the conversion result. In the 10-bit converter, the code widths are ideally 5 mV (V_{REF} / 1024). If such a converter is specified to have a maximum differential nonlinearity of 2 LSBs (10 mV), the maximum code width will be no greater than 10 mV larger than ideal, or 15 mV.

Because the A/D converter has *no missing codes*, the minimum code width will always be greater than –1 (negative one). The differential nonlinearity error on a particular code width is compensated for by other code widths in the transfer function, such that 1024 unique steps occur. The actual code widths in this converter typically vary from 2.5 mV to 7.5 mV.

Nonlinearity is the worst-case deviation of *code transitions* from the corresponding code transitions of the ideal characteristic. Nonlinearity describes the extent to which differential nonlinearities can add up to produce an overall maximum departure from a linear characteristic. If the differential nonlinearity errors are too large, it is possible for an A/D converter to miss codes or to exhibit non-monotonic behavior. Neither behavior is desirable in a closed-loop system. A converter has *no missing codes* if there exists for each output code a unique input voltage range that produces that code only. A converter is *monotonic* if every subsequent code change represents an input voltage change in the same direction.

Differential nonlinearity and nonlinearity are quantified by measuring the terminal-based linearity errors. A terminal-based characteristic results when an actual characteristic is translated and scaled to eliminate zero-offset and full-scale error, as shown in Figure 12-13. The terminal-based characteristic is similar to the actual characteristic that would result if zero-offset and full-scale error were externally trimmed away. In practice, this is done by using input circuits that include gain and offset trimming. In addition, V_{REF} could also be closely regulated and trimmed within the specified range to affect full-scale error.

Other factors that affect a real A/D converter system include temperature drift, failure to completely reject unwanted signals, multiplexer channel dissimilarities, and random noise. Fortunately, these effects are small. *Temperature drift* is the rate at which typical specifications change with a change in temperature. These changes are reflected in the *temperature coefficients*. Unwanted signals come from three main sources: noise on V_{CC}, input signal changes on the channel being converted (after the sample window has closed), and signals applied to channels not selected by the multiplexer. The effects of these unwanted signals are specified as *Vcc rejection*, *off-isolation*, and *feedthrough*, respectively. Finally, multiplexer on-channel resistances differ slightly from one channel to the next, which causes *channel-to-channel matching* errors and *repeatability* errors. Differences in DC leakage current from one channel to another and random noise in general contribute to repeatability errors.

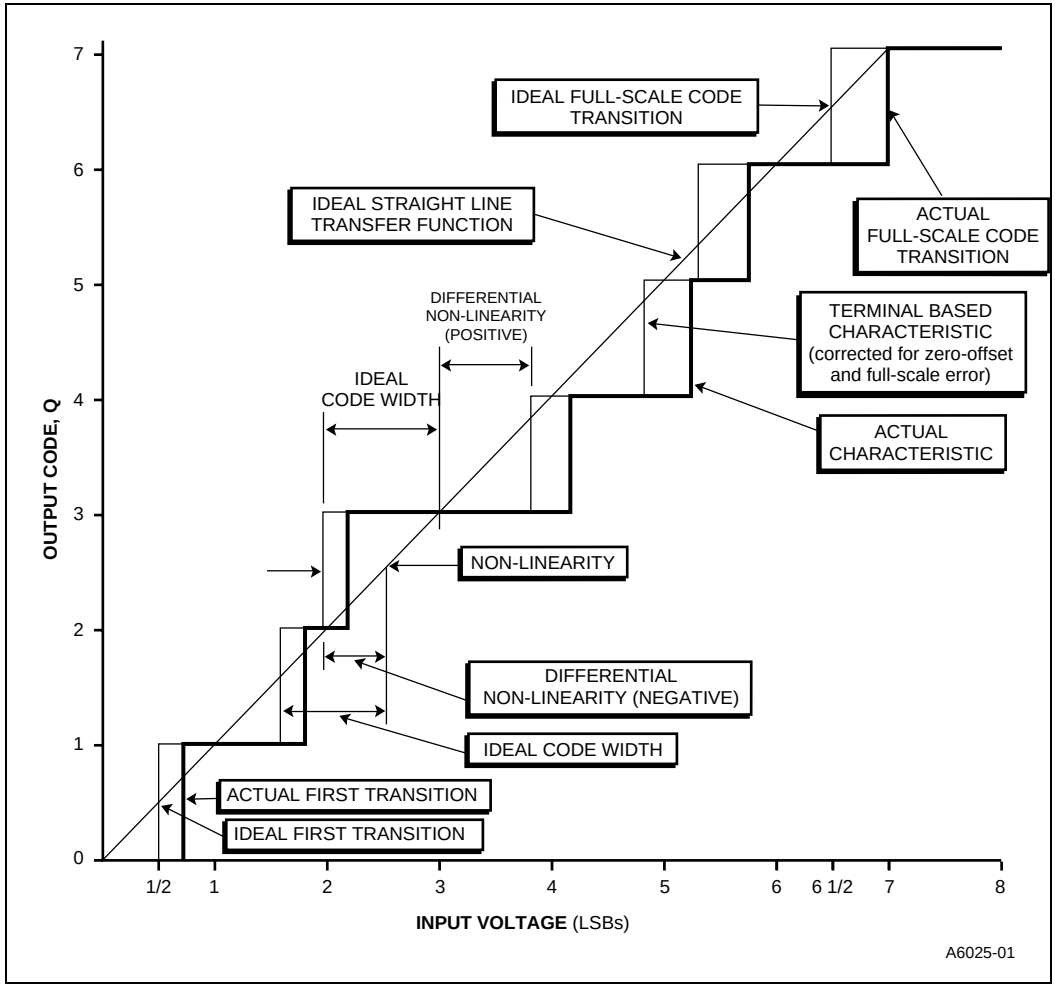


Figure 12-13. Terminal-based A/D Conversion Characteristic

13

Minimum Hardware Considerations

CHAPTER 13
MINIMUM HARDWARE CONSIDERATIONS

The 8XC196EA has several basic requirements for operation within a system. This chapter describes options for providing the basic requirements and discusses other hardware considerations.

13.1 MINIMUM CONNECTIONS

Table 13-1 lists the signals that are required for the device to function and Figure 13-1 shows the connections for a minimum configuration.

Table 13-1. Minimum Required Signals

Signal Name	Type	Description
ANGND	GND	Analog Ground ANGND must be connected for A/D converter operation. ANGND and V_{SS} should be nominally at the same potential.
EA#	I	External Access This input determines whether memory accesses to the upper 7 Kbytes of ROM (FF2400–FF3FFFH) are directed to internal or external memory. These accesses are directed to internal memory if EA# is held high and to external memory if EA# is held low. For an access to any other memory location, the value of EA# is irrelevant. EA# is sampled and latched only on the rising edge of RESET#. Changing the level of EA# after reset has no effect.
PLLEN	I	Phase-locked Loop Enable This active-high input pin enables the on-chip clock multiplier.
RESET#	I/O	Reset A level-sensitive reset input to, and an open-drain system reset output from, the microcontroller. Either a falling edge on RESET# or an internal reset turns on a pull-down transistor connected to the RESET# pin for 16 state times. In the powerdown and idle modes, asserting RESET# causes the microcontroller to reset and return to normal operating mode. After a reset, the first instruction fetch is from FF2080H (or 1F2080H in external memory).
V_{CC}	PWR	Digital Supply Voltage Connect each V_{CC} pin to the digital supply voltage.
V_{REF}	PWR	Reference Voltage for the A/D Converter This pin supplies operating voltage to the A/D converter.
V_{SS}	GND	Digital Circuit Ground These pins supply ground for the digital circuitry. Connect each V_{SS} pin to ground through the lowest possible impedance path.

Table 13-1. Minimum Required Signals (Continued)

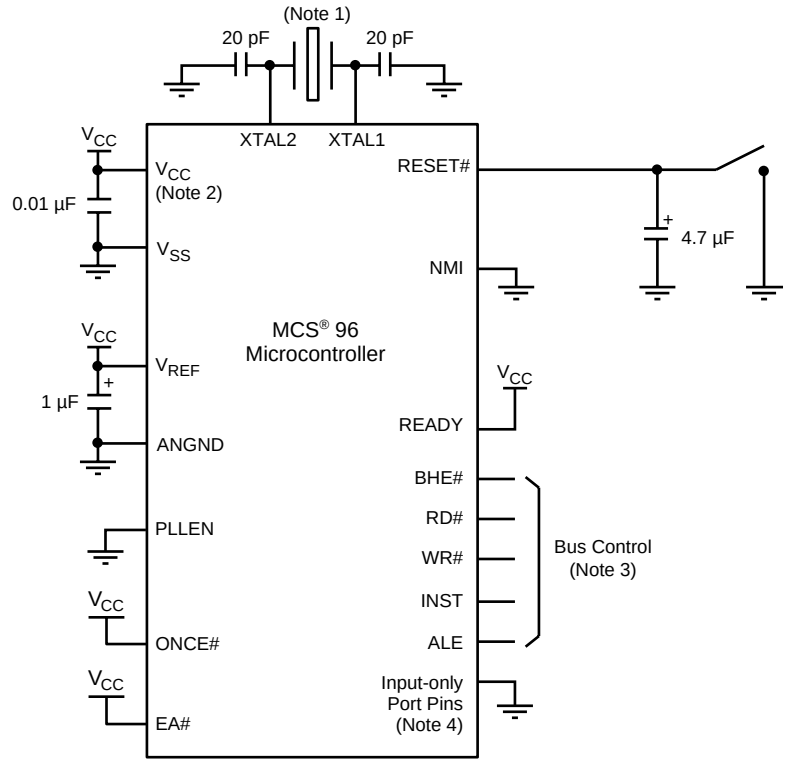
Signal Name	Type	Description
XTAL1	I	Input Crystal/Resonator or External Clock Input Input to the on-chip oscillator and the internal clock generators. The internal clock generators provide the peripheral clocks, CPU clock, and CLKOUT signal. When using an external clock source instead of the on-chip oscillator, connect the clock input to XTAL1. The external clock signal must meet the V_{IH} specification for XTAL1.
XTAL2	O	Inverted Output for the Crystal/Resonator Output of the on-chip oscillator inverter. Leave XTAL2 floating when the design uses an external clock source instead of the on-chip oscillator.

13.1.1 Unused Inputs

For predictable performance, it is important to tie unused inputs to V_{CC} or V_{SS}. Otherwise, they can float to a mid-voltage level and draw excessive current. Unused interrupt inputs may generate spurious interrupts if left unconnected.

13.1.2 I/O Port Pin Connections

Tie unused input-only port inputs to V_{SS} as shown in Figure 13-1. Chapter 7, "I/O Ports," contains information about initializing and configuring the ports. See "Configuring the Port Pins" on page 7-7.



Notes:
1. See the datasheet for the oscillator frequency range (F_{XTAL1}) and the crystal manufacturer's datasheet for recommended load capacitors.
2. The number of V_{CC} and V_{SS} pins varies with package type (see datasheet). Be sure to connect all V_{CC} pins to the supply voltage and all V_{SS} pins to ground, including the analog supply pins, V_{REF} and ANGND.
3. No connection is required.
4. Tie all input-only port pins to $V_{SS.}$

A3386-01

Figure 13-1. Minimum Hardware Connections

13.2 APPLYING AND REMOVING POWER

When power is first applied to the microcontroller, RESET# must remain continuously low for at least one state time after the power supply is within tolerance and the oscillator/clock has stabilized; otherwise, operation might be unpredictable. Similarly, when powering down a system, RESET# should be brought low before V_{CC} is removed; otherwise, an inadvertent write to an external location might occur. Carefully evaluate the possible effect of power-up and power-down sequences on a system.

13.3 NOISE PROTECTION TIPS

The fast rise and fall times of high-speed CMOS logic often produce noise spikes on the power supply lines and outputs. To minimize noise, it is important to follow good design and board layout techniques. We recommend liberal use of decoupling capacitors and transient absorbers. Add 0.01 µF bypass capacitors between V_{CC} and each V_{SS} pin and a 1.0 µF capacitor between V_{REF} and ANGND to reduce noise (Figure 13-2). Place the capacitors as close to the device as possible. Use the shortest possible path to connect V_{SS} lines to ground and to each other.

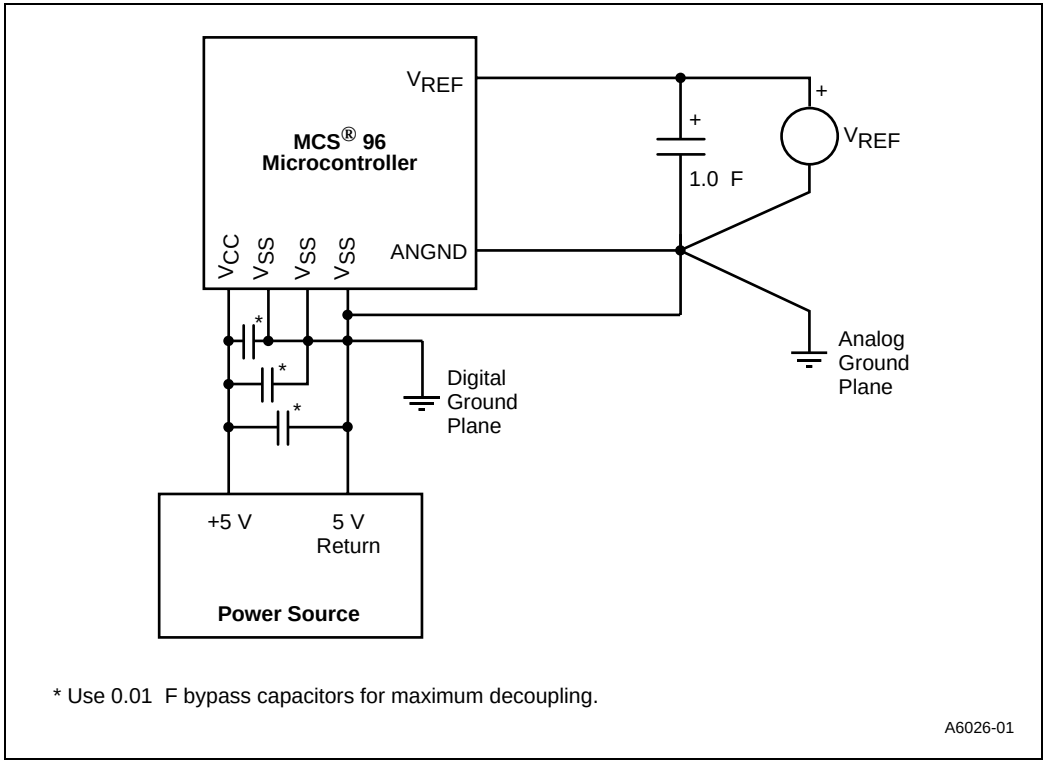


Figure 13-2. Power and Return Connections

If the A/D converter will be used, connect V_{REF} to a separate reference supply to minimize noise during A/D conversions. Even if the A/D converter will not be used, V_{REF} and ANGND must be connected. Refer to "Analog Ground and Reference Voltages" on page 12-16 for a detailed discussion of A/D power and ground recommendations.

Multilayer printed circuit boards with separate V_{CC} and ground planes also help to minimize noise. For more information on noise protection, refer to AP-125, *Designing Microcontroller Systems for Noisy Environments* (order number 210313) and AP-711, *EMI Design Techniques for Microcontrollers in Automotive Applications* (order number 272637).

13.4 THE ON-CHIP OSCILLATOR CIRCUITRY

The on-chip oscillator circuit (Figure 13-3) consists of a crystal-controlled, positive reactance oscillator. In this application, the crystal operates in a parallel resonance mode. The feedback resistor (Rf) consists of paralleled *n*-channel and *p*-channel FETs controlled by the internal powerdown signal. In powerdown mode, Rf acts as an open and the output drivers are disabled, which disables the oscillator. Both the XTAL1 and XTAL2 pins have built-in electrostatic discharge (ESD) protection.

NOTE

Although the maximum external clock input frequency is 40MHz, the maximum oscillator input frequency is limited to 20MHz.

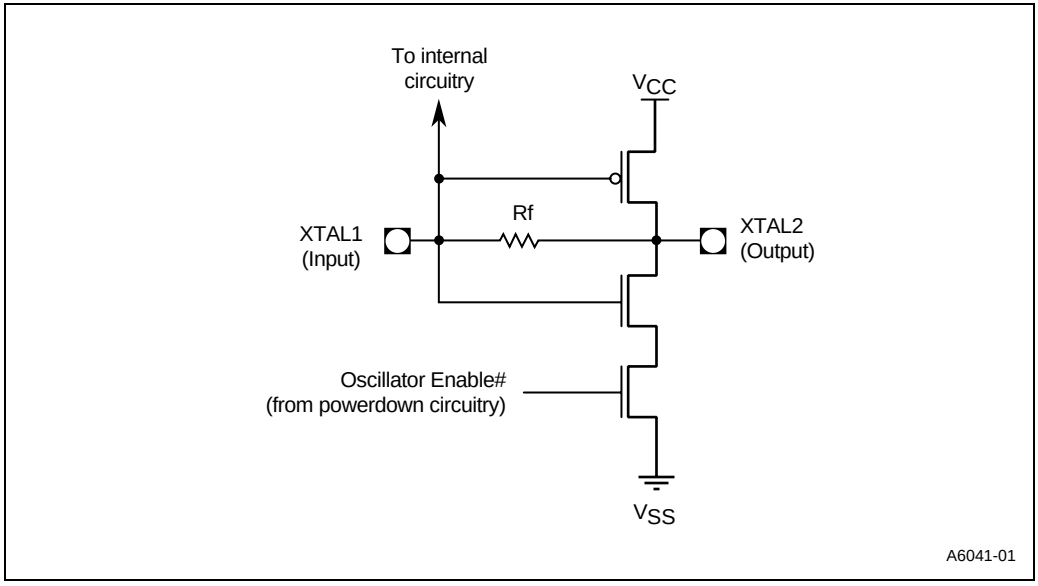
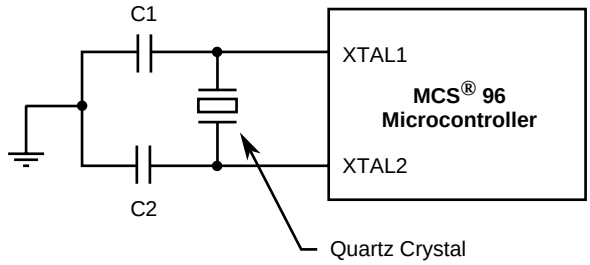


Figure 13-3. On-chip Oscillator Circuit

Figure 13-4 shows the connections between the external crystal and the device. When designing an external oscillator circuit, consider the effects of parasitic board capacitance, extended operating temperatures, and crystal specifications. Consult the manufacturer's datasheet for performance specifications and required capacitor values. With high-quality components, 20 pF load capacitors (C_L) are usually adequate for frequencies above 1 MHz.

Noise spikes on the XTAL1 or XTAL2 pin can cause a miscount in the internal clock-generating circuitry. Capacitive coupling between the crystal oscillator and traces carrying fast-rising digital signals can introduce noise spikes. To reduce this coupling, mount the crystal oscillator and capacitors near the device and use short, direct traces to connect to XTAL1, XTAL2, and V_{SS}. To further reduce the effects of noise, use grounded guard rings around the oscillator circuitry and ground the metallic crystal case.



Note:
Mount the crystal and capacitors close to the device using short, direct traces to XTAL1, XTAL2, and V_{SS}. When using a crystal, C1=C2»20 pF. When using a ceramic resonator, consult the manufacturer for recommended oscillator circuitry.

A6028-01

Figure 13-4. External Crystal Connections

In cost-sensitive applications, you may choose to use a ceramic resonator instead of a crystal oscillator. Ceramic resonators may require slightly different load capacitor values and circuit configurations. Consult the manufacturer's datasheet for the requirements.

13.5 USING AN EXTERNAL CLOCK SOURCE

To use an external clock source, apply a clock signal to XTAL1 and let XTAL2 float (Figure 13-5). To ensure proper operation, the external clock source must meet the minimum high and low times (T_{XHXX} and T_{XLXX}) and the maximum rise and fall transition times (T_{XLXH} and T_{XHXL}) illustrated in Figure 13-6. The longer the rise and fall times, the higher the probability that external noise will affect the clock generator circuitry and cause unreliable operation. See the datasheet for required XTAL1 voltage drive levels and actual specifications.

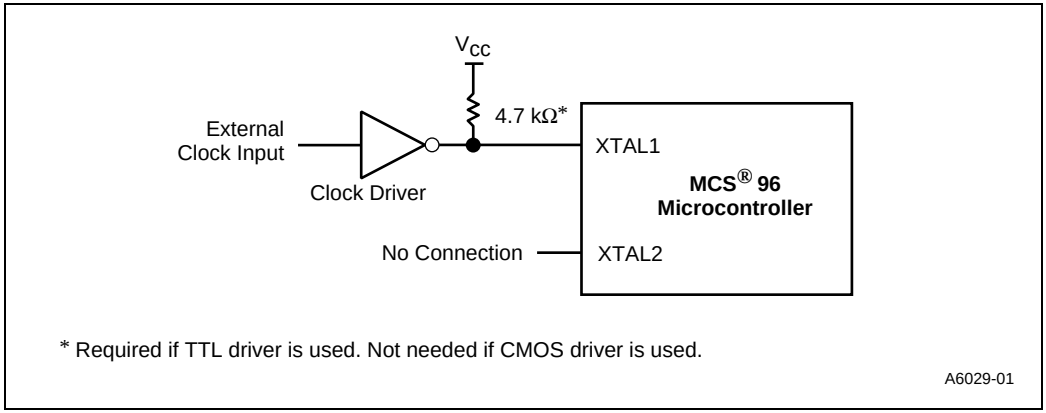


Figure 13-5. External Clock Connections

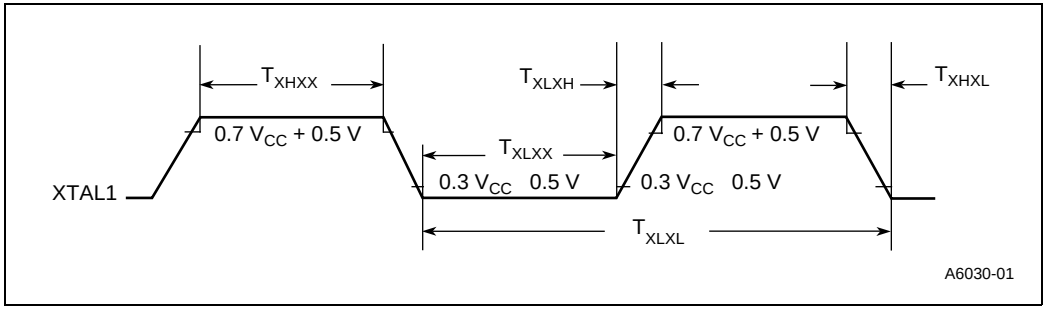
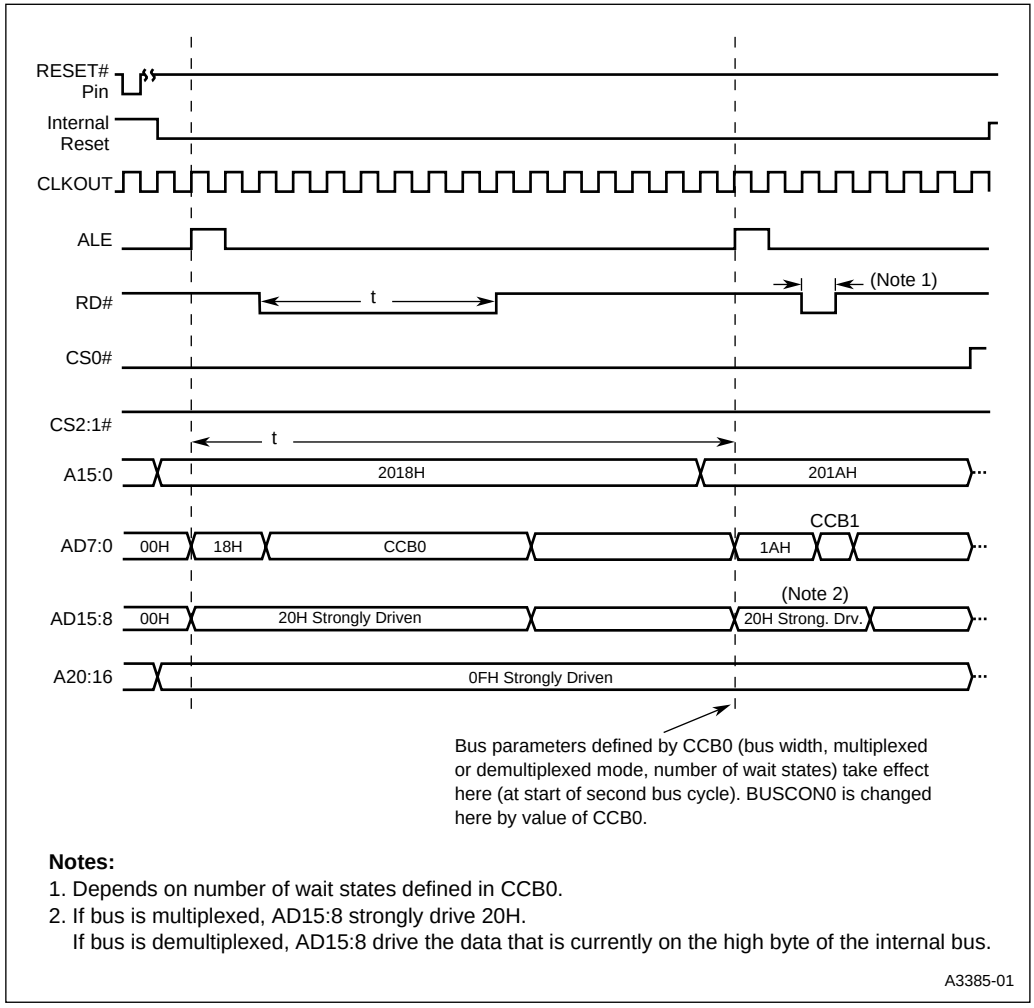


Figure 13-6. External Clock Drive Waveforms

At power-on, the interaction between the internal amplifier and its feedback capacitance (i.e., the Miller effect) may cause a load of up to 100 pF at the XTAL1 pin if the signal at XTAL1 is weak (such as might be the case during start-up of the external oscillator). This situation will go away when the XTAL1 input signal meets the V_{IL} and V_{IH} specifications (listed in the datasheet). If these specifications are met, the XTAL1 pin capacitance will not exceed 20 pF.

13.6 RESETTING THE MICROCONTROLLER

Reset forces the microcontroller into a known state. As soon as RESET# is asserted, the I/O pins, the control pins, and the registers are driven to their reset states. Table B-5 on page B-15 lists the reset states of the pins. See Table C-2 on page C-2 for the reset values of the SFRs. The microcontroller remains in its reset state until RESET# is deasserted. When RESET# is deasserted, the bus controller fetches the chip configuration bytes (CCBs), loads them into the chip configuration registers (CCRs), and then fetches the first instruction. Figure 13-7 shows the reset-sequence timing.



Notes:
1. Depends on number of wait states defined in CCB0.
2. If bus is multiplexed, AD15:8 strongly drive 20H.
   If bus is demultiplexed, AD15:8 drive the data that is currently on the high byte of the internal bus.

A3385-01

Figure 13-7. Reset Timing Sequence

The following events will reset the microcontroller (see Figure 13-8):

- an external device pulls the RESET# pin low
- the CPU executes a reset (RST) instruction
- the CPU executes an idle/powerdown (IDLPD) instruction with an illegal key operand
- the watchdog timer (WDT) overflows
- the clock failure detection (CFD) logic is enabled and a clock failure occurs

The following paragraphs describe each of these reset methods in more detail.

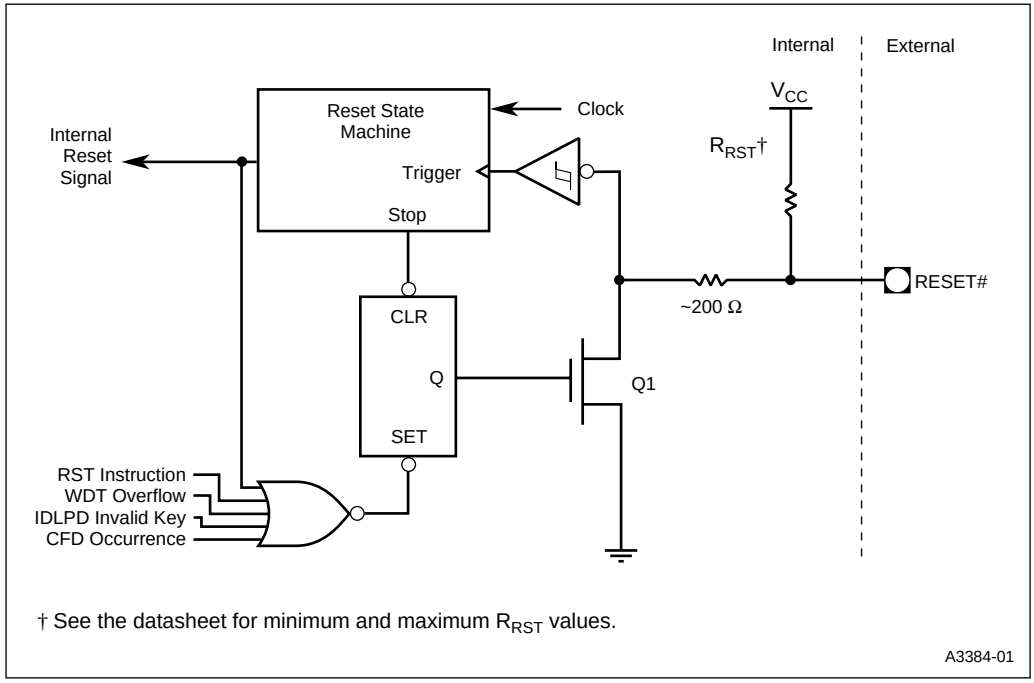


Figure 13-8. Internal Reset Circuitry

13.6.1 Generating an External Reset

To reset the device, hold the RESET# pin low for at least one state time after the power supply is within tolerance and the oscillator has stabilized. When RESET# is first asserted, the device turns on a pull-down transistor (Q1 in Figure 13-8) for 16 state times. This enables the RESET# signal to function as the system reset.

The simplest way to reset the microcontroller is to insert a capacitor between the RESET# pin and V_{SS}, as shown in Figure 13-9. The microcontroller has an internal pull-up resistor (R_{RST}), as shown in Figure 13-8. RESET# should remain asserted for at least one state time after V_{CC} and XTAL1 have stabilized and met the operating conditions specified in the datasheet. A capacitor of 4.7 µF or greater should provide sufficient reset time, as long as V_{CC} rises quickly.

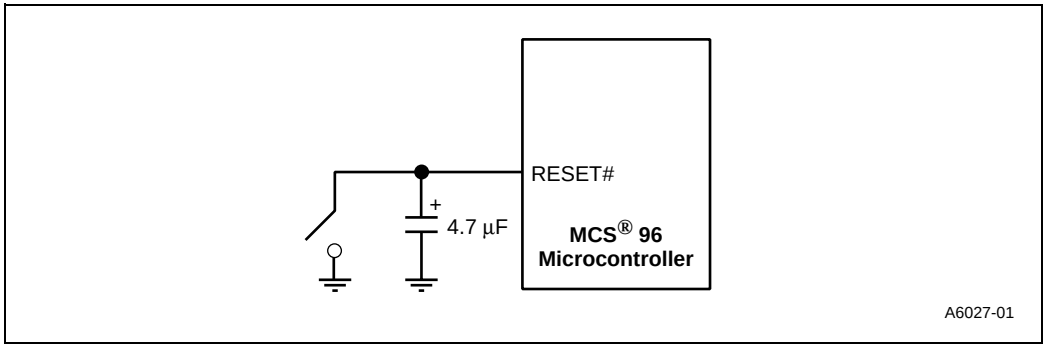


Figure 13-9. Minimum Reset Circuit

Other devices in the system may not be reset because the capacitor will keep the voltage above V_{IL}. Since RESET# is asserted for only 16 state times, it may be necessary to lengthen and buffer the system-reset pulse. Figure 13-10 shows an example of a system-reset circuit. In this example, D2 creates a wired-OR gate connection to the reset pin. An internal reset, system power-up, or SW1 closing will generate the system-reset signal.

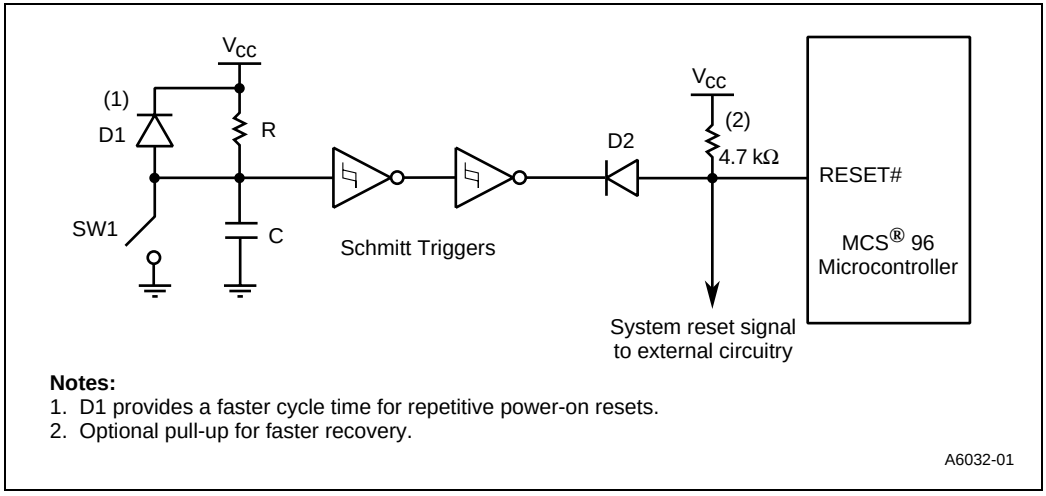


Figure 13-10. Example of a System Reset Circuit

13.6.2 Issuing the Reset (RST) Instruction

The RST instruction (opcode FFH) resets the device by pulling RESET# low for 16 state times. It also clears the processor status word (PSW), sets the master program counter (PC) to FF2080H, and resets the special function registers (SFRs). See Table 13-10 on page 13-11 for the reset values of the SFRs.

13.6.3 Issuing an Illegal IDLPD Key Operand

The device resets itself if an illegal key operand is used with the idle/powerdown (IDLPD) command. The legal keys are "1" for idle mode and "2" for powerdown mode. Any key greater than "3" will cause the device to execute a reset sequence. (See Appendix A for a description of the IDLPD command.)

13.6.4 Enabling the Watchdog Timer

The watchdog timer (WDT) is a 16-bit counter that resets the microcontroller when the counter overflows (every 64K state times). The WDE bit (bit 3) of CCR1 controls whether the watchdog is enabled immediately or is disabled until the first time it is cleared. If the WDE bit in CCR1 is clear, the watchdog is immediately activated. Software must clear the watchdog timer register to avoid a reset. If the WDE bit in CCR1 is set, the watchdog is disabled until the first time you clear it.

You must write two consecutive bytes to the watchdog register (location 0AH) to clear it. For the 83C196EA, the first byte must be 1EH; however, the second byte determines the reset interval and can be one of four values (Table 13-2). Only the values listed in the table are valid; an invalid value will not clear the register, so the counter will overflow and the watchdog will reset the device. It is recommended that you disable interrupts before writing to the watchdog register. If an interrupt occurs between the two writes, the watchdog register will not be cleared.

Table 13-2. Selecting the Watchdog Reset Interval

First Byte	Second Byte	Reset Interval
1EH	E1H	64K states
1EH	A1H	128K states (2 × 64K)
1EH	61H	256K states (4 × 64K)
1EH	21H	512K states (8 × 64K)

If enabled, the watchdog continues to run in idle mode. The device must be awakened before the end of the reset interval to clear the watchdog; otherwise, the watchdog will reset the microcontroller, which causes it to exit idle mode.

13.6.5 Detecting Clock Failure

The ability to sense a clock failure is important in safety-sensitive applications. This microcontroller provides a feature that can detect a failed clock and reset itself. Low-frequency oscillation, typically 100 kHz or below, is sensed as a failure. If enabled, the clock failure detection (CFD) circuitry resets the microcontroller in the event of a clock failure. This feature is enabled by programming the CFD bit in the chip configuration 1 (CCR1) register. (See Figure 2-10 on page 2-20 for details.)

13.7 IDENTIFYING THE RESET SOURCE

The reset source (RSTSRC) register indicates the sources of the last reset that the microcontroller encountered (see Figure 13-11). If more than one reset occurs at the same time, all of the corresponding RSTSRC bits will be set. Reading this SFR clears all the register bits.

RSTSRC Address: 1FA4H
Reset State: 00H†

The reset source (RSTSRC) register indicates the sources of the last reset that the microcontroller encountered.

7							0
—	—	—	—	CFDRST	WDTRST	SFWRST	EXTRST

Bit Number	Bit Mnemonic	Function
7:4	—	Reserved. These bits are undefined.
3	CFDRST	Clock Failure Detection Reset When set, this bit indicates that a failed clock caused the last reset.
2	WDTRST	Watchdog Timer Reset When set, this bit indicates that the watchdog timer caused the last reset.
1	SFWRST	Software Reset When set, this bit indicates that either the RST instruction or the IDLPD instruction used with an illegal key caused the last reset.
0	EXTRST	External Reset When set, this bit indicates that the RESET# pin being asserted caused the last reset.

† Applies to V_{CC} Powerup condition only.

Figure 13-11. Reset Source (RSTSRC) Register

14

Special Operating Modes

CHAPTER 14
SPECIAL OPERATING MODES

The 8XC196EA has two power saving modes: idle and powerdown. It also has an on-circuit emulation (ONCE) mode that electrically isolates the microcontroller from the other system components. This chapter describes each mode and explains how to enter and exit them.

14.1 SPECIAL OPERATING MODE SIGNALS AND REGISTERS

Table 14-1 lists the signals and Table 14-2 lists the registers that are mentioned in this chapter.

Table 14-1. Operating Mode Control Signals

Signal Name	Type	Description
CLKOUT	O	Clock Output Output of the internal clock generator. You can select one of five frequencies: f, f/2, f/4, f/8, or f/16. CLKOUT has a 50% duty cycle. CLKOUT shares a package pin with P2.7
EXTINT	I	External Interrupt In normal operating mode, a rising edge on EXTINT sets the EXTINT interrupt pending bit. EXTINT is sampled during phase 2 (CLKOUT high). The minimum high time is one state time. In powerdown mode, asserting the EXTINT signal for at least 50 ns causes the device to resume normal operation. The interrupt does not need to be enabled, but the pin must be configured as a special-function input. If the EXTINT interrupt is enabled, the CPU executes the interrupt service routine. Otherwise, the CPU executes the instruction that immediately follows the command that invoked the power-saving mode. In idle mode, asserting any enabled interrupt causes the device to resume normal operation. EXTINT shares a package pin with P2.2.
ONCE#	I	On-circuit Emulation Holding ONCE# low during the rising edge of RESET# places the device into on-circuit emulation (ONCE) mode. This mode puts all pins into a high-impedance state, thereby isolating the microcontroller from other components in the system. The value of ONCE# is latched when the RESET# pin goes inactive. While the device is in ONCE mode, you can debug the system using a clip-on emulator. To exit ONCE mode, reset the microcontroller by pulling the RESET# signal low. To prevent inadvertent entry into ONCE mode, either configure this pin as an output or hold it high during reset and ensure that your system meets the V_{IH} specification. ONCE# shares a package pin with P2.6 and HLDA#.
PLLEN	I	Phase-locked Loop Enable This active-high input pin enables the on-chip clock multiplier.

Table 14-1. Operating Mode Control Signals (Continued)

Signal Name	Type	Description
RESET#	I/O	Reset A level-sensitive reset input to, and an open-drain system reset output from, the microcontroller. Either a falling edge on RESET# or an internal reset turns on a pull-down transistor connected to the RESET# pin for 16 state times. In the powerdown and idle modes, asserting RESET# causes the microcontroller to reset and return to normal operating mode. After a reset, the first instruction fetch is from FF2080H (or 1F2080H in external memory).
RPD	I	Return from Powerdown Timing pin for the return-from-powerdown circuit. If your application uses powerdown mode, connect a capacitor between RPD and V_{SS} if **either** of the following conditions are true. • the internal oscillator is the clock source • the phase-locked loop (PLL) circuitry is enabled (see PLLEN signal description) The capacitor causes a delay that enables the oscillator and PLL circuitry to stabilize before the internal CPU and peripheral clocks are enabled. The capacitor is not required if your application uses powerdown mode and if **both** of the following conditions are true. • an external clock input is the clock source • the phase-locked loop circuitry is disabled If your application does not use powerdown mode, leave this pin unconnected. RPD shares a package pin with P5.7.
TMODE#	I	Test-Mode Entry If this pin is held low during reset, the microcontroller will enter a test mode. The value of several other pins defines the actual test mode. All test modes, except TROM execution, are reserved for Intel factory use. If you choose to configure this signal as an input, always hold it high during reset and ensure that your system meets the V_{IH} specification to prevent inadvertent entry into test mode. TMODE# shares a package pin with P5.4 and BREQ#.

Table 14-2. Operating Mode Control and Status Registers

Mnemonic	Address	Description
CCR0	2018H	Chip Configuration 0 Enables or disables the IDLPD #1 and IDLPD #2 instructions. When enabled, the IDLPD #1 instruction causes the microcontroller to enter idle mode and the IDLPD #2 instruction causes the microcontroller to enter powerdown mode. This register also selects the write-control mode and contains the bus-control parameters for fetching chip configuration byte 1.
INT_MASK	0008H	Interrupt Mask Bit 6 of this register enables and disables (masks) external interrupt EXTINT.
INT_PEND	0009H	Interrupt Pending Bit 6 of this register is set to indicate pending external interrupt EXTINT.
P2_DIR	1FD2H	Port Direction Register Each bit controls the configuration of the corresponding pin. Clearing a bit configures a pin as a complementary output; setting a bit configures a pin as a high-impedance input or an open-drain output.
P2_MODE	1FD0H	Port Mode Register Each bit controls the mode of the corresponding pin. Setting a bit configures a pin as a special-function signal; clearing a bit configures a pin as a general-purpose I/O signal.
P2_PIN	1FD6H	Port Pin Register Each bit reflects the current state of the corresponding pin, regardless of the pin's mode and configuration.
P2_REG	1FD4H	Port Data Output Register **For I/O Mode (P*x*_MODE.*x* = 0)** When a port pin is configured as a complementary output (P*x*_DIR.*x* = 0), setting the corresponding port data bit drives a one on the pin, and clearing the corresponding port data bit drives a zero on the pin. When a port pin is configured as a high-impedance input or an open-drain output (P*x*_DIR.*x* = 1), clearing the corresponding port data bit drives a zero on the pin, and setting the corresponding port data bit floats the pin, making it available as a high-impedance input. **For Special-function Mode (P*x*_MODE.*x* = 1)** When a port pin is configured as an output (either complementary or open-drain), the corresponding port data bit value is immaterial because the corresponding on-chip peripheral or system function controls the pin. To configure a pin as a high-impedance input, set both the P*x*_DIR and P*x*_REG bits.

14.2 REDUCING POWER CONSUMPTION

Both power-saving modes conserve power by disabling portions of the internal clock circuitry (Figure 14-1). The remainder of this section describes both power-saving modes in detail.

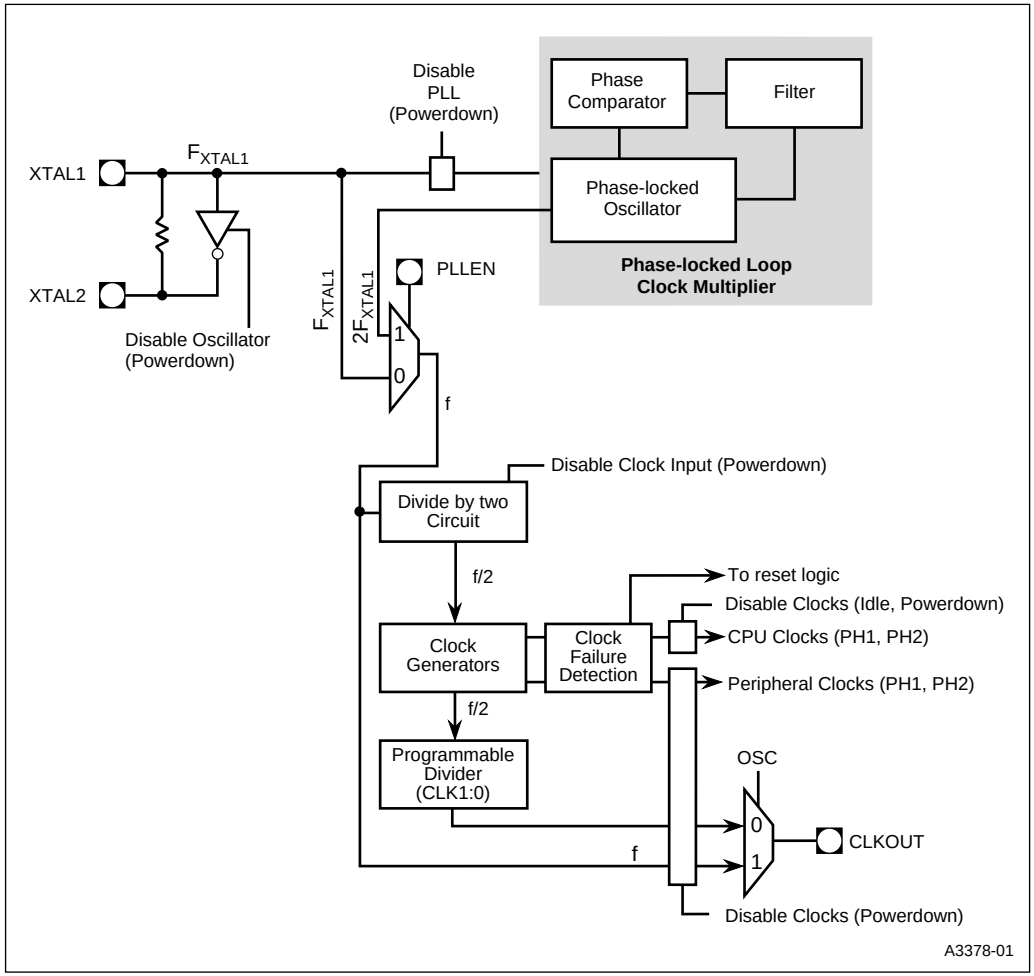


Figure 14-1. Clock Control During Power-saving Modes

14.3 IDLE MODE

In idle mode, the microcontroller's power consumption decreases to approximately 60% of normal consumption. Internal logic holds the CPU clocks at logic zero, causing the CPU to stop executing instructions. Neither the phase-locked loop circuitry, the peripheral clocks, nor CLKOUT is affected. So, the special-function registers (SFRs) and register RAM retain their data; and the peripherals and interrupt system remain active. Table B-5 on page B-15 lists the values of the pins during idle mode.

14.3.1 Enabling and Disabling Idle Mode

The PD bit in the chip configuration register 0 (CCR0.0) either enables or disables both idle and powerdown modes. CCR0 cannot be accessed by code; the PD bit value is defined in chip configuration byte 0 (CCB0.0). If the PD bit is set, both idle and powerdown modes are enabled. If the PD bit is clear, both are disabled. CCR0 is loaded from CCB0 when the microcontroller returns from reset.

14.3.2 Entering and Exiting Idle Mode

The microcontroller enters idle mode after executing the IDLPD #1 instruction. Any enabled interrupt source, either internal or external, or a hardware reset can cause the device to exit idle mode. When an interrupt occurs, the CPU clocks restart and the CPU executes the corresponding interrupt service or PTS routine. When the routine is complete, the CPU fetches and then executes the instruction that follows the IDLPD #1 instruction.

NOTE

To prevent an accidental return to full power, hold the external interrupt pin (EXTINT) low while the device is in idle mode.

14.4 POWERDOWN MODE

Powerdown mode places the microcontroller into a very low power state by disabling the internal oscillator, the phase-locked loop circuitry, and the clock generators. Internal logic holds the CPU and peripheral clocks at logic zero, which causes the CPU to stop executing instructions, the system bus-control signals to become inactive, the CLKOUT signal to become high, and the peripherals to turn off. Power consumption drops into the microwatt range (refer to the datasheet for exact specifications). I_{CC} is reduced to device leakage. Table B-5 on page B-15 lists the values of the pins during powerdown mode. If V_{CC} is maintained above the minimum specification, the special-function registers (SFRs) and register RAM retain their data.

14.4.1 Enabling and Disabling Powerdown Mode

The PD bit in the chip configuration register 0 (CCR0.0) either enables or disables both idle and powerdown modes. CCR0 cannot be accessed by code; the PD bit value is defined in chip configuration byte 0 (CCB0.0). If the PD bit is set, both idle and powerdown modes are enabled. If the PD bit is clear, both are disabled. CCR0 is loaded from CCB0 when the microcontroller returns from reset.

14.4.2 Entering Powerdown Mode

Before entering powerdown, complete the following tasks:

- Complete all serial port transmissions or receptions. Otherwise, when the device exits powerdown, the serial port activity will continue where it left off and incorrect data may be transmitted or received.
- Put all other peripherals into an inactive state.

After completing these tasks, execute the IDLPD #2 instruction to enter powerdown mode.

NOTE

To prevent an accidental return to full power, hold the external interrupt pin (EXTINT) low while the device is in powerdown mode.

14.4.3 Exiting Powerdown Mode

The microcontroller will exit powerdown mode when either of the following events occurs:

- a hardware reset is generated
- a transition occurs on the external interrupt pin

14.4.3.1 Generating a Hardware Reset

The microcontroller will exit powerdown if RESET# is asserted. Asserting RESET# causes the chip to reset and return to normal operating mode. If the phase-locked loop (PLL) clock circuitry is enabled or if the design uses an external clock input signal, you must hold RESET# low for at least 2 ms to allow the PLL to stabilize before the internal CPU and peripheral clocks are enabled. If the design uses the on-chip oscillator, then either a falling edge on RESET# or an internal reset turns on a pull-down transistor connected to the RESET# pin for 16 state times.

14.4.3.2 Asserting the External Interrupt Signal

The other way to exit powerdown mode is to assert the external interrupt signal (EXTINT) for at least 50 ns. Although EXTINT is normally a sampled input, the powerdown circuitry uses it as a level-sensitive input. An interrupt does not need to be enabled to bring the microcontroller out of powerdown, but the pin must be configured as a special-function input (see “Configuring the Port Pins” on page 7-7). Figure 14-2 shows the power-up and power-down sequence when using an external interrupt to exit powerdown.

When the external interrupt brings the microcontroller out of powerdown mode, the corresponding pending bit is set in the interrupt pending register. If the interrupt is enabled, the device executes the interrupt service routine, then fetches and executes the instruction following the IDLPD #2 instruction. If the interrupt is disabled (masked), the device fetches and executes the instruction following the IDLPD #2 instruction, and the pending bit remains set until the interrupt is serviced or software clears the pending bit.

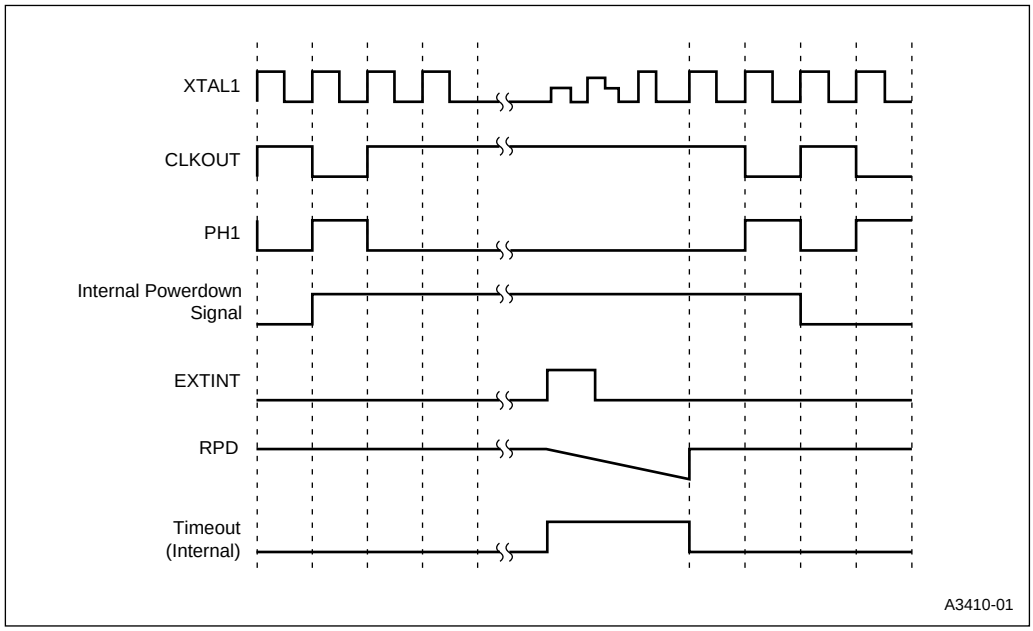


Figure 14-2. Power-up and Power-down Sequence When Using an External Interrupt

When using the external interrupt signal to exit powerdown mode, we recommend that you connect the external capacitor shown in Figure 14-3 to the RPD pin. The discharging of the capacitor causes a delay that allows the oscillator and phase-locked loop (PLL) circuitry to stabilize before the internal CPU and peripheral clocks are enabled.

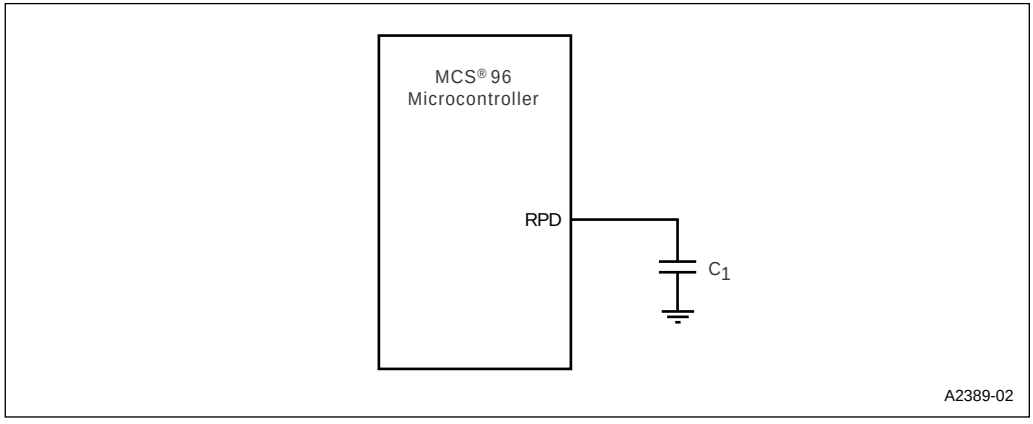


Figure 14-3. External RC Circuit

During normal operation (before entering powerdown mode), an internal pull-up holds the RPD pin at V_{CC}. When the external interrupt signal is asserted, the internal oscillator circuitry is enabled and turns on a weak internal pull-down (approximately 10 kΩ). This weak pull-down causes the external capacitor (C_1) to begin discharging at a typical rate of 200 µA. When the RPD pin voltage drops below the threshold voltage (about 2.5 V), the internal phase clocks are enabled and the device resumes code execution.

At this time, a Schmitt-triggered detection circuit prompted by the switching voltage levels strongly drives a logic one, quickly pulling the RPD pin back up to V_{CC} (see recovery time in Figure 14-4). The time constant (RC) follows an exponential charging curve. However, since there is no external resistor on the RPD pin, the time constant goes to zero and the recovery time is instantaneous.

$$V_c = V_{cc}[1 - e^{(t/\tau)}]; \quad (\tau = RC_1 = 0)$$

$$V_c = V_{cc}$$

where:

V_C = Charging capacitor voltage

14.4.3.3 Selecting an External Capacitor

With the resistance of the discharge path designed into the silicon with an internal pull-down resistor, the selection of an external capacitor (C_1) can be critical. Ideally, you want to select a component that will produce a sufficient discharge time to permit the internal oscillator circuitry to stabilize. Because many factors can influence the discharge time requirement, you should always fully characterize your design under worst-case conditions to verify proper operation.

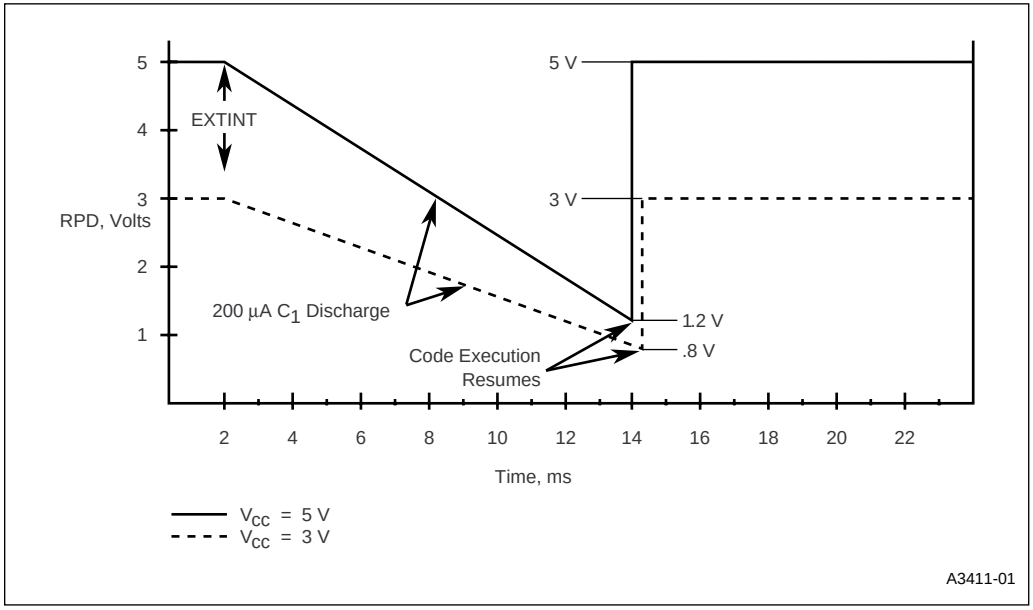


Figure 14-4. Typical Voltage on the RPD Pin While Exiting Powerdown

When selecting the capacitor, determine the worst-case discharge time needed for the oscillator to stabilize, then use this formula to calculate an appropriate value for C_1.

$$C_1 = \frac{T_{DIS} \times I}{V_t}$$

where:

C_1	is the capacitor value, in farads
T_{DIS}	is the worst-case discharge time, in seconds
I	is the discharge current, in amperes
V_t	is the threshold voltage

NOTE

If powerdown is re-entered and exited before C_1 charges to V_{CC}, it will take less time for the voltage to ramp down to the threshold. Therefore, the device will take less time to exit powerdown.

For example, assume that the oscillator needs at least 12.5 ms to stabilize (T_{DIS} = 12.5 ms), V_t is 2.5 V, and the discharge current is 200 µA. The minimum C_1 capacitor size is 1 µF.

$$C_1 = \frac{(0.0125)(0.0002)}{2.5} = 1\ \mu F$$

When using an external oscillator, the value of C_1 can be very small, allowing rapid recovery from powerdown. For example, a 100 pF capacitor discharges in 1.25 µs.

$$T_{DIS} = \frac{C_1 \times V_t}{I} = \frac{(1.0 \times 10^{-10})(2.5)}{0.0002} = 1.25\ \mu s$$

14.5 ONCE MODE

On-circuit emulation (ONCE) mode isolates the microcontroller from other components in the system to allow printed-circuit-board testing and debugging with a clip-on emulator. During ONCE mode, all pins except XTAL1, XTAL2, RESET#, V_{SS}, and V_{CC} are in a high impedence state. RESET# must be held high; otherwise, the microcontroller will exit ONCE mode and enter the reset state.

Holding the ONCE# signal low during the rising edge of RESET# causes the microcontroller to enter ONCE mode. The PLLEN pin must also be held low. The ONCE signal is latched when RESET goes inactive. Internally, the ONCE# pin is tied to a medium-strength pull-up. To prevent accidental entry into ONCE mode, connect the ONCE# pin to V_{CC}.

Exit ONCE mode by asserting the RESET# signal. Normal operations resume when RESET# goes high.

15

Interfacing with External Memory

CHAPTER 15
INTERFACING WITH EXTERNAL MEMORY

The microcontroller can interface with a variety of external memory devices. Three chip-selects can be individually programmed for bus width, the number of wait states, and a multiplexed or demultiplexed address/data bus. Other features of the external memory interface include ready control for inserting additional wait states, a bus-hold protocol that enables external devices to take control of the bus, and two write-control modes for writing words and bytes to memory. These features provide a great deal of flexibility when interfacing with external memory systems.

In addition to describing the signals and registers related to external memory, this chapter discusses the process of fetching the chip configuration bytes and configuring the external bus. It also provides examples of external memory configurations and chip-select setup.

15.1 INTERNAL AND EXTERNAL ADDRESSES

The address that external devices see is different from the address that the microcontroller generates internally. The microcontroller has 24 address bits internally, but only 21 address pins (A20:0) externally. The absence of the upper three address bits at the external pins causes different internal addresses to have the same external address. For example, the internal addresses FF2080H, 7F2080H, and 1F2080H all appear at the 21 external pins as 1F2080H. The upper three bits of the internal address have no effect on the external address.

The address seen by an external device also depends on the number of address lines that the external system uses. If the address on the external pins (A20:0) is 1F2080H, and only A17:0 are connected to the external device, the external device sees 32080H. The upper five address lines (A20:16) are implemented by the EPORT. Table 15-1 shows how the external address depends on the number of EPORT lines used to address the external device.

Table 15-1. Example of Internal and External Addresses

Internal Address	Address on the Microcontroller Pins	EPORT Pins Connected to the External Device	Address Seen by External Device
*x*F2080H (*x* = 1, 3, 5, 7, 9, B, D, F)	1F2080H	A16	12080H
		A17:16	32080H
		A18:16	72080H
		A19:16	F2080H
		A20:16	1F2080H

Table 15-1. Example of Internal and External Addresses (Continued)

Internal Address	Address on the Microcontroller Pins	EPORT Pins Connected to the External Device	Address Seen by External Device
*x*F2080H (*x* = 0, 2, 4, 6, 8, A, C, E)	0F2080H	A16	12080H
		A17:16	32080H
		A18:16	72080H
		A19:16	F2080H
		A20:16	0F2080H

15.2 EXTERNAL MEMORY INTERFACE SIGNALS AND REGISTERS

Table 15-2 lists the signals and Table 15-3 lists the registers that are discussed in this chapter. Some of the microcontroller port pins can function as either bus-control signals or general purpose I/O signals. "Using the Special-function Signals" on page 7-11 describes how to configure a port pin as either a general purpose I/O signal or a bus-control signal.

Table 15-2. Bus-control Signals

Signal Name	Port Pin	Type	Description
A15:0	—	O	System Address Bus These address pins provide address bits 0–15 during the entire external memory cycle during both multiplexed and demultiplexed bus modes.
A20:16	EPORT.4:0	O	Address Pins 16–20 These address pins provide address bits 16–20 during the entire external memory cycle during both multiplexed and demultiplexed bus modes, supporting extended addressing of the 2-Mbyte address space. **NOTE:** Internally, there are 24 address bits; however, only 21 external address pins (A20:0) are implemented. The internal address space is 16 Mbytes (000000–FFFFFFH) and the external address space is 2 Mbytes (000000–1FFFFFH). The microcontroller resets to FF2080H in internal memory or 1F2080H in external memory. A20:16 share package pins with EPORT.4:0.

Table 15-2. Bus-control Signals (Continued)

Signal Name	Port Pin	Type	Description
AD15:0	P4.7:0 P3.7:0	I/O	Address/Data Lines The function of these pins depends on the bus width and mode. When a bus access is not occurring, these pins revert to their I/O port function. **16-bit Multiplexed Bus Mode**: AD15:0 drive address bits 0–15 during the first half of the bus cycle and drive or receive data during the second half of the bus cycle. **8-bit Multiplexed Bus Mode**: AD15:8 drive address bits 8–15 during the entire bus cycle. AD7:0 drive address bits 0–7 during the first half of the bus cycle and drive or receive data during the second half of the bus cycle. **16-bit Demultiplexed Mode**: AD15:0 drive or receive data during the entire bus cycle. **8-bit Demultiplexed Mode**: AD7:0 drive or receive data during the entire bus cycle. AD15:8 drive the data that is currently on the high byte of the internal bus.
ALE	P5.0	O	Address Latch Enable This active-high output signal is asserted only during external memory cycles. ALE signals the start of an external bus cycle and indicates that valid address information is available on the system address/data bus (A20:16 and AD15:0 for a multiplexed bus; A20:0 for a demultiplexed bus). An external latch can use this signal to demultiplex address bits 0–15 from the address/data bus in multiplexed mode. ALE shares a package pin with P5.0.
BHE#	P5.5	O	Byte High Enable† During 16-bit bus cycles, this active-low output signal is asserted for word and high-byte reads and writes to external memory. BHE# indicates that valid data is being transferred over the upper half of the system data bus. Use BHE#, in conjunction with address bit 0 (A0 for a demultiplexed address bus, AD0 for a multiplexed address/data bus), to determine which memory byte is being transferred over the system bus: **BHE# AD0 or A0 Byte(s) Accessed** 0 0 both bytes 0 1 high byte only 1 0 low byte only BHE# shares a package pin with P5.5 and WRH#. † When this pin is configured as a special-function signal (P5_MODE.5 = 1), the chip configuration register 0 (CCR0) determines whether it functions as BHE# or WRH#. CCR0.2 = 1 selects BHE#; CCR0.2 = 0 selects WRH#.

Table 15-2. Bus-control Signals (Continued)

Signal Name	Port Pin	Type	Description
BREQ#	P5.4	O	Bus Request This active-low output signal is asserted during a hold cycle when the bus controller has a pending external memory cycle. When the bus-hold protocol is enabled (WSR.7 is set), the P5.4/BREQ# pin can function only as BREQ#, regardless of the configuration selected through the port configuration registers (P5_MODE, P5_DIR, and P5_REG). An attempt to change the pin configuration is ignored until the bus-hold protocol is disabled (WSR.7 is cleared). The microcontroller can assert BREQ# at the same time as or after it asserts HLDA#. Once it is asserted, BREQ# remains asserted until HOLD# is deasserted. BREQ# shares a package pin with P5.4 and TMODE#.
CLKOUT	P2.7	O	Clock Output Output of the internal clock generator. You can select one of five frequencies: f, f/2, f/4, f/8, or f/16. CLKOUT has a 50% duty cycle. CLKOUT shares a package pin with P2.7
CS2:0#	EPORT.7:5	O	Chip-select Lines 0–2 The active-low output CS*x*# is asserted during an external memory cycle when the address to be accessed is in the range programmed for chip select *x*. If the external memory address is outside the range assigned to the three chip selects, no chip-select output is asserted and the bus configuration defaults to the CS2# values. Immediately following reset, CS0# is automatically assigned to the range FF2000–FF20FFH (1F2000–1F20FFH if external). CS2:0# share package pins with EPORT.7:5.
EA#	—	I	External Access This input determines whether memory accesses to the upper 7 Kbytes of ROM (FF2400–FF3FFFH) are directed to internal or external memory. These accesses are directed to internal memory if EA# is held high and to external memory if EA# is held low. For an access to any other memory location, the value of EA# is irrelevant. EA# is sampled and latched only on the rising edge of RESET#. Changing the level of EA# after reset has no effect.
HLDA#	P2.6	O	Bus Hold Acknowledge This active-low output indicates that the CPU has released the bus as the result of an external device asserting HOLD#. When the bus-hold protocol is enabled (WSR.7 is set), the P2.6/HLDA# pin can function only as HLDA#, regardless of the configuration selected through the port configuration registers (P2_MODE, P2_DIR, and P2_REG). An attempt to change the pin configuration is ignored until the bus-hold protocol is disabled (WSR.7 is cleared). HLDA# shares a package pin with P2.6 and ONCE#.

Table 15-2. Bus-control Signals (Continued)

Signal Name	Port Pin	Type	Description
HOLD#	P2.5	I	Bus Hold Request An external device uses this active-low input signal to request control of the bus. When the bus-hold protocol is enabled (WSR.7 is set), the P2.5/HOLD# pin can function only as HOLD#, regardless of the configuration selected through the port configuration registers (P2_MODE, P2_DIR, and P2_REG). An attempt to change the pin configuration is ignored until the bus-hold protocol is disabled (WSR.7 is cleared). HOLD# shares a package pin with P2.5.
INST	P5.1	O	Instruction Fetch When high, INST indicates that an instruction is being fetched from external memory. The signal remains high during the entire bus cycle of an external instruction fetch. INST is low for data accesses, including interrupt vector fetches and chip configuration byte reads. INST is low during internal memory fetches. INST shares a package pin with P5.1.
RD#	P5.3	O	Read Read-signal output to external memory. RD# is asserted only during external memory reads. RD# shares a package pin with P5.3.
READY	P5.6	I	Ready Input This active-high input can be used to insert wait states in addition to those programmed in the chip configuration byte 0 (CCB0) and the bus control *x* register (BUSCON*x).* CCB0 is programmed with the minimum number of wait states (0–3) for an external fetch of CCB1, and BUSCON*x* is programmed with the minimum number of wait states (0–3) for all external accesses to the address range assigned to the chip-select *x* channel. If READY is low when the programmed number of wait states is reached, additional wait states are added until READY is pulled high. READY shares a package pin with P5.6.
WR#	P5.2	O	Write† This active-low output indicates that an external write is occurring. This signal is asserted only during external memory writes. WR# shares a package pin with P5.2 and WRL#. † When this pin is configured as a special-function signal (P5_MODE.2 = 1), the chip configuration register 0 (CCR0) determines whether it functions as WR# or WRL#. CCR0.2 = 1 selects WR#; CCR0.2 = 0 selects WRL#.

Table 15-2. Bus-control Signals (Continued)

Signal Name	Port Pin	Type	Description
WRH#	P5.5	O	Write High† During 16-bit bus cycles, this active-low output signal is asserted for high-byte writes and word writes to external memory. During 8-bit bus cycles, WRH# is asserted for all write operations. WRH# shares a package pin with P5.5 and BHE#. † When this pin is configured as a special-function signal (P5_MODE.5 = 1), the chip configuration register 0 (CCR0) determines whether it functions as BHE# or WRH#. CCR0.2 = 1 selects BHE#; CCR0.2 = 0 selects WRH#.
WRL#	P5.2	O	Write Low† During 16-bit bus cycles, this active-low output signal is asserted for low-byte writes and word writes to external memory. During 8-bit bus cycles, WRL# is asserted for all write operations. WRL# shares a package pin with P5.2 and WR#. † When this pin is configured as a special-function signal (P5_MODE.2 = 1), the chip configuration register 0 (CCR0) determines whether it functions as WR# or WRL#. CCR0.2 = 1 selects WR#; CCR0.2 = 0 selects WRL#.

Table 15-3. External Memory Interface Registers

Register Mnemonic	Address	Description
ADDRCOM0 ADDRCOM1 ADDRCOM2	1E78H 1E80H 1E88H	Address Compare Holds address bits 8–20 of the base address of the address range assigned to CS*x*#.
ADDRMSK0 ADDRMSK1 ADDRMSK2	1E7AH 1E82H 1E8AH	Address Mask Determines the size of the address range (256 bytes–2 Mbyte) assigned to CS*x*#.
BUSCON0 BUSCON1 BUSCON2	1E7CH 1E84H 1E8CH	Bus Control Determines the bus configuration for external accesses to the address range assigned to CS*x*#. The bus parameters are 8- or 16-bit bus width, multiplexed or demultiplexed address/data lines, and the number of wait states inserted into each bus cycle.
CCR0	†	Chip Configuration 0 Enables or disables the IDLPD #1 and IDLPD #2 instructions. When enabled, the IDLPD #1 instruction causes the microcontroller to enter idle mode and the IDLPD #2 instruction causes the microcontroller to enter powerdown mode. This register also selects the write-control mode and contains the bus-control parameters for fetching chip configuration byte 1.
CCR1	†	Chip Configuration 1 Selects the 64-Kbyte or 2-Mbyte addressing mode and controls whether the internal ROM is mapped only into page FFH or into both pages FFH and 00H.
EP_DIR	1FE3H	Extended Port Direction In I/O mode, each bit of the extended port I/O direction (EP_DIR) register controls the configuration of the corresponding pin. Clearing a bit configures a pin as a complementary signal; setting a bit configures a pin as an open-drain signal. Any pin that is configured for its extended-address function is forced to the complementary output mode except during reset, hold, idle, and powerdown.
EP_MODE	1FE1H	Extended Port Mode The EPORT pins 0–4 can function as general-purpose I/O signals or as extended address signals. EPORT pins 5–7 can function as general-purpose I/O signals or as chip-select signals. Each bit of the extended port mode (EP_MODE) register controls whether the corresponding pin functions as a general-purpose I/O signal or as an extended-address signal. Setting a bit configures a pin as an extended-address signal; clearing a bit configures a pin as a general-purpose I/O signal.
EP_PIN	1FE7H	Extended Port Input Each bit of the extended port input (EP_PIN) register reflects the current state of the corresponding pin, regardless of the pin configuration.

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after reset. The CCBs reside at addresses FF2018H (CCB0) and FF201AH (CCB1).

Table 15-3. External Memory Interface Registers (Continued)

Register Mnemonic	Address	Description
EP_REG	1FE5H	Extended Port Data Output The EPORT pins 0–4 can function as general purpose I/O signals or as extended address signals. EPORT pins 5–7 can function as general purpose I/O signals or as chip-select signals. **For I/O Mode (EP_MODE.*x* = 0)** When a port pin is configured as a complementary output (EP_DIR.*x* = 0), setting the corresponding EP_REG bit drives a one on the pin and clearing the corresponding EP_REG bit drives a zero on the pin. When a port pin is configured as a high impedance input or an open-drain output (EP_DIR.*x* = 1), clearing the corresponding EP_REG bit drives a zero on the pin and setting the corresponding EP_REG bit floats the pin, making it available as a high impedance input. **For Special-function Mode (EP_MODE.*x* = 1)** When an EPORT pin is configured as a special-function signal (either a chip-select or an extended-address signal), the EP_REG bit value is immaterial because the address bus or the chip-select unit controls the pin.
P2_DIR P5_DIR	1FD2H 1FF3H	Port Direction Register Each bit controls the configuration of the corresponding pin. Clearing a bit configures a pin as a complementary output; setting a bit configures a pin as a high-impedance input or an open-drain output.
P2_MODE P5_MODE	1FD0H 1FF1H	Port Mode Register Each bit controls the mode of the corresponding pin. Setting a bit configures a pin as a special-function signal; clearing a bit configures a pin as a general-purpose I/O signal.
P2_PIN P5_PIN	1FD6H 1FF7H	Port Pin Register Each bit reflects the current state of the corresponding pin, regardless of the pin's mode and configuration.
P2_REG P5_REG	1FD4H 1FF5H	Port Data Output Register **For I/O Mode (P*x*_MODE.*x* = 0)** When a port pin is configured as a complementary output (P*x*_DIR.*x* = 0), setting the corresponding port data bit drives a one on the pin, and clearing the corresponding port data bit drives a zero on the pin. When a port pin is configured as a high-impedance input or an open-drain output (P*x*_DIR.*x* = 1), clearing the corresponding port data bit drives a zero on the pin, and setting the corresponding port data bit floats the pin, making it available as a high-impedance input. **For Special-function Mode (P*x*_MODE.*x* = 1)** When a port pin is configured as an output (either complementary or open-drain), the corresponding port data bit value is immaterial because the corresponding on-chip peripheral or system function controls the pin. To configure a pin as a high-impedance input, set both the P*x*_DIR and P*x*_REG bits.

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after reset. The CCBs reside at addresses FF2018H (CCB0) and FF201AH (CCB1).

15.3 THE CHIP-SELECT UNIT

The chip-select unit provides three outputs, CS2:0#, for selecting an external device during an external bus cycle. During an external memory access, a chip-select output CS*x*# is asserted if the address falls within the address range assigned to that chip-select. The bus width, the number of wait states, and multiplexed or demultiplexed address/data lines are programmed independently for each of the three chip-selects. If the external address is outside the range of the three chip-selects, the chip-select 2 bus control register determines the wait states, bus width, and multiplexing for the current bus cycle, and no chip-select is asserted.

Figure 15-1 illustrates the microcontroller's calculation of a chip-select output CS*x*# for a given external memory address. Address bits 8–20 of the memory address are compared (XORed) bitwise with the 13 least-significant bits (BASE20:8) of the ADDRCOM*x* register. If all of the bits match, CS*x*# is asserted. Additionally, if some bits do not match, CS*x*# is still asserted if, for each non-matching bit in ADDRCOM*x*, the corresponding bit in ADDRMSK*x* is cleared. The 13 least-significant bits are named MASK20:8 for their function in masking bits BASE20:8.

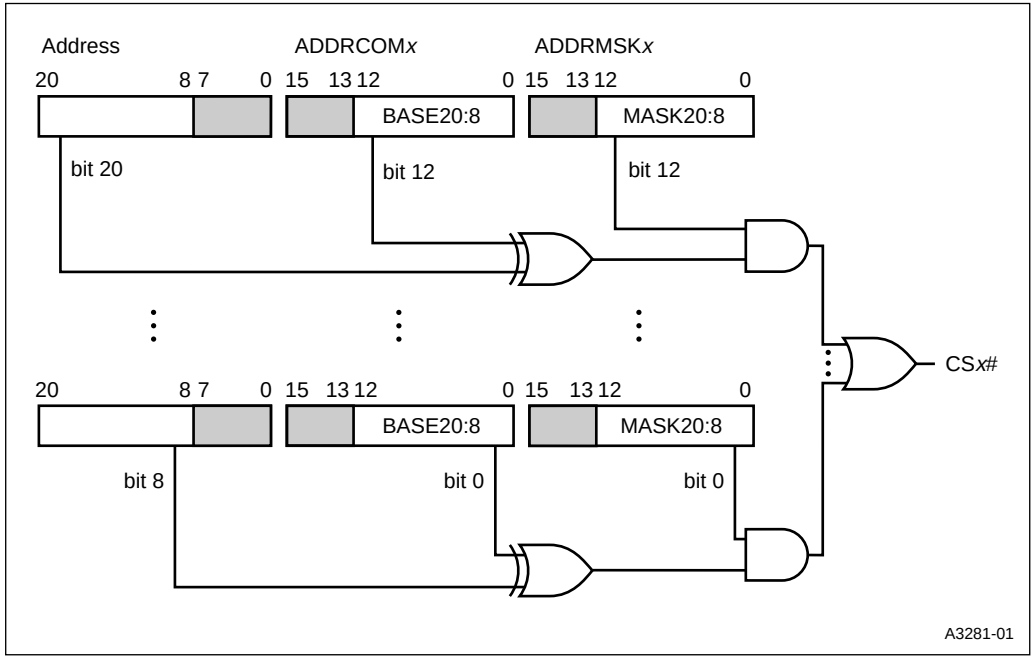


Figure 15-1. Calculation of a Chip-select Output

15.3.1 Defining Chip-select Address Ranges

This section describes the ADDRCOM*x* and ADDRMSK*x* registers and how to set them up for a desired address range. The ADDRCOM*x* register (Figure 15-2) and ADDRMSK*x* register (Figure 15-3) control the assertion of each chip-select output. The BASE20:8 bits in the ADDRCOM*x* register determine the base address of the address range. The MASK20:8 bits in the ADDRMSK*x* register determine the size of the address range.

ADDRCOM*x*
***x* = 0–2**

Address: 1E78H, 1E80H, 1E88H
Reset State: 1F20H, 0000H, 0000H

The address compare (ADDRCOM*x*) register specifies the base (lowest) address of the address range. The base address of a 2^n-byte address range must be on a 2^n-byte boundary.

15							8
—	—	—	BASE20	BASE19	BASE18	BASE17	BASE16
7							**0**
BASE15	BASE14	BASE13	BASE12	BASE11	BASE10	BASE9	BASE8

Bit Number	Bit Mnemonic	Function
15:13	—	Reserved; for compatibility with future devices, write zeros to these bits.
12:0	BASE20:8	Base Address Bits Write address bits 20–8 of the base address of the address range assigned to chip-select *x* to these bits.

Figure 15-2. Address Compare (ADDRCOM*x*) Registers

ADDRMSK*x*
***x* = 0–2**

Address: 1E7AH, 1E82H, 1E8AH
Reset State: 1FFFH

The address mask (ADDRMSK*x*) register, together with the address compare register, defines the address range that is assigned to the chip-select *x* output, CS*x*#. The address mask register determines the size of the address range, which must be 2^n bytes, where n = 8, 9, . . , 21. For a 2^n-byte address range, calculate n_1 = 21– n, and set the n_1 most-significant bits of MASK20:8 in the address mask register.

15							8
—	—	—	MASK20	MASK19	MASK18	MASK17	MASK16
7							**0**
MASK15	MASK14	MASK13	MASK12	MASK11	MASK10	MASK9	MASK8

Bit Number	Bit Mnemonic	Function
15:13	—	Reserved; for compatibility with future devices, write zeros to these bits.
12:0	MASK20:8	Address Mask Bits For a 2^n-byte address range, set the n_1 most-significant bits of MASK20:8, where n_1 = 21 – n.

Figure 15-3. Address Mask (ADDRMSK*x*) Registers

Observe the following restrictions in choosing an address range for a chip-select output:

- The addresses in the address range must be contiguous.
- The size of the address range must be 2^n bytes, where n = 8, 9, ..., 21. This corresponds to block sizes of 256 bytes, 512 bytes, ..., 2 Mbyte.
- The base address of a 2^n-byte address range must be on a 2^n-byte boundary (that is, the base address must be evenly divisible by 2^n). For example, the base address of a 256-Kbyte range must be 00000H, 40000H, 80000H, or C0000H. Table 15-4 shows the base addresses for some address-range sizes.
- The address ranges for different chip-selects must not overlap, unless their BUSCON*x* parameters (wait states, bus width, and multiplexing) have the same values. If BUSCON*x* registers have different parameter values and an address in their overlapping region is accessed, the results are unpredictable. See “Example of a Chip-select Setup” on page 15-15 for a chip-select initialization procedure that avoids this difficulty.

Table 15-4. Base Addresses for Several Sizes of the Address Range

Address-Range Size	2 Mbyte (2^{21})	1 Mbyte (2^{20})	512 Kbyte (2^{19})	256 Kbyte (2^{18})		512 bytes (2^{9})	256 bytes (2^{8})
Base Addresses				1C0000H		1FFE00H	1FFF00H
				180000H		1FFC00H	1FFE00H
				140000H		1FFA00H	1FFD00H
				100000H	• • •	• • •	• • •
			180000H	0C0000H		000600H	000300H
			100000H	080000H		000400H	000200H
		100000H	080000H	040000H		000200H	000100H
	000000H	000000H	000000H	000000H		000000H	000000H

For an address range satisfying these restrictions, set up the ADDRCOM*x* and ADDRMSK*x* registers as follows:

- Place address bits 20–8 of the base address into bits BASE20:8 in the ADDRCOM*x* register (Figure 15-2).
- For an address range of 2^n bytes, set the n_1 most-significant bits of MASK20:8 in the ADDRMSK*x* register (Figure 15-3), where $n_1 = 21 - n$.

For example, assume that chip-select output *x* is to be assigned to a 32-Kbyte address range with base address 1E0000H. The address range size is $32 \times 1024 = 2^{15}$, and $n_1 = 21 - 15 = 6$. To set up the registers, write address bits 20–8 of 1E0000H to BASE20:8 in the ADDRCOM*x* register, and set the 6 most-significant bits of MASK20:8 in the ADDRMSK*x* register:

ADDRCOM*x* = 1EF00H
ADDRMSK*x* = 1F80H

Note that the 32-Kbyte address range could not have 4000H as base address, for example, because 4000H is not on a 32-Kbyte boundary.

"Example of a Chip-select Setup" on page 15-15 shows another example of setting up the chip-select unit.

15.3.2 Controlling Bus Parameters

For each chip-select output address range, the bus control register BUSCON*x* (Figure 15-4) determines the wait states, the bus width, and the address/data multiplexing.

The chip-select output signals share package pins with port 3. Use the port registers to configure these pins as general-purpose I/O signals or as chip-select signals (see "Chip-select Signals (EPORT)" on page 7-16. The bus configuration programmed in BUSCON*x* applies to address range *x*, regardless of the port 3 pin configurations.

BUSCON*x*
***x* = 0–2**

Address: 1E7CH, 1E84H, 1E8CH
Reset State: 0003H, 0000H, 0000H

For the address range assigned to chip-select *x*, the bus control (BUSCON*x*) register specifies the number of wait states, the bus width, and the address/data multiplexing for all external bus cycles that access address range *x*.

The chip-select output signals share package pins with port 3. Use the port registers to configure these pins as general-purpose I/O signals or as chip-select signals (see "Chip-select Signals (EPORT)" on page 7-16). The bus configuration programmed in BUSCON*x* applies to address range *x*, regardless of the port 3 pin configurations.

7							0
DEMUX	BW16	—	—	—	—	WS1	WS0

Bit Number	Bit Mnemonic	Function
7	DEMUX	Address/Data Multiplexing This bit specifies the address/data multiplexing on AD15:0 for all external accesses to the address range assigned to chip-select *x* output. 0 = multiplexed 1 = demultiplexed
6	BW16	Bus Width This bit specifies the bus width for all external accesses to the address range assigned to chip-select *x* output. 0 = 8 bits 1 = 16 bits
5:2	—	Reserved; for compatibility with future devices, write zeros to these bits.
1:0	WS1:0	Wait States These bits, along with the READY pin, control the number of wait states for all external accesses to the address range assigned to the chip-select *x* channel. Write the desired minimum number of wait states (0–3) to WS1:0. If READY is low when this number is reached, additional wait states are added until READY is pulled high.

Figure 15-4. Bus Control (BUSCON*x*) Registers

15.3.3 Chip-select Unit Initial Conditions

A chip reset produces the following initial conditions for the chip-select unit:

- ADDRMSK*x* = 1FFFH.
- ADDRCOM0 = 1F20H. This asserts CS0# for the 256-byte address range 1F2000–1F20FFH.
- ADDRCOM1 and ADDRCOM2 = 0000H.
- For the fetch of chip configuration byte 0 (CCB0), BUSCON0 is initialized for an 8-bit bus width, multiplexed mode, and three wait states (DEMUX = 0, BW16 = 0, WS0 = 1, WS1 = 1).
- Before the fetch of chip configuration byte 1 (CCB1), the values of DEMUX, BW16, WS0, and WS1 in BUSCON0 are loaded from CCB0. The external bus is configured according to the new values.

15.3.4 Programming the Chip-select Registers

When programming the chip-select parameters, it is important to avoid a condition in which two chip-selects outputs have overlapping address ranges and different bus-parameter values (wait states, bus width, and multiplexing). Accessing a location in such an overlapping address range can cause unpredictable results.

Use the following sequence to program the chip-select registers after reset:

1. Program chip-select output 0:
    1.1. Clear ADDRMSK0.
    1.2. Write to ADDRCOM0 to establish the desired base address.
    1.3. Write to ADDRMSK0 to establish the desired address range.
    1.4. Write the desired bus-parameter values to BUSCON0.
2. While executing in the address range defined in step 1 for chip-select output 0, use the following sequence to program chip-select outputs 1–2. Begin with $x = 1$.
    2.1. Write to ADDRCOM*x* to establish the desired base address.
    2.2. Write to ADDRMSK*x* to establish the desired address range.
    2.3. Write the desired bus-parameter values to BUSCON*x*.
    2.4. Repeat steps 2.1–2.4 for $x = 2$.

15.3.5 Example of a Chip-select Setup

This section shows an example of setting up the chip-select unit and provides details of the chip-select output calculation. This example shows how to set up the chip-select registers for the system shown in Figure 15-5. For each address range, the BUSCON*x* register (Figure 15-4) specifies the address/data multiplexing (bit 7), the bus width (bit 6), and the number of wait states (bits 0–1). Table 15-5 lists the characteristics of the three chip-select outputs and shows the corresponding contents of BUSCON*x*.

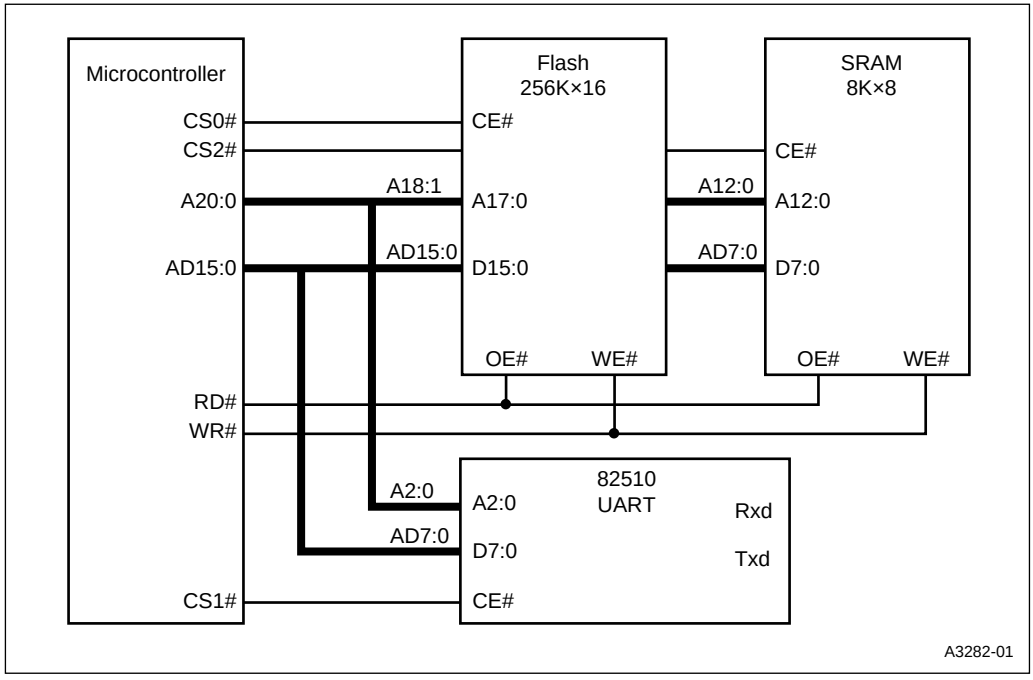


Figure 15-5. Example System for Setting Up Chip-select Outputs

Table 15-5. BUSCON*x* Registers for the Example System

Chip-select Output	Multiplexing	Bus Width	Wait States	Contents of BUSCON*x*
CS0#	Demultiplexed	16 bits	0	C0H
CS1#	Demultiplexed	8 bits	3	83H
CS2#	Demultiplexed	8 bits	0	80H

The location and size of an address range are specified by the ADDRCOM*x* register (Figure 15-2 on page 15-10) and the ADDRMSK*x* register (Figure 15-3 on page 15-7). The 8-Kbyte SRAM is assigned to address range 17E000–17FFFFH and uses chip-select output 2. Address bits 20–8 of the base address (17E000H) are written to the BASE20:8 bits in the ADDRCOM2 register, which then contains 17E0H.

The address range for CS2# is 8 Kbytes or 2^{13} bytes (n = 13). The number of bits to be set in MASK20:8 of ADDRMSK2 is 21 – 13 = 8. After the 8 most-significant bits of MASK20:8 are set, ADDRMSK2 contains 1FE0H. Results for CS0# and CS1# are found similarly, as shown in Table 15-6.

Table 15-6. Results for the Chip-select Example

Chip-Select Output	Address Range	Size of Address Range	Number of Bits to Set in ADDRMSK*x*	Contents of ADDRCOM*x*	Contents of ADDRMSK*x*
CS0#	180000–1FFFFFH	512 Kbytes = 2^{19} bytes	n_1 = 21 – 19 = 2	1800H	1800H
CS1#	101E00–101EFFH	256 bytes = 2^8 bytes	n_1 = 21 – 8 = 13	101EH	1FFFH
CS2#	17E000–17FFFFH	8 Kbytes = 2^{13} bytes	n_1 = 21 – 13 = 8	17E0H	1FE0H

15.4 CHIP CONFIGURATION REGISTERS AND CHIP CONFIGURATION BYTES

Two chip configuration registers (CCRs) have bits that set parameters for chip operation and external bus cycles. The CCRs cannot be accessed by code. They are loaded from the chip configuration bytes (CCBs), which reside at addresses FF2018H (CCB0) and FF201AH (CCB1). If the CCBs are stored in external memory, their external addresses depend on the number of EPORT lines used in the external system (see "Internal and External Addresses" on page 15-1).

When the microcontroller returns from reset, the bus controller fetches the CCBs and loads them into the CCRs. From this point, these CCR bit values define the chip configuration until the microcontroller is reset again. The CCR bits are described in Figures 15-6 and 15-7. The remainder of this section describes the state of the microcontroller following reset and the process of fetching the CCBs.

CCR0 no direct access†

The chip configuration 0 (CCR0) register controls ROM access, enables or disables the IDLPD #1 and IDLPD #2 instructions and selects the write-control mode. It also contains the bus-control parameters for fetching chip configuration byte 1.

7							0
LOC	1	WS1	WS0	DEMUX	BHE#	BW16	PD

Bit Number	Bit Mnemonic	Function
7	LOC	Lock Bit This bit controls read access to the ROM during normal operation. 0 = read protect 1 = no protection Refer to “Controlling Read Access to the Internal ROM” on page 4-26 for details.
6	1	To guarantee proper operation, write one to this bit.
5:4	WS1:0	Wait States These bits, along with the READY pin, control the number of wait states that are used for an external fetch of chip configuration byte 1 (CCB1). **WS1 WS0** 0 0 zero wait states 0 1 one wait state 1 0 two wait states 1 1 three wait states If READY is low when this number is reached, additional wait states are added until READY is pulled high.
3	DEMUX	Select Demultiplexed Bus Selects the demultiplexed bus mode for an external fetch of CCB1: 0 = multiplexed — address and data are multiplexed on AD15:0. 1 = demultiplexed — data only on AD15:0.
2	BHE#	Write-control Mode Selects the write-control mode, which determines the functions of the BHE#/WRH# and WR#/WRL# pins for external bus cycles: 0 = write strobe mode: the BHE#/WRH# pin operates as WRH#, and the WR#/WRL# pin operates as WRL#. 1 = standard write-control mode: the BHE#/WRH# pin operates as BHE#, and the WR#/WRL# pin operates as WR#.

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after reset. The CCBs reside at addresses FF2018H (CCB0) and FF201AH (CCB1); therefore, define your CCRs at these addresses.

Figure 15-6. Chip Configuration 0 (CCR0) Register

CCR0 (Continued) no direct access†

The chip configuration 0 (CCR0) register controls ROM access, enables or disables the IDLPD #1 and IDLPD #2 instructions and selects the write-control mode. It also contains the bus-control parameters for fetching chip configuration byte 1.

7							0
LOC	1	WS1	WS0	DEMUX	BHE#	BW16	PD

Bit Number	Bit Mnemonic	Function
1	BW16	Buswidth Control Selects the bus width for an external fetch of CCB1: 0 = 8-bit bus 1 = 16-bit bus
0	PD	Powerdown Enable Enables or disables the IDLPD #1 and IDLPD #2 instructions. When enabled, the IDLPD #1 instruction causes the microcontroller to enter idle mode and the IDLPD #2 instruction causes the microcontroller to enter powerdown mode.0 = disable idle and powerdown modes 1 = enable idle and powerdown modes If your design uses idle or powerdown mode, set this bit when you program the CCBs. If it does not, clearing this bit when you program the CCBs will prevent accidental entry into idle or powerdown mode. (Chapter 14, "Special Operating Modes," discusses idle and powerdown modes.)

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after reset. The CCBs reside at addresses FF2018H (CCB0) and FF201AH (CCB1); therefore, define your CCRs at these addresses.

Figure 15-6. Chip Configuration 0 (CCR0) Register (Continued)

CCR1 no direct access†

The chip configuration 1 (CCR1) register controls the clock failure detection circuitry, deferred mode, and the watchdog timer, selects the 64-Kbyte or 2-Mbyte addressing mode.

7							0
1	CFD	DM	0	WDD	REMAP	MODE64	0

7	1	To guarantee proper operation, write one to this bit.
6	CFD	Clock-failure Detection This bit enables or disables the clock failure detection circuitry. 0 = disables clock-failure detection circuitry 1 = enables clock-failure detection circuitry (See “Clock Failure Detection Logic” on page 2-12.)
5	DM	Deferred Mode Enables the deferred bus-cycle mode. If the microcontroller is using a demultiplexed bus and deferred mode is enabled, a delay of 2t occurs in the first bus cycle following a chip-select output change, the first write cycle following a read cycle, and the first read cycle following a write cycle. (See “Deferred Bus-cycle Mode” on page 15-41.) 0 = deferred bus-cycle mode disabled 1 = deferred bus-cycle mode enabled
4	0	To guarantee proper operation, write zero to this bit.
3	WDD	Watchdog Timer Disable Selects whether the watchdog timer is always enabled or disabled until the first time it is cleared. If this bit is clear, the watchdog is enabled at reset, so software must clear the watchdog within 64K state times to prevent another reset. If this bit is set, the watchdog is disabled until the first time you clear it. (See “Enabling the Watchdog Timer” on page 13-12.) 0 = always enabled 1 = disabled at reset; enabled the first time it is cleared

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after reset. Locate your CCBs at addresses FF2018H (CCB0) and FF201AH (CCB1).

Figure 15-7. Chip Configuration 1 (CCR1) Register

CCR1 (Continued) no direct access†

The chip configuration 1 (CCR1) register controls the clock failure detection circuitry, deferred mode, and the watchdog timer, selects the 64-Kbyte or 2-Mbyte addressing mode.

7							0
1	CFD	DM	0	WDD	REMAP	MODE64	0

2	REMAP	Internal ROM Mapping The EA# pin controls whether accesses to FF2000–FF3FFFH are directed to internal ROM or to external memory. When EA# is low (external execution), REMAP is ignored. When EA# is high (internal execution), REMAP controls whether the upper 7-Kbyte area (FF2400–FF3FFFH) of internal ROM is mapped only into page FFH or into both pages FFH and 00H. 0 = ROM maps to page FFH only 1 = ROM maps to page FFH and page 00H (See "Remapping Internal ROM" on page 4-29.)
1	MODE64	Addressing Mode Selects 64-Kbyte or 2-Mbyte addressing. 0 = selects 2-Mbyte addressing 1 = selects 64-Kbyte addressing In 2-Mbyte mode, code can execute from almost anywhere in the address space. In 64-Kbyte mode, code can execute only from page FFH. (See "Fetching Code and Data in the 2-Mbyte and 64-Kbyte Modes" on page 4-31 for more information.)
0	0	Reserved; for compatibility with future devices, write zero to this bit.

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after reset. Locate your CCBs at addresses FF2018H (CCB0) and FF201AH (CCB1).

Figure 15-7. Chip Configuration 1 (CCR1) Register (Continued)

Upon leaving the reset state, the microcontroller is configured for normal operation. This section describes the state of the microcontroller following reset and summarizes the steps in the configuration process. Following reset, the microcontroller automatically fetches the two chip configuration bytes from external memory if EA# = 0 and from internal ROM if EA# = 1.

If the CCBs are stored in external ROM, chip-select output 0 (CS0#) should be connected to that device. Chip-select output 0 is initialized for the address range FF2000–FF20FFH, which includes the CCB locations. Following the CCB fetches, the microcontroller fetches the instruction at FF2080H.

The microcontroller uses the following bus control parameters for the CCB0 fetch:

- 3 wait states. The READY pin is active for the CCB0 and CCB1 fetches and can be used to insert additional wait states. See "Wait States (Ready Control)" on page 15-29.
- 8-bit, multiplexed bus

CCB0 can be fetched over a 16-bit bus, even though the microcontroller defaults to an 8-bit bus for the CCB0 fetch. The upper address pins, A20:8 and AD15:8, are strongly driven during the CCB0 fetch because an 8-bit bus is assumed. Therefore, if you have a 16-bit data bus, write the value 20H to FF2019H to avoid contention on AD15:8. Pins A20:0 are driven in the multiplexed mode. You can access the memory using A20:0 and use AD15:0 for data only.

The CCB0 values (wait states, bus width, and multiplexing) control the CCB1 fetch. Following the fetch, the CCB0 values are stored in the chip-select output 0 bus control register, BUSCON0 (see “Chip-select Unit Initial Conditions” on page 15-14). CCB0 and CCB1 are described in “Chip Configuration Registers and Chip Configuration Bytes” on page 15-16.

After RESET# is deasserted, the bus-hold function is disabled internally (WSR.7 is clear), and the following pins are initialized:

- The P2.7/CLKOUT pin operates as CLKOUT (as during reset). Be sure that the CLKOUT signal does not damage external hardware.
- The EPORT.5/CS0# pin operates as CS0#, which is asserted for the CCB fetches. If you plan to use the EPORT.5 pin as an input, you must reconfigure it from its post-reset operation as an output.
- The BHE#/WRH# pin operates as BHE#.
- The WR#/WRL#/P5.2 pin operates as general-purpose I/O (P5.2) and is weakly driven high.
- The READY/P5.6 pin operates as READY.
- The INST/P5.1 pin is weakly driven low.
- The AD15:0 pins are active.
- The following port pins are weakly held high: P2.6:0, port 3, port 4, P5.0, P5.7:2, and ports 7–12.
- The EPORT.4:0 pins are forced high, regardless of the state of the EA# pin.

Following reset, you should initialize the stack pointer and initialize the chip-select outputs using the procedure in “Example of a Chip-select Setup” on page 15-15.

15.5 BUS WIDTH AND MULTIPLEXING

The external bus can operate with a 16-bit or an 8-bit data bus and with a multiplexed or a demultiplexed address/data bus. Figure 15-10 shows the external bus signals during operation in the four combinations of bus width and multiplexing.

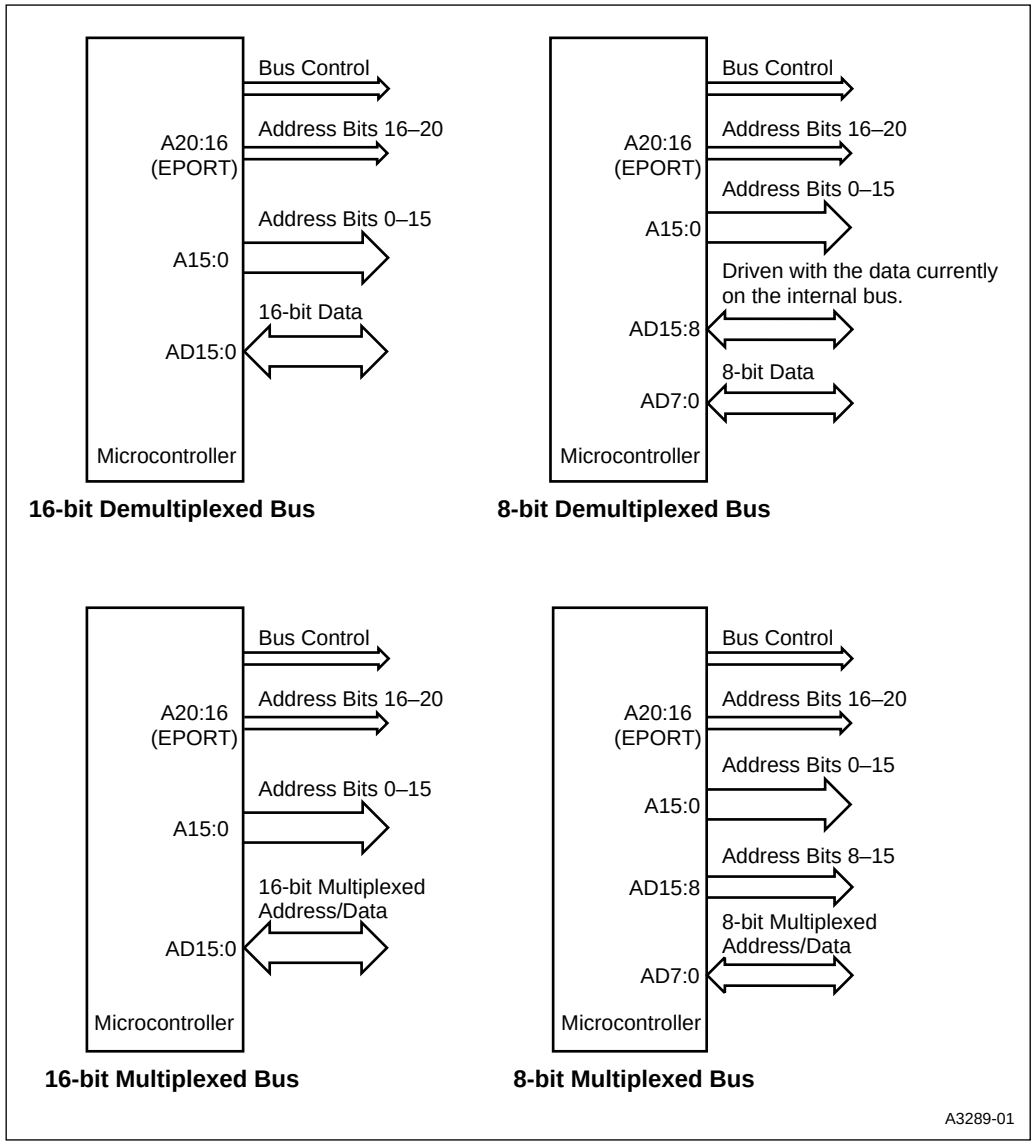


Figure 15-8. Multiplexing and Bus Width Options

A system can incorporate external devices that operate with different bus widths and multiplexing. The chip-select output assigned to the address being accessed determines the bus parameters for that particular bus cycle. Figure 15-9 shows the address and data bus configurations for the four combinations of bus width and multiplexing. For detailed waveforms, see “16-bit Bus Timings” on page 15-25 and “System Bus AC Timing Specifications” on page 15-40.

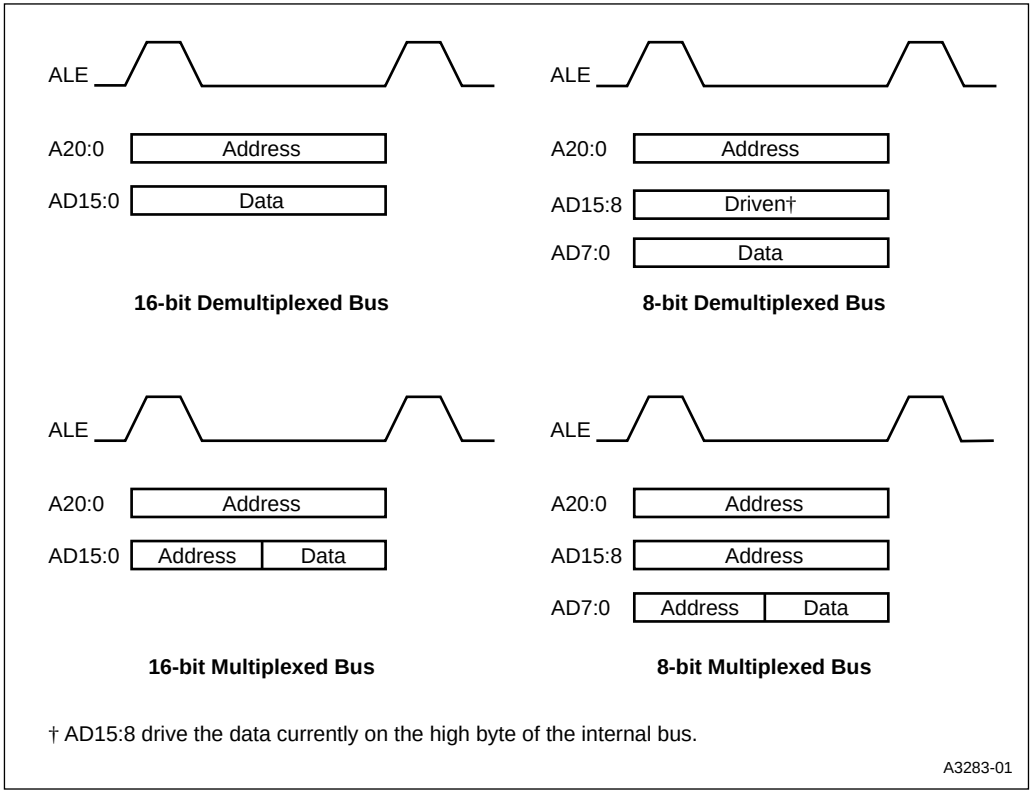


Figure 15-9. Bus Activity for Four Types of Buses

In demultiplexed mode (top of Figure 15-8 and Figure 15-9), the external device receives the address from A20:0. In a 16-bit system, the data is on AD15:0. In an 8-bit system, the data is on AD7:0, and AD15:8 drive the data currently on the high byte of the internal bus.

In multiplexed mode (bottom half of Figure 15-8 and Figure 15-9), both A20:0 and AD15:0 drive the address. A20:0 drive the address throughout the entire bus cycle. For a 16-bit bus width, AD15:0 drive the address for the first half of the bus cycle and drive or receive data during the second half. In the 8-bit case, AD15:8 drive the address during the entire bus cycle.

In multiplexed mode, with the full address on the bus for only half of the cycle, the external device has less time to receive it and to respond. As a result, for the same bus-cycle length (4t) a multiplexed system requires a faster external device (unless wait states are added to the bus cycle). Although the multiplexed mode has this disadvantage, it is useful for compatibility with devices designed for multiplexed operation.

In a 16-bit system (left side of Figure 15-8 and Figure 15-9) one data word can be transferred over AD15:0 in a single bus cycle. In an 8-bit system, one data word is transferred as two bytes over AD7:0 in successive bus cycles, and AD15:8 drive the upper eight address bits for the entire bus cycle.

The flexibility of the chip-select unit enables you to specify the bus width, the number of wait states, and a multiplexed or demultiplexed bus for each of the three chip-select outputs. The system in Figure 15-5 on page 15-15 illustrates a mixture of 16-bit and 8-bit devices with different numbers of wait states.

15.5.1 A 16-bit Example System

Figure 15-10 shows a 16-bit system in demultiplexed mode. The 256K×16 flash memory receives the address on A18:1; data is transferred on AD15:0. Using the WR# signal as shown, this system writes words, not single bytes, to the memory. (Using WRL# and WRH#, you can write single bytes on a 16-bit bus.)

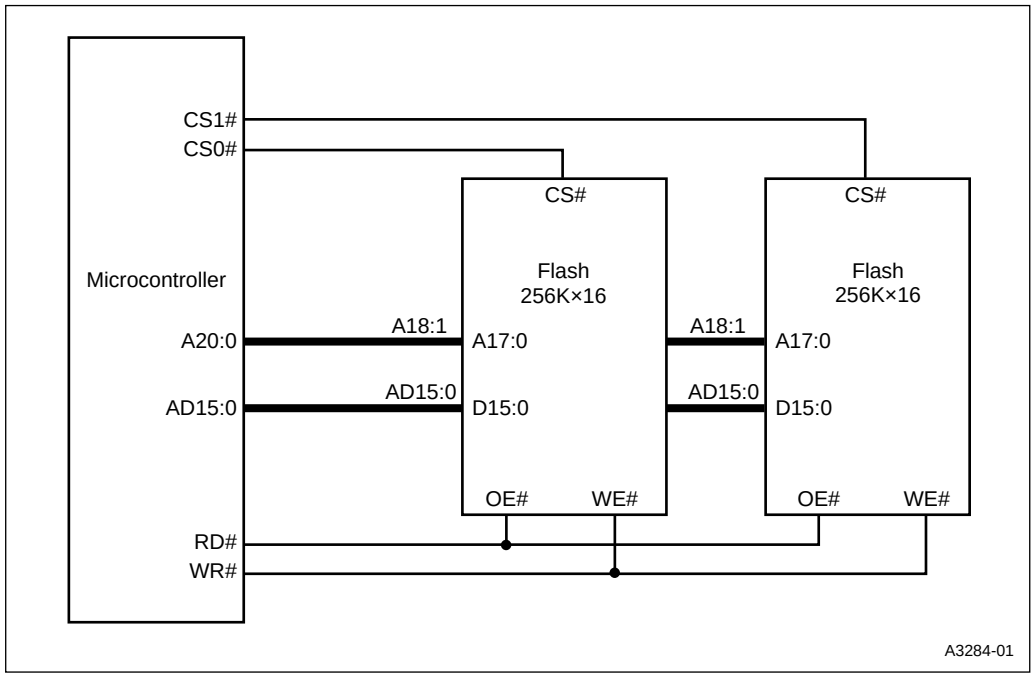


Figure 15-10. 16-bit External Devices in Demultiplexed Mode

15.5.2 16-bit Bus Timings

Figure 15-11 shows idealized 16-bit external-bus timings for the microcontroller. The signals are divided into two groups: signals for a demultiplexed bus (top) and signals for a multiplexed bus (bottom). Several bus signals are omitted from the figure to focus on a comparison of multiplexed and demultiplexed buses. The timing parameters are addressed in "Comparison of Multiplexed and Demultiplexed Buses" on page 15-29. Comprehensive timing specifications for the microcontroller are shown in Figures 15-19 and 15-20.

CLKOUT and ALE are the same in multiplexed and demultiplexed buses. The CLKOUT period is as programmed in the CLKOUT_CON register (see "External Timing" on page 2-12). The figures assume the CLKOUT period is twice the internal oscillator period (2t). The bus cycles shown here, which have no wait states, require two CLKOUT periods (two state times, or 4t).

The rising edge of the address latch enable (ALE) signal indicates that the microcontroller is driving an address onto the bus (A20:16 and AD15:0 for a multiplexed bus, A20:0 for a demultiplexed bus). The microcontroller presents a valid address before ALE falls. In a multiplexed system, the ALE signal is used to strobe a transparent latch (such as a 74AC373), which captures the address from AD15:0 and holds it while data is transferred on AD15:0.

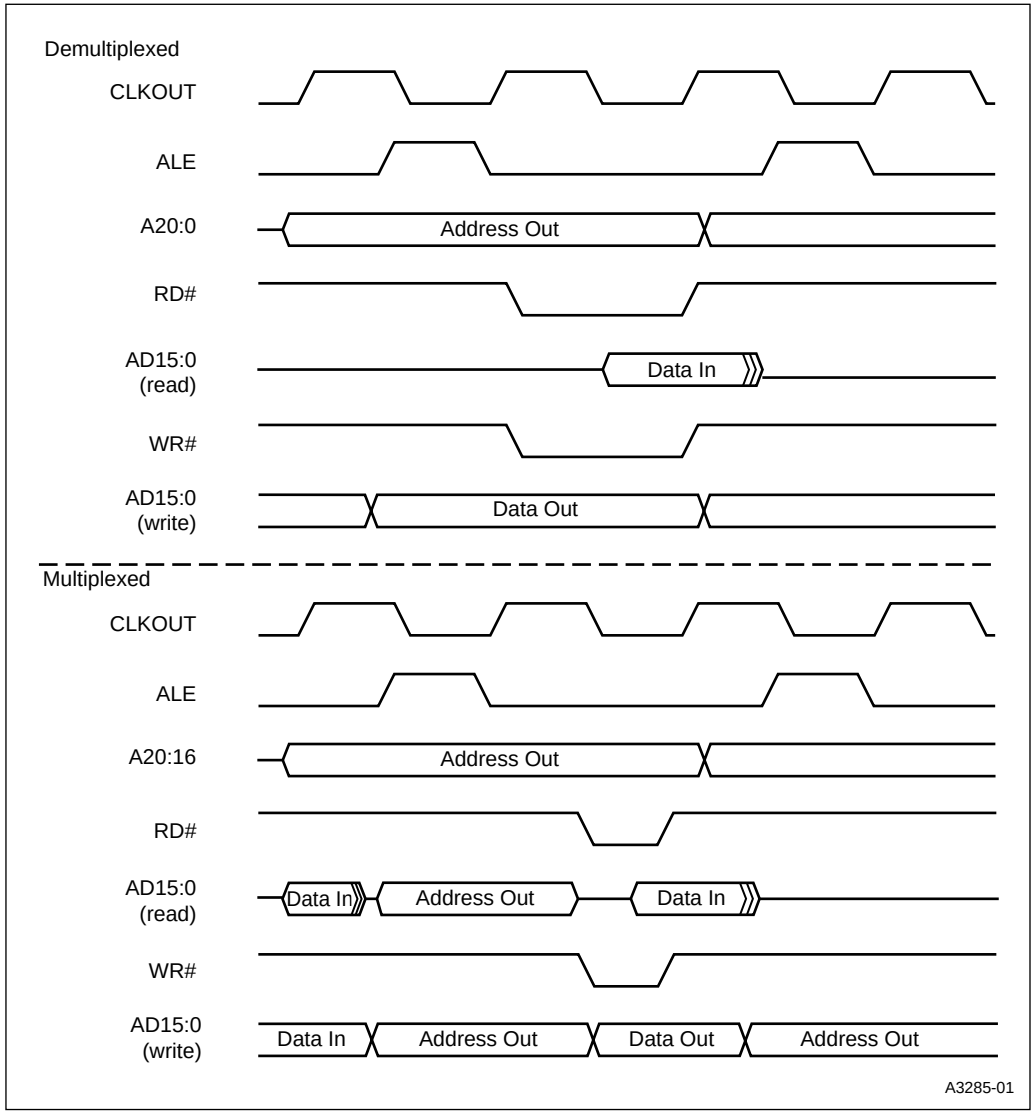


Figure 15-11. Timings for Multiplexed and Demultiplexed 16-bit Buses

15.5.3 8-bit Bus Timings

Figure 15-11 shows idealized 8-bit timings for the microcontroller. One cycle is required for an 8-bit read or write. A 16-bit access requires two cycles. The first cycle accesses the lower byte, and the second cycle accesses the upper byte. Except for requiring an extra cycle to write the bytes separately, the timings are the same as on the 16-bit bus, and the comparison between the multiplexed and demultiplexed cases is also the same. The demultiplexed bus can accommodate slower memory devices than the multiplexed bus can.

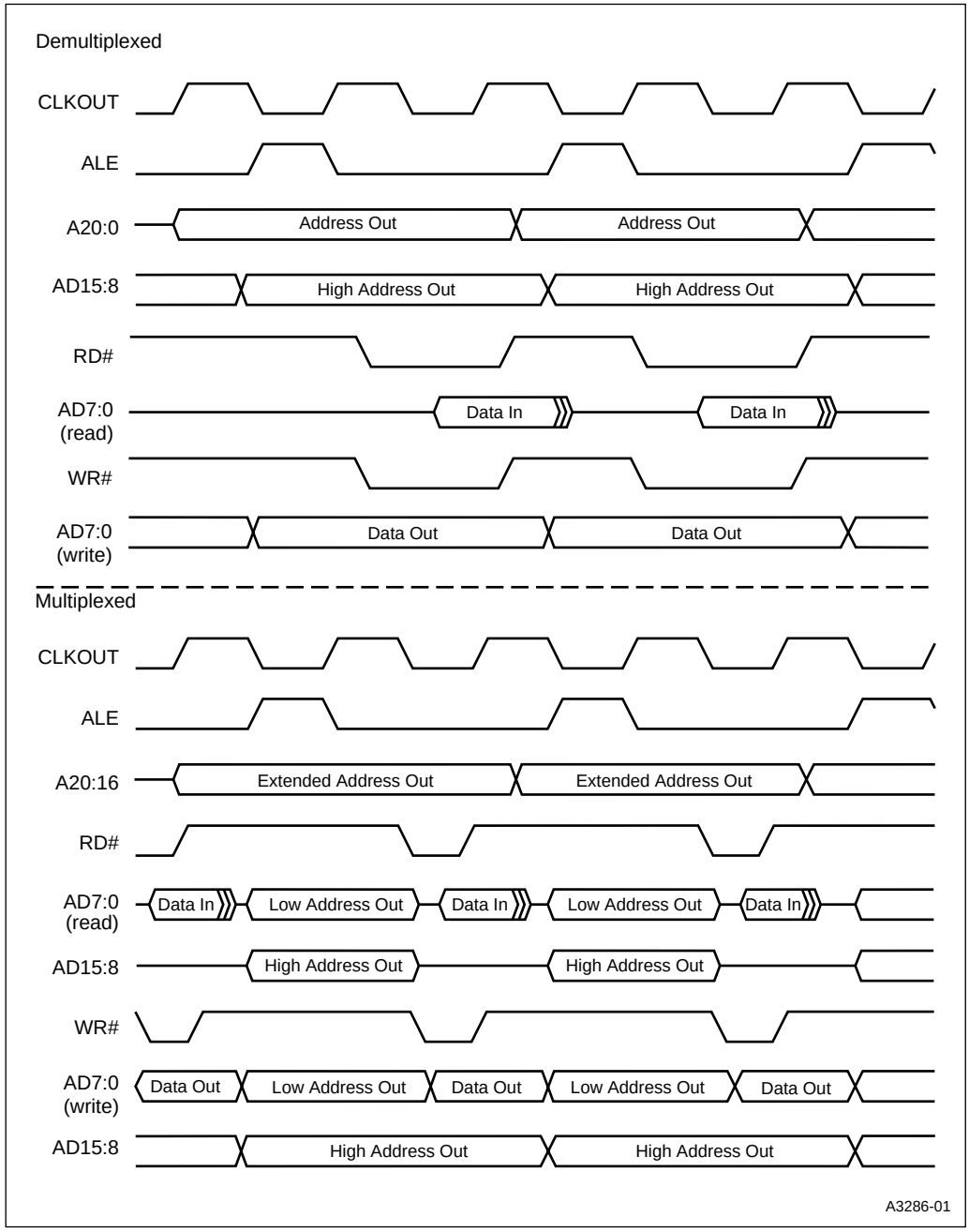


Figure 15-12. Timings for Multiplexed and Demultiplexed 8-bit Buses

15.5.4 Comparison of Multiplexed and Demultiplexed Buses

In a multiplexed system, where AD15:0 carry both address and data, bus activities are time-compressed in comparison with a demultiplexed system, where the address and data have separate pins (A20:0 and AD15:0). The compression is reflected in differences in specifications for the demultiplexed and multiplexed bus. The demultiplexed bus can accommodate slower memory devices than a multiplexed bus can. (Consult the datasheet for specifications.)

15.6 WAIT STATES (READY CONTROL)

An external device can use the READY input to lengthen an external bus cycle. When an external address is placed on the bus, the external device can pull the READY signal low to indicate it is not ready. In response, the microcontroller inserts wait states to lengthen the bus cycle until the external device asserts the READY signal. Each wait state adds one CLKOUT period to the bus cycle. The CLKOUT period is as programmed in the CLKOUT_CON register (see "External Timing" on page 2-12). The following figures assume the CLKOUT period is twice the internal oscillator period (2t).

The READY signal is effective for all bus cycles, including the CCB0 fetch (which has three internal wait states). Bits WS0 and WS1 in CCB0 specify the wait states for the CCB1 fetch. Thereafter, the WS1:0 bits in the BUSCON*x* registers control the wait states, and the READY signal can be used to insert additional wait states. (See "Controlling Bus Parameters" on page 15-13.)

The external device must meet setup and hold timings when using the READY signal to insert wait states into a bus cycle (see Figures 15-13 and 15-14 and Table 15-7). Because a decoded, valid address is used to generate the READY signal, the setup time is specified relative to the address being valid. This specification, T_{AVYV}, indicates how much time the external device has to decode the address and assert READY after the address is valid.

The external device must hold READY low until the minimum T_{CLYX} timing specification is met. Typically, this is a minimum of 0 ns from the time CLKOUT goes low. Do not exceed the maximum T_{CLYX} specification or additional (unwanted) wait states might be added. Refer to the datasheets for the current T_{AVYV} and T_{CLYX} specifications.

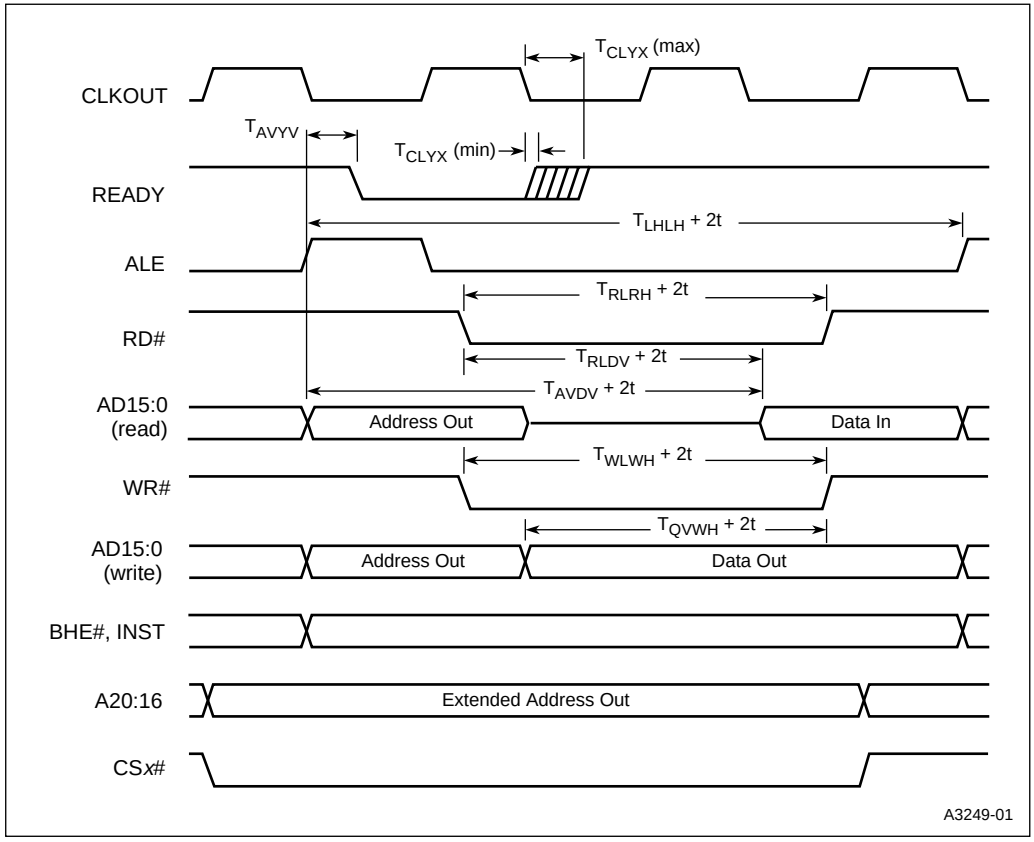


Figure 15-13. READY Timing Diagram — Multiplexed Mode

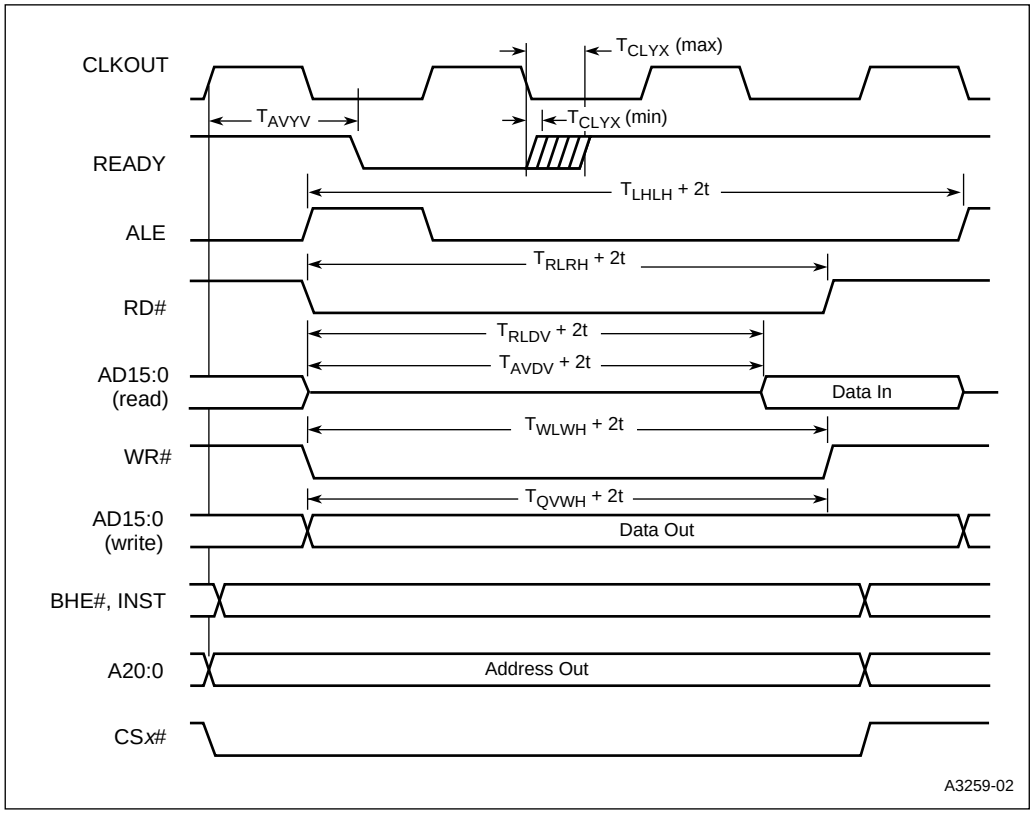


Figure 15-14. READY Timing Diagram — Demultiplexed Mode

Table 15-7. READY Signal Timing Definitions

Symbol	Definition
T_{AVDV}	Address Valid to Input Data Valid Maximum time the memory device has to output valid data after the microcontroller outputs a valid address.
T_{AVYV}	Address Valid to READY Setup Maximum time the external device has to pull READY low after the microcontroller outputs the address to guarantee that at least one wait state will occur.
T_{CHYX}	READY Hold after CLKOUT† High If maximum specification is exceeded, additional wait states may occur.
T_{CLYX}	READY Hold after CLKOUT† Low Minimum time the level of the READY signal must be valid after CLKOUT falls.
T_{LHLH}	ALE Cycle Time Minimum time between ALE pulses.
T_{QVWH}	Data Valid to WR# High Time between data being valid on the bus and the microcontroller deasserting WR#.
T_{RLDV}	RD# Low to Input Data Valid Maximum time the memory system has to output valid data after the microcontroller asserts RD#.
T_{RLRH}	RD# Low to RD# High RD# pulse width.
T_{WLWH}	WR# Low to WR# High WR# pulse width.

† Assumes CLKOUT is equal to twice the internal operating period (2t).

15.7 BUS-HOLD PROTOCOL

The microcontroller supports a bus-hold protocol that allows external devices to gain control of the address/data bus. The protocol uses three signals: HOLD#/P2.5 (bus-hold request), HLDA#/P2.6 (bus-hold acknowledge), and BREQ#/P5.4 (bus request).

When an external device wants to use the microcontroller bus, it asserts the HOLD# signal. The microcontroller samples HOLD# while CLKOUT is low. If HOLD# is asserted, the microcontroller responds by releasing the bus and asserting HLDA#. During this hold time, the address/data bus floats, and signals CS*x*#, ALE, RD#, WR#/WRL#, BHE#/WRH#, and INST are weakly held in their inactive states. Figure 15-15 shows the timing for the bus-hold protocol, and Table 15-8 lists the timing parameters and their definitions. Consult the datasheet for timing diagrams and specifications.

When the external device is finished with the bus, it relinquishes control by driving HOLD# high. In response, the microcontroller deasserts HLDA# and resumes control of the bus.

If the microcontroller has a pending external bus cycle while another device has control of the bus, it asserts BREQ# to request control of the bus. After the external device responds by releasing HOLD#, the microcontroller exits hold and then deasserts BREQ# and HLDA#.

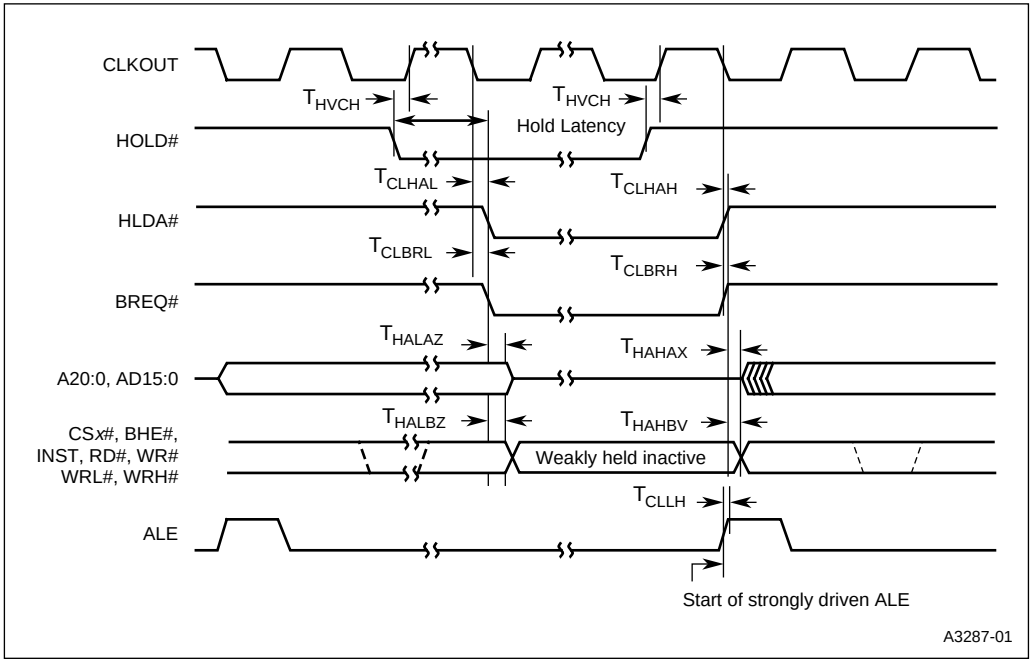


Figure 15-15. HOLD#, HLDA# Timing

Table 15-8. HOLD#, HLDA# Timing Definitions

Symbol	Parameter
T_{HVCH}	HOLD# Setup Time
T_{CLHAL}	CLKOUT† Low to HLDA# Low
T_{CLHAH}	CLKOUT† Low to HLDA# High
T_{CLBRL}	CLKOUT† Low to BREQ# Low
T_{CLBRH}	CLKOUT† Low to BREQ# High
T_{HALAZ}	HLDA# Low to Address Float
T_{HAHAX}	HLDA# High to Address No Longer Float
T_{HALBZ}	HLDA# Low to CS*x*#, BHE#, INST, RD#, WR#, WRL#, WRH# Weakly Driven
T_{HAHBV}	HLDA# High to CS*x*#, BHE#, INST, RD#, WR#, WRL#, WRH# valid
T_{CLLH}	Clock Falling to ALE Rising

† Assumes CLKOUT is equal to twice the internal operating period (2t).

When the external device is finished with the bus, it relinquishes control by driving HOLD# high. In response, the microcontroller deasserts HLDA# and resumes control of the bus.

If the microcontroller has a pending external bus cycle while another device has control of the bus, it asserts BREQ# to request control of the bus. After the external device responds by releasing HOLD#, the microcontroller exits hold and then deasserts BREQ# and HLDA#.

15.7.1 Enabling the Bus-hold Protocol

To use the bus-hold protocol, first configure P2.5/HOLD#, P2.6/HLDA#, and P5.4/BREQ# as special-function signals. (BREQ# and HLDA# are active-low outputs; HOLD# is an active-low input.) To enable the bus-hold protocol, set WSR.7. Once WSR.7 is set, an attempt to reconfigure HOLD#, BREQ#, or HLDA# to serve as a general-purpose I/O signal is ignored until the bus-hold protocol is disabled (that is, until you clear WSR.7 or reset the microcontroller).

15.7.2 Disabling the Bus-hold Protocol

To disable hold requests, clear WSR.7. The microcontroller does not take control of the bus immediately after WSR.7 is cleared. Instead, it waits for the current hold request to finish and then disables the bus-hold feature and ignores any new requests until the bit is set again.

Sometimes it is important to prevent another device from taking control of the bus while a block of code is executing. One way to protect a code segment is to clear WSR.7 and then execute a JBC instruction to check the status of the HLDA# signal. The JBC instruction prevents the RALU from executing the protected block until current hold requests are serviced and the hold feature is disabled. This is illustrated in the following code:

```
          DI                          ;Disable interrupts to prevent
                                      ;code interruption
          PUSH WSR                    ;Disable hold requests and
          LDB  WSR,#1FH               ;window Port 2
WAIT:     JBC  P2_PIN,6, WAIT         ;Check the HLDA# signal. If set,
                                      ;add protected instruction here
          POP  WSR                    ;Enable hold requests
          EI                          ;Enable interrupts
```

15.7.3 Hold Latency

When an external device asserts HOLD#, the microcontroller finishes the current bus cycle and then asserts HLDA#. The time it takes the microcontroller to assert HLDA# after the external device asserts HOLD# is called *hold latency* (see Figure 15-15). Table 15-9 lists the maximum hold latency for each type of bus cycle.

Table 15-9. Maximum Hold Latency

Bus Cycle Type	Maximum Hold Latency (state times)
Internal execution or idle mode	1.5
16-bit external execution	2.5 + 1 per wait state
8-bit external execution	2.5 + 2 per wait state

15.7.4 Regaining Bus Control

While HOLD# is asserted, the microcontroller continues executing code until it needs to access the external bus. If executing from internal memory, it continues until it needs to perform an external memory cycle. If executing from external memory, it continues executing until the queue is empty or until it needs to perform an external data cycle. As soon as it needs to access the external bus, the microcontroller asserts BREQ# and waits for the external device to deassert HOLD#. After asserting BREQ#, the microcontroller cannot respond to any interrupt requests, including NMI, until the external device deasserts HOLD#. One state time after HOLD# goes high, the microcontroller deasserts HLDA# and, with no delay, resumes control of the bus.

If the microcontroller is reset while in hold, bus contention can occur. For example, a device without internal ROM would try to fetch the chip configuration byte from external memory after RESET# was brought high. Bus contention would occur because both the external device and the microcontroller would attempt to access memory. One solution is to use the RESET# signal as the system reset; then all bus masters (including the microcontroller) are reset at once. Chapter 14, "Special Operating Modes,"shows system reset circuit examples.

15.8 WRITE-CONTROL MODES

The microcontroller has two write-control modes: the standard mode, which uses the WR# and BHE# signals; and the write strobe mode, which uses the WRL# and WRH# signals. Otherwise, the two modes are identical. The modes are selected by chip configuration register 0 (Figure 15-6 on page 15-17.)

Figure 15-16 shows the waveforms of the asserted write-control signals in the two modes. Note that only BHE# is valid throughout the bus cycle.

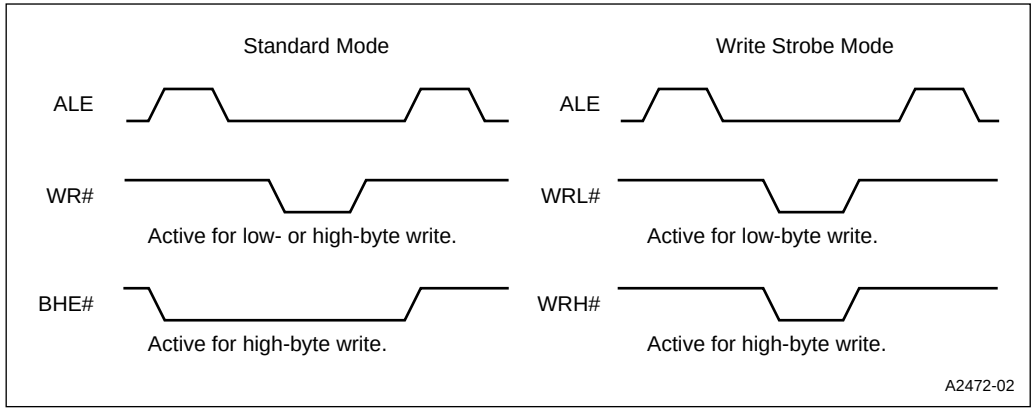


Figure 15-16. Write-control Signal Waveforms

compares the values of the write-control signals for write operations in the standard mode and the write strobe mode. The table lists values of WR# and BHE# and values of WRL# and WRH# for byte and word writes on a 16-bit and an 8-bit bus.

Table 15-10. Write Signals for Standard and Write Strobe Modes

Bus Width	Word/Byte Written	A0 or AD0†	Standard (CCR0.2 = 1)		Write Strobe (CCR0.2 = 0)	
			WR#	BHE#	WRL#	WRH#
16	Low Byte	0	0	1	0	1
	High Byte	1	0	0	1	0
	Word	0	0	0	0	0
		1	Illegal		Illegal	
8	Low Byte	0	0	1	0	1
	High Byte	1	0	1	0	1

† A0 for a demultiplexed address bus; AD0 for a multiplexed address/data bus.

To select the standard write-control mode, set CCR0.2. In standard mode, the WR#/WRL# pin operates as WR#, and the BHE#/WRH# pin operates as BHE#. WR# is asserted for every external memory write. BHE# is asserted for word accesses (read and write) and for byte accesses to odd addresses. BHE# can be used to select the bank of memory that stores the high (odd) byte. Figure 15-10 on page 15-25 illustrates use of the standard mode in a 16-bit system. In this example, WR# writes words to the 16-bit flash memory. To write individual bytes, you can use the decoding logic in Figure 15-17 or use the write strobe mode.

To write single bytes on a 16-bit bus requires separate low-byte and high-byte write signals (WRL# and WRH#). Figure 15-17 shows a sample circuit that combines WR#, BHE#, and address bit 0 (A0 for a demultiplexed address bus, AD0 for a mulitplexed address/data bus) to produce these signals. This additional logic is unnecessary, however. In the write strobe mode, WRL# and WRH# are available at the microcontroller's external pins.

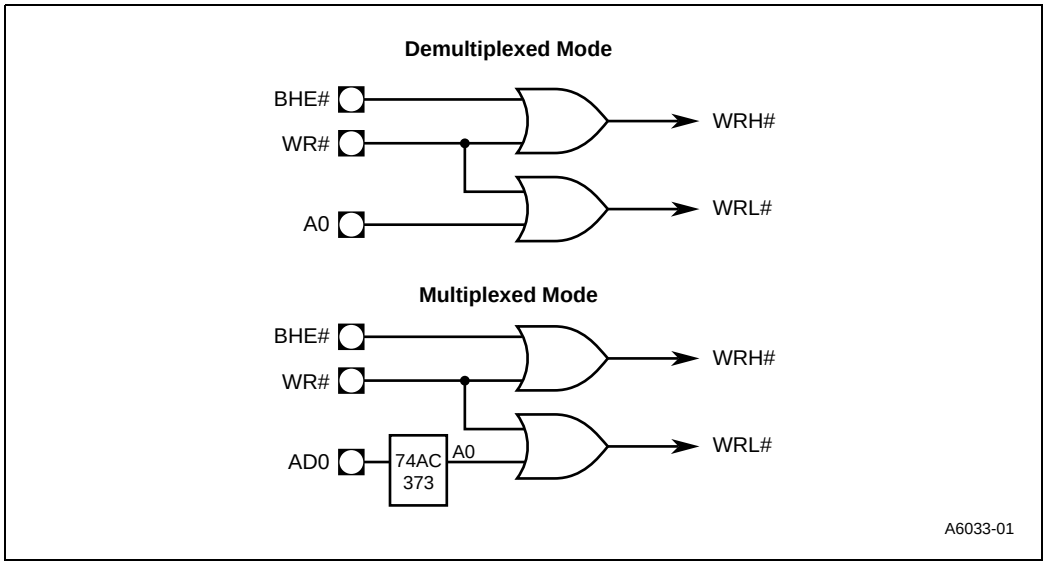


Figure 15-17. Decoding WRL# and WRH#

The write strobe mode eliminates the need to externally decode high-byte and low-byte write signals to external 16-bit memory on a 16-bit bus. When the write strobe mode is selected, the WR#/WRL# pin operates as WRL#, and the BHE#/WRH# pin operates as WRH#. In the 16-bit bus mode, WRL# is asserted for all low-byte writes (even addresses) and all word writes, and WRH# is asserted for all high-byte writes (odd addresses) and all word writes.

Figure 15-18 illustrates the use of the write strobe mode in a mixed 8-bit and 16-bit system with two flash memories and one SRAM. The WRL# signal, which is generated for all 8-bit writes (Table 15-10), is used to write bytes to the SRAM. Note that the RD# signal is sufficient for single-byte reads on a 16-bit bus. Both bytes are put onto the data bus and the memory controller discards the unwanted byte.

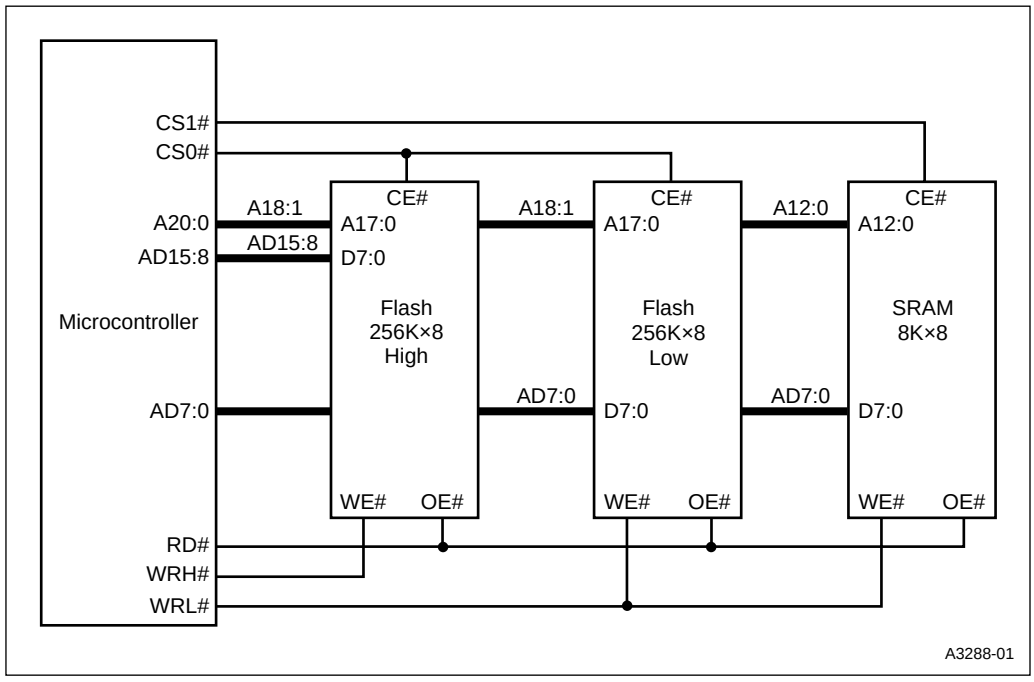


Figure 15-18. A System with 8-bit and 16-bit Buses

15.9 SYSTEM BUS AC TIMING SPECIFICATIONS

Refer to the latest datasheet for the AC timings to ensure your system meets specifications. The major external bus timing specifications are shown in Figures 15-19 and 15-20.

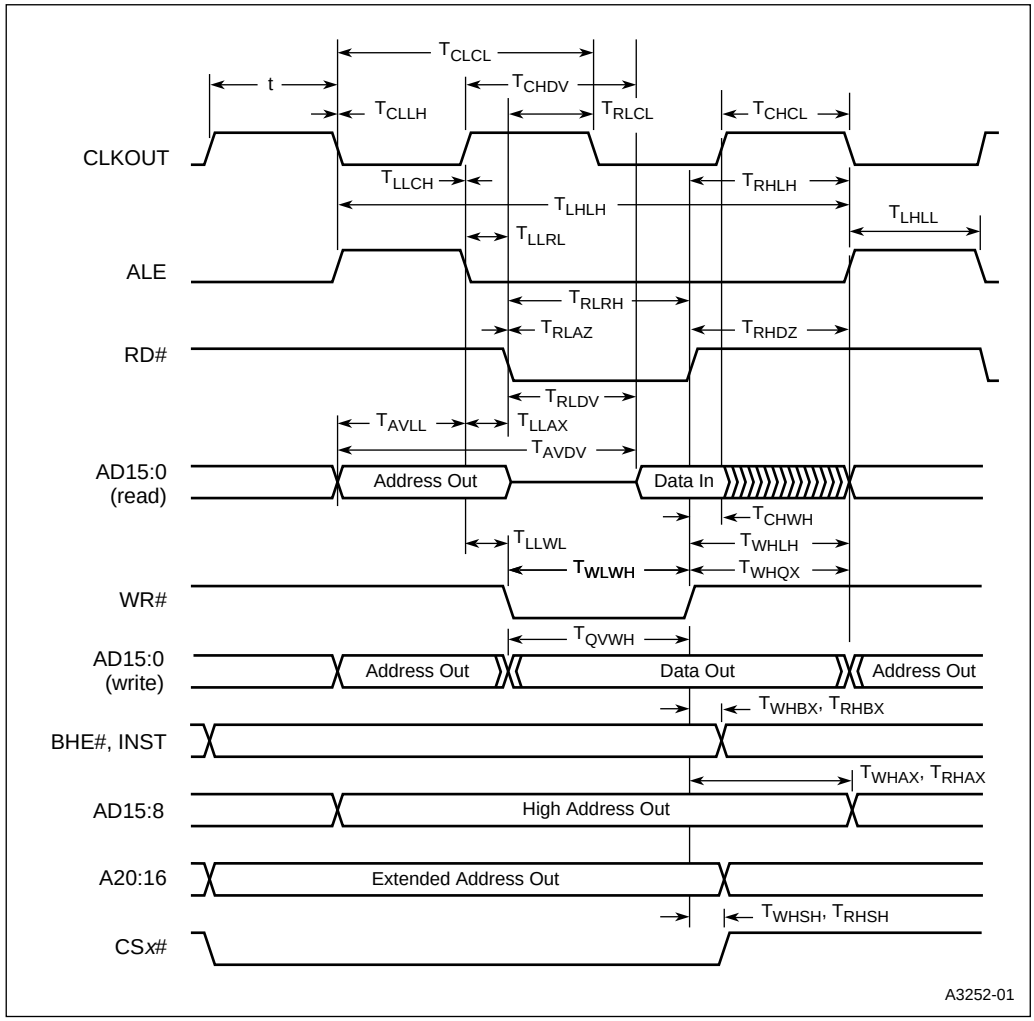


Figure 15-19. Multiplexed System Bus Timing

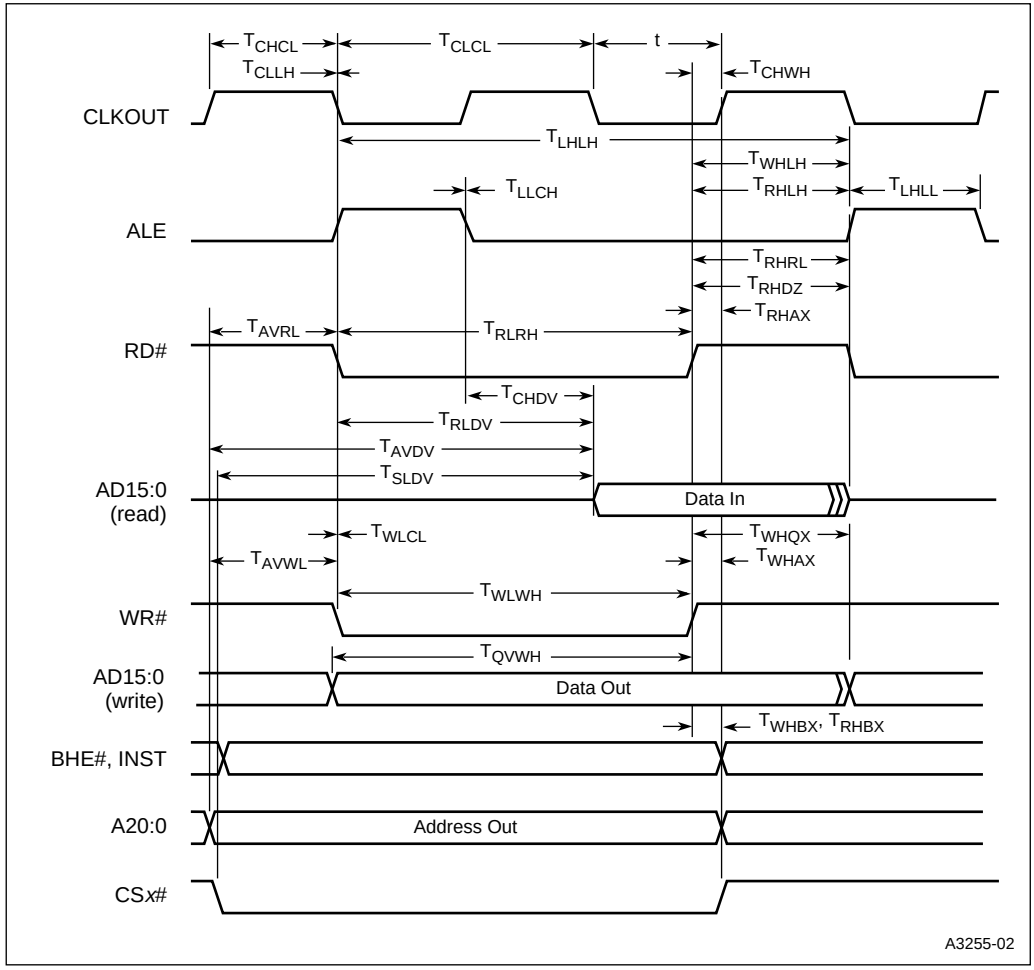


Figure 15-20. Demultiplexed System Bus Timing

15.9.1 Deferred Bus-cycle Mode

The microcontroller offers a deferred bus-cycle mode. This bus mode (enabled by CCR1.5; see Figure 15-7 on page 15-19) reduces bus contention when using the microcontroller in demultiplexed mode with slow memories. As shown in Figure 15-21, a delay of 2t occurs in the first bus cycle following a chip-select output change or the first write cycle following a read cycle.

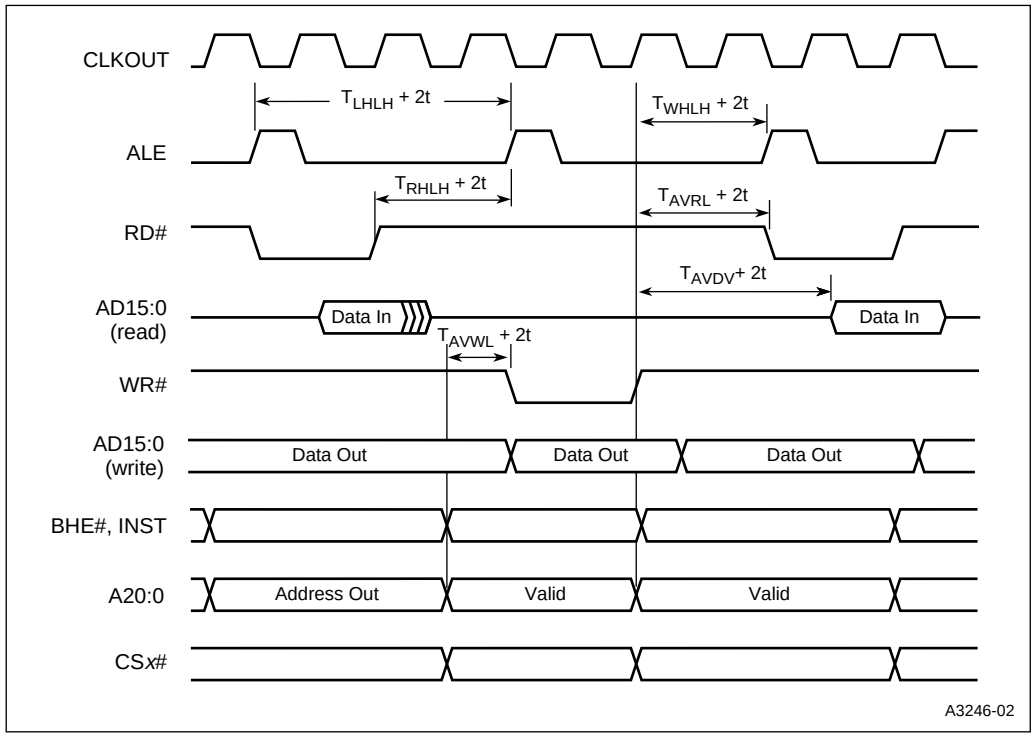


Figure 15-21. Deferred Bus-cycle Mode Timing Diagram

15.9.2 Explanation of AC Symbols

Each symbol consists of two pairs of letters prefixed by "T" (for time). The characters in a pair indicate a signal and its condition, respectively. Symbols represent the time between the two signal/condition points. For example, T_{LLRL} is the time between signal L (ALE) condition L (Low), and signal R (RD#) condition L (Low). Table 15-11 defines the signal and condition codes.

Table 15-11. AC Timing Symbol Definitions

A	AD15:0, A20:0
B	BHE#
C	CLKOUT
D	AD15:0, AD7:0, SD*x* (SSIO input data)
H	HOLD#
HA	HLDA#
L	ALE
Q	AD15:0, AD7:0, SD*x* (SSIO output data)
R	RD#
S	CS*x*#
W	WR#, WRL#

Character	Condition
H	High
L	Low
V	Valid
X	No Longer Valid
Z	Floating (low impedance)

15.9.3 AC Timing Definitions

Tables 15-12 and 15-13 define the AC timing specifications that the memory system must meet and those that the microcontroller will provide.

Table 15-12. External Memory Systems Must Meet These Specifications

Symbol	Definition
T_{AVDV}	Address Valid to Input Data Valid Maximum time the memory device has to output valid data after the microcontroller outputs a valid address.
T_{CHDV}	CLKOUT† High to Input Data Valid Maximum time the memory system has to output valid data after CLKOUT rises.
T_{QVWH}	Data Valid to WR# High Time between data being valid on the bus and the microcontroller deasserting WR#.
T_{RHDZ}	RD# High to Input Data Float Time after RD# is inactive until the memory system must float the bus. If this timing is not met, bus contention will occur.

† Assumes CLKOUT is equal to twice the internal operating period (2t).

Table 15-12. External Memory Systems Must Meet These Specifications (Continued)

Symbol	Definition
T_{RLDV}	RD# Low to Input Data Valid Maximum time the memory system has to output valid data after the microcontroller asserts RD#.

† Assumes CLKOUT is equal to twice the internal operating period (2t).

Table 15-13. The Microcontroller Meets These Specifications

f	Operating frequency Frequency of the signal input on the XTAL1 pin times the clock multiplier (*x*); *x* is 1 or 2 depending on the clock mode. The internal bus speed of the microcontroller is f/2.
t	Operating period (1/f) All AC Timings are referenced to t.
T_{AVLL}	Address Setup to ALE Low Length of time the address is valid before ALE falls. Use this specification when designing the external latch.
T_{AVRL}	Address Setup to RD# Low Length of time the address is valid before RD# falls.
T_{AVWL}	Address Setup to WR# Low Length of time the address is valid before WR# falls.
T_{CHCL}	CLKOUT† High Period Needed in systems that use CLKOUT as clock for external devices.
T_{CHWH}	CLKOUT† High to WR# High Time between CLKOUT going high and WR# going inactive.
T_{CLCL}	CLKOUT† Cycle Time Normally 2t.
T_{CLLH}	CLKOUT† Falling to ALE Rising Use to derive other timings.
T_{LHLH}	ALE Cycle Time Minimum time between ALE pulses.
T_{LHLL}	ALE High Period Use this specification when designing the external latch.
T_{LLAX}	Address Hold after ALE Low Length of time the address is valid after ALE falls. Use this specification when designing the external latch.
T_{LLCH}	ALE Falling to CLKOUT† Rising Use to derive other timings.

† Assumes CLKOUT is equal to twice the internal operating period (2t).

Table 15-13. The Microcontroller Meets These Specifications (Continued)

T_{LLRL}	ALE Low to RD# Low Length of time after ALE falls before RD# is asserted. Could be needed to ensure proper memory decoding takes place before a device is enabled.
T_{LLWL}	ALE Low to WR# Low Length of time after ALE falls before WR# is asserted. Could be needed to ensure proper memory decoding takes place before a device is enabled.
T_{RHAX}	(Multiplexed Mode) AD15:8/CS*x*# Hold after RD# High Minimum time that the high byte of the address in 8-bit mode will be valid after RD# inactive. (Demultiplexed Mode) A20:0/CS*x*# Hold after RD# High Minimum time that the address will be valid after RD# inactive.
T_{RHBX}	BHE#, INST Hold after RD# High Minimum time that these signals will be valid after RD# inactive.
T_{RHLH}	RD# High to ALE Rising Time between the microcontroller deasserting RD# and the next ALE. Useful in calculating time between RD# inactive and next address valid.
T_{RLAZ}	RD# Low to Address Float Used to calculate when the microcontroller stops driving the address on the bus.
T_{RLCL}	RD# Low to CLKOUT† Low Length of time from RD# asserted to CLKOUT falling edge.
T_{RLRH}	RD# Low to RD# High RD# pulse width.
T_{WHAX}	(Multiplexed Mode) AD15:8/CS*x*# Hold after WR# High Minimum time that the high byte of the address in 8-bit mode will be valid after WR# inactive. (Demultiplexed Mode) A20:0/CS*x*# Hold after WR# High Minimum time that the address will be valid after WR# inactive.
T_{WHBX}	BHE#, INST Hold after WR# High Minimum time that these signals will be valid after WR# inactive.
T_{WHLH}	WR# High to ALE High Time between the microcontroller deasserting WR# and next ALE. Also used to calculate WR# inactive and next Address valid.
T_{WHQX}	Data Hold after WR# High Minimum time after WR# rises that the data stays valid on the bus.
T_{WHSH}	A20:0/CS*x*# Hold after WR# High Minimum time that the address and chip-select output are held after WR# inactive.
T_{WLCL}	WR# Low to CLKOUT† Low Minimum and maximum time between WR# being asserted and CLKOUT going low.
T_{WLWH}	WR# Low to WR# High WR# pulse width.

† Assumes CLKOUT is equal to twice the internal operating period (2t).

16

Serial Debug Unit

CHAPTER 16
SERIAL DEBUG UNIT

The serial debug unit (SDU) is a new design for the MCS® 96 microcontroller architecture. This new peripheral allows you to read and write the contents of the code RAM using a high-speed dedicated serial link.

The SDU peripheral has four objectives:

- to provide a real-time method for developing and debugging code with no CPU overhead
- to provide a simple user interface to the device without requiring extensive external hardware
- to support reading and writing of all internal and external memory during interrogation mode
- to support breakpoints in both internal and external memory development systems

This chapter explains and illustrates how to transfer data to and from the code RAM. Examples using the SDU command instruction set and the reduced instruction set monitor (RISM) routine are provided for clarification.

16.1 SERIAL DEBUG UNIT (SDU) FUNCTIONAL OVERVIEW

The SDU (Figure 16-1) is a simple four-pin interface peripheral. It is similar to, and can communicate with, other synchronous serial I/O (SSIO) devices. It allows the code RAM to function as a dual-port RAM. This is because it can read from and write to the code RAM across a serial link, without CPU overhead.

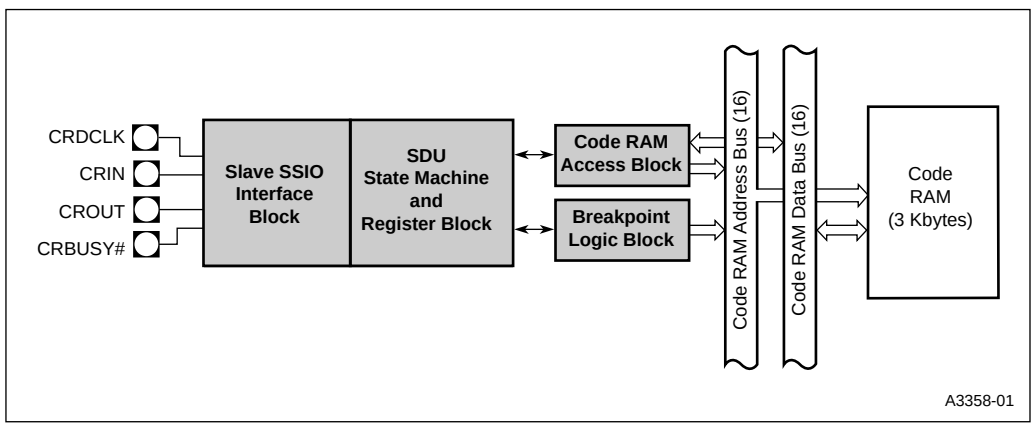


Figure 16-1. SDU Block Diagram

The SDU module consists of four main functional blocks:

- the slave SSIO interface block, which controls communication to and from the SDU
- the SDU state machine and register block, which decodes the SDU instruction set and contains the data registers
- the code RAM access block, which permits the SDU to read or write the code RAM without CPU intervention
- the breakpoint logic block, which permits the insertion of hardware breakpoints during the code RAM interrogation

16.2 SDU SIGNALS AND REGISTERS

Table 16-1 describes the SDU signals and Table 16-2 describes the control register.

Table 16-1. SDU Signals

SDU Signal(s)	SDU Signal Type	Description
CRBUSY#	O	Code RAM Busy When active, this signal indicates the SDU is busy processing a code RAM command. No data can be transferred during this time.
CRDCLK	I	Code RAM Data Clock Provides the clock signal for the SDU. The maximum clock frequency is one-half the operating frequency (f/2).
CRIN	I	Code RAM Data Input Serial input for test instructions and data into the SDU. Data is transferred eight bits at a time with the most-significant bit (MSB) first. Each bit is sampled and latched on the rising edge of CRDCLK.
CROUT	O	Code RAM Data Output Serial output for data from the SDU. Data is transferred eight bits at a time with the MSB first. Each data bit changes one CPU state time after the rising edge of CRDCLK.

Table 16-2. SDU Control Register

Mnemonic	Description
SDU_COM	SDU Command Register This byte register determines the instruction the SDU will execute, the data direction, and the size (byte or word) of the transaction. It can also force an interrupt to the CPU for accessing the RISM.

16.3 SDU OPERATION

The SDU functional block diagram (Figure 16-2) illustrates the operation of the SDU module.

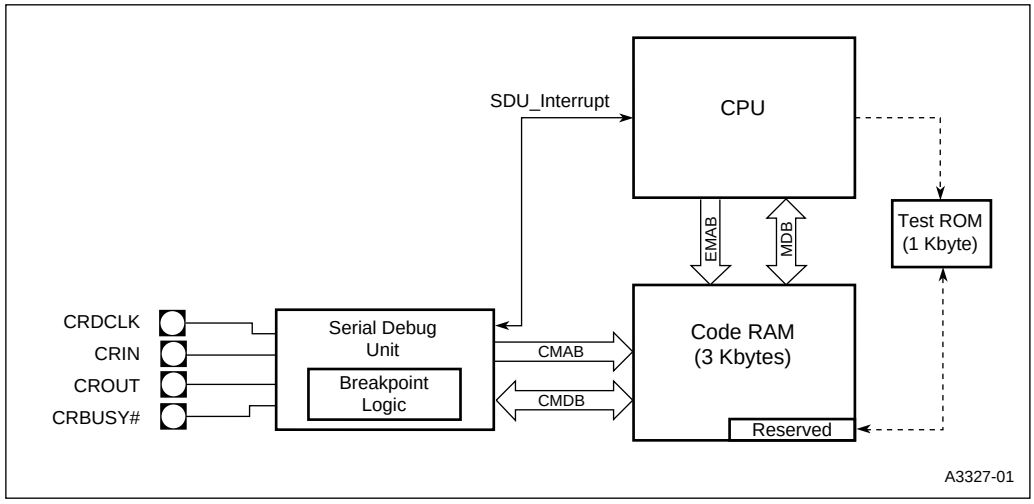


Figure 16-2. SDU Functional Block Diagram

Three internal data registers control the operation of the SDU:

- The code RAM address access (CR_ADDR) register specifies the code RAM address to be read or written. This register automatically increments after each transfer.
- The code RAM data access (CR_DATA) register functions as an internal buffer to transfer data to and from the code RAM.
- The breakpoint address access (BP_ADDR) register specifies the memory address from which you want to generate a TRAP instruction, instead of executing code.

The SDU uses the command (SDU_COM) register as a temporary storage location for command information that the master shifts in serially one bit at a time. The data transfer protocol for the SDU is always the command byte followed by up to four data bytes. A command byte counter tracks the number of data bytes expected until the next command byte.

The clock rate of the SDU is one-half the internal operating frequency (f/2). With a 32 MHz external clock (the maximum frequency), the internal operating frequency (f) is 16 MHz, so the SDU operates at 8 MHz.

16.3.1 SDU State Machine

The SDU state machine diagram (Figure 16-3) illustrates the serial data transfer decision-making process.

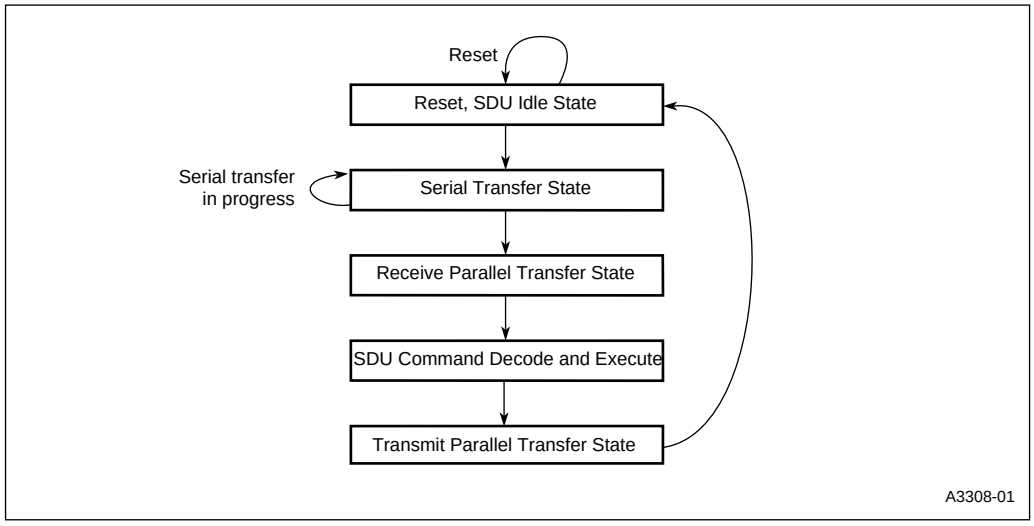


Figure 16-3. SDU State Machine Diagram

The following is a step-by-step account of the internal SDU operation as outlined in Figure 16-3:

1. The SDU begins in its idle state. When the first bit (the start bit) on CRIN is a zero and a rising edge is detected on CRDCLK, a serial data transmission starts. For synchronization purposes, a start bit of zero indicates that the first data frame (eight bits) being shifted in is a valid command byte.

   NOTE

   If the SDU was left in an unknown state and your application is attempting to re-establish communication with the SDU, the master must send it a synchronization sequence. The synchronization sequence consists of six consecutive command bytes containing the reset SDU instruction, 01111111B (7FH). This six-command sequence resynchronizes the SDU, regardless of its last state.

2. Data is shifted out of the transmit buffer and into the receive buffer at the same time.
3. After the transfer is completed, data in the receive buffer is moved to an internal register (command register, data register, or address register) pointed to by a receive pointer. The handshake signal, CRBUSY#, is asserted.

4. If the new data frame is a command, it is decoded and the command byte counter is updated.
   - If the new command is a code RAM data access command, control is transferred to the code RAM access state machine before resuming onto the next step. (Refer to “Code RAM Access State Machine” on page 16-5.)
   - If the new data frame is **not** a command byte, the receive and transmit pointers are updated.
5. Data from the register accessed by the last command is loaded into the transmit buffer. After the transmit buffer is loaded, the CRBUSY# signal is deasserted and the SDU returns to its idle state.

16.3.2 Code RAM Access State Machine

The code RAM access state machine diagram (Figure 16-4) illustrates the decision-making process of the code RAM read/write operation.

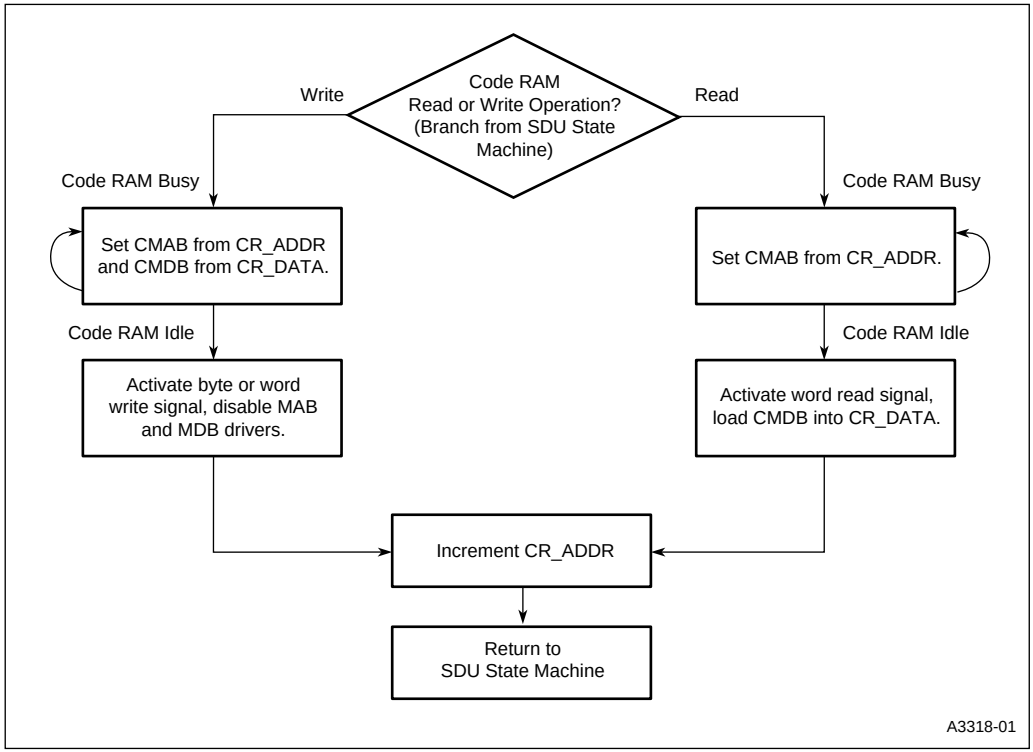


Figure 16-4. Code RAM Access State Machine Diagram

The code RAM data access command allows the SDU to read and write the code RAM without CPU intervention. The following is a step-by-step account of the read and write operation as outlined in Figure 16-4:

1. Perform a code RAM read or write operation.
   - If SDU_COM.1 is set, a write operation is performed.

     The memory address is loaded onto the code RAM memory address bus (CMAB) from the CR_ADDR register.

     Data is loaded onto the code RAM memory data bus (CMDB) from the CR_DATA register.
   - If SDU_COM.1 is cleared, a read operation is performed.

     The memory address is loaded onto the code RAM memory address bus (CMAB) from the CR_ADDR register.
2. Wait at least two CPU states for any user application code that is currently executing from the code RAM to complete.
3. Once the code RAM enters an idle state, the byte/word bit is activated (SDU_COM.0).
   - For a write operation, bit zero set indicates a byte transfer and bit zero clear indicates a word transfer. The memory address bus (MAB) and memory data bus (MDB) drivers to the code RAM are disabled.
   - For a read operation, a word transfer is performed by loading the CMDB into the CR_DATA register. The read operation does **not** support byte transfers.
4. The CR_ADDR register is automatically incremented to set up the next data transfer.
5. Control is returned to the SDU state machine.

16.3.3 Minimizing Latency

The SDU can access the code RAM only when the bus controller is starting an access to memory other than code RAM (i.e., Flash or external memory). Accesses to register RAM do not count in most cases, since the CPU directly accesses the register file. Therefore, to minimize latency, it is necessary to understand what the bus controller is doing in the user application at all times.

16.4 CODE RAM ACCESS

Communication to the code RAM through the SDU is accomplished using a limited SSIO format configuration with an additional handshaking operation. (Refer to Table 16-1 on page 16-2 for a description of the four pins required to interface to the SDU.) The SSIO of the SDU module has no master function. It can function only as a slave, allowing communication over a serial link in a bidirectional, single-byte transfer mode. All data transfers must be initiated by the clock source of an external master. Figure 16-5 illustrates a possible interface scheme using an 8XC196K*x* SSIO master to control the 8XC196EA SDU slave.

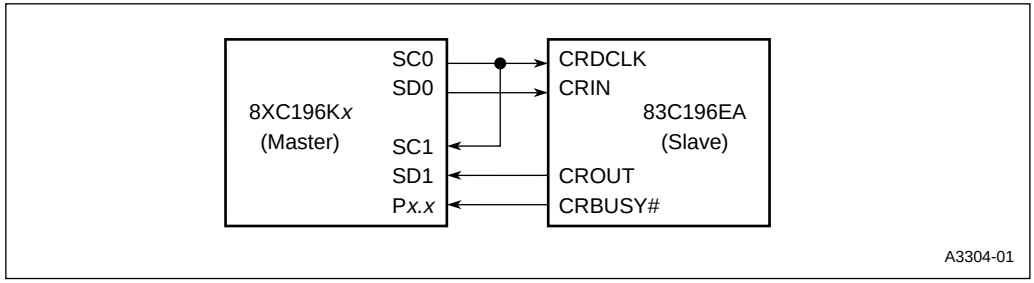


Figure 16-5. SDU Master/Slave Configuration

Figure 16-6 shows the standard 8-bit data-in (DI*x*) and data-out (DO*x*) transfer frame. The MSB is sent first, followed by another seven bits before the handshaking is asserted. Handshaking prevents a data underflow or overflow from occurring at the slave. The code RAM busy (CRBUSY#) signal serves this purpose for the slave. The CRBUSY# signal activates automatically one to two state times after the rising edge of the code RAM clock (CRDCLK) edge corresponding to the last data bit of the transmitted 8-bit packet. Each data bit is sampled and latched on the rising edge of CRDCLK.

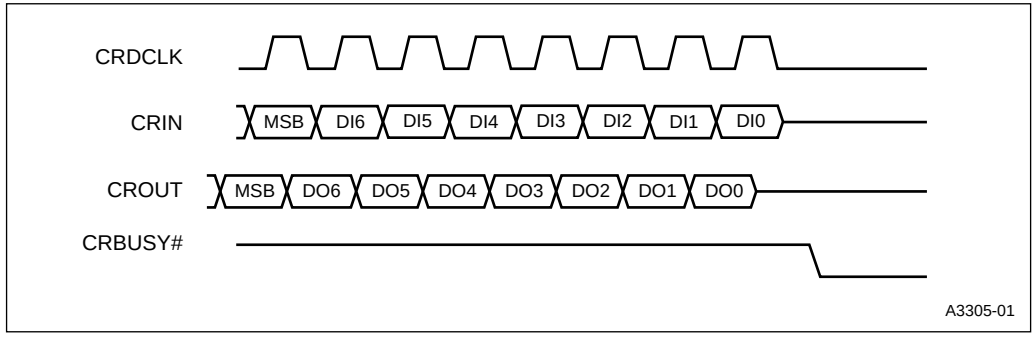


Figure 16-6. SDU Transmit/Receive Timings

16.4.1 Code RAM Data Transfer

The serial transfer of data frames using the SDU must be initiated by an external master device. The structure of a code RAM data transfer must be an eight-bit command byte followed by a specific number of bytes. The command instruction register, SDU_COM, specifies the data length.

Traffic control across the serial link is maintained by the handshaking pin, CRBUSY#. The SDU asserts the CRBUSY# signal to indicate that it is processing a command. The CRBUSY# signal toggles low for a few (three or fewer) states after every eight-bit command and data frame. This allows the state machine time to decode the command and to determine the register destination of a particular byte. (Refer to "Code RAM Data Transfer Example" on page 16-12 for a detailed example of the code RAM message structure.)

16.4.2 Code RAM Access Instructions

Figure 16-7 describes the SDU command byte, and Table 16-3 describes the code RAM access instructions.

SDU_COM

The serial debug unit command byte (SDU_COM) register specifies the instruction, the transfer direction, and the data size (byte or word). The command byte must precede each data frame string in the serial transfer.

7							0
SB	SDU_INT	MS3	MS2	MS1	MS0	W/R#	B/W#

Bit Number	Bit Mnemonic	Function
7	SB	Start Bit Clear this bit to indicate a valid command byte for synchronization.
6	SDU_INT	SDU Interrupt This bit is used to generate an interrupt to the CPU. 0 = disable interrupt 1 = enable interrupt
5:2	MS3:0	Mode Select These bits specify which instruction the SDU will execute. Refer to Table 16-3 for the complete set of instructions.

Figure 16-7. Serial Debug Unit Command Byte (SDU_COM) Register

SDU_COM

The serial debug unit command byte (SDU_COM) register specifies the instruction, the transfer direction, and the data size (byte or word). The command byte must precede each data frame string in the serial transfer.

7 **0**

SB	SDU_INT	MS3	MS2	MS1	MS0	W/R#	B/W#

Bit Number	Bit Mnemonic	Function
1	W/R#	Write/Read This bit specifies the transfer direction. 0 = read data from code RAM 1 = write data to code RAM
0	B/W#	Byte/Word This bit specifies the data transfer size. 0 = word 1 = byte

Figure 16-7. Serial Debug Unit Command Byte (SDU_COM) Register (Continued)

Table 16-3. Code RAM Access Instructions

Mode Select†				Instruction
MS3	MS2	MS1	MS0	
0	0	0	0	Reset SDU The reset SDU instruction resets all the SDU circuitry and reinitializes all the registers to their default value. If the SDU was left in an unknown state and your application is attempting to re-establish communication with the SDU, the master must send it a synchronization sequence. The synchronization sequence consists of six consecutive command bytes containing the reset SDU instruction 01111111B (7FH). This six-command sequence resynchronizes the SDU, regardless of its last state. The reset SDU instruction will not reset the microcontroller CPU.
0	0	0	1	Code RAM Address Access The code RAM address access instruction allows access to the code RAM address access (CR_ADDR) register. This instruction, in conjunction with the 16-bit CR_ADDR register, is used to set up the address for the code RAM access.
0	1	0	0	Code RAM Data Access The code RAM data access instruction allows access to the code RAM data access (CR_DATA) register. This instruction, in conjunction with the 16-bit CR_DATA register, is used for data transfers to and from the code RAM.
0	1	0	1	Breakpoint Address Access The breakpoint address access instruction is used to write to the breakpoint address access (BP_ADDR) register. This instruction, in conjunction with the 24-bit BP_ADDR register, is used to specify the address for inserting a hardware breakpoint.
0	1	1	0	Breakpoint Enable The breakpoint enable instruction enables immediate breakpoint generation when the address on the internal extended memory address bus (EMAB) matches the value in BP_ADDR. The breakpoint logic (Figure 16-8) includes a comparator that resides on the EMAB. If the breakpoint instruction has been programmed, the breakpoint logic compares the address on the bus with the address in the BP_ADDR register. A match sets an internal flag that exists as a ninth bit in the instruction queue. When the flagged instruction is transferred from the instruction queue into the instruction register, the CPU executes a TRAP instruction and vectors to the TRAP interrupt service routine at FF2010H (see "Breakpoint Logic Block Diagram" on page 16-11).
0	1	1	1	Breakpoint Disable The breakpoint disable instruction disables the breakpoint generation.

† All other bit combinations are reserved

Table 16-3. Code RAM Access Instructions (Continued)

Mode Select†				Instruction
MS3	MS2	MS1	MS0	
1	0	1	0	SDU Idle (NOP) The idle instruction puts the SDU into an idle state, causing it to wait for another instruction without performing any operation. The idle instruction is primarily used to write the high byte of data after a word read. For example, if during a word read operation a command byte is shifted in, the SDU can shift out only eight bits, which is assumed to be the high data byte. Eight additional clocks are required to shift out another data frame. To read out the next data byte, which is the low data byte, without issuing a new command, use the idle instruction.
1	1	1	1	Reset SDU The reset SDU instruction resets all the SDU circuitry and reinitializes all the registers to their default value. If the SDU was left in an unknown state and your application is attempting to re-establish communication with the SDU, the master must send it a synchronization sequence. The synchronization sequence consists of six consecutive command bytes containing the reset SDU instruction 01111111B (7FH). This six-command sequence resynchronizes the SDU, regardless of its last state. The reset SDU instruction will not reset the microcontroller CPU.

† All other bit combinations are reserved

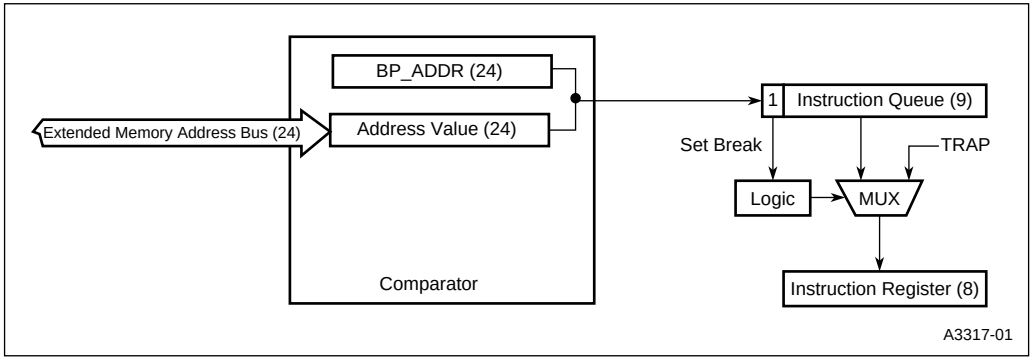


Figure 16-8. Breakpoint Logic Block Diagram

16.4.3 Code RAM Data Transfer Example

The following example, illustrated in Figure 16-9, reads two words at address locations 480H and 482H from the code RAM:

1. Shift in SDU command 06H to select the code RAM address access instruction.
2. Shift the high byte (04H) of the address, followed by the low byte (80H) of the address into the CR_ADDR register.
3. Shift in SDU command 10H to select the code RAM word read instruction.
4. The SDU asserts CRBUSY#, indicating that it is processing a code RAM command. Wait for CRBUSY# to be deasserted, indicating that the data is ready.
5. Shift in SDU command 10H to read the next word. CD_ADDR will automatically increment. (As this command byte is shifted in, the high byte of data0 will be shifted out on CROUT.)
6. Shift in SDU command 28H to put the SDU into an idle state. This shift causes the low byte of data0 to be shifted out on CROUT.
7. Shift in SDU command 00H to reset the SDU and terminate the data transfer. This shift causes the high byte of data1 to be shifted out on CROUT.
8. Shift in SDU command 28H to put the SDU into an idle state. This shift causes the low byte of data1 to be shifted out on CROUT.

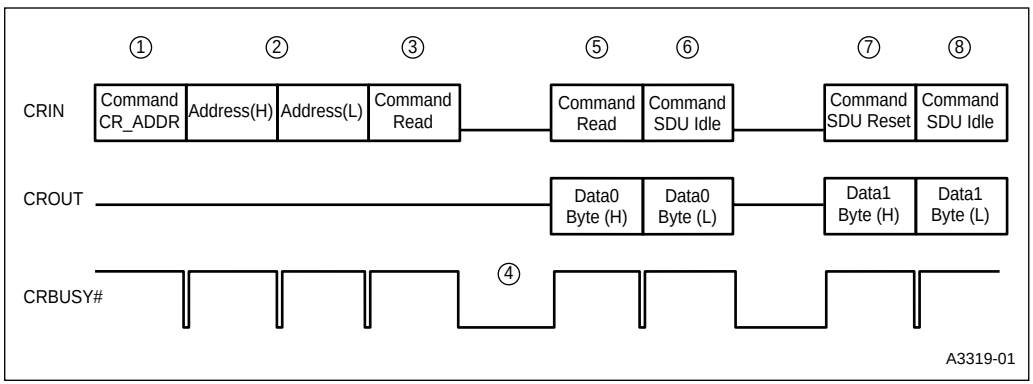


Figure 16-9. Code RAM Word Read Sequence

Figure 16-10 is an example of a code RAM byte write sequence.

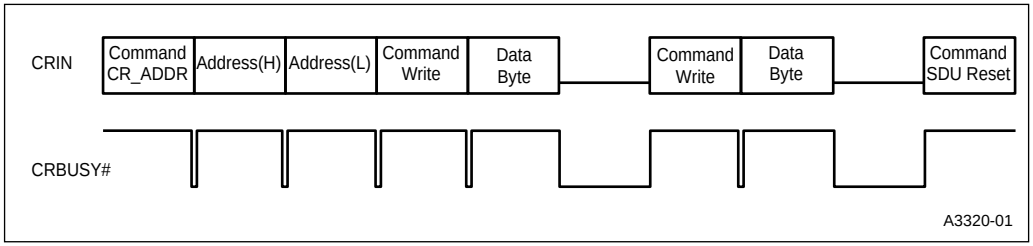


Figure 16-10. Code RAM Byte Write Sequence

16.5 SDU INTERFACE CONNECTOR

To standardize the connector interface to the SDU port, we recommend that you use a 10-pin, 2-row I/O expansion connector. This low cost, readily available standard connector can be easily added to any printed-circuit board (PCB) after manufacturing of the module.

Figure 16-11 describes the SDU port connector pinout.

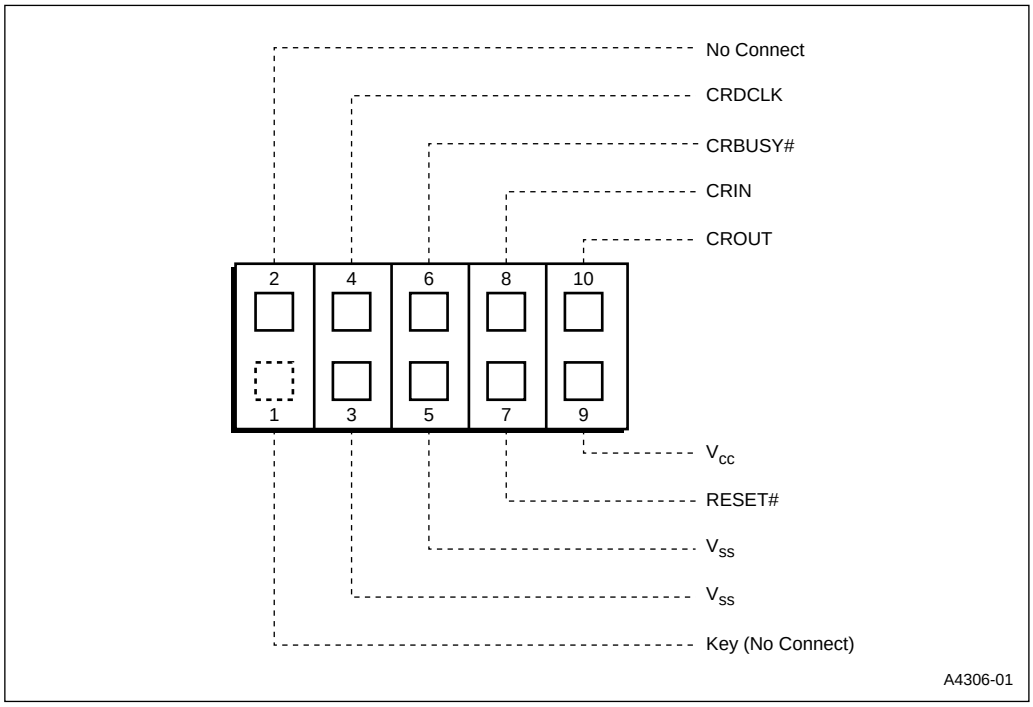


Figure 16-11. SDU Interface Connector

To prevent mis-socketing and ensure proper cable connection, we recommend that you remove the square pin at position #1 (Key) and block the adjoining cable position.

17

Using the Test-ROM Routines

CHAPTER 17
USING THE TEST-ROM ROUTINES

Within the memory array of the 8XC196EA resides 1 Kbyte of nonerasable memory, commonly referred to as test-ROM. Embedded in the test-ROM are three software routines: ROM-dump, serial port, and serial debug unit (SDU) RISM execution. These routines enable you to perform the following operations:

- Use the ROM-dump routine to write the entire ROM array to an external memory device to verify its contents.
- Use the serial port routine to download code and data (usually from a personal computer or workstation) to a microcontroller (the slave) through the serial I/O port.
- Use the SDU RISM execution routine to download code and data across a high-speed dedicated serial link.

17.1 SIGNALS AND REGISTERS

Table 17-1 describes the signals used in test-mode entry, and Figure 17-1 describes chip configuration register 0 (CCR0), which controls access to the ROM. Several pins combine to control entry into these modes, and three port 12 pins (P12.2:0) select the desired test-ROM routine. The serial port mode and ROM-dump routines configure additional pins, which are required to execute properly.

Table 17-1. Signal Descriptions

Port Pin	Special-function Signal	Type	Mode	Description
P12.2:0	—	I	All	Test-ROM Routine Select† Determines the test-ROM routine. These pins are sampled after a device reset and must be static while the part is operating. **P12.2 P12.1 P12.0** 0 0 0 Serial port mode 1 0 0 ROM-dump 1 1 1 SDU RISM execution † All other pin combinations are reserved.
P2.0	TXD0	O	Serial	Transmit Serial Data 0 During serial port mode, TXD0 transmits data to an external device.
P2.1	RXD0	I	Serial	Receive Serial Data 0 During serial port mode, RXD0 receives data from an external device.

Table 17-1. Signal Descriptions (Continued)

Port Pin	Special-function Signal	Type	Mode	Description
P1.2:1, P4.5:0, P3.7:0	PBUS	I/O	ROM-dump	Address/Command/Data Bus During ROM-dump, these pins serve as a regular system bus to access external memory. Leave P4.7:6 unconnected and connect P1.2:1 to serve as the upper address signals.
P5.4 P5.3 P5.0 PLLEN P2.0 P2.6 P5.2	TMODE# RD# ALE — TXD0 HLDA#/ONCE# WR#/WRL#	I I/O I/O I O I I/O	All	Test-mode Entry Pins To enter serial port mode, ROM-dump, or SDU RISM execution routine, you must hold four pins low during reset: P5.4, P5.3, P5.0, and PLLEN†. The remaining test-mode entry pins (P2.0, P2.6, and P5.2) must be high during reset. Because these pins have weak pull-ups internally, they will be in the correct state during reset. You need only ensure that your system does not drive these pins low. † PLLEN=0 disables the clock doubler, so the maximum frequency in test-ROM mode is F_{XTAL1}.

Table 17-2. Control and Status Register

Mnemonic	Description
CCR0	Chip Configuration 0 Register The chip configuration 0 (CCR0) register controls ROM access, enables or disables idle and powerdown modes, and selects the write-control mode. It also contains the bus-control parameters for fetching chip configuration byte 1.

17.2 MEMORY PROTECTION OPTIONS

The lock bit in chip configuration register 0 (CCR0) controls access to the ROM. Clearing CCB0.7 enables read protection. With read protection enabled, the bus controller will not read from protected areas of ROM. An attempt to read the ROM when the slave program counter contains an external address always returns F0FDH. Because the slave program counter can be as much as eight bytes ahead of the CPU program counter, the bus controller might prevent code execution from the last eight bytes of internal memory. The interrupt vectors and CCBs are **not** read protected because interrupts can occur even when executing from external memory. Figure 17-1 describes CCB0.

CCR0 no direct access†

The chip configuration 0 (CCR0) register controls ROM access, enables or disables idle and powerdown modes, and selects the write-control mode. It also contains the bus-control parameters for fetching chip configuration byte 1.

7							0
LOC	1	WS1	WS0	DEMUX	BHE#	BW16	PD

Bit Number	Bit Mnemonic	Function
7	LOC	Lock Bit This bit controls read access to the ROM during normal operation. 0 = read protect 1 = no protection Refer to “Controlling Read Access to the Internal ROM” on page 4-26 for details.
6	1	To guarantee proper operation, write one to this bit.
5:4	WS1:0	Wait States These bits, along with the READY pin, control the number of wait states that are used for an external fetch of chip configuration byte 1 (CCB1). **WS1 WS0** 0 0 zero wait states 0 1 one wait state 1 0 two wait states 1 1 three wait states If READY is low when this number is reached, additional wait states are added until READY is pulled high.
3	DEMUX	Select Demultiplexed Bus Selects the demultiplexed bus mode for an external fetch of CCB1: 0 = multiplexed — address and data are multiplexed on AD15:0. 1 = demultiplexed — data only on AD15:0.
2	BHE#	Write-control Mode Selects the write-control mode, which determines the functions of the BHE#/WRH# and WR#/WRL# pins for external bus cycles: 0 = write strobe mode: the BHE#/WRH# pin operates as WRH#, and the WR#/WRL# pin operates as WRL#. 1 = standard write-control mode: the BHE#/WRH# pin operates as BHE#, and the WR#/WRL# pin operates as WR#.
1	BW16	Buswidth Control Selects the bus width for an external fetch of CCB1: 0 = 8-bit bus 1 = 16-bit bus

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after reset. The CCBs reside at addresses FF2018H (CCB0) and FF201AH (CCB1

Figure 17-1. Chip Configuration 0 (CCR0) Register

CCR0 (Continued) no direct access†

The chip configuration 0 (CCR0) register controls ROM access, enables or disables idle and powerdown modes, and selects the write-control mode. It also contains the bus-control parameters for fetching chip configuration byte 1.

7							0
LOC	1	WS1	WS0	DEMUX	BHE#	BW16	PD

Bit Number	Bit Mnemonic	Function
0	PD	Powerdown Enable Enables or disables the IDLPD #1 and IDLPD #2 instructions. When enabled, the IDLPD #1 instruction causes the microcontroller to enter idle mode and the IDLPD #2 instruction causes the microcontroller to enter powerdown mode.0 = disable idle and powerdownmodes 1 = enable idle and powerdown modes If your design uses idle or powerdown mode, set this bit when you program the CCBs. If it does not, clearing this bit when you program the CCBs will prevent accidental entry into idle or powerdown mode. (Chapter 14, "Special Operating Modes," discusses idle and powerdown modes.)

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after reset. The CCBs reside at addresses FF2018H (CCB0) and FF201AH (CCB1

Figure 17-1. Chip Configuration 0 (CCR0) Register (Continued)

NOTE

The developers have made a substantial effort to provide an adequate program protection scheme. However, Intel cannot and does not guarantee that these protection methods will always prevent unauthorized access.

17.3 ENTERING TEST-ROM ROUTINES

To execute properly, the microcontroller must have these minimum hardware connections: XTAL1 driven, unused input pins strapped, and power and grounds applied. Follow the operating conditions specified in the datasheet. Place the device into test-ROM mode by configuring pins as described in Tables 17-1 and holding those values during the rising edge of RESET#.

17.3.1 Power-up and Power-down Sequences

To enter a test-ROM routine, follow these power-up and power-down procedures.

WARNING

Failure to observe these precautions will cause permanent device damage.

- Ensure that the port 12 pins are in their desired states before RESET# rises.
- Ensure that all voltages are within the ranges specified in the datasheet and that the oscillator is stable before RESET# rises.
- Ensure that the power supplies to the V_{CC}, EA# and RESET# pins are well regulated and free of glitches and spikes.
- Ensure that all V_{SS} pins are well grounded.

17.3.1.1 Power-up Sequence

1. Hold RESET# low while V_{CC} stabilizes.
2. While V_{CC} is ramping to within specifications, ensure that no pins are driven higher than V_{CC} + 0.5 volts.
3. Ensure that all pins needed to enter the desired mode are driven to the correct values (Table 17-1).
4. After the oscillator stabilizes, bring the RESET# pin high.

17.3.1.2 Power-down Sequence

1. Assert the RESET# signal and hold it low throughout the powerdown sequence.
2. Turn off the V_{CC} supply and allow time for it to reach 0 volts.

17.4 ROM-DUMP ROUTINE AND CIRCUIT

The ROM-dump routine provides an easy way to verify the contents of the ROM array. Figure 17-2 shows a recommended circuit. Although the illustration uses an EPROM, you could use a RAM device or you could dump the ROM contents to any 16-bit parallel port.

To access the ROM-dump routine, follow the power-up sequence on page 17-5. The ROM-dump routine checks the security key, regardless of the CCR security-lock bit. If you have programmed a security key, a matching key must reside in the external memory; otherwise, the device enters an endless loop. If the security key verifies, the ROM-dump routine writes the entire ROM array to external memory.

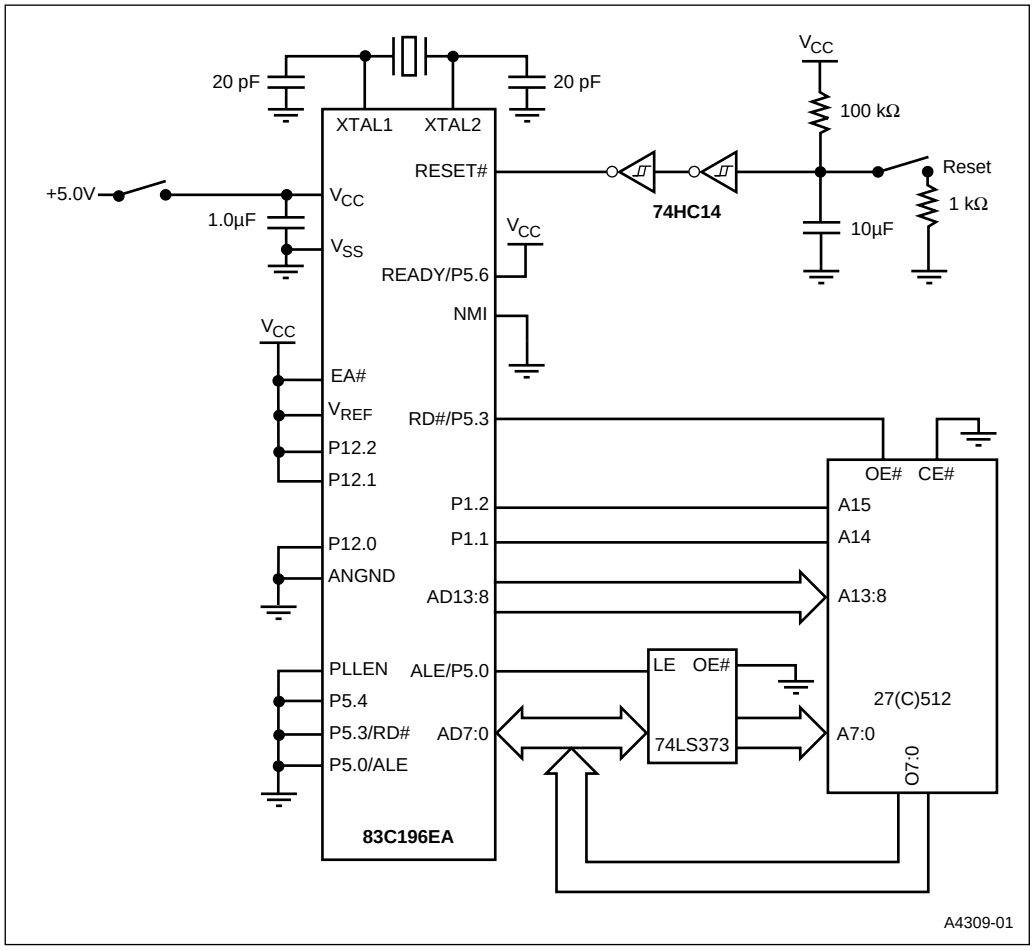


Figure 17-2. ROM-dump Circuit

Table 17-3. ROM-dump Memory Map

Address Output from 8XC196 Device (A15:0)	Internal ROM Address	Address Using Circuit in Figure 17-2 (P1.2:1, A13:0)	Description
4020–402FH	FF2020–FF202FH	C020–C02FH	Security key for verification.
4000–5FFFH	FF2000–FF3FFFH	C000–DFFFH	Code, data, and reserved locations.

17.5 SERIAL PORT MODE ROUTINE

The serial port mode test-ROM routine enables the serial I/O (SIO) port to write data through the TXD0 pin and read it through the RXD0 pin. In this mode, the device executes a routine from its internal test ROM. This routine is a modified version of the reduced instruction set monitor (RISM) that exists on all 8X9X evaluation boards. The simple hardware setup of this routine makes it useful for in-module testing and in-line diagnostics. Special software, called IBSP196, simplifies communication between the device and a smart terminal. This software is available free of charge through the Intel BBS. (See “Application Support Services” on page 1-8.)

NOTE

Serial port mode has no provision for security-key verification. If the LOC bit is cleared in CCB0, an attempt to enter serial port mode causes the device to enter an endless loop.

Entering serial port mode enables you to perform these functions:

- download a module-testing program into internal RAM and execute it
- examine the code programmed into the ROM
- examine the contents of any register
- manipulate RAM, SFRs, or pin states

17.5.1 Serial Port RISM

When you enter serial port mode test-ROM routine, the device begins executing its RISM (reduced instruction set monitor) program. The RISM executes in 16-Mbyte mode (24-bit addresses). You communicate with the device by sending RISM commands from any smart terminal across the TXD0 and RXD0 pins at a fixed baud rate. (For a complete list of all the available RISM commands, refer to “RISM Command Descriptions” on page 17-15.)

Upon entering serial port mode, the device enters a waiting loop, called Monitor_Pause, in which it waits for RISM commands to arrive across the serial port. The commands are each one byte in length and have values between 00H and 1FH. A value between 00H and 1FH is considered a command unless it follows a data latch enable (SET_DLE_FLAG) command. The SET_DLE_FLAG command sets the DLE flag in the MODE register (57H). The DLE flag alerts

the RISM to store the next byte in the DATA register, a 32-bit first-in-last-out (FILO) register located at 58H.

When a receive interrupt occurs, the RISM checks the data value and the DLE flag. If the data value is greater than 1FH or if the DLE flag is set, the received byte is considered data and is stored in the DATA register (58H). Each time new data is received, the DATA register is shifted left by eight bits. If the value is between 00H and 1FH and the DLE flag is clear, the received byte is considered a command. Commands are stored in the CHAR register (56H). After it executes each command, the RISM resumes Monitor_Pause, except where otherwise noted.

To access a particular address, you must first send the address across the serial port as data. Send it one byte at a time, with the high byte first (the address is always assumed to be 24 bits). The RISM stores the address data in the DATA register. Now you must transfer the address from the DATA register to the shadow address register (5CH) by sending the DATA_TO_ADDR command (0AH).

17.5.2 Serial Port Mode Circuit

Figure 17-3 shows the recommended circuit for serial port mode. In this mode, data is transmitted and received through the TXD0 (P2.0) and RXD0 (P2.1) pins. Connect these pins to any smart terminal capable of communicating with the RISM. Any host that requires an RS-232C interface (such as a PC) must be connected through an RS-232C driver/receiver such as the one shown within the dashed line in Figure 17-3. XTAL1 and XTAL2 can be connected to a crystal with a frequency of 9600 baud at 16 MHz. The frequency must correspond to the value in the SP_BAUD register.

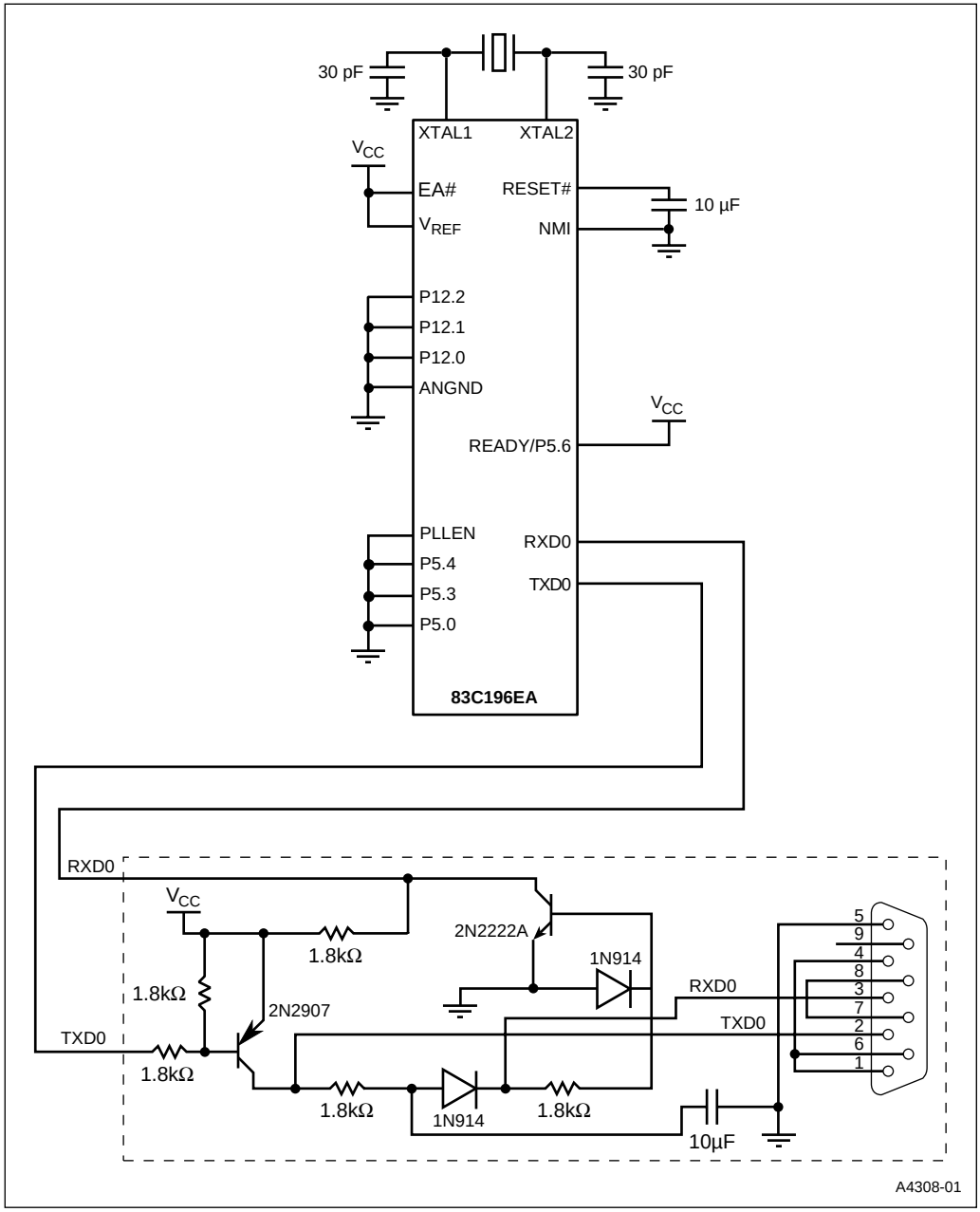


Figure 17-3. Serial Port Mode Circuit

Because the RISM begins at location FF2000H in serial port programming mode, the ROM locations are automatically remapped as shown in Table 17-4. For example, to access ROM location FF2000H in serial port mode, you must address it as FFA000H.

Table 17-4. Serial Port Mode Memory Map

Description	Address Range (Hex)	
	Normal Operation	Serial Port Programming Mode
Internal ROM	FF2000–FF3FFF	FFA000–FFBFFF
External memory	—	FF4000–FF5FFF
Do not address	—	FF2400–FF3FFF
Test ROM and RISM	—	FF2000–FF23FF

17.6 SDU RISM EXECUTION ROUTINE

The SDU RISM execution routine contains a built-in reduced instruction set monitor (RISM), which enables you to access user routines that reside in test-ROM. The on-chip RISM commands are accessed via the SDU module (see Table 17-7 on page 17-15). To access the RISM routine, set the interrupt (SDU_INT) bit of the SDU command byte and program your software to vector to the RISM. (Refer to Figure 16-7 on page 16-8 for additional information on using the SDU command byte.)

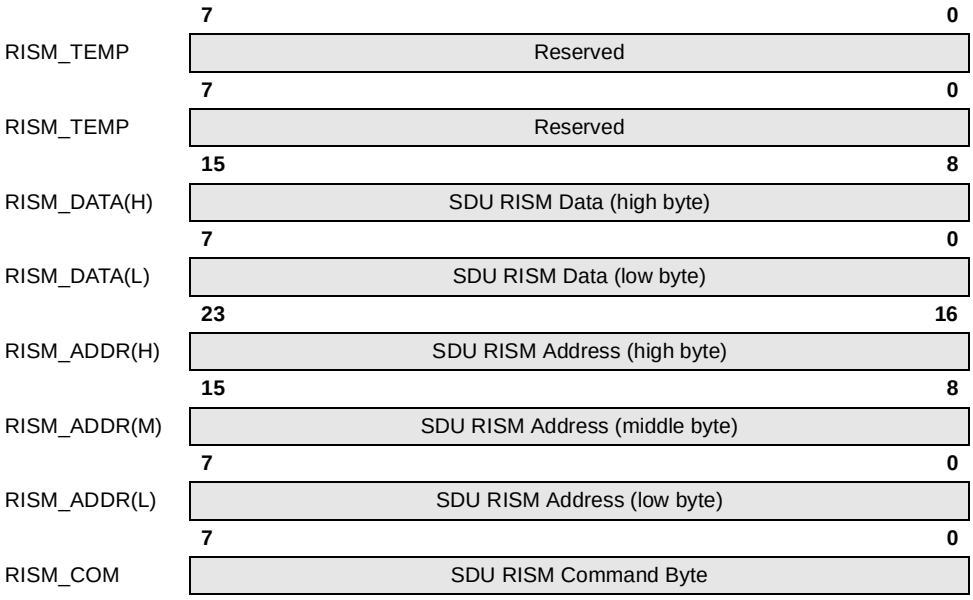
NOTE

The SDU interrupt occurs at the execution of the SDU command byte. The mode select (SDU_COM.5:2) bits must not equal 00H or 0FH, or the reset SDU instruction will take precedence. Use 68H as the SDU command byte to generate an interrupt.

When accessing the SDU RISM execution routine, code RAM address locations 0400–0407H are reserved (see Figure 17-4) and must not be used in any user routines that are called. The eight bytes of the SDU RISM control block are used to communicate to the CPU, since the CPU has no direct access to any SDU register.

SDU RISM Control Block

The SDU RISM control block contains data and address registers (RISM_DATA and RISM_ADDR) and a command byte register (RISM_COM) for initializing a data transfer.



Register	Address	Function
RISM_TEMP	407–406H	Reserved.
RISM_DATA	405–404H	SDU RISM Data Stores the data content to be written.
RISM_ADDR	403–401H	SDU RISM Address Stores the address to which the data content is to be read or written.
RISM_COM	400H	SDU RISM Command Byte See Table 17-7 on page 17-15 for the RISM commands supported by the SDU.

Figure 17-4. SDU RISM Control Block

17.6.1 SDU RISM Data Transfer

The structure of a data transfer is an 8-bit command byte followed by a specific number of data bytes, which are determined by the command instruction register, SDU_COM. With the RISM data transfer, after you load the RISM command, address, and data registers needed into the code RAM, the last byte loaded must command the SDU to issue the interrupt instruction. After the RISM responds to the interrupt request and executes the command, the most-significant bit of the RISM command byte (located at 400H) will be set to indicate that the RISM has completed the command. The command byte of the RISM must be polled to determine the status of the operation. (Refer to Table 17-5 and Table 17-6 for RISM data transfer command setups.)

17.6.1.1 SDU RISM Data Transfer Before

The following table (Table 17-5) shows what needs to be entered into each code RAM location before an interrupt is generated to execute each command.

Table 17-5. Before RISM Command Execution

RISM Command	400H	401H	402H	403H	404H	405H
RISM_IDLE	00H	—	—	—	—	—
READ_BYTE	01H	ADDR(L)	ADDR(M)	ADDR(H)	—	—
READ_WORD	02H	ADDR(L)	ADDR(M)	ADDR(H)	—	—
WRITE_BYTE	03H	ADDR(L)	ADDR(M)	ADDR(H)	DATA(L)	—
WRITE_WORD	04H	ADDR(L)	ADDR(M)	ADDR(H)	DATA(L)	DATA(H)
GO	05H	—	—	—	—	—
HALT	06H	—	—	—	—	—
READ_NEXT_BYTE	11H	—	—	—	—	—
READ_NEXT_WORD	12H	—	—	—	—	—
WRITE_NEXT_BYTE	13H	DATA(L)	—	—	—	—
WRITE_NEXT_WORD	14H	DATA(L)	DATA(H)	—	—	—

— indicates unused address location

17.6.1.2 SDU RISM Data Transfer After

Table 17-6 shows the code RAM contents after each command has been executed.

Table 17-6. After RISM Command Execution

RISM Command	400H	401H	402H	403H	404H	405H
RISM_IDLE	80H	U	U	U	U	U
READ_BYTE	81H	U	U	DATA(L)	U	U
READ_WORD	82H	U	DATA(L)	DATA(H)	U	U
WRITE_BYTE	83H	U	U	U	U	U
WRITE_WORD	84H	U	U	U	U	U
GO	85H	U	U	U	U	U
HALT	86H	U	U	U	U	U
READ_NEXT_BYTE	81H	U	U	DATA(L)	U	U
READ_NEXT_WORD	82H	U	DATA(L)	DATA(H)	U	U
WRITE_NEXT_BYTE	83H	U	U	U	U	U
WRITE_NEXT_WORD	84H	U	U	U	U	U

U = unchanged

NOTE

For the read byte commands, the data is returned in location 403H (not 402H). This is because the SDU cannot correctly perform a byte read on an even address. All reads from the code RAM are 16-bit transfers, and there is no logic to move the low byte to the high byte as would be required to correctly read an even byte.

In the command byte column of Table 17-6, notice that the RISM has set the most-significant bit of the command byte to signal to the monitor software that it has completed executing a command byte. This allows the host software to determine when the RISM is done processing one command so that it can accept another command.

17.6.2 SDU RISM Execution Circuit

Figure 17-5 shows the recommended circuit for accessing the SDU RISM execution routine.

To standardize the connector interface to the SDU port, we recommend that you use a 10-pin, 2-row I/O expansion connector. This low cost, readily available standard connector can be easily added to any printed-circuit board (PCB) after manufacturing of the module.

To prevent mis-socketing and ensure proper cable connection, we recommend that you remove the square pin at position #1 (Key) and block the adjoining cable position.

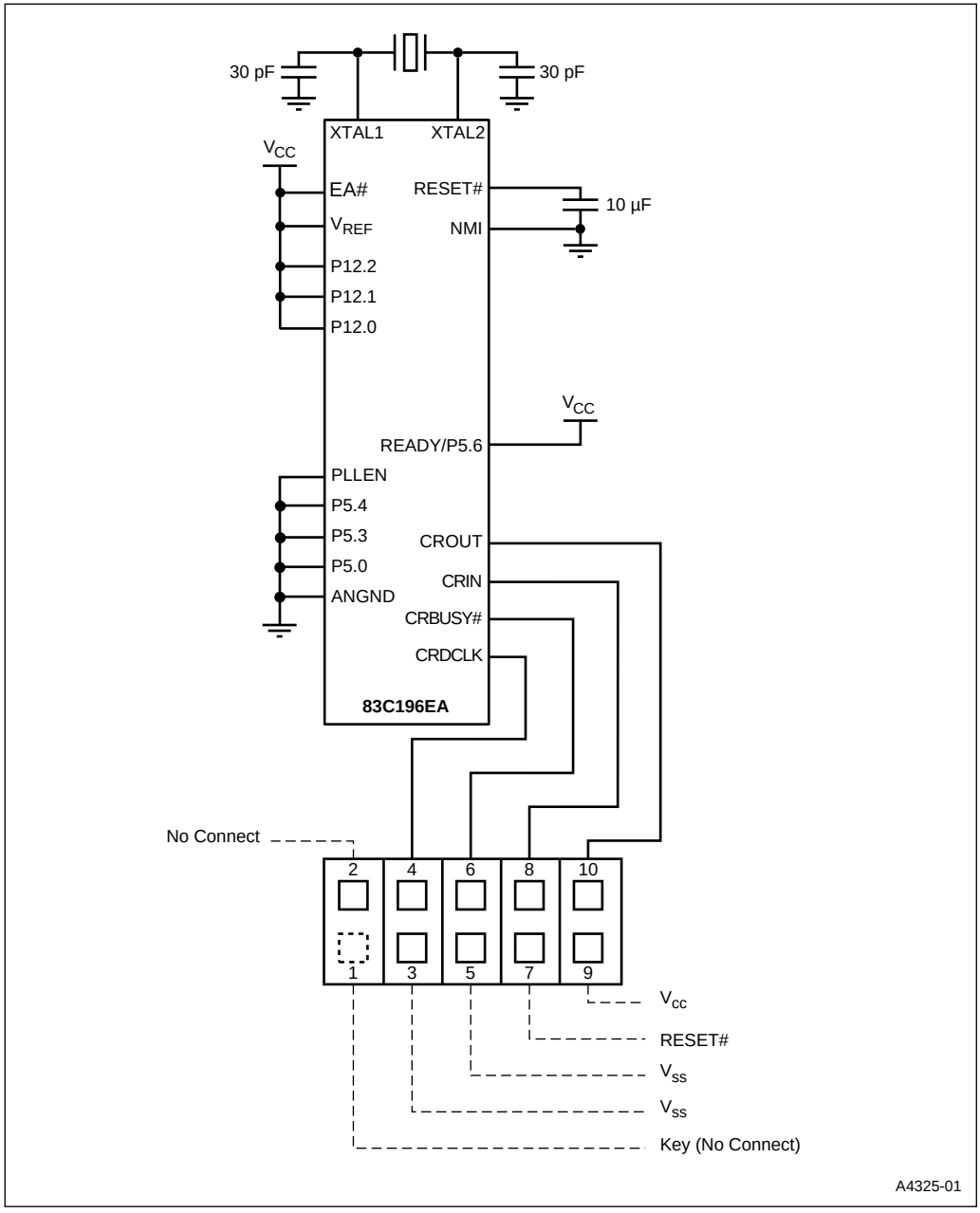


Figure 17-5. SDU RISM Execution Circuit

17.7 RISM COMMAND DESCRIPTIONS

Table 17-7 lists and describes all the RISM commands for both the serial port mode and SDU RISM execution routines, along with their associated opcodes. Some of the RISM commands listed in Table 17-7 are SDU RISM only and some are serial port RISM only. Also, the RISM commands that are shared between the SDU and serial port **do not** share the same opcode values.

Table 17-7. RISM Commands

Command	Serial Port RISM Opcode	SDU RISM Opcode	Description
SET_DLE_FLAG	00H	—	Sets the DLE (data latch enable) flag in bit 0 of the MODE register (57H) to tell the RISM that the next byte on the serial port is data that should be loaded into the DATA register (58H). The flag is cleared as soon as the byte is read.
RISM_IDLE	—	00H	The RISM routine halts and enters a loop waiting for another interrupt.
TRANSMIT	02H	—	Transmits the low byte of the DATA register to the serial port through the CHAR register, shifts the DATA register right (long) by eight bits, and increments ADDR by one.
READ_BYTE	04H	01H	Reads the byte contents of the memory address pointed to by the RISM_ADDR register and puts it into the low byte of the DATA register (location 403H in SDU RISM execution). The shadow address register, located in register RAM, is loaded with RISM_ADDR + 1.
READ_WORD	05H	02H	Reads the word contents of the memory address pointed to by the RISM_ADDR register and puts it into the low word of the DATA register (locations 402–403H in SDU RISM execution). The shadow address register, located in register RAM, is loaded with RISM_ADDR + 2.
WRITE_BYTE	07H	03H	Stores the byte contents of the RISM_DATA register in the memory address pointed to by the shadow address register and increments the shadow address register by one.
WRITE_WORD	08H	04H	Stores the word contents of the RISM_DATA register in the memory address pointed to by the shadow address register and increments the shadow address register by two.
DATA_TO_ADDR	0AH	—	Puts the low word of the DATA register into the ADDR register.
GO	12H	05H	Starts execution at location pointed to by register RAM address 60H.
HALT	13H	06H	Stops execution and returns to the Monitor_Pause state (RISM_IDLE state in SDU RISM execution).

— indicates the RISM command is not available

Table 17-7. RISM Commands (Continued)

Command	Serial Port RISM Opcode	SDU RISM Opcode	Description
READ_NEXT_BYTE	—	11H	Reads the byte contents of the next memory address pointed to by the shadow address register and increments by one.
READ_NEXT_WORD	—	12H	Reads the word contents of the next memory address pointed to by the shadow address register and increments by two.
WRITE_NEXT_BYTE	—	13H	Stores the byte contents of the next memory address pointed to by the shadow address register and increments by one.
WRITE_NEXT_WORD	—	14H	Stores the word contents of the next memory address pointed to by the shadow address register and increments by two.
REPORT_STATUS	14H	—	Loads a value into the DATA register. This value indicates the status of your program: **Value** **Status** 00 data latched 01 trapped 02 running

— indicates the RISM command is not available

17.8 EXECUTING PROGRAMS FROM REGISTER RAM

For those wanting to execute user programs from register RAM while in either serial port mode or SDU RISM execution, the RISM allows you to initialize the user program counter (UPC and UPC2), user processor status word (UPSW), and user window selection register (UWSR). Table 17-8 lists the registers and the register RAM addresses to which you may write new values.

Table 17-8. User Program Register and Register RAM Location

User Program Register	Register RAM Address
Shadow Address	5CH
UPC	60H
UPC2	62H
UPSW	64H
UWSR	66H

Before attempting to execute a program from RAM, write the beginning address of the program to UPC and UPC2 at the addresses shown in Table 17-8. You need not change the UWSR and UPSW unless other flags need to be set for the program you are executing. After writing the UPC and UPC2 values, issue the GO command, which automatically initializes the UPC and begins code execution (refer to "SDU RISM Execution Go Command Example" on page 17-20). When

the RISM interrupts or halts the program, it reinitializes the UPC, UPC2, UPSW (which includes INT_MASK), and UWSR (which includes INT_MASK1).

17.9 RISM COMMAND EXAMPLES

This section provides examples of ways in which you might use the RISM commands.

17.9.1 Serial Port Mode RISM Read Command Example

This example reads the contents of remapped ROM address FFA080H. This example assumes that the word at location 002080H is 8067H, the assembled hex value of the code.

Send	Comments	DATA					ADDR		
FF	Data. High byte of address to DATA register.					FF			
A0	Data. Middle byte of address to DATA register.				FF	A0			
80	Data. Low byte of address to DATA register.			FF	A0	80			
0A	DATA_TO_ADDR. Move DATA to address register.			FF	A0	80	FF	A0	80
05	READ_WORD. Put word at FFA080H into DATA.	FF	A0	80	80	67	FF	A0	80
02	TRANSMIT. Transmit low byte of DATA across the serial link, increment ADDR by one, and shift DATA right long by eight bits.								
		00	FF	A0	80	80	FF	A0	81
02	TRANSMIT. Transmit low byte of DATA across the serial link, increment ADDR by one, and shift DATA right long by eight bits.								
		00	00	FF	A0	80	FF	A0	82

Any address can be read in this manner, including register RAM, code RAM, and SFRs.

17.9.2 Serial Port Mode RISM Write Command Example

This example loads a program into internal code RAM one byte at a time. The following program is to be loaded:

```
400  A1221180  LD 80H, #1122H ;Puts 1122H into register RAM location 80H
404  27FE      SJMP 0404H     ;Jumps to itself to keep program running
                              ;indefinitely
```

The hex file must be loaded one byte at a time using the RISM commands.

Send	Comments	DATA					ADDR		
00	SET_DLE_FLAG. Next data byte is < 1FH.								
00	Data. High byte of address 0400H.					00			
00	SET_DLE_FLAG. Next data byte is < 1FH.					00			
04	Data. Middle byte of address 0400H.				00	04			
00	SET_DLE_FLAG. Next data byte is < 1FH.				00	04			
00	Data. Low byte of address 0400H.			00	04	00			
0A	DATA_TO_ADDR. Move data to address register.			00	04	00	00	04	00
A1	Data. High byte of hex file for location 000401H.		00	04	00	A1	00	04	00
22	Data. Low byte of hex file for location 000400H.	00	04	00	A1	22	00	04	00
08	WRITE_WORD. Low word of DATA to memory location 000400H (contents of ADDR). Increment ADDR by two.								
		00	04	00	A1	22	00	04	00
	Memory Addresses (Hex)				**000401**	**000400**			
					A1	22	00	04	02
00	SET_DLE_FLAG. Next data byte is < 1FH.	00	04	00	A1	22	00	04	02
11	Data. High byte of hex file for location 0403H.	04	00	A1	22	11	00	04	02
00	SET_DLE_FLAG. Next data byte is < 1FH.	04	00	A1	22	11	00	04	02
80	Data. Low byte of hex file for location 0402H.	00	A1	22	11	80	00	04	02
08	WRITE_WORD. Low word of DATA to memory location 000402H (contents of ADDR). Increment ADDR by two.								
		00	A1	22	11	80	00	04	02
	Memory Addresses (Hex)				**000403**	**000402**			
					11	80	00	04	04

Send	Comments	DATA	ADDR
27	Data. High byte of hex file for location 0405H.	A1 22 11 80 27	00 04 04
FE	Data. Low byte of hex file for location 0404H.	22 11 80 27 FE	00 04 04
08	WRITE_WORD. Low word of DATA to memory location 000404H (contents of ADDR). Increment ADDR by two.		
		22 11 80 27 FE	00 04 04
	Memory Addresses (Hex)	**000405 000404**	
		27 FE	00 04 06

17.9.3 SDU RISM Execution Write Command Example

This example writes the data word value 3355H to 001FBCH, an SFR address not in code RAM.

Upon sending a value of 68H, the SDU command to generate an interrupt, you must wait for the most-significant bit of location 000400H to become set. Also, when you are using the SDU WRITE_WORD command, the most-significant byte always precedes the least-significant byte onto the serial link.

Send	Comments	DATA	ADDR
06	SDU command. Write code RAM word address.		
04	Address. High byte of code RAM address.		04
00	Address. Low byte of code RAM address.		04 00
12	SDU command. Write code RAM data word.		04 00
BC	Data. Low byte of RISM address to 000401H.	BC →	04 01
04	Data. RISM write_word command to 000400H.	04 →	04 00
12	SDU command. Write code RAM data word.	BC 04	04 01
00	Data. High byte of RISM address to 000403H.	00 →	04 03
1F	Data. Middle byte of RISM address to 000402H.	1F →	04 02
12	SDU command. Write code RAM data word.	00 1F BC 04	04 03

Send	Comments	DATA	ADDR
33	Data. High byte of RISM data to 000405H.	33 →	04 05
55	Data. Low byte of RISM data to 000404H.	55 →	04 04
68	SDU_Interrupt command. Vectors to the SDU RISM routine in TROM and processes the SDU RISM control block.		
		33 55	00 1F BC
	Memory Addresses (Hex)	**001FBD 001FBC**	
		33 55	00 1F BC

17.9.4 SDU RISM Execution Go Command Example

To the start processor at address *zzyyxx*H, you must initiate the following five steps.

1. Write to the UPC (60H) to set up the low (*xx*) and middle (*yy*) bytes of the start address.
2. Write to the UPC2 (62H) to set up the high (*zz*) byte of the start address.
3. Write to the UPSW (64H) the value 0280H to enable servicing of the SDU interrupt.
4. Write to the UWSR (66H) the value 0000H to clear the WSR.
5. Execute the SDU RISM GO command.

Tabulated below are the byte sequences for accomplishing each step.

Send	Comments — UPC	DATA	ADDR
06	SDU command. Write code RAM word address.		
04	Address. High byte of code RAM address.		04
00	Address. Low byte of code RAM address.		04 00
12	SDU command. Write code RAM data word.		04 00
60	Data. Low byte of UPC address.	60 →	04 01
04	Data. RISM write_word command to 000400H.	04 →	04 00
12	SDU command. Write code RAM data word.	60 04	04 01
00	Data. High byte of UPC address.	00 →	04 03

Send	Comments — UPC	DATA						ADDR		
00	Data. Middle byte of UPC address.					00	→		04	02
12	SDU command. Write code RAM data word.		00	00	60	04			04	03
yy	Data. Middle byte of start address.					*yy*	→		04	05
xx	Data. Low byte of start address.					*xx*	→		04	04
68	SDU_Interrupt command. Vectors to the SDU RISM routine in TROM and processes the SDU RISM control block.									
					yy	*xx*		00	00	60

Memory Addresses (Hex)

000061	000060
yy	*xx*

Send	Comments — UPC2	DATA						ADDR		
06	SDU command. Write code RAM word address.									
04	Address. High byte of code RAM address.									04
00	Address. Low byte of code RAM address.								04	00
13	SDU command. Write code RAM data byte.								04	00
14	Data. RISM write_next_word command.					14	→		04	00
13	SDU command. Write code RAM data byte.					14			04	00
zz	Data. High byte of start address.					*zz*	→		04	01
13	SDU command. Write code RAM data byte.				*zz*	14			04	01
00	Data. Reserved; write to zero.					00	→		04	02
68	SDU_Interrupt command. Vectors to the SDU RISM routine in TROM and processes the SDU RISM control block.									
					00	*zz*		00	00	62

Send	Comments — UPC2	DATA	ADDR
	Memory Addresses (Hex)	**000063 000062** 00 *zz*	00 00 62

Send	Comments — UPSW	DATA	ADDR
06	SDU command. Write code RAM word address.		
04	Address. High byte of code RAM address.		04
00	Address. Low byte of code RAM address.		04 00
13	SDU command. Write code RAM data byte.		04 00
14	Data. RISM write_next_word command.	14 →	04 00
13	SDU command. Write code RAM data byte.	14	04 00
80	Data. Low byte of PSW. Setting bit seven enables the SDU interrupt request.	80 →	04 01
13	SDU command. Write code RAM data byte.	80 14	04 01
02	Data. High byte of PSW. Setting bit one enables the servicing of all maskable interrupts.	02 →	04 02
68	SDU_Interrupt command. Vectors to the SDU RISM routine in TROM and processes the SDU RISM control block.	02 80	00 00 64
	Memory Addresses (Hex)	**000065 000064** 02 80	00 00 64

Send	Comments — UWSR	DATA	ADDR
06	SDU command. Write code RAM word address.		
04	Address. High byte of code RAM address.		04

Send	Comments — UWSR	DATA	ADDR
00	Address. Low byte of code RAM address.		04 00
13	SDU command. Write code RAM data byte.		04 00
14	Data. RISM write_next_word command.	14 →	04 00
13	SDU command. Write code RAM data byte.	14	04 00
00	Data. Low byte of WSR. WSR = 0000H	00 →	04 01
13	SDU command. Write code RAM data byte.	00 14	04 01
00	Data. High byte of WSR.	00 →	04 02
68	SDU_Interrupt command. Vectors to the SDU RISM routine in TROM and processes the SDU RISM control block.		
		00 00	00 00 66
	Memory Addresses (Hex)	**000067 000066**	
		00 00	00 00 66

Send	Comments — Execute RISM GO Command	DATA	ADDR
06	SDU command. Write code RAM word address.		
04	Address. High byte of code RAM address.		04
00	Address. Low byte of code RAM address.		04 00
13	SDU command. Write code RAM data byte.		04 00
05	Data. RISM GO command to 000400H.	05 →	04 00
68	SDU_Interrupt command. Vectors to the SDU RISM routine in TROM and processes the SDU RISM control block.		
			zz yy xx

A

Instruction Set Reference

APPENDIX A
INSTRUCTION SET REFERENCE

This appendix provides reference information for the instruction set of the family of MCS® 96 microcontrollers. It defines the processor status word (PSW) flags, describes each instruction, shows the relationships between instructions and PSW flags, and shows hexadecimal opcodes, instruction lengths, and execution times. It includes the following tables.

- Table A-1 on page A-2 is a map of the opcodes.
- Table A-2 on page A-4 defines the processor status word (PSW) flags.
- Table A-3 on page A-5 shows the effect of the PSW flags or a specified register bit on conditional jump instructions.
- Table A-4 on page A-5 defines the symbols used in Table A-6.
- Table A-5 on page A-6 defines the variables used in Table A-6 to represent instruction operands.
- Table A-6 beginning on page A-7 lists the instructions alphabetically, describes each of them, and shows the effect of each instruction on the PSW flags.
- Table A-7 beginning on page A-45 lists the instruction opcodes, in hexadecimal order, along with the corresponding instruction mnemonics.
- Table A-8 on page A-51 lists instruction lengths and opcodes for each applicable addressing mode.
- Table A-9 on page A-57 lists instruction execution times, expressed in state times.

NOTE

The # symbol prefixes an immediate value in immediate addressing mode. Chapter 3, “Programming Considerations,” describes the operand types and addressing modes.

Table A-1. Opcode Map (Left Half)

Opcode	*x*0	*x*1	*x*2	*x*3	*x*4	*x*5	*x*6	*x*7
0*x*	SKIP	CLR	NOT	NEG	XCH di	DEC	EXT	INC
1*x*		CLRB	NOTB	NEGB	XCHB di	DECB	EXTB	INCB
2*x*	SJMP							
3*x*	JBC bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7
4*x*	AND 3op di	im	in	ix	ADD 3op di	im	in	ix
5*x*	ANDB 3op di	im	in	ix	ADDB 3op di	im	in	ix
6*x*	AND 2op di	im	in	ix	ADD 2op di	im	in	ix
7*x*	ANDB 2op di	im	in	ix	ADDB 2op di	im	in	ix
8*x*	OR di	im	in	ix	XOR di	im	in	ix
9*x*	ORB di	im	in	ix	XORB di	im	in	ix
A*x*	LD di	im	in	ix	ADDC di	im	in	ix
B*x*	LDB di	im	in	ix	ADDCB di	im	in	ix
C*x*	ST di	BMOV	ST in	ix	STB di	CMPL	STB in	ix
D*x*	JNST	JNH	JGT	JNC	JNVT	JNV	JGE	JNE
E*x*	DJNZ	DJNZW	TIJMP	BR/EBR in	EBMOVI		EJMP	LJMP
F*x*	RET	ECALL	PUSHF	POPF	PUSHA	POPA	IDLPD	TRAP

NOTE: The first digit of the opcode is listed vertically, and the second digit is listed horizontally. The related instruction mnemonic is shown at the intersection of the two digits. Shading indicates unimplemented (reserved) opcodes. If the CPU attempts to execute an unimplemented opcode, an interrupt occurs.

Table A-1. Opcode Map (Right Half)

Opcode	*x*8	*x*9	*x*A	*x*B	*x*C	*x*D	*x*E	*x*F
0*x*	SHR	SHL	SHRA	XCH ix	SHRL	SHLL	SHRAL	NORML
1*x*	SHRB	SHLB	SHRAB	XCHB ix	EST in	EST ix	ESTB in	ESTB ix
2*x*	SCALL							
3*x*	JBS bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7
4*x*	SUB 3op di	im	in	ix	MUL & MULU 3op (2) di	im	in	ix
5*x*	SUBB 3op di	im	in	ix	MULB & MULUB 3op (2) di	im	in	ix
6*x*	SUB 2op di	im	in	ix	MUL & MULU 2op (2) di	im	in	ix
7x	SUBB 2op di	im	in	ix	MULB & MULUB 2op (2) di	im	in	ix
8*x*	CMP di	im	in	ix	DIV & DIVU (2) di	im	in	ix
9*x*	CMPB di	im	in	ix	DIVB & DIVUB (2) di	im	in	ix
A*x*	SUBC di	im	in	ix	LDBZE di	im	in	ix
B*x*	SUBCB di	im	in	ix	LDBSE di	im	in	ix
C*x*	PUSH di	im	in	ix	POP di	BMOVI	POP in	ix
D*x*	JST	JH	JLE	JC	JVT	JV	JLT	JE
E*x*	ELD in	ELD ix	ELDB in	ELDB ix	DPTS	EPTS	(1)	LCALL
F*x*	CLRC	SETC	DI	EI	CLRVT	NOP	(3)	RST

NOTES:
1. This opcode is reserved, but it does not generate an unimplemented opcode interrupt.
2. Signed multiplication and division require a prefix opcode (FE).
3. Prefix opcode for signed multiplication and division instructions.

Table A-2. Processor Status Word (PSW) Flags

Mnemonic	Description
C	The carry flag is set to indicate an arithmetic carry from the MSB of the ALU or the state of the last bit shifted out of an operand. If a subtraction operation generates a borrow, the carry flag is cleared. **C** **Value of Bits Shifted Off** 0 < ½ LSB 1 ≥ ½ LSB Normally, the result is rounded up if the carry flag is set. The sticky bit flag allows a finer resolution in the rounding decision. **C ST** **Value of Bits Shifted Off** 0 0 = 0 0 1 > 0 and < ½ LSB 1 0 = ½ LSB 1 1 > ½ LSB and < 1 LSB
N	The negative flag is set to indicate that the result of an operation is negative. The flag is correct even if an overflow occurs. For all shift operations and the NORML instruction, the flag is set to equal the most-significant bit of the result, even if the shift count is zero.
ST	The sticky bit flag is set to indicate that, during a right shift, a "1" has been shifted into the carry flag and then shifted out. This bit is undefined after a multiply operation. The sticky bit flag can be used with the carry flag to allow finer resolution in rounding decisions. See the description of the carry (C) flag for details.
V	The overflow flag is set to indicate that the result of an operation is too large to be represented correctly in the available space. For shift operations, the flag is set if the most-significant bit of the operand changes during the shift. For divide operations, the quotient is stored in the low-order half of the destination operand and the remainder is stored in the high-order half. The overflow flag is set if the quotient is outside the range for the low-order half of the destination operand. **Instruction** **Quotient Stored in:** **V Flag Set if Quotient is:** DIVB Short-integer < –127 (< 81H) or > +128 (> 80H) DIV Integer < –32767(< 8001H) or > +32768 (> 8000H) DIVUB Byte > 255 (> FFH) DIVU Word > 65535 (> FFFFH)
VT	The overflow-trap flag is set when the overflow flag is set, but it is cleared only by the CLRVT, JVT, and JNVT instructions. This allows testing for a possible overflow at the end of a sequence of related arithmetic operations, which is generally more efficient than testing the overflow flag after each operation.
Z	The zero flag is set to indicate that the result of an operation was zero. For multiple-precision calculations, the zero flag cannot be set by the instructions that use the carry bit from the previous calculation (e.g., ADDC, SUBC). However, these instructions can clear the zero flag. This ensures that the zero flag will reflect the result of the entire operation, not just the last calculation. For example, if the result of adding together the lower words of two double words is zero, the zero flag would be set. When the upper words are added together using the ADDC instruction, the flag remains set if the result is zero and is cleared if the result is not zero.

Table A-3 shows the effect of the PSW flags or a specified condition on conditional jump instructions. Table A-4 defines the symbols used in Table A-6 to show the effect of each instruction on the PSW flags.

Table A-3. Effect of PSW Flags or Specified Conditions on Conditional Jump Instructions

Instruction	Jumps to Destination if	Continues if
DJNZ	decremented byte ≠ 0	decremented byte = 0
DJNZW	decremented word ≠ 0	decremented word = 0
JBC	specified register bit = 0	specified register bit = 1
JBS	specified register bit = 1	specified register bit = 0
JNC	C = 0	C = 1
JNH	C = 0 OR Z = 1	C = 1 AND Z = 0
JC	C = 1	C = 0
JH	C = 1 AND Z = 0	C = 0 OR Z = 1
JGE	N = 0	N = 1
JGT	N = 0 AND Z = 0	N = 1 OR Z = 1
JLT	N = 1	N = 0
JLE	N = 1 OR Z = 1	N = 0 AND Z = 0
JNST	ST = 0	ST = 1
JST	ST = 1	ST = 0
JNV	V = 0	V = 1
JV	V = 1	V = 0
JNVT	VT = 0	VT = 1 (clears VT)
JVT	VT = 1 (clears VT)	VT = 0
JNE	Z = 0	Z = 1
JE	Z = 1	Z = 0

Table A-4. PSW Flag Setting Symbols

Symbol	Description
✓	The instruction sets or clears the flag, as appropriate.
—	The instruction does not modify the flag.
↓	The instruction may clear the flag, if it is appropriate, but cannot set it.
↑	The instruction may set the flag, if it is appropriate, but cannot clear it.
1	The instruction sets the flag.
0	The instruction clears the flag.
?	The instruction leaves the flag in an indeterminate state.

Table A-5 defines the variables that are used in Table A-6 to represent the instruction operands.

Table A-5. Operand Variables

Variable	Description
aa	A 2-bit field within an opcode that selects the basic addressing mode used. This field is present only in those opcodes that allow addressing mode options. The field is encoded as follows: 00 register-direct 01 immediate 10 indirect 11 indexed
baop	A byte operand that is addressed by any addressing mode.
bbb	A 3-bit field within an opcode that selects a specific bit within a register.
bitno	A 3-bit field within an opcode that selects one of the eight bits in a byte.
breg	A byte register in the internal register file. When it could be unclear whether this variable refers to a source or a destination register, it is prefixed with an *S* or a *D*. The value must be in the range of 00–FFH.
cadd	An address in the program code.
Dbreg†	A byte register in the lower register file that serves as the destination of the instruction operation.
disp	Displacement. The distance between the end of an instruction and the target label.
Dlreg†	A 32-bit register in the lower register file that serves as the destination of the instruction operation. Must be aligned on an address that is evenly divisible by 4. The value must be in the range of 00–FCH.
Dwreg†	A word register in the lower register file that serves as the destination of the instruction operation. Must be aligned on an address that is evenly divisible by 2. The value must be in the range of 00–FEH.
lreg	A 32-bit register in the lower register file. Must be aligned on an address that is evenly divisible by 4. The value must be in the range of 00–FCH.
ptr2_reg	A double-pointer register, used with the EBMOVI instruction. Must be aligned on an address that is evenly divisible by 8. The value must be in the range of 00–F8H.
preg	A pointer register. Must be aligned on an address that is evenly divisible by 4. The value must be in the range of 00–FCH.
Sbreg†	A byte register in the lower register file that serves as the source of the instruction operation.
Slreg†	A 32-bit register in the lower register file that serves as the source of the instruction operation. Must be aligned on an address that is evenly divisible by 4. The value must be in the range of 00–FCH.
Swreg†	A word register in the lower register file that serves as the source of the instruction operation. Must be aligned on an address that is evenly divisible by 2. The value must be in the range of 00–FEH.
treg	A 24-bit register in the lower register file. Must be aligned on an address that is evenly divisible by 4. The value must be in the range of 00–FCH.
waop	A word operand that is addressed by any addressing mode.
w2_reg	A double-word register in the lower register file. Must be aligned on an address that is evenly divisible by 4. The value must be in the range of 00–FCH. Although *w2_reg* is similar to *lreg*, there is a distinction: *w2_reg* consists of two halves, each containing a 16-bit address; *lreg* is indivisible and contains a 32-bit number.
wreg	A word register in the lower register file. When it could be unclear whether this variable refers to a source or a destination register, it is prefixed with an *S* or a *D*. Must be aligned on an address that is evenly divisible by 2. The value must be in the range of 00–FEH.
xxx	The three high-order bits of displacement.

† The *D* or *S* prefix is used only when it could be unclear whether a variable refers to a destination or a source register.

Table A-6. Instruction Set

Mnemonic	Operation	Instruction Format
ADD (2 operands)	ADD WORDS. Adds the source and destination word operands and stores the sum into the destination operand. (DEST) ← (DEST) + (SRC) **PSW Flag Settings** Z: ✓ N: ✓ C: ✓ V: ✓ VT: ↑ ST: —	DEST, SRC ADD wreg, waop (011001aa) (waop) (wreg)
ADD (3 operands)	ADD WORDS. Adds the two source word operands and stores the sum into the destination operand. (DEST) ← (SRC1) + (SRC2) **PSW Flag Settings** Z: ✓ N: ✓ C: ✓ V: ✓ VT: ↑ ST: —	DEST, SRC1, SRC2 ADD Dwreg, Swreg, waop (010001aa) (waop) (Swreg) (Dwreg)
ADDB (2 operands)	ADD BYTES. Adds the source and destination byte operands and stores the sum into the destination operand. (DEST) ← (DEST) + (SRC) **PSW Flag Settings** Z: ✓ N: ✓ C: ✓ V: ✓ VT: ↑ ST: —	DEST, SRC ADDB breg, baop (011101aa) (baop) (breg)
ADDB (3 operands)	ADD BYTES. Adds the two source byte operands and stores the sum into the destination operand. (DEST) ← (SRC1) + (SRC2) **PSW Flag Settings** Z: ✓ N: ✓ C: ✓ V: ✓ VT: ↑ ST: —	DEST, SRC1, SRC2 ADDB Dbreg, Sbreg, baop (010101aa) (baop) (Sbreg) (Dbreg)
ADDC	ADD WORDS WITH CARRY. Adds the source and destination word operands and the carry flag (0 or 1) and stores the sum into the destination operand. (DEST) ← (DEST) + (SRC) + C **PSW Flag Settings** Z: ↓ N: ✓ C: ✓ V: ✓ VT: ↑ ST: —	DEST, SRC ADDC wreg, waop (101001aa) (waop) (wreg)

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>ADDCB</td><td>ADD BYTES WITH CARRY. Adds the source and destination byte operands and the carry flag (0 or 1) and stores the sum into the destination operand.
(DEST) ← (DEST) + (SRC) + C
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>↓</td><td>✓</td><td>✓</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>DEST, SRC
ADDCB breg, baop
(101101aa) (baop) (breg)</td></tr>
<tr><td>AND
(2 operands)</td><td>LOGICAL AND WORDS. ANDs the source and destination word operands and stores the result into the destination operand. The result has ones in only the bit positions in which both operands had a “1” and zeros in all other bit positions.
(DEST) ← (DEST) AND (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>0</td><td>0</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC
AND wreg, waop
(011000aa) (waop) (wreg)</td></tr>
<tr><td>AND
(3 operands)</td><td>LOGICAL AND WORDS. ANDs the two source word operands and stores the result into the destination operand. The result has ones in only the bit positions in which both operands had a “1” and zeros in all other bit positions.
(DEST) ← (SRC1) AND (SRC2)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>0</td><td>0</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC1, SRC2
AND Dwreg, Swreg, waop
(010000aa) (waop) (Swreg) (Dwreg)</td></tr>
<tr><td>ANDB
(2 operands)</td><td>LOGICAL AND BYTES. ANDs the source and destination byte operands and stores the result into the destination operand. The result has ones in only the bit positions in which both operands had a “1” and zeros in all other bit positions.
(DEST) ← (DEST) AND (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>0</td><td>0</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC
ANDB breg, baop
(011100aa) (baop) (breg)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>ANDB
(3 operands)</td><td>LOGICAL AND BYTES. ANDs the two source byte operands and stores the result into the destination operand. The result has ones in only the bit positions in which both operands had a “1” and zeros in all other bit positions.
(DEST) ← (SRC1) AND (SRC2)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>0</td><td>0</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC1, SRC2
ANDB Dbreg, Sbreg, baop
(010100aa) (baop) (Sbreg) (Dbreg)</td></tr>
<tr><td>BMOV</td><td>BLOCK MOVE. Moves a block of word data from one location in memory to another. The source and destination addresses are calculated using indirect addressing with autoincrement.
A long register (PTRS) addresses the source and destination pointers, which are stored in adjacent word registers. The source pointer (SRCPTR) is the low word and the destination pointer (DSTPTR) is the high word of PTRS.
A word register (CNTREG) specifies the number of transfers. CNTREG must reside in the lower register file; it cannot be windowed.
The blocks of word data can be located anywhere in page 00H, but should not overlap. Because the source (SRCPTR) and destination (DSTPTR) pointers are 16 bits wide, this instruction uses nonextended data moves. It cannot operate across page boundaries.
COUNT ← (CNTREG)
LOOP: SRCPTR ← (PTRS)
DSTPTR ← (PTRS + 2)
(DSTPTR) ← (SRCPTR)
(PTRS) ← SRCPTR + 2
(PTRS + 2) ← DSTPTR + 2
COUNT ← COUNT – 1
if COUNT ≠ 0 then
go to LOOP
end_if
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>PTRS, CNTREG
BMOV lreg, wreg
(11000001) (wreg) (lreg)
NOTE: The pointers are autoincremented during this instruction. However, CNTREG is not decremented. Therefore, it is easy to unintentionally create a long, uninterruptible operation with the BMOV instruction. Use the BMOVI instruction for an interruptible operation.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>BMOVI</td><td>INTERRUPTIBLE BLOCK MOVE. Moves a block of word data from one location in memory to another. The instruction is identical to BMOV, except that BMOVI is interruptible. The source and destination addresses are calculated using indirect addressing with autoincrement .

A long register (PTRS) addresses the source and destination pointers, which are stored in adjacent word registers. The source pointer (SRCPTR) is the low word and the destination pointer (DSTPTR) is the high word of PTRS.

A word register (CNTREG) specifies the number of transfers. CNTREG must reside in the lower register file; it cannot be windowed. The blocks of word data can be located anywhere in page 00H, but should not overlap. Because the source (SRCPTR) and destination (DSTPTR) pointers are 16 bits wide, this instruction uses nonexteneded data moves. It cannot operate across page boundaries. (If you need to cross page boundaries, use the EBMOVI instruction.)

COUNT ← (CNTREG)
LOOP: SRCPTR ← (PTRS)
DSTPTR ← (PTRS + 2)
(DSTPTR) ← (SRCPTR)
(PTRS) ← SRCPTR + 2
(PTRS + 2) ← DSTPTR + 2
COUNT ← COUNT – 1
if COUNT ≠ 0 then
go to LOOP
end_if

<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>PTRS, CNTREG
BMOVI lreg, wreg
(11001101) (wreg) (lreg)

NOTE: The pointers are autoincremented during this instruction. However, CNTREG is decremented only when the instruction is interrupted. When BMOVI is interrupted, CNTREG is updated to store the interim word count at the time of the interrupt. For this reason, you should always reload CNTREG before starting a BMOVI.</td></tr>
<tr><td>BR</td><td>BRANCH INDIRECT. Continues execution at the address specified in the operand word register.

PC ← (DEST)

<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>DEST
BR [wreg]
(11100011) (wreg)

NOTE: In 2-Mbyte mode, the BR instruction always branches to page FFH. Use the EBR instruction to branch to an address on any other page.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>CLR</td><td>CLEAR WORD. Clears the value of the operand.
(DEST) ← 0
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>—</td><td>—</td></tr></table></td><td>DEST
CLR wreg
(00000001) (wreg)</td></tr>
<tr><td>CLRB</td><td>CLEAR BYTE. Clears the value of the operand.
(DEST) ← 0
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>—</td><td>—</td></tr></table></td><td>DEST
CLRB breg
(00010001) (breg)</td></tr>
<tr><td>CLRC</td><td>CLEAR CARRY FLAG. Clears the carry flag.
C ← 0
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>0</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>CLRC
(11111000)</td></tr>
<tr><td>CLRVT</td><td>CLEAR OVERFLOW-TRAP FLAG. Clears the overflow-trap flag.
VT ← 0
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>0</td><td>—</td></tr></table></td><td>CLRVT
(11111100)</td></tr>
<tr><td>CMP</td><td>COMPARE WORDS. Subtracts the source word operand from the destination word operand. The flags are altered, but the operands remain unaffected. If a borrow occurs, the carry flag is cleared; otherwise, it is set.
(DEST) – (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>DEST, SRC
CMP wreg, waop
(100010aa) (waop) (wreg)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>CMPB</td><td>COMPARE BYTES. Subtracts the source byte operand from the destination byte operand. The flags are altered, but the operands remain unaffected. If a borrow occurs, the carry flag is cleared; otherwise, it is set.
(DEST) – (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>DEST, SRC
CMPB breg, baop
(100110aa) (baop) (breg)</td></tr>
<tr><td>CMPL</td><td>COMPARE LONG. Compares the magnitudes of two double-word (long) operands. The operands are specified using the direct addressing mode. The flags are altered, but the operands remain unaffected. If a borrow occurs, the carry flag is cleared; otherwise, it is set.
(DEST) – (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>—</td></tr></table></td><td>DEST, SRC
CMPL Dlreg, Slreg
(11000101) (Slreg) (Dlreg)</td></tr>
<tr><td>DEC</td><td>DECREMENT WORD. Decrements the value of the operand by one.
(DEST) ← (DEST) –1
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>DEST
DEC wreg
(00000101) (wreg)</td></tr>
<tr><td>DECB</td><td>DECREMENT BYTE. Decrements the value of the operand by one.
(DEST) ← (DEST) –1
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>DEST
DECB breg
(00010101) (breg)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>DI</td><td>DISABLE INTERRUPTS. Disables maskable interrupts. Interrupt calls cannot occur after this instruction.
Interrupt Enable (PSW.1) ← 0
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>DI
(11111010)</td></tr>
<tr><td>DIV</td><td>DIVIDE INTEGERS. Divides the contents of the destination long-integer operand by the contents of the source integer word operand, using signed arithmetic. It stores the quotient into the low-order word of the destination (i.e., the word with the lower address) and the remainder into the high-order word. The following two statements are performed concurrently.
(low word DEST) ← (DEST) / (SRC)
(high word DEST) ← (DEST) MOD (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>DEST, SRC
DIV lreg, waop
(11111110) (100011aa) (waop) (lreg)</td></tr>
<tr><td>DIVB</td><td>DIVIDE SHORT-INTEGERS. Divides the contents of the destination integer operand by the contents of the source short-integer operand, using signed arithmetic. It stores the quotient into the low-order byte of the destination (i.e., the word with the lower address) and the remainder into the high-order byte. The following two statements are performed concurrently.
(low byte DEST) ← (DEST) / (SRC)
(high byte DEST) ← (DEST) MOD (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>DEST, SRC
DIVB wreg, baop
(11111110) (100111aa) (baop) (wreg)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format
DIVU	DIVIDE WORDS, UNSIGNED. Divides the contents of the destination **double-word** operand by the contents of the source **word** operand, using unsigned arithmetic. It stores the quotient into the low-order word (i.e., the word with the lower address) of the destination operand and the remainder into the high-order word. The following two statements are performed concurrently. (low word DEST) ← (DEST) / (SRC) (high word DEST) ← (DEST) MOD (SRC) **PSW Flag Settings** Z: — N: — C: — V: ✓ VT: ↑ ST: —	DEST, SRC DIVU lreg, waop (100011aa) (waop) (lreg)
DIVUB	DIVIDE BYTES, UNSIGNED. This instruction divides the contents of the destination **word** operand by the contents of the source **byte** operand, using unsigned arithmetic. It stores the quotient into the low-order byte (i.e., the byte with the lower address) of the destination operand and the remainder into the high-order byte. The following two statements are performed concurrently. (low byte DEST) ← (DEST) / (SRC) (high byte DEST) ← (DEST) MOD (SRC) **PSW Flag Settings** Z: — N: — C: — V: ✓ VT: ↑ ST: —	DEST, SRC DIVUB wreg, baop (100111aa) (baop) (wreg)
DJNZ	DECREMENT AND JUMP IF NOT ZERO. Decrements the value of the byte operand by 1. If the result is 0, control passes to the next sequential instruction. If the result is not 0, the instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127. (COUNT) ← (COUNT) –1 if (COUNT) ≠ 0 then PC ← PC + 8-bit disp end_if **PSW Flag Settings** Z: — N: — C: — V: — VT: — ST: —	DJNZ breg, cadd (11100000) (breg) (disp) **NOTE:** The displacement (disp) is sign-extended to 24 bits.

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>DJNZW</td><td>DECREMENT AND JUMP IF NOT ZERO WORD. Decrements the value of the word operand by 1. If the result is 0, control passes to the next sequential instruction. If the result is not 0, the instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127.
(COUNT) ⟵ (COUNT) –1
if (COUNT) ≠ 0 then
PC ⟵ PC + 8-bit disp
end_if
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>DJNZW wreg, cadd
(11100001) (wreg) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits</td></tr>
<tr><td>DPTS</td><td>DISABLE PERIPHERAL TRANSACTION SERVER (PTS). Disables the peripheral transaction server (PTS).
PTS Disable (PSW.2) ⟵ 0
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>DPTS
(11101100)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>EBMOVI</td><td>EXTENDED INTERRUPTIBLE BLOCK MOVE. Moves a block of word data from one memory location to another. This instruction allows you to move blocks of up to 64K words between any two locations in the 16-Mbyte address space. This instruction is interruptible.

The source and destination addresses are calculated using the extended indirect with autoincrement addressing mode. A quad-word register (PTRS) addresses the 24-bit pointers, which are stored in adjacent double-word registers. The source pointer (SRCPTR) is the low double-word and the destination pointer is the high double-word of PTRS. A word register (CNTREG) specifies the number of transfers. This register must reside in the lower register file; it cannot be windowed. The blocks of data can reside anywhere in memory, but should not overlap.

COUNT ← (CNTREG)
LOOP: SRCPTR ← (PTRS)
DSTPTR ← (PTRS + 2)
(DSTPTR) ← (SRCPTR)
(PTRS) ← SRCPTR + 2
(PTRS + 2) ← DSTPTR + 2
COUNT ← COUNT 1
if COUNT ≠ 0 then
go to LOOP

<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>PTRS, CNTREG
EBMOVI prt2_reg, wreg
(11100100) (wreg) (prt2_reg)

NOTES: The pointers are autoincremented during this instruction. However, CNTREG is decremented only when the instruction is interrupted. When EBMOVI is interrupted, CNTREG is updated to store the interim word count at the time of the interrupt. For this reason, you should always reload CNTREG before starting an EBMOVI.

For 21-bit addresses, the offset must be in the range of +1048575 to –1048576.</td></tr>
<tr><td>EBR</td><td>EXTENDED BRANCH INDIRECT. Continues execution at the address specified in the operand word register. This instruction is an unconditional indirect jump to anywhere in the 16-Mbyte address space.

EBR shares its opcode (E3) with the BR instruction. To differentiate between the two, the compiler sets the least-significant bit of treg for the EBR instruction. For example: EBR [50] becomes E351 when compiled.

PC ← (DEST)

<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>DEST
EBR cadd
or
EBR [treg]
(11100011) (treg)

NOTE: For 21-bit addresses, the offset must be in the range of +1048575 to –1048576.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>ECALL</td><td>EXTENDED CALL. Pushes the contents of the program counter (the return address) onto the stack, then adds to the program counter the offset between the end of this instruction and the target label, effecting the call. The operand may be any address in the address space.

This instruction is an unconditional relative call to anywhere in the 16-Mbyte address space. It functions only in extended addressing mode.

SP ← SP – 4
(SP) ← PC
PC ← PC + 24-bit disp

<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>ECALL cadd
(1111 0001) (disp-low) (disp-high) (disp-ext)

NOTE: For 21-bit addresses, the offset must be in the range of +1048575 to –1048576.</td></tr>
<tr><td>EI</td><td>ENABLE INTERRUPTS. Enables maskable interrupts following the execution of the next statement. Interrupt calls cannot occur immediately following this instruction.

Interrupt Enable (PSW.1) ← 1

<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>EI
(11111011)</td></tr>
<tr><td>EJMP</td><td>EXTENDED JUMP. Adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The operand may be any address in the entire address space. The offset must be in the range of +8,388,607 to –8,388,608 for 24-bit addresses.

This instruction is an unconditional, relative jump to anywhere in the 16-Mbyte address space. It functions only in extended addressing mode.

PC ←PC + 24-bit disp

<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>?</td></tr></table></td><td>EJMP cadd
(11100110) (disp-low) (disp-high) (disp-ext)

NOTE: For 21-bit addresses, the offset must be in the range of +1048575 to –1048576.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>ELD</td><td>EXTENDED LOAD WORD. Loads the value of the source word operand into the destination operand.
This instruction allows you to move data from anywhere in the 16-Mbyte address space into the lower register file.
ext. indirect: (DEST) ← (SRC)
ext indexed: (DEST)← (SRC) + 24-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC
ELD wreg, [treg]
ext. indirect: (11101000) (treg) (wreg)
ext. indexed: (11101001) (treg) (disp-low) (disp-high) (disp-ext) (wreg)

NOTE: For 21-bit addresses, the offset must be in the range of +1048575 to –1048576.</td></tr>
<tr><td>ELDB</td><td>EXTENDED LOAD BYTE. Loads the value of the source byte operand into the destination operand.
This instruction allows you to move data from anywhere in the 16-Mbyte address space into the lower register file.
ext. indirect: (DEST) ← (SRC)
ext indexed: (DEST)← (SRC) + 24-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC
ELDB breg, [treg]
ext. indirect: (11101010) (treg) (breg)
ext. indexed: (11101011) (treg) (disp-low) (disp-high) (disp-ext) (breg)

NOTE: For 21-bit addresses, the offset must be in the range of +1048575 to –1048576.</td></tr>
<tr><td>EPTS</td><td>ENABLE PERIPHERAL TRANSACTION SERVER (PTS). Enables the peripheral transaction server (PTS).
PTS Enable (PSW.2) ← 1
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>EPTS
(11101101)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>EST</td><td>EXTENDED STORE WORD. Stores the value of the source (leftmost) word operand into the destination (rightmost) operand.
This instruction allows you to move data from the lower register file to anywhere in the 16-Mbyte address space.
ext. indirect: (DEST) ← (SRC)
ext indexed: (DEST)← (SRC) + 24-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>SRC, DEST
EST wreg, [treg]
ext. indirect: (00011100) (treg) (wreg)
ext. indexed: (00011101) (treg) (disp-low) (disp-high) (disp-ext) (wreg)

NOTE: For 21-bit addresses, the offset must be in the range of +1048575 to –1048576.</td></tr>
<tr><td>ESTB</td><td>EXTENDED STORE BYTE. Stores the value of the source (leftmost) byte operand into the destination (rightmost) operand.
This instruction allows you to move data from the lower register file to anywhere in the 16-Mbyte address space.
ext. indirect: (DEST) ← (SRC)
ext indexed: (DEST)← (SRC) + 24-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>SRC, DEST
ESTB breg, [treg]
ext. indirect: (00011110) (treg) (breg)
ext. indexed: (00011111) (treg) (disp-low) (disp-high) (disp-ext) (breg)

NOTE: For 21-bit addresses, the offset must be in the range of +1048575 to –1048576.</td></tr>
<tr><td>EXT</td><td>SIGN-EXTEND INTEGER INTO LONG-INTEGER. Sign-extends the low-order word of the operand throughout the high-order word of the operand.
if DEST.15 = 1 then
(high word DEST) ← 0FFFFH
else
(high word DEST) ← 0
end_if
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>0</td><td>0</td><td>—</td><td>—</td></tr></table></td><td>EXT lreg
(00000110) (lreg)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>EXTB</td><td>SIGN-EXTEND SHORT-INTEGER INTO INTEGER. Sign-extends the low-order byte of the operand throughout the high-order byte of the operand.
if DEST.7 = 1 then
(high byte DEST) ← 0FFH
else
(high byte DEST) ← 0
end_if
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>0</td><td>0</td><td>—</td><td>—</td></tr></table></td><td>EXTB wreg
(00010110) (wreg)</td></tr>
<tr><td>IDLPD</td><td>IDLE/POWERDOWN. Depending on the 8-bit value of the KEY operand, this instruction causes the device to:
• enter idle mode, if KEY=1,
• enter powerdown mode, if KEY=2,
• execute a reset sequence, if KEY > 3.
The bus controller completes any prefetch cycle in progress before the CPU stops or resets.
if KEY = 1 then
enter idle
else if KEY = 2 then
enter powerdown
else if key > 3 then
execute reset
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td colspan="6">KEY = 1 or 2</td></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr><tr><td colspan="6">KEY = any value other than 1, 2, or 3</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table></td><td>IDLPD #key
(11110110) (key)</td></tr>
<tr><td>INC</td><td>INCREMENT WORD. Increments the value of the word operand by 1.
(DEST) ← (DEST) + 1
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>↑</td><td>0</td></tr></table></td><td>INC wreg
(00000111) (wreg)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>INCB</td><td>INCREMENT BYTE. Increments the value of the byte operand by 1.
(DEST) ← (DEST) + 1
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>INCB breg
(00010111) (breg)</td></tr>
<tr><td>JBC</td><td>JUMP IF BIT IS CLEAR. Tests the specified bit. If the bit is set, control passes to the next sequential instruction. If the bit is clear, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127.
if (specified bit) = 0 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JBC breg, bitno, cadd
(00110bbb) (breg) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
<tr><td>JBS</td><td>JUMP IF BIT IS SET. Tests the specified bit. If the bit is clear, control passes to the next sequential instruction. If the bit is set, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127.
if (specified bit) = 1 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JBS breg, bitno, cadd
(00111bbb) (breg) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>JC</td><td>JUMP IF CARRY FLAG IS SET. Tests the carry flag. If the carry flag is clear, control passes to the next sequential instruction. If the carry flag is set, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127.
if C = 1 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JC cadd
(11011011) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
<tr><td>JE</td><td>JUMP IF EQUAL. Tests the zero flag. If the flag is clear, control passes to the next sequential instruction. If the zero flag is set, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127.
if Z = 1 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JE cadd
(11011111) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
<tr><td>JGE</td><td>JUMP IF SIGNED GREATER THAN OR EQUAL. Tests the negative flag. If the negative flag is set, control passes to the next sequential instruction. If the negative flag is clear, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127.
if N = 0 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JGE cadd
(11010110) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>JGT</td><td>JUMP IF SIGNED GREATER THAN. Tests both the zero flag and the negative flag. If either flag is set, control passes to the next sequential instruction. If both flags are clear, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127.
if N = 0 AND Z = 0 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JGT cadd
(11010010) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
<tr><td>JH</td><td>JUMP IF HIGHER (UNSIGNED). Tests both the zero flag and the carry flag. If either the carry flag is clear or the zero flag is set, control passes to the next sequential instruction. If the carry flag is set and the zero flag is clear, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in range of –128 to +127.
if C = 1 AND Z = 0 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JH cadd
(11011001) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
<tr><td>JLE</td><td>JUMP IF SIGNED LESS THAN OR EQUAL. Tests both the negative flag and the zero flag. If both flags are clear, control passes to the next sequential instruction. If either flag is set, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127.
if N = 1 OR Z = 1 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JLE cadd
(11011010) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>JLT</td><td>JUMP IF SIGNED LESS THAN. Tests the negative flag. If the flag is clear, control passes to the next sequential instruction. If the negative flag is set, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127.
if N = 1 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JLT cadd
(11011110) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
<tr><td>JNC</td><td>JUMP IF CARRY FLAG IS CLEAR. Tests the carry flag. If the flag is set, control passes to the next sequential instruction. If the carry flag is clear, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127.
if C = 0 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JNC cadd
(11010011) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
<tr><td>JNE</td><td>JUMP IF NOT EQUAL. Tests the zero flag. If the flag is set, control passes to the next sequential instruction. If the zero flag is clear, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127.
if Z = 0 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JNE cadd
(11010111) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>JNH</td><td>JUMP IF NOT HIGHER (UNSIGNED). Tests both the zero flag and the carry flag. If the carry flag is set and the zero flag is clear, control passes to the next sequential instruction. If either the carry flag is clear or the zero flag is set, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in range of –128 to +127.
if C = 0 OR Z = 1 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JNH cadd
(11010001) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
<tr><td>JNST</td><td>JUMP IF STICKY BIT FLAG IS CLEAR. Tests the sticky bit flag. If the flag is set, control passes to the next sequential instruction. If the sticky bit flag is clear, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in range of –128 to +127.
if ST = 0 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JNST cadd
(11010000) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
<tr><td>JNV</td><td>JUMP IF OVERFLOW FLAG IS CLEAR. Tests the overflow flag. If the flag is set, control passes to the next sequential instruction. If the overflow flag is clear, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in range of –128 to +127.
if V = 0 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JNV cadd
(11010101) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>JNVT</td><td>JUMP IF OVERFLOW-TRAP FLAG IS CLEAR. Tests the overflow-trap flag. If the flag is set, this instruction clears the flag and passes control to the next sequential instruction. If the overflow-trap flag is clear, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in range of –128 to +127.
if VT = 0 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>0</td><td>—</td></tr></table></td><td>JNVT cadd
(11010100) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
<tr><td>JST</td><td>JUMP IF STICKY BIT FLAG IS SET. Tests the sticky bit flag. If the flag is clear, control passes to the next sequential instruction. If the sticky bit flag is set, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in range of –128 to +127.
if ST = 1 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JST cadd
(11011000) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
<tr><td>JV</td><td>JUMP IF OVERFLOW FLAG IS SET. Tests the overflow flag. If the flag is clear, control passes to the next sequential instruction. If the overflow flag is set, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in range of –128 to +127.
if V = 1 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JV cadd
(11011101) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>JVT</td><td>JUMP IF OVERFLOW-TRAP FLAG IS SET. Tests the overflow-trap flag. If the flag is clear, control passes to the next sequential instruction. If the overflow-trap flag is set, this instruction clears the flag and adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in range of –128 to +127.
if VT = 1 then
PC ← PC + 8-bit disp

<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>0</td><td>—</td></tr></table></td><td>JVT cadd
(11011100) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
<tr><td>LCALL</td><td>LONG CALL. Pushes the contents of the program counter (the return address) onto the stack, then adds to the program counter the offset between the end of this instruction and the target label, effecting the call. The offset must be in the range of –32,768 to +32,767.
64-Kbyte mode:
SP ← SP – 2
(SP) ← PC
PC ← PC + 16-bit disp
2-Mbyte mode:
SP ← SP – 4
(SP) ← PC
PC ← PC + 24-bit disp

<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>LCALL cadd
(11101111) (disp-low) (disp-high)

NOTE: The displacement (disp) is sign-extended to 24 bits in the 2-Mbyte addressing mode. This displacement may cause the program counter to cross a page boundary.</td></tr>
<tr><td>LD</td><td>LOAD WORD. Loads the value of the source word operand into the destination operand.
(DEST) ← (SRC)

<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC
LD wreg, waop
(101000aa) (waop) (wreg)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>LDB</td><td>LOAD BYTE. Loads the value of the source byte operand into the destination operand.
(DEST) ← (SRC)

<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC
LDB breg, baop
(101100aa) (baop) (breg)</td></tr>
<tr><td>LDBSE</td><td>LOAD BYTE SIGN-EXTENDED. Sign-extends the value of the source short-integer operand and loads it into the destination integer operand.
(low byte DEST) ← (SRC)
if DEST.15 = 1 then
(high word DEST) ← 0FFH
else
(high word DEST) ← 0
end_if

<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC
LDBSE wreg, baop
(101111aa) (baop) (wreg)</td></tr>
<tr><td>LDBZE</td><td>LOAD BYTE ZERO-EXTENDED. Zero-extends the value of the source byte operand and loads it into the destination word operand.
(low byte DEST) ← (SRC)
(high byte DEST) ← 0

<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC
LDBZE wreg, baop
(101011aa) (baop) (wreg)</td></tr>
<tr><td>LJMP</td><td>LONG JUMP. Adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –32,768 to +32,767.
64-Kbyte mode:
PC ← PC + 16-bit disp
2-Mbyte mode:
PC ← PC + 24-bit disp

<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>?</td></tr></table></td><td>LJMP cadd
(11100111) (disp-low) (disp-high)

NOTE: The displacement (disp) is sign-extended to 24 bits in the 2-Mbyte addressing mode. This displacement may cause the program counter to cross a page boundary.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>MUL
(2 operands)</td><td>MULTIPLY INTEGERS. Multiplies the source and destination integer operands, using signed arithmetic, and stores the 32-bit result into the destination long-integer operand. The sticky bit flag is undefined after the instruction is executed.
(DEST) ← (DEST) × (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>?</td></tr></table></td><td>DEST, SRC
MUL lreg, waop
(11111110) (011011aa) (waop) (lreg)</td></tr>
<tr><td>MUL
(3 operands)</td><td>MULTIPLY INTEGERS. Multiplies the two source integer operands, using signed arithmetic, and stores the 32-bit result into the destination long-integer operand. The sticky bit flag is undefined after the instruction is executed.
(DEST) ← (SRC1) × (SRC2)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>?</td></tr></table></td><td>DEST, SRC1, SRC2
MUL lreg, wreg, waop
(11111110) (010011aa) (waop) (wreg) (lreg)</td></tr>
<tr><td>MULB
(2 operands)</td><td>MULTIPLY SHORT-INTEGERS. Multiplies the source and destination short-integer operands, using signed arithmetic, and stores the 16-bit result into the destination integer operand. The sticky bit flag is undefined after the instruction is executed.
(DEST) ← (DEST) × (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>?</td></tr></table></td><td>DEST, SRC
MULB wreg, baop
(11111110) (011111aa) (baop) (wreg)</td></tr>
<tr><td>MULB
(3 operands)</td><td>MULTIPLY SHORT-INTEGERS. Multiplies the two source short-integer operands, using signed arithmetic, and stores the 16-bit result into the destination integer operand. The sticky bit flag is undefined after the instruction is executed.
(DEST) ← (SRC1) × (SRC2)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>?</td></tr></table></td><td>DEST, SRC1, SRC2
MULB wreg, breg, baop
(11111110) (010111aa) (baop) (breg) (wreg)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>MULU
(2 operands)</td><td>MULTIPLY WORDS, UNSIGNED. Multiplies the source and destination word operands, using unsigned arithmetic, and stores the 32-bit result into the destination double-word operand. The sticky bit flag is undefined after the instruction is executed.
(DEST) ← (DEST) × (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>?</td></tr></table></td><td>DEST, SRC
MULU lreg, waop
(011011aa) (waop) (lreg)</td></tr>
<tr><td>MULU
(3 operands)</td><td>MULTIPLY WORDS, UNSIGNED. Multiplies the two source word operands, using unsigned arithmetic, and stores the 32-bit result into the destination double-word operand. The sticky bit flag is undefined after the instruction is executed.
(DEST) ← (SRC1) × (SRC2)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>?</td></tr></table></td><td>DEST, SRC1, SRC2
MULU lreg, wreg, waop
(010011aa) (waop) (wreg) (lreg)</td></tr>
<tr><td>MULUB
(2 operands)</td><td>MULTIPLY BYTES, UNSIGNED. Multiplies the source and destination operands, using unsigned arithmetic, and stores the word result into the destination operand. The sticky bit flag is undefined after the instruction is executed.
(DEST) ← (DEST) × (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>?</td></tr></table></td><td>DEST, SRC
MULUB wreg, baop
(011111aa) (baop) (wreg)</td></tr>
<tr><td>MULUB
(3 operands)</td><td>MULTIPLY BYTES, UNSIGNED. Multiplies the two source byte operands, using unsigned arithmetic, and stores the word result into the destination operand. The sticky bit flag is undefined after the instruction is executed.
(DEST) ← (SRC1) × (SRC2)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>?</td></tr></table></td><td>DEST, SRC1, SRC2
MULUB wreg, breg, baop
(010111aa) (baop) (breg) (wreg)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>NEG</td><td>NEGATE INTEGER. Negates the value of the integer operand.
(DEST) ← – (DEST)

<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>NEG wreg
(00000011) (wreg)</td></tr>
<tr><td>NEGB</td><td>NEGATE SHORT-INTEGER. Negates the value of the short-integer operand.
(DEST) ← – (DEST)

<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>NEGB breg
(00010011) (breg)</td></tr>
<tr><td>NOP</td><td>NO OPERATION. Does nothing. Control passes to the next sequential instruction.

<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>NOP
(11111101)</td></tr>
<tr><td>NORML</td><td>NORMALIZE LONG-INTEGER. Normalizes the source (leftmost) long-integer operand. (That is, it shifts the operand to the left until its most significant bit is “1” or until it has performed 31 shifts). If the most significant bit is still “0” after 31 shifts, the instruction stops the process and sets the zero flag. The instruction stores the actual number of shifts performed in the destination (rightmost) operand.
(COUNT) ← 0
do while
(MSB (DEST) = 0) AND (COUNT) < 31)
(DEST) ← (DEST) × 2
(COUNT) ← (COUNT) + 1
end_while

<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>?</td><td>0</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>SRC, DEST
NORML lreg, breg
(00001111) (breg) (lreg)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>NOT</td><td>COMPLEMENT WORD. Complements the value of the word operand (replaces each “1” with a “0” and each “0” with a “1”).
(DEST) ← NOT (DEST)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>0</td><td>0</td><td>—</td><td>—</td></tr></table></td><td>NOT wreg
(00000010) (wreg)</td></tr>
<tr><td>NOTB</td><td>COMPLEMENT BYTE. Complements the value of the byte operand (replaces each “1” with a “0” and each “0” with a “1”).
(DEST) ← NOT (DEST)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>0</td><td>0</td><td>—</td><td>—</td></tr></table></td><td>NOTB breg
(00010010) (breg)</td></tr>
<tr><td>OR</td><td>LOGICAL OR WORDS. ORs the source word operand with the destination word operand and replaces the original destination operand with the result. The result has a “1” in each bit position in which either the source or destination operand had a “1”.
(DEST) ← (DEST) OR (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>0</td><td>0</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC
OR wreg, waop
(100000aa) (waop) (wreg)</td></tr>
<tr><td>ORB</td><td>LOGICAL OR BYTES. ORs the source byte operand with the destination byte operand and replaces the original destination operand with the result. The result has a “1” in each bit position in which either the source or destination operand had a “1”.
(DEST) ← (DEST) OR (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>0</td><td>0</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC
ORB breg, baop
(100100aa) (baop) (breg)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>POP</td><td>POP WORD. Pops the word on top of the stack and places it at the destination operand.
(DEST) ← (SP)
SP ← SP + 2
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>POP waop
(110011aa) (waop)</td></tr>
<tr><td>POPA</td><td>POP ALL. This instruction is used instead of POPF, to support the eight additional interrupts. It pops two words off the stack and places the first word into the INT_MASK1/WSR register pair and the second word into the PSW/INT_MASK register-pair. This instruction increments the SP by 4. Interrupt calls cannot occur immediately following this instruction.
INT_MASK1/WSR ← (SP)
SP ← SP + 2
PSW/INT_MASK ← (SP)
SP ← SP + 2
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td></tr></table></td><td>POPA
(11110101)</td></tr>
<tr><td>POPF</td><td>POP FLAGS. Pops the word on top of the stack and places it into the PSW. Interrupt calls cannot occur immediately following this instruction.
(PSW)/INT_MASK ← (SP)
SP ← SP + 2
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td></tr></table></td><td>POPF
(11110011)</td></tr>
<tr><td>PUSH</td><td>PUSH WORD. Pushes the word operand onto the stack.
SP ← SP – 2
(SP) ← (DEST)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>PUSH waop
(110010aa) (waop)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>PUSHA</td><td>PUSH ALL. This instruction is used instead of PUSHF, to support the eight additional interrupts. It pushes two words — PSW/INT_MASK and INT_MASK1/WSR — onto the stack.
This instruction clears the PSW, INT_MASK, and INT_MASK1 registers and decrements the SP by 4. Interrupt calls cannot occur immediately following this instruction.
SP ← SP – 2
(SP) ← PSW/INT_MASK
PSW/INT_MASK ← 0
SP ← SP – 2
(SP) ← INT_MASK1/WSR
INT_MASK1 ← 0
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table></td><td>PUSHA
(11110100)</td></tr>
<tr><td>PUSHF</td><td>PUSH FLAGS. Pushes the PSW onto the top of the stack, then clears it. Clearing the PSW disables interrupt servicing. Interrupt calls cannot occur immediately following this instruction.
SP ← SP – 2
(SP) ← PSW/INT_MASK
PSW/INT_MASK ← 0
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table></td><td>PUSHF
(11110010)</td></tr>
<tr><td>RET</td><td>RETURN FROM SUBROUTINE. Pops the PC off the top of the stack.
64-Kbyte mode:
PC ← (SP)
SP ← SP + 2
2-Mbyte mode:
PC ← (SP)
SP ← SP + 4
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>RET
(11110000)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>RST</td><td>RESET SYSTEM. Initializes the PSW to zero, the PC to FF2080H, and the pins and SFRs to their reset values. Executing this instruction causes the RESET# pin to be pulled low for 16 state times.
SFR ← Reset Status
Pin ← Reset Status
PSW ← 0
PC ← FF2080H
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table></td><td>RST
(11111111)</td></tr>
<tr><td>SCALL</td><td>SHORT CALL. Pushes the contents of the program counter (the return address) onto the stack, then adds to the program counter the offset between the end of this instruction and the target label, effecting the call. The offset must be in the range of –1024 to +1023.
64-Kbyte mode:
SP ← SP – 2
(SP) ← PC
PC←PC +11-bit disp
2-Mbyte mode:
SP ← SP – 4
(SP) ← PC
PC←PC +11-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>SCALL cadd
(00101xxx) (disp-low)
NOTE: The displacement (disp) is sign-extended to 16-bits in the 64-Kbyte addressing mode and to 24 bits in the 2-Mbyte addressing mode. This displacement may cause the program counter to cross a page boundary in 2-Mbyte mode.</td></tr>
<tr><td>SETC</td><td>SET CARRY FLAG. Sets the carry flag.
C ← 1
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>1</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>SETC
(11111001)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>SHL</td><td>SHIFT WORD LEFT. Shifts the destination word operand to the left as many times as specified by the count operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH), inclusive, or as the content of any register (10–0FFH) with a value in the range of 0 to 31 (1FH), inclusive. The right bits of the result are filled with zeros. The last bit shifted out is saved in the carry flag.
temp ← (COUNT)
do while temp ≠ 0
C ← High order bit of (DEST)
(DEST) ← (DEST) × 2
temp ← temp – 1
end_while
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>SHL wreg, #count
(00001001) (count) (wreg)
or
SHL wreg, breg
(00001001) (breg) (wreg)</td></tr>
<tr><td>SHLB</td><td>SHIFT BYTE LEFT. Shifts the destination byte operand to the left as many times as specified by the count operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH), inclusive, or as the content of any register (10–0FFH) with a value in the range of 0 to 31 (1FH), inclusive. The right bits of the result are filled with zeros. The last bit shifted out is saved in the carry flag.
temp ← (COUNT)
do while temp ≠ 0
C ← High order bit of (DEST)
(DEST) ← (DEST) × 2
temp ← temp – 1
end_while
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>SHLB breg, #count
(00011001) (count) (breg)
or
SHLB breg, breg
(00011001) (breg) (breg)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>SHLL</td><td>SHIFT DOUBLE-WORD LEFT. Shifts the destination double-word operand to the left as many times as specified by the count operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH), inclusive, or as the content of any register (10–0FFH) with a value in the range of 0 to 31 (1FH), inclusive. The right bits of the result are filled with zeros. The last bit shifted out is saved in the carry flag.
temp ← (COUNT)
do while temp ≠ 0
C ← High order bit of (DEST)
(DEST) ← (DEST) × 2
temp ← temp – 1
end_while
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>SHLL lreg, #count
(00001101) (count) (lreg)
or
SHLL lreg, breg
(00001101) (breg) (lreg)</td></tr>
<tr><td>SHR</td><td>LOGICAL RIGHT SHIFT WORD. Shifts the destination word operand to the right as many times as specified by the count operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH), inclusive, or as the content of any register (10–0FFH) with a value in the range of 0 to 31 (1FH), inclusive. The left bits of the result are filled with zeros. The last bit shifted out is saved in the carry flag.
temp ← (COUNT)
do while temp ≠ 0
C ← Low order bit of (DEST)
(DEST) ← (DEST)/2
temp ← temp – 1
end_while
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>0</td><td>✓</td><td>0</td><td>—</td><td>✓</td></tr></table></td><td>SHR wreg, #count
(00001000) (count) (wreg)
or
SHR wreg, breg
(00001000) (breg) (wreg)

NOTES: This instruction clears the sticky bit flag at the beginning of the instruction. If at any time during the shift a “1” is shifted into the carry flag and another shift cycle occurs, the instruction sets the sticky bit flag.

In this operation, (DEST)/2 represents unsigned division.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>SHRA</td><td>ARITHMETIC RIGHT SHIFT WORD. Shifts the destination word operand to the right as many times as specified by the count operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH), inclusive, or as the content of any register (10–0FFH) with a value in the range of 0 to 31 (1FH), inclusive. If the original high order bit value was “0,” zeros are shifted in. If the value was “1,” ones are shifted in. The last bit shifted out is saved in the carry flag.
temp ← (COUNT)
do while temp ≠ 0
C ← Low order bit of (DEST)
(DEST) ← (DEST)/2
temp ← temp – 1
end_while
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>0</td><td>—</td><td>✓</td></tr></table></td><td>SHRA wreg, #count
(00001010) (count) (wreg)
or
SHRA wreg, breg
(00001010) (breg) (wreg)

NOTES: This instruction clears the sticky bit flag at the beginning of the instruction. If at any time during the shift a “1” is shifted into the carry flag and another shift cycle occurs, the instruction sets the sticky bit flag.

In this operation, (DEST)/2 represents signed division.</td></tr>
<tr><td>SHRAB</td><td>ARITHMETIC RIGHT SHIFT BYTE. Shifts the destination byte operand to the right as many times as specified by the count operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH), inclusive, or as the content of any register (10–0FFH) with a value in the range of 0 to 31 (1FH), inclusive. If the original high order bit value was “0,” zeros are shifted in. If the value was “1,” ones are shifted in. The last bit shifted out is saved in the carry flag.
temp ← (COUNT)
do while temp ≠ 0
C ← Low order bit of (DEST)
(DEST) ← (DEST)/2
temp ← temp – 1
end_while
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>0</td><td>—</td><td>✓</td></tr></table></td><td>SHRAB breg, #count
(00011010) (count) (breg)
or
SHRAB breg, breg
(00011010) (breg) (breg)

NOTES: This instruction clears the sticky bit flag at the beginning of the instruction. If at any time during the shift a “1” is shifted into the carry flag and another shift cycle occurs, the instruction sets the sticky bit flag.

In this operation, (DEST)/2 represents signed division.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>SHRAL</td><td>ARITHMETIC RIGHT SHIFT DOUBLE-WORD. Shifts the destination double-word operand to the right as many times as specified by the count operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH), inclusive, or as the content of any register (10–0FFH) with a value in the range of 0 to 31 (1FH), inclusive. If the original high order bit value was “0,” zeros are shifted in. If the value was “1,” ones are shifted in.
temp ← (COUNT)
do while temp ≠ 0
C ← Low order bit of (DEST)
(DEST) ← (DEST)/2
temp ← temp – 1
end_while
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>0</td><td>—</td><td>✓</td></tr></table></td><td>SHRAL lreg, #count
(00001110) (count) (lreg)
or
SHRAL lreg, breg
(00001110) (breg) (lreg)

NOTES: This instruction clears the sticky bit flag at the beginning of the instruction. If at any time during the shift a “1” is shifted into the carry flag and another shift cycle occurs, the instruction sets the sticky bit flag.

In this operation, (DEST)/2 represents signed division.</td></tr>
<tr><td>SHRB</td><td>LOGICAL RIGHT SHIFT BYTE. Shifts the destination byte operand to the right as many times as specified by the count operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH), inclusive, or as the content of any register (10–0FFH) with a value in the range of 0 to 31 (1FH), inclusive. The left bits of the result are filled with zeros. The last bit shifted out is saved in the carry flag.
temp ← (COUNT)
do while temp ≠ 0
C ← Low order bit of (DEST)
(DEST) ← (DEST)/2
temp ← temp–1
end_while
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>0</td><td>✓</td><td>0</td><td>—</td><td>✓</td></tr></table></td><td>SHRB breg, #count
(00011000) (count) (breg)
or
SHRB breg, breg
(00011000) (breg) (breg)

NOTES: This instruction clears the sticky bit flag at the beginning of the instruction. If at any time during the shift a “1” is shifted into the carry flag and another shift cycle occurs, the instruction sets the sticky bit flag.

In this operation, (DEST)/2 represents unsigned division.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>SHRL</td><td>LOGICAL RIGHT SHIFT DOUBLE-WORD. Shifts the destination double-word operand to the right as many times as specified by the count operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH), inclusive, or as the content of any register (10–0FFH) with a value in the range of 0 to 31 (1FH), inclusive. The left bits of the result are filled with zeros. The last bit shifted out is saved in the carry flag.
temp ← (COUNT)
do while temp ≠ 0
C ← Low order bit of (DEST)
(DEST) ← (DEST)/2)
temp ← temp – 1
end_while
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>0</td><td>✓</td><td>0</td><td>—</td><td>✓</td></tr></table></td><td>SHRL lreg, #count
(00001100) (count) (lreg)
or
SHRL lreg, breg
(00001100) (breg) (lreg)

NOTES: This instruction clears the sticky bit flag at the beginning of the instruction. If at any time during the shift a “1” is shifted into the carry flag and another shift cycle occurs, the instruction sets the sticky bit flag.

In this operation, (DEST)/2 represents unsigned division.</td></tr>
<tr><td>SJMP</td><td>SHORT JUMP. Adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –1024 to +1023, inclusive.
PC ← PC + 11-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>SJMP cadd
(00100xxx) (disp-low)

NOTE: The displacement (disp) is sign-extended to 16 bits in the 64-Kbyte addressing mode and to 24 bits in the 2-Mbyte addressing mode. This displacement may cause the program counter to cross a page boundary in 2-Mbyte mode.</td></tr>
<tr><td>SKIP</td><td>TWO BYTE NO-OPERATION. Does nothing. Control passes to the next sequential instruction. This is actually a two-byte NOP in which the second byte can be any value and is simply ignored.
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>SKIP breg
(00000000) (breg)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format
ST	STORE WORD. Stores the value of the source **(leftmost)** word operand into the destination **(rightmost)** operand. (DEST) ← (SRC) **PSW Flag Settings** Z N C V VT ST — — — — — —	SRC, DEST ST wreg, waop (110000aa) (waop) (wreg)
STB	STORE BYTE. Stores the value of the source **(leftmost)** byte operand into the destination **(rightmost)** operand. (DEST) ← (SRC) **PSW Flag Settings** Z N C V VT ST — — — — — —	SRC, DEST STB breg, baop (110001aa) (baop) (breg)
SUB (2 operands)	SUBTRACT WORDS. Subtracts the source word operand from the destination word operand, stores the result in the destination operand, and sets the carry flag as the complement of borrow. (DEST) ← (DEST) – (SRC) **PSW Flag Settings** Z N C V VT ST ✓ ✓ ✓ ✓ ↑ —	DEST, SRC SUB wreg, waop (011010aa) (waop) (wreg)
SUB (3 operands)	SUBTRACT WORDS. Subtracts the first source word operand from the second, stores the result in the destination operand, and sets the carry flag as the complement of borrow. (DEST) ← (SRC1) – (SRC2) **PSW Flag Settings** Z N C V VT ST ✓ ✓ ✓ ✓ ↑ —	DEST, SRC1, SRC2 SUB Dwreg, Swreg, waop (010010aa) (waop) (Swreg) (Dwreg)

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>SUBB
(2 operands)</td><td>SUBTRACT BYTES. Subtracts the source byte operand from the destination byte operand, stores the result in the destination operand, and sets the carry flag as the complement of borrow.
(DEST) ← (DEST) – (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>DEST, SRC
SUBB breg, baop
(011110aa) (baop) (breg)</td></tr>
<tr><td>SUBB
(3 operands)</td><td>SUBTRACT BYTES. Subtracts the second source byte operand from the first, stores the result in the destination operand, and sets the carry flag as the complement of borrow.
(DEST) ← (SRC1) – (SRC2)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>DEST, SRC1, SRC2
SUBB Dbreg, Sbreg, baop
(010110aa) (baop) (Sbreg) (Dbreg)</td></tr>
<tr><td>SUBC</td><td>SUBTRACT WORDS WITH BORROW. Subtracts the source word operand from the destination word operand. If the carry flag was clear, SUBC subtracts 1 from the result. It stores the result in the destination operand and sets the carry flag as the complement of borrow.
(DEST) ← (DEST) – (SRC) – (1–C)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>↓</td><td>✓</td><td>✓</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>DEST, SRC
SUBC wreg, waop
(101010aa) (waop) (wreg)</td></tr>
<tr><td>SUBCB</td><td>SUBTRACT BYTES WITH BORROW. Subtracts the source byte operand from the destination byte operand. If the carry flag was clear, SUBCB subtracts 1 from the result. It stores the result in the destination operand and sets the carry flag as the complement of borrow.
(DEST) ← (DEST) – (SRC) – (1–C)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>↓</td><td>✓</td><td>✓</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>DEST, SRC
SUBCB breg, baop
(101110aa) (baop) (breg)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>TIJMP</td><td>TABLE INDIRECT JUMP. Causes execution to continue at an address selected from a table of addresses.

The first word register, TBASE, contains the 16-bit address of the beginning of the jump table. TBASE can be located in RAM up to FEH without windowing or above FFH with windowing. The jump table itself can be placed at any nonreserved memory location on a word boundary in page FFH.

The second word register, INDEX, contains the 16-bit address that points to a register containing a 7-bit value. This value is used to calculate the offset into the jump table. Like TBASE, INDEX can be located in RAM up to FEH without windowing or above FFH with windowing. Note that the 16-bit address contained in INDEX is absolute; it disregards any windowing that may be in effect when the TIJMP instruction is executed.

The byte operand, #MASK, is 7-bit immediate data to mask INDEX. #MASK is ANDed with INDEX to determine the offset (OFFSET). OFFSET is multiplied by two, then added to the base address (TBASE) to determine the destination address (DEST X) in page FFH.

[INDEX] AND #MASK = OFFSET
(2 × OFFSET) + TBASE = DEST X
PC ← (DEST X)

<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>TIJMP TBASE, [INDEX], #MASK
(11100010) [INDEX] (#MASK) (TBASE)

NOTE: TIJMP multiplies OFFSET by two to provide for word alignment of the jump table.</td></tr>
<tr><td>TRAP</td><td>SOFTWARE TRAP. This instruction causes an interrupt call that is vectored through location FF2010H. The operation of this instruction is not affected by the state of the interrupt enable flag (I) in the PSW. Interrupt calls cannot occur immediately following this instruction.

64-Kbyte mode:
SP ← SP – 2
(SP) ← PC
PC ← (2010H)

2-Mbyte mode:
SP ← SP – 4
(SP) ← PC
PC ← (0FF2010H)

<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>TRAP
(11110111)

NOTE: This instruction is not supported by assemblers. The TRAP instruction is intended for use by development tools. These tools may not support user-application of this instruction.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>XCH</td><td>EXCHANGE WORD. Exchanges the value of the source word operand with that of the destination word operand.
(DEST) ↔ (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC
XCH wreg, waop
(00000100) (waop) (wreg) direct
(00001011) (waop) (wreg) indexed</td></tr>
<tr><td>XCHB</td><td>EXCHANGE BYTE. Exchanges the value of the source byte operand with that of the destination byte operand.
(DEST) ↔ (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC
XCHB breg, baop
(00010100) (baop) (breg) direct
(00011011) (baop) (breg) indexed</td></tr>
<tr><td>XOR</td><td>LOGICAL EXCLUSIVE-OR WORDS. XORs the source word operand with the destination word operand and stores the result in the destination operand. The result has ones in the bit positions in which either operand (but not both) had a “1” and zeros in all other bit positions.
(DEST) ← (DEST) XOR (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>0</td><td>0</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC
XOR wreg, waop
(100001aa) (waop) (wreg)</td></tr>
<tr><td>XORB</td><td>LOGICAL EXCLUSIVE-OR BYTES. XORs the source byte operand with the destination byte operand and stores the result in the destination operand. The result has ones in the bit positions in which either operand (but not both) had a “1” and zeros in all other bit positions.
(DEST) ← (DEST) XOR (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>0</td><td>0</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC
XORB breg, baop
(100101aa) (baop) (breg)</td></tr>
</table>

Table A-7 lists the instruction opcodes, in hexadecimal order, along with the corresponding instruction mnemonics.

Table A-7. Instruction Opcodes

Hex Code	Instruction Mnemonic
00	SKIP
01	CLR
02	NOT
03	NEG
04	XCH Direct
05	DEC
06	EXT
07	INC
08	SHR
09	SHL
0A	SHRA
0B	XCH Indexed
0C	SHRL
0D	SHLL
0E	SHRAL
0F	NORML
10	Reserved
11	CLRB
12	NOTB
13	NEGB
14	XCHB Direct
15	DECB
16	EXTB
17	INCB
18	SHRB
19	SHLB
1A	SHRAB
1B	XCHB Indexed
1C	EST Indirect
1D	EST Indexed
1E	ESTB Indirect
1F	ESTB Indexed
20–27	SJMP
28–2F	SCALL
30–37	JBC
38–3F	JBS
40	AND Direct (3 ops)
41	AND Immediate (3 ops)

Table A-7. Instruction Opcodes (Continued)

Hex Code	Instruction Mnemonic
42	AND Indirect (3 ops)
43	AND Indexed (3 ops)
44	ADD Direct (3 ops)
45	ADD Immediate (3 ops)
46	ADD Indirect (3 ops)
47	ADD Indexed (3 ops)
48	SUB Direct (3 ops)
49	SUB Immediate (3 ops)
4A	SUB Indirect (3 ops)
4B	SUB Indexed (3 ops)
4C (Note 2)	MUL, MULU Direct (3 ops)
4D (Note 2)	MUL, MULU Immediate (3 ops)
4E (Note 2)	MUL, MULU Indirect (3 ops)
4F (Note 2)	MUL, MULU Indexed (3 ops)
50	ANDB Direct (3 ops)
51	ANDB Immediate (3 ops)
52	ANDB Indirect (3 ops)
53	ANDB Indexed (3 ops)
54	ADDB Direct (3 ops)
55	ADDB Immediate (3 ops)
56	ADDB Indirect (3 ops)
57	ADDB Indexed (3 ops)
58	SUBB Direct (3 ops)
59	SUBB Immediate (3 ops)
5A	SUBB Indirect (3 ops)
5B	SUBB Indexed (3 ops)
5C	MULUB Direct (3 ops)
5D	MULUB Immediate (3 ops)
5E	MULUB Indirect (3 ops)
5F	MULUB Indexed (3 ops)
60	AND Direct (2 ops)
61	AND Immediate (2 ops)
62	AND Indirect (2 ops)
63	AND Indexed (2 ops)
64	ADD Direct (2 ops)
65	ADD Immediate (2 ops)
66	ADD Indirect (2 ops)
67	ADD Indexed (2 ops)
68	SUB Direct (2 ops)
69	SUB Immediate (2 ops)
6A	SUB Indirect (2 ops)

Table A-7. Instruction Opcodes (Continued)

Hex Code	Instruction Mnemonic
6B	SUB Indexed (2 ops)
6C (Note 2)	MUL, MULU Direct (2 ops)
6D (Note 2)	MUL, MULU Immediate(2 ops)
6E (Note 2)	MUL, MULU Indirect (2 ops)
6F (Note 2)	MUL, MULU Indexed (2 ops)
70	ANDB Direct (2 ops)
71	ANDB Immediate (2 ops)
72	ANDB Indirect (2 ops)
73	ANDB Indexed (2 ops)
74	ADDB Direct (2 ops)
75	ADDB Immediate (2 ops)
76	ADDB Indirect (2 ops)
77	ADDB Indexed (2 ops)
78	SUBB Direct (2 ops)
79	SUBB Immediate (2 ops)
7A	SUBB Indirect (2 ops)
7B	SUBB Indexed (2 ops)
7C	MULUB Direct (2 ops)
7D	MULUB Immediate (2 ops)
7E	MULUB Indirect (2 ops)
7F	MULUB Indexed (2 ops)
80	OR Direct
81	OR Immediate
82	OR Indirect
83	OR Indexed
84	XOR Direct
85	XOR Immediate
86	XOR Indirect
87	XOR Indexed
88	CMP Direct
89	CMP Immediate
8A	CMP Indirect
8B	CMP Indexed
8C	DIVU Direct
8D	DIVU Immediate
8E	DIVU Indirect
8F	DIVU Indexed
90	ORB Direct
91	ORB Immediate
92	ORB Indirect
93	ORB Indexed

Table A-7. Instruction Opcodes (Continued)

Hex Code	Instruction Mnemonic
94	XORB Direct
95	XORB Immediate
96	XORB Indirect
97	XORB Indexed
98	CMPB Direct
99	CMPB Immediate
9A	CMPB Indirect
9B	CMPB Indexed
9C	DIVUB Direct
9D	DIVUB Immediate
9E	DIVUB Indirect
9F	DIVUB Indexed
A0	LD Direct
A1	LD Immediate
A2	LD Indirect
A3	LD Indexed
A4	ADDC Direct
A5	ADDC Immediate
A6	ADDC Indirect
A7	ADDC Indexed
A8	SUBC Direct
A9	SUBC Immediate
AA	SUBC Indirect
AB	SUBC Indexed
AC	LDBZE Direct
AD	LDBZE Immediate
AE	LDBZE Indirect
AF	LDBZE Indexed
B0	LDB Direct
B1	LDB Immediate
B2	LDB Indirect
B3	LDB Indexed
B4	ADDCB Direct
B5	ADDCB Immediate
B6	ADDCB Indirect
B7	ADDCB Indexed
B8	SUBCB Direct
B9	SUBCB Immediate
BA	SUBCB Indirect
BB	SUBCB Indexed
BC	LDBSE Direct

Table A-7. Instruction Opcodes (Continued)

Hex Code	Instruction Mnemonic
BD	LDBSE Immediate
BE	LDBSE Indirect
BF	LDBSE Indexed
C0	ST Direct
C1	BMOV
C2	ST Indirect
C3	ST Indexed
C4	STB Direct
C5	CMPL
C6	STB Indirect
C7	STB Indexed
C8	PUSH Direct
C9	PUSH Immediate
CA	PUSH Indirect
CB	PUSH Indexed
CC	POP Direct
CD	BMOVI
CE	POP Indirect
CF	POP Indexed
D0	JNST
D1	JNH
D2	JGT
D3	JNC
D4	JNVT
D5	JNV
D6	JGE
D7	JNE
D8	JST
D9	JH
DA	JLE
DB	JC
DC	JVT
DD	JV
DE	JLT
DF	JE
E0	DJNZ
E1	DJNZW
E2	TIJMP
E3	BR Indirect, 64-Kbyte mode
	EBR Indirect, 2-Mbyte mode
E4	EBMOVI

Table A-7. Instruction Opcodes (Continued)

Hex Code	Instruction Mnemonic
E5	Reserved
E6	EJMP
E7	LJMP
E8	ELD Indirect
E9	ELD Indexed
EA	ELDB Indirect
EB	ELDB Indexed
EC	DPTS
ED	EPTS
EE (Note 1)	Reserved
EF	LCALL
F0	RET
F1	ECALL
F2	PUSHF
F3	POPF
F4	PUSHA
F5	POPA
F6	IDLPD
F7	TRAP
F8	CLRC
F9	SETC
FA	DI
FB	EI
FC	CLRVT
FD	NOP
FE (Note 2)	Prefix for signed multiplication and division.
FF	RST

NOTES:
1. This opcode is reserved, but it does not generate an unimplemented opcode interrupt.
2. Signed multiplication and division are two-byte instructions. For each signed instruction, the first byte is "FE" and the second is the opcode of the corresponding unsigned instruction. For example, the opcode for MULU (3 operands) direct is "4C," so the opcode for MUL (3 operands) direct is "FE 4C."

Table A-8 lists instructions along with the number of bytes and opcodes for each applicable addressing mode. A dash (—) in any column indicates “not applicable.”

Table A-8. Number of Bytes for Each Instruction and Hexadecimal Opcodes

Arithmetic (Group I)								
Mnemonic	Direct		Immediate		Indirect		Indexed (1)	
	Bytes	Opcode	Bytes	Opcode	Bytes	Opcode	Bytes S/L	Opcode
ADD (2 ops)	3	64	4	65	3	66	4/5	67
ADD (3 ops)	4	44	5	45	4	46	5/6	47
ADDB (2 ops)	3	74	3	75	3	76	4/5	77
ADDB (3 ops)	4	54	4	55	4	56	5/6	57
ADDC	3	A4	4	A5	3	A6	4/5	A7
ADDCB	3	B4	3	B5	3	B6	4/5	B7
CLR	2	01	—	—	—	—	—	—
CLRB	2	11	—	—	—	—	—	—
CMP	3	88	4	89	3	8A	4/5	8B
CMPB	3	98	3	99	3	9A	4/5	9B
CMPL	3	C5	—	—	—	—	—	—
DEC	2	05	—	—	—	—	—	—
DECB	2	15	—	—	—	—	—	—
EXT	2	06	—	—	—	—	—	—
EXTB	2	16	—	—	—	—	—	—
INC	2	07	—	—	—	—	—	—
INCB	2	17	—	—	—	—	—	—
SUB (2 ops)	3	68	4	69	3	6A	4/5	6B
SUB (3 ops)	4	48	5	49	4	4A	5/6	4B
SUBB (2 ops)	3	78	3	79	3	7A	4/5	7B
SUBB (3 ops)	4	58	4	59	4	5A	5/6	5B
SUBC	3	A8	4	A9	3	AA	4/5	AB
SUBCB	3	B8	3	B9	3	BA	4/5	BB

NOTES:

1. For indexed instructions, the first column lists instruction bytes as *S*/*L*, where *S* is the number of bytes for the short-indexed instruction and *L* is the number of bytes for the long-indexed instruction.
2. For the SCALL and SJMP instructions, the three least-significant bits of the opcode are concatenated with the eight bits to form an 11-bit, two’s complement offset.

Table A-8. Number of Bytes for Each Instruction and Hexadecimal Opcodes (Continued)

Arithmetic (Group II)								
Mnemonic	Direct		Immediate		Indirect		Indexed (1)	
	Bytes	Opcode	Bytes	Opcode	Bytes	Opcode	Bytes S/L	Opcode
DIV	4	FE 8C	5	FE 8D	4	FE 8E	5/6	FE 8F
DIVB	4	FE 9C	4	FE 9D	4	FE 9E	5/6	FE 9F
DIVU	3	8C	4	8D	3	8E	4/5	8F
DIVUB	3	9C	3	9D	3	9E	4/5	9F
MUL (2 ops)	4	FE 6C	5	FE 6D	4	FE 6E	5/6	FE 6F
MUL (3 ops)	5	FE 4C	6	FE 4D	5	FE 4E	6/7	FE 4F
MULB (2 ops)	4	FE 7C	4	FE 7D	4	FE 7E	5/6	FE 7F
MULB (3 ops)	5	FE 5C	5	FE 5D	5	FE 5E	6/7	FE 5F
MULU (2 ops)	3	6C	4	6D	3	6E	4/5	6F
MULU (3 ops)	4	4C	5	4D	4	4E	5/6	4F
MULUB (2 ops)	3	7C	3	7D	3	7E	4/5	7F
MULUB (3 ops)	4	5C	4	5D	4	5E	5/6	5F
Logical								
Mnemonic	Direct		Immediate		Indirect		Indexed (1)	
	Bytes	Opcode	Bytes	Opcode	Bytes	Opcode	Bytes S/L	Opcode
AND (2 ops)	3	60	4	61	3	62	4/5	63
AND (3 ops)	4	40	5	41	4	42	5/6	43
ANDB (2 ops)	3	70	3	71	3	72	4/5	73
ANDB (3 ops)	4	50	4	51	4	52	5/6	53
NEG	2	03	—	—	—	—	—	—
NEGB	2	13	—	—	—	—	—	—
NOT	2	02	—	—	—	—	—	—
NOTB	2	12	—	—	—	—	—	—
OR	3	80	4	81	3	82	4/5	83
ORB	3	90	3	91	3	92	4/5	93
XOR	3	84	4	85	3	86	4/5	87
XORB	3	94	3	95	3	96	4/5	97

NOTES:

1. For indexed instructions, the first column lists instruction bytes as *S*/*L*, where *S* is the number of bytes for the short-indexed instruction and *L* is the number of bytes for the long-indexed instruction.
2. For the SCALL and SJMP instructions, the three least-significant bits of the opcode are concatenated with the eight bits to form an 11-bit, two's complement offset.

Table A-8. Number of Bytes for Each Instruction and Hexadecimal Opcodes (Continued)

Stack								
Mnemonic	Direct		Immediate		Indirect		Indexed (1)	
	Bytes	Opcode	Bytes	Opcode	Bytes	Opcode	Bytes S/L	Opcode
POP	2	CC	—	—	2	CE	3/4	CF
POPA	1	F5	—	—	—	—	—	—
POPF	1	F3	—	—	—	—	—	—
PUSH	2	C8	3	C9	2	CA	3/4	CB
PUSHA	1	F4	—	—	—	—	—	—
PUSHF	1	F2	—	—	—	—	—	—
Data								
Mnemonic	Direct		Immediate		Extended-indirect		Extended-indexed	
	Bytes	Opcode	Bytes	Opcode	Bytes	Opcode	Bytes	Opcode
EBMOVI	—	—	—	—	3	E4	—	—
ELD	—	—	—	—	3	E8	6	E9
ELDB	—	—	—	—	3	EA	6	EB
EST	—	—	—	—	3	1C	6	1D
ESTB	—	—	—	—	3	1E	6	1F
Mnemonic	Direct		Immediate		Indirect		Indexed (1)	
	Bytes	Opcode	Bytes	Opcode	Bytes	Opcode	Bytes S/L	Opcode
BMOV	—	—	—	—	3	C1	—	—
BMOVI	—	—	—	—	3	CD	—	—
LD	3	A0	4	A1	3	A2	4/5	A3
LDB	3	B0	3	B1	3	B2	4/5	B3
LDBSE	3	BC	3	BD	3	BE	4/5	BF
LDBZE	3	AC	3	AD	3	AE	4/5	AF
ST	3	C0	—	—	3	C2	4/5	C3
STB	3	C4	—	—	3	C6	4/5	C7
XCH	3	04	—	—	—	—	4/5	0B
XCHB	3	14	—	—	—	—	4/5	1B

NOTES:
1. For indexed instructions, the first column lists instruction bytes as *S*/*L*, where *S* is the number of bytes for the short-indexed instruction and *L* is the number of bytes for the long-indexed instruction.
2. For the SCALL and SJMP instructions, the three least-significant bits of the opcode are concatenated with the eight bits to form an 11-bit, two's complement offset.

Table A-8. Number of Bytes for Each Instruction and Hexadecimal Opcodes (Continued)

Jump								
Mnemonic	**Direct**		**Immediate**		**Extended-indirect**		**Extended-indexed**	
	Bytes	**Opcode**	**Bytes**	**Opcode**	**Bytes**	**Opcode**	**Bytes**	**Opcode**
EBR	—	—	—	—	2	E3	—	—
EJMP	—	—	—	—	—	—	4	E6
Mnemonic	**Direct**		**Immediate**		**Indirect**		**Indexed (1)**	
	Bytes	**Opcode**	**Bytes**	**Opcode**	**Bytes**	**Opcode**	**Bytes S/L**	**Opcode**
BR	—	—	—	—	2	E3	—	—
LJMP	—	—	—	—	—	—	—/3	E7
SJMP (2)	—	—	—	—	—	—	2/—	20–27
TIJMP	—	—	—	—	—	—	—/4	E2
Call								
Mnemonic	**Direct**		**Immediate**		**Extended-indirect**		**Extended-indexed**	
	Bytes	**Opcode**	**Bytes**	**Opcode**	**Bytes**	**Opcode**	**Bytes**	**Opcode**
ECALL	—	—	—	—	—	—	4	F1
Mnemonic	**Direct**		**Immediate**		**Indirect**		**Indexed (1)**	
	Bytes	**Opcode**	**Bytes**	**Opcode**	**Bytes**	**Opcode**	**Bytes**	**Opcode**
LCALL	—	—	—	—	—	—	3	EF
RET	—	—	—	—	1	F0	—	—
SCALL (2)	—	—	—	—	—	—	2	28–2F
TRAP	1	F7	—	—	—	—	—	—

NOTES:
1. For indexed instructions, the first column lists instruction bytes as *S*/*L*, where *S* is the number of bytes for the short-indexed instruction and *L* is the number of bytes for the long-indexed instruction.
2. For the SCALL and SJMP instructions, the three least-significant bits of the opcode are concatenated with the eight bits to form an 11-bit, two's complement offset.

Table A-8. Number of Bytes for Each Instruction and Hexadecimal Opcodes (Continued)

Conditional Jump								
Mnemonic	Direct		Immediate		Indirect		Indexed (1)	
	Bytes	Opcode	Bytes	Opcode	Bytes	Opcode	Bytes S/L	Opcode
DJNZ	—	—	—	—	—	—	3/—	E0
DJNZW	—	—	—	—	—	—	3/—	E1
JBC	—	—	—	—	—	—	3/—	30–37
JBS	—	—	—	—	—	—	3/—	38–3F
JC	—	—	—	—	—	—	2/—	DB
JE	—	—	—	—	—	—	2/—	DF
JGE	—	—	—	—	—	—	2/—	D6
JGT	—	—	—	—	—	—	2/—	D2
JH	—	—	—	—	—	—	2/—	D9
JLE	—	—	—	—	—	—	2/—	DA
JLT	—	—	—	—	—	—	2/—	DE
JNC	—	—	—	—	—	—	2/—	D3
JNE	—	—	—	—	—	—	2/—	D7
JNH	—	—	—	—	—	—	2/—	D1
JNST	—	—	—	—	—	—	2/—	D0
JNV	—	—	—	—	—	—	2/—	D5
JNVT	—	—	—	—	—	—	2/—	D4
JST	—	—	—	—	—	—	2/—	D8
JV	—	—	—	—	—	—	2/—	DD
JVT	—	—	—	—	—	—	2/—	DC

NOTES:
1. For indexed instructions, the first column lists instruction bytes as *S*/*L*, where *S* is the number of bytes for the short-indexed instruction and *L* is the number of bytes for the long-indexed instruction.
2. For the SCALL and SJMP instructions, the three least-significant bits of the opcode are concatenated with the eight bits to form an 11-bit, two's complement offset.

Table A-8. Number of Bytes for Each Instruction and Hexadecimal Opcodes (Continued)

Shift								
Mnemonic	Direct		Immediate		Indirect		Indexed	
	Bytes	Opcode	Bytes	Opcode	Bytes	Opcode	Bytes	Opcode
NORML	3	0F	3	0F	—	—	—	—
SHL	3	09	3	09	—	—	—	—
SHLB	3	19	3	19	—	—	—	—
SHLL	3	0D	3	0D	—	—	—	—
SHR	3	08	3	08	—	—	—	—
SHRA	3	0A	3	0A	—	—	—	—
SHRAB	3	1A	3	1A	—	—	—	—
SHRAL	3	0E	3	0E	—	—	—	—
SHRB	3	18	3	18	—	—	—	—
SHRL	3	0C	3	0C	—	—	—	—
Special								
Mnemonic	Direct		Immediate		Indirect		Indexed	
	Bytes	Opcode	Bytes	Opcode	Bytes	Opcode	Bytes	Opcode
CLRC	1	F8	—	—	—	—	—	—
CLRVT	1	FC	—	—	—	—	—	—
DI	1	FA	—	—	—	—	—	—
EI	1	FB	—	—	—	—	—	—
IDLPD	—	—	1	F6	—	—	—	—
NOP	1	FD	—	—	—	—	—	—
RST	1	FF	—	—	—	—	—	—
SETC	1	F9	—	—	—	—	—	—
SKIP	2	00	—	—	—	—	—	—
PTS								
Mnemonic	Direct		Immediate		Indirect		Indexed	
	Bytes	Opcode	Bytes	Opcode	Bytes	Opcode	Bytes	Opcode
DPTS	1	EC	—	—	—	—	—	—
EPTS	1	ED	—	—	—	—	—	—

NOTES:
1. For indexed instructions, the first column lists instruction bytes as *S*/*L*, where *S* is the number of bytes for the short-indexed instruction and *L* is the number of bytes for the long-indexed instruction.
2. For the SCALL and SJMP instructions, the three least-significant bits of the opcode are concatenated with the eight bits to form an 11-bit, two's complement offset.

Table A-9 lists instructions alphabetically within groups, along with their execution times, expressed in state times.

Table A-9. Instruction Execution Times (in State Times)

Arithmetic (Group I)										
Mnemonic	Direct	Immed.	Indirect				Indexed			
			Normal		Autoinc.		Short		Long	
			Reg.	Mem.	Reg.	Mem.	Reg.	Mem.	Reg.	Mem.
ADD (2 ops)	4	5	6	8	7	9	6	8	7	9
ADD (3 ops)	5	6	7	10	8	11	7	10	8	11
ADDB (2 ops)	4	4	6	8	7	9	6	8	7	9
ADDB (3 ops)	5	5	7	10	8	11	7	10	8	11
ADDC	4	5	6	8	7	9	6	8	7	9
ADDCB	4	4	6	8	7	9	6	8	7	9
CLR	3	—	—	—	—	—	—	—	—	—
CLRB	3	—	—	—	—	—	—	—	—	—
CMP	4	5	6	8	7	9	6	8	7	9
CMPB	4	4	6	8	7	9	6	8	7	9
CMPL	7	—	—	—	—	—	—	—	—	—
DEC	3	—	—	—	—	—	—	—	—	—
DECB	3	—	—	—	—	—	—	—	—	—
EXT	4	—	—	—	—	—	—	—	—	—
EXTB	4	—	—	—	—	—	—	—	—	—
INC	3	—	—	—	—	—	—	—	—	—
INCB	3	—	—	—	—	—	—	—	—	—
SUB (2 ops)	4	5	6	8	7	9	6	8	7	9
SUB (3 ops)	5	6	7	10	8	11	7	10	8	11
SUBB (2 ops)	4	4	6	8	7	9	6	8	7	9
SUBB (3 ops)	5	5	7	10	8	11	7	10	8	11
SUBC	4	5	6	8	7	9	6	8	7	9
SUBCB	4	4	6	8	7	9	6	8	7	9

NOTE: The column entitled “Reg.” lists the instruction execution times for accesses to the register file or peripheral SFRs. The column entitled “Mem.” lists the instruction execution times for accesses to all memory-mapped registers, I/O, or memory.

Table A-9. Instruction Execution Times (in State Times) (Continued)

Arithmetic (Group II)										
Mnemonic	Direct	Immed.	Indirect				Indexed			
			Normal		Autoinc.		Short		Long	
			Reg.	Mem.	Reg.	Mem.	Reg.	Mem.	Reg.	Mem.
DIV	26	27	28	31	29	32	29	32	30	33
DIVB	18	18	20	23	21	24	21	24	22	25
DIVU	24	25	26	29	27	30	27	30	28	31
DIVUB	16	16	18	21	19	22	19	22	20	23
MUL (2 ops)	16	17	18	21	19	22	19	22	20	23
MUL (3 ops)	16	17	18	21	19	22	19	22	20	23
MULB (2 ops)	12	12	14	17	15	18	15	18	16	19
MULB (3 ops)	12	12	14	17	15	18	15	18	16	19
MULU (2 ops)	14	15	16	19	17	19	17	20	18	21
MULU (3 ops)	14	15	16	19	17	19	17	20	18	21
MULUB (2 ops)	10	10	12	15	13	15	12	16	14	17
MULUB (3 ops)	10	10	12	15	13	15	12	16	14	17

NOTE: The column entitled "Reg." lists the instruction execution times for accesses to the register file or peripheral SFRs. The column entitled "Mem." lists the instruction execution times for accesses to all memory-mapped registers, I/O, or memory. See Table 4-2 on page 4-8 for address information.

Table A-9. Instruction Execution Times (in State Times) (Continued)

Logical										
Mnemonic	Direct	Immed.	Indirect				Indexed			
			Normal		Autoinc.		Short		Long	
			Reg.	Mem.	Reg.	Mem.	Reg.	Mem.	Reg.	Mem.
AND (2 ops)	4	5	6	8	7	9	6	8	7	9
AND (3 ops)	5	6	7	10	8	11	7	10	8	11
ANDB (2 ops)	4	4	6	8	7	9	6	8	7	9
ANDB (3 ops)	5	5	7	10	8	11	7	10	8	11
NEG	3	—	—	—	—	—	—	—	—	—
NEGB	3	—	—	—	—	—	—	—	—	—
NOT	3	—	—	—	—	—	—	—	—	—
NOTB	3	—	—	—	—	—	—	—	—	—
OR	4	5	6	8	7	9	6	8	7	9
ORB	4	4	6	8	7	9	6	8	7	9
XOR	4	5	6	8	7	9	6	8	7	9
XORB	4	4	6	8	7	9	6	8	7	9
Stack (Register)										
Mnemonic	Direct	Immed.	Indirect				Indexed			
			Normal		Autoinc.		Short		Long	
			Reg.	Mem.	Reg.	Mem.	Reg.	Mem.	Reg.	Mem.
POP	8	—	10	12	11	13	11	13	12	14
POPA	12	—	—	—	—	—	—	—	—	—
POPF	7	—	—	—	—	—	—	—	—	—
PUSH	6	7	9	12	10	13	10	13	11	14
PUSHA	12	—	—	—	—	—	—	—	—	—
PUSHF	6	—	—	—	—	—	—	—	—	—

NOTE: The column entitled “Reg.” lists the instruction execution times for accesses to the register file or peripheral SFRs. The column entitled “Mem.” lists the instruction execution times for accesses to all memory-mapped registers, I/O, or memory. See Table 4-2 on page 4-8 for address information.

Table A-9. Instruction Execution Times (in State Times) (Continued)

Stack (Memory)										
Mnemonic	Direct	Immed.	Indirect				Indexed			
			Normal		Autoinc.		Short		Long	
			Reg.	Mem.	Reg.	Mem.	Reg.	Mem.	Reg.	Mem.
POP	11	—	13	15	14	16	14	16	15	17
POPA	18	—	—	—	—	—	—	—	—	—
POPF	10	—	—	—	—	—	—	—	—	—
PUSH	8	9	11	14	12	15	12	15	13	16
PUSHA	18	—	—	—	—	—	—	—	—	—
PUSHF	8	—	—	—	—	—	—	—	—	—

NOTE: The column entitled "Reg." lists the instruction execution times for accesses to the register file or peripheral SFRs. The column entitled "Mem." lists the instruction execution times for accesses to all memory-mapped registers, I/O, or memory. See Table 4-2 on page 4-8 for address information.

Table A-9. Instruction Execution Times (in State Times) (Continued)

Data										
Mnemonic	**Extended-indirect (Normal)**									
EBMOVI	register/register 8 + 14 per word + 16 per interrupt memory/register 8 + 17 per word + 16 per interrupt memory/memory 8 + 20 per word + 16 per interrupt									
Mnemonic	**Indirect**									
BMOV	register/register 6 + 8 per word memory/register 6 + 11 per word memory/memory 6 + 14 per word									
BMOVI	register/register 6 + 8 per word + 14 per interrupt memory/register 6 + 11 per word + 14 per interrupt memory/memory 6 + 14 per word + 14 per interrupt									
Mnemonic	**Direct**	**Immed.**	**Extended-indirect**				**Extended-indexed**			
			Normal		**Autoinc.**					
ELD	—	—	6	9	8	11	8		11	
ELDB	—	—	6	9	8	11	8		11	
EST	—	—	6	9	8	11	8		11	
ESTB	—	—	6	9	8	11	8		11	
Mnemonic	**Direct**	**Immed.**	**Indirect**				**Indexed**			
			Normal		**Autoinc.**		**Short**		**Long**	
			Reg.	**Mem.**	**Reg.**	**Mem.**	**Reg.**	**Mem.**	**Reg.**	**Mem.**
LD	4	4	5	8	6	8	6	9	7	10
LDB	4	4	5	8	6	8	6	9	7	10
LDBSE	4	4	5	8	6	8	6	9	7	10
LDBZE	4	4	5	8	6	8	6	9	7	10
ST	4	—	5	8	6	9	6	9	7	10
STB	4	—	5	8	6	8	6	9	7	10
XCH	5	—	—	—	—	—	8	13	9	14
XCHB	5	—	—	—	—	—	8	13	9	14

NOTE: The column entitled “Reg.” lists the instruction execution times for accesses to the register file or peripheral SFRs. The column entitled “Mem.” lists the instruction execution times for accesses to all memory-mapped registers, I/O, or memory. See Table 4-2 on page 4-8 for address information.

Table A-9. Instruction Execution Times (in State Times) (Continued)

Jump						
Mnemonic	**Direct**	**Immed.**	**Extended-indirect**		**Extended-indexed**	
			Normal	**Autoinc.**		
EBR	—	—	9	—	—	
EJMP	—	—	—	—	8	
Mnemonic	**Direct**	**Immed.**	**Indirect**		**Indexed**	
			Normal	**Autoinc.**	**Short**	**Long**
BR	—	—			—	—
LJMP	—	—	—	—	—	7
SJMP	—	—	—	—	7	—
TIJMP register/register memory/register memory/memory	—	—	—	—	—	15 18 21 —
Call (Register)						
Mnemonic	**Direct**	**Immed.**	**Extended-indirect**		**Extended-indexed**	
			Normal	**Autoinc.**		
ECALL2-Mbyte mode	—	—	—	—	16	
Mnemonic	**Direct**	**Immed.**	**Indirect**		**Indexed**	
			Normal	**Autoinc.**	**Short**	**Long**
LCALL2-Mbyte mode64-Kbyte mode	—	—	—	—	—	15 11
RET2-Mbyte mode 64-Kbyte mode	—	—	16 11	—	—	—
SCALL 2-Mbyte mode 64-Kbyte mode	—	—	—	—	—	15 11
TRAP 2-Mbyte mode 64-Kbyte mode	19 16	—	—	—	—	—

Table A-9. Instruction Execution Times (in State Times) (Continued)

Call (Memory)						
Mnemonic	**Direct**	**Immed.**	**Extended-indirect**		**Extended-indexed**	
			Normal	**Autoinc.**		
ECALL 2-Mbyte mode	—	—	—	—	22	
Mnemonic	**Direct**	**Immed.**	**Indirect**		**Indexed**	
			Normal	**Autoinc.**	**Short**	**Long**
LCALL 2-Mbyte mode 64-Kbyte mode	—	—	—	—	—	18 13
RET 2-Mbyte mode 64-Kbyte mode	—	—	22 14	—	—	—
SCALL 2-Mbyte mode 64-Kbyte mode	—	—	—	—	—	18 13
TRAP 2-Mbyte mode 64-Kbyte mode	25 18	—	—	—	—	—

Conditional Jump	
Mnemonic	**Short-Indexed**
DJNZ	5 (jump not taken), 9 (jump taken)
DJNZW	6 (jump not taken), 10 (jump taken)
JBC	5 (jump not taken), 9 (jump taken)
JBS	5 (jump not taken), 9 (jump taken)
JC	4 (jump not taken), 8 (jump taken)
JE	4 (jump not taken), 8 (jump taken)
JGE	4 (jump not taken), 8 (jump taken)
JGT	4 (jump not taken), 8 (jump taken)
JH	4 (jump not taken), 8 (jump taken)
JLE	4 (jump not taken), 8 (jump taken)
JLT	4 (jump not taken), 8 (jump taken)
JNC	4 (jump not taken), 8 (jump taken)
JNE	4 (jump not taken), 8 (jump taken)
JNH	4 (jump not taken), 8 (jump taken)
JNST	4 (jump not taken), 8 (jump taken)
JNV	4 (jump not taken), 8 (jump taken)
JNVT	4 (jump not taken), 8 (jump taken)
JST	4 (jump not taken), 8 (jump taken)
JV	4 (jump not taken), 8 (jump taken)
JVT	4 (jump not taken), 8 (jump taken)

Table A-9. Instruction Execution Times (in State Times) (Continued)

Shift						
Mnemonic	**Direct**					
NORML	8 + 1 per shift (9 for 0 shift)					
SHL	6 + 1 per shift (7 for 0 shift)					
SHLB	6 + 1 per shift (7 for 0 shift)					
SHLL	7 + 1 per shift (8 for 0 shift)					
SHR	6 + 1 per shift (7 for 0 shift)					
SHRA	6 + 1 per shift (7 for 0 shift)					
SHRAB	6 + 1 per shift (7 for 0 shift)					
SHRAL	7 + 1 per shift (8 for 0 shift)					
SHRB	6 + 1 per shift (7 for 0 shift)					
SHRL	7 + 1 per shift (8 for 0 shift)					
Special						
Mnemonic	**Direct**	**Immed.**	**Indirect**		**Indexed**	
			Normal	**Autoinc.**	**Short**	**Long**
CLRC	2	—	—	—	—	—
CLRVT	2	—	—	—	—	—
DI	2	—	—	—	—	—
EI	2	—	—	—	—	—
IDLPD Valid key Invalid key	 — —	 12 28	 — —	 — —	 — —	 — —
NOP	2	—	—	—	—	—
RST	4	—	—	—	—	—
SETC	2	—	—	—	—	—
SKIP	3	—	—	—	—	—
PTS						
Mnemonic	**Direct**	**Immed.**	**Indirect**		**Indexed**	
			Normal	**Autoinc.**	**Short**	**Long**
DPTS	2	—	—	—	—	—
EPTS	2	—	—	—	—	—

Table A-10. Jump Penalty (in State Times)

Instruction Length (bytes)	Number of Reads to Complete Instruction		Penalty for Jump to Code RAM		Penalty for Jump to External Memory			
	Even Address	Odd Address	Even Address	Odd Address	Even Address		Odd Address	
					Bus Idle	Bus Busy	Bus Idle	Bus Busy
1	1	1	4	4	4	5	4	5
2	1	2	4	5	4	5	6	7
3	2	2	5	5	6	7	6	7
4	2	3	5	6	6	7	8	9
5	3	3	6	6	8	9	8	9
6	3	4	6	7	8	9	10	11

B

Signal Descriptions

APPENDIX B
SIGNAL DESCRIPTIONS

This appendix provides reference information for the pin functions of the 8XC196EA.

B.1 FUNCTIONAL GROUPINGS OF SIGNALS

Table B-1 lists the signals for the 8XC196EA, grouped by function. A diagram of each package that is currently available shows the pin location of each signal.

NOTE

The datasheets are revised more frequently than this manual. As new packages are supported, the pin-out diagrams will be added to the datasheets first. If your package type is not shown in this appendix, refer to the latest datasheet to find the pin locations.

Table B-1. 8XC196EA Signals Arranged by Functions

Address & Data
Name
A20:0
AD15:0

Power & Ground
Name
ANGND
V_{CC}
V_{SS}
V_{REF}

Processor Control
Name
CLKOUT
EA#
EXTINT
NMI
ONCE#
PLLEN
RESET#
RPD
TMODE#
XTAL1
XTAL2

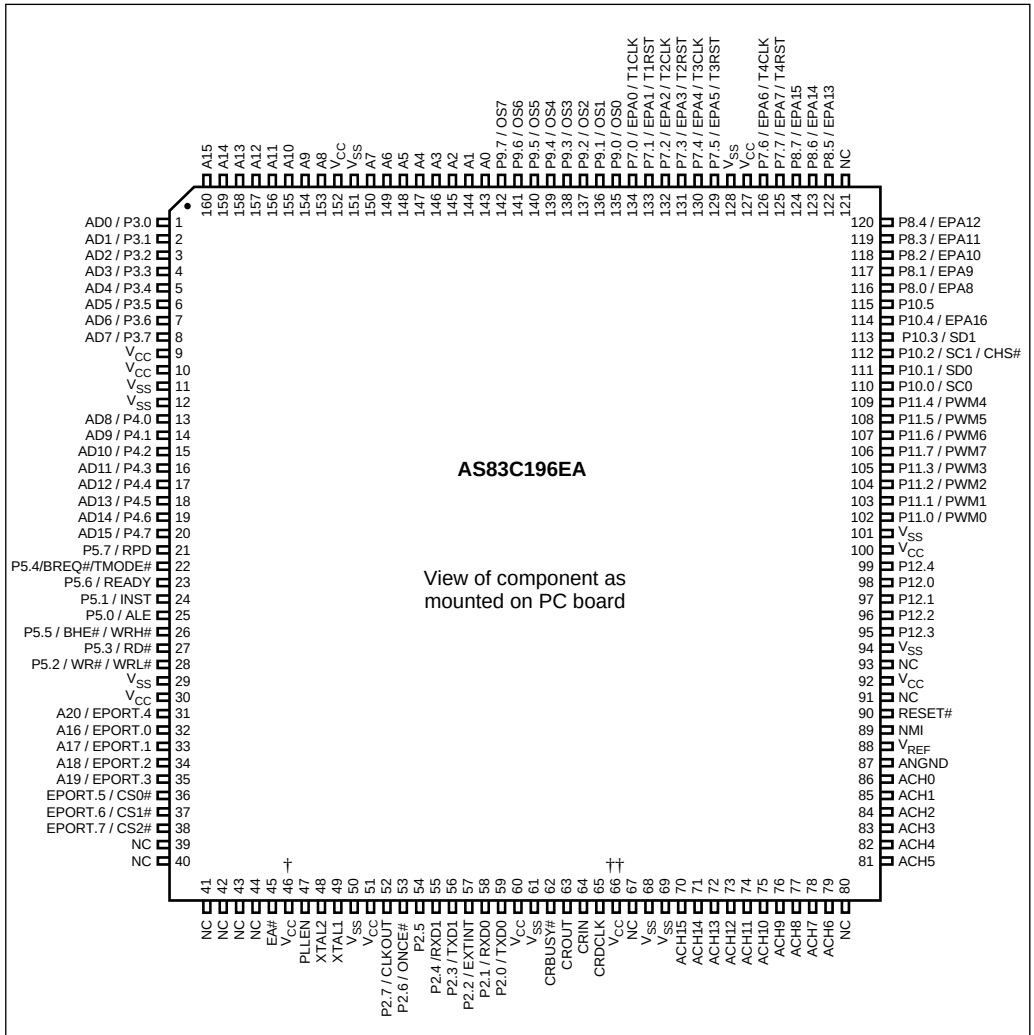
Input/Output
Name
P2.0 / TXD0
P2.1 / RXD0
P2.2
P2.3 / TXD1
P2.4 / RXD1
P2.7:5
P3.7:0
P4.7:0
P5.7:0
EPORT.7:0
P7.0 / EPA0 / T1CLK
P7.1 / EPA1 / T1RST
P7.2 / EPA2 / T2CLK
P7.3 / EPA3 / T2RST
P7.4 / EPA4 / T3CLK
P7.5 / EPA5 / T3RST
P7.6 / EPA6 / T4CLK
P7.7 / EPA7 / T4RST
P8.7:0 / EPA15:8
P9.7:0 / OS7:0
P10.0 / SC0
P10.1 / SD0
P10.2 / SC1 / CHS#

Input/Output (Cont'd)
Name
P10.3 / SD1
P10.4 / EPA16
P10.5
P11.7:0 / PWM7:0
P12.4:0

Bus Control & Status
Name
ALE
BHE# / WRH#
BREQ#
CS2:0#
HLDA#
HOLD#
INST
RD#
READY
WR# / WRL#

Serial Debug Unit
Name
CRBUSY#
CRDCLK
CRIN
CROUT

Analog Inputs
Name
ACH15:0



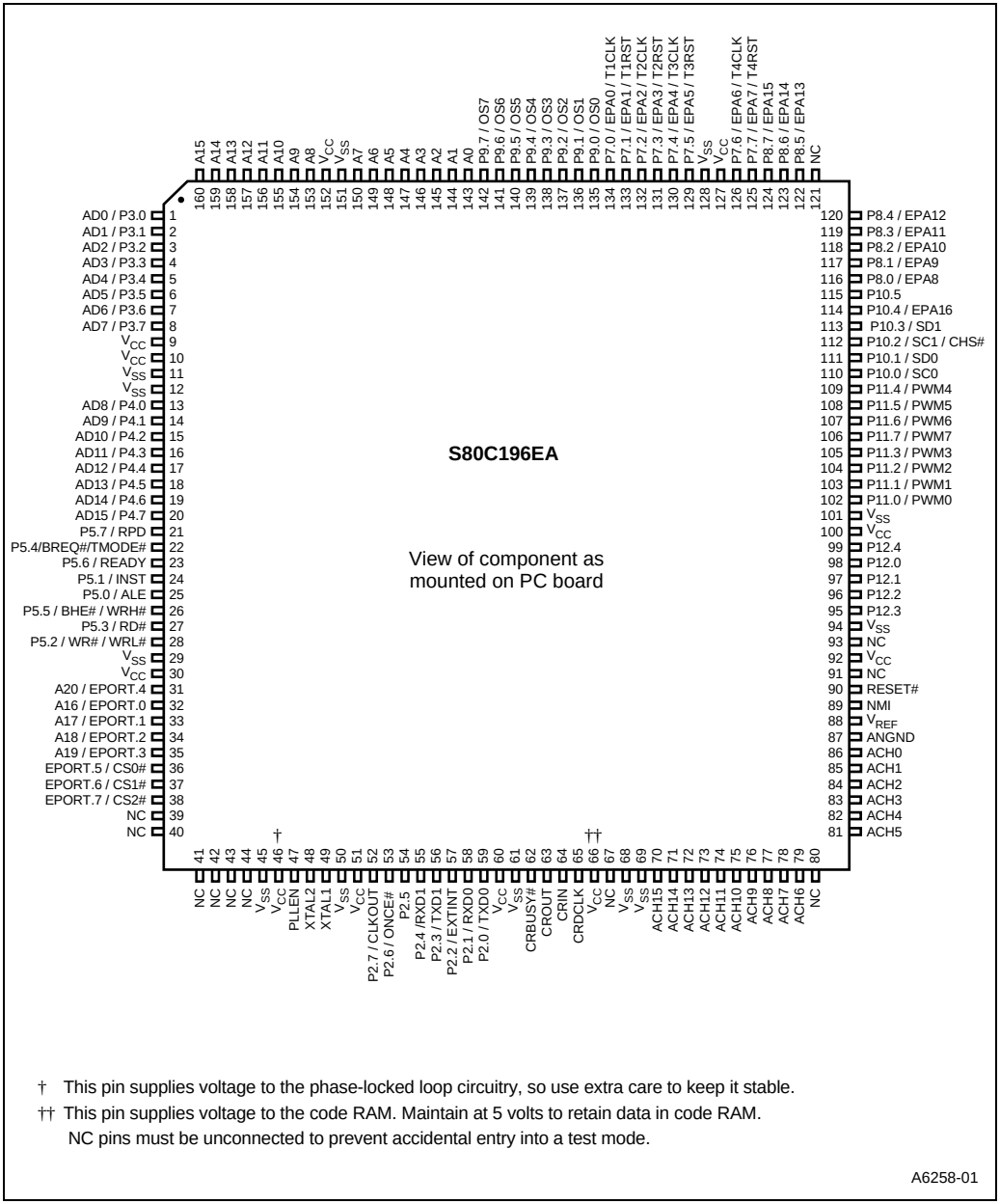
† This pin supplies voltage to the phase-locked loop circuitry, so use extra care to keep it stable.

†† This pin supplies voltage to the code RAM. Maintain at 5 volts to retain data in code RAM.

NC pins must be unconnected to prevent accidental entry into a test mode.

A3151-02

Figure B-1. 83C196EA 160-pin QFP Package



† This pin supplies voltage to the phase-locked loop circuitry, so use extra care to keep it stable.

†† This pin supplies voltage to the code RAM. Maintain at 5 volts to retain data in code RAM.

NC pins must be unconnected to prevent accidental entry into a test mode.

A6258-01

Figure B-2. 80C196EA 160-pin QFP Package

B.2 SIGNAL DESCRIPTIONS

Table B-2 defines the columns used in Table B-3, which describes the signals.

Table B-2. Description of Columns of Table B-3

Column Heading	Description
Name	Lists the signals, arranged alphabetically. Many pins have two functions, so there are more entries in this column than there are pins. Every signal is listed in this column.
Type	Identifies the pin function listed in the *Name* column as an input (I), output (O), bidirectional (I/O), power (PWR), or ground (GND). Note that all inputs except RESET# are *sampled inputs*. RESET# is a level-sensitive input. During powerdown mode, the powerdown circuitry uses EXTINT as a level-sensitive input.
Description	Briefly describes the function of the pin for the specific signal listed in the *Name* column. Also lists any alternate functions that share package pins with the signal.

Table B-3. Signal Descriptions

Name	Type	Description
A15:0	O	System Address Bus These address pins provide address bits 0–15 during the entire external memory cycle during both multiplexed and demultiplexed bus modes.
A20:16	I/O	Address Pins 16–20 These address pins provide address bits 16–20 during the entire external memory cycle during both multiplexed and demultiplexed bus modes, supporting extended addressing of the 2-Mbyte address space. **NOTE:** Internally, there are 24 address bits; however, only 21 external address pins (A20:0) are implemented. The internal address space is 16 Mbytes (000000–FFFFFFH) and the external address space is 2 Mbytes (000000–1FFFFFH). The microcontroller resets to FF2080H in internal memory or 1F2080H in external memory. A20:16 share package pins with EPORT.4:0.
ACH15:0	I	Analog Channels These signals are analog inputs to the A/D converter. The ANGND and V_{REF} pins must be connected for the A/D converter to function.

Table B-3. Signal Descriptions (Continued)

Name	Type	Description
AD15:0	I/O	Address/Data Lines The function of these pins depends on the bus width and mode. When a bus access is not occurring, these pins revert to their I/O port function. **16-bit Multiplexed Bus Mode**: AD15:0 drive address bits 0–15 during the first half of the bus cycle and drive or receive data during the second half of the bus cycle. **8-bit Multiplexed Bus Mode**: AD15:8 drive address bits 8–15 during the entire bus cycle. AD7:0 drive address bits 0–7 during the first half of the bus cycle and drive or receive data during the second half of the bus cycle. **16-bit Demultiplexed Mode**: AD15:0 drive or receive data during the entire bus cycle. **8-bit Demultiplexed Mode**: AD7:0 drive or receive data during the entire bus cycle. AD15:8 drive the data that is currently on the high byte of the internal bus. AD15:8 share package pins with P4.7:0. AD7:0 share package pins with P3.7:0.
ALE	O	Address Latch Enable This active-high output signal is asserted only during external memory cycles. ALE signals the start of an external bus cycle and indicates that valid address information is available on the system address/data bus (A20:16 and AD15:0 for a multiplexed bus; A20:0 for a demultiplexed bus). An external latch can use this signal to demultiplex address bits 0–15 from the address/data bus in multiplexed mode. ALE shares a package pin with P5.0.
ANGND	GND	Analog Ground ANGND must be connected for A/D converter operation. ANGND and V_{SS} should be nominally at the same potential.
BHE#	O	Byte High Enable† During 16-bit bus cycles, this active-low output signal is asserted for word and high-byte reads and writes to external memory. BHE# indicates that valid data is being transferred over the upper half of the system data bus. Use BHE#, in conjunction with address bit 0 (A0 for a demultiplexed address bus, AD0 for a multiplexed address/data bus), to determine which memory byte is being transferred over the system bus: **BHE# — AD0 or A0 — Byte(s) Accessed** 0 — 0 — both bytes 0 — 1 — high byte only 1 — 0 — low byte only BHE# shares a package pin with P5.5 and WRH#. † When this pin is configured as a special-function signal (P5_MODE.5 = 1), the chip configuration register 0 (CCR0) determines whether it functions as BHE# or WRH#. CCR0.2 = 1 selects BHE#; CCR0.2 = 0 selects WRH#.

Table B-3. Signal Descriptions (Continued)

Name	Type	Description
BREQ#	O	Bus Request This active-low output signal is asserted during a hold cycle when the bus controller has a pending external memory cycle. When the bus-hold protocol is enabled (WSR.7 is set), the P5.4/BREQ# pin can function only as BREQ#, regardless of the configuration selected through the port configuration registers (P5_MODE, P5_DIR, and P5_REG). An attempt to change the pin configuration is ignored until the bus-hold protocol is disabled (WSR.7 is cleared). The microcontroller can assert BREQ# at the same time as or after it asserts HLDA#. Once it is asserted, BREQ# remains asserted until HOLD# is deasserted. BREQ# shares a package pin with P5.4 and TMODE#.
CHS#	I	Channel Select This signal is available only when the SSIO is configured for channel-select operation. The function of the signal depends on whether the SSIO is configured as master or slave. When the SSIO is configured as a slave, an external master activates CHS# to communicate with the SSIO. When the SSIO is configured as a master, an external master activates CHS# when it wants the SSIO to give up the bus. CHS# shares a package pin with P10.2.
CLKOUT	O	Clock Output Output of the internal clock generator. You can select one of five frequencies: f, f/2, f/4, f/8, or f/16. CLKOUT has a 50% duty cycle. CLKOUT shares a package pin with P2.7
CRBUSY#	O	Code RAM Busy When active, this signal indicates that the serial debug unit (SDU) is busy processing a code RAM command. No data can be transferred during this time.
CRDCLK	I	Code RAM Clock Provides the clock signal for the serial debug unit (SDU). The maximum clock frequency equals one-half the operating frequency (f/2).
CRIN	I	Code RAM Data Input Serial input for test instructions and data into the serial debug unit (SDU). Data is transferred in 8-bit bytes with the most-significant bit (MSB) first. Each byte is sampled on the rising edge of CRDCLK.
CROUT	O	Code RAM Data Output Serial output for data from the serial debug unit (SDU). Data is transferred in 8-bit bytes with the most-significant bit (MSB) first. Each byte is valid on the rising edge of CRDCLK.

Table B-3. Signal Descriptions (Continued)

Name	Type	Description
CS2:0#	O	Chip-select Lines 0–2 The active-low output CS*x*# is asserted during an external memory cycle when the address to be accessed is in the range programmed for chip select *x*. If the external memory address is outside the range assigned to the three chip selects, no chip-select output is asserted and the bus configuration defaults to the CS2# values. Immediately following reset, CS0# is automatically assigned to the range FF2000–FF20FFH (1F2000–1F20FFH if external). CS2:0# share package pins with EPORT.7:5.
EA#	I	External Access This input determines whether memory accesses to the upper 7 Kbytes of ROM (FF2400–FF3FFFH) are directed to internal or external memory. These accesses are directed to internal memory if EA# is held high and to external memory if EA# is held low. For an access to any other memory location, the value of EA# is irrelevant. EA# is sampled and latched only on the rising edge of RESET#. Changing the level of EA# after reset has no effect.
EPA16:0	I/O	Event Processor Array (EPA) Capture/Compare Channels High-speed input/output signals for the EPA capture/compare channels. EPA16:0 share package pins with the following signals: EPA0/P7.0/T1CLK, EPA1/P7.1/T1RST, EPA2/P7.2/T2CLK, EPA3/P7.3/T2RST, EPA4/P7.4/T3CLK, EPA5/P7.5/T3RST, EPA6/P7.6/T4CLK, EPA7/P7.7/T4RST, EPA8/P8.0, EPA9/P8.1, EPA10/P8.2, EPA11/P8.3, EPA12/P8.4, EPA13/P8.5, EPA14/P8.6, EPA15/P8.7, and EPA16/P10.4.
EPORT.7:0	I/O	Extended Addressing Port This is a standard 8-bit, bidirectional port. EPORT.4:0 share package pins with A20:16. EPORT7:5 share package pins with CS2:0#.
EXTINT	I	External Interrupt In normal operating mode, a rising edge on EXTINT sets the EXTINT interrupt pending bit. EXTINT is sampled during phase 2 (CLKOUT high). The minimum high time is one state time. In powerdown mode, asserting the EXTINT signal for at least 50 ns causes the device to resume normal operation. The interrupt does not need to be enabled, but the pin must be configured as a special-function input. If the EXTINT interrupt is enabled, the CPU executes the interrupt service routine. Otherwise, the CPU executes the instruction that immediately follows the command that invoked the power-saving mode. In idle mode, asserting any enabled interrupt causes the device to resume normal operation. EXTINT shares a package pin with P2.2.

Table B-3. Signal Descriptions (Continued)

Name	Type	Description
HLDA#	O	Bus Hold Acknowledge This active-low output indicates that the CPU has released the bus as the result of an external device asserting HOLD#. When the bus-hold protocol is enabled (WSR.7 is set), the P2.6/HLDA# pin can function only as HLDA#, regardless of the configuration selected through the port configuration registers (P2_MODE, P2_DIR, and P2_REG). An attempt to change the pin configuration is ignored until the bus-hold protocol is disabled (WSR.7 is cleared). HLDA# shares a package pin with P2.6 and ONCE#.
HOLD#	I	Bus Hold Request An external device uses this active-low input signal to request control of the bus. When the bus-hold protocol is enabled (WSR.7 is set), the P2.5/HOLD# pin can function only as HOLD#, regardless of the configuration selected through the port configuration registers (P2_MODE, P2_DIR, and P2_REG). An attempt to change the pin configuration is ignored until the bus-hold protocol is disabled (WSR.7 is cleared). HOLD# shares a package pin with P2.5.
INST	O	Instruction Fetch When high, INST indicates that an instruction is being fetched from external memory. The signal remains high during the entire bus cycle of an external instruction fetch. INST is low for data accesses, including interrupt vector fetches and chip configuration byte reads. INST is low during internal memory fetches. INST shares a package pin with P5.1.
NMI	I	Nonmaskable Interrupt In normal operating mode, a rising edge on NMI generates a nonmaskable interrupt. NMI has the highest priority of all prioritized interrupts. Assert NMI for greater than one state time to guarantee that it is recognized.
ONCE#	I	On-circuit Emulation Holding ONCE# low during the rising edge of RESET# places the device into into on-circuit emulation (ONCE) mode. (The PLLEN pin must also be held low.) This mode puts all pins into a high-impedance state, thereby isolating the microcontroller from other components in the system. The value of ONCE# is latched when the RESET# pin goes inactive. While the microcontroller is in ONCE mode, you can debug the system using a clip-on emulator. To exit ONCE mode, reset the microcontroller by pulling the RESET# signal low. To prevent inadvertent entry into ONCE mode, either configure this pin as an output or hold it high during reset and ensure that your system meets the V_{IH} specification. ONCE# shares a package pin with P2.6 and HLDA#.
OS7:0	O	Event Processor Array (EPA) Compare-only Channels with Simulcapture Outputs of the EPA's compare-only channels. These pins are shared with port 9 and may be configured as general-purpose I/O signals. OS7:0 share package pins with P9.7:0.

Table B-3. Signal Descriptions (Continued)

Name	Type	Description
P2.7:0	I/O	Port 2 This is a standard, 8-bit, bidirectional port that shares package pins with individually selectable special-function signals. P2.6 is multiplexed with the ONCE function. If you choose to configure this pin as an input, always hold it during reset and ensure that your system meets the V_{IH} specification to prevent inadvertent entry into ONCE mode. Port 2 shares package pins with the following signals: P2.0/TXD0, P2.1/RXD0, P2.2/EXTINT, P2.3/TXD1, P2.4/RXD1, P2.5/HOLD#, P2.6/HLDA#/ONCE#/, and P2.7/CLKOUT.
P3.7:0	I/O	Port 3 This is a memory-mapped, 8-bit, bidirectional port with programmable open-drain or complementary output modes. The pins are shared with the multiplexed address/data bus, which has complementary drivers. P3.7:0 share package pins with AD7:0.
P4.7:0	I/O	Port 4 This is a memory-mapped, 8-bit, bidirectional port with programmable open-drain or complementary output modes. The pins are shared with the multiplexed address/data bus, which has complementary drivers. P4.7:0 share package pins with AD15:8.
P5.7:0	I/O	Port 5 This is a memory-mapped, 8-bit, bidirectional port that shares package pins with individually selectable control signals. P5.4 shares a package pin with TMODE#. If this pin is held low during reset, the device will enter a test mode. To prevent inadvertent entry into a reserved test mode, either configure this pin as an output or hold it high during reset and ensure that your system meets the V_{IH} specification. (see datasheet) Port 5 shares package pins with the following signals: P5.0/ALE, P5.1/INST, P5.2/WR#/WRL#, P5.3/RD#, P5.4/BREQ#/TMODE#, P5.5/BHE#/WRH#, P5.6/READY, and P5.7/RPD.
P7.7:0	I/O	Port 7 This is a standard, 8-bit, bidirectional port that shares package pins with individually selectable special-function signals. Port 7 shares package pins as follows: P7.0/EPA0/T1CLK, P7.1/EPA1/T1RST, P7.2/EPA2/T2CLK, P7.3/EPA3/T2RST, P7.4/EPA4/T3CLK, P7.5/EPA5/T3RST, P7.6/EPA6/T4CLK, and P7.7/EPA7/T4RST.
P8.7:0	I/O	Port 8 This is a standard, 8-bit, bidirectional port that shares package pins with individually selectable special-function signals. P8.7:0 share package pins with EPA15:8.
P9.7:0	I/O	Port 9 This is a standard, 8-bit, bidirectional port that shares package pins with individually selectable special-function signals. P9.7:0 share package pins with OS7:0.

Table B-3. Signal Descriptions (Continued)

Name	Type	Description
P10.5:0	I/O	Port 10 This is a standard, 6-bit, bidirectional port that is multiplexed with individually selectable special-function signals. Port 10 shares package pins as follows: P10.0/SC0, P10.1/SD0, P10.2/SC1, P10.3/SD1, P10.4/EPA16, and P10.5.
P11.7:0	I/O	Port 11 This is a standard, 8-bit, bidirectional port that is multiplexed with individually selectable special-function signals. P11.7:0 share package pins with PWM7:0.
P12.4:0	I/O	Port 12 This is a memory-mapped, 5-bit, bidirectional port. P12.2:0 select the TROM execution mode.
PLLEN	I	Phase-locked Loop Enable This active-high input pin enables the on-chip clock multiplier. The PLLEN pin must be held low along with the ONCE# pin to enter on-circuit emulation (ONCE) mode.
PWM7:0	O	Pulse Width Modulator Outputs These are PWM output pins with high-current drive capability. PWM7:0 share package pins with P11.7:0.
RD#	O	Read Read-signal output to external memory. RD# is asserted only during external memory reads. RD# shares a package pin with P5.3.
READY	I	Ready Input This active-high input can be used to insert wait states in addition to those programmed in the chip configuration byte 0 (CCB0) and the bus control *x* register (BUSCON*x*). CCB0 is programmed with the minimum number of wait states (0–3) for an external fetch of CCB1, and BUSCON*x* is programmed with the minimum number of wait states (0–3) for all external accesses to the address range assigned to the chip-select *x* channel. If READY is low when the programmed number of wait states is reached, additional wait states are added until READY is pulled high. READY shares a package pin with P5.6.
RESET#	I/O	Reset A level-sensitive reset input to, and an open-drain system reset output from, the microcontroller. Either a falling edge on RESET# or an internal reset turns on a pull-down transistor connected to the RESET# pin for 16 state times. In the powerdown and idle modes, asserting RESET# causes the microcontroller to reset and return to normal operating mode. After a reset, the first instruction fetch is from FF2080H (or 1F2080H in external memory).

Table B-3. Signal Descriptions (Continued)

Name	Type	Description
RPD	I	Return from Powerdown Timing pin for the return-from-powerdown circuit. If your application uses powerdown mode, connect a capacitor between RPD and V_{SS} if **either** of the following conditions are true. • the internal oscillator is the clock source • the phase-locked loop (PLL) circuitry is enabled (see PLLEN signal description) The capacitor causes a delay that enables the oscillator and PLL circuitry to stabilize before the internal CPU and peripheral clocks are enabled. Refer to the "Special Operating Modes" chapter of the for details on selecting the capacitor. The capacitor is not required if your application uses powerdown mode and if **both** of the following conditions are true. • an external clock input is the clock source • the phase-locked loop circuitry is disabled If your application does not use powerdown mode, leave this pin unconnected. RPD shares a package pin with P5.7.
RXD1:0	I/O	Receive Serial Data 0 and 1 In modes 1, 2, and 3, RXD0 and 1 receive serial port input data. In mode 0, they function as either inputs or open-drain outputs for data. RXD0 shares a package pin with P2.1, and RXD1 shares a package pin with P2.4.
SC1:0	I/O	Clock Pins for SSIO0 and 1 In standard mode, SC0 is the serial clock pin for channel 0 and SC1 is the serial clock pin for channel 1. In duplex and channel-select modes, SC0 is the serial clock pin for both channels 0 and 1 and SC1 is not available. SC0 shares a package pin with P10.0, and SC1 shares a package pin with P10.2.
SD1:0	I/O	Data Pins for SSIO0 and 1 These pins are the data I/O pins for SSIO0 and 1. For transmissions, configure SD*x* as a complementary output signal. For receptions, configure SD*x* as a high-impedance input signal. SD0 shares a package pin with P10.1, and SD1 shares a package pin with P10.3.
T1CLK	I	Timer 1 External Clock External clock for timer 1. Timer 1 is programmable to increment or decrement on the rising edge, the falling edge, or both rising and falling edges of T1CLK. **and** External clock for the serial I/O baud-rate generator input (program selectable). T1CLK shares a package pin with P7.0 and EPA0.
T2CLK	I	Timer 2 External Clock External clock for timer 2. Timer 2 is programmable to increment or decrement on the rising edge, the falling edge, or both rising and falling edges of T2CLK. T2CLK shares a package pin with P7.2 and EPA2.

Table B-3. Signal Descriptions (Continued)

Name	Type	Description
T3CLK	I	Timer 3 External Clock External clock for timer 3. Timer 3 is programmable to increment or decrement on the rising edge, the falling edge, or both rising and falling edges of T3CLK. T3CLK shares a package pin with P7.4 and EPA4.
T4CLK	I	Timer 4 External Clock External clock for timer 4. Timer 4 is programmable to increment or decrement on the rising edge, the falling edge, or both rising and falling edges of T4CLK. T4CLK shares a package pin with P7.6 and EPA6.
T1RST	I	Timer 1 External Reset External reset for timer 1. Timer 1 is programmable to reset on the rising edge, the falling edge, or both rising and falling edges of T1RST. T1RST shares a package pin with P7.1 and EPA1.
T2RST	I	Timer 2 External Reset External reset for timer 2. Timer 2 is programmable to reset on the rising edge, the falling edge, or both rising and falling edges of T2RST. T2RST shares a package pin with P7.3 and EPA3.
T3RST	I	Timer 3 External Reset External reset for timer 3. Timer 3 is programmable to reset on the rising edge, the falling edge, or both rising and falling edges of T3RST. T3RST shares a package pin with P7.5 and EPA5.
T4RST	I	Timer 4 External Reset External reset for timer 4. Timer 4 is programmable to reset on the rising edge, the falling edge, or both rising and falling edges of T4RST. T4RST shares a package pin with P7.7 and EPA7.
TMODE#	I	Test-Mode Entry If this pin is held low during reset, the microcontroller will enter a test mode. The value of several other pins defines the actual test mode. All test modes, except TROM execution, are reserved for Intel factory use. If you choose to configure this signal as an input, always hold it high during reset and ensure that your system meets the V_{IH} specification to prevent inadvertent entry into test mode. TMODE# shares a package pin with P5.4 and BREQ#.
TXD1:0	O	Transmit Serial Data 0 and 1 In serial I/O modes 1, 2, and 3, TXD0 and 1 transmit serial port output data. In mode 0, they are the serial clock output. TXD0 shares a package pin with P2.0 and TXD1 shares a package pin with P2.3.
V_{CC}	PWR	Digital Supply Voltage Connect each V_{CC} pin to the digital supply voltage.
V_{REF}	PWR	Reference Voltage for the A/D Converter This pin supplies operating voltage to the A/D converter.
V_{SS}	GND	Digital Circuit Ground These pins supply ground for the digital circuitry. Connect each V_{SS} pin to ground through the lowest possible impedance path.

Table B-3. Signal Descriptions (Continued)

Name	Type	Description
WR#	O	Write† This active-low output indicates that an external write is occurring. This signal is asserted only during external memory writes. WR# shares a package pin with P5.2 and WRL#. † When this pin is configured as a special-function signal (P5_MODE.2 = 1), the chip configuration register 0 (CCR0) determines whether it functions as WR# or WRL#. CCR0.2 = 1 selects WR#; CCR0.2 = 0 selects WRL#.
WRH#	O	Write High† During 16-bit bus cycles, this active-low output signal is asserted for high-byte writes and word writes to external memory. During 8-bit bus cycles, WRH# is asserted for all write operations. WRH# shares a package pin with P5.5 and BHE#. † When this pin is configured as a special-function signal (P5_MODE.5 = 1), the chip configuration register 0 (CCR0) determines whether it functions as BHE# or WRH#. CCR0.2 = 1 selects BHE#; CCR0.2 = 0 selects WRH#.
WRL#	O	Write Low† During 16-bit bus cycles, this active-low output signal is asserted for low-byte writes and word writes to external memory. During 8-bit bus cycles, WRL# is asserted for all write operations. WRL# shares a package pin with P5.2 and WR#. † When this pin is configured as a special-function signal (P5_MODE.2 = 1), the chip configuration register 0 (CCR0) determines whether it functions as WR# or WRL#. CCR0.2 = 1 selects WR#; CCR0.2 = 0 selects WRL#.
XTAL1	I	Input Crystal/Resonator or External Clock Input Input to the on-chip oscillator and the internal clock generators. The internal clock generators provide the peripheral clocks, CPU clock, and CLKOUT signal. When using an external clock source instead of the on-chip oscillator, connect the clock input to XTAL1. The external clock signal must meet the V_{IH} specification for XTAL1.
XTAL2	O	Inverted Output for the Crystal/Resonator Output of the on-chip oscillator inverter. Leave XTAL2 floating when the design uses an external clock source instead of the on-chip oscillator.

B.3 DEFAULT CONDITIONS

Table B-5 lists the values of the signals of the 8XC196EA during various operating conditions. Table B-4 defines the symbols used to represent the pin status. Refer to the DC characteristics table in the datasheet for actual specifications for V_{OL}, V_{IL}, V_{OH}, and V_{IH}.

Table B-4. Definition of Status Symbols

Symbol	Definition
0	Voltage less than or equal to V_{OL}, V_{IL}
1	Voltage greater than or equal to V_{OH}, V_{IH}
HiZ	High impedance
LoZ0	Low impedance; strongly driven low
LoZ1	Low impedance; strongly driven high

Symbol	Definition
MD0	Medium pull-down
MD1	Medium pull-up
WK0	Weak pull-down
WK1	Weak pull-up
ODIO	Open-drain I/O

Table B-5. 8XC196EA Default Signal Conditions

Port Signals	Alternate Functions	During RESET# Active	Upon RESET# Inactive (Note 8)	Idle	Powerdown
P2.0	TXD0	WK1	WK1	(Note 1)	(Note 1)
P2.1	RXD0	WK1	WK1	(Note 1)	(Note 1)
P2.2	EXTINT	WK1	WK1	(Note 1)	(Note 1)
P2.3	TXD1	WK1	WK1	(Note 1)	(Note 1)
P2.4	RXD1	WK1	WK1	(Note 1)	(Note 1)
P2.5	HOLD#	WK1	WK1	(Note 1)	(Note 1)
P2.6	HLDA#/ONCE#	MD1	MD1	(Note 1)	(Note 1)
P2.7	CLKOUT	CLKOUT active, LoZ0/1	CLKOUT active, LoZ0/1	CLKOUT active, LoZ0/1	LoZ0
P3.7:0	AD7:0	WK1	HiZ	(Note 2)	(Note 2)
P4.7:0	AD15:8	WK1	HiZ	(Note 2)	(Note 2)
P5.0	ALE	0	0	(Note 6)	(Note 6)
P5.1	INST	WK0	WK0	(Note 6)	(Note 6)
P5.2	WR#/WRL#	WK1	WK1	(Note 1)	(Note 1)
P5.3	RD#	WK1	WK1	(Note 1)	(Note 1)
P5.4	BREQ#/TMODE#	WK1	WK1	(Note 1)	(Note 1)
P5.5	BHE#/WRH#	WK1	WK1	(Note 6)	(Note 6)
P5.6	READY	WK1	WK1	(Note 7)	(Note 7)
P5.7	RPD	HiZ	HiZ	(Note 7)	(Note 7)
P7.0	EPA0/T1CLK	WK1	WK1	(Note 1)	(Note 1)
P7.1	EPA1/T1RST	WK1	WK1	(Note 1)	(Note 1)
P7.2	EPA2/T2CLK	WK1	WK1	(Note 1)	(Note 1)
P7.3	EPA3/T2RST	WK1	WK1	(Note 1)	(Note 1)
P7.4	EPA4/T3CLK	WK1	WK1	(Note 1)	(Note 1)
P7.5	EPA5/T3RST	WK1	WK1	(Note 1)	(Note 1)
P7.6	EPA6/T4CLK	WK1	WK1	(Note 1)	(Note 1)
P7.7	EPA7/T4RST	WK1	WK1	(Note 1)	(Note 1)
P8.7:0	EPA15:8	WK1	WK1	(Note 1)	(Note 1)
P9.7:0	OS7:0	WK1	WK1	(Note 1)	(Note 1)

Table B-5. 8XC196EA Default Signal Conditions (Continued)

Port Signals	Alternate Functions	During RESET# Active	Upon RESET# Inactive (Note 8)	Idle	Powerdown
P10.0	SC0	WK1	WK1	(Note 1)	(Note 1)
P10.1	SD0	WK1	WK1	(Note 1)	(Note 1)
P10.2	SC1/CHS#	WK1	WK1	(Note 1)	(Note 1)
P10.3	SD1	WK1	WK1	(Note 1)	(Note 1)
P10.4	EPA16	WK1	WK1	(Note 1)	(Note 1)
P10.5	—	WK1	WK1	(Note 1)	(Note 1)
P11.7:0	PWM7:0	WK1	WK1	(Note 1)	(Note 1)
P12.0	—	WK1	WK1	(Note 1)	(Note 1)
P12.1	—	WK1	WK1	(Note 1)	(Note 1)
P12.2	—	WK1	WK1	(Note 1)	(Note 1)
P12.3	—	WK1	WK1	(Note 1)	(Note 1)
P12.4	—	WK1	WK1	(Note 1)	(Note 1)
EPORT.4:0	A20:16	WK1	WK1	(Note 4)	(Note 4)
EPORT.7:5	CS2:0#	WK1	WK1	(Note 3)	(Note 3)
—	A15:0	WK1	LoZ0	(Note 5)	(Note 5)
—	ACH15:0	HiZ	HiZ	HiZ	HiZ
—	CRBUSY#	LoZ1	LoZ1	LoZ1	LoZ1
—	CROUT	LoZ0	LoZ0	LoZ0	LoZ0
—	CRIN	HiZ	HiZ	HiZ	HiZ
—	CRDCLK	HiZ	HiZ	HiZ	HiZ
—	EA#	WK1	WK1	WK1	WK1
—	NMI	WK0	WK0	WK0	WK0
—	PLLEN	HiZ	HiZ	HiZ	HiZ
—	RESET#	LoZ0	WK1	WK1	WK1
—	XTAL1	Osc input, HiZ	Osc input, HiZ	Osc input, HiZ	Osc input, HiZ
—	XTAL2	Osc output, LoZ0/1	Osc output, LoZ0/1	Osc output, LoZ0/1	HiZ

NOTES:
1. If P*x*_MODE.*y* = 0, port is as programmed. If P*x*_MODE.*y* = 1, pin is as specified by the associated peripheral.
2. If EA# = 0, port is HiZ. If EA# = 1, port is open-drain I/O.
3. When used as chip select, if HLDA# = 0, pin is WK1; if HLDA# = 1, pin is LoZ1. When used as EPORT, port is as programmed.
4. When used as extended address, if HLDA# = 1, pin is 0; if HLDA# = 0, pin is HiZ.
   When used as EPORT, port is as programmed.
5. If HLDA# = 1, pin is LoZ0. If HLDA# = 0, pin is HiZ.
6. If P5_MODE.y = 0, port is as programmed.
   If P5_MODE.y = 1 and HLDA# = 1, P5.0 and P5.1 are LoZ0; P5.5 is LoZ1.
   If P5_MODE.y = 1 and HLDA# = 0, port is HiZ.
7. If P5_MODE.y = 0, port is as programmed. If P5_MODE.y = 1, port is HiZ.
8. The values in this column are valid until your software writes to P*x*_MODE.

C

Registers

APPENDIX C
REGISTERS

This appendix provides reference information about the microcontroller registers. Table C-1 lists the modules and major components of the microcontroller with their related configuration and status registers. Table C-2 lists the registers, arranged alphabetically by mnemonic, along with their names, addresses, and reset values. Following the tables, individual descriptions of the registers are arranged alphabetically by mnemonic.

Table C-1. Modules and Related Registers

A/D Converter (*x* = 0–15)	**Chip Configuration**	**Chip-select Units (*x* = 0–2)**	**CPU**
AD_COMMAND	CCR0	ADDRCOM*x*	ONES_REG
AD_RESULT	CCR1	ADDRMSK*x*	PSW
AD_RESULT*x*	CLKOUT_CON	BUSCON*x*	SP
AD_SCAN	RSTSRC		STACK_BOTTOM
AD_TEST			STACK_TOP
AD_TIME			ZERO_REG
EPA (*x* = 0–16) (*y* = 0–7)	**I/O Ports (*x* = 2, 5, 7–12) (*y* = 2–5, 7–12)**	**Interrupts and PTS (*x* = 0–1)**	**Memory Control**
EPA*x*_CON	EP_DIR	INT_MASK	IRAM_CON
EPA*x*_TIME	EP_MODE	INT_MASK1	WSR
OS*y*_CON	EP_PIN	INT_PEND	WSR1
OS*y*_TIME	EP_REG	INT_PEND1	
	P*x*_DIR	PIH*x*_INT_MASK	
	P*x*_MODE	PIH*x*_INT_PEND	
	P*y*_PIN	PIH*x*_PTSSEL	
	P*y*_REG	PIH*x*_PTSSRV	
	P34_DRV	PIH*x*_VEC_BASE	
		PIH*x*_VEC_IDX	
		PTSSEL	
		PTSSRV	

Table C-1. Modules and Related Registers (Continued)

PWM (*x* = 0, 2, 4, 6) (*y* = 1, 3, 5, 7) (*z* = 0–7)	SIO (*x* = 0–1)	SSIO (*x* = 0–1)	Timers (*x* = 1–4)
PWM*x_y*_COUNT	SBUF*x*_RX	SSIO_BAUD	T*x*CONTROL
PWM*x_y*_PERIOD	SBUF*x*_TX	SSIO*x*_BUF	TIMER*x*
PWM*z*_CONTROL	SP*x*_BAUD	SSIO*x*_CLK	TIMER_MUX
	SP*x*_CON	SSIO*x*_CON	WATCHDOG
	SP*x*_STATUS		

Table C-2. Register Name, Address, and Reset Value

Register Mnemonic	Register Name	Hex Addr.	Binary Reset Value	
			High	Low
AD_COMMAND	A/D Command	1E74	0000 0000	0000 0000
AD_RESULT	A/D Result	1E72	0000 0000	0000 0000
AD_RESULT0	A/D Result 0	1E50	0000 0000	0000 0000
AD_RESULT1	A/D Result 1	1E52	0000 0000	0000 0000
AD_RESULT2	A/D Result 2	1E54	0000 0000	0000 0000
AD_RESULT3	A/D Result 3	1E56	0000 0000	0000 0000
AD_RESULT4	A/D Result 4	1E58	0000 0000	0000 0000
AD_RESULT5	A/D Result 5	1E5A	0000 0000	0000 0000
AD_RESULT6	A/D Result 6	1E5C	0000 0000	0000 0000
AD_RESULT7	A/D Result 7	1E5E	0000 0000	0000 0000
AD_RESULT8	A/D Result 8	1E60	0000 0000	0000 0000
AD_RESULT9	A/D Result 9	1E62	0000 0000	0000 0000
AD_RESULT10	A/D Result 10	1E64	0000 0000	0000 0000
AD_RESULT11	A/D Result 11	1E66	0000 0000	0000 0000
AD_RESULT12	A/D Result 12	1E68	0000 0000	0000 0000
AD_RESULT13	A/D Result 13	1E6A	0000 0000	0000 0000
AD_RESULT14	A/D Result 14	1E6C	0000 0000	0000 0000
AD_RESULT15	A/D Result 15	1E6E	0000 0000	0000 0000
AD_SCAN	A/D Scan	1E70	0000 0000	0000 0000
AD_TEST	A/D Test	1E76		0000 0000
AD_TIME	A/D Time	1E77		1111 1111

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after a device reset. The CCBs reside in internal nonvolatile memory at addresses FF2018H (CCB0) and FF201AH (CCB1).

†† Reset value is 80H if the EA# pin is high, A9H if EA# is low.

Table C-2. Register Name, Address, and Reset Value (Continued)

Register Mnemonic	Register Name	Hex Addr.	Binary Reset Value	
			High	Low
ADDRCOM0	Address Compare 0	1E78	0001 1111	0010 0000
ADDRCOM1	Address Compare 1	1E80	0000 0000	0000 0000
ADDRCOM2	Address Compare 2	1E88	0000 0000	0000 0000
ADDRMSK0	Address Mask 0	1E7A	0001 1111	1111 1111
ADDRMSK1	Address Mask 1	1E82	0001 1111	1111 1111
ADDRMSK2	Address Mask 2	1E8A	0001 1111	1111 1111
BUSCON0	Bus Control 0	1E7C		0000 0011
BUSCON1	Bus Control 1	1E84		0000 0000
BUSCON2	Bus Control 2	1E8C		0000 0000
CCR0	Chip Configuration 0	†		XXXX XXXX
CCR1	Chip Configuration 1	†		XXXX XXXX
CLKOUT_CON	Clock Out Control	1F80		0000 0000
EP_DIR	Extended Port I/O Direction	1FE3		1101 1111
EP_MODE	Extended Port Mode	1FE1		0011 1111
EP_PIN	Extended Port Pin Input	1FE7		XXXX XXXX
EP_REG	Extended Port Data Output	1FE5		1110 0000
EPA0_CON	EPA Capture/Compare 0 Control	1F5C	0000 0000	0000 0000
EPA1_CON	EPA Capture/Compare 1 Control	1F58	0000 0000	0000 0000
EPA2_CON	EPA Capture/Compare 2 Control	1F54	0000 0000	0000 0000
EPA3_CON	EPA Capture/Compare 3 Control	1F50	0000 0000	0000 0000
EPA4_CON	EPA Capture/Compare 4 Control	1F4C	0000 0000	0000 0000
EPA5_CON	EPA Capture/Compare 5 Control	1F48	0000 0000	0000 0000
EPA6_CON	EPA Capture/Compare 6 Control	1F44	0000 0000	0000 0000
EPA7_CON	EPA Capture/Compare 7 Control	1F40	0000 0000	0000 0000
EPA8_CON	EPA Capture/Compare 8 Control	1F3C	0000 0000	0000 0000
EPA9_CON	EPA Capture/Compare 9 Control	1F38	0000 0000	0000 0000
EPA10_CON	EPA Capture/Compare 10 Control	1F34	0000 0000	0000 0000
EPA11_CON	EPA Capture/Compare 11 Control	1F30	0000 0000	0000 0000
EPA12_CON	EPA Capture/Compare 12 Control	1F2C	0000 0000	0000 0000
EPA13_CON	EPA Capture/Compare 13 Control	1F28	0000 0000	0000 0000
EPA14_CON	EPA Capture/Compare 14 Control	1F24	0000 0000	0000 0000

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after a device reset. The CCBs reside in internal nonvolatile memory at addresses FF2018H (CCB0) and FF201AH (CCB1).

†† Reset value is 80H if the EA# pin is high, A9H if EA# is low.

Table C-2. Register Name, Address, and Reset Value (Continued)

Register Mnemonic	Register Name	Hex Addr.	Binary Reset Value			
			High		Low	
EPA15_CON	EPA Capture/Compare 15 Control	1F20	0000	0000	0000	0000
EPA16_CON	EPA Capture/Compare 16 Control	1F1C	0000	0000	0000	0000
EPA0_TIME	EPA Capture/Compare 0 Time	1F5E	XXXX	XXXX	XXXX	XXXX
EPA1_TIME	EPA Capture/Compare 1 Time	1F5A	XXXX	XXXX	XXXX	XXXX
EPA2_TIME	EPA Capture/Compare 2 Time	1F56	XXXX	XXXX	XXXX	XXXX
EPA3_TIME	EPA Capture/Compare 3 Time	1F52	XXXX	XXXX	XXXX	XXXX
EPA4_TIME	EPA Capture/Compare 4 Time	1F4E	XXXX	XXXX	XXXX	XXXX
EPA5_TIME	EPA Capture/Compare 5 Time	1F4A	XXXX	XXXX	XXXX	XXXX
EPA6_TIME	EPA Capture/Compare 6 Time	1F46	XXXX	XXXX	XXXX	XXXX
EPA7_TIME	EPA Capture/Compare 7 Time	1F42	XXXX	XXXX	XXXX	XXXX
EPA8_TIME	EPA Capture/Compare 8 Time	1F3E	XXXX	XXXX	XXXX	XXXX
EPA9_TIME	EPA Capture/Compare 9 Time	1F3A	XXXX	XXXX	XXXX	XXXX
EPA10_TIME	EPA Capture/Compare 10 Time	1F36	XXXX	XXXX	XXXX	XXXX
EPA11_TIME	EPA Capture/Compare 11 Time	1F32	XXXX	XXXX	XXXX	XXXX
EPA12_TIME	EPA Capture/Compare 12 Time	1F2E	XXXX	XXXX	XXXX	XXXX
EPA13_TIME	EPA Capture/Compare 13 Time	1F2A	XXXX	XXXX	XXXX	XXXX
EPA14_TIME	EPA Capture/Compare 14 Time	1F26	XXXX	XXXX	XXXX	XXXX
EPA15_TIME	EPA Capture/Compare 15 Time	1F22	XXXX	XXXX	XXXX	XXXX
EPA16_TIME	EPA Capture/Compare 16 Time	1F1E	XXXX	XXXX	XXXX	XXXX
INT_MASK	Interrupt Mask	0008			0000	0000
INT_MASK1	Interrupt Mask 1	0013			0000	0000
INT_PEND	Interrupt Pending	0009			0000	0000
INT_PEND1	Interrupt Pending 1	0012			0000	0000
IRAM_CON	Internal RAM Control	1FE0			0000	0000
ONES_REG	Ones Register	0002	1111	1111	1111	1111
OS0_CON	Output Simulcapture 0 Control	1EFC	0000	0000	0000	0000
OS1_CON	Output Simulcapture 1 Control	1EF8	0000	0000	0000	0000
OS2_CON	Output Simulcapture 2 Control	1EF4	0000	0000	0000	0000
OS3_CON	Output Simulcapture 3 Control	1EF0	0000	0000	0000	0000
OS4_CON	Output Simulcapture 4 Control	1EEC	0000	0000	0000	0000
OS5_CON	Output Simulcapture 5 Control	1EE8	0000	0000	0000	0000

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after a device reset. The CCBs reside in internal nonvolatile memory at addresses FF2018H (CCB0) and FF201AH (CCB1).

†† Reset value is 80H if the EA# pin is high, A9H if EA# is low.

Table C-2. Register Name, Address, and Reset Value (Continued)

Register Mnemonic	Register Name	Hex Addr.	Binary Reset Value	
			High	Low
OS6_CON	Output Simulcapture 6 Control	1EE4	0000 0000	0000 0000
OS7_CON	Output Simulcapture 7 Control	1EE0	0000 0000	0000 0000
OS0_TIME	Output Simulcapture 0 Time	1EFE	XXXX XXXX	XXXX XXXX
OS1_TIME	Output Simulcapture 1 Time	1EFA	XXXX XXXX	XXXX XXXX
OS2_TIME	Output Simulcapture 2 Time	1EF6	XXXX XXXX	XXXX XXXX
OS3_TIME	Output Simulcapture 3 Time	1EF2	XXXX XXXX	XXXX XXXX
OS4_TIME	Output Simulcapture 4 Time	1EEE	XXXX XXXX	XXXX XXXX
OS5_TIME	Output Simulcapture 5 Time	1EEA	XXXX XXXX	XXXX XXXX
OS6_TIME	Output Simulcapture 6 Time	1EE6	XXXX XXXX	XXXX XXXX
OS7_TIME	Output Simulcapture 7 Time	1EE2	XXXX XXXX	XXXX XXXX
P2_DIR	Port 2 I/O Direction	1FD2		0111 1111
P5_DIR	Port 5 I/O Direction	1FF3		1111 1111
P7_DIR	Port 7 I/O Direction	1FCA		1111 1111
P8_DIR	Port 8 I/O Direction	1FCB		1111 1111
P9_DIR	Port 9 I/O Direction	1FC2		1111 1111
P10_DIR	Port 10 I/O Direction	1FC3		0011 1111
P11_DIR	Port 11 I/O Direction	1FBA		1111 1111
P12_DIR	Port 12 I/O Direction	1FEA		0001 1111
P2_MODE	Port 2 Mode	1FD0		1000 0000
P5_MODE	Port 5 Mode	1FF1		††
P7_MODE	Port 7 Mode	1FC8		0000 0000
P8_MODE	Port 8 Mode	1FC9		0000 0000
P9_MODE	Port 9 Mode	1FC0		0000 0000
P10_MODE	Port 10 Mode	1FC1		0000 0000
P11_MODE	Port 11 Mode	1FB8		0000 0000
P12_MODE	Port 12 Mode	1FE8		0000 0000
P2_PIN	Port 2 Pin Input	1FD6		XXXX XXXX
P3_PIN	Port 3 Pin Input	1FFE		XXXX XXXX
P4_PIN	Port 4 Pin Input	1FFF		XXXX XXXX
P5_PIN	Port 5 Pin Input	1FF7		XXXX XXXX
P7_PIN	Port 7 Pin Input	1FCE		XXXX XXXX

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after a device reset. The CCBs reside in internal nonvolatile memory at addresses FF2018H (CCB0) and FF201AH (CCB1).

†† Reset value is 80H if the EA# pin is high, A9H if EA# is low.

Table C-2. Register Name, Address, and Reset Value (Continued)

Register Mnemonic	Register Name	Hex Addr.	Binary Reset Value	
			High	Low
P8_PIN	Port 8 Pin Input	1FCF		XXXX XXXX
P9_PIN	Port 9 Pin Input	1FC6		XXXX XXXX
P10_PIN	Port 10 Pin Input	1FC7		XXXX XXXX
P11_PIN	Port 11 Pin Input	1FBE		XXXX XXXX
P12_PIN	Port 12 Pin Input	1FEE		XXXX XXXX
P2_REG	Port 2 Data Output	1FD4		1111 1111
P3_REG	Port 3 Data Output	1FFC		1111 1111
P4_REG	Port 4 Data Output	1FFD		1111 1111
P5_REG	Port 5 Data Output	1FF5		1111 1111
P7_REG	Port 7 Data Output	1FCC		1111 1111
P8_REG	Port 8 Data Output	1FCD		1111 1111
P9_REG	Port 9 Data Output	1FC4		1111 1111
P10_REG	Port 10 Data Output	1FC5		0011 1111
P11_REG	Port 11 Data Output	1FBC		1111 1111
P12_REG	Port 12 Data Output	1FEC		0001 1111
P34_DRV	Port 3/4 Push-pull Enable	1FF4		0000 1111
PIH0_INT_MASK	Peripheral Int Handler 0 Int Mask	1E98	0000 0000	0000 0000
PIH1_INT_MASK	Peripheral Int Handler 1 Int Mask	1EA8	0000 0000	0000 0000
PIH0_INT_PEND	Peripheral Int Handler 0 Int Pending	1E9A	0000 0000	0000 0000
PIH1_INT_PEND	Peripheral Int Handler 1 Int Pending	1EAA	0000 0000	0000 0000
PIH0_PTSSEL	Peripheral Int Handler 0 PTS Select	1E96	0000 0000	0000 0000
PIH1_PTSSEL	Peripheral Int Handler 1 PTS Select	1EA6	0000 0000	0000 0000
PIH0_PTSSRV	Peripheral Int Handler 0 PTS Service	1E94	0000 0000	0000 0000
PIH1_PTSSRV	Peripheral Int Handler 1 PTS Service	1EA4	0000 0000	0000 0000
PIH0_VEC_BASE	Peripheral Int Handler 0 Vector Base	1E92	XXXX XXXX	XXXX XXXX
PIH1_VEC_BASE	Peripheral Int Handler 1 Vector Base	1EA2	XXXX XXXX	XXXX XXXX
PIH0_VEC_IDX	Peripheral Int Handler 0 Vector Index	1E90		0001 0000
PIH1_VEC_IDX	Peripheral Int Handler 1 Vector Index	1EA0		0001 0000
PSW	Processor Status Word	no direct access		
PTSSEL	PTS Select	0004	0000 0000	0000 0000

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after a device reset. The CCBs reside in internal nonvolatile memory at addresses FF2018H (CCB0) and FF201AH (CCB1).

†† Reset value is 80H if the EA# pin is high, A9H if EA# is low.

Table C-2. Register Name, Address, and Reset Value (Continued)

Register Mnemonic	Register Name	Hex Addr.	Binary Reset Value	
			High	Low
PTSSRV	PTS Service	0006	0000 0000	0000 0000
PWM0_1_COUNT	PWM 0 and 1 Count	1EDD		XXXX XXXX
PWM2_3_COUNT	PWM 2 and 3 Count	1ED9		XXXX XXXX
PWM4_5_COUNT	PWM 4 and 5 Count	1ED5		XXXX XXXX
PWM6_7_COUNT	PWM 6 and 7 Count	1ED1		XXXX XXXX
PWM0_1_PERIOD	PWM 0 and 1 Period	1EDF		1111 1111
PWM2_3_PERIOD	PWM 2 and 3 Period	1EDB		1111 1111
PWM4_5_PERIOD	PWM 4 and 5 Period	1ED7		1111 1111
PWM6_7_PERIOD	PWM 6 and 7 Period	1ED3		1111 1111
PWM0_CONTROL	PWM 0 Control	1EDE		0000 0000
PWM1_CONTROL	PWM 1 Control	1EDC		0000 0000
PWM2_CONTROL	PWM 2 Control	1EDA		0000 0000
PWM3_CONTROL	PWM 3 Control	1ED8		0000 0000
PWM4_CONTROL	PWM 4 Control	1ED6		0000 0000
PWM5_CONTROL	PWM 5 Control	1ED4		0000 0000
PWM6_CONTROL	PWM 6 Control	1ED2		0000 0000
PWM7_CONTROL	PWM 7 Control	1ED0		0000 0000
RSTSRC	Reset Source Indicator	1FA4		0000 0000
SBUF0_RX	Serial Port Receive Buffer 0	1F88		0000 0000
SBUF1_RX	Serial Port Receive Buffer 1	1F98		0000 0000
SBUF0_TX	Serial Port Transmit Buffer 0	1F8A		0000 0000
SBUF1_TX	Serial Port Transmit Buffer 1	1F9A		0000 0000
SP	Stack Pointer	0018	XXXX XXXX	XXXX XXXX
SP0_BAUD	Serial Port 0 Baud Rate	1F8C	0000 0000	0000 0000
SP1_BAUD	Serial Port 1 Baud Rate	1F9C	0000 0000	0000 0000
SP0_CON	Serial Port 0 Control	1F8B		0000 0000
SP1_CON	Serial Port 1 Control	1F9B		0000 0000
SP0_STATUS	Serial Port 0 Status	1F89		0000 1000
SP1_STATUS	Serial Port 1 Status	1F99		0000 1000
SSIO_BAUD	Synchronous Serial Port Baud Rate	1F94		0000 0000
SSIO0_BUF	Synchronous Serial Port 0 Buffer	1F90		XXXX XXXX

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after a device reset. The CCBs reside in internal nonvolatile memory at addresses FF2018H (CCB0) and FF201AH (CCB1).

†† Reset value is 80H if the EA# pin is high, A9H if EA# is low.

Table C-2. Register Name, Address, and Reset Value (Continued)

Register Mnemonic	Register Name	Hex Addr.	Binary Reset Value			
			High		Low	
SSIO1_BUF	Synchronous Serial Port 1 Buffer	1F92			XXXX	XXXX
SSIO0_CLK	Synchronous Serial Port 0 Clock	1F95			0000	0000
SSIO1_CLK	Synchronous Serial Port 1 Clock	1F97			0000	0000
SSIO0_CON	Synchronous Serial Port 0 Control	1F91			0000	0000
SSIO1_CON	Synchronous Serial Port 1 Control	1F93			0000	0000
STACK_BOTTOM	Stack Bottom	1FA0	0000	0000	0000	0000
STACK_TOP	Stack Top	1FA2	1111	1111	1111	1111
T1CONTROL	Timer 1 Control	1F7C	0000	0000	0000	0000
T2CONTROL	Timer 2 Control	1F78	0000	0000	0000	0000
T3CONTROL	Timer 3 Control	1F74	0000	0000	0000	0000
T4CONTROL	Timer 4 Control	1F70	0000	0000	0000	0000
TIMER1	Timer 1 Value	1F7E	0000	0000	0000	0000
TIMER2	Timer 2 Value	1F7A	0000	0000	0000	0000
TIMER3	Timer 3 Value	1F76	0000	0000	0000	0000
TIMER4	Timer 4 Value	1F72	0000	0000	0000	0000
TIMER_MUX	Timer Multiplexer	1F6E			0000	0000
WATCHDOG	Watchdog Timer	000A			XXXX	XXXX
WSR	Window Selection	0014			0000	0000
WSR1	Window Selection 1	0015			0000	0000
ZERO_REG	Zero Register	0000	0000	0000	0000	0000

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after a device reset. The CCBs reside in internal nonvolatile memory at addresses FF2018H (CCB0) and FF201AH (CCB1).

†† Reset value is 80H if the EA# pin is high, A9H if EA# is low.

AD_COMMAND

Address: 1E74H
Reset State: 00H

The A/D command (AD_COMMAND) register selects the A/D channel number to be converted, controls whether the A/D converter starts immediately or with an EPA command, initiates automatic scan on selected A/D channel inputs, and selects the conversion mode.

7							0
SCAN	M1	M0	GO	ACH3	ACH2	ACH1	ACH0

Bit Number	Bit Mnemonic	Function
7	SCAN	A/D Scan This bit enables and disables automatic scanning. 0 = disable scan 1 = enable scan
6:5	M1:0	A/D Mode† These bits determine the A/D mode. **M1 M0 Mode** 0 0 10-bit conversion 0 1 8-bit conversion 1 0 threshold detect low 1 1 threshold detect high
4	GO	A/D Conversion Trigger†† Writing this bit arms the A/D converter. The value that you write to it determines at what point a conversion is to start. 0 = EPA initiates conversion 1 = start immediately
3:0	ACH3:0	A/D Channel Selection Write the A/D conversion channel number to these bits. The 83C196EA has 16 A/D channel inputs, numbered 0–15.

† While a threshold-detection mode is selected for an analog input pin, no other conversion can be started. If another value is loaded into AD_COMMAND, the threshold-detection mode is disabled and the new command is executed.

†† It is the act of writing to the GO bit, rather than its value, that starts a conversion. Even if the GO bit has the desired value, you must set it again to start a conversion immediately or clear it again to arm it for an EPA-initiated conversion.

Figure C-1. A/D Command (AD_COMMAND) Register

AD_RESULT (Read)

Address: 1E72H
Reset State: 0000H

The A/D result (AD_RESULT) register consists of two bytes. The high byte contains the eight most-significant bits from the A/D converter. The low byte contains the two least-significant bits from a 10-bit A/D conversion, indicates the A/D channel number that was used for the conversion, and indicates whether a conversion is currently in progress.

15							8
ADRLT9	ADRLT8	ADRLT7	ADRLT6	ADRLT5	ADRLT4	ADRLT3	ADRLT2
7							**0**
ADRLT1	ADRLT0	—	STATUS	ACH3	ACH2	ACH1	ACH0

Bit Number	Bit Mnemonic	Function
15:6	ADRLT9:0	A/D Result These bits contain the A/D conversion result.
5	—	Reserved. This bit is undefined.
4	STATUS	A/D Status Indicates the status of the A/D converter. Up to 8 state times are required to set this bit following a start command. When testing this bit, wait at least the 8 state times. 0 = A/D is idle 1 = A/D conversion is in progress
3:0	ACH3:0	A/D Channel Number These bits indicate the A/D channel number that was used for the conversion. The 83C196EA has 16 A/D channel inputs, numbered 0–15.

Figure C-2. A/D Result (AD_RESULT) Register — Read Format

AD_RESULT (Write)

Address: 1E72H
Reset State: 0000H

The high byte of the A/D result (AD_RESULT) register can be written to set the reference voltage for the A/D threshold-detection modes.

15							8
REFV7	REFV6	REFV5	REFV4	REFV3	REFV2	REFV1	REFV0
7							**0**
—	—	—	—	—	—	—	—

Bit Number	Bit Mnemonic	Function
15:8	REFV7:0	Reference Voltage These bits specify the threshold value. This reference voltage is compared with an analog input pin. When the voltage on the analog input pin crosses over (detect high) or under (detect low) the threshold value, the A/D conversion complete interrupt pending bit is set. Use the following formula to determine the value to write to this register for a given threshold voltage. $\text{reference voltage} = \frac{\text{desired threshold voltage} \times 256}{V_{REF} - \text{ANGND}}$
7:0	—	Reserved; for compatibility with future devices, write zeros to these bits.

Figure C-3. A/D Result (AD_RESULT) Register — Write Format

AD_RESULT*x* (Read)

Address: 1E72H
Reset State: 0000H

The individual A/D result (AD_RESULT*x*) registers contain the result from the A/D channel *x* conversion. The low byte contains the eight LSBs from the conversion, while the high byte contains the two MSBs from a 10-bit conversion (undefined for an 8-bit conversion).

15							8
—	—	—	—	—	—	ADRLT9	ADRLT8
7							**0**
ADRLT7	ADRLT6	ADRLT5	ADRLT4	ADRLT3	ADRLT2	ADRLT1	ADRLT0

Bit Number	Bit Mnemonic	Function
15:10	—	Reserved. These bits are undefined.
9:0	ADRLT9:0	A/D Result on Channel *x* These bits contain the A/D conversion result.

Figure C-4. A/D Result *x* (AD_RESULT*x*) Register — Read Format

Table C-3. AD_RESULT*x* Addresses and Reset States

Register	Address	Reset State	Register	Address	Reset State
AD_RESULT0	1E50H	0000H	AD_RESULT8	1E60H	0000H
AD_RESULT1	1E52H	0000H	AD_RESULT9	1E62H	0000H
AD_RESULT2	1E54H	0000H	AD_RESULT10	1E64H	0000H
AD_RESULT3	1E56H	0000H	AD_RESULT11	1E66H	0000H
AD_RESULT4	1E58H	0000H	AD_RESULT12	1E68H	0000H
AD_RESULT5	1E5AH	0000H	AD_RESULT13	1E6AH	0000H
AD_RESULT6	1E5CH	0000H	AD_RESULT14	1E6CH	0000H
AD_RESULT7	1E5EH	0000H	AD_RESULT15	1E6EH	0000H

AD_SCAN

Address: 1E70H
Reset State: 0000H

The A/D scan (AD_SCAN) register allows you to select individual channels to include in the scan sequence and those to exclude. The automatic scan begins with the least-significant channel and ends with the most-significant channel.

15							8
ACH15	ACH14	ACH13	ACH12	ACH11	ACH10	ACH9	ACH8
7							**0**
ACH7	ACH6	ACH5	ACH4	ACH3	ACH2	ACH1	ACH0

Bit Number	Bit Mnemonic	Function
15:0	ACH15:0	A/D Channel *x* These bits select the channels to be included in the automatic scan. 0 = exclude A/D channel *x* from scan sequence 1 = include A/D channel *x* in scan sequence

Figure C-5. A/D Scan (AD_SCAN) Register

AD_TEST

Address: 1E76H
Reset State: 00H

The A/D test (AD_TEST) register enables conversions on ANGND and V_{REF} and specifies adjustments for DC offset errors. Its functions allow you to perform two conversions, one on ANGND and one on V_{REF}. With these results, a software routine can calculate the offset and gain errors.

7 **0**

—	—	—	OFF2	OFF1	OFF0	TV	TE

Bit Number	Bit Mnemonic	Function
7:5	—	Reserved; for compatibility with future devices, write zeros to these bits.
4:2	OFF2:0	Offset These bits allow you to set the zero-offset point. **OFF2 OFF1 OFF0** 0 0 0 no adjustment 0 0 1 add 2.5 mV 0 1 0 subtract 2.5 mV 0 1 1 subtract 5.0 mV 1 0 0 no adjustment 1 0 1 add 5.0 mV 1 1 0 subtract 7.5 mV 1 1 1 subtract 10 mV
1	TV	Test Voltage This bit selects the test voltage for a test mode conversion. (The TE bit must be set to enable test mode.) 0 = ANGND 1 = V_{REF}
0	TE	Test Enable This bit determines whether normal or test mode conversions will be performed. A normal conversion converts the analog signal input on one of the analog input channels. A test conversion allows you to perform a conversion on ANGND or V_{REF}, selected by the TV bit. 0 = normal 1 = test

Figure C-6. A/D Test (AD_TEST) Register

AD_TIME

Address: 1E77H
Reset State: FFH

The A/D time (AD_TIME) register programs the sample window time and the conversion time for each bit. This register programs the speed at which the A/D can run — not the speed at which it can convert correctly. Consult the data sheet for recommended values. Initialize the AD_TIME register before initializing the AD_COMMAND register. Do not write to this register while a conversion is in progress; the results are unpredictable.

7							0
SAM2	SAM1	SAM0	CONV4	CONV3	CONV2	CONV1	CONV0

Bit Number	Bit Mnemonic	Function
7:5	SAM2:0	A/D Sample Time These bits specify the sample time. Use the following formula to compute the sample time. $SAM = \frac{T_{SAM} \times f - 2}{8}$ where: SAM = 1 to 7 T_{SAM} = the sample time, in µsec, from the data sheet f = the internal operating frequency, in MHz
4:0	CONV4:0	A/D Bit Conversion Time These bits specify the conversion time for each bit. Use the following formula to compute the conversion time. $CONV = \left[\frac{T_{CONV} \times f - 3}{2 \times B}\right] - 1$ where: CONV = 2 to 31 T_{CONV} = the conversion time, in µsec, from the data sheet f = the internal operating frequency, in MHz B = the number of bits to be converted (8 or 10)

Figure C-7. A/D Time (AD_TIME) Register

ADDRCOM*x*
***x* = 0–2**

Address: 1E78H, 1E80H, 1E88H
Reset State: 1F20H, 0000H, 0000H

The address compare (ADDRCOM*x*) register specifies the base (lowest) address of the address range. The base address of a 2^n-byte address range must be on a 2^n-byte boundary.

15							8
—	—	—	BASE20	BASE19	BASE18	BASE17	BASE16
7							**0**
BASE15	BASE14	BASE13	BASE12	BASE11	BASE10	BASE9	BASE8

Bit Number	Bit Mnemonic	Function
15:13	—	Reserved; for compatibility with future devices, write zeros to these bits.
12:0	BASE20:8	Base Address Bits Write address bits 20–8 of the base address of the address range assigned to chip-select *x* to these bits.

Figure C-8. Address Compare (ADDRCOM*x*) Registers

Table C-4. ADDRCOM*x* Addresses and Reset States

Register	Address	Reset States
ADDRCOM0	1E78H	1F20H
ADDRCOM1	1E80H	0000H
ADDRCOM2	1E88H	0000H

ADDRMSK*x*
***x* = 0–2**

Address: 1E7AH, 1E82H, 1E8AH
Reset State: 1FFFH

The address mask (ADDRMSK*x*) register, together with the address compare register, defines the address range that is assigned to the chip-select *x* output, CS*x*#. The address mask register determines the size of the address range, which must be 2^n bytes, where n = 8, 9, . . , 21. For a 2^n-byte address range, calculate n_1 = 21– n, and set the n_1 most-significant bits of MASK20:8 in the address mask register.

15							8
—	—	—	MASK20	MASK19	MASK18	MASK17	MASK16
7							**0**
MASK15	MASK14	MASK13	MASK12	MASK11	MASK10	MASK9	MASK8

Bit Number	Bit Mnemonic	Function
15:13	—	Reserved; for compatibility with future devices, write zeros to these bits.
12:0	MASK20:8	Address Mask Bits For a 2^n-byte address range, set the n_1 most-significant bits of MASK20:8, where n_1 = 21 – n.

Figure C-9. Address Mask (ADDRMSK*x*) Registers

Table C-5. ADDRMSK*x* Addresses and Reset States

Register	Address	Reset State
ADDRMSK0	1E7AH	1FFFH
ADDRMSK1	1E82H	1FFFH
ADDRMSK2	1E8AH	1FFFH

BUSCON*x*
***x* = 0–2**

Address: 1E7CH, 1E84H, 1E8CH
Reset State: 0003H, 0000H, 0000H

For the address range assigned to chip-select *x*, the bus control (BUSCON*x*) register specifies the number of wait states, the bus width, and the address/data multiplexing for all external bus cycles that access address range *x*.

The chip-select output signals share package pins with port 3. Use the port registers to configure these pins as general-purpose I/O signals or as chip-select signals (see "Chip-select Signals (EPORT)" on page 7-16). The bus configuration programmed in BUSCON*x* applies to address range *x*, regardless of the port 3 pin configurations.

7							0
DEMUX	BW16	—	—	—	—	WS1	WS0

Bit Number	Bit Mnemonic	Function
7	DEMUX	Address/Data Multiplexing This bit specifies the address/data multiplexing on AD15:0 for all external accesses to the address range assigned to chip-select *x* output. 0 = multiplexed 1 = demultiplexed
6	BW16	Bus Width This bit specifies the bus width for all external accesses to the address range assigned to chip-select *x* output. 0 = 8 bits 1 = 16 bits
5:2	—	Reserved; for compatibility with future devices, write zeros to these bits.
1:0	WS1:0	Wait States These bits, along with the READY pin, control the number of wait states for all external accesses to the address range assigned to the chip-select *x* channel. Write the desired minimum number of wait states (0–3) to WS1:0. If READY is low when this number is reached, additional wait states are added until READY is pulled high.

Figure C-10. Bus Control (BUSCON*x*) Registers

Table C-6. BUSCON*x* Addresses and Reset States

Register	Address	Reset State
BUSCON0	1E7CH	03H
BUSCON1	1E84H	00H
BUSCON2	1E8CH	00H

CCR0 no direct access†

The chip configuration 0 (CCR0) register controls ROM access, enables or disables the IDLPD #1 and IDLPD #2 instructions and selects the write-control mode. It also contains the bus-control parameters for fetching chip configuration byte 1.

7							0
LOC	1	WS1	WS0	DEMUX	BHE#	BW16	PD

Bit Number	Bit Mnemonic	Function
7	LOC	Lock Bit This bit controls read access to the ROM during normal operation. 0 = read protect 1 = no protection Refer to “Controlling Read Access to the Internal ROM” on page 4-26 for details.
6	1	To guarantee proper operation, write one to this bit.
5:4	WS1:0	Wait States These bits, along with the READY pin, control the number of wait states that are used for an external fetch of chip configuration byte 1 (CCB1). **WS1 WS0** 0 0 zero wait states 0 1 one wait state 1 0 two wait states 1 1 three wait states If READY is low when this number is reached, additional wait states are added until READY is pulled high.
3	DEMUX	Select Demultiplexed Bus Selects the demultiplexed bus mode for an external fetch of CCB1: 0 = multiplexed — address and data are multiplexed on AD15:0. 1 = demultiplexed — data only on AD15:0.
2	BHE#	Write-control Mode Selects the write-control mode, which determines the functions of the BHE#/WRH# and WR#/WRL# pins for external bus cycles: 0 = write strobe mode: the BHE#/WRH# pin operates as WRH#, and the WR#/WRL# pin operates as WRL#. 1 = standard write-control mode: the BHE#/WRH# pin operates as BHE#, and the WR#/WRL# pin operates as WR#.

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after reset. The CCBs reside at addresses FF2018H (CCB0) and FF201AH (CCB1); therefore, define your CCRs at these addresses.

Figure C-11. Chip Configuration 0 (CCR0) Register

CCR0 (Continued) no direct access†

The chip configuration 0 (CCR0) register controls ROM access, enables or disables the IDLPD #1 and IDLPD #2 instructions and selects the write-control mode. It also contains the bus-control parameters for fetching chip configuration byte 1.

7							0
LOC	1	WS1	WS0	DEMUX	BHE#	BW16	PD

Bit Number	Bit Mnemonic	Function
1	BW16	Buswidth Control Selects the bus width for an external fetch of CCB1: 0 = 8-bit bus 1 = 16-bit bus
0	PD	Powerdown Enable Enables or disables the IDLPD #1 and IDLPD #2 instructions. When enabled, the IDLPD #1 instruction causes the microcontroller to enter idle mode and the IDLPD #2 instruction causes the microcontroller to enter powerdown mode.0 = disable idle and powerdownmodes 1 = enable idle and powerdown modes If your design uses idle or powerdown mode, set this bit when you program the CCBs. If it does not, clearing this bit when you program the CCBs will prevent accidental entry into idle or powerdown mode. (Chapter 14, "Special Operating Modes," discusses idle and powerdown modes.)

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after reset. The CCBs reside at addresses FF2018H (CCB0) and FF201AH (CCB1); therefore, define your CCRs at these addresses.

Figure C-11. Chip Configuration 0 (CCR0) Register (Continued)

CCR1 no direct access†

The chip configuration 1 (CCR1) register controls the clock failure detection circuitry, deferred mode, and the watchdog timer, selects the 64-Kbyte or 2-Mbyte addressing mode, and controls whether the upper 7-Kbyte section of internal ROM (FF2400–FF3FFFH) is mapped into page FFH only or into both pages FFH and 00H.

7							0
1	CFD	DM	0	WDD	REMAP	MODE64	0

Bit Number	Bit Mnemonic	Function
7	1	To guarantee proper operation, write one to this bit.
6	CFD	Clock-failure Detection This bit enables or disables the clock failure detection circuitry. 0 = disables clock-failure detection circuitry 1 = enables clock-failure detection circuitry (See “Clock Failure Detection Logic” on page 2-12.)
5	DM	Deferred Mode Enables the deferred bus-cycle mode. If the microcontroller is using a demultiplexed bus and deferred mode is enabled, a delay of 2t occurs in the first bus cycle following a chip-select output change, the first write cycle following a read cycle, and the first read cycle following a write cycle. (See “Deferred Bus-cycle Mode” on page 15-41.) 0 = deferred bus-cycle mode disabled 1 = deferred bus-cycle mode enabled
4	0	To guarantee proper operation, write zero to this bit.
3	WDD	Watchdog Timer Disable Selects whether the watchdog timer is always enabled or disabled until the first time it is cleared. If this bit is clear, the watchdog is enabled at reset, so software must clear the watchdog within 64K state times to prevent another reset. If this bit is set, the watchdog is disabled until the first time you clear it. (See “Enabling the Watchdog Timer” on page 13-12.) 0 = always enabled 1 = disabled at reset; enabled the first time it is cleared

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after reset. Locate your CCBs at addresses FF2018H (CCB0) and FF201AH (CCB1).

Figure C-12. Chip Configuration 1 (CCR1) Register

CCR1 (Continued) no direct access†

The chip configuration 1 (CCR1) register controls the clock failure detection circuitry, deferred mode, and the watchdog timer, selects the 64-Kbyte or 2-Mbyte addressing mode, and controls whether the upper 7-Kbyte section of internal ROM (FF2400–FF3FFFH) is mapped into page FFH only or into both pages FFH and 00H.

7 **0**

1	CFD	DM	0	WDD	REMAP	MODE64	0

Bit Number	Bit Mnemonic	Function
2	REMAP	Internal ROM Mapping The EA# pin controls whether accesses to FF2000–FF3FFFH are directed to internal ROM or to external memory. When EA# is low (external execution), REMAP is ignored. When EA# is high (internal execution), REMAP controls whether the upper 7-Kbyte area (FF2400–FF3FFFH) of internal ROM is mapped only into page FFH or into both pages FFH and 00H. 0 = ROM maps to page FFH only 1 = ROM maps to page FFH and page 00H (See "Remapping Internal ROM" on page 4-29.)
1	MODE64	Addressing Mode Selects 64-Kbyte or 2-Mbyte addressing. 0 = selects 2-Mbyte addressing 1 = selects 64-Kbyte addressing In 2-Mbyte mode, code can execute from almost anywhere in the address space. In 64-Kbyte mode, code can execute only from page FFH. (See "Fetching Code and Data in the 2-Mbyte and 64-Kbyte Modes" on page 4-31 for more information.)
0	0	Reserved; for compatibility with future devices, write zero to this bit.

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after reset. Locate your CCBs at addresses FF2018H (CCB0) and FF201AH (CCB1).

Figure C-12. Chip Configuration 1 (CCR1) Register (Continued)

CLKOUT_CON Address: 1F80H
Reset State: 00H

The CLKOUT control (CLKOUT_CON) register controls the output frequency on the CLKOUT pin.

7							0
—	—	—	—	—	OSC	CLK1	CLK0

Bit Number	Bit Mnemonic	Function
7:3	—	Reserved; for compatibility with future devices, write zeros to these bits.
2	OSC	This bit controls whether the CLKOUT output carries the undivided frequency, f, or the output of the programmable divider: 0 = use the divided frequency (divisor defined by CLK1:0) 1 = use the undivided frequency (CLKOUT = f)
1:0	CLK1:0	These bits specify the divisor to be used by the programmable divider: **CLK1 CLK0** 0 0 divide by 8 (CLKOUT = f/16) 0 1 divide by 4 (CLKOUT = f/8) 1 0 divide by 2 (CLKOUT = f/4) 1 1 divide by 1 (CLKOUT = f/2)

Figure C-13. CLKOUT Control (CLKOUT_CON) Register

EP_DIR Address: 1FE3H
Reset State: FFH

In I/O mode, each bit of the extended port I/O direction (EP_DIR) register controls the direction of the corresponding pin. Clearing a bit configures a pin as a complementary output; setting a bit configures a pin as either a high-impedance input or an open-drain output. (Open-drain outputs require external pull-ups.)

Any pin that is configured for its extended-address function is forced to the complementary output mode except during reset, hold, idle, powerdown.

7							0
PIN7	PIN6	PIN5	PIN4	PIN3	PIN2	PIN1	PIN0

Bit Number	Bit Mnemonic	Function
7:0	PIN7:0	Extended Address Port Pin *x* Direction Each bit controls the configuration of the corresponding pin. Clearing a bit configures a pin as a complementary output; setting a bit configures a pin as a high-impedance input or an open-drain output.

Figure C-14. Extended Port I/O Direction (EP_DIR) Register

EP_MODE

Address: 1FE1H
Reset State: FFH

Each bit of the extended port mode (EP_MODE) register controls whether the corresponding pin functions as a general-purpose I/O signal or as an extended-address or chip-select signal. EPORT pins 4–0 share package pins with the extended-address signals, A20:16, and EPORT pins 7–5 share package pins with the chip-select signals, CS2:0#. Setting a bit configures a pin as an extended-address or chip-select signal; clearing a bit configures a pin as a general-purpose I/O signal.

7							0
PIN7	PIN6	PIN5	PIN4	PIN3	PIN2	PIN1	PIN0

Bit Number	Bit Mnemonic	Function
7:0	PIN7:0	Extended Address Port Pin *x* Mode This bit determines the mode of EPORT.*x*: 0 = general-purpose I/O signal 1 = extended-address or chip-select signal

Figure C-15. Extended Port Mode (EP_MODE) Register

EP_PIN Address: 1FE7H
Reset State: XXH

Each bit of the extended port pin (EP_PIN) register reflects the current state of the corresponding pin, regardless of the pin configuration.

7							0
PIN7	PIN6	PIN5	PIN4	PIN3	PIN2	PIN1	PIN0

Bit Number	Bit Mnemonic	Function
7:0	PIN7:0	Extended Address Port Pin *x* Input This bit contains the current state of EPORT.*x*.

Figure C-16. Extended Port Input (EP_PIN) Register

EP_REG

Address: 1FE5H
Reset State: E0H

Each bit of the extended port data output (EP_REG) register contains data to be driven out by the corresponding pin. When a pin is configured as a general-purpose I/O signal (EP_MODE.*x* = 0), the result of a CPU write to EP_REG is immediately visible on the pin.

7							0
PIN7	PIN6	PIN5	PIN4	PIN3	PIN2	PIN1	PIN0

Bit Number	Bit Mnemonic	Function
7:0	PIN7:0	Extended Address Port Pin *x* Output If EPORT.*x* is to be used as an output, write the data that it is to drive out. If EPORT.*x* is to be used as an input, set this bit.

Figure C-17. Extended Port Data Output (EP_REG) Register

EPA*x*_CON
C*x* = 0–16

The EPA control (EPA*x*_CON) registers control the functions of their assigned capture/compare channels.

15							8
—	—	—	TB1	TB0	CE	M1	M0
7							**0**
MX1	MX0	RE	AD	RTS1	RTS0	ON/RT	EMC

Bit Number	Bit Mnemonic	Function
15:13	—	Reserved; always write as zeros.
12:11	TB1:0	Time Base Select Specifies the reference timer. **TB1 TB0** 0 0 timer 1 0 1 timer 2 1 0 timer 3 1 1 timer 4
10	CE	Compare Enable Determines whether the EPA channel operates in capture or compare mode. 0 = capture mode 1 = compare mode
9:8	M1:0	EPA Mode Select In capture mode, specifies the type of event that triggers an input capture. In compare mode, specifies the action that the EPA executes when the reference timer matches the event time. **M1 M0 Capture Mode Event** 0 0 no capture 0 1 capture on falling edge 1 0 capture on rising edge 1 1 capture on either edge **M1 M0 Compare Mode Action** 0 0 no output 0 1 clear output pin 1 0 set output pin 1 1 toggle output pin

Figure C-18. EPA Control (EPA*x*_CON) Registers

EPA*x*_CON (Continued)
C*x* = 0–16

The EPA control (EPA*x*_CON) registers control the functions of their assigned capture/compare channels.

15							8
—	—	—	TB1	TB0	CE	M1	M0
7							**0**
MX1	MX0	RE	AD	RTS1	RTS0	ON/RT	EMC

Bit Number	Bit Mnemonic	Function
7:6	MX1:0	Cross-coupled Mode Select If module cross-coupling is enabled, these bits specify the action that the EPA executes on the cross-coupled pin when the reference timer matches the event-time value. **MX1 MX0 Capture Mode Event** X 0 select associated pin for input X 1 select cross-coupled pin for input **MX1 MX0 Compare Mode Action** 0 0 no output 0 1 clear cross-coupled pin 1 0 set cross-coupled pin 1 1 toggle cross-coupled pin
5	RE	Re-enable Re-enable applies to the compare mode only. It allows a compare event to continue to execute each time the event-time register (EPA*x*_TIME) matches the reference timer rather than only upon the first time match. 0 = compare function is disabled after a single event 1 = compare function always enabled
4	AD	A/D Conversion Allows the EPA to start an A/D conversion that has been previously set up in the A/D command registers. To use this feature, you must select the EPA as the conversion source in the AD_COMMAND register. 0 = causes no A/D action 1 = EPA capture or compare event triggers an A/D conversion
3:2	RTS1:0	Reset Timer Select Selects the timer that is to be reset if the RT bit (EPA*x*_CON.1) is set. **RTS1 RTS0** 0 0 select timer 1 for reset 0 1 select timer 2 for reset 1 0 select timer 3 for reset 1 1 select timer 4 for reset

Figure C-18. EPA Control (EPA*x*_CON) Registers (Continued)

EPA*x*_CON (Continued)
C*x* = 0–16

The EPA control (EPA*x*_CON) registers control the functions of their assigned capture/compare channels.

15							8
—	—	—	TB1	TB0	CE	M1	M0
7							**0**
MX1	MX0	RE	AD	RTS1	RTS0	ON/RT	EMC

Bit Number	Bit Mnemonic	Function
1	ON/RT	Overwrite New/Reset Timer The ON/RT bit functions as overwrite new in capture mode and reset timer in compare mode. **In Capture Mode (ON):** An overrun error is generated when an input capture occurs while the event-time register (EPA*x*_TIME) and its buffer are both full. When an overrun occurs, the ON bit determines whether old data is overwritten or new data is ignored: 0 = ignores new data 1 = overwrites old data in the buffer **In Compare Mode (RT):** 0 = disables the reset function 1 = resets the RTS-selected timer
0	EMC	Enable Module Concatenation Used in conjunction with TxCONTROL.8, this bit allows an EPA channel to enable an adjacent channel, providing a 32-bit time base. 0 = disable 1 = enable concatenation

Figure C-18. EPA Control (EPA*x*_CON) Registers (Continued)

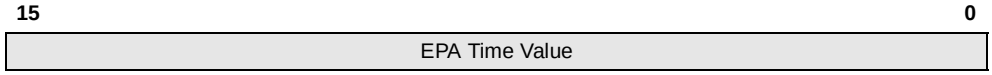
Table C-7. EPA*x*_CON Addresses and Reset States

Register	Address	Reset State
EPA0_CON	1F5CH	0000H
EPA1_CON	1F58H	0000H
EPA2_CON	1F54H	0000H
EPA3_CON	1F50H	0000H
EPA4_CON	1F4CH	0000H
EPA5_CON	1F48H	0000H
EPA6_CON	1F44H	0000H
EPA7_CON	1F40H	0000H
EPA8_CON	1F3CH	0000H

Register	Address	Reset State
EPA9_CON	1F38H	0000H
EPA10_CON	1F34H	0000H
EPA11_CON	1F30H	0000H
EPA12_CON	1F2CH	0000H
EPA13_CON	1F28H	0000H
EPA14_CON	1F24H	0000H
EPA15_CON	1F20H	0000H
EPA16_CON	1F1CH	0000H

EPA*x*_TIME
***x* = 0–16**

The EPA time (EPA*x*_TIME) registers are the event-time registers for the EPA channels. In capture mode, the value of the reference timer is captured in EPA*x*_TIME when an input transition occurs. Each event-time register is buffered, allowing the storage of two capture events at once. In compare mode, the EPA triggers a compare event when the reference timer matches the value in EPA*x*_TIME. EPA*x*_TIME is not buffered for compare mode.



Bit Number	Function
15:0	EPA Time Value When an EPA channel is configured for capture mode, this register contains the value of the reference timer when the specified event occurred. When an EPA channel is configured for compare mode, write the compare event time to this register.

Figure C-19. EPA Time (EPA*x*_TIME) Registers

Table C-8. EPA*x*_TIME Addresses and Reset States

Register	Addr.	Reset State
EPA0_TIME	1F5EH	XXXXH
EPA1_TIME	1F5AH	XXXXH
EPA2_TIME	1F56H	XXXXH
EPA3_TIME	1F52H	XXXXH
EPA4_TIME	1F4EH	XXXXH
EPA5_TIME	1F4AH	XXXXH
EPA6_TIME	1F46H	XXXXH
EPA7_TIME	1F42H	XXXXH
EPA8_TIME	1F3EH	XXXXH

Register	Addr.	Reset State
EPA9_TIME	1F3AH	XXXXH
EPA10_TIME	1F36H	XXXXH
EPA11_TIME	1F32H	XXXXH
EPA12_TIME	1F2EH	XXXXH
EPA13_TIME	1F2AH	XXXXH
EPA14_TIME	1F26H	XXXXH
EPA15_TIME	1F22H	XXXXH
EPA16_TIME	1F1EH	XXXXH

INT_MASK

Address: 0008H
Reset State: 00H

The interrupt mask (INT_MASK) register enables or disables (masks) individual interrupt requests. (The EI and DI instructions enable and disable servicing of all maskable interrupts.) INT_MASK is the low byte of the processor status word (PSW). PUSHF or PUSHA saves the contents of this register onto the stack and then clears this register. Interrupt calls cannot occur immediately following a push instruction. POPF or POPA restores it.

7 **0**

SDU	EXTINT	RI1	TI1	AD	EPA*x*_OVR	RI0	TI0

Bit Number	Function
7:0	Setting a bit enables the corresponding interrupt.

Bit Mnemonic	Interrupt Description
SDU	Serial Debug Unit Interrupt
EXTINT	External Interrupt Pin
RI1	SIO1 Receive
TI1	SIO1 Transmit
AD	A/D Conversion Complete
EPA*x*_OVR	EPA Channel 3–16 Overrun
RI0	SIO0 Receive
TI0	SIO0 Transmit

Figure C-20. Interrupt Mask (INT_MASK) Register

INT_MASK1

Address: 0013H
Reset State: 00H

The interrupt mask 1 (INT_MASK1) register enables or disables (masks) individual interrupt requests. (The EI and DI instructions enable and disable servicing of all maskable interrupts.) INT_MASK1 can be read from or written to as a byte register. PUSHA saves this register on the stack and POPA restores it.

7							0
NMI	STACK	PIH0_PTS	PIH0_INT	PIH1_PTS	PIH1_INT	SSIO1	SSIO0

Bit Number	Function
7:0	Setting a bit enables the corresponding interrupt. **Bit Mnemonic** — **Interrupt Description** NMI† — Nonmaskable Interrupt STACK† — Stack Overflow Error PIH0_PTS — PIH0 PTS Service Request PIH0_INT — PIH0 Interrupt Request PIH1_PTS — PIH1 PTS Service Request PIH1_INT — PIH1 Interrupt Request SSIO1 — SSIO1 Transfer SSIO0 — SSIO0 Transfer † The NMI and stack overflow interrupts are always enabled. These nonfunctional mask bits exist for design symmetry with the INT_PEND1 register. Always write zeros to these bits.

Figure C-21. Interrupt Mask 1 (INT_MASK1) Register

INT_PEND

Address: 0009H
Reset State: 00H

When hardware detects an interrupt request, it sets the corresponding bit in the interrupt pending (INT_PEND or INT_PEND1) registers. When the vector is taken, the hardware clears the pending bit. Software can generate an interrupt by setting the corresponding interrupt pending bit.

7 **0**

SDU	EXTINT	RI1	TI1	AD	EPA*x*_OVR	RI0	TI0

Bit Number	Function
7:0	Any set bit indicates that the corresponding interrupt is pending. The interrupt bit is cleared when processing transfers to the corresponding interrupt vector. **Bit Mnemonic** — **Interrupt Description** SDU — Serial Debug Unit Interrupt EXTINT — External Interrupt Pin RI1 — SIO1 Receive TI1 — SIO1 Transmit AD — A/D Conversion Complete EPA*x*_OVR — EPA Channel 3–16 Overrun† RI0 — SIO0 Receive TI0 — SIO0 Transmit † It is not possible to determine which EPA channel caused the EPA*x*_OVR interrupt request.

Figure C-22. Interrupt Pending (INT_PEND) Register

INT_PEND1 Address: 0012H
Reset State: 00H

When hardware detects an interrupt request, it sets the corresponding bit in the interrupt pending (INT_PEND or INT_PEND1) registers. When the vector is taken, the hardware clears the pending bit. Software can generate an interrupt by setting the corresponding interrupt pending bit.

7							0
NMI	STACK	PIH0_PTS	PIH0_INT	PIH1_PTS	PIH1_INT	SSIO1	SSIO0

Bit Number	Function
7:0	Any set bit indicates that the corresponding interrupt is pending. The interrupt bit is cleared when processing transfers to the corresponding interrupt vector.

Bit Mnemonic	Interrupt Description
NMI	Nonmaskable Interrupt
STACK	Stack Overflow Error
PIH0_PTS	PIH0 PTS Service Request
PIH0_INT	PIH0 Interrupt Request
PIH1_PTS	PIH1 PTS Service Request
PIH1_INT	PIH1 Interrupt Request
SSIO1	SSIO1 Transfer
SSIO0	SSIO0 Transfer

Figure C-23. Interrupt Pending 1 (INT_PEND1) Register

IRAM_CON Address: 1FE0H
Reset State: 00H

The internal RAM control (IRAM_CON) register has two functions related to memory accesses. The IRAM bit allows you to control access to locations 000400–000FFFH and FF0400–FF05FFFH. The EA_STAT bit allows you to determine the status of the EA# pin, which controls access to locations FF2000–FF3FFFH. (IRAM_CON is a memory-mapped SFR. It must be accessed using indirect or indexed addressing; it cannot be windowed.)

7							0
EA_STAT	IRAM	—	—	—	—	—	—

Bit Number	Bit Mnemonic	Function
7	EA_STAT	EA# Status: This **read-only** bit contains the complement of the EA# pin, which controls whether accesses to locations FF2000–FF3FFFH are directed to the internal ROM or to external memory. 0 = the EA# pin is inactive (accesses are directed to the ROM) 1 = the EA# pin is active (accesses are directed to external memory) (“Remapping Internal ROM” on page 4-29 describes additional options for OTPROM access.)
6	IRAM	Internal RAM Control: This bit controls whether accesses to locations 000400–000FFFH and FF0400–FF05FFFH are directed to internal code RAM or to external memory. 0 = use the internal code RAM 1 = use external memory
5:0	—	Reserved; for compatibility with future devices, write zeros to these bits.

Figure C-24. Internal RAM Control (IRAM_CON) Register

ONES_REG Address: 02H
Reset State: FFFFH

The two-byte ones register (ONES_REG) is always equal to FFFFH. It is useful as a fixed source of all ones for comparison operations.

15	0
Ones	

Bit Number	Function
15:0	Ones These bits are always equal to FFFFH.

Figure C-25. Ones Register (ONES_REG)

OS*x*_CON
***x* = 0–7**

The output/simulcapture control (OSx_CON) registers control the functions of their assigned output/simulcapture channels.

15							8
—	—	—	TB1	TB0	CE	M1	M0
7							**0**
MX1	MX0	RE	AD	CS1	CS0	SE	EMC

Bit Number	Bit Mnemonic	Function
15:13	—	Reserved; always write as zeros.
12:11	TB1:0	Time Base Select Specifies the reference timer. **TB1 TB0** 0 0 timer 1 0 1 timer 2 1 0 timer 3 1 1 timer 4
10	CE	Compare Enable Enables the compare function of the output/simulcapture channel. 0 = compare function disabled 1 = compare function enabled
9:8	M1:0	Mode Select Specifies the action that the EPA executes when the reference timer matches the event-time value. **M1 M0** 0 0 no output 0 1 clear output pin 1 0 set output pin 1 1 toggle output pin
7:6	MX1:0	Cross-coupled Mode Select If module cross-coupling is enabled, this bit specifies the action that the EPA executes on the cross-coupled pin when the reference timer matches the event-time value. **MX1 MX0** 0 0 no output 0 1 clear cross-coupled pin 1 0 set cross-coupled pin 1 1 toggle cross-coupled pin

Figure C-26. Output/Simulcapture *x* Control (OS*x*_CON) Register

OS*x*_CON (Continued)
***x* = 0–7**

The output/simulcapture control (OSx_CON) registers control the functions of their assigned output/simulcapture channels.

15							8
—	—	—	TB1	TB0	CE	M1	M0
7							**0**
MX1	MX0	RE	AD	CS1	CS0	SE	EMC

Bit Number	Bit Mnemonic	Function
5	RE	Re-enable Allows a compare event to continue to execute each time the reference timer matches the event-time register value, rather than only upon the first match. 0 = compare function will drive the output only once 1 = compare function always enabled
4	AD	A/D Conversion Allows the EPA to start an A/D conversion that has been previously set up in the A/D command registers. To use this feature, you must select the EPA as the conversion source in the AD_COMMAND register. 0 = causes no A/D action 1 = OS compare event triggers an A/D conversion
3:2	CS1:0	Simulcapture Timer Select Selects the timer whose value is to be captured when the output event is generated. **CS1 CS0 Reference Timer** 0 0 timer 1 0 1 timer 2 1 0 timer 3 1 1 timer 4
1	SE	Simulcapture Enable Enables the simulcapture function of the output/simulcapture channel. 0 = simulcapture function disabled 1 = simulcapture function enabled (if compare occurs)
0	EMC	Enable Module Concatenation Used in conjunction with TxCONTROL.8, this bit allows an output/simul-capture channel to enable an adjacent channel, providing a 32-bit time base. 0 =disable 1 =enable concatenation

Figure C-26. Output/Simulcapture *x* Control (OS*x*_CON) Register (Continued)

Table C-9. OS*x*_CON Addresses and Reset Values

Register	Address	Reset Value
OS0_CON	1EFCH	0000H
OS1_CON	1EF8H	0000H
OS2_CON	1EF4H	0000H
OS3_CON	1EF0H	0000H
OS4_CON	1EECH	0000H
OS5_CON	1EE8H	0000H
OS6_CON	1EE4H	0000H
OS7_CON	1EE0H	0000H

OS*x*_TIME
***x* = 0–7**

The output/simulcapture time registers (OS*x*_TIME) are the event-time registers for the output/simulcapture channels. The EPA triggers a compare event when the reference timer matches the value in OS*x*_TIME. If simulcapture is enabled, the EPA simultaneously captures the value of the specified timer into this register, overwriting the previous value.

15 **0**

OS Timer Value

Bit Number	Function
15:0	OS Timer Value After programming the OS*x*_CON register, write the compare event time to this register. The EPA generates the programmed event when the value of the reference timer matches the event-time value in this register. If simulcapture is enabled, the EPA simultaneously captures the value of the specified timer into this register, overwriting the event-time value.

Figure C-27. Output Simulcapture *x* Time (OS*x*_TIME) Register

Table C-10. OS*x*_TIME Addresses and Reset Values

Register	Address	Reset Value
OS0_TIME	1EFEH	XXXXH
OS1_TIME	1EFAH	XXXXH
OS2_TIME	1EF6H	XXXXH
OS3_TIME	1FE0H	XXXXH
OS4_TIME	1EEEH	XXXXH
OS5_TIME	1EEAH	XXXXH
OS6_TIME	1EE6H	XXXXH
OS7_TIME	1EE2H	XXXXH

P*x*_DIR
***x* = 2, 5, 7–12**

Each pin of port *x* can operate as a complementary output, a high-impedance input, or an open-drain output. The port *x* I/O direction (P*x*_DIR) register determines the configuration for each port *x* pin. When a port pin is configured as a complementary output, the microcontroller drives the signal high or low. When a port pin is configured as a high-impedance input or an open-drain output, the microcontroller drives the signal low or floats it.

	7							0
***x* = 2, 5, 7–9, 11 (ea)**	PIN7	PIN6	PIN5	PIN4	PIN3	PIN2	PIN1	PIN0
	7							0
***x* = 10 (ea)**	—	—	PIN5	PIN4	PIN3	PIN2	PIN1	PIN0
	7							0
***x* = 12 (ea)**	—	—	—	PIN4	PIN3	PIN2	PIN1	PIN0

Bit Number	Bit Mnemonic	Function
7:0†	PIN7:0	Port *x* Pin *y* Direction Each bit controls the configuration of the corresponding pin. Clearing a bit configures a pin as a complementary output; setting a bit configures a pin as a high-impedance input or an open-drain output.

† The bits shown as dashes (—) are reserved; for compatibility with future devices, write ones to these bits.

Figure C-28. Port *x* I/O Direction (P*x*_DIR) Register (Continued)

Table C-11. P*x*_DIR Addresses and Reset States

Register	Address	Reset State
P2_DIR	1FD2H	7FH
P5_DIR	1FF3H	FFH
P7_DIR	1FCAH	FFH
P8_DIR	1FCBH	FFH
P9_DIR	1FC2H	FFH
P10_DIR	1FC3H	3FH
P11_DIR	1FBAH	FFH
P12_DIR	1FEAH	1FH

P*x*_MODE
***x* = 2, 5, 7–12 ea**

Each bit of the port *x* mode (P*x*_MODE) register controls whether the corresponding pin functions as a general-purpose I/O signal or as a special-function signal.

	7							0
***x* = 2, 5, 7–9, 11**	PIN7	PIN6	PIN5	PIN4	PIN3	PIN2	PIN1	PIN0
	7							0
***x* = 10**	—	—	PIN5	PIN4	PIN3	PIN2	PIN1	PIN0
	7							0
***x* = 12**	—	—	—	PIN4	PIN3	PIN2	PIN1	PIN0

Bit Number	Bit Mnemonic	Function
7:0†	PIN7:0	Port *x* Pin *y* Mode This bit determines the mode of the corresponding port pin: 0 = general-purpose I/O signal 1 = special-function signal The following table lists the special-function signals for each pin.

† The bits shown as dashes (—) are reserved; for compatibility with future devices, write zeros to these bits.

Figure C-29. Port *x* Mode (P*x*_MODE) Register (Continued)

Table C-12. P*x*_MODE Addresses and Reset States

Register	Address	Reset State
P2_MODE	1FD0H	80H
P5_MODE	1FF1H	†
P7_MODE	1FC8H	00H
P8_MODE	1FC9H	00H
P9_MODE	1FC0H	00H
P10_MODE	1FC1H	00H
P11_MODE	1FB8H	00H
P12_MODE	1FE8H	00H

† Reset state is 80H if the EA# pin is high or A9H if EA# is low.

Table C-13. Special-function Signals for Ports 2, 5, 7–12

Port 2	
Pin	**Special-function Signal**
P2.0	TXD0
P2.1	RXD0
P2.2	EXTINT
P2.3	TXD1
P2.4	RXD1
P2.5	HOLD#
P2.6	HLDA#/ONCE#
P2.7	CLKOUT

Port 5	
Pin	**Special-function Signal**
P5.0	ALE
P5.1	INST
P5.2	WR#/WRL#
P5.3	RD#
P5.4	BREQ#
P5.5	BHE#/WRH#
P5.6	READY
P5.7	RPD

Port 7	
Pin	**Special-function Signal**
P7.0	EPA0/T1CLK
P7.1	EPA1/T1RST
P7.2	EPA2/T2CLK
P7.3	EPA3/T2RST
P7.4	EPA4/T3CLK
P7.5	EPA5/T3RST
P7.6	EPA6/T4CLK
P7.7	EPA7/T4RST

Port 8	
Pin	**Special-function Signal**
P8.0	EPA8
P8.1	EPA9
P8.2	EPA10
P8.3	EPA11
P8.4	EPA12
P8.5	EPA13
P8.6	EPA14
P8.7	EPA15

Port 9	
Pin	**Special-function Signal**
P9.0	OS0
P9.1	OS1
P9.2	OS2
P9.3	OS3
P9.4	OS4
P9.5	OS5
P9.6	OS6
P9.7	OS7

Port 10	
Pin	**Special-function Signal**
P10.0	SC0
P10.1	SD0
P10.2	SC1
P10.3	SD1
P10.4	EPA16
P10.5	—

Port 11	
Pin	**Special-function Signal**
P11.0	PWM0
P11.1	PWM1
P11.2	PWM2
P11.3	PWM3
P11.4	PWM4
P11.5	PWM5
P11.6	PWM6
P11.7	PWM7

Port 12	
Pin	**Special-function Signal**
P12.0	—
P12.1	—
P12.2	—
P12.3	—
P12.4	—

P*x*_PIN
***x* = 2–5, 7–12**

Each bit of the port *x* pin (P*x*_PIN) register reflects the current state of the corresponding pin, regardless of the pin configuration.

	7							0
***x* = 2–5, 7–9, 11**	PIN7	PIN6	PIN5	PIN4	PIN3	PIN2	PIN1	PIN0
	7							**0**
***x* = 10**	—	—	PIN5	PIN4	PIN3	PIN2	PIN1	PIN0
	7							**0**
***x* = 12**	—	—	—	PIN4	PIN3	PIN2	PIN1	PIN0

Bit Number	Bit Mnemonic	Function
7:0†	PIN7:0	Port *x* Pin *y* Input Value This bit contains the current state of P*x.y*.

† The bits shown as dashes (—) are reserved; their values are undefined.

Figure C-30. Port *x* Pin Input (P*x*_PIN) Register (Continued)

Table C-14. P*x*_PIN Addresses and Reset States

Register	Address	Reset State
P2_PIN	1FD6H	XXH
P3_PIN	1FFEH	XXH
P4_PIN	1FFFH	XXH
P5_PIN	1FF7H	XXH
P7_PIN	1FCEH	XXH
P8_PIN	1FCFH	XXH
P9_PIN	1FC6H	XXH
P10_PIN	1FC7H	XXH
P11_PIN	1FBEH	XXH
P12_PIN	1FEEH	XXH

P*x*_REG
***x* = 2–5, 7–12**

For an input, set the corresponding port *x* data output (P*x*_REG) register bit.

For an output, write the data to be driven out by each pin to the corresponding bit of P*x*_REG. When a pin is configured as a general-purpose I/O signal (P*x*_MODE.*y* = 0), the result of a CPU write to P*x*_REG is immediately visible on the pin. When a pin is configured as a special-function signal (P*x*_MODE.*y* = 1), the associated on-chip peripheral or off-chip component controls the pin. The CPU can still write to P*x*_REG, but the pin is unaffected until it is switched back to its standard I/O function.

This feature allows software to configure a pin as a general-purpose I/O signal (clear P*x*_MODE.*y*), initialize or overwrite the pin value, then configure the pin as a special-function signal (set P*x*_MODE.*y*). In this way, initialization, fault recovery, exception handling, etc., can be done without changing the operation of the associated peripheral.

	7							0
***x* = 2–5, 7–9, 11**	PIN7	PIN6	PIN5	PIN4	PIN3	PIN2	PIN1	PIN0
	7							**0**
***x* = 10**	—	—	PIN5	PIN4	PIN3	PIN2	PIN1	PIN0
	7							**0**
***x* = 12**	—	—	—	PIN4	PIN3	PIN2	PIN1	PIN0

Bit Number	Bit Mnemonic	Function
7:0†	PIN7:0	Port *x* Pin *y* Data Output **For I/O Mode (P*x*_MODE.*y* = 0)** When a port pin is configured as a complementary output (P*x*_DIR.*y* = 0), setting the corresponding port data output bit drives a one on the pin and clearing the corresponding port data output bit drives a zero on the pin. When a port pin is configured as a high-impedance input or an open-drain output (P*x*_DIR.*y* = 1), clearing the corresponding port data output bit drives a zero on the pin and setting the corresponding port data output bit floats the pin, making it available as a high-impedance input. **For Special-function Mode (P*x*_MODE.*y* = 1)** When a port pin is configured as an output (either complementary or open-drain), the corresponding port data output bit value is immaterial because the corresponding on-chip peripheral or system function controls the pin. To configure a pin as a high-impedance input, set both the P*x*_DIR and P*x*_REG bits.

† The bits shown as dashes (—) are reserved; for compatibility with future devices, write zeros to these bits.

Figure C-31. Port *x* Data Output (P*x*_REG) Register (Continued)

Table C-15. P*x*_REG Addresses and Reset States

Register	Address	Reset State
P2_REG	1FD4H	FFH
P3_REG	1FFCH	FFH
P4_REG	1FFDH	FFH
P5_REG	1FF5H	FFH
P7_REG	1FCCH	FFH
P8_REG	1FCDH	FFH
P9_REG	1FC4H	FFH
P10_REG	1FC5H	3FH
P11_REG	1FBCH	FFH
P12_REG	1FECH	1FH

P34_DRV

Address: 1FF4H
Reset State: 00H

Bits 7 and 6 of the port 3/4 complementary enable (P34_DRV) register control whether ports 3 and 4, respectively, are configured as complementary or open-drain. Setting a bit configures a port as complementary; clearing a bit configures a port as open-drain. These bits affect port operation only in I/O mode.

7							0
P3DRV	P4DRV	—	—	—	—	—	—

Bit Number	Bit Mnemonic	Function
7	P3DRV	Port 3 I/O Mode This bit controls whether port 3 is configured as complementary or open-drain outputs. 0 = selects open-drain operation 1 = selects complementary operation
6	P4DRV	Port 4 I/O Mode This bit controls whether port 4 is configured as complementary or open-drain outputs. 0 = selects open-drain operation 1 = selects complementary operation
5:0	—	Reserved; for compatibility with future devices, write zeros to these bits.

Figure 17-6. Port 3/4 Push-pull Enable (P34_DRV) Register

PIH0_INT_MASK

Address: 1E98H
Reset State: 0000H

The PIH0 interrupt mask (PIH0_INT_MASK) register enables or disables (masks) individual interrupt requests to peripheral interrupt handler 0. (The EI and DI instructions enable and disable servicing of all maskable interrupts.)

15							8
EPA15	EPA14	EPA13	EPA12	EPA11	EPA10	EPA9	EPA8
7							**0**
EPA7	EPA6	EPA5	EPA4	EPA3	EPA2	EPA1	EPA0

Bit Number	Function
15:0	Setting a bit enables the corresponding interrupt request to peripheral interrupt handler 0. **Bit Mnemonic** **Interrupt Description** EPA15:0 EPA Capture/Compare Channels 0–15

Figure 17-7. PIH0 Interrupt Mask (PIH0_INT_MASK) Register

PIH1_INT_MASK

Address: 1EA8H
Reset State: 0000H

The PIH1 interrupt mask (PIH1_INT_MASK) register enables or disables (masks) individual interrupt requests to the peripheral interrupt handler 1. (The EI and DI instructions enable and disable servicing of all maskable interrupts.)

15							8
EPA16	OS7	OS6	OS5	OS4	OS3	OS2	OS1
7							**0**
OS0	OVRTM1	OVRTM2	OVRTM3	OVRTM4	OVR0	OVR1	OVR2

Bit Number	Function
15:0	Setting a bit enables the corresponding interrupt request to peripheral interrupt handler 1. **Bit Mnemonic** — **Interrupt Description** EPA16 — EPA Capture/Compare Channel 16 OS7:0 — Output Simulcapture Channel 0–7 OVRTM1:4 — Timer 1–4 Overflow/Underflow OVR0:2 — EPA Channel 0–2 Capture Overrun

Figure 17-8. PIH1 Interrupt Mask (PIH1_INT_MASK) Register

PIH0_INT_PEND

Address: 1E9AH
Reset State: 0000H

When the peripheral interrupt handler 0 detects an interrupt request, it sets the corresponding bit in the interrupt pending (PIH0_INT_PEND) register. When the vector is taken, the hardware clears the pending bit. Software can generate an interrupt by setting the corresponding interrupt pending bit.

15							8
EPA15	EPA14	EPA13	EPA12	EPA11	EPA10	EPA9	EPA8
7							**0**
EPA7	EPA6	EPA5	EPA4	EPA3	EPA2	EPA1	EPA0

Bit Number	Function
15:0	Any set bit indicates that the corresponding PIH0 interrupt is pending. The interrupt bit is cleared when processing transfers to the corresponding interrupt vector. **Bit Mnemonic** **Interrupt Description** EPA15:0 EPA Capture/Compare Channels 0–15

Figure 17-9. PIH0 Interrupt Pending (PIH0_INT_PEND) Register

PIH1_INT_PEND

Address: 1EAAH
Reset State: 0000H

When the peripheral interrupt handler 1 detects an interrupt request, it sets the corresponding bit in the interrupt pending (PIH1_INT_PEND) register. When the vector is taken, the hardware clears the pending bit. Software can generate an interrupt by setting the corresponding interrupt pending bit.

15							8
EPA16	OS7	OS6	OS5	OS4	OS3	OS2	OS1
7							**0**
OS0	OVRTM1	OVRTM2	OVRTM3	OVRTM4	OVR0	OVR1	OVR2

Bit Number	Function
15:0	Any set bit indicates that the corresponding PIH1 interrupt is pending. The interrupt bit is cleared when processing transfers to the corresponding interrupt vector. **Bit Mnemonic** — **Interrupt Description** EPA16 — EPA Capture/Compare Channel 16 OS7:0 — Output Simulcapture Channel 0–7 OVRTM1:4 — Timer 1–4 Overflow/Underflow OVR0:2 — EPA Channel 0–2 Capture Overrun

Figure 17-10. PIH1 Interrupt Pending (PIH1_INT_PEND) Register

PIH0_PTSSEL Address: 1E96H
Reset State: 0000H

The PTS select (PIH0_PTSSEL) register for the peripheral interrupt handler 0 selects either a PTS service request or a standard interrupt service request for the corresponding interrupt. Setting a bit selects a PTS service request; clearing a bit selects a standard interrupt service request. In PTS modes that use the PTSCOUNT register, hardware clears the corresponding bit in both the PIH0_PTSSEL and PTSSEL registers when PTSCOUNT reaches zero. The end-of-PTS interrupt service routine must reset the PIH0_PTSSEL bit to re-enable the PTS channel.

15							8
EPA15	EPA14	EPA13	EPA12	EPA11	EPA10	EPA9	EPA8
7							**0**
EPA7	EPA6	EPA5	EPA4	EPA3	EPA2	EPA1	EPA0

Bit Number	Function
15:0	Setting a bit causes the peripheral interrupt handler 0 (PIH0) to generate the PIH0_PTS interrupt request when the corresponding interrupt is generated and enabled. The PTS interrupt vector locations are as follows:

Bit Mnemonic	Interrupt	PTS Vector
EPA15	EPA Capture/Compare Channel 15	FF20FEH
EPA14	EPA Capture/Compare Channel 14	FF20FAH
EPA13	EPA Capture/Compare Channel 13	FF20F6H
EPA12	EPA Capture/Compare Channel 12	FF20F2H
EPA11	EPA Capture/Compare Channel 11	FF20EEH
EPA10	EPA Capture/Compare Channel 10	FF20EAH
EPA9	EPA Capture/Compare Channel 9	FF20E6H
EPA8	EPA Capture/Compare Channel 8	FF20E2H
EPA7	EPA Capture/Compare Channel 7	FF20DEH
EPA6	EPA Capture/Compare Channel 6	FF20DAH
EPA5	EPA Capture/Compare Channel 5	FF20D6H
EPA4	EPA Capture/Compare Channel 4	FF20D2H
EPA3	EPA Capture/Compare Channel 3	FF20CEH
EPA2	EPA Capture/Compare Channel 2	FF20CAH
EPA1	EPA Capture/Compare Channel 1	FF20C6H
EPA0	EPA Capture/Compare Channel 0	FF20C2H

Figure 17-11. PIH0 PTS Select (PIH0_PTSSEL) Register

PIH1_PTSSEL

Address: 1EA6H
Reset State: 0000H

The PTS select register for the peripheral interrupt handler 1 (PIH1_PTSSEL) selects either a PTS service request or a standard interrupt service request for the corresponding interrupt. Setting a bit selects a PTS service request; clearing a bit selects a standard interrupt service request. In PTS modes that use the PTSCOUNT register, hardware clears the corresponding bit in both the PIH1_PTSSEL and PTSSEL registers when PTSCOUNT reaches zero. The end-of-PTS interrupt service routine must reset the PIH1_PTSSEL bit to re-enable the PTS channel.

15							8
EPA16	OS7	OS6	OS5	OS4	OS3	OS2	OS1
7							**0**
OS0	OVRTM1	OVRTM2	OVRTM3	OVRTM4	OVR0	OVR1	OVR2

Bit Number	Function
15:0	Setting a bit causes the peripheral interrupt handler 1(PIH1) to generate the PIH1_PTS interrupt request when the corresponding interrupt is generated and enabled. The PTS interrupt vector locations are as follows:

Bit Mnemonic	Interrupt	PTS Vector
EPA16	EPA Capture/Compare Channel 16	FF213EH
OS7	Output Simulcapture Channel 7	FF213AH
OS6	Output Simulcapture Channel 6	FF2136H
OS5	Output Simulcapture Channel 5	FF2132H
OS4	Output Simulcapture Channel 4	FF212EH
OS3	Output Simulcapture Channel 3	FF212AH
OS2	Output Simulcapture Channel 2	FF2126H
OS1	Output Simulcapture Channel 1	FF2122H
OS0	Output Simulcapture Channel 0	FF211EH
OVRTM1	Timer 1 Overflow/Underflow	FF211AH
OVRTM2	Timer 2 Overflow/Underflow	FF2116H
OVRTM3	Timer 3 Overflow/Underflow	FF2112H
OVRTM4	Timer 4 Overflow/Underflow	FF210EH
OVR0	EPA Channel 0 Capture Overrun	FF210AH
OVR1	EPA Channel 1 Capture Overrun	FF2106H
OVR2	EPA Channel 2 Capture Overrun	FF2102H

Figure 17-12. PIH1 PTS Select (PIH1_PTSSEL) Register

PIH0_PTSSRV Address: 1E94H
Reset State: 0000H

The PTS service (PIH0_PTSSRV) register for peripheral interrupt handler 0 is used by the hardware to indicate that the final PTS interrupt has been serviced by the PTS routine. When PTSCOUNT reaches zero, hardware clears the corresponding PIH0_PTSSEL bit and sets the PIH0_PTSSRV bit, which requests the end-of-PTS interrupt. When the end-of-PTS interrupt is called, hardware clears the PIH0_PTSSRV bit. The end-of-PTS interrupt service routine must set the PIH0_PTSSEL bit to re-enable the PTS channel.

15							8
EPA15	EPA14	EPA13	EPA12	EPA11	EPA10	EPA9	EPA8
7							**0**
EPA7	EPA6	EPA5	EPA4	EPA3	EPA2	EPA1	EPA0

Bit Number	Function
15:0	A bit is set by hardware to request an end-of-PTS interrupt for the corresponding interrupt through its standard interrupt vector. The standard interrupt vector locations are as follows:

Bit Mnemonic	Interrupt	Standard Vector
EPA15	EPA Capture/Compare Channel 15	FF20FCH
EPA14	EPA Capture/Compare Channel 14	FF20F8H
EPA13	EPA Capture/Compare Channel 13	FF20F4H
EPA12	EPA Capture/Compare Channel 12	FF20F0H
EPA11	EPA Capture/Compare Channel 11	FF20ECH
EPA10	EPA Capture/Compare Channel 10	FF20E8H
EPA9	EPA Capture/Compare Channel 9	FF20E4H
EPA8	EPA Capture/Compare Channel 8	FF20E0H
EPA7	EPA Capture/Compare Channel 7	FF20DCH
EPA6	EPA Capture/Compare Channel 6	FF20D8H
EPA5	EPA Capture/Compare Channel 5	FF20D4H
EPA4	EPA Capture/Compare Channel 4	FF20D0H
EPA3	EPA Capture/Compare Channel 3	FF20CCH
EPA2	EPA Capture/Compare Channel 2	FF20C8H
EPA1	EPA Capture/Compare Channel 1	FF20C4H
EPA0	EPA Capture/Compare Channel 0	FF20C0H

Figure 17-13. PIH0 PTS Service (PIH0_PTSSRV) Register

PIH1_PTSSRV

Address: 1EA4H
Reset State: 0000H

The PTS service (PIH1_PTSSRV) register for peripheral interrupt handler 1 is used by the hardware to indicate that the final PTS interrupt has been serviced by the PTS routine. When PTSCOUNT reaches zero, hardware clears the corresponding PIH1_PTSSEL bit and sets the PIH1_PTSSRV bit, which requests the end-of-PTS interrupt. When the end-of-PTS interrupt is called, hardware clears the PIH1_PTSSRV bit. The end-of-PTS interrupt service routine must set the PIH1_PTSSEL bit to re-enable the PTS channel.

15							8
EPA16	OS7	OS6	OS5	OS4	OS3	OS2	OS1
7							**0**
OS0	OVRTM1	OVRTM2	OVRTM3	OVRTM4	OVR0	OVR1	OVR2

Bit Number	Function
15:0	A bit is set by hardware to request an end-of-PTS interrupt for the corresponding interrupt through its standard interrupt vector. The standard interrupt vector locations are as follows:

Bit Mnemonic	Interrupt	Standard Vector
EPA16	EPA Capture/Compare Channel 16	FF213CH
OS7	Output Simulcapture Channel 7	FF2138H
OS6	Output Simulcapture Channel 6	FF2134H
OS5	Output Simulcapture Channel 5	FF2130H
OS4	Output Simulcapture Channel 4	FF212CH
OS3	Output Simulcapture Channel 3	FF2128H
OS2	Output Simulcapture Channel 2	FF2124H
OS1	Output Simulcapture Channel 1	FF2120H
OS0	Output Simulcapture Channel 0	FF211CH
OVRTM1	Timer 1 Overflow/Underflow	FF2118H
OVRTM2	Timer 2 Overflow/Underflow	FF2114H
OVRTM3	Timer 3 Overflow/Underflow	FF2110H
OVRTM4	Timer 4 Overflow/Underflow	FF210CH
OVR0	EPA Channel 0 Capture Overrun	FF2108H
OVR1	EPA Channel 1 Capture Overrun	FF2104H
OVR2	EPA Channel 2 Capture Overrun	FF2100H

Figure 17-14. PIH1 PTS Service (PIH1_PTSSRV) Register

PIH*x*_VEC_BASE
***x* = 0–1**

Address: 1E92H, 1EA2H
Reset State: XXXXH

The peripheral interrupt handler *x* vector base-address (PIH*x*_VEC_BASE) registers contain address bits 6–15 of the interrupt vector. Your code must initialize this register before enabling any PIH interrupt sources.

15							8
VA15	VA14	VA13	VA12	VA11	VA10	VA9	VA8
7							**0**
VA7	VA6	—	—	—	—	—	—

Bit Number	Bit Mnemonic	Function
15:6	VA15:6	Peripheral Interrupt Handler Vector Base-address Bits 6 through 15 This register contains address bits 6 through 15 for the PIH*x* interrupt vectors. Always initialize these registers to the following values: PIH0_VEC_BASE = 20C0H PIH1_VEC_BASE = 2100H
5:0	—	Reserved. These bits are undefined; for compatibility with future devices, do not modify these bits.

Figure 17-15. Peripheral Interrupt Handler *x* Vector Base (PIH*x*_VEC_BASE) Registers

PIH*x*_VEC_IDX
***x* = 0–1**

Address: 1E90H, 1EA0H
Reset State: 10H

The peripheral interrupt handler vector index (PIH*x*_VEC_IDX) registers indicate which standard PIH input is requesting interrupt service.

15							8
—	—	—	—	—	—	—	—
7							**0**
—	—	—	NI	IS3	IS2	IS1	IS0

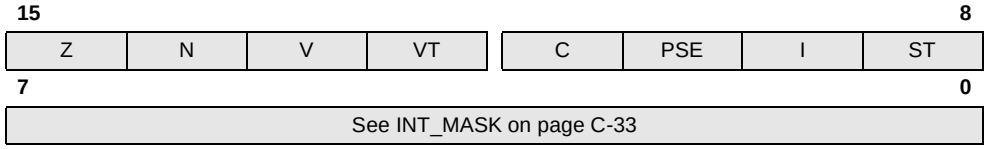
Bit Number	Bit Mnemonic	Function
15:5	—	Reserved. These bits are undefined.
4	NI	No Interrupt Pending This bit is set when the last PIH*x* interrupt request is serviced.
3:0	IS3:0	Interrupt Source These bits indicate the hex value of the highest-priority pending, standard PIH*x* interrupt request. PTS interrupt requests are not recorded in this register.

Figure C-32. Peripheral Interrupt Handler *x* Vector Index (PIH*x*_VEC_IDX) Registers

PSW no direct access

The processor status word (PSW) actually consists of two bytes. The high byte is the status word, which is described here; the low byte is the INT_MASK register. The status word contains one bit (PSW.1) that globally enables or disables servicing of all maskable interrupts, one bit (PSW.2) that enables or disables the peripheral transaction server (PTS), and six Boolean flags that reflect the state of a user's program.

The status word portion of the PSW cannot be accessed directly. To access the status word, push the value onto the stack (PUSHF), then pop the value to a register (POP *test_reg*). The PUSHF and PUSHA instructions save the PSW in the system stack and then clear it; POPF and POPA restore it.



See INT_MASK on page C-33

Bit Number	Bit Mnemonic	Function
15	Z	Zero Flag This flag is set to indicate that the result of an operation was zero. For multiple-precision calculations, the zero flag cannot be set by the instructions that use the carry bit from the previous calculation (e.g., ADDC, SUBC). However, these instructions can clear the zero flag. This ensures that the zero flag will reflect the result of the entire operation, not just the last calculation. For example, if the result of adding together the lower words of two double words is zero, the zero flag would be set. When the upper words are added together using the ADDC instruction, the flag remains set if the result is zero and is cleared if the result is not zero.
14	N	Negative Flag This flag is set to indicate that the result of an operation is negative. The flag is correct even if an overflow occurs. For all shift operations and the NORML instruction, the flag is set to equal the most-significant bit of the result, even if the shift count is zero.
13	V	Overflow Flag This flag is set to indicate that the result of an operation is too large to be represented correctly in the available space. For shift operations (SHL, SHLB, and SHLL), the flag is set if the most-significant bit of the operand changes during the shift. For divide operations, the quotient is stored in the low-order half of the destination operand and the remainder is stored in the high-order half. The overflow flag is set if the quotient is outside the range for the low-order half of the destination operand. (Chapter 3, "Programming Considerations," defines the operands and possible values for each. See the PSW flag descriptions in Appendix A for details.)

Figure C-33. Processor Status Word (PSW)

PSW (Continued) no direct access

The processor status word (PSW) actually consists of two bytes. The high byte is the status word, which is described here; the low byte is the INT_MASK register. The status word contains one bit (PSW.1) that globally enables or disables servicing of all maskable interrupts, one bit (PSW.2) that enables or disables the peripheral transaction server (PTS), and six Boolean flags that reflect the state of a user's program.

The status word portion of the PSW cannot be accessed directly. To access the status word, push the value onto the stack (PUSHF), then pop the value to a register (POP *test_reg*). The PUSHF and PUSHA instructions save the PSW in the system stack and then clear it; POPF and POPA restore it.

15							8
Z	N	V	VT	C	PSE	I	ST
7							0

See INT_MASK on page C-33

Bit Number	Bit Mnemonic	Function
12	VT	Overflow-trap Flag This flag is set when the overflow flag is set, but it is cleared only by the CLRVT, JVT, and JNVT instructions. This allows testing for a possible overflow at the end of a sequence of related arithmetic operations, which is generally more efficient than testing the overflow flag after each operation.
11	C	Carry Flag This flag is set to indicate an arithmetic carry or the last bit shifted out of an operand. It is cleared if a subtraction operation generates a borrow. Normally, the result is rounded up if the carry flag is set. The sticky bit flag allows a finer resolution in the rounding decision. (See the PSW flag descriptions in Appendix A for details.)
10	PSE	PTS Enable This bit globally enables or disables the peripheral transaction server (PTS). The EPTS instruction sets this bit; DPTS clears it. 0 = disable PTS 1 = enable PTS
9	I	Interrupt Disable (Global) This bit globally enables or disables the servicing of all *maskable interrupts*. The bits in INT_MASK and INT_MASK1 individually enable or disable the interrupts. The EI instruction sets this bit; DI clears it. 0 = disable interrupt servicing 1 = enable interrupt servicing
8	ST	Sticky Bit Flag This flag is set to indicate that, during a right shift, a "1" was shifted into the carry flag and then shifted out. It can be used with the carry flag to allow finer resolution in rounding decisions.

Figure C-33. Processor Status Word (PSW) (Continued)

PTSSEL

Address: 0004H
Reset State: 0000H

The PTS select (PTSSEL) register selects either a PTS microcode routine or a standard interrupt service routine for each interrupt request. Setting a bit selects a PTS microcode routine; clearing a bit selects a standard interrupt service routine. In PTS modes that use the PTSCOUNT register, hardware clears the corresponding PTSSEL bit when PTSCOUNT reaches zero. The end-of-PTS interrupt service routine must reset the PTSSEL bit to re-enable the PTS channel.

15							8
—	—	PIH0_PTS	0	PIH1_PTS	0	SSIO1	SSIO0
7							**0**
SDU	EXTINT	RI1	TI1	AD	EPA*x*_OVR	RI0	TI0

Bit Number	Function
15, 14	Reserved; for compatibility with future devices, write zeros to these bits.
12, 10	To guarantee proper device operation, write zeros to these bits.
13, 11, 9:0	Setting a bit causes the corresponding interrupt to be handled by a PTS microcode routine. The PTS interrupt vector locations are as follows:

Bit Mnemonic	Interrupt	PTS Vector
PIH0_PTS†	PIH0 PTS Service Request	see PIH0_PTSSEL
PIH1_PTS†	PIH1 PTS Service Request	see PIH1_PTSSEL
SSIO1	SSIO 1 Transfer	FF2052H
SSIO0	SSIO 0 Transfer	FF2050H
SDU	Serial Debug Unit Interrupt	FF204EH
EXTINT	External Interrupt Pin	FF204CH
RI1	SIO1 Receive	FF204AH
TI1	SIO1 Transmit	FF2048H
AD	A/D Conversion Complete	FF2046H
EPA*x*_OVR	EPA Channel 3–16 Overrun	FF2044H
RI0	SIO0 Receive	FF2042H
TI0	SIO0 Transmit	FF2040H

† Unless these interrupts are masked in the INT_MASK1 register, always write ones to these bits.

Figure C-34. PTS Select (PTSSEL) Register

PTSSRV

Address: 0006H
Reset State: 0000H

The PTS service (PTSSRV) register is used by the hardware to indicate that the final PTS interrupt has been serviced by the PTS routine. When PTSCOUNT reaches zero, hardware clears the corresponding PTSSEL bit and sets the PTSSRV bit, which requests the end-of-PTS interrupt. When the end-of-PTS interrupt is called, hardware clears the PTSSRV bit. The end-of-PTS interrupt service routine must set the PTSSEL bit to re-enable the PTS channel.

15							8
—	—	PIH0_PTS	—	PIH1_PTS	—	SSIO1	SSIO0

7							0
SDU	EXTINT	RI1	TI1	AD	EPA*x*_OVR	RI0	TI0

Bit Number	Function
15, 14, 12, 10	Reserved; always write as zeros.
13, 11, 9:0	A bit is set by hardware to request an end-of-PTS interrupt for the corresponding interrupt through its standard interrupt vector. The standard interrupt vector locations are as follows:

Bit Mnemonic	Interrupt	Standard Vector
PIH0_PTS	PIH0 PTS Service Request	see PIH0_PTSSRV
PIH1_PTS	PIH1 PTS Service Request	see PIH1_PTSSRV
SSIO0	SSIO 0 Transfer	FF2032H
SSIO0	SSIO 0 Transfer	FF2030H
SDU	Serial Debug Unit Interrupt	FF200EH
EXTINT	External Interrupt Pin	FF200CH
RI1	SIO1 Receive	FF200AH
TI1	SIO1 Transmit	FF2008H
AD	A/D Conversion Complete	FF2006H
EPA*x*_OVR	EPA Channel 3–16 Overrun	FF2004H
RI0	SIO0 Receive	FF2002H
TI0	SIO0 Transmit	FF2000H

Figure C-35. PTS Service (PTSSRV) Register

PWM*x_y*_COUNT
***x* = 0, 2, 4, 6**
***y* = 1, 3, 5, 7**
(read only)

Address: See Table 10-2 on page 10-3
Reset State: 00H

The PWM count (PWM*x_y*_COUNT) register provides the current value of the period counter.

7							0
PWM Count Value							

Bit Number	Function
7:0	PWM Count Value This register contains the current value of the period counter.

Figure C-36. PWM Count (PWM*x_y*_COUNT) Register

Table C-16. PWM*x_y*_COUNT Addresses and Reset States

Register	Address	Reset State
PWM0_1_COUNT	1EDDH	XXH
PWM2_3_COUNT	1ED9H	XXH
PWM4_5_COUNT	1ED5H	XXH
PWM6_7_COUNT	1ED1H	XXH

PWM*x_y*_PERIOD
***x* = 0, 2, 4, 6**
***y* = 1, 3, 5, 7**

Address: See Table 10-2 on page 10-3
Reset State: 00H

The PWM period (PWM*x_y*_PERIOD) register of each module controls the period of the two PWM outputs. Write a value to this register to specify the number of state counts necessary for decrementing the PWM counter. The value of PWM*x_y*_PERIOD is loaded into the PWM period count register whenever the count equals zero.

7							0
PWM Period							

Bit Number	Function
7:0	PWM Period This register controls the period of the PWM outputs. The value of PWM*x_y*_PERIOD is loaded into the PWM*x_y*_COUNT register whenever the count equals zero.

Figure C-37. PWM Period (PWM*x_y*_PERIOD) Register

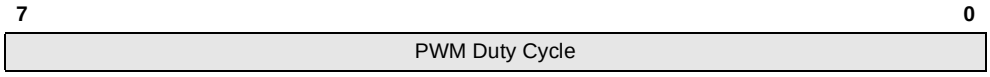
Table C-17. PWM*x*_*y*_PERIOD Addresses and Reset States

Register	Address	Reset State
PWM0_1_PERIOD	1EDFH	FFH
PWM2_3_PERIOD	1EDBH	FFH
PWM4_5_PERIOD	1ED7H	FFH
PWM6_7_PERIOD	1ED3H	FFH

PWM*x*_CONTROL, PWM*y*_CONTROL
***x* = 0, 2, 4, 6**
***y* = 1, 3, 5, 7**

Address: See Table 10-2 on page 10-3
Reset: 00H

The PWM control (PWM*x*_CONTROL, PWM*y*_CONTROL) registers determine the duty cycle of the PWM channel pair. A zero loaded into this register causes the PWM to output a low continuously (0% duty cycle). An FFH in this register causes the PWM to have its maximum duty cycle (99.6% duty cycle). Variables *x* and *y* represent the even- and odd-numbered members of an adjacent PWM channel pair, respectively.



Bit Number	Function
7:0	PWM Duty Cycle This register controls the PWM duty cycle. A zero loaded into this register causes the PWM to output a low continuously (0% duty cycle). An FFH in this register causes the PWM to have its maximum duty cycle (99.6% duty cycle).

Figure C-38. PWM Control (PWM*x*_CONTROL, PWM*y*_CONTROL) Registers

Table C-18. PWM*x*_CONTROL and PWM*y*_CONTROL Addresses and Reset States

Register	Address	Reset State
PWM0_CONTROL	1EDEH	00H
PWM1_CONTROL	1EDCH	00H
PWM2_CONTROL	1EDAH	00H
PWM3_CONTROL	1ED8H	00H
PWM4_CONTROL	1ED6H	00H
PWM5_CONTROL	1ED4H	00H
PWM6_CONTROL	1ED2H	00H
PWM7_CONTROL	1ED0H	00H

RSTSRC

Address: 1FA4H
Reset State: 00H†

The reset source (RSTSRC) register indicates the sources of the last reset that the microcontroller encountered.

7							0
—	—	—	—	CFDRST	WDTRST	SFWRST	EXTRST

Bit Number	Bit Mnemonic	Function
7:4	—	Reserved. These bits are undefined.
3	CFDRST	Clock Failure Detection Reset When set, this bit indicates that a failed clock caused the last reset.
2	WDTRST	Watchdog Timer Reset When set, this bit indicates that the watchdog timer caused the last reset.
1	SFWRST	Software Reset When set, this bit indicates that either the RST instruction or the IDLPD instruction used with an illegal key caused the last reset.
0	EXTRST	External Reset When set, this bit indicates that the RESET# pin being asserted caused the last reset.

† Only applies to a V_{CC} Powerup condition.

Figure C-39. Reset Source (RSTSRC) Register

SBUF*x*_RX
***x* = 0–1**

Address: 1F88H, 1F98H
Reset State: 00H

The serial port receive buffer (SBUF_RX) register contains data received from serial port. The serial port receiver is buffered and can begin receiving a second data byte before the first byte is read. Data is held in the receive shift register until the last data bit is received, then the data byte is loaded into SBUF_RX. If data in the shift register is loaded into SBUF_RX before the previous byte is read, the overflow error bit is set (SP_STATUS.2). The data in SBUF_RX will always be the last byte received, never a combination of the last two bytes.

7	0
Data Received	

Bit Number	Function
7:0	Data Received This register contains the last byte of data received from the serial port.

Figure C-40. Serial Port Receive Buffer (SBUF_RX) Register

SBUF*x*_TX
***x* = 0–1**

Address: 1F8AH, 1F9AH
Reset State: 00H

The serial port transmit buffer (SBUF_TX) register contains data that is ready for transmission. In modes 1, 2, and 3, writing to SBUF_TX starts a transmission. In mode 0, writing to SBUF_TX starts a transmission only if the receiver is disabled (SP_CON.3=0).

7							0
Data to Transmit							

Bit Number	Function
7:0	Data to Transmit This register contains a byte of data to be transmitted by the serial port.

SP Address: 18H
Reset State: XXXXH

The system's stack pointer (SP) can point anywhere in page 00H; it must be word aligned and must always be initialized before use. The stack pointer is decremented before a PUSH and incremented after a POP, so you should initialize the stack pointer to two bytes (in 64-Kbyte mode) or four bytes (in 2-Mbyte mode) above the highest stack location. If stack operations are not being performed, locations 18H and 19H may be used as standard registers.

15	0
Stack Pointer	

Bit Number	Function
15:0	Stack Pointer This register makes up the system's stack pointer.

Figure C-41. Stack Pointer (SP)

SP*x*_BAUD
***x* = 0–1**

Address: 1F8CH, 1F9CH
Reset State: 0000H

The serial port baud rate *x* (SP*x*_BAUD) register selects the clock source and serial port baud rate. The most-significant bit selects the clock source. The lower 15 bits represent baud value, an unsigned integer that determines the baud rate.

The maximum baud value is 32,767 (7FFFH). In asynchronous modes 1, 2, and 3, the minimum baud value is 0000H when using the internal clock source (f) and 0001H when using T2CLK. In synchronous mode 0, the minimum baud value is 0001H for transmissions and 0002H for receptions.

WARNING: Writing to the SP*x*_BAUD register during a reception or transmission can corrupt the received or transmitted data. Before writing to SP*x*_BAUD, check SP*x*_STATUS or the interrupt pending register to ensure that the reception or transmission is complete.

15							**8**
CLKSRC	BV14	BV13	BV12	BV11	BV10	BV9	BV8
7							**0**
BV7	BV6	BV5	BV4	BV3	BV2	BV1	BV0

Bit Number	Bit Mnemonic	Function
15	CLKSRC	Serial Port Clock Source This bit determines whether the baud-rate generator is clocked from an internal or an external source. 0 = signal on the T2CLK pin (external source) 1 = internal operating frequency (f)
14:0	BV14:0	These bits constitute the baud value. Use the following equations to determine the baud value for a given baud rate. Synchronous mode 0:† $\text{Baud Value} = \frac{f}{\text{Baud Rate} \times 2} - 1$ or $\frac{\text{T2CLK}}{\text{Baud Rate}}$ Asynchronous modes 1, 2, and 3: $\text{Baud Value} = \frac{f}{\text{Baud Rate} \times 16} - 1$ or $\frac{\text{T2CLK}}{\text{Baud Rate} \times 8}$ † For mode 0 receptions, the baud value must be 0002H or greater. For mode 0 transmissions, the baud value must be 0001H or greater.

Figure C-42. Serial Port *x* Baud Rate (SP*x*_BAUD) Register

SP*x*_CON
***x* = 0–1**

Address: 1F8BH, 1F9BH
Reset State: 00H

The serial port control (SP*x*_CON) register selects the communications mode and enables or disables the receiver for all modes. For modes 1 and 3, it enables or disables even or odd parity. For modes 2 and 3, it contains the ninth data bit to be transmitted. Use this register to delay the setting of the TI and TXE flags in the serial port status register and the TI*x* interrupt pending bit by one, three, or seven bit times.

7							0
ID1	ID0	PAR	TB8	REN	PEN	M1	M0

Bit Number	Bit Mnemonic	Function
7:6	ID1:0	Interrupt Delay These bits delay the setting of the TI and TXE flags in the SP*x*_STATUS register and the TI*x* interrupt pending bit after transmissions. For mode 0, the SIO sets the TI and TXE flags and the TI*x* pending bit immediately after it shifts out the eighth data bit unless a delay is selected. For modes 1, 2, and 3, the SIO sets TI and TXE flags and the TI*x* pending bit when it starts to shift out the stop bit unless a delay is selected. **ID1 ID0** 0 0 set immediately (no delay) 0 1 delay by one bit time 1 0 delay by three bit times 1 1 delay by seven bit times
5	PAR	Parity Selection Bit In modes 1 and 3, this bit selects even or odd parity. 0 = even parity 1 = odd parity For modes 0 and 2, this bit is ignored.
4	TB8	Transmit Ninth Data Bit This is the ninth data bit that will be transmitted in mode 2 or 3. This bit is cleared after each transmission, so you must write to this bit before writing to SBUF*x*_TX. For mode 3, when parity is enabled (SP*x*_CON.2 = 1), the transmitter sets or clears this bit so that the byte being transmitted contains the correct parity.
3	REN	Receive Enable In mode 1, 2, or 3, setting this bit enables receptions. When this bit is set, a falling edge on the RXD*x* pin starts a reception. In these modes, this bit has no effect on transmissions. In mode 0, clearing this bit enables transmissions and setting it enables receptions. Clearing this bit stops a reception in progress and inhibits further receptions. In mode 0, clearing the RI flag in the SP*x*_STATUS register starts a reception; therefore, to avoid corrupting your reception, clear this bit before clearing the RI bit.

Figure C-43. Serial Port Control (SP*x*_CON) Register

SP*x*_CON (Continued)
***x* = 0–1**

Address: 1F8BH, 1F9BH
Reset State: 00H

The serial port control (SP*x*_CON) register selects the communications mode and enables or disables the receiver for all modes. For modes 1 and 3, it enables or disables even or odd parity. For modes 2 and 3, it contains the ninth data bit to be transmitted. Use this register to delay the setting of the TI and TXE flags in the serial port status register and the TI*x* interrupt pending bit by one, three, or seven bit times.

7							0
ID1	ID0	PAR	TB8	REN	PEN	M1	M0

Bit Number	Bit Mnemonic	Function
2	PEN	Parity Enable In modes 1 and 3, setting this bit enables parity. For mode 1, when this bit is set, the seventh data bit takes the parity value on transmissions and SP*x*_STATUS.7 becomes the receiver parity error bit. For mode 3, when this bit is set, SP*x*_CON.4 (TB8) takes the parity value on transmissions and SP*x*_STATUS.7 becomes the receive parity error bit. Clear this bit for mode 2. For mode 0, this bit is ignored.
1:0	M1:0	Mode Selection These bits select the communications mode. **M1 M0** 0 0 mode 0, synchronous 0 1 mode 1, 8-bit asynchronous with optional parity 1 0 mode 2, 9-bit asynchronous with optional receive interrupt 1 1 mode 3, 9-bit asynchronous with optional parity

Figure C-43. Serial Port Control (SP*x*_CON) Register (Continued)

SP*x*_STATUS
***x* = 0–1**

Address: 1F89H, 1F99H
Reset State: 00H

The serial port status (SP*x*_STATUS) register contains bits that indicate the status of the serial port *x*.

7							0
RPE/RB8	RI	TI	FE	TXE	OE	RIP	—

Bit Number	Bit Mnemonic	Function
7	RPE/RB8	Received Parity Error/Received Bit 8 For modes 1 and 3, RPE is set if parity is enabled (SP*x*_CON.2 = 1) and the data received does not contain the correct parity, as programmed in SP*x*_CON. For mode 2, and for mode 3 with parity disabled, this bit is the ninth data bit received. (The serial port receive buffer contains the received data bits 0–7. The received data bit 8 is written to this bit.) Reading SP*x*_STATUS clears this bit.
6	RI	Receive Interrupt This bit indicates whether an incoming data byte has been received. For modes 0, 1, and 3, this bit is set when the last bit (eighth bit for mode 0, or stop bit for modes 1 and 3) is sampled. For mode 2, this bit is set when the stop bit is detected only if the ninth bit received (SP*x*_STATUS, RB8) is a one. Reading SP*x*_STATUS clears this bit.
5	TI	Transmit Interrupt This bit indicates whether a data byte has finished transmitting. For mode 0 transmissions, the SIO sets this bit immediately after it transmits the eighth data bit, unless a delay is selected in the SP*x*_CON register. For mode 1, 2, and 3 transmissions, the SIO sets this bit immediately after it starts to transmit the stop bit, unless a delay is selected in the SP*x*_CON register. You can delay setting this bit by one, three, or seven bit times. Reading SP*x*_STATUS clears this bit.
4	FE	Framing Error For modes 1, 2, and 3, this bit is set if the receiver does not detect a valid stop bit within the appropriate period of time. Reading SP*x*_STATUS clears this bit. For mode 0, this bit has no function.
3	TXE	SBUF*x*_TX Empty The SIO sets this bit, along with the TI flag, if the transmit buffer and the transmit shift register are both empty. Using the SP*x*_CON register, you can delay the setting of this bit by one, three, or seven bit times. When set, this bit indicates that two bytes can be written to the transmit buffer. Writing to the transmit buffer clears this bit.

Figure C-44. Serial Port Status (SP*x*_STATUS) Register

SP*x*_STATUS (Continued)
***x* = 0–1**

Address: 1F89H, 1F99H
Reset State: 00H

The serial port status (SP*x*_STATUS) register contains bits that indicate the status of the serial port *x*.

7							0
RPE/RB8	RI	TI	FE	TXE	OE	RIP	—

Bit Number	Bit Mnemonic	Function
2	OE	Overrun Error The SIO sets this bit if data in the receive shift register is loaded into SBUF*x*_RX before the previous byte in SBUF*x*_RX is read. Reading SP*x*_STATUS clears this bit.
1	RIP	Reception In Progress For modes 1, 2, and 3, this bit simplifies detecting an idle data line for asynchronous, half-duplex operations. A falling edge on RXD (indicating the possibility of a valid start bit) sets this bit. Either of three conditions clears this bit: • the start bit is found to be invalid, • the stop bit is received, or • a framing error occurs (there is no valid stop bit within the proper time). 0 = receiver is idle 1 = data is being received For mode 0, this bit is not useful. A falling edge on RXD sets this bit and a rising edge clears it.
0	—	Reserved; for compatibility with future devices, write zero to this bit.

Figure C-44. Serial Port Status (SP*x*_STATUS) Register (Continued)

SSIO_BAUD Address: 1F94H
Reset State: 00H

The synchronous serial port baud (SSIO_BAUD) register enables and disables the baud-rate generator and selects the baud rate for the internal baud clock. During read operations, SSIO_BAUD serves as the down-counter monitor. The down-counter is decremented once every four state times when the baud-rate generator is enabled.

7							0
BE	BV6	BV5	BV4	BV3	BV2	BV1	BV0

Bit Number	Bit Mnemonic	Function
7	BE	Baud-rate Generator Enable This bit enables and disables the baud-rate generator. **For write operations:** 0 = disable the baud-rate generator and clear BV6:0 1 = enable the baud-rate generator and start the down-counter **For read operations:** 0 = baud-rate generator is disabled 1 = baud-rate generator is enabled and down-counter is running
6:0	BV6:0	Baud Value **For write operations:** These bits represent baud value, an unsigned integer that determines the baud rate. The maximum value of baud value is 7FH; the minimum value is 0. Use the following equation to determine baud value for a given baud rate. $\text{Baud Value} = \frac{f}{\text{Baud Rate} \times 8} - 1$ **For read operations:** These bits contain the current value of the down-counter.

Figure C-45. Synchronous Serial Port Baud (SSIO_BAUD) Register

Table C-19. Common SSIO_BAUD Values at 40 MHz Operating Frequency

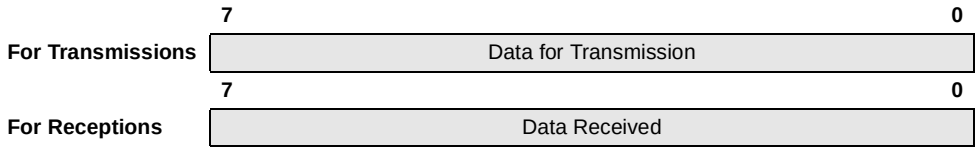
Baud Rate		SSIO_BAUD Value†
(Maximum)	5.0 MHz	80H
	100.0 KHz	B1H
	50.0 KHz	E3H
	40.0 KHz	FCH
(Minimum)	39.1 KHz	FFH

† Bit 7 must be set to enable the baud-rate generator.

SSIO*x*_BUF
***x* = 0–1**

Address: 1F90H, 1F92H
Reset State: 00H

The SSIO *x* buffer (SSIO*x*_BUF) register contains either data for transmission or data received, depending on channel *x*'s configuration. For transmissions, data is shifted from this register to the SD*x* pin; for receptions, data is sampled into this register from the SD*x* pin. For both transmissions and receptions, the data is transferred with most-significant bit first.



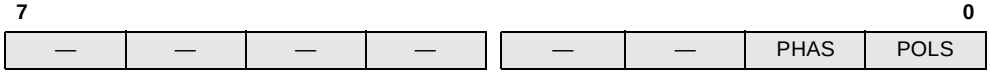
Bit Number	Function
7:0	Data for Transmission During transmissions, this register contains a byte of data to be transmitted by the synchronous serial port.
	Data Received During receptions, this register contains the last byte of data received from the synchronous serial port.

Figure C-46. Synchronous Serial *x* Buffer (SSIO*x*_BUF) Register

SSIO0_CLK

Address: 1F95H
Reset State: 00H

The SSIO 0 clock (SSIO0_CLK) register configures the serial clock for channel 0. In standard mode, the SC0 is channel 0's clock signal. In duplex and channel-select modes, SC0 is the common clock signal for both SSIO channels.



Bit Number	Bit Mnemonic	Function
7:2	—	Reserved; for compatibility with future devices, write zeros to these bits.

Figure C-47. SSIO 0 Clock (SSIO0_CLK) Register

SSIO0_CLK (Continued)

Address: 1F95H
Reset State: 00H

The SSIO 0 clock (SSIO0_CLK) register configures the serial clock for channel 0. In standard mode, the SC0 is channel 0's clock signal. In duplex and channel-select modes, SC0 is the common clock signal for both SSIO channels.

7							0
—	—	—	—	—	—	PHAS	POLS

Bit Number	Bit Mnemonic	Function
1	PHAS	Phase and Polarity Select
0	POLS	For normal transfers, these bits determine the idle state of the serial clock and select the serial clock signal edge on which the SSIO samples incoming data bits or shifts out outgoing data bits. These bits are ignored for handshaking transfers. Use SSIO0_ CON to select the type of data transfer (normal or handshaking) for channel 0. **For transmissions** PHAS POLS 0 0 low idle state; shift on falling edges 0 1 high idle state; shift on rising edges 1 0 low idle state; shift on rising edges 1 1 high idle state; shift on falling edges **For receptions** PHAS POLS 0 0 low idle state; sample on rising edges 0 1 high idle state; sample on falling edges 1 0 low idle state; sample on falling edges 1 1 high idle state; sample on rising edges

Figure C-47. SSIO 0 Clock (SSIO0_CLK) Register (Continued)

SSIO1_CLK

Address: 1F97H
Reset State: 00H

The SSIO 1 clock (SSIO1_CLK) register selects the SSIO mode of operation (standard, duplex, or channel-select), enables the channel-select master contention interrupt request, and selects the phase and polarity for the serial clock (SC1) for channel 1.

7							0
—	—	CHS	DUP	CONINT	CONPND	PHAS	POLS

Bit Number	Bit Mnemonic	Function
7:6	—	Reserved; for compatibility with future devices, write zeros to these bits.

Figure C-48. SSIO 1 Clock (SSIO1_CLK) Register

SSIO1_CLK (Continued)

Address: 1F97H
Reset State: 00H

The SSIO 1 clock (SSIO1_CLK) register selects the SSIO mode of operation (standard, duplex, or channel-select), enables the channel-select master contention interrupt request, and selects the phase and polarity for the serial clock (SC1) for channel 1.

7							0
—	—	CHS	DUP	CONINT	CONPND	PHAS	POLS

Bit Number	Bit Mnemonic	Function
5	CHS	These bits determine the SSIO operating mode. **CHS DUP** 0 0 standard mode 0 1 duplex mode 1 0 channel-select half-duplex mode (uses SD1 only) 1 1 channel-select full-duplex mode (uses both SD0 and SD1)
4	DUP	
3	CONINT	Master Contention Interrupt For channel-select master operations, the SSIO sets the master contention interrupt pending bit (CONPND) when the CHS# pin is externally activated. In a system with multiple masters, an external master activates the CHS# signal to request control of the serial clock. CONINT determines whether the SSIO sets both CONPND and the SSIO0 interrupt pending bit or only CONPND when the CHS# pin is externally activated. 0 = SSIO sets only CONPND 1 = SSIO sets both CONPND and the SSIO0 interrupt pending bit This bit is valid for channel-select master operations and ignored for all other operations.
2	CONPND	Master Contention Interrupt Pending For channel-select master operations, the SSIO sets this bit when the CHS# pin is externally activated. In a system with multiple masters, an external master activates the CHS# signal to request control of the serial clock. This bit is valid for channel-select master operations and ignored for all other operations.

Figure C-48. SSIO 1 Clock (SSIO1_CLK) Register (Continued)

SSIO1_CLK (Continued)

Address: 1F97H
Reset State: 00H

The SSIO 1 clock (SSIO1_CLK) register selects the SSIO mode of operation (standard, duplex, or channel-select), enables the channel-select master contention interrupt request, and selects the phase and polarity for the serial clock (SC1) for channel 1.

7							0
—	—	CHS	DUP	CONINT	CONPND	PHAS	POLS

Bit Number	Bit Mnemonic	Function
1	PHAS	Phase and Polarity Select
0	POLS	For normal transfers, these bits determine the idle state of the serial clock and select the serial clock signal edge that the SSIO samples incoming data bits or shifts out outgoing data bits. **For transmissions** PHAS POLS 0 0 low idle state; shift on falling edges 0 1 high idle state; shift on rising edges 1 0 low idle state; shift on rising edges 1 1 high idle state; shift on falling edges **For receptions** PHAS POLS 0 0 low idle state; sample on rising edges 0 1 high idle state; sample on falling edges 1 0 low idle state; sample on falling edges 1 1 high idle state; sample on rising edges These bits are ignored for duplex and channel-select modes; these modes use SC0 as the common clock signal. The SSIO0_CLK register contains the phase and polarity select bits for the SC0 clock signal. These bits are also ignored for handshaking transfers. Use SSIO1_ CON to select the type of data transfer (normal or handshaking) for channel 1.

Figure C-48. SSIO 1 Clock (SSIO1_CLK) Register (Continued)

SSIO*x*_CON
***x* = 0–1**

Address: 1F91H, 1F93H
Reset State: 00H

The synchronous serial control *x* (SSIO*x*_CON) register selects the channel configuration (master or slave, transmitter or receiver), the data transfer type (normal or handshaking), and indicates buffer status. SSIO*x*_CON also determines whether the SSIO re-enables the channel at the completion of a transfer.

7							0
M/S#	T/R#	TRT	THS	STE	ATR	OUF	TBS

Bit Number	Bit Mnemonic	Function
7	M/S#	Master/Slave Select Configures the channel as either master or slave. 0 = slave (externally generated clocking signal is input on SC*x*) 1 = master (internally generated clocking signal is output on SC*x*) In standard mode, either channel can act as master or slave. In channel-select and duplex modes, configure channel 1 as a slave (SSIO1_CON.7 = 0) and configure channel 0 as desired.
6	T/R#	Transmit/Receive Select Configures the channel as either transmitter or receiver. 0 = receiver (data is input on SD*x*) 1 = transmitter (data is output on SD*x*) In standard mode, either channel can act as transmitter or receiver. In channel-select and duplex modes, configure channel 0 as the transmitter (SSIO0_CON.6 = 1) and channel 1 as the receiver (SSIO0_CON.6 = 0).
5	TRT	Transmitter/Receiver Toggle Setting this bit causes the SSIO to toggle the channel configuration (transmitter or receiver) at the completion of a transfer, thus avoiding possible contention on the data line. The SSIO clears this bit at the completion of each transfer.
4	THS	Transceiver Handshake Select Selects the type of data transfer. 0 = normal transfers (all modes) 1 = handshaking transfers (standard mode) In standard mode, the channels can perform either normal or handshaking transfers. In channel-select and duplex modes, always configure the channels for normal transfers by clearing this bit. Set STE and ATR, along with THS, for handshaking transfers.

Figure C-49. Synchronous Serial Control *x* (SSIO*x*_CON) Registers

SSIO*x*_CON (Continued) Address: 1F91H, 1F93H
***x* = 0–1** Reset State: 00H

The synchronous serial control *x* (SSIO*x*_CON) register selects the channel configuration (master or slave, transmitter or receiver), the data transfer type (normal or handshaking), and indicates buffer status. SSIO*x*_CON also determines whether the SSIO re-enables the channel at the completion of a transfer.

7 **0**

M/S#	T/R#	TRT	THS	STE	ATR	OUF	TBS

Bit Number	Bit Mnemonic	Function
3	STE	Single Transfer Enable Enables and disables transfer of a single byte. Unless ATR is set, STE is automatically cleared at the end of a transfer. 0 = disable transfer 1 = enable transfer Set THS, STE, and ATR for handshaking transfers.
2	ATR	Automatic Transfer Re-enable Enables and disables subsequent transfers. 0 = disable subsequent transfers (allow automatic clearing of STE) 1 = enable subsequent transfers (prevent automatic clearing of STE) Set THS, STE, and ATR for handshaking transfers.
1	OUF	Overflow/Underflow Flag Indicates whether an overflow or underflow has occurred. An attempt to access SSIO*x*_BUF during a byte transfer sets this bit. **For the master (M/S# = 1)** 0 = no overflow or underflow has occurred 1 = the core attempted to access SSIO*x*_BUF during the current transfer **For the slave (M/S# = 0)** 0 = no overflow or underflow has occurred 1 = the core attempted to access SSIO*x*_BUF during the current transfer **or** the master attempted to clock data into or out of the slave's SSIO*x*_BUF before the buffer was available
0	TBS	Transceiver Buffer Status Indicates the status of the channel's SSIO*x*_BUF. **For transmissions (T/R# =1)** 0 = SSIO*x*_BUF is full; waiting to transmit 1 = SSIO*x*_BUF is empty; buffer available **For receptions (T/R# = 0)** 0 = SSIO*x*_BUF is empty; waiting to receive 1 = SSIO*x*_BUF is full; data available

Figure C-49. Synchronous Serial Control *x* (SSIO*x*_CON) Registers (Continued)

STACK_BOTTOM

Address: 1FA0H
Reset State: 0000H

The value you write to this register specifies the lower address limit for the stack. The overflow detection circuitry compares the stack pointer (SP) value on the peripheral address bus to the STACK_TOP and STACK_BOTTOM values.

15	8
Lower Stack Limit (high byte)	

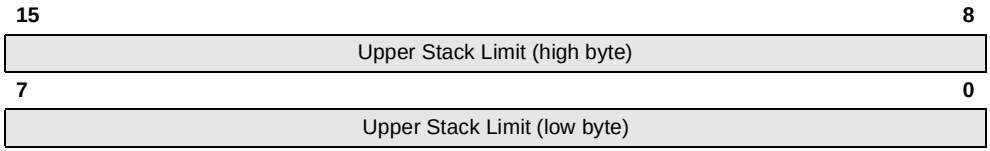
7	0
Lower Stack Limit (low byte)	

Bit Number	Function
15:0	Lower Stack Limit This register specifies the lower address limit for the stack. If the stack pointer value is equal to or less than this value, the stack overflow module sets the nonmaskable stack error interrupt (INT14) pending bit in INT_PEND1.

Figure C-50. Lower Stack Limit (STACK_BOTTOM) Register

STACK_TOP Address: 1FA2H
Reset State: FFFFH

The value you write to this register specifies the upper address limit for the stack. You must write to this register to enable the stack overflow module. The overflow detection circuitry compares the stack pointer (SP) value on the peripheral address bus to the STACK_TOP and STACK_BOTTOM values.



Bit Number	Function
15:0	Upper Stack Limit This register specifies the upper address limit for the stack. If the stack pointer value is equal to or greater than this value, the stack overflow module sets the nonmaskable stack error interrupt (INT14) pending bit in INT_PEND1.

Figure C-51. Upper Stack Limit (STACK_TOP) Register

T*x*CONTROL
***x* = 1–4**

The timer *x* control (T*x*CONTROL) register enables the associated timer/counter and specifies the counting direction, concatenation, reset source, clock source, and count rate for timer *x*.

15							8
—	—	—	—	—	—	CE	UD
7							**0**
ETC†	RM1	RM0	CM1	CM0	P2	P1	P0

Bit Number	Bit Mnemonic	Function
15:10	—	Reserved; always write as zeros.
9	CE	Counter Enable This bit enables or disables the timer. From reset, the timers are disabled and not free running. 0 = disable timer/counter 1 = enable timer/counter
8	UD	Up/Down This bit specifies the timer's counting direction. 0 = count down 1 = count up If T2CONTROL.7 is set, this bit in T2CONTROL controls the direction of both timers 2 and 1. If T4CONTROL.7 is set, this bit in T4CONTROL controls the direction of both timers 4 and 3.
7	ETC†	Enable Timer Concatenation This bit allows you to concatenate timers 2 and 1 and timers 4 and 3 to provide two 32-bit time bases. 0 = no concatenation 1 = concatenate timers Set this bit in T2CONTROL to concatenate timers 1 and 2. Set this bit in T4CONTROL to concatenate timers 3 and 4.
6:5	RM1:0	Reset Mode This bit specifies whether an output event causes a reset and, if so, the edge (falling, rising, or either) that causes the reset. **RM1 RM0 Reset** 0 0 internal reset only 0 1 reset reference timer *x* on falling edge 1 0 reset reference timer *x* on rising edge 1 1 reset reference timer *x* on falling or rising edge

† For T1CONTROL and T3CONTROL, this bit is reserved. Write zero to this bit for proper operation.

Figure C-52. Timer *x* Control (T*x*CONTROL) Register

T*x*CONTROL (Continued)
***x* = 1–4**

The timer *x* control (T*x*CONTROL) register enables the associated timer/counter and specifies the counting direction, concatenation, reset source, clock source, and count rate for timer *x*.

15							8
—	—	—	—	—	—	CE	UD
7							**0**
ETC†	RM1	RM0	CM1	CM0	P2	P1	P0

Bit Number	Bit Mnemonic	Function
4:3	CM1:0	Clock Mode Select These bits specify whether the clock signal is provided by the internal frequency (f) or an external pin (T*x*CLK) **CM1 CM0 Clocking** 0 0 internal clocking (f) 0 1 external clocking on falling edge of T*x*CLK 1 0 external clocking on rising edge of T*x*CLK 1 1 external clocking on falling and rising edges of T*x*CLK
2:0	P2:0	EPA Clock Prescaler Bits These bits determine the clock prescaler value. **P2 P1 P0 Prescaler Divisor Resolution††** 0 0 0 divide by 1 (disabled) 125 ns 0 0 1 divide by 2 250 ns 0 1 0 divide by 4 500 ns 0 1 1 divide by 8 1 µs 1 0 0 divide by 16 2 µs 1 0 1 divide by 32 4 µs 1 1 0 divide by 64 8 µs 1 1 1 — reserved †† At f = 32 MHz and TIMER_MUX.0 = 0, use the formula on page 11-8 to calculate the resolution at other frequencies or with TIMER_MUX.0 =1.

† For T1CONTROL and T3CONTROL, this bit is reserved. Write zero to this bit for proper operation.

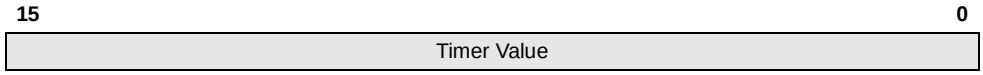
Figure C-52. Timer *x* Control (T*x*CONTROL) Register (Continued)

Table C-20. T*x*CONTROL Addresses

Register	Address
T1CONTROL	1F7CH
T2CONTROL	1F78H
T3CONTROL	1F74H
T4CONTROL	1F70H

TIMER*x*
***x* = 1–4**

This register contains the value of timer *x*. This register can be written, allowing timer *x* to be initialized to a value other than zero.



Bit Number	Function
15:0	Timer Value Read the current timer *x* value from this register or write a new timer *x* value to this register.

Figure C-53. Timer *x* Time (TIMER*x*) Registers

Table C-21. TIMER*x* Addresses and Reset States

Register	Address	Reset State
TIMER1	1F7EH	0000H
TIMER2	1F7AH	0000H
TIMER3	1F76H	0000H
TIMER4	1F72H	0000H

TIMER_MUX Address: 1F6EH
Reset State: 00H

The timer multiplexing (TIMER_MUX) register controls the number of timer values that can be time-multiplexed on the bus and available to the EPA for capture/compare or. With a two-state time field, only timers 1 and 2 can be used, and each timer's maximum count rate is f/4. With a four-state time field, all four timers can be used, and each timer's maximum count rate is f/8.

7							0
—	—	—	—	—	—	—	MXS

Bit Number	Bit Mnemonic	Function
7:1	—	Reserved; always write as zeros.
0	MXS	MUX Select Selects a two-state or a four-state time field. 0 = two-state time field; timers 1 and 2 share the time bus, so each of them can count every two state times (timers 3 and 4 cannot be used) 1 = four-state time field; timers 1 through 4 share the time bus, so each timer can count every four state times

Figure C-54. Timer/Counter Multiplexer (TIMER_MUX) Register

Glossary

GLOSSARY

This glossary defines acronyms, abbreviations, and terms that have special meaning in this manual. (Chapter 1 discusses notational conventions and general terminology.)

2-Mbyte mode — The addressing mode that allows code to reside anywhere in the addressing space.

64-Kbyte mode — The addressing mode that allows code to reside only in page FFH.

absolute error — The maximum difference between corresponding actual and ideal *code transitions*. Absolute error accounts for all deviations of an actual A/D converter from an ideal converter.

accumulator — A register or storage location that forms the result of an arithmetic or logical operation.

actual characteristic — A graph of output code versus input voltage of an actual *A/D converter.* An actual characteristic may vary with temperature, supply voltage, and frequency conditions.

A/D converter — Analog-to-digital converter. An internal peripheral that converts an analog input to a digital value.

ALU — Arithmetic-logic unit. The part of the *RALU* that processes arithmetic and logical operations.

assert — The act of making a signal active (enabled). The polarity (high or 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 ALE is to drive it high.

attenuation — A decrease in amplitude; voltage decay.

bit — A binary digit.

BIT — A single-bit operand that can take on the Boolean values, “true” and “false.”

break-before-make The property of a multiplexer which guarantees that a previously selected channel is deselected before a new channel is selected. (That is, break-before-make ensures that the *A/D converter* will not short inputs together.)

byte Any 8-bit unit of data.

BYTE An unsigned, 8-bit variable with values from 0 through 2^8–1.

CCBs Chip configuration bytes. The chip configuration registers (*CCRs*) are loaded with the contents of the CCBs after a reset.

CCRs Chip configuration registers. Registers that define the environment in which the microcontroller will be operating. The chip configuration registers are loaded with the contents of the *CCBs* after a reset.

channel-to-channel matching error The difference between corresponding *code transitions* of actual characteristics taken from different *A/D converter* channels under the same temperature, voltage, and frequency conditions. This error is caused by differences in *DC input leakage* and on-channel resistance from one multiplexer channel to another.

characteristic A graph of output code versus input voltage; the *transfer function* of an *A/D converter*.

chip-select unit The integrated module that selects an external memory device during an external bus cycle.

clear The "0" value of a bit or the act of giving it a "0" value. See also *set*.

code 1) A set of instructions that perform a specific function; a program.
2) The digital value output by the *A/D converter.*

code center The voltage corresponding to the midpoint between two adjacent *code transitions* on the *A/D converter.*

code transition The point at which the *A/D converter*'s output code changes from "Q" to "Q+1." The input voltage corresponding to a code transition is defined as the voltage that is equally likely to produce either of two adjacent codes.

code width — The voltage change corresponding to the difference between two adjacent *code transitions*. Code width deviations cause *differential nonlinearity* and *nonlinearity* errors.

crosstalk — See *off-isolation.*

DC input leakage — Leakage current from an analog input pin to ground or to the reference voltage (V_{REF}).

deassert — The act of making a signal inactive (disabled). The polarity (high or 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 ALE is to drive it low.

demultiplexed bus — The configuration in which the microcontroller uses separate lines for address and data (address on A20:0; data on AD15:0 for a 16-bit bus or AD7:0 for an 8-bit bus). See also *multiplexed bus*.

differential nonlinearity — The difference between the actual *code width* and the ideal one-LSB code width of the *terminal-based characteristic* of an A/D converter. It provides a measure of how much the input voltage may have changed in order to produce a one-count change in the conversion result. *Differential nonlinearity* is a measure of local code-width error; *nonlinearity* is a measure of overall code-transition error.

doping — The process of introducing a periodic table Group III or Group V element into a Group IV element (e.g., silicon). A Group III impurity (e.g., indium or gallium) results in a *p-type material*. A Group V impurity (e.g., arsenic or antimony) results in an *n-type material*.

double-word — Any 32-bit unit of data.

DOUBLE-WORD — An unsigned, 32-bit variable with values from 0 through 2^{32}–1.

EPA — Event processor array. An integrated peripheral that provides high-speed input/output capability.

EPORT — Extended addressing port. The port that provides the additional address lines to support extended addressing.

EPROM Erasable, programmable read-only-memory.

ESD Electrostatic discharge.

external address A 21-bit address is presented on the microcontroller's pins. The address decoded by an external device depends on how many of these address pins the external system uses. See also *internal address*.

far constants Constants that can be accessed only with extended instructions. See also *near constants*.

far data Data that can be accessed only with extended instructions. See also *near data*.

feedthrough The *attenuation* from an input voltage on the selected channel to the A/D output after the *sample window* closes. The ability of the *A/D converter* to reject an input on its selected channel after the sample window closes.

FET Field-effect transistor.

f Lowercase "f" represents the frequency of the internal clock.

full-scale error The difference between the ideal and actual input voltage corresponding to the final (full-scale) *code transition* of an *A/D converter.*

hold latency The time it takes the microcontroller to assert HLDA# after an external device asserts HOLD#.

ideal characteristic The *characteristic* of an ideal *A/D converter.* An ideal characteristic is unique: its first *code transition* occurs when the input voltage is 0.5 LSB, its full-scale (final) code transition occurs when the input voltage is 1.5 LSB less than the full-scale reference, and its code widths are all exactly 1.0 LSB. These properties result in a conversion without *zero-offset*, *full-scale*, or *linearity* errors. *Quantizing error* is the only error seen in an ideal A/D converter.

input leakage Current leakage from an input pin to power or ground.

input series resistance The effective series resistance from an analog input pin to the *sample capacitor* of an *A/D converter.*

integer Any member of the set consisting of the positive and negative whole numbers and zero.

INTEGER — A 16-bit, signed variable with values from -2^{15} through $+2^{15}-1$.

internal address — The 24-bit address that the microcontroller generates. See also *external address*.

interrupt controller — The module responsible for handling interrupts that are to be serviced by *interrupt service routines* that you provide. Also called the *programmable interrupt controller (PIC)*.

interrupt latency — The total delay between the time that an interrupt is generated (not acknowledged) and the time that the microcontroller begins executing the *interrupt service routine* or *PTS routine*. Determine the instruction in your code that has the longest execution time and use that execution time in calculating interrupt latency.

interrupt service routine — A software routine that you provide to service a standard interrupt request. See also *PTS routine*.

interrupt vector — A location in *special-purpose memory* that holds the starting address of an *interrupt service routine*.

ISR — See *interrupt service routine*.

linearity errors — See *differential nonlinearity* and *nonlinearity.*

LONG-INTEGER — A 32-bit, signed variable with values from -2^{31} through $+2^{31}-1$.

LSB — 1) Least-significant bit of a byte or least-significant byte of a word.

2) In an A/D converter, the reference voltage divided by 2^n, where n is the number of bits to be converted. For a 10-bit converter with a reference voltage of 5.12 volts, one LSB is equal to 5.0 millivolts ($5.12 \div 2^{10}$).

LSW — Least-significant word of a double-word or quad-word.

maskable interrupts — All interrupts except stack overflow, unimplemented opcode, software trap, and NMI. Maskable interrupts can be disabled (masked) by the individual mask bits in the interrupt mask registers, and their servicing can be disabled by the DI (disable interrupt service) instruction. Each *maskable interrupt* can be assigned to the *PTS* for processing.

monotonic — The property of *successive approximation* converters which guarantees that increasing input voltages produce adjacent *codes* of increasing value, and that decreasing input voltages produce adjacent codes of decreasing value. (In other words, a converter is monotonic if every code change represents an input voltage change in the same direction.) Large *differential nonlinearity* errors can cause the converter to exhibit nonmonotonic behavior.

MSB — Most-significant bit of a *byte* or most-significant byte of a *word*.

MSW — Most-significant word of a double-word or quad-word.

multiplexed bus — The configuration in which the microcontroller uses both A20:0 and AD15:0 for address and also uses AD15:0 for data. See also *demultiplexed bus*.

***n*-channel FET** — A field-effect transistor with an *n*-type conducting path (channel).

***n*-type material** — Semiconductor material with introduced impurities (*doping*) causing it to have an excess of negatively charged carriers.

near constants — Constants that can be accessed with nonextended instructions. Constants in page 00H are near constants. See also *far constants.*

near data — Data that can be accessed with nonextended instructions. Data in page 00H is near data. See also *far data*.

no missing codes — An A/D converter has *no missing codes* if, for every output code, there is a unique input voltage range which produces that code only. Large *differential nonlinearity* errors can cause the converter to miss codes.

nonlinearity — The maximum deviation of *code transitions* of the *terminal-based characteristic* from the corresponding code transitions of the *ideal characteristic*.

nonmaskable interrupts — Interrupts that cannot be masked (disabled) and cannot be assigned to the PTS for processing. The nonmaskable interrupts are stack overflow, unimplemented opcode, software trap, and NMI. The DI (disable interrupt service) and EI (enable interrupt service) instructions have no effect on nonmaskable interrupts.

nonvolatile memory — Read-only memory that retains its contents when power is removed. Many MCS® 96 microcontrollers are available with either masked ROM, *EPROM*, or *OTPROM*. Consult the *Automotive Products* databook to determine which type of memory is available for a specific microcontroller.

***npn* transistor** — A transistor consisting of one part *p-type material* and two parts *n-type material*.

off-isolation — The ability of an *A/D converter* to reject (isolate) the signal on a deselected (off) output.

output/simulcapture — The specialized output-only channels of the 83C196EA's event processor array (EPA). These channels can trigger the compare event output and simultaneously capture the value of any other timer, providing easy conversion between angle and time domains.

***p*-channel FET** — A field-effect transistor with a *p*-type conducting path.

***p*-type material** — Semiconductor material with introduced impurities (*doping*) causing it to have an excess of positively charged carriers.

PC — Program counter.

phase-locked loop — (A component of the clock generation circuitry. The phase-locked loop (PLL) and the input pin (PLLEN) combine to enable the microcontroller to attain its maximum operating frequency with an external clock whose frequency is either equal to or one-half that maximum frequency or with an external oscillator whose frequency is one-half that maximum frequency.

PIC Programmable interrupt controller. The module responsible for handling interrupts that are to be serviced by *interrupt service routines* that you provide. Also called simply the *interrupt controller.*

PIH Peripheral interrupt handler. An integrated module that provides interrupt vectors for specific *EPA* interrupt requests to the *interrupt controller* or *PTS.*

PLL See *phase-locked loop*.

prioritized interrupt NMI, stack overflow, or any *maskable interrupt*. Two of the *nonmaskable interrupts* (unimplemented opcode and software trap) are not prioritized; they vector directly to the *interrupt service routine* when executed.

program memory A partition of memory where instructions can be stored for fetching and execution.

protected instruction An instruction that prevents an interrupt from being acknowledged until after the next instruction executes. The protected instructions are DI, EI, DPTS, EPTS, POPA, POPF, PUSHA, and PUSHF.

PSW Processor status word. The high byte of the PSW is the status byte, which contains one bit that globally enables or disables servicing of all maskable interrupts, one bit that enables or disables the *PTS*, and six Boolean flags that reflect the state of the current program. The low byte of the PSW is the INT_MASK register. A PUSHA or POPA instruction saves or restores both bytes (PSW + INT_MASK); a PUSHF or POPF saves or restores only the PSW.

PTS Peripheral transaction server. The microcoded hardware interrupt processor.

PTSCB See *PTS control block.*

PTS control block A block of data required for each *PTS interrupt*. The microcode executes the proper *PTS routine* based on the contents of the PTS control block.

PTS cycle The microcoded response to a **single** PTS interrupt request.

PTS interrupt Any *maskable interrupt* that is assigned to the *PTS* for interrupt processing.

PTS mode — A microcoded response that enables the *PTS* to complete a specific task quickly.

PTS routine — The entire microcoded response to multiple PTS interrupt requests. The PTS routine is controlled by the contents of the *PTS control block*.

PTS transfer — The movement of a single byte or word from the source memory location to the destination memory location.

PTS vector — A location in *special-purpose memory* that holds the starting address of a *PTS control block*.

PWM — Pulse-width modulator. A peripheral that generates waveforms with a fixed, selectable frequency and a variable duty cycle.

QUAD-WORD — An unsigned, 64-bit variable with values from 0 through 2^{64}–1. The QUAD-WORD variable is supported only as the operand for the EBMOVI instruction.

quantizing error — An unavoidable A/D conversion error that results simply from the conversion of a continuous voltage to its integer digital representation. Quantizing error is always ± 0.5 LSB and is the only error present in an ideal *A/D converter*.

RALU — Register arithmetic-logic unit. A part of the CPU that consists of the *ALU*, the *PSW*, the master *PC*, the microcode engine, a loop counter, and six registers.

repeatability error — The variation in *code transitions* when comparing a number of *actual characteristics* from the same converter on the same channel with the same temperature, voltage, and frequency conditions. The amount of repeatability error depends on the comparator's ability to resolve very similar voltages and the extent to which random noise contributes to the error.

reserved memory — A memory location that is reserved for factory use or for future expansion. Do not use a reserved memory location except to initialize it.

resolution — The number of input voltage levels that an *A/D converter* can unambiguously distinguish between. The number of useful bits of information that the converter can return.

sample capacitor — A small (2–3 pF) capacitor used in the *A/D converter* circuitry to store the input voltage on the selected input channel.

sample delay — The time period between the time that *A/D converter* receives the "start conversion" signal and the time that the *sample capacitor* is connected to the selected channel.

sample delay uncertainty — The variation in the *sample delay.*

sample time — The period of time that the *sample window* is open. (That is, the length of time that the input channel is actually connected to the *sample capacitor.*)

sample time uncertainty — The variation in the *sample time.*

sample window — The period of time that begins when the *sample capacitor* is attached to a selected channel of an *A/D converter* and ends when the sample capacitor is disconnected from the selected channel.

sampled inputs — All input pins, with the exception of RESET#, are sampled inputs. The input pin is sampled one state time before the read buffer is enabled. Sampling occurs during PH1 (while CLKOUT is low) and resolves the value (high or low) of the pin before it is presented to the internal bus. If the pin value changes during the sample time, the new value may or may not be recorded during the read.

RESET# is a level-sensitive input. EXTINT is normally a sampled input; however, the powerdown circuitry uses EXTINT as a level-sensitive input during powerdown mode.

SAR — *Successive approximation* register. A component of the *A/D converter.*

SDU — Serial debug unit, the module that enables read and write access to the code/data RAM via a dedicated serial link.

set — The "1" value of a bit or the act of giving it a "1" value. See also *clear.*

SFR — Special-function register.

SHORT-INTEGER — An 8-bit, signed variable with values from -2^7 through $+2^7-1$.

sign extension — A method for converting data to a larger format by filling the upper bit positions with the value of the sign. This conversion preserves the positive or negative value of signed integers.

sink current — Current flowing **into** a device to ground. Always a positive value.

SOM — Stack overflow module. The peripheral that monitors the stack pointer and generates a nonmaskable interrupt request if it crosses specified boundaries.

source current — Current flowing **out of** a device from V_{CC}. Always a negative value.

SP — Stack pointer.

special interrupt — Any of the three *nonmaskable interrupts* (unimplemented opcode, software trap, or NMI).

special-purpose memory — A partition of memory used for storing the *interrupt vectors*, *PTS vectors*, chip configuration bytes, and several reserved locations.

standard interrupt — Any *maskable interrupt* that is assigned to the *interrupt controller* for processing by an *interrupt service routine*.

stack overflow module — The peripheral that monitors the stack pointer and generates a nonmaskable interrupt request if it crosses specified boundaries.

state time (or state) — The basic time unit of the microcontroller; the combined period of the two internal timing signals, PH1 and PH2. Because the microcontroller can operate at many frequencies, this manual defines time requirements in terms of *state times* rather than in specific units of time.

successive approximation — An A/D conversion method that uses a binary search to arrive at the best digital representation of an analog input.

t — Lowercase “t” represents the period of the internal clock.

temperature coefficient — Change in the stated variable for each degree Centigrade of temperature change.

temperature drift — The change in a specification due to a change in temperature. Temperature drift can be calculated by using the *temperature coefficient* for the specification.

terminal-based characteristic — An *actual characteristic* that has been translated and scaled to remove *zero-offset error* and *full-scale error.* A terminal-based characteristic resembles an *actual characteristic* with zero-offset error and full-scale error removed.

transfer function — A graph of output *code* versus input voltage; the *characteristic* of the *A/D converter.*

transfer function errors — Errors inherent in an analog-to-digital conversion process: *quantizing error, zero-offset error, full-scale error, differential nonlinearity,* and *nonlinearity.* Errors that are hardware-dependent, rather than being inherent in the process itself, include *feedthrough, repeatability, channel-to-channel matching, off-isolation,* and V_{CC} *rejection* errors.

UART — Universal asynchronous receiver and transmitter. A part of the serial I/O port.

V_{CC} rejection — The property of an A/D converter that causes it to ignore (reject) changes in V_{CC} so that the *actual characteristic* is unaffected by those changes. The effectiveness of V_{CC} *rejection* is measured by the ratio of the change in V_{CC} to the change in the *actual characteristic*.

wait state — Time spent waiting for an operation to take place. Wait states are added to external bus cycles to allow a slow memory device to respond to a request from the microcontroller.

watchdog timer — An internal timer that resets the microcontroller if software fails to respond before the timer overflows.

WDT — *Watchdog timer.* An internal timer that resets the microcontroller if software fails to respond before the timer overflows.

word — Any 16-bit unit of data.

WORD — An unsigned, 16-bit variable with values from 0 through 2^{16}–1.

zero extension A method for converting data to a larger format by filling the upper bit positions with zeros.

zero-offset error An ideal *A/D converter*'s first *code transition* occurs when the input voltage is 0.5 LSB. Zero-offset error is the difference between 0.5 LSB and the actual input voltage that triggers an A/D converter's first code transition.

Index

A

B

C

D

E

F

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z