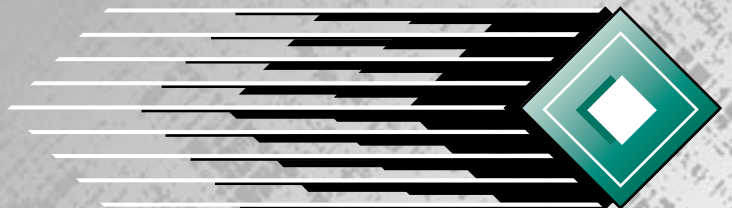


8XC196MC, 8XC196MD, 8XC196MH Microcontroller User's Manual



intel®



**8XC196MC,
8XC196MD, 8XC196MH
Microcontroller
User's Manual**

August 2004

Order Number 272181-003

Information in this document is provided solely to enable use of Intel products. Intel assumes no liability whatsoever, including infringement of any patent or copyright, for sale and use of Intel products except as provided in Intel's Terms and Conditions of Sale for such products.

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

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

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

MDS is an ordering code only and is not used as a product name or trademark of Intel Corporation.

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

*Other brands and names are the property of their respective owners.

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

Intel Corporation
Literature Sales
P.O. Box 7641
Mt. Prospect, IL 60056-7641
or call 1-800-879-4683

CHAPTER 1
GUIDE TO THIS MANUAL

1.1	MANUAL CONTENTS	1-1
1.2	NOTATIONAL CONVENTIONS AND TERMINOLOGY	1-3
1.3	RELATED DOCUMENTS	1-5
1.4	ELECTRONIC SUPPORT SYSTEMS	1-8
1.4.4	World Wide Web	1-11
1.5	TECHNICAL SUPPORT	1-11
1.6	PRODUCT LITERATURE	1-11

CHAPTER 2
ARCHITECTURAL OVERVIEW

2.1	TYPICAL APPLICATIONS	2-1
2.2	MICROCONTROLLER FEATURES	2-1
2.3	FUNCTIONAL OVERVIEW	2-2
2.3.1	CPU Control	2-4
2.3.2	Register File	2-4
2.3.3	Register Arithmetic-logic Unit (RALU)	2-4
2.3.3.1	Code Execution	2-5
2.3.3.2	Instruction Format	2-5
2.3.4	Memory Interface Unit	2-6
2.3.5	Interrupt Service	2-6
2.4	INTERNAL TIMING	2-7
2.5	INTERNAL PERIPHERALS	2-8
2.5.1	I/O Ports	2-9
2.5.2	Serial I/O (SIO) Port	2-9
2.5.3	Event Processor Array (EPA) and Timer/Counters	2-10
2.5.4	Pulse-width Modulator (PWM)	2-10
2.5.5	Frequency Generator	2-10
2.5.6	Waveform Generator	2-10
2.5.7	Analog-to-digital Converter	2-11
2.5.8	Watchdog Timer	2-11
2.6	SPECIAL OPERATING MODES	2-11
2.6.1	Reducing Power Consumption	2-11
2.6.2	Testing the Printed Circuit Board	2-11
2.6.3	Programming the Nonvolatile Memory	2-12

CHAPTER 3**PROGRAMMING CONSIDERATIONS**

3.1	OVERVIEW OF THE INSTRUCTION SET	3-1
3.1.1	BIT Operands	3-2
3.1.2	BYTE Operands	3-2
3.1.3	SHORT-INTEGER Operands	3-2
3.1.4	WORD Operands	3-2
3.1.5	INTEGER Operands	3-3
3.1.6	DOUBLE-WORD Operands	3-3
3.1.7	LONG-INTEGER Operands	3-4
3.1.8	Converting Operands	3-4
3.1.9	Conditional Jumps	3-4
3.1.10	Floating Point Operations	3-4
3.2	ADDRESSING MODES	3-5
3.2.1	Direct Addressing	3-6
3.2.2	Immediate Addressing	3-6
3.2.3	Indirect Addressing	3-6
3.2.3.1	Indirect Addressing with Autoincrement	3-7
3.2.3.2	Indirect Addressing with the Stack Pointer	3-7
3.2.4	Indexed Addressing	3-7
3.2.4.1	Short-indexed Addressing	3-7
3.2.4.2	Long-indexed Addressing	3-8
3.2.4.3	Zero-indexed Addressing	3-8
3.3	ASSEMBLY LANGUAGE ADDRESSING MODE SELECTIONS	3-9
3.3.1	Direct Addressing	3-9
3.3.2	Indexed Addressing	3-9
3.4	SOFTWARE STANDARDS AND CONVENTIONS	3-9
3.4.1	Using Registers	3-9
3.4.2	Addressing 32-bit Operands	3-10
3.4.3	Linking Subroutines	3-10
3.5	SOFTWARE PROTECTION FEATURES AND GUIDELINES	3-11

CHAPTER 4**MEMORY PARTITIONS**

4.1	MEMORY PARTITIONS	4-1
4.1.1	External Devices (Memory or I/O)	4-1
4.1.2	Program and Special-purpose Memory	4-1
4.1.3	Program Memory	4-2
4.1.4	Special-purpose Memory	4-3
4.1.4.1	Reserved Memory Locations	4-3
4.1.4.2	Interrupt and PTS Vectors	4-3
4.1.4.3	Security Key	4-4
4.1.4.4	Chip Configuration Bytes (CCBs)	4-4
4.1.5	Special-function Registers (SFRs)	4-4

4.1.5.1	Memory-mapped SFRs	4-5
4.1.5.2	Peripheral SFRs	4-5
4.1.6	Register File	4-9
4.1.6.1	General-purpose Register RAM	4-10
4.1.6.2	Stack Pointer (SP)	4-10
4.1.6.3	CPU Special-function Registers (SFRs)	4-11
4.2	WINDOWING	4-12
4.2.1	Selecting a Window	4-13
4.2.2	Addressing a Location Through a Window	4-14
4.2.2.1	32-byte Windowing Example	4-16
4.2.2.2	64-byte Windowing Example	4-16
4.2.2.3	128-byte Windowing Example	4-16
4.2.2.4	Unsupported Locations Windowing Example	4-16
4.2.2.5	Using the Linker Locator to Set Up a Window	4-17
4.2.3	Windowing and Addressing Modes	4-19

CHAPTER 5

STANDARD AND PTS INTERRUPTS

5.1	OVERVIEW OF INTERRUPTS.....	5-1
5.2	INTERRUPT SIGNALS AND REGISTERS	5-3
5.3	INTERRUPT SOURCES AND PRIORITIES.....	5-4
5.3.1	Special Interrupts	5-6
5.3.1.1	Unimplemented Opcode	5-6
5.3.1.2	Software Trap	5-6
5.3.1.3	NMI	5-6
5.3.2	External Interrupt Pin	5-6
5.3.3	Multiplexed Interrupt Sources	5-7
5.3.4	End-of-PTS Interrupts	5-9
5.4	INTERRUPT LATENCY.....	5-9
5.4.1	Situations that Increase Interrupt Latency	5-9
5.4.2	Calculating Latency	5-10
5.4.2.1	Standard Interrupt Latency	5-10
5.4.2.2	PTS Interrupt Latency	5-11
5.5	PROGRAMMING THE INTERRUPTS.....	5-12
5.5.1	Modifying Interrupt Priorities	5-18
5.5.2	Determining the Source of an Interrupt	5-20
5.6	INITIALIZING THE PTS CONTROL BLOCKS.....	5-24
5.6.1	Specifying the PTS Count	5-25
5.6.2	Selecting the PTS Mode	5-27
5.6.3	Single Transfer Mode	5-27
5.6.4	Block Transfer Mode	5-30
5.6.5	A/D Scan Mode	5-32
5.6.5.1	A/D Scan Mode Cycles	5-35
5.6.5.2	A/D Scan Mode Example 1	5-35
5.6.5.3	A/D Scan Mode Example 2	5-37

5.6.6	Serial I/O Modes	5-37
5.6.6.1	Synchronous SIO Transmit Mode Example	5-43
5.6.6.2	Synchronous SIO Receive Mode Example	5-47
5.6.6.3	Asynchronous SIO Transmit Mode Example	5-50
5.6.6.4	Asynchronous SIO Receive Mode Example	5-55

CHAPTER 6 I/O PORTS

6.1	I/O PORTS OVERVIEW	6-1
6.2	INPUT-ONLY PORTS 1 (MC, MD ONLY) AND 0.....	6-2
6.2.1	Standard Input-only Port Operation	6-3
6.2.2	Standard Input-only Port Considerations	6-4
6.3	BIDIRECTIONAL PORTS 1 (MH ONLY), 2, 5, AND 7 (MD ONLY).....	6-4
6.3.1	Bidirectional Port Operation	6-6
6.3.2	Bidirectional Port Pin Configurations	6-9
6.3.3	Bidirectional Port Pin Configuration Example	6-11
6.3.4	Bidirectional Port Considerations	6-12
6.4	BIDIRECTIONAL PORTS 3 AND 4 (ADDRESS/DATA BUS).....	6-14
6.4.1	Bidirectional Ports 3 and 4 (Address/Data Bus) Operation	6-15
6.4.2	Using Ports 3 and 4 as I/O	6-16
6.4.3	Design Considerations for Ports 3 and 4	6-16
6.5	STANDARD OUTPUT-ONLY PORT 6	6-16
6.5.1	Output-only Port Operation	6-17
6.5.2	Configuring Output-only Port Pins	6-17

CHAPTER 7 SERIAL I/O (SIO) PORT

7.1	SERIAL I/O (SIO) PORT FUNCTIONAL OVERVIEW	7-1
7.2	SERIAL I/O PORT SIGNALS AND REGISTERS	7-2
7.3	SERIAL PORT MODES.....	7-4
7.3.1	Synchronous Modes (Modes 0 and 4)	7-5
7.3.1.1	Mode 0	7-5
7.3.1.2	Mode 4	7-6
7.3.2	Asynchronous Modes (Modes 1, 2, and 3)	7-7
7.3.2.1	Mode 1	7-7
7.3.2.2	Mode 2	7-8
7.3.2.3	Mode 3	7-9
7.3.2.4	Mode 2 and 3 Timings	7-9
7.3.2.5	Multiprocessor Communications	7-9
7.4	PROGRAMMING THE SERIAL PORT	7-10
7.4.1	Configuring the Serial Port Pins	7-10
7.4.2	Programming the Control Register	7-10
7.4.3	Programming the Baud Rate and Clock Source	7-12
7.4.4	Enabling the Serial Port Interrupts	7-14

7.4.5 Determining Serial Port Status7-15

CHAPTER 8

FREQUENCY GENERATOR

8.1 FUNCTIONAL OVERVIEW..... 8-1

8.2 PROGRAMMING THE FREQUENCY GENERATOR 8-3

 8.2.1 Configuring the Output8-3

 8.2.2 Programming the Frequency8-3

 8.2.3 Determining the Current Value of the Down-counter8-4

8.3 APPLICATION EXAMPLE 8-4

CHAPTER 9

WAVEFORM GENERATOR

9.1 WAVEFORM GENERATOR FUNCTIONAL OVERVIEW..... 9-1

9.2 WAVEFORM GENERATOR SIGNALS AND REGISTERS..... 9-3

9.3 WAVEFORM GENERATOR OPERATION..... 9-4

 9.3.1 Timebase Generator9-4

 9.3.2 Phase Driver Channels9-5

 9.3.3 Control and Protection Circuitry9-5

 9.3.4 Register Buffering and Synchronization9-6

 9.3.5 Operating Modes9-7

 9.3.5.1 Center-aligned Modes9-9

 9.3.5.2 Edge-Aligned Modes9-10

9.4 PROGRAMMING THE WAVEFORM GENERATOR..... 9-12

 9.4.1 Configuring the Outputs9-12

 9.4.2 Controlling the Protection Circuitry and EXTINT Interrupt Generation9-15

 9.4.3 Specifying the Carrier Period and Duty Cycle9-16

 9.4.4 Specifying the Operating Mode and Dead Time and Starting the Counter9-17

9.5 DETERMINING THE WAVEFORM GENERATOR'S STATUS 9-19

9.6 ENABLING THE WAVEFORM GENERATOR INTERRUPTS..... 9-19

9.7 DESIGN CONSIDERATIONS..... 9-20

 9.7.1 Dead Time and Duty Cycle9-20

 9.7.2 EXTINT Interrupts and Protection Circuitry9-21

9.8 PROGRAMMING EXAMPLE 9-21

CHAPTER 10

PULSE-WIDTH MODULATOR

10.1 PWM FUNCTIONAL OVERVIEW..... 10-1

10.2 PWM SIGNALS AND REGISTERS 10-2

10.3 PWM OPERATION..... 10-3

10.4 PROGRAMMING THE FREQUENCY AND PERIOD..... 10-4

10.5 PROGRAMMING THE DUTY CYCLE 10-6

 10.5.1 Sample Calculations 10-7

10.5.2	Reading the Current Value of the Down-counter	10-7
10.5.3	Enabling the PWM Outputs	10-8
10.5.4	Generating Analog Outputs	10-10

CHAPTER 11

EVENT PROCESSOR ARRAY (EPA)

11.1	EPA FUNCTIONAL OVERVIEW	11-1
11.2	EPA AND TIMER/COUNTER SIGNALS AND REGISTERS	11-2
11.3	TIMER/COUNTER FUNCTIONAL OVERVIEW.....	11-5
11.3.1	Cascade Mode (Timer 2 Only)	11-7
11.3.2	Quadrature Clocking Modes	11-7
11.4	EPA CHANNEL FUNCTIONAL OVERVIEW	11-9
11.4.1	Operating in Capture Mode	11-10
11.4.1.1	EPA Overruns	11-12
11.4.1.2	Preventing EPA Overruns	11-13
11.4.2	Operating in Compare Mode	11-13
11.4.2.1	Generating a Low-speed PWM Output	11-13
11.4.2.2	Generating the Highest-speed PWM Output	11-14
11.5	PROGRAMMING THE EPA AND TIMER/COUNTERS.....	11-15
11.5.1	Configuring the EPA and Timer/Counter Signals	11-15
11.5.2	Programming the Timers	11-15
11.5.3	Programming the Capture/Compare Channels	11-18
11.5.4	Programming the Compare-only Channels	11-22
11.6	ENABLING THE EPA INTERRUPTS	11-23
11.7	DETERMINING EVENT STATUS.....	11-24

CHAPTER 12

ANALOG-TO-DIGITAL (A/D) CONVERTER

12.1	A/D CONVERTER FUNCTIONAL OVERVIEW	12-1
12.2	A/D CONVERTER SIGNALS AND REGISTERS	12-2
12.3	A/D CONVERTER OPERATION	12-3
12.4	PROGRAMMING THE A/D CONVERTER	12-4
12.4.1	Programming the A/D Test Register	12-5
12.4.2	Programming the A/D Result Register (for Threshold Detection Only)	12-5
12.4.3	Programming the A/D Time Register	12-6
12.4.4	Programming the A/D Command Register	12-7
12.4.5	Enabling the A/D Interrupt	12-8
12.5	DETERMINING A/D STATUS AND CONVERSION RESULTS	12-9
12.6	DESIGN CONSIDERATIONS.....	12-10
12.6.1	Designing External Interface Circuitry	12-10
12.6.1.1	Minimizing the Effect of High Input Source Resistance	12-11
12.6.1.2	Suggested A/D Input Circuit	12-12
12.6.1.3	Analog Ground and Reference Voltages	12-12

12.6.1.4 Using Mixed Analog and Digital Inputs12-13
 12.6.2 Understanding A/D Conversion Errors12-13

CHAPTER 13

MINIMUM HARDWARE CONSIDERATIONS

13.1 MINIMUM CONNECTIONS 13-1
 13.1.1 Unused Inputs 13-2
 13.1.2 I/O Port Pin Connections 13-2
 13.2 APPLYING AND REMOVING POWER 13-4
 13.3 NOISE PROTECTION TIPS 13-4
 13.4 THE ON-CHIP OSCILLATOR CIRCUITRY 13-5
 13.5 USING AN EXTERNAL CLOCK SOURCE 13-7
 13.6 RESETTING THE DEVICE 13-8
 13.6.1 Generating an External Reset 13-10
 13.6.2 Issuing the Reset (RST) Instruction 13-12
 13.6.3 Issuing an Illegal IDLPD Key Operand 13-12
 13.6.4 Generating Wait States 13-12
 13.6.5 Enabling the Watchdog Timer 13-12

CHAPTER 14

SPECIAL OPERATING MODES

14.1 SPECIAL OPERATING MODE SIGNALS AND REGISTERS 14-1
 14.2 REDUCING POWER CONSUMPTION 14-3
 14.3 IDLE MODE 14-4
 14.4 POWERDOWN MODE 14-5
 14.4.1 Enabling and Disabling Powerdown Mode 14-5
 14.4.2 Entering Powerdown Mode 14-6
 14.4.3 Exiting Powerdown Mode 14-6
 14.4.3.1 Driving the V_{pp} Pin Low 14-6
 14.4.3.2 Generating a Hardware Reset 14-6
 14.4.3.3 Asserting the External Interrupt Signal 14-7
 14.4.3.4 Selecting R_1 and C_1 14-8
 14.5 ONCE MODE 14-10
 14.6 RESERVED TEST MODES 14-11

CHAPTER 15

INTERFACING WITH EXTERNAL MEMORY

15.1 EXTERNAL MEMORY INTERFACE SIGNALS AND REGISTERS 15-1
 15.2 CHIP CONFIGURATION REGISTERS AND CHIP CONFIGURATION BYTES 15-5
 15.3 BUS WIDTH AND MULTIPLEXING 15-10
 15.3.1 Timing Requirements for BUSWIDTH 15-13
 15.3.2 16-bit Bus Timings 15-14
 15.3.3 8-bit Bus Timings 15-16

15.4	WAIT STATES (READY CONTROL).....	15-17
15.5	BUS-CONTROL MODES.....	15-21
15.5.1	Standard Bus-control Mode	15-22
15.5.2	Write Strobe Mode	15-25
15.5.3	Address Valid Strobe Mode	15-27
15.5.4	Address Valid with Write Strobe Mode	15-30
15.6	SYSTEM BUS AC TIMING SPECIFICATIONS	15-31
15.6.1	Explanation of AC Symbols	15-33
15.6.2	AC Timing Definitions	15-33

CHAPTER 16

PROGRAMMING THE NONVOLATILE MEMORY

16.1	PROGRAMMING METHODS.....	16-1
16.2	OTPROM MEMORY MAP	16-2
16.3	SECURITY FEATURES.....	16-3
16.3.1	Controlling Access to Internal Memory	16-3
16.3.1.1	Controlling Access to the OTPROM During Normal Operation	16-4
16.3.1.2	Controlling Access to the OTPROM During Programming Modes	16-4
16.3.2	Controlling Fetches from External Memory	16-6
16.4	PROGRAMMING PULSE WIDTH	16-8
16.5	MODIFIED QUICK-PULSE ALGORITHM.....	16-9
16.6	PROGRAMMING MODE PINS.....	16-11
16.7	ENTERING PROGRAMMING MODES	16-13
16.7.1	Selecting the Programming Mode	16-13
16.7.2	Power-up and Power-down Sequences	16-14
16.7.2.1	Power-up Sequence	16-14
16.7.2.2	Power-down Sequence	16-14
16.8	SLAVE PROGRAMMING MODE.....	16-15
16.8.1	Reading the Signature Word and Programming Voltages	16-15
16.8.2	Slave Programming Circuit and Memory Map	16-16
16.8.3	Operating Environment	16-17
16.8.4	Slave Programming Routines	16-19
16.8.5	Timing Mnemonics	16-24
16.9	AUTO PROGRAMMING MODE	16-25
16.9.1	Auto Programming Circuit and Memory Map	16-25
16.9.2	Operating Environment	16-27
16.9.3	Auto Programming Routine	16-27
16.9.4	Auto Programming Procedure	16-29
16.9.5	ROM-dump Mode	16-30
16.10	PCCB AND UPROM PROGRAMMING (8XC196MH ONLY)	16-30
16.11	RUN-TIME PROGRAMMING	16-32

**APPENDIX A
INSTRUCTION SET REFERENCE**

**APPENDIX B
SIGNAL DESCRIPTIONS**

B.1	SIGNAL NAME CHANGES.....	B-1
B.2	FUNCTIONAL GROUPINGS OF SIGNALS	B-1
B.3	SIGNAL DESCRIPTIONS.....	B-12
B.4	DEFAULT CONDITIONS.....	B-22

**APPENDIX C
REGISTERS**

GLOSSARY

INDEX

FIGURES

Figure	Page
2-1	8XC196Mx Block Diagram 2-3
2-2	Block Diagram of the Core 2-3
2-3	Clock Circuitry 2-7
2-4	Internal Clock Phases 2-8
4-1	Register File Memory Map 4-9
4-2	Windowing 4-12
4-3	Window Selection (WSR) Register 4-13
5-1	Flow Diagram for PTS and Standard Interrupts 5-2
5-2	Waveform Generator Protection Circuitry 5-7
5-3	Flow Diagram for the OVRTM Interrupt 5-8
5-4	Standard Interrupt Response Time 5-11
5-5	PTS Interrupt Response Time 5-11
5-6	PTS Select (PTSSEL) Register 5-14
5-7	Interrupt Mask (INT_MASK) Register 5-15
5-8	Interrupt Mask 1 (INT_MASK1) Register 5-16
5-9	Peripheral Interrupt Mask (PI_MASK) Register 5-17
5-10	Interrupt Pending (INT_PEND) Register 5-21
5-11	Interrupt Pending 1 (INT_PEND1) Register 5-22
5-12	Peripheral Interrupt Pending (PI_PEND) Register 5-23
5-13	PTS Control Blocks 5-25
5-14	PTS Service (PTSSRV) Register 5-26
5-15	PTS Mode Selection Bits (PTSCON Bits 7:5) 5-27
5-16	PTS Control Block — Single Transfer Mode 5-28
5-17	PTS Control Block — Block Transfer Mode 5-31
5-18	PTS Control Block — A/D Scan Mode 5-33
5-19	PTS Control Block 1 — Serial I/O Mode 5-38
5-20	PTS Control Block 2 — Serial I/O Mode 5-41
5-21	Synchronous SIO Transmit Mode Timing 5-43
5-22	Synchronous SIO Transmit Mode — End-of-PTS Interrupt Routine Flowchart 5-46
5-23	Synchronous SIO Receive Timing 5-47
5-24	Synchronous SIO Receive Mode — End-of-PTS Interrupt Routine Flowchart 5-50
5-25	Asynchronous SIO Transmit Timing 5-51
5-26	Asynchronous SIO Transmit Mode — End-of-PTS Interrupt Routine Flowchart 5-54
5-27	Asynchronous SIO Receive Timing 5-55
5-28	Asynchronous SIO Receive Mode — End-of-PTS Interrupt Routine Flowchart 5-58
6-1	Standard Input-only Port Structure 6-3
6-2	Bidirectional Port Structure 6-8
6-3	Address/Data Bus (Ports 3 and 4) Structure 6-15
6-4	Output-only Port 6-18
6-5	Port 6 Output Configuration (WG_OUTPUT) Register 6-18
7-1	SIO Block Diagram 7-1
7-2	Typical Shift Register Circuit for Mode 0 7-5
7-3	Mode 0 Timing 7-6
7-4	Serial Port Frames for Mode 1 7-8

FIGURES

Figure	Page
7-5	Serial Port Frames in Mode 2 and 3.....7-9
7-6	Serial Port Control (SPx_CON) Register.....7-10
7-7	Serial Port x Baud Rate (SPx_BAUD) Register.....7-12
7-8	Serial Port Status (SPx_STATUS) Register.....7-15
8-1	Frequency Generator Block Diagram.....8-1
8-2	Frequency (FREQ_GEN) Register.....8-3
8-3	Frequency Generator Count (FREQ_CNT) Register.....8-4
8-4	Infrared Remote Control Application Block Diagram.....8-5
8-5	Data Encoding Example.....8-5
9-1	Waveform Generator Block Diagram.....9-2
9-2	Dead-time Generator Circuitry.....9-5
9-3	Protection Circuitry.....9-6
9-4	Center-aligned Modes — Counter Operation.....9-9
9-5	Center-aligned Modes — Output Operation.....9-10
9-6	Edge-aligned Modes — Counter Operation.....9-11
9-7	Edge-aligned Modes — Output Operation.....9-11
9-8	WG Output Configuration (WG_OUTPUT) Register.....9-13
9-9	Waveform Generator Protection (WG_PROTECT) Register.....9-15
9-10	Waveform Generator Reload (WG_RELOAD) Register.....9-16
9-11	Phase Compare (WG_COMPx) Register.....9-17
9-12	Waveform Generator Control (WG_CONTROL) Register.....9-18
9-13	Waveform Generator Counter (WG_COUNTER) Register.....9-19
9-14	Effect of Dead Time on Duty Cycle.....9-20
10-1	PWM Block Diagram.....10-2
10-2	PWM Output Waveforms.....10-4
10-3	PWM Period (PWM_PERIOD) Register.....10-6
10-4	PWM Control (PWMx_CONTROL) Register.....10-7
10-5	PWM Count (PWM_COUNT) Register.....10-8
10-6	Waveform Generator Output Configuration (WG_OUTPUT) Register.....10-9
10-7	D/A Buffer Block Diagram.....10-10
10-8	PWM to Analog Conversion Circuitry.....10-10
11-1	EPA Block Diagram.....11-2
11-2	EPA Timer/Counters.....11-6
11-3	Quadrature Mode Interface.....11-8
11-4	Quadrature Mode Timing and Count.....11-9
11-5	A Single EPA Capture/Compare Channel.....11-10
11-6	EPA Simplified Input-capture Structure.....11-11
11-7	Valid EPA Input Events.....11-11
11-8	Timer 1 Control (T1CONTROL) Register.....11-16
11-9	Timer 2 Control (T2CONTROL) Register.....11-17
11-10	EPA Control (EPAx_CON) Registers.....11-19
11-11	EPA Compare Control (COMPx_CON) Registers.....11-22
12-1	A/D Converter Block Diagram.....12-1
12-2	A/D Test (AD_TEST) Register.....12-5

FIGURES

Figure	Page
12-3	A/D Result (AD_RESULT) Register — Write Format 12-6
12-4	A/D Time (AD_TIME) Register 12-7
12-5	A/D Command (AD_COMMAND) Register 12-8
12-6	A/D Result (AD_RESULT) Register — Read Format 12-9
12-7	Idealized A/D Sampling Circuitry 12-10
12-8	Suggested A/D Input Circuit 12-12
12-9	Ideal A/D Conversion Characteristic 12-15
12-10	Actual and Ideal A/D Conversion Characteristics 12-16
12-11	Terminal-based A/D Conversion Characteristic 12-18
13-1	Minimum Hardware Connections 13-3
13-2	Power and Return Connections 13-4
13-3	On-chip Oscillator Circuit 13-5
13-4	External Crystal Connections 13-6
13-5	External Clock Connections 13-7
13-6	External Clock Drive Waveforms 13-7
13-7	Reset Timing Sequence 13-8
13-8	General Configuration Register (GEN_CON) 13-9
13-9	Internal Reset Circuitry 13-10
13-10	Minimum Reset Circuit 13-11
13-11	Example of a System Reset Circuit 13-11
14-1	Clock Control During Power-saving Modes 14-4
14-2	Power-up and Power-down Sequence When Using an External Interrupt 14-7
14-3	External RC Circuit 14-8
14-4	Typical Voltage on the V _{PP} Pin While Exiting Powerdown 14-9
15-1	Chip Configuration 0 (CCR0) Register 15-7
15-2	Chip Configuration 1 (CCR1) Register 15-9
15-3	Multiplexing and Bus Width Options 15-11
15-4	BUSWIDTH Timing Diagram (8XC196MC, MD) 15-12
15-5	BUSWIDTH Timing Diagram (8XC196MH) 15-12
15-6	Timings for 16-bit Buses 15-15
15-7	Timings for 8-bit Buses 15-17
15-8	READY Timing Diagram — One Wait State (8XC196MC, MD) 15-19
15-9	READY Timing Diagram — One Wait State (8XC196MH) 15-20
15-10	Standard Bus Control 15-22
15-11	Decoding WRL# and WRH# 15-22
15-12	8-bit System with Flash and RAM 15-23
15-13	16-bit System with Dynamic Bus Width 15-24
15-14	Write Strobe Mode 15-25
15-15	16-bit System with Writes to Byte-wide RAMs 15-26
15-16	Address Valid Strobe Mode 15-27
15-17	Comparison of ALE and ADV# Bus Cycles 15-27
15-18	8-bit System with Flash 15-28
15-19	16-bit System with EPROM 15-29
15-20	Timings of Address Valid with Write Strobe Mode 15-30

FIGURES

Figure	Page
15-21 16-bit System with RAM	15-31
15-22 System Bus Timing	15-32
16-1 Unerasable PROM (USFR) Register.....	16-7
16-2 Programming Pulse Width (PPW) Register.....	16-8
16-3 Modified Quick-pulse Algorithm.....	16-10
16-4 Pin Functions in Programming Modes.....	16-11
16-5 Slave Programming Circuit.....	16-16
16-6 Chip Configuration Registers (CCRs).....	16-18
16-7 Address/Command Decoding Routine	16-20
16-8 Program Word Routine.....	16-21
16-9 Program Word Waveform.....	16-22
16-10 Dump Word Routine.....	16-23
16-11 Dump Word Waveform	16-24
16-12 Auto Programming Circuit	16-26
16-13 Auto Programming Routine	16-28
16-14 PCCB and UPROM Programming Circuit	16-31
16-15 Run-time Programming Code Example.....	16-33
B-1 8XC196MC 64-lead Shrink DIP (SDIP) Package.....	B-3
B-2 8XC196MC 84-lead PLCC Package	B-4
B-3 8XC196MC 80-lead Shrink EIAJ/QFP Package.....	B-5
B-4 8XC196MD 84-lead PLCC Package	B-7
B-5 8XC196MD 80-lead Shrink EIAJ/QFP Package.....	B-8
B-6 8XC196MH 64-lead Shrink DIP (SDIP) Package.....	B-10
B-7 8XC196MH 84-lead PLCC Package	B-11
B-8 8XC196MH 80-lead Shrink EIAJ/QFP Package.....	B-12

TABLES

Table	Page
1-1	Handbooks and Product Information 1-6
1-2	Application Notes, Application Briefs, and Article Reprints 1-6
1-3	MCS® 96 Microcontroller Datasheets (Commercial/Express) 1-7
1-4	MCS® 96 Microcontroller Datasheets (Automotive) 1-7
1-5	MCS® 96 Microcontroller Quick References 1-8
2-1	Features of the 8XC196Mx Product Family 2-2
2-2	State Times at Various Frequencies 2-8
3-1	Operand Type Definitions 3-1
3-2	Equivalent Operand Types for Assembly and C Programming Languages 3-2
3-3	Definition of Temporary Registers 3-6
4-1	Memory Map 4-2
4-2	Special-purpose Memory Addresses 4-3
4-3	Memory-mapped SFRs 4-5
4-4	Peripheral SFRs — 8XC196MC 4-6
4-5	Peripheral SFRs — 8XC196MD 4-7
4-6	Peripheral SFRs — 8XC196MH 4-8
4-7	Register File Memory Addresses 4-10
4-8	CPU SFRs 4-11
4-9	Selecting a Window of Peripheral SFRs 4-13
4-10	Selecting a Window of the Upper Register File 4-14
4-11	Windows 4-15
4-12	Windowed Base Addresses 4-15
5-1	Interrupt Signals 5-3
5-2	Interrupt and PTS Control and Status Registers 5-3
5-3	Interrupt Sources, Vectors, and Priorities 5-5
5-4	Execution Times for PTS Cycles 5-12
5-5	Single Transfer Mode PTSCB 5-30
5-6	Block Transfer Mode PTSCB 5-30
5-7	A/D Scan Mode Command/Data Table 5-34
5-8	Command/Data Table (Example 1) 5-36
5-9	A/D Scan Mode PTSCB (Example 1) 5-36
5-10	Command/Data Table (Example 2) 5-37
5-11	A/D Scan Mode PTSCB (Example 2) 5-37
5-13	SSIO Transmit Mode PTSCBs 5-45
5-14	SSIO Receive Mode PTSCBs 5-48
5-15	ASIO Transmit Mode PTSCBs 5-52
5-16	ASIO Receive Mode PTSCBs 5-56
6-1	Device I/O Ports 6-1
6-2	Standard Input-only Port Pins 6-2
6-3	Input-only Port Registers 6-3
6-4	Bidirectional Port Pins 6-5
6-5	Bidirectional Port Control and Status Registers 6-6
6-6	Logic Table for Bidirectional Ports in I/O Mode 6-9
6-7	Logic Table for Bidirectional Ports in Special-function Mode 6-9

TABLES

Table	Page
6-8 Control Register Values for Each Configuration.....	6-11
6-9 Port Configuration Example	6-11
6-10 Port Pin States After Reset and After Example Code Execution.....	6-12
6-11 Ports 3 and 4 Pins	6-14
6-12 Ports 3 and 4 Control and Status Registers	6-14
6-13 Logic Table for Ports 3 and 4 as Open-drain I/O.....	6-16
6-14 Standard Output-only Port Pins.....	6-17
6-15 Output-only Port Control Register	6-17
7-1 Serial Port Signals.....	7-2
7-2 Serial Port Control and Status Registers.....	7-2
7-3 SP _X _BAUD Values When Using XTAL1 at 16 MHz.....	7-14
8-1 Frequency Generator Signal	8-2
8-2 Frequency Generator Control and Status Registers	8-2
9-1 Waveform Generator Signals	9-3
9-2 Waveform Generator Control and Status Registers.....	9-3
9-3 Operation in Center-aligned and Edge-aligned Modes	9-8
9-4 Register Updates.....	9-8
9-5 Output Configuration	9-12
10-1 PWM Signals.....	10-2
10-2 PWM Control and Status Registers.....	10-3
10-3 PWM Output Frequencies (F_{PWM}).....	10-5
10-4 PWM Output Alternate Functions	10-8
11-1 EPA Channels	11-1
11-2 EPA and Timer/Counter Signals.....	11-2
11-3 EPA Control and Status Registers	11-3
11-4 Quadrature Mode Truth Table	11-8
11-5 Action Taken When a Valid Edge Occurs	11-12
11-6 Example EPA Control Register Settings for Channels 1, 3, or 5.....	11-18
12-1 A/D Converter Pins.....	12-2
12-2 A/D Control and Status Registers.....	12-2
13-1 Minimum Required Signals.....	13-1
13-2 I/O Port Configuration Guide	13-2
13-3 Selecting the Watchdog Reset Interval (8XC196MH only).....	13-13
14-1 Operating Mode Control Signals	14-1
14-2 Operating Mode Control and Status Registers.....	14-2
15-1 External Memory Interface Signals.....	15-1
15-2 External Memory Interface Registers	15-4
15-3 Register Settings for Configuring External Memory Interface Signals.....	15-5
15-4 BUSWIDTH Signal Timing Definitions.....	15-13
15-5 READY Signal Timing Definitions.....	15-20
15-6 Bus-control Modes	15-21
15-7 AC Timing Symbol Definitions.....	15-33
15-8 External Memory Systems Must Meet These Specifications.....	15-33
15-9 Microcontroller Meets These Specifications.....	15-34

TABLES

Table	Page
16-1	87C196Mx OTPROM Memory Map 16-3
16-2	Memory Protection for Normal Operating Mode..... 16-4
16-3	Memory Protection Options for Programming Modes 16-5
16-4	UPROM Programming Values and Locations for Slave Mode 16-7
16-5	Example PPW_VALUE Calculations 16-9
16-6	Pin Descriptions 16-11
16-7	PMODE Values 16-13
16-8	Device Signature Word and Programming Voltages 16-16
16-9	Slave Programming Mode Memory Map 16-17
16-10	Timing Mnemonics 16-24
16-11	8XC196MC/MD Auto Programming Memory Map 16-27
16-12	8XC196MH Auto Programming Memory Map 16-27
16-13	PCCB and UPROM Programming Values..... 16-32
A-1	Opcode Map (Left Half) A-2
A-1	Opcode Map (Right Half)..... A-3
A-2	Processor Status Word (PSW) Flags A-4
A-3	Effect of PSW Flags or Specified Conditions on Conditional Jump Instructions A-5
A-4	PSW Flag Setting Symbols A-5
A-5	Operand Variables A-6
A-6	Instruction Set A-7
A-7	Instruction Opcodes A-41
A-8	Instruction Lengths and Hexadecimal Opcodes A-47
A-9	Instruction Execution Times (in State Times) A-52
B-1	Signal Name Changes B-1
B-2	8XC196MC Signals Arranged by Functional Categories..... B-2
B-3	8XC196MD Signals Arranged by Functional Categories..... B-6
B-4	8XC196MH Signals Arranged by Functional Categories..... B-9
B-5	Description of Columns of Table B-6..... B-13
B-6	Signal Descriptions B-13
B-7	Definition of Status Symbols B-23
B-8	8XC196MC and MD Default Signal Conditions B-23
B-9	8XC196MH Default Signal Conditions B-25
C-1	Modules and Related Registers C-1
C-2	Register Name, Address, and Reset Status..... C-2
C-3	COMPX_TIME Addresses and Reset Values C-17
C-4	EPAX_CON Addresses and Reset Values C-20
C-5	EPAX_TIME Addresses and Reset Values..... C-21
C-6	PX_DIR Addresses and Reset Values C-30
C-7	PX_MODE Addresses and Reset Values C-31
C-8	Special-function Signals for Ports 1, 2, 5, 6..... C-32
C-9	PX_PIN Addresses and Reset Values C-33
C-10	PX_REG Addresses and Reset Values C-34
C-11	Output Configuration C-65
C-12	WSR Settings and Direct Addresses for Windowable SFRs C-68



1

Guide to This Manual

CHAPTER 1

GUIDE TO THIS MANUAL

This manual describes the 8XC196MC, 8XC196MD, and 8XC196MH embedded microcontrollers. 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.

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 addressable memory space of the device. It describes the memory partitions and explains how to use windows to increase the amount of memory that can be accessed with register-direct instructions.

Chapter 5 — 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 6 — I/O Ports — describes the input/output ports and explains how to configure the ports for input, output, or special functions.

Chapter 7 — Serial I/O (SIO) Port — describes the 8XC196MH's asynchronous/synchronous serial I/O (SIO) port and explains how to program it.

Chapter 8 — Frequency Generator — describes the 8XC196MD's frequency generator and explains how to configure it. For additional information and application examples, consult AP-483, *Application Examples Using the 8XC196MC/MD Microcontroller* (order number 272282).

Chapter 9 — Waveform Generator — describes the waveform generator and explains how to configure it. For additional information and application examples, consult AP-483, *Application Examples Using the 8XC196MC/MD Microcontroller* (order number 272282).

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 external memory signals and describes the registers that control the external memory interface. 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 — Programming the Nonvolatile Memory — provides recommended circuits, the corresponding memory maps, and flow diagrams. It also provides procedures for auto programming.

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 device 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.

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.

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 Px_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.
- 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., P1.0, P1.1); a range of pins is represented by Px.y:z (e.g., P1.4:0 represents five port pins: P1.4, P1.3, P1.2, P1.1, P1.0). A pound symbol (#) appended to a signal name identifies an active-low signal.

units of measure

The following abbreviations are used to represent units of measure:

A	amps, amperes
DCV	direct current volts
Kbytes	kilobytes
kHz	kilohertz
k Ω	kilo-ohms
mA	milliamps, milliamperes
Mbytes	megabytes
MHz	megahertz
ms	milliseconds
mW	milliwatts
ns	nanoseconds
pF	picofarads
W	watts
V	volts
μ A	microamps, microamperes
μ F	microfarads
μ s	microseconds
μ W	microwatts

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.

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 (telephone numbers are listed on page 1-11).

Table 1-1. Handbooks and Product Information

Title and Description	Order Number
<i>Intel Embedded Quick Reference Guide</i>	272439
<i>Solutions for Embedded Applications Guide</i>	240691
<i>Data on Demand</i> fact sheet	240952
<i>Data on Demand</i> annual subscription (6 issues; Windows* version) Complete set of Intel handbooks on CD-ROM.	240897
<i>Handbook Set</i> — handbooks and product overview Complete set of Intel's product line handbooks. Contains datasheets, application notes, article reprints and other design information on microprocessors, peripherals, embedded controllers, memory components, single-board computers, microcommunications, software development tools, and operating systems.	231003
<i>Automotive Products</i> † Application notes and article reprints on topics including the MCS 51 and MCS 96 microcontrollers. Documents in this handbook discuss hardware and software implementations and present helpful design techniques.	231792
<i>Embedded Applications</i> handbook (2 volume set) † Datasheets, architecture descriptions, and application notes on topics including flash memory devices, networking chips, and MCS 51 and MCS 96 microcontrollers. Documents in this handbook discuss hardware and software implementations and present helpful design techniques.	270648
<i>Embedded Microcontrollers</i> † Datasheets and architecture descriptions for Intel's three industry-standard microcontrollers, the MCS 48, MCS 51, and MCS 96 microcontrollers.	270646
<i>Peripheral Components</i> † Comprehensive information on Intel's peripheral components, including datasheets, application notes, and technical briefs.	296467
<i>Flash Memory</i> (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
<i>Packaging</i> † Detailed information on the manufacturing, applications, and attributes of a variety of semiconductor packages.	240800
<i>Development Tools Handbook</i> Information on third-party hardware and software tools that support Intel's embedded microcontrollers.	272326

† Included in handbook set (order number 231003)

Table 1-2. Application Notes, Application Briefs, and Article Reprints

Title	Order Number
AB-71, <i>Using the SIO on the 8XC196MH</i> (application brief)	272594
AP-125, <i>Designing Microcontroller Systems for Electrically Noisy Environments</i> †††	210313
AP-155, <i>Oscillators for Microcontrollers</i> †††	230659
AR-375, <i>Motor Controllers Take the Single-Chip Route</i> (article reprint)	270056

† Included in *Automotive Products* handbook (order number 231792)

†† Included in *Embedded Applications* handbook (order number 270648)

††† Included in *Automotive Products* and *Embedded Applications* handbooks

Table 1-2. Application Notes, Application Briefs, and Article Reprints (Continued)

Title	Order Number
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-475, Using the 8XC196NT ††	272315
AP-477, Low Voltage Embedded Design ††	272324
AP-483, Application Examples Using the 8XC196MC/MD Microcontroller	272282
AP-700, Intel Fuzzy Logic Tool Simplifies ABS Design †	272595
AP-711, EMI Design Techniques for Microcontrollers in Automotive Applications	272324
AP-715, Interfacing an I ² C Serial EEPROM to an MCS [®] 96 Microcontroller	272680

† Included in *Automotive Products* handbook (order number 231792)

†† Included in *Embedded Applications* handbook (order number 270648)

††† Included in *Automotive Products* and *Embedded Applications* handbooks

Table 1-3. MCS[®] 96 Microcontroller Datasheets (Commercial/Express)

Title	Order Number
8XC196KR/KQ/JR/JQ Commercial/Express CHMOS Microcontroller †	270912
8XC196KT Commercial CHMOS Microcontroller †	272266
87C196KT/87C196KS 20 MHz Advanced 16-Bit CHMOS Microcontroller †	272513
8XC196MC Industrial Motor Control Microcontroller †	272323
87C196MD Industrial Motor Control CHMOS Microcontroller †	270946
8XC196NP Commercial CHMOS 16-Bit Microcontroller †	272459
8XC196NT CHMOS Microcontroller with 1-Mbyte Linear Address Space †	272267
80C196NU Commercial CHMOS 16-Bit Microcontroller	272644

† Included in *Embedded Microcontrollers* handbook (order number 270646)

Table 1-4. MCS[®] 96 Microcontroller Datasheets (Automotive)

Title and Description	Order Number
87C196CA/87C196CB 20 MHz 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
87C196KR/KQ, 87C196JV/JT, 87C196JR/JQ Advanced 16-Bit CHMOS Microcontroller †	270827
87C196KT/87C196KS Advanced 16-Bit CHMOS Microcontroller †	270999
87C196KT/KS 20 MHz Advanced 16-Bit CHMOS Microcontroller †	272513

† Included in *Automotive Products* handbook (order number 231792)

This Page Left Intentionally Blank





This Page Left Intentionally Blank

This Page Left Intentionally Blank



1.4.4 World Wide Web

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

1.5 TECHNICAL SUPPORT

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

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

1.6 PRODUCT LITERATURE

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

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



2

Architectural Overview

CHAPTER 2 ARCHITECTURAL OVERVIEW

The 16-bit 8XC196MC, 8XC196MD, and 8XC196MH CMOS microcontrollers are designed to handle high-speed calculations and fast input/output (I/O) operations. They share a common architecture and instruction set with other members of the MCS[®] 96 microcontroller family.

NOTE

This manual describes a family of microcontrollers. For brevity, the name 8XC196Mx is used when the discussion applies to all family members. When information applies to specific microcontrollers, individual product names are used.

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).

2.2 MICROCONTROLLER FEATURES

Table 2-1 lists the features of each member of the 8XC196Mx family.

Table 2-1. Features of the 8XC196Mx Product Family

Device	Pins	OTPROM/ ROM Bytes (Note 1)	Register RAM Bytes (Note 2)	I/O Pins	EPA Pins	SIO Ports (Note 3)	PWM Channels (Note 4)	A/D Channels	External Interrupt Pins
8XC196MH	84	32 K	744	52	6	2	8	8	1
8XC196MH	80	32 K	744	52	6	2	8	8	1
8XC196MH	64	32 K	744	50	6	2	7	8	1
8XC196MC	84	16 K	488	53	8	0	8	13	1
8XC196MC	80	16 K	488	53	8	0	8	13	1
8XC196MC	64	16 K	488	49	7	0	7	12	1
8XC196MD	84	16 K	488	64	12	0	9	14	1
8XC196MD	80	16 K	488	64	12	0	9	14	1

NOTES:

1. Nonvolatile memory is optional. The second character of the device name indicates the presence and type of nonvolatile memory. 80C196Mx = none; 83C196Mx = ROM; 87C196Mx = OTPROM.
2. Register RAM amounts include the 24 bytes allocated to core SFRs and the stack pointer.
3. The 8XC196MC and 8XC196MD have no serial I/O ports, but have PTS modes that allow asynchronous or synchronous serial communication.
4. The number of PWM channels includes the outputs from the PWM peripheral and the waveform generator. For the 8XC196MD, it also includes the output from the frequency generator.

2.3 FUNCTIONAL OVERVIEW

Figure 2-1 shows the major blocks within the microcontroller. The core of the microcontroller (Figure 2-2) consists of the central processing unit (CPU) and memory controller. The CPU contains the register file and the register arithmetic-logic unit (RALU). 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.

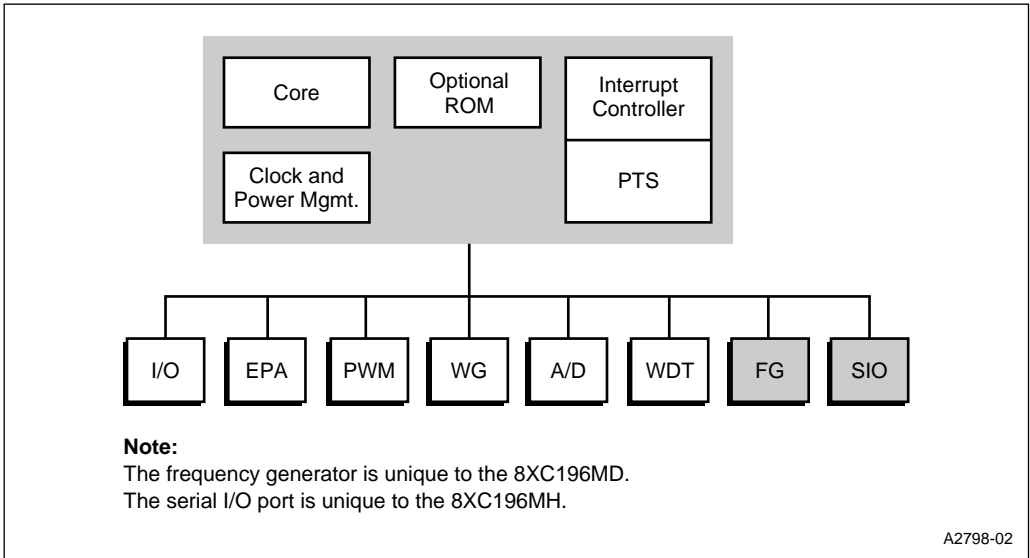


Figure 2-1. 8XC196Mx Block Diagram

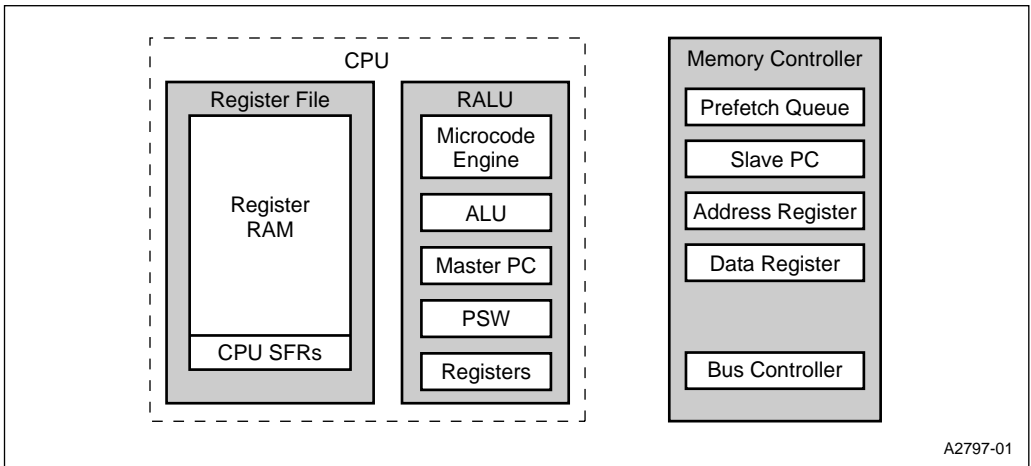


Figure 2-2. Block Diagram of the Core

2.3.1 CPU Control

The CPU is controlled by the microcode engine, which instructs the RALU to perform operations using bytes, words, or double-words from either the 256-byte lower register file or through a *window* that directly accesses the upper register file. (See Chapter 4, “Memory Partitions,” for more information about the register file and windowing.) CPU instructions move from the 4-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.

2.3.2 Register File

The register file is divided into an upper and a lower file. In the lower register file, the lowest 24 bytes are 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 “Addressing Modes” on page 3-5). The upper register file is accessible with direct addressing only when *windowing* is enabled. 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.

2.3.3 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 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 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. Appendix A, “Instruction Set Reference,” provides a detailed description of the PSW.

All registers, except the 3-bit bit-select register and the 6-bit loop counter, 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.

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 six-bit loop counter counts repetitive shifts. 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 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. In addition, the constants register generates single-bit masks, based on the bit-select register, for bit-test instructions.

2.3.3.1 Code Execution

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.

2.3.3.2 Instruction Format

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)
```

2.3.4 Memory Interface Unit

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, and the bus controller.

The bus controller drives the memory bus, which consists of an internal memory bus and the external address/data 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.

2.3.5 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, trap, and unimplemented opcode) to be serviced by the PTS instead of the interrupt controller.

The PTS can transfer bytes or words, either individually or in blocks, between any memory locations, manage multiple analog-to-digital (A/D) conversions, and can generate pulse-width modulated (PWM) signals. The 8XC196MC and 8XC196MD have additional modes that allow asynchronous or synchronous serial communication. PTS interrupts have a higher priority than standard interrupts and may temporarily suspend interrupt service routines. See Chapter 5, “Standard and PTS Interrupts,” for more information.

2.4 INTERNAL TIMING

The clock circuitry (Figure 2-3) receives an input clock signal on XTAL1 provided by an external crystal or oscillator and divides the frequency by two. The clock generators accept the divided input frequency from the divide-by-two circuit and produce two nonoverlapping internal timing signals, PH1 and PH2. These signals are active when high.

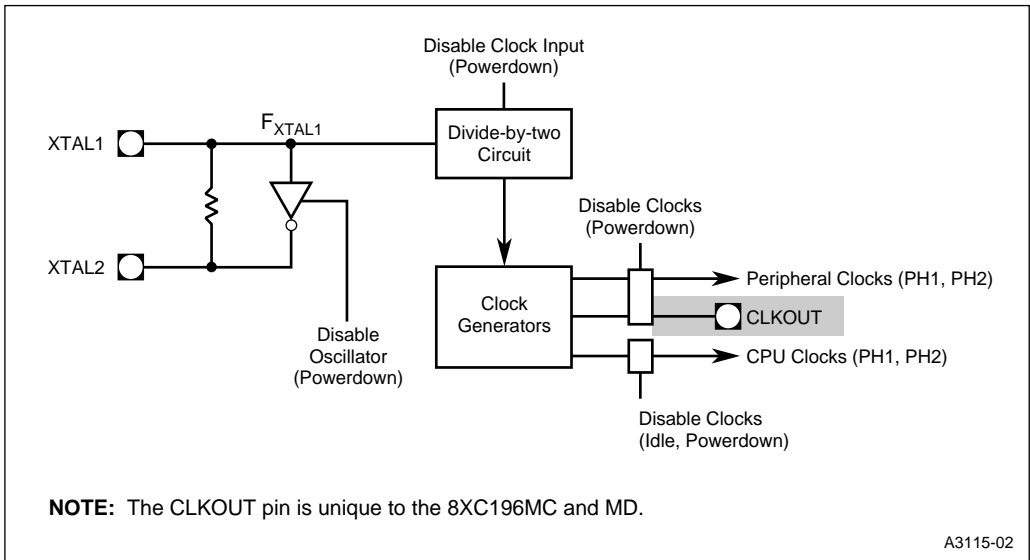


Figure 2-3. Clock Circuitry

The rising edges of PH1 and PH2 generate the internal CLKOUT signal (Figure 2-4). The clock circuitry routes separate internal clock signals to the CPU and the peripherals to provide flexibility in power management. (“Reducing Power Consumption” on page 14-3 describes the power management modes.) The 8XC196MC and 8XC196MD microcontrollers output the CLKOUT signal on the CLKOUT pin. 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.

The 8XC196MH microcontroller has no CLKOUT pin. If your 8XC196MH design requires a system clock, we recommend that you use an external oscillator and add external logic to generate the system clock signal.

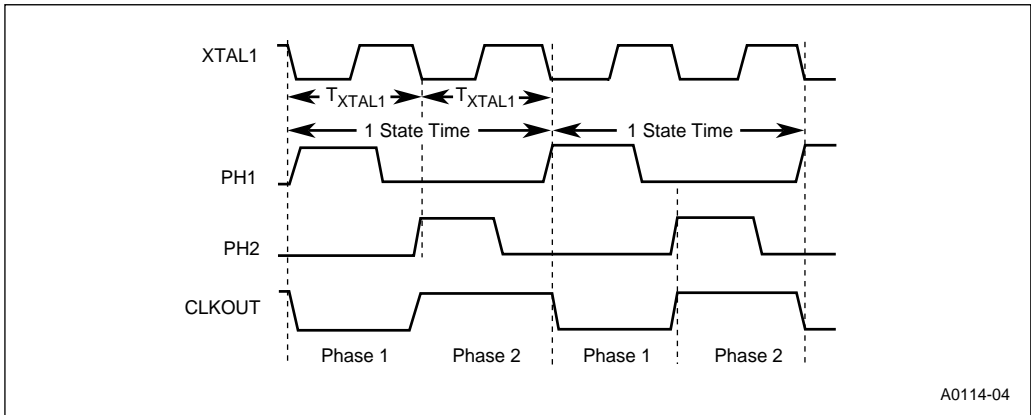


Figure 2-4. Internal Clock Phases

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 2-2 lists state time durations at various frequencies.

Table 2-2. State Times at Various Frequencies

F_{XTAL1} (Frequency Input to the Divide-by-two Circuit)	State Time
8 MHz	250 ns
12 MHz	167 ns
16 MHz	125 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_{XTAL1}).

$$PH1 \text{ (in MHz)} = \frac{F_{XTAL1}}{2} = PH2 \quad \text{State Time (in } \mu\text{s)} = \frac{2}{F_{XTAL1}} \quad T_{XTAL1} = \frac{1}{F_{XTAL1}}$$

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_{XTAL1} or T_{OSC}).

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 8XC196Mx microcontrollers have seven I/O ports, ports 0–6. The 8XC196MD has an additional port, port 7. Individual port pins are multiplexed to serve as standard I/O or to carry special-function signals associated with an on-chip peripheral or an off-chip component. If a particular special-function signal is not used in an application, the associated pin can be individually configured to serve as a standard I/O pin. 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 address/data bus, it takes control of the ports. When the address/data bus is idle, you can use the ports for I/O.

Port 0 is an input-only port that is also the analog input for the A/D converter. On the 8XC196MH, port 0 provides two pins for the EPA. On the 8XC196MC and 8XC196MD, port 1 is also an input-only port that provides analog inputs for the A/D converter. On the 8XC196MH, port 1 is a bidirectional port that shares pins with the serial I/O port.

Port 2 is a standard, bidirectional I/O port that provides pins for the EPA and timers. Port 7, which is unique to the 8XC196MD, is a standard, bidirectional I/O port that provides additional pins for the EPA and also provides pins for the frequency generator. Ports 3, 4, and 5 are memory-mapped, bidirectional I/O ports. Ports 3 and 4 serve as the external address/data bus, while port 5 provides bus-control signals. Port 6 is a standard, output-only port that provides pins for the pulse-width modulator and waveform generator. Chapter 6, “I/O Ports,” describes the I/O ports in more detail.

2.5.2 Serial I/O (SIO) Port

The 8XC196MH microcontroller has a two-channel serial I/O port that shares pins with ports 1 and 2. (The 8XC196MC and 8XC196MD have no serial I/O ports, but have PTS modes that allow asynchronous or synchronous serial communication. See Chapter 5, “Standard and PTS Interrupts,” for more information.) The serial I/O (SIO) port is an asynchronous/synchronous port that includes a universal asynchronous receiver and transmitter (UART). The UART has two synchronous modes (modes 0 and 4) and three asynchronous modes (modes 1, 2, and 3) for both transmission and reception. The asynchronous modes are full duplex, meaning that they can transmit and receive data simultaneously. 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 7, “Serial I/O (SIO) Port,” for details.

2.5.3 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.

Timer 1 and timer 2 are both 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. See Chapter 11, “Event Processor Array (EPA),” for additional information on the EPA and timer/counters.

2.5.4 Pulse-width Modulator (PWM)

The output waveform from each PWM channel is a variable duty-cycle pulse. 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.

2.5.5 Frequency Generator

The 8XC196MD has a peripheral not found on other 8XC196Mx microcontrollers — the frequency generator. This peripheral produces a waveform with a fixed duty cycle (50%) and a programmable frequency (ranging from 4 kHz to 1 MHz with a 16 MHz input clock). See Chapter 8, “Frequency Generator,” for details.

2.5.6 Waveform Generator

A waveform generator simplifies the task of generating synchronized, pulse-width modulated (PWM) outputs. This waveform generator is optimized for motion control applications such as driving 3-phase AC induction motors, 3-phase DC brushless motors, or 4-phase stepping motors. The waveform generator can produce three independent pairs of complementary PWM outputs, which share a common carrier period, dead time, and operating mode. Once it is initialized, the waveform generator operates without CPU intervention unless you need to change a duty cycle. See Chapter 9, “Waveform Generator,” for more information.

2.5.7 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, which can be used to generate an interrupt when a programmable threshold voltage is crossed in either direction. The A/D scan mode of the PTS facilitates automated A/D conversions and result storage. See Chapter 12, “Analog-to-digital (A/D) Converter,” for more information.

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 mode, the microcontroller operates in several special-purpose modes. Idle and powerdown modes conserve power when the microcontroller is inactive. On-circuit emulation (ONCE) mode electrically isolates the microcontroller from the system, and several other modes provide programming options for nonvolatile memory. See Chapter 14, “Special Operating Modes,” for more information about idle, powerdown, and ONCE modes, and see Chapter 16, “Programming the Nonvolatile Memory,” for details about programming options.

2.6.1 Reducing Power Consumption

In idle mode, the CPU stops executing instructions, but the peripheral clocks remain active. Power consumption drops to about 40% 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 and most peripherals retain their data if V_{CC} is maintained. Power consumption drops into the μW range.

2.6.2 Testing the Printed Circuit Board

The 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.

2.6.3 Programming the Nonvolatile Memory

MCS 96 microcontrollers that have internal OTPROM provide several programming options:

- Slave programming allows a master EPROM programmer to program and verify one or more slave MCS 96 microcontrollers. Programming vendors and Intel distributors typically use this mode to program a large number of microcontrollers with a customer's code and data.
- Auto programming allows an MCS 96 microcontroller to program itself with code and data located in an external memory device. Customers typically use this low-cost method to program a small number of microcontrollers after development and testing are complete.
- Run-time programming allows you to program individual nonvolatile memory locations during normal code execution, under complete software control. Customers typically use this mode to download a small amount of information to the microcontroller after the rest of the array has been programmed. For example, you might use run-time programming to download a unique identification number to a security device.
- ROM dump mode allows you to dump the contents of the microcontroller's nonvolatile memory to a tester or to a memory device (such as flash memory or RAM).

Chapter 16, "Programming the Nonvolatile Memory," provides recommended circuits, the corresponding memory maps, and flow diagrams. It also provides procedures for auto programming.



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 operand types likely to be useful in control applications (see Table 3-1).

NOTE

The operand-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. Operand Type Definitions

Operand 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 (Note 2)
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 (Note 2)

NOTES:

1. The 32-bit variables are supported 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.
2. For consistency with third-party software, you should adopt the C programming conventions for addressing 32-bit operands. For more information, refer to page 3-9.

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

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

Operand Types	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

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. It does not support the direct addressing of BITS.

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. There are no alignment restrictions for 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 into 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 Converting Operands

The instruction set supports conversions between the operand types. 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.

3.1.9 Conditional Jumps

The instructions for addition, subtraction, and comparison do not distinguish between unsigned (BYTE, WORD) and signed (SHORT-INTEGER, INTEGER) operands. 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.

3.1.10 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.

3.2 ADDRESSING MODES

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.

An instruction can contain only one immediate, indirect, or indexed reference; any remaining operands must be direct references.

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. The assembly language hides some of the details of how these addressing modes work. “Assembly Language Addressing Mode Selections” on page 3-9 describes how the assembly language handles direct and indexed addressing modes.

The examples in this section assume that temporary registers are defined as shown in this segment of assembly code and described in Table 3-3.

```
                Oseg at lch
AL:             DSB 1
BL:             DSB 1
CL:             DSB 1
DL:             DSB 1
AX:             DSW 1
BX:             DSW 1
CX:             DSW 1
DX:             DSW 1
THISVAR:       DSW 1
```


Table 3-3. 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; BH is the high byte of BX and BL is the low byte
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

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 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

```

3.2.3.2 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 255 byte locations, long-indexed addressing can access up to 65,535 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 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
```

3.3 ASSEMBLY LANGUAGE ADDRESSING MODE SELECTIONS

The assembly language simplifies the choice of addressing modes. Use these features wherever possible.

3.3.1 Direct Addressing

The assembly language chooses between direct and zero-indexed addressing depending on the memory location of the operand. Simply refer to the operand by its symbolic name. If the operand is in the lower register file, the assembly language chooses a direct reference. If the operand is elsewhere in memory, it chooses a zero-indexed reference.

3.3.2 Indexed Addressing

The assembly language chooses between short-indexed and long-indexed addressing depending on the value of the index expression. If the value can be expressed in eight bits, the assembly language chooses a short-indexed reference. If the value is greater than eight bits, it chooses a long-indexed reference.

3.4 SOFTWARE STANDARDS AND CONVENTIONS

For a software project of any size, it is a good idea to develop the program in modules and to establish standards that control communication between the modules. These standards vary with the needs of the final application. However, all standards must include some mechanism for passing parameters to procedures and returning results from procedures. We recommend that you use the conventions adopted by the C programming language for procedure linkage. These standards are usable for both the assembly language and C programming environments, and they offer compatibility between these environments.

3.4.1 Using Registers

The 256-byte lower register file contains the CPU special-function registers and the stack pointer. The remainder of the lower register file and all of the upper register file is available for your use. Peripheral special-function registers (SFRs) and memory-mapped SFRs reside in higher memory. The peripheral SFRs can be *windowed* into the lower register file for direct access. Memory-mapped SFRs cannot be windowed; you must use indirect or indexed addressing to access them. All SFRs can be operated on as BYTES or WORDs, unless otherwise specified. See “Special-function Registers (SFRs)” on page 4-4 and “Register File” on page 4-9 for more information.

To use these registers effectively, you must have some overall strategy for allocating them. The C programming language adopts a simple, effective strategy. It allocates the eight bytes beginning at address 1CH as temporary storage and treats the remaining area in the register file as a segment of memory that is allocated as required.

NOTE

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

3.4.2 Addressing 32-bit Operands

The 32-bit operands (DOUBLE-WORDS and LONG-INTEGERS) are formed by two adjacent 16-bit words in memory. The least-significant word of a DOUBLE-WORD is always in the lower address, even when the data is in the stack (which means that the most-significant word must be pushed into the stack first). The address of a 32-bit operand is that of its least-significant byte.

The hardware supports the 32-bit data types as operands in shift operations, as dividends of 32-by-16 divide operations, and as products of 16-by-16 multiply operations. For these operations, the 32-bit operand must reside in the lower register file and must be aligned at an address that is evenly divisible by four.

3.4.3 Linking Subroutines

Parameters are passed to subroutines via the stack. Parameters are pushed into the stack from the rightmost parameter to the left. The 8-bit parameters are pushed into the stack with the high-order byte undefined. The 32-bit parameters are pushed into the stack as two 16-bit values; the most-significant half of the parameter is pushed into the stack first. As an example, consider the following procedure:

```
void example_procedure (char param1, long param2, int param3);
```

When this procedure executes at run-time, the stack will contain the parameters in the following order:

```
param3
low word of param2
high word of param2
undefined;param1
return address      ← Stack Pointer
```

If a procedure returns a value to the calling code (as opposed to modifying more global variables), the result is returned in the temporary storage space (TMPREG0, in this example) starting at 1CH. TMPREG0 is viewed as either an 8-, 16-, or 32bit variable, depending on the type of the procedure.

The standard calling convention adopted by the C programming language has several key features:

- Procedures can always assume that the eight bytes of register file memory starting at 1CH can be used as temporary storage within the body of the procedure.
- Code that calls a procedure must assume that the procedure modifies the eight bytes of register file memory starting at 1CH.
- Code that calls a procedure must assume that the procedure modifies the processor status word (PSW) condition flags because procedures do not save and restore the PSW.
- Function results from procedures are always returned in the variable TMPREG0.

The C programming language allows the definition of interrupt procedures, which are executed when a predefined interrupt request occurs. Interrupt procedures do not conform to the rules of normal procedures. Parameters cannot be passed to these procedures and they cannot return results. Since interrupt procedures can execute essentially at any time, they must save and restore both the PSW and TMPREG0.

3.5 SOFTWARE PROTECTION FEATURES AND GUIDELINES

The device 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 device in the event of a hardware or software error.

We recommend that you 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 will provide protection only for catastrophic failures.



4

Memory Partitions

CHAPTER 4

MEMORY PARTITIONS

This chapter describes the address space, its major partitions, and a *windowing* technique for accessing the upper register file and peripheral SFRs with register-direct instructions.

4.1 MEMORY PARTITIONS

Table 4-1 is a memory map of the 8XC196Mx devices. The remainder of this section describes the partitions.

4.1.1 External Devices (Memory or I/O)

Several partitions are assigned to external devices (see Table 4-1). Data can be stored in any part of this memory. Chapter 15, “Interfacing with External Memory,” describes the external memory interface and shows examples of external memory configurations. These partitions can also be used to interface with external peripherals connected to the address/data bus.

4.1.2 Program and Special-purpose Memory

Internal nonvolatile memory is an optional component of the 8XC196Mx devices. Various devices are available with masked ROM, EPROM, QROM, or OTPROM. Please consult the datasheets in the *Embedded Microcontrollers* databook for details.

If present, the nonvolatile memory occupies the special-purpose memory and program memory partitions (locations 2000H and above; see Table 4-1 on page 4-2). The EA# signal controls access to these memory partitions. Accesses to these partitions are directed to internal memory if EA# is held high and to external memory if EA# is held low. For devices without internal nonvolatile memory, the EA# signal must be tied low. EA# is latched at reset.

Table 4-1. Memory Map

Device and Hex Address Range		Description	Addressing Modes
MC, MD	MH		
FFFF 6000	FFFF A000	External device (memory or I/O) connected to the address/data bus	Indirect or indexed
5FFF 2080	9FFF 2080	Program memory (internal nonvolatile or external memory); see Note 1.	Indirect or indexed
207F 2000	207F 2000	Special-purpose memory (internal nonvolatile or external memory)	Indirect or indexed
1FFF 1FE0	1FFF 1FE0	Memory-mapped SFRs	Indirect or indexed
1FDF 1F00	1FDF 1F00	Peripheral SFRs	Indirect, indexed, or windowed direct
1EFF 0200	1EFF 0300	External device (memory or I/O) connected to the address/data bus (future SFR expansion; see Note 2)	Indirect or indexed
01FF 0100	02FF 0100	Upper register file (general-purpose register RAM)	Indirect, indexed, or windowed direct
00FF 0000	00FF 0000	Lower register file (general-purpose register RAM, stack pointer, and CPU SFRs)	Direct, indirect, or indexed

NOTES:

1. After a reset, the device fetches its first instruction from 2080H.
2. The content or function of these locations may change in future device revisions, in which case a program that relies on a location in this range might not function properly.

4.1.3 Program Memory

Program memory occupies a memory partition beginning at 2080H. (See Table 4-1 for the ending address for each device.) This entire partition is available for storing executable code and data. The EA# signal controls access to program memory. Accesses to this address range are directed to internal memory if EA# is held high and to external memory if EA# is held low. For devices without internal nonvolatile memory, the EA# signal must be tied low. EA# is latched at reset.

NOTE

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

4.1.4 Special-purpose Memory

Special-purpose memory resides in locations 2000–207FH (Table 4-2). It contains several reserved memory locations, the chip configuration bytes (CCBs), and vectors for both peripheral transaction server (PTS) and standard interrupts. Accesses to this address range are directed to internal memory if EA# is held high and to external memory if EA# is held low. For devices without internal nonvolatile memory, the EA# signal must be tied low. EA# is latched at reset.

Table 4-2. Special-purpose Memory Addresses

Hex Address	Description
207F 205E	Reserved (each byte must contain FFH)
205D 2040	PTS vectors
203F 2030	Upper interrupt vectors
202F 2020	Security key
201F	Reserved (must contain 20H)
201E	Reserved (must contain FFH)
201D	Reserved (must contain 20H)
201C	Reserved (must contain FFH)
201B	Reserved (must contain 20H)
201A	CCB1
2019	Reserved (must contain 20H)
2018	CCB0
2017 2014	Reserved (each byte must contain FFH)
2013 2000	Lower interrupt vectors

4.1.4.1 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. The function or contents of these locations may change in future revisions; software that uses reserved locations may not function properly. Always initialize reserved locations to the values listed in Table 4-2.

4.1.4.2 Interrupt and PTS Vectors

The upper and lower interrupt vectors contain the addresses of the interrupt service routines. The peripheral transaction server (PTS) vectors contain the addresses of the PTS control blocks. See Chapter 5, “Standard and PTS Interrupts,” for more information on interrupt and PTS vectors.

4.1.4.3 Security Key

The security key prevents unauthorized programming access to the nonvolatile memory. See Chapter 16, “Programming the Nonvolatile Memory,” for details.

4.1.4.4 Chip Configuration Bytes (CCBs)

The chip configuration bytes (CCBs) specify the operating environment. They specify the bus width, bus-control mode, and wait states. They also control powerdown mode, the watchdog timer, and nonvolatile memory protection.

The CCBs are the first bytes fetched from memory when the device 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 device reset. Typically, the CCBs are programmed once when the user program is compiled and are not redefined during normal operation. “Chip Configuration Registers and Chip Configuration Bytes” on page 15-5 describes the CCBs and CCRs.

For devices with customer-programmable nonvolatile memory, the CCBs are loaded for normal operation, but the PCCBs are loaded into the CCRs if the device is entering programming modes. See Chapter 16, “Programming the Nonvolatile Memory,” for details.

4.1.5 Special-function Registers (SFRs)

These devices have both memory-mapped SFRs and peripheral SFRs. The memory-mapped SFRs must be accessed using indirect or indexed addressing modes, and they **cannot** be windowed. The peripheral SFRs are physically located in the on-chip peripherals, and they can be windowed (see “Windowing” on page 4-12). 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 because external events can change the contents of SFRs. Also, because some SFRs are cleared when read, consider the implications of using an SFR as an operand in a read-modify-write instruction (e.g., XORB).

4.1.5.1 Memory-mapped SFRs

Locations 1FE0–1FFFH contain memory-mapped SFRs (see Table 4-3). Locations in this range that are omitted from the table are reserved. The memory-mapped SFRs must be accessed 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.

The memory-mapped SFRs are accessed through the memory controller, so instructions that operate on these SFRs execute as they would from external memory with zero wait states.

Table 4-3. Memory-mapped SFRs

Ports 3, 4, 5, UPROM SFRs		
Hex Address	High (Odd) Byte	Low (Even) Byte
1FFE	P4_PIN	P3_PIN
1FFC	P4_REG	P3_REG
...
1FF6	P5_PIN	USFR
1FF4	P5_REG	Reserved
1FF2	P5_DIR	Reserved
1FF0	P5_MODE	Reserved

4.1.5.2 Peripheral SFRs

Locations 1F00–1FDFH provide access to the peripheral SFRs. Table 4-6 on page 4-8, Table 4-6 on page 4-8, and Table 4-6 on page 4-8 list the peripheral SFRs of the 8XC196MC, 8XC196MD, and 8XC196MH, respectively. Locations that are omitted from the tables are reserved. The peripheral SFRs are I/O control registers; they are physically located in the on-chip peripherals. These peripheral SFRs can be windowed and they can be addressed either as words or bytes, except as noted in the tables.

The peripheral SFRs are accessed directly, without using the memory controller, so instructions that operate on these SFRs execute as they would if they were operating on the register file.

Table 4-4. Peripheral SFRs — 8XC196MC

Port 2 SFRs			EPA and Timer SFRs		
Address	High (Odd) Byte	Low (Even) Byte	Address	High (Odd) Byte	Low (Even) Byte
1FDEH	Reserved	Reserved	†1F7EH	TIMER2 (H)	TIMER2 (L)
...	1F7CH	Reserved	T2CONTROL
1FD6H	Reserved	P2_PIN	†1F7AH	TIMER1 (H)	TIMER1 (L)
1FD4H	Reserved	P2_REG	1F78H	Reserved	T1CONTROL
1FD2H	Reserved	P2_DIR	1F76H	Reserved	Reserved
1FD0H	Reserved	P2_MODE	1F74H	Reserved	Reserved
Waveform Generator SFRs			1F72H	T1RELOAD (H)	T1RELOAD (L)
Address	High (Odd) Byte	Low (Even) Byte	1F70H	Reserved	Reserved
1FCEH	Reserved	WG_PROTECT
1FCCH	WG_CONTROL(H)	WG_CONTROL(L)	†1F66H	COMP3_TIME (H)	COMP3_TIME (L)
1FCAH	WG_COUNTER(H)	WG_COUNTER(L)	1F64H	Reserved	COMP3_CON
1FC8H	WG_RELOAD (H)	WG_RELOAD (L)	†1F62H	COMP2_TIME (H)	COMP2_TIME (L)
1FC6H	WG_COMP3 (H)	WG_COMP3 (L)	1F60H	Reserved	COMP2_CON
1FC4H	WG_COMP2 (H)	WG_COMP2 (L)	†1F5EH	COMP1_TIME (H)	COMP1_TIME (L)
1FC2H	WG_COMP1 (H)	WG_COMP1 (L)	1F5CH	Reserved	COMP1_CON
1FC0H	WG_OUTPUT (H)	WG_OUTPUT (L)	†1F5AH	COMP0_TIME (H)	COMP0_TIME (L)
Peripheral Interrupt and PWM SFRs			1F58H	Reserved	COMP0_CON
Address	High (Odd) Byte	Low (Even) Byte	1F56H	Reserved	Reserved
1FBEH	Reserved	PI_PEND
1FBCH	Reserved	PI_MASK	†1F4EH	EPA3_TIME (H)	EPA3_TIME (L)
...	1F4CH	Reserved	EPA3_CON
1FB6H	Reserved	PWM_COUNT	†1F4AH	EPA2_TIME (H)	EPA2_TIME (L)
1FB4H	Reserved	PWM_PERIOD	1F48H	Reserved	EPA2_CON
1FB2H	Reserved	PWM1_CONTROL	†1F46H	EPA1_TIME (H)	EPA1_TIME (L)
1FB0H	Reserved	PWM0_CONTROL	1F44H	Reserved	EPA1_CON
A/D SFRs			†1F42H	EPA0_TIME (H)	EPA0_TIME (L)
Address	High (Odd) Byte	Low (Even) Byte	1F40H	Reserved	EPA0_CON
1FAEH	AD_TIME	AD_TEST			
1FACH	Reserved	AD_COMMAND			
1FAAH	AD_RESULT (H)	AD_RESULT (L)			
1FA8H	P1_PIN	P0_PIN			
1FA6H	Reserved	Reserved			
...			
1F80H	Reserved	Reserved			

† Must be addressed as a word.

Table 4-5. Peripheral SFRs — 8XC196MD

Ports 2 and 7 SFRs			EPA and Timer SFRs		
Address	High (Odd) Byte	Low (Even) Byte	Address	High (Odd) Byte	Low (Even) Byte
1FDEH	Reserved	Reserved	†1F7EH	TIMER2 (H)	TIMER2 (L)
1FDCH	Reserved	Reserved	1F7CH	Reserved	T2CONTROL
1FDAH	Reserved	Reserved	†1F7AH	TIMER1 (H)	TIMER1 (L)
1FD8H	Reserved	Reserved	1F78H	Reserved	T1CONTROL
1FD6H	P7_PIN	P2_PIN	1F76H	Reserved	Reserved
1FD4H	P7_REG	P2_REG	1F74H	Reserved	Reserved
1FD2H	P7_DIR	P2_DIR	1F72H	T1RELOAD (H)	T1RELOAD (L)
1FD0H	P7_MODE	P2_MODE	1F70H	Reserved	Reserved
Waveform Generator SFRs			†1F6EH	COMP5_TIME (H)	COMP5_TIME (L)
Address	High (Odd) Byte	Low (Even) Byte	1F6CH	Reserved	COMP5_CON
1FCEH	Reserved	WG_PROTECT	†1F6AH	COMP4_TIME (H)	COMP4_TIME (L)
1FCCH	WG_CONTROL(H)	WG_CONTROL(L)	1F68H	Reserved	COMP4_CON
1FCAH	WG_COUNTER(H)	WG_COUNTER(L)	†1F66H	COMP3_TIME (H)	COMP3_TIME (L)
1FC8H	WG_RELOAD (H)	WG_RELOAD (L)	1F64H	Reserved	COMP3_CON
1FC6H	WG_COMP3 (H)	WG_COMP3 (L)	†1F62H	COMP2_TIME (H)	COMP2_TIME (L)
1FC4H	WG_COMP2 (H)	WG_COMP2 (L)	1F60H	Reserved	COMP2_CON
1FC2H	WG_COMP1 (H)	WG_COMP1 (L)	†1F5EH	COMP1_TIME (H)	COMP1_TIME (L)
1FC0H	WG_OUTPUT (H)	WG_OUTPUT (L)	1F5CH	Reserved	COMP1_CON
Periph. Int., Freq. Gen., and PWM SFRs			†1F5AH	COMP0_TIME (H)	COMP0_TIME (L)
Address	High (Odd) Byte	Low (Even) Byte	1F58H	Reserved	COMP0_CON
1FBEH	Reserved	PI_PEND	†1F56H	EPA5_TIME (H)	EPA5_TIME (L)
1FBCH	Reserved	PI_MASK	1F54H	Reserved	EPA5_CON
1FBAH	Reserved	FREQ_CNT	†1F52H	EPA4_TIME (H)	EPA4_TIME (L)
1FB8H	Reserved	FREQ_GEN	1F50H	Reserved	EPA4_CON
1FB6H	Reserved	PWM_COUNT	†1F4EH	EPA3_TIME (H)	EPA3_TIME (L)
1FB4H	Reserved	PWM_PERIOD	1F4CH	Reserved	EPA3_CON
1FB2H	Reserved	PWM1_CONTROL	†1F4AH	EPA2_TIME (H)	EPA2_TIME (L)
1FB0H	Reserved	PWM0_CONTROL	1F48H	Reserved	EPA2_CON
A/D SFRs			†1F46H	EPA1_TIME (H)	EPA1_TIME (L)
Address	High (Odd) Byte	Low (Even) Byte	1F44H	Reserved	EPA1_CON
1FAEH	AD_TIME	AD_TEST	†1F42H	EPA0_TIME (H)	EPA0_TIME (L)
1FACH	Reserved	AD_COMMAND	1F40H	Reserved	EPA0_CON
1FAAH	AD_RESULT (H)	AD_RESULT (L)			
1FA8H	P1_PIN	P0_PIN			
...			
1F80H	Reserved	Reserved			

† Must be addressed as a word.

Table 4-6. Peripheral SFRs — 8XC196MH

Port 0 and 2 SFRs		
Address	High (Odd) Byte	Low (Even) Byte
1FDEH	Reserved	Reserved
1FDCH	Reserved	Reserved
1FDAH	Reserved	P0_PIN
1FD8H	Reserved	Reserved
1FD6H	Reserved	P2_PIN
1FD4H	Reserved	P2_REG
1FD2H	Reserved	P2_DIR
1FD0H	Reserved	P2_MODE
Waveform Generator SFRs		
Address	High (Odd) Byte	Low (Even) Byte
1FCEH	Reserved	WG_PROTECT
1FCCH	WG_CONTROL(H)	WG_CONTROL (L)
1FCAH	WG_COUNTER(H)	WG_COUNTER (L)
1FC8H	WG_RELOAD (H)	WG_RELOAD (L)
1FC6H	WG_COMP3 (H)	WG_COMP3 (L)
1FC4H	WG_COMP2 (H)	WG_COMP2 (L)
1FC2H	WG_COMP1 (H)	WG_COMP1 (L)
1FC0H	WG_OUTPUT (H)	WG_OUTPUT (L)
Peripheral Interrupt and PWM SFRs		
Address	High (Odd) Byte	Low (Even) Byte
1FBEH	Reserved	PI_PEND
1FBCH	Reserved	PI_MASK
1FBAH	Reserved	Reserved
1FB8H	Reserved	Reserved
1FB6H	Reserved	PWM_COUNT
1FB4H	Reserved	PWM_PERIOD
1FB2H	Reserved	PWM1_CONTROL
1FB0H	Reserved	PWM0_CONTROL
A/D SFRs		
Address	High (Odd) Byte	Low (Even) Byte
1FAEH	AD_TIME	AD_TEST
1FACH	Reserved	AD_COMMAND
1FAAH	AD_RESULT (H)	AD_RESULT (L)
Reset Control SFR		
Address	High (Odd) Byte	Low (Even) Byte
1FA8H	Reserved	Reserved
...
1FA0H	Reserved	GEN_CON

Port 1 SFRs		
Address	High (Odd) Byte	Low (Even) Byte
1F9EH	P1_PIN	Reserved
1F9CH	P1_REG	Reserved
1F9AH	P1_DIR	Reserved
1F98H	P1_MODE	Reserved
...
Serial I/O Port SFRs		
Address	High (Odd) Byte	Low (Even) Byte
1F8EH	Reserved	Reserved
1F8CH	SP1_BAUD(H)	SP1_BAUD (L)
1F8AH	SP1_CON	SBUF1_TX
1F88H	SP1_STATUS	SBUF1_RX
1F86H	Reserved	Reserved
1F84H	SP0_BAUD (H)	SP0_BAUD (L)
1F82H	SP0_CON	SBUF0_TX
1F80H	SP0_STATUS	SBUF0_RX
EPA and Timer SFRs		
Address	High (Odd) Byte	Low (Even) Byte
†1F7EH	TIMER2 (H)	TIMER2 (L)
1F7CH	Reserved	T2CONTROL
†1F7AH	TIMER1 (H)	TIMER1 (L)
1F78H	Reserved	T1CONTROL
...
1F72H	T1RELOAD (H)	T1RELOAD (L)
...
†1F62H	COMP2_TIME(H)	COMP2_TIME(L)
1F60H	Reserved	COMP2_CON
†1F5EH	COMP1_TIME(H)	COMP1_TIME(L)
1F5CH	Reserved	COMP1_CON
†1F5AH	COMP0_TIME(H)	COMP0_TIME(L)
1F58H	Reserved	COMP0_CON
...
†1F4EH	COMP3_TIME(H)	COMP3_TIME(L)
1F4CH	Reserved	COMP3_CON
...
†1F46H	EPA1_TIME (H)	EPA1_TIME (L)
1F44H	Reserved	EPA1_CON
†1F42H	EPA0_TIME (H)	EPA0_TIME (L)
1F40H	Reserved	EPA0_CON

† Must be addressed as a word.

4.1.6 Register File

The register file (Figure 4-1) is divided into an upper register file and a lower register file. The upper register file consists of general-purpose register RAM. The lower register file contains general-purpose register RAM along with the stack pointer (SP) and the CPU special-function registers (SFRs).

Table 4-1 on page 4-2 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 4-12). 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 register-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.

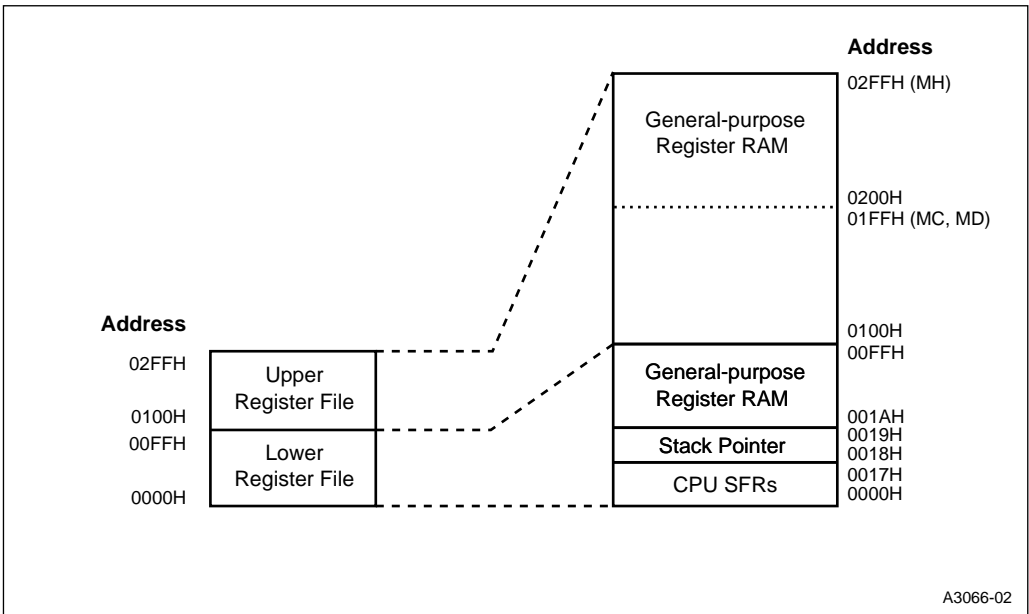


Figure 4-1. Register File Memory Map

Table 4-7. Register File Memory Addresses

Device and Hex Address Range		Description	Addressing Modes
MC, MD	MH		
01FF 0100	02FF 0100	Upper register file (register RAM)	Indirect, indexed, or windowed direct
00FF 001A	00FF 001A	Lower register file (register RAM)	Direct, indirect, or indexed
0019 0018	0019 0018	Lower register file (stack pointer)	Direct, indirect, or indexed
0017 0000	0017 0000	Lower register file (CPU SFRs)	Direct, indirect, or indexed

4.1.6.1 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 register-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 register-direct addressing to access this memory. (See Chapter 3, “Programming Considerations,” for a discussion of addressing modes.) Windowing can provide for fast context switching of interrupt tasks and faster program execution. (See “Windowing” on page 4-12.) PTS control blocks and the stack are most efficient when located in the upper register file.

4.1.6.2 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 greater than the desired starting address. Before the CPU executes a subroutine call or interrupt service routine, it decrements the SP by two and 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 and increments the SP by two.

Subroutines may be nested. That is, each subroutine may call other subroutines. 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 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 either the internal register file or external RAM. The stack can be used most efficiently when it is located in the register file.

The following example initializes the top of the upper register file (8XC196MC, MD) as the stack. (For the 8XC196MH, the immediate value would be #300H.)

```
LD    SP, #200H                ;Load stack pointer
```

The following example shows how to allow the linker locator to determine where the stack fits in the memory map that you specify.

```
LD    SP, #STACK
```

4.1.6.3 CPU Special-function Registers (SFRs)

Locations 0000–0017H in the lower register file are the CPU SFRs (Table 4-8). Appendix C describes the CPU SFRs.

Table 4-8. CPU SFRs

Address	High (Odd) Byte	Low (Even) Byte
0016H	Reserved	Reserved
0014H	Reserved	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)

NOTE

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

4.2 WINDOWING

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

Windowing maps a segment of higher memory (the upper register file or peripheral SFRs) into the lower register file. The window selection register (WSR) selects a 32-, 64-, or 128-byte segment of higher memory to be windowed into the top of the lower register file space. Figure 4-2 illustrates a 128-byte window.

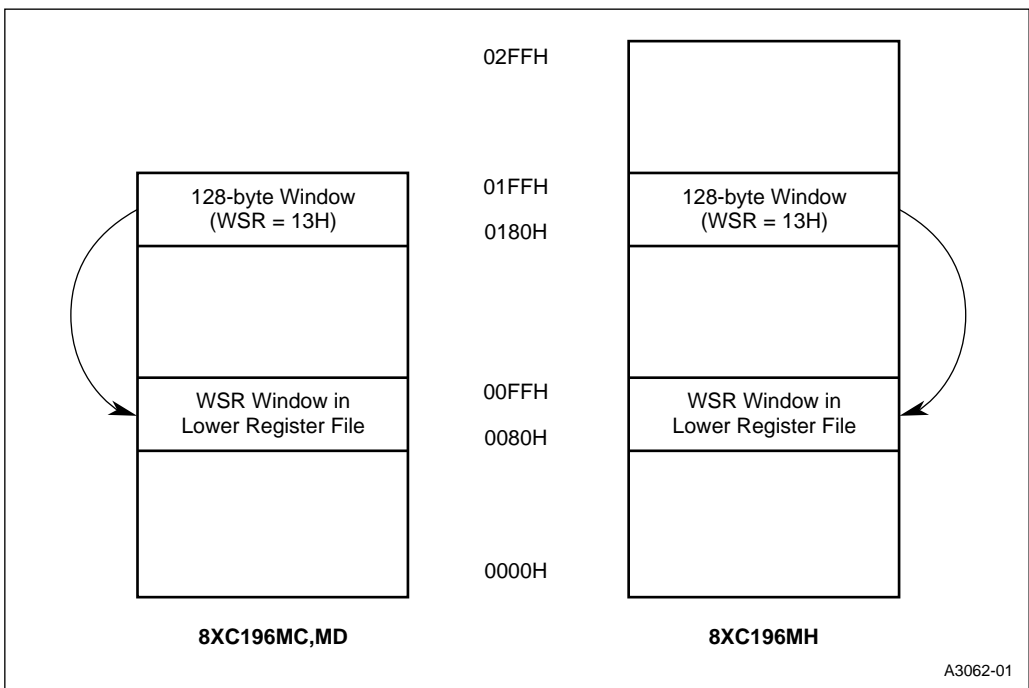


Figure 4-2. Windowing

NOTE

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

4.2.1 Selecting a Window

The window selection register (Figure 4-3) selects a window to be mapped into the top of the lower register file.

Table 4-9 provides a quick reference of WSR values for windowing the peripheral SFRs. Table 4-10 on page 4-14 lists the WSR values for windowing the upper register file.

WSR		Address:	0014H
		Reset State:	00H
The window selection register (WSR) maps 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.			
7			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 4-9 on page 4-13 or Table 4-10 on page 4-14. See Table 4-9 for peripheral SFR windows or Table 4-10 for upper register file windows.	

Figure 4-3. Window Selection (WSR) Register

Table 4-9. Selecting a Window of Peripheral SFRs

Peripherals	WSR Value for 32-byte Window (00E0–00FFH)	WSR Value for 64-byte Window (00C0–00FFH)	WSR Value for 128-byte Window (0080–00FFH)
Port 2 Waveform generator Port 7 (MD only)	7EH	3FH	1FH
Peripheral interrupts Pulse-width modulator A/D converter Frequency generator (MD only) Reset control (MH only)	7DH	3EH	
Port 1 (MH only) Serial I/O port (MH only)	7CH		
Timer 1–2 EPA compare 2–3 (MC) EPA compare 0–5 (MD) EPA compare 0–2 (MH)	7BH	3DH	1EH
EPA capture/compare 0–3 (MC) EPA compare 0–1 (MC) EPA capture/compare 0–5 (MD) EPA capture/compare 0–1 (MH) EPA compare 3 (MH)	7AH		

Table 4-10. Selecting a Window of the Upper Register File

Register RAM Locations	WSR Value for 32-byte Window (00E0–00FFH)	WSR Value for 64-byte Window (00C0–00FFH)	WSR Value for 128-byte Window (0080–00FFH)
8XC196MH Only			
02E0–02FFH	57H	2BH	15H
02C0–02DFH	56H		
02A0–02BFH	55H	2AH	14H
0280–029FH	54H		
0260–027FH	53H	29H	14H
0240–025FH	52H		
0220–023FH	51H	28H	14H
0200–021FH	50H		
8XC196MC, 8XC196MD, and 8XC196MH			
01E0–01FFH	4FH	27H	13H
01C0–01DFH	4EH		
01A0–01BFH	4DH	26H	12H
0180–019FH	4CH		
0160–017FH	4BH	25H	12H
0140–015FH	4AH		
0120–013FH	49H	24H	12H
0100–011FH	48H		

4.2.2 Addressing a Location Through a Window

After you have selected the desired window, you need to know the windowed direct address of the memory location (the address in the lower register file). Calculate the windowed direct address as follows:

1. Subtract the base address of the area to be remapped (from Table 4-11 on page 4-15) 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 4-12 on page 4-15). The result is the windowed direct address.

Appendix C includes a table of the windowable SFRs with the WSR values and windowed direct addresses for each window size. Examples beginning on page 4-16 explain how to determine the WSR value and windowed direct address for any windowable location. An additional example shows how to set up a window by using the linker locator.

Table 4-11. Windows

Base Address	WSR Value for 32-byte Window (00E0–00FFH)	WSR Value for 64-byte Window (00C0–00FFH)	WSR Value for 128-byte Window (0080–00FFH)
Peripheral SFRs			
1FE0H	7FH (Note)	3FH (Note)	1FH (Note)
1FC0H	7EH		
1FA0H	7DH		
1F80H	7CH	3EH	1EH
1F60H	7BH		
1F40H	7AH		
1F20H	79H	3DH	1EH
1F00H	78H		
02E0H	57H		
02C0H	56H	2BH	15H
02A0H	55H		
0280H	54H		
0260H	53H	2AH	14H
0240H	52H		
0220H	51H		
0200H	50H	29H	13H
01E0H	4FH		
01C0H	4EH		
01A0H	4DH	27H	12H
0180H	4CH		
0160H	4BH		
0140H	4AH	26H	12H
0120H	49H		
0100H	48H		
		25H	12H
		24H	12H

NOTE: Locations 1FE0–1FFFH contain memory-mapped SFRs that cannot be accessed through a window. Reading these locations through a window returns FFH; writing these locations through a window has no effect.

Table 4-12. Windowed Base Addresses

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

Appendix C includes a table of the windowable SFRs with the WSR 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.

4.2.2.1 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 register-direct addressing through a 32-byte window. Table 4-11 on page 4-15 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 (00E0H, from Table 4-12). The direct address is 00EBH ($000BH + 00E0H$).

4.2.2.2 64-byte Windowing Example

Assume that you wish to access the WG_CONTROL register (location 1FCCH) with register-direct addressing through a 64-byte window. Table 4-11 shows that you need to write 3FH to the window selection register. It also shows that the base address of the 64-byte memory area is 1FC0H. To determine the offset, subtract that base address from the address to be accessed ($1FCCH - 1FC0H = 000CH$). Add the offset to the base address of the window in the lower register file (00C0H, from Table 4-12). The direct address is 00CCH ($000CH + 00C0H$).

4.2.2.3 128-byte Windowing Example

Assume that you wish to access location 1F42H (the EPA0_TIME register) with register-direct addressing through a 128-byte window. Table 4-11 shows that you need to write 1EH to the window selection register. It also shows that the base address of the 128-byte memory area is 1F00H. To determine the offset, subtract that base address from the address to be accessed ($1F42H - 1F00H = 0042H$). Add the offset to the base address of the window in the lower register file (0080H, from Table 4-12). The direct address is 00C2H ($0042H + 0080H$).

4.2.2.4 Unsupported Locations Windowing Example

Assume that you wish to access location 1FF1H (the P5_MODE register, a memory-mapped SFR) with register-direct addressing through a 128-byte window. This location is in the range of addresses (1FE0–1FFFH) that cannot be windowed. 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 access the peripheral SFRs in the range of 1F80–1FDFH with their windowed direct addresses.

4.2.2.5 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
var2: dsw 1                    ;overlayable segment
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-01ffh) windowsize(32)
```

The above linker controls tell the linker to use registers 0100–01FFH 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*.

4.2.3 Windowing and Addressing Modes

Once windowing is enabled, the windowed locations can be accessed both through the window using direct (8-bit) addressing and by the usual 16-bit 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 the WSR. To enable direct access to a particular location in the lower register file, you can select a smaller window that does not cover that location.

When windowing is enabled:

- a register-direct instruction that uses an address within the lower register file actually accesses the window in the upper register file;
- an indirect, indexed, or zero-register 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 register-direct and indexed addressing when using windowing.

```
PUSHA                ; pushes the contents of WSR onto the stack
LDB WSR, #12H       ; select window 12H, a 128-byte block
                    ; The next instruction uses register-direct addr
                    ; mem_word(40H)←mem_word(40H) + mem_word(380H)
ADD 40H, 80H        ; The next two instructions use indirect addr
                    ; mem_word(40H)←mem_word(40H) + mem_word(80H +0)
ADD 40H, 80H[0]     ; mem_word(40H)←mem_word(40H) + mem_word(380H +0)
ADD 40H, 380H[0]
POPA                ; reloads the previous contents into WSR
```




5

Standard and PTS Interrupts

CHAPTER 5

STANDARD AND PTS INTERRUPTS

This chapter describes the interrupt control circuitry, priority scheme, and timing for standard and peripheral transaction server (PTS) interrupts. It discusses the three special interrupts and the seven PTS modes, four of which are used with the EPA to provide a software serial I/O channel for both synchronous and asynchronous transfers and receptions. It also explains interrupt programming and control.

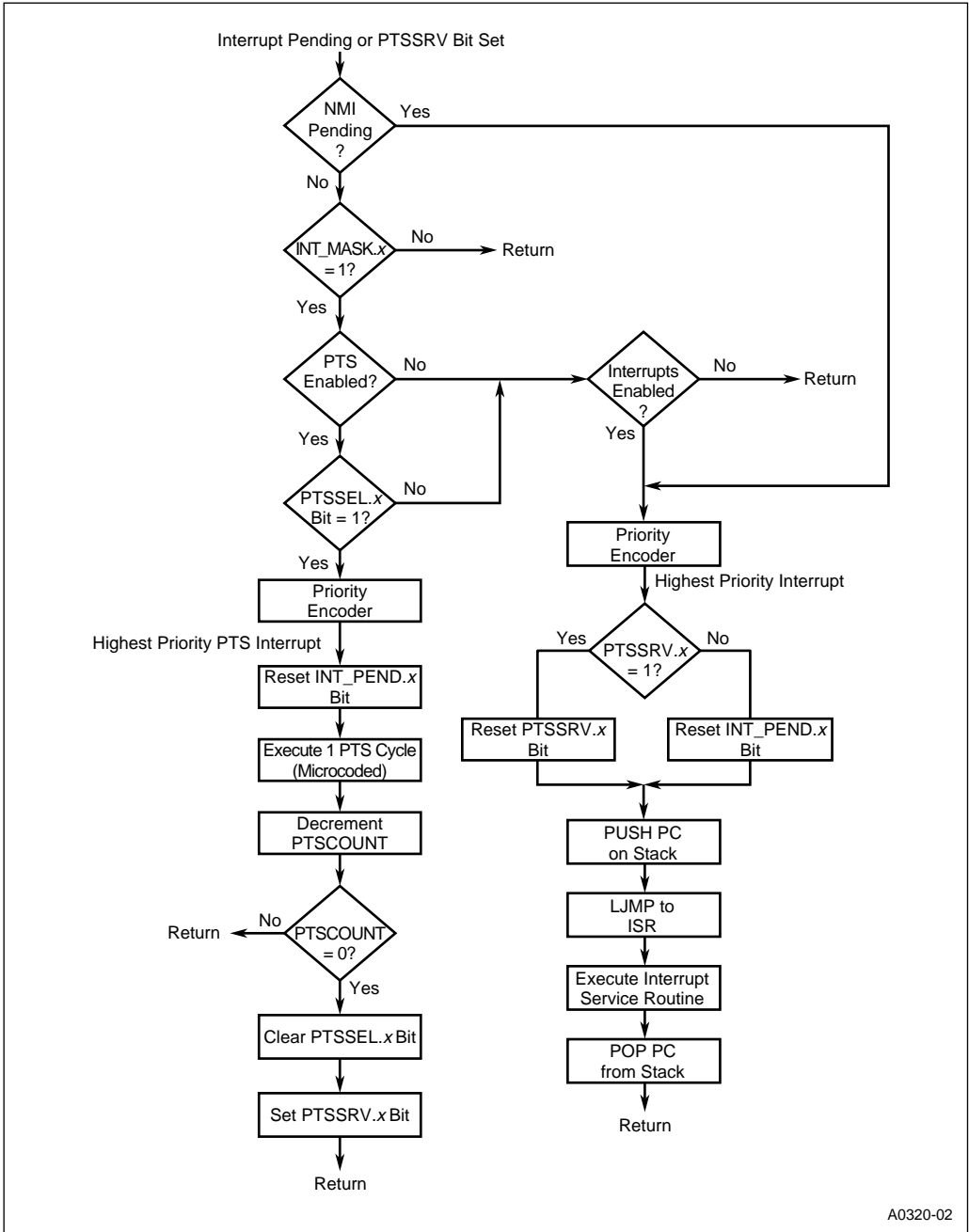
5.1 OVERVIEW OF INTERRUPTS

The interrupt control circuitry within a microcontroller permits real-time events to control program flow. When an event generates an interrupt, the device 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 device receives the request, performs the service, and returns to the task that was interrupted.

This 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 upper and lower interrupt vectors in special-purpose memory (see Chapter 4, "Memory Partitions") contain the interrupt service routines' addresses. 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, trap, and unimplemented opcode) to be serviced by the PTS instead of the interrupt controller.

The PTS supports seven 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; manage multiple analog-to-digital (A/D) conversions; and transmit and receive serial data in either asynchronous or synchronous mode (MC, MD only). 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 5-24). When a PTS interrupt occurs, the priority encoder selects the appropriate vector and fetches the PTS control block (PTSCB).



A0320-02

Figure 5-1. Flow Diagram for PTS and Standard Interrupts

Figure 5-1 illustrates the interrupt processing flow. In this flow diagram, “INT_MASK” represents both the INT_MASK and INT_MASK1 registers, and “INT_PEND” represents both the INT_PEND and INT_PEND1 registers.

5.2 INTERRUPT SIGNALS AND REGISTERS

Table 5-1 describes the external interrupt signals and Table 5-2 describes the control and status registers for both the interrupt controller and PTS.

Table 5-1. Interrupt Signals

Port Pin	Interrupt Signal	Type	Description
—	EXTINT	I	<p>External Interrupt</p> <p>This programmable interrupt is controlled by the WG_PROTECT register. This register controls whether the interrupt is edge triggered or sampled and whether a rising edge/high level or falling edge/low level activates the interrupt.</p> <p>In powerdown mode, asserting the EXTINT signal for at least 50 ns causes the device to resume normal operation. The interrupt need not be enabled. 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.</p> <p>In idle mode, asserting any enabled interrupt causes the device to resume normal operation.</p>
—	NMI	I	<p>Nonmaskable Interrupt</p> <p>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.</p>

Table 5-2. Interrupt and PTS Control and Status Registers

Mnemonic	Address	Description
INT_MASK INT_MASK1	0008H 0013H	<p>Interrupt Mask Registers</p> <p>These registers enable/disable each maskable interrupt (that is, each interrupt except unimplemented opcode, software trap, and NMI).</p>
INT_PEND INT_PEND1	0009H 0012H	<p>Interrupt Pending Registers</p> <p>The bits in this register are set by hardware to indicate that an interrupt is pending.</p>
PI_MASK	1FBCH	<p>Peripheral Interrupt Mask</p> <p>The bits in this register enable and disable (mask) the timer 1 and 2 overflow/underflow interrupt requests, the waveform generator interrupt request (MC, MD), the EPA compare-only channel 5 interrupt request (MD), and the serial port error interrupts (MH).</p>

Table 5-2. Interrupt and PTS Control and Status Registers (Continued)

Mnemonic	Address	Description
PI_PEND	1FBEH	Peripheral Interrupt Pending Any bit set indicates a pending 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 routine or a standard interrupt service routine for each of the maskable interrupt requests.
PTSSRV	0006H, 0007H	PTS Service Register The bits in this register are set by hardware to request an end-of-PTS interrupt.

5.3 INTERRUPT SOURCES AND PRIORITIES

Table 5-3 lists the interrupts sources, their default priorities (30 is highest and 0 is lowest), and their vector addresses. The unimplemented opcode and software trap interrupts are not prioritized; they go directly to the interrupt controller for servicing. The priority encoder determines the priority of all other pending interrupt requests. NMI has the highest priority of all prioritized interrupts, PTS interrupts have the next highest priority, and standard interrupts have the lowest. The priority encoder 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.

Table 5-3. Interrupt Sources, Vectors, and Priorities

Interrupt Source	Mnemonic	Interrupt Controller Service			PTS Service		
		Name	Vector	Priority	Name	Vector	Priority
Nonmaskable Interrupt	NMI	INT15	203EH	30	—	—	—
EXTINT Pin	EXTINT	INT14	203CH	14	PTS14	205CH	29
WF Gen (MC)	PI	INT13	203AH	13	PTS13	205AH	28
WF Gen & EPA Comp 5 (MD)	PI †						
Waveform Generator (MH)	WG						
Reserved (MC)	—	INT12	2038H	12	PTS12	2058H	27
EPA Cap/Comp 5 (MD)	EPA5						
SIO0 & 1 Receive Err (MH)	SPI †						
Reserved (MC)	—	INT11	2036H	11	PTS11	2056H	26
EPA Compare 4 (MD)	COMP4						
SIO1 Receive (MH)	RI1						
Reserved (MC)	—	INT10	2034H	10	PTS10	2054H	25
EPA Cap/Comp 4 (MD)	EPA4						
SIO0 Receive (MH)	RI0						
EPA Compare 3 (MC, MD)	COMP3	INT09	2032H	09	PTS09	2052H	24
SIO1 Transmit (MH)	TI1						
EPA Cap/Comp 3 (MC, MD)	EPA3	INT08	2030H	08	PTS08	2050H	23
SIO0 Transmit (MH)	TI0						
Unimplemented Opcode	—	—	2012H	—	—	—	—
Software TRAP Instruction	—	—	2010H	—	—	—	—
EPA Compare 2 (MC, MD)	COMP2	INT07	200EH	07	PTS07	204EH	22
EPA Compare 3 (MH)	COMP3						
EPA Cap/Comp 2 (MC, MD)	EPA2	INT06	200CH	06	PTS06	204CH	21
EPA Compare 2 (MH)	COMP2						
EPA Compare 1	COMP1	INT05	200AH	05	PTS05	204AH	20
EPA Capture/Compare 1	EPA1	INT04	2008H	04	PTS04	2048H	19
EPA Compare 0	COMP0	INT03	2006H	03	PTS03	2046H	18
EPA Capture/Compare 0	EPA0	INT02	2004H	02	PTS02	2044H	17
A/D Conversion Complete	AD_DONE	INT01	2002H	01	PTS01	2042H	16
Timer 1 or 2 Overflow	OVRTM †	INT00	2000H	00	PTS00	2040H	15

† PTS service is not useful for multiplexed interrupts because the PTS cannot readily determine the source of these interrupts.

5.3.1 Special Interrupts

This microcontroller has three special interrupt sources that are always enabled: unimplemented opcode, software trap, and NMI. 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. Of these three, only NMI goes through the transition detector and priority encoder. 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.

5.3.1.1 Unimplemented Opcode

If the CPU attempts to execute an unimplemented opcode, an indirect vector through location 2012H 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. The unimplemented opcode interrupt prevents other interrupt requests from being acknowledged until after the next instruction is executed.

5.3.1.2 Software Trap

The TRAP instruction (opcode F7H) causes an interrupt call that is vectored through location 2010H. The TRAP instruction provides a single-instruction interrupt that is useful when debugging software or generating software interrupts. The TRAP instruction prevents other interrupt requests from being acknowledged until after the next instruction is executed.

5.3.1.3 NMI

The external NMI pin generates a nonmaskable interrupt for implementation of critical interrupt routines. NMI has the highest priority of all the prioritized interrupts. It is passed directly from the transition detector to the priority encoder, and it vectors indirectly through location 203EH. 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.

5.3.2 External Interrupt Pin

The protection circuitry in the waveform generator (Figure 5-2) monitors the external interrupt (EXTINT) signal. When it detects a valid event on the input, it simultaneously disables the waveform generator outputs and generates an EXTINT interrupt request. Bits 2 and 3 in the waveform generator protection (WG_PROTECT) register (Figure 9-9 on page 9-15) select the type of external event that will generate an interrupt request: a falling or rising edge or a low or high level.

When the level-sensitive event is selected, the external interrupt signal must remain asserted for at least $24 T_{XTAL1}$ ($24/F_{XTAL1}$) to be recognized as a valid interrupt. When the signal is asserted, the level sampler samples the level of the signal three times during a $24 T_{XTAL1}$ period. When a valid level occurs, the level sampler generates a single output pulse. The output pulse generates the EXTINT interrupt request. The level-sensitive mode is useful in noisy environments, where a noise spike might cause an unintentional interrupt request.

When an edge-triggered event is selected, the input must remain asserted for at least two T_{XTAL1} ($2/F_{XTAL1}$) to be recognized as a valid interrupt. When a valid transition occurs, the transition detector generates a single output pulse. The output pulse generates the EXTINT interrupt request.

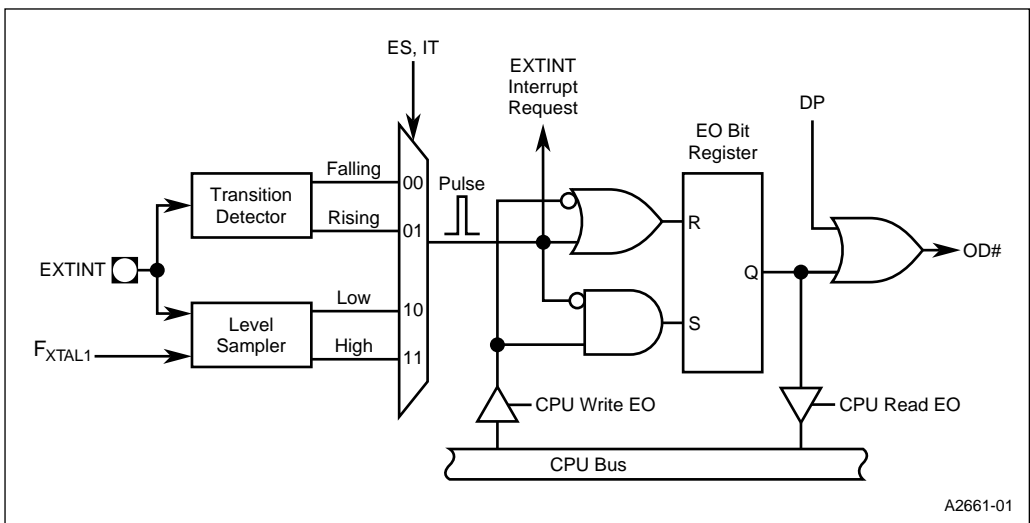


Figure 5-2. Waveform Generator Protection Circuitry

5.3.3 Multiplexed Interrupt Sources

The PI (MD), OVRTM (M_x), and SPI (MH) interrupts have multiple sources (see Table 5-3 on page 5-5). An individual source will generate the interrupt only if software enables both the interrupt source and multiplexed interrupt. To enable the multiplexed interrupt, set the appropriate bit in the interrupt mask register (Figures 5-7 and 5-8). To enable an interrupt source, set the appropriate bit in the PI_MASK register (Figure 5-9 on page 5-17). Figure 5-3 shows the flow for the timer interrupt.

NOTE

Although the PI interrupt on the 8XC196MC has a single source (the waveform generator), software must still enable both the source interrupt (WG) in the PI_PEND register and the PI interrupt in the INT_MASK register.

The interrupt service routine should read the PI_PEND (Figure 5-12 on page 5-23) register to determine the source of the interrupt. Before executing the return instruction, the interrupt service routine should check to see if any of the other interrupt sources are pending. Generally, PTS interrupt service is not useful for multiplexed interrupts because the PTS cannot readily determine the interrupt source.

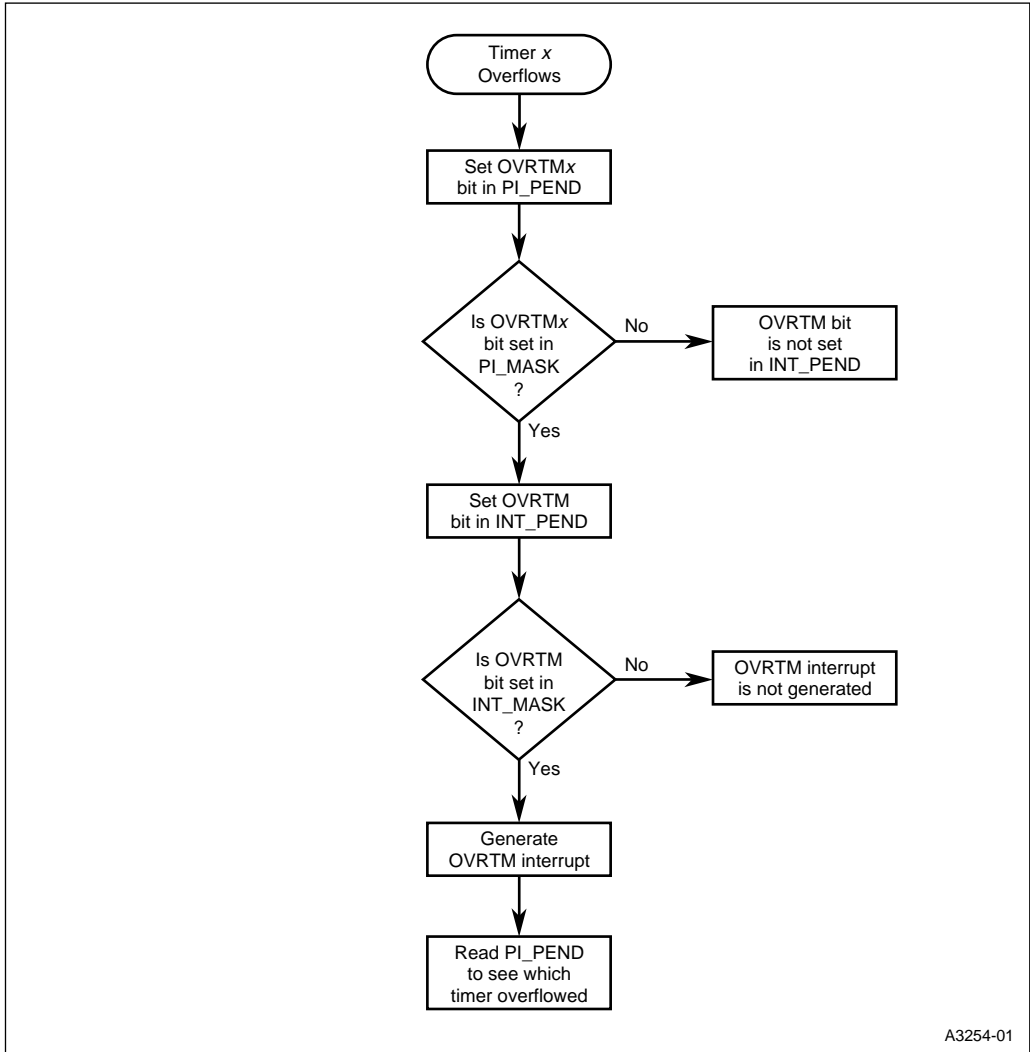


Figure 5-3. Flow Diagram for the OVRMx Interrupt

5.3.4 End-of-PTS Interrupts

When the PTSCOUNT register decrements to zero at the end of a single transfer, block transfer, A/D scan, or serial I/O routine, hardware clears the corresponding bit in the PTSSEL register, (Figure 5-6 on page 5-14) 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 EPA0 interrupt if PTSSEL.2 is set. The interrupt vectors through 2044H, but the corresponding end-of-PTS interrupt vectors through 2004H, the standard EPA0 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.

5.4 INTERRUPT LATENCY

Interrupt latency is the total delay between the time that the interrupt request is generated (not acknowledged) and the time that the device begins executing either the standard 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 finishes executing. 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 and prepared a few state times before they are executed. Thus, the maximum delay between interrupt request and acknowledgment is four state times plus 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.

5.4.1 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
- any of these eight *protected instructions*: DI, EI, DPTS, EPTS, POPA, POPF, PUSHA, PUSHF (see Appendix A for descriptions of these instructions)
- any of the read-modify-write instructions: AND, ANDB, OR, ORB, XOR, XORB

Both the unimplemented opcode interrupt and the software trap interrupt prevent other interrupt requests from being acknowledged until after the next instruction is executed.

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 5-4 on page 5-12 for PTS cycle execution times.

5.4.2 Calculating Latency

The maximum latency occurs when the interrupt request occurs too late 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 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. (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 instruction in your calculation instead of NORML. See Appendix A for instruction execution times.)
- For standard interrupts only, the response time to get the vector and force the call.
 - 11 state times for an internal stack or 13 for an external stack (assuming a zero-wait-state bus)

5.4.2.1 Standard Interrupt Latency

The worst-case delay for a standard interrupt is 56 state times ($4 + 39 + 11 + 2$) if the stack is in external memory (Figure 5-4). 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.

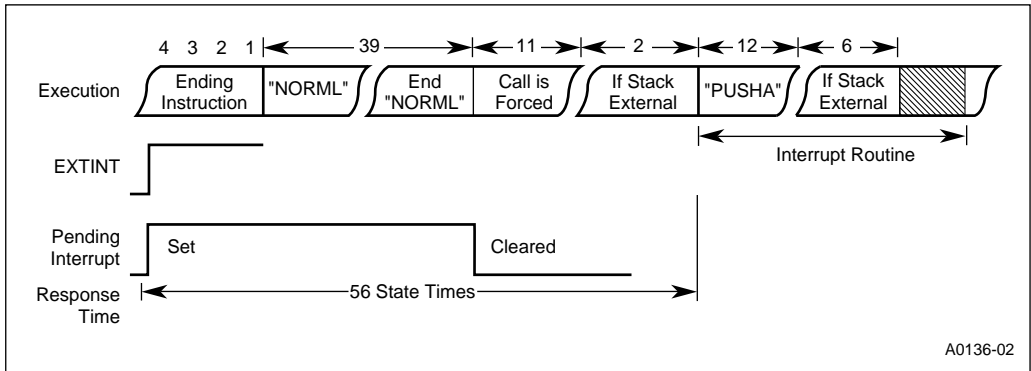


Figure 5-4. Standard Interrupt Response Time

5.4.2.2 PTS Interrupt Latency

The maximum delay for a PTS interrupt is 43 state times (4 + 39) as shown in Figure 5-5. 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 5-4 for execution times for PTS cycles.

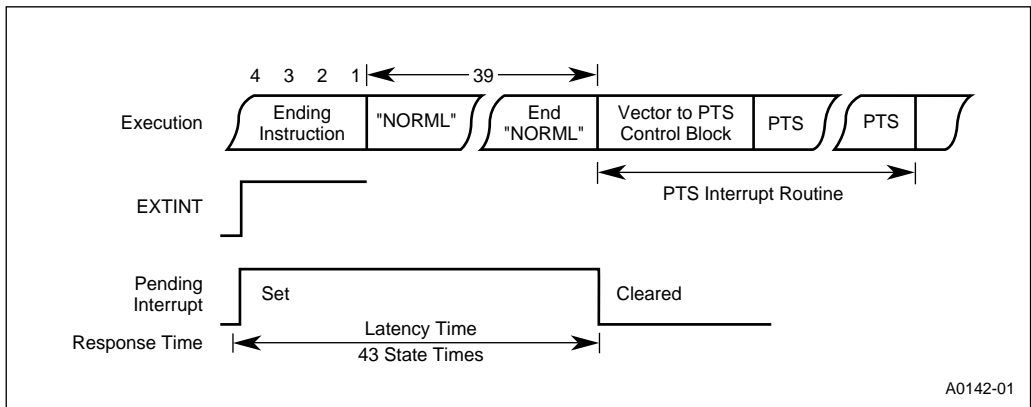


Figure 5-5. PTS Interrupt Response Time

Table 5-4. Execution Times for PTS Cycles

PTS Mode	Execution Time (in State Times)
Single transfer mode register/register† memory/register† memory/memory†	18 per byte or word transfer + 1 21 per byte or word transfer + 1 24 per byte or word transfer + 1
Block transfer mode register/register† memory/register† memory/memory†	13 + 7 per byte or word transfer (1 minimum) 16 + 7 per byte or word transfer (1 minimum) 19 + 7 per byte or word transfer (1 minimum)
A/D scan mode register/register† register/memory†	21 25
ASIO receive mode (MC, MD only) Majority disabled	24 + 2 (if parity enabled)
Majority enabled	36 + sample time (second sample) 36 + 7 + sample time (third sample) 36 + 2 (if parity enabled)
ASIO transmit mode (MC, MD only)	29 + 3 (if parity enabled)
SSIO receive mode (MC, MD only)	29 (receive data bit) 21 (no reception)
SSIO transmit mode (MC, MD only)	30 (transmit data bit) 20 (no transmission)

† *Register* indicates an access to the register file or peripheral SFR. *Memory* indicates an access to a memory-mapped register, I/O, or memory. See Table 4-1 on page 4-2 for address information.

5.5 PROGRAMMING THE INTERRUPTS

The PTS select register (PTSSEL) selects either PTS service or a standard software interrupt service routine for each of the maskable interrupt requests (see Figure 5-6). The bits in the interrupt mask registers, INT_MASK and INT_MASK1, enable or disable (mask) individual interrupts (see Figures 5-7 and 5-8). For the multiplexed interrupt sources, bits in the PI_MASK register (Figure 5-9 on page 5-17) enable or disable (mask) the individual interrupt sources. With the exception of the nonmaskable interrupt (NMI) bit (INT_MASK1.7), setting a bit enables the corresponding interrupt source and clearing a bit disables the source.

To disable any interrupt, clear its mask bit. To enable an interrupt for standard interrupt service, set its mask bit and clear its PTS select bit. To enable an interrupt for PTS service, set both the mask bit and the PTS select bit.

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 5-24) and use the EPTS instruction to globally enable the PTS. When you assign an interrupt to a standard software service routine, use the EI (enable interrupts) instruction to globally enable interrupt servicing.

NOTE

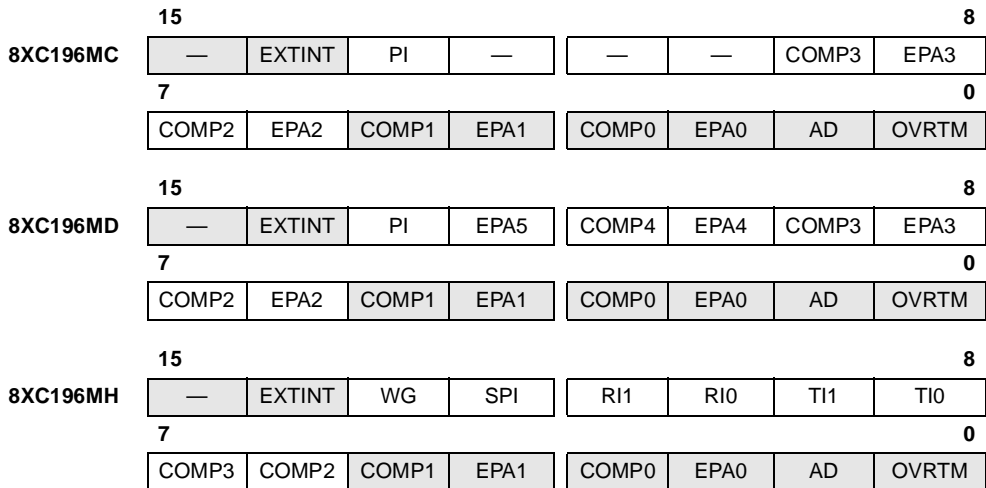
The DI (disable interrupts) instruction does not disable PTS service. However, it does disable service for the end-of-PTS interrupt request. If an interrupt request occurs while interrupts are disabled, the corresponding pending bit is set in the INT_PEND or INT_PEND1 register.

PTS service is not useful for multiplexed interrupts because the PTS cannot readily determine the source of these interrupts.

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. When PTSCOUNT reaches zero, hardware clears the corresponding PTSSEL bit. The PTSSEL bit must be set manually to re-enable the PTS channel.



Bit Number	Function																																																				
15	Reserved; for compatibility with future devices, write zero to this bit.																																																				
14:0 [†]	<p>Setting a bit causes the corresponding interrupt to be handled by a PTS microcode routine. The PTS interrupt vector locations are as follows:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 25%;">Bit Mnemonic</th> <th style="width: 25%;">PTS Vector</th> <th style="width: 25%;">Bit Mnemonic</th> <th style="width: 25%;">PTS Vector</th> </tr> </thead> <tbody> <tr> <td>EXTINT</td> <td>205CH</td> <td>TI0 (MH)</td> <td>2050H</td> </tr> <tr> <td>PI (MC, MD)^{††}</td> <td>205AH</td> <td>COMP2 (MC,MD)</td> <td>204EH</td> </tr> <tr> <td>WG (MH)</td> <td>205AH</td> <td>COMP3 (MH)</td> <td>204EH</td> </tr> <tr> <td>EPA5 (MD)</td> <td>2058H</td> <td>EPA2 (MC, MD)</td> <td>204CH</td> </tr> <tr> <td>SPI (MH)^{††}</td> <td>2058H</td> <td>COMP2 (MH)</td> <td>204CH</td> </tr> <tr> <td>COMP4 (MD)</td> <td>2056H</td> <td>COMP1</td> <td>204AH</td> </tr> <tr> <td>R11 (MH)</td> <td>2056H</td> <td>EPA1</td> <td>2048H</td> </tr> <tr> <td>EPA4 (MD)</td> <td>2054H</td> <td>COMP0</td> <td>2046H</td> </tr> <tr> <td>R10 (MH)</td> <td>2054H</td> <td>EPA0</td> <td>2044H</td> </tr> <tr> <td>COMP3 (MC, MD)</td> <td>2052H</td> <td>AD</td> <td>2042H</td> </tr> <tr> <td>TI1 (MH)</td> <td>2052H</td> <td>OVRTM^{††}</td> <td>2040H</td> </tr> <tr> <td>EPA3 (MC, MD)</td> <td>2050H</td> <td></td> <td></td> </tr> </tbody> </table> <p>^{††} PTS service is not useful for multiplexed interrupts because the PTS cannot readily determine the source of these interrupts.</p>	Bit Mnemonic	PTS Vector	Bit Mnemonic	PTS Vector	EXTINT	205CH	TI0 (MH)	2050H	PI (MC, MD) ^{††}	205AH	COMP2 (MC,MD)	204EH	WG (MH)	205AH	COMP3 (MH)	204EH	EPA5 (MD)	2058H	EPA2 (MC, MD)	204CH	SPI (MH) ^{††}	2058H	COMP2 (MH)	204CH	COMP4 (MD)	2056H	COMP1	204AH	R11 (MH)	2056H	EPA1	2048H	EPA4 (MD)	2054H	COMP0	2046H	R10 (MH)	2054H	EPA0	2044H	COMP3 (MC, MD)	2052H	AD	2042H	TI1 (MH)	2052H	OVRTM ^{††}	2040H	EPA3 (MC, MD)	2050H		
Bit Mnemonic	PTS Vector	Bit Mnemonic	PTS Vector																																																		
EXTINT	205CH	TI0 (MH)	2050H																																																		
PI (MC, MD) ^{††}	205AH	COMP2 (MC,MD)	204EH																																																		
WG (MH)	205AH	COMP3 (MH)	204EH																																																		
EPA5 (MD)	2058H	EPA2 (MC, MD)	204CH																																																		
SPI (MH) ^{††}	2058H	COMP2 (MH)	204CH																																																		
COMP4 (MD)	2056H	COMP1	204AH																																																		
R11 (MH)	2056H	EPA1	2048H																																																		
EPA4 (MD)	2054H	COMP0	2046H																																																		
R10 (MH)	2054H	EPA0	2044H																																																		
COMP3 (MC, MD)	2052H	AD	2042H																																																		
TI1 (MH)	2052H	OVRTM ^{††}	2040H																																																		
EPA3 (MC, MD)	2050H																																																				

[†] On the 8XC196MC device bits 10–12 are reserved. For compatibility with future devices, write zeros to these bits.

Figure 5-6. PTS Select (PTSSEL) Register

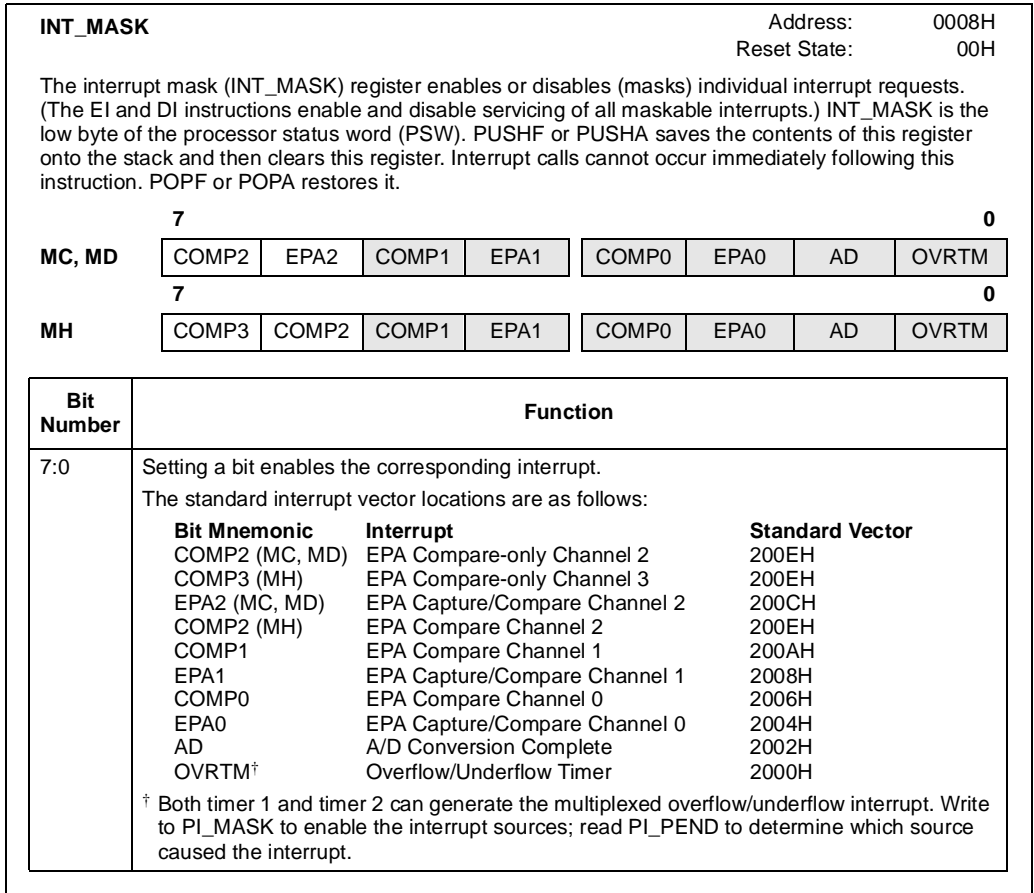
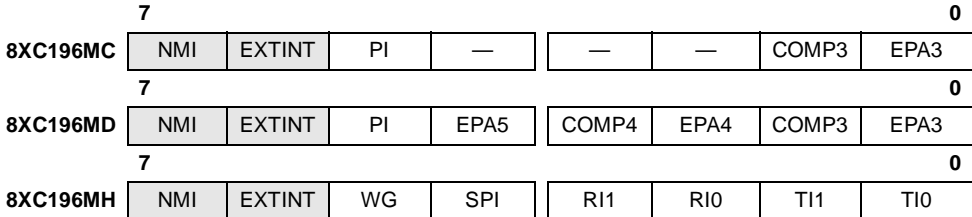


Figure 5-7. Interrupt Mask (INT_MASK) Register

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.



Bit Number	Function																																													
7:0 [†]	Setting a bit enables the corresponding interrupt. The standard interrupt vector locations are as follows: <table style="width: 100%; margin-top: 10px;"> <thead> <tr> <th style="text-align: left;">Bit Mnemonic</th> <th style="text-align: left;">Interrupt</th> <th style="text-align: left;">Standard Vector</th> </tr> </thead> <tbody> <tr><td>NMI</td><td>Nonmaskable Interrupt</td><td>203EH</td></tr> <tr><td>EXTINT</td><td>EXTINT pin</td><td>203CH</td></tr> <tr><td>PI (MC, MD)^{††}</td><td>Multiplexed Peripheral Interrupt</td><td>203AH</td></tr> <tr><td>WG (MH)</td><td>Waveform Generator</td><td>203AH</td></tr> <tr><td>EPA5 (MD)</td><td>EPA Capture/Compare Channel 5</td><td>2038H</td></tr> <tr><td>SPI (MH)^{†††}</td><td>Serial Port</td><td>2038H</td></tr> <tr><td>COMP4 (MD)</td><td>EPA Compare Channel 4</td><td>2036H</td></tr> <tr><td>RI1 (MH)</td><td>SIO 1 Receive</td><td>2036H</td></tr> <tr><td>EPA4 (MD)</td><td>EPA Capture/Compare Channel 4</td><td>2034H</td></tr> <tr><td>RI0 (MH)</td><td>SIO 0 Receive</td><td>2034H</td></tr> <tr><td>COMP3 (MC, MD)</td><td>EPA Compare Channel 3</td><td>2032H</td></tr> <tr><td>TI1 (MH)</td><td>SIO 1 Transmit</td><td>2032H</td></tr> <tr><td>EPA3 (MC, MD)</td><td>EPA Capture/Compare Channel 3</td><td>2030H</td></tr> <tr><td>TI0 (MH)</td><td>SIO 0 Transmit</td><td>2030H</td></tr> </tbody> </table> <p>^{††} The waveform generator and the EPA compare-only channel 5 can generate this interrupt. Write to PI_MASK to enable the interrupt sources; read PI_PEND to determine which source caused the interrupt.</p> <p>^{†††} SIO 0 and SIO 1 can generate this interrupt. Write to PI_MASK to enable the interrupt sources; read PI_PEND to determine which source caused the interrupt.</p>	Bit Mnemonic	Interrupt	Standard Vector	NMI	Nonmaskable Interrupt	203EH	EXTINT	EXTINT pin	203CH	PI (MC, MD) ^{††}	Multiplexed Peripheral Interrupt	203AH	WG (MH)	Waveform Generator	203AH	EPA5 (MD)	EPA Capture/Compare Channel 5	2038H	SPI (MH) ^{†††}	Serial Port	2038H	COMP4 (MD)	EPA Compare Channel 4	2036H	RI1 (MH)	SIO 1 Receive	2036H	EPA4 (MD)	EPA Capture/Compare Channel 4	2034H	RI0 (MH)	SIO 0 Receive	2034H	COMP3 (MC, MD)	EPA Compare Channel 3	2032H	TI1 (MH)	SIO 1 Transmit	2032H	EPA3 (MC, MD)	EPA Capture/Compare Channel 3	2030H	TI0 (MH)	SIO 0 Transmit	2030H
Bit Mnemonic	Interrupt	Standard Vector																																												
NMI	Nonmaskable Interrupt	203EH																																												
EXTINT	EXTINT pin	203CH																																												
PI (MC, MD) ^{††}	Multiplexed Peripheral Interrupt	203AH																																												
WG (MH)	Waveform Generator	203AH																																												
EPA5 (MD)	EPA Capture/Compare Channel 5	2038H																																												
SPI (MH) ^{†††}	Serial Port	2038H																																												
COMP4 (MD)	EPA Compare Channel 4	2036H																																												
RI1 (MH)	SIO 1 Receive	2036H																																												
EPA4 (MD)	EPA Capture/Compare Channel 4	2034H																																												
RI0 (MH)	SIO 0 Receive	2034H																																												
COMP3 (MC, MD)	EPA Compare Channel 3	2032H																																												
TI1 (MH)	SIO 1 Transmit	2032H																																												
EPA3 (MC, MD)	EPA Capture/Compare Channel 3	2030H																																												
TI0 (MH)	SIO 0 Transmit	2030H																																												

[†] On the 8XC196MC device bits 4–3 are reserved. For compatibility with future devices, write zeros to these bits.

Figure 5-8. Interrupt Mask 1 (INT_MASK1) Register

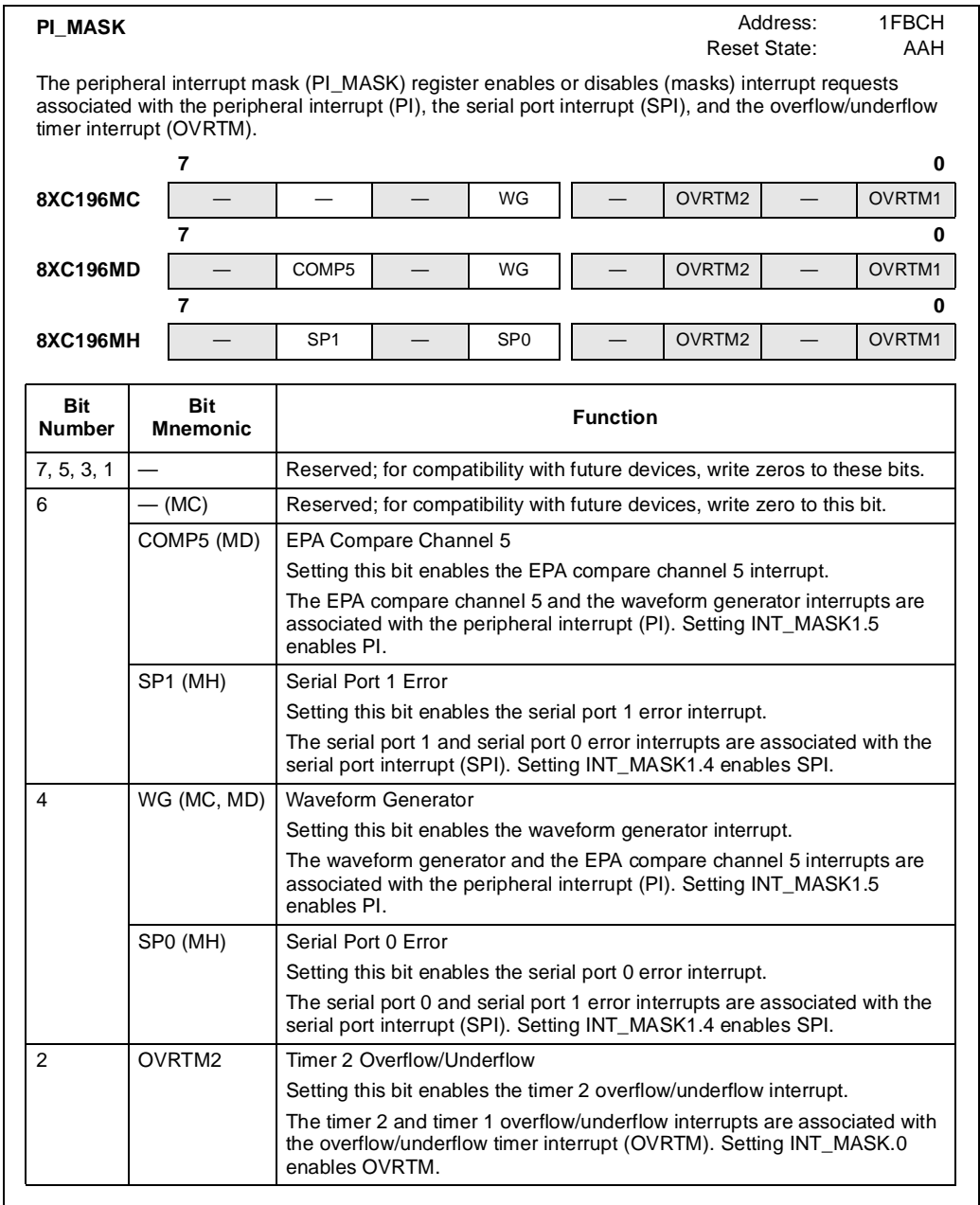


Figure 5-9. Peripheral Interrupt Mask (PI_MASK) Register

PI_MASK (Continued)		Address:	1FBCH								
		Reset State:	AAH								
The peripheral interrupt mask (PI_MASK) register enables or disables (masks) interrupt requests associated with the peripheral interrupt (PI), the serial port interrupt (SPI), and the overflow/underflow timer interrupt (OVRTM).											
	7		0								
8XC196MC	<table border="1" style="display: inline-table;"><tr><td style="width: 20px; height: 15px;">—</td><td style="width: 20px; height: 15px;">—</td><td style="width: 20px; height: 15px;">—</td><td style="width: 20px; height: 15px;">WG</td></tr></table>	—	—	—	WG	<table border="1" style="display: inline-table;"><tr><td style="width: 20px; height: 15px;">—</td><td style="width: 20px; height: 15px;">OVRTM2</td><td style="width: 20px; height: 15px;">—</td><td style="width: 20px; height: 15px;">OVRTM1</td></tr></table>	—	OVRTM2	—	OVRTM1	
—	—	—	WG								
—	OVRTM2	—	OVRTM1								
	7		0								
8XC196MD	<table border="1" style="display: inline-table;"><tr><td style="width: 20px; height: 15px;">—</td><td style="width: 20px; height: 15px;">COMP5</td><td style="width: 20px; height: 15px;">—</td><td style="width: 20px; height: 15px;">WG</td></tr></table>	—	COMP5	—	WG	<table border="1" style="display: inline-table;"><tr><td style="width: 20px; height: 15px;">—</td><td style="width: 20px; height: 15px;">OVRTM2</td><td style="width: 20px; height: 15px;">—</td><td style="width: 20px; height: 15px;">OVRTM1</td></tr></table>	—	OVRTM2	—	OVRTM1	
—	COMP5	—	WG								
—	OVRTM2	—	OVRTM1								
	7		0								
8XC196MH	<table border="1" style="display: inline-table;"><tr><td style="width: 20px; height: 15px;">—</td><td style="width: 20px; height: 15px;">SP1</td><td style="width: 20px; height: 15px;">—</td><td style="width: 20px; height: 15px;">SP0</td></tr></table>	—	SP1	—	SP0	<table border="1" style="display: inline-table;"><tr><td style="width: 20px; height: 15px;">—</td><td style="width: 20px; height: 15px;">OVRTM2</td><td style="width: 20px; height: 15px;">—</td><td style="width: 20px; height: 15px;">OVRTM1</td></tr></table>	—	OVRTM2	—	OVRTM1	
—	SP1	—	SP0								
—	OVRTM2	—	OVRTM1								
Bit Number	Bit Mnemonic	Function									
0	OVRTM1	Timer 1 Overflow/Underflow Setting this bit enables the timer 1 overflow/underflow interrupt. The timer 1 and timer 2 overflow/underflow interrupts are associated with the overflow/underflow timer interrupt (OVRTM). Setting INT_MASK.0 enables OVRTM.									

Figure 5-9. Peripheral Interrupt Mask (PI_MASK) Register (Continued)

5.5.1 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 14), from interrupting an A/D conversion-complete interrupt service routine (priority 01).

```

SERIAL_RI_ISR:
    PUSHA                                ; Save PSW, INT_MASK, INT_MASK1, & WSR
                                        ; (this disables all interrupts)
    LDB INT_MASK1, #01000000B           ; Enable EXTINT only
    EI                                    ; Enable interrupt servicing

                                        ; Service the AD_DONE interrupt

    POPA                                  ; Restore PSW, INT_MASK, INT_MASK1, &
                                        ; WSR registers
    RET

CSEG AT 02002H                           ; fill in interrupt table
    DCW AD_DONE_ISR                      END
  
```

Note that location 2002H in the interrupt vector table must be loaded with the value of the label `AD_DONE_ISR` before the interrupt request occurs and that the A/D conversion complete 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 4-12).

5.5.2 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 5-10 and 5-11). It sets the bit even if the individual interrupt is disabled (masked). Hardware clears the pending bit when the program vectors to the interrupt service routine. The interrupt service routine can read INT_PEND and INT_PEND1 to determine which interrupts are pending.

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

```
ANDB INT_PEND, #11111110B      ; Clears the OVRTM pending bit
ORB  INT_PEND, #00000001B      ; Sets the OVRTM 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 jump to the interrupt vector will not occur.

The PI (MD), SPI (MH), and OVRTM (Mx) interrupts have multiple sources. Read PI_PEND (Figure 5-12 on page 5-23) to determine which source generated the interrupt request. Reading PI_PEND clears all the bits. PI_PEND is a read-only register.

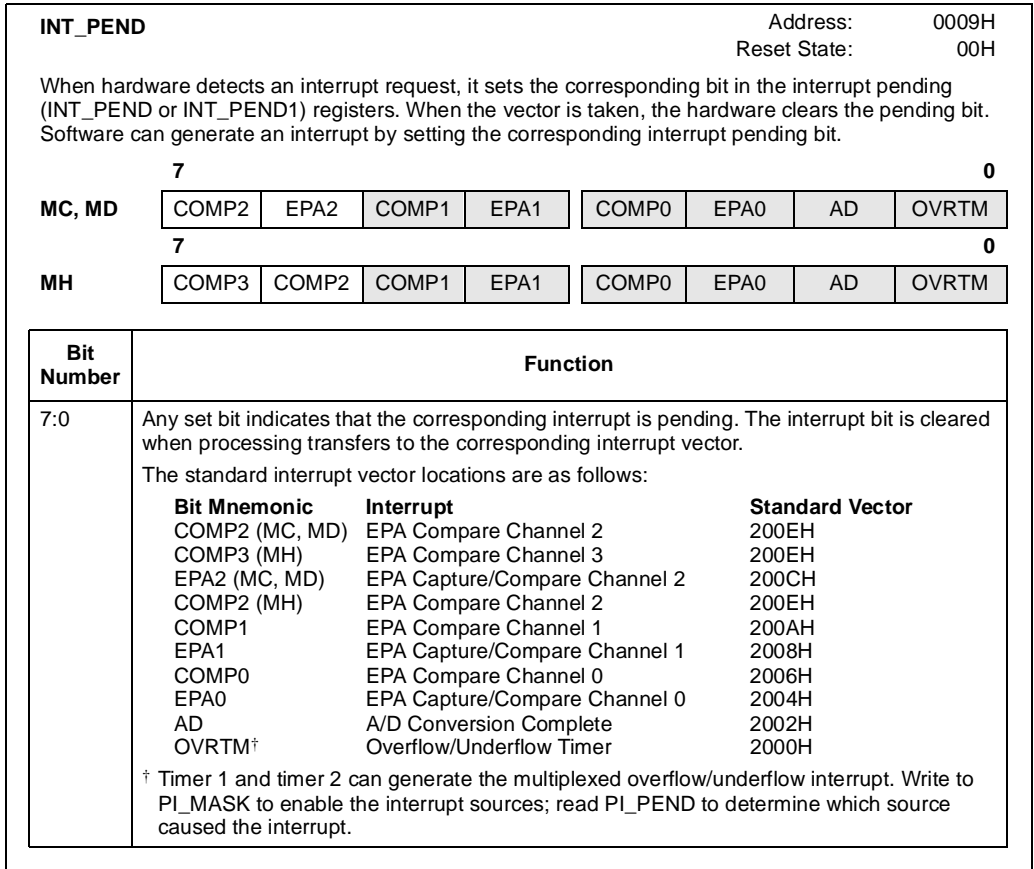


Figure 5-10. Interrupt Pending (INT_PEND) Register

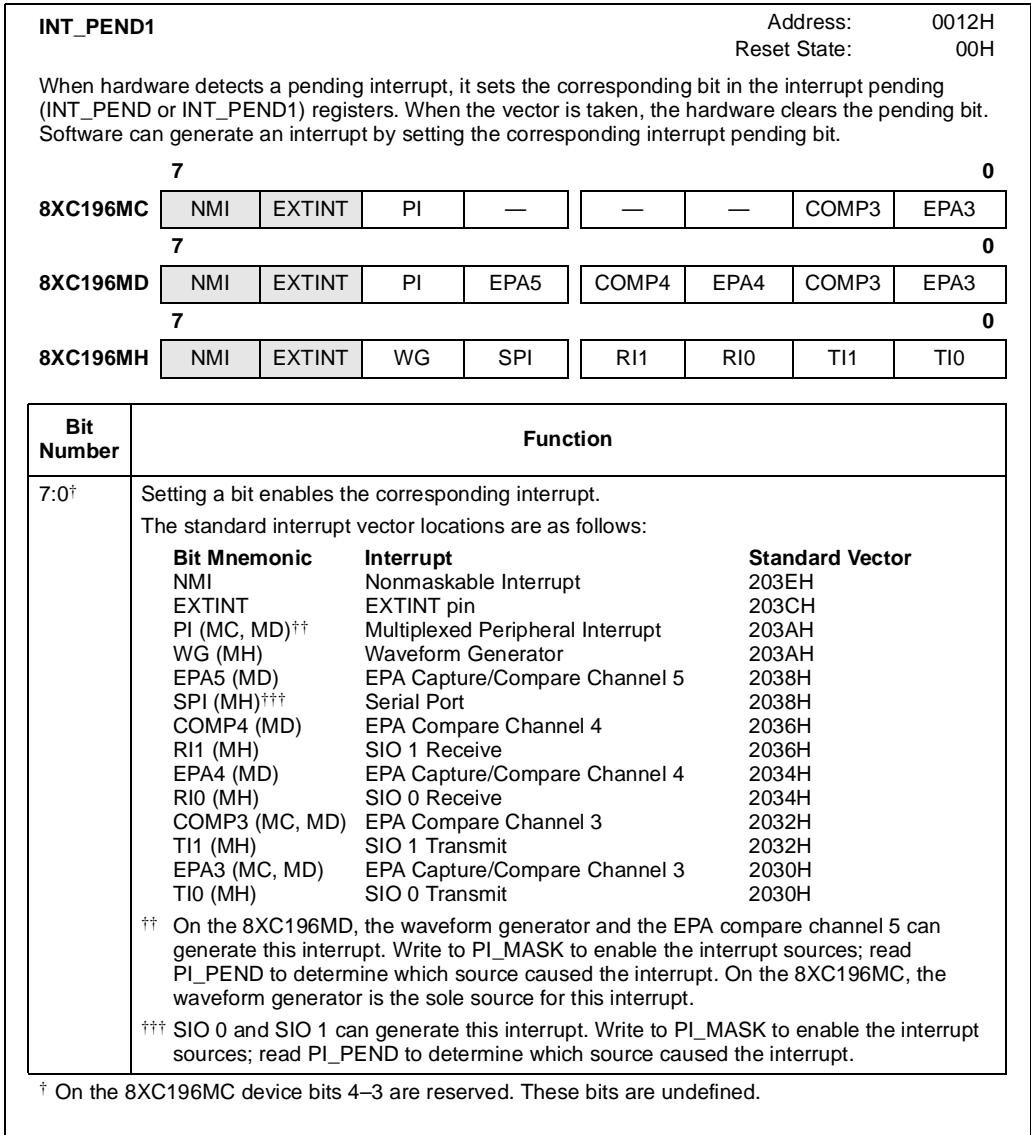
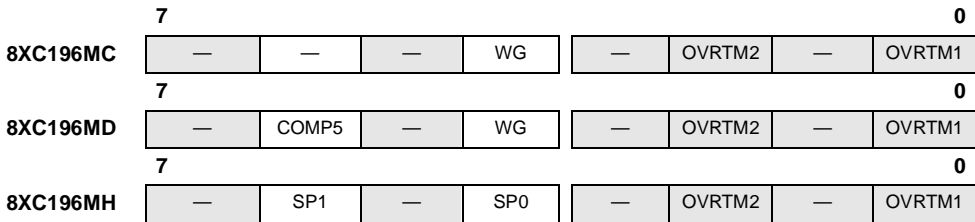


Figure 5-11. Interrupt Pending 1 (INT_PEND1) Register

PI_PEND

Address: 1FBEH
Reset State: AAH

When hardware detects a pending peripheral or timer interrupt, it sets the corresponding bit in the interrupt pending (INT_PEND or INT_PEND1) registers and the peripheral interrupt pending (PI_PEND) register. When the vector is taken, the hardware clears the INT_PEND/INT_PEND1 pending bit. Reading this register clears all the PI_PEND bits. Software can generate an interrupt by setting a PI_PEND bit.



Bit Number	Bit Mnemonic	Function
7, 5, 3, 1	—	Reserved. These bits are undefined.
6	— (MC)	Reserved. This bit is undefined.
	COMP5 (MD)	EPA Compare Channel 5 When set, this bit indicates a pending EPA compare channel 5 interrupt. The EPA compare channel 5 and the waveform generator interrupts are associated with the peripheral interrupt (PI). Setting INT_MASK1.5 enables PI. Setting PI_MASK.6 enables COMP5.
	SP1 (MH)	Serial Port 1 Error When set, this bit indicates a pending serial port 1 error interrupt. The serial port 1 and 0 error interrupts are associated with the serial port interrupt (SPI). Setting INT_MASK1.4 enables SPI. Setting PI_MASK.6 enables SP1.
4	WG (MC, MD)	Waveform Generator When set, this bit indicates a pending waveform generator interrupt. The waveform generator and the EPA compare channel 5 interrupts are associated with the peripheral interrupt (PI). Setting INT_MASK1.5 enables PI. Setting PI_MASK.5 enables WG.
	SP0 (MH)	Serial Port 0 Error When set, this bit indicates a pending serial port 0 error interrupt. The serial port 0 and 1 error interrupts are associated with the serial port interrupt (SPI). Setting INT_MASK1.4 enables SPI. Setting PI_MASK.4 enables SP0.

Figure 5-12. Peripheral Interrupt Pending (PI_PEND) Register

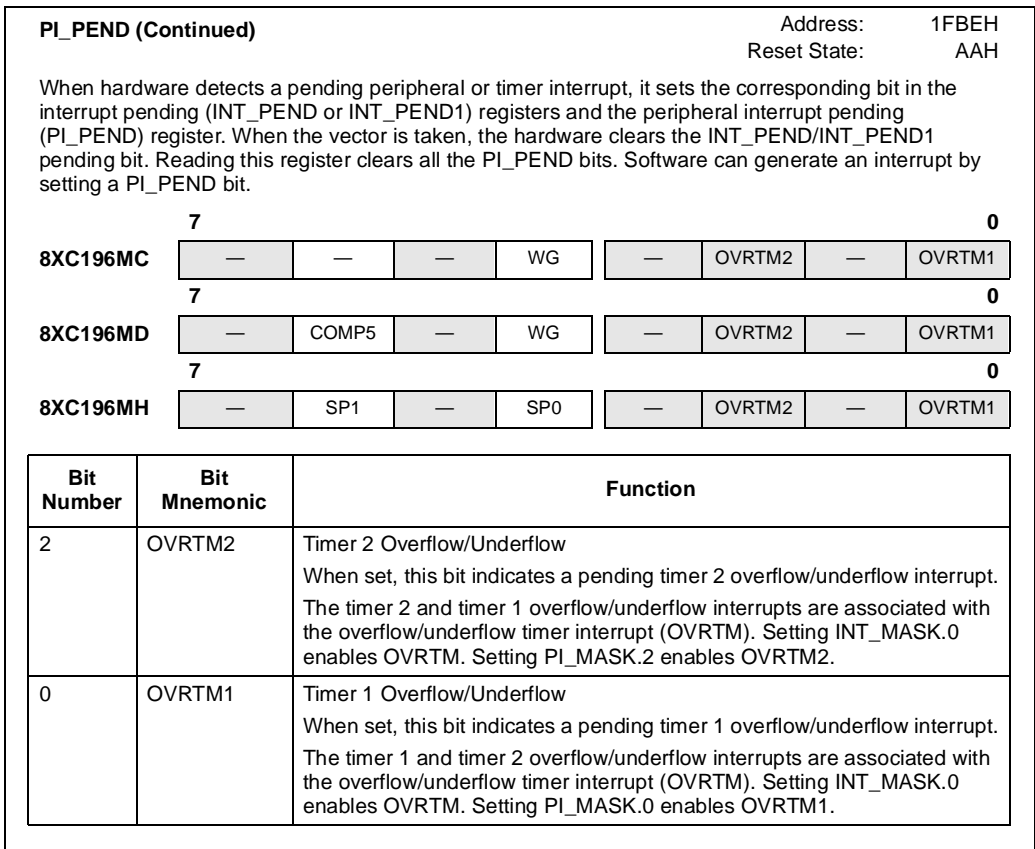


Figure 5-12. Peripheral Interrupt Pending (PI_PEND) Register (Continued)

5.6 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.

The address of the first (lowest) PTSCB byte is stored in the PTS vector table in special-purpose memory (see “Special-purpose Memory” on page 4-3). Figure 5-13 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	A/D Scan Mode	SIO #1 [†]	SIO #2 [†]
	Unused	Unused	Unused	PTSVEC1 (H)	Unused
	Unused	PTSBLOCK	Unused	PTSVEC1 (L)	SAMPTIME
	PTSDST(H)	PTSDST (H)	PTSPTR2 (H)	BAUD (H)	DATA (H)
	PTSDST (L)	PTSDST (L)	PTSPTR2 (L)	BAUD (L)	DATA (L)
	PTSSRC (H)	PTSSRC (H)	PTSPTR1 (H)	EPAREG (H)	PTSCON1
	PTSSRC (L)	PTSSRC (L)	PTSPTR1 (L)	EPAREG (L)	PORTMASK
	PTSCON	PTSCON	PTSCON	PTSCON	PORTREG (H)
PTSVECT	PTSCOUNT	PTSCOUNT	PTSCOUNT	PTSCOUNT	PORTREG (L)

[†] 8XC196MC and MD only.

Figure 5-13. PTS Control Blocks

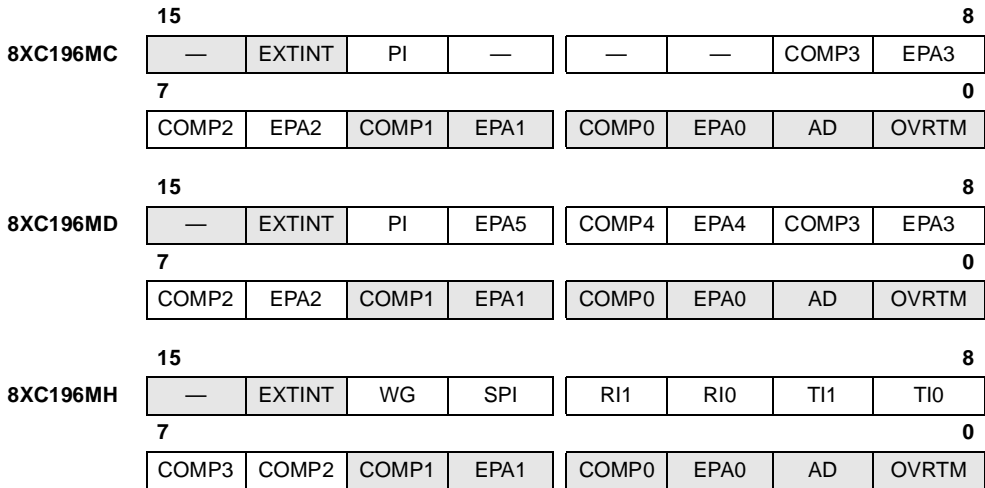
5.6.1 Specifying the PTS Count

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 PTSSSEL bit and sets the PTSSRV bit (Figure 5-6), which requests an end-of-PTS interrupt. The end-of-PTS interrupt service routine should reinitialize the PTSCB, if required, and set the appropriate PTSSSEL bit 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 PTSSEL bit must be set manually to re-enable the PTS channel.



Bit Number	Function																																																				
15	Reserved. This bit is undefined.																																																				
14:0 [†]	<p>A bit is set by hardware to request an end-of-PTS interrupt for the corresponding interrupt through its standard interrupt vector.</p> <p>The PTS interrupt vector locations are as follows:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Bit Mnemonic</th> <th>PTS Vector</th> <th>Bit Mnemonic</th> <th>PTS Vector</th> </tr> </thead> <tbody> <tr> <td>EXTINT</td> <td>205CH</td> <td>T10 (MH)</td> <td>2050H</td> </tr> <tr> <td>PI (MC, MD)^{††}</td> <td>205AH</td> <td>COMP2 (MC,MD)</td> <td>204EH</td> </tr> <tr> <td>WG (MH)</td> <td>205AH</td> <td>COMP3 (MH)</td> <td>204EH</td> </tr> <tr> <td>EPA5 (MD)</td> <td>2058H</td> <td>EPA2 (MC, MD)</td> <td>204CH</td> </tr> <tr> <td>SPI (MH)^{††}</td> <td>2058H</td> <td>COMP2 (MH)</td> <td>204CH</td> </tr> <tr> <td>COMP4 (MD)</td> <td>2056H</td> <td>COMP1</td> <td>204AH</td> </tr> <tr> <td>R11 (MH)</td> <td>2056H</td> <td>EPA1</td> <td>2048H</td> </tr> <tr> <td>EPA4 (MD)</td> <td>2054H</td> <td>COMP0</td> <td>2046H</td> </tr> <tr> <td>R10 (MH)</td> <td>2054H</td> <td>EPA0</td> <td>2044H</td> </tr> <tr> <td>COMP3 (MC, MD)</td> <td>2052H</td> <td>AD</td> <td>2042H</td> </tr> <tr> <td>T11 (MH)</td> <td>2052H</td> <td>OVRTM^{††}</td> <td>2040H</td> </tr> <tr> <td>EPA3 (MC, MD)</td> <td>2050H</td> <td></td> <td></td> </tr> </tbody> </table> <p>^{††} PTS service is not useful for multiplexed interrupts because the PTS cannot readily determine the source of these interrupts.</p>	Bit Mnemonic	PTS Vector	Bit Mnemonic	PTS Vector	EXTINT	205CH	T10 (MH)	2050H	PI (MC, MD) ^{††}	205AH	COMP2 (MC,MD)	204EH	WG (MH)	205AH	COMP3 (MH)	204EH	EPA5 (MD)	2058H	EPA2 (MC, MD)	204CH	SPI (MH) ^{††}	2058H	COMP2 (MH)	204CH	COMP4 (MD)	2056H	COMP1	204AH	R11 (MH)	2056H	EPA1	2048H	EPA4 (MD)	2054H	COMP0	2046H	R10 (MH)	2054H	EPA0	2044H	COMP3 (MC, MD)	2052H	AD	2042H	T11 (MH)	2052H	OVRTM ^{††}	2040H	EPA3 (MC, MD)	2050H		
Bit Mnemonic	PTS Vector	Bit Mnemonic	PTS Vector																																																		
EXTINT	205CH	T10 (MH)	2050H																																																		
PI (MC, MD) ^{††}	205AH	COMP2 (MC,MD)	204EH																																																		
WG (MH)	205AH	COMP3 (MH)	204EH																																																		
EPA5 (MD)	2058H	EPA2 (MC, MD)	204CH																																																		
SPI (MH) ^{††}	2058H	COMP2 (MH)	204CH																																																		
COMP4 (MD)	2056H	COMP1	204AH																																																		
R11 (MH)	2056H	EPA1	2048H																																																		
EPA4 (MD)	2054H	COMP0	2046H																																																		
R10 (MH)	2054H	EPA0	2044H																																																		
COMP3 (MC, MD)	2052H	AD	2042H																																																		
T11 (MH)	2052H	OVRTM ^{††}	2040H																																																		
EPA3 (MC, MD)	2050H																																																				

[†] On the 8XC196MC device bits 10–12 are reserved. These bits are undefined.

Figure 5-14. PTS Service (PTSSRV) Register

5.6.2 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 5-15). 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 5-4 on page 5-12 lists the cycle execution times for each PTS mode.

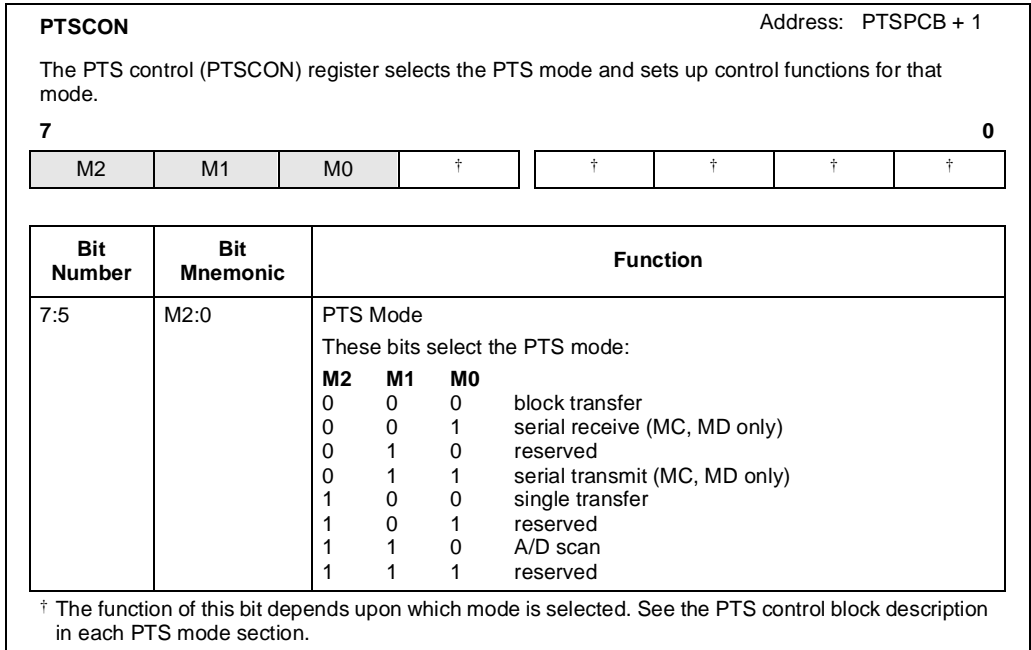


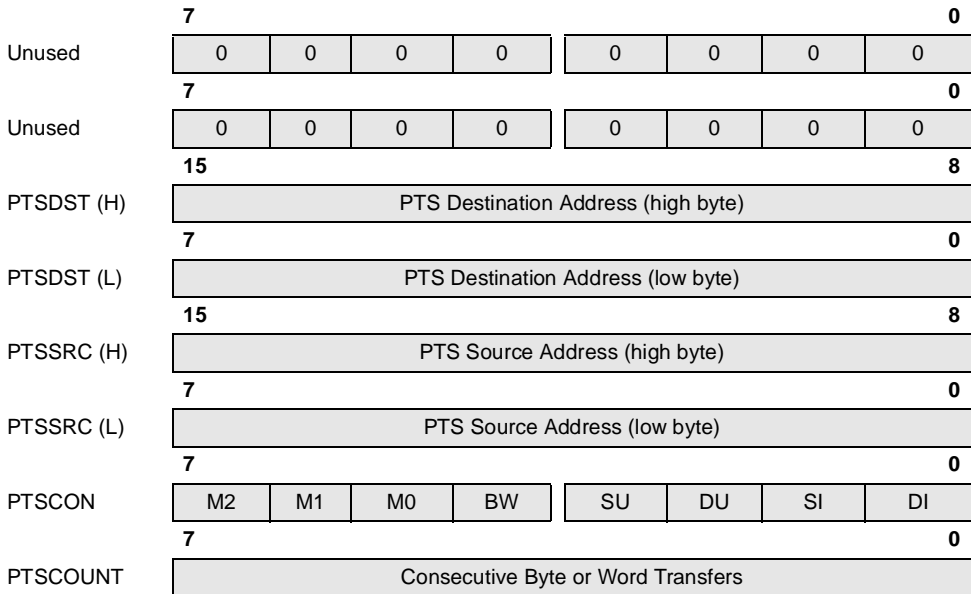
Figure 5-15. PTS Mode Selection Bits (PTSCON Bits 7:5)

5.6.3 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 the EPA to move captured time values from the event-time register to internal RAM for further processing. See AP-483, *Application Examples Using the 8XC196MC/MD Microcontroller*, for application examples with code. Figure 5-16 shows the PTS control block for single transfer mode.

PTS Single Transfer Mode Control Block

In single transfer mode, the PTS control block contains a source and 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; 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; however, it must point to an even address if word transfers are selected.

Figure 5-16. 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 0 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 = do not increment the contents of PTSSRC after each byte or word transfer 1 = increment the contents of PTSSRC after each byte or word transfer
		DI [†]	PTSDST Autoincrement 0 = do not increment the contents of PTSDST after each byte or word transfer 1 = increment the contents of PTSDST 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, need not be equal.

Figure 5-16. PTS Control Block — Single Transfer Mode (Continued)

The PTSCB in Table 5-5 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 5-5. Single Transfer Mode PTSCB

Unused
Unused
PTSDST (H) = 60H
PTSDST (L) = 00H
PTSSRC (H) = 00H
PTSSRC (L) = 20H
PTSCON = 85H (Mode = 100, BW = 0, SI/SU = 0, DI/DU = 1)
PTSCOUNT = 09H

5.6.4 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-483, *Application Examples Using the 8XC196MC/MD Microcontroller*, for application examples with code. Figure 5-17 shows the PTS control block for block transfer modes.

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 5-4 on page 5-12 for execution times of PTS cycles.

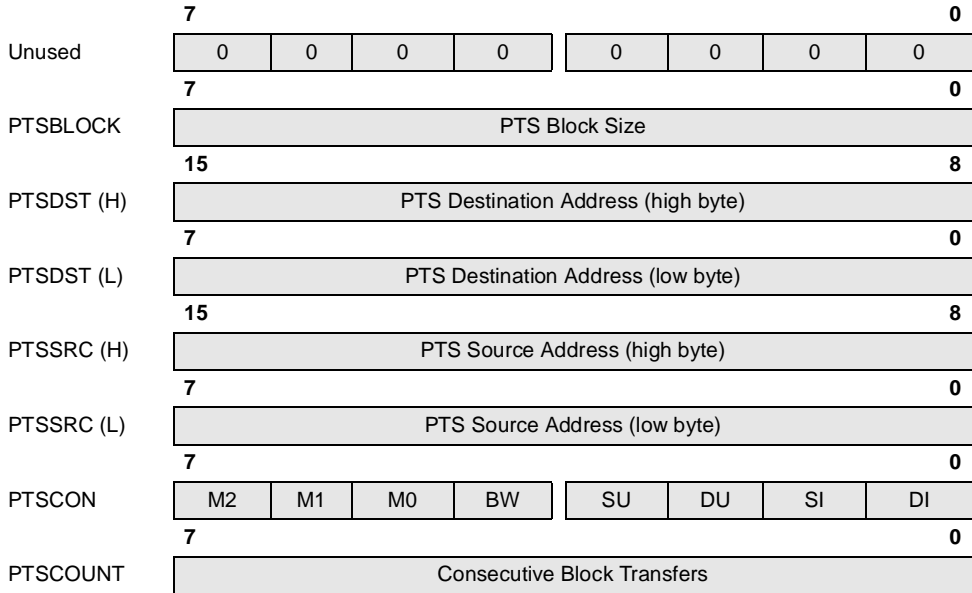
The PTSCB in Table 5-6 sets up three PTS cycles that will transfer five bytes from memory locations 20–24H to 6000–6004H (cycle 1), 6005–6009H (cycle 2), and 600A–600EH (cycle 3). The source and destination are incremented after each byte transfer, but the original source address is reloaded into PTSSRC at the end of each block-transfer cycle. In this routine, the PTS always gets the first byte from location 20H.

Table 5-6. Block Transfer Mode PTSCB

Unused
PTSBLOCK = 05H
PTSDST (H) = 60H
PTSDST (L) = 00H
PTSSRC (H) = 00H
PTSSRC (L) = 20H
PTSCON = 17H (Mode = 000; 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; 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; however, it must point to an even address if word transfers are selected.

Figure 5-17. 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 0 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 = do not increment the contents of PTSSRC after each byte or word transfer 1 = increment the contents of PTSSRC after each byte or word transfer
		DI	PTSDST Autoincrement 0 = do not increment the contents of PTSDST after each byte or word transfer 1 = increment the contents of PTSDST 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.	

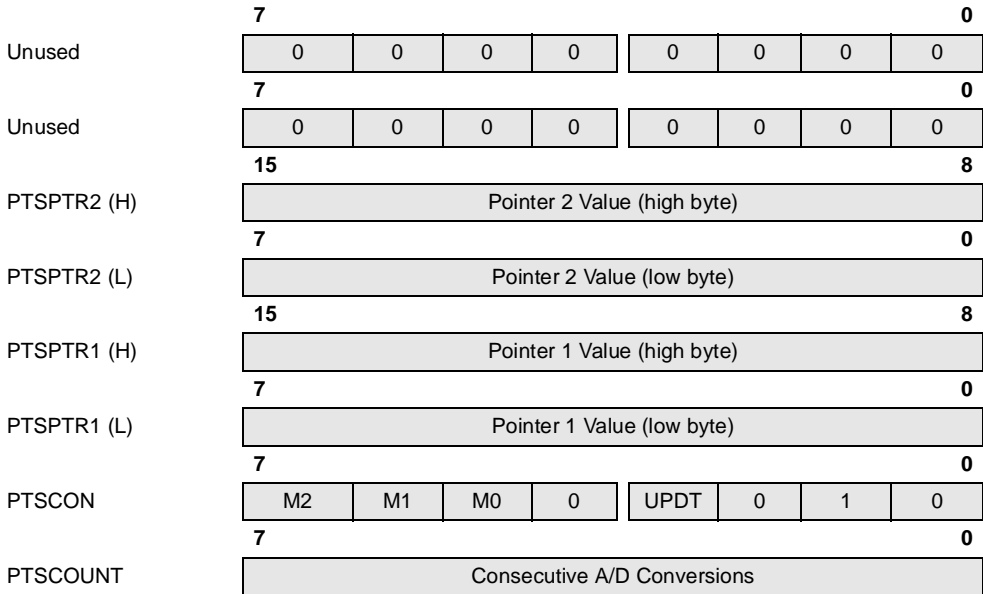
Figure 5-17. PTS Control Block — Block Transfer Mode (Continued)

5.6.5 A/D Scan Mode

In the A/D scan mode, the PTS causes the A/D converter to perform multiple conversions on one or more channels and then stores the results in a table in memory. Figure 5-19 shows the PTS control block for A/D scan mode.

PTS A/D Scan Mode Control Block

In A/D scan mode, the PTS causes the A/D converter to perform multiple conversions on one or more channels and then stores the results. The control block contains pointers to both the AD_RESULT register (PTSPTR1) and a table of A/D conversion commands and results (PTSPTR2), a control register (PTSCON), and an A/D conversion count (PTSCOUNT).



Register	Location	Function	
PTSPTR2	PTSCB + 4	Pointer 2 Value This register contains the address of the A/D result register (AD_RESULT).	
PTSPTR1	PTSCB + 2	Pointer 1 Value This register contains the address of the table of A/D conversion commands and results.	
PTSCON	PTSCB + 1	PTS Control Bits	
		M2:0	PTS Mode These bits select the PTS mode. M2 M1 M0 1 1 0 A/D Scan Mode
		UPDT	Update 0 = reload original PTSPTR1 value after each A/D scan 1 = retain current PTSPTR1 value after each A/D scan

Figure 5-18. PTS Control Block – A/D Scan Mode

PTS A/D Scan Mode Control Block (Continued)		
PTSCOUNT	PTSCB + 0	Consecutive A/D Conversions Defines the number of A/D conversions that will be completed during the A/D scan routine. Each cycle consists of the PTS transferring the A/D conversion results into the command/data table, and then loading a new command into the AD_COMMAND register. Maximum number is 255.

Figure 5-18. PTS Control Block – A/D Scan Mode (Continued)

To use the A/D scan mode, you must first set up a command/data table in memory (Table 5-7). The command/data table contains A/D commands that are interleaved with blank memory locations. The PTS stores the conversion results in these blank locations. Only the amount of available memory limits the table size; it can reside in internal or external RAM.

Table 5-7. A/D Scan Mode Command/Data Table

Address	Contents	
XXXX + AH	A/D Result 2	
XXXX + 8H	Unused	A/D Command 3 [†]
XXXX + 6H	A/D Result 1	
XXXX + 4H	Unused	A/D Command 2
XXXX + 2H	A/D Result 0 ^{††}	
XXXX	Unused	A/D Command 1

[†] Write 0000H to prevent a new conversion at the end of the routine.

^{††} Result of the A/D conversion that initiated the PTS routine.

To initiate A/D scan mode, enable the A/D conversion complete interrupt and assign it to the PTS. Software must initiate the first conversion. When the A/D finishes the first conversion and generates an A/D conversion complete interrupt, the interrupt vectors to the PTSCB and initiates the A/D scan routine. The PTS stores the conversion results, loads a new command into AD_COMMAND, and then decrements the number in PTSCOUNT. As each additional conversion complete interrupt occurs, the PTS repeats the A/D scan cycle; it stores the conversion results, loads the next conversion command into the AD_COMMAND register, and decrements PTSCOUNT. The routine continues until PTSCOUNT decrements to zero. When this occurs, hardware clears the enable bit in the PTSSSEL register, which disables PTS service, and sets the PTSSRV bit, which requests an end-of-PTS interrupt.

The interrupt service routine could process the conversion results and then re-enable PTS service for the A/D conversion complete interrupt. Because the lower six bits of the AD_RESULT register contain status information, the end-of-PTS interrupt service routine could shift the results data to the right six times to leave only the conversion results in the memory locations. See AP-483, *Application Examples Using the 8XC196MC/MD Microcontroller*, for application examples with code.

5.6.5.1 A/D Scan Mode Cycles

Software must start the first A/D conversion. After the A/D conversion complete interrupt initiates the PTS routine, the following actions occur.

1. The PTS reads the first command (from address XXXX), stores it in a temporary location, and increments the PTSPTR1 register twice. PTSPTR1 now points to the first blank location in the command/data table (address XXXX + 2).
2. The PTS reads the AD_RESULT register, stores the results of the first conversion into location XXXX + 2 in the command/data table, and increments the PTSPTR1 register twice. PTSPTR1 now points to XXXX + 4.
3. The PTS loads the command from the temporary location into the AD_COMMAND register. This completes the first A/D scan cycle and initiates the next A/D conversion.
4. If UPDT (PTSCON.3) is clear, the original address is reloaded into the PTSPTR1 register. The next cycle uses the same command and overwrites previous data. If UPDT is set, the updated address remains in PTSPTR1 and the next cycle uses a new command and stores the conversion results at the new address.
5. PTSCOUNT is decremented and the CPU returns to regular program execution. When the next A/D conversion complete interrupt occurs, the cycle repeats. When PTSCOUNT reaches zero, hardware clears the corresponding PTSSSEL bit and sets the PTSSRV bit, which requests the end-of-PTS interrupt.

5.6.5.2 A/D Scan Mode Example 1

The command/data table shown in Table 5-8 sets up a series of A/D conversions, beginning with channel 7 and ending with channel 4. Each table entry is a word (two bytes). Table 5-9 shows the corresponding PTSCB.

Software starts a conversion on channel 7. Upon completion of the conversion, the A/D conversion complete interrupt initiates the A/D scan mode routine. Step 1 stores the channel 6 command in a temporary location and increments PTSPTR1 to 3002H. Step 2 stores the result of the channel 7 conversion in location 3002H and increments PTSPTR1 to 3004H. Step 3 loads the channel 6 command from the temporary location into the AD_COMMAND register to start the next con-

version. Step 4 updates PTSPTR1 (PTSPTR1 now points to 3004H) and step 5 decrements PTSCOUNT to 3. The next cycle begins by storing the channel 5 command in the temporary location. During the last cycle (PTSCOUNT = 1), the dummy command is loaded into the AD_COMMAND register and no conversion is performed. PTSCOUNT is decremented to zero and the end-of-PTS interrupt is requested.

Table 5-8. Command/Data Table (Example 1)

Address	Contents	
300EH	AD_RESULT for ACH4	
300CH	Unused	0000H (Dummy command)
300AH	AD_RESULT for ACH5	
3008H	Unused	AD_COMMAND for ACH4
3006H	AD_RESULT for ACH6	
3004H	Unused	AD_COMMAND for ACH5
3002H	AD_RESULT for ACH7	
3000H	Unused	AD_COMMAND for ACH6

Table 5-9. A/D Scan Mode PTSCB (Example 1)

Unused
Unused
PTSPTR2 (H) = 1FH
PTSPTR2 (L) = AAH
PTSPTR1 (H) = 30H
PTSPTR1 (L) = 00H
PTSCON = CBH (Mode = 110, UPDT = 1)
PTSCOUNT = 04H

5.6.5.3 A/D Scan Mode Example 2

Table 5-11 sets up a series of ten PTS cycles, each of which reads a single A/D channel and stores the result in a single location (3002H). The UPDT bit (PTSCON.3) is cleared so that original contents of PTSPTR1 are restored after the cycle. The command/data table is shown in Table 5-10.

Table 5-10. Command/Data Table (Example 2)

Address	Contents	
3002H	AD_RESULT for ACHx	
3000H	Unused	AD_COMMAND for ACHx

Table 5-11. A/D Scan Mode PTSCB (Example 2)

Unused
Unused
PTSPTR2 (H) = 1FH
PTSPTR2 (L) = AAH
PTSPTR1 (H) = 30H
PTSPTR1 (L) = 00H
PTSCON = C3H (Mode = 110, UPDT = 0)
PTSCOUNT = 0AH

Software starts a conversion on channel *x*. When the conversion is finished and the A/D conversion complete interrupt is generated, the A/D scan mode routine begins. The PTS reads the command in location 3000H and stores it in a temporary location. Then it increments PTSPTR1 twice and stores the value of the AD_RESULT register in location 3002H. The final step is to copy the conversion command from the temporary location to the AD_COMMAND register. The CPU could process or move the conversion results data from the table before the next conversion completes and a new PTS cycle begins. When the next cycle begins, PTSPTR1 again points to 3000H and the repeats the events of the first cycle. The value of the AD_RESULT register is written to location 3002H and the command at location 3000H is re-executed.

5.6.6 Serial I/O Modes

The 8XC196MH has a two-channel serial I/O port. The 8XC196MC and MD have no serial I/O ports, but the serial I/O modes of the PTS provide a software serial I/O channel for both synchronous and asynchronous transfers and receptions. There are four basic modes of operation: synchronous transmit, synchronous receive, asynchronous transmit, and asynchronous receive. A standard I/O port signal is configured to function as either the transmit data (TXD) or receive data (RXD) signal and an EPA channel sets the baud rate. You may configure any port 2 signal (MC and MD) or P7.3:0 (MD) as TXD or RXD. In the synchronous modes, an EPA channel can either generate the serial clock (SCK) signal or input an external clock signal. Up to 16 bits may be

transmitted or received including the parity and stop bits in the asynchronous modes. The serial I/O modes require two PTS control blocks to configure all options (see Figures 5-19 and 5-20). These blocks need not be contiguous, but they must each be located in register RAM on a quad-word boundary. See AP-483, *Application Examples Using the 8XC196MC/MD Microcontroller*, for application examples with code.

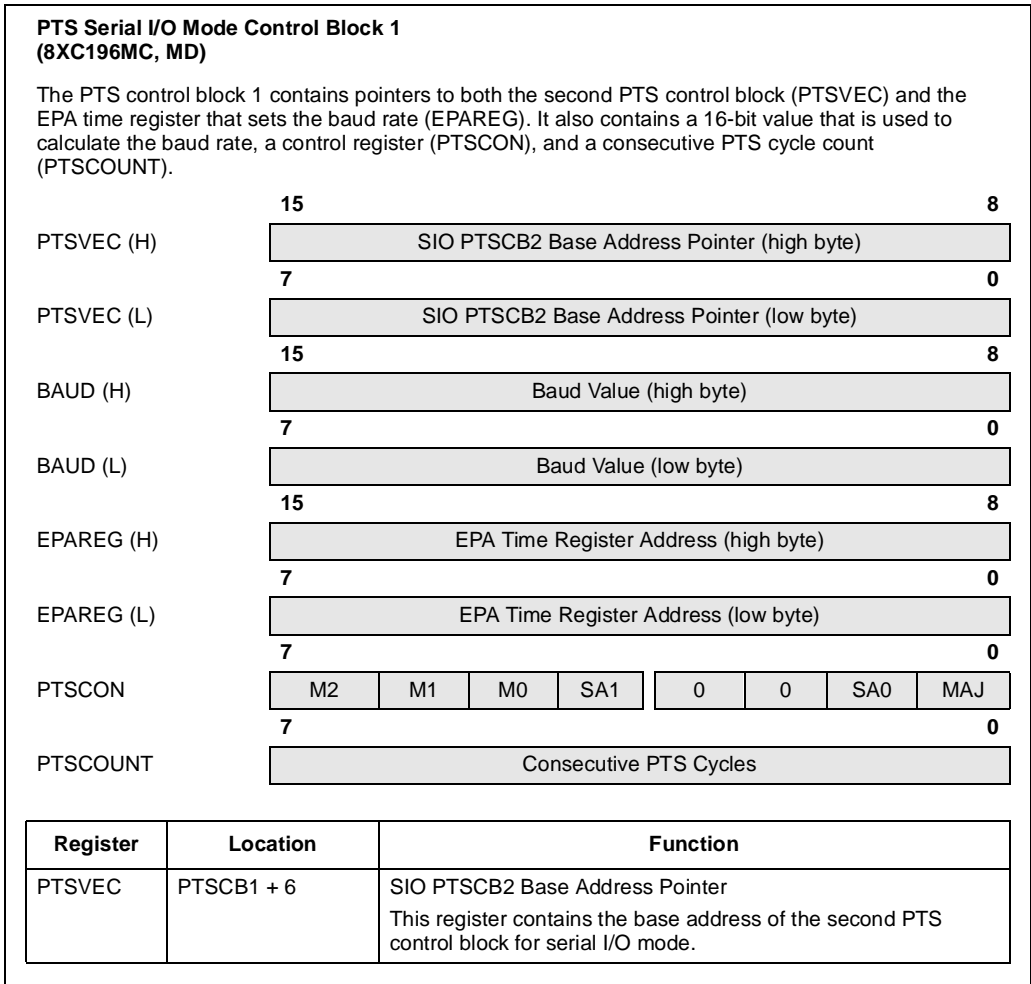


Figure 5-19. PTS Control Block 1 – Serial I/O Mode

PTS Serial I/O Mode Control Block 1 (Continued) (8XC196MC, MD)																				
Register	Location	Function																		
BAUD	PTSCB1 + 4	<p>Baud Value</p> <p>This register contains the 16-bit value that the PTS uses to generate the desired baud rate. Use the following formula to calculate the value to load into the BAUD register.</p> $\text{Baud_value} = \frac{F_{\text{XTAL1}}}{\text{Multiplier} (\text{Baud_rate} \times \text{EPA_prescale})}$ <p>where:</p> <ul style="list-style-type: none"> Baud_value is a 16-bit integer that is loaded into the BAUD register F_{XTAL1} is the input frequency on XTAL1, in MHz Multiplier is the number 4 in asynchronous modes and the number 8 in synchronous modes Baud_rate is the desired baud rate, in bits per second EPA_prescale is the EPA timer prescale number, 1–64 																		
EPAREG	PTSCB1 + 2	<p>EPA Time Register Address</p> <p>This register contains the 16-bit address of the EPA_x_TIME or COMP_x_TIME register.</p>																		
PTSCON	PTSCB1 + 2	<p>PTS Control Bits</p> <table border="1"> <tr> <td rowspan="3">M2:0</td> <td colspan="3">PTS Mode</td> <td></td> </tr> <tr> <td>M2</td> <td>M1</td> <td>M0</td> <td></td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> <td>SIO Receive Mode</td> </tr> <tr> <td></td> <td>0</td> <td>1</td> <td>1</td> <td>SIO Transmit Mode</td> </tr> </table>	M2:0	PTS Mode				M2	M1	M0		0	0	1	SIO Receive Mode		0	1	1	SIO Transmit Mode
		M2:0		PTS Mode																
				M2	M1	M0														
0	0		1	SIO Receive Mode																
	0	1	1	SIO Transmit Mode																
SA1:0	Asynchronous, Synchronous Mode Select	<p>SA1 SA0[†]</p> <p>0 0 enables the asynchronous serial I/O modes</p> <p>1 1 enables the synchronous serial I/O modes</p> <p>[†] Always write the same value to both bits.</p>																		
MAJ	Majority Sampling	<p>0 = disable majority sampling in asynchronous receive mode; always clear in all other modes</p> <p>1 = enable majority sampling in asynchronous receive mode</p>																		

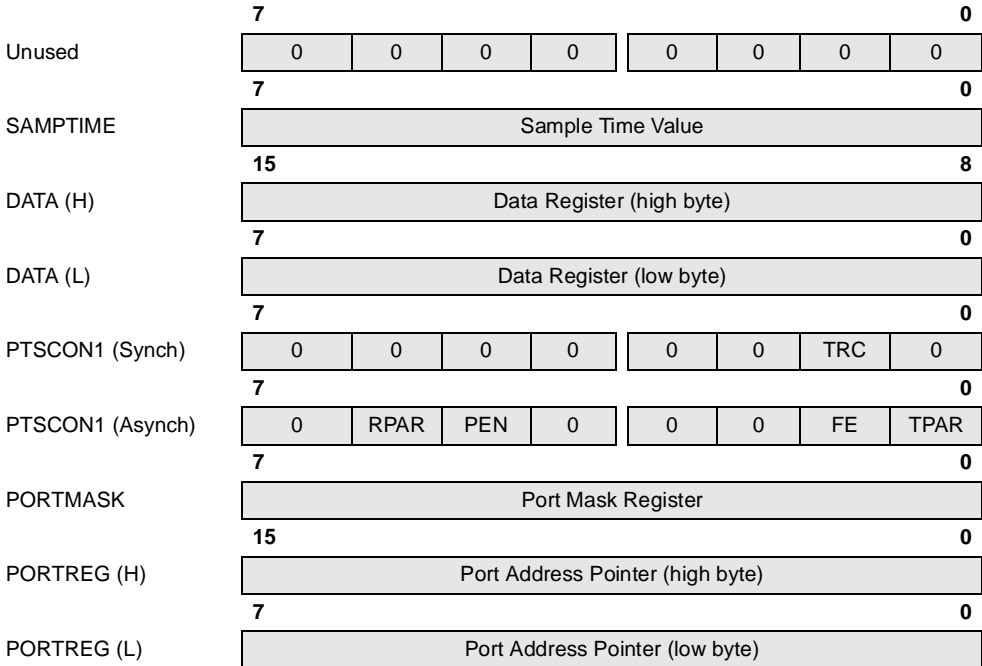
Figure 5-19. PTS Control Block 1 – Serial I/O Mode (Continued)

PTS Serial I/O Mode Control Block 1 (Continued) (8XC196MC, MD)		
Register	Location	Function
PTSCOUNT	PTSCB1 + 0	<p>Consecutive PTS Cycles</p> <p>Defines the number of bits to be transmitted or received, including parity and stop bits, but not the start bit. For asynchronous modes, program a number that is between 1–16. For synchronous modes, program a number that is twice the number of bits to be transmitted or received (2–32). PTSCOUNT is decremented at the end of each PTS cycle. When it reaches zero, hardware clears the corresponding PTSSEL bit and sets the PTSSRV bit, which requests the end-of-PTS interrupt.</p>

Figure 5-19. PTS Control Block 1 – Serial I/O Mode (Continued)

**PTS Serial I/O Mode Control Block 2
(8XC196MC, MD)**

The PTS control block 2 contains pointers to both the port register (PORTREG) and the data register (DATA). It also contains a 16-bit value that is used to calculate the sample time for asynchronous receptions when majority sampling is selected (SAMPTIME), a control register (PTSCON1), and a 16-bit value that is used to select the port signal that functions as the TXD or RXD signal (PORTMASK).



Register	Location	Function
SAMPTIME	PTSCB2 + 6	<p>Sample Time Value</p> <p>This register controls the time between samples during asynchronous receive mode when majority sampling is selected. Use the following formula to calculate the value to load into the SAMPTIME register.</p> $\text{Sample_time} = \frac{T_{\text{SAM}} \times F_{\text{XTAL1}}}{2} - 9$ <p>where:</p> <ul style="list-style-type: none"> Sample_time is an integer, 1–31, that is loaded into the SAMPTIME register T_{SAM} is the desired time between samples, in μs F_{XTAL1} is the input frequency on XTAL1, in MHz

Figure 5-20. PTS Control Block 2 – Serial I/O Mode

PTS Serial I/O Mode Control Block 2 (Continued) (8XC196MC, MD)			
Register	Location	Function	
DATA	PTSCB2 + 4	<p>Data Register</p> <p>This 16-bit register holds the data to be transmitted or the data that has been received. During transmit mode, the least-significant bit (bit 0) is transmitted first. Data shifts to the right with each successive transmission. During receive mode, the first bit is loaded into the most-significant bit (bit 15). Data shifts to the right with each successive reception.</p>	
PTSCON1	PTSCB2 + 3	PTS Control Bits	
		Synchronous Mode	
		TRC	<p>Transmit/Receive Control</p> <p>0 = transmit or receive data during even numbered PTS cycles 1 = transmit or receive data during odd numbered PTS cycles</p> <p>Initialize this bit at the start of every transmission or reception.</p>
		Asynchronous Mode	
		RPAR	<p>Receive Parity Control and Status</p> <p>Initialize this bit as indicated before beginning a reception.</p> <p>0 = TPAR bit is set to select even parity 1 = TPAR bit is cleared to select odd parity</p> <p>If this bit is set at the end of a reception, a parity error has occurred.</p>
		PEN	<p>Parity Enable</p> <p>0 = disable parity 1 = enable parity</p>
		FE	<p>Framing Error Flag</p> <p>0 = stop bit was 1 1 = stop bit was 0</p> <p>Clear this bit at the start of every reception.</p>
TPAR	<p>Transmit Parity Control</p> <p>0 = even parity 1 = odd parity</p>		

Figure 5-20. PTS Control Block 2 – Serial I/O Mode (Continued)

PTS Serial I/O Mode Control Block 2 (Continued) (8XC196MC, MD)		
Register	Location	Function
PORTMASK	PTSCB2 + 2	Port Mask Register Select the port signal that will function as the transmit data (TXD) or receive data (RXD) signal by setting the corresponding bit. Clear all other bits to mask those signals.
PORTREG	PTSCB2 + 0	Port Address Pointer This 16-bit register contains the address of the port that will be used to transmit or receive data.

Figure 5-20. PTS Control Block 2 – Serial I/O Mode (Continued)

5.6.6.1 Synchronous SIO Transmit Mode Example

In synchronous serial I/O (SSIO) transmit mode, an EPA channel controls the transmission baud rate by generating or capturing a serial clock signal (SCK). To generate the SCK signal, configure the EPA channel in compare mode and set the output-pin toggle option. Whenever a match occurs between the EPA event-time register and a timer register, the EPA channel toggles SCK and generates an interrupt. If an external source will provide the SCK signal, configure the EPA channel in capture mode with capture on either edge set. In this case, the EPA channel generates an interrupt whenever the SCK input toggles. On every other EPA interrupt, the PTS shifts a data bit out onto a port pin that is configured to function as the Transmit Data signal (TXD). PTSCON1 (Figure 5-19 on page 5-38) controls whether the transmission occurs on even or odd PTS cycles. Because transmissions occur only on a rising or falling clock edge, two PTS cycles occur for every one data bit transmission (Figure 5-21). It takes 16 PTS cycles to transmit eight data bits. In SSIO transmit mode, only data bits can be transmitted; parity and stop bits are not included.

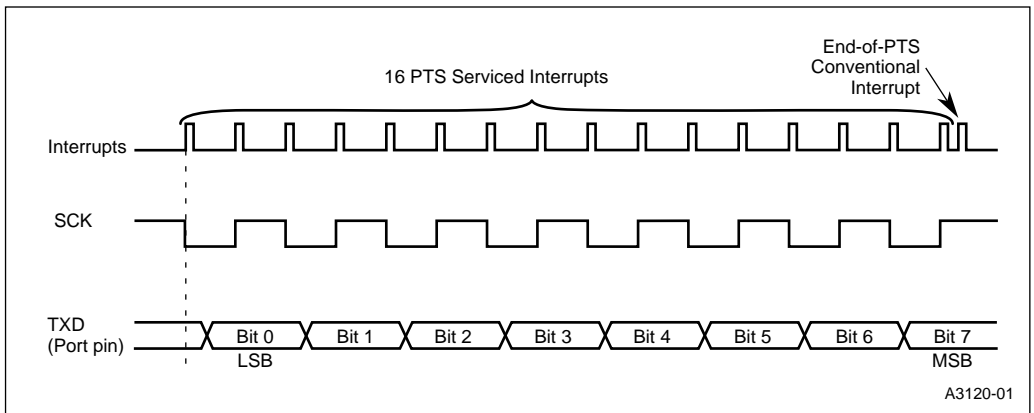


Figure 5-21. Synchronous SIO Transmit Mode Timing

If the SCK signal is generated by the EPA channel, the first PTS cycle must be started manually.

- Initialize the TXD port pin and the SCK signal to the system-required logic level before starting a transmission.
- Add the contents of the timer register to the Baud_value (Figure 5-19 on page 5-38) and store the result into the EPA time register. This sets up the timing for the first interrupt and causes the first bit transmission to occur at the proper baud rate.

The following example uses EPA0 to control the baud rate and output the SCK signal and P2.2 to output the data (TXD). It sets up a synchronous serial I/O PTS routine that transmits 16 bytes with eight data bits at 9600 baud. This example uses several user-defined registers. T_COUNT defines the number of bytes to transfer and TXDDONE is a flag that is set when all bytes are transferred.

1. Disable the interrupts and the PTS.
 - Use the DI instruction to disable all standard interrupts and the DPTS instruction to disable the PTS.
2. Set-up the stack pointer.
3. Reset all interrupt mask registers.
 - Clear INT_MASK, INT_MASK1 and PI_MASK.
4. Initialize P2.0 to function as the EPA0 output (SCK) and P2.2 to function as TXD.
 - Clear P2_DIR (selects output).
 - Set P2_MODE.0 (selects special function).
 - Clear P2_MODE.2 (selects LSIO function).
 - Set P2_REG bits 0 and 2 (initializes TXD and SCK outputs to “1”).
5. Initialize and enable the timer. Select up counting, internal clock, and prescaler disabled.
 - Set T1CONTROL bits 6 and 7 (Figure 11-8 on page 11-16).
6. Initialize the PTSCB as shown in Table 5-13.

Table 5-13. SSIO Transmit Mode PTSCBs

PTSCB1	PTSCB2
PTSVEC (H) = pointer to PTSCB2	Unused
PTSVEC (L) = pointer to PTSCB2	SAMPTIME = unused
BAUD (H) = 00H (9600 baud at 16 MHz)	DATA (H) = unused
BAUD (L) = D0H (9600 baud at 16 MHz)	DATA (L) = <i>nnH</i> (8 data bits)
EPAREG (H) = 1FH (EPA0_TIME)	PTSCON1 = 02H (transmit data on odd PTS cycles)
EPAREG (L) = 42H (EPA0_TIME)	PORTMASK = 04H (P2.2 = TXD)
PTSCON = 72H (SSIO transmit mode)	PORTREG (H) = 1FH (P2_REG)
PTSCOUNT = 10H (8 data bits x 2)	PORTREG (L) = D4H (P2_REG)

7. Enable EPA0 interrupt.
 - Set INT_MASK.2.
8. Load the number of bytes to transmit into the user_defined transmit count register (T_COUNT) and clear the user-defined transfer-done flag (TXDDONE).
 - LD T_COUNT, #16
 - CLRB TXDDONE
9. Select PTS service for EPA0.
 - Set PTSSEL.2.
10. Set-up EPA0 as a compare-only channel.
 - Set EPA0_CON.6 (Figure 11-10 on page 11-19).
11. Start the operation of the EPA0 channel by writing the time of the first interrupt to EPA0_TIME. To set-up the correct value, add the baud_value (0D0H) to the current TIMER1 value and store the result in EPA0_TIME. The baud_value determines the time to the first PTS interrupt and the first transition on SCK. The PTS transmits the first data bit on first transition of SCK in this example. The baud_value of 0D0H selects a baud rate of 9600.
12. Enable the PTS and conventional interrupts.
 - Use the EI instruction to enable all standard interrupts and the EPTS instruction to enable the PTS.
13. The transmission will begin. Data is shifted out with the least-significant (rightmost) bit first. Each time a timer match occurs between EPA0_TIME and TIMER1, the EPA0 channel generates an interrupt and toggles the SCK signal. The PTS outputs the next bit of data on the pin configured as TXD on odd PTS cycles. When PTSCOUNT decrements to zero, the PTS calls the end-of-PTS interrupt (Figure 5-22). The interrupt service routine should disable the EPA channel because the final PTS cycle loads the next SCK toggle

time into the event-time register. If this toggle occurs, the clock polarity will change because of the odd number of toggles and erroneous data may be output. The interrupt service routine should also load the next data byte, reload the PTSCOUNT and PTSCON1 registers, select PTS service for EPA0, reload both the EPA0_CONTROL and EPA0_TIME registers.

- To determine when all bytes have been transmitted, create a loop routine to check the status of the TXDDONE flag.

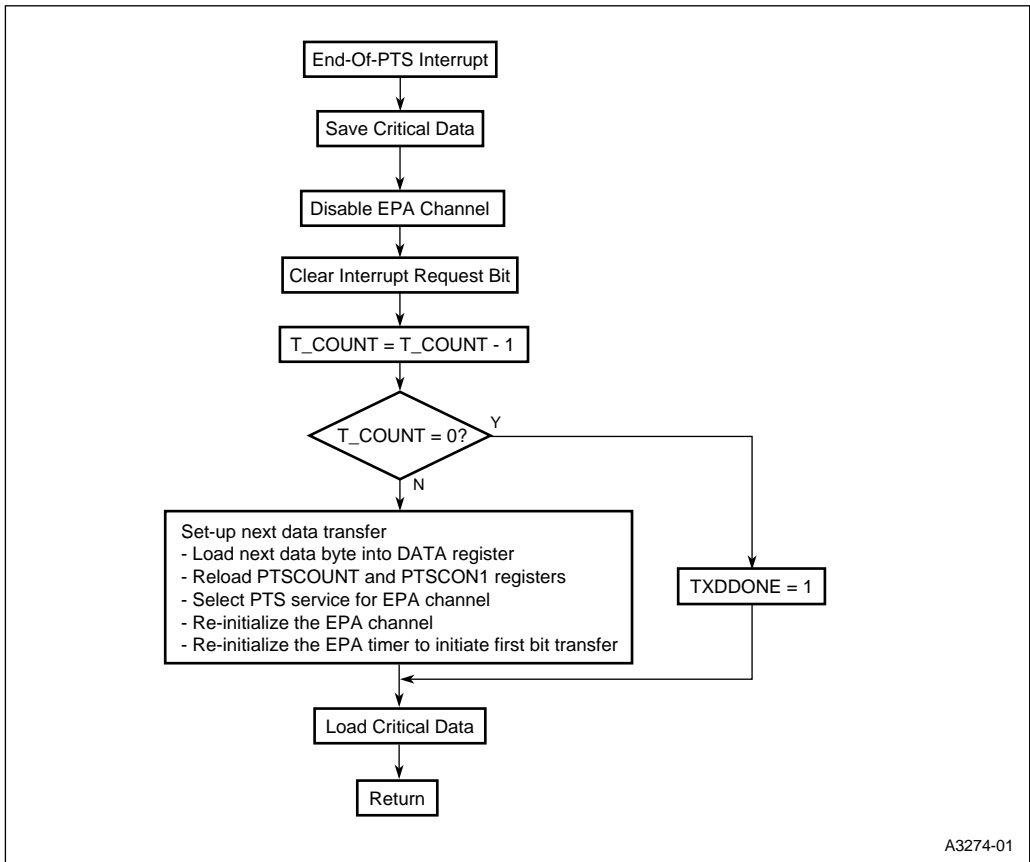


Figure 5-22. Synchronous SIO Transmit Mode — End-of-PTS Interrupt Routine Flowchart

5.6.6.2 Synchronous SIO Receive Mode Example

In synchronous serial I/O (SSIO) receive mode, an EPA channel controls the reception baud rate by generating or capturing a serial clock signal (SCK). To generate the SCK signal, configure the EPA channel in compare mode and set the output-pin toggle option. Whenever a match occurs between the EPA event-time register and a timer register, the EPA channel toggles SCK and generates an interrupt. If an external source will provide the SCK signal, configure the EPA channel in capture mode with capture on either edge set. In this case, the EPA channel generates an interrupt whenever the SCK input toggles. On every other EPA interrupt, the PTS inputs a data bit from a port pin that is configured to function as the Receive Data signal (RXD). PTSCON1 (Figure 5-19 on page 5-38) controls whether the reception occurs on even or odd PTS cycles. Because receptions occur only on a rising or falling clock edge, two PTS cycles occur for every one data bit reception (Figure 5-23). It takes 16 PTS cycles to receive eight data bits. SSIO receptions do not include parity or stop bits.

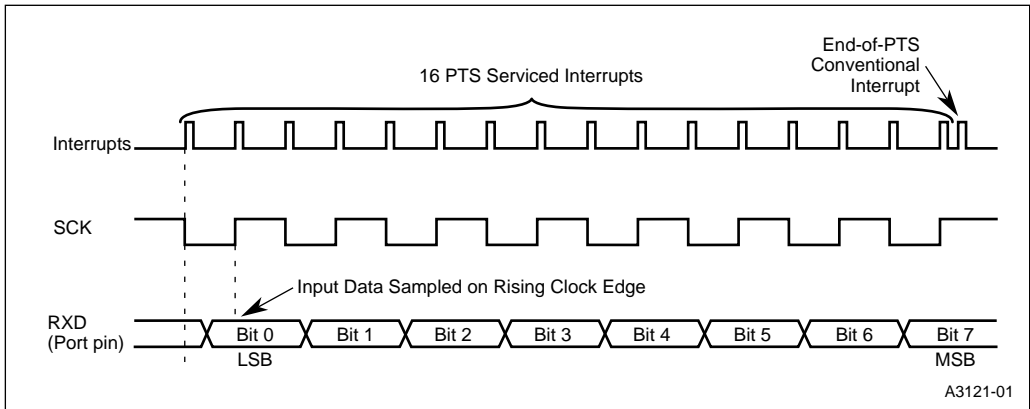


Figure 5-23. Synchronous SIO Receive Timing

If the SCK signal is generated by the EPA channel, the first PTS cycle must be started manually.

- Initialize the RXD port pin and the SCK signal to the system-required logic level before starting a reception.
- Add the contents of the timer register to the Baud_value (Figure 5-19 on page 5-38) and store the result into the EPA time register. This sets up the timing for the first interrupt and causes the first interrupt to occur at the proper baud rate.

The following example uses EPA0 to capture the SCK signal and P2.3 to receive the data (RXD). It sets up a synchronous serial I/O PTS routine that receives 16 bytes with eight data bits. Because this example uses an external serial clock input, the TIMER1 and BAUD registers are not used. The external clock source controls the baud rate. This example uses several user-defined registers. R_COUNT defines the number of bytes to receive and RXDDONE is a flag that is set when all bytes are received.

1. Disable the interrupts and the PTS.
 - Use the DI instruction to disable all standard interrupts and the DPTS instruction to disable the PTS.
2. Set-up the stack pointer.
3. Reset all interrupt mask registers.
 - Clear INT_MASK, INT_MASK1 and PI_MASK.
4. Initialize P2.0 to function as the EPA0 input (SCK) and P2.3 to function as RXD.
 - Set P2_DIR bits 0 and 3 (selects input).
 - Set P2_MODE.0 (selects special function).
 - Clear P2_MODE.3 (selects LSIO function).
 - Set P2_REG bits 0 and 3 (initializes SCK and RXD input to “1”).
5. Initialize the PTSCB as shown in Table 5-14.

Table 5-14. SSIO Receive Mode PTSCBs

PTSCB1	PTSCB2
PTSVEC (H) = pointer to PTSCB2	Unused
PTSVEC (L) = pointer to PTSCB2	SAMPTIME = unused
BAUD (H) = unused	DATA (H) = unused
BAUD (L) = unused	DATA (L) = 00H (clear register to receive data)
EPAREG (H) = 1FH (EPA0_TIME)	PTSCON1 = 00H (receive data on even PTS cycles)
EPAREG (L) = 42H (EPA0_TIME)	PORTMASK = 08H (P2.3 = RXD)
PTSCON = 32H (SSIO receive mode)	PORTREG (H) = 1FH (P2_REG)
PTSCOUNT = 10H (8 data bits x 2)	PORTREG (L) = D4H (P2_REG)

6. Enable EPA0 interrupt.
 - Set INT_MASK.2.
7. Load the number of bytes to transmit into the user_defined transmit count register (R_COUNT) and clear the user-defined reception-done flag (RXDDONE).
 - LD R_COUNT, #16
 - CLRB RXDDONE

8. Select PTS service for EPA0.
 - Set PTSSEL.2.
9. Set-up EPA0 to capture on both rising and falling edges.
 - Set EPA0_CON bits 4 and 5 (Figure 11-10 on page 11-19).
10. Enable the PTS and conventional interrupts.
 - Use the EI instruction to enable all standard interrupts and the EPTS instruction to enable the PTS.
11. Toggle the SCK input to start the reception. Data is shifted into the most-significant (leftmost) bit first and shifts right with each successive bit received. The EPA generates an interrupt each time that the SCK input toggles. The PTS receives the next bit of data on the pin configured as RXD on even PTS cycles. When PTSCOUNT decrements to zero, the PTS calls the end-of-PTS interrupt (Figure 5-24). The interrupt service routine should disable the EPA channel, clear the DATA register, reload the PTSCOUNT and PTSCON1 registers, reload EPA0_CON, and select PTS service for EPA0. If the EPA were generating the SCK signal, the end-of-PTS interrupt service routine would also have to reload the EPA0_TIME register.
12. To determine when all bytes have been transmitted, create a loop routine to check the status of the RXDDONE flag.

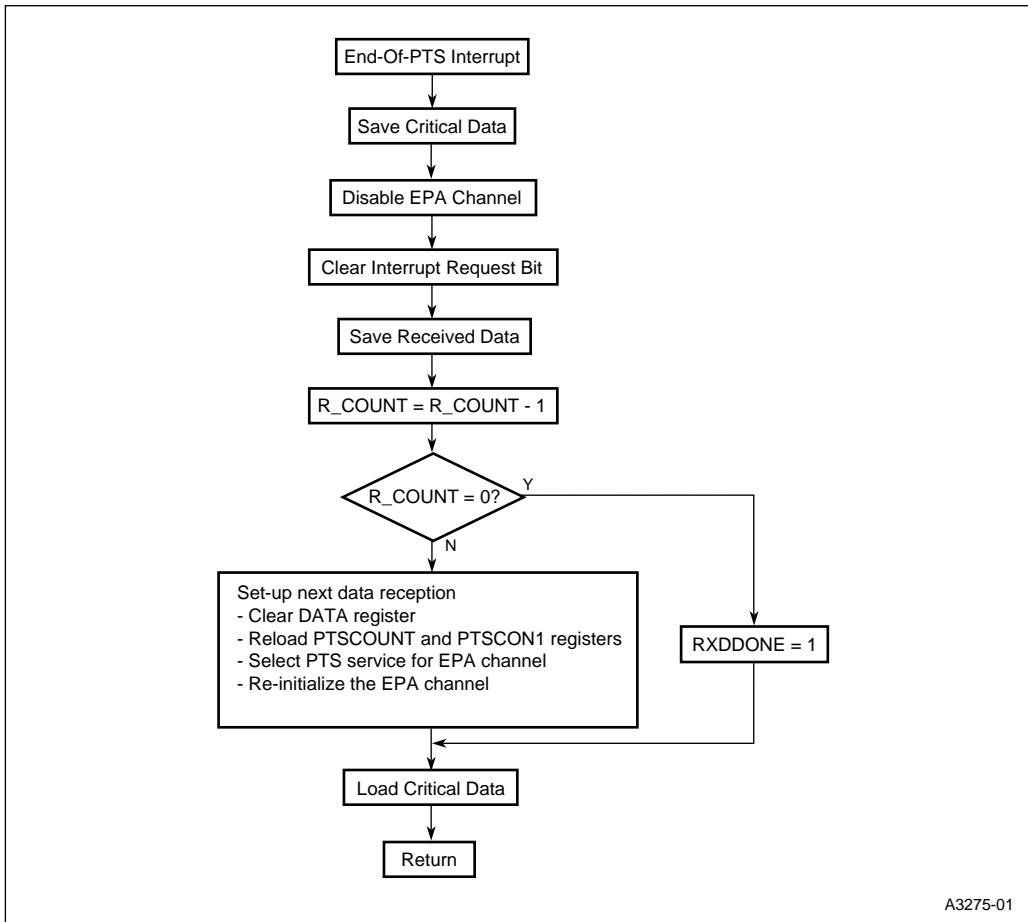


Figure 5-24. Synchronous SIO Receive Mode — End-of-PTS Interrupt Routine Flowchart

5.6.6.3 Asynchronous SIO Transmit Mode Example

In asynchronous serial I/O (ASIO) transmit mode, an EPA channel controls the transmission baud rate by generating an interrupt whenever a match occurs between the EPA event-time register and a timer register. The PTS shifts a data bit out onto a port pin that is configured to function as the Transmit Data signal (TXD) when the selected EPA channel generates a compare interrupt (Figure 5-25). In ASIO transmit mode, the PTS automatically transmits up to 16 bits (data + 1 optional parity + 1 stop bit). The maximum number of data bits is 14 with parity, or 15 without.

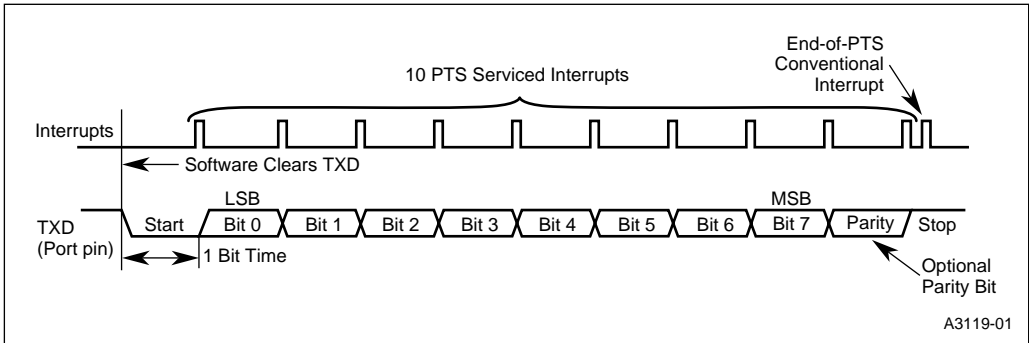


Figure 5-25. Asynchronous SIO Transmit Timing

The first PTS cycle must be started manually by generating a start bit and then setting up the timing for the first EPA interrupt.

- Initialize the TXD port pin to one before starting a transmission.
- Write a zero to the TXD port pin to start the transmission. The PTS uses this as the start bit.
- Add the contents of the timer register to the Baud_value (Figure 5-19 on page 5-38) and store the result into the EPA time register. This sets up the timing for the first interrupt and causes the first bit transmission to occur at the proper baud rate.

The following example uses P2.0 to output the data (TXD) and EPA0 to control the baud rate. It sets up an asynchronous serial I/O PTS routine that transmits 16 bytes with eight data bits, one parity bit, and one stop bit at 9600 baud. This example uses several user-defined registers. T_COUNT defines the number of bytes to transfer and TXDDONE is a flag that is set when all bytes are transferred.

1. Disable the interrupts and the PTS.
 - Use the DI instruction to disable all standard interrupts and the DPTS instruction to disable the PTS.
2. Set-up the stack pointer.
3. Reset all interrupt mask registers.
 - Clear INT_MASK, INT_MASK1 and PI_MASK.
4. Initialize P2.0 to function as TXD.
 - Clear P2_DIR.0 (selects output).
 - Clear P2_MODE.0 (selects LSIO function).
 - Set P2_REG.0 (initializes TXD output to “1”).

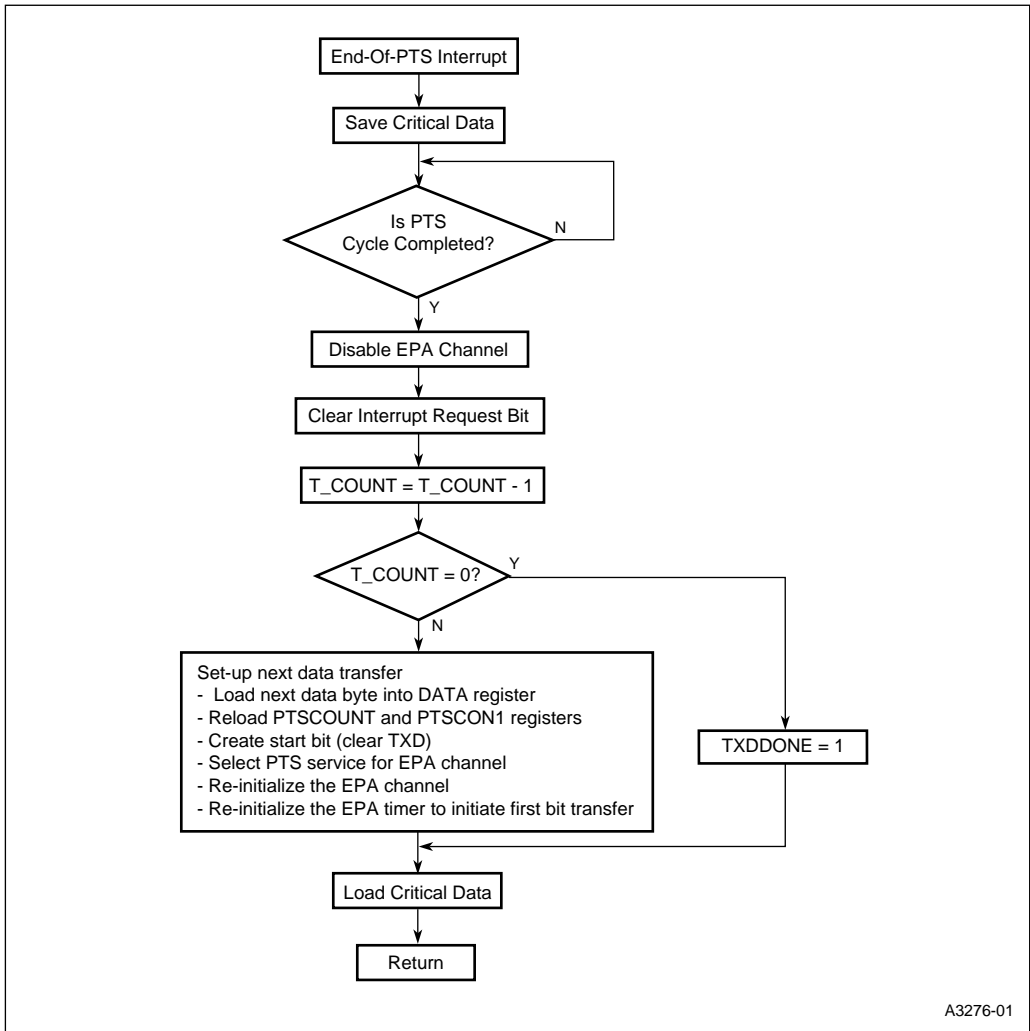
5. Initialize and enable the timer; select up counting, internal clock, and prescaler disabled.
 - Set T1CONTROL bits 6 and 7 (Figure 11-8 on page 11-16).
6. Initialize the PTSCB as shown in Table 5-15.

Table 5-15. ASIO Transmit Mode PTSCBs

PTSCB1	PTSCB2
PTSVEC (H) = pointer to PTSCB2	Unused
PTSVEC (L) = pointer to PTSCB2	SAMPTIME = unused
BAUD (H) = 01H (9600 baud at 16 MHz)	DATA (H) = unused
BAUD (L) = A0H (9600 baud at 16 MHz)	DATA (L) = <i>nn</i> H (8 data bits)
EPAREG (H) = 1FH (EPA0_TIME)	PTSCON1 = 21H (enable odd parity)
EPAREG (L) = 42H (EPA0_TIME)	PORTMASK = 01H (P2.0 = TXD)
PTSCON = 60H (ASIO transmit mode)	PORTREG (H) = 1FH (P2_REG)
PTSCOUNT = 0AH (8 data bits, 1 parity, & 1 stop bit)	PORTREG (L) = D4H (P2_REG)

7. Enable EPA0 interrupt.
 - Set INT_MASK.2.
8. Load the number of bytes to transmit into the user_defined transmit count register (T_COUNT) and clear the user-defined transfer-done flag (TXDDONE).
 - LD T_COUNT, #16
 - CLRB TXDDONE
9. Select PTS service for EPA0.
 - Set PTSSEL.2.
10. Set-up the transmission start bit.
 - Clear P2.0.
11. Set-up EPA0 as a compare-only channel.
 - Set EPA0_CON.6 (Figure 11-10 on page 11-19).
12. Start the operation of the EPA0 channel by writing the time of the first interrupt to EPA0_TIME. To set-up the correct value, add the baud_value (1A0H) to the current TIMER1 value and store the result in EPA0_TIME. The baud_value determines the time to the first PTS interrupt. When the interrupt occurs, the PTS transmits the first data bit. The baud_value of 1A0H selects a baud rate of 9600.
13. Enable the PTS and conventional interrupts.
 - Use the EI instruction to enable all standard interrupts and the EPTS instruction to enable the PTS.

14. The transmission will begin. Data is shifted out with the least-significant (rightmost) bit first. Each time a timer match occurs between EPA0_TIME and TIMER1, the EPA0 channel generates an interrupt and the PTS outputs the next bit of data on the pin configured as TXD. When PTSCOUNT decrements to zero, the PTS calls the end-of-PTS interrupt (Figure 5-26). The interrupt service routine should load the next data byte, reload the PTSCOUNT and PTSCON1 registers, clear the TXD bit to create the start bit for the next word to be transmitted, select PTS service for EPA0, and reload both the EPA0_CONTROL and EPA0_TIME registers.
15. To determine when all bytes have been transmitted, create a loop routine to check the status of the TXDDONE flag.



A3276-01

Figure 5-26. Asynchronous SIO Transmit Mode — End-of-PTS Interrupt Routine Flowchart

5.6.6.4 Asynchronous SIO Receive Mode Example

In asynchronous serial I/O (ASIO) receive mode, an EPA channel is set up to capture the falling edge when the data start bit toggles on a port pin that is configured to function as the Receive Data signal (RXD). When the capture occurs, the EPA generates a conventional interrupt which starts the asynchronous receive process. This conventional interrupt service routine would be the same as the end-of-PTS interrupt service routine. It changes the EPA channel to the compare mode and sets the time of the next compare to 1.5 bit times and enables the PTS. At exactly 1.5 bit times from the beginning of the start bit, the first PTS cycle samples the input data on RXD and shifts it into the DATA register (Figure 5-27). If majority sampling is enabled, an additional sample occurs. If the two samples differ, a third sample occurs to determine which of the first two samples is correct. The SAMPTIME register (Figure 5-19 on page 5-38) controls the time between samples. Majority sampling causes a substantial increase in PTS cycle execution time (Table 5-4 on page 5-12).

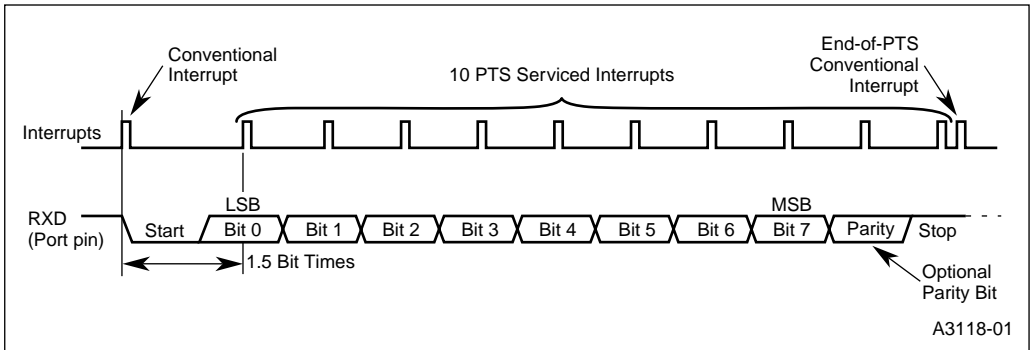


Figure 5-27. Asynchronous SIO Receive Timing

The first PTS cycle must be started manually.

- Initialize the RXD port pin and the SCK signal to the system-required logic level before starting a reception.
- Add the contents of the timer register to the Baud_value (Figure 5-19 on page 5-38) and store the result into the EPA time register. This sets up the timing for the first interrupt and causes the first interrupt to occur at the proper baud rate.

The following example uses EPA0 to capture the start bit and P2.0 to receive the data (RXD). It sets up an asynchronous serial I/O PTS routine that receives 16 bytes each with eight data bits and a parity bit. This example uses several user-defined registers. R_COUNT defines the number of bytes to receive and RXDDONE is a flag that is set when all bytes are received.

1. Disable the interrupts and the PTS.
 - Use the DI instruction to disable all standard interrupts and the DPTS instruction to disable the PTS.

2. Set-up the stack pointer.
3. Reset all interrupt mask registers.
 - Clear INT_MASK, INT_MASK1 and PI_MASK.
4. Initialize P2.0 to function as the RXD signal.
 - Set P2_DIR.0 (selects input).
 - Clear P2_MODE.0 (selects LSIO function).
 - Set P2_REG.0 (initializes RXD input to “1”).
5. Initialize and enable the timer; select up counting, internal clock, and prescaler disabled.
 - Set T1CONTROL bits 6 and 7 (Figure 11-8 on page 11-16).
6. Initialize the PTSCB as shown in Table 5-16.

Table 5-16. ASIO Receive Mode PTSCBs

PTSCB1	PTSCB2
PTSVEC (H) = pointer to PTSCB2	Unused
PTSVEC (L) = pointer to PTSCB2	SAMPTIME = 01H
BAUD (H) = 01H	DATA (H) = 00H (clear register to receive data)
BAUD (L) = A0H	DATA (L) = 00H (clear register to receive data)
EPAREG (H) = 1FH (EPA0_TIME)	PTSCON1 = 60H (enable odd parity)
EPAREG (L) = 42H (EPA0_TIME)	PORTMASK = 01H (P2.0 = RXD)
PTSCON = 21H (SSIO receive mode, majority sampling)	PORTREG (H) = 1FH (P2_PIN)
PTSCOUNT = 0AH (receive 8 data bits, 1 parity bit)	PORTREG (L) = D6H (P2_PIn)

7. Enable EPA0 interrupt.
 - Set INT_MASK.2.
8. Load the number of bytes to transmit into the user_defined transmit count register (R_COUNT) and clear the user-defined reception-done flag (RXDDONE).
 - LD R_COUNT, #16
 - CLRB RXDDONE
9. Select PTS service for EPA0.
 - Set PTSSEL.2.
10. Set-up EPA0 to capture on falling edges.
 - Set EPA0_CON.4 (Figure 11-10 on page 11-19).

11. Enable the PTS and conventional interrupts.
 - Use the EI instruction to enable all standard interrupts and the EPTS instruction to enable the PTS.
12. Toggle the RXD input to start the reception. The EPA will generate a conventional interrupt. This interrupt service routine should be the same as the end-of-PTS routine. The service routine can determine if this is a start reception interrupt or a end-of-PTS interrupt by reading the EPA0_CON register. If the register is set to capture mode, then this is the start reception interrupt.

The interrupt service routine contains the following steps.

1. Reconfigure the EPA channel to compare mode.
 - Set EPA0_CON.6 (selects compare mode).
2. Initialize the first PTS cycle time by writing the time of the first interrupt to EPA0_TIME. To set-up the correct value, multiply the baud_value (1A0H) by 1.5, add the product to the current TIMER1 value, and store the result in EPA0_TIME. The baud_value determines the time to the first PTS interrupt. When the interrupt occurs, the PTS transmits the first data bit. The baud_value of 1A0H selects a baud rate of 9600. In this example, the value added to the current TIMER1 value is 270H.
3. Select PTS service for EPA0.
 - Set PTSSEL.2.
4. The PTS routine begins sampling the data every 1.5 bit times. When PTSCOUNT decrements to zero, the PTS calls the end-of-PTS interrupt (Figure 5-28). The interrupt service routine should check to see if a framing or parity error occurred (Figure 5-19 on page 5-38), clear the DATA registers to prepare for the next reception, reload the PTSCOUNT and PTSCON1 registers, and reconfigure the EPA channel to select capture on falling edge mode. Note that PTS service is not enabled for the EPA channel at this time because the next interrupt must be a conventional interrupt.
5. To determine when all bytes have been transmitted, create a loop routine to check the status of the RXDDONE flag.

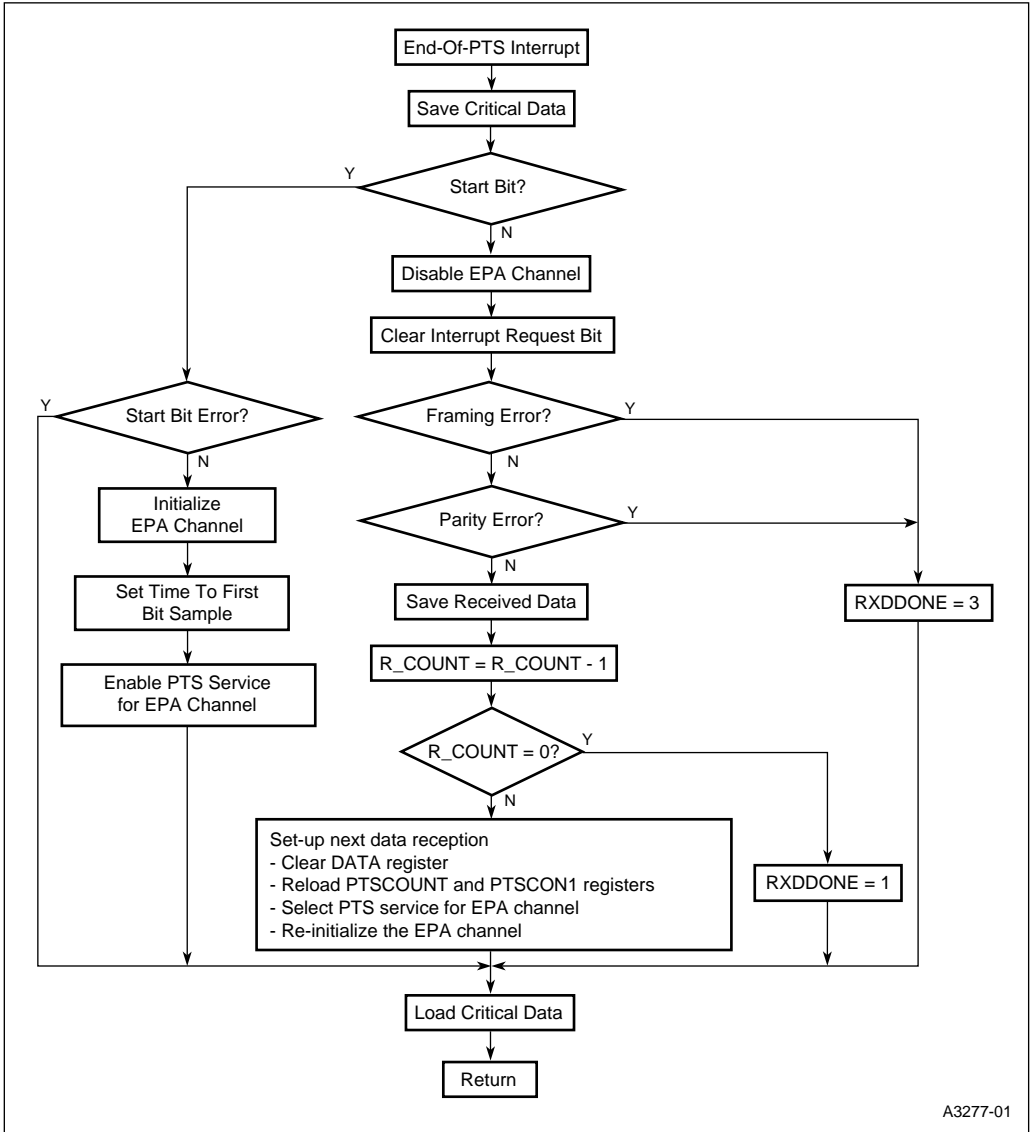


Figure 5-28. Asynchronous SIO Receive Mode — End-of-PTS Interrupt Routine Flowchart



6

I/O Ports

CHAPTER 6

I/O PORTS

I/O ports provide a mechanism to transfer information between the device and the surrounding system circuitry. They can read system status, monitor system operation, output device status, configure system options, generate control signals, provide serial communication, and so on. Their usefulness in an application is limited only by the number of I/O pins available and the imagination of the engineer.

6.1 I/O PORTS OVERVIEW

Standard I/O port registers are located in the SFR address space and they can be windowed. Memory-mapped I/O port registers are located in memory-mapped address space. Memory-mapped registers must be accessed with indirect or indexed addressing; they cannot be windowed. All ports can provide low-speed input/output pins or serve alternate functions. Table 6-1 provides an overview of the device I/O ports. The remainder of this chapter describes the ports in more detail and explains how to configure the pins. The chapters that cover the associated peripherals discuss using the pins for their special functions.

Table 6-1. Device I/O Ports

Port	Bits	Type	Direction	Associated Peripheral(s)
Port 0	8	Standard	Input-only	A/D converter (MC, MD) A/D converter, EPA (MH)
Port 1	5 (MC) 8 (MD)	Standard	Input-only	A/D converter, EPA (MC, MD) P1.7:6 are digital input-only channels for the 8XC196MD.
	4 (MH)	Standard	Bidirectional	SIO
Port 2	8	Standard	Bidirectional	EPA and timers (MC, MD) EPA and timers, SIO (MH)
Port 3	8	Memory-mapped	Bidirectional	Address/data bus
Port 4	8	Memory-mapped	Bidirectional	Address/data bus
Port 5	8	Memory-mapped	Bidirectional	Bus control
Port 6	8	Standard	Output-only	PWM, waveform generator
Port 7 (MD)	8	Standard	Bidirectional	EPA, frequency generator P7.6:4 are low-speed input/output pins only; they have no peripheral functions.

6.2 INPUT-ONLY PORTS 1 (MC, MD ONLY) AND 0

Port 0 is an eight-bit, high-impedance, input-only port that provides analog and digital inputs. The input-only pins can be read as digital inputs; most of them are also inputs to the A/D converter. The input-only ports differ from the other standard ports in that their pins can be used only as inputs to the digital or analog circuitry. On the 8XC196MC and 8XC196MD, port 1 is an input-only port that serves the same purpose as port 0. The 8XC196MC implements five pins, while the 8XC196MD implements all eight.

8XC196MH: On the 8XC196MH, port 1 is a standard bidirectional port that shares pins with the serial I/O port. (See “Bidirectional Ports 1 (MH Only), 2, 5, and 7 (MD Only)” on page 6-4.)

Because port 0 (and port 1 of the 8XC196MC, MD) is permanently configured as an input-only port, it has no configuration registers. Its single register, P_x_PIN, can be read to determine the current state of the pin. The register is byte-addressable and can be windowed. (See “Windowing” on page 5-16)

Table 6-2 lists the standard input-only port pins and Table 6-3 describes the P_x_PIN status register.

Table 6-2. Standard Input-only Port Pins

Port Pin	Special-function Signal(s)	Special-function Signal Type	Associated Peripheral
P0.5:0	ACH5:0	Input	A/D converter
P0.6	ACH6	Input	A/D converter
	T1CLK (MH)	Input	EPA
P0.7	ACH7	Input	A/D converter
	T1DIR (MH)	Input	EPA
P1.0 (MC, MD)	ACH8	Input	A/D converter
P1.1 (MC, MD)	ACH9	Input	A/D converter
P1.2 (MC, MD)	ACH10	Input	A/D converter
	T1CLK	Input	EPA
P1.3 (MC, MD)	ACH11	Input	A/D converter
	T1DIR	Input	EPA
P1.4 (MC, MD)	ACH12	Input	A/D converter
P1.5 (MD)	ACH13	Input	A/D converter
P1.6 (MD)	—	—	—
P1.7 (MD)	—	—	—

Table 6-3. Input-only Port Registers

Mnemonic	Address	Description
P0_PIN	1FA8H (MC, MD) 1FDAH (MH)	Each bit of P0_PIN reflects the current state of the corresponding port 0 pin.
P1_PIN (MC, MD)	1FA9H (MC, MD)	Each bit of P1_PIN reflects the current state of the corresponding port 1 pin.

6.2.1 Standard Input-only Port Operation

Figure 6-1 is a schematic of an input-only port pin. Transistors Q1 and Q2 serve as electrostatic discharge (ESD) protection devices; they are referenced to V_{REF} and ANGND. Transistor Q3 is an additional ESD protection device; it is referenced to V_{SS} (digital ground). Resistor R1 limits current flow through Q3 to acceptable levels. At this point, the input signal is sent to the analog multiplexer and to the digital level-translation buffer. The level-translation buffer converts the input signals to work with the V_{CC} and V_{SS} digital voltage levels used by the CPU core. This buffer is Schmitt-triggered for improved noise immunity. The signals are latched in the P0_PIN or P1_PIN register and are output onto the internal bus when P0_PIN or P1_PIN is read.

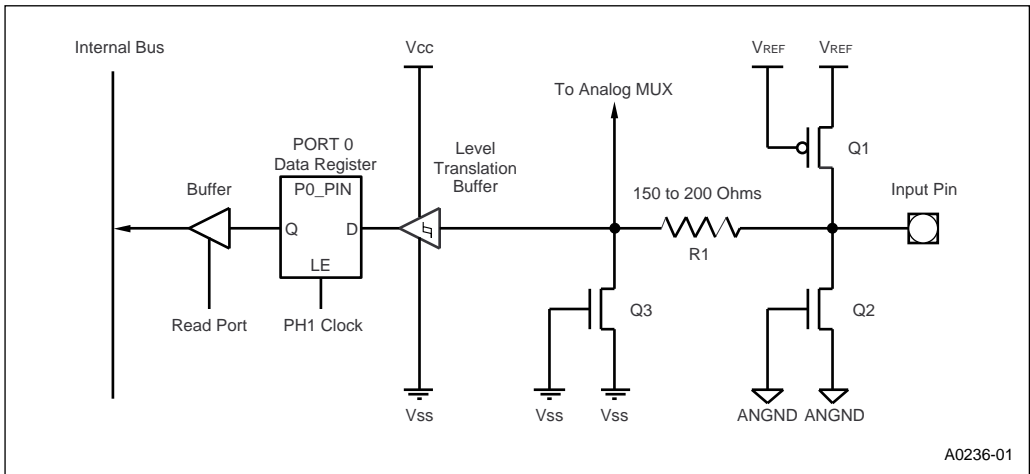


Figure 6-1. Standard Input-only Port Structure

A0236-01

6.2.2 Standard Input-only Port Considerations

Port 0 and 1 pins are unique in that they may individually be used as digital inputs and analog inputs at the same time. However, reading the port induces noise into the A/D converter, decreasing the accuracy of any conversion in progress. We strongly recommend that you **not** read the port while an A/D conversion is in progress. To reduce noise, the P0_PIN or P1_PIN register is clocked only when the port is read.

These port pins are powered by the analog reference voltage (V_{REF}) and analog ground (ANGND) pins. If the port pins are to function as either analog or digital inputs, the V_{REF} and ANGND pins must provide power. If the voltage applied to the analog input exceeds V_{REF} or ANGND by more than 0.5 volts, current will be driven through Q1 or Q2 into the reference circuitry, decreasing the accuracy of all analog conversions.

The port pin is sampled one state time before the read buffer is enabled. Sampling occurs during phase 1 (while CLKOUT is low) and resolves the value of the pin before it is presented to the internal bus. To ensure that the value is recognized, it must be valid 45 ns before the rising edge of CLKOUT and must remain valid until CLKOUT falls. If the pin value changes during the sample time, the new value may or may not be recorded.

As a digital input, a pin acts as a high-impedance input. However, as an analog input, a pin must provide current for a short time to charge the internal sample capacitor when a conversion begins. This means that if a conversion is taking place on a port pin, its input characteristics change momentarily.

6.3 BIDIRECTIONAL PORTS 1 (MH ONLY), 2, 5, AND 7 (MD ONLY)

Although the bidirectional ports are very similar in both circuitry and configuration, port 5 differs from the others in some ways. Port 5, a memory-mapped port, uses a standard CMOS input buffer because of the high speeds required for system control functions. The remaining bidirectional ports use Schmitt-triggered input buffers for improved noise immunity.

NOTE

Ports 3 and 4 are significantly different from the other bidirectional ports. See “Bidirectional Ports 3 and 4 (Address/Data Bus)” on page 6-14 for details on the structure and operation of these ports.

Table 6-4 lists the bidirectional port pins with their special-function signals and associated peripherals.

Table 6-4. Bidirectional Port Pins

Port Pin	Special-function Signal(s)	Special-function Signal Type	Associated Peripheral
P1.0 (MH)	TXD0	O	SIO
P1.1 (MH)	RXD0	I/O	SIO
P1.2 (MH)	TXD1	O	SIO
P1.3 (MH)	RXD1	I/O	SIO
P2.0	EPA0	I/O	EPA
P2.1	EPA1 (MC, MD)	I/O	EPA
	SCLK0# (MH) BCLK0 (MH)	I/O I	SIO SIO
P2.2	EPA2 (MC, MD)	I/O	EPA
	EPA1 (MH)	I/O	EPA
P2.3	EPA3 (MC, MD)	I/O	EPA
	COMP3 (MH)	O	EPA
P2.4	COMP0	O	EPA
P2.5	COMP1	O	EPA
P2.6	COMP2	O	EPA
P2.7	COMP3 (MC, MD)	O	EPA
	SCLK1# (MH) BCLK1 (MH)	I/O I	SIO SIO
P5.0	ALE/ADV#	O	Bus controller
P5.1	INST	O	Bus controller
P5.2	WR#/WRL#	O	Bus controller
P5.3	RD#	O	Bus controller
P5.4	ONCE	I	Bus controller
P5.5	BHE#/WRH#	O	Bus controller
P5.6	READY	I	Bus controller
P5.7	BUSWIDTH	I	Bus controller
P7.0 (MD)	EPA4	I/O	EPA
P7.1 (MD)	EPA5	I/O	EPA
P7.2 (MD)	COMP4	O	EPA
P7.3 (MD)	COMP5	O	EPA
P7.4 (MD)	—	I/O	—
P7.5 (MD)	—	I/O	—
P7.6 (MD)	—	I/O	—
P7.7 (MD)	FREQOUT	O	Frequency generator

Table 6-5 lists the registers associated with the bidirectional ports. Each port has three control registers (Px_MODE, Px_DIR, and Px_REG); they can be both read and written. The Px_PIN register is a status register that returns the logic level present on the pins; it can only be read. The registers for the standard ports are byte-addressable and can be windowed. The port 5 registers must be accessed using 16-bit addressing and **cannot** be windowed. “Bidirectional Port Considerations” on page 6-12 discusses special considerations for reading P2_REG.7 and P6_REG.7:4.

Table 6-5. Bidirectional Port Control and Status Registers

Mnemonic	Address	Description
P1_DIR (MH) P2_DIR P5_DIR P7_DIR (MD)	1F9BH 1FD2H 1FF3H 1FD3H	Port x Direction Each bit of Px_DIR controls the direction of the corresponding pin. 0 = complementary output (output only) 1 = input or open-drain output (input, output, or bidirectional) Open-drain outputs require external pull-ups.
P1_MODE (MH) P2_MODE P5_MODE P7_MODE (MD)	1F99H 1FD0H 1FF1H 1FD1H	Port x Mode Each bit of Px_MODE controls whether the corresponding pin functions as a standard I/O port pin or as a special-function signal. 0 = standard I/O port pin 1 = special-function signal
P1_PIN (MH) P2_PIN P5_PIN P7_PIN (MD)	1F9FH 1FD6H 1FF7H 1FD7H	Port x Input Each bit of Px_PIN reflects the current state of the corresponding pin, regardless of the pin configuration.
P1_REG (MH) P2_REG P5_REG P7_REG (MD)	1F9DH 1FD4H 1FF5H 1FD5H	Port x Data Output For an input, set the corresponding Px_REG bit. For an output, write the data to be driven out by each pin to the corresponding bit of Px_REG. When a pin is configured as standard I/O (Px_MODE.y = 0), the result of a CPU write to Px_REG is immediately visible on the pin. When a pin is configured as a special-function signal (Px_MODE.y = 1), the associated on-chip peripheral or off-chip component controls the pin. The CPU can still write to Px_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 standard I/O (clear Px_MODE.y), initialize or overwrite the pin value, then configure the pin as a special-function signal (set Px_MODE.y). In this way, initialization, fault recovery, exception handling, etc., can be done without changing the operation of the associated peripheral.

6.3.1 Bidirectional Port Operation

Figure 6-2 shows the logic for driving the output transistors, Q1 and Q2. Q1 can source at least -3 mA at $V_{CC} - 0.7$ volts. Q2 can sink at least 3 mA at 0.45 volts. (Consult the datasheet for specifications.)

In I/O mode (selected by clearing $Px_MODE.y$), Px_REG and Px_DIR 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. Table 6-6 is a logic table for I/O operation of these ports.

In special-function mode (selected by setting $Px_MODE.y$), $SFDIR$ 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 $SFDIR$. Table 6-7 is a logic table for special-function operation of these ports. 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 Px_DIR .

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 Px_PIN sample latch and output onto the internal bus when the Px_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 can source approximately $-10\ \mu A$; consult the datasheet for exact specifications.) Q4 remains on, weakly holding the pin high, until your software writes to the Px_MODE register.

NOTE (8XC196MC, MD Only)

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

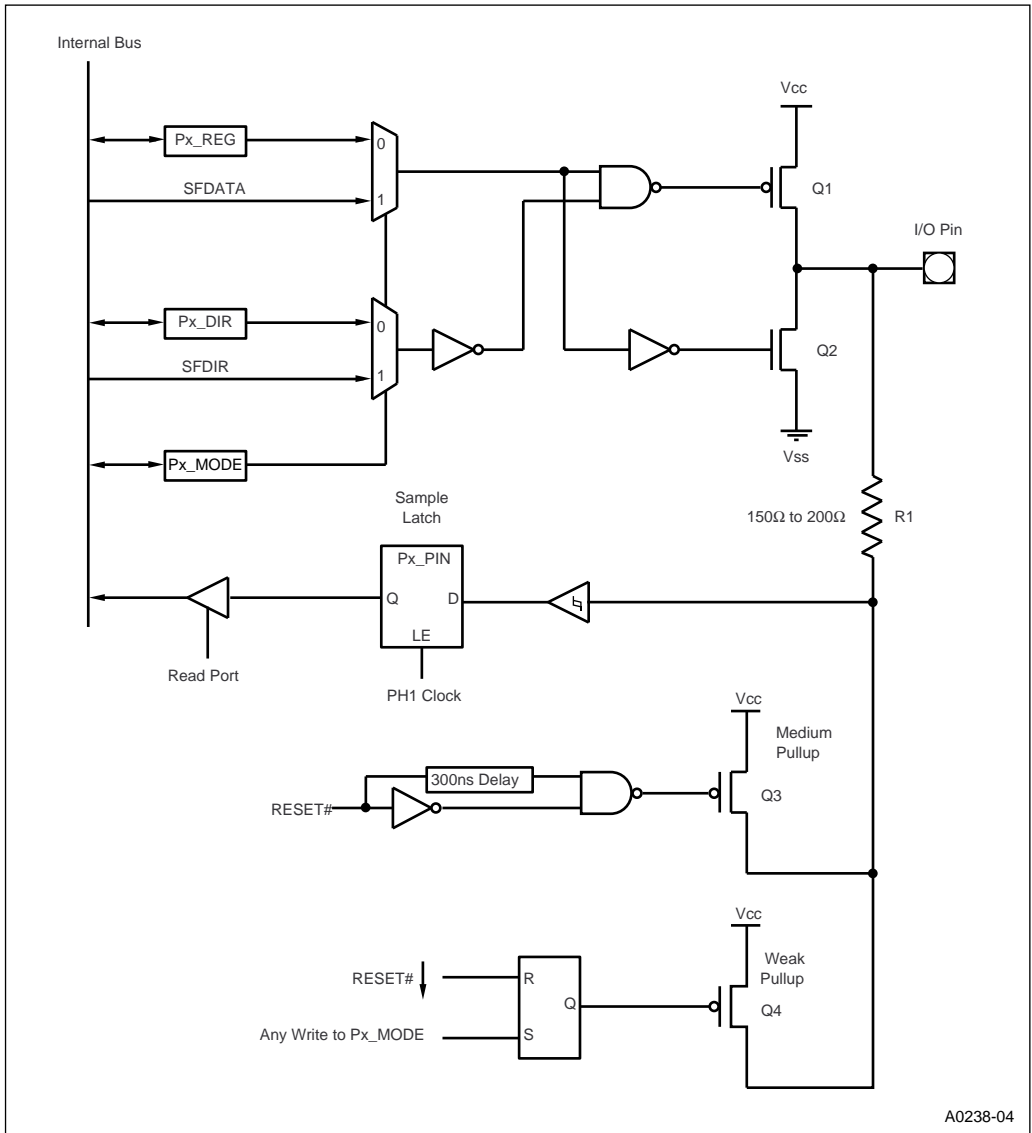


Figure 6-2. Bidirectional Port Structure

Table 6-6. Logic Table for Bidirectional Ports in I/O Mode

Configuration	Complementary Output		Open-drain Output	Input
Px_MODE	0	0	0	0
Px_DIR	0	0	1	1
SFDIR	X	X	X	X
SFDATA	X	X	X	X
Px_REG	0	1	0, 1 (Note 2)	1
Q1	off	on	off	off
Q2	on	off	on, off (Note 2)	off
Px_PIN	0	1	X (Note 3)	high impedance (Note 4)

NOTES:

1. X = Don't care.
2. If Px_REG is cleared, Q2 is on; if Px_REG is set, Q2 is off.
3. Px_PIN contains the current value on the pin.
4. During reset and until the first write to Px_MODE, Q4 is on.

Table 6-7. Logic Table for Bidirectional Ports in Special-function Mode

Configuration	Complementary Output		Open-drain Output	Input
Px_MODE	1	1	1	1
Px_DIR	0	0	1	1
SFDIR	0	0	1	1
SFDATA	0	1	0, 1 (Note 2)	1
Px_REG	X	X	X	1
Q1	off	on	off	off
Q2	on	off	on, off (Note 2)	off
Px_PIN	0	1	X (Note 3)	high impedance (Note 4)

NOTES:

1. X = Don't care.
2. If Px_REG is cleared, Q2 is on; if Px_REG is set, Q2 is off.
3. Px_PIN contains the current value on the pin.
4. During reset and until the first write to Px_MODE, Q4 is on.

6.3.2 Bidirectional Port Pin Configurations

Each bidirectional port pin can be individually configured to operate either as an I/O pin or as a pin for a special-function signal. In the special-function configuration, the signal is controlled by an on-chip peripheral or an off-chip component. In either configuration, two modes are possible:

- complementary output (output only)
- high-impedance input or open-drain output (input, output, or bidirectional)

To prevent the CMOS inputs from floating, the bidirectional port pins are weakly pulled high during and after reset, until your software writes to Px_MODE. The default values of the control registers after reset configure the pins as high-impedance inputs with weak pull-ups. To ensure that the ports are initialized correctly and that the weak pull-ups are turned off, follow this suggested initialization sequence:

1. Write to Px_DIR to establish the individual pins as either inputs or outputs. (Outputs will drive the data that you specify in step 3.)
 - For a complementary output, clear its Px_DIR bit.
 - For a high-impedance input or an open-drain output, set its Px_DIR bit. (Open-drain outputs require external pull-ups.)
2. Write to Px_MODE to select either I/O or special-function mode. Writing to Px_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 Px_MODE to ensure that the weak pull-ups are turned off.**
 - For a standard I/O pin, clear its Px_MODE bit. In this mode, the pin is driven as defined in steps 1 and 3.
 - For a special-function signal, set its Px_MODE bit. In this mode, the associated peripheral controls the pin.
3. Write to Px_REG.
 - For output pins defined in step 1, write the data that is to be driven by the pins to the corresponding Px_REG bits. For special-function outputs, the value is immaterial because the peripheral controls the pin. However, you must still write to Px_REG to initialize the pin.
 - For input pins defined in step 1, set the corresponding Px_REG bits.

Table 6-8 lists the control register values for each possible configuration. For special-function outputs, the Px_REG value is irrelevant (don't care) because the associated peripheral controls the pin in special-function mode. However, you must still write to Px_REG to initialize the pin. For a bidirectional pin to function as an input (either special function or port pin), you must set Px_REG.

Table 6-8. Control Register Values for Each Configuration

Desired Pin Configuration	Configuration Register Settings		
	Px_DIR	Px_MODE†	Px_REG
Standard I/O Signal			
Complementary output, driving 0	0	0	0
Complementary output, driving 1	0	0	1
Open-drain output, strongly driving 0	1	0	0
Open-drain output, high impedance	1	0	1
Input	1	0	1
Special-function signal			
	Px_DIR	Px_MODE†	Px_REG
Complementary output, output value controlled by peripheral	0	1	X
Open-drain output, output value controlled by peripheral	1	1	X
Input	1	1	1

† During reset and until the first write to Px_MODE, the pins are weakly held high.

6.3.3 Bidirectional Port Pin Configuration Example

Assume that you wish to configure the pins of a bidirectional port as shown in Table 6-9.

Table 6-9. Port Configuration Example

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

To do so, you could use the following example code segment. Table 6-10 shows the state of each pin after reset and after execution of each line of the example code.

```
LDB Px_DIR, #00011111B
LDB Px_MODE, #00000000B
LDB Px_REG, #10010011B
```

Table 6-10. Port Pin States After Reset and After Example Code Execution

Action or Code	Resulting Pin States [†]							
	Px.7	Px.6	Px.5	Px.4	Px.3	Px.2	Px.1	Px.0
Reset	wk1	wk1	wk1	wk1	wk1	wk1	wk1	wk1
LDB Px_DIR, #00011111B	1	1	1	wk1	wk1	wk1	wk1	wk1
LDB Px_MODE, #00000000B	1	1	1	HZ1	HZ1	HZ1	HZ1	HZ1
LDB Px_REG, #10010011B	1	0	0	HZ1	0	0	HZ1	HZ1

[†] wk1 = weakly pulled high, HZ1 = high impedance (actually a "1" with an external pull-up).

6.3.4 Bidirectional Port Considerations

This section outlines special considerations for using the pins of these ports.

Port 1 (8XC196MH) After reset, your software must configure the device to match the external system. This is accomplished by writing appropriate configuration data into P1_MODE. Writing to P1_MODE not only configures the pins but also turns off the transistor that weakly holds the pins high (Q4 in Figure 6-2 on page 6-8). For this reason, even if port 1 is to be used as it is configured at reset, you should still write data into P1_MODE.

Port 2 After reset, your software must configure the device to match the external system. This is accomplished by writing appropriate configuration data into P2_MODE. Writing to P2_MODE not only configures the pins but also turns off the transistor that weakly holds the pins high (Q4 in Figure 6-2 on page 6-8). For this reason, even if port 2 is to be used as it is configured at reset, you should still write data into P2_MODE.

P2.7 A value written to P2_REG.7 is held in a buffer until P2_MODE.7 is cleared, at which time the value is loaded into P2_REG.7. A value read from P2_REG.7 is the value currently in the register, not the value in the buffer. Therefore, any change to P2_REG.7 can be read only after P2_MODE.7 is cleared.

Port 5 After reset, the device configures port 5 to match the external system. The following paragraphs describe the states of the port 5 pins after reset and until your software writes to the P5_MODE register. Writing to P5_MODE not only configures the pins but also turns off the transistor that weakly holds the pins high (Q4 in Figure 6-2 on page 6-8). For this reason, even if port 5 is to be used as it is configured at reset, you should still write data into P5_MODE.

- P5.0/ALE** If EA# is high on reset (internal access), the pin is weakly held high until your software writes to P5_MODE. If EA# is low on reset (external access), either ALE or ADV# is activated as a system control pin, depending on the ALE bit of CCR0. In either case, the pin becomes a true complementary output.
- P5.1/INST** This pin remains weakly held high until your software writes configuration data into P5_MODE.
- P5.2/WR#/WRL#** This pin remains weakly held high until your software writes configuration data into P5_MODE.
- P5.3/RD#** If EA# is high on reset (internal access), the pin is weakly held high until your software writes to P5_MODE. If EA# is low on reset (external access), RD# is activated as a system control pin and the pin becomes a true complementary output.
- P5.4** This pin is weakly held high until your software writes to P5_MODE. P5.4 is the enable pin for ONCE mode (see Chapter 14, “Special Operating Modes”) and one of the enable pins for Intel-reserved test modes. Because a low input during reset could cause the device to enter ONCE mode or a reserved test mode, **exercise caution** if you use this pin for input. Be certain that your system meets the V_{IH} specification (listed in the datasheet) during reset to prevent inadvertent entry into ONCE mode or a test mode.
- P5.5/BHE#/WRH#** This pin is weakly held high until the CCB fetch is completed. At that time, the state of this pin depends on the value of the BW0 bit of the CCRs. If BW0 is clear, the pin remains weakly held high until your software writes to P5_MODE. If BW0 is set, BHE# is activated as a system control pin and the pin becomes a true complementary output.
- P5.6/READY** This pin remains weakly held high until the CCB fetch is completed. At that time, the state of this pin depends on the value of the IRC0–IRC2 bits of the CCRs. If IRC0–IRC2 are all set (111B), READY is activated as a system control pin. This prevents the insertion of infinite wait states upon the first access to external memory. For any other values of IRC0–IRC2, the pin is configured as I/O upon reset.
- NOTE:** If IRC0–IRC2 of the CCB are all set (activating READY as a system control pin) and P5_MODE.6 is cleared (configuring the pin as I/O), an external memory access may cause the processor to lock up.
- P5.7/BUSWIDTH** This pin remains weakly held high until your software writes configuration data into P5_MODE.

6.4 BIDIRECTIONAL PORTS 3 AND 4 (ADDRESS/DATA BUS)

Ports 3 and 4 are eight-bit, bidirectional, memory-mapped I/O ports. They can be addressed only with indirect or indexed addressing and cannot be windowed. Ports 3 and 4 provide the multiplexed address/data bus. In programming modes, ports 3 and 4 serve as the programming bus (PBUS). Port 5 supplies the bus-control signals.

During external memory bus cycles, the processor takes control of ports 3 and 4 and automatically configures them as complementary output ports for driving address/data or as inputs for reading data. For this reason, these ports have no mode registers.

Systems with EA# tied inactive have idle time between external bus cycles. When the address/data bus is idle, you can use the ports for I/O. Like port 5, these ports use standard CMOS input buffers. However, ports 3 and 4 must be configured entirely as complementary or open-drain ports; their pins cannot be configured individually. Systems with EA# tied active cannot use ports 3 and 4 as standard I/O; when EA# is active, these ports will function only as the address/data bus.

Table 6-11 lists the port 3 and 4 pins with their special-function signals and associated peripherals. Table 6-12 lists the registers that affect the function and indicate the status of ports 3 and 4.

Table 6-11. Ports 3 and 4 Pins

Port Pins	Special-function Signal(s)	Special-function Signal Type	Associated Peripheral
P3.7:0	AD7:0	I/O	Address/data bus, low byte
	PBUS7:0	I/O	Programming bus, low byte
P4.7:0	AD15:8	I/O	Address/data bus, high byte
	PBUS15:8	I/O	Programming bus, high byte

Table 6-12. Ports 3 and 4 Control and Status Registers

Mnemonic	Address	Description
P3_PIN P4_PIN	1FFEh 1FFFh	Port x Input Each bit of P _x _PIN reflects the current state of the corresponding pin, regardless of the pin configuration.
P3_REG P4_REG	1FFCh 1FFDh	Port x Data Output Each bit of P _x _REG contains data to be driven out by the corresponding pin. When the device requires access to external memory, it takes control of the port and drives the address/data bit onto the pin. The address/data bit replaces your output during this time. When the external access is completed, the device restores your data onto the pin.

6.4.1 Bidirectional Ports 3 and 4 (Address/Data Bus) Operation

Figure 6-3 shows the ports 3 and 4 logic. During reset, the active-low level of RESET# turns off Q1 and Q2 and turns on transistor Q3, which weakly pulls the pin high. (Q1 can source at least – 3 mA at $V_{CC} - 0.7$ volts; Q2 can sink at least 3 mA at 0.45 volts; and Q3 can source approximately –10 μ A at $V_{CC} - 1.0$ volts. Consult the datasheet for exact specifications.) During normal operation, an internal control signal, BUS CONTROL SELECT, controls the port.

When the microcontroller needs to access external memory, it clears BUS CONTROL SELECT, which selects 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 microcontroller sets BUS CONTROL SELECT, which selects P_x_REG as the input to the multiplexer. P_x_REG then drives Q1 and Q2 as open-drain outputs. (Open-drain outputs require external pull-up resistors.) In this configuration, a port 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 microcontroller and which are driven high by external drivers. Table 6-13 is a logic table for ports 3 and 4 as I/O.

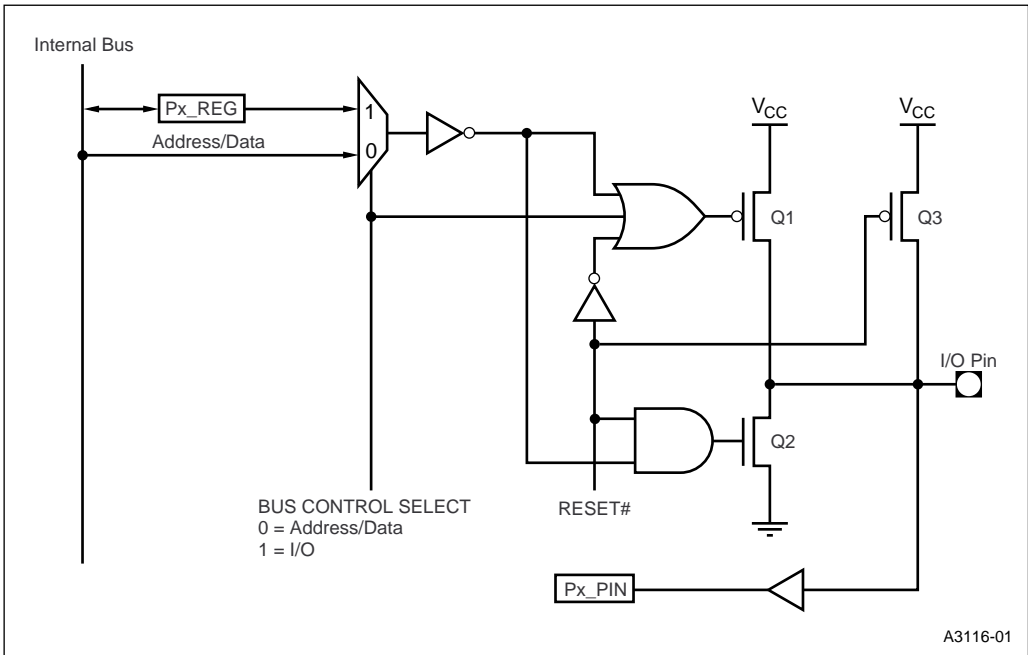


Figure 6-3. Address/Data Bus (Ports 3 and 4) Structure

Table 6-13. Logic Table for Ports 3 and 4 as Open-drain I/O

Configuration	Open-drain	
	0	1
Px_REG	0	1
Q1	off	off
Q2	on	off
Px_PIN	0	high impedance

6.4.2 Using Ports 3 and 4 as I/O

To use a port pin as an output, write the output data to the corresponding Px_REG bit. When the device requires access to external memory, it takes control of the port and drives the address/data bit onto the pin. The address/data bit replaces your output during this time. When the external access is completed, the device restores your data onto the pin.

To use a port pin as an input, set the corresponding Px_REG bit to drive the pin to a high-impedance state. You may then read the pin's input value in the Px_PIN register. When the device requires access to external memory, it takes control of the port. You must configure the input source to avoid contention on the bus.

6.4.3 Design Considerations for Ports 3 and 4

When EA# is active, ports 3 and 4 will function **only** as the address/data bus. In these circumstances, an instruction that operates on P3_REG or P4_REG causes a bus cycle that reads from or writes to the external memory location corresponding to the SFR's address. (For example, writing to P4_REG causes a bus cycle that writes to external memory location 1FFDH.) Because P3_REG and P4_REG have no effect when EA# is active, the bus will float during long periods of inactivity (such as during a BMOV or TIJMP instruction).

When EA# is inactive, ports 3 and 4 output the contents of the P3_REG and P4_REG registers, which reset to FFH, placing the pins in a high-impedance state. Ports 3 and 4 will float unless you either connect external resistors to the pins or write zeros to the P3_REG and P4_REG registers.

6.5 STANDARD OUTPUT-ONLY PORT 6

Port 6 is an output-only port that provides output pins for the waveform generator and pulse-width modulator (PWM). The port 6 pins can be configured to operate either as port pins or as output pins for the waveform generator or pulse-width modulator. Table 6-2 lists the pins with their special-function signals and associated peripherals.

Table 6-14. Standard Output-only Port Pins

Port Pin	Special-function Signal(s)	Special-function Signal Type	Associated Peripheral
P6.0	WG1#	Output	Waveform generator
P6.1	WG1	Output	Waveform generator
P6.2	WG2#	Output	Waveform generator
P6.3	WG2	Output	Waveform generator
P6.4	WG3#	Output	Waveform generator
P6.5	WG3	Output	Waveform generator
P6.6	PWM0	Output	PWM
P6.7	PWM1	Output	PWM

Table 6-15. Output-only Port Control Register

Mnemonic	Address	Description
WG_OUTPUT	1FC0H	Port 6 Output Control Register This register controls the port 6 pins in I/O mode. Its functions differ when the port 6 pins are being used as waveform generator or PWM outputs.

6.5.1 Output-only Port Operation

Figure 6-4 shows a simplified circuit schematic for port 6. Port 6 has a single configuration and control register, WG_OUTPUT. Transistor Q1 can source at least $-200\ \mu\text{A}$ at $V_{CC}-0.3$ volts. For pins P6.0–P6.5, transistor Q2 can sink at least 10 mA at 0.45 volts. For pins P6.6 and P6.7, Q2 can sink at least $200\ \mu\text{A}$ at 0.3 volts.

6.5.2 Configuring Output-only Port Pins

Port 6 has a single configuration register, WG_OUTPUT (Figure 6-5). This register controls the pin functions, values, and output polarity. This register can be addressed either as a word or as separate bytes, and it can be windowed. The functions of this register are different for configuring general-purpose outputs than for configuring waveform generator and PWM outputs.

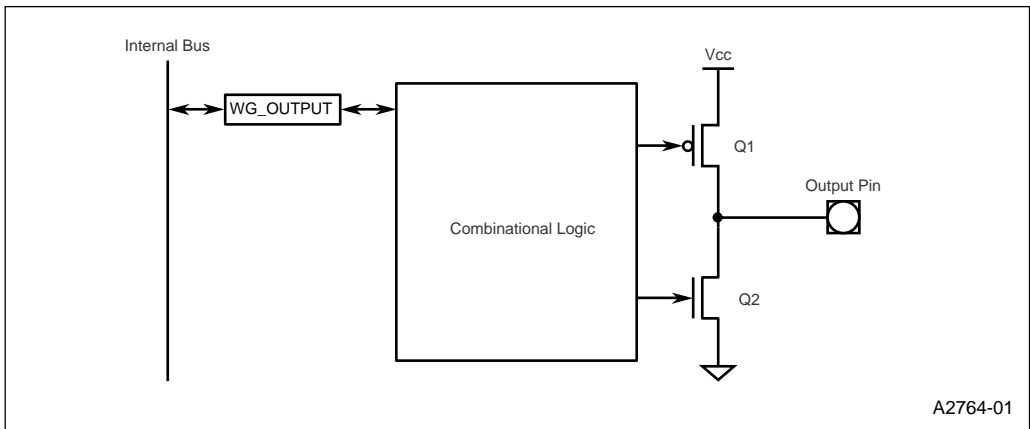


Figure 6-4. Output-only Port

WG_OUTPUT (Port 6) Address: 1FC0H
Reset State: 0000H

The port 6 output configuration (WG_OUTPUT) register controls port 6 functions. If you are using port 6 for general-purpose outputs, write C0H (for active-high outputs) or 00H (for active-low outputs) to the high byte of WG_OUTPUT, and write the desired pin values to the low byte. If you are using port 6 for waveform generator or PWM outputs, please refer to Chapter 9, "Waveform Generator" or Chapter 10, "Pulse-width Modulator" for a description of the WG_OUTPUT register functions.

15								8			
OP1	OP0	—	M7	M6	M5:4	M3:2	M1:0				
7								0			
D7	D6	D5	D4	D3	D2	D1	D0				

Bit Number	Bit Mnemonic	Function
15:14	OP1:0	Output Polarity These bits select the output polarity of the port 6 pins. 0 = active-low outputs 1 = active-high outputs
13	—	Reserved; for compatibility with future devices, write zero to this bit.
12:8	M7:0	Mode These bits select either the peripheral function or general-purpose output function of the port 6 pins. Clear these bits for general-purpose output.

Figure 6-5. Port 6 Output Configuration (WG_OUTPUT) Register

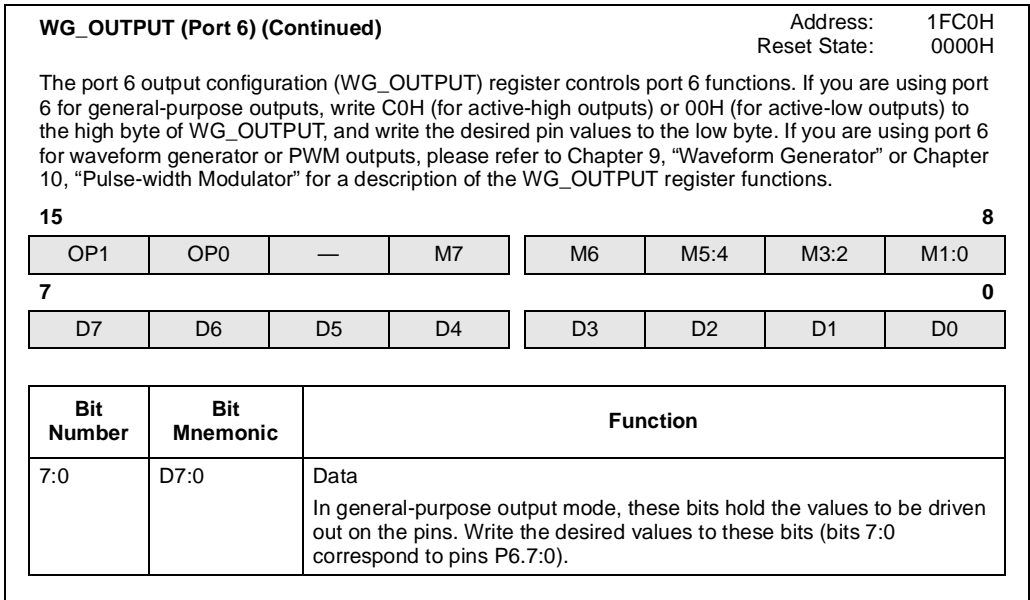


Figure 6-5. Port 6 Output Configuration (WG_OUTPUT) Register (Continued)



7

Serial I/O (SIO) Port

CHAPTER 7 SERIAL I/O (SIO) PORT

A serial input/output (SIO) port provides a means for the system to communicate with external devices. The 8XC196MH device has a two-channel serial I/O port that shares pins with ports 1 and 2. (The 8XC196MC and 8XC196MD devices do **not** have serial I/O ports.) This chapter describes the SIO port and explains how to configure it. Chapter 6, “I/O Ports,” explains how to configure the port pins for their special functions. Refer to Appendix B for details about the signals discussed in this chapter.

7.1 SERIAL I/O (SIO) PORT FUNCTIONAL OVERVIEW

The serial I/O port (Figure 7-1) is an asynchronous/synchronous port that includes a universal asynchronous receiver and transmitter (UART). The UART has two synchronous modes (modes 0 and 4) and three asynchronous modes (modes 1, 2, and 3) for both transmission and reception.

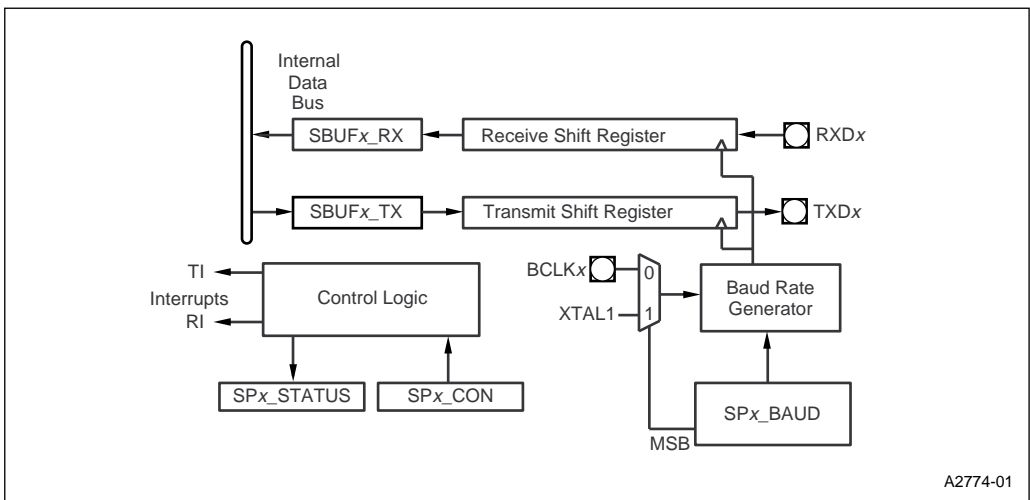


Figure 7-1. SIO Block Diagram

The serial port receives data into the receive buffer (SBUF_x_RX) and transmits data from the port through the transmit buffer (SBUF_x_RX). The transmit and receive buffers are separate registers, permitting simultaneous reads and writes to both. These buffers support continuous transmissions and allow reception of a second byte before the first byte has been read.

An independent, 15-bit baud-rate generator controls the baud rate of the serial port. Either XTAL1 or BCLK_x can provide the clock signal for modes 0–3. In mode 4, the internal shift clock is output on SCLK_x# or an external shift clock is input on SCLK_x# (in which case the baud-rate generator is not used). The baud-rate register (SP_x_BAUD) selects the clock source and the baud rate. The serial port control register (SP_x_CON) register controls whether SCLK_x# outputs the internal shift clock or inputs an external shift clock.

7.2 SERIAL I/O PORT SIGNALS AND REGISTERS

Table 7-1 describes the SIO signals and Table 7-2 describes the control and status registers.

Table 7-1. Serial Port Signals

Port Pin	Serial Port Signal	Serial Port Signal Type	Description
P1.0 P1.2	TXD0 TXD1	O	Transmit Serial Data. In modes 1, 2, 3, and 4, TXD _x transmits serial port output data. In mode 0, it is the serial clock output.
P1.1 P1.3	RXD0 RXD1	I/O	Receive Serial Data. In modes 1, 2, 3, and 4, RXD _x receives serial port input data. In mode 0, it functions as an input or an open-drain output for data.
P2.1 P2.7	SCLK0#/BCLK0 SCLK1#/BCLK1	I/O	Baud Clock. BCLK _x can provide an external clock source for the baud-rate generator input. Shift Clock. In mode 4 only, SCLK _x # are bidirectional shift clock signals that synchronize the serial data transfer. The DIR bit in the SP_CON register controls the direction of SCLK _x #: DIR = 0 causes SCLK _x # to output the internal shift clock. DIR = 1 allows an external shift clock to be input on SCLK _x #.

Table 7-2. Serial Port Control and Status Registers

Mnemonic	Address	Description
INT_MASK1	0013H	Interrupt Mask 1 Setting the T1x bit enables the transmit interrupt; clearing the bit disables (masks) the interrupt. Setting the R1x bit enables the receive interrupt; clearing the bit disables (masks) the interrupt. Setting the SPE bit enables the serial port receive error interrupt.

Table 7-2. Serial Port Control and Status Registers (Continued)

Mnemonic	Address	Description
INT_PEND1	0012H	<p>Interrupt Pending 1</p> <p>When set, the T1x bit indicates a pending transmit interrupt.</p> <p>When set, the R1x bit indicates a pending receive interrupt.</p> <p>When set, the SPE bit indicates a pending serial port receive error interrupt. You must read the PI_PEND register to determine which channel generated the interrupt.</p>
P1_DIR	1F9BH	<p>Port 1 Direction</p> <p>This register selects the direction of each port 1 pin. Set P1_DIR.1 and P1_DIR.3 to configure RXD0 (P1.1) and RXD1 (P1.3) as high-impedance inputs/open-drain outputs, and clear P1_DIR.0 and P1_DIR.2 to configure TXD0 (P1.0) and TXD1 (P1.2) as complementary outputs.</p>
P1_MODE	1F99H	<p>Port 1 Mode</p> <p>This register selects either the general-purpose input/output function or the peripheral function for each pin of port 1. Set P1_MODE.3:0 to configure TXD0 (P1.0), TXD1 (P1.2), RXD0 (P1.1), and RXD1 (P1.3) for the SIO port.</p>
P1_PIN	1F9FH	<p>Port 1 Pin State</p> <p>Four bits of this register contain the values of the TXD0 (P1.0), RXD0 (P1.1), TXD1 (P1.2), and RXD1 (P1.3) pins. Read P1_PIN to determine the current value of the TXDx and RXDx pins.</p>
P1_REG	1F9DH	<p>Port 1 Output Data</p> <p>This register holds data to be driven out on the pins of port 1. Set P1_REG.1 and P1_REG.3 for the RXDx pins. Set P1_REG.0 and P1_REG.2 for the TXDx pins.</p>
P2_DIR	1FD2H	<p>Port 2 Direction</p> <p>This register selects the direction of each port 2 pin. To use BCLKx as the baud-rate generator clock source for modes 0–3, or to use mode 4 (in which case SCLKx# is the shift clock), set P2_DIR.1 and P2_DIR.7 to configure SCLK0#/BCLK0 (P2.1) and SCLK1#/BCLK1 (P2.7) as high-impedance inputs/open-drain outputs.</p>
P2_MODE	1FD0H	<p>Port 2 Mode</p> <p>This register selects either the general-purpose input/output function or the peripheral function for each pin of port 2. Set P2_MODE.1 and P2_MODE.7 to configure SCLK0#/BCLK0 (P2.1) and SCLK1#/BCLK1 (P2.7) for the SIO port.</p>
P2_PIN	1FD6H	<p>Port 2 Pin State</p> <p>Two bits of this register contain the values of the SCLK0#/BCLK0 (P2.1) and SCLK1#/BCLK1 (P2.7) pins. Read P2_PIN to determine the current value of the pins.</p>
P2_REG	1FD4H	<p>Port 2 Output Data</p> <p>This register holds data to be driven out on the pins of port 2. Set P2_REG.1 and P2_REG.7 for the SCLK0#/BCLK0 (P2.1) and SCLK1#/BCLK1 (P2.7) pins.</p>

Table 7-2. Serial Port Control and Status Registers (Continued)

Mnemonic	Address	Description
PI_MASK	1FBCH	Peripheral Interrupt Mask This register enables and disables multiplexed peripheral interrupts. Setting an SPx bit enables a serial port receive error interrupt; clearing the bit disables (masks) the interrupt.
PI_PEND	1FBEH	Peripheral Interrupt Pending This register indicates pending multiplexed peripheral interrupts. When set, an SPx bit indicates a pending serial port receive error interrupt.
SBUF0_RX SBUF1_RX	1F80H 1F88H	Serial Port x Receive Buffer This register contains data received from the serial port.
SBUF0_TX SBUF1_TX	1F82H 1F8AH	Serial Port x Transmit Buffer This register contains data that is ready for transmission. In modes 1, 2, and 3, writing to SBUFx_TX starts a transmission. In mode 0, writing to SBUFx_TX starts a transmission only if the receiver is disabled (SPx_CON.3 = 0).
SP0_BAUD SP1_BAUD	1F84H,1F85H 1F8CH,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.
SP0_CON SP1_CON	1F83H 1F8BH	Serial Port x Control This register selects the communications mode and enables or disables the receiver, parity checking, and ninth-bit data transmissions. The TB8 bit is cleared after each transmission. For mode 4, this register also selects the direction (input or output) of the SCLKx# signal.
SP0_STATUS SP1_STATUS	1F81H 1F89H	Serial Port x Status This register contains the serial port status bits. It has status bits for receive overrun errors (OE), transmit buffer empty (TXE), framing errors (FE), transmit interrupt (TI), receive interrupt (RI), and received parity error (RPE) or received bit 8 (RB8). Reading SPx_STATUS clears all bits except TXE; writing a byte to SBUFx_TX clears the TXE bit.

7.3 SERIAL PORT MODES

The serial port has both synchronous and asynchronous operating modes for transmission and reception. This section describes the operation of each mode.

7.3.1 Synchronous Modes (Modes 0 and 4)

The 8XC196MH serial port has two synchronous modes, mode 0 and mode 4. Mode 0 is the synchronous mode available on all the 8XC196 devices that have serial ports. Mode 4 is an enhanced, full-duplex synchronous mode.

7.3.1.1 Mode 0

The most common use of mode 0 is to expand the I/O capability of the device with shift registers (see Figure 7-2). In this mode, 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 7-3 shows a diagram of the relative timing of these signals. Note that only mode 0 uses RXD_x as an open-drain output.

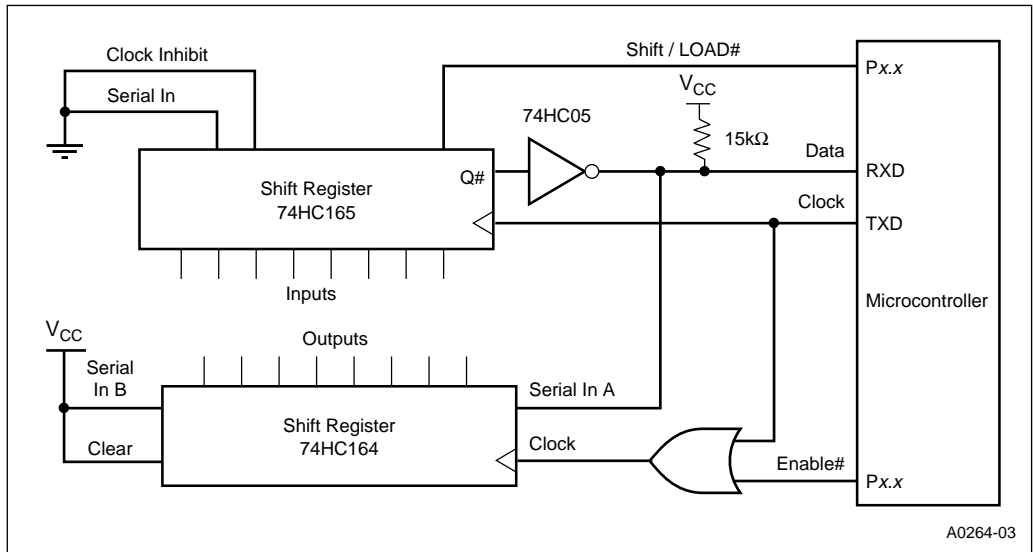


Figure 7-2. Typical Shift Register Circuit for Mode 0

In mode 0, RXD_x must be enabled for receptions and disabled for transmissions. (See “Programming the Control Register” on page 7-10.) When RXD_x is enabled, either a rising edge on the RXD_x input or clearing the receive interrupt (RI) flag in SP_x_STATUS starts a reception. When RXD_x is disabled, writing to SBUF_x_TX starts a transmission.

Disabling RXD_x stops a reception in progress and inhibits further receptions. To avoid a partial or undesired complete reception, disable RXD_x before clearing the RI flag in SP_x_STATUS. This can be handled in an interrupt environment by using software flags or in straight-line code by using the interrupt pending register to signal the completion of a reception.

During a reception, the RI flag in SPx_STATUS is set after the stop bit is sampled. The RLx pending bit in the interrupt pending register is set immediately before the RI flag is set. During a transmission, the TI flag is set immediately after the end of the last (eighth) data bit is transmitted. The TLx pending bit in the interrupt pending register is generated when the TI flag in SPx_STATUS is set.

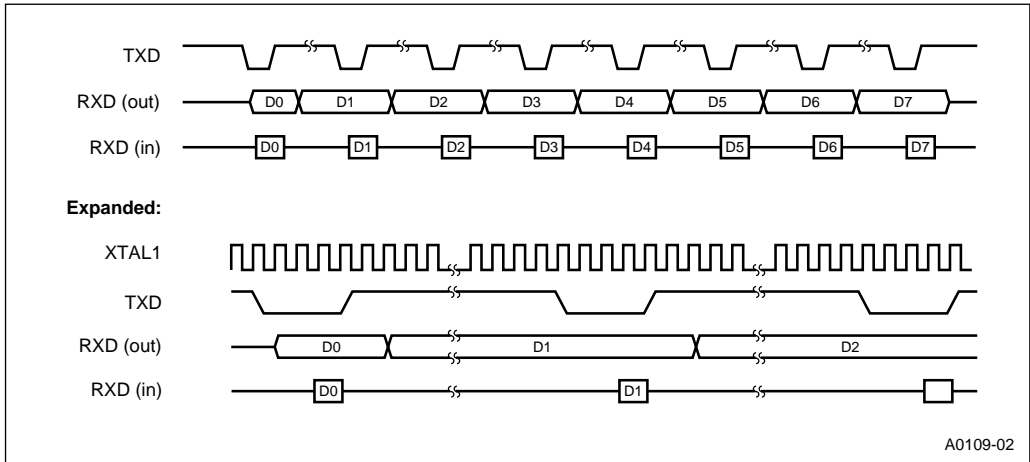


Figure 7-3. Mode 0 Timing

7.3.1.2 Mode 4

Mode 4 is an enhanced synchronous mode that is similar to mode 0 in many ways, but there are three main differences:

- In mode 0, $TXDx$ inputs or outputs the clock signal and $RXDx$ transmits or receives data. In mode 4, $TXDx$ transmits data, $RXDx$ receives data, and the $SCLKx\#$ pin inputs or outputs the clock signal.
- For mode 4, a direction bit (DIR) was added to the SPx_CON register. This bit controls whether $SCLKx\#$ outputs the internal shift clock or inputs an external shift clock.
- In mode 0, $RXDx$ must be enabled to start a transmission (because it must transmit the data). In mode 4, $TXDx$ transmits the data, so the $RXDx$ status is unrelated to transmissions. Writing to $SBUFx_TX$ starts a transmission regardless of whether $RXDx$ is enabled.

In mode 4, $SCLKx\#$ outputs a set of eight clock pulses while $TXDx$ transmits data, or $SCLKx\#$ accepts a set of eight clock pulses while $RXDx$ receives data. Data is transferred eight bits at a time with the least-significant bit first. When $SCLKx\#$ is in output mode, its timing is identical to that of $TXDx$ shown in Figure 7-3. When $SCLKx\#$ is in input mode, the input shift clock signal is internally synchronized with the internal clock.

In mode 4, writing to `SBUFx_TX` starts a transmission regardless of whether `RXDx` is enabled. However, `RXDx` must be enabled to allow a reception. If `RXDx` is enabled, either a rising edge on the `RXDx` input or clearing the receive interrupt (RI) flag starts a reception.

Disabling `RXDx` stops a reception in progress and inhibits further receptions. To avoid a partial or undesired complete reception, disable `RXDx` before clearing the RI flag. This can be handled in an interrupt environment by using software flags or in straight-line code by using the interrupt pending register to signal the completion of a reception.

During a reception, the RI flag is set one clock bit time after the last data bit is received. The receive interrupt signal is generated immediately before the RI flag is set. During a transmission, the TI flag is set immediately after the end of the last (eighth) data bit is transmitted. The transmit interrupt signal is generated when the TI flag is set.

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

Modes 1, 2, and 3 are full-duplex serial transmit/receive modes, meaning that they can transmit and receive data simultaneously. Mode 1 is the standard 8-bit, asynchronous mode used for normal serial communications. Modes 2 and 3 are 9-bit asynchronous modes typically used for interprocessor communications (see “Multiprocessor Communications” on page 7-9). In mode 2, the serial port sets an interrupt pending bit only if the ninth data bit is set. In mode 3, the serial port always sets an interrupt pending bit upon completion of a data transmission or reception.

When the serial port is configured for mode 1, 2, or 3, writing to `SBUFx_TX` causes the serial port to start transmitting data. New data placed in `SBUFx_TX` is transmitted only after the stop bit of the previous data has been sent. A falling edge on the `RXDx` input causes the serial port to begin receiving data if `RXDx` is enabled. Disabling `RXDx` stops a reception in progress and inhibits further receptions. (See “Programming the Control Register” on page 7-10.)

7.3.2.1 Mode 1

Mode 1 is the standard asynchronous communications mode. The data frame used in this mode (Figure 7-4) consists of ten bits: a start bit (0), eight data bits (LSB first), and a stop bit (1). If parity is enabled, a parity bit is sent instead of the eighth data bit, and parity is checked on reception.

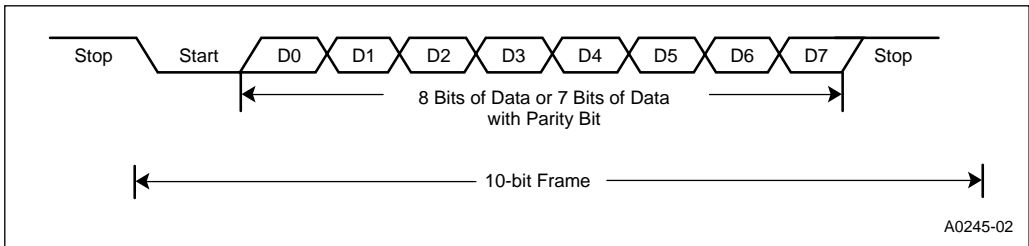


Figure 7-4. Serial Port Frames for Mode 1

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

The transmit interrupt (TI) and receive interrupt (RI) flags in SP_x_STATUS are set to indicate completed operations. During a reception, both the RI flag and the RL_x interrupt pending bit are set just before the end of the stop bit. During a transmission, both the TI flag and the TL_x interrupt pending bit are set at the beginning of the stop bit. The next byte cannot be sent until the stop bit is sent.

Use caution when connecting more than two devices with the serial port in half-duplex (i.e., with one wire for transmit and receive). 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 devices listening on the link.

7.3.2.2 Mode 2

Mode 2 is the asynchronous, ninth-bit recognition mode. This mode is commonly used with mode 3 for multiprocessor communications. Figure 7-5 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, setting the TB8 bit in the SP_x_CON register before writing to $SBUF_x_TX$ sets the ninth transmission bit. The hardware clears the TB8 bit after every transmission, so it must be set (if desired) before each write to $SBUF_x_TX$. During receptions, the RI flag and RL_x interrupt pending bit are set only if the TB8 bit is set. This provides an easy way to have selective reception on a data link. (See “Multiprocessor Communications” on page 7-9.) Parity cannot be enabled in this mode.

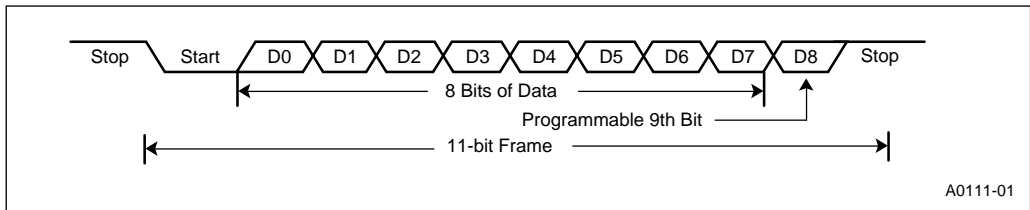


Figure 7-5. Serial Port Frames in Mode 2 and 3

7.3.2.3 Mode 3

Mode 3 is the asynchronous, ninth-bit mode. The data frame for this mode is identical to that of mode 2. 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, and the ninth data bit is written to SP_x_CON.4 (TB8). In mode 3, a reception always sets the RL_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. If parity is enabled, then bit 7 (RB8) is the received parity error (RPE) flag.

7.3.2.4 Mode 2 and 3 Timings

Operation in modes 2 and 3 is similar to mode 1 operation. The only difference is that the data consists of 9 bits, so 11-bit packages are transmitted and received. During a reception, the RI flag and the RL_x interrupt pending bit are set just after the end of the stop bit. During a transmission, the TI flag and the TL_x interrupt pending bit are set at the beginning of the stop bit. The ninth bit can be used for parity or multiprocessor communications.

7.3.2.5 Multiprocessor Communications

Modes 2 and 3 are provided for multiprocessor communications. In mode 2, the serial port sets the RL_x interrupt pending bit only when the ninth data bit is set. In mode 3, the serial port sets the RL_x interrupt pending bit regardless of the value of the ninth bit. The ninth bit is always set in address frames and always cleared in data frames.

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. Because the ninth bit is set, 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, while the slaves that are not addressed remain in mode 2 and are not interrupted.

7.4 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.

7.4.1 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 7-1 on page 7-2 lists the pins associated with the serial port. Table 7-2 on page 7-2 lists the port configuration registers, and Chapter 6, "I/O Ports," explains how to configure the pins.

7.4.2 Programming the Control Register

The SP_x_CON register (Figure 7-6) selects the communication mode and enables or disables the receiver, parity checking, and nine-bit data transmissions. Selecting a new mode resets the serial I/O port and aborts any transmission or reception in progress on the channel.

SP_x_CON x = 0–1 (8XC196MH)		Address: 1F83H, 1F8BH	
		Reset State: 00H	
The serial port control (SP _x _CON) register selects the communications mode and enables or disables the receiver, parity checking, and nine-bit data transmission.			
7		0	
8XC196MH	M2	DIR	PAR
		TB8	REN
			PEN
			M1
			M0

Bit Number	Bit Mnemonic	Function
7	M2	See description for bits 0 and 1.
6	DIR	Synchronous Clock Direction This bit determines the direction of the clock during synchronous mode. 0 = output 1 = input
5	PAR	Parity Selection Bit This bit selects even or odd parity. 0 = even parity 1 = odd parity
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 it must be set before SBUF _x _TX is written. When parity is enabled (SP _x _CON.2 = 1), this bit takes on the even parity value.

Figure 7-6. Serial Port Control (SP_x_CON) Register

<p>SP_x_CON (Continued) x = 0–1 (8XC196MH)</p> <p>The serial port control (SP_x_CON) register selects the communications mode and enables or disables the receiver, parity checking, and nine-bit data transmission.</p>	<p>Address: 1F83H, 1F8BH Reset State: 00H</p>																									
<div style="display: flex; justify-content: space-between; width: 100%;"> 7 0 </div> <div style="display: flex; justify-content: space-around; margin-top: 5px;"> M2 DIR PAR TB8 REN PEN M1 M0 </div>																										
<p>8XC196MH</p>																										
Bit Number	Bit Mnemonic	Function																								
3	REN	<p>Receive Enable</p> <p>Setting this bit enables receptions. When this bit is set, a falling edge on the RXD_x pin starts a reception in mode 1, 2, or 3. In mode 0, this bit must be clear for transmission to begin and must be set for reception to begin.</p> <p>Clearing this bit stops a reception in progress and inhibits further receptions. To avoid a partial or undesired reception, clear this bit before clearing the RI flag in SP_x_STATUS. This can be handled in an interrupt environment by using software flags or in straight-line code by using the interrupt pending register to signal the completion of a reception.</p>																								
2	PEN	<p>Parity Enable</p> <p>In modes 1 and 3, setting this bit enables the parity function. This bit must be cleared if mode 2 is used. When this bit is set, TB8 takes the parity value on transmissions and SP_x_STATUS.7 becomes the receive parity error bit.</p>																								
1:0	M1:0	<p>Mode Selection</p> <p>These bits along with bit 7 select the communications mode.</p> <table border="0" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding-right: 10px;">M2</td> <td style="padding-right: 10px;">M1</td> <td style="padding-right: 10px;">M0</td> <td></td> </tr> <tr> <td>0</td> <td>0</td> <td>0</td> <td>synchronous mode 0</td> </tr> <tr> <td>X</td> <td>0</td> <td>1</td> <td>mode 1</td> </tr> <tr> <td>X</td> <td>1</td> <td>0</td> <td>mode 2</td> </tr> <tr> <td>X</td> <td>1</td> <td>1</td> <td>mode 3</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>synchronous mode 4</td> </tr> </table>	M2	M1	M0		0	0	0	synchronous mode 0	X	0	1	mode 1	X	1	0	mode 2	X	1	1	mode 3	1	0	0	synchronous mode 4
M2	M1	M0																								
0	0	0	synchronous mode 0																							
X	0	1	mode 1																							
X	1	0	mode 2																							
X	1	1	mode 3																							
1	0	0	synchronous mode 4																							

Figure 7-6. Serial Port Control (SP_x_CON) Register (Continued)

7.4.3 Programming the Baud Rate and Clock Source

The SP_x_BAUD register (Figure 7-7) selects the clock input for the baud-rate generator and defines the baud rate for all serial I/O modes. (For mode 4 with SCLK_x# configured for input, the baud-rate generator is not used.) This register acts as a control register during write operations and as a down-counter monitor during read operations.

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 the SP_x_STATUS register to ensure that the reception or transmission is complete.

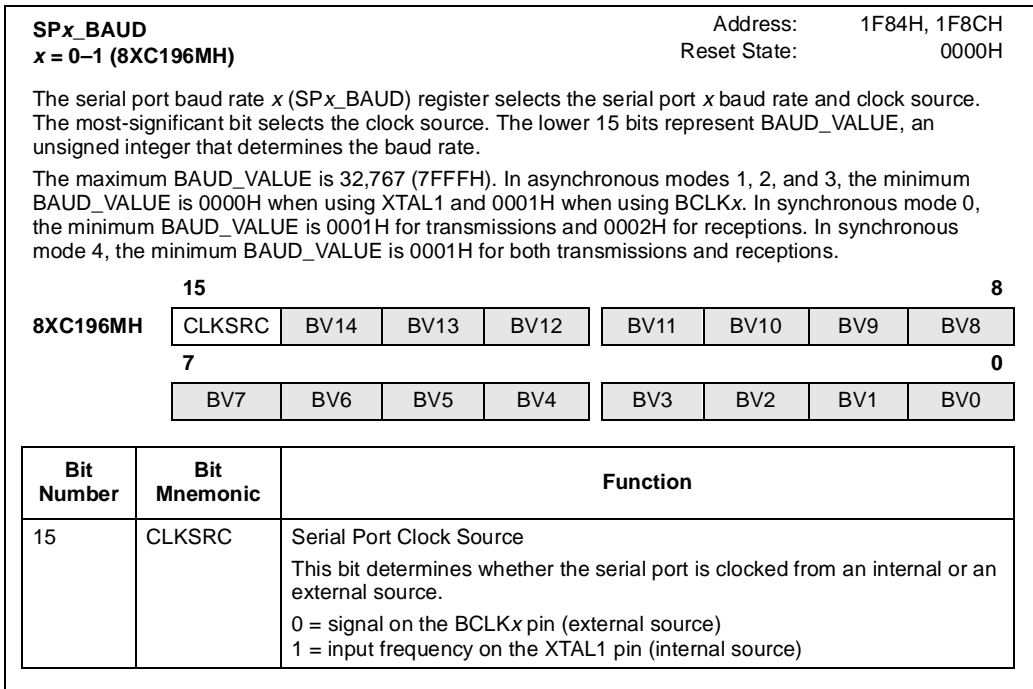


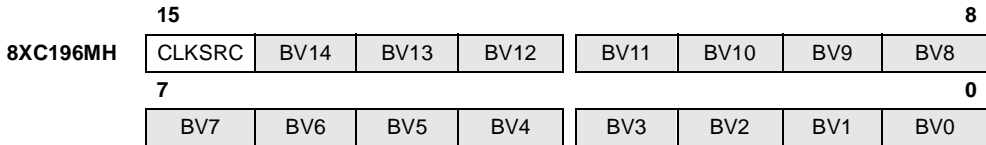
Figure 7-7. Serial Port x Baud Rate (SP_x_BAUD) Register

SP_x_BAUD (Continued)
x = 0–1 (8XC196MH)

Address: 1F84H, 1F8CH
 Reset State: 0000H

The serial port baud rate x (SP_x_BAUD) register selects the serial port x baud rate and clock source. 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 XTAL1 and 0001H when using BCLK_x. In synchronous mode 0, the minimum BAUD_VALUE is 0001H for transmissions and 0002H for receptions. In synchronous mode 4, the minimum BAUD_VALUE is 0001H for both transmissions and receptions.



Bit Number	Bit Mnemonic	Function
14:0	BV14:0	<p>These bits constitute the BAUD_VALUE.</p> <p>Use the following equations to determine the BAUD_VALUE for a given baud rate.</p> <p>Synchronous mode 0:†</p> $\text{BAUD_VALUE} = \frac{F_{\text{XTAL1}}}{\text{Baud Rate} \times 2} - 1 \quad \text{or} \quad \frac{\text{BCLK}_x}{\text{Baud Rate}}$ <p>Asynchronous modes 1, 2, and 3:</p> $\text{BAUD_VALUE} = \frac{F_{\text{XTAL1}}}{\text{Baud Rate} \times 16} - 1 \quad \text{or} \quad \frac{\text{BCLK}_x}{\text{Baud Rate} \times 8}$ <p>Synchronous mode 4 (SCLK_x# output):</p> $\text{BAUD_VALUE} = \frac{F_{\text{XTAL1}}}{\text{Baud Rate} \times 4} - 1$ <p>† For mode 0 receptions, the BAUD_VALUE must be 0002H or greater. Otherwise, the resulting data in the receive shift register will be incorrect.</p>

Figure 7-7. Serial Port x Baud Rate (SP_x_BAUD) Register (Continued)

CAUTION

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 XTAL1 at 16 MHz, the maximum baud rates are 2.76 Mbaud (SP_x_BAUD = 8002H or 0002H) for mode 0 and 1.0 Mbaud for modes 1, 2, and 3. Table 7-3 shows the SP_x_BAUD values for common baud rates when using a 16 MHz XTAL1 clock input. Because of rounding, the BAUD_VALUE formula is not exact and the resulting baud rate is slightly different than desired. Table 7-3 shows 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 7-3. SP_x_BAUD Values When Using XTAL1 at 16 MHz

Baud Rate	SP _x _BAUD Register Value [†]		% Error	
	Mode 0	Mode 1, 2, 3	Mode 0, 4	Mode 1, 2, 3
9600	8340H	8067H	0.04	0.16
4800	8682H	80CFH	0.02	0.16
2400	8D04H	81A0H	0.01	0.08
1200	9A0AH	8340H	0	0.04
300	E82BH	8D04H	0	0.01

[†] Bit 15 is always set when XTAL1 is selected as the clock source for the baud-rate generator.

7.4.4 Enabling the Serial Port Interrupts

Each serial port channel has both a transmit interrupt (Tl_x) and a receive interrupt (Rl_x). Each channel can also generate a serial port receive error interrupt (SP_x). To enable an interrupt, set the corresponding mask bit in the interrupt mask register or peripheral interrupt mask register (see Table 7-2 on page 7-2) and execute the EI instruction to globally enable servicing of interrupts. See Chapter 5, “Standard and PTS Interrupts,” for more information about interrupts.

7.4.5 Determining Serial Port Status

You can read the SP_x_STATUS register (Figure 7-8) to determine the status of the serial port. Reading SP_x_STATUS **clears all bits** except TXE. For this reason, we recommend that you 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, executing a bit-test instruction clears the flags, so any subsequent bit-test instructions will return false values. You can also read the interrupt pending register or peripheral interrupt pending register (see Table 7-2 on page 7-2) to determine the status of the serial port interrupts.

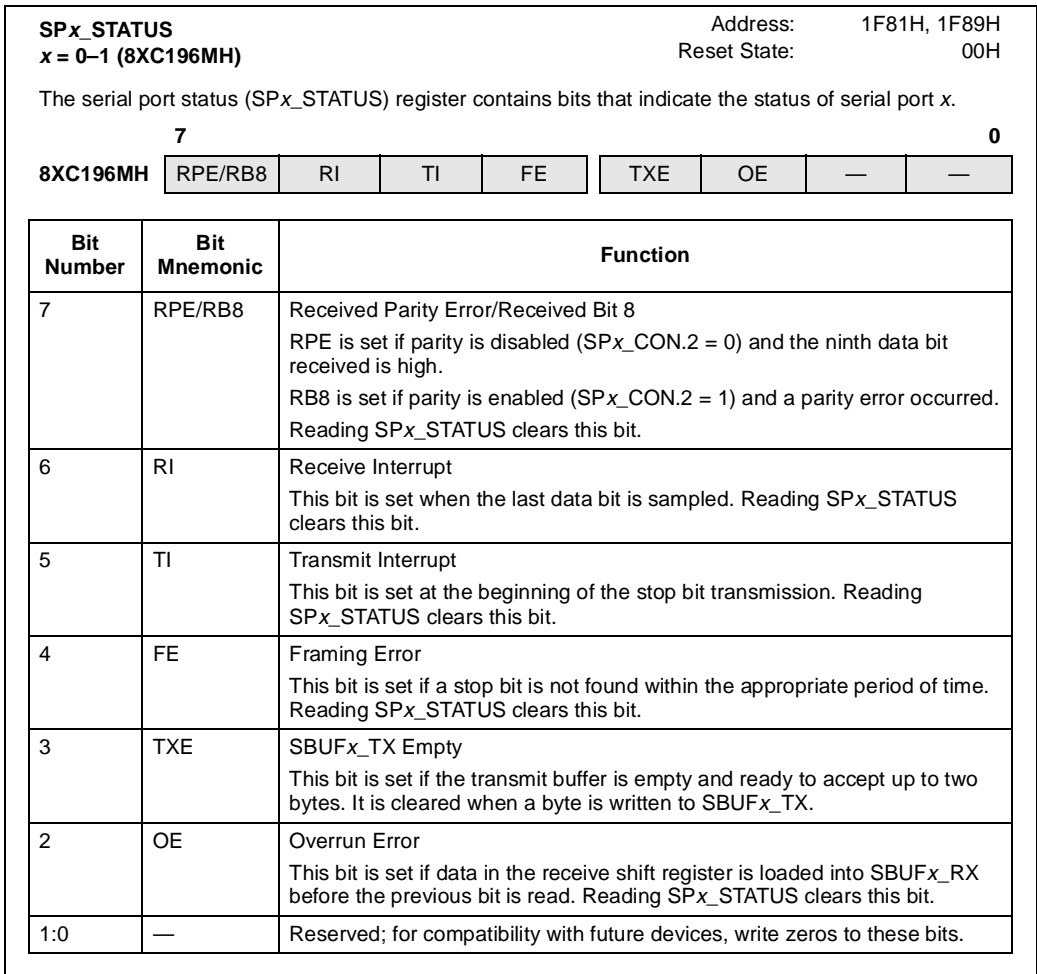


Figure 7-8. Serial Port Status (SP_x_STATUS) Register

The receiver checks for a valid stop bit. Unless a stop bit is found within the appropriate time, the framing error (FE) bit in the SP_x_STATUS register is set. When the stop bit is detected, the data in the receive shift register is loaded into $SBUF_x_RX$ and the receive interrupt (RI) flag is set. If this happens before the previous byte in $SBUF_x_RX$ is read, the overrun error (OE) bit is set. $SBUF_x_RX$ always contains the latest byte received; it is never a combination of the last two bytes.

The receive interrupt (RI) flag indicates whether an incoming data byte has been received. The transmit interrupt (TI) flag indicates whether a data byte has finished transmitting. These flags also set the corresponding bits in the interrupt pending register. A reception or transmission sets the RI or TI flag in SP_x_STATUS and the corresponding interrupt pending bit. However, a software write to the RI or TI flag in SP_x_STATUS has no effect on the interrupt pending bits and does not cause an interrupt. Similarly, reading SP_x_STATUS clears the RI and TI flags, but does not clear the corresponding interrupt pending bits. The RI and TI flags in the SP_x_STATUS and the corresponding interrupt pending bits can be set even if the RI_x and TI_x interrupts are masked.

The transmitter empty (TXE) bit is set if $SBUF_x_TX$ and its buffer are empty and ready to accept up to two bytes. TXE is cleared as soon as a byte is written to $SBUF_x_TX$. One byte may be written if TI alone is set. By definition, if TXE has just been set, a transmission has completed and TI is set.

The received parity error (RPE) flag or the received bit 8 (RB8) flag applies for parity enabled or disabled, respectively. If parity is enabled, RPE is set if a parity error is detected. If parity is disabled, RB8 is the ninth data bit received in modes 2 and 3.



8

Frequency Generator

CHAPTER 8 FREQUENCY GENERATOR

The 8XC196MD has a peripheral not found on other 8XC196Mx devices — the frequency generator. This peripheral produces a waveform with a fixed duty cycle (50%) and a programmable frequency (ranging from 4 kHz to 1 MHz with a 16-MHz input clock). One application for the frequency generator is to drive an infrared LED to transmit remote control data and control signals.

This chapter describes the frequency generator and explains how to configure it. For detailed descriptions of the signals discussed in this chapter, refer to Appendix B, “Signal Descriptions.” For additional information and application examples, consult AP-483, *Application Examples Using the 8XC196MC/MD Microcontroller* (order number 272282).

8.1 FUNCTIONAL OVERVIEW

The frequency generator (Figure 8-1) has a frequency register, a count register, and an output signal. The output signal shares pin P7.7, so you must configure the pin for its frequency generator output function.

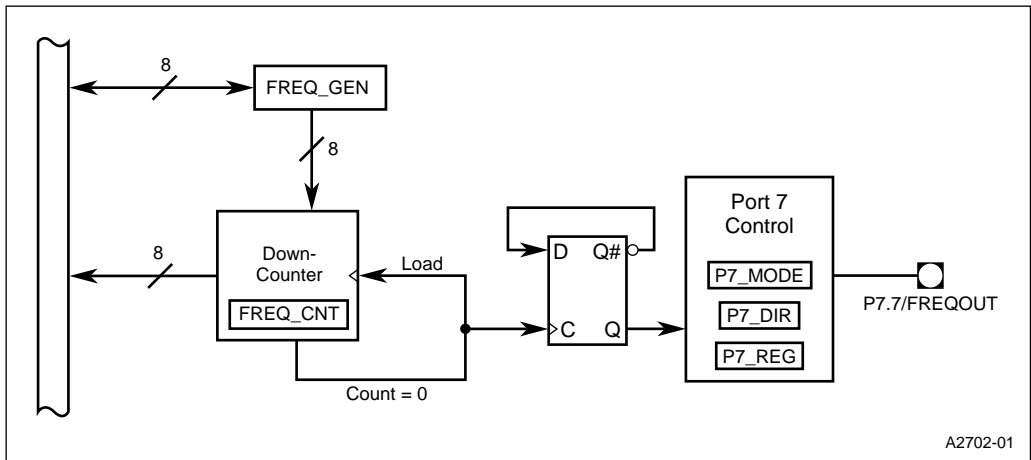


Figure 8-1. Frequency Generator Block Diagram

The frequency register (FREQ_GEN) controls the output frequency. The frequency generator loads the FREQ_GEN value into the counter. The counter counts down until it reaches zero, at which time the value is reloaded from the FREQ_GEN register. Each load toggles the D flip-flop, producing the 50% duty cycle output. The count register (FREQ_CNT) reflects the current value of the down-counter. Table 8-1 describes the frequency generator's output signal and Table 8-2 describes the control and status registers.

Table 8-1. Frequency Generator Signal

Port Pin	Frequency Generator Signal	Frequency Generator Signal Type	Description
P7.7	FREQOUT	O	Frequency Generator Output This signal carries the output of the frequency generator.

Table 8-2. Frequency Generator Control and Status Registers

Mnemonic	Address	Description
FREQ_GEN	1FB8H	Frequency The frequency register holds a programmed value that determines the output frequency. This value is reloaded into the down-counter each time the counter reaches 0.
FREQ_CNT	1FBAH	Count The read-only counter register reflects the current counter value.
P7_DIR	1FD3H	Port 7 Direction Bit 7 controls the direction of P7.7/FREQOUT. Clear this bit to configure FREQOUT as a complementary output.
P7_MODE	1FD1H	Port 7 Mode Bit 7 controls the mode (general-purpose I/O or special-function signal) of P7.7/FREQOUT. Set this bit to configure the pin for its FREQOUT function.
P7_PIN	1FD7H	Port 7 Input Bit 7 reflects the current state of P7.7/FREQOUT, regardless of its configuration.
P7_REG	1FD5H	Port 7 Data Output Bit 7 contains data to be driven out by P7.7/FREQOUT in general-purpose I/O mode. In special-function mode, the frequency generator controls the pin.

8.2 PROGRAMMING THE FREQUENCY GENERATOR

This section explains how to configure the frequency generator and determine its status.

8.2.1 Configuring the Output

The frequency generator’s output is multiplexed with P7.7, so you must configure it as a special-function output signal. To do so, follow this sequence:

1. Clear bit 7 of P7_DIR.
2. Set bit 7 of P7_MODE.
3. Clear bit 7 of P7_REG.

Refer to Chapter 6, “I/O Ports,” for additional information about configuring port pins.

8.2.2 Programming the Frequency

Program the frequency register (Figure 8-2) to control the frequency of the output.

FREQ_GEN (8XC196MD)	Address: 1FB8H Reset State: 00H
The frequency (FREQ_GEN) register holds a programmed value that specifies the output frequency. This value is reloaded into the down-counter each time the counter reaches 0.	
7	0
8XC196MD	Output Frequency

Bit Number	Function
7:0	Output Frequency Use the following formula to calculate the FREQ value for the desired output frequency and write this value to the frequency register. $FREQ = \frac{F_{XTAL1}}{16 \times FREQ_OUT} - 1$ where: FREQ = 8-bit value to load into FREQ_GEN register F _{XTAL1} = input frequency on XTAL1 pin, in MHz FREQ_OUT = output frequency on FREQOUT pin, in MHz

Figure 8-2. Frequency (FREQ_GEN) Register

8.2.3 Determining the Current Value of the Down-counter

You can read the `FREQ_CNT` register (Figure 8-3) to determine the current value of the down-counter.

FREQ_CNT (8XC196MD)		Address: 1FBAH
		Reset State: 00H
Read the frequency generator count (<code>FREQ_CNT</code>) register to determine the current value of the down-counter.		
7		0
8XC196MD	Count	
Bit Number	Function	
7:0	Count This register contains the current down-counter value.	

Figure 8-3. Frequency Generator Count (`FREQ_CNT`) Register

8.3 APPLICATION EXAMPLE

One application for the frequency generator is to drive an infrared LED to transmit remote control data and control signals (Figure 8-4). In this example, the frequency generator is configured with a 40-kHz frequency and is switched on and off by writing to bit 7 of the `P7_MODE` register. Information is transmitted serially. Zero is represented by a one-millisecond carrier burst followed by a one-millisecond pause; one is represented by a two-millisecond carrier burst followed by a two-millisecond pause (Figure 8-5). A photodiode receives the light pulses, and a high-pass filter rejects low-frequency ambient light and allows the 40-kHz carrier to pass through. This carrier is amplified and detected to reproduce the original pulse sequence.

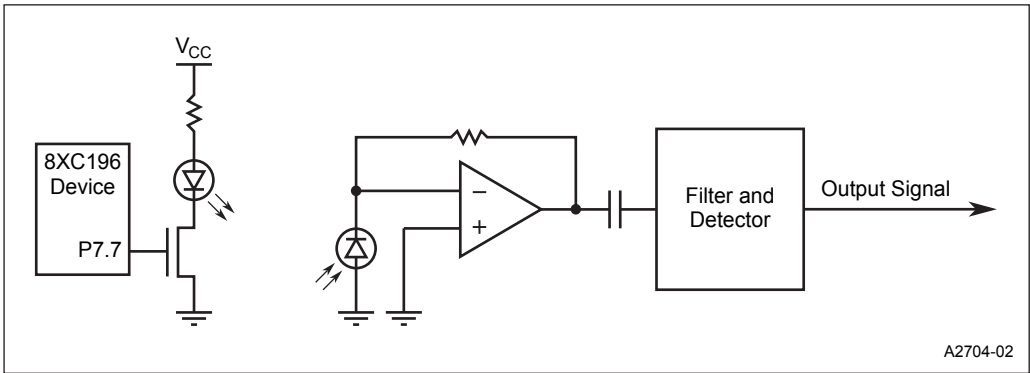


Figure 8-4. Infrared Remote Control Application Block Diagram

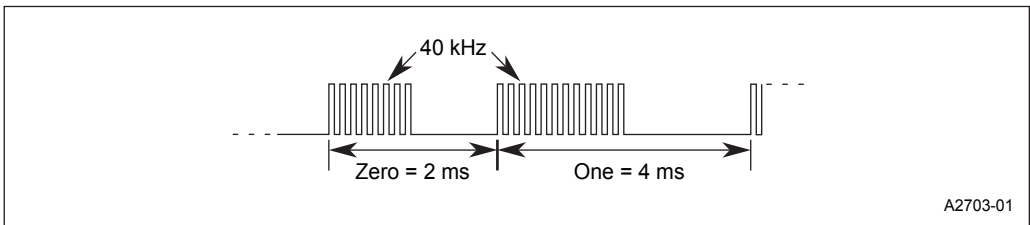


Figure 8-5. Data Encoding Example

This program example was designed to run on an 8XC196MD demo board. It uses an EPA timer (timer 1) and compare channel (COMP3) to provide the timebase for the ones and zeros.

```

$debug
$nolist
#include (c:\ecm\196mc\mc.inc)
$list
;
;*****
; PROGRAM FREQ.A96
;
; This program transmits a block of data serially
; by gating the frequency generator on and off.
;
; The carrier frequency is programmed for 40 kHz.
;
; Ones are represented by a long (2 ms) carrier burst
; followed by a long (2 ms) pause (no carrier).
; Zeros are represented by a short (1 ms) carrier burst

```



```

; followed by a short (1 ms) pause, thus generating a MFM waveform.
;
; This program is assembled to run on the MD demo board.
;
;*****
; CONSTANT AND VARIABLE DECLARATIONS
;*****
; Program equates
; This section defines the constants used by this program.
; They can be changed at assembly time as required.
;*****
;
zero_time      equ      1000      ;1 ms
zero_pause_time equ      zero_time
one_time       equ      2000      ;2 ms
one_pause_time equ      one_time
carrier_freq   equ      25        ;40 KHz
buf_size       equ      8         ;size of data buffer (bytes)
fill_char      equ      10100011b ;initial data for buffer
;
;*****
; SFR equates in a 32-byte window
; This section defines SFR locations as seen through a window
; to allow using compact read-modify-write instructions.
;*****
;
p2_mode_w      equ      0f0H      ;WSR = 7EH 1FD0H
p2_dir_w       equ      0F2H      ;WSR = 7EH 1FD2H
p2_reg_w       equ      0F4H      ;WSR = 7EH 1FD4H
p2_pin_w       equ      0F6H      ;WSR = 7EH 1FD6H
;
p7_mode_w      equ      0f1H      ;WSR = 7EH 1FD1H
p7_dir_w       equ      0F3H      ;WSR = 7EH 1FD3H
p7_reg_w       equ      0F5H      ;WSR = 7EH 1FD5H
p7_pin_w       equ      0F7H      ;WSR = 7EH 1FD7H
;
timer1_w       equ      0FAH      ;WSR = 7BH 1F7AH
comp3_con_w    equ      0E4H      ;WSR = 7BH 1F64H
comp3_time_w   equ      0E6H      ;WSR = 7BH 1F66H
;
;*****
; Other SFR equates
; This section defines the locations of the frequency generator SFRs.
;*****
;
freq_gen       equ      1FB8H
freq_cnt       equ      1FBAH
;
;*****
; Variable storage area
; This section defines the variables used by this program.
;*****
;
rseg at 30h
;
rism:          dsb 13             ;reserved for RISM
;
rseg at 40h

```

```

;
temp:          dsw  1
temp1:         dsw  1
temp2:         dsw  1
buf_start:    dsw  1
buf_cnt:      dsb  1
bit_cnt:      dsb  1
flag:         dsb  1

;bit 0 = zero being sent
;bit 1 = one being sent
;bit 5 = get next bit
;bit 6 = get next byte
;bit 7 = buffer send in progress

;
xmit_buf:     dsb  buf_size      ;block of data to send
shift_reg:    dsb  1
;
;*****
; MAIN PROGRAM
;*****
; Define the program location and set up the interrupts and stack.
;*****
;
cseg at 0e000h          ;RISM user space
;
start:         di              ;set up interrupts
               dpts
               andb int_pend1,#10000000b;
               orb  int_mask1,#00000010b;unmask compare3
               ld   sp,#0200h      ;set up stack
;*****
; Initialize pin P7.7/FREQOUT for I/O and set the pin low.
;*****
;
               ldb  wsr,#7EH        ;move SFR's into window
               andb p7_reg_w,#01111111b ;P7.7 = low
               andb p7_dir_w,#01111111b ;P7.7 comp output
               andb p7_mode_w,#01111111b;P7.7 = I/O
               ldb  wsr,zero_reg
;*****
; This section fills the data buffer with a "fill" character.
; An application would typically place a block of data here.
;*****
;
               ld   temp,#xmit_buf   ;initialize buffer data
               ldb  temp1,#fill_char
               ldb  temp2,#buf_size
fill:          stb  temp1,[temp]+
               djnz temp2,fill
;*****
; Start timer 1 with a 1 microsecond clock period,
; load the carrier frequency into FREQ_GEN, and
; enable the interrupts.
;*****
;
               ldb  temp,#11000010b  ;init EPA timer1
               stb  temp,tlcontrol[0] ;1 uS ticks
               ldb  temp,#carrier_freq ;load carrier freq

```

```

        stb temp,freq_gen[0]    ;into freq gen
        ei                      ;enable interrupts
;
;
;*****
; Now send buffer out as serial data bytes
;*****
; This section issues a 1 millisecond pulse on P2.0
; for use with an oscilloscope monitor.
;*****
;
        ld    wsr,#7EH
        ld    temp,#0400
        andb p2_reg_w,#11111110b ;strobe p2.0 to sync
        andb p2_dir_w,#11111110b ;scope
        andb p2_mode_w,#11111110b;
        orb  p2_reg_w,#00000001b ;set pin high
        djnzw temp,$           ;pause
        andb p2_reg_w,#11111110b ;set pin low
        ld    wsr,zero_reg
;*****
; Initialize the buffers and flag register that the interrupt routine needs.
;*****
;
        ld    buf_start,#xmit_buf ;pointer reg
        ld    buf_cnt,#buf_size  ;number to send
        ldb  flag,#11000000b     ;set "buffer send in progress"
;*****
; and "get next byte" flags
;*****
; Set the compare module's interrupt pending bit
; to start sending the buffer. Loop until the buffer
; send is complete, then start main program over.
;*****
        ldb  int_pend1,#00000010b;force interrupt
wait:    jbs  flag,7,wait         ;loop here until done
        ljmp start              ;then start over
;
;
;*****
; COMPARE3 INTERRUPT ROUTINE
;*****
; This routine executes each time the EPA compare channel times out.
; The "flag" register identifies the reason for the interrupt request.
;*****
;
cseg at 0f120H                                ;comp3 Int vector demo board
;
        pusha                    ;save cpu status
        jbs  flag,1,one_pause     ;jump if one being sent
        jbs  flag,0,zero_pause   ;jump if zero being sent
        jbs  flag,5,get_bit      ;get next bit
        jbs  flag,6,get_byte     ;get next byte
        sjmp all_done            ;if nothing set, done
;
get_byte:  andb flag,#10111111b   ;clear get byte flag
        ldb  shift_reg,[buf_start]+ ;get byte to send into temp
        ldb  bit_cnt,#8           ;# of bits to send per char

```

```

        cmpb buf_cnt,#0                ;see if last byte has been sent
        jne  dec_buf_cnt              ;no!
        ljmp all_done                ;yes!
dec_buf_cnt:  decb buf_cnt              ;decrement byte count
;
get_bit:     andb flag,#11011111b     ;clear get bit flag
            shl  shift_reg,#1         ;shift MSB into carry flag
            jc   send_one              ;send a one
            ;else send zero

;
send_zero:   orb  flag,#00000001b     ;set zero's flag
            ldb  wsr,#7BH              ;set up EPA
            ldb  comp3_con_w,#01000000b
            add  comp3_time_w,timer1_w,#zero_time
            ldb  wsr,#7EH
            orb  p7_mode_w,#10000000b ;start FREQOUT
            ldb  wsr,zero_reg
            sjmp done

send_one:    orb  flag,#00000010b     ;set one's flag
            ldb  wsr,#7BH              ;set up EPA
            ldb  comp3_con_w,#01000000b
            add  comp3_time_w,timer1_w,#one_time
            ldb  wsr,#7EH
            orb  p7_mode_w,#10000000b ;start FREQOUT
            ldb  wsr,zero_reg
            sjmp done

;
zero_pause:  andb flag,#11111110b     ;turn off zero flag
            ldb  wsr,#7EH
            andb p7_mode_w,#01111111b ;turn off FREQOUT
            ldb  wsr,#7BH              ;set up EPA
            ldb  comp3_con_w,#01000000b
            add  comp3_time_w,timer1_w,#zero_pause_time
            ldb  wsr,zero_reg
            sjmp next

;
one_pause:   andb flag,#11111101b     ;turn off one flag
            ldb  wsr,#7EH
            andb p7_mode_w,#01111111b ;turn off FREQOUT
            ldb  wsr,#7BH              ;set up EPA
            ldb  comp3_con_w,#01000000b
            add  comp3_time_w,timer1_w,#one_pause_time
            ldb  wsr,zero_reg

;
next:        decb bit_cnt              ;decrement bit count
            cmpb bit_cnt,#00H         ;check if 8 bits sent
            jne  next1                ;no, get next bit
            orb  flag,#01000000b     ;yes, get next byte
            sjmp done                ;done for now
next1:       orb  flag,#00100000b     ;set get bit flag
            sjmp done

;
all_done:    ldb  flag,#00H            ;clear all flags
done:        popa
            ret

;
end

```




9

Waveform Generator

CHAPTER 9 WAVEFORM GENERATOR

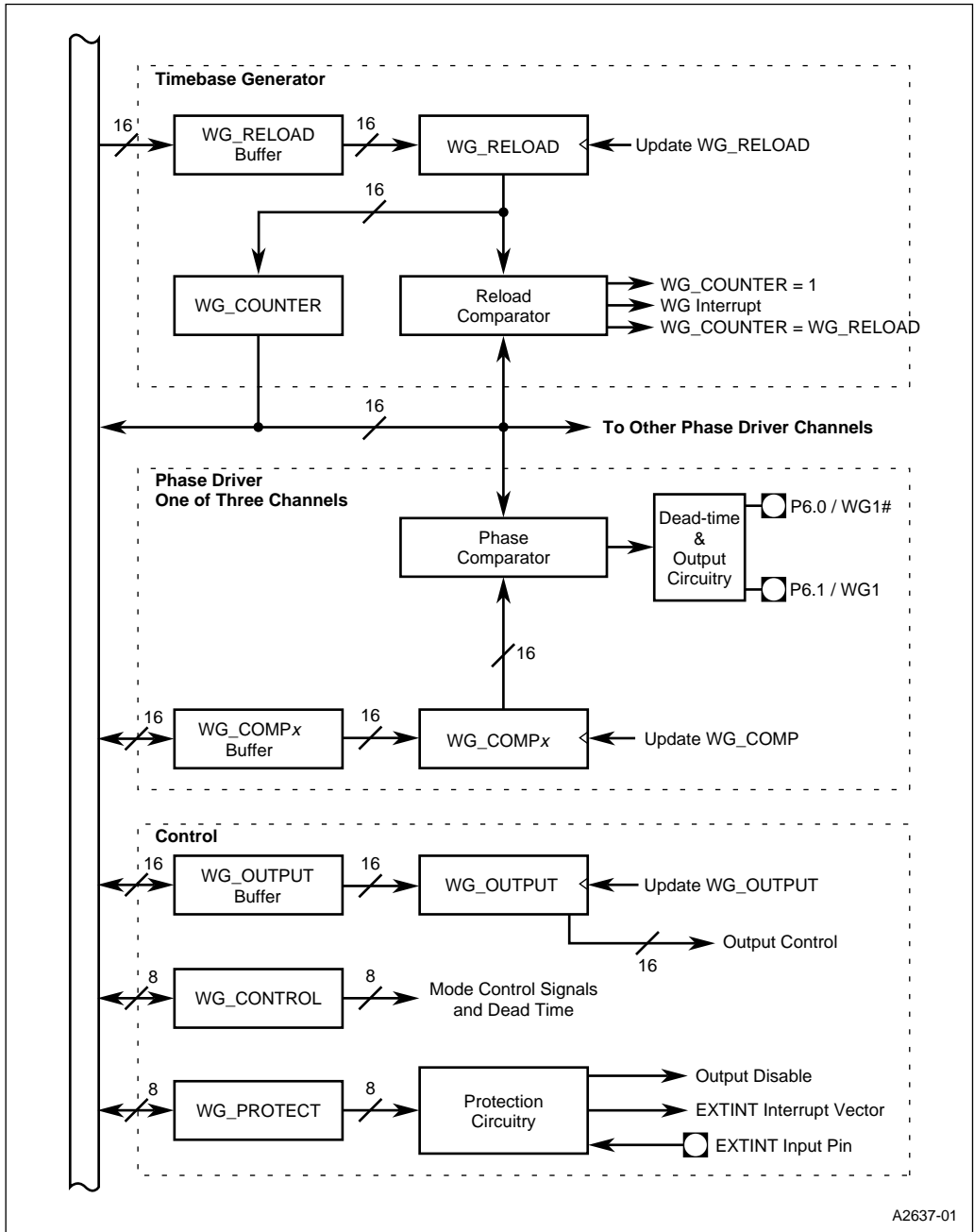
A waveform generator simplifies the task of generating synchronized, pulse-width modulated (PWM) outputs. This waveform generator is optimized for motion control applications such as driving 3-phase AC induction motors, 3-phase DC brushless motors, or 4-phase stepping motors. The waveform generator can produce three independent pairs of complementary PWM outputs that share a common carrier period, dead time, and operating mode. Once it is initialized, the waveform generator operates without CPU intervention unless you need to change a duty cycle.

This chapter describes the waveform generator and explains how to configure it. For detailed descriptions of the signals discussed in this chapter, refer to Appendix B, “Signal Descriptions.” For additional information and application examples, consult AP-483, *Application Examples Using the 8XC196MC/MD Microcontroller* (order number 272282).

9.1 WAVEFORM GENERATOR FUNCTIONAL OVERVIEW

The waveform generator (Figure 9-1) has three main parts: a timebase generator, phase driver channels, and control circuitry. The timebase generator establishes the carrier period, the phase driver channels determine the duty cycle, and the control circuitry determines the operating mode and controls interrupt generation. The waveform generator’s maximum frequency is 15.625 kHz for center-aligned modes and 31.250 kHz for edge-aligned modes.

There are three independent phases, each of which has two programmable, complementary outputs. A programmable “dead-time” generator prevents the complementary outputs from being active at the same time. The carrier period, dead time, and operating mode are the same for all three phases; the duty cycle is independently programmable.



A2637-01

Figure 9-1. Waveform Generator Block Diagram

9.2 WAVEFORM GENERATOR SIGNALS AND REGISTERS

Table 9-1 describes the waveform generator’s signals, and Table 9-2 briefly describes the control and status registers.

Table 9-1. Waveform Generator Signals

Port Pin	Waveform Generator Signal	Type	Description
P6.0	WG1#	O	Waveform generator phase 1 negative output.
P6.1	WG1	O	Waveform generator phase 1 positive output.
P6.2	WG2#	O	Waveform generator phase 2 negative output.
P6.3	WG2	O	Waveform generator phase 2 positive output.
P6.4	WG3#	O	Waveform generator phase 3 negative output.
P6.5	WG3	O	Waveform generator phase 3 positive output.
—	EXTINT	I	Input to the waveform generator’s protection circuitry.

Table 9-2. Waveform Generator Control and Status Registers

Mnemonic	Address	Description
INT_MASK1	0013H	Interrupt Mask 1 The EXTINT bit enables or disables the EXTINT interrupt. 8XC196MH: The WG bit enables or disables the waveform generator interrupt. 8XC196MC, MD: The PI bit enables or disables the multiplexed peripheral interrupt. The corresponding bit in the PI_MASK register enables or disables the individual sources of the peripheral interrupt.
INT_PEND1	0014H	Interrupt Pending 1 Any set bit indicates a pending interrupt request.
PI_MASK (MC, MD)	1FBCH	Peripheral Interrupt Mask 8XC196MC, MD: The WG bit enables or disables the waveform generator interrupt as one of the possible sources of the multiplexed peripheral interrupt. The PI bit in INT_MASK1 must be set to enable the multiplexed peripheral interrupt.
PI_PEND (MC, MD)	1FBEH	Peripheral Interrupt Pending Any set bit indicates a pending interrupt request.
WG_COMP1 WG_COMP2 WG_COMP3	1FC2H 1FC4H 1FC6H	Waveform Generator Compare Buffers Each phase compare buffer contains a value that is compared with the counter value. The action that is performed when a match occurs depends on the operating mode.

Table 9-2. Waveform Generator Control and Status Registers (Continued)

Mnemonic	Address	Description
WG_CONTROL	1FCCH	Waveform Generator Control The control register determines the waveform generator's operating mode, starts and stops the counter, specifies the dead time for all phases, and indicates the current count direction.
WG_COUNTER	1FCAH	Waveform Generator Count Value The read-only counter register reflects the current counter value.
WG_OUTPUT	1FC0H	Waveform Generator Output Control The output control register configures the waveform generator's outputs and selects their active polarity.
WG_PROTECT	1FCEH	Waveform Generator Protection The protection register enables and disables the protection circuitry and the outputs, selects level-sensitive or edge-triggered interrupts, and controls which value of the edge or level will trigger an interrupt request. 8XC196MH only: This register also selects the method for disabling the outputs: inactive states or weak pull-ups.
WG_RELOAD	1FC8H	Waveform Generator Reload Value The reload register contains a value that is compared with the counter value. The actions performed based on this comparison depend on the operating mode.

9.3 WAVEFORM GENERATOR OPERATION

This section describes the major components of the waveform generator: the timebase generator, the phase driver channels, and the control and protection circuitry. It also explains how the buffered registers are updated and describes the similarities and differences between the center-aligned and edge-aligned operating modes. Finally, it describes the two types of interrupt requests that the waveform generator can generate and explains how to enable the interrupts.

9.3.1 Timebase Generator

The timebase generator establishes the carrier period of the PWM outputs. You specify this period by writing a value to the reload register (WG_RELOAD). This value is loaded into the counter register (WG_COUNTER) when the system is initialized and periodically (depending on the operating mode) thereafter. You can read the counter register to determine the current counter value and you can write to the reload register to change the reload value at any time.

The 16-bit timebase counter is clocked every state time. The control register (WG_CONTROL) enables and disables the counter, controls the counting mode, and reflects the count direction. When the counter is enabled, it continuously counts between 0001H and the reload value. Writing 0000H to the reload register or clearing the enable bit in the control register stops the counter.

9.3.2 Phase Driver Channels

The phase driver channels determine the duty cycle of the outputs. You specify the duty cycle by writing a value to each phase’s compare register (WG_COMP x). In all operating modes, the outputs are initially asserted, and they remain asserted until the counter value (WG_COUNTER) matches the phase’s compare register (WG_COMP x) value. At this point, the outputs are deasserted and remain deasserted until another event occurs. The event that causes the outputs to be asserted again depends on the operating mode. (See “Operating Modes” on page 9-7.)

The dead-time generator circuitry (Figure 9-2) prevents an output and its complement from being asserted at the same time. It uses two internal signals, WFG and DT, to generate the nonoverlapping outputs. The edge-detection circuitry generates the WFG signal, while a 10-bit dead-time counter generates the DT signal. When a valid edge is detected, the dead-time counter is loaded with the 10-bit dead-time value from the control register and DT is driven low. The counter decrements once every state time until it reaches zero, at which point the counter stops and DT is driven high. The WFG signal is ANDed with DT to produce the WG_EVEN signal; the WFG# signal is ANDed with DT to produce the WG_ODD signal. The waveform generator’s outputs can be connected to the WG_EVEN and WG_ODD signals. (See “Configuring the Outputs” on page 9-12.)

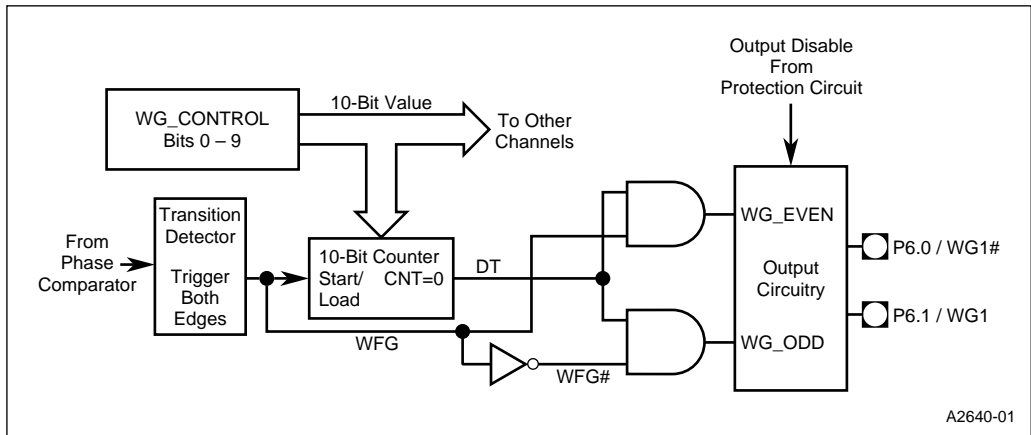


Figure 9-2. Dead-time Generator Circuitry

9.3.3 Control and Protection Circuitry

The control circuitry contains the control (WG_CONTROL) and output (WG_OUTPUT) registers. The control register enables or disables the counter, specifies the count direction, controls the operating mode, and specifies the dead time for all three phases. The output register configures the pins, specifies the output polarity (active high or active low), and controls whether the outputs are updated immediately or are synchronized with an event.

The protection circuitry (Figure 9-3) monitors the EXTINT pin. When it detects a valid event on the input, it simultaneously disables the outputs and generates an EXTINT interrupt request. Software can also disable the outputs by clearing the enable outputs (EO) bit in the protection (WG_PROTECT) register.

For the 8XC196MC and 8XC196MD, disabled outputs go to their inactive states based on the programmed polarity. The protection circuitry of the 8XC196MH operates in the same way as that of the 8XC196MC and 8XC196MD, but it allows you to choose the method used to disable the outputs. It can either place outputs in their inactive states, as the other devices do, or it can apply weak pull-ups to them. The protection type (PT) bit in the protection register controls the method.

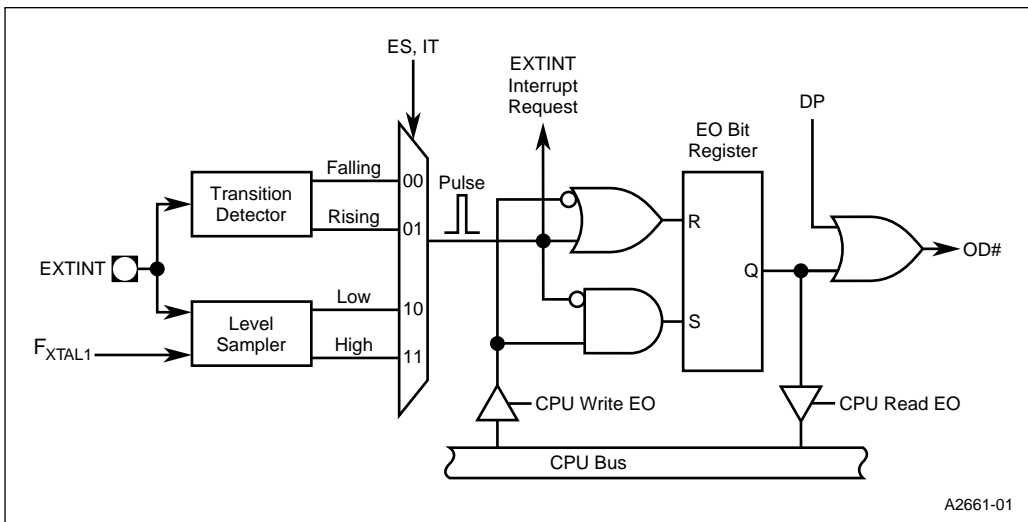


Figure 9-3. Protection Circuitry

9.3.4 Register Buffering and Synchronization

The WG_RELOAD, WG_COMP_x, and WG_OUTPUT registers are buffered; you read and write the buffers rather than the registers. The waveform generator updates the registers synchronously to prevent erroneous or nonsymmetrical duty cycles.

When you write to the WG_COMP_x buffers while the counter is stopped (either when the counter register is zero or when the enable counter bit in the control register is clear), the registers are updated one-half state time later.

The WG_RELOAD register is updated when the counter value reaches the reload value. The WG_COUNTER register is loaded with the updated WG_RELOAD value, so a new reload value takes effect for the next cycle. In mode 3 (and mode 4 for the 8XC196MH), the WG_RELOAD register can be updated when an EPA event occurs. This requires you to enable an EPA channel's peripheral function. (See Chapter 11, "Event Processor Array (EPA)" for details.)

The WG_OUTPUT register contains a synchronization bit that controls whether changes to the output signals are reflected immediately or are synchronized with an event. The synchronization bit is not buffered, so changes to it take effect immediately. You should initialize the synchronization to zero (changes take effect immediately) to ensure that the pins are in the desired states when the counter starts.

9.3.5 Operating Modes

The waveform generator can operate in a center-aligned or an edge-aligned mode. In the center-aligned modes, the counter counts both up and down; in the edge-aligned modes, it counts up only. The center-aligned modes generate PWM outputs that are more efficient for driving 3-phase AC induction motors, while the edge-aligned modes generate conventional PWM outputs. Center-aligned outputs have less harmonic content than edge-aligned outputs, with effectively twice the carrier period.

The initial condition of the waveform generator is the same for all operating modes. Following a system power-up or reset, the counter is stopped, outputs are deasserted, and all registers are cleared. Values written to the registers take effect one-half state time later.

The main differences between center-aligned and edge-aligned modes are the counter's initial value, the count direction, and the conditions that cause a change in the state of the outputs. Table 9-3 summarizes the operation of the center-aligned and edge-aligned modes.

Table 9-3. Operation in Center-aligned and Edge-aligned Modes

Step	Center-aligned Modes	Edge-aligned Modes
1	Load WG_COUNTER with WG_RELOAD. Leave outputs deasserted.	Load WG_COUNTER with 0001H. Leave outputs deasserted.
2	When counter is enabled, begin counting down. When WG_COUNTER reaches 1, wait 1 state, then begin counting up. Assert outputs when up count begins.	When counter is enabled, begin counting up. Assert outputs when up count begins.
3	When WG_COUNTER reaches the WG_COMPx value during the up count, deassert the corresponding phase's outputs and continue counting up.	When WG_COUNTER reaches the WG_COMPx value, deassert the corresponding phase's outputs and continue counting up.
4	When WG_COUNTER reaches the WG_RELOAD value, begin counting down.	When WG_COUNTER reaches the WG_RELOAD value, update WG_RELOAD and go to step 1.
5	When WG_COUNTER reaches the WG_COMPx value during the down count, assert the corresponding phase's outputs and continue counting down.	
6	When WG_COUNTER reaches 1, deassert outputs, update WG_RELOAD, and go to step 1.	

The main differences between the center-aligned modes and among the edge-aligned modes are the events that control register updates. Table 9-4 lists the events that can cause register updates and the registers that are updated in each mode.

Table 9-4. Register Updates

Event	Center-aligned Modes		Edge-aligned Modes		
	Mode 0	Mode 1	Mode 2	Mode 3	Mode 4 (MH Only)
	Registers Updated		Registers Updated		
WG_COUNTER = WG_RELOAD	WG_RELOAD WG_COUNTER WG_COMPx WG_OUTPUT [†]	WG_RELOAD WG_COUNTER WG_COMPx WG_OUTPUT [†]	WG_RELOAD WG_COUNTER WG_COMPx WG_OUTPUT [†]	WG_RELOAD WG_COUNTER WG_COMPx WG_OUTPUT [†]	WG_RELOAD WG_COUNTER WG_COMPx WG_OUTPUT [†]
WG_COUNTER = 1	—	WG_COMPx	—	—	—
EPA event	WG_OUTPUT [†]	WG_OUTPUT [†]	WG_OUTPUT [†]	WG_RELOAD WG_COUNTER WG_COMPx WG_OUTPUT [†]	WG_OUTPUT [†]

[†] The WG_OUTPUT register is updated under these conditions if its synchronization bit is set; otherwise, changes take effect immediately.

9.3.5.1 Center-aligned Modes

In the center-aligned modes, the counter counts down from the WG_RELOAD value to 1, then counts back up from 1 to WG_RELOAD. When you write to the WG_RELOAD register, WG_COUNTER is loaded with the reload value. When you set the enable bit in the control register, the counter begins counting down and continues counting until it reaches 1, waits one state time, and starts counting up until it reaches WG_RELOAD. At this point, WG_RELOAD is updated and WG_COUNTER is reloaded with the updated value, so a new reload value takes effect for the next cycle. The counter resumes counting down from WG_RELOAD to 1. This produces a symmetrical ascending and descending count, illustrated by the triangular wave in Figure 9-4, with a period that is twice the WG_RELOAD value. Figure 9-5 shows the operation of outputs and interrupts in center-aligned modes.

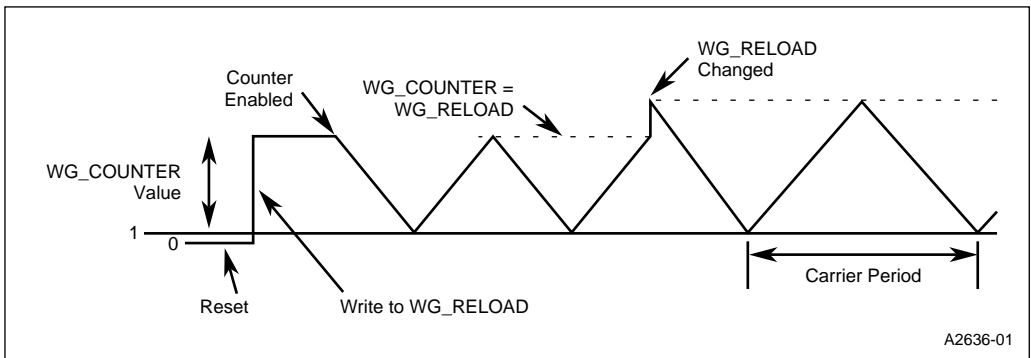


Figure 9-4. Center-aligned Modes — Counter Operation

In mode 0, the WG_COMPx and WG_OUTPUT registers are updated only once during the carrier period, when the counter reaches the reload value. In mode 1, these registers are updated twice during the carrier period: first when the counter is set to 1, then again when it reaches the reload value.

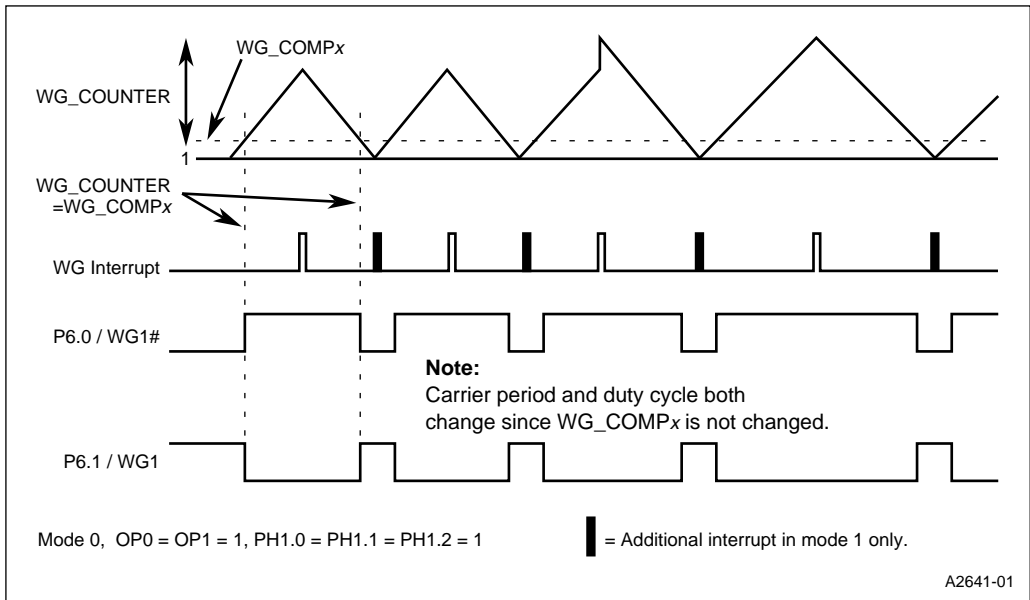


Figure 9-5. Center-aligned Modes — Output Operation

9.3.5.2 Edge-Aligned Modes

In the edge-aligned modes, the counter begins at 1 and counts up to the WG_RELOAD value. When you write to the WG_RELOAD register, WG_COUNTER is loaded with 0001H. When you set the enable bit in the control register, the counter begins counting up and continues counting until it reaches the WG_RELOAD value or, in mode 3 only, until an EPA event occurs. At this point, WG_COUNTER is reloaded with 0001H and WG_RELOAD is updated, so a new reload value takes effect for the next cycle. The counter resumes counting up from 0001H to WG_RELOAD. This produces a smoothly ascending count, illustrated by the sawtooth wave in Figure 9-6, with a period that is equal to the WG_RELOAD value. Figure 9-7 shows the operation of outputs and interrupts in edge-aligned modes.

In mode 2, the registers are updated only once during the carrier period, when the counter reaches the reload value. In mode 3, the registers are also updated when an EPA peripheral function event occurs. (You must configure an EPA channel for this function. See Chapter 11, “Event Processor Array (EPA)” for information.)

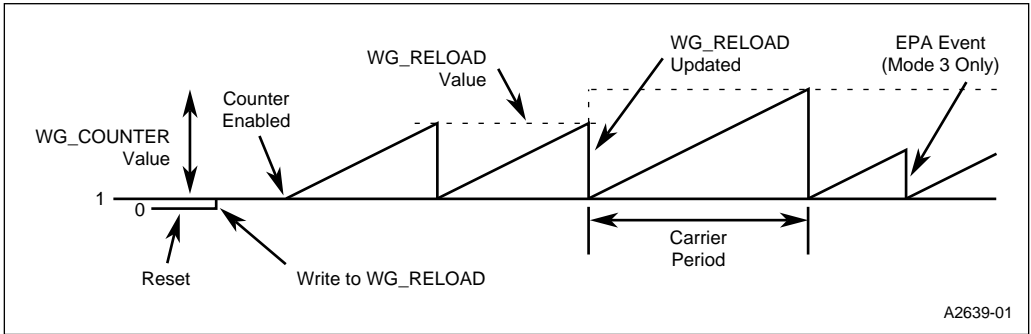


Figure 9-6. Edge-aligned Modes — Counter Operation

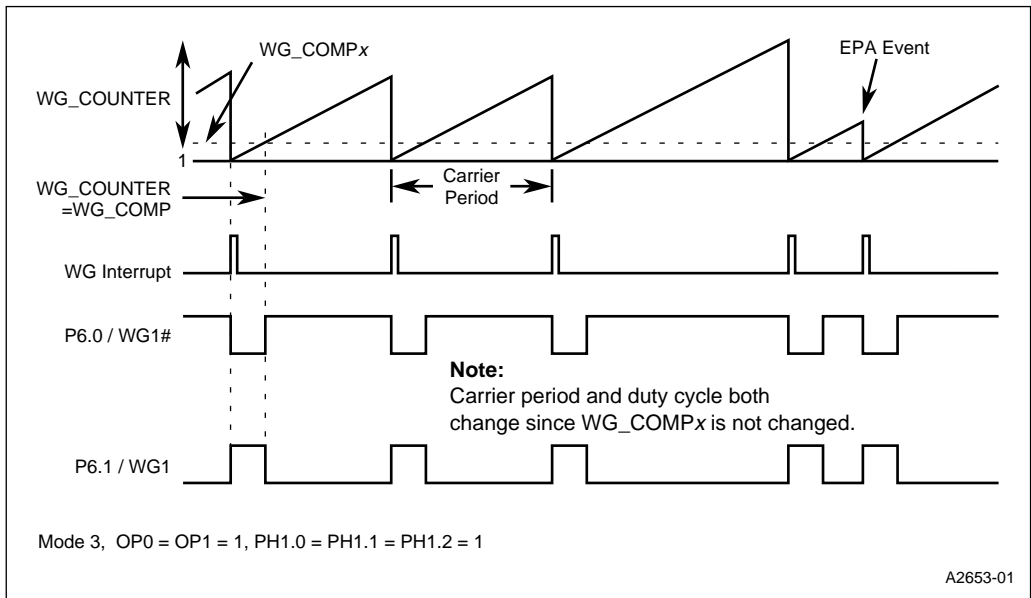


Figure 9-7. Edge-aligned Modes — Output Operation

8XC196MH only: The 8XC196MH device has an additional edge-aligned mode, mode 4. This mode prevents the output “jitter” that can occur in mode 3 when an EPA event reloads WG_COUNTER, which also causes a change in the duty cycle. In mode 4, an EPA event reloads WG_OUTPUT, but WG_COMPx and WG_COUNTER are reloaded only when the counter reaches the reload value.

9.4 PROGRAMMING THE WAVEFORM GENERATOR

This section explains how to configure the waveform generator and determine its status.





9.4.1 Configuring the Outputs

The waveform generator's outputs are multiplexed with general-purpose output port 6, so you must configure them as special-function signals to use them as waveform-generator outputs. The WG_OUTPUT register (Figure 9-8) configures the pins, establishes the output polarity, and controls whether changes to the outputs are synchronized with an event or take effect immediately.

Four bits of WG_OUTPUT are unrelated to the waveform generator; they configure the outputs for the pulse-width modulator (PWM) peripheral, which also shares pins with port 6. The P6 and PE6 bits control the P6.6/PWM0 pin, and the P7 and PE7 bits control the P6.7/PWM1 pin. Their placement in this register allows you to configure all the port 6 pins with a single write to WG_OUTPUT.

Table 9-5 shows the bit combinations necessary to drive the waveform generator's outputs high or low or to connect them to the WG_EVEN or WG_ODD signal. Note that PHx.2 is always set to select the waveform-generator signal function (clearing PHx.2 selects the general-purpose I/O port function). The "Output Polarities" column shows the output polarities. The drawings show a duty cycle of about 15%, and for these cases, the high portion of the waveforms increases as dead time increases.

Table 9-5. Output Configuration

PHx.2	PHx.1	PHx.0	Output Values		Output Polarities	
			WGx	WGx#	WGx	WGx#
1	0	0	Low	Low	Always Low	Always Low
1	0	1	Low	WG_EVEN#	Always Low	
1	1	0	WG_ODD	Low		Always Low
1	1	1	WG_ODD	WG_EVEN		

NOTE: This table assumes active-high outputs (OP1=OP0=1).

WG_OUTPUT (Waveform Generator)

 Address: 1FC0H
 Reset State: 0000H

The waveform generator output configuration (WG_OUTPUT) register controls the configuration of the waveform generator and PWM module pins. Both the waveform generator and the PWM module share pins with port 6. Having these control bits in a single register enables you to configure all port 6 pins with a single write to WG_OUTPUT.

15
8

OP1	OP0	SYNC	PE7	PE6	PH3.2	PH2.2	PH1.2
-----	-----	------	-----	-----	-------	-------	-------

7
0

P7	P6	PH3.1	PH3.0	PH2.1	PH2.0	PH1.1	PH1.0
----	----	-------	-------	-------	-------	-------	-------

Bit Number	Bit Mnemonic	Function
15	OP1	Output Polarity Selects the output polarity for negative-phase outputs WG1#, WG2#, and WG3#. 0 = active-low outputs 1 = active-high outputs
14	OP0	Output Polarity Selects the output polarity for positive-phase outputs WG1, WG2, and WG3. 0 = active-low outputs 1 = active-high outputs
13	SYNC	Synchronize Selects whether updating the WG_OUTPUT register is synchronized with another event or occurs immediately after you change it. 0 = update WG_OUTPUT immediately 1 = synchronize WG_OUTPUT update with an event To ensure that the outputs are in the desired states when the waveform generator starts, you should initially clear this bit, then set it later if you want subsequent WG_OUTPUT updates to be synchronized with an event. (Table 9-4 on page 9-8 lists the events that update WG_OUTPUT in each mode.)
12	PE7	P6.7/PWM1 Function Selects the port function or the PWM output function of P6.7/PWM1. 0 = P6.7 1 = PWM1
11	PE6	P6.6/PWM0 Function Selects the port function or the PWM output function of P6.6/PWM0. 0 = P6.6 1 = PWM0

Figure 9-8. WG Output Configuration (WG_OUTPUT) Register

WG_OUTPUT (Waveform Generator) (Continued)Address: 1FC0H
Reset State: 0000H

The waveform generator output configuration (WG_OUTPUT) register controls the configuration of the waveform generator and PWM module pins. Both the waveform generator and the PWM module share pins with port 6. Having these control bits in a single register enables you to configure all port 6 pins with a single write to WG_OUTPUT.

15

8

OP1	OP0	SYNC	PE7	PE6	PH3.2	PH2.2	PH1.2
-----	-----	------	-----	-----	-------	-------	-------

7

0

P7	P6	PH3.1	PH3.0	PH2.1	PH2.0	PH1.1	PH1.0
----	----	-------	-------	-------	-------	-------	-------

Bit Number	Bit Mnemonic	Function
10	PH3.2	Phase 3 Function Selects either the port function or the waveform generator output function for pins P6.4/WG3# and P6.5/WG3. 0 = P6.4, P6.5 1 = WG3#, WG3
9	PH2.2	Phase 2 Function Selects either the port function or the waveform generator output function for pins P6.2/WG2# and P6.3/WG2. 0 = P6.2, P6.3 1 = WG2#, WG2
8	PH1.2	Phase 1 Function Selects either the port function or the waveform generator output function for pins P6.0/WG1# and P6.1/WG1. 0 = P6.0, P6.1 1 = WG1#, WG1
7	P7	P6.7/PWM1 Value Write the desired P6.7/PWM1 value to this bit.
6	P6	P6.6/PWM0 Value Write the desired P6.6/PWM0 value to this bit.
5:4	PH3.1:0	P6.4/WG3#, P6.5/WG3 Value Write the desired output values to these bits. See Table 9-5 on page 9-12.
3:2	PH2.1:0	P6.2/WG2#, P6.3/WG2 Values Write the desired output values to these bits. See Table 9-5 on page 9-12.
1:0	PH1.1:0	P6.0/WG1#, P6.1/WG1 Values Write the desired output values to these bits. See Table 9-5 on page 9-12.

Figure 9-8. WG Output Configuration (WG_OUTPUT) Register (Continued)

9.4.2 Controlling the Protection Circuitry and EXTINT Interrupt Generation

The protection register (Figure 9-9) controls the protection circuitry and EXTINT interrupt requests.

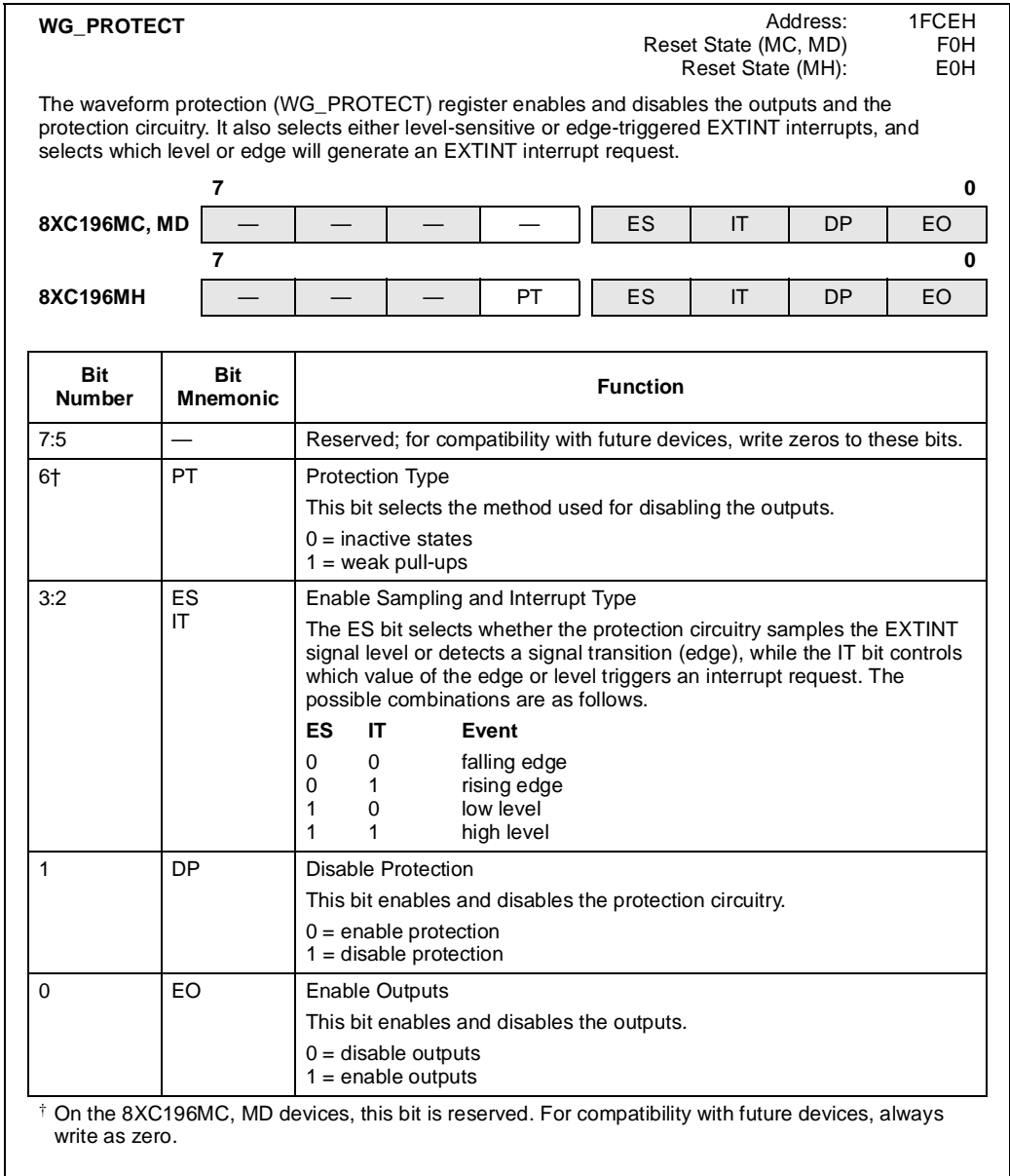


Figure 9-9. Waveform Generator Protection (WG_PROTECT) Register

9.4.3 Specifying the Carrier Period and Duty Cycle

The reload register (WG_RELOAD) and the phase compare registers (WG_COMP_x) control the carrier period and duty cycle. Write a value to the reload register (Figure 9-10) to establish the carrier period. Write a value to each phase compare register to specify the length of time that the associated outputs will remain asserted.

WG_RELOAD	Address: 1FC8H Reset State: 0000H
<p>The waveform generator reload (WG_RELOAD) register and the phase compare registers (WG_COMP_x) control the carrier period and duty cycle. Write a value to the reload register to establish the carrier period.</p> <p>Changing the WG_RELOAD value changes both the carrier period and the duty cycle because the outputs remain asserted for a constant length of time, while the counter takes longer to cycle. To change the carrier period without changing the duty cycle, you must proportionally change both WG_RELOAD and WG_COMP_x at the same time, immediately after the interrupt.</p>	
15	0
Reload	
Bit Number	Function
15:0	<p>Reload</p> <p>This register determines the carrier period.</p> <p>Use the following formulas to calculate carrier period and duty cycle.</p> $T_{\text{CARRIER}} = \frac{\text{multiplier} \times \text{WG_RELOAD}}{F_{\text{XTAL1}}}$ $\text{Duty Cycle} = \frac{\text{WG_COMP}_x}{\text{WG_RELOAD}} \times 100\%$ <p>where:</p> <p>T_{CARRIER} = carrier period, in μs</p> <p>F_{XTAL1} = input frequency on XTAL1 pin, in MHz</p> <p><i>multiplier</i> = 4 for center-aligned modes; 2 for edge-aligned modes</p> <p>WG_RELOAD = 16-bit WG_RELOAD value \geq WG_COMP_x</p> <p>WG_COMP_x = 16-bit WG_COMP_x value \leq WG_RELOAD</p>

Figure 9-10. Waveform Generator Reload (WG_RELOAD) Register

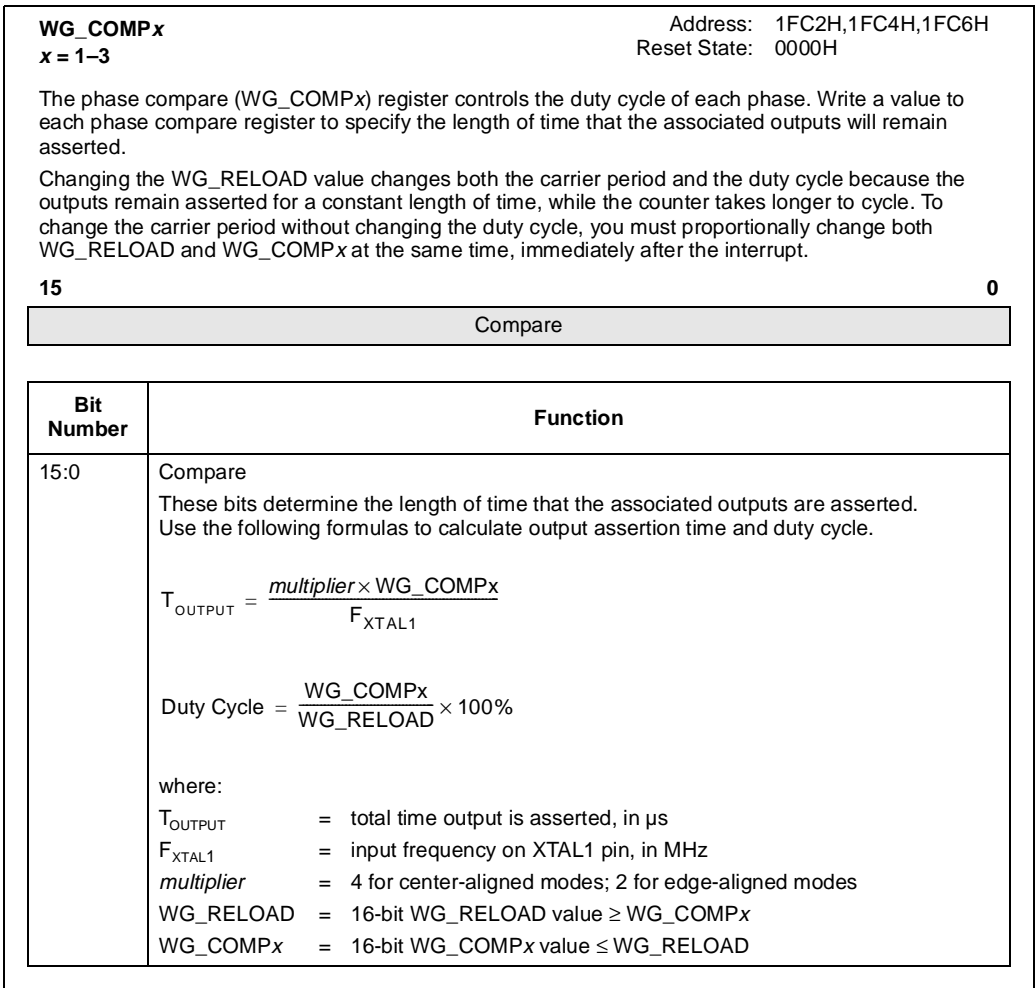


Figure 9-11. Phase Compare (WG_COMP_x) Register

9.4.4 Specifying the Operating Mode and Dead Time and Starting the Counter

The control register (Figure 9-12) specifies the dead time and operating mode and enables and disables the counters. A read-only bit (CS) indicates the current count direction.

WG_CONTROL

Address: 1FCCH
 Reset State (MC, MD): 00C0H
 Reset State (MH): 8000H

The waveform generator control (WG_CONTROL) register controls the operating mode, dead time, and count direction, and enables and disables the counter.

15

8

—	M2	M1	M0	CS	EC	DT9	DT8
---	----	----	----	----	----	-----	-----

7

0

DT7	DT6	DT5	DT4	DT3	DT2	DT1	DT0
-----	-----	-----	-----	-----	-----	-----	-----

Bit Number	Bit Mnemonic	Function																								
15	—	Reserved; for compatibility with future devices, write zero to this bit.																								
14:12	M2:0	<p>Operating Mode This field controls the waveform generator's operating mode.</p> <table border="1"> <thead> <tr> <th>M2</th> <th>M1</th> <th>M0</th> <th>Mode</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>0 center-aligned; update registers once</td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> <td>1 center-aligned; update registers twice</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> <td>2 edge-aligned; update registers once</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> <td>3 edge-aligned; update registers twice</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>4 (8XC196MH only) edge-aligned; update WG_COMPx and WG_COUNTER only when WG_COUNTER = WG_RELOAD</td> </tr> </tbody> </table>	M2	M1	M0	Mode	0	0	0	0 center-aligned; update registers once	0	0	1	1 center-aligned; update registers twice	0	1	0	2 edge-aligned; update registers once	0	1	1	3 edge-aligned; update registers twice	1	1	1	4 (8XC196MH only) edge-aligned; update WG_COMPx and WG_COUNTER only when WG_COUNTER = WG_RELOAD
M2	M1	M0	Mode																							
0	0	0	0 center-aligned; update registers once																							
0	0	1	1 center-aligned; update registers twice																							
0	1	0	2 edge-aligned; update registers once																							
0	1	1	3 edge-aligned; update registers twice																							
1	1	1	4 (8XC196MH only) edge-aligned; update WG_COMPx and WG_COUNTER only when WG_COUNTER = WG_RELOAD																							
11	CS	<p>Counter Status This read-only bit indicates whether the counter is counting up or counting down. 0 = down counting 1 = up counting</p>																								
10	EC	<p>Enable Counter This bit starts and stops the counter. 0 = disable (stop) counter 1 = enable (start) counter</p>																								
9:0	DT9:0	<p>Dead-time This field specifies the dead-time for all three phases. Use the following formula to calculate the appropriate DT_VALUE.</p> $DT_VALUE = \frac{T_{DEAD} \times F_{XTAL1}}{2}$ <p>where: T_{DEAD} = dead-time, in μs F_{XTAL1} = input frequency on XTAL1 pin, in MHz</p>																								

Figure 9-12. Waveform Generator Control (WG_CONTROL) Register

9.5 DETERMINING THE WAVEFORM GENERATOR'S STATUS

Read WG_CONTROL (Figure 9-12 on page 9-18) to determine the current dead-time value, counter status, count direction, and operating mode. Read WG_COUNTER (Figure 9-13) to determine the current counter value.

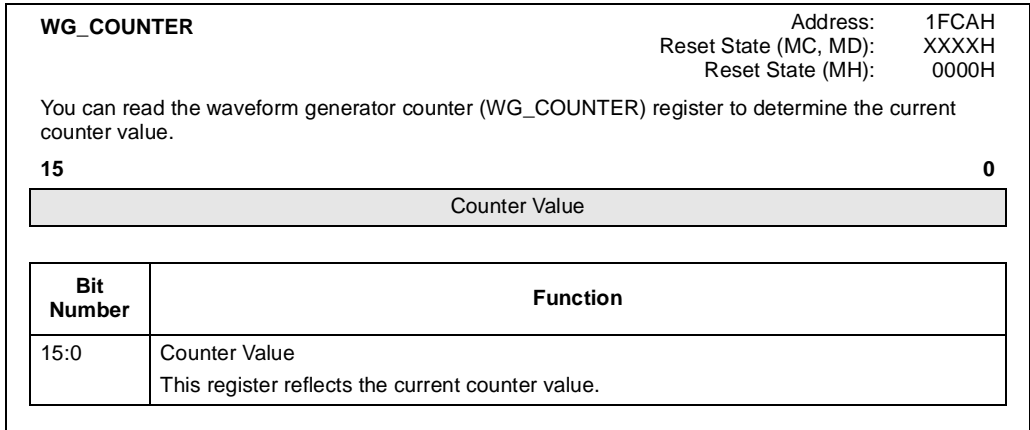


Figure 9-13. Waveform Generator Counter (WG_COUNTER) Register

9.6 ENABLING THE WAVEFORM GENERATOR INTERRUPTS

The waveform generator can generate two types of interrupt requests. The WG interrupt request is triggered by the counter, while the EXTINT interrupt is triggered by an external event.

Mode 0 generates a WG interrupt request once per period, when the counter reaches the WG_RELOAD value. Mode 1 generates a WG interrupt request twice per period, first when the counter reaches 1 and again when it reaches the WG_RELOAD value. The edge-aligned modes generate a WG interrupt request once at the end of each period, when the counter is reloaded with 1.

The protection circuitry controls the EXTINT interrupt. Two bits in the protection register control the type of external event that will generate an interrupt request: a falling or rising edge or a low or high level. (See “Controlling the Protection Circuitry and EXTINT Interrupt Generation” on page 9-15.)

The edge detection circuitry requires a signal to remain asserted for at least 2 state times to be considered a valid edge. The sample circuitry requires a signal to remain asserted for at least 24 state times to be considered a valid level. It samples the input level 3 times during this 24-state period and recognizes the signal as valid only if it is asserted for each sample. Level sampling is useful for environments in which noise spikes might cause unintended interrupts if edge detection were used.

To enable the interrupts, set the corresponding mask bits in the mask register (see Table 9-2 on page 9-3) and execute the EI instruction to enable interrupt servicing. You can read the interrupt pending register to determine whether there are any pending interrupts. Refer to Chapter 5, “Standard and PTS Interrupts” for details.

9.7 DESIGN CONSIDERATIONS

This section describes design and programming considerations for using the waveform generator.

9.7.1 Dead Time and Duty Cycle

Short dead times have little effect on the duty cycle if the pulse is relatively wide; however, longer dead times with narrower pulses can affect the duty cycle (Figure 9-14). No minimum pulse width is imposed by the hardware, so it is possible to deassert an output for the entire period if the total dead time is greater than the pulse width. For this reason, software should ensure that the pulse width is at least $3 \times T_{DEAD}$.

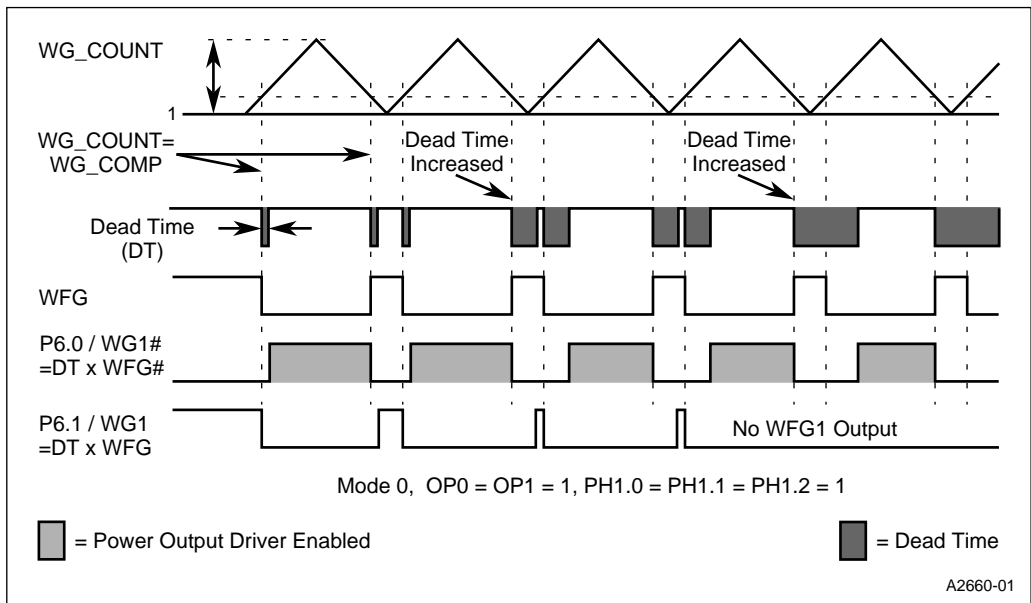


Figure 9-14. Effect of Dead Time on Duty Cycle

9.7.2 EXTINT Interrupts and Protection Circuitry

The protection register contains two bits, disable protection (DP) and enable output (EO), that together enable and disable the waveform generator's outputs. The EXTINT event generates a single short pulse that clears the EO bit, so if software sets the EO bit immediately following an EXTINT event, the outputs will be disabled only for the time between the EXTINT event and the CPU write. The CPU can immediately set the EO bit again, even if the EXTINT signal remains asserted.

9.8 PROGRAMMING EXAMPLE

This example was designed to run on an 8XC196MC demo board, but it can easily be modified for an evaluation board. The program allows you to test the waveform generator's registers and observe their effects on the output waveforms. All variables are defined as words and are masked to the appropriate length before they are written to the registers. (This method is not compact, but it is easy to code and debug.) When running the program under the reduced instruction set monitor (RISM) software, you can use the following command to change any variable and immediately see the result on the outputs:

WORD.variable_name

```
$debug
;Program to test WFG peripheral
;
$noList
$include (c:\ecm\196mc\mc.inc)
$list
;
; This program allows modifying the WFG input parameters "on the fly"
; on the MC demo board. This allows you to see what is really going on.
;
; First, set up the variables that you want to control:
;
rseg at 40h
;
mode:      dsw 1      ;mode = 0-3
op0:      dsw 1      ;P6.0,2,4 polarity--0=low, 1=high
op1:      dsw 1      ;P6.1,3,5 polarity--0=low, 1=high
sync:     dsw 1      ;0=load now, 1 = synchronized
pe7:      dsw 1      ;P6.7 0=i/o, 1=PWM
pe6:      dsw 1      ;P6.6 0=i/o, 1=PWM
p7:       dsw 1      ;P6.7 I/O value
p6:       dsw 1      ;P6.7 I/O value
ph1:      dsw 1      ;P6.0,1 config
ph2:      dsw 1      ;P6.2,3 config
```

```

ph3:      dsw 1      ;P6.4,5 config
eo:       dsw 1      ;0=disable output, 1=enable output
dp:       dsw 1      ;0=enable protection, 1=disable
it:       dsw 1      ;0=falling edge trig, 1=rising edge
es:       dsw 1      ;0=edge, 1=sample
ec:       dsw 1      ;0=stop cntr, 1=start
dead:     dsw 1      ;10-bit dead time
reload:   dsw 1
comp1:    dsw 1
comp2:    dsw 1
comp3:    dsw 1
temp:     dsw 1
temp1:    dsw 1
temp2:    dsw 1
;
;
;*****
; first, initialize the values for these variables:
;*****
;
cseg at 0e000h          ;demo board RAM
;
        di
        dpts
        ld  sp,#200h
        ld  mode,#0000h          ;mode0
        ld  ec,#0001h           ;enable cntr
        ld  dead,#0010h         ;1.6 us
        ld  op0,#0001h          ;active high
        ld  opl,#0001h          ;active high
        ld  sync,#0001h         ;synchronized WG_OUTPUT load
        ld  pe7,#00000h         ;P6.7 in I/O mode
        ld  pe6,#00000h         ;P6.6 in I/O mode
        ld  p7,#00000h         ;P6.7 I/O = 0
        ld  p6,#00000h         ;P6.6 I/O = 0
        ld  ph1,#0007h          ;wfg both outputs
        ld  ph2,#0007h          ;wfg both outputs
        ld  ph3,#0007h          ;wfg both outputs
        ld  eo,#0001h           ;enable outputs
        ld  dp,#0001h           ;disable protection
        ld  it,#0001h           ;rising/high edge trigger
        ld  es,#0001h           ;24-state trigger
        ld  reload,#1000h       ;1.024 ms mode0
        ld  comp1,#0100h        ;64 us
        ld  comp2,#0200h        ;128 us
        ld  comp3,#0400h        ;256 us
;
;*****
; now initialize the WFG
;*****
;
;set up interrupts
;
        ldb  temp,#00010000b     ;
        stb  temp,PI_MASK[0]     ;unmask WG interrupt
        ldb  temp,#00100000b     ;
        ldb  int_mask1,temp      ;
;

```

```

;load WFG registers
;
        call wgout          ;initialize WG_OUTPUT register
        call loadregs       ;initialize reload & compare regs
        call protect        ;initialize protection
        call wgcon          ;initialize WG_CONTROL
;
;enable interrupts & loop here
;
        ei
        sjmp $
;
;*****
; form WG_OUTPUT value from variable data
;*****
;
wgout:  ld    temp,opl        ;get opl
        and  temp,#0001h     ;mask
        shl  temp,#15       ;move bit to correct location

        ld    temp1,op0      ;get op0
        and  temp1,#0001h    ;mask
        shl  temp1,#14      ;move bit to correct location
        or   temp1,temp      ;combine

        ld    temp,sync      ;get sync bit
        and  temp,#0001h     ;mask
        shl  temp,#13       ;move to correct location
        or   temp1,temp      ;combine

        ld    temp,pe7       ;get pe7 bit
        and  temp,#0001h     ;mask
        shl  temp,#12       ;move to correct location
        or   temp1,temp      ;combine

        ld    temp,pe6       ;get pe6 bit
        and  temp,#0001h     ;mask
        shl  temp,#11       ;move to correct location
        or   temp1,temp      ;combine

        ld    temp,ph3       ;get ph3 bits
        and  temp,#0004h     ;mask for ph3.2
        shl  temp,#8h        ;move
        or   temp1,temp      ;combine

        ld    temp,ph2       ;get ph2 bits
        and  temp,#0004h     ;mask for ph2.2
        shl  temp,#7h        ;move
        or   temp1,temp      ;combine

        ld    temp,ph1       ;get ph1 bits
        and  temp,#0004h     ;mask for ph1.2
        shl  temp,#6h        ;move
        or   temp1,temp      ;combine

        ld    temp,p7        ;get p7 bit
        and  temp,#0001h     ;mask
        shl  temp,#7         ;move to correct location

```

```

    or    temp1,temp        ;combine

    ld    temp,p6           ;get p6 bit
    and   temp,#0001h      ;mask
    shl   temp,#6          ;move to correct location
    or    temp1,temp       ;combine

    ld    temp,ph3         ;get ph3 bits again
    and   temp,#0003h      ;mask for ph3.0 & 1
    shl   temp,#4h        ;move
    or    temp1,temp       ;combine1

    ld    temp,ph2         ;get ph2 bits again
    and   temp,#0003h      ;mask for ph2.0 & 1
    shl   temp,#2h        ;move
    or    temp1,temp       ;combine

    ld    temp,ph1         ;get ph1 bits again
    and   temp,#0003h      ;mask for ph3.0 & 1
    or    temp1,temp       ;combine, don't need to move

    st    temp1,WG_OUTPUT[0] ;now store it
    ret

;
;*****
; form WG_CONTROL value
;*****
;
wgcon:  ld    temp,mode     ;get mode
        and   temp,#0003h   ;mask
        shl   temp,#12      ;shift to correct location

        ld    temp1,ec      ;start/stop bit
        and   temp1,#0001h  ;mask
        shl   temp1,#10     ;shift to correct location
        or    temp,temp1    ;combine into temp

        ld    temp1,dead    ;get dead time
        and   temp1,#03ffh  ;mask to 10 bits
        or    temp,temp1    ;combine into temp
        st    temp,WG_CONTROL[0] ;store WG_CONTROL
        ret

;
;*****
;load the WG_RELOAD and WG_COMPx registers
;*****
;
loadregs: st    reload,WG_RELOAD[0]
          st    comp1,WG_COMP1[0]
          st    comp2,WG_COMP2[0]
          st    comp3,WG_COMP3[0]
          ret

;
;*****
;load WG_PROTECT options
;*****
;
protect:ldtemp,es          ;get sample control bit

```

```
and temp,#0001h      ;mask
shl temp,#3          ;shift to correct location

ld temp1,it          ;interrupt type bit
and temp1,#0001h     ;mask
shl temp1,#2         ;shift to correct location
or temp,temp1        ;combine into temp

ld temp1,dp          ;disable protection bit
and temp1,#0001h     ;mask
shl temp1,#1         ;shift to correct location
or temp,temp1        ;combine into temp

ld temp1,eo          ;enable output bit
and temp1,#0001h     ;mask
or temp,temp1        ;combine into temp

stb temp,WG_PROTECT[0];store byte into WG_PROTECT
ret

;
;*****
;interrupt routine for WFG
;*****
;
cseg at 0f1a0h        ;demo board PI interrupt

pusha
call wgout           ;update WG_OUTPUT register
call loadregs        ;update reload & compare regs
call protect         ;update protection options
call wgcon           ;update WG_CONTROL
popa
ret

end
```


intel[®]

10

Pulse-width Modulator

CHAPTER 10

PULSE-WIDTH MODULATOR

The pulse-width modulator (PWM) module has two output pins, each of which can output a PWM signal 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. For detailed descriptions of the signals and registers discussed in this chapter, please refer to Appendix B, “Signal Descriptions” and Appendix C, “Registers.”

10.1 PWM FUNCTIONAL OVERVIEW

The PWM module has two channels, each of which consists of a control register (PWMx_CONTROL), a buffer, a comparator, an RS flip-flop, and an output pin. Two other components, an eight-bit counter (PWM_COUNT) and a period register (PWM_PERIOD), are shared across the PWM module’s two channels, completing the circuitry (see Figure 10-1).

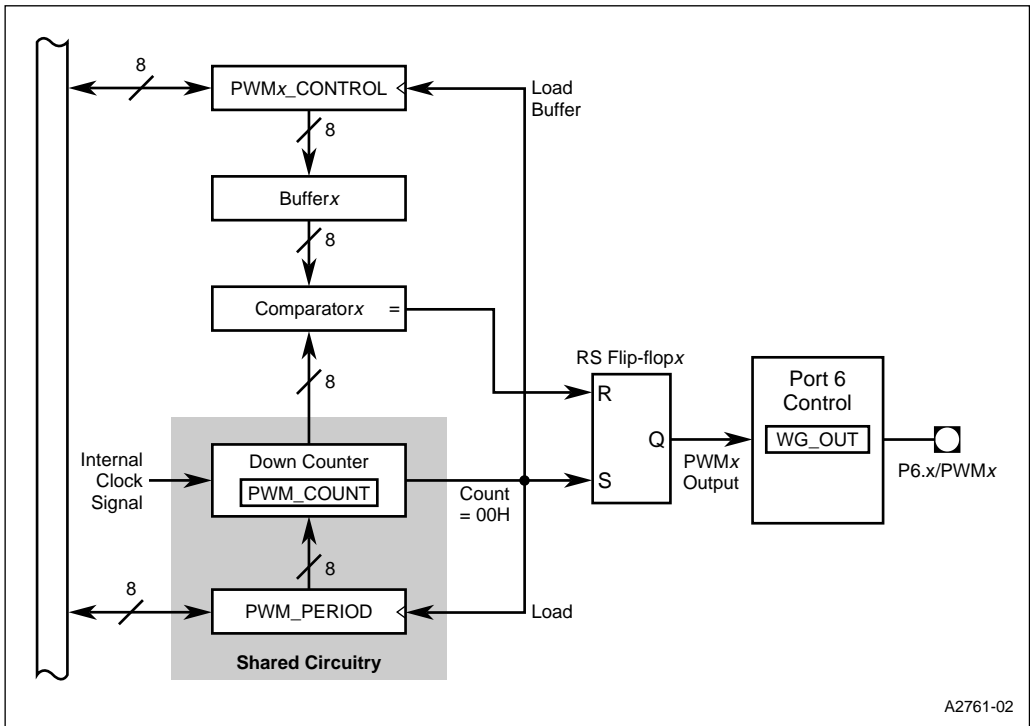


Figure 10-1. PWM Block Diagram

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
P6.6	PWM0	O	Pulse-width modulator 0 output with high-drive capability.
P6.7	PWM1	O	Pulse-width modulator 1 output with high-drive capability.

Table 10-2. PWM Control and Status Registers

Mnemonic	Address	Description												
PWM0_CONTROL PWM1_CONTROL	1FB0H 1FB2H	<p>PWM Duty Cycle</p> <p>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).</p>												
PWM_PERIOD	1FB4H	<p>PWM Period</p> <p>This register holds a programmed value that determines the output period of the PWM outputs. The value is reloaded into the counter each time the count resets to FFH.</p>												
PWM_COUNT	1FB6H	<p>PWM Counter</p> <p>This read-only register contains the current value of the decremented counter.</p>												
WG_OUTPUT	1FC0H	<p>Waveform Generator Output</p> <p>Bits 11 and 12 (PE6 and PE7) determine whether the corresponding pin functions as a standard I/O port pin or as a PWM output. Bits 6 and 7 (P6 and P7) define the pin output when the port pin function is selected.</p> <table border="1"> <thead> <tr> <th>PE_x</th> <th>P_x</th> <th>Pin Output</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>X</td> <td>PWM Output</td> </tr> </tbody> </table>	PE _x	P _x	Pin Output	0	0	0	0	1	1	1	X	PWM Output
PE _x	P _x	Pin Output												
0	0	0												
0	1	1												
1	X	PWM Output												

10.3 PWM OPERATION

The period register (PWM_PERIOD) controls the output frequency of both PWM outputs. Each control register (PWM_x_CONTROL) controls the duty cycle (the pulsewidth stated as a percentage of the period) of the corresponding PWM output. Each control register contains an 8-bit value that is loaded into a buffer when the 8-bit counter rolls over from 00H to FFH. 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 8-bit value to PWM_x_CONTROL at any time. However, the comparators do not recognize the new value until the counter has expired the remainder of the current 8-bit count. The new value is used during the next PWM output period.

The counter counts down to 00H, at which time the PWM output is driven high, the counter value is reloaded from the PWM_PERIOD register, 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_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 PWMx_CONTROL with 00H forces the output to remain low.) Figure 10-2 shows typical PWM output waveforms.

NOTE

The PWMx_CONTROL register value and corresponding duty cycle result, in Figure 10-2, are true only when the PWM_PERIOD register value is FFH.

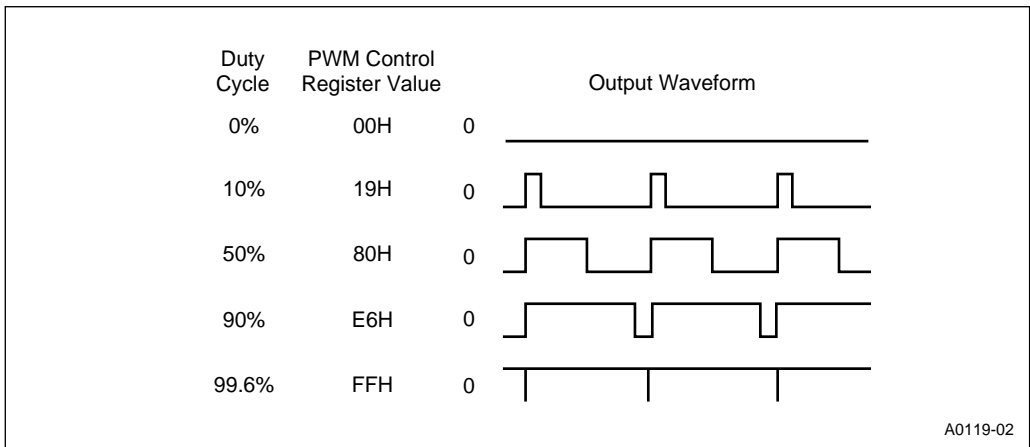


Figure 10-2. PWM Output Waveforms

10.4 PROGRAMMING THE FREQUENCY AND PERIOD

The input frequency on XTAL1 (F_{XTAL1}) and the contents of the PWM_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_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_PERIOD register.

$$T_{PWM} \text{ (in } \mu\text{s)} = \frac{512 \times (\text{PWM_PERIOD} + 1)}{F_{XTAL1}}$$

$$F_{PWM} \text{ (in MHz)} = \frac{F_{XTAL1}}{512 \times (\text{PWM_PERIOD} + 1)}$$

where:

- PWM_PERIOD = 8-bit value to load into the PWM_PERIOD register
- F_{XTAL1} = input frequency on XTAL1 pin, in MHz
- T_{PWM} = output period on the PWM output pins, in μs
- F_{PWM} = output frequency on the PWM output pins, in MHz

Table 10-3. PWM Output Frequencies (F_{PWM})

PWM_PERIOD	XTAL1 Frequency (F_{XTAL1})		
	8 MHz	10 MHz	16 MHz
00H	15.6 kHz	19.5 kHz	31.2 kHz
0FH	976.6 Hz	1220.7 Hz	1953.1 Hz
1FH	488.3 Hz	610.3 Hz	976.6 Hz
2FH	325.5 Hz	406.9 Hz	651.0 Hz
3FH	244.1 Hz	395.2 Hz	488.3 Hz
4FH	195.3 Hz	244.1 Hz	390.6 Hz
5FH	162.8 Hz	203.4 Hz	325.5 Hz
6FH	139.5 Hz	174.4 Hz	279.0 Hz
7FH	122.1 Hz	152.6 Hz	244.1 Hz
8FH	108.5 Hz	135.6 Hz	217.0 Hz
9FH	97.7 Hz	122.1 Hz	195.3 Hz
AFH	88.8 Hz	111.0 Hz	177.6 Hz
BFH	81.4 Hz	101.7 Hz	162.8 Hz
CFH	75.1 Hz	93.9 Hz	150.2 Hz
DFH	69.7 Hz	87.2 Hz	139.5 Hz
EFH	65.1 Hz	81.4 Hz	130.2 Hz
FFH	61.0 Hz	76.0 Hz	122.0 Hz

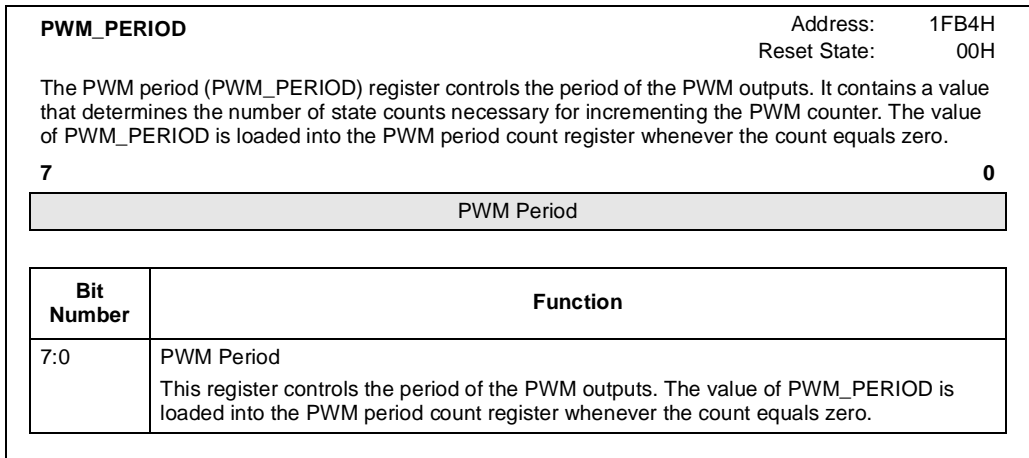


Figure 10-3. PWM Period (PWM_PERIOD) Register

10.5 PROGRAMMING THE DUTY CYCLE

The values written to the PWM_x_CONTROL and PWM_PERIOD registers control the width of the high pulse, effectively controlling the duty cycle. The 8-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 given values of PWM_x_CONTROL and PWM_PERIOD, and then write these values to the appropriate registers.

$$\text{Duty Cycle (in \%)} = \frac{\text{PWM}_x\text{_CONTROL}}{\text{PWM_PERIOD} + 1} \times 100$$

$$\text{Pulsewidth (in } \mu\text{s)} = \frac{\text{Duty Cycle} \times T_{\text{PWM}}}{100}$$

where:

- | | | |
|---------------------------|---|---|
| PWM _x _CONTROL | = | 8-bit value to load into the PWM _x _CONTROL register |
| PWM_PERIOD | = | 8-bit value to load into the PWM_PERIOD register |
| Pulsewidth | = | width of each high pulse |
| T _{PWM} | = | output period on the PWM pin, in μs |

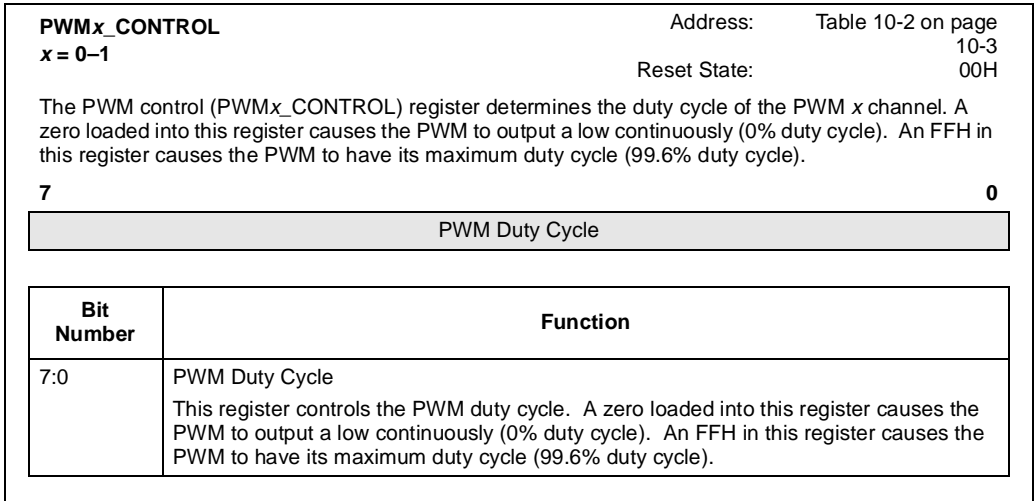


Figure 10-4. PWM Control (PWM_x_CONTROL) Register

10.5.1 Sample Calculations

For example, assume that F_{XTAL1} equals 16 MHz and the value written to the PWM_PERIOD register is FFH, thus the desired period of the PWM output waveform is 8.19 ms. If PWM_x_CONTROL equals 8AH (138 decimal), the pulsewidth is held high for 4.42 ms (and low for 3.77 ms) of the total 8.19 ms period, resulting in a duty cycle of approximately 54%.

10.5.2 Reading the Current Value of the Down-counter

You can read the PWM_COUNT register to find the current value of the down-counter (see Figure 10-5).

PWM_COUNT (read only)		Address: 1FB6H
		Reset State: 00H
The PWM count (PWM_COUNT) register provides the current value of the decremented period counter.		
7	0	
PWM Count Value		
Bit Number	Function	
7:0	PWM Count Value This register contains the current value of the decremented period counter.	

Figure 10-5. PWM Count (PWM_COUNT) Register

10.5.3 Enabling the PWM Outputs

Each PWM output is multiplexed with a port pin, so you must configure it as a special-function output signal before using the PWM function. To determine whether the corresponding pin functions as a standard I/O port pin or as a PWM output you must make the proper selections by writing to the WG_OUTPUT register (see Figure 10-6). 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	P6.6	WG_OUT.11 = 1
PWM1	P6.7	WG_OUT.12 = 1

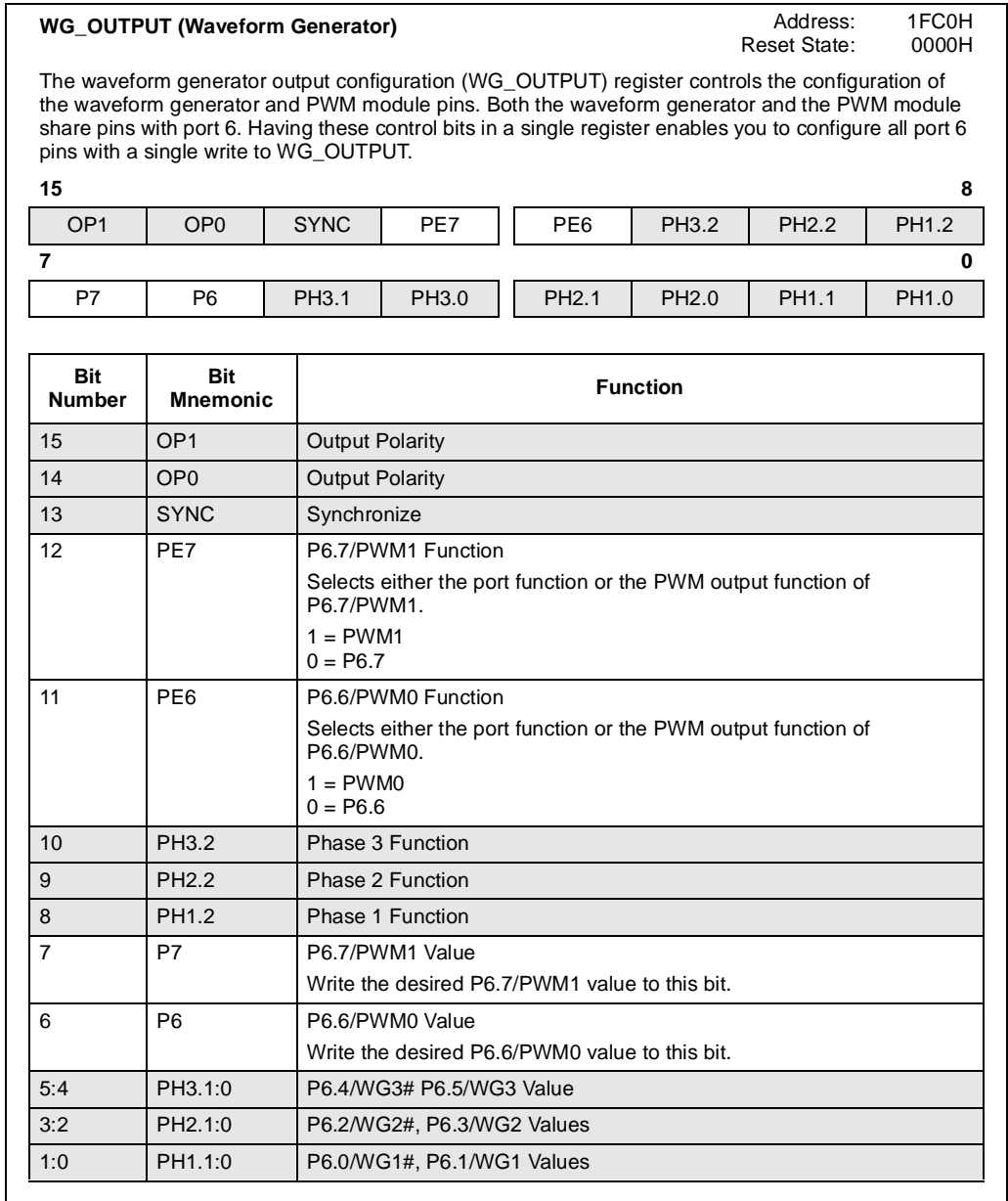


Figure 10-6. Waveform Generator Output Configuration (WG_OUTPUT) Register

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 10-7 is a block diagram of the type of circuit needed to create the smooth analog signal.

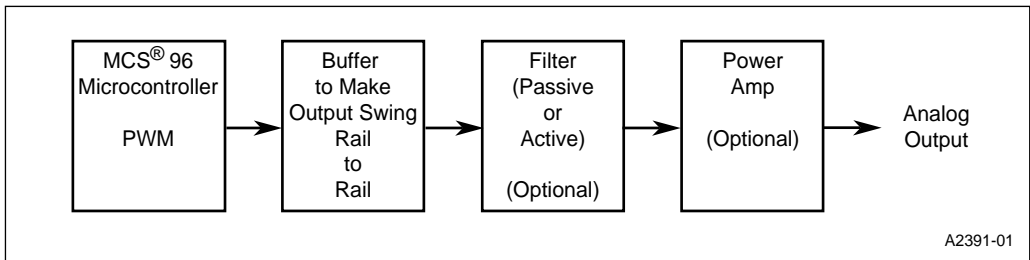


Figure 10-7. D/A Buffer Block Diagram

Figure 10-8 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.

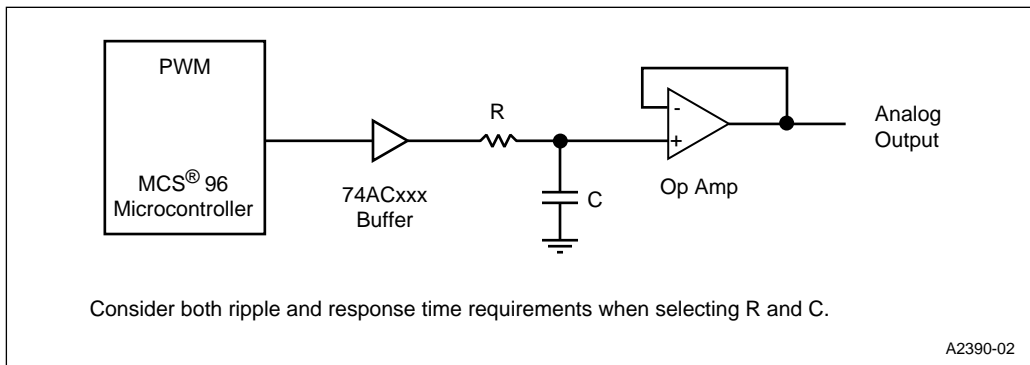


Figure 10-8. PWM to Analog Conversion Circuitry



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 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 CPU 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 two timer/counters, timer 1 and timer 2 (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, a waveform generator reload, 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 an output compare sets an interrupt pending bit. This bit can optionally cause an interrupt. Table 11-1 lists the capture/compare and compare-only channels for each device in the 8XC196Mx family.

Table 11-1. EPA Channels

Device	Capture/Compare Channels	Compare-only Channels
8XC196MC	EPA3:0	COMP3:0
8XC196MD	EPA5:0	COMP5:0
8XC196MH	EPA1:0	COMP3:0

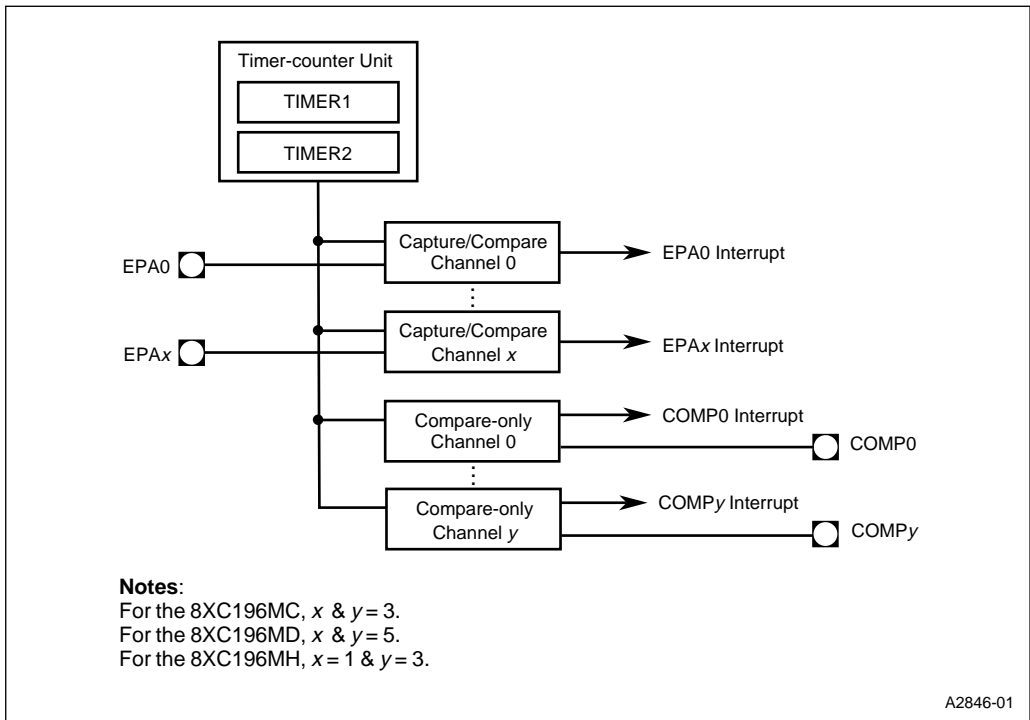


Figure 11-1. EPA Block Diagram

11.2 EPA AND TIMER/COUNTER SIGNALS AND REGISTERS

Table 11-2 describes the EPA and timer/counter input and output signals. Each signal is multiplexed with a port pin as shown in the first column. Table 11-3 briefly describes the registers for the EPA capture/compare channels, EPA compare-only channels, and timer/counters.

Table 11-2. EPA and Timer/Counter Signals

Port Pin			EPA Signals	EPA Signal Type	Description
8XC196MC	8XC196MD	8XC196MH			
P1.2	P1.2	P0.6	T1CLK	I	External clock source for timer 1.
P1.3	P1.3	P0.7	T1DIR	I	External direction control for timer 1.
P2.0	P2.0	P2.0	EPA0	I/O	High-speed input/output for the capture/compare channels.
P2.1	P2.1	P2.2	EPA1		
P2.2	P2.2	—	EPA2		
P2.3	P2.3	—	EPA3		
—	P7.0	—	EPA4		
—	P7.1	—	EPA5		

Table 11-2. EPA and Timer/Counter Signals (Continued)

Port Pin			EPA Signals	EPA Signal Type	Description
8XC196MC	8XC196MD	8XC196MH			
P2.4	P2.4	P2.4	COMP0	O	Output of the compare-only channels.
P2.5	P2.5	P2.5	COMP1		
P2.6	P2.6	P2.6	COMP2		
P2.7	P2.7	P2.3	COMP3		
—	P7.2	—	COMP4		
—	P7.3	—	COMP5		

Table 11-3. EPA Control and Status Registers

Mnemonic	Address			Description
	MC	MD	MH	
COMP0_CON COMP1_CON COMP2_CON COMP3_CON COMP4_CON COMP5_CON	1F58H 1F5CH 1F60H 1F64H — —	1F58H 1F5CH 1F60H 1F64H 1F68H 1F6CH	1F58H 1F5CH 1F60H 1F4CH — —	EPAx Compare Control These registers control the functions of the compare-only channels.
COMP0_TIME COMP1_TIME COMP2_TIME COMP3_TIME COMP4_TIME COMP5_TIME	1F5AH 1F5EH 1F62H 1F66H — —	1F5AH 1F5EH 1F62H 1F66H 1F6AH 1F6EH	1F5AH 1F5EH 1F62H 1F4EH — —	EPAx Compare Time These registers contain the time at which an event is to occur on the compare-only channels.
EPA0_CON EPA1_CON EPA2_CON EPA3_CON EPA4_CON EPA5_CON	1F40H 1F44H 1F48H 1F4CH — —	1F40H 1F44H 1F48H 1F4CH 1F50H 1F54H	1F40H 1F44H — — — —	EPAx Capture/Compare Control These registers control the functions of the capture/compare channels. EPA1_CON and EPA3_CON require an extra byte because they contain an additional bit for PWM remap mode. These two registers must be addressed as words; the others can be addressed as bytes.
EPA0_TIME EPA1_TIME EPA2_TIME EPA3_TIME EPA4_TIME EPA5_TIME	1F42H 1F46H 1F4AH 1F4EH — —	1F42H 1F46H 1F4AH 1F4EH 1F52H 1F56H	1F42H 1F46H — — — —	EPAx 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. However, they are not buffered in compare mode.
INT_MASK	0008H	0008H	0008H	Interrupt Mask The bits in this 8-bit register enable and disable (mask) the interrupts associated with the corresponding bits in the INT_PEND register.
INT_MASK1	0013H	0013H	0013H	Interrupt Mask 1 The bits in this 8-bit register enable and disable (mask) the interrupts associated with the corresponding bits in the INT_PEND1 register.

Table 11-3. EPA Control and Status Registers (Continued)

Mnemonic	Address			Description
	MC	MD	MH	
INT_PEND	0009H	0009H	0009H	Interrupt Pending Any set bit in this 8-bit register indicates a pending interrupt request.
INT_PEND1	0012H	0012H	0012H	Interrupt Pending 1 Any set bit in this 8-bit register indicates a pending interrupt request.
P2_DIR P7_DIR	1FD2H —	1FD2H 1FD3H	1FD2H —	Port x Direction Each bit of Px_DIR controls the direction of the corresponding pin. Clearing a bit configures a pin as a complementary output; setting a bit configures a pin as an input or open-drain output. (Open-drain outputs require external pull-ups.)
P2_MODE P7_MODE	1FD0H —	1FD0H 1FD1H	1FD0H —	Port x Mode Each bit of Px_MODE controls whether the corresponding pin functions as a standard I/O port pin or as a special-function signal. Setting a bit configures a pin as a special-function signal; clearing a bit configures a pin as a standard I/O port pin.
P0_PIN P1_PIN P2_PIN P7_PIN	1FA8H 1FA9H 1FD6H —	1FA8H 1FA9H 1FD6H 1FD7H	1FDAH 1F9FH 1FD6H —	Port x Input Each bit of Px_PIN reflects the current state of the corresponding pin, regardless of the pin configuration.
P2_REG P7_REG	1FD4H —	1FD4H 1FD5H	1FD4H —	Port x Data Output For an input, set the corresponding Px_REG bit. For an output, write the data to be driven out by each pin to the corresponding bit of Px_REG. When a pin is configured as standard I/O (Px_MODE.y = 0), the result of a CPU write to Px_REG is immediately visible on the pin. When a pin is configured as a special-function signal (Px_MODE.y = 1), the associated on-chip peripheral or off-chip component controls the pin. The CPU can still write to Px_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 standard I/O (clear Px_MODE.y), initialize or overwrite the pin value, then configure the pin as a special-function signal (set Px_MODE.y). In this way, initialization, fault recovery, exception handling, etc., can be done without changing the operation of the associated peripheral.

Table 11-3. EPA Control and Status Registers (Continued)

Mnemonic	Address			Description
	MC	MD	MH	
PI_MASK	1FBCH	1FBCH	1FBCH	Peripheral Interrupt Mask The bits in this register enable and disable (mask) the timer 1 and 2 overflow/underflow interrupt requests, the waveform generator interrupt request (MC, MD), the EPA compare-only channel 5 interrupt request (MD), and the serial port error interrupts (MH).
PI_PEND	1FBEH	1FBEH	1FBEH	Peripheral Interrupt Pending Any bit set indicates a pending interrupt request.
T1CONTROL	1F78H	1F78H	1F78H	Timer 1 Control This register enables/disables timer 1, controls whether it counts up or down, selects the clock source and direction, and determines the clock prescaler setting.
T1RELOAD	1F72H	1F72H	1F72H	Timer 1 Reload This register contains an initialization value for timer 1. A timer 1 overflow or underflows loads the T1RELOAD value into the TIMER1 register if both quadrature clocking and the reload function are enabled (T1CONTROL.5:0 = 1).
T2CONTROL	1F7CH	1F7CH	1F7CH	Timer 2 Control This register enables/disables timer 2, controls whether it counts up or down, selects the clock source and direction, and determines the clock prescaler setting.
TIMER1	1F7AH	1F7AH	1F7AH	Timer 1 Value This register contains the current value of timer 1.
TIMER2	1F7EH	1F7EH	1F7EH	Timer 2 Value This register contains the current value of timer 2.

11.3 TIMER/COUNTER FUNCTIONAL OVERVIEW

The EPA has two 16-bit up/down timer/counters, timer 1 and timer 2, 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.

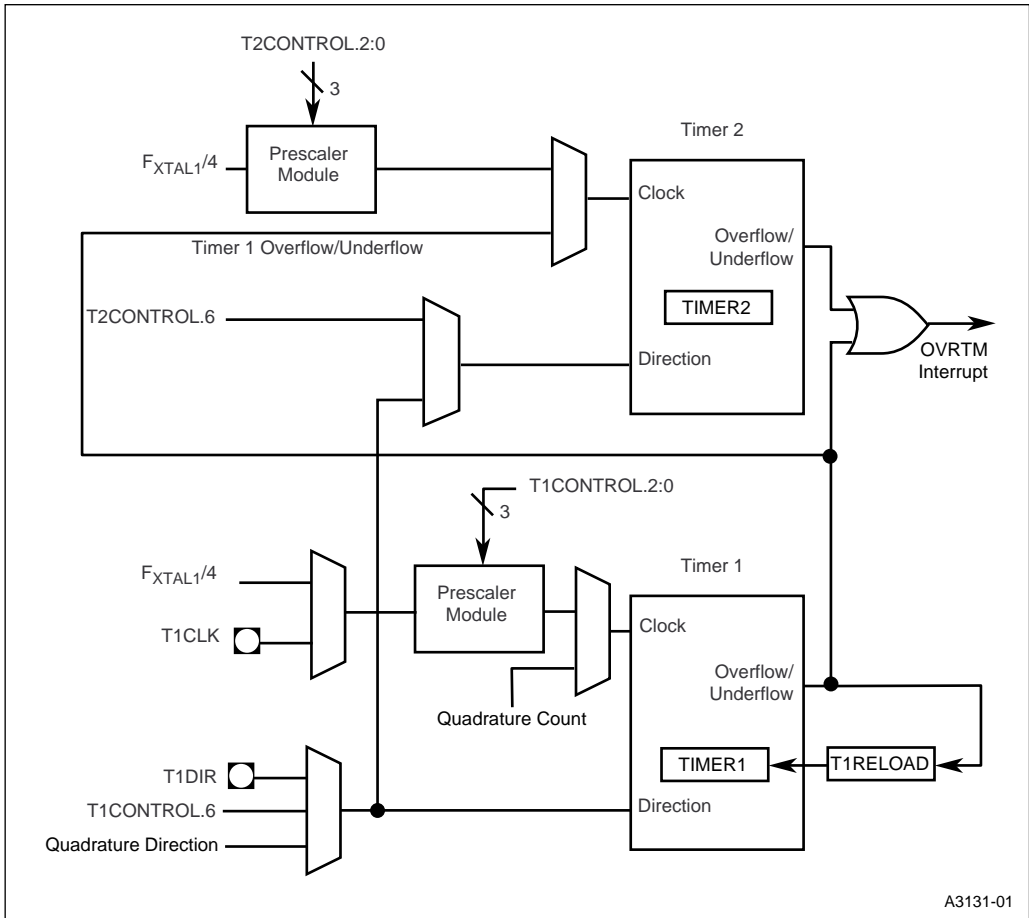


Figure 11-2. EPA Timer/Counters

The timer/counters can be used as time bases for input captures, output compares, and programmed interrupts (software timers). When a counter increments from FFFE_H to FFFF_H or decrements from 0001_H to 0000_H, the counter-overflow/underflow interrupt pending bit is set. This bit can optionally cause an interrupt. The clock source, direction-control 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 one-half the internal clock rate, or $F_{XTAL1}/4$ (see “Internal Timing” on page 2-7). This provides a minimum resolution for an input capture or output compare of 250 ns (at 16 MHz).

$$\text{resolution} = \frac{4 \times \text{prescaler_divisor}}{F_{XTAL1}}$$

where:

prescaler_divisor is the clock prescaler divisor from the TxCONTROL registers (see “Timer 1 Control (T1CONTROL) Register” on page 11-16 and “Timer 2 Control (T2CONTROL) Register” on page 11-17).

F_{XTAL1} is the input frequency on XTAL1.

11.3.1 Cascade Mode (Timer 2 Only)

Timer 2 can be used in cascade mode. In this mode, the timer 1 overflow output is used as the timer 2 clock input. Either the direction control bit of the timer 2 control register or the direction control assigned to timer 1 controls the count direction. This method, called *cascading*, can provide a slow clock for idle mode timeout control or for slow pulse-width modulation (PWM) applications (see “Generating a Low-speed PWM Output” on page 11-13).

11.3.2 Quadrature Clocking Modes

Timer 1 can be used in two quadrature clocking modes. Both modes use the T1CLK and T1DIR pins as quadrature inputs, as shown in Figure 11-3. External quadrature-encoded signals (two signals at the same frequency that differ in phase by 90°) are input, and the timer increments or decrements by one count on each rising edge and each falling edge. Because the T1CLK and T1DIR inputs are sampled by the internal phase clocks, transitions must be separated by at least two state times for proper operation. The count is clocked by PH2, which is PH1 delayed by one-half period. The sequence of the signal edges and levels controls the count direction. Refer to Figure 11-4 and Table 11-4 for sequencing information.

A typical source of quadrature-encoded signals is a shaft-angle decoder, shown in Figure 11-3. Its output signals X and Y are input to T1CLK and T1DIR, which in turn output signals X_internal and Y_internal. These signals are used in Figure 11-4 and Table 11-4 to describe the direction of the shaft.

In the default quadrature clocking mode, software must reload the TIMER1 register when timer 1 overflows or underflows. In mode 2 (T1CONTROL.2:0 = 1), timer 1 automatically loads the value from the T1RELOAD register into the TIMER1 register when an overflow or underflow occurs. Mode 2 is useful for interfacing to an incremental shaft encoder that turns in only one direction. For this application, initialize T1RELOAD with a value that is one less than the encoder’s resolution. This method allows timer 1 to track the absolute position of the shaft encoder with no software overhead.

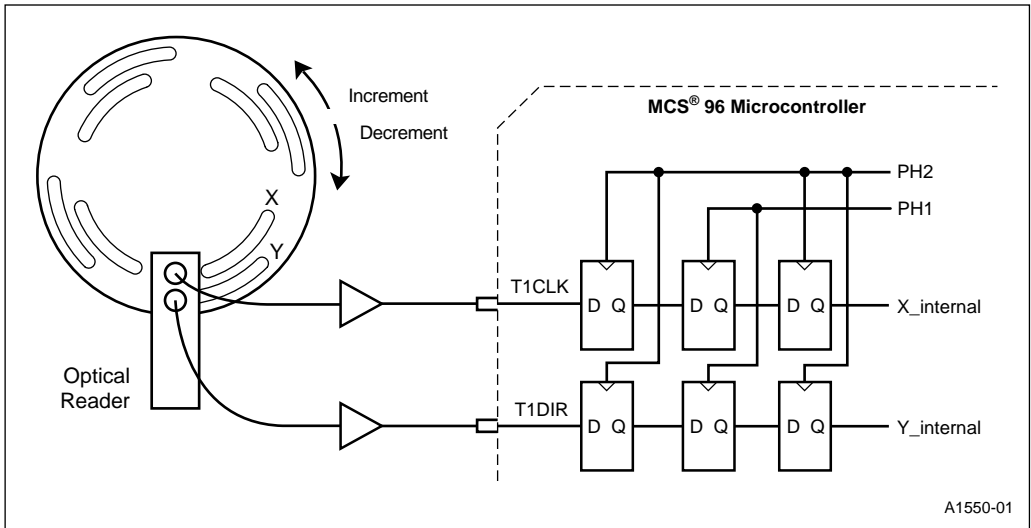


Figure 11-3. Quadrature Mode Interface

Table 11-4. Quadrature Mode Truth Table

State of X_internal (T1CLK)	State of Y_internal (T1DIR)	Count Direction
↑	0	Increment
↓	1	Increment
0	↓	Increment
1	↑	Increment
↓	0	Decrement
↑	1	Decrement
0	↑	Decrement
1	↓	Decrement

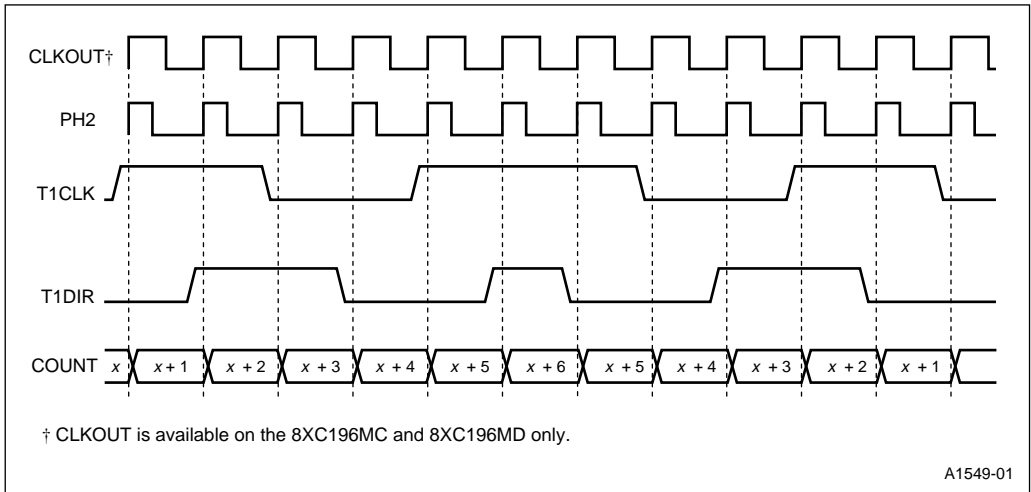


Figure 11-4. Quadrature Mode Timing and Count

11.4 EPA CHANNEL FUNCTIONAL OVERVIEW

The EPA has both programmable capture/compare and 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.)

- capture the current timer value when a specified transition occurs on the EPA pin
- start an A/D conversion or reload the waveform generator when an event is captured or 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
- reset its own base timer in compare mode
- reset the opposite timer in both compare and capture mode

Each EPA channel has a control register, EPA_x_CON (capture/compare channels) or COMP_x_CON (compare-only channels); an event-time register, EPA_x_TIME (capture/compare channels) or COMP_x_TIME (compare-only channels); and a timer input (Figure 11-5). The control register selects the timer, the mode, and either the event to be captured or the event that is to occur. The event-time register holds the captured timer value in capture mode and the event time in compare mode. See “Programming the Capture/Compare Channels” on page 11-18 and “Programming the Compare-only Channels” on page 11-22 for configuration information.

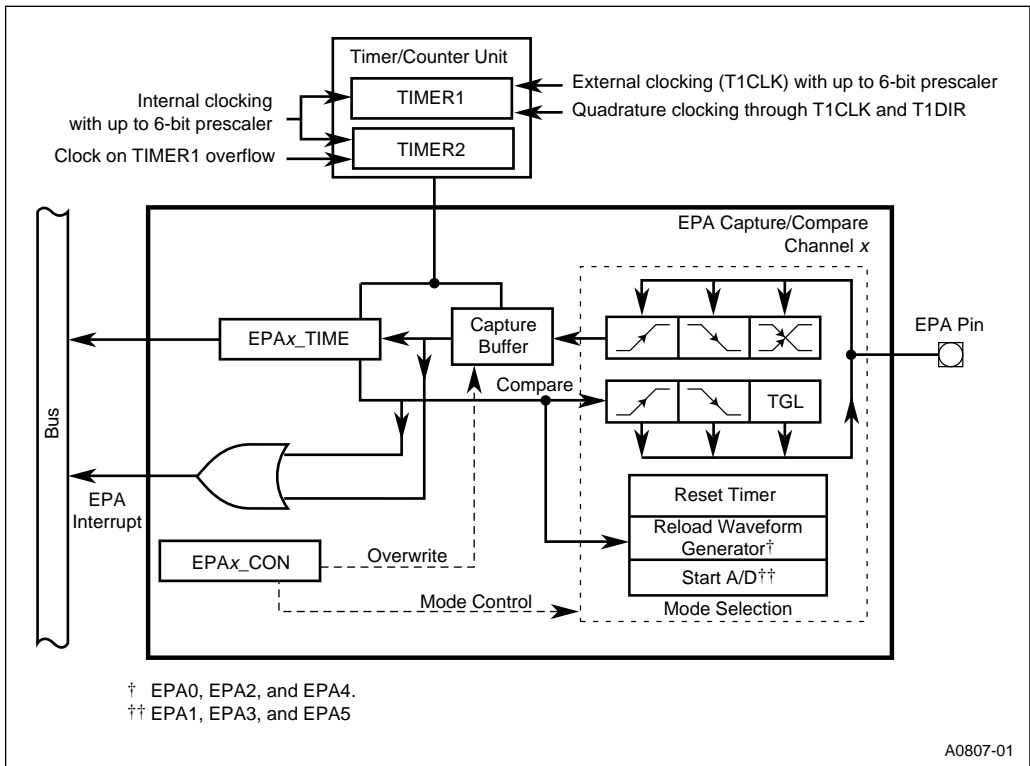


Figure 11-5. A Single EPA Capture/Compare Channel

11.4.1 Operating in 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 11-6. If enabled, an interrupt is generated. If a second event occurs before the CPU reads the first timer value in EPA_x_TIME, the current timer value is loaded into the buffer and held there. After the CPU reads the EPA_x_TIME register, the contents of the capture buffer are automatically transferred into EPA_x_TIME and the EPA interrupt pending bit is set.

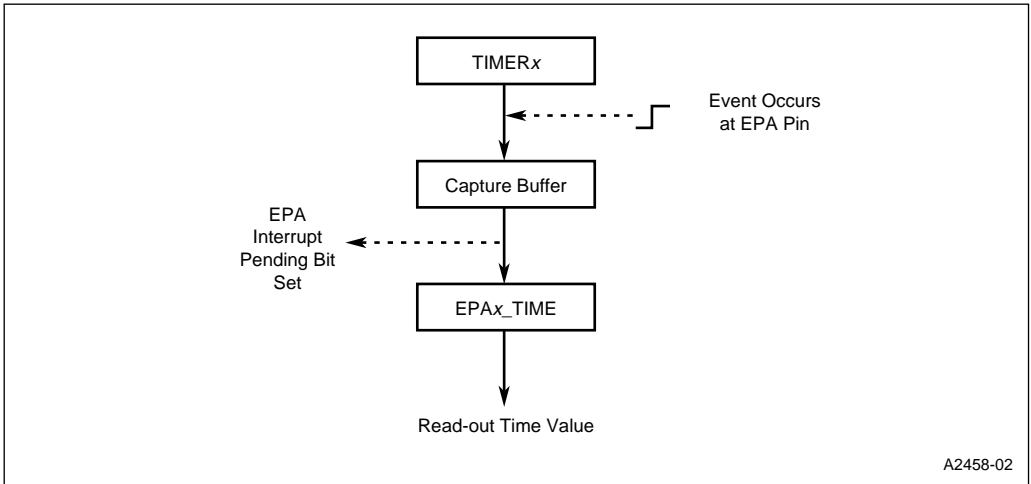


Figure 11-6. EPA Simplified Input-capture Structure

If a third event occurs before the CPU reads the event-time register, the overwrite bit (EPAx_CON.0) 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 11-5 summarizes the possible actions when a valid event occurs.

NOTE

In order 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 11-7).

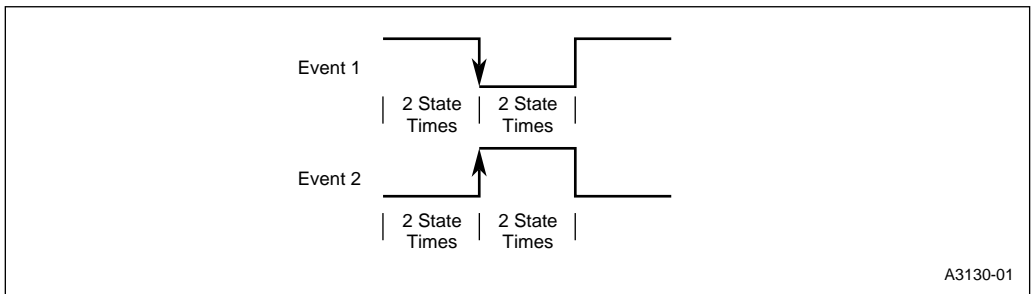


Figure 11-7. Valid EPA Input Events

Table 11-5. Action Taken When a Valid Edge Occurs

Overwrite Bit (EPAx_CON.0)	Status of Capture Buffer & EPAx_TIME	Action Taken When a Valid Edge Occurs
0	empty	Edge is captured and event time is loaded into the capture buffer and EPAx_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 EPAx_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 EPAx_TIME register. If the buffer contains data and the PTS is used to service the interrupts, then two PTS interrupts occur almost back-to-back (that is, with one instruction executed between the interrupts).

11.4.1.1 EPA Overruns

Overruns occur when an EPA input transitions at a rate that cannot be handled by the EPA interrupt service routine. If no overrun handling strategy is in place, and if the following three conditions exist, a situation may occur where both the capture buffer and the EPAx_TIME register contain data, and no EPA interrupt is generated.

- an input signal with a frequency high enough to cause overruns is present on an enabled EPA pin, and
- the overwrite bit is set (EPAx_CON.0 = 1; old data is overwritten on overrun), and
- the EPAx_TIME register is read at the exact instant that the EPA recognizes the captured edge as valid.

The input frequency at which this occurs depends on the length of the interrupt service routine as well as other factors. Unless the interrupt service routine includes a check for overruns, this situation will remain the same until the device is reset or the EPAx_TIME register is read. The act of reading EPAx_TIME allows the buffered time value to be moved into EPAx_TIME. This clears the buffer and allows another event to be captured. Remember that the act of the transferring the buffer contents to the EPAx_TIME register is what actually sets the EPAx interrupt pending bit and generates the interrupt.

11.4.1.2 Preventing EPA Overruns

Any one of the following methods can be used to prevent or recover from an EPA overrun situation.

- Clear EPA_x_CON.0

When the overwrite bit (EPA_x_CON.0) is zero, the EPA does not consider the captured edge until the EPA_x_TIME register is read and the data in the capture buffer is transferred to EPA_x_TIME. This prevents overruns by ignoring new input capture events when both the capture buffer and EPA_x_TIME contain valid capture times.

- Check for pending EPA_x interrupts before exiting an EPA_x ISR

Another method for avoiding this situation is to check for pending EPA interrupts before exiting the EPA interrupt service routine. This is an easy way to detect overruns and additional interrupts. It can also save loop time by eliminating the latency necessary to service the pending interrupt. However, this method cannot be used with the peripheral transaction server (PTS).

11.4.2 Operating in Compare Mode

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, an A/D conversion is initiated, or the waveform generator is reloaded). If the re-enable bit (EPA_x_CON.3) 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 11-18 and “Programming the Compare-only Channels” on page 11-22 for configuration information.

In compare mode, you can use the EPA to produce a pulse-width modulated (PWM) output. The following sections describe two possible methods.

11.4.2.1 Generating a Low-speed PWM Output

You can generate a low-speed, pulse-width modulated output with a single EPA channel and a standard interrupt service routine. Configure the EPA channel as follows: compare mode, toggle output, and the compare function re-enabled. Select standard interrupt service, enable the EPA interrupt, and globally enable interrupts with the EI instruction. When the assigned timer/counter value matches the value in the event-time register, the EPA toggles the output pin and generates an interrupt. The interrupt service routine loads a new value into EPA_x_TIME.

The maximum output frequency depends upon the total interrupt latency and the interrupt-service execution times used by your system. As additional EPA channels and the other functions of the microcontroller are used, the maximum PWM frequency decreases because the total interrupt latency and interrupt-service execution time increases. To determine the maximum, low-speed PWM frequency in your system, calculate your system's worst-case interrupt latency and worst-case interrupt-service execution time, and then add them together. The worst-case interrupt latency is the total latency of all the interrupts (both normal and PTS) used in your system. The worst-case interrupt-service execution time is the total execution time of all interrupt service routines and PTS routines.

Assume a system with a single EPA channel, a single enabled interrupt, and the following interrupt service routine.

```
;If EPA0-x interrupt is generated
EPA0-x_ISR:
    PUSHA
    LD EPAX_CON, #toggle_command
    ADD EPAX_TIME, TIMERx, [next_duty_ptr]; Load next event time
    POPA
    RET
```

The worst-case interrupt latency for a single-interrupt system is 56 state times for external stack usage and 54 state times for internal stack usage (see “Standard Interrupt Latency” on page 5-10). To determine the execution time for an interrupt service routine, add up the execution time of the instructions (Table A-9).

The total execution time for the ISR that services the EPA interrupts is 79 state times for external stack usage or 71 state times for internal stack usage. Therefore, a single capture/compare channel can be updated every 125 state times assuming internal stack usage (54 + 71). Each PWM period requires two updates (one setting and one clearing), so the execution time for a PWM period equals 250 state times. When the input frequency on XTAL1 is 16 MHz, the PWM period is 31.25 μ s and the maximum PWM frequency is 32 kHz.

11.4.2.2 Generating the Highest-speed PWM Output

You can generate a highest-speed, pulse-width modulated output with a pair of EPA channels and a dedicated timer/counter. The first channel toggles the output when the timer value matches `EPAX_TIME`, and at some later time, the second channel toggles the output again **and** resets the timer/counter. This restarts the cycle. No interrupts are required, resulting in the highest possible speed. Software must calculate and load the appropriate `EPAX_TIME` values and load them at the correct time in the cycle in order to change the frequency or duty cycle.

With this method, the resolution of the EPA (selected by the TxCONTROL registers; see Figure 11-8 on page 11-16 and Figure 11-9 on page 11-17) determines the maximum PWM output frequency. (Resolution is the minimum time required between consecutive captures or compares.) When the input frequency on XTAL1 is 16 MHz, a 250 ns resolution results in a maximum PWM of 4 MHz.

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 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 “Bidirectional Ports 1 (MH Only), 2, 5, and 7 (MD Only)” on page 6-4 for information about configuring the ports.

Table 11-2 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 standard I/O.

11.5.2 Programming the Timers

The control registers for the timers are T1CONTROL (Figure 11-8) and T2CONTROL (Figure 11-9). Write to these registers to configure the timers. Write to the TIMER1 and TIMER2 registers (see Table 11-3 on page 11-3 for addresses) to load a specific timer value.

T1CONTROL

Address: 1F78H

Reset State: 00H

The timer 1 control (T1CONTROL) register determines the clock source, counting direction, and count rate for timer 1.

7**0**

CE	UD	M2	M1	M0	P2	P1	P0
----	----	----	----	----	----	----	----

Bit Number	Bit Mnemonic	Function																																													
7	CE	Counter Enable This bit enables or disables the timer. From reset, the timers are disabled and not free running. 0 = disables timer 1 = enables timer																																													
6	UD	Up/Down This bit determines the timer counting direction, in selected modes (see mode bits, M2:0). 0 = count down 1 = count up																																													
5:3	M2:0	EPA Clock Direction Mode Bits These bits determine the timer clocking source and direction control source. <table border="1"> <thead> <tr> <th>M2</th> <th>M1</th> <th>M0</th> <th>Clock Source</th> <th>Direction Source</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>F_{XTAL}1/4</td> <td>UD bit (T1CONTROL.6)</td> </tr> <tr> <td>X</td> <td>0</td> <td>1</td> <td>T1CLK pin[†]</td> <td>UD bit (T1CONTROL.6)</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> <td>F_{XTAL}1/4</td> <td>T1DIR pin</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> <td>T1CLK pin[†]</td> <td>T1DIR pin</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td colspan="2">quadrature clocking using T1CLK and T1DIR</td> </tr> </tbody> </table> [†] If an external clock is selected, the timer counts on both the rising and falling edges of the clock.	M2	M1	M0	Clock Source	Direction Source	0	0	0	F _{XTAL} 1/4	UD bit (T1CONTROL.6)	X	0	1	T1CLK pin [†]	UD bit (T1CONTROL.6)	0	1	0	F _{XTAL} 1/4	T1DIR pin	0	1	1	T1CLK pin [†]	T1DIR pin	1	1	1	quadrature clocking using T1CLK and T1DIR																
M2	M1	M0	Clock Source	Direction Source																																											
0	0	0	F _{XTAL} 1/4	UD bit (T1CONTROL.6)																																											
X	0	1	T1CLK pin [†]	UD bit (T1CONTROL.6)																																											
0	1	0	F _{XTAL} 1/4	T1DIR pin																																											
0	1	1	T1CLK pin [†]	T1DIR pin																																											
1	1	1	quadrature clocking using T1CLK and T1DIR																																												
2:0	P2:0	EPA Clock Prescaler Bits These bits determine the clock prescaler value. <table border="1"> <thead> <tr> <th>P2</th> <th>P1</th> <th>P0</th> <th>Prescaler Divisor</th> <th>Resolution[†]</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>divide by 1 (disabled)</td> <td>250 ns</td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> <td>divide by 2</td> <td>500 ns</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> <td>divide by 4</td> <td>1 μs</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> <td>divide by 8</td> <td>2 μs</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>divide by 16</td> <td>4 μs</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> <td>divide by 32</td> <td>8 μs</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> <td>divide by 64</td> <td>16 μs</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>enable T1RELOAD</td> <td>—</td> </tr> </tbody> </table> [†] At 16 MHz. Use the formula on page 11-6 to calculate the resolution at other frequencies.	P2	P1	P0	Prescaler Divisor	Resolution [†]	0	0	0	divide by 1 (disabled)	250 ns	0	0	1	divide by 2	500 ns	0	1	0	divide by 4	1 μs	0	1	1	divide by 8	2 μs	1	0	0	divide by 16	4 μs	1	0	1	divide by 32	8 μs	1	1	0	divide by 64	16 μs	1	1	1	enable T1RELOAD	—
P2	P1	P0	Prescaler Divisor	Resolution [†]																																											
0	0	0	divide by 1 (disabled)	250 ns																																											
0	0	1	divide by 2	500 ns																																											
0	1	0	divide by 4	1 μs																																											
0	1	1	divide by 8	2 μs																																											
1	0	0	divide by 16	4 μs																																											
1	0	1	divide by 32	8 μs																																											
1	1	0	divide by 64	16 μs																																											
1	1	1	enable T1RELOAD	—																																											

Figure 11-8. Timer 1 Control (T1CONTROL) Register

T2CONTROL				Address: 1F7CH																																																
				Reset State: 00H																																																
The timer 2 control (T2CONTROL) register determines the clock source, counting direction, and count rate for timer 2.																																																				
7 0																																																				
CE	UD	M2	M1	M0	P2	P1	P0																																													
Bit Number	Bit Mnemonic	Function																																																		
7	CE	Counter Enable This bit enables or disables the timer. From reset, the timers are disabled and not free running. 0 = disables timer 1 = enables timer																																																		
6	UD	Up/Down This bit determines the timer counting direction, in selected modes (see mode bits, M2:0). 0 = count down 1 = count up																																																		
5:3	M2:0	EPA Clock Direction Mode Bits These bits determine the timer clocking source and direction source. <table border="1" style="width: 100%; border-collapse: collapse; margin-top: 5px;"> <thead> <tr> <th style="width: 10%;">M2</th> <th style="width: 10%;">M1</th> <th style="width: 10%;">M0</th> <th style="width: 30%;">Clock Source</th> <th style="width: 30%;">Direction Source</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td>F_{XTAL1}/4</td> <td>UD bit (T2CONTROL.6)</td> </tr> <tr> <td style="text-align: center;">X</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td>reserved</td> <td>—</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td>reserved</td> <td>—</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td>reserved</td> <td>—</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td>timer 1 overflow</td> <td>UD bit (T2CONTROL.6)</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td>timer 1 overflow</td> <td>same as timer 1</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td>reserved</td> <td>—</td> </tr> </tbody> </table>						M2	M1	M0	Clock Source	Direction Source	0	0	0	F _{XTAL1} /4	UD bit (T2CONTROL.6)	X	0	1	reserved	—	0	1	0	reserved	—	0	1	1	reserved	—	1	0	0	timer 1 overflow	UD bit (T2CONTROL.6)	1	1	0	timer 1 overflow	same as timer 1	1	1	1	reserved	—					
M2	M1	M0	Clock Source	Direction Source																																																
0	0	0	F _{XTAL1} /4	UD bit (T2CONTROL.6)																																																
X	0	1	reserved	—																																																
0	1	0	reserved	—																																																
0	1	1	reserved	—																																																
1	0	0	timer 1 overflow	UD bit (T2CONTROL.6)																																																
1	1	0	timer 1 overflow	same as timer 1																																																
1	1	1	reserved	—																																																
2:0	P2:0	EPA Clock Prescaler Bits These bits determine the clock prescaler value. <table border="1" style="width: 100%; border-collapse: collapse; margin-top: 5px;"> <thead> <tr> <th style="width: 10%;">P2</th> <th style="width: 10%;">P1</th> <th style="width: 10%;">P0</th> <th style="width: 30%;">Prescaler</th> <th style="width: 30%;">Resolution[†]</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td>divide by 1 (disabled)</td> <td>250 ns</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td>divide by 2</td> <td>500 ns</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td>divide by 4</td> <td>1 μs</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td>divide by 8</td> <td>2 μs</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td>divide by 16</td> <td>4 μs</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td>divide by 32</td> <td>8 μs</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td>divide by 64</td> <td>16 μs</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td>reserved</td> <td>—</td> </tr> </tbody> </table> <p style="margin-top: 5px;">[†] Resolution at 16 MHz. Use the formula on page 11-6 to calculate the resolution at other frequencies.</p>						P2	P1	P0	Prescaler	Resolution [†]	0	0	0	divide by 1 (disabled)	250 ns	0	0	1	divide by 2	500 ns	0	1	0	divide by 4	1 μs	0	1	1	divide by 8	2 μs	1	0	0	divide by 16	4 μs	1	0	1	divide by 32	8 μs	1	1	0	divide by 64	16 μs	1	1	1	reserved	—
P2	P1	P0	Prescaler	Resolution [†]																																																
0	0	0	divide by 1 (disabled)	250 ns																																																
0	0	1	divide by 2	500 ns																																																
0	1	0	divide by 4	1 μs																																																
0	1	1	divide by 8	2 μs																																																
1	0	0	divide by 16	4 μs																																																
1	0	1	divide by 32	8 μs																																																
1	1	0	divide by 64	16 μs																																																
1	1	1	reserved	—																																																

Figure 11-9. Timer 2 Control (T2CONTROL) Register

11.5.3 Programming the Capture/Compare Channels

The EPA_x_CON register controls the function of its assigned capture/compare channel. The registers are identical with the exception of bit 2. For EPA channels 0, 2, and 4, setting this bit enables an EPA event to cause a waveform generator reload. For EPA channels 1, 3, and 5, setting this bit enables an EPA event to cause an A/D conversion. To program a compare event, always write to EPA_x_CON (Figure 11-10) first to configure the EPA capture/compare channel, and then load the event time into EPA_x_TIME. To program a capture event, you need only write to EPA_x_CON. Table 11-6 shows the effects of various combinations of EPA_x_CON bit settings for channels 1, 3, or 5.

Table 11-6. Example EPA Control Register Settings for Channels 1, 3, or 5

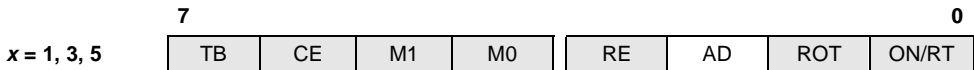
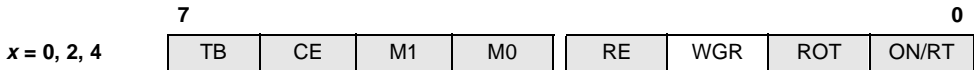
Capture Mode								
TB	CE	MODE		RE	WGR /AD	ROT	ON/RT	Operation
7	6	5	4	3	2	1	0	
X	0	0	0	—	—	—	0	None
X	0	0	1	—	X	X	X	Capture on falling edges
X	0	1	0	—	X	X	X	Capture on rising edges
X	0	1	1	—	X	X	X	Capture on both edges
X	0	X	1	—	X	1	X	Capture on falling edge and reset opposite timer
X	0	1	X	—	X	1	X	Capture on rising edge and reset opposite timer
X	0	0	1	—	1	X	X	Start A/D conversion (EPA1, 3, 5) or reload waveform generator (EPA0, 2, 4) on falling edge
X	0	1	0	—	1	X	X	Start A/D conversion (EPA1, 3, 5) or reload waveform generator (EPA0, 2, 4) on rising edge
Compare Mode								
TB	CE	MODE		RE	WGR /AD	ROT	ON/RT	Operation
7	6	5	4	3	2	1	0	
X	1	0	0	X	—	—	0	None
X	1	0	0	X	0	X	0	Generate interrupt only (software timer)
X	1	0	1	X	X	X	X	Clear output pin
X	1	1	0	X	X	X	X	Set output pin
X	1	1	1	X	X	X	X	Toggle output pin
X	1	X	X	X	X	0	1	Reset reference timer
X	1	X	X	X	X	1	1	Reset opposite timer
X	1	X	X	X	1	X	X	Start A/D conversion (EPA1, 3, 5) or reload waveform generator (EPA0, 2, 4)

NOTES:

1. — = bit is not used
2. X = bit may be used, but has no effect on the described operation. These bits cause other operations to occur.

EPA_x_CON
x = 0–1 (8XC196MH)
x = 0–3 (8XC196MC)
x = 0–5 (8XC196MD)

 Address: See Table 11-3 on page 11-3
 Reset State: 00H

 The EPA control (EPA_x_CON) registers control the functions of their assigned capture/compare channels.


Bit Number	Bit Mnemonic	Function																														
7	TB	Time Base Select Specifies the reference timer. 0 = timer 1 is the reference timer and timer 2 is the opposite timer 1 = timer 2 is the reference timer and timer 1 is the opposite timer A compare event (reloading the waveform generator; starting an A/D conversion; clearing, setting, or toggling an output pin; and/or resetting either timer) occurs when the reference timer matches the time programmed in the event-time register. When a capture event (falling edge, rising edge, or an edge change on the EPA _x pin) occurs, the reference timer value is saved in the EPA event-time register (EPA _x _TIME).																														
6	CE	Compare Enable Determines whether the EPA channel operates in capture or compare mode. 0 = capture mode 1 = compare mode																														
5:4	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. <table style="margin-left: 20px; border-collapse: collapse;"> <tr> <td style="text-align: center;">M1</td> <td style="text-align: center;">M0</td> <td style="text-align: left;">Capture Mode Event</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td>no capture</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td>capture on falling edge</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td>capture on rising edge</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td>capture on either edge</td> </tr> <tr> <td style="text-align: center;">M1</td> <td style="text-align: center;">M0</td> <td style="text-align: left;">Compare Mode Action</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td>no output</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td>clear output pin</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td>set output pin</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td>toggle output pin</td> </tr> </table>	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
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 11-10. EPA Control (EPA_x_CON) Registers

EPA_x_CON (Continued)

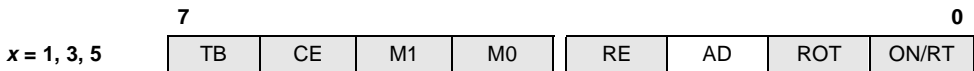
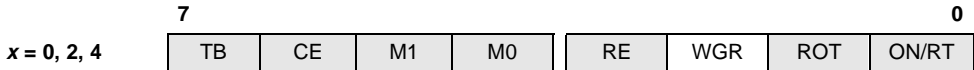
x = 0–1 (8XC196MH)

x = 0–3 (8XC196MC)

x = 0–5 (8XC196MD)

Address: See Table 11-3 on page 11-3
 Reset State: 00H

The EPA control (EPA_x_CON) registers control the functions of their assigned capture/compare channels.



Bit Number	Bit Mnemonic	Function
3	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
2	WGR AD	Waveform Generator Reload; A/D Conversion The function of this bit depends on the EPA channel. For EPA capture/compare channels 0, 2, 4: The WGR bit allows you to use the EPA activities to cause the reload of new values in the waveform generator. 0 = no action 1 = enables waveform generator reload For EPA capture/compare channels 1, 3, 5: The AD bit allows you to use the EPA activities to start an A/D conversion that has been previously set up in the A/D control registers. 0 = causes no A/D action 1 = starts an A/D conversion on an output compare

Figure 11-10. EPA Control (EPA_x_CON) Registers (Continued)

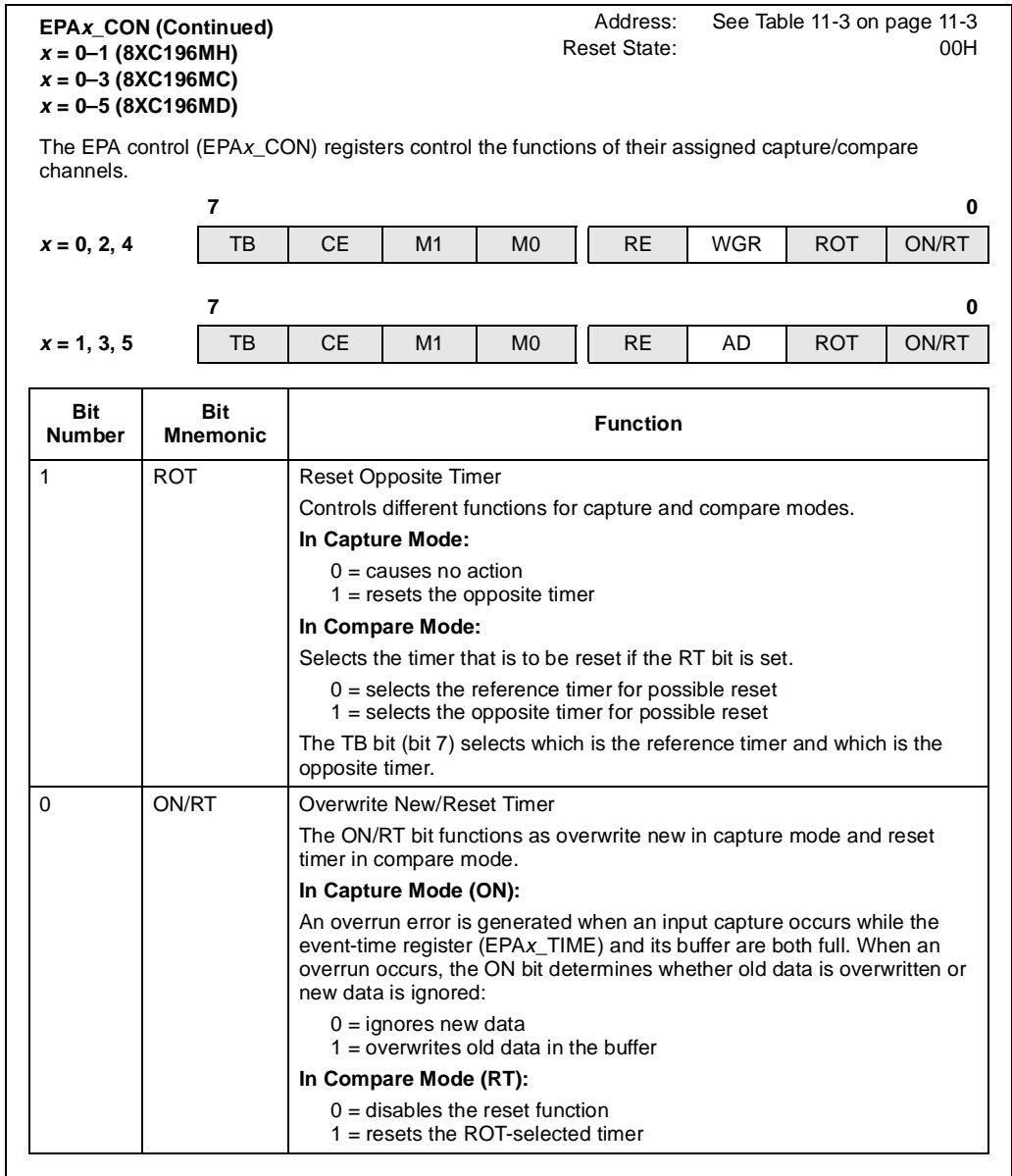


Figure 11-10. EPA Control (EPA_x CON) Registers (Continued)

11.5.4 Programming the Compare-only Channels

To program a compare event, you must first write to the $COMP_x_CON$ register (Figure 11-11) to configure the compare-only channel and then load the event time into $COMP_x_TIME$. $COMP_x_CON$ has the same bits and settings as EPA_x_CON . $COMP_x_TIME$ is functionally identical to EPA_x_TIME .

COMP_x_CON		Address: See Table 11-3 on page 11-3								
x = 0–3 (8XC196MC, MH)		Reset State: 00H								
x = 0–5 (8XC196MD)										
The EPA compare control ($COMP_x_CON$) registers determine the function of the EPA compare channels.										
	7	0								
x = 0, 2, 4	<table border="1" style="width: 100%; text-align: center;"> <tr> <td>TB</td> <td>CE</td> <td>M1</td> <td>M0</td> <td>RE</td> <td>WGR</td> <td>ROT</td> <td>RT</td> </tr> </table>	TB	CE	M1	M0	RE	WGR	ROT	RT	
TB	CE	M1	M0	RE	WGR	ROT	RT			
	7	0								
x = 1, 3, 5	<table border="1" style="width: 100%; text-align: center;"> <tr> <td>TB</td> <td>CE</td> <td>M1</td> <td>M0</td> <td>RE</td> <td>AD</td> <td>ROT</td> <td>RT</td> </tr> </table>	TB	CE	M1	M0	RE	AD	ROT	RT	
TB	CE	M1	M0	RE	AD	ROT	RT			
7	TB	Time Base Select Specifies the reference timer. 0 = timer 1 is the reference timer and timer 2 is the opposite timer 1 = timer 2 is the reference timer and timer 1 is the opposite timer A compare event (start of an A/D conversion; clearing, setting, or toggling an output pin; and/or resetting either timer) occurs when the reference timer matches the time programmed in the event-time register.								
6	CE	Compare Enable This bit enables the compare function. 0 = compare function disabled 1 = compare function enabled								
5:4	M1:0	EPA Mode Select Specifies the type of compare event. M1 M0 0 0 no output 0 1 clear output pin 1 0 set output pin 1 1 toggle output pin								
3	RE	Re-enable Allows a compare event to continue to execute each time the event-time register ($COMP_x_TIME$) matches the reference timer rather than only upon the first time match. 0 = compare function will drive the output only once 1 = compare function always enabled								

Figure 11-11. EPA Compare Control ($COMP_x_CON$) Registers

COMP_x_CON (Continued)		Address: See Table 11-3 on page 11-3					
x = 0–3 (8XC196MC, MH)		Reset State: 00H					
x = 0–5 (8XC196MD)							
The EPA compare control (COMP _x _CON) registers determine the function of the EPA compare channels.							
x = 0, 2, 4	7	0					
	TB	CE	M1	M0	RE	WGR	ROT
x = 1, 3, 5	7	0					
	TB	CE	M1	M0	RE	AD	ROT
2	WGR AD	<p>A/D Conversion, Waveform Generator Reload</p> <p>The function of this bit depends on the EPA channel.</p> <p>For EPA capture/compare channels 0, 2, 4:</p> <p>The WGR bit allows you to use the EPA activities to cause the reload of new values in the waveform generator.</p> <p>0 = no action 1 = enables waveform generator reload</p> <p>For EPA capture/compare channels 1, 3, 5:</p> <p>The AD bit allows you to use the EPA activities to start an A/D conversion that has been previously set up in the A/D control registers.</p> <p>0 = causes no A/D action 1 = starts an A/D conversion on an output compare</p>					
1	ROT	<p>Reset Opposite Timer</p> <p>Selects the timer that is to be reset if the RT bit is set.</p> <p>0 = selects the reference timer for possible reset 1 = selects the opposite timer for possible reset</p> <p>The state of the TB bit determines which timer is the reference timer and which timer is the opposite timer.</p>					
0	RT	<p>Reset Timer</p> <p>This bit controls whether the timer selected by the ROT bit will be reset.</p> <p>1 = resets the timer selected by the ROT bit 0 = disables the reset function</p>					

Figure 11-11. EPA Compare Control (COMP_x_CON) Registers (Continued)

11.6 ENABLING THE EPA INTERRUPTS

To enable the interrupts, set the corresponding bits in the INT_MASK register (Figure 5-7 on page 5-15). To enable the individual sources of the multiplexed PI (MC, MD), SPI (MH), and OVRTM (M_x) interrupts, set the corresponding bits in the PI_MASK register (Figure 5-9 on page 5-17). Chapter 5, “Standard and PTS Interrupts,” discusses the interrupts in greater detail.

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.

The pending bits are located in the INT_PEND and INT_PEND1 registers (Figure 5-7 on page 5-15 and Figure 5-11 on page 5-22). The pending bits for the multiplexed interrupts (those that share the PI interrupt) are located in the PI_PEND register (Figure 5-12 on page 5-23). Timer overflows/underflows also set interrupt pending bits. Even if an interrupt is masked, software can poll the interrupt pending registers to determine whether an event has occurred.



12

Analog-to-digital 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. It can also monitor an input and set the A/D interrupt pending bit when the input voltage crosses over or under the programmed threshold voltage.

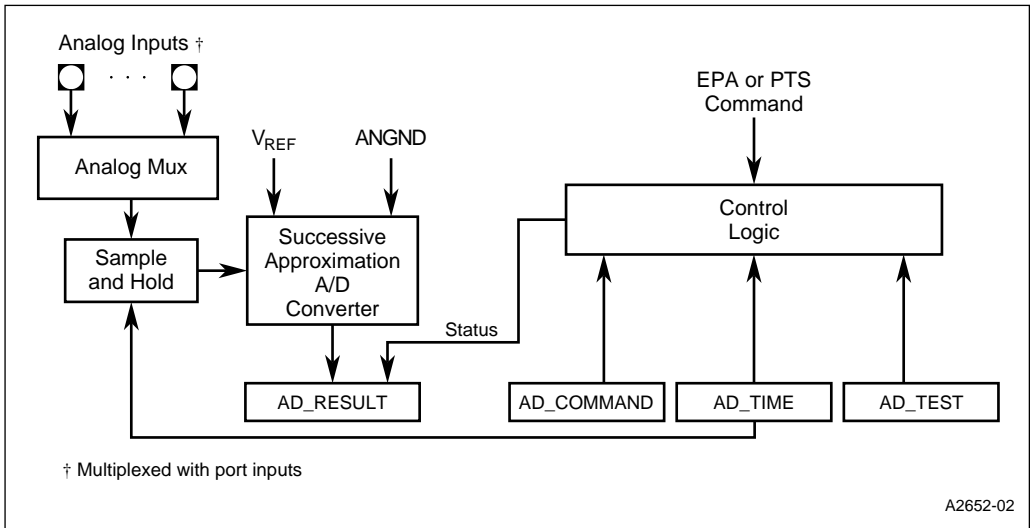


Figure 12-1. A/D Converter Block Diagram

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. Although the analog inputs are multiplexed with I/O port pins, no configuration is necessary.

Table 12-1. A/D Converter Pins

Port Pin	A/D Signal	A/D Signal Type	Description
P1.4:0 P1.5:0	ACH12:8 (MC) ACH13:8 (MD)	I	Analog inputs. See the "Voltage on Analog Input Pin" specification in the datasheet.
P0.7:0	ACH7:0 (MC, MD, MH)	I	Analog inputs. See the "Voltage on Analog Input Pin" specification in the datasheet.
—	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	1FACH	A/D Command This register selects the A/D channel, controls whether the A/D conversion starts immediately or is triggered by the EPA, and selects the operating mode.
AD_RESULT	1FAAH, 1FABH	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 channel 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_TEST	1FAEH	A/D Conversion Test This register specifies adjustments for zero-offset errors.
AD_TIME	1FAFH	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.

Table 12-2. A/D Control and Status Registers (Continued)

Mnemonic	Address	Description
P0_PIN	1FA8H (MC, MD) 1FDAH (MH)	Port 0 Pin State Read P0_PIN to determine the current values of the port 0 pins. Reading the port induces noise into the A/D converter, decreasing the accuracy of any conversion in progress. We strongly recommend that you not read the port while an A/D conversion is in progress. To reduce noise, the P0_PIN register is clocked only when the port is read.
P1_PIN (MC,MD)	1FA9H (MC, MD)	Port 1 Pin State Read P1_PIN to determine the current values of the port 1 pins. Reading the port induces noise into the A/D converter, decreasing the accuracy of any conversion in progress. We strongly recommend that you not read the port while an A/D conversion is in progress. To reduce noise, the P1_PIN register is clocked only when the port is read.

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 register, 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 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 “Programming the EPA and Timer/Counters” on page 11-15.) The A/D scan mode of the peripheral transaction server (PTS) allows you to perform multiple conversions and store their results. (See “A/D Scan Mode” on page 5-32.)

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, it 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 $\frac{1}{2}$ 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 (011111111B). If the analog input was less than the test voltage, bit 10 of the SAR is left at zero, and a new test voltage of $\frac{1}{4}$ full scale (001111111B) is tried. If the analog input was greater than the test voltage, bit 9 of SAR is set. Bit 8 is then cleared for the next test (010111111B). This binary search continues until 10 (or 8) tests have occurred, at which time the valid conversion result resides in the AD_RESULT register 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
- zero-offset adjustment — no adjustment, plus 2.5 mV, minus 2.5 mV, or minus 5.0 mV
- 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) analog 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 12-13 describes zero-offset and other errors inherent in A/D conversions.

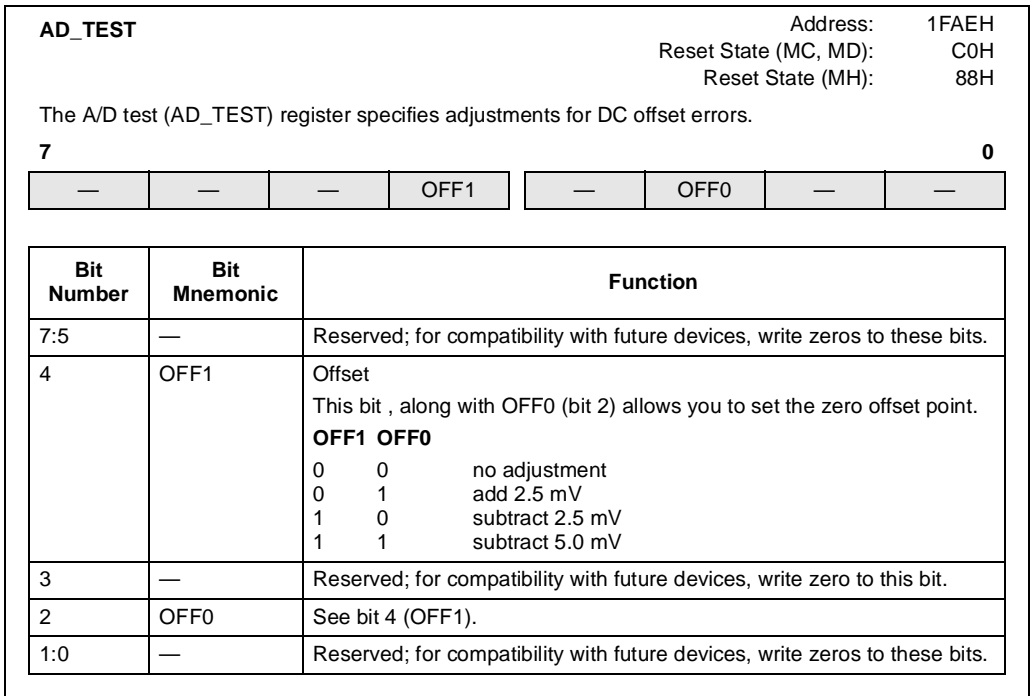


Figure 12-2. A/D Test (AD_TEST) Register

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 to set the desired reference (threshold) voltage.

AD_RESULT (Write)				Address: 1FAAH			
				Reset State (MC, MD): FFC0H			
				Reset State (MH): 7FC0H			
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 selects a reference voltage that 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. $\text{reference voltage} = \frac{\text{desired threshold voltage} \times 256}{V_{\text{REF}} - \text{ANGND}}$					
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 on the datasheet to determine appropriate values.

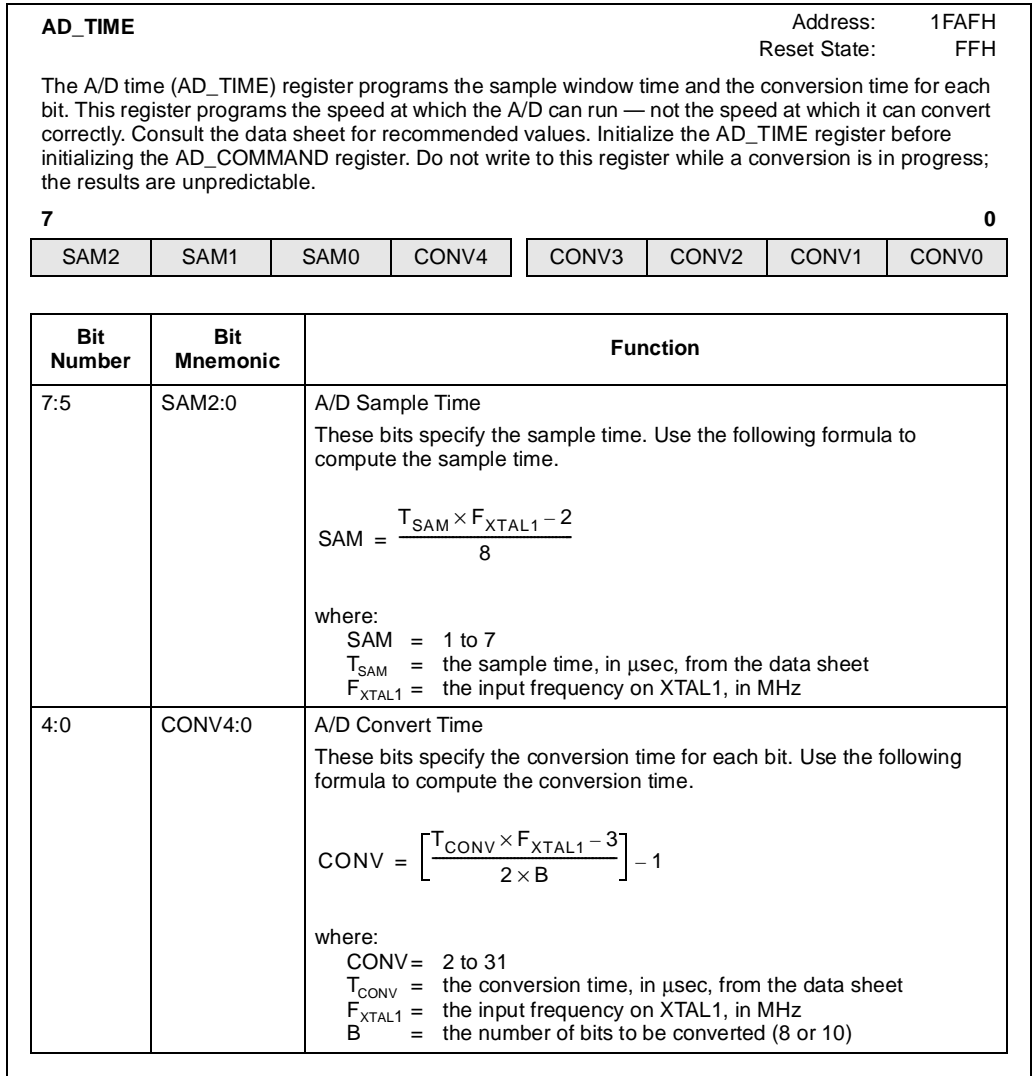


Figure 12-4. A/D Time (AD_TIME) Register

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: 1FACH			
				Reset State: 80H			
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, and selects the conversion mode.							
7				0			
—	M1	M0	GO	ACH3	ACH2	ACH1	ACH0
Bit Number	Bit Mnemonic	Function					
7	—	Reserved; for compatibility with future devices, write zeros to these bits.					
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 high 1 1 threshold detect low					
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.					
† 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 12-5. A/D Command (AD_COMMAND) Register

12.4.5 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 Table 12-2 on page 12-2) and execute the EI instruction to globally enable servicing of interrupts. The A/D interrupt can cause the PTS to begin a new conversion. See Chapter 5, “Standard and PTS Interrupts,” for details about interrupts and a description of using the PTS in A/D scan mode.

12.5 DETERMINING A/D STATUS AND CONVERSION RESULTS

You can read the AD_RESULT register (Figure 12-6) 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. If you read AD_RESULT before the conversion is complete, the result is not guaranteed to be accurate.

The conversion result is the ratio of the input voltage to the reference voltage:

$$\text{RESULT (8-bit)} = 255 \times \frac{V_{\text{IN}} - \text{ANGND}}{V_{\text{REF}} - \text{ANGND}} \qquad \text{RESULT (10-bit)} = 1023 \times \frac{V_{\text{IN}} - \text{ANGND}}{V_{\text{REF}} - \text{ANGND}}$$

You can also read the interrupt pending register (see Table 12-2 on page 12-2) to determine the status of the A/D interrupt.

AD_RESULT (Read)				Address: 1FAAH			
				Reset State (MC, MD): FFC0H			
				Reset State (MH): 7FC0H			
<p>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 ten-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.</p>							
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.					

Figure 12-6. A/D Result (AD_RESULT) 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 12-7.

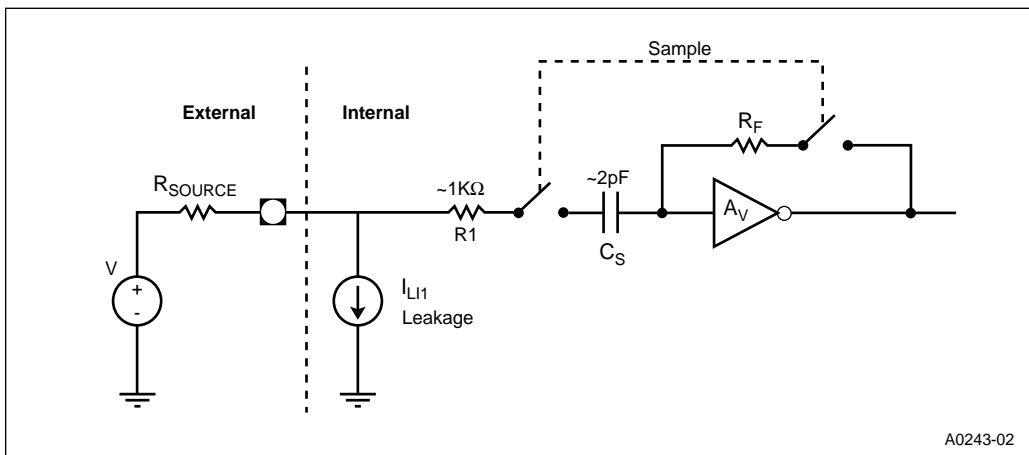


Figure 12-7. Idealized A/D Sampling Circuitry

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 kilo-ohm 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 12-8 provides limited protection against over-voltage 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 270 Ω 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.

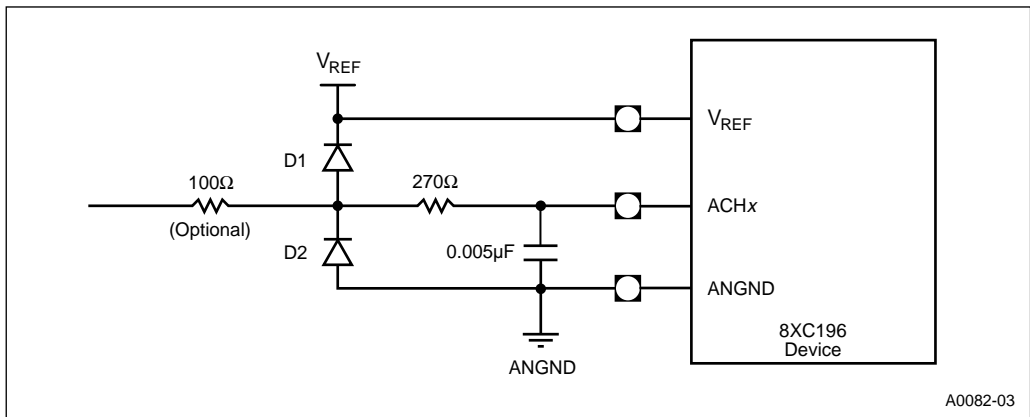


Figure 12-8. Suggested A/D Input Circuit

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 μ F 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. It is also the supply to port 0 (and port 1 for the MC and MD) if the A/D converter is not being used. 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} .

12.6.1.4 Using Mixed Analog and Digital Inputs

Port 0 (and port 1 for the MC and MD) may be used for both analog and digital input signals at the same time. However, reading the port may inject some noise into the analog circuitry. For this reason, make certain that an analog conversion is **not** in progress when the port is read. Refer to Chapter 6, "I/O Ports," for information about using the port as digital inputs.

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 12-9 shows the transfer function for an ideal 3-bit A/D converter.

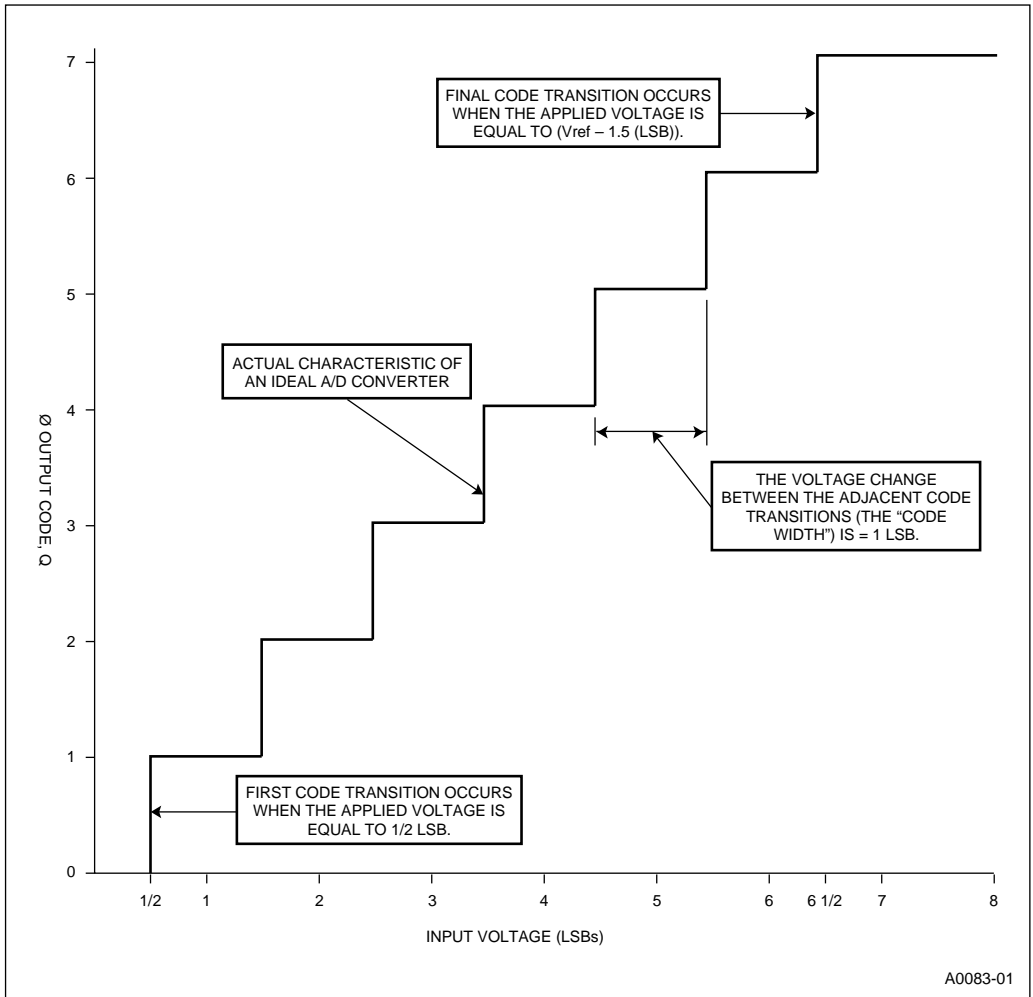


Figure 12-9. Ideal A/D Conversion Characteristic

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 ($V_{REF} - 1.5\text{LSB}$); 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.

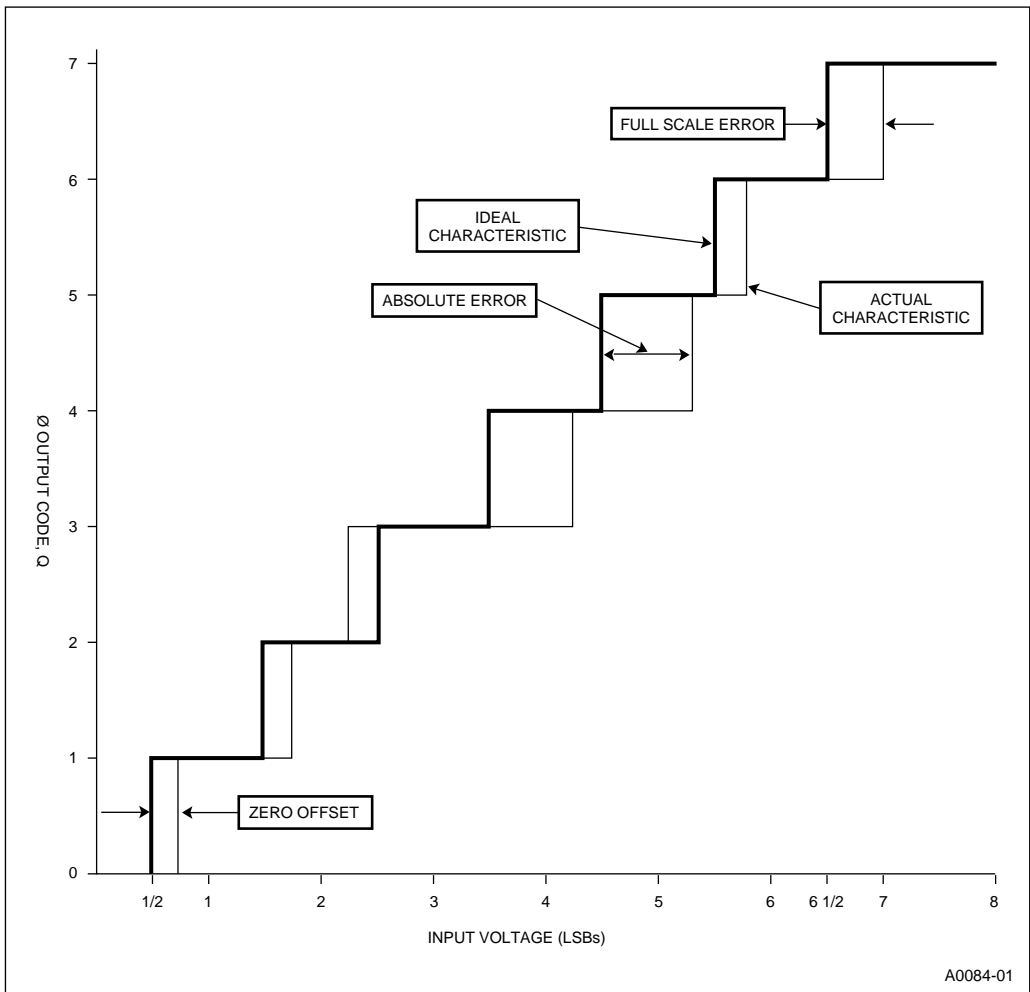


Figure 12-10. Actual and Ideal A/D Conversion Characteristics

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 12-10. 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 12-11. 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.

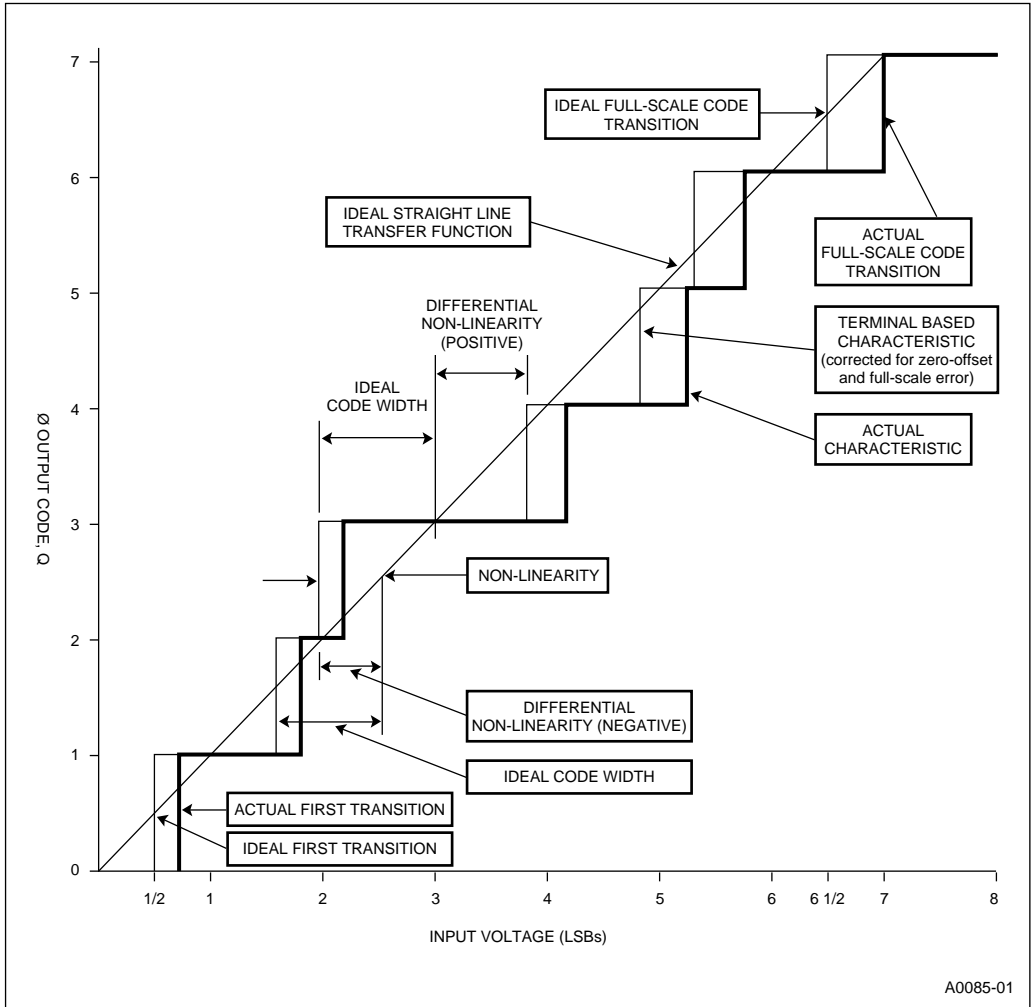


Figure 12-11. Terminal-based A/D Conversion Characteristic



13

Minimum Hardware Considerations

CHAPTER 13

MINIMUM HARDWARE CONSIDERATIONS

The 8XC196MC, MD, and MH have 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 and port 0 operation (also port 1 on the 8XC196MC and MD). ANGND and V_{SS} should be nominally at the same potential.
RESET#	I/O	Reset A level-sensitive reset input to and 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 chip to reset and return to normal operating mode. The 8XC196MH provides the option of preventing an internal reset from generating a reset on the external pin (see "Resetting the Device" on page 13-8). After a device reset, the first instruction fetch is from 2080H.
V_{CC}	PWR	Digital Supply Voltage Connect each V_{CC} pin to the digital supply voltage.
V_{PP}	PWR	Programming Voltage During programming, the V_{PP} pin is typically at +12.5 V (V_{PP} voltage). Exceeding the maximum V_{PP} voltage specification can damage the device. V_{PP} also causes the device to exit powerdown mode when it is driven low for at least 50 ns. Use this method to exit powerdown only when using an external clock source because it enables the internal phase clocks, but not the internal oscillator. On devices with no internal nonvolatile memory, connect V_{PP} to V_{CC} .
V_{REF}	PWR	Reference Voltage for the A/D Converter This pin also supplies operating voltage to both the analog portion of the A/D converter and the logic used to read port 0 (also port 1 in the 8XC196MC and 8XC196MD).
V_{SS}	GND	Digital Circuit Ground 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 (MC/MD only). When using an external clock or crystal, instead of the on-chip oscillator, connect the clock input to XTAL1. The external clock signal must meet the V_{IH} specification for XTAL1 (see datasheet).
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 6, "I/O Ports," contains information about initializing and configuring the ports. Table 13-2 lists the sections, with page numbers, that contain the information for each port.

Table 13-2. I/O Port Configuration Guide

Port	Where to Find Configuration Information
Port 0	"Standard Input-only Port Considerations" on page 6-4
Port 1	"Standard Input-only Port Considerations" on page 6-4 (MC, MD) "Bidirectional Port Pin Configurations" on page 6-9 and "Bidirectional Port Considerations" on page 6-12 (MH)
Port 2	"Bidirectional Port Pin Configurations" on page 6-9 and "Bidirectional Port Considerations" on page 6-12
Ports 3 and 4	"Bidirectional Ports 3 and 4 (Address/Data Bus) Operation" on page 6-15
Port 5	"Bidirectional Port Pin Configurations" on page 6-9 and "Bidirectional Port Considerations" on page 6-12
Port 6	"Configuring Output-only Port Pins" on page 6-17
Port 7 (MD)	"Bidirectional Port Pin Configurations" on page 6-9 and "Bidirectional Port Considerations" on page 6-12

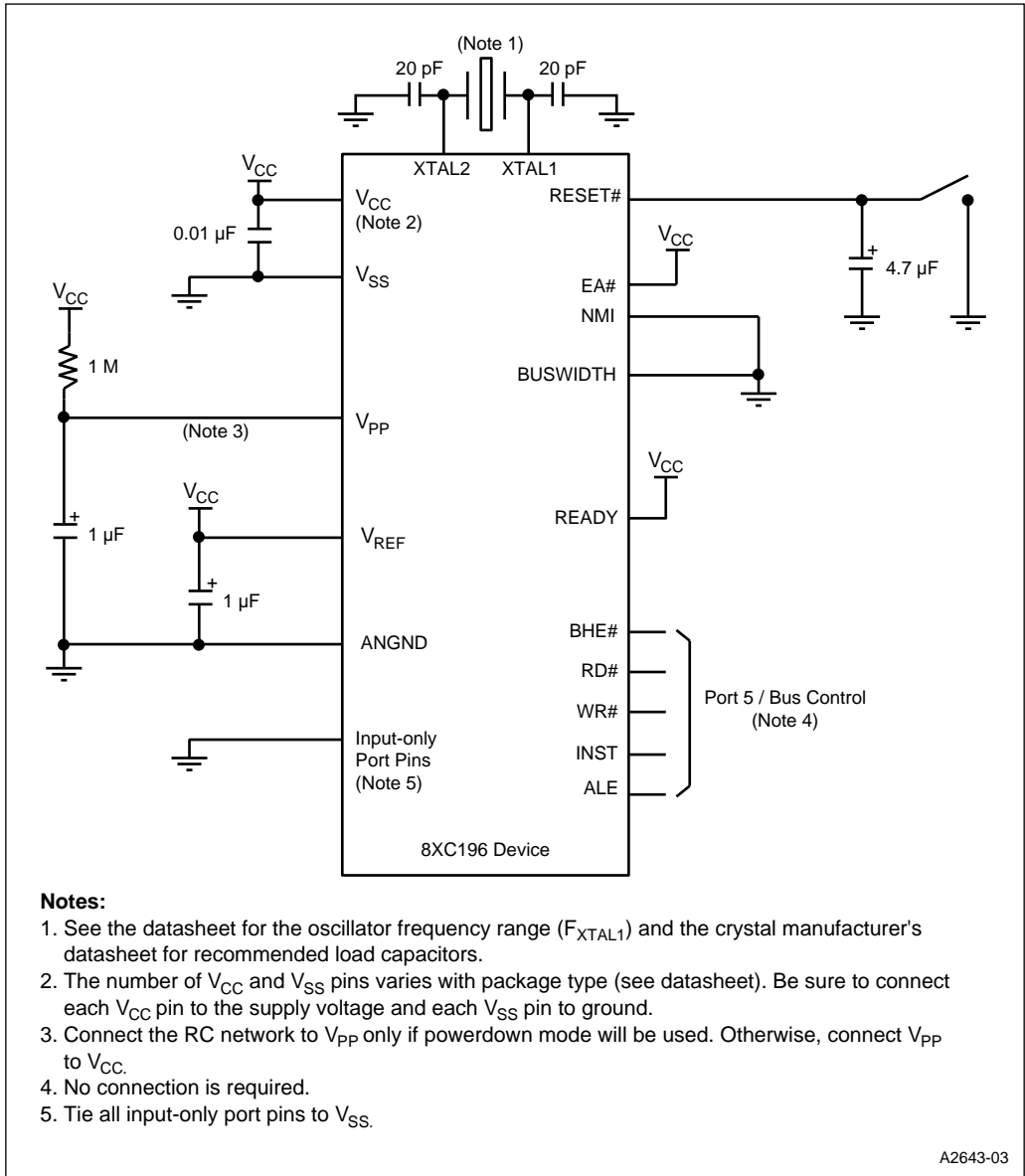


Figure 13-1. Minimum Hardware Connections

13.2 APPLYING AND REMOVING POWER

When power is first applied to the device, 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 $0.01\ \mu\text{F}$ bypass capacitors between V_{CC} and each V_{SS} pin and a $1.0\ \mu\text{F}$ 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 each other.

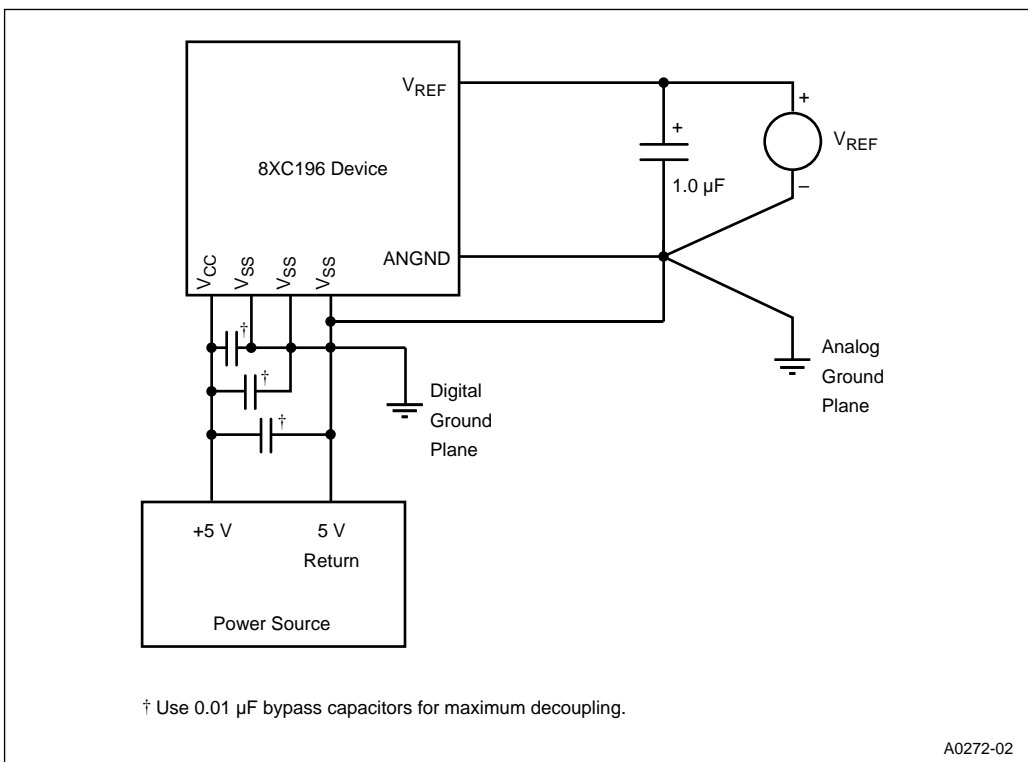


Figure 13-2. Power and Return Connections

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 to provide power to port 0. On the 8XC196MC and MD, they also provide power to port 1. Refer to “Analog Ground and Reference Voltages” on page 12-12 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* and AP-711, *EMI Design Techniques for Microcontrollers in Automotive Applications*.

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, R_f , consists of paralleled n -channel and p -channel FETs controlled by the internal powerdown signal. In powerdown mode, R_f 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.

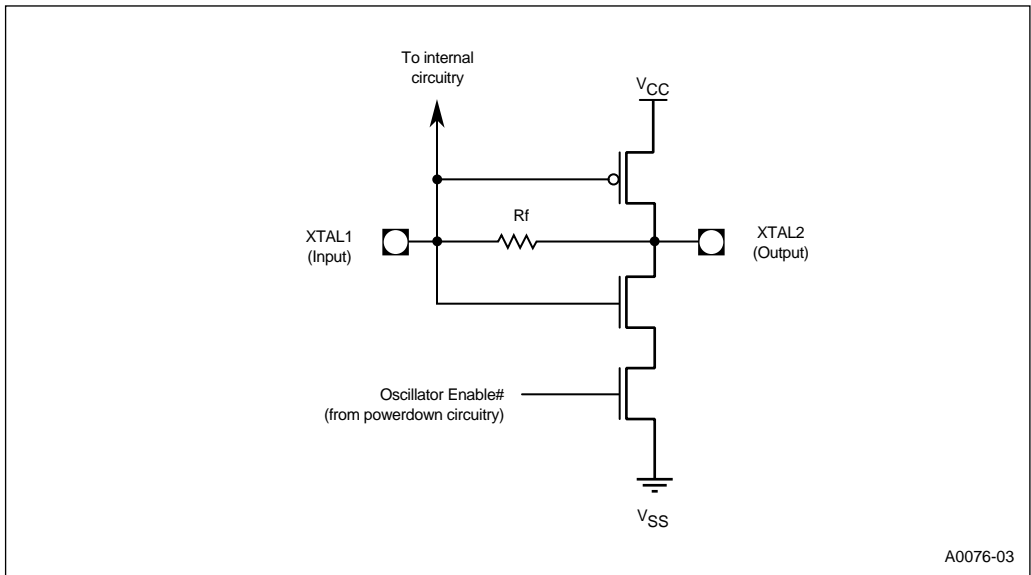


Figure 13-3. On-chip Oscillator Circuit

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.

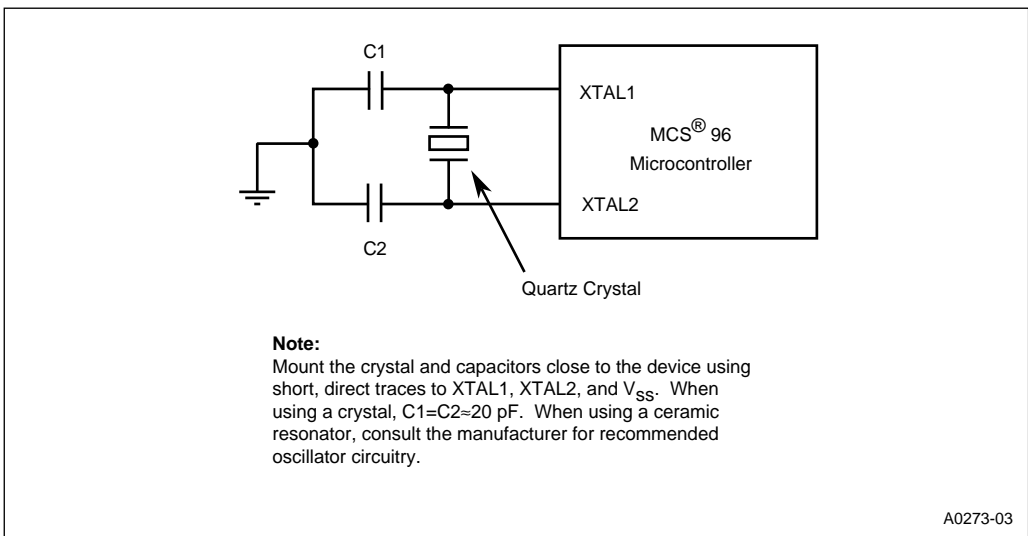


Figure 13-4. External Crystal Connections

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}) (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.

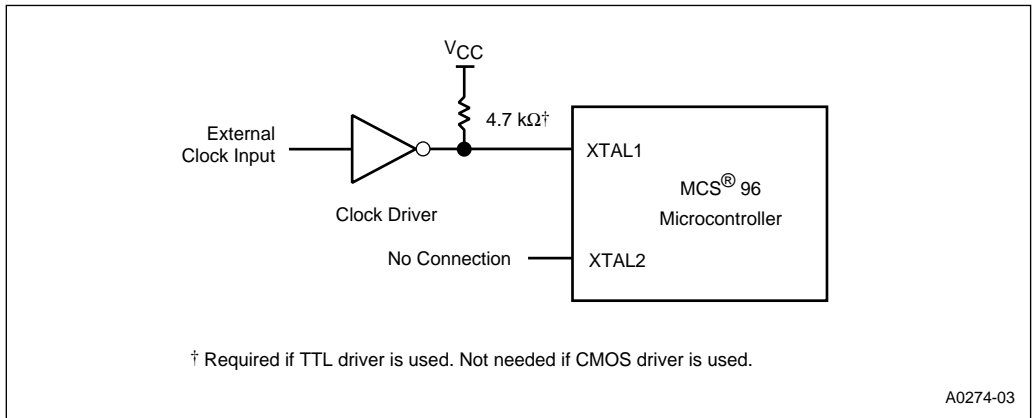


Figure 13-5. External Clock Connections

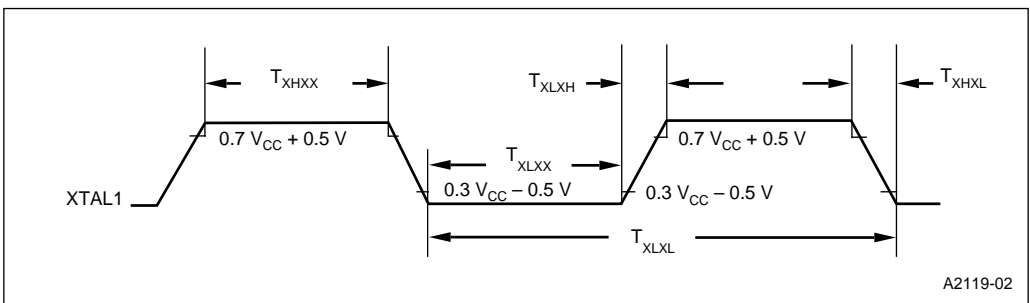


Figure 13-6. External Clock Drive Waveforms

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 DEVICE

Reset forces the device 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. (Tables in Appendix B list the reset states of the pins (see Table B-8 on page B-23 for the 8XC196MC and 8XC196MD or Table B-9 on page B-25 for the 8XC196MH). See Table C-2 on page C-2 for the reset values of the SFRs.) The device 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. Depending upon when RESET# is brought high, the CLKOUT signal (MC and MD only) may become out of phase with the PH1 internal clock. When this occurs, the clock generator immediately resynchronizes CLKOUT as shown in Case 2.

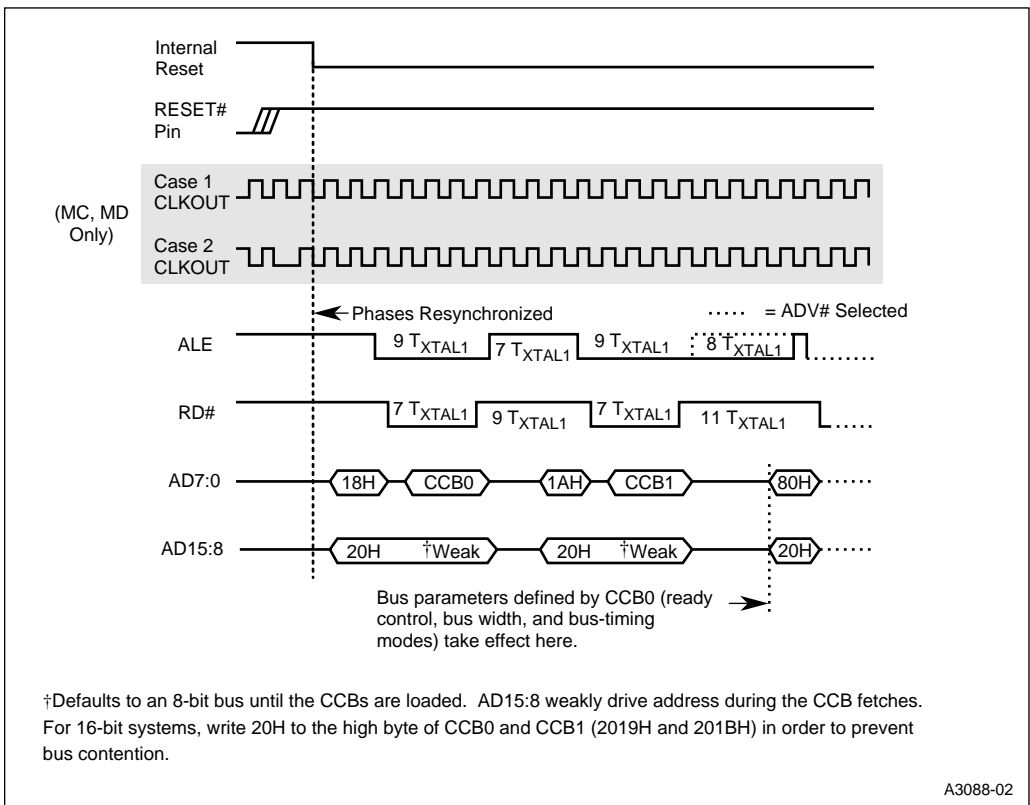


Figure 13-7. Reset Timing Sequence

The 8XC196MH provides the option of an internal-only reset or an internal reset that is also reflected externally (by the RESET# pin). The GEN_CON register controls whether an internal reset asserts the external RESET# signal and indicates the source of the most recent reset. Figure 13-8 describes the general configuration register, GEN_CON.

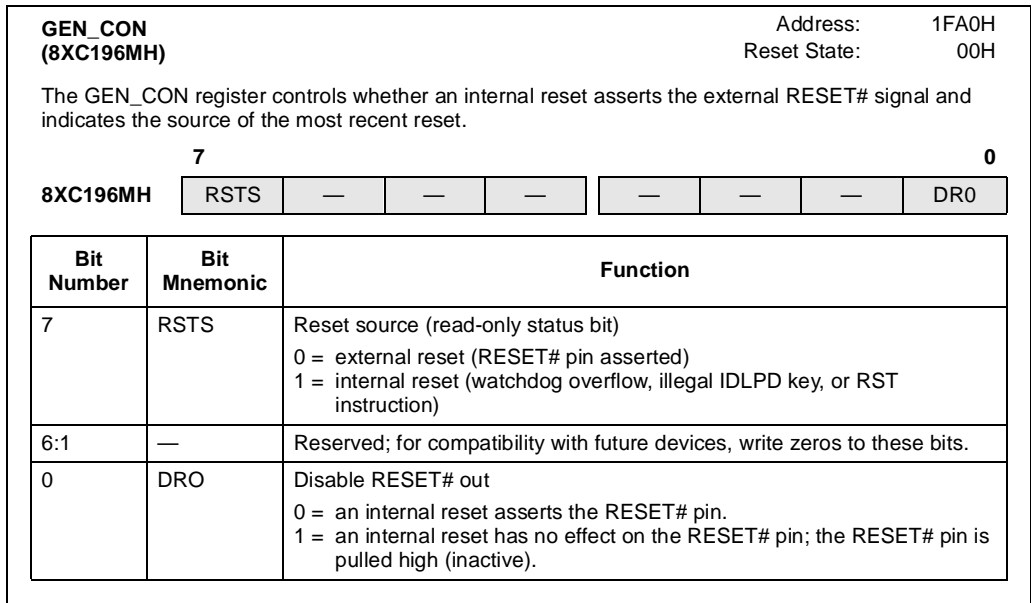


Figure 13-8. General Configuration Register (GEN_CON)

The following events will reset the device (see Figure 13-9):

- an external device pulls the RESET# pin low
- the CPU issues the reset (RST) instruction
- the CPU issues an idle/powerdown (IDLPD) instruction with an illegal key operand
- the watchdog timer (WDT) overflows

The following paragraphs describe each of these reset methods in more detail.

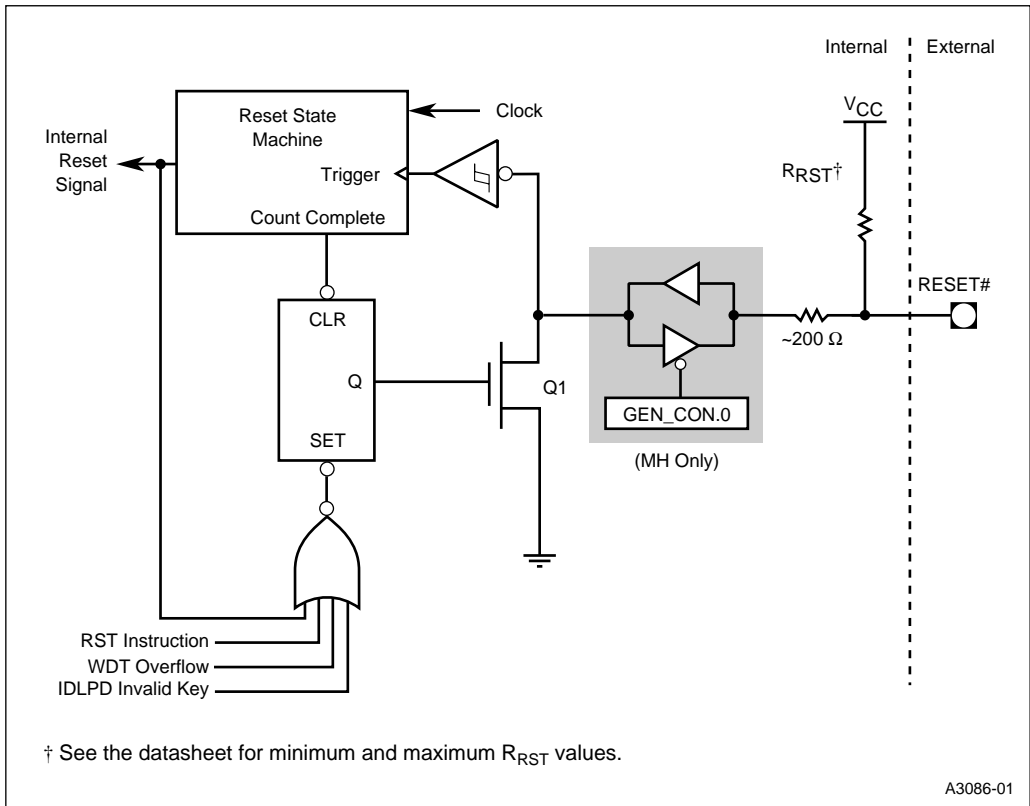


Figure 13-9. Internal Reset Circuitry

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) for 16 state times. This enables the RESET# signal to function as the system reset.

The simplest way to reset the device is to insert a capacitor between the RESET# pin and V_{SS} , as shown in Figure 13-10. The device has an internal pull-up resistor (R_{RST}) shown in Figure 13-9. 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.

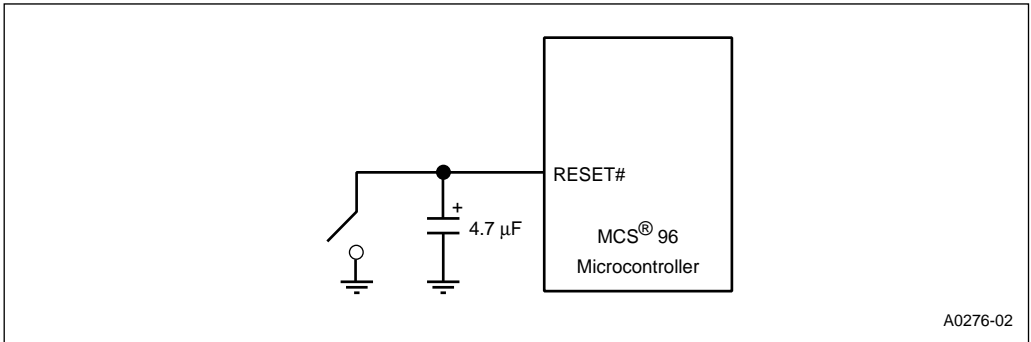


Figure 13-10. Minimum Reset Circuit

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 13-11 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.

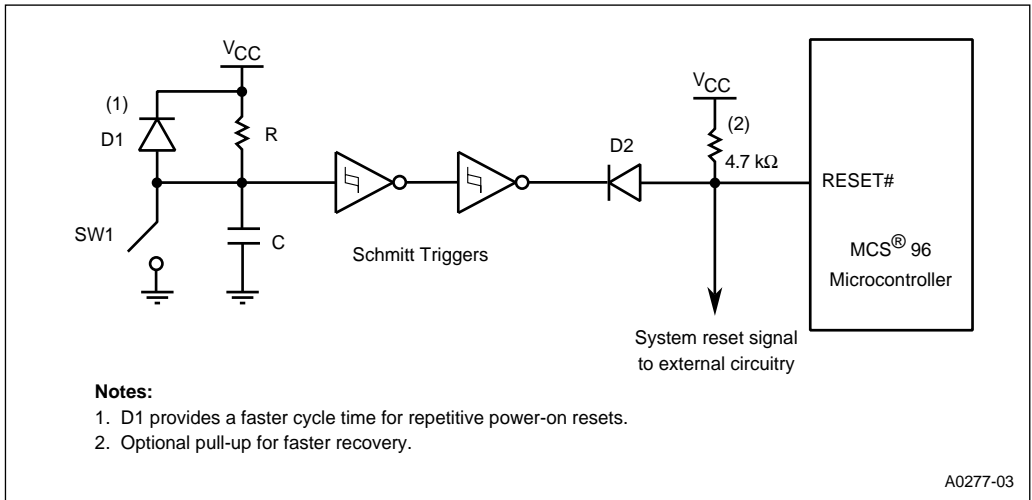


Figure 13-11. Example of a System Reset Circuit

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 2080H, and resets the special function registers (SFRs). See Table C-2 on page C-2 for the reset values of the SFRs.

Putting pull-ups on the address/data bus causes unimplemented areas of memory to be read as FFH. If unused internal OTPROM memory is set to FFH, then execution from any unused memory locations will reset the device.

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. If any other value is used, the device executes a reset sequence. (See Appendix A for a description of the IDLPD command.)

13.6.4 Generating Wait States

The 8XC196M_x devices can interface with a variety of external memory devices. With slower external devices this requires inserting wait states to lengthen the bus cycle. An external device can use the READY input to request wait states in addition to the wait states that are generated internally by the 8XC196M_x device. The READY signal must be held valid until the T_{CLYX} (for the 8XC196MC, MD) or T_{LLYX} (for the 8XC196MH) timing specification is met. Do not exceed the maximum T_{CLYX} (for the 8XC196MC, MD) or T_{LLYX} (for the 8XC196MH) specification or additional (unwanted) wait states might be added (see Chapter 15, “Wait States (Ready Control),” for a detailed explanation).

13.6.5 Enabling the Watchdog Timer

The watchdog timer (WDT) is a 16-bit counter that resets the device when the counter overflows. The WDE bit (bit 3) of CCR1 controls whether the watchdog is enabled immediately or is disabled until the first time it is cleared. Clearing WDE activates the watchdog. Setting WDE makes the watchdog timer inactive, but you can activate it by clearing the watchdog register. Once the watchdog is activated, only a reset can disable it.

The 8XC196MC and 8XC196MD allow only one reset interval, 64K state times. This requires your software to interrupt itself every 65,535 state times to reset the watchdog. The 8XC196MH allows you to choose a longer interval.

You must write two consecutive bytes to the watchdog register (location 0AH) to clear it. For the 8XC196MC and MD, the first byte must be 1EH and the second must be E1H. For the 8XC196MH, the first byte must also be 1EH; however, the second byte can be one of four values. The second byte determines the reset interval (Table 13-3). 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. We recommend 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 13-3. Selecting the Watchdog Reset Interval (8XC196MH only)

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)

NOTE (8XC196MH Only)

If the WDE bit of CCR1 is cleared, the watchdog is activated immediately after a system power-up or a device reset. You must clear the watchdog register within 64K state times after a system power-up or a device reset. At that time, you can choose a longer interval for subsequent watchdog resets.

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 device, which causes it to exit idle mode.



14

Special Operating Modes

CHAPTER 14

SPECIAL OPERATING MODES

The 8XC196MC, MD, and MH provide two power saving modes: idle and powerdown. They also provide an on-circuit emulation (ONCE) mode that electrically isolates the device from the other system components. This chapter describes each mode and explains how to enter and exit each. (Refer to Appendix A for descriptions of the instructions discussed in this chapter, to Appendix B for descriptions of signal status during each mode, and to Appendix C for details about the registers.)

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

Port Pin	Signal Name	Type	Description
P2.7	CLKOUT (MC/MD only)	O	<p>Clock Output</p> <p>Output of the internal clock generator. The CLKOUT frequency is $\frac{1}{2}$ the oscillator input frequency (F_{XTAL1}). CLKOUT has a 50% duty cycle. CLKOUT is not implemented on the 8XC196MH.</p>
—	EXTINT	I	<p>External Interrupt</p> <p>This programmable interrupt is controlled by the WG_PROTECT register. This register controls whether the interrupt is edge triggered or sampled and whether a rising edge/high level or falling edge/low level activates the interrupt.</p> <p>In powerdown mode, asserting the EXTINT signal for at least 50 ns causes the device to resume normal operation. The interrupt need not be enabled. 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.</p> <p>In idle mode, asserting any enabled interrupt causes the device to resume normal operation.</p>

Table 14-1. Operating Mode Control Signals (Continued)

Port Pin	Signal Name	Type	Description
P5.4	ONCE#	I	<p>On-circuit Emulation</p> <p>Holding ONCE# low during the rising edge of RESET# places the device into on-circuit emulation (ONCE) mode. This mode puts all pins, except XTAL1 and XTAL2, into a high-impedance state, thereby isolating the device 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 device 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 (see datasheet).</p>
P2.6	Test-mode entry	I/O	<p>Test-mode entry</p> <p>If this pin is held low during reset, the device will enter a reserved test mode, so exercise caution if you use this pin for input. If you choose to configure this pin as an input, always hold it high during reset and ensure that your system meets the V_{IH} specification (see datasheet) to prevent inadvertent entry into a test mode.</p>
—	RESET#	I/O	<p>Reset</p> <p>A level-sensitive reset input to and 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 chip to reset and return to normal operating mode. The 8XC196MH provides the option of preventing an internal reset from generating a reset on the external pin (see "Resetting the Device" on page 13-8). After a device reset, the first instruction fetch is from 2080H.</p>
—	V_{PP}	PWR	<p>Programming Voltage</p> <p>During programming, the V_{PP} pin is typically at +12.5 V (V_{PP} voltage). Exceeding the maximum V_{PP} voltage specification can damage the device.</p> <p>V_{PP} also causes the device to exit powerdown mode when it is driven low for at least 50 ns. Use this method to exit powerdown only when using an external clock source because it enables the internal phase clocks, but not the internal oscillator.</p> <p>On devices with no internal nonvolatile memory, connect V_{PP} to V_{CC}.</p>

Table 14-2. Operating Mode Control and Status Registers

Mnemonic	Address	Description
CCR0	2018H	<p>Chip Configuration 0 Register</p> <p>Bit 0 of this register enables and disables powerdown mode.</p>
INT_MASK1	0013H	<p>Interrupt Mask 1</p> <p>Bit 6 of this 8-bit register enables and disables (masks) the external interrupt (EXTINT).</p>

Table 14-2. Operating Mode Control and Status Registers (Continued)

Mnemonic	Address	Description
P1_DIR (MH) P2_DIR P5_DIR P7_DIR (MD)	1F9BH 1FD2H 1FF3H 1FD3H	Port x Direction Each bit of Px_DIR controls the direction of the corresponding pin. Clearing a bit configures a pin as a complementary output; setting a bit configures a pin as an input or open-drain output. (Open-drain outputs require external pull-ups.)
P1_MODE(MH) P2_MODE P5_MODE P7_MODE(MD)	1F99H 1FD0H 1FF1H 1FD1H	Port x Mode Each bit of Px_MODE controls whether the corresponding pin functions as a standard I/O port pin or as a special-function signal. Setting a bit configures a pin as a special-function signal; clearing a bit configures a pin as a standard I/O port pin.
P1_REG (MH) P2_REG P5_REG P7_REG (MD)	1F9DH 1FD4H 1FF5H 1FD5H	Port x Data Output For an input, set the corresponding Px_REG bit. For an output, write the data to be driven out by each pin to the corresponding bit of Px_REG. When a pin is configured as standard I/O (Px_MODE.y = 0), the result of a CPU write to Px_REG is immediately visible on the pin. When a pin is configured as a special-function signal (Px_MODE.y = 1), the associated on-chip peripheral or off-chip component controls the pin. The CPU can still write to Px_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 standard I/O (clear Px_MODE.y), initialize or overwrite the pin value, then configure the pin as a special-function signal (set Px_MODE.y). In this way, initialization, fault recovery, exception handling, etc., can be done without changing the operation of the associated peripheral.

14.2 REDUCING POWER CONSUMPTION

Both power-saving modes conserve power by disabling portions of the internal clock circuitry (Figure 14-1). The following paragraphs describe both modes in detail.

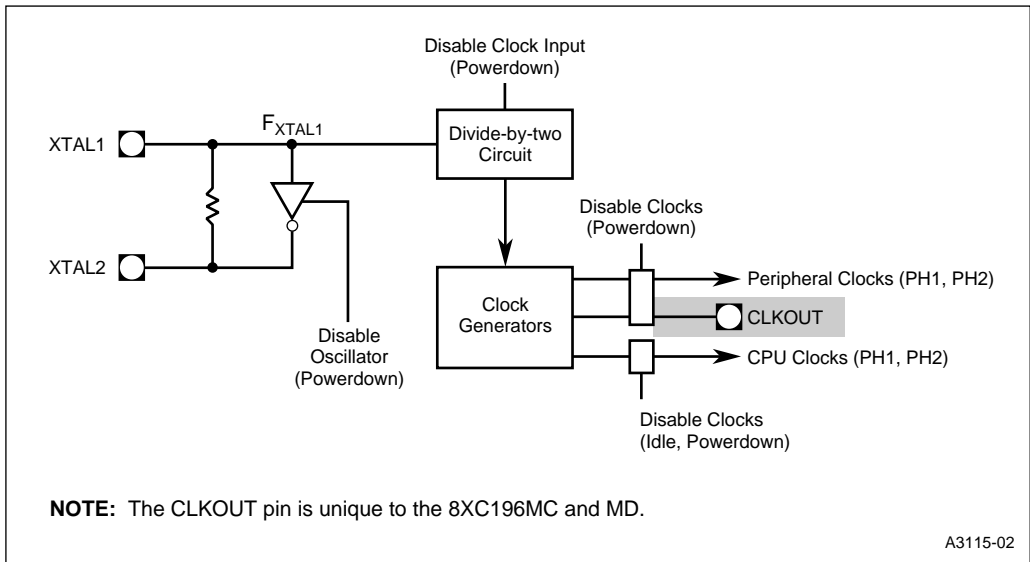


Figure 14-1. Clock Control During Power-saving Modes

14.3 IDLE MODE

In idle mode, the device's power consumption decreases to approximately 40% of normal consumption. Internal logic holds the CPU clocks at logic zero, causing the CPU to stop executing instructions. Neither the peripheral clocks nor CLKOUT are affected, so the special-function registers (SFRs) and register RAM retain their data and the peripherals and interrupt system remain active. Tables in Appendix B list the values of the pins during idle mode (see Table B-8 on page B-23 for the 8XC196MC and 8XC196MD or Table B-9 on page B-25 for the 8XC196MH).

The device 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

If enabled, the watchdog timer continues to run in idle mode. The device must be awakened before the counter overflows; otherwise, the timer will reset the device. The watchdog timer interval is always 64K state times in the 8XC196MC and MD, but a longer interval can be selected in the 8XC196MH (see “Enabling the Watchdog Timer” on page 13-12).

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 device into a very low power state by disabling the internal oscillator and 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. Tables in Appendix B list the values of the pins during powerdown mode (see Table B-8 on page B-23 for the 8XC196MC and 8XC196MD or Table B-9 on page B-25 for the 8XC196MH). 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 powerdown mode. Because CCR0 cannot be accessed by code, the PD bit value is defined in chip configuration byte 0 (CCB0.0). Setting the PD bit enables powerdown mode and clearing it disables powerdown. CCR0 is loaded from CCB0 when the device returns from reset. Refer to “Operating Environment” on page 16-17 for descriptions of the methods for programming the CCBs.

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.
- Complete all analog conversions. If powerdown occurs during the conversion, the result will be incorrect.
- If the watchdog timer (WDT) is enabled, clear the WATCHDOG register just before issuing the powerdown instruction. This ensures that the device can exit powerdown cleanly. Otherwise, the WDT could reset the device before the oscillator stabilizes. (The WDT cannot reset the device during powerdown because the clock is stopped.)
- 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 device will exit powerdown mode when any of the following events occurs:

- an external device drives the V_{PP} pin low for at least 50 ns
- a hardware reset is generated
- a transition occurs on the external interrupt pin

14.4.3.1 Driving the V_{PP} Pin Low

If the design uses an external clock input signal rather than the on-chip oscillator, the fastest way to exit powerdown mode is to drive the V_{PP} pin low for at least 50 ns. Use this method **only** when using an external clock input because the internal CPU and peripheral clocks will be enabled, but the internal oscillator will not.

14.4.3.2 Generating a Hardware Reset

The device will exit powerdown if RESET# is asserted. If the design uses an external clock input signal rather than the on-chip oscillator, RESET# must remain low for at least 16 state times. If the design uses the on-chip oscillator, then RESET# must be held low until the oscillator has stabilized.

14.4.3.3 Asserting the External Interrupt Signal

The final 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. The interrupt need not be enabled to bring the device out of powerdown, but the pin must be configured as a special-function input (see “Bidirectional Port Pin Configurations” on page 6-9). Figure 14-2 shows the power-up and power-down sequence when using an external interrupt to exit powerdown.

When an external interrupt brings the device 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.

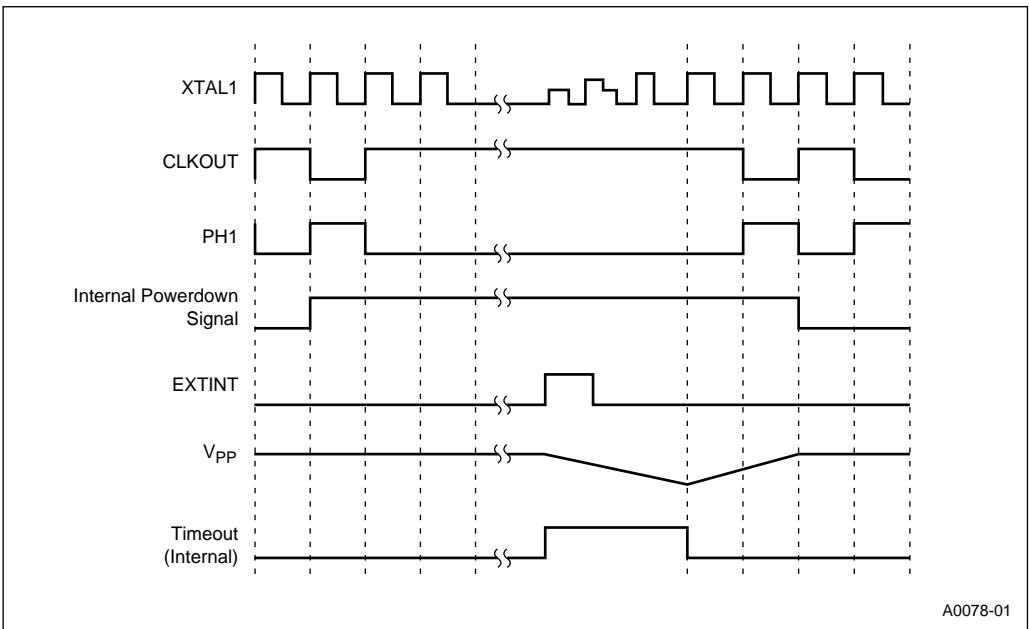


Figure 14-2. Power-up and Power-down Sequence When Using an External Interrupt

When using an external interrupt signal to exit powerdown mode, we recommend that you connect the external RC circuit shown in Figure 14-3 to the V_{pp} pin. The discharging of the capacitor causes a delay that allows the oscillator to stabilize before the internal CPU and peripheral clocks are enabled.

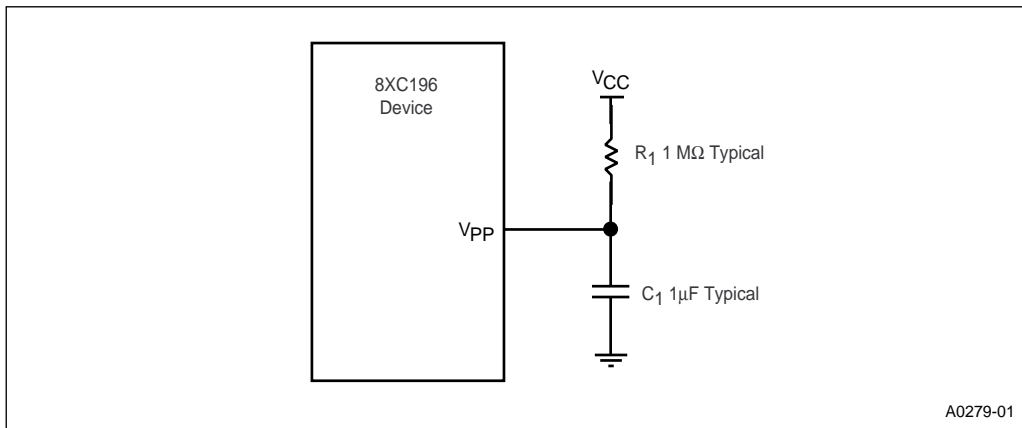


Figure 14-3. External RC Circuit

During normal operation (before entering powerdown mode), an internal pull-up holds the V_{PP} pin at V_{CC} . When an external interrupt signal is asserted, the internal oscillator circuitry is enabled and turns on a weak internal pull-down. This weak pull-down causes the external capacitor (C_1) to begin discharging at a typical rate of 200 μA . When the V_{PP} 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, the internal pull-up transistor turns on and quickly pulls the pin back up to about 3.5 V. The pull-up becomes ineffective and the external resistor (R_1) takes over and pulls the voltage up to V_{CC} (see recovery time in Figure 14-4). The time constant follows an exponential charging curve. If $R_1 = 1 \text{ M}\Omega$ and $C_1 = 1 \mu\text{F}$, the recovery time will be one second.

14.4.3.4 Selecting R_1 and C_1

The values of R_1 and C_1 are not critical. Select components that 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.

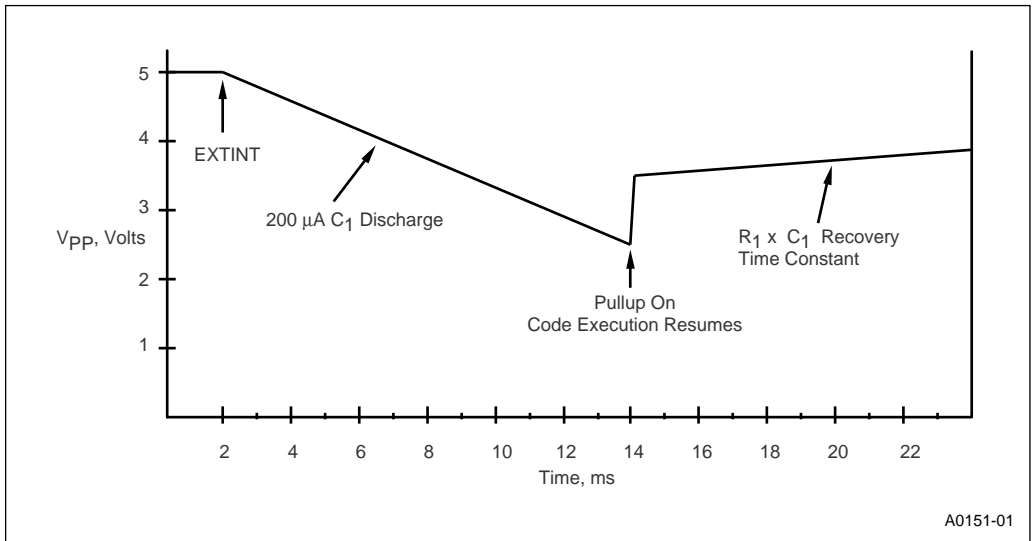


Figure 14-4. Typical Voltage on the V_{PP} Pin While Exiting Powerdown

Select a resistor that will not interfere with the discharge current. In most cases, values between 200 k Ω and 1 M Ω should perform satisfactorily.

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 discharge ($T_{DIS} = 12.5$ ms), V_t is 2.5 V, and the discharge current is 200 μ A. The minimum C_1 capacitor size is 1 μ F.

$$C_1 = \frac{(0.0125) (0.0002)}{2.5} = 1 \mu\text{F}$$

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 μ s.

$$T_{DIS} = \frac{C_1 \times V_t}{I} = \frac{(1.0 \times 10^{-10}) (2.5)}{0.0002} = 1.25 \mu\text{s}$$

14.5 ONCE MODE

On-circuit emulation (ONCE) mode isolates the device from other components in the system to allow printed-circuit-board testing or debugging with a clip-on emulator. During ONCE mode, all pins except XTAL1, XTAL2, V_{SS} , and V_{CC} are weakly pulled high or low. During ONCE mode, RESET# must be held high or the device will exit ONCE mode and enter the reset state.

Holding the ONCE# signal low during the rising edge of RESET# causes the device to enter ONCE mode. To prevent accidental entry into ONCE mode, we highly recommend configuring this pin as an output. If you choose to configure this pin as an input, always hold it high during reset and ensure that your system meets the V_{IH} specification (see datasheet) to prevent inadvertent entry into ONCE mode.

Exit ONCE mode by asserting the RESET# signal and allowing the ONCE# pin to float or be pulled high. Normal operations resume when RESET# goes high.

14.6 RESERVED TEST MODES

A special test-mode-entry pin (P2.6) is provided for Intel's in-house testing only. These test modes can be entered accidentally if you configure the test-mode-entry pin as an input and hold it low during the rising edge of RESET#. To prevent accidental entry into an unsupported test mode, we highly recommend configuring the test-mode-entry pin as an output. If you choose to configure this pin as an input, always hold it high during reset and ensure that your system meets the V_{IH} specification (see datasheet) to prevent inadvertent entry into an unsupported test mode.



15

Interfacing with External Memory

CHAPTER 15

INTERFACING WITH EXTERNAL MEMORY

The microcontroller can interface with a variety of external memory devices. It supports either a fixed 8-bit data bus width, a fixed 16-bit data bus width, or a dynamic 8-bit/16-bit data bus width; internal control of wait states for slow external memory devices; and several bus-control modes. 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.

15.1 EXTERNAL MEMORY INTERFACE SIGNALS AND REGISTERS

Table 15-1 lists the signals and Table 15-2 lists the registers that are mentioned in this chapter. Many of the external memory interface signals are multiplexed with standard I/O port signals, as shown in the *Port Pin* column. Table 15-3 gives the port register settings to configure the pins as external memory interface signals rather than standard I/O port signals. See Chapter 6, “I/O Ports,” to configure the pins as standard I/O port signals.

Table 15-1. External Memory Interface Signals

Signal Name	Port Pin	Type	Description
AD15:0	P4.7:0 P3.7:0	I/O	<p>Address/Data Lines</p> <p>These pins provide a multiplexed address and data bus. During the address phase of the bus cycle, address bits 0–15 are presented on the bus and can be latched using ALE or ADV#. During the data phase, 8- or 16-bit data is transferred.</p>
ADV#	P5.0	O	<p>Address Valid</p> <p>This active-low output signal is asserted only during external memory accesses. ADV# indicates that valid address information is available on the system address/data bus. The signal remains low while a valid bus cycle is in progress and is returned high as soon as the bus cycle completes.</p> <p>An external latch can use this signal to demultiplex the address from the address/data bus. A decoder can also use this signal to generate chip selects for external memory.</p>

Table 15-1. External Memory Interface Signals (Continued)

Signal Name	Port Pin	Type	Description																				
ALE	P5.0	O	<p>Address Latch Enable</p> <p>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. ALE differs from ADV# in that it does not remain active during the entire bus cycle.</p> <p>An external latch can use this signal to demultiplex the address from the address/data bus.</p>																				
BHE#	P5.5	O	<p>Byte High Enable[†]</p> <p>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 AD0, to determine which memory byte is being transferred over the system bus:</p> <table border="1"> <thead> <tr> <th>BHE#</th> <th>AD0</th> <th>Byte(s) Accessed</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>both bytes</td> </tr> <tr> <td>0</td> <td>1</td> <td>high byte only</td> </tr> <tr> <td>1</td> <td>0</td> <td>low byte only</td> </tr> </tbody> </table> <p>[†] The chip configuration register 0 (CCR0) determines whether this pin functions as BHE# or WRH#. CCR0.2 = 1 selects BHE#; CCR0.2 = 0 selects WRH#.</p>	BHE#	AD0	Byte(s) Accessed	0	0	both bytes	0	1	high byte only	1	0	low byte only								
BHE#	AD0	Byte(s) Accessed																					
0	0	both bytes																					
0	1	high byte only																					
1	0	low byte only																					
BUSWIDTH	P5.7	I	<p>Bus Width</p> <p>Two chip configuration register bits, CCR0.1 and CCR1.2, along with the BUSWIDTH pin, control the data bus width. When both CCR bits are set, the BUSWIDTH signal selects the external data bus width. When only one CCR bit is set, the bus width is fixed at either 16 or 8 bits, and the BUSWIDTH signal has no effect.</p> <table border="1"> <thead> <tr> <th>CCR0.1</th> <th>CCR1.2</th> <th>BUSWIDTH</th> <th></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1</td> <td>X</td> <td>fixed 8-bit data bus</td> </tr> <tr> <td>1</td> <td>0</td> <td>X</td> <td>fixed 16-bit data bus</td> </tr> <tr> <td>1</td> <td>1</td> <td>high</td> <td>16-bit data bus</td> </tr> <tr> <td>1</td> <td>1</td> <td>low</td> <td>8-bit data bus</td> </tr> </tbody> </table>	CCR0.1	CCR1.2	BUSWIDTH		0	1	X	fixed 8-bit data bus	1	0	X	fixed 16-bit data bus	1	1	high	16-bit data bus	1	1	low	8-bit data bus
CCR0.1	CCR1.2	BUSWIDTH																					
0	1	X	fixed 8-bit data bus																				
1	0	X	fixed 16-bit data bus																				
1	1	high	16-bit data bus																				
1	1	low	8-bit data bus																				
CLKOUT (MC, MD)	—	O	<p>Clock Output</p> <p>Output of the internal clock generator. The CLKOUT frequency is $\frac{1}{2}$ the oscillator input frequency (F_{XTAL1}). CLKOUT has a 50% duty cycle.</p>																				

Table 15-1. External Memory Interface Signals (Continued)

Signal Name	Port Pin	Type	Description
EA#	—	I	<p>External Access</p> <p>This input determines whether memory accesses to special-purpose and program memory partitions are directed to internal or external memory. (See Table 4-1 on page 4-2 for address ranges of special-purpose and program memory partitions.) 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.</p> <p>EA# also controls entry into the programming modes. If EA# is at V_{PP} voltage (typically +12.5 V) on the rising edge of RESET#, the microcontroller enters a programming mode.</p> <p>NOTE: Systems with EA# tied inactive have idle time between external bus cycles. When the address/data bus is idle, you can use ports 3 and 4 for I/O. Systems with EA# tied active cannot use ports 3 and 4 as standard I/O; when EA# is active, these ports will function only as the address/data bus.</p> <p>EA# is sampled and latched only on the rising edge of RESET#. Changing the level of EA# after reset has no effect.</p> <p>Always connect EA# to V_{SS} when using a microcontroller that has no internal nonvolatile memory.</p>
INST	P5.1	O	<p>Instruction Fetch</p> <p>This active-high output signal is valid only during external memory bus cycles. 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.</p>
RD#	P5.3	O	<p>Read</p> <p>Read-signal output to external memory. RD# is asserted only during external memory reads.</p>
READY	P5.6	I	<p>Ready Input</p> <p>This active high input along with the chip configuration registers determine the number of wait states inserted into the bus cycle. The chip configuration registers selects the maximum number of wait states (0, 1, 2, 3, or infinite) that can be inserted into the bus cycle. While READY is low, wait states are inserted into the bus cycle until the programmed number of wait states is reached. If READY is pulled high before the programmed number of wait states is reached, no additional wait states will be inserted into the bus cycle.</p>
WR#	P5.2	O	<p>Write[†]</p> <p>This active-low output indicates that an external write is occurring. This signal is asserted only during external memory writes.</p> <p>[†] 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#.</p>

Table 15-1. External Memory Interface 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. [†] The chip configuration register 0 (CCR0) determines whether this pin 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. [†] 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#.

Table 15-2. External Memory Interface Registers

Register Mnemonic	Address	Description
CCR0	2018H	Chip Configuration 0 Controls powerdown mode, bus-control signals, and internal memory protection. Three of its bits combine with two bits of CCR1 to control wait states and bus width.
CCR1	201AH	Chip Configuration 1 Enables the watchdog timer and selects the bus timing mode. Two of its bits combine with three bits of CCR0 to control wait states and bus width.
P5_DIR	1FF3H	Port 5 Direction Each bit of P5_DIR controls the direction of the corresponding pin. Clearing a bit configures a pin as a complementary output; setting a bit configures a pin as an input or open-drain output. (Open-drain outputs require external pull-ups.)
P5_MODE	1FF1H	Port 5 Mode Each bit of P5_MODE controls whether the corresponding pin functions as a standard I/O port pin or as a special-function signal. Setting a bit configures a pin as a special-function signal; clearing a bit configures a pin as a standard I/O port pin.
P5_PIN	1FF7H	Port 5 Input Each bit of P5_PIN reflects the current state of the corresponding pin, regardless of the pin configuration.

Table 15-2. External Memory Interface Registers (Continued)

Register Mnemonic	Address	Description
P5_REG	1FF5H	<p>Port 5 Data Output</p> <p>For an input, regardless of the pin's configuration, set the corresponding P5_REG bit.</p> <p>For an output, write the data to be driven out by each pin to the corresponding bit of P5_REG. When a pin is configured as standard I/O (P5_MODE.y = 0), the result of a CPU write to P5_REG is immediately visible on the pin. When a pin is configured as a special-function signal (P5_MODE.y = 1), the associated on-chip peripheral or off-chip component controls the pin. The CPU can still write to P5_REG, but the pin is unaffected until it is switched back to its standard I/O function.</p> <p>This feature allows software to configure a pin as standard I/O (clear P5_MODE.y), initialize or overwrite the pin value, then configure the pin as a special-function signal (set P5_MODE.y). In this way, initialization, fault recovery, exception handling, etc., can be done without changing the operation of the associated peripheral.</p>

Table 15-3. Register Settings for Configuring External Memory Interface Signals

Port Pin	External Memory Interface Signal Name	Signal Type	Port Register Settings
P5.0	ALE/ADV#	O	P5_DIR = 110X 0000B
P5.1	INST	O	P5_MODE = 111X 1111B
P5.2	WR#/WRL# [†]	O	P5_REG = 11XX XXXXB
P5.3	RD#	O	
P5.5	BHE#/WRH# [†]	O	
P5.6	READY	I	
P5.7	BUSWIDTH (Kx)	I	

[†] The chip configuration register 0 (CCR0), which is loaded at reset from the chip configuration byte 0 (CCB0, 2018H) during normal operation, determines whether P5.2 functions as BHE# or WRH# and whether P5.5 functions as WR# or WRL#. CCR0.2 = 1 selects BHE# and WR#; CCR0.2 = 1 selects WRH# and WRL#.

15.2 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 in nonvolatile memory at addresses 2018H (CCB0) and 201AH (CCB1).

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 15-1 and 15-2. (Refer to Chapter 16, “Programming the Nonvolatile Memory,” for descriptions of the methods for programming the CCBs.)

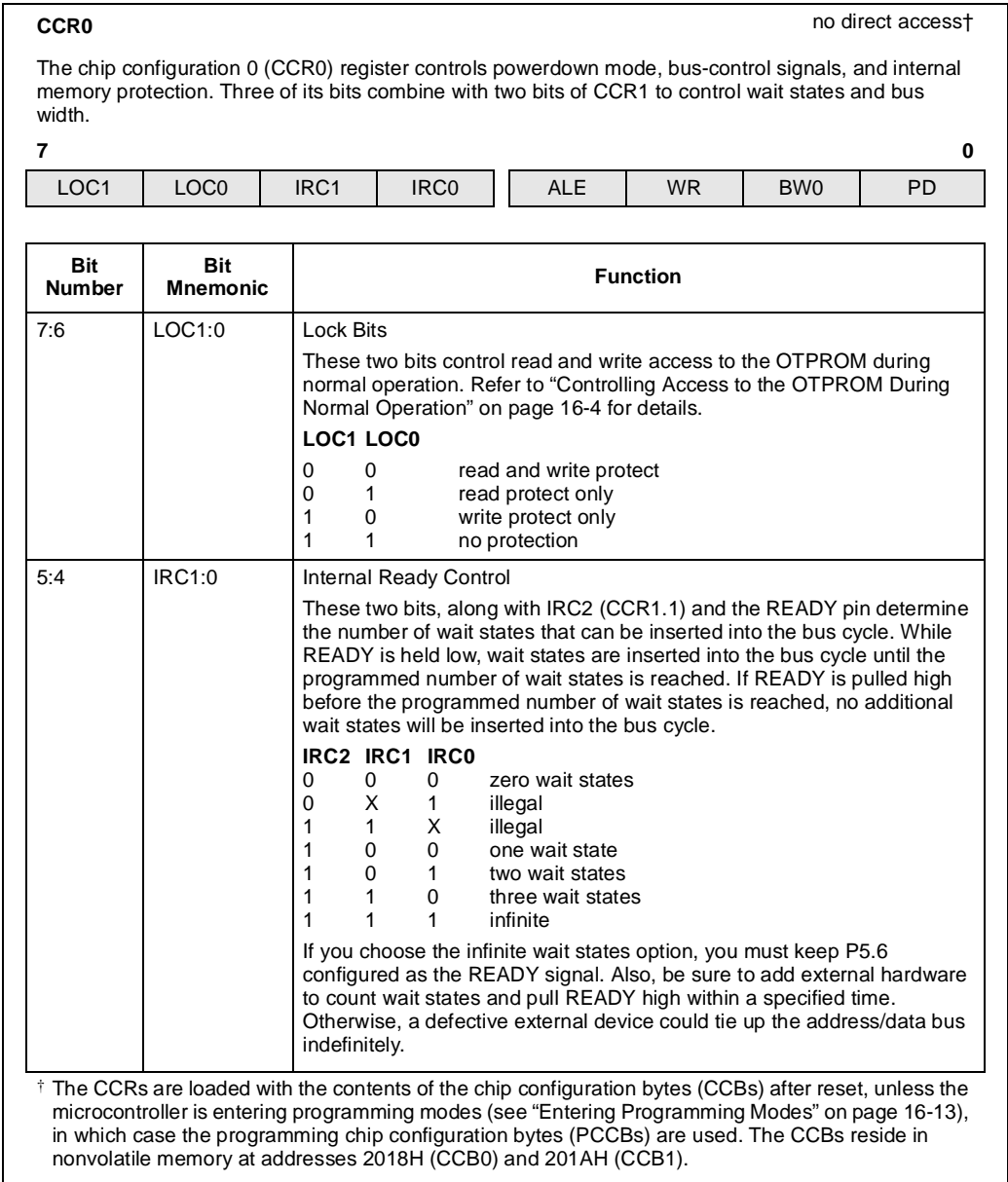


Figure 15-1. Chip Configuration 0 (CCR0) Register

CCR0 (Continued)				no direct access†			
The chip configuration 0 (CCR0) register controls powerdown mode, bus-control signals, and internal memory protection. Three of its bits combine with two bits of CCR1 to control wait states and bus width.							
7				0			
LOC1	LOC0	IRC1	IRC0	ALE	WR	BW0	PD
Bit Number	Bit Mnemonic	Function					
3	ALE	Address Valid Strobe and Write Strobe					
2	WR	These bits define which bus-control signals will be generated during external read and write cycles.					
		ALE WR 0 0 address valid with write strobe mode (ADV#, RD#, WRL#, WRH#) 0 1 address valid strobe mode (ADV#, RD#, WR#, BHE#) 1 0 write strobe mode (ALE, RD#, WRL#, WRH#) 1 1 standard bus-control mode (ALE, RD#, WR#, BHE#)					
1	BW0	Buswidth Control This bit, along with the BW1 bit (CCR1.2), selects the bus width. BW1 BW0 0 0 illegal 0 1 16-bit only 1 0 8-bit only 1 1 BUSWIDTH pin controlled					
0	PD	Powerdown Enable Controls whether the IDLPD #2 instruction causes the microcontroller to enter powerdown mode. If your design uses 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 powerdown mode. 0 = disable powerdown mode 1 = enable powerdown mode					
† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after reset, unless the microcontroller is entering programming modes (see "Entering Programming Modes" on page 16-13), in which case the programming chip configuration bytes (PCCBs) are used. The CCBs reside in nonvolatile memory at addresses 2018H (CCB0) and 201AH (CCB1).							

Figure 15-1. Chip Configuration 0 (CCR0) Register (Continued)

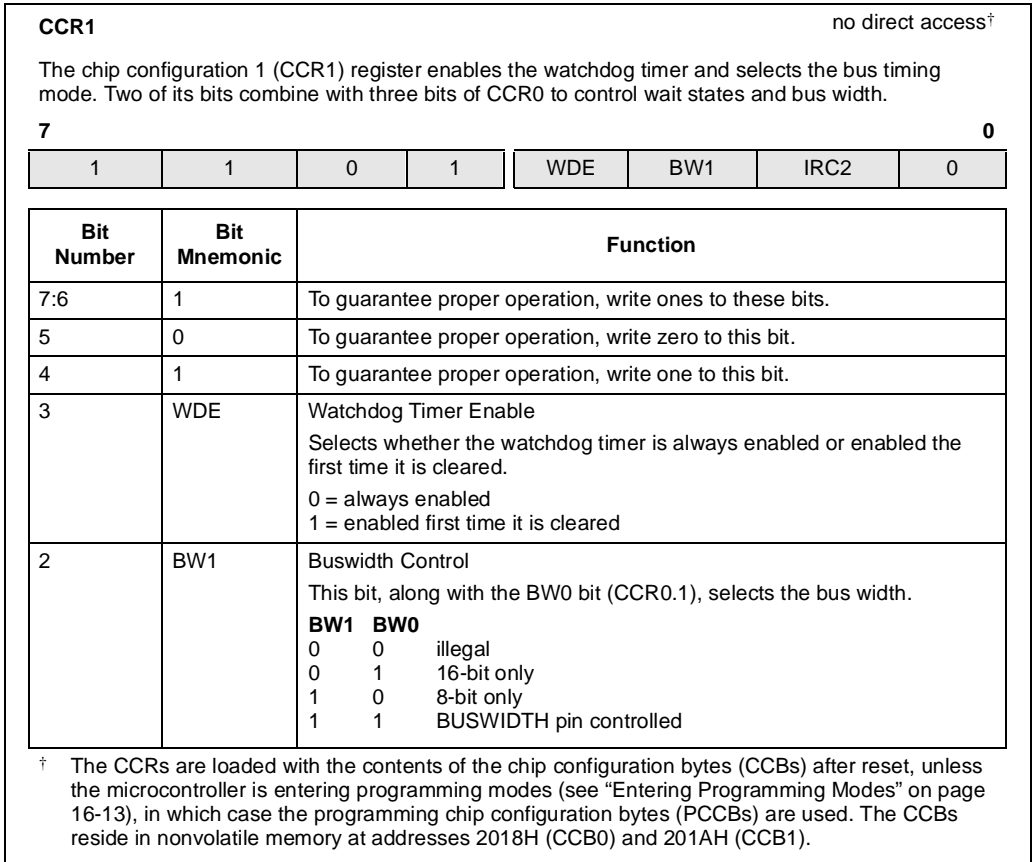


Figure 15-2. Chip Configuration 1 (CCR1) Register

CCR1 (Continued)				no direct access [†]																																			
The chip configuration 1 (CCR1) register enables the watchdog timer and selects the bus timing mode. Two of its bits combine with three bits of CCR0 to control wait states and bus width.																																							
7				0																																			
1	1	0	1	WDE	BW1	IRC2	0																																
Bit Number	Bit Mnemonic	Function																																					
1	IRC2	<p>Ready Control</p> <p>This bit, along with IRC0 (CCR0.4), IRC1 (CCR0.5), and the READY pin determine the number of wait states that can be inserted into the bus cycle. While READY is held low, wait states are inserted into the bus cycle until the programmed number of wait states is reached. If READY is pulled high before the programmed number of wait states is reached, no additional wait states will be inserted into the bus cycle.</p> <table border="1"> <thead> <tr> <th>IRC2</th> <th>IRC1</th> <th>IRC0</th> <th></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>zero wait states</td> </tr> <tr> <td>0</td> <td>X</td> <td>1</td> <td>illegal</td> </tr> <tr> <td>1</td> <td>1</td> <td>X</td> <td>illegal</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>one wait state</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> <td>two wait states</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> <td>three wait states</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>infinite</td> </tr> </tbody> </table> <p>If you choose the infinite wait states option, you must keep P5.6 configured as the READY signal. Also, be sure to add external hardware to count wait states and pull READY high within a specified time. Otherwise, a defective external device could tie up the address/data bus indefinitely.</p>						IRC2	IRC1	IRC0		0	0	0	zero wait states	0	X	1	illegal	1	1	X	illegal	1	0	0	one wait state	1	0	1	two wait states	1	1	0	three wait states	1	1	1	infinite
IRC2	IRC1	IRC0																																					
0	0	0	zero wait states																																				
0	X	1	illegal																																				
1	1	X	illegal																																				
1	0	0	one wait state																																				
1	0	1	two wait states																																				
1	1	0	three wait states																																				
1	1	1	infinite																																				
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, unless the microcontroller is entering programming modes (see "Entering Programming Modes" on page 16-13), in which case the programming chip configuration bytes (PCCBs) are used. The CCBs reside in nonvolatile memory at addresses 2018H (CCB0) and 201AH (CCB1).

Figure 15-2. Chip Configuration 1 (CCR1) Register

15.3 BUS WIDTH AND MULTIPLEXING

The external bus can operate as either a 16-bit multiplexed address/data bus or as a multiplexed 16-bit address/8-bit data bus (Figure 15-3).

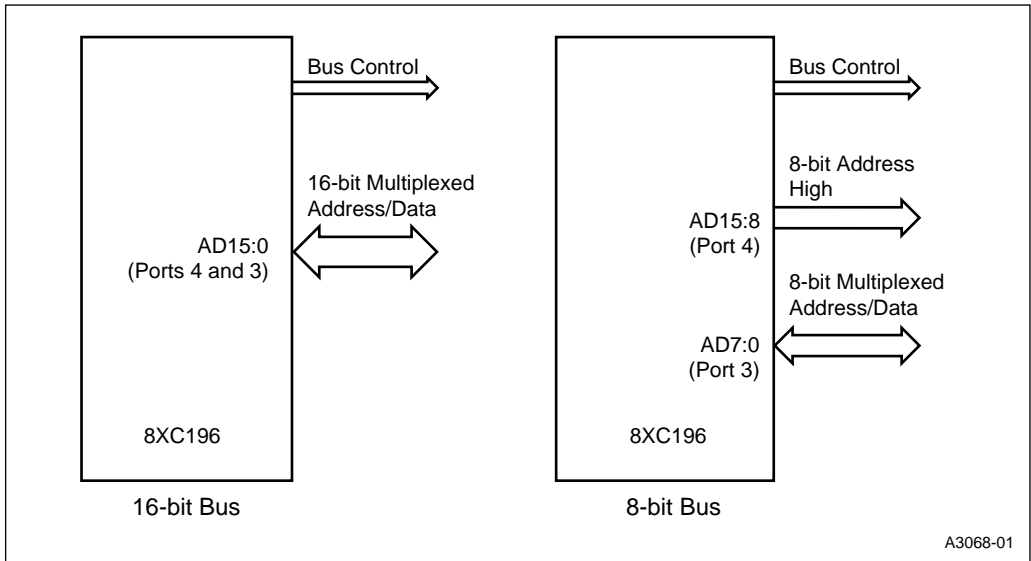


Figure 15-3. Multiplexing and Bus Width Options

After reset, but before the CCB fetch, the microcontroller is configured for 8-bit bus mode, regardless of the BUSWIDTH input. The upper address lines (AD15:8) are weakly driven throughout the CCB0 and CCB1 bus cycles. To prevent bus contention, neither pull-ups nor pull-downs should be used on AD15:8. Also, the upper bytes of the CCB words (locations 2019H and 201BH) should be loaded with 20H. If the external memory outputs 20H on its high byte, there will be no bus contention.

After the CCBs are loaded into the CCRs, the values of BW0 and BW1 define the data bus width as either a fixed 8-bit, a fixed 16-bit, or a dynamic 16-bit/8-bit bus width controlled by the BUSWIDTH signal. (The BW0 and BW1 bits are defined in Figures 15-1 and 15-2.)

If BW0 is clear and BW1 is set, the bus controller is locked into an 8-bit bus mode. In comparing an 8-bit bus system to a 16-bit bus system, expect some performance degradation. In a 16-bit bus system, a word fetch is done with a single word fetch. However, in an 8-bit bus system, a word fetch takes an additional bus cycle because it must be done with two byte fetches.

If BW0 is set and BW1 is clear, the bus controller is locked into a 16-bit bus mode. If both BW0 and BW1 are set, the BUSWIDTH signal controls the bus width. The bus is 16 bits wide when BUSWIDTH is high, and 8 bits wide when BUSWIDTH is low. The BUSWIDTH signal is sampled after the address is on the bus, as shown in Figures 15-4 and 15-5.

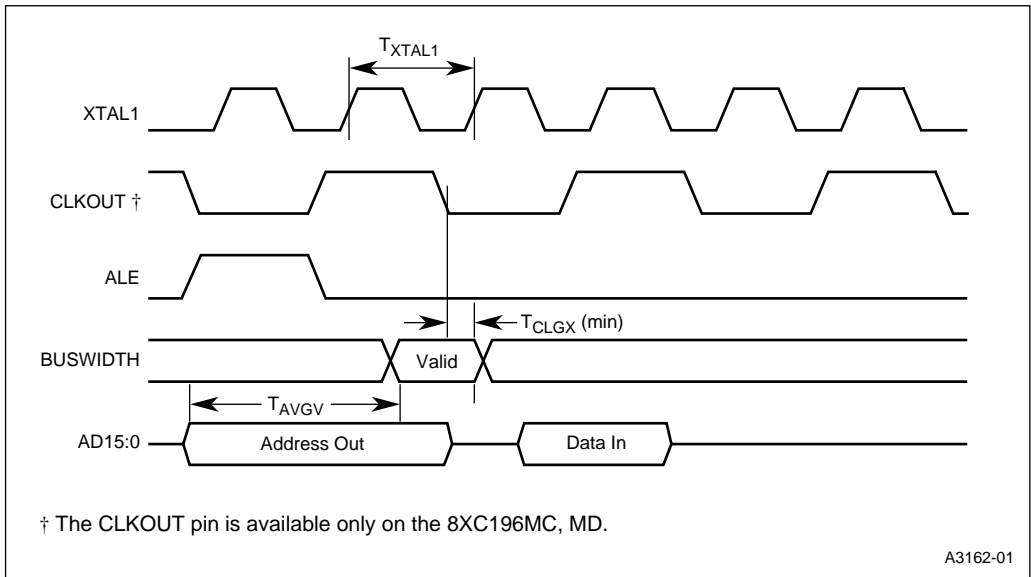


Figure 15-4. BUSWIDTH Timing Diagram (8XC196MC, MD)

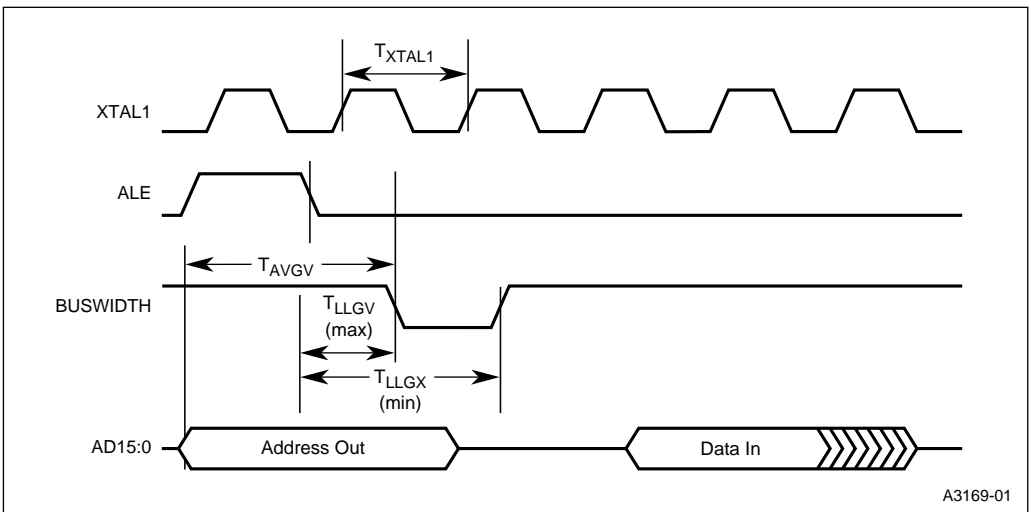


Figure 15-5. BUSWIDTH Timing Diagram (8XC196MH)

Table 15-4. BUSWIDTH Signal Timing Definitions

Symbol	Definition
T_{AVGV}	Address Valid to BUSWIDTH Setup Maximum time the external device has to assert or deassert BUSWIDTH after the microcontroller outputs the address.
T_{CLGX}^{\dagger}	BUSWIDTH Hold after CLKOUT Low Minimum time the level of the BUSWIDTH signal must be valid after CLKOUT falls.
$T_{LLGV}^{\dagger\dagger}$	ALE Low to BUSWIDTH Setup Maximum time the external device has to assert or deassert BUSWIDTH after ALE falls.
$T_{LLGX}^{\dagger\dagger}$	BUSWIDTH Hold after ALE Low Minimum time the level of the BUSWIDTH signal must be valid after ALE falls.
T_{XTAL1}	$1/F_{XTAL1}$ All AC timings are referenced to T_{XTAL1} .

\dagger This specification applies to the 8XC196MC, MD microcontrollers only.

$\dagger\dagger$ This specification applies to the 8XC196MH microcontroller only.

The BUSWIDTH signal can be used in numerous applications. For example, a system could store code in a 16-bit memory device and data in an 8-bit memory device. The BUSWIDTH signal could be tied to the chip-select input of the 8-bit memory device (shown in Figure 15-13 on page 15-24). When BUSWIDTH is low, it enables 8-bit bus mode and selects the 8-bit memory device. When BUSWIDTH is high, it enables 16-bit bus mode and deselects the 8-bit memory device.

15.3.1 Timing Requirements for BUSWIDTH

When using BUSWIDTH to dynamically change between 8-bit and 16-bit bus widths, setup and hold timings must be met for proper operation (see Figures 15-4 and 15-5, and Table 15-4). Because a decoded, valid address is used to generate the BUSWIDTH signal, the setup time is specified relative to the address being valid. This specification, T_{AVGV} , indicates how much time an external device has to decode the valid address and generate a valid BUSWIDTH signal. As shown in Figure 15-5, the 8XC196MH has an additional BUSWIDTH setup timing specification. This specification, T_{LLGV} , indicates how much time an external device has to generate a valid BUSWIDTH signal after ALE falls.

BUSWIDTH must be held valid until the minimum hold specification, T_{CLGX} (for 8XC196MC, MD) or T_{LLGX} (for 8XC196MH), has been met. Refer to the datasheet for the current T_{AVGV} , T_{CLGX} , and T_{LLGX} specifications.

15.3.2 16-bit Bus Timings

When the microcontroller is configured to operate in the 16-bit bus-width mode, lines AD15:0 form a 16-bit multiplexed address/data bus. Figure 15-6 shows an idealized timing diagram for the external read and write cycles. Comprehensive timing specifications are shown in Figure 15-22 on page 15-32.

The rising edge of the address latch enable (ALE) signal indicates that the microcontroller is driving an address onto the bus (AD15:0). The microcontroller presents a valid address before ALE falls. 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 the bus controller puts data onto AD15:0.

For 16-bit read cycles, the bus controller floats the bus and then drives RD# low so that it can receive data. The external memory must put data (Data In) onto the bus before the rising edge of RD#. The datasheet specifies the maximum time the memory device has to output valid data after RD# is asserted (T_{RLDV}). When INST is asserted, it indicates that the read operation is an instruction fetch.

For 16-bit write cycles, the bus controller drives WR# low, then puts data onto the bus. The rising edge of WR# signifies that data is valid. At this time, the external system must latch the data.

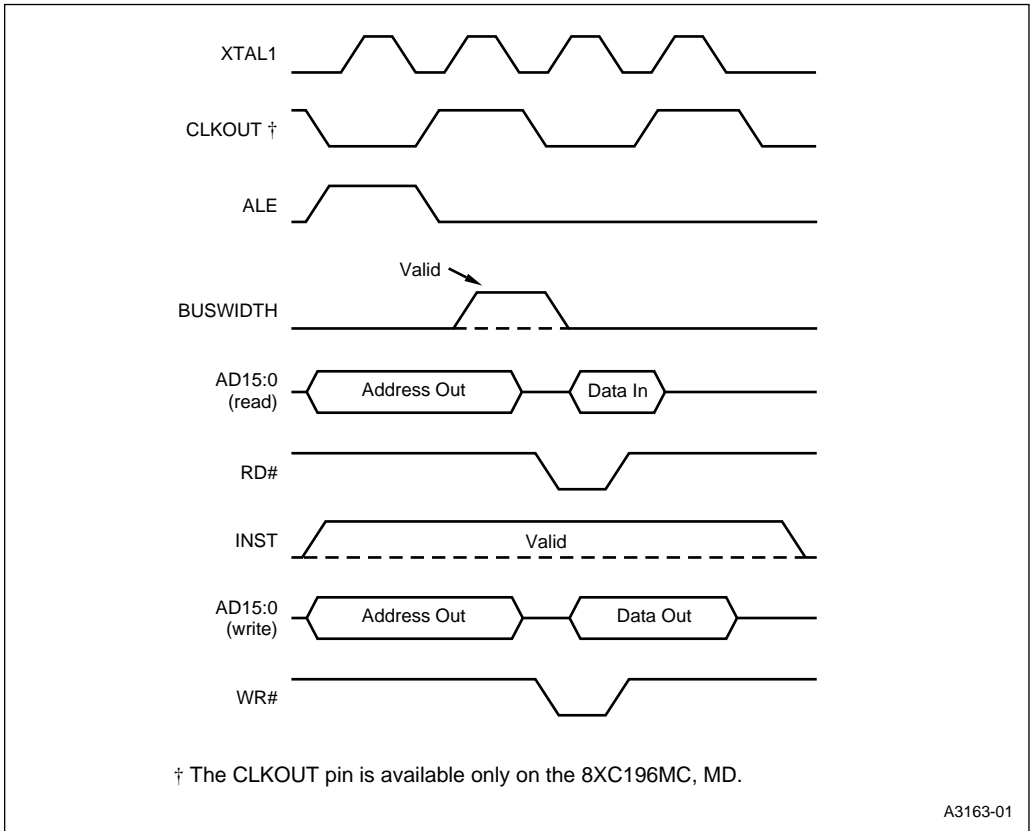


Figure 15-6. Timings for 16-bit Buses

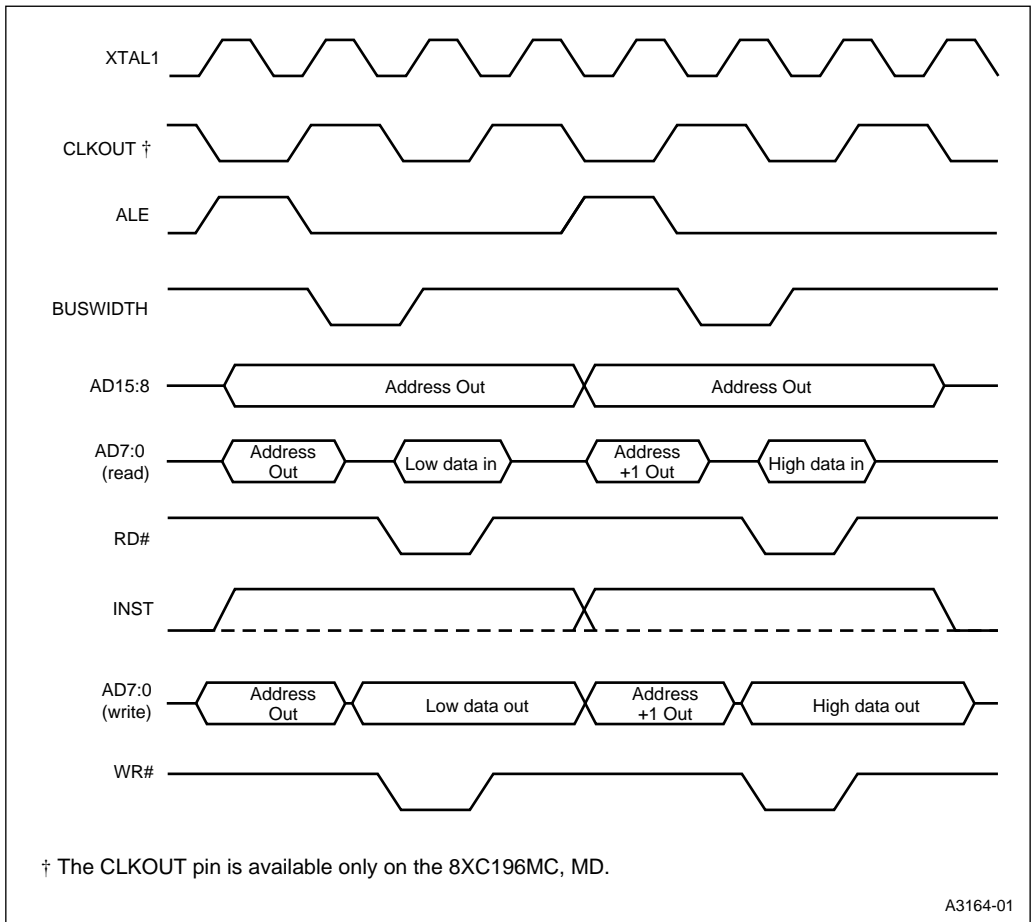
15.3.3 8-bit Bus Timings

When the microcontroller is configured to operate in the 8-bit bus mode, lines AD7:0 form a multiplexed lower address and data bus. Lines AD15:8 are not multiplexed; the upper address is latched and remains valid throughout the bus cycle. Figure 15-7 shows an idealized timing diagram for the external read and write cycles. 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.

The ALE signal is used to demultiplex the lower address by strobing a transparent latch (such as a 74AC373).

For 8-bit bus read cycles, after ALE falls, the bus controller floats the bus and drives the RD# signal low. The external memory then must put its data on the bus. That data must be valid at the rising edge of the RD# signal. To read a data word, the bus controller performs two consecutive reads, reading the low byte first, followed by the high byte.

For 8-bit bus write cycles, after ALE falls, the bus controller outputs data on AD7:0 and then drives WR# low. The external memory must latch the data by the time WR# goes high. That data will be valid on the bus until slightly after WR# goes high. To write a data word, the bus controller performs two consecutive writes, writing the low byte first, followed by the high byte.


Figure 15-7. Timings for 8-bit Buses

15.4 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 raises the READY signal. Each wait state adds one state time ($2T_{XTAL1}$) to the bus cycle.

After reset and until CCB1 is fetched, the bus controller always inserts three wait states into bus cycles. Then, until P5.6 has been configured to operate as the READY signal, the internal ready control bits (IRC2:0) control the wait states.

After the CCBI fetch, the internal ready control circuitry allows slow external memory devices to increase the length of the read and write bus cycles. If the external memory device is not ready for access, it pulls the READY signal low and holds it low until it is ready to complete the operation, at which time it releases READY. While READY is low, the bus controller inserts wait states into the bus cycle.

The internal ready control bits, CCR0.5, CCR0.4, and CCR1.1 shown in Figures 15-1 and 15-2, define the maximum number of wait states (0, 1, 2, 3, or infinite) that will be inserted into the bus cycle. While READY is low, wait states are inserted into the bus cycle until the programmed number of wait states is reached. If READY is pulled high before the programmed number of wait states is reached, no additional wait states will be inserted into the bus cycle.

If you choose the infinite wait states option, you must keep P5.6 configured as the READY signal. Also, be sure to add external hardware to count wait states and pull READY high within a specified time. Otherwise, a defective external device could tie up the address/data bus indefinitely.

Setup and hold timings must be met when using the READY signal to insert wait states into a bus cycle (see Figures 15-8 and 15-9, and Table 15-5). 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. As shown in Figure 15-9, the 8XC196MH has an additional READY setup timing specification. This specification, T_{LLYV} , indicates how much time an external device has to deassert the READY signal after ALE falls.

The READY signal must be held low until the minimum T_{CLYX} timing specification is met. Do not exceed the maximum T_{CLYX} or T_{LLYX} specification or additional (unwanted) wait states might be added. Refer to the datasheets for the current T_{AVYV} , T_{CLYX} , and T_{LLYX} specifications.

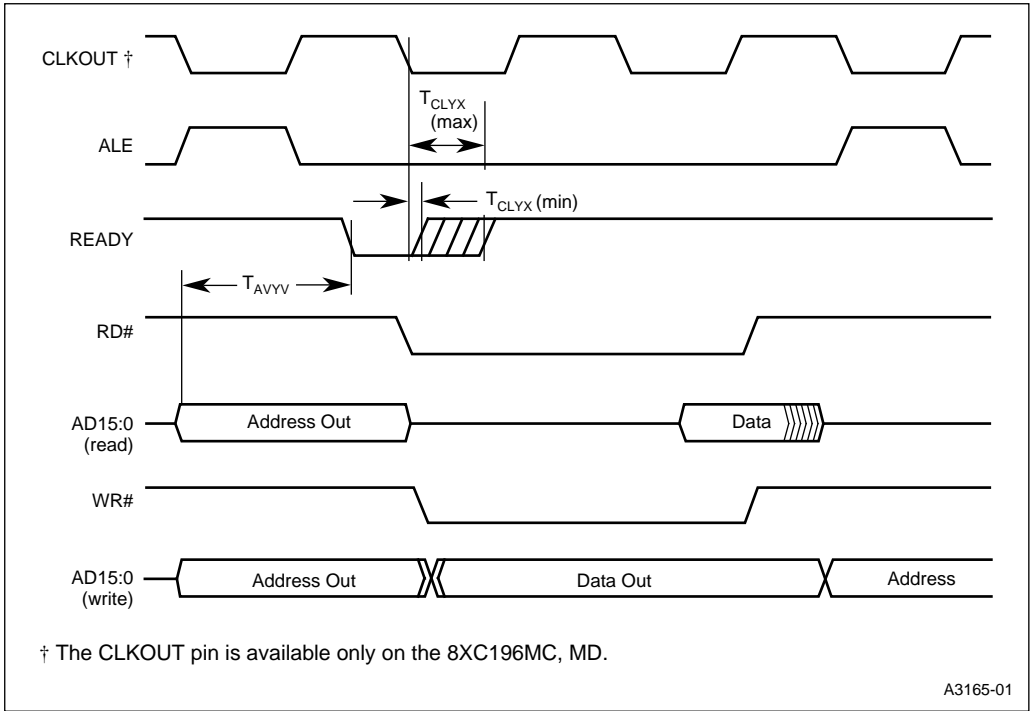


Figure 15-8. READY Timing Diagram — One Wait State (8XC196MC, MD)

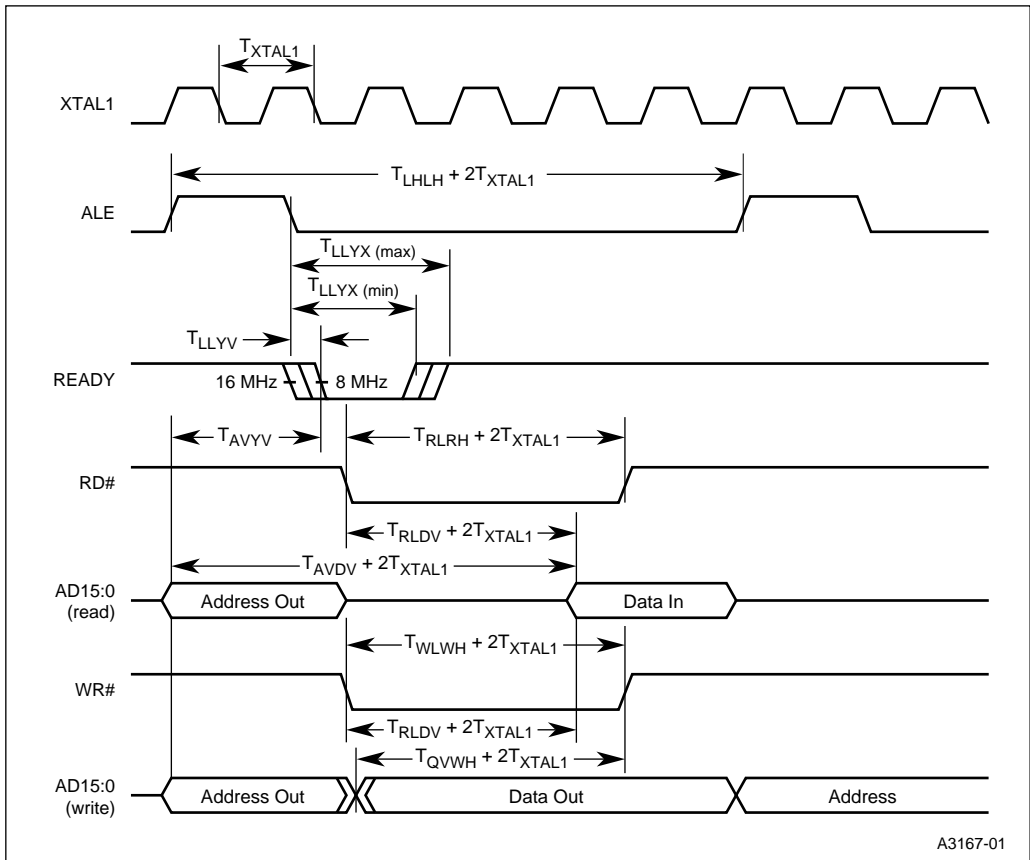


Figure 15-9. READY Timing Diagram — One Wait State (8XC196MH)

Table 15-5. READY Signal Timing Definitions

Symbol	Definition
T_{AVYV}	Address Valid to READY Setup Maximum time the external device has to deassert READY after the microcontroller outputs the address to guarantee that at least one wait state will occur.
T_{CLYX}^{\dagger}	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.

[†] This specification applies to the 8XC196MC, MD microcontrollers only.

^{††} This specification applies to the 8XC196MH microcontroller only.

Table 15-5. READY Signal Timing Definitions (Continued)

Symbol	Definition
$T_{LLYX}^{\dagger\dagger}$	READY Hold after ALE Low Minimum time the level of the READY signal must be valid after ALE falls. If the maximum value is exceeded, additional wait states will occur.
$T_{LLYV}^{\dagger\dagger}$	ALE Low to READY Setup Maximum time the external device has to deassert READY after ALE falls.
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.
T_{XTAL1}	$1/F_{XTAL1}$ All AC timings are referenced to T_{XTAL1} .

† This specification applies to the 8XC196MC, MD microcontrollers only.

†† This specification applies to the 8XC196MH microcontroller only.

15.5 BUS-CONTROL MODES

The ALE and WR bits (CCR0.3 and CCR0.2) define which bus-control signals will be generated during external read and write cycles. Table 15-6 lists the four bus-control modes and shows the CCR0.3 and CCR0.2 settings for each.

Table 15-6. Bus-control Modes

Bus-control Mode	Bus-control Signals	CCR0.3 (ALE)	CCR0.2 (WR)
Standard Bus-control Mode	ALE, RD#, WR#, BHE#	1	1
Write Strobe Mode	ALE, RD#, WRL#, WRH#	1	0
Address Valid Strobe Mode	ADV#, RD#, WR#, BHE#	0	1
Address Valid with Write Strobe Mode	ADV#, RD#, WRL#, WRH#	0	0

15.5.1 Standard Bus-control Mode

In the standard bus-control mode, the microcontroller generates the standard bus-control signals: ALE, RD#, WR#, and BHE# (see Figure 15-10). ALE is asserted while the address is driven, and it can be used to latch the address externally. RD# is asserted for every external memory read, and WR# is asserted for every external memory write. When asserted, BHE# selects the bank of memory that is addressed by the high byte of the data bus.

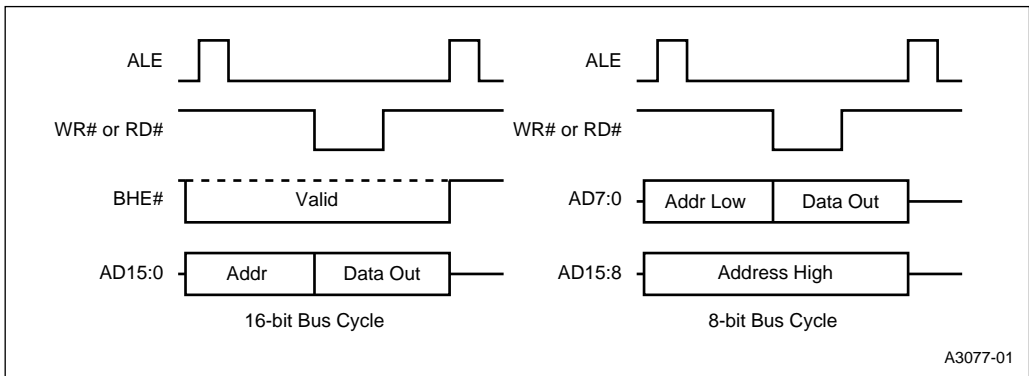


Figure 15-10. Standard Bus Control

When the microcontroller is configured to use a 16-bit bus, separate low- and high-byte write signals must be generated for single-byte writes. Figure 15-11 shows a sample circuit that combines BHE# and AD0 to produce these signals (WRL# and WRH#). A similar pair of signals for read is unnecessary. For a single-byte read with the 16-bit bus, both bytes are placed on the data bus and the processor discards the unwanted byte.

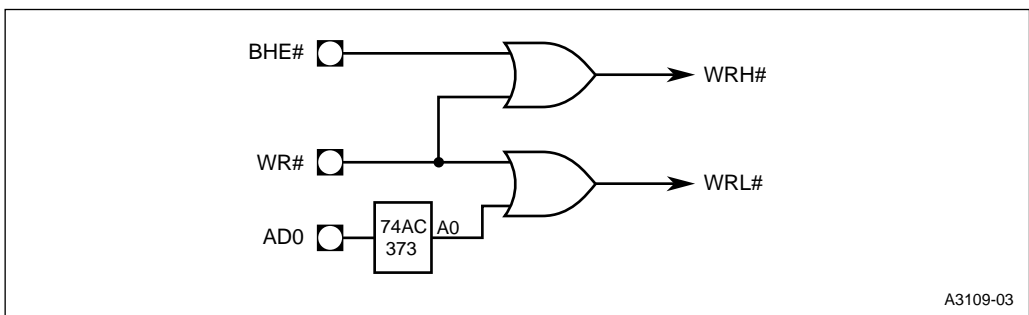


Figure 15-11. Decoding WRL# and WRH#

Figure 15-12 shows an 8-bit system with both flash and RAM. The flash is the lower half of memory, and the RAM is the upper half. This system configuration uses the most-significant address bit (AD15) as the chip-select signal and ALE as the address-latch signal.

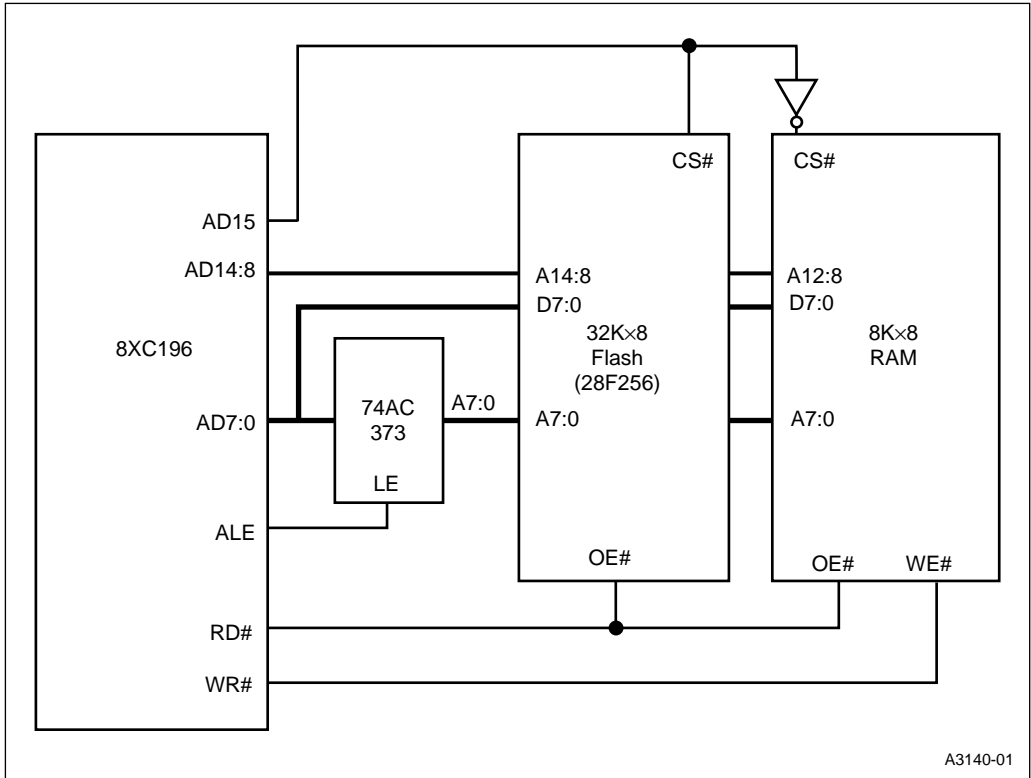


Figure 15-12. 8-bit System with Flash and RAM

A3140-01

Figure 15-13 shows a system that uses the dynamic bus-width feature. (The CCR bits, BW0 and BW1, are set.) Code is executed from the two EPROMs and data is stored in the byte-wide RAM. The RAM is in high memory. It is selected by driving AD15 high, which also selects the 8-bit bus-width mode by driving the BUSWIDTH signal low.

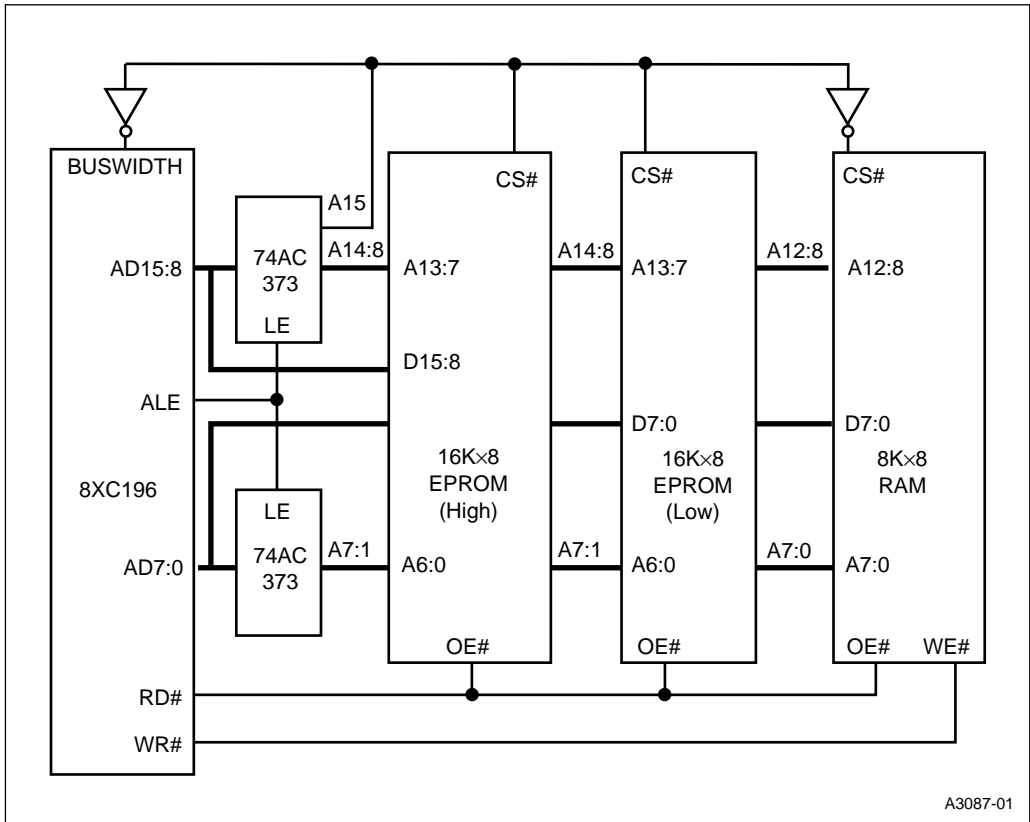


Figure 15-13. 16-bit System with Dynamic Bus Width

15.5.2 Write Strobe Mode

The write strobe mode eliminates the need to externally decode high- and low-byte writes to an external 16-bit RAM or flash device in 16-bit bus mode. When the write strobe mode is selected, the microcontroller generates WRL# and WRH# instead of WR# and BHE#. WRL# is asserted for all low byte writes (even addresses) and all word writes. WRH# is asserted for all high byte writes (odd addresses) and all word writes. In the 8-bit bus mode, WRH# and WRL# are asserted for both even and odd addresses. Figure 15-14 shows write strobe mode timing.

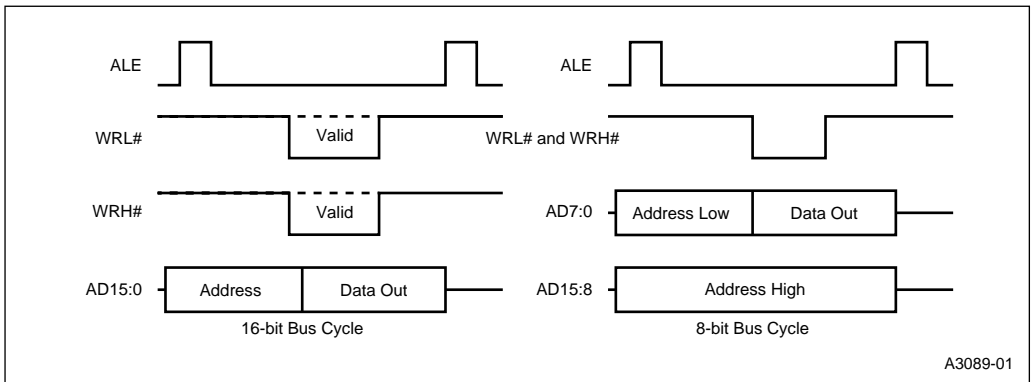


Figure 15-14. Write Strobe Mode

Figure 15-15 shows a 16-bit system with two EPROMs and two RAMs. It is configured to use the write strobe mode. ALE latches the address; AD15 is the chip-select signal for the memory devices. WRL# is asserted during low byte writes and word writes. WRH# is asserted during high byte writes and word writes. Note that the RAM devices do not use AD0. WRL# and WRH# determine whether the low byte (AD0=0) or high byte (AD0=1) is selected.

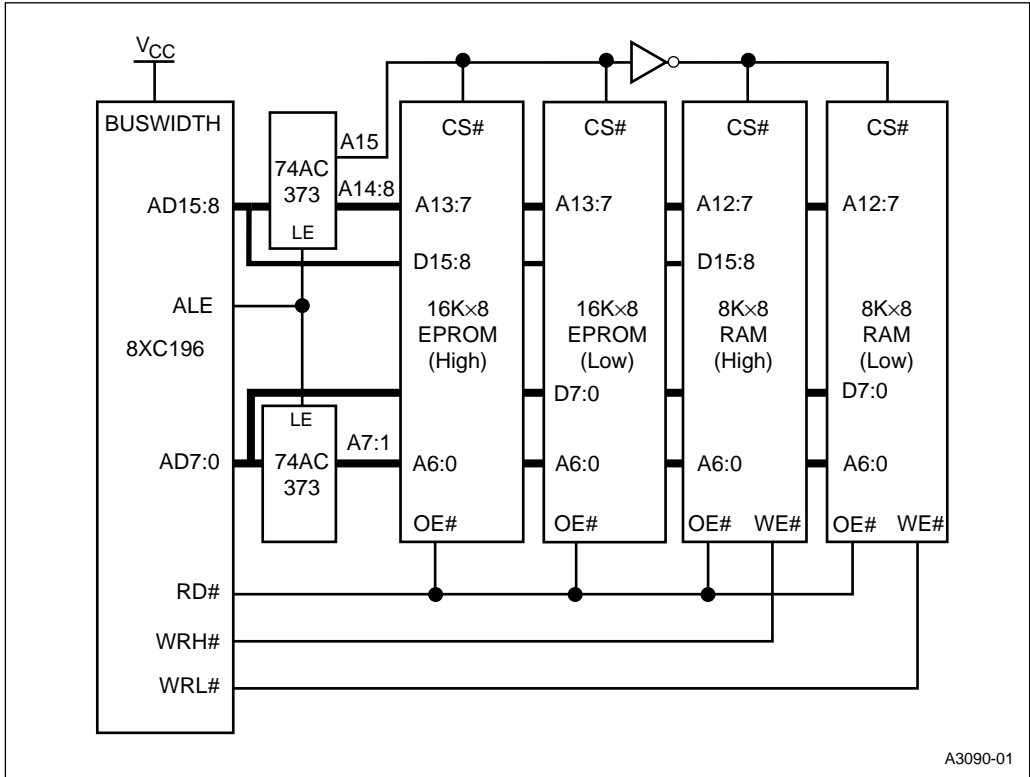


Figure 15-15. 16-bit System with Writes to Byte-wide RAMs

15.5.3 Address Valid Strobe Mode

When the address valid strobe mode is selected, the microcontroller generates the address valid signal (ADV#) instead of the address latch enable signal (ALE). ADV# is asserted after an external address is valid (see Figure 15-16). This signal can be used to latch the valid address and simultaneously enable an external memory device. (See the examples in Figures 15-18 and 15-19.)

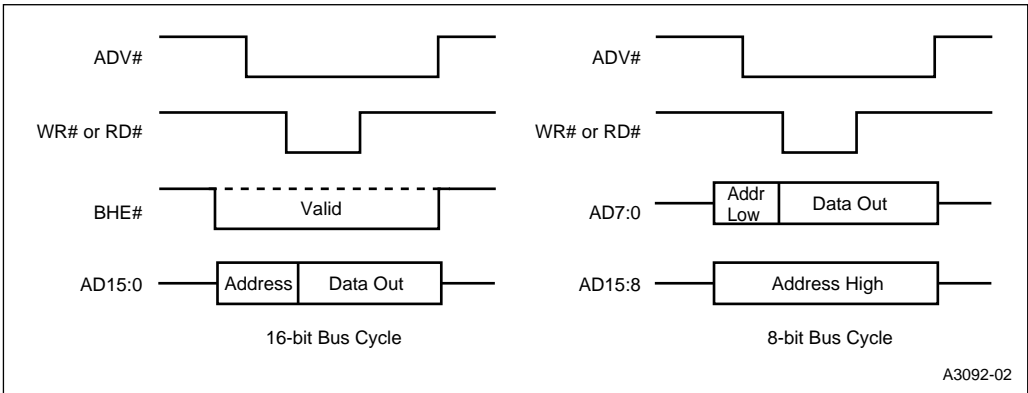


Figure 15-16. Address Valid Strobe Mode

The difference between ALE and ADV# is that ADV# is asserted for the entire bus cycle, not just to latch the address. Figure 15-17 shows the difference between ALE and ADV# for a single read or write cycle. Note that for back-to-back bus access, the ADV# function will look identical to the ALE function. The difference becomes apparent only when the bus is idle. Because ADV# is high during these periods, external memory will be disabled, thus saving power.

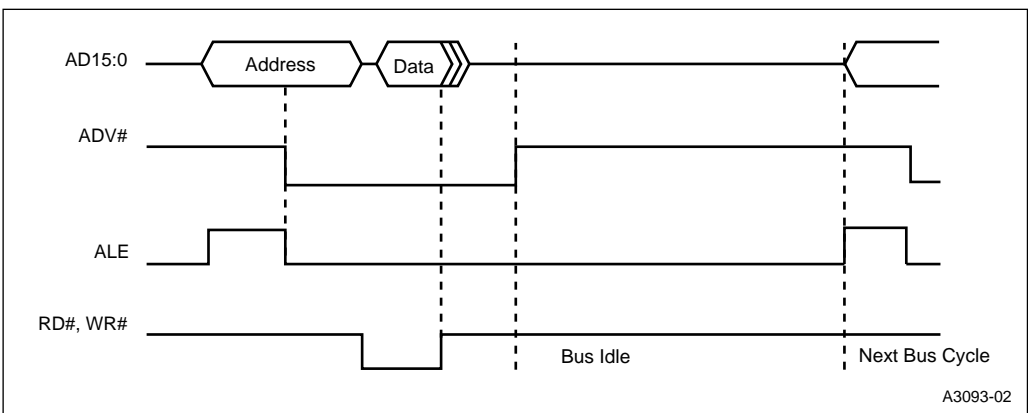


Figure 15-17. Comparison of ALE and ADV# Bus Cycles

Figure 15-18 and Figure 15-19 show sample circuits that use the address valid strobe mode. Figure 15-18 shows a simple 8-bit system with a single flash. It is configured for the address valid strobe mode. This system configuration uses the ADV# signal as both the flash chip-select signal and the address-latch signal.

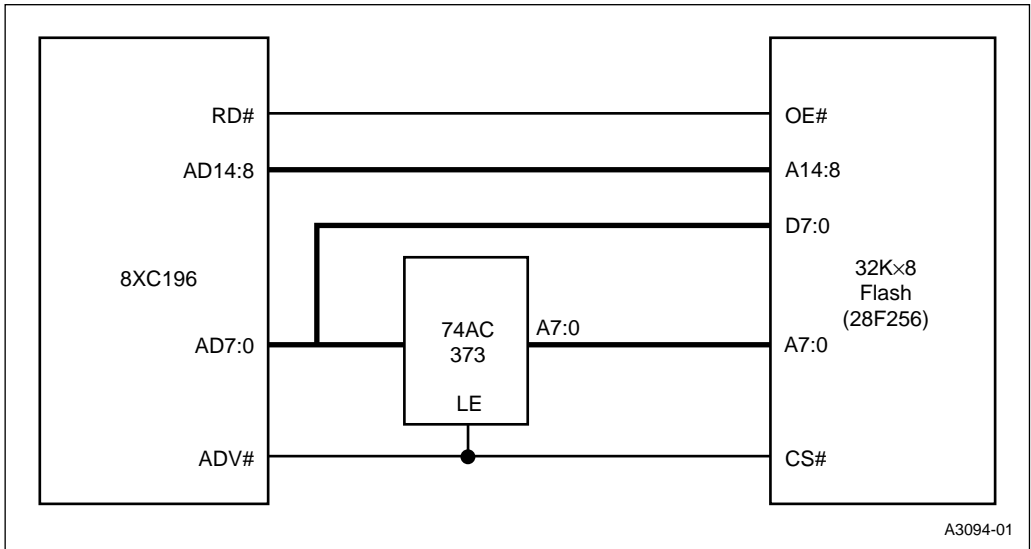


Figure 15-18. 8-bit System with Flash

Figure 15-19 shows a 16-bit system with two EPROMs. This system configuration uses the ADV# signal as both the EPROM chip-select signal and the address-latch signal.

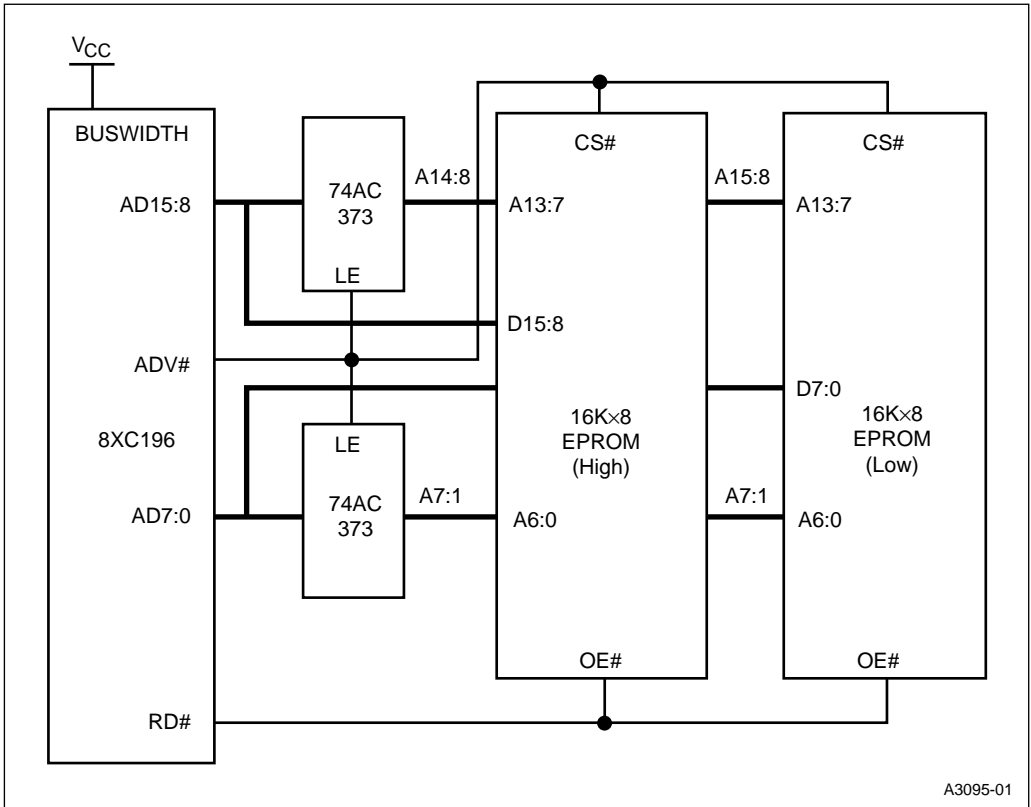


Figure 15-19. 16-bit System with EPROM

15.5.4 Address Valid with Write Strobe Mode

When the address valid with write strobe mode is selected, the microcontroller generates the ADV#, RD#, WRL#, and WRH# bus-control signals. This mode is used for a simple system using an external 16-bit data bus. Figure 15-20 shows the timing. The RD# signal (not shown) is similar to WRL# and WRH#. The example system of Figure 15-21 uses address valid with write strobe to access byte-wide RAMs with a 16-bit data bus.

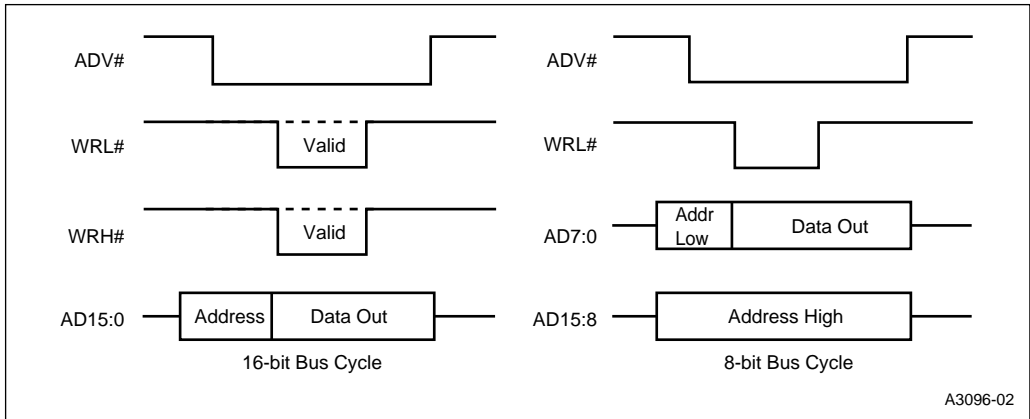


Figure 15-20. Timings of Address Valid with Write Strobe Mode

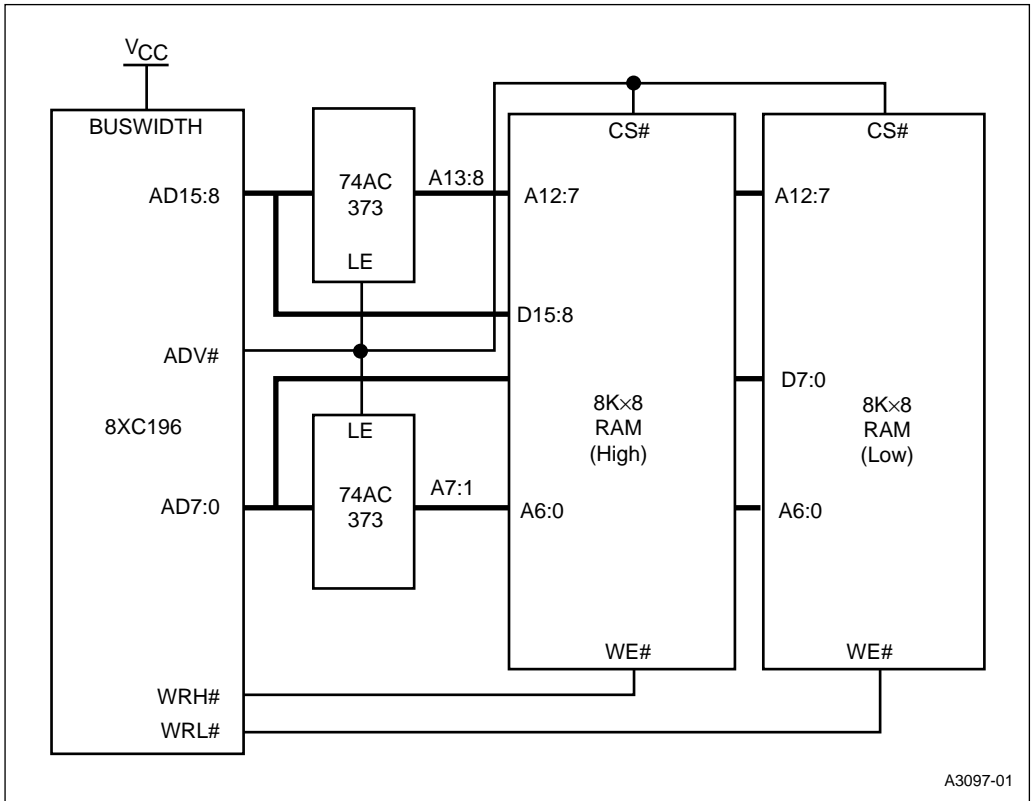


Figure 15-21. 16-bit System with RAM

15.6 SYSTEM BUS AC TIMING SPECIFICATIONS

Refer to the latest datasheet for the AC timings to make sure your system meets specifications. The major external bus timing specifications are shown in Figure 15-22.

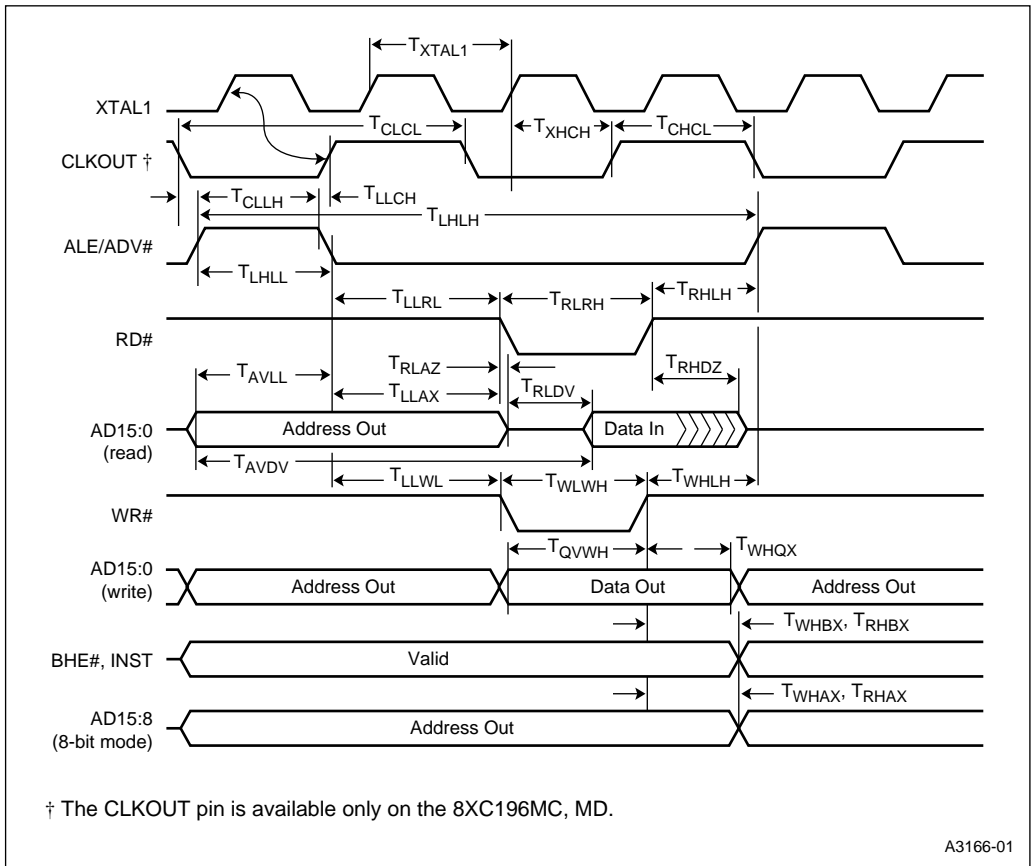


Figure 15-22. System Bus Timing

15.6.1 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/ADV#) condition L (Low), and signal R (RD#) condition L (Low). Table 15-7 defines the signal and condition codes.

Table 15-7. AC Timing Symbol Definitions

Signals		Conditions	
A	Address	Q	Output Data
B	BHE#	R	RD#
C [†]	CLKOUT	W	WR#, WRH#, WRL#
D	Input Data	X	XTAL1
G	BUSWIDTH	Y	READY
L	ALE/ADV#		

[†] The CLKOUT pin is available only on the 8XC196MC, MD microcontrollers.

15.6.2 AC Timing Definitions

Tables 15-8 and 15-9 define the AC timing specifications that the memory system must meet and those that the microcontroller will provide.

Table 15-8. External Memory Systems Must Meet These Specifications

Symbol	Definition
T_{AVDV}	Address Valid to Input Data Valid Maximum time the memory system has to output valid data after the microcontroller outputs a valid address.
T_{RHDZ}	RD# High to Input Data Float Time after the microcontroller deasserts RD# until the memory system must float the bus. If this timing is not met, bus contention will occur.
T_{RLDV}	RD# Low to Input Data Valid Maximum time the memory system has to output valid data after the microcontroller asserts RD#.

[†] The CLKOUT pin is available only on the 8XC196MC, MD microcontrollers.

Table 15-9. Microcontroller Meets These Specifications

Symbol	Definition
T_{AVLL}	Address Setup to ALE/ADV# Low Length of time address is valid before ALE/ADV# falls. Useful when using an external latch to demultiplex the address from the address data bus.
T_{CHCL}^{\dagger}	CLKOUT High CLKOUT Low CLKOUT pulse width. Useful when using CLKOUT to clock external devices.
T_{CLCL}^{\dagger}	CLKOUT Cycle Time The period of the CLKOUT signal; equal to $2T_{XTAL1}$.
T_{CLLH}^{\dagger}	CLKOUT Low to ALE/ADV# High
T_{LHLH}	ALE Cycle Time Minimum time between two ALE rising edges.
T_{LHLL}	ALE High to ALE Low ALE pulse width. Useful when using an external latch to demultiplex the address from the address data bus.
T_{LLAX}	AD15:0 Hold after ALE/ADV# Low Length of time address is valid after ALE/ADV# falls. Useful when using an external latch to demultiplex the address from the address data bus.
T_{LLCH}^{\dagger}	ALE/ADV# Low to CLKOUT High
T_{LLRL}	ALE/ADV# Low to RD# Low Length of time after ALE/ADV# falls before the microcontroller asserts RD#. Maximum time a memory system has to decode the address before the microcontroller asserts RD#.
T_{LLWL}	ALE/ADV# Low to WR# Low Length of time after ALE/ADV# falls before the microcontroller asserts WR#. Maximum time a memory system has to decode the address before the microcontroller asserts WR#.
T_{QVWH}	Output Data Valid to WR# High Length of time output data is valid before the microcontroller deasserts WR#.
T_{RHAX}	Address (high byte) Hold after RD# High Minimum time the high byte of the address (when using an 8-bit data bus) is valid after the microcontroller deasserts RD#.
T_{RHBX}	BHE#, INST Hold after RD# High Minimum time these signals are valid after the microcontroller deasserts RD#.
T_{RHLH}	RD# High to ALE High Time between the microcontroller deasserting RD# and the next ALE pulse. Maximum time the memory system has to put data on the bus before the next address cycle.
T_{RLAZ}	RD# Low to Address Float Time after the microcontroller deasserts RD# until it stops driving the address on the bus.
T_{RLRH}	RD# Low to RD# High RD# pulse width.

[†] The CLKOUT pin is available only on the 8XC196MC, MD microcontrollers.

Table 15-9. Microcontroller Meets These Specifications (Continued)

Symbol	Definition
T_{WHAX}	Address (high byte) Hold after WR# High Minimum time the high byte of the address (when using an 8-bit data bus) is valid after the microcontroller deasserts WR#.
T_{WHBX}	BHE#, INST Hold after WR# High Minimum time these signals are valid after the microcontroller deasserts WR#.
T_{WHLH}	WR# High to ALE High Time between the microcontroller deasserting WR# and next ALE pulse. Maximum time the memory system has to get data off the bus before the next address cycle.
T_{WHQX}	Output Data Hold after WR# High Length of time output data is valid on the bus after the microcontroller deasserts WR#.
T_{WLWH}	WR# Low to WR# High WR# pulse width.
T_{XHCH}^{\dagger}	XTAL1 High to CLKOUT High or Low
T_{XTAL1}	$1/F_{XTAL1}$ The period of the frequency on the XTAL1 input (F_{XTAL1}). All AC timings are referenced to T_{XTAL1} .

[†] The CLKOUT pin is available only on the 8XC196MC, MD microcontrollers.



16

Programming the Nonvolatile Memory

CHAPTER 16

PROGRAMMING THE NONVOLATILE MEMORY

The 87C196MC and 87C196MD contain 16 Kbytes of one-time-programmable read-only memory (OTPROM); the 87C196MH contains 32 Kbytes. OTPROM is similar to EPROM, but it comes in an unwindowed package and cannot be erased. You can either program the OTPROM yourself or have the factory program it as a quick-turn ROM product (this option may not be available for all devices). This chapter provides procedures and guidelines to help you program the device. The information is organized as follows.

- overview of programming methods
- OTPROM memory map (page 16-2)
- security features (page 16-3)
- programming pulse width (page 16-8)
- modified quick-pulse algorithm (page 16-9)
- programming mode pins (page 16-11)
- entering programming modes (page 16-13)
- slave programming (page 16-15)
- auto programming (page 16-25)
- PCCB and UPROM programming (8XC196MH only; page 16-30)
- run-time programming (page 16-32)

16.1 PROGRAMMING METHODS

You can program the OTPROM by configuring a circuit that allows the device to enter a programming mode. In programming modes, the device executes an algorithm that resides in the internal test ROM.

- Slave programming mode allows you to use an EPROM programmer as a master to program several microcontrollers (the slaves). The code and data to be programmed into the nonvolatile memory typically resides on a diskette. The EPROM programmer transfers the code and data from the diskette to its memory, then manipulates the slave's pins to define the addresses to be programmed and the contents to be written to those addresses. Using this

mode, you can program and verify single or multiple words in the OTPROM. This mode allows you to read the signature word and programming voltages and to program the PCCBs and unerasable PROM (UPROM) bits. Programming vendors and Intel distributors typically use this mode to program a large number of microcontrollers with a customer's code and data.

- Auto programming mode enables the microcontroller to act as a master to program itself with code and data that reside in an external memory device. Using this mode, you can program the entire OTPROM array except the UPROM bits and PCCBs. (For the 8XC196MH, PCCB and UPROM modes allow you to program those locations. For the 8XC196MC and 8XC196MD, only slave mode allows you to program them.) After programming, you can use the ROM-dump mode to write the entire OTPROM array to an external memory device to verify its contents. Customers typically use this low-cost method to program a small number of microcontrollers after development and testing are complete.

You can also program individual OTPROM locations without entering a programming mode. With this method, called run-time programming, your software controls the number and duration of programming pulses. Customers typically use this mode to download small sections of code to the microcontroller during software development and testing.

16.2 OTPROM MEMORY MAP

The OTPROM contains customer-specified special-purpose and program memory (Table 16-1). The 128-byte special-purpose memory partition is used for interrupt vectors, the chip configuration bytes (CCBs), and the security key. Several locations are reserved for testing or for use in future products. Write the value (20H or FFH) indicated in Table 16-1 to each reserved location. The remainder of the OTPROM is available for code storage.

Table 16-1. 87C196Mx OTPROM Memory Map

Address Range (Hex)	Description
9FFF (MH) 2080	Program memory
5FFF (MC, MD) 2080	Program memory
207F 205E	Reserved (each location must contain FFH)
205D 2040	PTS vectors
203F 2030	Upper interrupt vectors
202F 2020	Security key
201F 201C	Reserved (each location must contain FFH)
201B	Reserved (must contain 20H)
201A	CCB1
2019	Reserved (must contain 20H)
2018	CCB0
2017 2014	Reserved (each location must contain FFH)
2013 2000	Lower interrupt vectors

16.3 SECURITY FEATURES

Several security features enable you to control access to both internal and external memory. Read and write protection bits in the chip configuration register (CCR0), combined with a security key, allow various levels of internal memory protection. Two UPROM bits disable fetches of instructions and data from external memory. (See Figure 16-1 on page 16-7 for more information.)

16.3.1 Controlling Access to Internal Memory

The lock bits in the chip configuration register (CCR0) control access to the OTPROM. The reset sequence loads the CCRs from the CCBs for normal operation and from the PCCBs when entering programming modes. You can program the CCBs using any of the programming methods, but only slave and PCCB programming modes allow you to program the PCCBs.

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.

16.3.1.1 Controlling Access to the OTPROM During Normal Operation

During normal operation, the lock bits in CCB0 control read and write accesses to the OTPROM. Table 16-2 describes the options. You can program the CCBs using any of the programming methods.

Table 16-2. Memory Protection for Normal Operating Mode

Read Protect LOC1 (CCR0.7)	Write Protect LOC0 (CCR0.6)	Protection Status
1	1	No protection. Run-time programming is permitted, and the entire OTPROM array can be read.
1	0	Write protection only. Run-time programming is disabled, but the entire OTPROM array can be read.
0	1	Read protection. Run-time programming is disabled. If program execution is external, only the interrupt vectors and CCBs can be read. The security key is write protected.
0	0	Read and write protection. Run-time programming is disabled. If program execution is external, only the interrupt vectors and CCBs can be read.

Clearing CCB0.6 enables write protection. With write protection enabled, a write attempt causes the bus controller to cycle through the write sequence, but it does not enable V_{pp} or write data to the OTPROM. This protects the entire OTPROM array from inadvertent or unauthorized programming.

Clearing CCB0.7 enables read protection and also **write** protects the security key to protect it from being overwritten. With read protection enabled, the bus controller will not read from protected areas of OTPROM. An attempt to load the slave program counter with an external address causes the device to reset itself. Because the slave program counter can be as much as four bytes ahead of the CPU program counter, the bus controller might prevent code execution from the last four bytes of internal memory. The interrupt vectors and CCBs are **not** read protected because interrupts can occur even when executing from external memory.

16.3.1.2 Controlling Access to the OTPROM During Programming Modes

For programming modes, three levels of protection are available:

- prohibit all programming
- prohibit all programming, but permit authorized ROM dumps
- permit authorized ROM dumps, auto programming, and slave programming

These protection levels are provided by the PCCB0 lock bits, the CCB0 lock bits, and the internal security key (Table 16-3). When entering programming modes, the reset sequence loads the PCCBs into the chip configuration registers. It also loads CCB0 into internal RAM to provide an additional level of security.

You can program the CCBs using any of the programming methods, but only slave and PCCB programming modes permits access to the PCCBs, and only slave and auto programming allow you to program the internal security key.

Table 16-3. Memory Protection Options for Programming Modes

LOC1 (CCR0.7)		LOC0 (CCR0.6)		Security Key Programmed ?	Protection Status
PCCB	CCB	PCCB	CCB		
1	1	1	1	No	No protection. All programming modes allowed.
1	X	0	X	Yes	All programming disabled. ROM-dump permitted with matching security key.
1	0	1	0	Yes	Auto and slave programming permitted with matching security key.
0	X	0	X	X	All programming unconditionally disabled.

If you want to prohibit all programming, clear both PCCB0 lock bits. If these bits are cleared, they prevent the device from entering any programming mode.

If you want to prevent programming, but allow ROM dumps, leave the PCCB0 read-protection bit (PCCB0.7) unprogrammed and clear the PCCB0 write-protection bit (PCCB0.6). To protect against unauthorized reads, program an internal security key. The ROM-dump mode compares the internal security key location with an externally supplied security key, regardless of the CCB0 lock bits. If the security keys match, the routine continues; otherwise, the device enters an endless internal loop.

If you want to allow slave and auto programming as well as ROM dumps, leave both PCCB0 lock bits unprogrammed. To protect against unauthorized programming, clear the CCB0 lock bits and program an internal security key. After the device enters either slave or auto programming mode, the corresponding test ROM routine reads the CCB0 lock bits. If either CCB0 lock bit is enabled, the routine compares the internal security key location with an externally supplied security key. If the security keys match, the routine continues; otherwise, the device enters an endless internal loop.

You can program the internal security key in either auto or slave programming mode. Once the security key is programmed, you must provide a matching key to gain access to any programming mode. For auto programming and ROM-dump modes, a matching security key must reside in external memory. For slave programming mode, you must “program” a matching security key into the appropriate OTPROM locations with the program word command. The locations are not actually programmed, but the data is compared to the internal security key.

WARNING

If you leave the internal security key locations unprogrammed (filled with FFFFH), an unauthorized person could gain access to the OTPROM by using an external EPROM with an unprogrammed external security key location or by using slave programming mode.

16.3.2 Controlling Fetches from External Memory

Two UPROM bits disable external instruction fetches and external data fetches. If you program the UPROM bits, an attempt to fetch data or instructions from external memory causes a device reset. You can program the UPROM bits using slave or UPROM (8XC196MH only) programming mode.

Programming the DEI bit prevents the bus controller from executing external instruction fetches. An attempt to load the slave program counter with an external address causes the device to reset itself. Because the slave program counter can be as much as four bytes ahead of the CPU program counter, the bus controller might prevent code execution from the last four bytes of internal memory. The automatic reset also gives extra protection against runaway code.

Programming the DED bit prevents the bus controller from executing external data reads and writes. An attempt to access data through the bus controller causes the device to reset itself. Setting this bit disables ROM-dump mode.

To program these bits, write the correct value to the location shown in Table 16-4 using slave programming mode. During normal operation, you can determine the values of these bits by reading the UPROM special-function register (Figure 16-1).

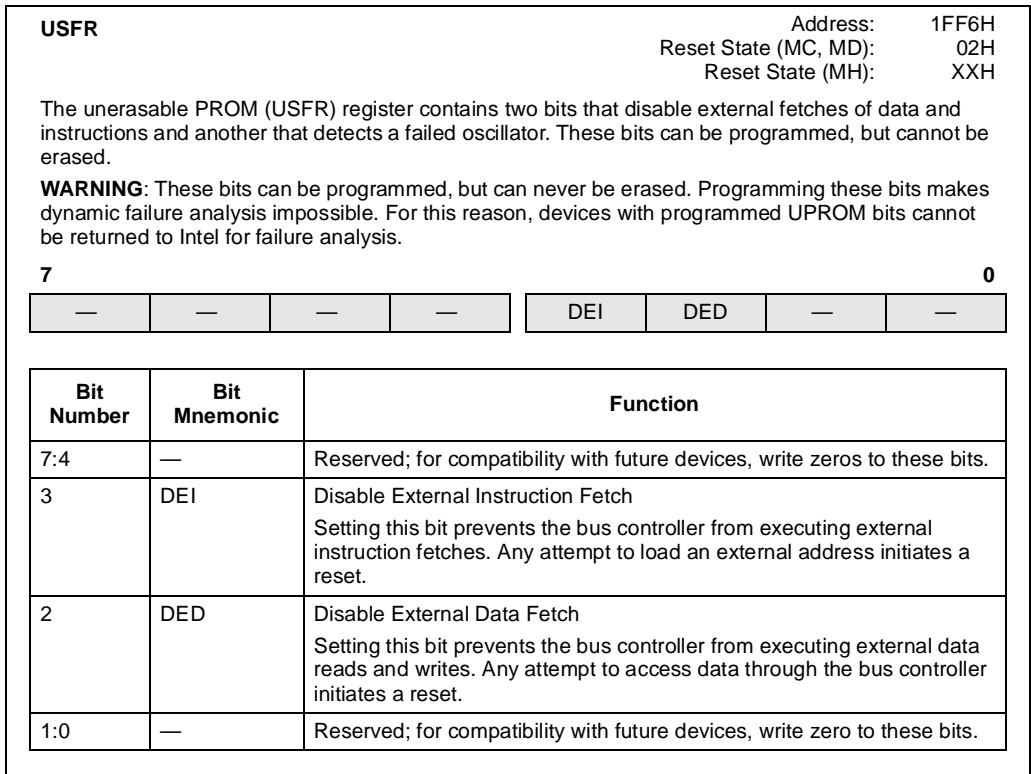


Figure 16-1. Unerasable PROM (USFR) Register

You can verify a UPROM bit to make sure it programmed, but you cannot erase it. For this reason, Intel cannot test the bits before shipment. However, Intel does test the features that the UPROM bits enable, so the only undetectable defects are (unlikely) defects within the UPROM cells themselves.

Table 16-4. UPROM Programming Values and Locations for Slave Mode

To set this bit	Write this value	To this location
DEI	08H	0718H
DED	04H	0758H

16.4 PROGRAMMING PULSE WIDTH

The programming pulse width is controlled in different ways, depending on the programming mode. In slave programming mode, the pulse width is controlled by the PALE# signal. In auto programming mode, it is loaded from the external EPROM into the PPW register. In the UPROM (8XC196MH only) and PCCB programming modes, the pulse width is controlled by the test-ROM routine. (For run-time programming, your software controls the pulse width.)

To determine the correct PPW_VALUE for the frequency of the device, use the following formula and round the result to the next higher integer.

$$\begin{aligned} \text{PPW_VALUE for 8XC196MC, MD} &= 62.5 \times F_{\text{XTAL1}} \\ \text{PPW_VALUE for 8XC196MH} &= 25 \times F_{\text{XTAL1}} \end{aligned}$$

where:

- PPW_VALUE is a 16-bit word
- F_{XTAL1} is the input frequency on XTAL1, in MHz

PPW								no direct access
<p>The programming pulse width (PPW) register is loaded from the external EPROM (locations 14H and 15H for the 8XC196MC and MD; locations 4014H and 4015H for the 8XC196MH) in auto programming mode. The PPW_VALUE determines the programming pulse width.</p>								
15								8
7	PPW15	PPW14	PPW13	PPW12	PPW11	PPW10	PPW9	PPW8
0	PPW7	PPW6	PPW5	PPW4	PPW3	PPW2	PPW1	PPW0

Bit Number	Bit Mnemonic	Function
15:0	PPW15:0	<p>PPW_VALUE</p> <p>This value establishes the programming pulse width for auto programming. Use the appropriate formula to calculate the PPW_VALUE, then write the result to the PPW register. (Table 16-5 shows the calculations and results for 8 MHz and 16 MHz operation.)</p> <p style="text-align: right;"> $\text{PPW_VALUE for 8XC196MC, MD} = 62.5 \times F_{\text{XTAL1}}$ $\text{PPW_VALUE for 8XC196MH} = 25 \times F_{\text{XTAL1}}$ </p>

Figure 16-2. Programming Pulse Width (PPW) Register

The examples in Table 16-5 calculate the required minimum pulse width (100 μs for the 8XC196MH and 250 μs for the 8XC196MC, MD) for an 8-MHz and a 16-MHz crystal.

Table 16-5. Example PPW_VALUE Calculations

F_{XTAL1}	8XC196MC, MD (Two 250-μs pulses required)	8XC196MH (Five 100-μs pulses required)
8 MHz	PPW_VALUE = $62.5 \times 8 = 504 = 01F4H$	PPW_VALUE = $25 \times 8 = 200 = 00C8H$
16 MHz	PPW_VALUE = $62.5 \times 16 = 1000 = 03F8H$	PPW_VALUE = $25 \times 16 = 400 = 0190H$

16.5 MODIFIED QUICK-PULSE ALGORITHM

Both the slave and auto programming routines use the modified quick-pulse algorithm (Figure 16-3). The modified quick-pulse algorithm sends programming pulses to each OTPROM word location. After the required number of programming pulses, a verification routine compares the contents of the programmed location to the input data. A verification error deasserts the PVER signal, but does not stop the programming routine. This process repeats until each OTPROM word has been programmed and verified. Intel guarantees lifetime data retention for a device programmed with the modified quick-pulse algorithm.

NOTE

The 87C196MC, MD devices use two pulses; the 87C196MH uses five.

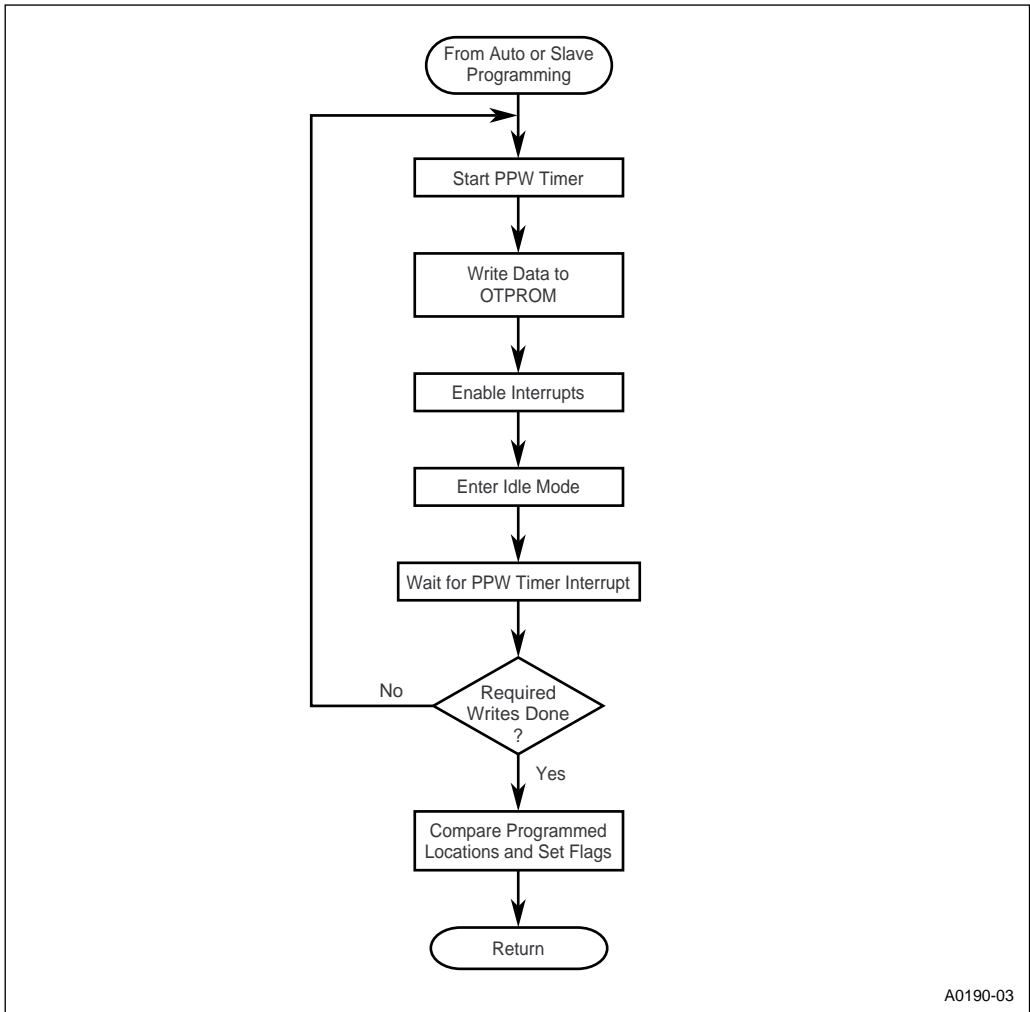


Figure 16-3. Modified Quick-pulse Algorithm

Auto programming repeats the pulse twice (for the 87C196MC, MD) or five times (for the 87C196MH), using the pulse width you specify in the external EPROM. Slave mode repeats the pulse until PROG# is deasserted. In slave programming mode, the PALE# signal controls the pulse width. In all cases, the pulse width must be at least 100 μ s for successful programming.

16.6 PROGRAMMING MODE PINS

Figure 16-4 illustrates the signals used in programming and Table 16-6 describes them. The EA#, V_{PP}, and PMODE pins combine to control entry into programming modes. You must configure the PMODE (P0.7:4) pins to select the desired programming mode (see Table 16-7 on page 16-13). Each programming routine configures the port 2 pins to operate as the appropriate special-function signals. Ports 3 and 4 automatically serve as the PBUS during programming.

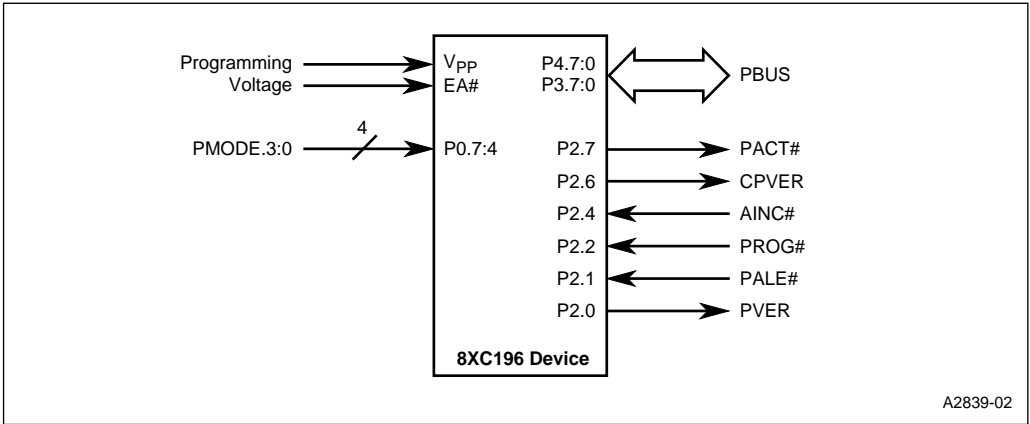


Figure 16-4. Pin Functions in Programming Modes

Table 16-6. Pin Descriptions

Port Pin	Special-function Signal	Type	Programming Mode	Description
P0.7:4	PMODE.3: PMODE.0	I	All	Programming Mode Select Determines the programming mode. PMODE is sampled after a device reset and must be static while the part is operating. (Table 16-7 on page 16-13 lists the PMODE values and programming modes.)
P2.0	PVER	O	Slave Auto	Programming Verification During slave or auto programming, PVER is updated after each programming pulse. A high output signal indicates successful programming of a location, while a low signal indicates a detected error.
P2.1	PALE#	I	Slave	Programming ALE Input During slave programming, a falling edge causes the device to read a command and address from the PBUS.

Table 16-6. Pin Descriptions (Continued)

Port Pin	Special-function Signal	Type	Programming Mode	Description
P2.2	PROG#	I	Slave	<p>Programming</p> <p>During programming, a falling edge latches data on the PBUS and begins programming, while a rising edge ends programming. The current location is programmed with the same data as long as PROG# remains asserted, so the data on the PBUS must remain stable while PROG# is active.</p> <p>During a word dump, a falling edge causes the contents of an OTPROM location to be output on the PBUS, while a rising edge ends the data transfer.</p>
P2.4	AINC#	I	Slave	<p>Auto-increment</p> <p>During slave programming, this active-low input enables the auto-increment feature. (Auto increment allows reading or writing of sequential OTPROM locations, without requiring address transactions across the PBUS for each read or write.) AINC# is sampled after each location is programmed or dumped. If AINC# is asserted, the address is incremented and the next data word is programmed or dumped.</p>
P2.6	CPVER	O	Slave	<p>Cumulative Program Verification</p> <p>During slave programming, a high signal indicates that all locations programmed correctly, while a low signal indicates that an error occurred during one of the programming operations.</p>
P2.7	PACT#	O	Auto ROM-dump	<p>Programming Active</p> <p>During auto programming or ROM-dump, a low signal indicates that programming or dumping is in progress, while a high signal indicates that the operation is complete.</p>
P4.7:0, P3.7:0	PBUS	I/O	Slave	<p>Address/Command/Data Bus</p> <p>During slave programming, ports 3 and 4 serve as a bidirectional port with open-drain outputs to pass commands, addresses, and data to or from the device. Slave programming requires external pull-up resistors.</p>
P4.7:0, (MC,MD) P3.7:0 (MC, MD) P1.3:0, (MH) P4.3:0, (MH) P3.7:0 (MH)	PBUS	I/O	Auto ROM-dump	<p>Address/Command/Data Bus</p> <p>During auto programming and ROM-dump, ports 3 and 4 serve as a regular system bus to access external memory. For the 8XC196MH, P4.7:4 are left unconnected; P1.3:0 serve as the upper address lines.</p>

Table 16-6. Pin Descriptions (Continued)

Port Pin	Special-function Signal	Type	Programming Mode	Description
—	EA#	I	All	<p>External Access</p> <p>Controls program mode entry. If EA# is at V_{PP} voltage on the rising edge of RESET#, the device enters programming mode.</p> <p>EA# is sampled and latched only on the rising edge of RESET#. Changing the level of EA# after reset has no effect.</p>
—	V_{PP}	I	All	<p>Programming Voltage</p> <p>During programming, the V_{PP} pin is typically at +12.5V (V_{PP} voltage). Exceeding the maximum V_{PP} voltage specification can damage the device.</p>

16.7 ENTERING PROGRAMMING MODES

To execute programs properly, the device 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 programming mode by applying V_{PP} voltage (+12.5 V) to EA# during the rising edge of RESET#.

16.7.1 Selecting the Programming Mode

The PMODE (P0.7:4) value controls the programming mode. PMODE is sampled on the rising edge of RESET#. You must reset the device to switch programming modes. Table 16-7 lists the PMODE value for each programming mode. All other PMODE values are reserved.

Table 16-7. PMODE Values

PMODE Value (Hex)	Programming Mode
5	Slave programming
6	ROM-dump
9	UPROM programming (MH only)
C	Auto programming
D	PCCB programming (MH only)

16.7.2 Power-up and Power-down Sequences

When you are ready to begin programming, follow these power-up and power-down procedures.

WARNING

Failure to observe these warnings will cause permanent device damage.

- Voltage must **not** be applied to V_{PP} while V_{CC} is low.
- The V_{PP} voltage must be within 1 volt of V_{CC} while V_{CC} is less than 4.5 volts. V_{PP} must not go above 4.5 volts until V_{CC} is at least 4.5 volts.
- The V_{PP} maximum voltage must **not** be exceeded.
- EA# must reach programming voltage before V_{PP} does so.
- The PMODE pins (P0.7:4) must be in their desired states before RESET# rises.
- All voltages must be within the ranges specified in the datasheet and the oscillator must be stable before RESET# rises.
- The power supplies to the V_{CC} , V_{PP} , EA# and RESET# pins must be well regulated and free of glitches and spikes.
- All V_{SS} pins must be well grounded.

16.7.2.1 Power-up Sequence

1. Hold RESET# low while V_{CC} stabilizes. Allow V_{PP} and EA# to float during this time.
2. After V_{CC} and the oscillator stabilize, continue to hold RESET# low and apply V_{PP} voltage to EA#.
3. After EA# stabilizes, apply V_{PP} voltage (+12.5V) to the V_{PP} pin.
4. Set the PMODE value to select a programming algorithm.
5. Bring the RESET# pin high.
6. Complete the selected programming algorithm.

16.7.2.2 Power-down Sequence

1. Assert the RESET# signal and hold it low throughout the powerdown sequence.
2. Remove the V_{PP} voltage from the V_{PP} pin and allow the pin to float.
3. Remove the V_{PP} voltage from the EA# pin and allow the pin to float.
4. Turn off the V_{CC} supply and allow time for it to reach 0 volts.

16.8 SLAVE PROGRAMMING MODE

Slave programming mode allows you to program and verify the entire OTPROM array, including the PCCBs and UPROM bits, by using an EPROM programmer.

In this mode, ports 3 and 4 serve as the PBUS, transferring commands, addresses, and data. The least-significant bit of the PBUS (P3.0) controls the command (1 = program word; 0 = dump word) and the remaining 15 bits contain the address of the word to be programmed or dumped. Some port 2 pins provide handshaking signals. The AINC# signal controls whether the address is automatically incremented, enabling programming or dumping sequential OTPROM locations. This speeds up the programming process, since it eliminates the need to generate and decode each sequential address.

NOTE

If a glitch or reset occurs during programming of the security key, an unknown security key might accidentally be written, rendering the device inaccessible for further programming. To prevent this possibility during slave programming, program the rest of the OTPROM array before you program the CCB security-lock bits (CCB0.6 and CCB0.7).

16.8.1 Reading the Signature Word and Programming Voltages

The signature word identifies the device; the programming voltages specify the V_{PP} and V_{CC} voltages required for programming. This information resides in the test ROM at locations 2070H, 2072H, and 2073H; however, these locations are remapped to 0070H, 0072H, and 0073H. You can use the dump word command in slave programming mode to read the signature word and programming voltages at the locations shown in Table 16-8. The external programmer can use this information to determine the device type and operating conditions. You should **never** write to these locations. The voltages are calculated by using the following equation (after converting the test ROM value to decimal).

$$\text{Voltage} = \frac{20 \times \text{test ROM value}}{256}$$

$$V_{CC} (40H) = \frac{20 \times 64}{256} = 5 \text{ volts}$$

$$V_{PP} (0A0H) = \frac{20 \times 160}{256} = 12.5 \text{ volts}$$

Table 16-8. Device Signature Word and Programming Voltages

Device	Signature Word		Programming V _{CC}		Programming V _{PP}	
	Location	Value	Location	Value	Location	Value
8XC196MC, MD	0070H	8794H	0072H	40H	0073H	0A0H
8XC196MH	0070H	87DEH	0072H	40H	0073H	0A0H

16.8.2 Slave Programming Circuit and Memory Map

Figure 16-5 shows the circuit diagram and Table 16-9 shows the memory map for slave programming mode. The external clock signal can be supplied by either a clock or a crystal. Refer to the device datasheet for acceptable clock frequencies.

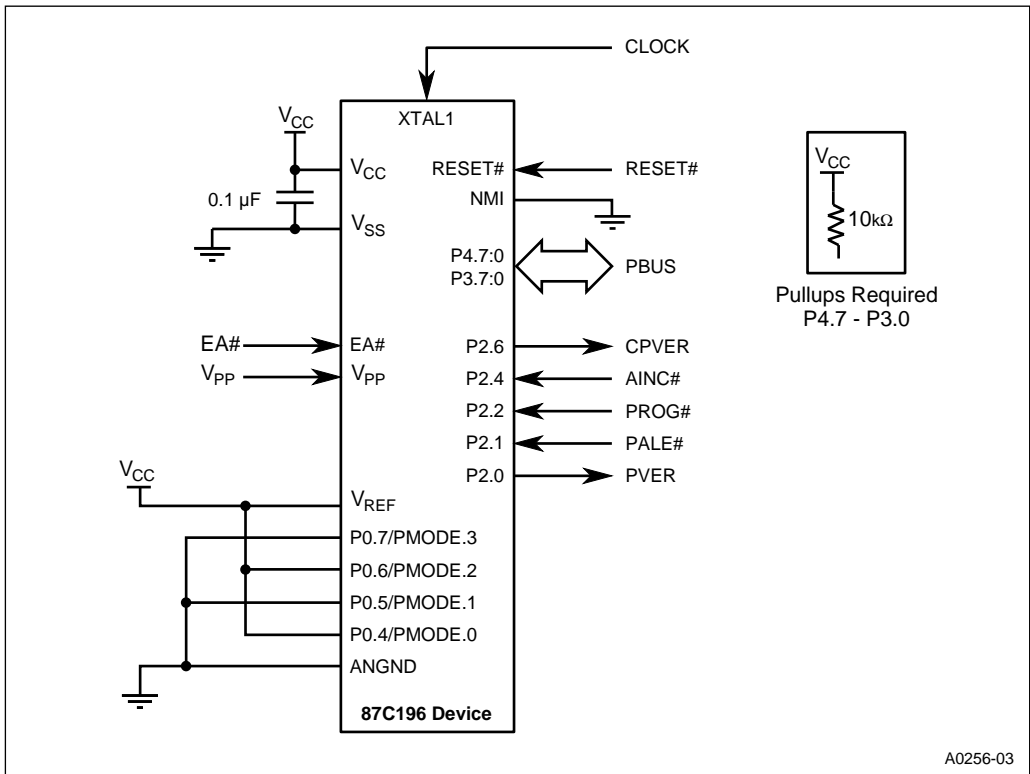


Figure 16-5. Slave Programming Circuit

Table 16-9. Slave Programming Mode Memory Map

Description	Address	Comments
OTPROM	(MH) 2000–9FFFH (MC, MD) 2000–5FFFH	OTPROM Cells
DED†	0758H	UPROM Cell
DEI†	0718H	UPROM Cell
PCCB	0218H	Test EPROM
Programming voltages (see Table 16-8 on page 16-16)	0072H, 0073H	Read Only
Signature word	0070H	Read Only

†These bits program the UPROM cells. Once these bits are programmed, they cannot be erased and dynamic failure analysis of the device is impossible.

16.8.3 Operating Environment

The chip configuration registers (CCRs) define the system environment. Since the programming environment is not necessarily the same as the application environment, the device provides a means for specifying different configurations. Specify your application environment in the chip configuration bytes (CCBs) located in the OTPROM. Specify your programming environment in the programming chip configuration bytes (PCCBs) located in the test ROM.

Figure 16-6 shows an abbreviated description of the CCRs with the default PCCB environment settings. The reset sequence loads the CCRs from the CCBs for normal operation and from the PCCBs when entering programming modes. You can program the CCBs using any of the programming methods, but only slave mode allows you to program the PCCBs. Chapter 15, “Interfacing with External Memory,” describes the system configuration options, and “Controlling Access to Internal Memory” on page 16-3 describes the memory protection options.

CCR1, CCR0		no direct access					
The chip configuration registers (CCRs) control wait states, powerdown mode, and internal memory protection. These registers are loaded from the PCCBs during programming modes and from the CCBs for normal operation.							
7		0					
1	1	0	1	WDE	BW1	IRC2	0
7		0					
LOC1	LOC0	IRC1	IRC0	ALE	WR	BW0	PD
Bit Mnemonic	Function						
WDE	Watchdog Timer Enable PCCB default is initially disabled (enabled the first time WDT is cleared).						
BW1	Buswidth Control PCCB default selects BUSWIDTH pin control.						
IRC2	Internal Ready Control PCCB default selects READY pin control.						
LOC1:0	Security Bits PCCB default selects no protection.						
IRC1:0	Internal Ready Control PCCB default selects READY pin control.						
ALE	Select Address Valid Strobe Mode PCCB default selects ALE.						
WR	Select Write Strobe Mode PCCB default selects WR# and BHE#.						
BW0	Buswidth Control PCCB default selects BUSWIDTH pin control.						
PD	Powerdown Enable PCCB default enables powerdown.						

Figure 16-6. Chip Configuration Registers (CCRs)

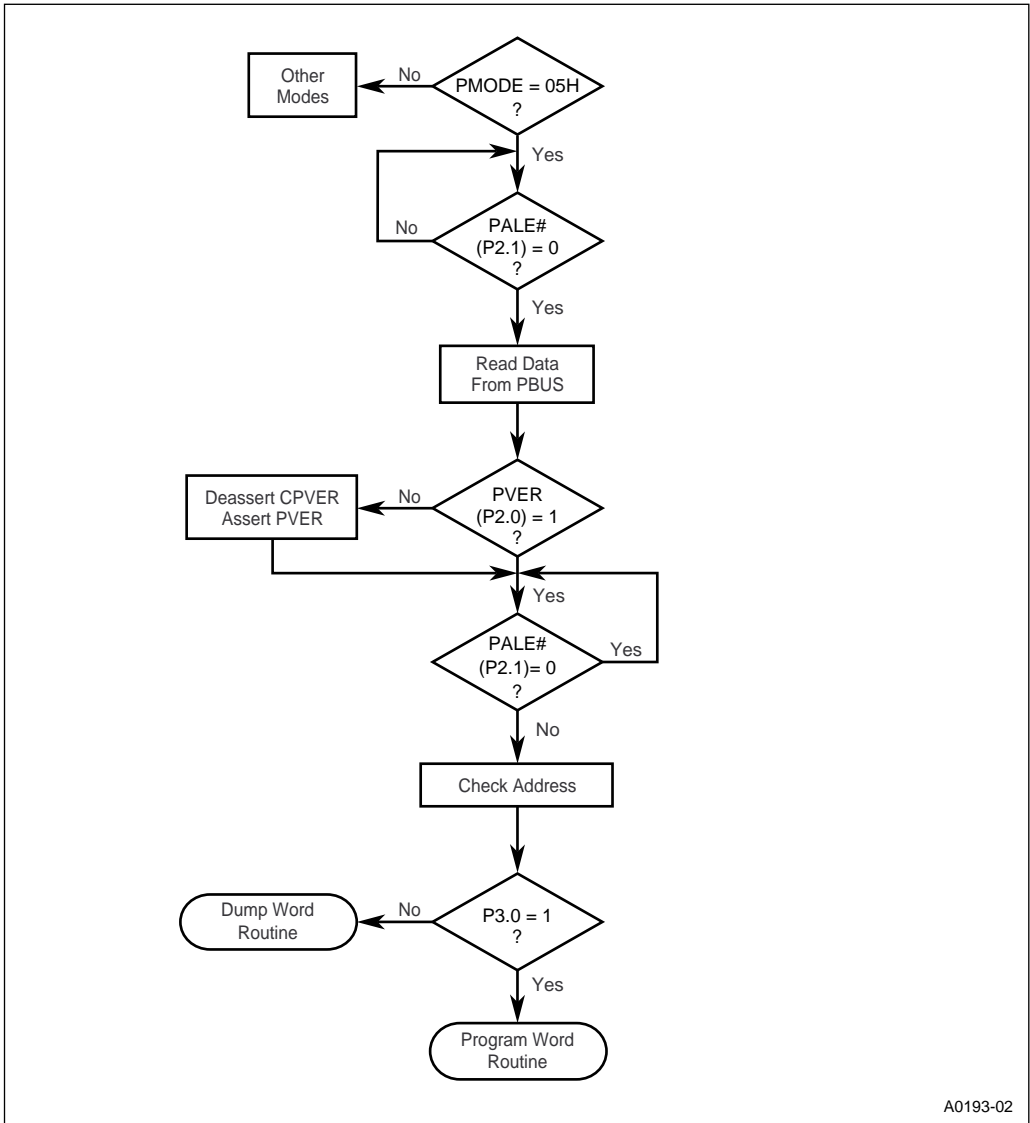
16.8.4 Slave Programming Routines

The slave programming mode algorithm consists of three routines: the address/command decoding routine, the program word routine, and the dump word routine.

The address/command decoding routine (Figure 16-7) reads the PBUS and transfers control to the program word or dump word routine based on the value of P3.0. A one on P3.0 selects the program word command and the remaining bits specify the address. For example, a PBUS value of 3501H programs a word of data at location 3500H. A zero on P3.0 selects the dump word command and the remaining bits specify the address. For example, a PBUS value of 3500H places the word at location 3500H on the PBUS.

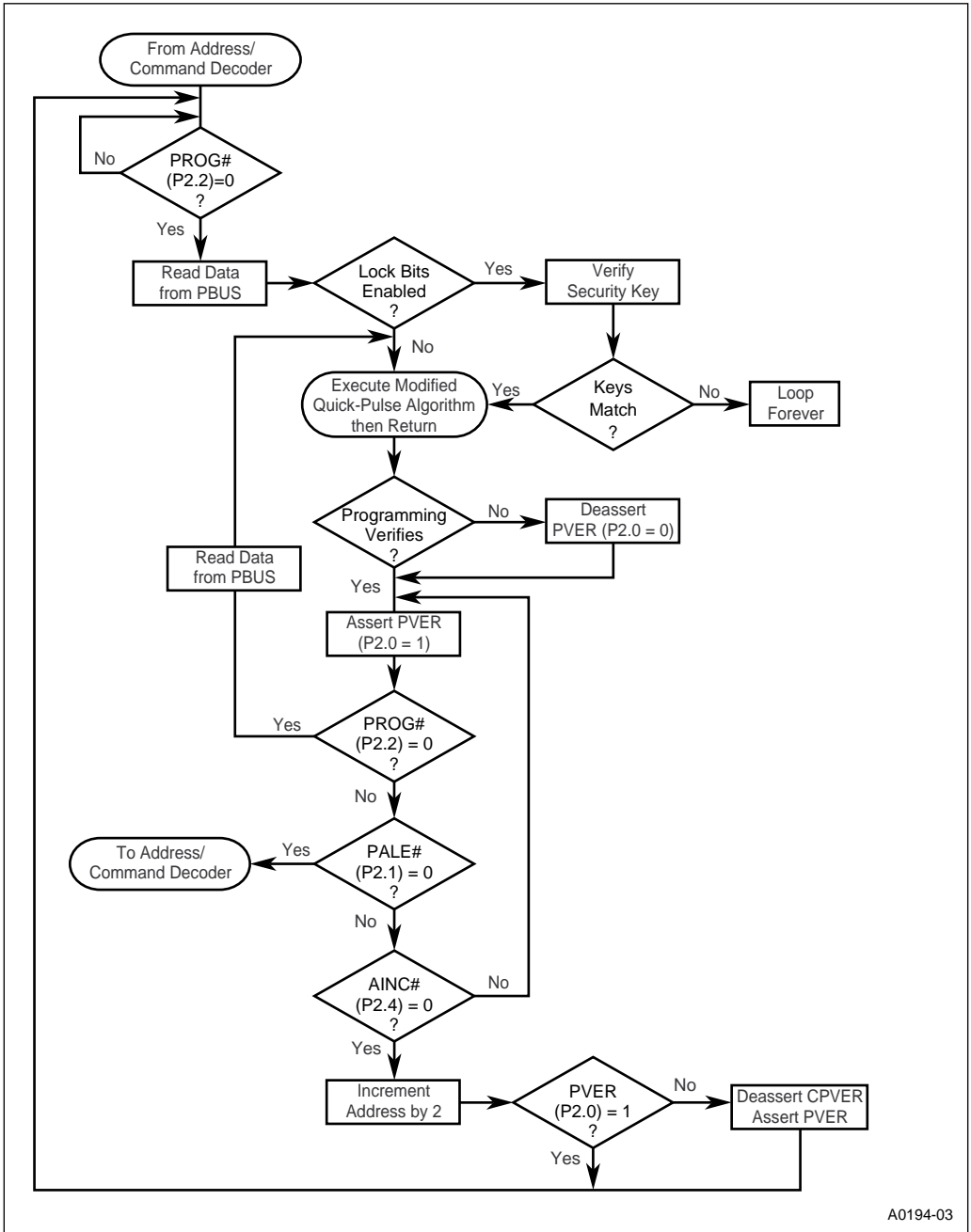
The program word routine (Figure 16-8) checks the CCB security-lock bits. If either security lock bit (CCB0.6 or CCB0.7) has been programmed, you must provide a matching security key to gain access to the device. Using the program word command, write eight consecutive words to the device, starting at location 2020H and continuing to 202FH. The routine stores these eight words in an internal register and compares their value with the internal key. If the keys match, the routine allows you to program individual or sequential OTPROM locations; otherwise, the device enters an endless loop.

The dump word routine (Figure 16-10) also checks the CCB security-lock bits, but it has no provision for security key verification. If the lock bits are unprogrammed, the routine fetches a word of data from the OTPROM and writes that data to the PBUS. If either lock bit is programmed, the routine performs a write cycle without first getting data from the OTPROM.



A0193-02

Figure 16-7. Address/Command Decoding Routine



A0194-03

Figure 16-8. Program Word Routine

Figure 16-9 shows the timings of the program word command with a repeated programming pulse and auto increment. Asserting PALE# latches the command and address on the PBUS. Asserting PROG# latches the data on the PBUS and starts the programming sequence. The PROG# signal controls the programming pulse width. (Slave programming mode does not use the PPW register.) After the rising edge of PROG#, the routine verifies the contents of the location that was just programmed and asserts PVER to indicate successful programming. AINC# is optional and can automatically increment the address for the next location. If you do not use AINC#, you must send a new program word command to access the next word location.

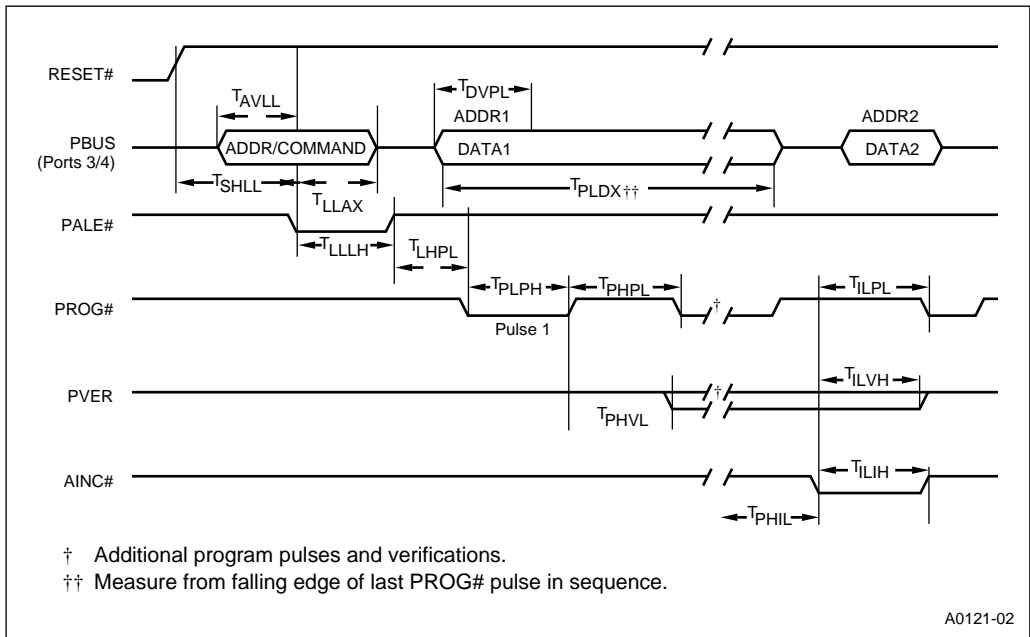
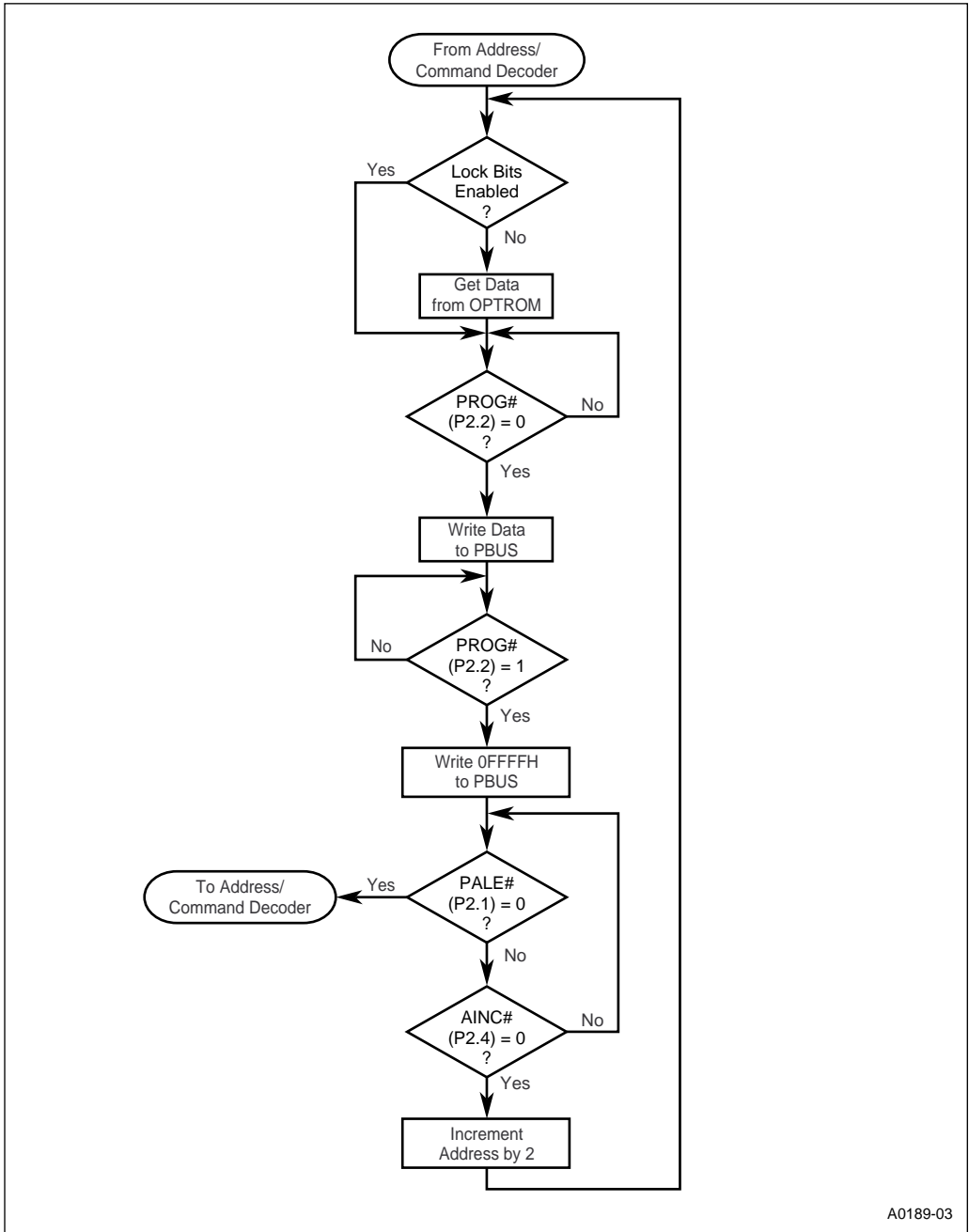


Figure 16-9. Program Word Waveform



A0189-03

Figure 16-10. Dump Word Routine

Figure 16-11 shows the timings of the dump word command. PROG# governs when the device drives the bus. The timings before the dump word command are the same as those shown in Figure 16-9. In the dump word mode, the AINC# pin can remain active and toggling. The PROG# pin automatically increments the address.

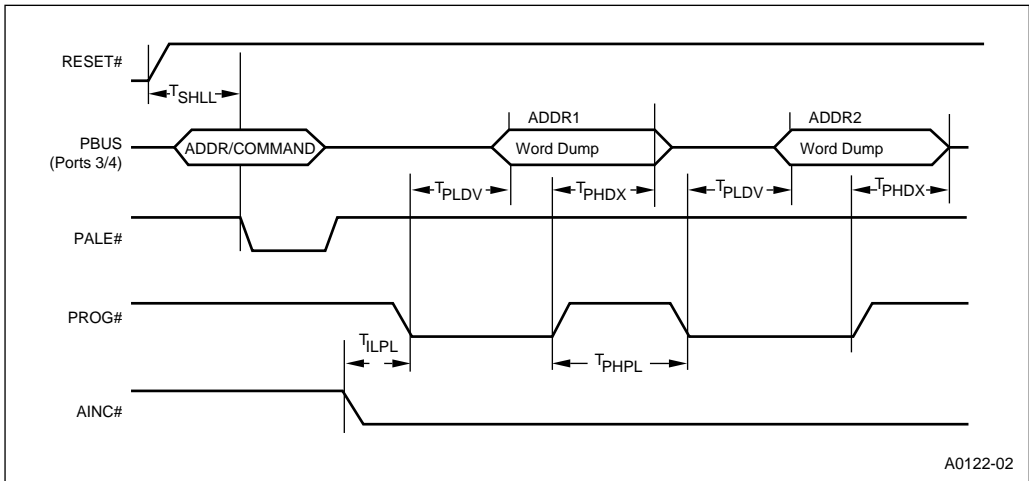


Figure 16-11. Dump Word Waveform

16.8.5 Timing Mnemonics

Table 16-10 defines the timing mnemonics used in the program word and dump word waveforms. The datasheets include timing specifications for these signals.

Table 16-10. Timing Mnemonics

Mnemonic	Description
T_{SHLL}	Reset High to First PALE# Low.
T_{LLLH}	PALE# Pulse Width.
T_{AVLL}	Address Setup Time.
T_{LLAX}	Address Hold Time.
T_{PLDV}	PROG# Low to Word Dump Valid.
T_{PHDX}	Word Dump Data Hold.
T_{DVPL}	Data Setup Time.
T_{PLDX}	Data Hold Time.
T_{PLPH}	PROG# Pulse Width.
T_{PHLL}	PROG# High to Next PALE# Low.
T_{LHPL}	PALE# High to PROG# Low.

Table 16-10. Timing Mnemonics (Continued)

Mnemonic	Description
T _{PHPL}	PROG# High to Next PROG# Low.
T _{PHIL}	PROG# High to AINC# Low.
T _{ILIH}	AINC# Pulse Width.
T _{ILVH}	PVER Hold After AINC# Low.
T _{ILPL}	AINC# Low to PROG# Low.
T _{PHVL}	PROG# High to PVER Valid.

16.9 AUTO PROGRAMMING MODE

The auto programming mode is a low-cost programming alternative. Using this programming mode, the device programs itself with data from an external EPROM (external locations 4000H and above; see Table 16-1 on page 16-3). A bank switching mechanism provided by port 1 pins (see Figure 16-12) supports auto programming of devices with more than 16 Kbytes of internal memory.

16.9.1 Auto Programming Circuit and Memory Map

Figure 16-12 shows the recommended circuit for auto programming mode. Table 16-11 shows the 8XC196MC/MD memory map and Table 16-11 shows the 8XC196MH auto programming memory map. Auto programming is specified for a crystal frequency of 6 to 8 MHz. At 8 MHz, use a 27(C)512 EPROM with t_{ACC} = 250 ns and t_{OE} = 100 ns or faster specifications.

Tie the BUSWIDTH pin low to configure an 8-bit data bus. Connect P1.3:0 (8XC196MH only) as shown to generate the high-order bits of the external EPROM address. Connect P0.7:4 to V_{SS} and V_{CC} to select auto programming (1100B = 0CH). PACT# and PVER are status outputs, buffered by the 74HC14s. They drive LEDs that indicate programming active (PACT#) and programming verification (PVER). Connect all unused inputs to ground (V_{SS}) and leave unused outputs floating. READY and NMI are active; connect them as indicated.

NOTE

All external EPROM addresses specified in this section are given for the circuit in Figure 16-12. If you choose a different circuit, you must adjust the addresses accordingly.

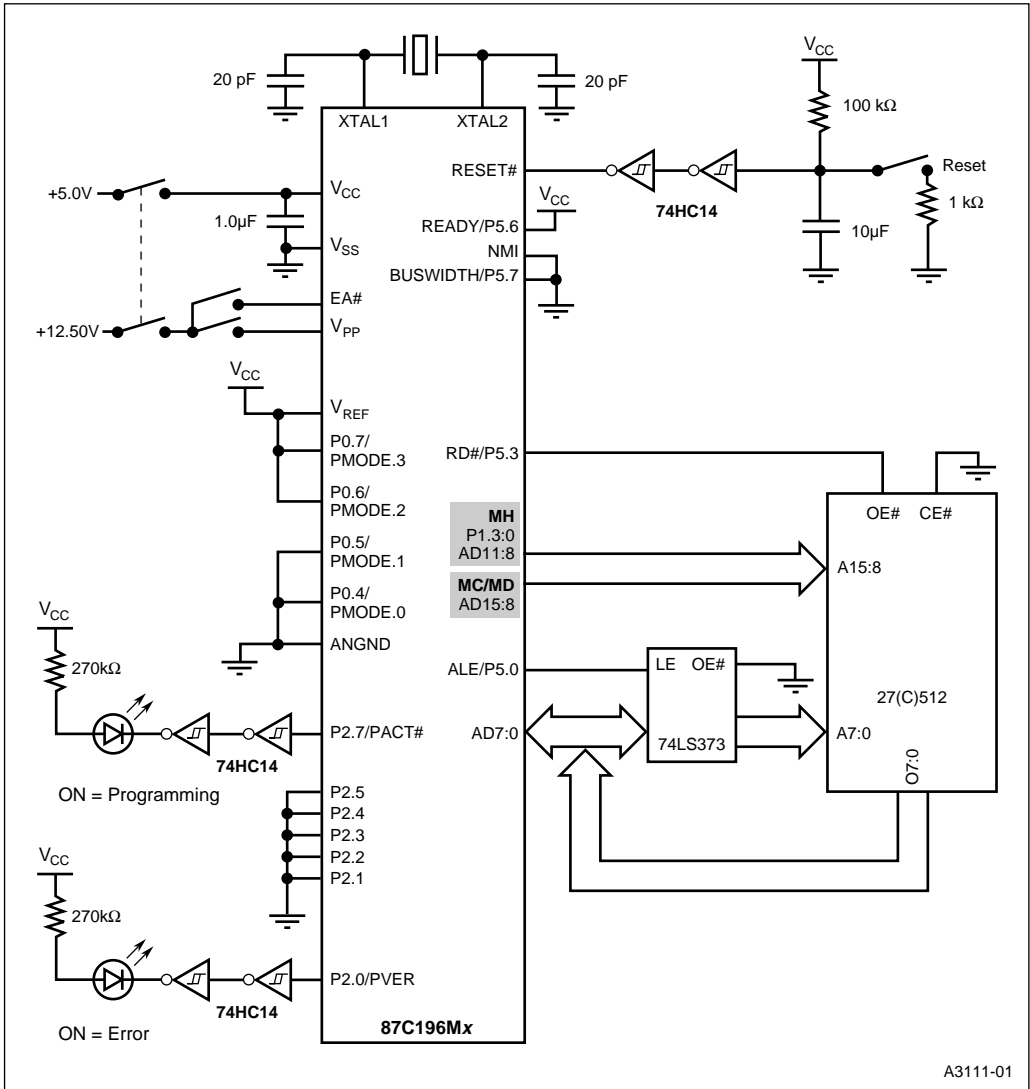


Figure 16-12. Auto Programming Circuit

Table 16-11. 8XC196MC/MD Auto Programming Memory Map

Address Output from 8XC196MC, 8XC196MD	Internal OTPROM Address	Address Using Circuit in Figure 16-12 (A15:0)	Description
4014H	N/A	14H	Programming pulse width (PPW) LSB.
4015H	N/A	15H	Programming pulse width (PPW) MSB.
4020–402FH	2020–202FH	0020–002FH	Security key for verification.
4000–7FFFH	2000–5FFFH	4000–7FFFH	Code, data, and reserved locations.

Table 16-12. 8XC196MH Auto Programming Memory Map

Address Output from 8XC196MH	Internal OTPROM Address	Address Using Circuit in Figure 16-12 (P1.3:0, A11:0)	Description
105EH	N/A	105EH	Programming pulse width (PPW) LSB.
105FH	N/A	105FH	Programming pulse width (PPW) MSB.
0020–002FH	2020–202FH	0020–002FH	Security key for verification.
2000–9FFFH	2000–9FFFH	2000–9FFFH	Code, data, and reserved locations.

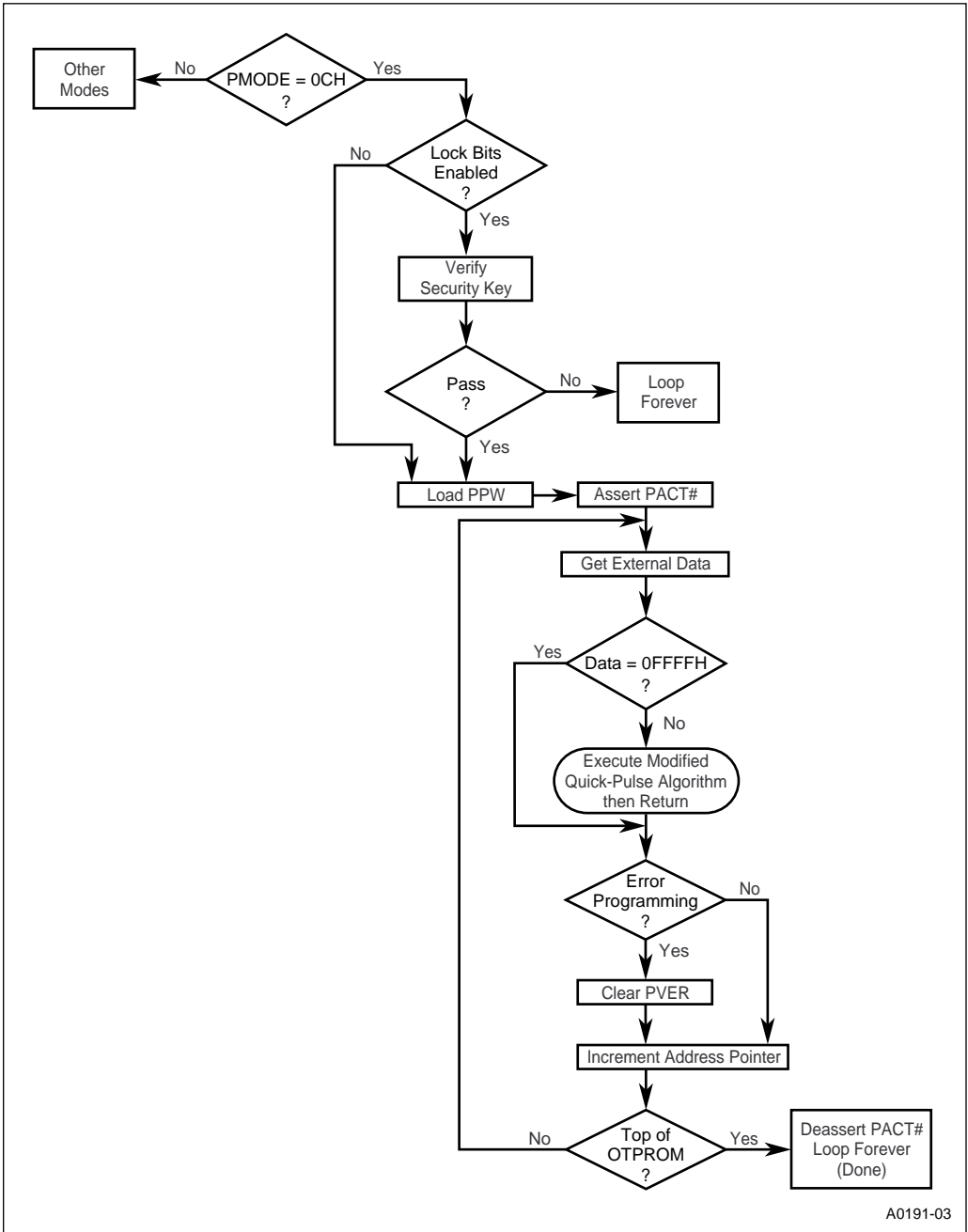
16.9.2 Operating Environment

In the auto programming mode, the PCCBs are loaded into the chip configuration registers. Since the device gets programming data through the external bus, the memory device in the programming system must correspond to the default configuration (Figure 16-6 on page 16-18). Auto programming requires an 8-bit bus configuration, so the circuit must tie the BUSWIDTH pin low. The PCCB defaults allow you to use any standard EPROM that satisfies the AC specifications listed in the device datasheet.

The auto programming mode also loads CCB0 into an internal RAM location and checks the lock bits. If either lock bit is programmed, the auto programming routine compares the internal security key to the external security key location. If the verification fails, the device enters an endless internal loop. If the security keys match, the routine continues. The auto programming routine uses the modified quick-pulse algorithm and the pulse width value programmed into the external EPROM.

16.9.3 Auto Programming Routine

Figure 16-13 illustrates the auto programming routine. This routine checks the security lock bits in CCB0; if either bit is programmed, it compares the internal security key to the external security key locations. If the security keys match, the routine continues; otherwise, the device enters an endless loop.



A0191-03

Figure 16-13. Auto Programming Routine

If the security key verification is successful, the routine loads the programming pulse width (PPW) value from the external EPROM into the internal PPW register. It then asserts PACT#, indicating that programming has begun. (PACT# is also active during reset, although no programming is in progress.) PVER is initially asserted and remains asserted unless an error is detected, in which case it is deasserted.

The routine then reads the contents of the external EPROM, beginning at 4000H (2000H for the 8XC196MH). It skips any word that contains FFFFH (unprogrammed state). When it reads a word that contains any value other than FFFFH, the routine calls the modified quick-pulse algorithm, which writes that value to the OTPROM, using the appropriate number of pulses for the device, then verifies the result. The routine repeats this activity until the entire OTPROM is programmed, then deasserts PACT# and enters an endless loop.

16.9.4 Auto Programming Procedure

If a glitch or reset occurs while programming the security key and lock bits, an unknown security key might accidentally be written, rendering the device inaccessible for further programming. To minimize this possibility, follow this recommended programming procedure.

NOTE

All addresses are given for the circuit shown in Figure 16-12 on page 16-26. If you choose a different circuit, you must adjust the addresses accordingly.

1. Using a blank EPROM device, follow these steps to skip programming of CCB0 and program the rest of the OTPROM array, including the security key.
 - Place the programming pulse width (PPW) in external EPROM locations 14H–15H (for the 8XC196MC, MD) or 105E–105FH (for the 8XC196MH).
 - Leave the external CCB0 location (4018H) unprogrammed (0FFFFH).
 - Place the appropriate CCB1 value at external location 401AH.
 - Place the security key to be programmed in external EPROM locations 4020H–402FH (for the 8XC196MC, MD) or 0020–002FH (for the 8XC196MH).
 - Place the value 20H in external EPROM locations 4019H and 401BH (for the reserved OTPROM locations that require this value).
 - Place the desired code in the remaining external EPROM locations 4000–7FFFH (for the 8XC196MC, MD) or 2000–9FFFH (for the 8XC196MH).
 - Execute the power-up sequence (page 16-14) to initiate auto programming.
 - When programming is complete, execute the powerdown sequence (page 16-14) before continuing to step 2.

2. Using another blank EPROM device, follow these steps to program only CCB0.
 - Place the programming pulse width (PPW) in external locations 14H–15H.
 - Place the appropriate CCB0 value in external location 4018H.
 - Place the security key to be verified in external EPROM locations 0020H–002FH. This value must match the security key programmed in step 1.
 - Leave the remaining EPROM locations unprogrammed (0FFFFH).
 - Execute the power-up sequence (page 16-14) to initiate auto programming.
 - When programming is complete, follow the powerdown sequence (page 16-14).

At this point, you can modify the circuit, then use ROM-dump mode to write the entire OTPROM array to an external memory device and verify its contents. (See “ROM-dump Mode” for details.)

16.9.5 ROM-dump Mode

The ROM-dump mode provides an easy way to verify the contents of the OTPROM array after auto programming. Use the same circuit as for auto programming, but change the connections of the PMODE (P0.7:4) pins. To select ROM-dump mode (PMODE=6H), connect P0.6 and P0.5 to V_{CC} and connect P0.7 and P0.4 to ground. The same bank switching mechanism is used and the memory map is the same as that for auto programming. The example circuit (Figure 16-12 on page 16-26) does not show the necessary $WR\#$ and V_{PP} connections to allow writing to the EPROM. And although the example uses an EPROM, you could also use a RAM device. Alternatively, you could dump the OTPROM contents to any 16-bit parallel port.

NOTE

If you have programmed the DED bit (USFR.2), ROM-dump mode is disabled. (See “Controlling Fetches from External Memory” on page 16-6).

To enter ROM-dump mode, follow the power-up sequence on page 16-14. The ROM-dump mode checks the security key, regardless of the CCR security-lock bits. 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, ROM-dump mode fetches the PPW, then writes the entire OTPROM array to external memory. $PACT\#$ remains low while the dump is in progress, then goes high to indicate that the dump is complete.

16.10 PCCB AND UPROM PROGRAMMING (8XC196MH ONLY)

The PCCB and UPROM programming modes are useful if you have auto programmed and verified the rest of the OTPROM array and now need to program only the PCCB and UPROM bits. (With slave programming mode, you can program the PCCB and UPROM bits along with the rest of the array.)

Figure 16-14 shows the recommended circuit for PCCB and UPROM programming. In these circuits, the PBUS holds data to be written to the OTPROM, PALE# begins programming, and PVER drives an LED that lights to indicate successful programming.

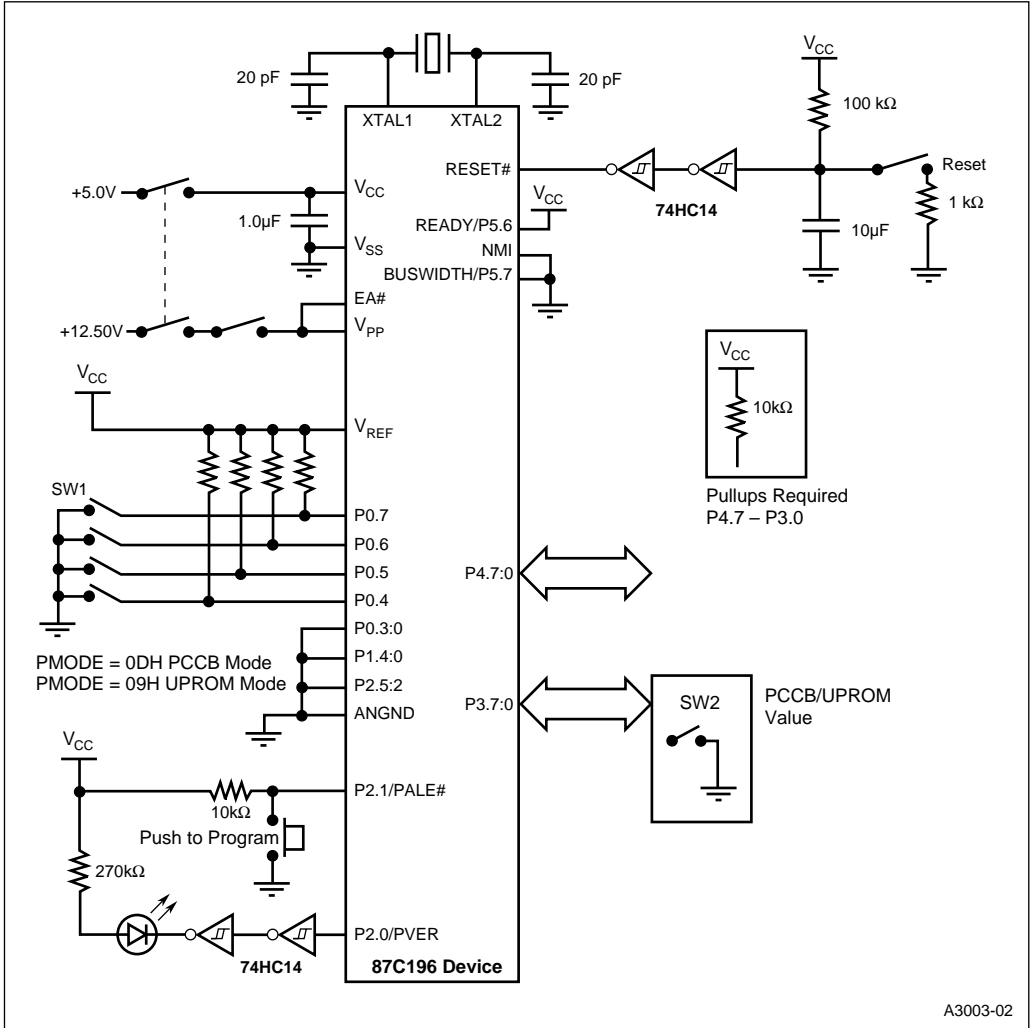


Figure 16-14. PCCB and UPROM Programming Circuit

To enter either mode, follow the standard power-up sequence (page 16-14) after setting the values listed in Table 16-13. If you want to program both the PCCB and the UPROM bits, program one of them and complete the power-down sequence. Then reconfigure the PMODE and port 3 pins and repeat the power-up sequence.

Table 16-13. PCCB and UPROM Programming Values

Pins	PCCB Programming	UPROM Programming
PMODE3:0	0DH	09H
P4.7:0	FFH	FFH
P3.7:0	Data to be programmed in PCCB (See CCR descriptions in Appendix C)	Value to program UPROM bits: 04H to program DED only 08H to program DEI only 0CH to program both DED and DEI

Assert PALE# to begin programming. The algorithm sends five programming pulses that write the port 3 data to the OTPROM, then it compares the input data with the programmed data. If the programming verifies, the PVER signal lights the LED to indicate successful programming. Otherwise, you can pulse PALE# to repeat programming. Complete the procedure by following the power-down sequence (page 16-14).

NOTE

The PCCB and UPROM programming modes are available only for the 8XC196MH device. The pulse width is 200 μ s at 8 MHz or 266 μ s at 6 MHz.

16.11 RUN-TIME PROGRAMMING

You can program an OTPROM location during normal code execution. To make the OTPROM array accessible, apply V_{CC} voltage to EA# while you reset the device. Apply V_{PP} voltage to the V_{PP} pin during the entire programming process. Then simply write to the location to be programmed.

NOTE

Programming either security-lock bit in CCB0 disables run-time programming. (For details, see "Controlling Access to the OTPROM During Normal Operation" on page 16-4.)

Immediately after writing to the OTPROM, the device must either enter idle mode or execute code from external memory. An access to OTPROM would abort the current programming cycle. Each programming cycle begins when a word is written to the OTPROM and ends when the next OTPROM access occurs. Each word requires a total of five programming cycles, each of which must be approximately 100 μ s in duration.

Figure 16-15 is a run-time programming example. It performs five programming cycles for each word. After each programming cycle, the code causes the device to enter idle mode.

The calling routine must pass two parameters to this routine — the data to be programmed (in DATA_TEMP) and the address (in ADDR_TEMP).

```
PROGRAM:
    PUSHA                                ;clear PSW, WSR, INT_MASK, INT_MASK1
    LD  WSR,#7BH                          ;select 32-byte window with EPA0_CON
    LD  COUNT,#5                          ;set up for 5 programming cycles
    ANDB INT_PEND,#CLEAR_EPA0             ;clear EPA0 pending bit
    LDB INT_MASK,#ENABLE_EPA0            ;enable EPA0 interrupt
    LDB EPA0_CON,#EPA0_TIMER              ;set up EPA0 as software timer
LOOP:
    LD  TEMP0,TIMER1                      ;load TIMER1 value into TEMP0
    ADD EPA0_TIME,TEMP0,#PGM_PULSE        ;load EPA0_TIME with TIMER1 + PGM_PULSE
    EI                                     ;enable unmasked interrupt(EPA0)
    ST  DATA_TEMP,[ADDR_TEMP]           ;store passed data at passed address
    IDLPD #1                              ;enter idle mode
    DJNZ COUNT,LOOP                      ;decrement COUNT and loop if not 0
                                           ;to complete 5 programming cycles
    POPA                                  ;restore PSW, WSR, and INT_MASKs
    RET
EPA0_ISR:
    RET
```

Figure 16-15. Run-time Programming Code Example



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-41 lists the instruction opcodes, in hexadecimal order, along with the corresponding instruction mnemonics.
- Table A-8 on page A-47 lists instruction lengths and opcodes for each applicable addressing mode.
- Table A-9 on page A-52 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	x0	x1	x2	x3	x4	x5	x6	x7
0x	SKIP	CLR	NOT	NEG	XCH di	DEC	EXT	INC
1x		CLRB	NOTB	NEGB	XCHB di	DECB	EXTB	INCB
2x	SJMP							
3x	JBC							
	bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7
4x	AND 3op di im in ix				ADD 3op di im in ix			
5x	ANDB 3op di im in ix				ADDB 3op di im in ix			
6x	AND 2op di im in ix				ADD 2op di im in ix			
7x	ANDB 2op di im in ix				ADDB 2op di im in ix			
8x	OR di im in ix				XOR di im in ix			
9x	ORB di im in ix				XORB di im in ix			
Ax	LD di im in ix				ADDC di im in ix			
Bx	LDB di im in ix				ADDCB di im in ix			
Cx	ST di	BMOV	ST in ix		STB di	CMPL	STB in ix	
Dx	JNST	JNH	JGT	JNC	JNVT	JNV	JGE	JNE
Ex	DJNZ	DJNZW	TIJMP	BR in				LJMP
Fx	RET		PUSHF	POPF	PUSHA	POPA	IDLDP	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 reserved opcodes. If the CPU attempts to execute an unimplemented opcode, an interrupt occurs. For more information, see "Unimplemented Opcode" on page 5-6.

Table A-1. Opcode Map (Right Half)

Opcode	x8	x9	xA	xB	xC	xD	xE	xF
0x	SHR	SHL	SHRA	XCH ix	SHRL	SHLL	SHRAL	NORML
1x	SHRB	SHLB	SHRAB	XCHB ix				
2x	SCALL							
3x	JBS							
	bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7
4x	SUB 3op				MULU 3op (Note 2)			
	di	im	in	ix	di	im	in	ix
5x	SUBB 3op				MULUB 3op (Note 2)			
	di	im	in	ix	di	im	in	ix
6x	SUB 2op				MULU 2op (Note 2)			
	di	im	in	ix	di	im	in	ix
7x	SUBB 2op				MULUB 2op (Note 2)			
	di	im	in	ix	di	im	in	ix
8x	CMP				DIVU (Note 2)			
	di	im	in	ix	di	im	in	ix
9x	CMPB				DIVUB (Note 2)			
	di	im	in	ix	di	im	in	ix
Ax	SUBC				LDBZE			
	di	im	in	ix	di	im	in	ix
Bx	SUBCB				LDBSE			
	di	im	in	ix	di	im	in	ix
Cx	PUSH				POP	BMOVI	POP	
	di	im	in	ix	di		in	ix
Dx	JST	JH	JLE	JC	JVT	JV	JLT	JE
Ex					DPTS		(Note 1)	LCALL
Fx	CLRC	SETC	DI	EI	CLRVT	NOP	signed MUL/DIV (Note 2)	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. The first byte is "FE" and the second is the opcode of the corresponding unsigned instruction.

Table A-2. Processor Status Word (PSW) Flags

Mnemonic	Description																					
C	<p>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.</p> <table> <tr> <td>C</td> <td>Value of Bits Shifted Off</td> </tr> <tr> <td>0</td> <td>< ½ LSB</td> </tr> <tr> <td>1</td> <td>≥ ½ LSB</td> </tr> </table> <p>Normally, the result is rounded up if the carry flag is set. The sticky bit flag allows a finer resolution in the rounding decision.</p> <table> <tr> <td>C</td> <td>ST</td> <td>Value of Bits Shifted Off</td> </tr> <tr> <td>0</td> <td>0</td> <td>= 0</td> </tr> <tr> <td>0</td> <td>1</td> <td>> 0 and < ½ LSB</td> </tr> <tr> <td>1</td> <td>0</td> <td>= ½ LSB</td> </tr> <tr> <td>1</td> <td>1</td> <td>> ½ LSB and < 1 LSB</td> </tr> </table>	C	Value of Bits Shifted Off	0	< ½ LSB	1	≥ ½ LSB	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
C	Value of Bits Shifted Off																					
0	< ½ LSB																					
1	≥ ½ LSB																					
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	<p>The overflow flag is set to indicate that the result of an operation is too large to be represented correctly in the available space.</p> <p>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. (Chapter 3, "Programming Considerations," defines the operands and possible values for each.)</p> <table> <tr> <td>Instruction</td> <td>Quotient Stored in:</td> <td>V Flag Set if Quotient is:</td> </tr> <tr> <td>DIVB</td> <td>Short-integer</td> <td>< -128 or > +127 (< 81H or > 7FH)</td> </tr> <tr> <td>DIV</td> <td>Integer</td> <td>< -32768 or > +32767 (< 8001H or > 7FFFH)</td> </tr> <tr> <td>DIVUB</td> <td>Byte</td> <td>> 255 (FFH)</td> </tr> <tr> <td>DIVU</td> <td>Word</td> <td>> 65535 (FFFFH)</td> </tr> </table>	Instruction	Quotient Stored in:	V Flag Set if Quotient is:	DIVB	Short-integer	< -128 or > +127 (< 81H or > 7FH)	DIV	Integer	< -32768 or > +32767 (< 8001H or > 7FFFH)	DIVUB	Byte	> 255 (FFH)	DIVU	Word	> 65535 (FFFFH)						
Instruction	Quotient Stored in:	V Flag Set if Quotient is:																				
DIVB	Short-integer	< -128 or > +127 (< 81H or > 7FH)																				
DIV	Integer	< -32768 or > +32767 (< 8001H or > 7FFFH)																				
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 $\neq 0$	decremented byte = 0
DJNZW	decremented word $\neq 0$	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 <i>S</i> or a <i>D</i> . 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.
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.
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 <i>w2_reg</i> is similar to <i>lreg</i> , there is a distinction: <i>w2_reg</i> consists of two halves, each containing a 16-bit address; <i>lreg</i> 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 <i>S</i> or a <i>D</i> . 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. $(DEST) \leftarrow (DEST) + (SRC)$ <table border="1" data-bbox="325 447 646 543"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>↑</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	✓	✓	↑	—	DEST, SRC ADD wreg, waop (011001aa) (waop) (wreg)
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	✓	✓	↑	—															
ADD (3 operands)	ADD WORDS. Adds the two source word operands and stores the sum into the destination operand. $(DEST) \leftarrow (SRC1) + (SRC2)$ <table border="1" data-bbox="325 696 646 791"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>↑</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	✓	✓	↑	—	DEST, SRC1, SRC2 ADD Dwreg, Swreg, waop (010001aa) (waop) (Swreg) (Dwreg)
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	✓	✓	↑	—															
ADDB (2 operands)	ADD BYTES. Adds the source and destination byte operands and stores the sum into the destination operand. $(DEST) \leftarrow (DEST) + (SRC)$ <table border="1" data-bbox="325 944 646 1039"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>↑</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	✓	✓	↑	—	DEST, SRC ADDB breg, baop (011101aa) (baop) (breg)
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	✓	✓	↑	—															
ADDB (3 operands)	ADD BYTES. Adds the two source byte operands and stores the sum into the destination operand. $(DEST) \leftarrow (SRC1) + (SRC2)$ <table border="1" data-bbox="325 1194 646 1289"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>↑</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	✓	✓	↑	—	DEST, SRC1, SRC2 ADDB Dbreg, Sbreg, baop (010101aa) (baop) (Sbreg) (Dbreg)
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	✓	✓	↑	—															
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. $(DEST) \leftarrow (DEST) + (SRC) + C$ <table border="1" data-bbox="325 1463 646 1558"> <thead> <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> </thead> <tbody> <tr> <td>↓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>↑</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	↓	✓	✓	✓	↑	—	DEST, SRC ADDC wreg, waop (101001aa) (waop) (wreg)
PSW Flag Settings																				
Z	N	C	V	VT	ST															
↓	✓	✓	✓	↑	—															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
ADDCB	<p>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.</p> $(DEST) \leftarrow (DEST) + (SRC) + C$ <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>↓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>↑</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	↓	✓	✓	✓	↑	—	<p>DEST, SRC</p> <p>ADDCB breg, baop (101101aa) (baop) (breg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
↓	✓	✓	✓	↑	—															
AND (2 operands)	<p>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.</p> $(DEST) \leftarrow (DEST) \text{ AND } (SRC)$ <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>0</td> <td>0</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	0	0	—	—	<p>DEST, SRC</p> <p>AND wreg, waop (011000aa) (waop) (wreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	0	0	—	—															
AND (3 operands)	<p>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.</p> $(DEST) \leftarrow (SRC1) \text{ AND } (SRC2)$ <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>0</td> <td>0</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	0	0	—	—	<p>DEST, SRC1, SRC2</p> <p>AND Dwreg, Swreg, waop (010000aa) (waop) (Swreg) (Dwreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	0	0	—	—															
ANDB (2 operands)	<p>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.</p> $(DEST) \leftarrow (DEST) \text{ AND } (SRC)$ <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>0</td> <td>0</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	0	0	—	—	<p>DEST, SRC</p> <p>ANDB breg, baop (011100aa) (baop) (breg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	0	0	—	—															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
ANDB (3 operands)	<p>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.</p> <p>(DEST) ← (SRC1) AND (SRC2)</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>0</td> <td>0</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	0	0	—	—	<p>DEST, SRC1, SRC2</p> <p>ANDB Dbreg, Sbreg, baop (010100aa) (baop) (Sbreg) (Dbreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	0	0	—	—															
BMOV	<p>BLOCK MOVE. Moves a block of word data from one location in memory to another. The source and destination addresses are calculated using the indirect with autoincrement addressing mode. 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. The blocks of word data can be located anywhere in register RAM, but should not overlap.</p> <p>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</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>PTRS, CNTREG</p> <p>BMOV lreg, wreg (11000001) (wreg) (lreg)</p> <p>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.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
BMOVI	<p>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 the indirect with autoincrement addressing mode. 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. The blocks of word data can be located anywhere in register RAM, but should not overlap.</p> <p>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</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>PTRS, CNTREG</p> <p>BMOVI Ireg, wreg (11001101) (wreg) (Ireg)</p> <p>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.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															
BR	<p>BRANCH INDIRECT. Continues execution at the address specified in the operand word register.</p> <p>PC ← (DEST)</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>DEST</p> <p>BR [wreg] (11100011) (wreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															
CLR	<p>CLEAR WORD. Clears the value of the operand.</p> <p>(DEST) ← 0</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>1</td> <td>0</td> <td>0</td> <td>0</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	1	0	0	0	—	—	<p>DEST</p> <p>CLR wreg (00000001) (wreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
1	0	0	0	—	—															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
CLRB	<p>CLEAR BYTE. Clears the value of the operand. $(\text{DEST}) \leftarrow 0$</p> <table border="1"> <thead> <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> </thead> <tbody> <tr> <td>1</td> <td>0</td> <td>0</td> <td>0</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	1	0	0	0	—	—	<p>DEST CLRB breg (00010001) (breg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
1	0	0	0	—	—															
CLRC	<p>CLEAR CARRY FLAG. Clears the carry flag. $C \leftarrow 0$</p> <table border="1"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>0</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	0	—	—	—	<p>CLRC (11111000)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	0	—	—	—															
CLRVT	<p>CLEAR OVERFLOW-TRAP FLAG. Clears the overflow-trap flag. $VT \leftarrow 0$</p> <table border="1"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>0</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	0	—	<p>CLRVT (11111100)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	0	—															
CMP	<p>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. $(\text{DEST}) - (\text{SRC})$</p> <table border="1"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>↑</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	✓	✓	↑	—	<p>DEST, SRC CMP wreg, waop (100010aa) (waop) (wreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	✓	✓	↑	—															
CMPB	<p>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. $(\text{DEST}) - (\text{SRC})$</p> <table border="1"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>↑</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	✓	✓	↑	—	<p>DEST, SRC CMPB breg, baop (100110aa) (baop) (breg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	✓	✓	↑	—															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
CMPL	<p>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.</p> <p>(DEST) – (SRC)</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	✓	✓	✓	—	<p>DEST, SRC</p> <p>CMPL Dreg, Sreg (11000101) (Sreg) (Dreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	✓	✓	✓	—															
DEC	<p>DECREMENT WORD. Decrements the value of the operand by one.</p> <p>(DEST) ← (DEST) – 1</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>↑</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	✓	✓	↑	—	<p>DEST</p> <p>DEC wreg (00000101) (wreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	✓	✓	↑	—															
DECB	<p>DECREMENT BYTE. Decrements the value of the operand by one.</p> <p>(DEST) ← (DEST) – 1</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>↑</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	✓	✓	↑	—	<p>DEST</p> <p>DECB breg (00010101) (breg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	✓	✓	↑	—															
DI	<p>DISABLE INTERRUPTS. Disables interrupts. Interrupt calls cannot occur after this instruction.</p> <p>Interrupt Enable (PSW.1) ← 0</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>DI (11111010)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
DIV	<p>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.</p> <p>(low word DEST) ← (DEST) / (SRC) (high word DEST) ← (DEST) MOD (SRC)</p> <table border="1" data-bbox="325 562 645 661"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>✓</td> <td>↑</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	✓	↑	—	<p>DEST, SRC</p> <p>DIV lreg, waop (11111110) (100011aa) (waop) (lreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	✓	↑	—															
DIVB	<p>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.</p> <p>(low byte DEST) ← (DEST) / (SRC) (high byte DEST) ← (DEST) MOD (SRC)</p> <table border="1" data-bbox="325 979 645 1078"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>✓</td> <td>↑</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	✓	↑	—	<p>DEST, SRC</p> <p>DIVB wreg, baop (11111110) (100111aa) (baop) (wreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	✓	↑	—															
DIVU	<p>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.</p> <p>(low word DEST) ← (DEST) / (SRC) (high word DEST) ← (DEST) MOD (SRC)</p> <table border="1" data-bbox="325 1395 645 1494"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>✓</td> <td>↑</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	✓	↑	—	<p>DEST, SRC</p> <p>DIVU lreg, waop (100011aa) (waop) (lreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	✓	↑	—															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
DIVUB	<p>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.</p> <p>(low byte DEST) ← (DEST) / (SRC) (high byte DEST) ← (DEST) MOD (SRC)</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>✓</td> <td>↑</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	✓	↑	—	<p>DEST, SRC</p> <p>DIVUB wreg, baop (100111aa) (baop) (wreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	✓	↑	—															
DJNZ	<p>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.</p> <p>(COUNT) ← (COUNT) - 1 if (COUNT) ≠ 0 then PC ← PC + 8-bit disp end_if</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>DJNZ breg, cadd (11100000) (breg) (disp)</p> <p>NOTE: The displacement (disp) is sign-extended to 16 bits.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															
DJNZW	<p>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.</p> <p>(COUNT) ← (COUNT) - 1 if (COUNT) ≠ 0 then PC ← PC + 8-bit disp end_if</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>DJNZW wreg, cadd (11100001) (wreg) (disp)</p> <p>NOTE: The displacement (disp) is sign-extended to 16 bits</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
DPTS	<p>DISABLE PERIPHERAL TRANSACTION SERVER (PTS). Disables the peripheral transaction server (PTS). PTS Disable (PSW.2) ← 0</p> <table border="1" data-bbox="325 395 646 491"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>DPTS (11101100)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															
EI	<p>ENABLE INTERRUPTS. Enables interrupts following the execution of the next statement. Interrupt calls cannot occur immediately following this instruction. Interrupt Enable (PSW.1) ← 1</p> <table border="1" data-bbox="325 670 646 765"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>EI (11111011)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															
EPTS	<p>ENABLE PERIPHERAL TRANSACTION SERVER (PTS). Enables the peripheral transaction server (PTS). PTS Enable (PSW.2) ← 1</p> <table border="1" data-bbox="325 916 646 1012"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>EPTS (11101101)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															
EXT	<p>SIGN-EXTEND INTEGER INTO LONG-INTEGGER. 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</p> <table border="1" data-bbox="325 1286 646 1381"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>0</td> <td>0</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	0	0	—	—	<p>EXT lreg (00000110) (lreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	0	0	—	—															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																																				
EXTB	<p>SIGN-EXTEND SHORT-INTEGER INTO INTEGER. Sign-extends the low-order byte of the operand throughout the high-order byte of the operand.</p> <p>if DEST.7 = 1 then (high byte DEST) ← 0FFH else (high byte DEST) ← 0 end_if</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>0</td> <td>0</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	0	0	—	—	<p>EXTB wreg (00010110) (wreg)</p>																		
PSW Flag Settings																																						
Z	N	C	V	VT	ST																																	
✓	✓	0	0	—	—																																	
IDLPD	<p>IDLE/POWERDOWN. Depending on the 8-bit value of the KEY operand, this instruction causes the device to:</p> <ul style="list-style-type: none"> enter idle mode, if KEY=1, enter powerdown mode, if KEY=2, execute a reset sequence, if KEY = any value other than 1 or 2. <p>The bus controller completes any prefetch cycle in progress before the CPU stops or resets.</p> <p>if KEY = 1 then enter idle else if KEY = 2 then enter powerdown else execute reset</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td colspan="6" style="text-align: center;">KEY = 1 or 2</td> </tr> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> <tr> <td colspan="6" style="text-align: center;">KEY = any value other than 1 or 2</td> </tr> <tr> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	KEY = 1 or 2						—	—	—	—	—	—	KEY = any value other than 1 or 2						0	0	0	0	0	0	<p>IDLPD #key (11110110) (key)</p>
PSW Flag Settings																																						
Z	N	C	V	VT	ST																																	
KEY = 1 or 2																																						
—	—	—	—	—	—																																	
KEY = any value other than 1 or 2																																						
0	0	0	0	0	0																																	
INC	<p>INCREMENT WORD. Increments the value of the word operand by 1.</p> <p>(DEST) ← (DEST) + 1</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>↑</td> <td>0</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	✓	✓	↑	0	<p>INC wreg (00000111) (wreg)</p>																		
PSW Flag Settings																																						
Z	N	C	V	VT	ST																																	
✓	✓	✓	✓	↑	0																																	

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
INCB	<p>INCREMENT BYTE. Increments the value of the byte operand by 1.</p> $(DEST) \leftarrow (DEST) + 1$ <table border="1" style="margin: 10px auto;"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>↑</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	✓	✓	↑	—	<p>INCB breg (00010111) (breg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	✓	✓	↑	—															
JBC	<p>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.</p> <p>if (specified bit) = 0 then $PC \leftarrow PC + 8\text{-bit disp}$</p> <table border="1" style="margin: 10px auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>JBC breg, bitno, cadd (00110bbb) (breg) (disp)</p> <p>NOTE: The displacement (disp) is sign-extended to 16 bits.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															
JBS	<p>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.</p> <p>if (specified bit) = 1 then $PC \leftarrow PC + 8\text{-bit disp}$</p> <table border="1" style="margin: 10px auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>JBS breg, bitno, cadd (00111bbb) (breg) (disp)</p> <p>NOTE: The displacement (disp) is sign-extended to 16 bits.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
JC	<p>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 $+127$.</p> <p>if C = 1 then $PC \leftarrow PC + 8\text{-bit disp}$</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>JC cadd (11011011) (disp)</p> <p>NOTE: The displacement (disp) is sign-extended to 16 bits.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															
JE	<p>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 $+127$.</p> <p>if Z = 1 then $PC \leftarrow PC + 8\text{-bit disp}$</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>JE cadd (11011111) (disp)</p> <p>NOTE: The displacement (disp) is sign-extended to 16 bits.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															
JGE	<p>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 $+127$.</p> <p>if N = 0 then $PC \leftarrow PC + 8\text{-bit disp}$</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>JGE cadd (11010110) (disp)</p> <p>NOTE: The displacement (disp) is sign-extended to 16 bits.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
JGT	<p>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.</p> <p>if N = 0 AND Z = 0 then $PC \leftarrow PC + 8\text{-bit disp}$</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>JGT cadd (11010010) (disp)</p> <p>NOTE: The displacement (disp) is sign-extended to 16 bits.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															
JH	<p>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.</p> <p>if C = 1 AND Z = 0 then $PC \leftarrow PC + 8\text{-bit disp}$</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>JH cadd (11011001) (disp)</p> <p>NOTE: The displacement (disp) is sign-extended to 16 bits.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															
JLE	<p>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.</p> <p>if N = 1 OR Z = 1 then $PC \leftarrow PC + 8\text{-bit disp}$</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>JLE cadd (11011010) (disp)</p> <p>NOTE: The displacement (disp) is sign-extended to 16 bits.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
JLT	<p>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.</p> <p>if N = 1 then PC ← PC + 8-bit disp</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>JLT cadd (11011110) (disp)</p> <p>NOTE: The displacement (disp) is sign-extended to 16 bits.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															
JNC	<p>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.</p> <p>if C = 0 then PC ← PC + 8-bit disp</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>JNC cadd (11010011) (disp)</p> <p>NOTE: The displacement (disp) is sign-extended to 16 bits.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															
JNE	<p>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.</p> <p>if Z = 0 then PC ← PC + 8-bit disp</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>JNE cadd (11010111) (disp)</p> <p>NOTE: The displacement (disp) is sign-extended to 16 bits.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
<p>JNH</p>	<p>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 $+127$.</p> <p>if $C = 0$ OR $Z = 1$ then $PC \leftarrow PC + 8\text{-bit disp}$</p> <table border="1" data-bbox="325 586 645 683"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>JNH cadd (11010001) (disp)</p> <p>NOTE: The displacement (disp) is sign-extended to 16 bits.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															
<p>JNST</p>	<p>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 $+127$.</p> <p>if $ST = 0$ then $PC \leftarrow PC + 8\text{-bit disp}$</p> <table border="1" data-bbox="325 979 645 1076"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>JNST cadd (11010000) (disp)</p> <p>NOTE: The displacement (disp) is sign-extended to 16 bits.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															
<p>JNV</p>	<p>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 $+127$.</p> <p>if $V = 0$ then $PC \leftarrow PC + 8\text{-bit disp}$</p> <table border="1" data-bbox="325 1367 645 1465"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>JNV cadd (11010101) (disp)</p> <p>NOTE: The displacement (disp) is sign-extended to 16 bits.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
JNVT	<p>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.</p> <p>if VT = 0 then $PC \leftarrow PC + 8\text{-bit disp}$</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>0</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	0	—	<p>JNVT cadd (11010100) (disp)</p> <p>NOTE: The displacement (disp) is sign-extended to 16 bits.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	0	—															
JST	<p>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.</p> <p>if ST = 1 then $PC \leftarrow PC + 8\text{-bit disp}$</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>JST cadd (11011000) (disp)</p> <p>NOTE: The displacement (disp) is sign-extended to 16 bits.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															
JV	<p>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.</p> <p>if V = 1 then $PC \leftarrow PC + 8\text{-bit disp}$</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>JV cadd (11011101) (disp)</p> <p>NOTE: The displacement (disp) is sign-extended to 16 bits.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
JVT	<p>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.</p> <p>if VT = 1 then $PC \leftarrow PC + 8\text{-bit disp}$</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>0</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	0	—	<p>JVT cadd (11011100) (disp)</p> <p>NOTE: The displacement (disp) is sign-extended to 16 bits.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	0	—															
LCALL	<p>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.</p> <p>$SP \leftarrow SP - 2$ $(SP) \leftarrow PC$ $PC \leftarrow PC + 16\text{-bit disp}$</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>LCALL cadd (11101111) (disp-low) (disp-high)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															
LD	<p>LOAD WORD. Loads the value of the source word operand into the destination operand. $(DEST) \leftarrow (SRC)$</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>DEST, SRC LD wreg, waop (101000aa) (waop) (wreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															
LDB	<p>LOAD BYTE. Loads the value of the source byte operand into the destination operand. $(DEST) \leftarrow (SRC)$</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>DEST, SRC LDB breg, baop (101100aa) (baop) (breg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
LDBSE	<p>LOAD BYTE SIGN-EXTENDED. Sign-extends the value of the source short-integer operand and loads it into the destination integer operand.</p> <p>(low byte DEST) ← (SRC)</p> <p>if DEST.15 = 1 then (high word DEST) ← 0FFH else (high word DEST) ← 0 end_if</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>DEST, SRC</p> <p>LDBSE wreg, baop (101111aa) (baop) (wreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															
LDBZE	<p>LOAD BYTE ZERO-EXTENDED. Zero-extends the value of the source byte operand and loads it into the destination word operand.</p> <p>(low byte DEST) ← (SRC) (high byte DEST) ← 0</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>DEST, SRC</p> <p>LDBZE wreg, baop (101011aa) (baop) (wreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															
LJMP	<p>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.</p> <p>PC ← PC + 16-bit disp</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>?</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	?	<p>LJMP cadd (11100111) (disp-low) (disp-high)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	?															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
MUL (2 operands)	<p>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.</p> $(DEST) \leftarrow (DEST) \times (SRC)$ <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>?</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	?	<p>DEST, SRC</p> <p>MUL Ireg, waop (11111110) (011011aa) (waop) (Ireg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	?															
MUL (3 operands)	<p>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.</p> $(DEST) \leftarrow (SRC1) \times (SRC2)$ <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>?</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	?	<p>DEST, SRC1, SRC2</p> <p>MUL Ireg, wreg, waop (11111110) (010011aa) (waop) (wreg) (Ireg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	?															
MULB (2 operands)	<p>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.</p> $(DEST) \leftarrow (DEST) \times (SRC)$ <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>?</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	?	<p>DEST, SRC</p> <p>MULB wreg, baop (11111110) (011111aa) (baop) (wreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	?															
MULB (3 operands)	<p>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.</p> $(DEST) \leftarrow (SRC1) \times (SRC2)$ <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>?</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	?	<p>DEST, SRC1, SRC2</p> <p>MULB wreg, breg, baop (11111110) (010111aa) (baop) (breg) (wreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	?															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
MULU (2 operands)	<p>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.</p> $(DEST) \leftarrow (DEST) \times (SRC)$ <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>?</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	?	<p>DEST, SRC</p> <p>MULU <i>lreg, waop</i> (011011aa) (waop) (lreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	?															
MULU (3 operands)	<p>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.</p> $(DEST) \leftarrow (SRC1) \times (SRC2)$ <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>?</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	?	<p>DEST, SRC1, SRC2</p> <p>MULU <i>lreg, wreg, waop</i> (010011aa) (waop) (wreg) (lreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	?															
MULUB (2 operands)	<p>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.</p> $(DEST) \leftarrow (DEST) \times (SRC)$ <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>?</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	?	<p>DEST, SRC</p> <p>MULUB <i>wreg, baop</i> (011111aa) (baop) (wreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	?															
MULUB (3 operands)	<p>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.</p> $(DEST) \leftarrow (SRC1) \times (SRC2)$ <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>?</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	?	<p>DEST, SRC1, SRC2</p> <p>MULUB <i>wreg, breg, baop</i> (010111aa) (baop) (breg) (wreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	?															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
NEG	<p>NEGATE INTEGER. Negates the value of the integer operand. $(DEST) \leftarrow -(DEST)$</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>↑</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	✓	✓	↑	—	<p>NEG wreg (00000011) (wreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	✓	✓	↑	—															
NEGB	<p>NEGATE SHORT-INTEGGER. Negates the value of the short-integer operand. $(DEST) \leftarrow -(DEST)$</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>↑</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	✓	✓	↑	—	<p>NEGB breg (00010011) (breg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	✓	✓	↑	—															
NOP	<p>NO OPERATION. Does nothing. Control passes to the next sequential instruction.</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>NOP (11111101)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															
NORML	<p>NORMALIZE LONG-INTEGGER. 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. $(COUNT) \leftarrow 0$ do while $(MSB(DEST) = 0) \text{ AND } (COUNT) < 31$ $(DEST) \leftarrow (DEST) \times 2$ $(COUNT) \leftarrow (COUNT) + 1$ end_while</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>?</td> <td>0</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	?	0	—	—	—	<p>SRC, DEST NORML lreg, breg (00001111) (breg) (lreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	?	0	—	—	—															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
NOT	<p>COMPLEMENT WORD. Complements the value of the word operand (replaces each "1" with a "0" and each "0" with a "1").</p> <p>(DEST) ← NOT (DEST)</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>0</td> <td>0</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	0	0	—	—	<p>NOT wreg (00000010) (wreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	0	0	—	—															
NOTB	<p>COMPLEMENT BYTE. Complements the value of the byte operand (replaces each "1" with a "0" and each "0" with a "1").</p> <p>(DEST) ← NOT (DEST)</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>0</td> <td>0</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	0	0	—	—	<p>NOTB breg (00010010) (breg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	0	0	—	—															
OR	<p>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".</p> <p>(DEST) ← (DEST) OR (SRC)</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>0</td> <td>0</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	0	0	—	—	<p>DEST, SRC OR wreg, waop (100000aa) (waop) (wreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	0	0	—	—															
ORB	<p>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".</p> <p>(DEST) ← (DEST) OR (SRC)</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>0</td> <td>0</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	0	0	—	—	<p>DEST, SRC ORB breg, baop (100100aa) (baop) (breg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	0	0	—	—															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
POP	<p>POP WORD. Pops the word on top of the stack and places it at the destination operand.</p> <p>(DEST) ← (SP) SP ← SP + 2</p> <table border="1" data-bbox="325 421 646 517"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>POP waop (110011aa) (waop)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															
POPA	<p>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.</p> <p>INT_MASK1/WSR ← (SP) SP ← SP + 2 PSW/INT_MASK ← (SP) SP ← SP + 2</p> <table border="1" data-bbox="325 881 646 977"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	✓	✓	✓	✓	<p>POPA (11110101)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	✓	✓	✓	✓															
POPF	<p>POP FLAGS. Pops the word on top of the stack and places it into the PSW. Interrupt calls cannot occur immediately following this instruction.</p> <p>(PSW)/INT_MASK ← (SP) SP ← SP + 2</p> <table border="1" data-bbox="325 1177 646 1272"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	✓	✓	✓	✓	<p>POPF (11110011)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	✓	✓	✓	✓															
PUSH	<p>PUSH WORD. Pushes the word operand onto the stack.</p> <p>SP ← SP – 2 (SP) ← (DEST)</p> <table border="1" data-bbox="325 1425 646 1520"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>PUSH waop (110010aa) (waop)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
PUSHA	<p>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.</p> <p>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.</p> <p>$SP \leftarrow SP - 2$ $(SP) \leftarrow PSW/INT_MASK$ $PSW/INT_MASK \leftarrow 0$ $SP \leftarrow SP - 2$ $(SP) \leftarrow INT_MASK1/WSR$ $INT_MASK1 \leftarrow 0$</p> <table border="1"> <thead> <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> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	0	0	0	0	0	0	<p>PUSHA (11110100)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
0	0	0	0	0	0															
PUSHF	<p>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.</p> <p>$SP \leftarrow SP - 2$ $(SP) \leftarrow PSW/INT_MASK$ $PSW/INT_MASK \leftarrow 0$</p> <table border="1"> <thead> <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> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	0	0	0	0	0	0	<p>PUSHF (11110010)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
0	0	0	0	0	0															
RET	<p>RETURN FROM SUBROUTINE. Pops the PC off the top of the stack.</p> <p>$PC \leftarrow (SP)$ $SP \leftarrow SP + 2$</p> <table border="1"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>RET (11110000)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
RST	<p>RESET SYSTEM. Initializes the PSW to zero, the PC to 2080H, and the pins and SFRs to their reset values. Executing this instruction causes the RESET# pin to be pulled low for 16 state times.</p> <p>SFR ← Reset Status Pin ← Reset Status PSW ← 0 PC ← 2080H</p> <table border="1" data-bbox="325 517 646 612"> <thead> <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> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	0	0	0	0	0	0	<p>RST (11111111)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
0	0	0	0	0	0															
SCALL	<p>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.</p> <p>SP ← SP - 2 (SP) ← PC PC ← PC + 11-bit disp</p> <table border="1" data-bbox="325 907 646 1003"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>SCALL cadd (00101xxx) (disp-low)</p> <p>NOTE: The displacement (disp) is sign-extended to 16-bits.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															
SETC	<p>SET CARRY FLAG. Sets the carry flag.</p> <p>C ← 1</p> <table border="1" data-bbox="325 1107 646 1203"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>1</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	1	—	—	—	<p>SETC (11111001)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	1	—	—	—															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
SHL	<p>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 (10H – 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.</p> <p>Temp ← (COUNT) do while Temp ≠ 0 C ← High order bit of (DEST) (DEST) ← (DEST) × 2 Temp ← Temp – 1 end_while</p> <table border="1" data-bbox="325 682 646 777"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>↑</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	✓	✓	↑	—	<p>SHL wreg, #count (00001001) (count) (wreg)</p> <p>or</p> <p>SHL wreg, breg (00001001) (breg) (wreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	✓	✓	↑	—															
SHLB	<p>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 (10H – 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.</p> <p>Temp ← (COUNT) do while Temp ≠ 0 C ← High order bit of (DEST) (DEST) ← (DEST) × 2 Temp ← Temp – 1 end_while</p> <table border="1" data-bbox="325 1215 646 1310"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>↑</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	✓	✓	↑	—	<p>SHLB breg, #count (00011001) (count) (breg)</p> <p>or</p> <p>SHLB breg, breg (00011001) (breg) (breg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	✓	✓	↑	—															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
SHLL	<p>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 (10H – 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.</p> <p>Temp ← (COUNT) do while Temp ≠ 0 C ← High order bit of (DEST) (DEST) ← (DEST) × 2 Temp ← Temp – 1 end_while</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>↑</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	✓	✓	↑	—	<p>SHLL Ireg, #count (00001101) (count) (Ireg)</p> <p>or</p> <p>SHLL Ireg, breg (00001101) (breg) (Ireg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	✓	✓	↑	—															
SHR	<p>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 (10H – 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.</p> <p>Temp ← (COUNT) do while Temp ≠ 0 C ← Low order bit of (DEST) (DEST) ← (DEST)/2 Temp ← Temp – 1 end_while</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>0</td> <td>✓</td> <td>0</td> <td>—</td> <td>✓</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	0	✓	0	—	✓	<p>SHR wreg, #count (00001000) (count) (wreg)</p> <p>or</p> <p>SHR wreg, breg (00001000) (breg) (wreg)</p> <p>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.</p> <p>In this operation, (DEST)/2 represents unsigned division.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	0	✓	0	—	✓															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
SHRA	<p>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 (10H – 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.</p> <p>Temp ← (COUNT) do while Temp ≠ 0 C ← Low order bit of (DEST) (DEST) ← (DEST)/2 Temp ← Temp – 1 end_while</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>✓</td> <td>0</td> <td>—</td> <td>✓</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	✓	0	—	✓	<p>SHRA wreg, #count (00001010) (count) (wreg)</p> <p>or</p> <p>SHRA wreg, breg (00001010) (breg) (wreg)</p> <p>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.</p> <p>In this operation, (DEST)/2 represents signed division.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	✓	0	—	✓															
SHRAB	<p>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 (10H – 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.</p> <p>Temp ← (COUNT) do while Temp ≠ 0 C = Low order bit of (DEST) (DEST) ← (DEST)/2 Temp ← Temp – 1 end_while</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>✓</td> <td>0</td> <td>—</td> <td>✓</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	✓	0	—	✓	<p>SHRAB breg, #count (00011010) (count) (breg)</p> <p>or</p> <p>SHRAB breg, breg (00011010) (breg) (breg)</p> <p>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.</p> <p>In this operation, (DEST)/2 represents signed division.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	✓	0	—	✓															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
SHRAL	<p>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 (10H – 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.</p> <p>Temp ← (COUNT) do while Temp ≠ 0 C ← Low order bit of (DEST) (DEST) ← (DEST)/2 Temp ← Temp – 1 end_while</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>✓</td> <td>0</td> <td>—</td> <td>✓</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	✓	0	—	✓	<p>SHRAL Ireg, #count (00001110) (count) (Ireg)</p> <p>or</p> <p>SHRAL Ireg, breg (00001110) (breg) (Ireg)</p> <p>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.</p> <p>In this operation, (DEST)/2 represents signed division.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	✓	0	—	✓															
SHRB	<p>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 (10H – 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.</p> <p>Temp ← (COUNT) do while Temp ≠ 0 C ← Low order bit of (DEST) (DEST) ← (DEST)/2 Temp ← Temp – 1 end_while</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>0</td> <td>✓</td> <td>0</td> <td>—</td> <td>✓</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	0	✓	0	—	✓	<p>SHRB breg, #count (00011000) (count) (breg)</p> <p>or</p> <p>SHRB breg, breg (00011000) (breg) (breg)</p> <p>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.</p> <p>In this operation, (DEST)/2 represents unsigned division.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	0	✓	0	—	✓															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
SHRL	<p>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 (10H – 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.</p> <p>Temp ← (COUNT) do while Temp ≠ 0 C ← Low order bit of (DEST) (DEST) ← (DEST)/2 Temp ← Temp – 1 end_while</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>0</td> <td>✓</td> <td>0</td> <td>—</td> <td>✓</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	0	✓	0	—	✓	<p>SHRL Ireg, #count (00001100) (count) (Ireg)</p> <p>or</p> <p>SHRL Ireg, breg (00001100) (breg) (Ireg)</p> <p>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.</p> <p>In this operation, (DEST)/2 represents unsigned division.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	0	✓	0	—	✓															
SJMP	<p>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.</p> <p>PC ← PC + 11-bit disp</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>SJMP cadd (00100xxx) (disp-low)</p> <p>NOTE: The displacement (disp) is sign-extended to 16 bits.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															
SKIP	<p>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.</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>SKIP breg (00000000) (breg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
ST	<p>STORE WORD. Stores the value of the source (leftmost) word operand into the destination (rightmost) operand. (DEST) ← (SRC)</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>SRC, DEST ST wreg, waop (110000aa) (waop) (wreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															
STB	<p>STORE BYTE. Stores the value of the source (leftmost) byte operand into the destination (rightmost) operand. (DEST) ← (SRC)</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>SRC, DEST STB breg, baop (110001aa) (baop) (breg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															
SUB (2 operands)	<p>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)</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>↑</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	✓	✓	↑	—	<p>DEST, SRC SUB wreg, waop (011010aa) (waop) (wreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	✓	✓	↑	—															
SUB (3 operands)	<p>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)</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>↑</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	✓	✓	↑	—	<p>DEST, SRC1, SRC2 SUB Dwreg, Swreg, waop (010010aa) (waop) (Swreg) (Dwreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	✓	✓	↑	—															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
SUBB (2 operands)	<p>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.</p> $(DEST) \leftarrow (DEST) - (SRC)$ <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>↑</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	✓	✓	↑	—	<p>DEST, SRC</p> <p>SUBB breg, baop (011110aa) (baop) (breg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	✓	✓	↑	—															
SUBB (3 operands)	<p>SUBTRACT BYTES. Subtracts the first source byte operand from the second, stores the result in the destination operand, and sets the carry flag as the complement of borrow.</p> $(DEST) \leftarrow (SRC1) - (SRC2)$ <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>↑</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	✓	✓	↑	—	<p>DEST, SRC1, SRC2</p> <p>SUBB Dbreg, Sbreg, baop (010110aa) (baop) (Sbreg) (Dbreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	✓	✓	↑	—															
SUBC	<p>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.</p> $(DEST) \leftarrow (DEST) - (SRC) - (1-C)$ <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>↓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>↑</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	↓	✓	✓	✓	↑	—	<p>DEST, SRC</p> <p>SUBC wreg, waop (101010aa) (waop) (wreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
↓	✓	✓	✓	↑	—															
SUBCB	<p>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.</p> $(DEST) \leftarrow (DEST) - (SRC) - (1-C)$ <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>↓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>↑</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	↓	✓	✓	✓	↑	—	<p>DEST, SRC</p> <p>SUBCB breg, baop (101110aa) (baop) (breg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
↓	✓	✓	✓	↑	—															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
TIJMP	<p>TABLE INDIRECT JUMP. Causes execution to continue at an address selected from a table of addresses.</p> <p>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.</p> <p>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.</p> <p>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).</p> <p>[INDEX] AND #MASK = OFFSET $(2 \times \text{OFFSET}) + \text{TBASE} = \text{DEST X}$ $\text{PC} \leftarrow (\text{DEST X})$</p> <table border="1" data-bbox="325 1013 645 1111"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>TIJMP TBASE, [INDEX], #MASK (11100010) [INDEX] (#MASK) (TBASE)</p> <p>NOTE: TIJMP multiplies OFFSET by two to provide for word alignment of the jump table.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															
TRAP	<p>SOFTWARE TRAP. This instruction causes an interrupt call that is vectored through location 2010H. 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.</p> <p>$\text{SP} \leftarrow \text{SP} - 2$ $(\text{SP}) \leftarrow \text{PC}$ $\text{PC} \leftarrow (2010\text{H})$</p> <table border="1" data-bbox="325 1430 645 1527"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>TRAP (11110111)</p> <p>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.</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format																		
XCH	<p>EXCHANGE WORD. Exchanges the value of the source word operand with that of the destination word operand. (DEST) ↔ (SRC)</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>DEST, SRC</p> <p>XCH wreg, waop (00000100) (waop) (wreg) direct (00001011) (waop) (wreg) indexed</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															
XCHB	<p>EXCHANGE BYTE. Exchanges the value of the source byte operand with that of the destination byte operand. (DEST) ↔ (SRC)</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	—	—	—	—	—	—	<p>DEST, SRC</p> <p>XCHB breg, baop (00010100) (baop) (breg) direct (00011011) (baop) (breg) indexed</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
—	—	—	—	—	—															
XOR	<p>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)</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>0</td> <td>0</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	0	0	—	—	<p>DEST, SRC</p> <p>XOR wreg, waop (100001aa) (waop) (wreg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	0	0	—	—															
XORB	<p>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)</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <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> </thead> <tbody> <tr> <td>✓</td> <td>✓</td> <td>0</td> <td>0</td> <td>—</td> <td>—</td> </tr> </tbody> </table>	PSW Flag Settings						Z	N	C	V	VT	ST	✓	✓	0	0	—	—	<p>DEST, SRC</p> <p>XORB breg, baop (100101aa) (baop) (breg)</p>
PSW Flag Settings																				
Z	N	C	V	VT	ST															
✓	✓	0	0	—	—															

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–1F	Reserved
20–27	SJMP
28–2F	SCALL
30–37	JBC
38–3F	JBS
40	AND Direct (3 ops)
41	AND Immediate (3 ops)
42	AND Indirect (3 ops)
43	AND Indexed (3 ops)
44	ADD Direct (3 ops)

Table A-7. Instruction Opcodes (Continued)

Hex Code	Instruction Mnemonic
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	MULU Direct (3 ops)
4D	MULU Immediate (3 ops)
4E	MULU Indirect (3 ops)
4F	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)
6B	SUB Indexed (2 ops)
6C	MULU Direct (2 ops)
6D	MULU Immediate (2 ops)

Table A-7. Instruction Opcodes (Continued)

Hex Code	Instruction Mnemonic
6E	MULU Indirect (2 ops)
6F	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
94	XORB Direct
95	XORB Immediate
96	XORB Indirect

Table A-7. Instruction Opcodes (Continued)

Hex Code	Instruction Mnemonic
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
BD	LDBSE Immediate
BE	LDBSE Indirect
BF	LDBSE Indexed

Table A-7. Instruction Opcodes (Continued)

Hex Code	Instruction Mnemonic
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
E4–EB	Reserved
EC	DPTS
ED	EPTS
EE	Reserved (Note 1)
EF	LCALL

Table A-7. Instruction Opcodes (Continued)

Hex Code	Instruction Mnemonic
F0	RET
F2	PUSHF
F3	POPF
F4	PUSHA
F5	POPA
F6	IDLDP
F7	TRAP
F8	CLRC
F9	SETC
FA	DI
FB	EI
FC	CLRVT
FD	NOP
FE	DIV/DIVB/MUL/MULB (Note 2)
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 their lengths and opcodes for each applicable addressing mode. A dash (—) in any column indicates “not applicable.”

Table A-8. Instruction Lengths and Hexadecimal Opcodes

Arithmetic (Group I)								
Mnemonic	Direct		Immediate		Indirect		Indexed (Note 1)	
	Length	Opcode	Length	Opcode	Length	Opcode	Length 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 lengths as *S/L*, where *S* is the short-indexed instruction length and *L* is the long-indexed instruction length.
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. Instruction Lengths and Hexadecimal Opcodes (Continued)

Arithmetic (Group II)								
Mnemonic	Direct		Immediate		Indirect		Indexed (Note 1)	
	Length	Opcode	Length	Opcode	Length	Opcode	Length 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 (Note 1)	
	Length	Opcode	Length	Opcode	Length	Opcode	Length 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:

- For indexed instructions, the first column lists instruction lengths as *S/L*, where *S* is the short-indexed instruction length and *L* is the long-indexed instruction length.
- 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. Instruction Lengths and Hexadecimal Opcodes (Continued)

Stack								
Mnemonic	Direct		Immediate		Indirect		Indexed (Note 1)	
	Length	Opcode	Length	Opcode	Length	Opcode	Length 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		Indirect		Indexed (Note 1)	
	Length	Opcode	Length	Opcode	Length	Opcode	Length 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
Jump								
Mnemonic	Direct		Immediate		Indirect		Indexed (Note 1)	
	Length	Opcode	Length	Opcode	Length	Opcode	Length S/L	Opcode
BR	—	—	—	—	2	E3	—	—
LJMP	—	—	—	—	—	—	—/3	E7
SJMP (Note 2)	—	—	—	—	—	—	2/—	20–27
TIJMP	4	E2	4	E2	—	—	—/4	E2

NOTES:

- For indexed instructions, the first column lists instruction lengths as *S/L*, where *S* is the short-indexed instruction length and *L* is the long-indexed instruction length.
- 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. Instruction Lengths and Hexadecimal Opcodes (Continued)

Call								
Mnemonic	Direct		Immediate		Indirect		Indexed (Note 1)	
	Length	Opcode	Length	Opcode	Length	Opcode	Length	Opcode
LCALL	—	—	—	—	—	—	3	EF
RET	—	—	—	—	1	F0	—	—
SCALL (Note 2)	—	—	—	—	—	—	2	28–2F
TRAP	1	F7	—	—	—	—	—	—
Conditional Jump								
Mnemonic	Direct		Immediate		Indirect		Indexed (Note 1)	
	Length	Opcode	Length	Opcode	Length	Opcode	Length 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:

- For indexed instructions, the first column lists instruction lengths as *S/L*, where *S* is the short-indexed instruction length and *L* is the long-indexed instruction length.
- 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. Instruction Lengths and Hexadecimal OpCodes (Continued)

Shift								
Mnemonic	Direct		Immediate		Indirect		Indexed	
	Length	Opcode	Length	Opcode	Length	Opcode	Length	Opcode
NORML	3	0F	—	—	—	—	—	—
SHL	3	09	—	—	—	—	—	—
SHLB	3	19	—	—	—	—	—	—
SHLL	3	0D	—	—	—	—	—	—
SHR	3	08	—	—	—	—	—	—
SHRA	3	0A	—	—	—	—	—	—
SHRAB	3	1A	—	—	—	—	—	—
SHRAL	3	0E	—	—	—	—	—	—
SHRB	3	18	—	—	—	—	—	—
SHRL	3	0C	—	—	—	—	—	—
Special								
Mnemonic	Direct		Immediate		Indirect		Indexed	
	Length	Opcode	Length	Opcode	Length	Opcode	Length	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	
	Length	Opcode	Length	Opcode	Length	Opcode	Length	Opcode
DPTS	1	EC	—	—	—	—	—	—
EPTS	1	ED	—	—	—	—	—	—

NOTES:

- For indexed instructions, the first column lists instruction lengths as *S/L*, where *S* is the short-indexed instruction length and *L* is the long-indexed instruction length.
- 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. See Table 4-1 on page 4-2 for address information.

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
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

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 4-1 on page 4-2 for address information.

Table A-9. Instruction Execution Times (in State Times) (Continued)

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	—	—	—	—	—	—	—	—	—
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 4-1 on page 4-2 for address information.

Table A-9. Instruction Execution Times (in State Times) (Continued)

Data										
Mnemonic	Indirect									
BMOV	register/register	6 + 8 per word								
	memory/register	6 + 11 per word								
	memory/memory	6 + 14 per word								
BMOVI	register/register	7 + 8 per word + 14 per interrupt								
	memory/register	7 + 11 per word + 14 per interrupt								
	memory/memory	7 + 14 per word + 14 per interrupt								
Mnemonic	Direct	Immed.	Indirect				Indexed			
			Normal		Autoinc.		Short		Long	
			Reg.	Mem.	Reg.	Mem.	Reg.	Mem.	Reg.	Mem.
LD	4	5	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
Jump										
Mnemonic	Direct	Immed.	Indirect				Indexed			
			Normal		Autoinc.		Short		Long	
			Reg.	Mem.	Reg.	Mem.	Reg.	Mem.	Reg.	Mem.
BR	—	—	7		7		—		—	
LJMP	—	—	—		—		—		7	
SJMP	—	—	—		—		7		—	
TIJMP	—	—	15		—		—		—	
register/register			18		—		—		—	
memory/register			21		—		—		—	
memory/memory	—		—		—		—		—	
Call (Register)										
Mnemonic	Direct	Immed.	Indirect				Indexed			
			Normal		Autoinc.		Short		Long	
			Reg.	Mem.	Reg.	Mem.	Reg.	Mem.	Reg.	Mem.
LCALL	—	—	—		—		—		11	
RET	—	—	11		—		—		—	
SCALL	—	—	—		—		—		11	
TRAP	16	—	—		—		—		—	

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 4-1 on page 4-2 for address information.

Table A-9. Instruction Execution Times (in State Times) (Continued)

Call (Memory)						
Mnemonic	Direct	Immed.	Indirect		Indexed	
			Normal	Autoinc.	Short	Long
LCALL	—	—	—	—	—	13
RET	—	—	14	—	—	—
SCALL	—	—	—	—	—	13
TRAP	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)					

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 4-1 on page 4-2 for address information.

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	—	12	—	—	—	—
Valid key	—	28	—	—	—	—
Invalid key	—	—	—	—	—	—
NOP	2	—	—	—	—	—
RST	4	—	—	—	—	—
SETC	2	—	—	—	—	—
SKIP	3	—	—	—	—	—
PTS						
Mnemonic	Direct	Immed.	Indirect		Indexed	
			Normal	Autoinc.	Short	Long
DPTS	2	—	—	—	—	—
EPTS	2	—	—	—	—	—

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 4-1 on page 4-2 for address information.



B

Signal Descriptions

APPENDIX B

SIGNAL DESCRIPTIONS

This appendix provides reference information for the pin functions of the 8XC196MC, 8XC196MD, and 8XC196MH.

B.1 SIGNAL NAME CHANGES

The names of some 8XC196MC and 8XC196MD signals have been changed for consistency with other MCS[®] 96 microcontrollers. Table B-1 lists the old and new names.

Table B-1. Signal Name Changes

<i>Name in 8XC196MC User's Manual</i>	New Name
AGND	ANGND
CAPCOMP _x	EPA _x
COMPARE _x	COMP _x

B.2 FUNCTIONAL GROUPINGS OF SIGNALS

Tables B-2, B-3, and B-4 list the signals for the 8XC196MC, 8XC196MD, and 8XC196MH respectively, 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-2. 8XC196MC Signals Arranged by Functional Categories

Address & Data	Programming Control	Input/Output	Input/Output (Cont'd)
AD15:0	AINC#	P0.7:0/ACH7:0	P6.5/WG3
	CPVER	P1.0/ACH8	P6.6/PWM0
Bus Control & Status	PACT#	P1.1/ACH9	P6.7/PWM1
ALE/ADV#	PALE#	P1.2/ACH10/T1CLK	
BHE#/WRH#	PBUS.15:0	P1.3/ACH11/T1DIR	
BUSWIDTH	PMODE.3:0	P1.4/ACH12	
INST	PROG#	P2.3:0/EPA3:0	
READY	PVER	P2.6:4/COMP2:0	
RD#		P2.7/COMP3	
WR#/WRL#	Processor Control	P3.7:0	
	CLKOUT	P4.7:0	
Power & Ground	EA#	P5.1	
ANGND	EXTINT	P5.7:0	
V _{CC}	NMI	P6.0/WG1#	
V _{PP}	ONCE#	P6.1/WG1	
V _{REF}	RESET#	P6.2/WG2#	
V _{SS}	XTAL1	P6.3/WG2	
	XTAL2	P6.4/WG3#	

NOTE: The shaded signals are not available in the 64-pin package.

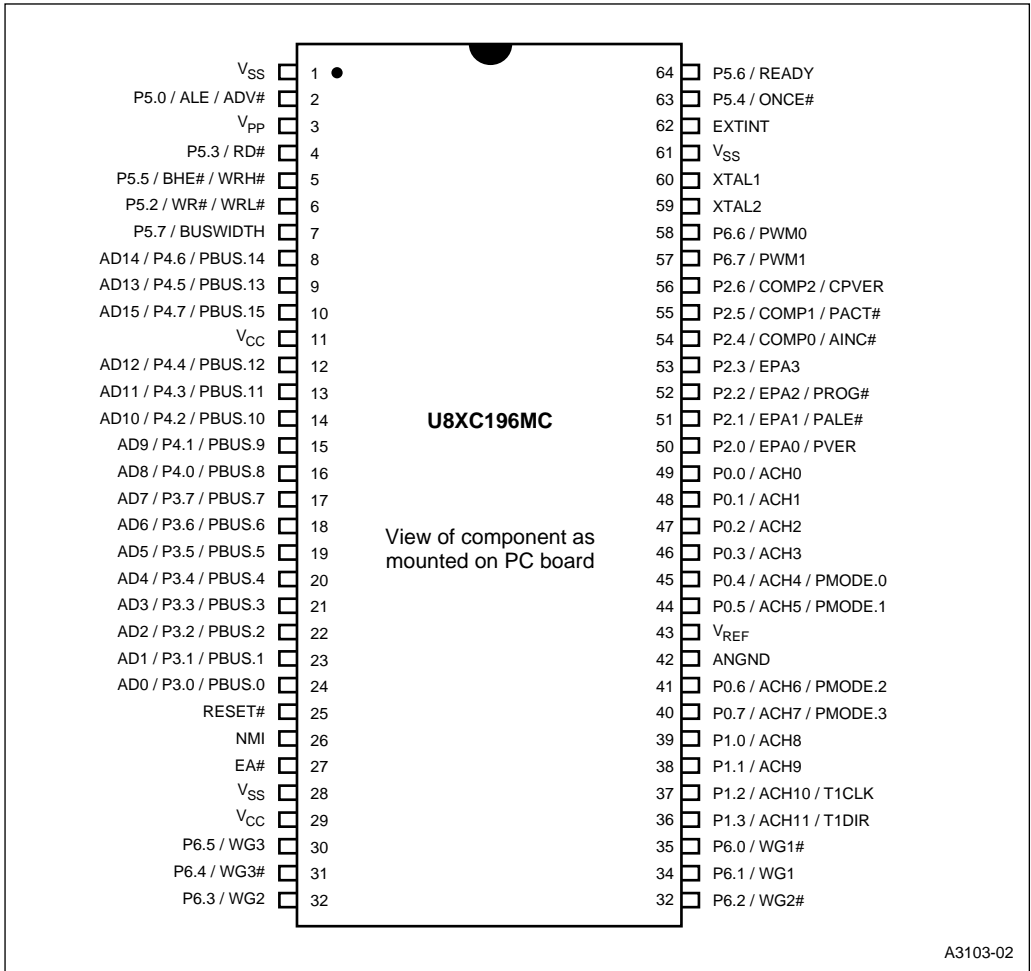


Figure B-1. 8XC196MC 64-lead Shrink DIP (SDIP) Package

A3103-02

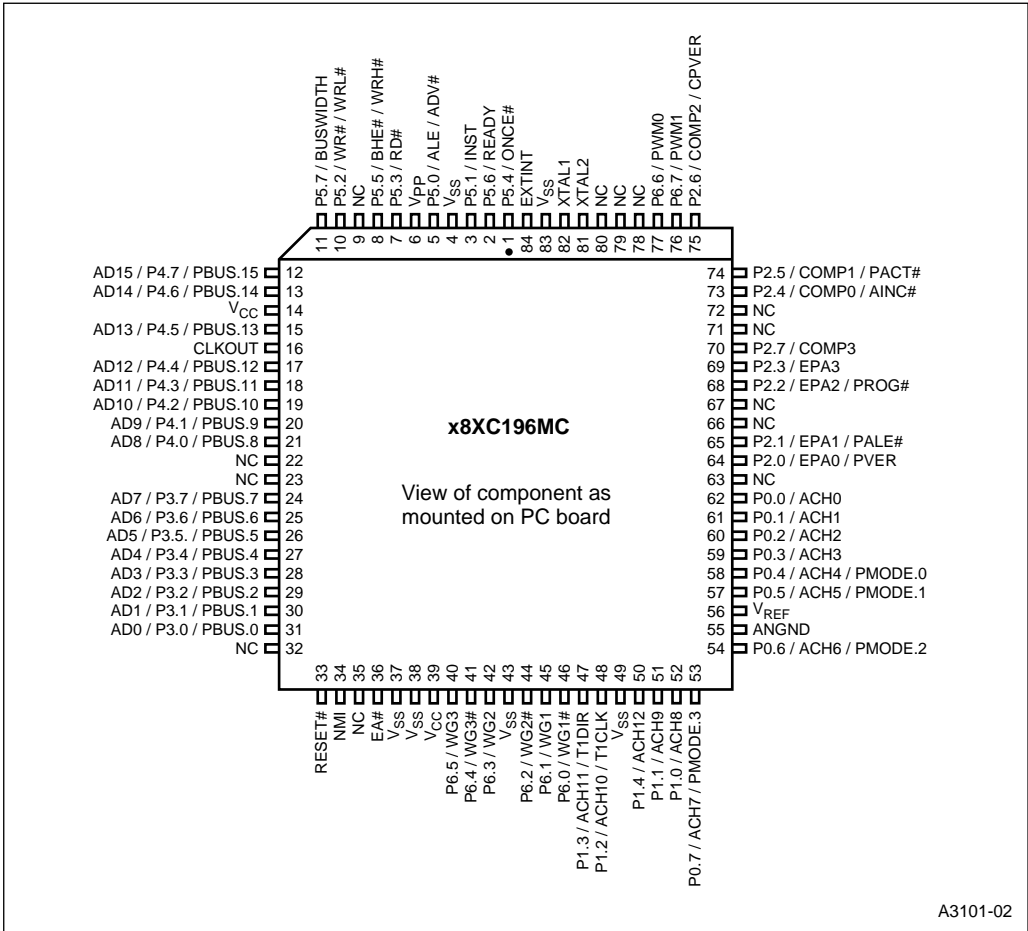


Figure B-2. 8XC196MC 84-lead PLCC Package

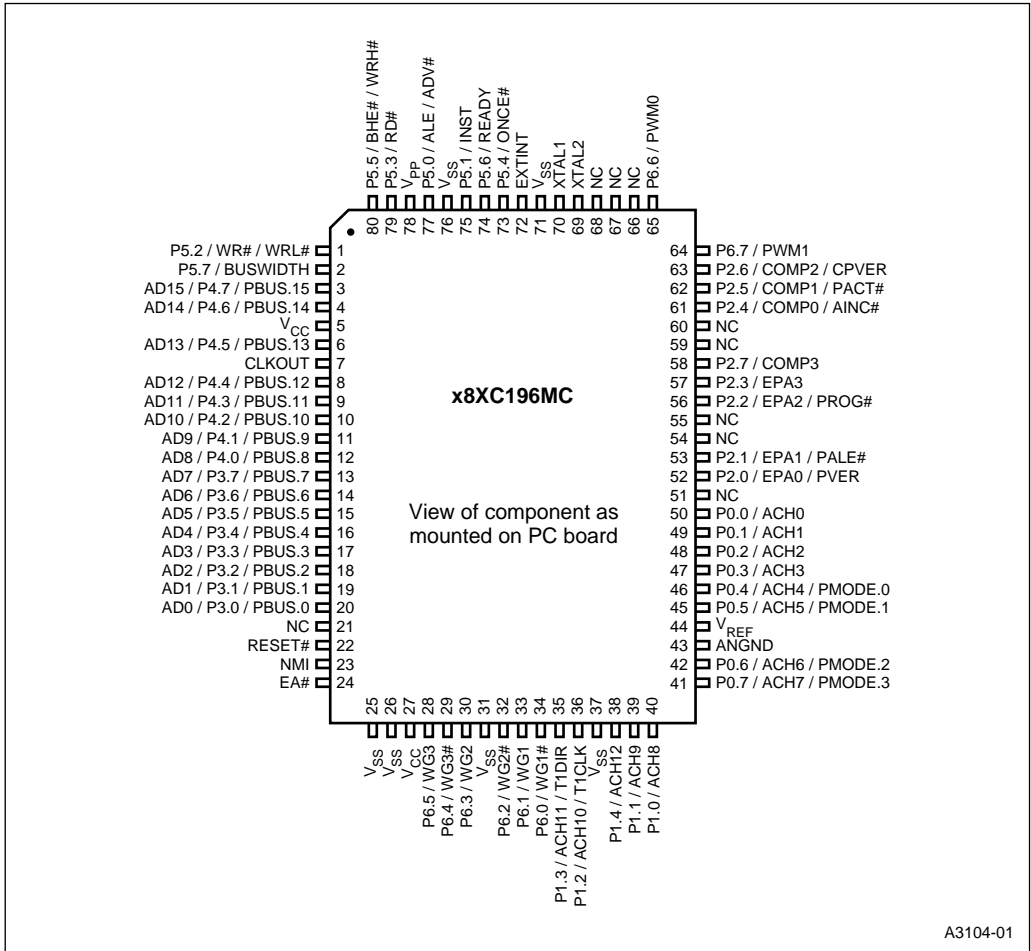


Figure B-3. 8XC196MC 80-lead Shrink EIAJ/QFP Package

A3104-01

Table B-3. 8XC196MD Signals Arranged by Functional Categories

Address & Data	Programming Control	Input/Output	Input/Output (Cont'd)
AD15:0	AINC#	P0.7:0/ACH7:0	P7.1:0/EPA5:4
	CPVER	P1.1:0/ACH9:8	P7.3:2/COMP5:4
Bus Control & Status	PACT#	P1.2/ACH10/T1CLK	P7.6:4
ALE/ADV#	PALE#	P1.3/ACH11/T1DIR	P7.7/FREQOUT
BHE#/WRH#	PBUS.15:0	P1.5:4/ACH13:12	
BUSWIDTH	PMODE.3:0	P1.7:6	
INST	PROG#	P2.3:0/EPA3:0	
READY	PVER	P2.7:4/COMP3:0	
RD#		P3.7:0	
WR#/WRL#	Processor Control	P4.7:0	
	CLKOUT	P5.7:0	
Power & Ground	EA#	P6.0/WG1#	
ANGND	EXTINT	P6.1/WG1	
V _{CC}	NMI	P6.2/WG2#	
V _{PP}	ONCE#	P6.3/WG2	
V _{REF}	RESET#	P6.4/WG3#	
V _{SS}	XTAL1	P6.5/WG3	
	XTAL2	P6.7:6/PWM1:0	

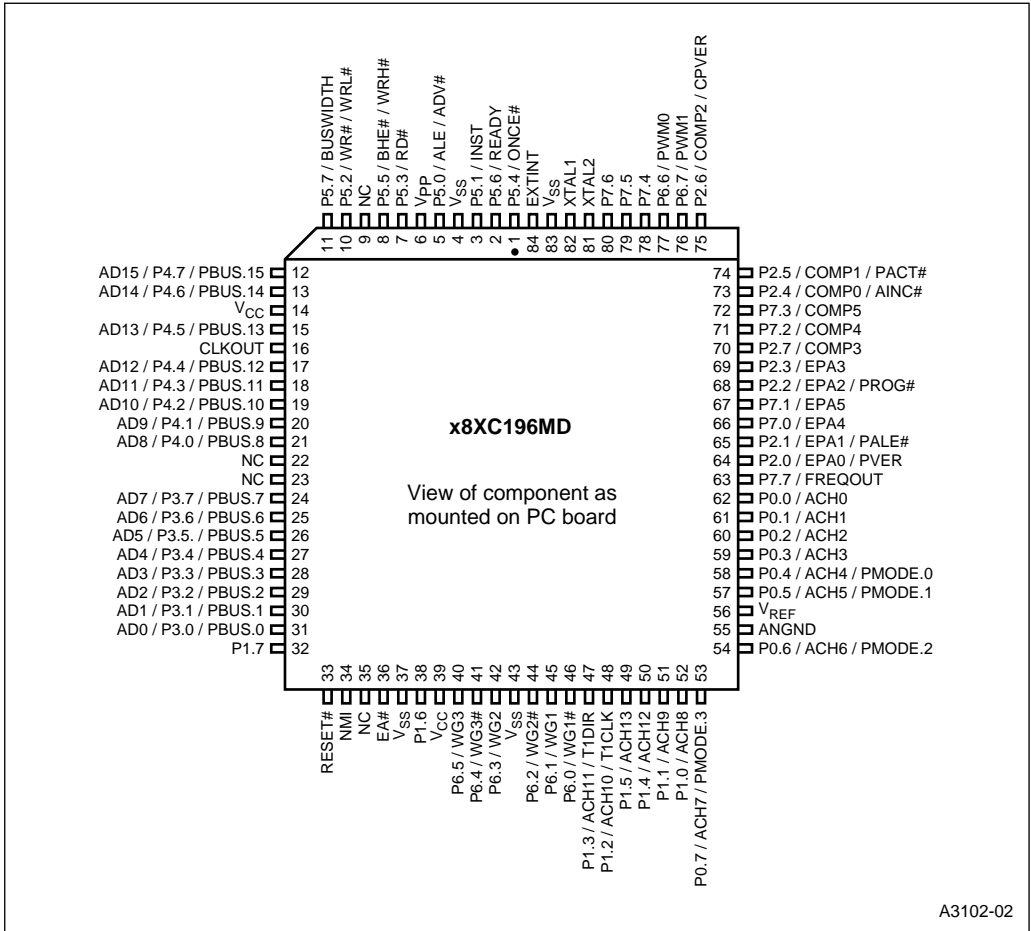


Figure B-4. 8XC196MD 84-lead PLCC Package

A3102-02

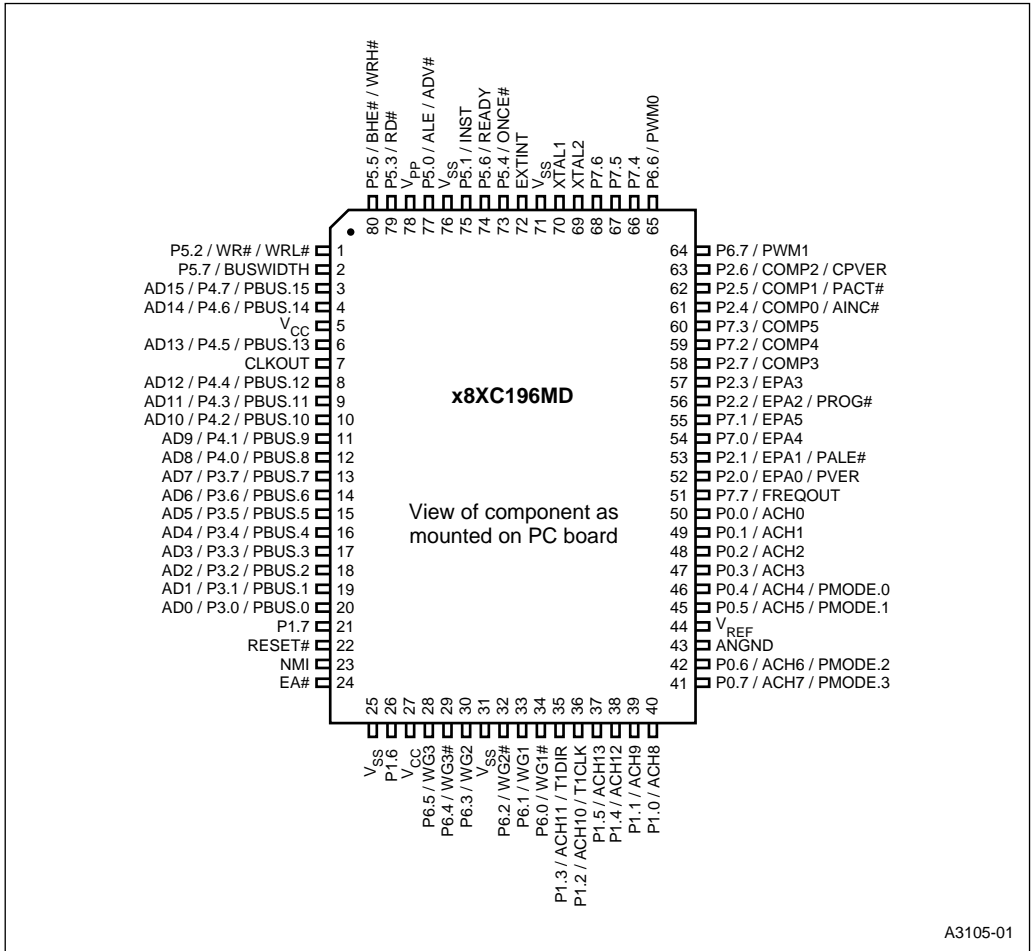


Figure B-5. 8XC196MD 80-lead Shrink EIAJ/QFP Package

A3105-01

Table B-4. 8XC196MH Signals Arranged by Functional Categories

Address & Data	Programming Control	Input/Output	Input/Output (Cont'd)
AD15:0	AINC#	P0.5:0/ACH5:0	P5.0
	CPVER	P0.6/ACH6/T1CLK	P5.1
Bus Control & Status	PACT#	P0.7/ACH7/T1DIR	P5.7:2
ALE/ADV#	PALE#	P1.0/TXD0	P6.0/WG1#
BHE#/WRH#	PBUS.15:0	P1.1/RXD0	P6.1/WG1
BUSWIDTH	PMODE.3:0	P1.2/TXD1	P6.2/WG2#
INST	PROG#	P1.3/RXD1	P6.3/WG2
READY	PVER	P2.0/EPA0	P6.4/WG3#
RD#		P2.1/SCLK0#/BCLK0	P6.5/WG3
WR#/WRL#	Processor Control	P2.2/EPA1	P6.6/PWM0
	EA#	P2.3/COMP3	P6.7/PWM1
Power & Ground	EXTINT	P2.4/COMP0	
ANGND	NMI	P2.5/COMP1	
V _{CC}	ONCE#	P2.6/COMP2	
V _{PP}	RESET#	P2.7/SCLK1#/BCLK1	
V _{REF}	XTAL1	P3.7:0	
V _{SS}	XTAL2	P4.7:0	

NOTE: The shaded signals are not available in the 64-pin package.

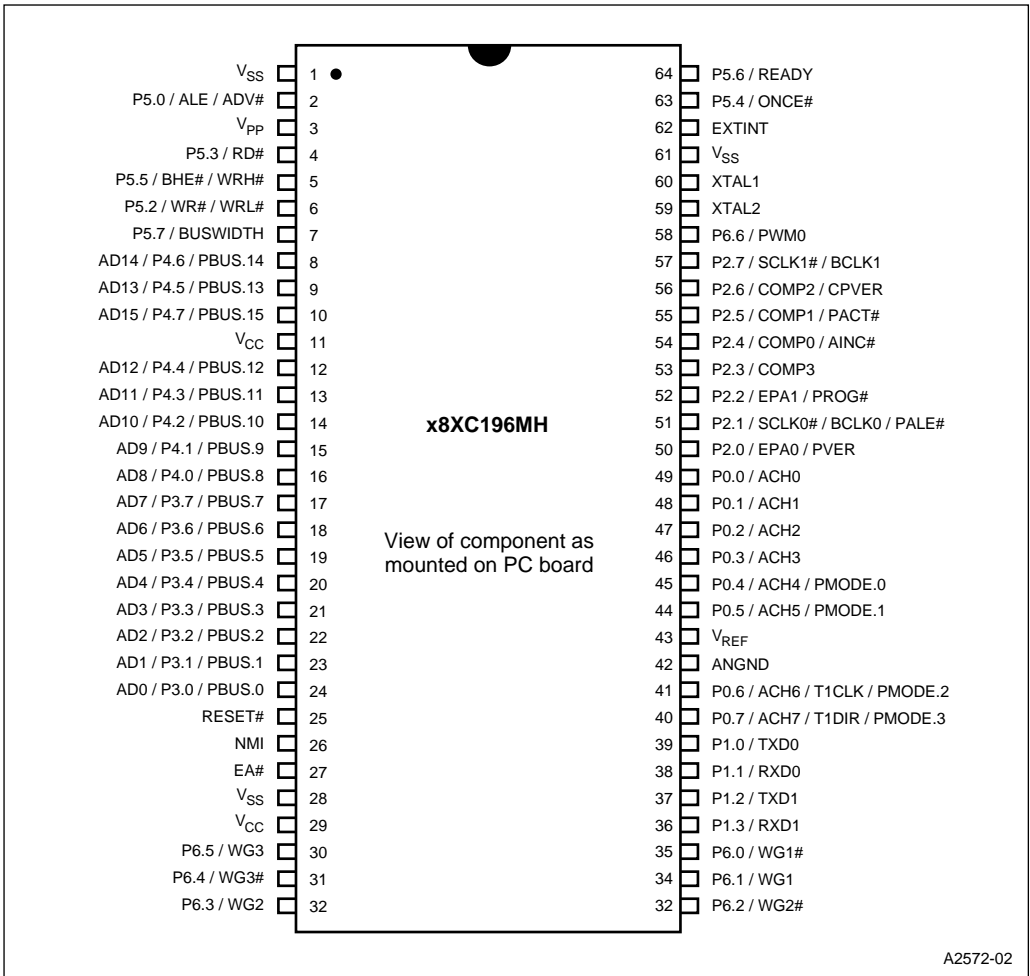
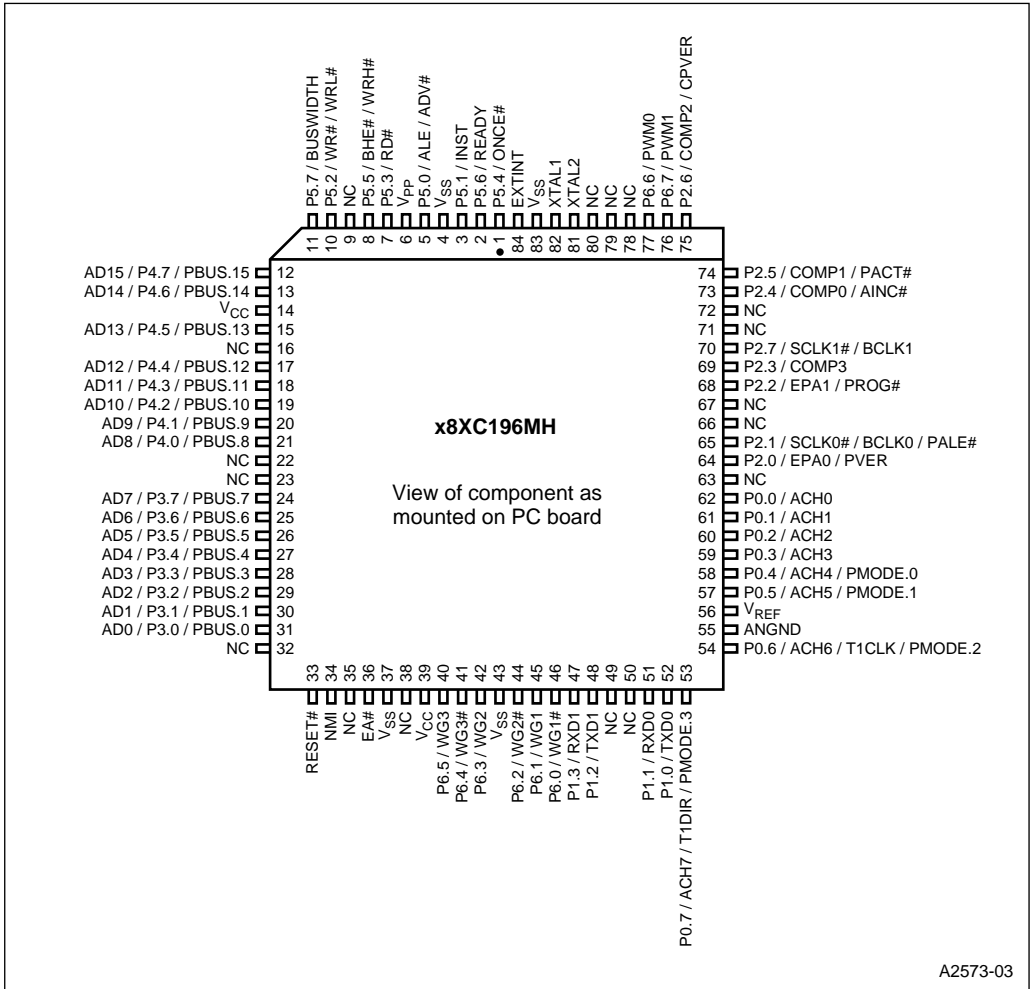
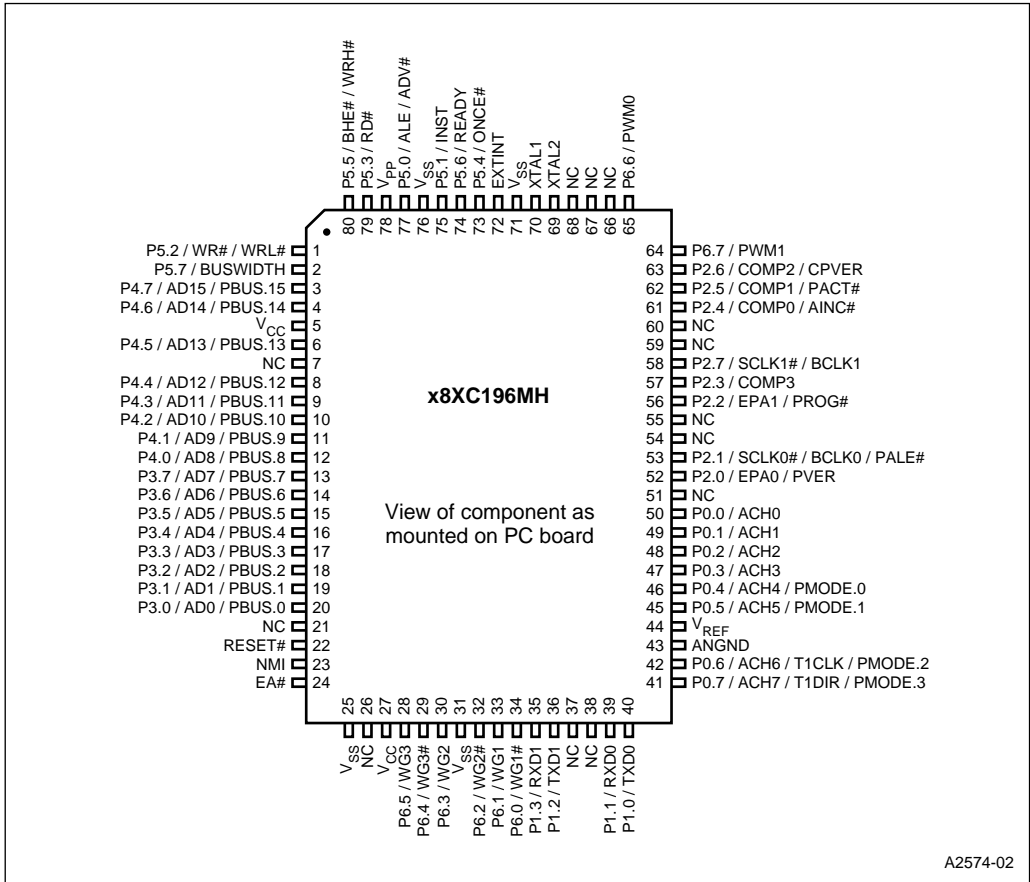


Figure B-6. 8XC196MH 64-lead Shrink DIP (SDIP) Package



A2573-03

Figure B-7. 8XC196MH 84-lead PLCC Package



A2574-02

Figure B-8. 8XC196MH 80-lead Shrink EIAJ/QFP Package

B.3 SIGNAL DESCRIPTIONS

Table B-5 defines the columns used in Table B-6, which describes the signals.

Table B-5. Description of Columns of Table B-6

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 <i>Name</i> column as an input (I), output (O), bidirectional (I/O), power (PWR), or ground (GND). Note that all inputs except RESET# are <i>sampled inputs</i> . 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 <i>Name</i> column. Also lists any alternate functions that are multiplexed with the signal.

Table B-6. Signal Descriptions

Name	Type	Description
ACH12:0 (MC) ACH13:0 (MD) ACH7:0 (MH)	I	<p>Analog Channels</p> <p>These pins are analog inputs to the A/D converter.</p> <p>These pins may individually be used as analog inputs (ACHx) or digital inputs (P0.y). While it is possible for the pins to function simultaneously as analog and digital inputs, this is not recommended because reading port 0 while a conversion is in process can produce unreliable conversion results.</p> <p>The ANGND and V_{REF} pins must be connected for the A/D converter and port 0 to function.</p> <p>ACH7:0 are multiplexed as follows: ACH0/P0.0, ACH1/P0.1, ACH2/P0.2, ACH3/P0.3, ACH4/P0.4/PMODE.0, ACH5/P0.5/PMODE.1, ACH6/P0.6/PMODE.2, ACH7/P0.7/PMODE.3, ACH8/P1.0, ACH9/P1.1, ACH10/P1.2/T1CLK, ACH11/P1.3/T1DIR, and ACH12/P1.4, and ACH13/P1.5.</p> <p>ACH13 is not implemented on the 8XC196MC and ACH13:8 are not implemented on the 8XC196MH.</p>
AD15:0	I/O	<p>Address/Data Lines</p> <p>These pins provide a multiplexed address and data bus. During the address phase of the bus cycle, address bits 0–15 are presented on the bus and can be latched using ALE or ADV#. During the data phase, 8- or 16-bit data is transferred.</p> <p>AD7:0 are multiplexed with P3.7:0, and PBUS.7:0. AD15:8 are multiplexed with P4.7:0 and PBUS.15:8.</p>
ADV#	O	<p>Address Valid</p> <p>This active-low output signal is asserted only during external memory accesses. ADV# indicates that valid address information is available on the system address/data bus. The signal remains low while a valid bus cycle is in progress and is returned high as soon as the bus cycle completes.</p> <p>An external latch can use this signal to demultiplex the address from the address/data bus. A decoder can also use this signal to generate chip selects for external memory.</p> <p>ADV# is multiplexed with P5.0 and ALE.</p>

Table B-6. Signal Descriptions (Continued)

Name	Type	Description												
AINC#	I	<p>Auto Increment</p> <p>During slave programming, this active-low input enables the auto-increment feature. (Auto increment allows reading or writing of sequential OTPROM locations, without requiring address transactions across the PBUS for each read or write.) AINC# is sampled after each location is programmed or dumped. If AINC# is asserted, the address is incremented and the next data word is programmed or dumped.</p> <p>AINC# is multiplexed with P2.4 and COMP0.</p>												
ALE	O	<p>Address Latch Enable</p> <p>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. ALE differs from ADV# in that it does not remain active during the entire bus cycle.</p> <p>An external latch can use this signal to demultiplex the address from the address/data bus.</p> <p>ALE is multiplexed with P5.0 and ADV#.</p>												
ANGND	GND	<p>Analog Ground</p> <p>ANGND must be connected for A/D converter and port 0 operation (also Port 1 on the 8XC196MC and MD). ANGND and V_{SS} should be nominally at the same potential.</p>												
BCLK1:0 (MH only)	I	<p>Baud Clock 0 and 1</p> <p>BCLK0 and 1 are alternate clock sources for the baud-rate generator input. The maximum input frequency is $F_{XTAL1}/4$.</p> <p>BCLK0 is multiplexed with P2.1, SCLK0#, and PALE#. BCLK1 is multiplexed with P2.7 and SCLK1#.</p>												
BHE#	O	<p>Byte High Enable[†]</p> <p>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 AD0, to determine which memory byte is being transferred over the system bus:</p> <table border="1"> <thead> <tr> <th>BHE#</th> <th>AD0</th> <th>Byte(s) Accessed</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>both bytes</td> </tr> <tr> <td>0</td> <td>1</td> <td>high byte only</td> </tr> <tr> <td>1</td> <td>0</td> <td>low byte only</td> </tr> </tbody> </table> <p>BHE# is multiplexed with P5.5 and WRH#.</p> <p>[†] The chip configuration register 0 (CCR0) determines whether this pin functions as BHE# or WRH#. CCR0.2 = 1 selects BHE#; CCR0.2 = 0 selects WRH#.</p>	BHE#	AD0	Byte(s) Accessed	0	0	both bytes	0	1	high byte only	1	0	low byte only
BHE#	AD0	Byte(s) Accessed												
0	0	both bytes												
0	1	high byte only												
1	0	low byte only												

Table B-6. Signal Descriptions (Continued)

Name	Type	Description																				
BUSWIDTH	I	<p>Bus Width</p> <p>Two chip configuration register bits, CCR0.1 and CCR1.2, along with the BUSWIDTH pin, control the data bus width. When both CCR bits are set, the BUSWIDTH signal selects the external data bus width. When only one CCR bit is set, the bus width is fixed at either 16 or 8 bits, and the BUSWIDTH signal has no effect.</p> <table border="1" data-bbox="379 430 899 548"> <thead> <tr> <th>CCR0.1</th> <th>CCR1.2</th> <th>BUSWIDTH</th> <th></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1</td> <td>X</td> <td>fixed 8-bit data bus</td> </tr> <tr> <td>1</td> <td>0</td> <td>X</td> <td>fixed 16-bit data bus</td> </tr> <tr> <td>1</td> <td>1</td> <td>high</td> <td>16-bit data bus</td> </tr> <tr> <td>1</td> <td>1</td> <td>low</td> <td>8-bit data bus</td> </tr> </tbody> </table> <p>BUSWIDTH is multiplexed with P5.7.</p>	CCR0.1	CCR1.2	BUSWIDTH		0	1	X	fixed 8-bit data bus	1	0	X	fixed 16-bit data bus	1	1	high	16-bit data bus	1	1	low	8-bit data bus
CCR0.1	CCR1.2	BUSWIDTH																				
0	1	X	fixed 8-bit data bus																			
1	0	X	fixed 16-bit data bus																			
1	1	high	16-bit data bus																			
1	1	low	8-bit data bus																			
CLKOUT (MC, MD)	O	<p>Clock Output</p> <p>Output of the internal clock generator. The CLKOUT frequency is ½ the oscillator input frequency (F_{XTAL1}). CLKOUT has a 50% duty cycle. CLKOUT is not implemented on the 8XC196MH.</p>																				
COMP3:0 (MC, MH) COMP5:0 (MD)	O	<p>Event Processor Array (EPA) Compare Pins</p> <p>These signals are the output of the EPA compare-only channels. These pins are multiplexed with other signals and may be configured as standard I/O. COMP5:0 are multiplexed as follows: COMP0/P2.4/AINC#, COMP1/P2.5/PACT#, COMP2/P2.6/CPVER, COMP3/P2.7 (MC, MD), COMP3/P2.3 (MH), COMP4/P7.2, and COMP5/P7.3. COMP4 and COMP5 are not implemented on the 8XC196MC and MH.</p>																				
CPVER	O	<p>Cumulative Program Verification</p> <p>During slave programming, a high signal indicates that all locations programmed correctly, while a low signal indicates that an error occurred during one of the programming operations. CPVER is multiplexed with P2.6 and COMP2.</p>																				
EA#	I	<p>External Access</p> <p>This input determines whether memory accesses to special-purpose and program memory partitions are directed to internal or external memory. (See Table 4-1 on page 4-2 for address ranges of special-purpose and program memory partitions.) 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.</p> <p>EA# also controls entry into the programming modes. If EA# is at V_{PP} voltage (typically +12.5 V) on the rising edge of RESET#, the microcontroller enters a programming mode.</p> <p>NOTE: Systems with EA# tied inactive have idle time between external bus cycles. When the address/data bus is idle, you can use ports 3 and 4 for I/O. Systems with EA# tied active cannot use ports 3 and 4 as standard I/O; when EA# is active, these ports will function only as the address/data bus.</p> <p>EA# is sampled and latched only on the rising edge of RESET#. Changing the level of EA# after reset has no effect.</p> <p>Always connect EA# to V_{SS} when using a microcontroller that has no internal nonvolatile memory.</p>																				

Table B-6. Signal Descriptions (Continued)

Name	Type	Description
EPA3:0 (MC) EPA5:0 (MD) EPA1:0 (MH)	I/O	Event Processor Array (EPA) Capture/Compare Channels High-speed input/output signals for the EPA capture/compare channels. EPA5:0 are multiplexed as follows: EPA0/P2.0/PVER, EPA1/P2.1/PALE# (MC, MD), EPA1/P2.2/PROG# (MH), EPA2/P2.2/PROG#, EPA3/P2.3, EPA4/P7.0, and EPA5/P7.1. EPA5:4 are not implemented on the 8XC196MC and EPA5:2 are not implemented on the 8XC196MH.
EXTINT	I	External Interrupt This programmable interrupt is controlled by the WG_PROTECT register. This register controls whether the interrupt is edge triggered or sampled and whether a rising edge/high level or falling edge/low level activates the interrupt. In powerdown mode, asserting the EXTINT signal for at least 50 ns causes the device to resume normal operation. The interrupt need not be enabled. 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.
FREQOUT (MD only)	O	Frequency Generator Output A fixed 50% duty cycle waveform that can vary in frequency from 4 KHz to 1 MHz (assuming $F_{XTAL1} = 16$ MHz). FREQOUT is multiplexed with P7.7. FREQOUT is not implemented on the 8XC196MC or MH.
INST	O	Instruction Fetch This active-high output signal is valid only during external memory bus cycles. 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 is multiplexed 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.

Table B-6. Signal Descriptions (Continued)

Name	Type	Description
ONCE#	I	<p>On-circuit Emulation</p> <p>Holding ONCE# low during the rising edge of RESET# places the device into on-circuit emulation (ONCE) mode. This mode puts all pins, except XTAL1 and XTAL2, into a high-impedance state, thereby isolating the device 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.</p> <p>To exit ONCE mode, reset the device 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 (see datasheet).</p> <p>ONCE# is multiplexed with P5.4.</p>
P0.7:0	I	<p>Port 0</p> <p>This is a high-impedance, input-only port. Port 0 pins should not be left floating. These pins may individually be used as analog inputs (ACHx) or digital inputs (P0.y). While it is possible for the pins to function simultaneously as analog and digital inputs, this is not recommended because reading port 0 while a conversion is in process can produce unreliable conversion results.</p> <p>ANGND and V_{REF} must be connected for port 0 to function.</p> <p>Port 0 is multiplexed as follows: P0.0/ACH0, P0.1/ACH1, P0.2/ACH2, P0.3/ACH3, P0.4/ACH4/PMODE.0, P0.5/ACH5/PMODE.1, P0.6/ACH6/PMODE.2 (MC, MD), P0.6/ACH6/T1CLK/PMODE.2 (MH), P0.7/ACH7/PMODE.3 (MC, MD), and P0.7/ACH7/T1DIR/PMODE.3 (MH).</p>
P1.4:0 (MC) P1.7:0 (MD)	I	<p>Port 1</p> <p>This is a high-impedance, input-only port.</p> <p>On the 8XC196MC and MD, some port 1 pins may individually be used as analog inputs (ACHx) or digital inputs (P1.y). While it is possible for the pins to function simultaneously as analog and digital inputs, this is not recommended because reading port 1 while a conversion is in process can produce unreliable conversion results.</p> <p>ANGND and V_{REF} must be connected for port 1 to function.</p> <p>Port 1 is multiplexed as follows: P1.0/ACH8, P1.1/ACH9, P1.2/ACH10/T1CLK, P1.3/ACH11/T1DIR, P1.4/ACH12, and P1.5/ACH13).</p> <p>P1.6 and P1.7 are not multiplexed with any other signals. P1.7:5 are not implemented on the 8XC196MC.</p>
P1.3:0 (MH)	I/O	<p>Port 1</p> <p>This is a standard, bidirectional port that is multiplexed with individually selectable special-function signals.</p> <p>On the 8XC196MH, port 1 is multiplexed as follows: P1.0/ TXD0, P1.1/RXD0, P1.2/TXD1, and P1.3/RXD1.</p>

Table B-6. Signal Descriptions (Continued)

Name	Type	Description
P2.7:0	I/O	<p>Port 2</p> <p>This is a standard, 8-bit, bidirectional port that is multiplexed with individually selectable special-function signals.</p> <p>P2.6 is multiplexed with a special test-mode-entry function. If this pin is held low during reset, the device will enter a reserved test mode, so exercise caution if you use this pin for input. If you choose to configure this pin as an input, always hold it high during reset and ensure that your system meets the V_{IH} specification (see datasheet) to prevent inadvertent entry into ONCE mode.</p> <p>On the 8XC196MC and MD, port 2 is multiplexed as follows: P2.0/EPA0/PVER, P2.1/EPA1/PALE#, P2.2/EPA2/PROG#, P2.3/EPA3, P2.4/COMP0/AINC#, P2.5/COMP1/PACT#, P2.6/COMP2/CPVER, and P2.7/COMP3.</p> <p>On the 8XC196MH, port 2 is multiplexed as follows: P2.0/EPA0/PVER, P2.1/SCLK0#/BCLK0/PALE#, P2.2/EPA1/PROG#, P2.3/COMP3, P2.4/COMP0/AINC#, P2.5/COMP1/PACT#, P2.6/COMP2/CPVER, and P2.7/SCLK1#/BCLK1.</p>
P3.7:0	I/O	<p>Port 3</p> <p>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.</p> <p>P3.7:0 are multiplexed with AD7:0 and PBUS.7:0.</p>
P4.7:0	I/O	<p>Port 4</p> <p>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.</p> <p>P4.7:0 are multiplexed with AD15:8 and PBUS15:8.</p>
P5.7:0	I/O	<p>Port 5</p> <p>This is a memory-mapped, 8-bit, bidirectional port that is multiplexed with individually selectable control signals.</p> <p>P5.4 is multiplexed with the ONCE# function. If this pin is held low during reset, the device will enter ONCE mode, so exercise caution if you use this pin for input. If you choose to configure this pin as an input, always hold it high during reset and ensure that your system meets the V_{IH} specification (see datasheet) to prevent inadvertent entry into ONCE mode.</p> <p>Port 5 is multiplexed as follows: P5.0/ALE/ADV#, P5.1/INST, P5.2/WR#/WRL#, P5.3/RD#, P5.4/ONCE#, P5.5/BHE#/WRH#, P5.6/READY, and P5.7/BUSWIDTH.</p>
P6.7:0	O	<p>Port 6</p> <p>This is a standard, 8-bit, output-only port that is multiplexed with the special functions of the waveform generator and PWM peripherals. The WG_OUT register configures the pins, establishes the output polarity, and controls whether changes to the outputs are synchronized with an event or take effect immediately.</p> <p>Port 6 is multiplexed as follows: P6.0/WG1#, P6.1/WG1, P6.2/WG2#, P6.3/WG2, P6.4/WG3#, P6.5/WG3, P6.6/PWM0, and P6.7/PWM1.</p>

Table B-6. Signal Descriptions (Continued)

Name	Type	Description
P7.7:0 (MD only)	I/O	<p>Port 7</p> <p>This is a standard, 8-bit, bidirectional port with Schmitt-trigger inputs.</p> <p>Port 7 is multiplexed as follows: P7.0/EPA4, P7.1/EPA5, P7.2/COMP4, P7.3/COMP5, and P7.7/FREQOUT. P7.6:4 are not multiplexed.</p> <p>Port 7 is not implemented on the 8XC196MC and 8XC196MH.</p>
PACT#	O	<p>Programming Active</p> <p>During auto programming or ROM-dump, a low signal indicates that programming or dumping is in progress, while a high signal indicates that the operation is complete.</p> <p>PACT# is multiplexed with P2.5 and COMP1.</p>
PALE#	I	<p>Programming ALE</p> <p>During slave programming, a falling edge causes the device to read a command and address from the PBUS.</p> <p>PALE# is multiplexed with P2.1 and EPA1 (MC, MD) and P2.1, SCLK0#, and BCLK0 (MH).</p>
PBUS.15:0	I/O	<p>Address/Command/Data Bus</p> <p>During slave programming, ports 3 and 4 serve as a bidirectional port with open-drain outputs to pass commands, addresses, and data to or from the device. Slave programming requires external pull-up resistors.</p> <p>During auto programming and ROM-dump, ports 3 and 4 serve as a regular system bus to access external memory. P4.6 and P4.7 are left unconnected; P1.1 and P1.2 serve as the upper address lines.</p> <p>Slave programming:</p> <p>PBUS.7:0 are multiplexed with AD7:0 and P3.7:0.</p> <p>PBUS.15:8 are multiplexed with AD15:8 and P4.7:0.</p> <p>Auto programming:</p> <p>On the 8XC196MC, MC, and MH, PBUS.7:0 are multiplexed with AD7:0 and P3.7:0.</p> <p>On the 8XC196MC and MD, PBUS.13:8 are multiplexed with AD13:8 and P4.5:0; PBUS15:14 are multiplexed with P1.2:1.</p> <p>On the 8XC196MH, PBUS.11:8 are multiplexed with AD11:8 and P4.3:0; PBUS15:12 are multiplexed with P1.3:0.</p>
PMODE.3:0	I	<p>Programming Mode Select</p> <p>Determines the programming mode. PMODE is sampled after a device reset and must be static while the microcontroller is operating. (Table 16-7 on page 16-13 lists the PMODE values and programming modes.)</p> <p>PMODE.3:0 are multiplexed with P0.7:4 and ACH7:4. On the 8XC196MH, PMODE.2 is also is also multiplexed with T1CLK, and PMODE.3 is also multiplexed with T1DIR.</p>

Table B-6. Signal Descriptions (Continued)

Name	Type	Description
PROG#	I	<p>Programming Start</p> <p>During programming, a falling edge latches data on the PBUS and begins programming, while a rising edge ends programming. The current location is programmed with the same data as long as PROG# remains asserted, so the data on the PBUS must remain stable while PROG# is active.</p> <p>During a word dump, a falling edge causes the contents of an OTPROM location to be output on the PBUS, while a rising edge ends the data transfer.</p> <p>On the 8XC196MC and MD, PROG# is multiplexed with P2.2 and EPA2. On the 8XC196MH, PROG# is multiplexed with P2.2 and EPA1.</p>
PVER	O	<p>Program Verification</p> <p>During slave or auto programming, PVER is updated after each programming pulse. A high output signal indicates successful programming of a location, while a low signal indicates a detected error.</p> <p>PVER is multiplexed with P2.0 and EPA0.</p>
PWM1:0	O	<p>Pulse Width Modulator Outputs</p> <p>These are PWM output pins with high-current drive capability. PWM1:0 are multiplexed with P6.7:6.</p>
RD#	O	<p>Read</p> <p>Read-signal output to external memory. RD# is asserted only during external memory reads.</p> <p>RD# is multiplexed with P5.3.</p>
READY	I	<p>Ready Input</p> <p>This active high input along with the chip configuration registers determine the number of wait states inserted into the bus cycle. The chip configuration registers selects the maximum number of wait states (0, 1, 2, 3, or infinite) that can be inserted into the bus cycle. While READY is low, wait states are inserted into the bus cycle until the programmed number of wait states is reached. If READY is pulled high before the programmed number of wait states is reached, no additional wait states will be inserted into the bus cycle.</p> <p>READY is multiplexed with P5.6.</p>
RESET#	I/O	<p>Reset</p> <p>A level-sensitive reset input to and 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 chip to reset and return to normal operating mode. The 8XC196MH provides the option of preventing an internal reset from generating a reset on the external pin (see "Resetting the Device" on page 13-8). After a device reset, the first instruction fetch is from FF2080H.</p>
RXD1:0 (MH only)	I/O	<p>Receive Serial Data 0 and 1</p> <p>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.</p> <p>RXD0 is multiplexed with P1.1 and RXD1 is multiplexed with P1.3.</p>

Table B-6. Signal Descriptions (Continued)

Name	Type	Description
SCLK1:0# (MH only)	I/O	<p>Shift Clock 0 and 1</p> <p>In SIO mode 4, SCLKx# are bidirectional shift clock signals that synchronize the serial data transfer. The DIR bit in the SP_CON register controls the direction of SCLKx#.</p> <p>DIR = 1 allows an external shift clock to be input on SCLKx#.</p> <p>DIR = 0 causes SCLKx# to output the internal shift clock.</p> <p>SCLK0# is multiplexed with P2.1, BCLK0, and PALE#. SCLK1# is multiplexed with P2.7 and BCLK1.</p>
T1CLK	I	<p>Timer 1 External Clock</p> <p>External clock for timer 1. Timer 1 increments (or decrements) on both rising and falling edges of T1CLK. Also used in conjunction with T1DIR for quadrature counting mode.</p> <p>and</p> <p>External clock for the serial I/O baud-rate generator input (program selectable).</p> <p>On the 8XC196MC and MD, T1CLK is multiplexed with P1.2 and ACH10. On the 8XC196MH, T1CLK is multiplexed with P0.6, ACH6, and PMODE.2.</p>
T1DIR	I	<p>Timer 1 External Direction</p> <p>External direction (up/down) for timer 1. Timer 1 increments when T1DIR is high and decrements when it is low. Also used in conjunction with T1CLK for quadrature counting mode.</p> <p>On the 8XC196MC and MD, T1DIR is multiplexed with P1.3 and ACH11. On the 8XC196MH, T1DIR is multiplexed with P0.7, ACH7, and PMODE.3.</p>
TXD1:0 (MH only)	O	<p>Transmit Serial Data 0 and 1</p> <p>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 outputs.</p> <p>TXD0 is multiplexed with P1.0 and TXD1 is multiplexed with P1.2.</p> <p>TXD0 and 1 are not implemented on the 8XC196MC and MD.</p>
V _{CC}	PWR	<p>Digital Supply Voltage</p> <p>Connect each V_{CC} pin to the digital supply voltage.</p>
V _{PP}	PWR	<p>Programming Voltage</p> <p>During programming, the V_{PP} pin is typically at +12.5 V (V_{PP} voltage). Exceeding the maximum V_{PP} voltage specification can damage the device.</p> <p>V_{PP} also causes the device to exit powerdown mode when it is driven low for at least 50 ns. Use this method to exit powerdown only when using an external clock source because it enables the internal phase clocks, but not the internal oscillator. See "Driving the Vpp Pin Low" on page 14-6.</p> <p>On devices with no internal nonvolatile memory, connect V_{PP} to V_{CC}.</p>
V _{REF}	PWR	<p>Reference Voltage for the A/D Converter</p> <p>This pin also supplies operating voltage to both the analog portion of the A/D converter and the logic used to read port 0 (also port 1 in the 8XC196MC and 8XC196MD).</p>
V _{SS}	GND	<p>Digital Circuit Ground</p> <p>These pins supply ground for the digital circuitry. Connect each V_{SS} pin to ground through the lowest possible impedance path.</p>

Table B-6. Signal Descriptions (Continued)

Name	Type	Description
WG3:1	O	Waveform Generator Phase 1–3 Positive Outputs 3-phase output signals used in motion-control applications. WG1 is multiplexed with P6.1, WG2 is multiplexed with P6.3, and WG3 is multiplexed with P6.5.
WG3:1#	O	Waveform Generator Phase 1–3 Negative Outputs Complimentary 3-phase output signals used in motion-control applications. WG1# is multiplexed with P6.0, WG2# is multiplexed with P6.2, and WG3# is multiplexed with P6.4.
WR#	O	Write [†] This active-low output indicates that an external write is occurring. This signal is asserted only during external memory writes. WR# is multiplexed with P5.2 and WRL#. [†] 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#.
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# is multiplexed with P5.5 and BHE#. [†] The chip configuration register 0 (CCR0) determines whether this pin 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# is multiplexed with P5.2 and WR#. [†] 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#.
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 (MC/MD only). 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 (see datasheet).
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.4 DEFAULT CONDITIONS

Table B-8 lists the values of the signals of the 8XC196MC and 8XC196MD during various operating conditions. The shaded rows indicate those signals that are available only on the

8XC196MD. Table B-9 lists the same information for the 8XC196MH. Table B-7 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-7. Definition of Status Symbols

Symbol	Definition	Symbol	Definition
0	Voltage less than or equal to V_{OL} , V_{IL}	MD0	Medium pull-down
1	Voltage greater than or equal to V_{OH} , V_{IH}	MD1	Medium pull-up
HiZ	High impedance	WK0	Weak pull-down
LoZ0	Low impedance; strongly driven low	WK1	Weak pull-up
LoZ1	Low impedance; strongly driven high	ODIO	Open-drain I/O

Table B-8. 8XC196MC and MD Default Signal Conditions

Port Signals	Alternate Functions	During RESET# Active	Upon RESET# Inactive (Note 14)	Idle	Powerdown
P0.7:0	ACH7:0	HiZ	—	HiZ	HiZ
P1.1:0	ACH9:8	HiZ	—	HiZ	HiZ
P1.2	ACH10/T1CLK	HiZ	—	HiZ	HiZ
P1.3	ACH11/T1DIR	HiZ	—	HiZ	HiZ
P1.4	ACH12	HiZ	—	HiZ	HiZ
P1.5 (Note 15)	ACH13	HiZ	—	HiZ	HiZ
P1.7:6 (Note 15)	—	HiZ	—	HiZ	HiZ
P2.0	EPA0	WK1 (Note 1)	WK1	(Note 12)	(Note 12)
P2.1	EPA1	WK1	WK1	(Note 12)	(Note 12)
P2.3:2	EPA3:2	WK1	WK1	(Note 12)	(Note 12)
P2.6:4	COMP2:0	WK1	WK1; except P2.5 = LZ (Note 1)	(Note 12)	(Note 12)
P2.7	COMP3	WK1	WK1	(Note 12)	(Note 12)
P3.7:0	AD7:0	WK1	HiZ	(Note 3)	(Note 3)
P4.7:0	AD15:8	WK1	HiZ	(Note 3)	(Note 3)
P5.0	ADV#/ALE	WK1 (Note 1)	WK1 (Note 4)	(Note 8)	(Note 8)
P5.1	INST	WK1	WK1	(Note 9)	(Note 9)
P5.2	WR#/WRL#	WK1 (Note 1)	WK1	(Note 10)	(Note 10)
P5.3	RD#	WK1 (Note 1)	WK1 (Note 4)	(Note 10)	(Note 10)
P5.4	ONCE#	MD1	LZ (Note 1)	(Note 10)	(Note 10)
P5.5	BHE#/WRH#	WK1	WK1 (Note 4)	(Note 10)	(Note 10)
P5.6	READY	WK1	(Note 5)	(Note 11)	(Note 11)
P5.7	BUSWIDTH	WK1	(Note 6)	(Note 11)	(Note 11)
P6.0	WG1#	WK1	WK1	(Note 13)	(Note 13)
P6.1	WG1	WK1	WK1	(Note 13)	(Note 13)
P6.2	WG2#	WK1	WK1	(Note 13)	(Note 13)

Table B-8. 8XC196MC and MD Default Signal Conditions (Continued)

Port Signals	Alternate Functions	During RESET# Active	Upon RESET# Inactive (Note 14)	Idle	Powerdown
P6.3	WG2	WK1	WK1	(Note 13)	(Note 13)
P6.4	WG3#	WK1	WK1	(Note 13)	(Note 13)
P6.5	WG3	WK1	WK1	(Note 13)	(Note 13)
P6.6	PWM0	WK0	—	(Note 13)	(Note 13)
P6.7	PWM1	WK0	—	(Note 13)	(Note 13)
P7.1:0 (Note 15)	EPA5:4	WK1 (Note 1)	—	(Note 12)	(Note 12)
P7.3:2 (Note 15)	COMP5:4	WK1	—	(Note 12)	(Note 12)
P7.6:4 (Note 15)	—	WK1	—	(Note 12)	(Note 12)
P7.7 (Note 15)	FREQOUT	WK1	—	(Note 12)	(Note 12)
—	CLKOUT	CLKOUT active, LoZ0/1	—	CLKOUT active, LoZ0/1	LoZ0
—	EA#	HiZ	—	HiZ	HiZ
—	EXTINT	HiZ	—	HiZ	HiZ
—	NMI	WK0	—	WK0	WK0
—	RESET#	LoZ0/HiZ (Note 2)	—	HiZ	HiZ
—	V _{PP}	HiZ	—	LoZ1	LoZ1
—	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	(Note 7)

NOTES:

1. These pins also control test mode entry.
2. If Disable Reset Out = 0, pin is LoZ0. If Disable Reset Out = 1, pin is HiZ.
3. If EA# = 0, Port 3 and Port 4 = HiZ. If EA# = 1, Port 3 and Port 4 = ODIO.
4. If EA# = 1, pin is WK1. If EA# = 0, P5.0, P5.3, and P5.5 are configured as outputs and function as ADV#, RD#, or BHE# respectively.
5. READY function not selected and three wait states inserted to allow fetch of CCB.
6. BUSWIDTH function is selected.
7. If XTAL1 = 1, pin is LoZ0. If XTAL1 = 0, pin is LoZ1.
8. If P5_MODE.0 = 0, port is as programmed.
If P5_MODE.0 = 1 and CCR.3 = 1 (ALE mode), pin is LoZ0. If P5_MODE.0 = 1 and CCR.3 = 0 (ADV# mode), pin is LoZ1.
9. If P5_MODE.1 = 0, port is as programmed. If P5_MODE.1 = 1, pin is LoZ0.
10. If P5_MODE.y = 0, port is as programmed. If P5_MODE.y = 1, pin is LoZ1.
11. If P5_MODE.y = 0, port is as programmed. If P5_MODE.y = 1, pin is HiZ.
12. If P_x_MODE.y = 0, port is as programmed. If P_x_MODE.y = 1, pin is as specified by the associated peripheral.
13. If output port, pin is as programmed. If special function, pin is as specified by the associated peripheral.
14. The values in this column are valid until your software writes to P_x_MODE.
15. Shaded rows indicate those signals that are available only on the 8XC196MD.

Table B-9. 8XC196MH Default Signal Conditions

Port Signals	Alternate Functions	During RESET# Active	Upon RESET# Inactive (Note 12)	Idle	Powerdown
P0.5:0	ACH5:0	HiZ	—	HiZ	HiZ
P0.6	ACH6/T1CLK	HiZ	—	HiZ	HiZ
P0.7	ACH7/T1DIR	HiZ	—	HiZ	HiZ
P1.0	TXD0	WK1	WK1	(Note 10)	(Note 10)
P1.1	RXD0	WK1	WK1	(Note 10)	(Note 10)
P1.2	TXD1	WK1	WK1	(Note 10)	(Note 10)
P1.3	RXD1	WK1	WK1	(Note 10)	(Note 10)
P2.0	EPA0	WK1 (Note 1)	WK1	(Note 10)	(Note 10)
P2.1	BCLK0/SCLK0#	WK1	WK1	(Note 10)	(Note 10)
P2.2	EPA1	WK1	WK1	(Note 10)	(Note 10)
P2.3	COMP3	WK1	WK1	(Note 10)	(Note 10)
P2.5:4	COMP1:0	WK1	WK1	(Note 10)	(Note 10)
P2.6	COMP2	MD1 (Note 1)	MD1	(Note 10)	(Note 10)
P2.7	BCLK1/SCLK1#	WK1	WK1	(Note 10)	(Note 10)
P3.7:0	AD7:0	WK1	HiZ	(Note 10)	(Note 3)
P4.7:0	AD15:8	WK1	HiZ	(Note 3)	(Note 3)
P5.0	ADV#/ALE	WK1 (Note 1)	(Note 6)	(Note 6)	(Note 6)
P5.1	INST	WK1	WK1	(Note 7)	(Note 7)
P5.2	WR#/WRL#	WK1 (Note 1)	WK1	(Note 8)	(Note 8)
P5.3	RD#	WK1 (Note 1)	(Note 4)	(Note 8)	(Note 8)
P5.4	ONCE#	MD1 (Note 1)	MD1	(Note 8)	(Note 8)
P5.5	BHE#/WRH#	WK1	(Note 5)	(Note 10)	(Note 10)
P5.6	READY	WK1	WK1	(Note 9)	(Note 9)
P5.7	BUSWIDTH	WK1	WK1	(Note 9)	(Note 9)
P6.0	WG1#	WK1	WK1	(Note 11)	(Note 11)
P6.1	WG1	WK1	WK1	(Note 11)	(Note 11)
P6.2	WG2#	WK1	WK1	(Note 11)	(Note 11)
P6.3	WG2	WK1	WK1	(Note 11)	(Note 11)
P6.4	WG3#	WK1	WK1	(Note 11)	(Note 11)
P6.5	WG3	WK1	WK1	(Note 11)	(Note 11)
P6.6	PWM0	WK0	WK1	(Note 11)	(Note 11)
P6.7	PWM1	WK0	WK1	(Note 11)	(Note 11)
—	EA#	HiZ	—	HiZ	HiZ
—	EXTINT	HiZ	—	HiZ	HiZ
—	NMI	WK0	—	WK0	WK0
—	RESET#	LoZ0/HiZ (Note 2)	—	HiZ	HiZ
—	V _{PP}	HiZ	—	LoZ1	LoZ1

Table B-9. 8XC196MH Default Signal Conditions (Continued)

Port Signals	Alternate Functions	During RESET# Active	Upon RESET# Inactive (Note 12)	Idle	Powerdown
—	XTAL1	Osc input, HiZ	—	Osc input, HiZ	Osc input, HiZ
—	XTAL2	Osc output, LoZ0/1	—	Osc output, LoZ0/1	(Note 5)

NOTES:

1. These pins also control test mode entry.
2. If Disable Reset Out = 0, pin is LoZ0. Else if Disable Reset Out = 1, pin is HiZ.
3. If EA# = 0, Port 3 and Port 4 = HiZ. If EA# = 1, Port 3 and Port 4 = ODIO.
4. If EA# = 1, pin is WK1. If EA# = 0, P5.0, P5.3, and P5.5 are configured as outputs and function as ADV#, RD#, or BHE# respectively.
5. If XTAL1 = 1, pin is LoZ0. If XTAL1 = 0, pin is LoZ1.
6. If P5_MODE.0 = 0, port is as programmed.
If P5_MODE.0 = 1 and CCR.3 = 1 (ALE mode), pin is LoZ0. If P5_MODE.0 = 1 and CCR.3 = 0 (ADV# mode), pin is LoZ1.
7. If P5_MODE.1 = 0, port is as programmed. If P5_MODE.1 = 1, pin is LoZ0.
8. If P5_MODE.y = 0, port is as programmed. If P5_MODE.y = 1, pin is LoZ1.
9. If P5_MODE.y = 0, port is as programmed. If P5_MODE.y = 1, pin is HiZ.
10. If Px_MODE.y = 0, port is as programmed. If Px_MODE.y = 1, pin is as specified by the associated peripheral.
11. If output port, pin is as programmed. If special function, pin is as specified by the associated peripheral.
12. The values in this column are valid until your software writes to Px_MODE.



C

Registers

APPENDIX C REGISTERS

This appendix provides reference information about the device registers. Table C-1 lists the modules and major components of the device 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	Chip Configuration	CPU	EPA (8XC196MC, MH, x = 0–3) (8XC196MD, x = 0–5)
AD_COMMAND AD_RESULT AD_TEST AD_TIME	CCR0 CCR1 GEN_CON (8XC196MH) PPW (or SP_PPW) USFR	ONES_REG PSW SP ZERO_REG	COMP _x _CON COMP _x _TIME EPA _x _CON EPA _x _TIME
Frequency Generator (8XC196MD)	I/O Ports	Interrupts and PTS	Memory Control
FREQ_CNT FREQ_GEN	P _x _DIR [†] P _x _MODE [†] P _x _PIN ^{††} P _x _REG [†]	INT_MASK INT_MASK1 INT_PEND INT_PEND1 PI_MASK PI_PEND PTSEL PTSSRV	WSR
PWM (x = 0–1)	Serial Port (8XC196MH, x = 0–1)	Timers (x = 0–1)	Waveform Generator (x = 1–3)
PWM_COUNT PWM_PERIOD PWM _x _CONTROL	SBUF _x _RX SBUF _x _TX SP _x _BAUD SP _x _CON SP _x _STATUS	TxCONTROL T1RELOAD TIMER _x WATCHDOG	WG_COMP _x WG_CONTROL WG_COUNTER WG_OUTPUT WG_PROTECT WG_RELOAD

[†] For the 8XC196MC, x = 2, 5; for the 8XC196MD, x = 2, 5, 7; for the 8XC196MH, x = 1, 2, 5.

^{††} For the 8XC196MC and 8XC196MH, x = 0–5; for the 8XC196MD, x = 0–5, 7.

Table C-2. Register Name, Address, and Reset Status

Register Mnemonic	Register Name	Hex Addr	Binary Reset Value			
			High		Low	
AD_COMMAND	A/D Command	1FAC	1000 0000			
AD_RESULT (MC, MD) AD_RESULT (MH)	A/D Result	1FAA	1111 1111	1100	0000	
AD_TEST (MC, MD) AD_TEST (MH)	A/D Test	1FAE			1100	0000
AD_TIME	A/D Time	1FAF			1000	1000
CCR0	Chip Configuration 0	†††	XXXX		XXXX	
CCR1	Chip Configuration 1	†††	XXXX		XXXX	
COMP0_CON	EPA Compare 0 Control	1F58			0000	0000
COMP1_CON	EPA Compare 1 Control	1F5C			0000	0000
COMP2_CON	EPA Compare 2 Control	1F60			0000	0000
COMP3_CON (MC, MD) COMP3_CON (MH)	EPA Compare 3 Control	1F64 1F4C			0000	0000
COMP4_CON (MD)	EPA Compare 4 Control	1F68			0000	0000
COMP5_CON (MD)	EPA Compare 5 Control	1F6C			0000	0000
COMP0_TIME	EPA Compare 0 Time	1F5A	XXXX	XXXX	XXXX	XXXX
COMP1_TIME	EPA Compare 1 Time	1F5E	XXXX	XXXX	XXXX	XXXX
COMP2_TIME	EPA Compare 2 Time	1F62	XXXX	XXXX	XXXX	XXXX
COMP3_TIME (MC, MD) COMP3_TIME (MH)	EPA Compare 3 Time	1F66 1F4E	XXXX	XXXX	XXXX	XXXX
COMP4_TIME (MD)	EPA Compare 4 Time	1F6A	XXXX	XXXX	XXXX	XXXX
COMP5_TIME (MD)	EPA Compare 5 Time	1F6E	XXXX	XXXX	XXXX	XXXX
EPA0_CON	EPA Capture/Comp 0 Control	1F40			0000	0000
EPA1_CON	EPA Capture/Comp 1 Control	1F44			0000	0000
EPA2_CON (MC, MD)	EPA Capture/Comp 2 Control	1F48			0000	0000
EPA3_CON (MC, MD)	EPA Capture/Comp 3 Control	1F4C			0000	0000
EPA4_CON (MD)	EPA Capture/Comp 4 Control	1F50			0000	0000
EPA5_CON (MD)	EPA Capture/Comp 5 Control	1F54			0000	0000
EPA0_TIME	EPA Capture/Comp 0 Time	1F42	XXXX	XXXX	XXXX	XXXX

† Reset value is FFH when pin is not driven.

†† Reset value is 80H if the EA# pin is high, A9H if EA# is low.

††† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after a device reset, unless the device is entering programming modes (see "Entering Programming Modes" on page 16-13), in which case the programming chip configuration bytes (PCCBs) are used. The CCBs reside in internal nonvolatile memory at addresses 2018H (CCB0) and 201AH (CCB1).

Table C-2. Register Name, Address, and Reset Status (Continued)

Register Mnemonic	Register Name	Hex Addr	Binary Reset Value			
			High		Low	
EPA1_TIME	EPA Capture/Comp 1 Time	1F46	XXXX	XXXX	XXXX	XXXX
EPA2_TIME (MC, MD)	EPA Capture/Comp 2 Time	1F4A	XXXX	XXXX	XXXX	XXXX
EPA3_TIME (MC, MD)	EPA Capture/Comp 3 Time	1F4E	XXXX	XXXX	XXXX	XXXX
EPA4_TIME (MD)	EPA Capture/Comp 4 Time	1F52	XXXX	XXXX	XXXX	XXXX
EPA5_TIME (MD)	EPA Capture/Comp 5 Time	1F56	XXXX	XXXX	0000	0000
FREQ_CNT (MD)	Frequency Gen Count	1FBA			0000	0000
FREQ_GEN (MD)	Frequency	1FB8			0000	0000
GEN_CON (MH)	General Configuration	1FA0			0000	0000
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
ONES_REG	Ones Register	0002	1111	1111	1111	1111
P1_DIR (MH)	Port 1 I/O Direction	1F9B			1111	1111
P2_DIR	Port 2 I/O Direction	1FD2			1111	1111
P5_DIR	Port 5 I/O Direction	1FF3			1111	1111
P7_DIR (MD)	Port 7 I/O Direction	1FD3			1111	1111
P1_MODE (MH)	Port 1 Mode	1F99			0000	0000
P2_MODE	Port 2 Mode	1FD0			0000	0000
P5_MODE	Port 5 Mode	1FF1				††
P7_MODE (MD)	Port 7 Mode	1FD1			0000	0000
P0_PIN (MC, MD)	Port 0 Pin Input	1FA8				†
P0_PIN (MH)		1FDA				
P1_PIN (MC, MD)	Port 1 Pin Input	1FA9				†
P1_PIN (MH)		1F9F				
P2_PIN	Port 2 Pin Input	1FD6				†
P3_PIN	Port 3 Pin Input	1FFE				†
P4_PIN	Port 4 Pin Input	1FFF				†
P5_PIN	Port 5 Pin Input	1FF7			1111	1111

† Reset value is FFH when pin is not driven.

†† Reset value is 80H if the EA# pin is high, A9H if EA# is low.

††† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after a device reset, unless the device is entering programming modes (see "Entering Programming Modes" on page 16-13), in which case the programming chip configuration bytes (PCCBs) are used. The CCBs reside in internal nonvolatile memory at addresses 2018H (CCB0) and 201AH (CCB1).

Table C-2. Register Name, Address, and Reset Status (Continued)

Register Mnemonic	Register Name	Hex Addr	Binary Reset Value			
			High		Low	
P7_PIN (MD)	Port 7 Pin Input	1FD7	XXXX XXXX			
P1_REG (MH)	Port 1 Data Output	1F9D	1111 1111			
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 (MC, MD) P5_REG (MH)	Port 5 Data Output	1FF5	†			
P7_REG (MD)	Port 7 Data Output	1FD5	1111 1111			
PI_MASK	Peripheral Interrupt Mask	1FBC	1010 1010			
PI_PEND	Peripheral Interrupt Pending	1FBE	1010 1010			
PPW	Programming Pulse Width	no direct access				
PSW	Processor Status Word					
PTSSEL	PTS Select	0004	0000	0000	0000	0000
PTSSRV	PTS Service	0006	0000	0000	0000	0000
PWM_COUNT	PWM Count	1FB6	0000 0000			
PWM_PERIOD	PWM Period	1FB4	0000 0000			
PWM0_CONTROL	PWM 0 Control	1FB0	0000 0000			
PWM1_CONTROL	PWM 1 Control	1FB2	0000 0000			
SBUF0_RX (MH)	Serial Port Receive Buffer	1F80	0000 0000			
SBUF1_RX (MH)	Serial Port Receive Buffer	1F88	0000 0000			
SBUF0_TX (MH)	Serial Port Transmit Buffer	1F82	0000 0000			
SBUF1_TX (MH)	Serial Port Transmit Buffer	1F8A	0000 0000			
SP	Stack Pointer	0018	XXXX	XXXX	XXXX	XXXX
SP0_BAUD (MH)	Serial Port 0 Baud Rate	1F84	0000	0000	0000	0000
SP1_BAUD (MH)	Serial Port 1 Baud Rate	1F8C	0000	0000	0000	0000
SP0_CON (MH)	Serial Port Control	1F83	0000 0000			
SP1_CON (MH)	Serial Port Control	1F8B	0000 0000			
SP0_STATUS (MH)	Serial Port Status	1F81	0000 0000			
SP1_STATUS (MH)	Serial Port Status	1F89	0000 0000			

† Reset value is FFH when pin is not driven.

†† Reset value is 80H if the EA# pin is high, A9H if EA# is low.

††† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after a device reset, unless the device is entering programming modes (see "Entering Programming Modes" on page 16-13), in which case the programming chip configuration bytes (PCCBs) are used. The CCBs reside in internal nonvolatile memory at addresses 2018H (CCB0) and 201AH (CCB1).

Table C-2. Register Name, Address, and Reset Status (Continued)

Register Mnemonic	Register Name	Hex Addr	Binary Reset Value			
			High		Low	
T1CONTROL	Timer 1 Control	1F78	0000 0000			
T2CONTROL	Timer 2 Control	1F7C	0000 0000			
T1RELOAD	Timer 1 Reload	1F72	XXXX	XXXX	XXXX	XXXX
TIMER1	Timer 1 Value	1F7A	0000	0000	0000	0000
TIMER2	Timer 2 Value	1F7E	0000	0000	0000	0000
USFR (MC, MD) USFR (MH)	UPROM Special Function	1FF6	0000		0010	
			XXXX		XXXX	
WATCHDOG	Watchdog Timer	000A	XXXX		XXXX	
WG_COMP1	Waveform Gen Phase Comp 1	1FC2	0000	0000	0000	0000
WG_COMP2	Waveform Gen Phase Comp 2	1FC4	0000	0000	0000	0000
WG_COMP3	Waveform Gen Phase Comp 3	1FC6	0000	0000	0000	0000
WG_CONTROL (MC, MD) WG_CONTROL (MH)	Waveform Gen Control	1FCC	0000	0000	1100	0000
			1000	0000	0000	0000
WG_COUNTER	Waveform Gen Count	1FCA	XXXX	XXXX	XXXX	XXXX
WG_OUTPUT	Waveform Gen Output Config	1FC0	0000	0000	0000	0000
WG_PROTECT (MC, MD) WG_PROTECT (MH)	Waveform Gen Protection	1FCE	1111		0000	
			1110		0000	
WG_RELOAD	Waveform Gen Reload	1FC8	0000	0000	0000	0000
WSR	Window Selection	0014	0000		0000	
ZERO_REG	Zero Register	0000	0000	0000	0000	0000

† Reset value is FFH when pin is not driven.

†† Reset value is 80H if the EA# pin is high, A9H if EA# is low.

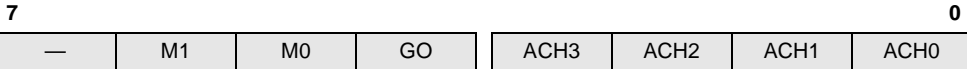
††† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after a device reset, unless the device is entering programming modes (see "Entering Programming Modes" on page 16-13), in which case the programming chip configuration bytes (PCCBs) are used. The CCBs reside in internal nonvolatile memory at addresses 2018H (CCB0) and 201AH (CCB1).

AD_COMMAND

AD_COMMAND

Address: 1FACH
Reset State: 80H

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, and selects the conversion mode.



Bit Number	Bit Mnemonic	Function															
7	—	Reserved; for compatibility with future devices, write zeros to these bits.															
6:5	M1:0	A/D Mode† These bits determine the A/D mode. <table style="margin-left: 20px;"> <tr> <td>M1</td> <td>M0</td> <td>Mode</td> </tr> <tr> <td>0</td> <td>0</td> <td>10-bit conversion</td> </tr> <tr> <td>0</td> <td>1</td> <td>8-bit conversion</td> </tr> <tr> <td>1</td> <td>0</td> <td>threshold detect high</td> </tr> <tr> <td>1</td> <td>1</td> <td>threshold detect low</td> </tr> </table>	M1	M0	Mode	0	0	10-bit conversion	0	1	8-bit conversion	1	0	threshold detect high	1	1	threshold detect low
M1	M0	Mode															
0	0	10-bit conversion															
0	1	8-bit conversion															
1	0	threshold detect high															
1	1	threshold detect low															
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.															

† 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.

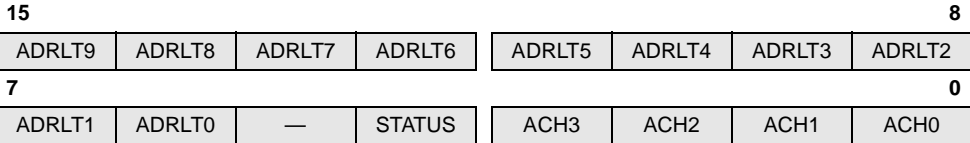
†† It is the act of writing to the GO bit, rather than its value, that starts a conversion. Even if the GO bit has the desired value, you must set it again to start a conversion immediately or clear it again to arm it for an EPA-initiated conversion.

AD_RESULT (Read)

AD_RESULT (Read)

Address: 1FAAH
 Reset State (MC, MD): FFC0H
 Reset State (MH): 7FC0H

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 ten-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.



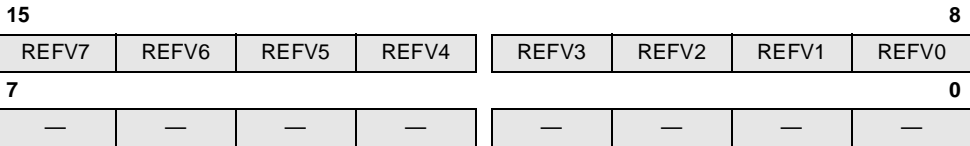
Bit Number	Bit Mnemonic	Function
15:6	ADRLT9:0	A/D Result These bits contain the A/D conversion result.
5	—	Reserved. This bit is undefined.
4	STATUS	A/D Status Indicates the status of the A/D converter. Up to 8 state times are required to set this bit following a start command. When testing this bit, 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.

AD_RESULT (Write)

AD_RESULT (Write)

Address: 1FAAH
 Reset State (MC, MD): FFC0H
 Reset State (MH): 7FC0H

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.



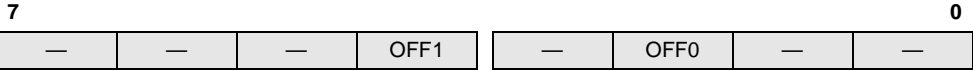
Bit Number	Bit Mnemonic	Function
15:8	REFV7:0	Reference Voltage These bits specify the threshold value. This selects a reference voltage that is compared with an analog input pin. When the voltage on the analog input pin crosses over (detect high) or under (detect low) the threshold value, the A/D conversion complete interrupt pending bit is set. Use the following formula to determine the value to write to this register for a given threshold voltage. $\text{reference voltage} = \frac{\text{desired threshold voltage} \times 256}{V_{\text{REF}} - \text{ANGND}}$
7:0	—	Reserved; for compatibility with future devices, write zeros to these bits.

AD_TEST

AD_TEST

Address: 1FAEH
 Reset State (MC, MD): C0H
 Reset State (MH): 88H

The A/D test (AD_TEST) register specifies adjustments for DC offset errors.



Bit Number	Bit Mnemonic	Function
7:5	—	Reserved; for compatibility with future devices, write zeros to these bits.
4	OFF1	Offset This bit , along with OFF0 (bit 2) allows you to set the zero offset point. OFF1 OFF0 0 0 no adjustment 0 1 add 2.5 mV 1 0 subtract 2.5 mV 1 1 subtract 5.0 mV
3	—	Reserved; for compatibility with future devices, write zero to this bit.
2	OFF0	See bit 4 (OFF1).
1:0	—	Reserved; for compatibility with future devices, write zeros to these bits.

AD_TIME

AD_TIME

Address: 1FAFH
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	<p>A/D Sample Time</p> <p>These bits specify the sample time. Use the following formula to compute the sample time.</p> $SAM = \frac{T_{SAM} \times F_{XTAL1} - 2}{8}$ <p>where: SAM = 1 to 7 T_{SAM} = the sample time, in μsec, from the data sheet F_{XTAL1} = the input frequency on XTAL1, in MHz</p>
4:0	CONV4:0	<p>A/D Convert Time</p> <p>These bits specify the conversion time for each bit. Use the following formula to compute the conversion time.</p> $CONV = \left[\frac{T_{CONV} \times F_{XTAL1} - 3}{2 \times B} \right] - 1$ <p>where: CONV = 2 to 31 T_{CONV} = the conversion time, in μsec, from the data sheet F_{XTAL1} = the input frequency on XTAL1, in MHz B = the number of bits to be converted (8 or 10)</p>

CCR0

CCR0

no direct access†

The chip configuration 0 (CCR0) register controls powerdown mode, bus-control signals, and internal memory protection. Three of its bits combine with two bits of CCR1 to control wait states and bus width.

7

0

LOC1	LOC0	IRC1	IRC0	ALE	WR	BW0	PD
------	------	------	------	-----	----	-----	----

Bit Number	Bit Mnemonic	Function																												
7:6	LOC1:0	<p>Lock Bits</p> <p>These two bits control read and write access to the OTPROM during normal operation. Refer to “Controlling Access to the OTPROM During Normal Operation” on page 16-4 for details.</p> <p>LOC1 LOC0</p> <table> <tr> <td>0</td> <td>0</td> <td>read and write protect</td> </tr> <tr> <td>0</td> <td>1</td> <td>read protect only</td> </tr> <tr> <td>1</td> <td>0</td> <td>write protect only</td> </tr> <tr> <td>1</td> <td>1</td> <td>no protection</td> </tr> </table>	0	0	read and write protect	0	1	read protect only	1	0	write protect only	1	1	no protection																
0	0	read and write protect																												
0	1	read protect only																												
1	0	write protect only																												
1	1	no protection																												
5:4	IRC1:0	<p>Internal Ready Control</p> <p>These two bits, along with IRC2 (CCR1.1) and the READY pin determine the number of wait states that can be inserted into the bus cycle. While READY is held low, wait states are inserted into the bus cycle until the programmed number of wait states is reached. If READY is pulled high before the programmed number of wait states is reached, no additional wait states will be inserted into the bus cycle.</p> <p>IRC2 IRC1 IRC0</p> <table> <tr> <td>0</td> <td>0</td> <td>0</td> <td>zero wait states</td> </tr> <tr> <td>0</td> <td>X</td> <td>1</td> <td>illegal</td> </tr> <tr> <td>1</td> <td>1</td> <td>X</td> <td>illegal</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>one wait state</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> <td>two wait states</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> <td>three wait states</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>infinite</td> </tr> </table> <p>If you choose the infinite wait states option, you must keep P5.6 configured as the READY signal. Also, be sure to add external hardware to count wait states and pull READY high within a specified time. Otherwise, a defective external device could tie up the address/data bus indefinitely.</p>	0	0	0	zero wait states	0	X	1	illegal	1	1	X	illegal	1	0	0	one wait state	1	0	1	two wait states	1	1	0	three wait states	1	1	1	infinite
0	0	0	zero wait states																											
0	X	1	illegal																											
1	1	X	illegal																											
1	0	0	one wait state																											
1	0	1	two wait states																											
1	1	0	three wait states																											
1	1	1	infinite																											

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after reset, unless the microcontroller is entering programming modes (see “Entering Programming Modes” on page 16-13), in which case the programming chip configuration bytes (PCCBs) are used. The CCBs reside in nonvolatile memory at addresses 2018H (CCB0) and 201AH (CCB1).

CCRO

CCRO (Continued)

no direct access†

The chip configuration 0 (CCRO) register controls powerdown mode, bus-control signals, and internal memory protection. Three of its bits combine with two bits of CCR1 to control wait states and bus width.

7

0

LOC1	LOC0	IRC1	IRC0	ALE	WR	BW0	PD
------	------	------	------	-----	----	-----	----

Bit Number	Bit Mnemonic	Function												
3	ALE	Address Valid Strobe and Write Strobe												
2	WR	<p>These bits define which bus-control signals will be generated during external read and write cycles.</p> <p>ALE WR</p> <table> <tr> <td>0</td> <td>0</td> <td>address valid with write strobe mode (ADV#, RD#, WRL#, WRH#)</td> </tr> <tr> <td>0</td> <td>1</td> <td>address valid strobe mode (ADV#, RD#, WR#, BHE#)</td> </tr> <tr> <td>1</td> <td>0</td> <td>write strobe mode (ALE, RD#, WRL#, WRH#)</td> </tr> <tr> <td>1</td> <td>1</td> <td>standard bus-control mode (ALE, RD#, WR#, BHE#)</td> </tr> </table>	0	0	address valid with write strobe mode (ADV#, RD#, WRL#, WRH#)	0	1	address valid strobe mode (ADV#, RD#, WR#, BHE#)	1	0	write strobe mode (ALE, RD#, WRL#, WRH#)	1	1	standard bus-control mode (ALE, RD#, WR#, BHE#)
0	0	address valid with write strobe mode (ADV#, RD#, WRL#, WRH#)												
0	1	address valid strobe mode (ADV#, RD#, WR#, BHE#)												
1	0	write strobe mode (ALE, RD#, WRL#, WRH#)												
1	1	standard bus-control mode (ALE, RD#, WR#, BHE#)												
1	BW0	<p>Buswidth Control</p> <p>This bit, along with the BW1 bit (CCR1.2), selects the bus width.</p> <p>BW1 BW0</p> <table> <tr> <td>0</td> <td>0</td> <td>illegal</td> </tr> <tr> <td>0</td> <td>1</td> <td>16-bit only</td> </tr> <tr> <td>1</td> <td>0</td> <td>8-bit only</td> </tr> <tr> <td>1</td> <td>1</td> <td>BUSWIDTH pin controlled</td> </tr> </table>	0	0	illegal	0	1	16-bit only	1	0	8-bit only	1	1	BUSWIDTH pin controlled
0	0	illegal												
0	1	16-bit only												
1	0	8-bit only												
1	1	BUSWIDTH pin controlled												
0	PD	<p>Powerdown Enable</p> <p>Controls whether the IDLPD #2 instruction causes the microcontroller to enter powerdown mode. If your design uses 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 powerdown mode.</p> <p>0 = disable powerdown mode 1 = enable powerdown mode</p>												

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after reset, unless the microcontroller is entering programming modes (see "Entering Programming Modes" on page 16-13), in which case the programming chip configuration bytes (PCCBs) are used. The CCBs reside in nonvolatile memory at addresses 2018H (CCB0) and 201AH (CCB1).

CCR1

CCR1

no direct access[†]

The chip configuration 1 (CCR1) register enables the watchdog timer and selects the bus timing mode. Two of its bits combine with three bits of CCR0 to control wait states and bus width.

7

0

1	1	0	1	WDE	BW1	IRC2	0
---	---	---	---	-----	-----	------	---

Bit Number	Bit Mnemonic	Function
7:6	1	To guarantee proper operation, write ones to these bits.
5	0	To guarantee proper operation, write zero to this bit.
4	1	To guarantee proper operation, write one to this bit.
3	WDE	<p>Watchdog Timer Enable</p> <p>Selects whether the watchdog timer is always enabled or enabled the first time it is cleared.</p> <p>0 = always enabled 1 = enabled first time it is cleared</p>
2	BW1	<p>Buswidth Control</p> <p>This bit, along with the BW0 bit (CCR0.1), selects the bus width.</p> <p>BW1 BW0</p> <p>0 0 illegal 0 1 16-bit only 1 0 8-bit only 1 1 BUSWIDTH pin controlled</p>

[†] The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after reset, unless the microcontroller is entering programming modes (see “Entering Programming Modes” on page 16-13), in which case the programming chip configuration bytes (PCCBs) are used. The CCBs reside in nonvolatile memory at addresses 2018H (CCB0) and 201AH (CCB1).

CCR1

CCR1 (Continued)

no direct access[†]

The chip configuration 1 (CCR1) register enables the watchdog timer and selects the bus timing mode. Two of its bits combine with three bits of CCR0 to control wait states and bus width.

7

0

1	1	0	1	WDE	BW1	IRC2	0
---	---	---	---	-----	-----	------	---

Bit Number	Bit Mnemonic	Function																																
1	IRC2	<p>Ready Control</p> <p>This bit, along with IRC0 (CCR0.4), IRC1 (CCR0.5), and the READY pin determine the number of wait states that can be inserted into the bus cycle. While READY is held low, wait states are inserted into the bus cycle until the programmed number of wait states is reached. If READY is pulled high before the programmed number of wait states is reached, no additional wait states will be inserted into the bus cycle.</p> <table border="1"> <thead> <tr> <th>IRC2</th> <th>IRC1</th> <th>IRC0</th> <th></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>zero wait states</td> </tr> <tr> <td>0</td> <td>X</td> <td>1</td> <td>illegal</td> </tr> <tr> <td>1</td> <td>1</td> <td>X</td> <td>illegal</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>one wait state</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> <td>two wait states</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> <td>three wait states</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>infinite</td> </tr> </tbody> </table> <p>If you choose the infinite wait states option, you must keep P5.6 configured as the READY signal. Also, be sure to add external hardware to count wait states and pull READY high within a specified time. Otherwise, a defective external device could tie up the address/data bus indefinitely.</p>	IRC2	IRC1	IRC0		0	0	0	zero wait states	0	X	1	illegal	1	1	X	illegal	1	0	0	one wait state	1	0	1	two wait states	1	1	0	three wait states	1	1	1	infinite
IRC2	IRC1	IRC0																																
0	0	0	zero wait states																															
0	X	1	illegal																															
1	1	X	illegal																															
1	0	0	one wait state																															
1	0	1	two wait states																															
1	1	0	three wait states																															
1	1	1	infinite																															
0	0	Reserved; for compatibility with future devices, write zero to this bit.																																

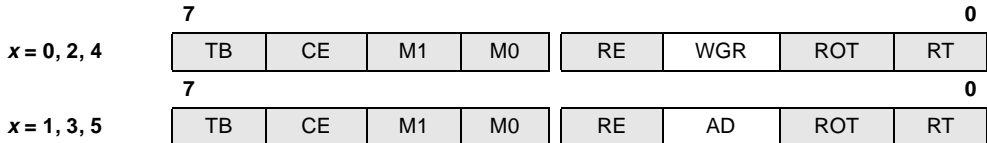
[†] The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after reset, unless the microcontroller is entering programming modes (see "Entering Programming Modes" on page 16-13), in which case the programming chip configuration bytes (PCCBs) are used. The CCBs reside in nonvolatile memory at addresses 2018H (CCB0) and 201AH (CCB1).

COMPx_CON

COMP_x_CON
x = 0–3 (8XC196MC, MH)
x = 0–5 (8XC196MD)

Address: Table C-3
 Reset State:

The EPA compare control (COMP_x_CON) registers determine the function of the EPA compare channels.



7	TB	Time Base Select Specifies the reference timer. 0 = timer 1 is the reference timer and timer 2 is the opposite timer 1 = timer 2 is the reference timer and timer 1 is the opposite timer A compare event (start of an A/D conversion; clearing, setting, or toggling an output pin; and/or resetting either timer) occurs when the reference timer matches the time programmed in the event-time register.
6	CE	Compare Enable This bit enables the compare function. 0 = compare function disabled 1 = compare function enabled
5:4	M1:0	EPA Mode Select Specifies the type of compare event. M1 M0 0 0 no output 0 1 clear output pin 1 0 set output pin 1 1 toggle output pin
3	RE	Re-enable Allows a compare event to continue to execute each time the event-time register (COMP _x _TIME) matches the reference timer rather than only upon the first time match. 0 = compare function will drive the output only once 1 = compare function always enabled

COMPx_CON

COMPx_CON (Continued)

x = 0–3 (8XC196MC, MH)

x = 0–5 (8XC196MD)

Address: Table C-3
Reset State:

The EPA compare control (COMPx_CON) registers determine the function of the EPA compare channels.

	7							0
x = 0, 2, 4	TB	CE	M1	M0	RE	WGR	ROT	RT
	7							0
x = 1, 3, 5	TB	CE	M1	M0	RE	AD	ROT	RT

2	WGR AD	<p>A/D Conversion, Waveform Generator Reload</p> <p>The function of this bit depends on the EPA channel.</p> <p>For EPA capture/compare channels 0, 2, 4:</p> <p>The WGR bit allows you to use the EPA activities to cause the reload of new values in the waveform generator.</p> <p>0 = no action 1 = enables waveform generator reload</p> <p>For EPA capture/compare channels 1, 3, 5:</p> <p>The AD bit allows you to use the EPA activities to start an A/D conversion that has been previously set up in the A/D control registers.</p> <p>0 = causes no A/D action 1 = starts an A/D conversion on an output compare</p>
1	ROT	<p>Reset Opposite Timer</p> <p>Selects the timer that is to be reset if the RT bit is set.</p> <p>0 = selects the reference timer for possible reset 1 = selects the opposite timer for possible reset</p> <p>The state of the TB bit determines which timer is the reference timer and which timer is the opposite timer.</p>
0	RT	<p>Reset Timer</p> <p>This bit controls whether the timer selected by the ROT bit will be reset.</p> <p>1 = resets the timer selected by the ROT bit 0 = disables the reset function</p>

COMP_x_TIME

<p>COMP_x_TIME x = 0–3 (8XC196MC, MH) x = 0–5 (8XC196MD)</p> <p>The EPA compare <i>x</i> time (COMP_x_TIME) registers are the event-time registers for the EPA compare channels; they are functionally identically to the EPA_x_TIME registers. The EPA triggers a compare event when the reference timer matches the value in COMP_x_TIME.</p>	<p>Address: Table C-3 Reset State:</p>				
<p>15 0</p> <div style="border: 1px solid black; background-color: #f0f0f0; padding: 5px; text-align: center; margin: 5px 0;"> EPA Event Time Value </div>					
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 15%;">Bit Number</th> <th>Function</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">15:0</td> <td>EPA Event Time Value Write the desired compare event time to this register.</td> </tr> </tbody> </table>		Bit Number	Function	15:0	EPA Event Time Value Write the desired compare event time to this register.
Bit Number	Function				
15:0	EPA Event Time Value Write the desired compare event time to this register.				

Table C-3. COMP_x_TIME Addresses and Reset Values

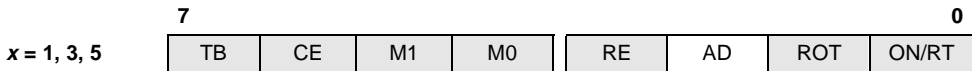
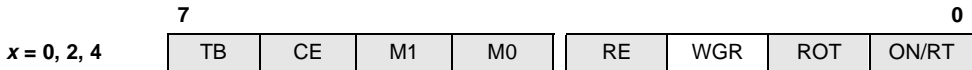
Register	Address	Reset Value
COMP0_TIME (8XC196M _x)	1F5AH	XXXXH
COMP1_TIME (8XC196M _x)	1F5EH	XXXXH
COMP2_TIME (8XC196M _x)	1F62H	XXXXH
COMP3_TIME (8XC196MC, MD) COMP3_TIME (8XC196MH)	1F66H 1F4EH	XXXXH XXXXH
COMP4_TIME (8XC196MD)	1F6AH	XXXXH
COMP5_TIME (8XC196MD)	1F6EH	XXXXH

EPAx_CON

EPAx_CON
x = 0–1 (8XC196MH)
x = 0–3 (8XC196MC)
x = 0–5 (8XC196MD)

Address: Table C-4
 Reset State:

The EPA control (EPAx_CON) registers control the functions of their assigned capture/compare channels.



Bit Number	Bit Mnemonic	Function																														
7	TB	Time Base Select Specifies the reference timer. 0 = timer 1 is the reference timer and timer 2 is the opposite timer 1 = timer 2 is the reference timer and timer 1 is the opposite timer A compare event (reloading the waveform generator; starting an A/D conversion; clearing, setting, or toggling an output pin; and/or resetting either timer) occurs when the reference timer matches the time programmed in the event-time register. When a capture event (falling edge, rising edge, or an edge change on the EPAx pin) occurs, the reference timer value is saved in the EPA event-time register (EPAx_TIME).																														
6	CE	Compare Enable Determines whether the EPA channel operates in capture or compare mode. 0 = capture mode 1 = compare mode																														
5:4	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. <table style="margin-left: 20px; border-collapse: collapse;"> <tr> <td style="text-align: center;">M1</td> <td style="text-align: center;">M0</td> <td style="text-align: left;">Capture Mode Event</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td>no capture</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td>capture on falling edge</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td>capture on rising edge</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td>capture on either edge</td> </tr> <tr> <td style="text-align: center;">M1</td> <td style="text-align: center;">M0</td> <td style="text-align: left;">Compare Mode Action</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td>no output</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td>clear output pin</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td>set output pin</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td>toggle output pin</td> </tr> </table>	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
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																														

EPAx_CON

EPAx_CON (Continued)

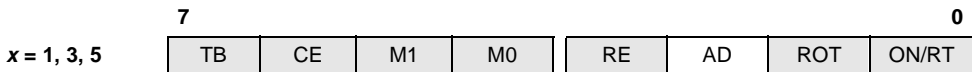
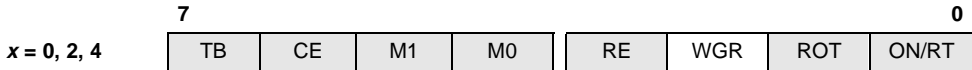
x = 0–1 (8XC196MH)

x = 0–3 (8XC196MC)

x = 0–5 (8XC196MD)

Address: Table C-4
Reset State:

The EPA control (EPAx_CON) registers control the functions of their assigned capture/compare channels.



Bit Number	Bit Mnemonic	Function
3	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 (EPAx_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
2	WGR AD	Waveform Generator Reload; A/D Conversion The function of this bit depends on the EPA channel. For EPA capture/compare channels 0, 2, 4: The WGR bit allows you to use the EPA activities to cause the reload of new values in the waveform generator. 0 = no action 1 = enables waveform generator reload For EPA capture/compare channels 1, 3, 5: The AD bit allows you to use the EPA activities to start an A/D conversion that has been previously set up in the A/D control registers. 0 = causes no A/D action 1 = starts an A/D conversion on an output compare
1	ROT	Reset Opposite Timer Controls different functions for capture and compare modes. In Capture Mode: 0 = causes no action 1 = resets the opposite timer In Compare Mode: Selects the timer that is to be reset if the RT bit is set. 0 = selects the reference timer for possible reset 1 = selects the opposite timer for possible reset The TB bit (bit 7) selects which is the reference timer and which is the opposite timer.

EPAx_CON

EPAx_CON (Continued)

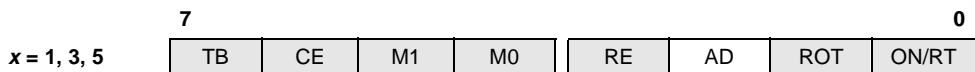
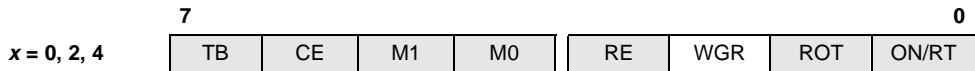
x = 0–1 (8XC196MH)

x = 0–3 (8XC196MC)

x = 0–5 (8XC196MD)

Address: Table C-4
Reset State:

The EPA control (EPAx_CON) registers control the functions of their assigned capture/compare channels.



Bit Number	Bit Mnemonic	Function
0	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 (EPAx_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 ROT-selected timer

Table C-4. EPAx_CON Addresses and Reset Values

Register	Address	Reset Value
EPA0_CON (8XC196Mx)	1F40H	00H
EPA1_CON (8XC196Mx)	1F44H	00H
EPA2_CON (8XC196MC, MD)	1F48H	00H
EPA3_CON (8XC196MC, MD)	1F4CH	00H
EPA4_CON (8XC196MD)	1F50H	00H
EPA5_CON (8XC196MD)	1F54H	00H

EPA_x_TIME

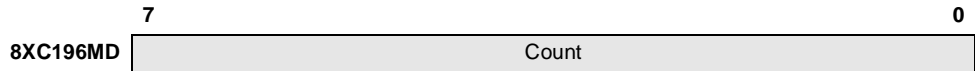
<p>EPA_x_TIME x = 0–1 (8XC196MH) x = 0–3 (8XC196MC) x = 0–5 (8XC196MD)</p> <p>The EPA time (EPA_x_TIME) registers are the event-time registers for the EPA channels. In capture mode, the value of the reference timer is captured in EPA_x_TIME when an input transition occurs. Each event-time register is 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.</p>	<p>Address: Table C-5 Reset State:</p>				
<div style="display: flex; justify-content: space-between; width: 100%;"> 15 0 </div> <div style="background-color: #e0e0e0; text-align: center; padding: 5px; margin-top: 5px;">EPA Timer Value</div>					
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 15%;">Bit Number</th> <th>Function</th> </tr> </thead> <tbody> <tr> <td style="text-align: center; vertical-align: top;">15:0</td> <td> <p>EPA Timer Value</p> <p>When an EPA channel is configured for capture mode, this register contains the value of the reference timer when the specified event occurred.</p> <p>When an EPA channel is configured for compare mode, write the compare event time to this register.</p> </td> </tr> </tbody> </table>		Bit Number	Function	15:0	<p>EPA Timer Value</p> <p>When an EPA channel is configured for capture mode, this register contains the value of the reference timer when the specified event occurred.</p> <p>When an EPA channel is configured for compare mode, write the compare event time to this register.</p>
Bit Number	Function				
15:0	<p>EPA Timer Value</p> <p>When an EPA channel is configured for capture mode, this register contains the value of the reference timer when the specified event occurred.</p> <p>When an EPA channel is configured for compare mode, write the compare event time to this register.</p>				

Table C-5. EPA_x_TIME Addresses and Reset Values

Register	Address	Reset Value
EPA0_TIME (8XC196Mx)	1F42H	XXXXH
EPA1_TIME (8XC196Mx)	1F46H	XXXXH
EPA2_TIME (8XC196MC, MD)	1F4AH	XXXXH
EPA3_TIME (8XC196MC, MD)	1F4EH	XXXXH
EPA4_TIME (8XC196MD)	1F52H	XXXXH
EPA5_TIME (8XC196MD)	1F56H	XX00H

FREQ_CNT**FREQ_CNT**
(8XC196MD)Address: 1FBAH
Reset State: 00H

Read the frequency generator count (FREQ_CNT) register to determine the current value of the down-counter.



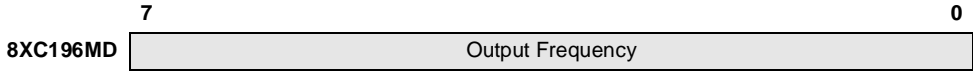
Bit Number	Function
7:0	Count This register contains the current down-counter value.

FREQ_GEN

FREQ_GEN
(8XC196MD)

Address: 1FB8H
Reset State: 00H

The frequency (FREQ_GEN) register holds a programmed value that specifies the output frequency. This value is reloaded into the down-counter each time the counter reaches 0.



Bit Number	Function
7:0	<p>Output Frequency</p> <p>Use the following formula to calculate the FREQ value for the desired output frequency and write this value to the frequency register.</p> $FREQ = \frac{F_{XTAL1}}{16 \times FREQ_OUT} - 1$ <p>where:</p> <p>FREQ = 8-bit value to load into FREQ_GEN register F_{XTAL1} = input frequency on XTAL1 pin, in MHz FREQ_OUT = output frequency on FREQOUT pin, in MHz</p>

GEN_CON

GEN_CON
(8XC196MH)

Address: 1FA0H
Reset State: 00H

The GEN_CON register controls whether an internal reset asserts the external RESET# signal and indicates the source of the most recent reset.



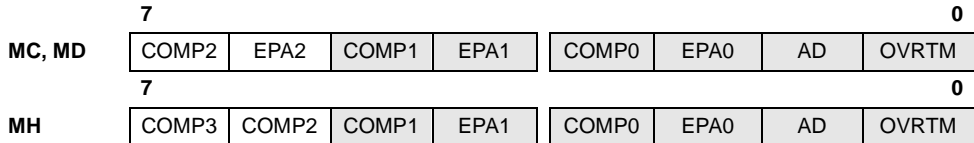
Bit Number	Bit Mnemonic	Function
7	RSTS	Reset source (read-only status bit) 0 = external reset (RESET# pin asserted) 1 = internal reset (watchdog overflow, illegal IDLPD key, or RST instruction)
6:1	—	Reserved; for compatibility with future devices, write zeros to these bits.
0	DRO	Disable RESET# out 0 = an internal reset asserts the RESET# pin. 1 = an internal reset has no effect on the RESET# pin; the RESET# pin is pulled high (inactive).

INT_MASK

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 this instruction. POPF or POPA restores it.



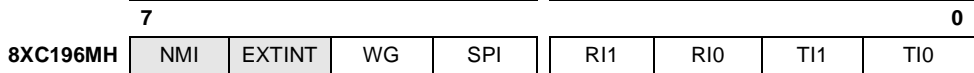
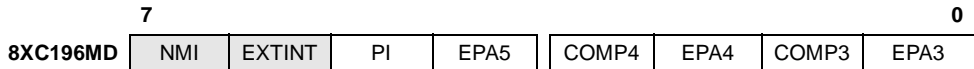
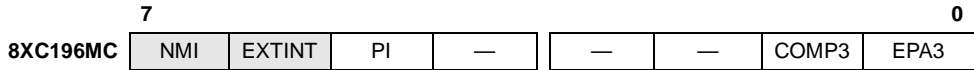
Bit Number	Function																																	
7:0	<p>Setting a bit enables the corresponding interrupt.</p> <p>The standard interrupt vector locations are as follows:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Bit Mnemonic</th> <th style="text-align: left;">Interrupt</th> <th style="text-align: left;">Standard Vector</th> </tr> </thead> <tbody> <tr> <td>COMP2 (MC, MD)</td> <td>EPA Compare-only Channel 2</td> <td>200EH</td> </tr> <tr> <td>COMP3 (MH)</td> <td>EPA Compare-only Channel 3</td> <td>200EH</td> </tr> <tr> <td>EPA2 (MC, MD)</td> <td>EPA Capture/Compare Channel 2</td> <td>200CH</td> </tr> <tr> <td>COMP2 (MH)</td> <td>EPA Compare Channel 2</td> <td>200EH</td> </tr> <tr> <td>COMP1</td> <td>EPA Compare Channel 1</td> <td>200AH</td> </tr> <tr> <td>EPA1</td> <td>EPA Capture/Compare Channel 1</td> <td>2008H</td> </tr> <tr> <td>COMP0</td> <td>EPA Compare Channel 0</td> <td>2006H</td> </tr> <tr> <td>EPA0</td> <td>EPA Capture/Compare Channel 0</td> <td>2004H</td> </tr> <tr> <td>AD</td> <td>A/D Conversion Complete</td> <td>2002H</td> </tr> <tr> <td>OVRTM[†]</td> <td>Overflow/Underflow Timer</td> <td>2000H</td> </tr> </tbody> </table> <p>[†] Both timer 1 and timer 2 can generate the multiplexed overflow/underflow interrupt. Write to PI_MASK to enable the interrupt sources; read PI_PEND to determine which source caused the interrupt.</p>	Bit Mnemonic	Interrupt	Standard Vector	COMP2 (MC, MD)	EPA Compare-only Channel 2	200EH	COMP3 (MH)	EPA Compare-only Channel 3	200EH	EPA2 (MC, MD)	EPA Capture/Compare Channel 2	200CH	COMP2 (MH)	EPA Compare Channel 2	200EH	COMP1	EPA Compare Channel 1	200AH	EPA1	EPA Capture/Compare Channel 1	2008H	COMP0	EPA Compare Channel 0	2006H	EPA0	EPA Capture/Compare Channel 0	2004H	AD	A/D Conversion Complete	2002H	OVRTM [†]	Overflow/Underflow Timer	2000H
Bit Mnemonic	Interrupt	Standard Vector																																
COMP2 (MC, MD)	EPA Compare-only Channel 2	200EH																																
COMP3 (MH)	EPA Compare-only Channel 3	200EH																																
EPA2 (MC, MD)	EPA Capture/Compare Channel 2	200CH																																
COMP2 (MH)	EPA Compare Channel 2	200EH																																
COMP1	EPA Compare Channel 1	200AH																																
EPA1	EPA Capture/Compare Channel 1	2008H																																
COMP0	EPA Compare Channel 0	2006H																																
EPA0	EPA Capture/Compare Channel 0	2004H																																
AD	A/D Conversion Complete	2002H																																
OVRTM [†]	Overflow/Underflow Timer	2000H																																

INT_MASK1

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.



Bit Number	Function																																													
7:0 [†]	<p>Setting a bit enables the corresponding interrupt. The standard interrupt vector locations are as follows:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Bit Mnemonic</th> <th style="text-align: left;">Interrupt</th> <th style="text-align: left;">Standard Vector</th> </tr> </thead> <tbody> <tr> <td>NMI</td> <td>Nonmaskable Interrupt</td> <td>203EH</td> </tr> <tr> <td>EXTINT</td> <td>EXTINT pin</td> <td>203CH</td> </tr> <tr> <td>PI (MC, MD)^{††}</td> <td>Multiplexed Peripheral Interrupt</td> <td>203AH</td> </tr> <tr> <td>WG (MH)</td> <td>Waveform Generator</td> <td>203AH</td> </tr> <tr> <td>EPA5 (MD)</td> <td>EPA Capture/Compare Channel 5</td> <td>2038H</td> </tr> <tr> <td>SPI (MH)^{†††}</td> <td>Serial Port</td> <td>2038H</td> </tr> <tr> <td>COMP4 (MD)</td> <td>EPA Compare Channel 4</td> <td>2036H</td> </tr> <tr> <td>RI1 (MH)</td> <td>SIO 1 Receive</td> <td>2036H</td> </tr> <tr> <td>EPA4 (MD)</td> <td>EPA Capture/Compare Channel 4</td> <td>2034H</td> </tr> <tr> <td>RI0 (MH)</td> <td>SIO 0 Receive</td> <td>2034H</td> </tr> <tr> <td>COMP3 (MC, MD)</td> <td>EPA Compare Channel 3</td> <td>2032H</td> </tr> <tr> <td>TI1 (MH)</td> <td>SIO 1 Transmit</td> <td>2032H</td> </tr> <tr> <td>EPA3 (MC, MD)</td> <td>EPA Capture/Compare Channel 3</td> <td>2030H</td> </tr> <tr> <td>TI0 (MH)</td> <td>SIO 0 Transmit</td> <td>2030H</td> </tr> </tbody> </table> <p>^{††} The waveform generator and the EPA compare-only channel 5 can generate this interrupt. Write to PI_MASK to enable the interrupt sources; read PI_PEND to determine which source caused the interrupt.</p> <p>^{†††} SIO 0 and SIO 1 can generate this interrupt. Write to PI_MASK to enable the interrupt sources; read PI_PEND to determine which source caused the interrupt.</p>	Bit Mnemonic	Interrupt	Standard Vector	NMI	Nonmaskable Interrupt	203EH	EXTINT	EXTINT pin	203CH	PI (MC, MD) ^{††}	Multiplexed Peripheral Interrupt	203AH	WG (MH)	Waveform Generator	203AH	EPA5 (MD)	EPA Capture/Compare Channel 5	2038H	SPI (MH) ^{†††}	Serial Port	2038H	COMP4 (MD)	EPA Compare Channel 4	2036H	RI1 (MH)	SIO 1 Receive	2036H	EPA4 (MD)	EPA Capture/Compare Channel 4	2034H	RI0 (MH)	SIO 0 Receive	2034H	COMP3 (MC, MD)	EPA Compare Channel 3	2032H	TI1 (MH)	SIO 1 Transmit	2032H	EPA3 (MC, MD)	EPA Capture/Compare Channel 3	2030H	TI0 (MH)	SIO 0 Transmit	2030H
Bit Mnemonic	Interrupt	Standard Vector																																												
NMI	Nonmaskable Interrupt	203EH																																												
EXTINT	EXTINT pin	203CH																																												
PI (MC, MD) ^{††}	Multiplexed Peripheral Interrupt	203AH																																												
WG (MH)	Waveform Generator	203AH																																												
EPA5 (MD)	EPA Capture/Compare Channel 5	2038H																																												
SPI (MH) ^{†††}	Serial Port	2038H																																												
COMP4 (MD)	EPA Compare Channel 4	2036H																																												
RI1 (MH)	SIO 1 Receive	2036H																																												
EPA4 (MD)	EPA Capture/Compare Channel 4	2034H																																												
RI0 (MH)	SIO 0 Receive	2034H																																												
COMP3 (MC, MD)	EPA Compare Channel 3	2032H																																												
TI1 (MH)	SIO 1 Transmit	2032H																																												
EPA3 (MC, MD)	EPA Capture/Compare Channel 3	2030H																																												
TI0 (MH)	SIO 0 Transmit	2030H																																												

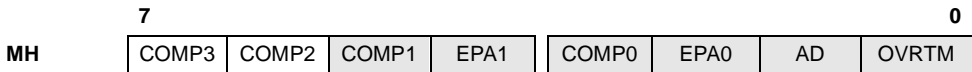
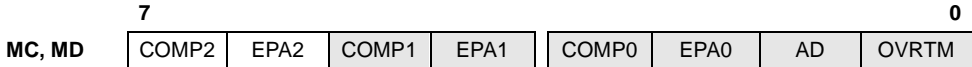
[†] On the 8XC196MC device bits 4–3 are reserved. For compatibility with future devices, write zeros to these bits.

INT_PEND

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.



Bit Number	Function																																	
7:0	<p>Any set bit indicates that the corresponding interrupt is pending. The interrupt bit is cleared when processing transfers to the corresponding interrupt vector.</p> <p>The standard interrupt vector locations are as follows:</p> <table border="1"> <thead> <tr> <th style="text-align: left;">Bit Mnemonic</th> <th style="text-align: left;">Interrupt</th> <th style="text-align: left;">Standard Vector</th> </tr> </thead> <tbody> <tr> <td>COMP2 (MC, MD)</td> <td>EPA Compare Channel 2</td> <td>200EH</td> </tr> <tr> <td>COMP3 (MH)</td> <td>EPA Compare Channel 3</td> <td>200EH</td> </tr> <tr> <td>EPA2 (MC, MD)</td> <td>EPA Capture/Compare Channel 2</td> <td>200CH</td> </tr> <tr> <td>COMP2 (MH)</td> <td>EPA Compare Channel 2</td> <td>200EH</td> </tr> <tr> <td>COMP1</td> <td>EPA Compare Channel 1</td> <td>200AH</td> </tr> <tr> <td>EPA1</td> <td>EPA Capture/Compare Channel 1</td> <td>2008H</td> </tr> <tr> <td>COMP0</td> <td>EPA Compare Channel 0</td> <td>2006H</td> </tr> <tr> <td>EPA0</td> <td>EPA Capture/Compare Channel 0</td> <td>2004H</td> </tr> <tr> <td>AD</td> <td>A/D Conversion Complete</td> <td>2002H</td> </tr> <tr> <td>OVRTM[†]</td> <td>Overflow/Underflow Timer</td> <td>2000H</td> </tr> </tbody> </table> <p>[†] Timer 1 and timer 2 can generate the multiplexed overflow/underflow interrupt. Write to PI_MASK to enable the interrupt sources; read PI_PEND to determine which source caused the interrupt.</p>	Bit Mnemonic	Interrupt	Standard Vector	COMP2 (MC, MD)	EPA Compare Channel 2	200EH	COMP3 (MH)	EPA Compare Channel 3	200EH	EPA2 (MC, MD)	EPA Capture/Compare Channel 2	200CH	COMP2 (MH)	EPA Compare Channel 2	200EH	COMP1	EPA Compare Channel 1	200AH	EPA1	EPA Capture/Compare Channel 1	2008H	COMP0	EPA Compare Channel 0	2006H	EPA0	EPA Capture/Compare Channel 0	2004H	AD	A/D Conversion Complete	2002H	OVRTM [†]	Overflow/Underflow Timer	2000H
Bit Mnemonic	Interrupt	Standard Vector																																
COMP2 (MC, MD)	EPA Compare Channel 2	200EH																																
COMP3 (MH)	EPA Compare Channel 3	200EH																																
EPA2 (MC, MD)	EPA Capture/Compare Channel 2	200CH																																
COMP2 (MH)	EPA Compare Channel 2	200EH																																
COMP1	EPA Compare Channel 1	200AH																																
EPA1	EPA Capture/Compare Channel 1	2008H																																
COMP0	EPA Compare Channel 0	2006H																																
EPA0	EPA Capture/Compare Channel 0	2004H																																
AD	A/D Conversion Complete	2002H																																
OVRTM [†]	Overflow/Underflow Timer	2000H																																

INT_PEND1

INT_PEND1

Address: 0012H
Reset State: 00H

When hardware detects a pending interrupt, 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								
8XC196MC	<table border="1" style="width: 100%; text-align: center;"> <tr> <td>NMI</td> <td>EXTINT</td> <td>PI</td> <td>—</td> <td>—</td> <td>—</td> <td>COMP3</td> <td>EPA3</td> </tr> </table>	NMI	EXTINT	PI	—	—	—	COMP3	EPA3
NMI	EXTINT	PI	—	—	—	COMP3	EPA3		

7	0								
8XC196MD	<table border="1" style="width: 100%; text-align: center;"> <tr> <td>NMI</td> <td>EXTINT</td> <td>PI</td> <td>EPA5</td> <td>COMP4</td> <td>EPA4</td> <td>COMP3</td> <td>EPA3</td> </tr> </table>	NMI	EXTINT	PI	EPA5	COMP4	EPA4	COMP3	EPA3
NMI	EXTINT	PI	EPA5	COMP4	EPA4	COMP3	EPA3		

7	0								
8XC196MH	<table border="1" style="width: 100%; text-align: center;"> <tr> <td>NMI</td> <td>EXTINT</td> <td>WG</td> <td>SPI</td> <td>RI1</td> <td>RI0</td> <td>TI1</td> <td>TI0</td> </tr> </table>	NMI	EXTINT	WG	SPI	RI1	RI0	TI1	TI0
NMI	EXTINT	WG	SPI	RI1	RI0	TI1	TI0		

Bit Number	Function																																													
7:0 [†]	Setting a bit enables the corresponding interrupt. The standard interrupt vector locations are as follows: <table border="1" style="width: 100%; margin-top: 10px;"> <thead> <tr> <th style="text-align: left;">Bit Mnemonic</th> <th style="text-align: left;">Interrupt</th> <th style="text-align: left;">Standard Vector</th> </tr> </thead> <tbody> <tr> <td>NMI</td> <td>Nonmaskable Interrupt</td> <td>203EH</td> </tr> <tr> <td>EXTINT</td> <td>EXTINT pin</td> <td>203CH</td> </tr> <tr> <td>PI (MC, MD)^{††}</td> <td>Multiplexed Peripheral Interrupt</td> <td>203AH</td> </tr> <tr> <td>WG (MH)</td> <td>Waveform Generator</td> <td>203AH</td> </tr> <tr> <td>EPA5 (MD)</td> <td>EPA Capture/Compare Channel 5</td> <td>2038H</td> </tr> <tr> <td>SPI (MH)^{†††}</td> <td>Serial Port</td> <td>2038H</td> </tr> <tr> <td>COMP4 (MD)</td> <td>EPA Compare Channel 4</td> <td>2036H</td> </tr> <tr> <td>RI1 (MH)</td> <td>SIO 1 Receive</td> <td>2036H</td> </tr> <tr> <td>EPA4 (MD)</td> <td>EPA Capture/Compare Channel 4</td> <td>2034H</td> </tr> <tr> <td>RI0 (MH)</td> <td>SIO 0 Receive</td> <td>2034H</td> </tr> <tr> <td>COMP3 (MC, MD)</td> <td>EPA Compare Channel 3</td> <td>2032H</td> </tr> <tr> <td>TI1 (MH)</td> <td>SIO 1 Transmit</td> <td>2032H</td> </tr> <tr> <td>EPA3 (MC, MD)</td> <td>EPA Capture/Compare Channel 3</td> <td>2030H</td> </tr> <tr> <td>TI0 (MH)</td> <td>SIO 0 Transmit</td> <td>2030H</td> </tr> </tbody> </table> <p>^{††} On the 8XC196MD, the waveform generator and the EPA compare channel 5 can generate this interrupt. Write to PI_MASK to enable the interrupt sources; read PI_PEND to determine which source caused the interrupt. On the 8XC196MC, the waveform generator is the sole source for this interrupt.</p> <p>^{†††} SIO 0 and SIO 1 can generate this interrupt. Write to PI_MASK to enable the interrupt sources; read PI_PEND to determine which source caused the interrupt.</p>	Bit Mnemonic	Interrupt	Standard Vector	NMI	Nonmaskable Interrupt	203EH	EXTINT	EXTINT pin	203CH	PI (MC, MD) ^{††}	Multiplexed Peripheral Interrupt	203AH	WG (MH)	Waveform Generator	203AH	EPA5 (MD)	EPA Capture/Compare Channel 5	2038H	SPI (MH) ^{†††}	Serial Port	2038H	COMP4 (MD)	EPA Compare Channel 4	2036H	RI1 (MH)	SIO 1 Receive	2036H	EPA4 (MD)	EPA Capture/Compare Channel 4	2034H	RI0 (MH)	SIO 0 Receive	2034H	COMP3 (MC, MD)	EPA Compare Channel 3	2032H	TI1 (MH)	SIO 1 Transmit	2032H	EPA3 (MC, MD)	EPA Capture/Compare Channel 3	2030H	TI0 (MH)	SIO 0 Transmit	2030H
Bit Mnemonic	Interrupt	Standard Vector																																												
NMI	Nonmaskable Interrupt	203EH																																												
EXTINT	EXTINT pin	203CH																																												
PI (MC, MD) ^{††}	Multiplexed Peripheral Interrupt	203AH																																												
WG (MH)	Waveform Generator	203AH																																												
EPA5 (MD)	EPA Capture/Compare Channel 5	2038H																																												
SPI (MH) ^{†††}	Serial Port	2038H																																												
COMP4 (MD)	EPA Compare Channel 4	2036H																																												
RI1 (MH)	SIO 1 Receive	2036H																																												
EPA4 (MD)	EPA Capture/Compare Channel 4	2034H																																												
RI0 (MH)	SIO 0 Receive	2034H																																												
COMP3 (MC, MD)	EPA Compare Channel 3	2032H																																												
TI1 (MH)	SIO 1 Transmit	2032H																																												
EPA3 (MC, MD)	EPA Capture/Compare Channel 3	2030H																																												
TI0 (MH)	SIO 0 Transmit	2030H																																												

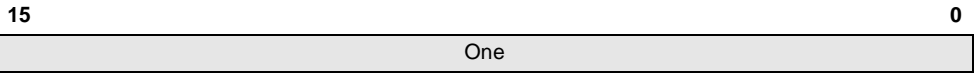
[†] On the 8XC196MC device bits 4–3 are reserved. These bits are undefined.

ONES_REG

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.



Bit Number	Function
15:0	One These bits are always equal to FFFFH.

Px_DIR

Px_DIR

x = 2, 5 (8XC196MC)

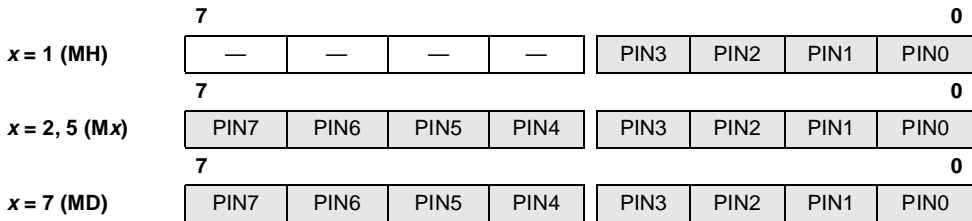
x = 2, 5, 7 (8XC196MD)

x = 1, 2, 5 (8XC196MH)

Address: Table C-6

Reset State:

Each pin of port *x* can operate in any of the standard I/O modes of operation: complementary output, open-drain output, or high-impedance input. The port *x* I/O direction (Px_DIR) register determines the I/O direction for each port *x* pin. The register settings for an open-drain output or a high-impedance input are identical. An open-drain output configuration requires an external pull-up. A high-impedance input configuration requires that the corresponding bit in Px_REG be set.



Bit Number	Bit Mnemonic	Function
7:0 [†]	PIN7:0	Port <i>x</i> Pin <i>y</i> Direction This bit selects the Px.y direction: 0 = complementary output (output only) 1 = input or open-drain output (input, output, or bidirectional open-drain outputs require external pull-ups.

[†] The bits shown as dashes (—) are reserved; for compatibility with future devices, write ones to these bits.

Table C-6. Px_DIR Addresses and Reset Values

Register	Address	Reset Value
P1_DIR (8XC196MH)	1F9BH	FFH
P2_DIR (8XC196Mx)	1FD2H	FFH
P5_DIR (8XC196Mx)	1FF3H	FFH
P7_DIR (8XC196MD)	1FD3H	FFH

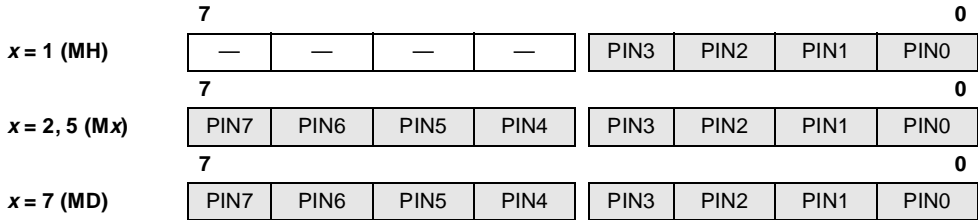
Px_MODE

Px_MODE

x = 2, 5 (8XC196MC)
x = 2, 5, 7 (8XC196MD)
x = 1, 2, 5 (8XC196MH)

Address: Table C-7
 Reset State:

Each bit of the port x mode (Px_MODE) register controls whether the corresponding pin functions as a standard I/O port pin or as a special-function signal.



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 = standard I/O port pin 1 = special-function signal Table C-8 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.

Table C-7. Px_MODE Addresses and Reset Values

Register	Address	Reset Value
P1_MODE (8XC196MH)	1F99H	00H
P2_MODE (8XC196Mx)	1FD0H	00H
P5_MODE (8XC196MC, MD) P5_MODE (8XC196MH)	1FF1H 1FF1H	FFH when pin is not driven 80H if the EA# pin is high, A9H if EA# is low
P7_MODE (8XC196MD)	1FD1H	00H

Px_MODE

Table C-8. Special-function Signals for Ports 1, 2, 5, 6

Port 1 (8XC196MH)		Port 2 (8XC196MC, MD)		Port 2 (8XC196MH)	
Pin	Special-function Signal	Pin	Special-function Signal	Pin	Special-function Signal
P1.0	TXD0	P2.0	EPA0/PVER	P2.0	EPA0/PVER
P1.1	RXD0	P2.1	EPA1/PALE#	P2.1	SCLK0#/BCLK0/PALE#
P1.2	TXD1	P2.2	EPA2/PROG#	P2.2	EPA1/PROG#
P1.3	RXD1	P2.3	EPA3	P2.3	COMP3
		P2.4	COMP0/AINC#	P2.4	COMP0/AINC#
		P2.5	COMP1/PACT#	P2.5	COMP1/PACT#
		P2.6	COMP2/CPVER	P2.6	COMP2/CPVER
		P2.7	COMP3	P2.7	SCLK1#BCLK1

Port 5 (8XC196Mx)		Port 7 (8XC196MD)	
Pin	Special-function Signal	Pin	Special-function Signal
P5.0	ALE/ADV#	P7.0	EPA4
P5.1	INST	P7.1	EPA5
P5.2	WR#/WRL#	P7.2	COMP4
P5.3	RD#	P7.3	COMP5
P5.4	ONCE#	P7.4	—
P5.5	BHE#/WRH#	P7.5	—
P5.6	READY	P7.6	—
P5.7	BUSWIDTH	P7.7	FREQOUT

Px_PIN

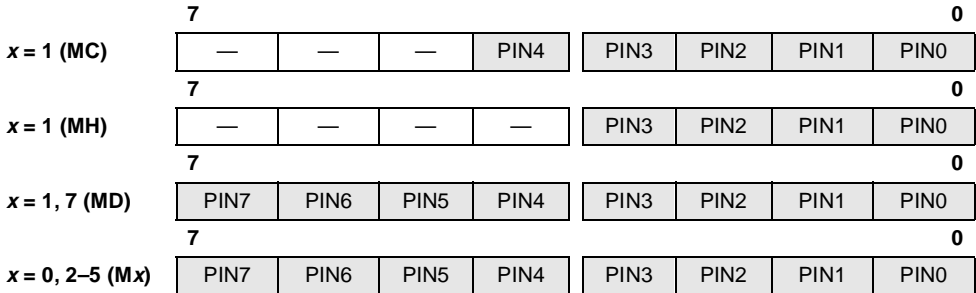
Px_PIN

x = 0–5 (8XC196MC, MH)

x = 0–5, 7 (8XC196MD)

Address: Table C-9
Reset State:

Each bit of the port x pin input (Px_PIN) register reflects the current state of the corresponding pin, regardless of the pin configuration.



Bit Number	Bit Mnemonic	Function
7:0†	PIN7:0	Port x Pin y Input Value This bit contains the current state of Px.y.

† The bits shown as dashes (—) are reserved; their values are undefined.

Table C-9. Px_PIN Addresses and Reset Values

Register	Address	Reset Value
P0_PIN (8XC196MC, MD) P0_PIN (8XC196MH)	1FA8H 1FDAH	FFH when pin is not driven
P1_PIN (8XC196MC, MD) P1_PIN (8XC196MH)	1FA9H 1F9FH	FFH when pin is not driven
P2_PIN (8XC196Mx)	1FD6H	FFH when pin is not driven
P3_PIN (8XC196Mx)	1FFEh	FFH when pin is not driven
P4_PIN (8XC196Mx)	1FFFH	FFH when pin is not driven
P5_PIN (8XC196MC, MD) P5_PIN (8XC196MH)	1FF7H 1FF7H	FFH FFH when pin is not driven
P7_PIN (8XC196MD)	1FD7H	XXH

Px_REG

Px_REG

x = 2–5 (8XC196MC)

x = 2–5, 7 (8XC196MD)

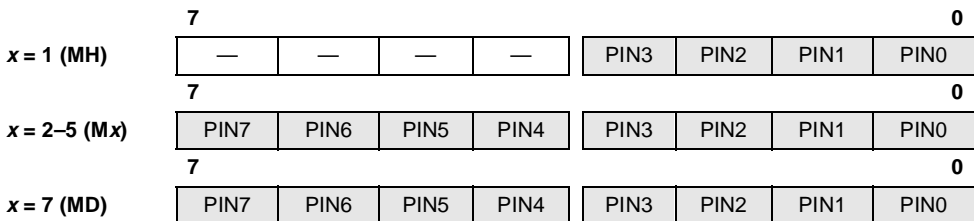
x = 1–5 (8XC196MH)

Address: Table C-10
Reset State:

For an input, set the corresponding port x data output (Px_REG) register bit.

For an output, write the data to be driven out by each pin to the corresponding bit of Px_REG. When a pin is configured as standard I/O (Px_MODE.y = 0), the result of a CPU write to Px_REG is immediately visible on the pin. When a pin is configured as a special-function signal (Px_MODE.y = 1), the associated on-chip peripheral or off-chip component controls the pin. The CPU can still write to Px_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 standard I/O (clear Px_MODE.y), initialize or overwrite the pin value, then configure the pin as a special-function signal (set Px_MODE.y). In this way, initialization, fault recovery, exception handling, etc., can be done without changing the operation of the associated peripheral.



Bit Number	Bit Mnemonic	Function
7:0 [†]	PIN7:0	Port x Pin y Output To use Px.y for output, write the desired output data to this bit. To use Px.y for input, set this bit.

[†] The bits shown as dashes (—) are reserved; for compatibility with future devices, write zeros to these bits.

Table C-10. Px_REG Addresses and Reset Values

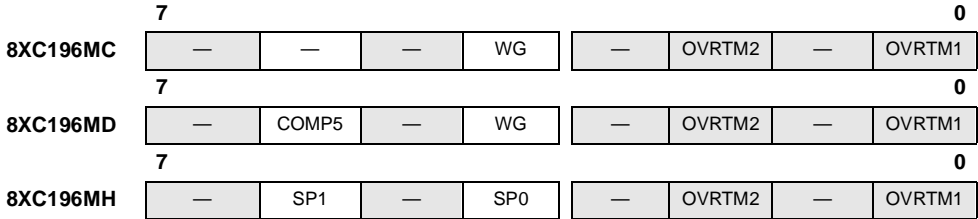
Register	Address	Reset Value
P1_REG (8XC196MH)	1F9DH	FFH
P2_REG (8XC196Mx)	1FD4H	FFH
P3_REG (8XC196Mx)	1FFCH	FFH
P4_REG (8XC196Mx)	1FFDH	FFH
P5_REG (8XC196MC, MD) P5_REG (8XC196MH)	1FF5H 1FF5H	FFH when pin is not driven FFH
P7_REG (8XC196MD)	1FD5H	FFH

PI_MASK

PI_MASK

Address: 1FBCH
Reset State: AAH

The peripheral interrupt mask (PI_MASK) register enables or disables (masks) interrupt requests associated with the peripheral interrupt (PI), the serial port interrupt (SPI), and the overflow/underflow timer interrupt (OVRTM).



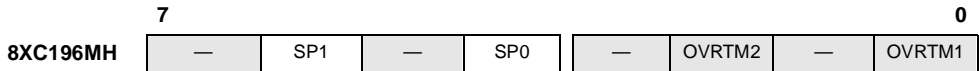
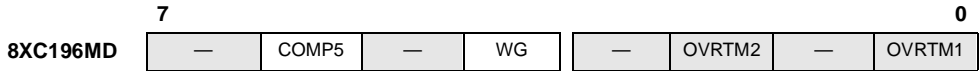
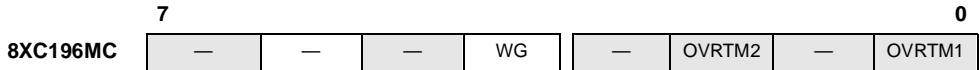
Bit Number	Bit Mnemonic	Function
7, 5, 3, 1	—	Reserved; for compatibility with future devices, write zeros to these bits.
6	— (MC)	Reserved; for compatibility with future devices, write zero to this bit.
	COMP5 (MD)	EPA Compare Channel 5 Setting this bit enables the EPA compare channel 5 interrupt. The EPA compare channel 5 and the waveform generator interrupts are associated with the peripheral interrupt (PI). Setting INT_MASK1.5 enables PI.
	SP1 (MH)	Serial Port 1 Error Setting this bit enables the serial port 1 error interrupt. The serial port 1 and serial port 0 error interrupts are associated with the serial port interrupt (SPI). Setting INT_MASK1.4 enables SPI.
4	WG (MC, MD)	Waveform Generator Setting this bit enables the waveform generator interrupt. The waveform generator and the EPA compare channel 5 interrupts are associated with the peripheral interrupt (PI). Setting INT_MASK1.5 enables PI.
	SP0 (MH)	Serial Port 0 Error Setting this bit enables the serial port 0 error interrupt. The serial port 0 and serial port 1 error interrupts are associated with the serial port interrupt (SPI). Setting INT_MASK1.4 enables SPI.
2	OVRTM2	Timer 2 Overflow/Underflow Setting this bit enables the timer 2 overflow/underflow interrupt. The timer 2 and timer 1 overflow/underflow interrupts are associated with the overflow/underflow timer interrupt (OVRTM). Setting INT_MASK.0 enables OVRTM.

PI_MASK

PI_MASK (Continued)

Address: 1FBCH
Reset State: AAH

The peripheral interrupt mask (PI_MASK) register enables or disables (masks) interrupt requests associated with the peripheral interrupt (PI), the serial port interrupt (SPI), and the overflow/underflow timer interrupt (OVRTM).



Bit Number	Bit Mnemonic	Function
0	OVRTM1	Timer 1 Overflow/Underflow Setting this bit enables the timer 1 overflow/underflow interrupt. The timer 1 and timer 2 overflow/underflow interrupts are associated with the overflow/underflow timer interrupt (OVRTM). Setting INT_MASK.0 enables OVRTM.

PI_PEND

PI_PEND

Address: 1FBEH
Reset State: AAH

When hardware detects a pending peripheral or timer interrupt, it sets the corresponding bit in the interrupt pending (INT_PEND or INT_PEND1) registers and the peripheral interrupt pending (PI_PEND) register. When the vector is taken, the hardware clears the INT_PEND/INT_PEND1 pending bit. Reading this register clears all the PI_PEND bits. Software can generate an interrupt by setting a PI_PEND bit.

	7	0								
8XC196MC	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 25%; text-align: center;">—</td> <td style="width: 25%; text-align: center;">—</td> <td style="width: 25%; text-align: center;">—</td> <td style="width: 25%; text-align: center;">WG</td> </tr> </table>	—	—	—	WG	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 25%; text-align: center;">—</td> <td style="width: 25%; text-align: center;">OVRTM2</td> <td style="width: 25%; text-align: center;">—</td> <td style="width: 25%; text-align: center;">OVRTM1</td> </tr> </table>	—	OVRTM2	—	OVRTM1
—	—	—	WG							
—	OVRTM2	—	OVRTM1							
	7	0								
8XC196MD	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 25%; text-align: center;">—</td> <td style="width: 25%; text-align: center;">COMP5</td> <td style="width: 25%; text-align: center;">—</td> <td style="width: 25%; text-align: center;">WG</td> </tr> </table>	—	COMP5	—	WG	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 25%; text-align: center;">—</td> <td style="width: 25%; text-align: center;">OVRTM2</td> <td style="width: 25%; text-align: center;">—</td> <td style="width: 25%; text-align: center;">OVRTM1</td> </tr> </table>	—	OVRTM2	—	OVRTM1
—	COMP5	—	WG							
—	OVRTM2	—	OVRTM1							
	7	0								
8XC196MH	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 25%; text-align: center;">—</td> <td style="width: 25%; text-align: center;">SP1</td> <td style="width: 25%; text-align: center;">—</td> <td style="width: 25%; text-align: center;">SP0</td> </tr> </table>	—	SP1	—	SP0	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 25%; text-align: center;">—</td> <td style="width: 25%; text-align: center;">OVRTM2</td> <td style="width: 25%; text-align: center;">—</td> <td style="width: 25%; text-align: center;">OVRTM1</td> </tr> </table>	—	OVRTM2	—	OVRTM1
—	SP1	—	SP0							
—	OVRTM2	—	OVRTM1							

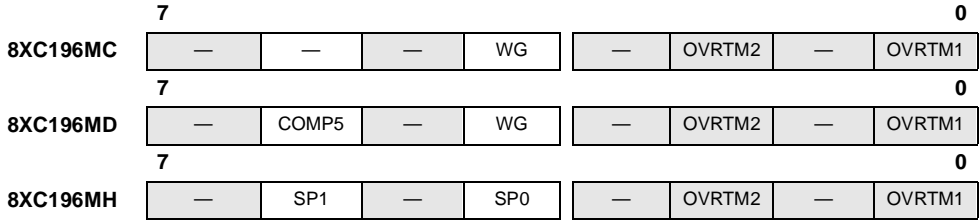
Bit Number	Bit Mnemonic	Function
7, 5, 3, 1	—	Reserved. These bits are undefined.
6	— (MC)	Reserved. This bit is undefined.
	COMP5 (MD)	EPA Compare Channel 5 When set, this bit indicates a pending EPA compare channel 5 interrupt. The EPA compare channel 5 and the waveform generator interrupts are associated with the peripheral interrupt (PI). Setting INT_MASK1.5 enables PI. Setting PI_MASK.6 enables COMP5.
	SP1 (MH)	Serial Port 1 Error When set, this bit indicates a pending serial port 1 error interrupt. The serial port 1 and 0 error interrupts are associated with the serial port interrupt (SPI). Setting INT_MASK1.4 enables SPI. Setting PI_MASK.6 enables SP1.
4	WG (MC, MD)	Waveform Generator When set, this bit indicates a pending waveform generator interrupt. The waveform generator and the EPA compare channel 5 interrupts are associated with the peripheral interrupt (PI). Setting INT_MASK1.5 enables PI. Setting PI_MASK.5 enables WG.
	SP0 (MH)	Serial Port 0 Error When set, this bit indicates a pending serial port 0 error interrupt. The serial port 0 and 1 error interrupts are associated with the serial port interrupt (SPI). Setting INT_MASK1.4 enables SPI. Setting PI_MASK.4 enables SP0.
2	OVRTM2	Timer 2 Overflow/Underflow When set, this bit indicates a pending timer 2 overflow/underflow interrupt. The timer 2 and timer 1 overflow/underflow interrupts are associated with the overflow/underflow timer interrupt (OVRTM). Setting INT_MASK.0 enables OVRTM. Setting PI_MASK.2 enables OVRTM2.

PI_PEND

PI_PEND (Continued)

Address: 1FBEH
Reset State: AAH

When hardware detects a pending peripheral or timer interrupt, it sets the corresponding bit in the interrupt pending (INT_PEND or INT_PEND1) registers and the peripheral interrupt pending (PI_PEND) register. When the vector is taken, the hardware clears the INT_PEND/INT_PEND1 pending bit. Reading this register clears all the PI_PEND bits. Software can generate an interrupt by setting a PI_PEND bit.



Bit Number	Bit Mnemonic	Function
0	OVRTM1	Timer 1 Overflow/Underflow When set, this bit indicates a pending timer 1 overflow/underflow interrupt. The timer 1 and timer 2 overflow/underflow interrupts are associated with the overflow/underflow timer interrupt (OVRTM). Setting INT_MASK.0 enables OVRTM. Setting PI_MASK.0 enables OVRTM1.

PPW

PPW

no direct access

The programming pulse width (PPW) register is loaded from the external EPROM (locations 14H and 15H for the 8XC196MC and MD; locations 4014H and 4015H for the 8XC196MH) in auto programming mode. The PPW_VALUE determines the programming pulse width.

15

8

PPW15	PPW14	PPW13	PPW12	PPW11	PPW10	PPW9	PPW8
-------	-------	-------	-------	-------	-------	------	------

7

0

PPW7	PPW6	PPW5	PPW4	PPW3	PPW2	PPW1	PPW0
------	------	------	------	------	------	------	------

Bit Number	Bit Mnemonic	Function
15:0	PPW15:0	<p>PPW_VALUE</p> <p>This value establishes the programming pulse width for auto programming. Use the appropriate formula to calculate the PPW_VALUE, then write the result to the PPW register.</p> <p style="text-align: center;"> $PPW_VALUE \text{ for } 8XC196MC, MD = 62.5 \times F_{XTAL1}$ $PPW_VALUE \text{ for } 8XC196MH = 25 \times F_{XTAL1}$ </p>

PSW

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.

15

8

Z	N	V	VT	C	PSE	I	ST
---	---	---	----	---	-----	---	----

7

0

See INT_MASK on page C-25

Bit Number	Bit Mnemonic	Function
15	Z	<p>Zero Flag</p> <p>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.</p>
14	N	<p>Negative Flag</p> <p>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.</p>
13	V	<p>Overflow Flag</p> <p>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.)</p>

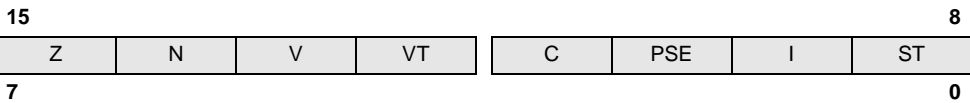
PSW

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.



See INT_MASK on page C-25

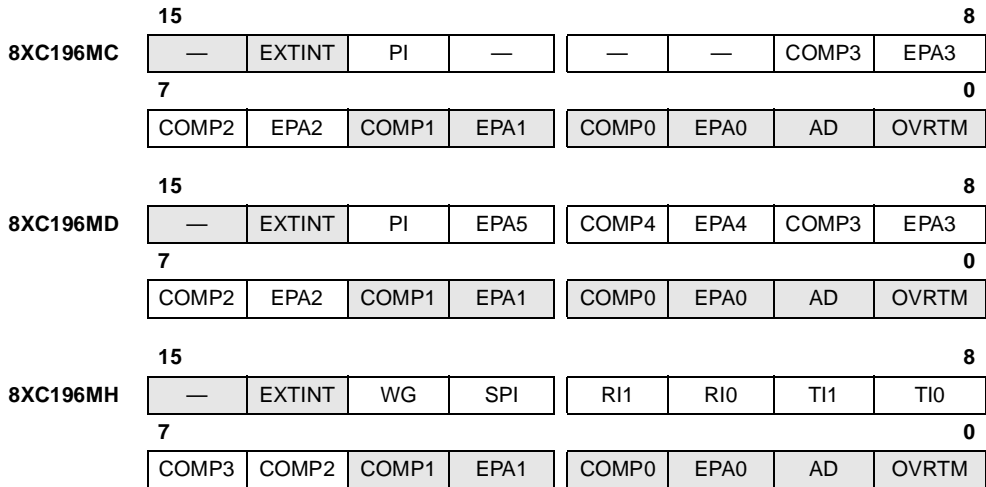
Bit Number	Bit Mnemonic	Function
12	VT	<p>Overflow-trap Flag</p> <p>This flag is set when the overflow flag is set, but it is cleared only by the CLRVT, JVT, and JNVT instructions. This allows testing for a possible overflow at the end of a sequence of related arithmetic operations, which is generally more efficient than testing the overflow flag after each operation.</p>
11	C	<p>Carry Flag</p> <p>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.)</p>
10	PSE	<p>PTS Enable</p> <p>This bit globally enables or disables the peripheral transaction server (PTS). The EPTS instruction sets this bit; DPTS clears it.</p> <p>0 = disable PTS 1 = enable PTS</p>
9	I	<p>Interrupt Disable (Global)</p> <p>This bit globally enables or disables the servicing of all <i>maskable interrupts</i>. The bits in INT_MASK and INT_MASK1 individually enable or disable the interrupts. The EI instruction sets this bit; DI clears it.</p> <p>0 = disable interrupt servicing 1 = enable interrupt servicing</p>
8	ST	<p>Sticky Bit Flag</p> <p>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.</p>

PTSSEL

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. When PTSCOUNT reaches zero, hardware clears the corresponding PTSSEL bit. The PTSSEL bit must be set manually to re-enable the PTS channel.



Bit Number	Function																																																				
15	Reserved; for compatibility with future devices, write zero to this bit.																																																				
14:0 [†]	<p>Setting a bit causes the corresponding interrupt to be handled by a PTS microcode routine. The PTS interrupt vector locations are as follows:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Bit Mnemonic</th> <th style="text-align: left;">PTS Vector</th> <th style="text-align: left;">Bit Mnemonic</th> <th style="text-align: left;">PTS Vector</th> </tr> </thead> <tbody> <tr> <td>EXTINT</td> <td>205CH</td> <td>TI0 (MH)</td> <td>2050H</td> </tr> <tr> <td>PI (MC, MD)^{††}</td> <td>205AH</td> <td>COMP2 (MC,MD)</td> <td>204EH</td> </tr> <tr> <td>WG (MH)</td> <td>205AH</td> <td>COMP3 (MH)</td> <td>204EH</td> </tr> <tr> <td>EPA5 (MD)</td> <td>2058H</td> <td>EPA2 (MC, MD)</td> <td>204CH</td> </tr> <tr> <td>SPI (MH)^{††}</td> <td>2058H</td> <td>COMP2 (MH)</td> <td>204CH</td> </tr> <tr> <td>COMP4 (MD)</td> <td>2056H</td> <td>COMP1</td> <td>204AH</td> </tr> <tr> <td>R11 (MH)</td> <td>2056H</td> <td>EPA1</td> <td>2048H</td> </tr> <tr> <td>EPA4 (MD)</td> <td>2054H</td> <td>COMP0</td> <td>2046H</td> </tr> <tr> <td>R10 (MH)</td> <td>2054H</td> <td>EPA0</td> <td>2044H</td> </tr> <tr> <td>COMP3 (MC, MD)</td> <td>2052H</td> <td>AD</td> <td>2042H</td> </tr> <tr> <td>TI1 (MH)</td> <td>2052H</td> <td>OVRTM^{††}</td> <td>2040H</td> </tr> <tr> <td>EPA3 (MC, MD)</td> <td>2050H</td> <td></td> <td></td> </tr> </tbody> </table> <p>^{††} PTS service is not useful for multiplexed interrupts because the PTS cannot readily determine the source of these interrupts.</p>	Bit Mnemonic	PTS Vector	Bit Mnemonic	PTS Vector	EXTINT	205CH	TI0 (MH)	2050H	PI (MC, MD) ^{††}	205AH	COMP2 (MC,MD)	204EH	WG (MH)	205AH	COMP3 (MH)	204EH	EPA5 (MD)	2058H	EPA2 (MC, MD)	204CH	SPI (MH) ^{††}	2058H	COMP2 (MH)	204CH	COMP4 (MD)	2056H	COMP1	204AH	R11 (MH)	2056H	EPA1	2048H	EPA4 (MD)	2054H	COMP0	2046H	R10 (MH)	2054H	EPA0	2044H	COMP3 (MC, MD)	2052H	AD	2042H	TI1 (MH)	2052H	OVRTM ^{††}	2040H	EPA3 (MC, MD)	2050H		
Bit Mnemonic	PTS Vector	Bit Mnemonic	PTS Vector																																																		
EXTINT	205CH	TI0 (MH)	2050H																																																		
PI (MC, MD) ^{††}	205AH	COMP2 (MC,MD)	204EH																																																		
WG (MH)	205AH	COMP3 (MH)	204EH																																																		
EPA5 (MD)	2058H	EPA2 (MC, MD)	204CH																																																		
SPI (MH) ^{††}	2058H	COMP2 (MH)	204CH																																																		
COMP4 (MD)	2056H	COMP1	204AH																																																		
R11 (MH)	2056H	EPA1	2048H																																																		
EPA4 (MD)	2054H	COMP0	2046H																																																		
R10 (MH)	2054H	EPA0	2044H																																																		
COMP3 (MC, MD)	2052H	AD	2042H																																																		
TI1 (MH)	2052H	OVRTM ^{††}	2040H																																																		
EPA3 (MC, MD)	2050H																																																				

[†] On the 8XC196MC device bits 10–12 are reserved. For compatibility with future devices, write zeros to these bits.

PTSSRV

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 PTSSSEL 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 PTSSSEL bit must be set manually to re-enable the PTS channel.

8XC196MC	15	—	EXTINT	PI	—	—	—	COMP3	EPA3	8
	7	COMP2	EPA2	COMP1	EPA1	COMP0	EPA0	AD	OVRTM	0

8XC196MD	15	—	EXTINT	PI	EPA5	COMP4	EPA4	COMP3	EPA3	8
	7	COMP2	EPA2	COMP1	EPA1	COMP0	EPA0	AD	OVRTM	0

8XC196MH	15	—	EXTINT	WG	SPI	RI1	RI0	TI1	TI0	8
	7	COMP3	COMP2	COMP1	EPA1	COMP0	EPA0	AD	OVRTM	0

Bit Number	Function																																																				
15	Reserved. This bit is undefined.																																																				
14:0 [†]	<p>A bit is set by hardware to request an end-of-PTS interrupt for the corresponding interrupt through its standard interrupt vector.</p> <p>The PTS interrupt vector locations are as follows:</p> <table border="1"> <thead> <tr> <th>Bit Mnemonic</th> <th>PTS Vector</th> <th>Bit Mnemonic</th> <th>PTS Vector</th> </tr> </thead> <tbody> <tr> <td>EXTINT</td> <td>205CH</td> <td>TI0 (MH)</td> <td>2050H</td> </tr> <tr> <td>PI (MC, MD)^{††}</td> <td>205AH</td> <td>COMP2 (MC,MD)</td> <td>204EH</td> </tr> <tr> <td>WG (MH)</td> <td>205AH</td> <td>COMP3 (MH)</td> <td>204EH</td> </tr> <tr> <td>EPA5 (MD)</td> <td>2058H</td> <td>EPA2 (MC, MD)</td> <td>204CH</td> </tr> <tr> <td>SPI (MH)^{††}</td> <td>2058H</td> <td>COMP2 (MH)</td> <td>204CH</td> </tr> <tr> <td>COMP4 (MD)</td> <td>2056H</td> <td>COMP1</td> <td>204AH</td> </tr> <tr> <td>RI1 (MH)</td> <td>2056H</td> <td>EPA1</td> <td>2048H</td> </tr> <tr> <td>EPA4 (MD)</td> <td>2054H</td> <td>COMP0</td> <td>2046H</td> </tr> <tr> <td>RI0 (MH)</td> <td>2054H</td> <td>EPA0</td> <td>2044H</td> </tr> <tr> <td>COMP3 (MC, MD)</td> <td>2052H</td> <td>AD</td> <td>2042H</td> </tr> <tr> <td>TI1 (MH)</td> <td>2052H</td> <td>OVRTM^{††}</td> <td>2040H</td> </tr> <tr> <td>EPA3 (MC, MD)</td> <td>2050H</td> <td></td> <td></td> </tr> </tbody> </table> <p>^{††} PTS service is not useful for multiplexed interrupts because the PTS cannot readily determine the source of these interrupts.</p>	Bit Mnemonic	PTS Vector	Bit Mnemonic	PTS Vector	EXTINT	205CH	TI0 (MH)	2050H	PI (MC, MD) ^{††}	205AH	COMP2 (MC,MD)	204EH	WG (MH)	205AH	COMP3 (MH)	204EH	EPA5 (MD)	2058H	EPA2 (MC, MD)	204CH	SPI (MH) ^{††}	2058H	COMP2 (MH)	204CH	COMP4 (MD)	2056H	COMP1	204AH	RI1 (MH)	2056H	EPA1	2048H	EPA4 (MD)	2054H	COMP0	2046H	RI0 (MH)	2054H	EPA0	2044H	COMP3 (MC, MD)	2052H	AD	2042H	TI1 (MH)	2052H	OVRTM ^{††}	2040H	EPA3 (MC, MD)	2050H		
Bit Mnemonic	PTS Vector	Bit Mnemonic	PTS Vector																																																		
EXTINT	205CH	TI0 (MH)	2050H																																																		
PI (MC, MD) ^{††}	205AH	COMP2 (MC,MD)	204EH																																																		
WG (MH)	205AH	COMP3 (MH)	204EH																																																		
EPA5 (MD)	2058H	EPA2 (MC, MD)	204CH																																																		
SPI (MH) ^{††}	2058H	COMP2 (MH)	204CH																																																		
COMP4 (MD)	2056H	COMP1	204AH																																																		
RI1 (MH)	2056H	EPA1	2048H																																																		
EPA4 (MD)	2054H	COMP0	2046H																																																		
RI0 (MH)	2054H	EPA0	2044H																																																		
COMP3 (MC, MD)	2052H	AD	2042H																																																		
TI1 (MH)	2052H	OVRTM ^{††}	2040H																																																		
EPA3 (MC, MD)	2050H																																																				

[†] On the 8XC196MC device bits 10–12 are reserved. These bits are undefined.

PWM_COUNT**PWM_COUNT**
(read only)Address: 1FB6H
Reset State: 00H

The PWM count (PWM_COUNT) register provides the current value of the decremented period counter.

7 **0**

PWM Count Value

Bit Number	Function
7:0	PWM Count Value This register contains the current value of the decremented period counter.

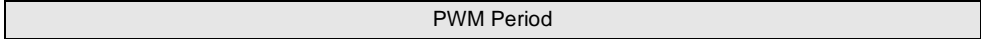
PWM_PERIOD

PWM_PERIOD

Address: 1FB4H
Reset State: 00H

The PWM period (PWM_PERIOD) register controls the period of the PWM outputs. It contains a value that determines the number of state counts necessary for incrementing the PWM counter. The value of PWM_PERIOD is loaded into the PWM period count register whenever the count equals zero.

7 **0**



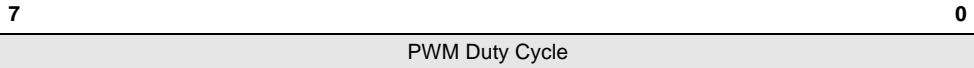
Bit Number	Function
7:0	PWM Period This register controls the period of the PWM outputs. The value of PWM_PERIOD is loaded into the PWM period count register whenever the count equals zero.

PWMx_CONTROL

PWMx_CONTROL
x = 0–1

Address: 1FB0H, 1FB2H
 Reset State: 00H

The PWM control (PWMx_CONTROL) register determines the duty cycle of the PWM x channel. 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).



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).

SBUF_x_RX

SBUF_x_RX
x = 0–1 (8XC196MH)

Address: 1F80H, 1F88H
 Reset State: 00H

The serial port receive buffer *x* (SBUF_{*x*}_RX) register contains data received from serial port *x*. 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_{*x*}_RX. If data in the shift register is loaded into SBUF_{*x*}_RX before the previous byte is read, the overflow error bit is set (SP_{*x*}_STATUS.2). The data in SBUF_{*x*}_RX will always be the last byte received, never a combination of the last two bytes.



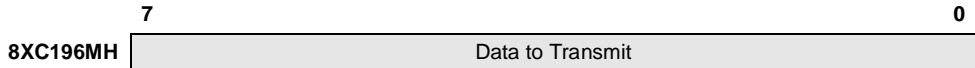
Bit Number	Function
7:0	Data Received This register contains the last byte of data received from the serial port.

SBUF_x_TX

SBUF_x_TX
x = 0–1 (8XC196MH)

Address: 1F82H, 1F8AH
 Reset State: 00H

The serial port transmit buffer *x* (SBUF_x_TX) 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).



Bit Number	Function
7:0	Data to Transmit This register contains a byte of data to be transmitted by the serial port.

SP

SP

Address: 18H
Reset State: XXXXH

The system's stack pointer (SP) can point anywhere in internal or external memory; 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 the stack pointer should be initialized to two bytes (in 64-Kbyte mode) or four bytes (in 1-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



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

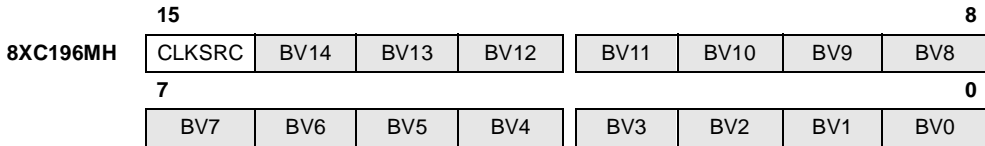
SP_x_BAUD

SP_x_BAUD
x = 0–1 (8XC196MH)

Address: 1F84H, 1F8CH
 Reset State: 0000H

The serial port baud rate *x* (SP_{*x*}_BAUD) register selects the serial port *x* baud rate and clock source. 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 XTAL1 and 0001H when using BCLK_{*x*}. In synchronous mode 0, the minimum BAUD_VALUE is 0001H for transmissions and 0002H for receptions. In synchronous mode 4, the minimum BAUD_VALUE is 0001H for both transmissions and receptions.



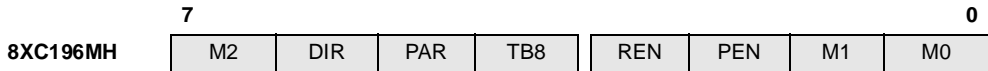
Bit Number	Bit Mnemonic	Function
15	CLKSRC	Serial Port Clock Source This bit determines whether the serial port is clocked from an internal or an external source. 0 = signal on the T1CLK pin (external source) 1 = input frequency on the XTAL1 pin (internal source)
14:0	BV14:0	These bits constitute the BAUD_VALUE. Use the following equations to determine the BAUD_VALUE for a given baud rate. Synchronous mode 0:† $\text{BAUD_VALUE} = \frac{F_{\text{XTAL1}}}{\text{Baud Rate} \times 2} - 1 \quad \text{or} \quad \frac{\text{BCLK}_x}{\text{Baud Rate}}$ Asynchronous modes 1, 2, and 3: $\text{BAUD_VALUE} = \frac{F_{\text{XTAL1}}}{\text{Baud Rate} \times 16} - 1 \quad \text{or} \quad \frac{\text{BCLK}_x}{\text{Baud Rate} \times 8}$ Synchronous mode 4 (SCLK _{<i>x</i>} # output): $\text{BAUD_VALUE} = \frac{F_{\text{XTAL1}}}{\text{Baud Rate} \times 4} - 1$ † For mode 0 receptions, the BAUD_VALUE must be 0002H or greater. Otherwise, the resulting data in the receive shift register will be incorrect.

SPx_CON

SPx_CON
x = 0–1 (8XC196MH)

Address: 1F83H, 1F8BH
 Reset State: 00H

The serial port control (SPx_CON) register selects the communications mode and enables or disables the receiver, parity checking, and nine-bit data transmission.



Bit Number	Bit Mnemonic	Function																								
7	M2	See description for bits 0 and 1.																								
6	DIR	Synchronous Clock Direction This bit determines the direction of the clock during synchronous mode. 0 = output 1 = input																								
5	PAR	Parity Selection Bit This bit selects even or odd parity. 0 = even parity 1 = odd parity																								
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 it must be set before SBUFx_TX is written. When parity is enabled (SPx_CON.2 = 1), this bit takes on the even parity value.																								
3	REN	Receive Enable Setting this bit enables receptions. When this bit is set, a falling edge on the RXDx pin starts a reception in mode 1, 2, or 3. In mode 0, this bit must be clear for transmission to begin and must be set for reception to begin. Clearing this bit stops a reception in progress and inhibits further receptions. To avoid a partial or undesired reception, clear this bit before clearing the RI flag in SPx_STATUS. This can be handled in an interrupt environment by using software flags or in straight-line code by using the interrupt pending register to signal the completion of a reception.																								
2	PEN	Parity Enable In modes 1 and 3, setting this bit enables the parity function. This bit must be cleared if mode 2 is used. When this bit is set, TB8 takes the parity value on transmissions and SPx_STATUS.7 becomes the receive parity error bit.																								
1:0	M1:0	Mode Selection These bits along with bit 7 select the communications mode. <table style="margin-left: 20px; border-collapse: collapse;"> <tr> <td style="text-align: right;">M2</td> <td style="text-align: right;">M1</td> <td style="text-align: right;">M0</td> <td></td> </tr> <tr> <td style="text-align: right;">0</td> <td style="text-align: right;">0</td> <td style="text-align: right;">0</td> <td>synchronous mode 0</td> </tr> <tr> <td style="text-align: right;">X</td> <td style="text-align: right;">0</td> <td style="text-align: right;">1</td> <td>mode 1</td> </tr> <tr> <td style="text-align: right;">X</td> <td style="text-align: right;">1</td> <td style="text-align: right;">0</td> <td>mode 2</td> </tr> <tr> <td style="text-align: right;">X</td> <td style="text-align: right;">1</td> <td style="text-align: right;">1</td> <td>mode 3</td> </tr> <tr> <td style="text-align: right;">1</td> <td style="text-align: right;">0</td> <td style="text-align: right;">0</td> <td>synchronous mode 4</td> </tr> </table>	M2	M1	M0		0	0	0	synchronous mode 0	X	0	1	mode 1	X	1	0	mode 2	X	1	1	mode 3	1	0	0	synchronous mode 4
M2	M1	M0																								
0	0	0	synchronous mode 0																							
X	0	1	mode 1																							
X	1	0	mode 2																							
X	1	1	mode 3																							
1	0	0	synchronous mode 4																							

SP_x_STATUS

SP_x_STATUS
x = 0–1 (8XC196MH)

Address: 1F81H, 1F89H
 Reset State: 00H

The serial port status (SP_x_STATUS) register contains bits that indicate the status of serial port x.

7	0								
8XC196MH	<table border="1" style="display: inline-table;"> <tr> <td>RPE/RB8</td> <td>RI</td> <td>TI</td> <td>FE</td> <td>TXE</td> <td>OE</td> <td>—</td> <td>—</td> </tr> </table>	RPE/RB8	RI	TI	FE	TXE	OE	—	—
RPE/RB8	RI	TI	FE	TXE	OE	—	—		

Bit Number	Bit Mnemonic	Function
7	RPE/RB8	Received Parity Error/Received Bit 8 RPE is set if parity is disabled (SP _x _CON.2 = 0) and the ninth data bit received is high. RB8 is set if parity is enabled (SP _x _CON.2 = 1) and a parity error occurred. Reading SP _x _STATUS clears this bit.
6	RI	Receive Interrupt This bit is set when the last data bit is sampled. Reading SP _x _STATUS clears this bit.
5	TI	Transmit Interrupt This bit is set at the beginning of the stop bit transmission. Reading SP _x _STATUS clears this bit.
4	FE	Framing Error This bit is set if a stop bit is not found within the appropriate period of time. Reading SP _x _STATUS clears this bit.
3	TXE	SBUF _x _TX Empty This bit is set if the transmit buffer is empty and ready to accept up to two bytes. It is cleared when a byte is written to SBUF _x _TX.
2	OE	Overrun Error This bit is set if data in the receive shift register is loaded into SBUF _x _RX before the previous bit is read. Reading SP _x _STATUS clears this bit.
1:0	—	Reserved; for compatibility with future devices, write zeros to these bits.

T1CONTROL

T1CONTROL

Address: 1F78H
Reset State: 00H

The timer 1 control (T1CONTROL) register determines the clock source, counting direction, and count rate for timer 1.

7 0

CE	UD	M2	M1	M0	P2	P1	P0
----	----	----	----	----	----	----	----

Bit Number	Bit Mnemonic	Function																																													
7	CE	<p>Counter Enable</p> <p>This bit enables or disables the timer. From reset, the timers are disabled and not free running.</p> <p>0 = disables timer 1 = enables timer</p>																																													
6	UD	<p>Up/Down</p> <p>This bit determines the timer counting direction, in selected modes (see mode bits, M2:0).</p> <p>0 = count down 1 = count up</p>																																													
5:3	M2:0	<p>EPA Clock Direction Mode Bits</p> <p>These bits determine the timer clocking source and direction control source.</p> <table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">M2</th> <th style="text-align: center;">M1</th> <th style="text-align: center;">M0</th> <th style="text-align: left;">Clock Source</th> <th style="text-align: left;">Direction Source</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td>$F_{XTAL}/4$</td> <td>UD bit (T1CONTROL.6)</td> </tr> <tr> <td style="text-align: center;">X</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td>T1CLK pin[†]</td> <td>UD bit (T1CONTROL.6)</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td>$F_{XTAL}/4$</td> <td>T1DIR pin</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td>T1CLK pin[†]</td> <td>T1DIR pin</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td colspan="2">quadrature clocking using T1CLK and T1DIR</td> </tr> </tbody> </table> <p>[†] If an external clock is selected, the timer counts on both the rising and falling edges of the clock.</p>	M2	M1	M0	Clock Source	Direction Source	0	0	0	$F_{XTAL}/4$	UD bit (T1CONTROL.6)	X	0	1	T1CLK pin [†]	UD bit (T1CONTROL.6)	0	1	0	$F_{XTAL}/4$	T1DIR pin	0	1	1	T1CLK pin [†]	T1DIR pin	1	1	1	quadrature clocking using T1CLK and T1DIR																
M2	M1	M0	Clock Source	Direction Source																																											
0	0	0	$F_{XTAL}/4$	UD bit (T1CONTROL.6)																																											
X	0	1	T1CLK pin [†]	UD bit (T1CONTROL.6)																																											
0	1	0	$F_{XTAL}/4$	T1DIR pin																																											
0	1	1	T1CLK pin [†]	T1DIR pin																																											
1	1	1	quadrature clocking using T1CLK and T1DIR																																												
2:0	P2:0	<p>EPA Clock Prescaler Bits</p> <p>These bits determine the clock prescaler value.</p> <table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">P2</th> <th style="text-align: center;">P1</th> <th style="text-align: center;">P0</th> <th style="text-align: left;">Prescaler Divisor</th> <th style="text-align: left;">Resolution[†]</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td>divide by 1 (disabled)</td> <td>250 ns</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td>divide by 2</td> <td>500 ns</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td>divide by 4</td> <td>1 μs</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td>divide by 8</td> <td>2 μs</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td>divide by 16</td> <td>4 μs</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td>divide by 32</td> <td>8 μs</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td>divide by 64</td> <td>16 μs</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td>enable T1RELOAD</td> <td>—</td> </tr> </tbody> </table> <p>[†] At 16 MHz. Use the formula on page 11-6 to calculate the resolution at other frequencies.</p>	P2	P1	P0	Prescaler Divisor	Resolution [†]	0	0	0	divide by 1 (disabled)	250 ns	0	0	1	divide by 2	500 ns	0	1	0	divide by 4	1 μ s	0	1	1	divide by 8	2 μ s	1	0	0	divide by 16	4 μ s	1	0	1	divide by 32	8 μ s	1	1	0	divide by 64	16 μ s	1	1	1	enable T1RELOAD	—
P2	P1	P0	Prescaler Divisor	Resolution [†]																																											
0	0	0	divide by 1 (disabled)	250 ns																																											
0	0	1	divide by 2	500 ns																																											
0	1	0	divide by 4	1 μ s																																											
0	1	1	divide by 8	2 μ s																																											
1	0	0	divide by 16	4 μ s																																											
1	0	1	divide by 32	8 μ s																																											
1	1	0	divide by 64	16 μ s																																											
1	1	1	enable T1RELOAD	—																																											

T1RELOAD**T1RELOAD**

Address: 1F72H
 Reset State: XXXXH

The timer 1 reload (T1RELOAD) register contains a reinitialization value for timer 1. The value of T1RELOAD is loaded into TIMER1 when timer 1 overflows or underflows and both quadrature clocking and the reload function are enabled (i.e., T1CONTROL.5:0 = 1).

15

0

Timer 1 Reload Value

Bit Number	Function
15:0	Timer 1 Reload Value Write the timer 1 reinitialization value to this register.

T2CONTROL

T2CONTROL

Address: 1F7CH
Reset State: 00H

The timer 2 control (T2CONTROL) register determines the clock source, counting direction, and count rate for timer 2.

7 0

CE	UD	M2	M1	M0	P2	P1	P0
----	----	----	----	----	----	----	----

Bit Number	Bit Mnemonic	Function																																													
7	CE	<p>Counter Enable</p> <p>This bit enables or disables the timer. From reset, the timers are disabled and not free running.</p> <p>0 = disables timer 1 = enables timer</p>																																													
6	UD	<p>Up/Down</p> <p>This bit determines the timer counting direction, in selected modes (see mode bits, M2:0).</p> <p>0 = count down 1 = count up</p>																																													
5:3	M2:0	<p>EPA Clock Direction Mode Bits</p> <p>These bits determine the timer clocking source and direction source.</p> <table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">M2</th> <th style="text-align: center;">M1</th> <th style="text-align: center;">M0</th> <th style="text-align: left;">Clock Source</th> <th style="text-align: left;">Direction Source</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td>F_{XTAL}/4</td> <td>UD bit (T2CONTROL.6)</td> </tr> <tr> <td style="text-align: center;">X</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td>reserved</td> <td>—</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td>reserved</td> <td>—</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td>reserved</td> <td>—</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td>timer 1 overflow</td> <td>UD bit (T2CONTROL.6)</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td>timer 1 overflow</td> <td>same as timer 1</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td>reserved</td> <td>—</td> </tr> </tbody> </table>	M2	M1	M0	Clock Source	Direction Source	0	0	0	F _{XTAL} /4	UD bit (T2CONTROL.6)	X	0	1	reserved	—	0	1	0	reserved	—	0	1	1	reserved	—	1	0	0	timer 1 overflow	UD bit (T2CONTROL.6)	1	1	0	timer 1 overflow	same as timer 1	1	1	1	reserved	—					
M2	M1	M0	Clock Source	Direction Source																																											
0	0	0	F _{XTAL} /4	UD bit (T2CONTROL.6)																																											
X	0	1	reserved	—																																											
0	1	0	reserved	—																																											
0	1	1	reserved	—																																											
1	0	0	timer 1 overflow	UD bit (T2CONTROL.6)																																											
1	1	0	timer 1 overflow	same as timer 1																																											
1	1	1	reserved	—																																											
2:0	P2:0	<p>EPA Clock Prescaler Bits</p> <p>These bits determine the clock prescaler value.</p> <table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">P2</th> <th style="text-align: center;">P1</th> <th style="text-align: center;">P0</th> <th style="text-align: left;">Prescaler</th> <th style="text-align: left;">Resolution[†]</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td>divide by 1 (disabled)</td> <td>250 ns</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td>divide by 2</td> <td>500 ns</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td>divide by 4</td> <td>1 μs</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td>divide by 8</td> <td>2 μs</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td>divide by 16</td> <td>4 μs</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td>divide by 32</td> <td>8 μs</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td>divide by 64</td> <td>16 μs</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td>reserved</td> <td>—</td> </tr> </tbody> </table> <p>[†] Resolution at 16 MHz. Use the formula on page 11-6 to calculate the resolution at other frequencies.</p>	P2	P1	P0	Prescaler	Resolution [†]	0	0	0	divide by 1 (disabled)	250 ns	0	0	1	divide by 2	500 ns	0	1	0	divide by 4	1 μs	0	1	1	divide by 8	2 μs	1	0	0	divide by 16	4 μs	1	0	1	divide by 32	8 μs	1	1	0	divide by 64	16 μs	1	1	1	reserved	—
P2	P1	P0	Prescaler	Resolution [†]																																											
0	0	0	divide by 1 (disabled)	250 ns																																											
0	0	1	divide by 2	500 ns																																											
0	1	0	divide by 4	1 μs																																											
0	1	1	divide by 8	2 μs																																											
1	0	0	divide by 16	4 μs																																											
1	0	1	divide by 32	8 μs																																											
1	1	0	divide by 64	16 μs																																											
1	1	1	reserved	—																																											

TIMERx**TIMERx**
x = 1–2Address: 1F7AH,
Reset State: 1F7EH
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.

USFR

USFR

Address: 1FF6H
 Reset State (MC, MD): 02H
 Reset State (MH): XXH

The unerasable PROM (USFR) register contains two bits that disable external fetches of data and instructions and another that detects a failed oscillator. These bits can be programmed, but cannot be erased.

WARNING: These bits can be programmed, but can never be erased. Programming these bits makes dynamic failure analysis impossible. For this reason, devices with programmed UPROM bits cannot be returned to Intel for failure analysis.

7

0



Bit Number	Bit Mnemonic	Function
7:4	—	Reserved; for compatibility with future devices, write zeros to these bits.
3	DEI	Disable External Instruction Fetch Setting this bit prevents the bus controller from executing external instruction fetches. Any attempt to load an external address initiates a reset.
2	DED	Disable External Data Fetch Setting this bit prevents the bus controller from executing external data reads and writes. Any attempt to access data through the bus controller initiates a reset.
1:0	—	Reserved; for compatibility with future devices, write zero to these bits.

WATCHDOG**WATCHDOG**

Address: 0AH
Reset State: XXH

Unless it is cleared every 64K state times, the watchdog timer resets the device. To clear the watchdog timer, send "1EH" followed immediately by "E1H" to location 0AH. Clearing this register the first time enables the watchdog with an initial value of 0000H, which is incremented once every state time. After it is enabled, the watchdog can be disabled only by a reset.

The WDE bit (bit 3) of CCR1 controls whether the watchdog is enabled immediately or is disabled until the first time it is cleared. Clearing WDE activates the watchdog. Setting WDE makes the watchdog timer inactive, but you can activate it by clearing the watchdog register. Once the watchdog is activated, only a reset can disable it.

7

0

Watchdog Timer Value

Bit Number	Function
7:0	Watchdog Timer Value This register contains the 8 most-significant bits of the current value of the watchdog timer.

WG_COMPx

WG_COMPx
x = 1–3

Address: 1FC2H,1FC4H,1FC6H
 Reset State: 0000H

The phase compare (WG_COMPx) register controls the duty cycle of each phase. Write a value to each phase compare register to specify the length of time that the associated outputs will remain asserted.

Changing the WG_RELOAD value changes both the carrier period and the duty cycle because the outputs remain asserted for a constant length of time, while the counter takes longer to cycle. To change the carrier period without changing the duty cycle, you must proportionally change both WG_RELOAD and WG_COMPx at the same time, immediately after the interrupt.

15

0

Compare

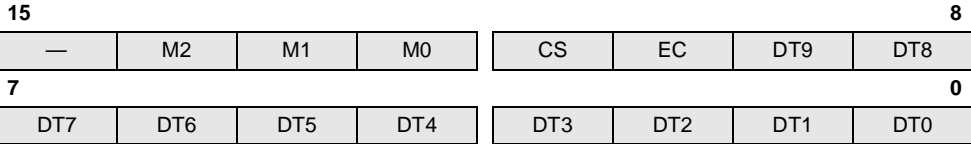
Bit Number	Function
15:0	<p>Compare</p> <p>These bits determine the length of time that the associated outputs are asserted. Use the following formulas to calculate output assertion time and duty cycle.</p> $T_{\text{OUTPUT}} = \frac{\text{multiplier} \times \text{WG_COMPx}}{F_{\text{XTAL1}}}$ $\text{Duty Cycle} = \frac{\text{WG_COMPx}}{\text{WG_RELOAD}} \times 100\%$ <p>where:</p> <ul style="list-style-type: none"> T_{OUTPUT} = total time output is asserted, in μs F_{XTAL1} = input frequency on XTAL1 pin, in MHz <i>multiplier</i> = 4 for center-aligned modes; 2 for edge-aligned modes WG_RELOAD = 16-bit WG_RELOAD value \geq WG_COMPx WG_COMPx = 16-bit WG_COMPx value \leq WG_RELOAD

WG_CONTROL

WG_CONTROL

Address: 1FCCH
 Reset State (MC, MD): 00C0H
 Reset State (MH): 8000H

The waveform generator control (WG_CONTROL) register controls the operating mode, dead time, and count direction, and enables and disables the counter.



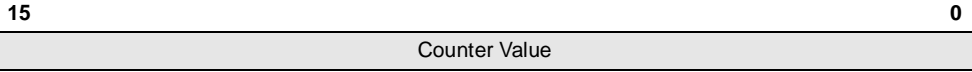
Bit Number	Bit Mnemonic	Function																														
15	—	Reserved; for compatibility with future devices, write zero to this bit.																														
14:12	M2:0	Operating Mode This field controls the waveform generator's operating mode: <table style="width: 100%; border-collapse: collapse;"> <tr> <th style="width: 5%;">M2</th> <th style="width: 5%;">M1</th> <th style="width: 5%;">M0</th> <th style="width: 5%;">Mode</th> <th style="width: 80%;"></th> </tr> <tr> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>center-aligned; update registers once</td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> <td>1</td> <td>center-aligned; update registers twice</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> <td>2</td> <td>edge-aligned; update registers once</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> <td>3</td> <td>edge-aligned; update registers twice</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>4</td> <td>(8XC196MH only) edge-aligned; update WG_COMPx and WG_COUNTER only when WG_COUNTER = WG_RELOAD</td> </tr> </table>	M2	M1	M0	Mode		0	0	0	0	center-aligned; update registers once	0	0	1	1	center-aligned; update registers twice	0	1	0	2	edge-aligned; update registers once	0	1	1	3	edge-aligned; update registers twice	1	1	1	4	(8XC196MH only) edge-aligned; update WG_COMPx and WG_COUNTER only when WG_COUNTER = WG_RELOAD
M2	M1	M0	Mode																													
0	0	0	0	center-aligned; update registers once																												
0	0	1	1	center-aligned; update registers twice																												
0	1	0	2	edge-aligned; update registers once																												
0	1	1	3	edge-aligned; update registers twice																												
1	1	1	4	(8XC196MH only) edge-aligned; update WG_COMPx and WG_COUNTER only when WG_COUNTER = WG_RELOAD																												
11	CS	Counter Status This read-only bit indicates whether the counter is counting up or counting down. 0 = down counting 1 = up counting																														
10	EC	Enable Counter This bit starts and stops the counter. 0 = disable (stop) counter 1 = enable (start) counter																														
9:0	DT9:0	Dead-time This field specifies the dead-time for all three phases. Use the following formula to calculate the appropriate DT_VALUE. $DT_VALUE = \frac{T_{DEAD} \times F_{XTAL1}}{2}$ where: T _{DEAD} = dead-time, in μs F _{XTAL1} = input frequency on XTAL1 pin, in MHz																														

WG_COUNTER

WG_COUNTER

Address: 1FCAH
 Reset State (MC, MD): XXXXH
 Reset State (MH): 0000H

You can read the waveform generator counter (WG_COUNTER) register to determine the current counter value.



Bit Number	Function
15:0	Counter Value This register reflects the current counter value.

WG_OUTPUT (Port 6)

WG_OUTPUT (Port 6)

Address: 1FC0H
Reset State: 0000H

The port 6 output configuration (WG_OUTPUT) register controls port 6 functions. If you are using port 6 for general-purpose outputs, write C0H (for active-high outputs) or 00H (for active-low outputs) to the high byte of WG_OUTPUT and write the desired pin values to the low byte.

15

8

OP1	OP0	—	M7	M6	M5:4	M3:2	M1:0
-----	-----	---	----	----	------	------	------

7

0

D7	D6	D5	D4	D3	D2	D1	D0
----	----	----	----	----	----	----	----

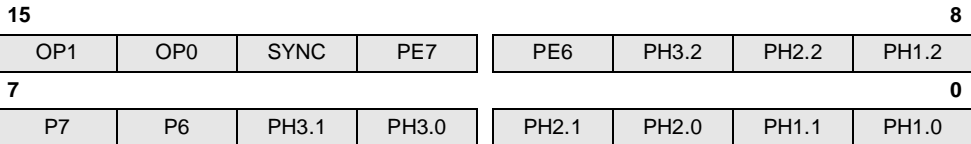
Bit Number	Bit Mnemonic	Function
15:14	OP1:0	Output Polarity These bits select the output polarity of the port 6 pins. 0 = active-low outputs 1 = active-high outputs
13	—	Reserved; for compatibility with future devices, write zero to this bit.
12:8	M7:0	Mode These bits select either the peripheral function or general-purpose output function of the port 6 pins. Clear these bits for general-purpose output.
7:0	D7:0	Data In general-purpose output mode, these bits hold the values to be driven out on the pins. Write the desired values to these bits (bits 7:0 correspond to pins P6.7:0).

WG_OUTPUT (Waveform Generator)

WG_OUTPUT (Waveform Generator)

Address: 1FC0H
Reset State: 0000H

The waveform generator output configuration (WG_OUTPUT) register controls the configuration of the waveform generator and PWM module pins. Both the waveform generator and the PWM module share pins with port 6. Having these control bits in a single register enables you to configure all port 6 pins with a single write to WG_OUTPUT.



Bit Number	Bit Mnemonic	Function
15	OP1	Output Polarity Selects the output polarity for negative-phase outputs WG1#, WG2#, and WG3#. 0 = active-low outputs 1 = active-high outputs
14	OP0	Output Polarity Selects the output polarity for positive-phase outputs WG1, WG2, and WG3. 0 = active-low outputs 1 = active-high outputs
13	SYNC	Synchronize Selects whether updating the WG_OUTPUT register is synchronized with another event or occurs immediately after you change it. 0 = update WG_OUTPUT immediately 1 = synchronize WG_OUTPUT update with an event To ensure that the outputs are in the desired states when the waveform generator starts, you should initially clear this bit, then set it later if you want subsequent WG_OUTPUT updates to be synchronized with an event. (Table 9-4 on page C-8 lists the events that update WG_OUTPUT in each mode.)
12	PE7	P6.7/PWM1 Function Selects the port function or the PWM output function of P6.7/PWM1. 0 = P6.7 1 = PWM1
11	PE6	P6.6/PWM0 Function Selects the port function or the PWM output function of P6.6/PWM0. 0 = P6.6 1 = PWM0

WG_OUTPUT (Waveform Generator)**WG_OUTPUT (Waveform Generator) (Continued)**Address: 1FC0H
Reset State: 0000H

The waveform generator output configuration (WG_OUTPUT) register controls the configuration of the waveform generator and PWM module pins. Both the waveform generator and the PWM module share pins with port 6. Having these control bits in a single register enables you to configure all port 6 pins with a single write to WG_OUTPUT.

15

8

OP1	OP0	SYNC	PE7	PE6	PH3.2	PH2.2	PH1.2
-----	-----	------	-----	-----	-------	-------	-------

7

0

P7	P6	PH3.1	PH3.0	PH2.1	PH2.0	PH1.1	PH1.0
----	----	-------	-------	-------	-------	-------	-------

Bit Number	Bit Mnemonic	Function
10	PH3.2	Phase 3 Function Selects either the port function or the waveform generator output function for pins P6.4/WG3# and P6.5/WG3. 0 = P6.4, P6.5 1 = WG3#, WG3
9	PH2.2	Phase 2 Function Selects either the port function or the waveform generator output function for pins P6.2/WG2# and P6.3/WG2. 0 = P6.2, P6.3 1 = WG2#, WG2
8	PH1.2	Phase 1 Function Selects either the port function or the waveform generator output function for pins P6.0/WG1# and P6.1/WG1. 0 = P6.0, P6.1 1 = WG1#, WG1
7	P7	P6.7/PWM1 Value Write the desired P6.7/PWM1 value to this bit.
6	P6	P6.6/PWM0 Value Write the desired P6.6/PWM0 value to this bit.
5:4	PH3.1:0	P6.4/WG3#, P6.5/WG3 Value Write the desired output values to these bits. See Table C-11 on page C-65.
3:2	PH2.1:0	P6.2/WG2#, P6.3/WG2 Values Write the desired output values to these bits. See Table C-11 on page C-65.
1:0	PH1.1:0	P6.0/WG1#, P6.1/WG1 Values Write the desired output values to these bits. See Table C-11 on page C-65.

WG_OUTPUT (Waveform Generator)

Table C-11. Output Configuration

PHx.2	PHx.1	PHx.0	Output Values		Output Polarities	
			WGx	WGx#	WGx	WGx#
1	0	0	Low	Low	Always Low	Always Low
1	0	1	Low	WG_EVEN#	Always Low	
1	1	0	WG_ODD	Low		Always Low
1	1	1	WG_ODD	WG_EVEN		

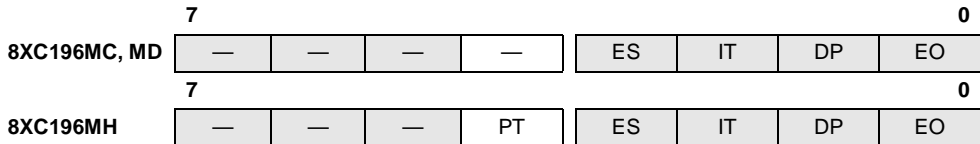
NOTE: This table assumes active-high outputs (OP1=OP0=1).

WG_PROTECT

WG_PROTECT

Address: 1FCEH
 Reset State (MC, MD) F0H
 Reset State (MH): E0H

The waveform protection (WG_PROTECT) register enables and disables the outputs and the protection circuitry. It also selects either level-sensitive or edge-triggered EXTINT interrupts, and selects which level or edge will generate an EXTINT interrupt request.



Bit Number	Bit Mnemonic	Function															
7:5	—	Reserved; for compatibility with future devices, write zeros to these bits.															
6†	PT	Protection Type This bit selects the method used for disabling the outputs. 0 = inactive states 1 = weak pull-ups															
3:2	ES IT	Enable Sampling and Interrupt Type The ES bit selects whether the protection circuitry samples the EXTINT signal level or detects a signal transition (edge), while the IT bit controls which value of the edge or level triggers an interrupt request. The possible combinations are as follows. <table border="1" style="margin-left: 20px; border-collapse: collapse;"> <thead> <tr> <th>ES</th> <th>IT</th> <th>Event</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td>falling edge</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td>rising edge</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td>low level</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td>high level</td> </tr> </tbody> </table>	ES	IT	Event	0	0	falling edge	0	1	rising edge	1	0	low level	1	1	high level
ES	IT	Event															
0	0	falling edge															
0	1	rising edge															
1	0	low level															
1	1	high level															
1	DP	Disable Protection This bit enables and disables the protection circuitry. 0 = enable protection 1 = disable protection															
0	EO	Enable Outputs This bit enables and disables the outputs. 0 = disable outputs 1 = enable outputs															

† On the 8XC196MC, MD devices, this bit is reserved. For compatibility with future devices, always write as zero.

WG_RELOAD
WG_RELOAD

 Address: 1FC8H
 Reset State: 0000H

The waveform generator reload (WG_RELOAD) register and the phase compare registers (WG_COMPx) control the carrier period and duty cycle. Write a value to the reload register to establish the carrier period.

Changing the WG_RELOAD value changes both the carrier period and the duty cycle because the outputs remain asserted for a constant length of time, while the counter takes longer to cycle. To change the carrier period without changing the duty cycle, you must proportionally change both WG_RELOAD and WG_COMPx at the same time, immediately after the interrupt.

15
0

Reload

Bit Number	Function
15:0	Reload This register determines the carrier period. Use the following formulas to calculate carrier period and duty cycle. $T_{\text{CARRIER}} = \frac{\text{multiplier} \times \text{WG_RELOAD}}{F_{\text{XTAL1}}}$ $\text{Duty Cycle} = \frac{\text{WG_COMPx}}{\text{WG_RELOAD}} \times 100\%$ where: T_{CARRIER} = carrier period, in μs F_{XTAL1} = input frequency on XTAL1 pin, in MHz <i>multiplier</i> = 4 for center-aligned modes; 2 for edge-aligned modes WG_RELOAD = 16-bit WG_RELOAD value \geq WG_COMPx WG_COMPx = 16-bit WG_COMPx value \leq WG_RELOAD

WSR

WSR	Address: 0014H						
	Reset State: 00H						
The window selection register (WSR) maps 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.							
7	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. Table C-12 shows the WSR settings and direct addresses for windowable SFRs.					

Table C-12. WSR Settings and Direct Addresses for Windowable SFRs

Register Mnemonic	Memory Location	32-byte Windows (00E0–00FFH)		64-byte Windows (00C0–00FFH)		128-byte Windows (0080–00FFH)	
		WSR	Direct Address	WSR	Direct Address	WSR	Direct Address
AD_COMMAND	1FACH	7DH	00ECH	3EH	00ECH	1FH	00ACH
AD_RESULT	1FAAH	7DH	00EAH	3EH	00EAH	1FH	00AAH
AD_TEST	1FAEH	7DH	00EEH	3EH	00EEH	1FH	00AEH
AD_TIME	1FAFH	7DH	00EFH	3EH	00EFH	1FH	00AFH
COMP0_CON	1F58H	7AH	00F8H	3DH	00D8H	1EH	00D8H
COMP1_CON	1F5CH	7AH	00FCH	3DH	00DCH	1EH	00DCH
COMP2_CON	1F60H	7BH	00E0H	3DH	00E0H	1EH	00E0H
COMP3_CON	1F64H	7BH	00E4H	3DH	00E4H	1EH	00E4H
COMP4_CON (MD)	1F68H	7BH	00E8H	3DH	00E8H	1EH	00E8H
COMP5_CON (MD)	1F6CH	7BH	00ECH	3DH	00ECH	1EH	00ECH
COMP0_TIME [†]	1F5AH	7AH	00FAH	3DH	00DAH	1EH	00DAH
COMP1_TIME [†]	1F5EH	7AH	00FEH	3DH	00DEH	1EH	00DEH
COMP2_TIME [†]	1F62H	7BH	00E2H	3DH	00E2H	1EH	00E2H
COMP3_TIME [†]	1F66H	7BH	00E6H	3DH	00E6H	1EH	00E6H
COMP4_TIME [†] (MD)	1F6AH	7BH	00EAH	3DH	00EAH	1EH	00EAH
COMP5_TIME [†] (MD)	1F6EH	7BH	00EEH	3DH	00EEH	1EH	00EEH
EPA0_CON	1F40H	7AH	00E0H	3DH	00C0H	1EH	00C0H
EPA1_CON [†]	1F44H	7AH	00E4H	3DH	00C4H	1EH	00C4H

[†] Must be addressed as a word.

WSR
Table C-12. WSR Settings and Direct Addresses for Windowable SFRs (Continued)

Register Mnemonic	Memory Location	32-byte Windows (00E0–00FFH)		64-byte Windows (00C0–00FFH)		128-byte Windows (0080–00FFH)	
		WSR	Direct Address	WSR	Direct Address	WSR	Direct Address
EPA2_CON (MC, MD)	1F48H	7AH	00E8H	3DH	00C8H	1EH	00C8H
EPA3_CON [†] (MC, MD)	1F4CH	7AH	00ECH	3DH	00CCH	1EH	00CCH
EPA4_CON (MD)	1F50H	7AH	00F0H	3DH	00D0H	1EH	00D0H
EPA5_CON (MD)	1F54H	7AH	00F4H	3DH	00D4H	1EH	00D4H
EPA0_TIME [†]	1F42H	7AH	00E2H	3DH	00C2H	1EH	00C2H
EPA1_TIME [†]	1F46H	7AH	00E6H	3DH	00C6H	1EH	00C6H
EPA2_TIME [†] (MC, MD)	1F4AH	7AH	00EAH	3DH	00CAH	1EH	00CAH
EPA3_TIME [†] (MC, MD)	1F4EH	7AH	00EEH	3DH	00CEH	1EH	00CEH
EPA4_TIME [†] (MD)	1F52H	7AH	00F2H	3DH	00D2H	1EH	00D2H
EPA5_TIME [†] (MD)	1F56H	7AH	00F6H	3DH	00D6H	1EH	00D6H
FREQ_CNT (MD)	1FBAH	7DH	00FAH	3EH	00FAH	1FH	00BAH
FREQ_GEN (MD)	1FB8H	7DH	00FBH	3EH	00F8H	1FH	00B8H
GEN_CON (MH)	1FA0H	7DH	00E0H	3EH	00E0H	1FH	00A0H
P1_DIR (MH)	1F9AH	7CH	00FAH	3EH	00DAH	1FH	009AH
P2_DIR	1FD2H	7EH	00F2H	3FH	00D2H	1FH	00D2H
P7_DIR (MD)	1FD3H	7EH	00F3H	3FH	00D3H	1FH	00D3H
P1_MODE (MH)	1F98H	7CH	00F8H	3EH	00D8H	1FH	0098H
P2_MODE	1FD0H	7EH	00F0H	3FH	00D0H	1FH	00D0H
P7_MODE (MD)	1FD1H	7EH	00F1H	3FH	00D1H	1FH	00D1H
P0_PIN (MC, MD)	1FA8H	7DH	00E8H	3EH	00E8H	1FH	00A8H
P0_PIN (MH)	1FDAH	7EH	00FAH	3FH	00DAH	1FH	00DAH
P1_PIN (MC)	1FA8H	7DH	00E8H	3EH	00E8H	1FH	00A8H
P1_PIN (MH)	1F9EH	7CH	00FEH	3EH	00DEH	1FH	009EH
P2_PIN	1FD6H	7EH	00F6H	3FH	00D6H	1FH	00D6H
P7_PIN (MD)	1FD7H	7EH	00F7H	3FH	00D7H	1FH	00D7H
P1_REG (MH)	1F9CH	7CH	00FCH	3EH	00DCH	1FH	009CH
P2_REG	1FD4H	7EH	00F4H	3FH	00D4H	1FH	00D4H
P7_REG (MD)	1FD5H	7EH	00F5H	3FH	00D5H	1FH	00D5H
PL_MASK	1FBCH	7DH	00FCH	3EH	00FCH	1FH	00BCH
PI_PEND	1FBEH	7DH	00FEH	3EH	00FEH	1FH	00BEH
PWM_COUNT	1FB6H	7DH	00F6H	3EH	00F6H	1FH	00B6H
PWM_PERIOD	1FB4H	7DH	00F4H	3EH	00F4H	1FH	00B4H

[†] Must be addressed as a word.

WSR

Table C-12. WSR Settings and Direct Addresses for Windowable SFRs (Continued)

Register Mnemonic	Memory Location	32-byte Windows (00E0–00FFH)		64-byte Windows (00C0–00FFH)		128-byte Windows (0080–00FFH)	
		WSR	Direct Address	WSR	Direct Address	WSR	Direct Address
PWM0_CONTROL	1FB0H	7DH	00F0H	3EH	00F0H	1FH	00B0H
PWM1_CONTROL	1FB2H	7DH	00F2H	3EH	00F2H	1FH	00B2H
SBUF0_RX (MH)	1F80H	7CH	00E0H	3EH	00C0H	1FH	0080H
SBUF1_RX (MH)	1F88H	7CH	00E8H	3EH	00C8H	1FH	0088H
SBUF0_TX (MH)	1F82H	7CH	00E2H	3EH	00C2H	1FH	0082H
SBUF1_TX (MH)	1F8AH	7CH	00EAH	3EH	00CAH	1FH	008AH
SP0_BAUD (MH)	1F84H	7CH	00E4H	3EH	00C4H	1FH	0084H
SP1_BAUD (MH)	1F8CH	7CH	00ECH	3EH	00CCH	1FH	008CH
SP0_CON (MH)	1F83H	7CH	00E3H	3EH	00C3H	1FH	0083H
SP1_CON (MH)	1F8BH	7CH	00EBH	3EH	00CBH	1FH	008BH
SP0_STATUS (MH)	1F81H	7CH	00E1H	3EH	00C1H	1FH	0081H
SP1_STATUS (MH)	1F89H	7CH	00E9H	3EH	00C9H	1FH	0089H
T1CONTROL	1F78H	7BH	00F8H	3DH	00F8H	1EH	00F8H
T1RELOAD	1F72H	7BH	00F2H	3DH	00F2H	1EH	00F2H
T2CONTROL	1F7CH	7BH	00FCH	3DH	00FCH	1EH	00FCH
TIMER1 [†]	1F7AH	7BH	00FAH	3DH	00FAH	1EH	00FAH
TIMER2 [†]	1F7EH	7BH	00FEH	3DH	00FEH	1EH	00FEH
WG_COMP1	1FC2H	7EH	00E2H	3FH	00C2H	1FH	00C2H
WG_COMP2	1FC4H	7EH	00E4H	3FH	00C4H	1FH	00C4H
WG_COMP3	1FC6H	7EH	00E6H	3FH	00C6H	1FH	00C6H
WG_CONTROL	1FCCH	7EH	00ECH	3FH	00CCH	1FH	00CCH
WG_COUNTER	1FCAH	7EH	00EAH	3FH	00CAH	1FH	00CAH
WG_OUTPUT	1FC0H	7EH	00E0H	3FH	00C0H	1FH	00C0H
WG_PROTECT	1FCEH	7EH	00EEH	3FH	00CEH	1FH	00CEH
WG_RELOAD	1FC8H	7EH	00E8H	3FH	00C8H	1FH	00C8H

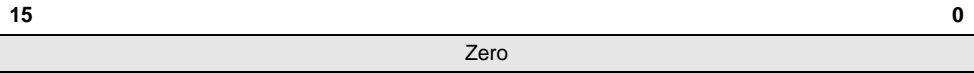
[†] Must be addressed as a word.

ZERO_REG

ZERO_REG

Address: 00H
 Reset State: 0000H

The two-byte zero register (ZERO_REG) is always equal to zero. It is useful as a fixed source of the constant zero for comparisons and calculations.



Bit Number	Function
15:0	Zero This register is always equal to zero.



Glossary

This glossary defines acronyms, abbreviations, and terms that have special meaning in this manual. (Chapter 1 discusses notational conventions and general terminology.)

absolute error	The maximum difference between corresponding actual and ideal <i>code transitions</i> . 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 <i>A/D converter</i> . An actual characteristic may vary with temperature, supply voltage, and frequency conditions.
A/D converter	Analog-to-digital converter.
ALU	Arithmetic-logic unit. The part of the <i>RALU</i> 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 <i>A/D converter</i> 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 (<i>CCRs</i>) are loaded with the contents of the <i>CCBs</i> after a device reset, unless the device is entering programming modes, in which case the <i>PCCBs</i> are used.
CCRs	Chip configuration registers. Registers that specify the environment in which the device will be operating. The chip configuration registers are loaded with the contents of the <i>CCBs</i> after a device reset unless the device is entering programming modes, in which case the <i>PCCBs</i> are used.
channel-to-channel matching error	The difference between corresponding <i>code transitions</i> of actual characteristics taken from different <i>A/D converter</i> channels under the same temperature, voltage, and frequency conditions. This error is caused by differences in <i>DC input leakage</i> and on-channel resistance from one multiplexer channel to another.
characteristic	A graph of output code versus input voltage; the <i>transfer function</i> of an <i>A/D converter</i> .
clear	The “0” value of a bit or the act of giving it a “0” value. See also <i>set</i> .
code	<ol style="list-style-type: none"> 1) A set of instructions that perform a specific function; a program. 2) The digital value output by the <i>A/D converter</i>.
code center	The voltage corresponding to the midpoint between two adjacent <i>code transitions</i> on the <i>A/D converter</i> .
code transition	The point at which the <i>A/D converter</i> '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 <i>code transitions</i> . Code width deviations cause <i>differential nonlinearity</i> and <i>nonlinearity</i> errors.
crosstalk	See <i>off-isolation</i> .
DC input leakage	Leakage current from an analog input pin to ground.

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.
differential nonlinearity	The difference between the actual <i>code width</i> and the ideal one-LSB code width of the <i>terminal-based characteristic</i> 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. <i>Differential nonlinearity</i> is a measure of local code-width error; <i>nonlinearity</i> 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 <i>p-type</i> material. A Group V impurity (e.g., arsenic or antimony) results in an <i>n-type</i> 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.
EPROM	Erasable, programmable read-only-memory.
ESD	Electrostatic discharge.
feedthrough	The <i>attenuation</i> from an input voltage on the selected channel to the A/D output after the <i>sample window</i> closes. The ability of the <i>A/D converter</i> to reject an input on its selected channel after the sample window closes.
FET	Field-effect transistor.
frequency generator	The 8XC196MD peripheral that generates outputs with a fixed 50% duty cycle and a programmable frequency. The frequency generator can be used for infrared transmission.

full-scale error	The difference between the ideal and actual input voltage corresponding to the final (full-scale) <i>code transition</i> of an <i>A/D converter</i> .
hold latency	The time it takes the microcontroller to assert HLDA# after an external device asserts HOLD#.
ideal characteristic	The <i>characteristic</i> of an ideal <i>A/D converter</i> . An ideal characteristic is unique: its first <i>code transition</i> 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 <i>zero-offset</i> , <i>full-scale</i> , or <i>linearity</i> errors. <i>Quantizing error</i> is the only error seen in an ideal <i>A/D converter</i> .
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 <i>sample capacitor</i> of an <i>A/D converter</i> .
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 $+2^{15}-1$.
interrupt controller	The module responsible for handling interrupts that are to be serviced by <i>interrupt service routines</i> that you provide. Also called the <i>programmable interrupt controller (PIC)</i> .
interrupt latency	The total delay between the time that an interrupt is generated (not acknowledged) and the time that the device begins executing the <i>interrupt service routine</i> or <i>PTS routine</i> .
interrupt service routine	A software routine that you provide to service a standard interrupt. See also <i>PTS routine</i> .
interrupt vector	A location in <i>special-purpose memory</i> that holds the starting address of an <i>interrupt service routine</i> .
ISR	See <i>interrupt service routine</i> .
linearity errors	See <i>differential nonlinearity</i> and <i>nonlinearity</i> .
LONG-INTEGERS	A 32-bit, signed variable with values from -2^{31} through $+2^{31}-1$.

LSB	<p>1) Least-significant bit of a byte or least-significant byte of a word.</p> <p>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 ($5.12 \div 2^{10}$).</p>
maskable interrupts	All interrupts except 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 global interrupt enable bit. Each <i>maskable interrupt</i> can be assigned to the <i>PTS</i> for processing.
monotonic	The property of <i>successive approximation</i> converters which guarantees that increasing input voltages produce adjacent <i>codes</i> 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 <i>differential nonlinearity</i> errors can cause the converter to exhibit nonmonotonic behavior.
MSB	Most-significant bit of a <i>byte</i> or most-significant byte of a <i>word</i> .
<i>n</i>-channel FET	A field-effect transistor with an <i>n</i> -type conducting path (channel).
<i>n</i>-type material	Semiconductor material with introduced impurities (<i>doping</i>) causing it to have an excess of negatively charged carriers.
no missing codes	An A/D converter has <i>no missing codes</i> if, for every output code, there is a unique input voltage range which produces that code only. Large <i>differential nonlinearity</i> errors can cause the converter to miss codes.
nonlinearity	The maximum deviation of <i>code transitions</i> of the <i>terminal-based characteristic</i> from the corresponding code transitions of the <i>ideal characteristic</i> .

nonmaskable interrupts	Interrupts that cannot be masked (disabled) and cannot be assigned to the PTS for processing. The nonmaskable interrupts are unimplemented opcode, software trap, and NMI.
nonvolatile memory	Read-only memory that retains its contents when power is removed. Many MCS [®] 96 microcontrollers are available with either masked ROM, <i>EPROM</i> , or <i>OTPROM</i> . Consult the <i>Automotive Products</i> or <i>Embedded Microcontrollers</i> databook to determine which type of memory is available for a specific device.
npn transistor	A transistor consisting of one part <i>p</i> -type material and two parts <i>n</i> -type material.
off-isolation	The ability of an <i>A/D converter</i> to reject (isolate) the signal on a deselected (off) output.
OTPROM	One-time-programmable read-only memory. Similar to <i>EPROM</i> , but it comes in an unwindowed package and cannot be erased.
p-channel FET	A field-effect transistor with a <i>p</i> -type conducting path.
p-type material	Semiconductor material with introduced impurities (<i>doping</i>) causing it to have an excess of positively charged carriers.
PC	Program counter.
PCCBs	Programming chip configuration bytes, which are loaded into the chip configuration registers (<i>CCRs</i>) when the device is entering programming modes; otherwise, the <i>CCBs</i> are used.
PIC	Programmable interrupt controller. The module responsible for handling interrupts that are to be serviced by <i>interrupt service routines</i> that you provide. Also called simply the <i>interrupt controller</i> .
prioritized interrupt	Any <i>maskable interrupt</i> or nonmaskable NMI. Two of the <i>nonmaskable interrupts</i> (unimplemented opcode and software trap) are not prioritized; they vector directly to the <i>interrupt service routine</i> 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 <i>PTS</i> , and six Boolean flags that reflect the state of the current program. The low byte of the PSW is the INT_MASK register. A push or pop instruction saves or restores both bytes (PSW + INT_MASK).
PTS	Peripheral transaction server. The microcoded hardware interrupt processor.
PTSCB	See <i>PTS control block</i> .
PTS control block	A block of data required for each <i>PTS interrupt</i> . The microcode executes the proper <i>PTS routine</i> based on the contents of the PTS control block.
PTS cycle	The microcoded response to a single PTS interrupt request.
PTS interrupt	Any <i>maskable interrupt</i> that is assigned to the <i>PTS</i> for interrupt processing.
PTS mode	A microcoded response that enables the <i>PTS</i> to complete a specific task quickly. These tasks include transferring a single byte or word, transferring a block of bytes or words, managing multiple A/D conversions, and generating <i>PWM</i> outputs.
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 <i>special-purpose memory</i> that holds the starting address of a <i>PTS control block</i> .

PWM	Pulse-width modulated (outputs). The 8XC196M _x devices have several options for producing PWM outputs: the generic pulse-width modulator modules, the <i>waveform generator</i> , and the <i>EPA</i> with or without the <i>PTS</i> . The 8XC196MD also has a <i>frequency generator</i> that produces PWM outputs.
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 <i>ALU</i> , the <i>PSW</i> , the master <i>PC</i> , the microcode engine, a loop counter, and six registers.
repeatability error	The difference between corresponding <i>code transitions</i> from different <i>actual characteristics</i> taken 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 with FFH.
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 <i>sample capacitor</i> is connected to the selected channel.
sample delay uncertainty	The variation in the <i>sample delay</i> .

sample time	The period of time that the <i>sample window</i> is open. (That is, the length of time that the input channel is actually connected to the <i>sample capacitor</i> .)
sample time uncertainty	The variation in the <i>sample time</i> .
sample window	The period of time that begins when the <i>sample capacitor</i> is attached to a selected channel of an <i>A/D converter</i> 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	<i>Successive approximation</i> register. A component of the <i>A/D converter</i> .
set	The “1” value of a bit or the act of giving it a “1” value. See also <i>clear</i> .
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.
source current	Current flowing out of a device from V_{CC} . Always a negative value.
SP	Stack pointer.

special interrupt	Any of the three <i>nonmaskable interrupts</i> (unimplemented opcode, software trap, or NMI).
special-purpose memory	A partition of memory used for storing the <i>interrupt vectors</i> , <i>PTS vectors</i> , chip configuration bytes, and several reserved locations.
standard interrupt	Any <i>maskable interrupt</i> that is assigned to the <i>interrupt controller</i> for processing by an <i>interrupt service routine</i> .
state time (or state)	The basic time unit of the device; the combined period of the two internal timing signals, PH1 and PH2. (The internal clock generator produces PH1 and PH2 by halving the frequency of the signal on XTAL1. The rising edges of the active-high PH1 and PH2 signals generate CLKOUT, the output of the internal clock generator.) Because the device can operate at many frequencies, this manual defines time requirements in terms of <i>state times</i> 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.
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 <i>temperature coefficient</i> for the specification.
terminal-based characteristic	An <i>actual characteristic</i> that has been translated and scaled to remove <i>zero-offset error</i> and <i>full-scale error</i> . A terminal-based characteristic resembles an <i>actual characteristic</i> with zero-offset error and full-scale error removed.
transfer function	A graph of output <i>code</i> versus input voltage; the <i>characteristic</i> of the A/D converter.

transfer function errors	Errors inherent in an analog-to-digital conversion process: <i>quantizing error</i> , <i>zero-offset error</i> , <i>full-scale error</i> , <i>differential nonlinearity</i> , and <i>nonlinearity</i> . Errors that are hardware-dependent, rather than being inherent in the process itself, include <i>feedthrough</i> , <i>repeatability</i> , <i>channel-to-channel matching</i> , <i>off-isolation</i> , and <i>V_{CC} rejection errors</i> .
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 <i>actual characteristic</i> is unaffected by those changes. The effectiveness of <i>V_{CC} rejection</i> is measured by the ratio of the change in V _{CC} to the change in the <i>actual characteristic</i> .
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 device if software fails to respond before the timer overflows.
waveform generator	One of the 8XC196Mx peripherals that can be used to produce pulse-width modulated (PWM) outputs. The waveform generator is optimized for controlling 3-phase AC induction motors, brushless DC motors, and other devices requiring multiple PWM outputs.
WDT	See <i>watchdog timer</i> .
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 <i>A/D converter's</i> first <i>code transition</i> 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 <i>A/D converter's</i> first code transition.



Index

#, defined, 1-3, A-1

16-bit data bus

read cycles, 15-14

timing diagram, 15-15

write cycles, 15-14

8-bit data bus

read cycles, 15-16

timing diagram, 15-17

write cycles, 15-16

A

A/D command register, 12-8, C-6

A/D converter, 2-11, 12-1–12-18

actual characteristic, 12-16

and port 0 reads, 12-13

and PTS, 5-32–5-37

block diagram, 12-1

calculating result, 12-9, 12-13

calculating series resistance, 12-10

characteristics, 12-15–12-18

conversion time, 12-6

determining status, 12-9

errors, 12-13–12-18

hardware considerations, 12-10–12-13

ideal characteristic, 12-15, 12-16

input circuit, suggested, 12-12

input protection devices, 12-12

interfacing with, 12-10–12-13

interpreting results, 12-9

interrupt, 12-8

minimizing input source resistance, 12-11

overview, 12-3–12-4

programming, 12-4–12-8

sample delay, 12-4

sample time, 12-6

sample window, 12-4

SFRs, 12-2

signals, 12-2

starting with PTS, 5-32–5-37

successive approximation

algorithm, 12-4

register (SAR), 12-4

terminal-based characteristic, 12-18

threshold-detection modes, 12-5

transfer function, 12-15–12-18

zero-offset adjustment, 12-3, 12-5

zero-offset error, 12-16

See also port 0

A/D result register (read), 12-9, C-7

A/D result register (write), 12-6, C-8

A/D scan mode, *See* PTS

A/D test register, 12-5, C-9

A/D time register, 12-7, C-10

AC timing

specifications, 15-31–15-34

symbol explanations, 15-33

Accumulator, RALU, 2-5

ACH13:0, B-13

AD_COMMAND, C-68

AD_RESULT, 12-9, C-68

AD_TEST, C-68

AD_TIME, C-68

AD15:0, B-13

ADD instruction, A-2, A-7, A-41, A-42, A-47, A-52

ADDB instruction, A-2, A-7, A-42, A-43, A-47, A-52

ADDC instruction, A-2, A-7, A-44, A-47, A-52

ADDCB instruction, A-2, A-8, A-44, A-47, A-52

Address space

map, 4-2

See also memory partitions

Address valid strobe mode

ALE/ADV# comparison, 15-27

example system, 15-28, 15-29

signals, 15-27

Address valid with write strobe mode

example system, 15-31

signals, 15-30

Address/data bus, 2-6

multiplexing, 15-10–15-17

Addressing modes, 3-5–3-6, A-6

ADV#, B-13

AINC#, 16-12, B-14

ALE, B-14

idle, powerdown, reset status, B-23, B-25

Analog outputs, generating, 10-10

Analog-to-digital converter, *See* A/D converter

AND instruction, A-2, A-8, A-41, A-42, A-48, A-53
 ANDB instruction, A-2, A-8, A-9, A-42, A-43, A-48, A-53
 ANGND, 12-5, 13-1, B-14
*Ap*BUILDER software, downloading, 1-10
 Application notes, ordering, 1-6
 Arithmetic instructions, A-47, A-48, A-52, A-53
 Assert, defined, 1-3
 Auto programming mode, 16-25–16-28
 algorithm, 16-28
 circuit, 16-25–16-26
 memory map, 16-27
 PCCB, 16-27
 security key programming, 16-29

B

Baud rate
 SIO port, 7-12–7-14
 Baud-rate generator
 SIO port, 7-12
 BAUD_VALUE, 7-13
 BCLK1:0, B-14
 BCLKx, 7-2
 BHE#, B-14
 BIT, defined, 3-2
 Bit-test instructions, A-17
 Block diagram
 A/D converter, 12-1
 address/data bus, 6-15
 clock circuitry, 2-7
 core and peripherals, 2-3
 EPA, 11-2
 frequency generator, 8-1
 infrared remote control application, 8-5
 I/O ports, 6-3, 6-8, 6-15, 6-16
 SIO port, 7-1
 waveform generator, 9-2
 Block transfer mode, *See* PTS
 BMOV instruction, A-2, A-9, A-45, A-49
 BMOVI instruction, A-3, A-9, A-10, A-45, A-49
 BR (indirect) instruction, A-2, A-10, A-49, A-55
 Bus controller, 2-6, 16-6
 Bus-control modes, 15-21–15-31
 address valid strobe, 15-27–15-29
 address valid with write strobe, 15-30–15-31

 standard, 15-22–15-24
 write strobe, 15-25–15-26
 Bus-control signals, 15-21
 Bus-width modes
 16-bit data bus, 15-14
 8-bit data bus, 15-16
 dynamic data bus, 15-13
 dynamic example, 15-24
 BUSWIDTH, 16-25, 16-27, B-15
 idle, powerdown, reset status, B-23, B-25
 timing, 15-11
 definitions, 15-13
 diagram, 15-12
 requirements, 15-13
 BYTE, defined, 3-2

C

Call instructions, A-50, A-55, A-56
 Carry (C) flag, 3-4, A-4, A-5, A-11, A-18, A-19, A-20, A-21, A-31
 Cascading timers, 11-7
 CCB fetch
 and BHE#, 6-13
 and P5.5, 6-13
 and P5.6, 6-13
 and READY, 6-13
 CCBs, 4-3, 13-8
 security-lock bits, 16-29–16-30
 CCBs, *See also* chip configuration bytes
 CCR0, 14-2
 CCRs, 13-8, 14-5
 security-lock bits, 16-17
 CCRs, *See also* chip configuration registers
 Chip configuration
 and bus contention, 15-11
 and reset, 15-6
 bytes, 15-5
 chip configuration register 0, 15-7, C-11
 chip configuration register 1, 15-9, C-13
 registers, 15-5
 Clear, defined, 1-3
 CLKOUT, 14-1, B-15
 and internal timing, 2-7
 and interrupts, 5-6
 and RESET#, 13-8
 idle, powerdown, reset status, B-24
 reset status, 6-7

Clock

- external, 13-7
- generator, 2-7, 13-7, 13-8
- internal, and idle mode, 14-4, 14-5
- phases, internal, 2-8

- CLR instruction, A-2, A-10, A-41, A-47, A-52
- CLRB instruction, A-2, A-11, A-41, A-47, A-52
- CLRC instruction, A-3, A-11, A-46, A-51, A-57
- CLRVT instruction, A-3, A-11, A-46, A-51, A-57
- CMP instruction, A-3, A-11, A-43, A-47, A-52
- COMPB instruction, A-3, A-11, A-44, A-47, A-52
- CMPL instruction, A-2, A-12, A-45, A-47, A-52
- Code execution, 2-5, 2-6
- COMP0_TIME, C-68
- COMP1_CON, C-68
- COMP1_TIME, C-68
- COMP5:0, 11-3, B-15
- CompuServe forums, 1-10
- Conditional jump instructions, A-5
- Configuring external memory pins, 15-5
- CPU, 2-4
- CPVER, 16-12, B-15
- Customer service, 1-8

D

- D/A converter, 10-10
- Data instructions, A-49, A-55
- Data types, 3-1–3-4
 - addressing restrictions, 3-1
 - converting between, 3-4
 - defined, 3-1
 - iC-96, 3-1
 - PLM-96, 3-1
 - signed and unsigned, 3-1, 3-4
 - values permitted, 3-1
- Datasheets
 - online, 1-10
 - ordering, 1-7
- Deassert, defined, 1-3
- DEC instruction, A-2, A-12, A-41, A-47, A-52
- DECB instruction, A-2, A-12, A-41, A-47, A-52
- DED bit, 16-6–16-7, 16-30
- DEI bit, 16-6–16-7, 16-17
- Design considerations
 - waveform generator, 9-19, 9-20
- Device
 - minimum hardware configuration, 13-1

- pin reset status, B-23, B-25
- programming, 16-1–16-33
- reset, 13-8, 13-9, 13-10, 13-11, 13-12

DI instruction, A-3, A-12, A-46, A-51, A-57

Digital-to-analog converter, 10-10

DIR bit, 7-2, 7-6

Direct addressing, 3-6, 3-9

DIV instruction, A-13, A-46, A-48, A-53

DIVB instruction, A-13, A-46, A-48, A-53

DIVU instruction, A-3, A-13, A-43, A-48, A-53

DIVUB instruction, A-3, A-14, A-44, A-48, A-53

DJNZ instruction, A-2, A-5, A-14, A-45, A-50, A-56

DJNZW instruction, A-2, A-5, A-14, A-45, A-50, A-56

Documents, related, 1-5–1-8

DOUBLE-WORD, defined, 3-3

DPTS instruction, A-3, A-15, A-45, A-51, A-57

Dump-word routine, 16-24

E

- EA#, 16-13, B-15
 - and P5.0, 6-13
 - and P5.3, 6-13
 - and programming modes, 16-14
 - idle, powerdown, reset status, B-24, B-25
- EE opcode, and unimplemented opcode interrupt, A-3, A-46
- EI instruction, 5-13, A-3, A-15, A-46, A-51, A-57
- EPA, 2-10, 11-1–11-24
 - and PTS, 11-12
 - block diagram, 11-2
 - capture data overruns, 11-21
 - capture/compare channels
 - programming, 11-18
 - choosing capture or compare mode, 11-19
 - clock prescaler, 11-16, 11-17
 - compare channels
 - programming, 11-18
 - compare modules
 - programming, 11-18
 - controlling the clock source and direction, 11-16, 11-17
 - determining event status, 11-24
 - enabling a timer/counter, 11-16, 11-17
 - enabling the compare function, 11-22
 - overruns, 11-12, 11-13

- re-enabling the compare event, 11-20, 11-22
- reloading the waveform generator, 11-20, 11-23, C-16
- resetting the timer in compare mode, 11-21
- resetting the timers, 11-21, 11-23
- selecting the capture/compare event, 11-19
- selecting the compare event, 11-22
- selecting the time base, 11-19, 11-22
- selecting up or down counting, 11-16, 11-17
- signals, 11-2
- starting an A/D conversion, 11-20, 11-23, C-16

See also port 1, port 6, PWM, timer/counters

- EPA compare control *x* register, 11-22, C-15
- EPA compare *x* time register, C-17
- EPA control *x* register, 11-19, C-18
- EPA time registers, C-21
- EPA0_CON, C-68
- EPA0_TIME, C-69
- EPA1_CON, C-68
- EPA1_TIME, C-69
- EPA2_CON, C-69
- EPA2_TIME, C-69
- EPA3_CON, C-69
- EPA3_TIME, C-69
- EPA4_CON, C-69
- EPA4_TIME, C-69
- EPA5:0, 11-2, B-16
- EPA5_CON, C-69
- EPA5_TIME, C-69
- EPTS instruction, 5-13, A-15, A-45, A-51, A-57
- ESD protection, 6-3, 6-7, 13-5
- Event processor array, *See* EPA
- EXT instruction, A-2, A-15, A-41, A-47, A-52
- EXTB instruction, A-2, A-16, A-41, A-47, A-52
- EXTINT, 5-3, 14-7, B-16
 - and idle mode, 14-5
 - and powerdown mode, 14-6, 14-7
 - hardware considerations, 14-7

F

- FaxBack service, 1-8
- FE opcode
 - and inhibiting interrupts, 5-9
- Floating point library, 3-4
- Formulas
 - A/D conversion result, 12-9, 12-13

- A/D conversion time, 12-7
- A/D error, 12-11
- A/D sample time, 12-7
- A/D series resistance, 12-10
- A/D threshold voltage, 12-5
- A/D voltage drop, 12-11
- capacitor size (powerdown circuit), 14-10
- PH1 and PH2 frequency, 2-8
- programming pulse width, OTPROM, 16-8
- programming voltage, 16-15
- SIO baud rate, 7-13
- state time, 2-8

FPAL-96, 3-4

FREQOUT, B-16

- Frequency generator, 8-1–8-9
 - application example, 8-4–8-9
 - data encoding example, 8-5
 - block diagram, 8-1
 - infrared remote control application, 8-5
 - overview, 8-1–8-2
 - programming
 - frequency, 8-3
 - output, 8-3
 - registers, 8-2
 - status, 8-4
- Frequency generator count register, 8-4, C-22
- Frequency register, 8-3, C-23

H

Handbooks, ordering, 1-6

Hardware

- A/D converter considerations, 12-10–12-13
- addressing modes, 3-5
- auto programming circuit, 16-26
- device considerations, 13-1–13-13
- device reset, 13-8, 13-10, 13-11, 13-12
- interrupt processor, 2-6, 5-1
- memory protection, 16-7, 16-17
- minimum configuration, 13-1
- NMI considerations, 5-6
- noise protection, 13-4
- pin reset status, B-23, B-25
- programming mode requirements, 16-13
- reset instruction, 3-11
- SIO port considerations, 7-8
- slave programming circuit, 16-16
- UPROM considerations, 16-7

Hypertext manuals and datasheets, downloading, 1-10

I

Idle mode, 2-11, 13-13, 14-4–14-5
 entering, 14-5
 pin status, B-23, B-25
 timeout control, 11-7

IDLPD instruction, A-2, A-16, A-46, A-51, A-57
 IDLPD #1, 14-5
 IDLPD #2, 14-6
 illegal operand, 13-9, 13-12

Immediate addressing, 3-6

INC instruction, A-2, A-16, A-41, A-47, A-52

INCB instruction, A-2, A-17, A-41, A-47, A-52

Indexed addressing, 3-9
 and register RAM, 4-10
 and windows, 4-19

Indirect addressing, 3-6
 and register RAM, 4-10
 with autoincrement, 3-7

Input pins
 level-sensitive, B-13
 sampled, B-13
 unused, 13-2

INST, B-16

Instruction set, 3-1
 and PSW flags, A-5
 code execution, 2-5, 2-6
 conventions, 1-3
 execution times, A-52–A-53
 lengths, A-47–A-52
 opcode map, A-2–A-3
 opcodes, A-41–A-46
 overview, 3-1–3-4
 protected instructions, 5-9
 reference, A-1–A-3
See also RISM

INT_MASK, 14-2

INTEGER, defined, 3-3

Interfacing with external memory
 configuring port pins, 15-5
 registers, 15-4
 signals, 15-1

Interrupt mask 1 register, 5-16, C-26

Interrupt mask register, 5-15, C-25

Interrupt pending 1 register, 5-22, C-28

Interrupt pending register, 5-21, C-27

Interrupts, 5-1–5-58
 controller, 2-6, 5-1
 end-of-PTS, 5-25
 inhibiting, 5-9
 latency, 5-9–5-11
 calculating, 5-10
 priorities, 5-4, 5-5
 modifying, 5-18–5-19
 procedures, PLM-96, 3-11
 processing, 5-2
 programming, 5-12–5-19
 selecting PTS or standard service, 5-12
 service routine
 processing, 5-19
 sources, 5-5
 unused inputs, 13-2
 vectors, 5-1, 5-5
 vectors, memory locations, 4-3

Italics, defined, 1-3

J

JBC instruction, A-2, A-5, A-17, A-41, A-50, A-56

JBS instruction, A-3, A-5, A-17, A-41, A-50, A-56

JC instruction, A-3, A-5, A-18, A-45, A-50, A-56

JE instruction, A-3, A-5, A-18, A-45, A-50, A-56

JGE instruction, A-2, A-5, A-18, A-45, A-50, A-56

JGT instruction, A-2, A-5, A-19, A-45, A-50, A-56

JH instruction, A-3, A-5, A-19, A-45, A-50, A-56

JLE instruction, A-3, A-5, A-19, A-45, A-50, A-56

JLT instruction, A-3, A-5, A-20, A-45, A-50, A-56

JNC instruction, A-2, A-5, A-20, A-45, A-50,
 A-56

JNE instruction, A-2, A-5, A-20, A-45, A-50, A-56

JNH instruction, A-2, A-5, A-21, A-45, A-50,
 A-56

JNST instruction, A-2, A-5, A-21, A-45, A-50,
 A-56

JNV instruction, A-2, A-5, A-21, A-45, A-50,
 A-56

JNVT instruction, A-2, A-5, A-22, A-45, A-50,
 A-56

JST instruction, A-3, A-5, A-22, A-45, A-50, A-56

Jump instructions, A-55
 conditional, A-5, A-50, A-56
 unconditional, A-49

JV instruction, A-3, A-5, A-22, A-45, A-50, A-56

JVT instruction, A-3, A-5, A-23, A-45, A-50, A-56

L

Latency, *See* bus-hold protocol, interrupts

LCALL instruction, A-3, A-23, A-45, A-50, A-56

LD instruction, A-2, A-23, A-44, A-49, A-55

LDB instruction, A-2, A-23, A-44, A-49, A-55

LDBSE instruction, A-3, A-24, A-44, A-49, A-55

LDBZE instruction, A-3, A-24, A-44, A-49, A-55

Level-sensitive input, B-13

Literature, 1-11

LJMP instruction, A-2, A-24, A-49, A-55

Logical instructions, A-48, A-53

LONG-INTEGGER, defined, 3-4

Lookup tables, software protection, 3-11

M

Manual contents, summary, 1-1

Manuals, online, 1-10

Measurements, defined, 1-5

Memory bus, 2-6

Memory controller, 2-4, 2-6

Memory map, 4-1, 4-2

Memory mapping

 auto programming mode, 16-27

Memory partitions, 4-1–4-19

 chip configuration bytes, 4-4

 chip configuration registers, 4-4

 interrupt and PTS vectors, 4-3

 OTPROM, 16-2

 program memory, 4-2, 16-2

 register file, 4-9

 register RAM, 4-10

 reserved memory, 4-3

 security key, 4-4

 SFRs, 4-4

 special-purpose memory, 4-2, 4-3, 16-2

Memory protection, 16-3–16-7

 CCR security-lock bits, 16-17

 UPROM security bits, 16-7

Memory space, *See* memory partitions

Microcode engine, 2-4

Miller effect, 13-7

Mode 0, SIO, 7-5

Mode 1, SIO, 7-7

Mode 2, SIO, 7-9

Mode 3, SIO, 7-9

Mode 4, SIO, 7-6

Modified quick-pulse algorithm, 16-9

MUL instruction, A-25, A-46, A-48, A-53

MULB instruction, A-25, A-46, A-48, A-53

Multiprocessor communications

 SIO port, 7-8, 7-9

MULU instruction, A-3, A-26, A-42, A-43, A-46, A-48, A-53

MULUB instruction, A-3, A-26, A-42, A-43, A-48, A-53

N

Naming conventions, 1-3–1-4

NEG instruction, A-2, A-27, A-41, A-48, A-53

Negative (N) flag, A-4, A-5, A-18, A-19, A-20

NEGB instruction, A-2, A-27, A-41, A-48, A-53

NMI, 5-3, 5-4, 5-6, B-16

 hardware considerations, 5-6

 idle, powerdown, reset status, B-24, B-25

Noise, reducing, 6-3, 6-4, 6-7, 12-12, 12-13, 13-4, 13-5, 13-6

NOP instruction, 3-11, A-3, A-27, A-46, A-51, A-57

 two-byte, *See* SKIP instruction

NORML instruction, 3-4, A-3, A-27, A-41, A-51, A-57

NOT instruction, A-2, A-28, A-41, A-48, A-53

Notational conventions, 1-3–1-4

NOTB instruction, A-2, A-28, A-41, A-48, A-53

Numbers, conventions, 1-4

O

ONCE mode, 2-11, 14-10

 entering, 14-11

 exiting, 14-11

ONCE#, 14-2, B-17

Ones register, C-29

Opcodes, A-41

 EE, and unimplemented opcode interrupt, A-3, A-46

 FE, and signed multiply and divide, A-3 map, A-2

 reserved, A-3, A-46

Operand types, *See* data types

Operands, addressing, 3-10

Operating modes, 2-11

OR instruction, A-2, A-28, A-43, A-48, A-53

ORB instruction, A-2, A-28, A-43, A-48, A-53

Oscillator

- and powerdown mode, 14-5
- external crystal, 13-6
- on-chip, 13-5

OTPROM

- controlling access to internal memory, 16-3–16-6
- controlling fetches from external memory, 16-6–16-7
- memory map, 16-2
- programming, 16-1–16-33
 - See also* programming modes
- ROM-dump mode, 16-30
- verifying, 16-30

Overflow (V) flag, A-4, A-5, A-21, A-22

Overflow-trap (VT) flag, A-4, A-5, A-11, A-22, A-23

P

P0.7:0, B-17

P0.7:4

- and programming modes, 16-14

P0_PIN, C-69

P1.3:0, B-17

P1.7:0, B-17

P1_DIR, C-69

P1_MODE, C-69

P1_PIN, C-69

P1_REG, C-69

P2.2 considerations, 14-7

P2.7:0, B-18

P2_DIR, C-69

P2_MODE, C-69

P2_PIN, C-69

P2_REG, C-69

P3.7:0, B-18

P4.7:0, B-18

P5.0–P5.7

- See also* port 5

P5.7:0, B-18

P5_PIN

- SFRs, 6-6

P6.7:0, B-18

P6_DIR, C-69

P6_MODE, C-69

P6_PIN, C-69

P6_REG, C-69

P7.7:0, B-19

PACT#, B-19

PALE#, 16-8, 16-10, 16-11, B-19

Parameters, passing to subroutines, 3-10

Parity, 7-7, 7-8, 7-9

PBUS, 16-12

PBUS15:0, B-19

PC (program counter), 2-4

- master, 2-6

- slave, 2-6

Peripheral Interrupt mask register, 5-17, C-35

Peripheral interrupt pending register, 5-23, C-37

Peripherals, internal, 2-8

Phase compare register, 9-17, C-59

Pin-out diagrams, B-3, B-4, B-5, B-7, B-8, B-11, B-12

PLM-96

- conventions, 3-9, 3-10, 3-11

- interrupt procedures, 3-11

PMODE, 16-11, 16-13

- and programming modes, 16-14

PMODE3:0, B-19

POP instruction, A-3, A-29, A-45, A-49, A-54

POPA instruction, A-2, A-29, A-46, A-49, A-54

POPF instruction, A-2, A-29, A-46, A-49, A-54

Port 0, 6-2, B-17

- considerations, 6-4, 12-13, 13-5

- idle, powerdown, reset status, B-25

- input only pins, 6-2

- overview, 6-1

- structure, 6-3

Port 1, 6-2, B-17

- considerations, 6-4, 6-12

- idle, powerdown, reset status, B-23, B-25

- input buffer, 6-7

- input only pins, 6-2

- logic tables, 6-9

- operation, 6-4

- overview, 6-1

- SFRs, 6-6, 14-3

Port 2, 14-2, B-18

- considerations, 6-12

- idle, powerdown, reset status, B-23, B-25

- operation, 6-4

- overview, 6-1

- P2.7 considerations, 6-12

- P2.7 reset status, 6-7

- SFRs, 6-6, 14-3
- Port 3, B-18
 - addressing, 6-14
 - idle, powerdown, reset status, B-23, B-25
 - operation, 6-15–6-16
 - overview, 6-1
 - pin configuration, 6-14
 - structure, 6-15
- Port 4, B-18
 - addressing, 6-14
 - idle, powerdown, reset status, B-23, B-25
 - operation, 6-15–6-16
 - overview, 6-1
 - pin configuration, 6-14
 - structure, 6-15
- Port 5, B-18
 - considerations, 6-12
 - idle, powerdown, reset status, B-23, B-25
 - operation, 6-4, 6-12
 - overview, 6-1
 - pin configuration, 6-12
 - SFRs, 6-6, 14-3
- Port 6, B-18
 - configuration, 6-17
 - idle, powerdown, reset status, B-23, B-25
 - operation, 6-17
 - output configuration register, 6-18, C-62
 - overview, 6-1
- Port 7, B-19
 - idle, powerdown, reset status, B-24
 - operation, 6-4
 - overview, 6-1
 - SFRs, 6-6, 14-3
- Port *x* data output register, C-34
- Port *x* I/O direction register, C-30
- Port *x* mode register, C-31
- Port *x* pin input register, C-33
- Port, serial, *See* SIO port
- Ports
 - general-purpose I/O, 2-9
 - unused inputs, 13-2
- Power and ground pins
 - minimum hardware connections, 13-5
- Power consumption, reducing, 2-11, 14-5
- Power-up sequence, programming modes, 16-14
- Powerdown mode, 2-11, 14-5–14-10
 - circuitry, external, 14-8, 14-10
 - disabling, 14-5
 - enabling, 14-5
 - entering, 14-6
 - exiting, 14-6
 - with EXTINT, 14-7–14-10
 - with RESET#, 14-6
 - with V_{pp} , 14-6
 - pin status, B-23
 - reset status, B-25
- Powerdown sequence, programming modes, 16-14
- Prefetch queue, 2-6
- Priority encoder, 5-4
- Processor status word, *See* PSW
- Product information, ordering, 1-6
- PROG#, 16-10, 16-12, B-20
- Program counter, *See* PC
- Program memory, 4-2
- Program-word routine, 16-22
- Programming
 - frequency generator
 - frequency, 8-3
 - output, 8-3
- Programming modes, 16-1–16-33
 - algorithms, 16-20, 16-21, 16-23, 16-28
 - auto, 16-2
 - entering, 16-13, 16-14
 - exiting, 16-14
 - hardware requirements, 16-13
 - pin functions, 16-11–16-13
 - selecting, 16-13
 - serial port, 16-2
 - slave, 16-1
- Programming pulse width register, 16-8, C-39
- Programming voltages, 13-1, 14-2, 16-13, B-21
 - calculating, 16-15
- PSW, 2-4, 3-11, C-40
 - flags, and instructions, A-5
- PTS, 2-4, 2-6, 2-11, 5-1
 - A/D scan mode, 5-32–5-37
 - and A/D converter, 5-33
 - asynchronous serial I/O receive mode, 5-55–5-58
 - asynchronous serial I/O transmit mode, 5-50–5-54
 - block transfer mode, 5-30
 - control block, *See* PTSCB
 - cycle execution time, 5-12
 - cycle, defined, 5-30
 - initializing PTS control blocks, 5-24

- instructions, A-51, A-57
- interrupt latency, 5-11
- interrupt processing flow, 5-2
- routine, defined, 5-1
- serial I/O modes, 5-37–5-58
- single transfer mode, 5-27
- synchronous serial I/O receive mode, 5-47–5-50
- synchronous serial I/O transmit mode, 5-43–5-46
- vectors, memory locations, 4-3
- See also* PWM
- PTS select register, 5-14, C-42
- PTS service register, 5-26, C-43
- PTSCB, 5-4, 5-9
 - A/D scan mode, 5-33
 - block transfer mode, 5-31
 - memory locations, 4-3
 - PTSCON register, 5-27
 - PTSCOUNT register, 5-25
 - single transfer mode, 5-28
- PTSCB1
 - serial I/O mode, 5-38
- PTSCB2
 - serial I/O mode, 5-41
- Pulse-width modulator, *See* PWM
- PUSH instruction, A-3, A-29, A-45, A-49, A-54
- PUSHA instruction, A-2, A-30, A-46, A-49, A-54
- PUSHF instruction, A-2, A-30, A-46, A-49, A-54
- PVER, 16-9, 16-11, B-20
- PWM, 10-1
 - and cascading timer/counters, 11-7
 - block diagram, 10-2
 - D/A converter, 10-10
 - duty cycle, 10-4
 - enabling outputs, 10-8
 - generating analog outputs, 10-10
 - highest-speed, 11-14
 - low-speed, 11-7, 11-13
 - overview, 10-1
 - programming duty cycle, 10-4
 - typical waveforms, 10-4
 - with dedicated timer/counter, 11-14
- PWM control *x* register, 10-7, C-46
- PWM count register, 10-8, C-44
- PWM period register, 10-6, C-45
- PWM1:0, B-20

Q

- Quadrature clocking, 11-7
- Quick reference guides, ordering, 1-8

R

- RALU, 2-4–2-6
- RD#, B-20
 - considerations, 6-13
 - idle, powerdown, reset status, B-23, B-25
- Read cycles
 - 16-bit data bus, 15-14
 - 8-bit data bus, 15-16
- READY, 15-17–15-21, 16-25
 - and wait states, 15-18
 - considerations, 6-13
 - idle, powerdown, reset status, B-23, B-25
 - timing definitions, 15-20
 - timing diagram, 15-19
 - timing requirements, 15-18
- REAL variables, 3-4
- Register bits
 - naming conventions, 1-4
 - reserved, 1-4
- Register file, 2-4, 4-9
 - and windowing, 4-9, 4-12
 - See also* windows
- Register RAM, 4-10
 - and idle mode, 14-4
 - and powerdown mode, 14-5
- Register-direct addressing, 4-10
 - and register RAM, 4-10
 - and windows, 4-12, 4-19
- Registers
 - AD_COMMAND, 12-8, C-6
 - AD_RESULT (read), 12-9, C-7
 - AD_RESULT (write), 12-6, C-8
 - AD_TEST, 12-5, C-9
 - AD_TIME, 12-7, C-10
 - addresses and reset values, C-2
 - allocating, 3-10
 - CCR0, 15-7, C-11
 - CCR1, 15-9, C-13
 - COMP_x_CON, 11-22, C-15
 - COMP_x_TIME, C-17
 - EPAX_CON, 11-19, C-18
 - EPAX_TIME, C-21
 - external memory, 15-4

- FREQ_CNT, 8-4, C-22
- FREQ_GEN, 8-3, C-23
- GEN_CON, 13-9, C-24
 - grouped by modules, C-1
- INT_MASK, 5-15, C-25
- INT_MASK1, 5-16, 7-2, C-26
- INT_PEND, 5-21, 7-3, C-27
- INT_PEND1, 5-22, 7-3, C-28
 - naming conventions, 1-4
- ONES_REG, C-29
- P0_PIN, 6-3, 6-4
- P1_MODE
 - considerations, 6-12
- P1_PIN, 6-3, 6-4, 7-3
- P2_MODE
 - considerations, 6-12
- P2_REG
 - considerations, 6-12
- P5_MODE
 - considerations, 6-12, 6-13
- PI_MASK, 5-17, 7-4, C-35
- PI_PEND, 5-23, 7-4, C-37
- PPW, 16-8, C-39
- PSW, C-40
- PTSSEL, 5-14, C-42
- PTSSRV, 5-26, C-43
- PWM_COUNT, 10-8, C-44
- PWM_PERIOD, 10-6, C-45
- PWMx_CONTROL, 10-7, C-46
- Px_DIR, 6-6, 6-9, 6-10, 6-11, C-30
- Px_MODE, 6-6, 6-9, 6-10, 6-11, C-31
- Px_PIN, 6-2, 6-6, 6-7, 6-9, 6-14, 6-16, C-33
- Px_REG, 6-6, 6-9, 6-10, 6-11, 6-14, 6-16, C-34
- RALU, 2-4, 2-5, 4-10
 - reset values and addresses, C-2
- SBUF0_RX, 7-4
- SBUF0_TX, 7-4
- SBUF1_RX, 7-4
- SBUF1_TX, 7-4
- SBUFx_RX, C-47
- SBUFx_TX, C-48
- SP, C-49
- SP_PPW, 16-8
- SP0_BAUD, 7-4
- SP0_CON, 7-4
- SP0_STATUS, 7-4
- SPI_BAUD, 7-4
- SPI_CON, 7-4
- SPI_STATUS, 7-4
- SPx_BAUD, 7-12, C-50
- SPx_CON, 7-10, C-51
- SPx_STAUS, 7-15, C-52
- T1CONTROL, 11-16, C-53
- T1RELOAD, C-54
- T2CONTROL, 11-17, C-55
- TIMERx, C-56
- USFR, 16-7, C-57
 - using, 3-9
- WATCHDOG, C-58
- WG_COMPx, 9-17, C-59
- WG_CONTROL, 9-18, C-60
- WG_COUNTER, 9-19, C-61
- WG_OUTPUT (port 6), 6-17, 6-18, C-62
- WG_OUTPUT (waveform generator), 9-13, C-63
- WG_PROTECT, 9-15, C-66
- WG_RELOAD, 9-16, C-67
- WSR, 4-12, C-68
- ZERO_REG, C-71
- Reserved bits, defined, 1-4
- Reset, 13-9
 - and CCB fetches, 4-4
 - circuit diagram, 13-11
 - general configuration register, 13-9, C-24
 - pin status, B-23, B-25
 - status
 - CLKOUT/P2.7, 6-7
 - with illegal IDLPD operand, 13-12
 - with RESET# pin, 13-10
 - with RST instruction, 13-9, 13-12
 - with watchdog timer, 13-12
- RESET#, 13-1, 14-2, B-20
 - and CCB fetch, 13-8
 - and CLKOUT, 13-8
 - and device reset, 13-8, 13-9, 13-10
 - and ONCE mode, 14-11
 - and powerdown mode, 14-6
 - and programming modes, 16-13, 16-14
 - idle, powerdown, reset status, B-24, B-25
- Resonator, ceramic, 13-6
- RET instruction, A-2, A-30, A-46, A-50, A-55, A-56
- ROM-dump mode, 16-30
 - security key verification, 16-30

RST instruction, 3-11, 13-9, 13-12, A-3, A-31, A-46, A-51, A-57

Run-time programming, 16-32–16-33
code example, 16-33

RXD, B-20
and SIO port mode 0, 7-5, 7-7
and SIO port modes 1, 2, and 3, 7-7

S

Sampled input, B-13

SBUF_x_RX, C-70

SBUF_x_TX, C-70

SCALL instruction, A-3, A-31, A-41, A-47, A-50, A-55, A-56

SCLK_x#, 7-2

Security key

verification, 16-30

Serial I/O modes, *See* PTS

Serial I/O port, *See* SIO port

Serial port control *x* register, 7-10, C-51

Serial port receive buffer *x* register, C-47

Serial port status *x* register, 7-15, C-52

Serial port transmit buffer *x* register, C-48

Serial port *x* baud rate register, 7-12, C-50

Set, defined, 1-3

SETC instruction, A-3, A-31, A-46, A-51, A-57

SFRs

and idle mode, 14-4

and powerdown mode, 14-5

CPU, 4-11

memory-mapped, 4-5

peripheral, 4-4, 4-5

and windowing, 4-12

reserved, 3-10, 4-4, 4-11

with indirect or indexed operations, 3-10, 4-4, 4-11

with read-modify-write instructions, 4-4

Shift instructions, A-51, A-57

SHL instruction, A-3, A-32, A-41, A-51, A-57

SHLB instruction, A-3, A-32, A-41, A-51, A-57

SHLL instruction, A-3, A-33, A-41, A-51

SHORT-INTEGGER, defined, 3-2

SHR instruction, A-3, A-33, A-41, A-51, A-57

SHRA instruction, A-3, A-34, A-41, A-51, A-57

SHRAB instruction, A-3, A-34, A-41, A-51, A-57

SHRAL instruction, A-3, A-35, A-41, A-51, A-57

SHRB instruction, A-3, A-35, A-41, A-51, A-57

SHRL instruction, A-3, A-36, A-41, A-51, A-57

Signals

configuring for external memory interfacing, 15-5

default conditions, B-23, B-25

descriptions, B-13–B-22

external memory, 15-1

AD15:0, 15-1

ADV#, 15-1

ALE, 15-2

BHE#, 15-2

BUSWIDTH, 15-2

CLKOUT, 15-2

EA#, 15-3

INST, 15-3

RD#, 15-3

READY, 15-3

WR#, 15-3

WRH#, 15-4

WRL#, 15-4

functional listings, B-2, B-6, B-9

name changes, B-1

naming conventions, 1-4

Single transfer mode, *See* PTS

SIO port, 2-9, 7-1

9-bit data, *See* mode 2, mode 3

block diagram, 7-1

calculating baud rate, 7-13, 7-14

enabling interrupts, 7-14

enabling parity, 7-10–??

framing error, 7-16

half-duplex considerations, 7-8

interrupts, 7-7, 7-9, 7-16

mode 0, 7-5–7-6

mode 1, 7-7

mode 2, 7-7, 7-8

mode 3, 7-7, 7-9

multiprocessor communications, 7-8, 7-9

overrun error, 7-16

programming, 7-10

receive interrupt (RI) flag, 7-16

selecting baud rate, 7-12–7-14

SFRs, 7-2

signals, 7-2

status, 7-15–7-16

transmit interrupt (TI) flag, 7-16

See also mode 0, mode 1, mode 2, mode 3, port 2

SJMP instruction, A-2, A-36, A-41, A-47, A-49, A-55
 SKIP instruction, A-2, A-36, A-41, A-51, A-57
 Slave programming mode, 16-15–16-24
 address/command decoder routine, 16-19, 16-20
 algorithm, 16-19–16-24
 circuit, 16-16
 dump-word routine, 16-19, 16-23
 entering, 16-19
 program-word routine, 16-19, 16-21
 security key programming, 16-15
 timings, 16-22, 16-24
 Software
 addressing modes, 3-9
 conventions, 3-9–3-11
 device reset, 13-12
 interrupt service routines, 5-19
 linking subroutines, 3-10
 protection, 3-11
 trap interrupt, 5-4, 5-6, 5-9
 SP_STATUS, 7-15
 SPE bit, 7-3
 Special instructions, A-51, A-57
 Special operating modes
 SFRs, 14-2
 Special-purpose memory, 4-2
 SPx bit, 7-4
 SPx_BAUD, C-70
 SPx_CON, C-70
 SPx_STATUS, C-70
 ST instruction, A-2, A-37, A-45, A-49, A-55
 Stack instructions, A-49, A-54
 Stack pointer, 4-10, C-49
 and subroutine call, 4-10
 initializing, 4-11
 Standard bus-control mode
 decoding WRL# and WRH#, 15-22
 example system, 15-23
 signals, 15-22
 State time, defined, 2-8
 STB instruction, A-2, A-37, A-45, A-49, A-55
 Sticky bit (ST) flag, 3-4, A-4, A-5, A-21, A-22
 SUB instruction, A-3, A-37, A-42, A-47, A-52
 SUBB instruction, A-3, A-38, A-42, A-43, A-47, A-52
 SUBC instruction, A-3, A-38, A-44, A-47, A-52
 SUBCB instruction, A-3, A-38, A-44, A-47, A-52

Subroutines
 linking, 3-10
 nested, 4-10
 Symbols
 signal status, B-23
 System bus timing, 15-32

T

T1CLK, 11-2, B-21
 T1CONTROL, C-70
 T1DIR, 11-2, B-21
 T1RELOAD, C-70
 T2CONTROL, C-70
 Technical support, 1-11
 Terminology, 1-3
 T1JMP instruction, A-2, A-39, A-45, A-49, A-55
 Timer 1 control register, 11-16, C-53
 Timer 1 reload register, C-54
 Timer 2 control register, 11-17, C-55
 Timer *x* register, C-56
 Timer, watchdog, *See* watchdog timer
 Timer/counters, 2-10
 and PWM, 11-13, 11-14
 cascading, 11-7
 count rate, 11-6
 programming, 11-15
 quadrature clocking, 11-7
 resolution, 11-6
 signals, 11-2
 See also EPA
 TIMER1, C-70
 TIMER2, C-70
 Timing
 dump-word routine, 16-24
 instruction execution, A-52–A-53
 internal, 2-7, 2-8
 interrupt latency, 5-9–5-12, 5-30
 program-word routine, 16-22
 PTS cycles, 5-12
 SIO port mode 0, 7-6, 7-7
 SIO port mode 1, 7-8
 SIO port mode 2, 7-9
 SIO port mode 3, 7-9
 slave programming routines, 16-22, 16-24
 Timing definitions
 BUSWIDTH, 15-13
 READY, 15-20

- Timing diagrams
 - 16-bit data bus, 15-15
 - 8-bit data bus, 15-17
 - BUSWIDTH, 15-12
 - READY, 15-19
 - system bus timing, 15-32
 - Timing requirements
 - BUSWIDTH, 15-13
 - READY, 15-18
 - TRAP instruction, 5-6, A-2, A-39, A-46, A-50, A-55, A-56
 - TRAP interrupt, 5-4
 - TXD, B-21
 - and SIO port mode 0, 7-5
- U**
- UART, 2-9, 7-1
 - Unerasable PROM register, 16-7, C-57
 - Unimplemented opcode interrupt, 3-11, 5-4, 5-6, 5-9
 - Units of measure, defined, 1-5
 - Universal asynchronous receiver and transmitter, *See* UART
 - UPROM, 16-6
 - programming, 16-6–16-7
 - USFR, 16-7
- V**
- V_{CC}, 13-1, B-21
 - and programming modes, 16-14
 - V_{PP}, 13-1, 14-2, 16-13, B-21
 - and programming modes, 16-14
 - V_{REF}, 12-5, 13-1, B-21
 - V_{SS}, 13-1, B-21
 - and programming modes, 16-14
- W**
- Wait states, 15-17–15-20
 - controlling, 15-18
 - Watchdog timer, 2-11, 3-11, 3-12, 13-9, 13-12
 - and idle mode, 14-5
 - selecting reset interval, 13-13
 - Watchdog timer register, C-58
 - Waveform generator, 9-1, 9-25
 - block diagram, 9-2
 - control and protection circuitry, 9-5
 - dead time
 - and duty cycle, 9-19
 - generator circuitry, 9-5
 - design considerations, 9-19, 9-20
 - EXTINT interrupts and protection circuitry, 9-21
 - interrupts, 9-20
 - operating modes, 9-7, 9-11
 - center-aligned, 9-8, 9-9, 9-10
 - edge-aligned, 9-8, 9-10, 9-11
 - overview, 9-1
 - phase driver channels, 9-5
 - programming, 9-12, 9-20
 - carrier period, 9-16
 - EXTINT
 - interrupts, 9-15
 - outputs, 9-12, C-63–C-65
 - protection circuitry, 9-15
 - programming example, 9-21, 9-25
 - protection circuitry, 9-6
 - register
 - buffering and synchronization, 9-6, 9-7
 - updates, 9-8
 - registers, 9-3, 9-4
 - signals, 9-3
 - status, 9-19
 - timebase generator, 9-4
 - Waveform generator control register, 9-18, C-60
 - Waveform generator counter register, 9-19, C-61
 - Waveform generator output configuration register, 9-13, C-63
 - Waveform generator reload register, 9-16, C-67
 - Waveform protection register, 9-15, C-66
 - WDE bit, 13-12
 - WG3:1, B-22
 - WG3:1#, B-22
 - Window selection register, *See* WSR
 - Windowing, 4-12–4-19
 - examples, 4-16–4-19
 - See also* windows
 - Windows, 4-12–4-19
 - addressing, 4-16
 - and addressing modes, 4-19
 - and memory-mapped SFRs, 4-15
 - base address, 4-14, 4-15
 - locations that cannot be windowed, 4-15
 - offset address, 4-14
 - selecting, 4-13
 - setting up with linker loader, 4-17

WSR values and direct addresses, 4-15
WORD, defined, 3-2
World Wide Web, 1-11
WR#, B-22
 idle, powerdown, reset status, B-23, B-25
WRH#, B-22
Write cycles
 16-bit data bus, 15-14
 8-bit data bus, 15-16
Write strobe mode
 example system, 15-26
 signals, 15-25
WRL#, B-22
WSR, 4-13, C-68

X

X, defined, 1-5
x, defined, 1-3
XCH instruction, A-2, A-3, A-40, A-41, A-49,
 A-55
XCHB instruction, A-2, A-3, A-40, A-41, A-49,
 A-55
XOR instruction, A-2, A-40, A-43, A-48, A-53
XORB instruction, A-2, A-40, A-43, A-44, A-48,
 A-53
XTAL1, 13-2, B-22
 and Miller effect, 13-7
 and programming modes, 16-13
 and SIO baud rate, 7-14
 hardware connections, 13-6, 13-7
XTAL2, 13-2, B-22
 hardware connections, 13-6, 13-7

Y

y, defined, 1-3

Z

Zero (Z) flag, A-4, A-5, A-18, A-19, A-20, A-21
Zero register, C-71

Free Manuals Download Website

<http://myh66.com>

<http://usermanuals.us>

<http://www.somanuals.com>

<http://www.4manuals.cc>

<http://www.manual-lib.com>

<http://www.404manual.com>

<http://www.luxmanual.com>

<http://aubethermostatmanual.com>

Golf course search by state

<http://golfingnear.com>

Email search by domain

<http://emailbydomain.com>

Auto manuals search

<http://auto.somanuals.com>

TV manuals search

<http://tv.somanuals.com>