PIC32MX
Family Reference Manual

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Trademarks

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
ISO/TS 16949:2002

Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

Section 1. Introduction

HIGHLIGHTS

This section of the manual contains the following topics:

1.1 INTRODUCTION

Microchip's PIC32MX series of 32-bit microcontrollers is designed to fulfill customers' requirements for enhanced features and performance for their MCU-based applications.

Common attributes among all devices in the PIC32MX series are:

- Pin, peripheral and source code compatibility with the PIC24F128GAXXX family
- MIPS32® M4K™ processor core
- Common development tools

1.2 OBJECTIVE OF THIS MANUAL

This manual describes the PIC32MX series of 32-bit microcontrollers. It explains the family architecture and operation of the peripheral modules, but does not cover the specifics of each device in the family. Users should refer to the respective device's data sheet for device-specific details, such as:

- Pinout and packaging details
- Memory map
- List of peripherals included on the device, including multiple instances of peripherals
- Device-specific electrical specifications and characteristics

1.3 DEVICE STRUCTURE

The PIC32MX architecture has been broken down into the following functional blocks:

- MCU Core
- System Memory
- System Integration
- Peripherals

1.3.1 MCU Core

The MCU core consists of these essential basic features.

- 32-bit RISC MIPS32 M4K Core
- Single Cycle ALU
- Load-Store Execution Unit
- 5-Stage Pipeline
- 32-bit Address and 32-bit Data Buses
- Two 32-element, 32-bit General Purpose Register Files
- FMT – Fixed Mapping Translation Memory Management
- FMDU – Fast-Multiply-Divide Unit
- MIPS32® Compatible Instruction Set
- MIPS16e™ Code Compression Instruction Set Architecture Support

The CPU section of this manual discusses the PIC32MX MCU core.

1.3.2 System Memory

The system memory provides on-chip nonvolatile Flash memory and volatile SRAM memory, featuring user and protected kernel-segment-partitioning for real-time operating systems. The following sections of this manual discuss the PIC32MX system memory:

- Section 3. Memory Organization
- Section 5. Flash Programming

Flash Memory Technology

- The Flash can be used for program memory or data.
- The Flash allows program memory to be electrically erased or programmed under software control during normal device operation.
- The PIC32MX series has full-speed execution directly from program Flash through the use of on-chip prefetch buffering by the Prefetch module.
- The Flash has the capability to page erase, word or row program.

1.3.3 System Integration

System integration consists of a comprehensive set of modules and features that tie the MCU core and peripheral modules into a single operational unit. System integration features also provide these advantages:

- Decreased system cost, by bringing traditionally off-chip functions into the microcontroller
- Increased design flexibility, by adding a wider range of operating modes
- Increased system reliability, by enhancing the ability to recover from unexpected events

The following sections of this manual discuss the PIC32MX system integration:

- Section 3. Memory Organization
- Section 4. Prefetch Module
- Section 5. Flash Programming
- Section 6. Oscillator
- Section 7. Resets
- Section 8. Interrupts
- Section 9. Watchdog Timer and Power-up Timer
- Section 10. Power-Saving Modes
- Section 31. Direct Memory Access (DMA) Controller with programmable Cyclic Redundancy Check (CRC)
- Section 32. High-Level Integration (Configuration, Code Protection and Voltage Regulation)
- Section 33. Device Programming, Debugging, In-Circuit and In-Circuit Testing

1.3.4 Peripherals

The PIC32MX devices have many peripherals that allow it to interface with the external world. The following sections of this manual discuss the PIC32MX peripherals:

- Section 12. I/O Ports
- Section 13. Parallel Master Port
- Section 14. Timers
- Section 15. Input Capture Module
- Section 16. Output Compare/Pulse Width Modulation (PWM) Module
- Section 17. 10-bit A/D Converter
- Section 19. Comparator Module
- Section 20. Comparator Voltage Reference Module
- Section 21. UART Module
- Section 23. SPI Module
- Section 24. I^2C™ Module
- Section 27. USB OTG
- Section 29. Real-Time Clock/Calendar (RTCC) Module

1.4 DEVELOPMENT SUPPORT

Microchip offers a wide range of development tools that allow users to efficiently develop and debug application code. Microchip's development tools can be broken down into four categories:

- Code generation
- Hardware/software debug
- Device programmer
- Product evaluation boards

As new tools are developed, the latest product briefs and user guides can be obtained from the Microchip web site (www.microchip.com) or from local Microchip Sales Offices.

Microchip offers other references and support to speed the development cycle. These include:

- Application notes
- Reference designs
- Microchip web site
- Local sales offices with field application support
- Corporate Applications support line
- Getting Stated guide
- "How to" brochures
- Masters Conferences
- Webinars
- Design Centers

These can all be found on the Microchip web site. Also, the Microchip web site lists other sites that may provide useful references.

1.5 STYLE AND SYMBOL CONVENTIONS

Throughout this document, certain style, format, and font conventions are used to signal particular distinctions for the affected text. Table 1-1 lists these conventions, the MCU-industry-specific symbols, and non-conventional word definitions and abbreviations used in this manual.

Located at the rear of this document, a glossary provides additional word and abbreviation definitions for content used in this manual.

1.5.1 Document Conventions

Table 1-1 defines some of the symbols, terms and typographic conventions used in this manual.

Table 1-1: Document Conventions

SYMBOL AND TERM CONVENTIONS:	
Convention	**Description**
set	To force a bit/register to a value of logic '1'.
clear	To force a bit/register to a value of logic '0'.
reset	1. To force a register/bit to its default state. 2. A condition in which the device places itself after a device Reset occurs. Some bits will be forced to '0' (such as interrupt enable bits), while others will be forced to '1' (such as the I/O data direction bits).
: (colon)	Specifies a range or concatenation of registers/bits/pins. Concatenation order (left to right) usually specifies a positional relationship (MSb to LSb, higher to lower). For example, TMR3:TMR2 indicates the concatenation of two 16-bit registers to form a 32-bit timer value, with the value of TMR3 representing the most significant half-word of the value.
< >	Specifies a bit location or range of locations within a particular register or field of similarly-named bits. For example, PTCON<2:0> specifies the range of the 3 Least Significant bits of the register PTCON.
MSb LSb	Most Significant bit and Least Significant bit.
MSB, LSB	Most Significant Byte, Least Significant Byte. (A Byte is 8-bits wide.)
mshw, lshw	Most Significant half-word and least significant half-word A Half-Word is 16-bits wide
msw, lsw	Most Significant Word and Least Significant Word. (A Word is 32-bits wide.)
0xnn	Designates the number 'nn' in the hexadecimal number system. This convention is used in code examples, and is equivalent to the notation 'nnh' used in text. For example, 0x13 is equivalent to 13h.
FONT CONVENTIONS:	
Arial Font	The standard font used for all text, figures and tables within this manual. Other fonts, as described below, are used to set off mathematical and logical expressions, or device instruction code, from descriptive text.
Courier New Font	Within text, this font is used for contrast with the standard text font and specifically denote the following: 1. an instruction set mnemonic or assembler code fragment. 2. the binary value of a bit, range of bits, or a register. 3. the logical state of a digital signal. Within code examples, this font is used exclusively to denote an assembly or high-level language instruction sequence.
Times New Roman Font	The standard font for mathematical expressions and variables.
GRAPHIC CONVENTIONS:	
Note	A note presents information that requires emphasis: either to help users avoid a common pitfall, or to make them aware of operating differences between some device family members. A note is usually in a shaded box, unless it is used in a bit description, or as a table or diagram footnote. **Note:** This is a Note in a shaded note box.
Register Cells	A bit name that appears in a grayed-out cell of a register signals that the bit is not relevant to the peripheral module described in that particular section of the manual. FRZ

1.5.2 Electrical Specifications

Throughout this manual, there are references to electrical specifications and their parameter numbers. Table 1-2 shows the parameter numbering convention for PIC32MX devices. A parameter number represents a unique set of characteristics and conditions that is consistent between every data sheet, though the actual parameter value may vary from device to device.

This manual describes a family of devices and, therefore, does not specify the parameter values. To determine the parameter values for a specific device, users should refer to the “Electrical Specifications” section of that device’s data sheet.

Table 1-2: Electrical Specification Parameter Numbering Convention

Parameter Number Format	Comment
DXXX	DC Specification
AXXX	DC Specification for Analog Peripherals
XXX	Timing (AC) Specification
PDXXX	Device Programming DC Specification
PXXX	Device Programming Timing (AC) Specification

Legend: XXX represents a parameter number.

1.6 RELATED DOCUMENTS

Microchip, as well as other sources, offers additional documentation to aid you as you develop PIC32MX-based applications. The list below contains the most common documentation, but other documents may also be available. Please check the Microchip web site (www.microchip.com) for the latest published technical documentation.

1.6.1 Microchip Documentation

The following PIC32MX documentation is available from Microchip. Many of these documents provide application-specific information that gives actual examples of using, programming, and designing with PIC32MX microcontrollers.

1. *PIC32MX Family Reference Manual*
   The family reference manual describes the PIC32MX architecture and operation of the peripheral modules, but does not cover the specifics of each device in the family.

2. PIC32MX Data Sheets
   The data sheets contain device-specific information, such as pinout and packaging details, electrical specifications and memory maps.

3. *PIC32MX Programming Specification*
   The programming specifications contain detailed descriptions of, and electrical and timing specifications for, the programming process. Both In-Circuit Serial Programming™ (ICSP™) and Enhanced ICSP are described in detail.

1.6.2 Third-Party Documentation

Microchip does not review third-party documentation for technical accuracy, but these references may be helpful to understand operation of the devices. The Microchip web site may have information on these third-party documents.

1.7 REVISION HISTORY

Revision A (September 2007)

This is the initial version of this document.

Revision B (October 2007)

Updated document to remove Confidential status.

Revision C (April 2008)

Revised status to Preliminary; Revised Section 1.1.

NOTES:

Section 2. MCU

HIGHLIGHTS

This section of the manual contains the following topics:

2.1 INTRODUCTION

The PIC32MX Microcontroller Unit (MCU) is a complex system-on-a-chip that is based on a M4K™ core from MIPS® Technologies. M4K™ is a state-of-the-art 32-bit, low-power, RISC processor core with the enhanced MIPS32® Release 2 Instruction Set Architecture. This chapter provides an overview of the CPU features and system architecture of the PIC32MX family of microcontrollers.

Key Features

- Up to 1.5 DMIPS/MHz of performance
- Programmable prefetch cache memory to enhance execution from Flash memory
- 16-bit Instruction mode (MIPS16e) for compact code
- Vectored interrupt controller with 63 priority levels
- Programmable User and Kernel modes of operation
- Atomic bit manipulations on peripheral registers (Single cycle)
- Multiply-Divide unit with a maximum issue rate of one 32 × 16 multiply per clock
- High speed Microchip ICD port with hardware-based non-intrusive data monitoring and application data streaming functions
- EJTAG debug port allows extensive third party debug, programming and test tools support
- Instruction controlled power management modes
- Five stage piplined instruction execution
- Internal Code protection to help protect intellectual property

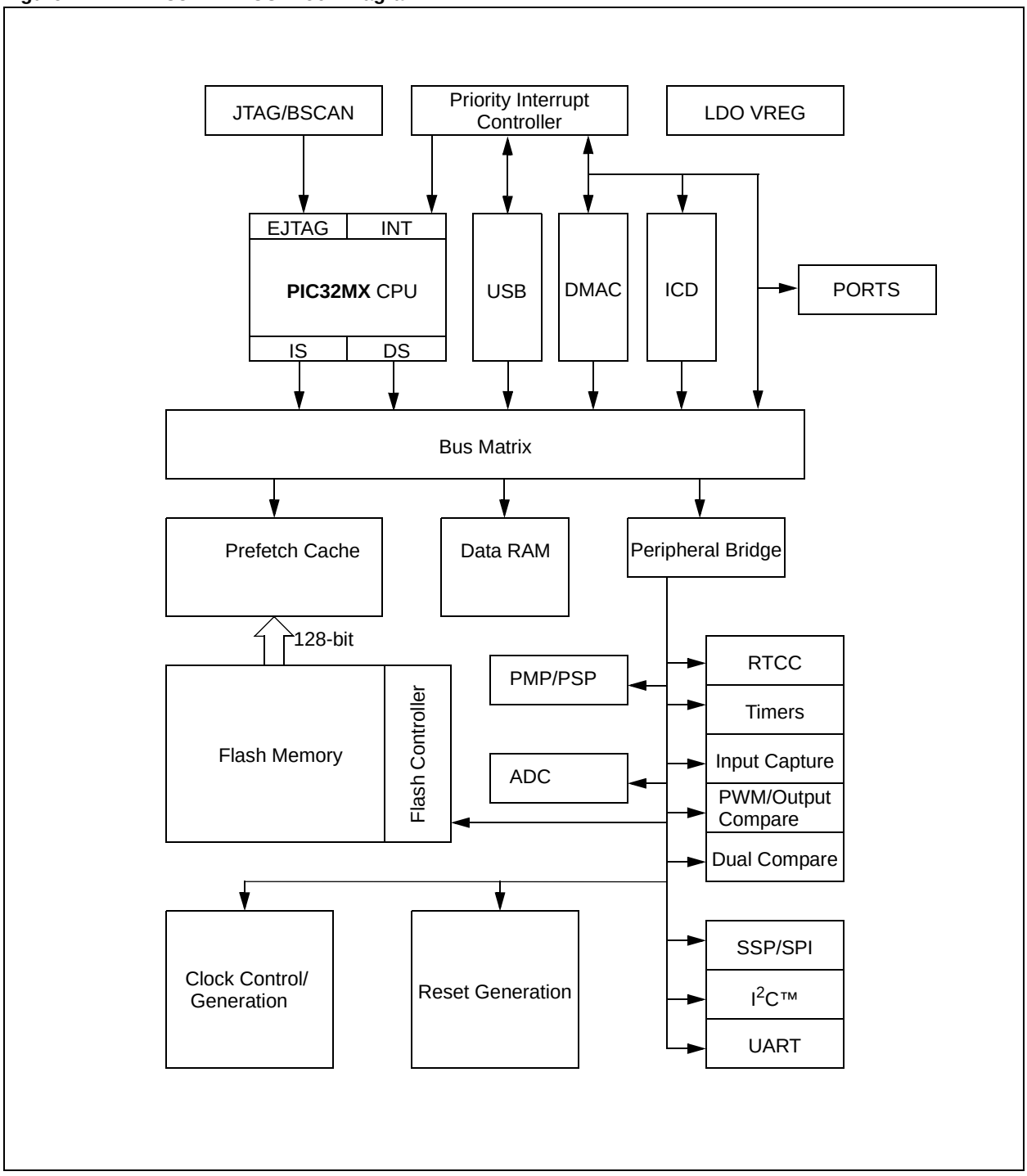
Related MIPS® Documentation

- MIPS M4K™ Software User's Manual – MD00249-2B-M4K-SUM
- MIPS® Instruction Set – MD00086-2B-MIPS32BIS-AFP
- MIPS16e™ – MD00076-2B-MIPS1632-AFP
- MIPS32® Privileged Resource Architecture – MD00090-2B-MIPS32PRA-AFP

2.2 ARCHITECTURE OVERVIEW

The PIC32MX family processors are complex systems-on-a-chip that contain many features. Included in all processors of the PIC32MX family is a high-performance RISC CPU, which can be programmed in 32-bit and 16-bit modes, and even mixed modes. The PIC32MX MCU contains a high-performance interrupt controller, DMA controller, USB controller, in-circuit debugger, high performance switching matrix for high-speed data accesses to the peripherals, on-chip data RAM memory that holds data and programs. The unique prefetch cache and prefetch buffer for the Flash memory, which hides the latency of the Flash, gives zero Wait state equivalent performance.

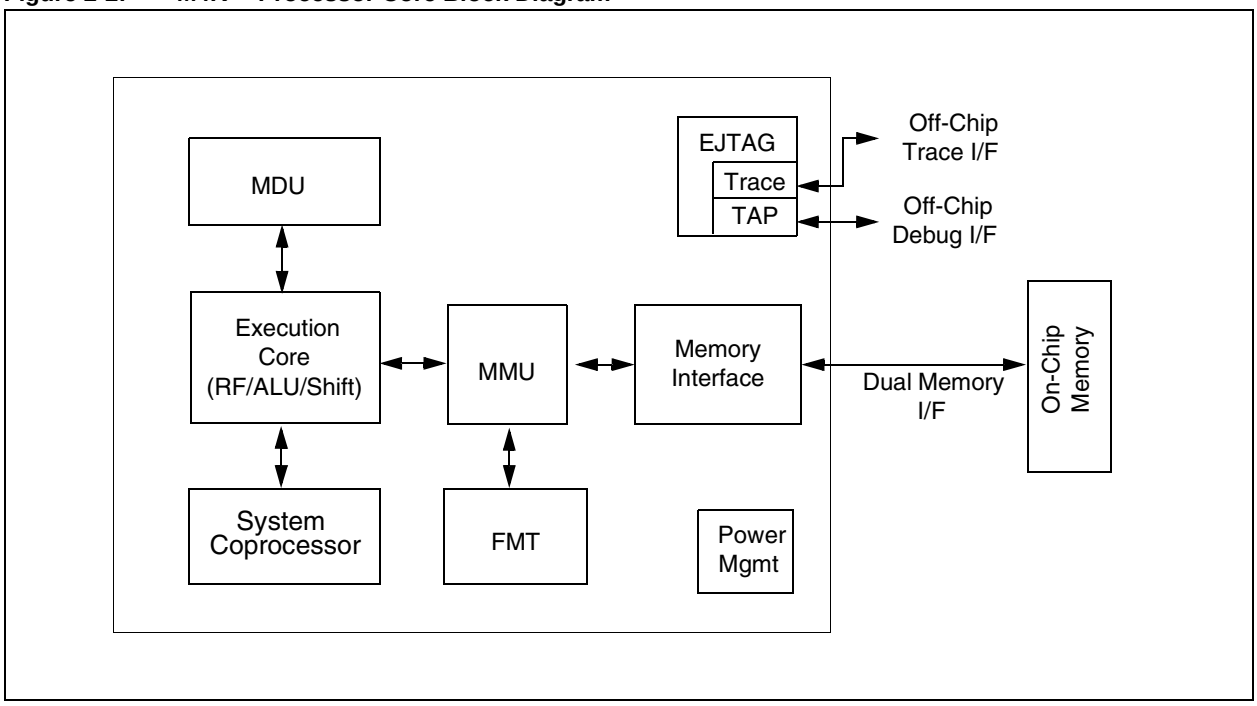
Figure 2-1: PIC32MX MCU Block Diagram



There are two internal buses in the chip to connect all the peripherals. The main peripheral bus connects most of the peripheral units to the bus matrix through a peripheral bridge. There is also a high-speed peripheral bridge that connects the interrupt controller DMA controller, in-circuit debugger, and USB peripherals. The heart of the PIC32MX MCU is the M4K CPU core. The CPU performs operations under program control. Instructions are fetched by the CPU, decoded and executed synchronously. Instructions exist in either the Program Flash memory or Data RAM memory.

The PIC32MX CPU is based on a load/store architecture and performs most operations on a set of internal registers. Specific load and store instructions are used to move data between these internal registers and the outside world.

Figure 2-2: M4K™ Processor Core Block Diagram



2.2.1 Busses

There are two separate busses on the PIC32MX MCU. One bus is responsible for the fetching of instructions to the CPU, and the other is the data path for load and store instructions. Both the instruction, or I-side bus, and the data, or D-side bus, are connected to the bus matrix unit. The bus matrix is a switch that allows multiple accesses to occur concurrently in a system. The bus matrix allows simultaneous accesses between different bus masters that are not attempting accesses to the same target. The bus matrix serializes accesses between different masters to the same target through an arbitration algorithm.

Since the CPU has two different data paths to the bus matrix, the CPU is effectively two different bus masters to the system. When running from Flash memory, load and store operations to SRAM and the internal peripherals will occur in parallel to instruction fetches from Flash memory.

In addition to the CPU, there are three other bus masters in the PIC32MX MCU – the DMA controller, In-Circuit-Debugger Unit, and the USB controller.

2.2.2 Introduction to the Programming Model

The PIC32MX processor has the following features:

- 5-stage pipeline
- 32-bit Address and Data Paths
- DSP-like Multiply-add and multiply-subtract instructions (MADD, MADDU, MSUB, MSUBU)
- Targeted multiply instruction (MUL)
- Zero and One detect instructions (CLZ, CLO)
- Wait instruction (WAIT)
- Conditional move instructions (MOVZ, MOVN)
- Implements MIPS32 Enhanced Architecture (Release 2)
- Vectored interrupts
- Programmable exception vector base
- Atomic interrupt enable/disable
- General Purpose Register (GPR) shadow sets
- Bit field manipulation instructions
- MIPS16e Application Specific Extension improves code density
- Special PC-relative instructions for efficient loading of addresses and constants
- Data type conversion instructions (ZEB, SEB, ZEH, SEH)
- Compact jumps (JRC, JALRC)
- Stack frame set-up and tear down “macro” instructions (SAVE and RESTORE)
- Memory Management Unit with simple Fixed Mapping Translation (FMT)
- Processor to/from Coprocessor register data transfers
- Direct memory to/from Coprocessor register data transfers
- Performance-optimized Multiply-Divide Unit (High-performance build-time option)
- Maximum issue rate of one 32 × 16 multiply per clock
- Maximum issue rate of one 32 × 32 multiply every other clock
- Early-in divide control – 11 to 34 clock latency
- Low-Power mode (triggered by WAIT instruction)
- Software breakpoints via the SDBBP instruction

2.2.3 Core Timer

The PIC32MX architecture includes a core timer that is available to application programs. This timer is implemented in the form of two co-processor registers–the Count register (CP0_COUNT), and the Compare register (CP0_COMPARE). The Count register is incremented every two system clock (SYSCLK) cycles. The incrementing of Count can be optionally suspended during Debug mode. The Compare register is used to cause a timer interrupt if desired. An interrupt is generated when the Compare register matches the Count register. An interrupt is taken only if it is enabled in the interrupt controller.

For more information on the core timer, see **Section 2.12. “CP0 Registers”** and **Section 8. “Interrupts.”**

2.3 PIC32MX CPU DETAILS

2.3.1 Pipeline Stages

The pipeline consists of five stages:

- Instruction (I) Stage
- Execution (E) Stage
- Memory (M) Stage
- Align (A) Stage
- Writeback (W) Stage

2.3.1.1 I Stage – Instruction Fetch

During I stage:

• An instruction is fetched from the instruction SRAM.

• MIPS16e instructions are converted into MIPS32-like instructions.

2.3.1.2 E Stage – Execution

During E stage:

- Operands are fetched from the register file.
- Operands from the M and A stage are bypassed to this stage.
- The Arithmetic Logic Unit (ALU) begins the arithmetic or logical operation for register-to-register instructions.
- The ALU calculates the data virtual address for load and store instructions and the MMU performs the fixed virtual-to-physical address translation.
- The ALU determines whether the branch condition is true and calculates the virtual branch target address for branch instructions.
- Instruction logic selects an instruction address and the MMU performs the fixed virtual-to-physical address translation.
- All multiply divide operations begin in this stage.

2.3.1.3 M Stage – Memory Fetch

During M stage:

- The arithmetic or logic ALU operation completes.
- The data SRAM access is performed for load and store instructions.
- A 16 × 16 or 32 × 16 MUL operation completes in the array and stalls for one clock in the M stage to complete the carry-propagate-add in the M stage.
- A 32 × 32 MUL operation stalls for two clocks in the M stage to complete the second cycle of the array and the carry-propagate-add in the M stage.
- Multiply and divide calculations proceed in the MDU. If the calculation completes before the IU moves the instruction past the M stage, then the MDU holds the result in a temporary register until the IU moves the instructions to the A stage (and it is consequently known that it won't be killed).

2.3.1.4 A Stage – Align

During A stage:

- A separate aligner aligns loaded data with its word boundary.
- A MUL operation makes the result available for writeback. The actual register writeback is performed in the W stage.
- From this stage, load data or a result from the MDU are available in the E stage for bypassing.

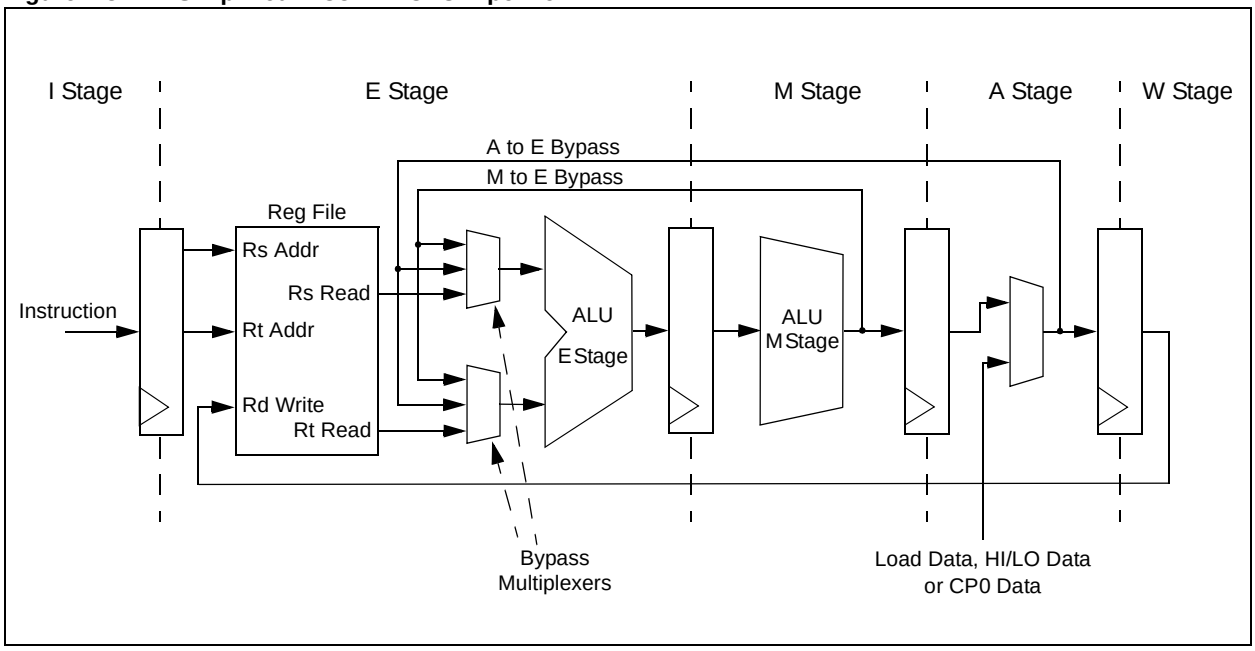
2.3.1.5 W Stage – Writeback

During W stage:

For register-to-register or load instructions, the result is written back to the register file.

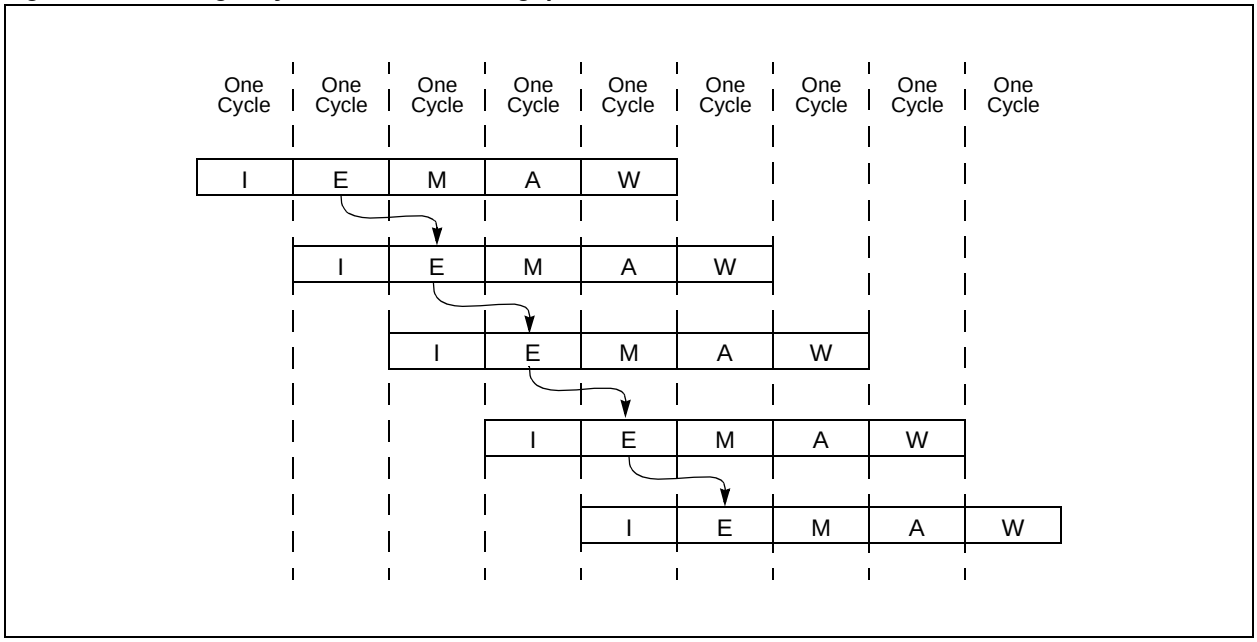
A M4K core implements a “Bypass” mechanism that allows the result of an operation to be sent directly to the instruction that needs it without having to write the result to the register and then read it back.

Figure 2-3: Simplified PIC32MX CPU Pipeline



The results of using instruction pipelining in the PIC32MX core is a fast, single-cycle instruction execution environment.

Figure 2-4: Single-Cycle Execution Throughput



2.3.2 Execution Unit

The PIC32MX Execution Unit is responsible for carrying out the processing of most of the instructions of the MIPS instruction set. The Execution Unit provides single-cycle throughput for most instructions by means of pipelined execution. Pipelined execution, sometimes referred to as “pipelining”, is where complex operations are broken into smaller pieces called stages. Operation stages are executed over multiple clock cycles.

The Execution Unit contains the following features:

- 32-bit adder used for calculating the data address
- Address unit for calculating the next instruction address
- Logic for branch determination and branch target address calculation
- Load aligner
- Bypass multiplexers used to avoid stalls when executing instructions streams where data producing instructions are followed closely by consumers of their results
- Leading Zero/One detect unit for implementing the CLZ and CLO instructions
- Arithmetic Logic Unit (ALU) for performing bitwise logical operations
- Shifter and Store Aligner

2.3.3 MDU

The Multiply/Divide unit performs multiply and divide operations. The MDU consists of a 32 × 16 multiplier, result-accumulation registers (HI and LO), multiply and divide state machines, and all multiplexers and control logic required to perform these functions. The high-performance, pipelined MDU supports execution of a 16 × 16 or 32 × 16 multiply operation every clock cycle; 32 × 32 multiply operations can be issued every other clock cycle. Appropriate interlocks are implemented to stall the issue of back-to-back 32 × 32 multiply operations. Divide operations are implemented with a simple 1 bit per clock iterative algorithm and require 35 clock cycles in worst case to complete. Early-in to the algorithm detects sign extension of the dividend, if it is actual size is 24, 16 or 8 bit. the divider will skip 7, 15, or 23 of the 32 iterations. An attempt to issue a subsequent MDU instruction while a divide is still active causes a pipeline stall until the divide operation is completed.

The M4K implements an additional multiply instruction, MUL, which specifies that lower 32-bits of the multiply result be placed in the register file instead of the HI/LO register pair. By avoiding the explicit move from LO (MFLO) instruction, required when using the LO register, and by supporting multiple destination registers, the throughput of multiply-intensive operations is increased. Two instructions, multiply-add (MADD/MADDU) and multiply-subtract (MSUB/MSUBU), are used to perform the multiply-add and multiply-subtract operations. The MADD instruction multiplies two numbers and then adds the product to the current contents of the HI and LO registers. Similarly, the MSUB instruction multiplies two operands and then subtracts the product from the HI and LO registers. The MADD/MADDU and MSUB/MSUBU operations are commonly used in Digital Signal Processor (DSP) algorithms.

2.3.4 Shadow Register Sets

The PIC32MX processor implements a copy of the General Purpose Registers (GPR) for use by high-priority interrupts. This extra bank of registers is known as a shadow register set. When a high-priority interrupt occurs the processor automatically switches to the shadow register set without software intervention. This reduces overhead in the interrupt handler and reduces effective latency.

The shadow register set is controlled by registers located in the System Coprocessor (CP0) as well as the interrupt controller hardware located outside of the CPU core.

For more information on shadow register sets, see the XREF Interrupt chapter.

2.3.5 Pipeline Interlock Handling

Smooth pipeline flow is interrupted when an instruction in a pipeline stage can not advance due to a data dependency or a similar external condition. Pipeline interruptions are handled entirely in hardware. These dependencies, are referred to as *interlocks*. At each cycle, interlock conditions are checked for all active instructions. An instruction that depends on the result of a previous instruction is an example of an interlock condition.

In general, MIPS processors support two types of hardware interlocks:

- Stalls

  Stalls are resolved by halting the entire pipeline. All instructions currently executing in each pipeline stage are affected by a stall.
- Slips

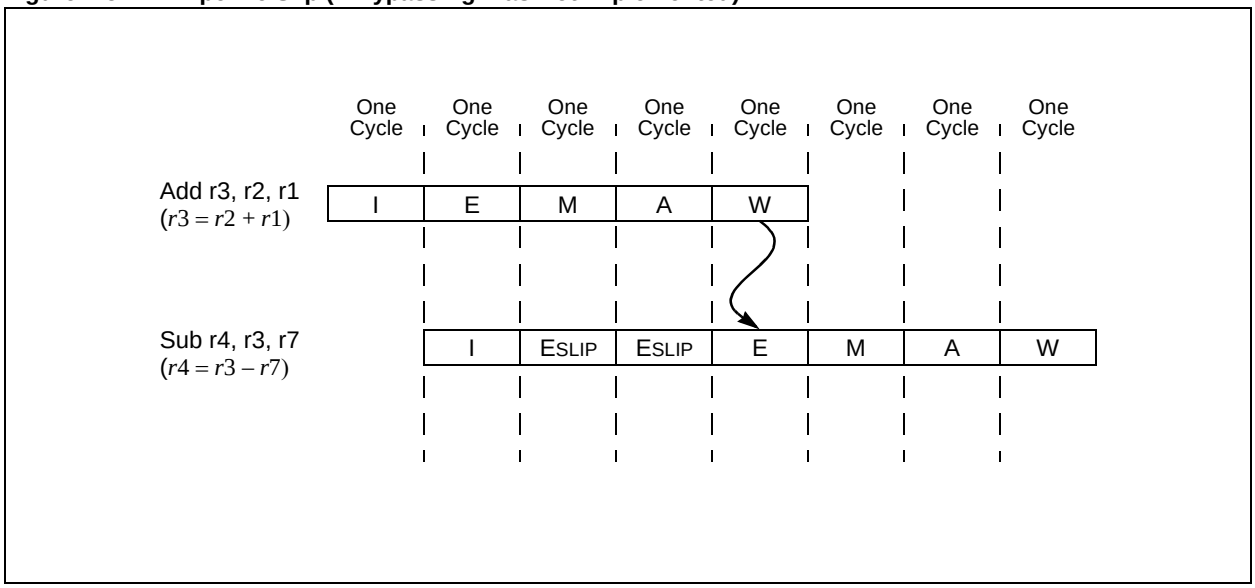
  Slips allow one part of the pipeline to advance while another part of the pipeline is held static.

In the PIC32MX processor core, all interlocks are handled as slips. These slips are minimized by grabbing results from other pipeline stages by using a method called register bypassing, which is described below.

Note:	To illustrate the concept of a pipeline slip, the following example is what would happen if the PIC32MX core did not implement register bypassing.

As shown in Figure 2-5, the sub instruction has a source operand dependency on register r3 with the previous add instruction. The sub instruction slips by two clocks waiting until the result of the add is written back to register r3. This slipping does *not* occur on the PIC32MX family of processors.

Figure 2-5: Pipeline Slip (If Bypassing Was Not Implemented)



2.3.6 Register Bypassing

As mentioned previously, the PIC32MX processor implements a mechanism called register bypassing that helps reduce pipeline slips during execution. When an instruction is in the E stage of the pipeline, the operands must be available for that instruction to continue. If an instruction has a source operand that is computed from another instruction in the execution pipeline, register bypassing allows a shortcut to get the source operands directly from the pipeline. An instruction in the E stage can retrieve a source operand from another instruction that is executing in either the M stage or the A stage of the pipeline. As seen in Figure 2-6, a sequence of three instructions with interdependencies does not slip at all during execution. This example uses both A to E, and M to E register bypassing. Figure 2-7 shows the operation of a load instruction utilizing A to E bypassing. Since the result of load instructions are not available until the A pipeline stage, M to E bypassing is not needed.

The performance benefit of register bypassing is that instruction throughput is increased to the rate of one instruction per clock for ALU operations, even in the presence of register dependencies.

Figure 2-6: IU Pipeline M to E Bypass

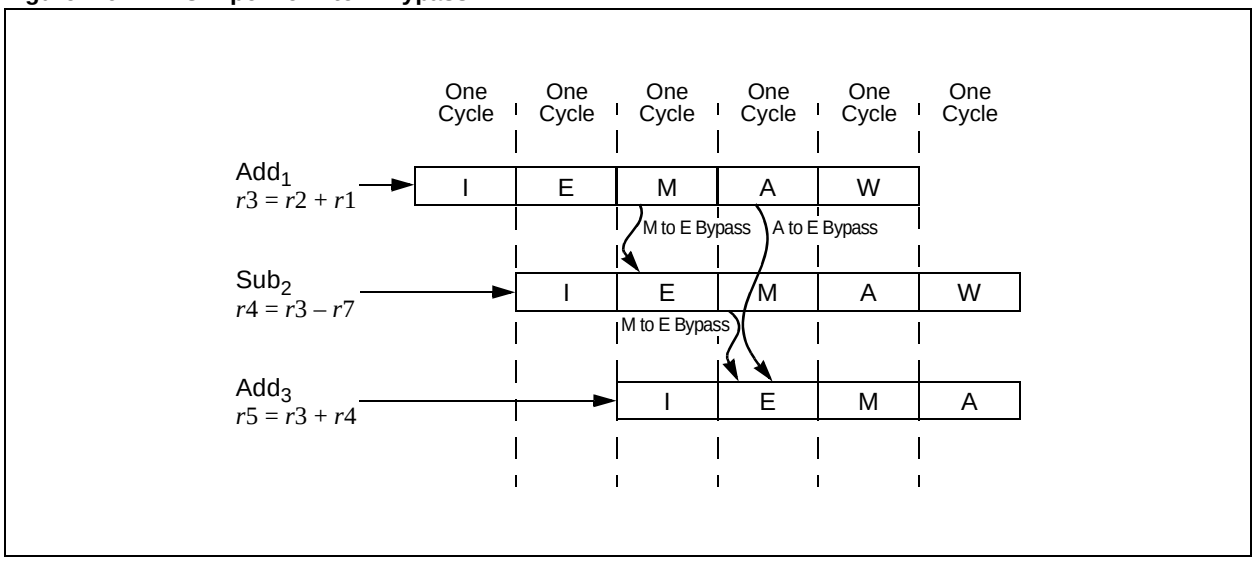
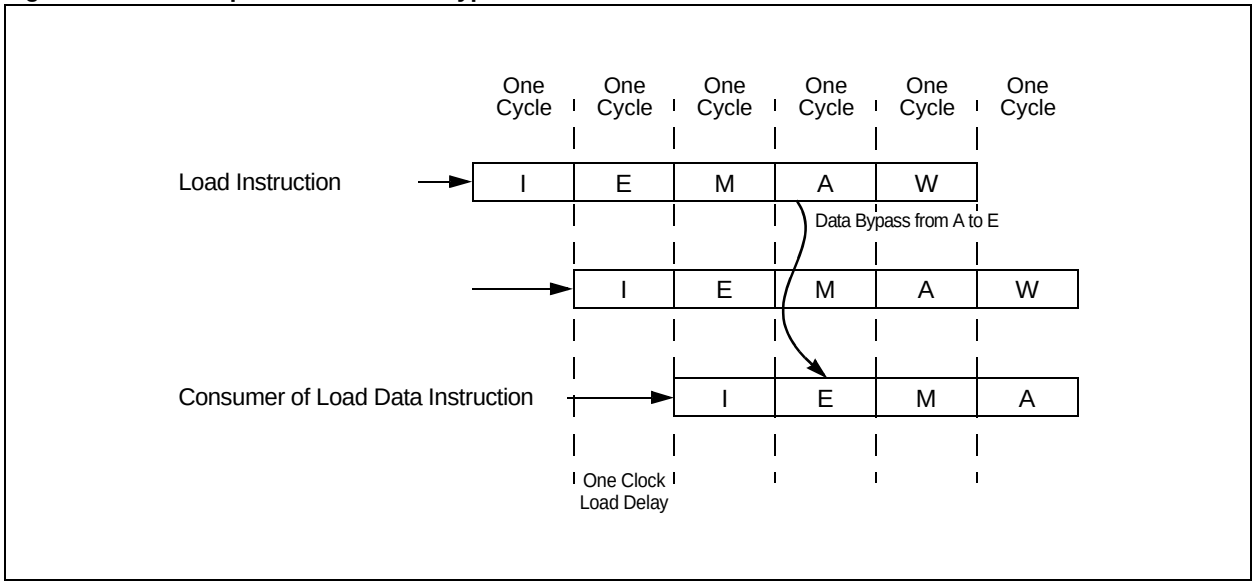


Figure 2-7: IU Pipeline A to E Data Bypass



2.4 SPECIAL CONSIDERATIONS WHEN WRITING TO CP0 REGISTERS

In general, the PIC32MX core ensures that instructions are executed following a fully sequential program model. Each instruction in the program sees the results of the previous instruction. There are some deviations to this model. These deviations are referred to as *hazards*.

In privileged software, there are two different types of hazards:

- Execution Hazards
- Instruction Hazards

2.4.0.1 Execution Hazards

Execution hazards are those created by the execution of one instruction, and seen by the execution of another instruction. Table 2-1 lists execution hazards.

Table 2-1: Execution Hazards

Producer	=	Consumer	Hazard On	Spacing (Instructions)
MTC0	=	Coprocessor instruction execution depends on the new value of $Status_{CU}$	$Status_{CU}$	1
MTC0	=	ERET	EPC DEPC ErrorEPC	1
MTC0	=	ERET	Status	0
MTC0, EI, DI	=	Interrupted Instruction	$Status_{IE}$	1
MTC0	=	Interrupted Instruction	$Cause_{IP}$	3
MTC0	=	RDPGPR WRPGPR	$SRSCtl_{PSS}$	1
MTC0	=	Instruction not seeing a Timer Interrupt	Compare update that clears Timer Interrupt	4
MTC0	=	Instruction affected by change	Any other CP0 register	2

2.4.0.2 Instruction Hazards

Instruction hazards are those created by the execution of one instruction, and seen by the instruction fetch of another instruction. Table 2-2 lists instruction hazards.

Table 2-2: Instruction Hazards

Producer	=	Consumer	Hazard On
MTC0	=	Instruction fetch seeing the new value (including a change to ERL followed by an instruction fetch from the useg segment)	Status

2.5 ARCHITECTURE RELEASE 2 DETAILS

The PIC32MX CPU utilizes Release 2 of the MIPS 32-bit architecture. The PIC32MX CPU implements the following Release 2 features:

- Vectored interrupts using and external-to-core interrupt controller

  Provide the ability to vector interrupts directly to a handler for that interrupt.
- Programmable exception vector base

  Allows the base address of the exception vectors to be moved for exceptions that occur when $Status_{BEV}$ is '0'. This allows any system to place the exception vectors in memory that is appropriate to the system environment.
- Atomic interrupt enable/disable

  Two instructions have been added to atomically enable or disable interrupts, and return the previous value of the *Status* register.
- The ability to disable the *Count* register for highly power-sensitive applications.
- GPR shadow registers

  Provides the addition of GPR shadow registers and the ability to bind these registers to a vectored interrupt or exception.
- Field, Rotate and Shuffle instructions

  Add additional capability in processing bit fields in registers.
- Explicit hazard management

  Provides a set of instructions to explicitly manage hazards, in place of the cycle-based SSNOP method of dealing with hazards.

2.6 SPLIT CPU BUS

The PIC32MX CPU core has two distinct busses to help improve system performance over a single-bus system. This improvement is achieved through parallelism. Load and store operations occur at the same time as instruction fetches. The two busses are known as the I-side bus which is used for feeding instructions into the CPU, and the D-side bus used for data transfers.

The CPU fetches instructions during the I pipeline stage. A fetch is issued to the I-side bus and is handled by the bus matrix unit. Depending on the address, the BMX will do one of the following:

- Forward the fetch request to the Prefetch Cache Unit
- Forward the fetch request to the DRM unit or
- Cause an exception

Instruction fetches always use the I-side bus independent of the addresses being fetched. The BMX decides what action to perform for each fetch request based on the address and the values in the BMX registers. (See BMX chapter).

The D-side bus processes all load and store operations executed by the CPU. When a load or store instruction is executed the request is routed to the BMX by the D-side bus. This operation occurs during the M pipeline stage and is routed to one of several targets devices:

- Data Ram
- Prefetch Cache/Flash Memory
- Fast Peripheral Bus (Interrupt controller, DMA, Debug unit, USB, GPIO Ports)
- General Peripheral Bus (UART, SPI, Flash Controller, EPMP/EPSP, TRCC Timers, Input Capture, PWM/Output Compare, ADC, Dual Compare, I^2C, Clock SIB, and Reset SIB)

2.7 INTERNAL SYSTEM BUSSES

The PIC32MX processor internal busses connect the peripherals to the bus matrix unit. The bus matrix routes bus accesses from 5 different initiators to a set of targets utilizing several data paths throughout the chip to help eliminate performance bottlenecks.

Some of the paths that the bus matrix uses serve a dedicated purpose, while others are shared between several targets.

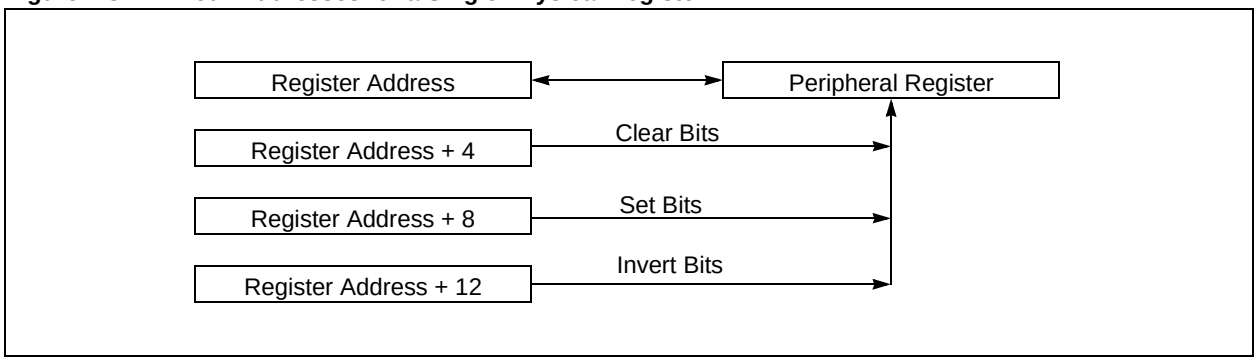
The data RAM and Flash memory read paths are dedicated paths, allowing low-latency access to the memory resources without being delayed by peripheral bus activity. The high-bandwidth peripherals are placed on a high-speed bus. These include the Interrupt controller, debug unit, DMA engine, and the USB host/peripheral unit.

Peripherals that do not require high-bandwidth are located on a separate peripheral bus to save power.

2.8 SET/CLEAR/INVERT

To provide single-cycle bit operations on peripherals, the registers in the peripheral units can be accessed in three different ways depending on peripheral addresses. Each register has four different addresses. Although the four different addresses appear as different registers, they are really just four different methods to address the same physical register.

Figure 2-8: Four Addresses for a Single Physical Register



The base register address provides normal Read/Write access, the other three provide special write-only functions.

1. Normal access
2. Set bit atomic RMW access
3. Clear bit atomic RMW access
4. Invert bit atomic RMW access

Peripheral reads must occur from the base address of each peripheral register. Reading from a set/clear/invert address has an undefined meaning, and may be different for each peripheral.

Writing to the base address writes an entire value to the peripheral register. All bits are written. For example, assume a register contains 0xaaaa5555 before a write of 0x000000ff. After the write, the register will contain 0x000000ff (assuming that all bits are R/W bits).

Writing to the Set address for any peripheral register causes only the bits written as '1's to be set in the destination register. For example, assume that a register contains 0xaaaa5555 before a write of 0x000000ff to the set register address. After the write to the Set register address, the value of the peripheral register will contain 0xaaaa55ff.

Writing to the Clear address for any peripheral register causes only the bits written as '1's to be cleared to '0's in the destination register. For example, assume that a register contains 0xaaaa5555 before a write of 0x000000ff to the Clear register address. After the write to the Clear register address, the value of the peripheral register will contain 0xaaaa5500.

Writing to the Invert address for any peripheral register causes only the bits written as ‘1’s to be inverted, or toggled, in the destination register. For example, assume that a register contains 0xaaaa5555 before a write of 0x000000ff to the invert register address. After the write to the Invert register, the value of the peripheral register will contain 0xaaaa55aa.

2.9 ALU STATUS BITS

Unlike most other PIC® microcontrollers, the PIC32MX Processor does not use STATUS register flags. Condition flags are used on many processors to help perform decision making operations during program execution. Flags are set based on the results of comparison operations or some arithmetic operations. Conditional branch instructions on these machines then make decisions based on the values of the single set of condition codes.

The PIC32MX processor, instead, uses instructions that perform a comparison and stores a flag or value into a General Purpose Register. A conditional branch is then executed with this general purpose register used as an operand.

2.10 INTERRUPT AND EXCEPTION MECHANISM

The PIC32MX family of processors implement an efficient and flexible interrupt and exception handling mechanism. Interrupts and exceptions both behave similarly in that the current instruction flow is changed temporarily to execute special procedures to handle an interrupt or exception. The difference between the two is that interrupts are usually a result of normal operation, and exceptions are a result of error conditions such as bus errors.

When an interrupt or exception occurs, the processor does the following:

1. The PC of the next instruction to execute after the handler returns is saved into a coprocessor register.
2. Cause register is updated to reflect the reason for exception or interrupt
3. Status EXL or ERL is set to cause Kernel mode execution
4. Handler PC is calculated from EBASE and SPACING values
5. Processor starts execution from new PC

This is a simplified overview of the interrupt and exception mechanism. See **Section 8. “Interrupts”** for more information regarding interrupt and exception handling.

2.11 PROGRAMMING MODEL

The PIC32MX family of processors is designed to be used with a high-level language such as the C programming language. It supports several data types and uses simple but flexible addressing modes needed for a high-level language. There are 32 General Purpose Registers and two special registers for multiplying and dividing.

There are three different formats for the machine language instructions on the PIC32MX processor:

- immediate or I-type CPU instructions
- jump or J-type CPU instructions and
- registered or R-type CPU instructions

Most operations are performed in registers. The register type CPU instructions have three operands; two source operands and a destination operand.

Having three operands and a large register set allows assembly language programmers and compilers to use the CPU resources efficiently. This creates faster and smaller programs by allowing intermediate results to stay in registers rather than constantly moving data to and from memory.

The immediate format instructions have an immediate operand, a source operand and a destination operand. The jump instructions have a 26-bit relative instruction offset field that is used to calculate the jump destination.

2.11.1 CPU Instruction Formats

A CPU instruction is a single 32-bit aligned word. The CPU instruction formats are shown below:

- Immediate (see Figure 2-9)
- Jump (see Figure 2-10)
- Register (see Figure 2-11)

Table 2-3 describes the fields used in these instructions.

Table 2-3: CPU Instruction Format Fields

Field	Description
opcode	6-bit primary operation code
rd	5-bit specifier for the destination register
rs	5-bit specifier for the source register
rt	5-bit specifier for the target (source/destination) register or used to specify functions within the primary opcode REGIMM
immediate	16-bit signed immediate used for logical operands, arithmetic signed operands, load/store address byte offsets, and PC-relative branch signed instruction displacement
instr_index	26-bit index shifted left two bits to supply the low-order 28 bits of the jump target address
sa	5-bit shift amount
function	6-bit function field used to specify functions within the primary opcode SPECIAL

Figure 2-9: Immediate (I-Type) CPU Instruction Format

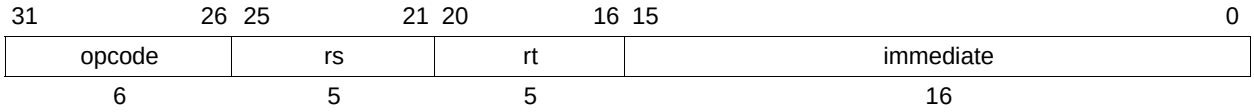


Figure 2-10: Jump (J-Type) CPU Instruction Format

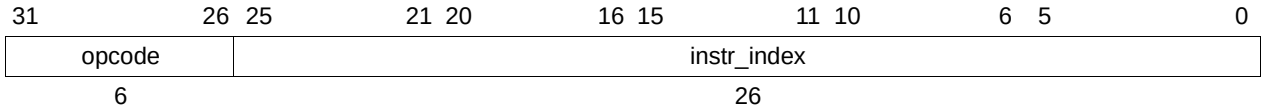
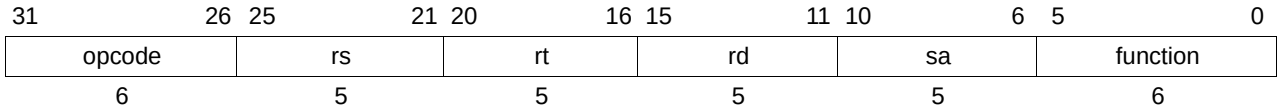


Figure 2-11: Register (R-Type) CPU Instruction Format



2.11.2 CPU Registers

The PIC32MX architecture defines the following CPU registers:

- 32 32-bit General Purpose Registers (GPRs)
- 2 special purpose registers to hold the results of integer multiply, divide, and multiply-accumulate operations (HI and LO)
- a special purpose program counter (PC), which is affected only indirectly by certain instructions – it is not an architecturally visible register.

2.11.2.1 CPU General Purpose Registers

Two of the CPU General Purpose Registers have assigned functions:

- r0

  r0 is hard-wired to a value of '0', and can be used as the target register for any instruction the result of which will be discarded. r0 can also be used as a source when a '0' value is needed.

- r31

  r31 is the destination register used by JAL, BLTZAL, BLTZALL, BGEZAL, and BGEZALL, without being explicitly specified in the instruction word. Otherwise r31 is used as a normal register.

The remaining registers are available for general purpose use.

2.11.2.2 Register Conventions

Although most of the registers in the PIC32MX architecture are designated as General Purpose Registers, there are some recommended uses of the registers for correct software operation with high-level languages such as the Microchip C compiler.

Table 2-4: Register Conventions

CPU Register	Symbolic Register	Usage
r0	zero	Always 0(1)
r1	at	Assembler Temporary
r2 - r3	v0-v1	Function Return Values
r4 - r7	a0-a3	Function Arguments
r8 - r15	t0-t7	Temporary – Caller does not need to preserve contents
r16 - r23	s0-s7	Saved Temporary – Caller must preserve contents
r24 - r25	t8 - t9	Temporary – Caller does not need to preserve contents
r26 - r27	k0 - k1	Kernel temporary – Used for interrupt and exception handling
r28	gp	Global Pointer – Used for fast-access common data
r29	sp	Stack Pointer – Software stack
r30	s8 or fp	Saved Temporary – Caller must preserve contents *OR* Frame Pointer – Pointer to procedure frame on stack
r31	ra	Return Address(1)

Note 1: Hardware enforced, not just convention.

2.11.2.3 CPU Special Purpose Registers

The CPU contains three special purpose registers:

- PC – Program Counter register
- HI – Multiply and Divide register higher result
- LO – Multiply and Divide register lower result
  - During a multiply operation, the HI and LO registers store the product of integer multiply.
  - During a multiply-add or multiply-subtract operation, the HI and LO registers store the result of the integer multiply-add or multiply-subtract.
  - During a division, the HI and LO registers store the quotient (in LO) and remainder (in HI) of integer divide.
  - During a multiply-accumulate, the HI and LO registers store the accumulated result of the operation.

Figure 2-12 shows the layout of the CPU registers.

Table 2-5: CPU Register

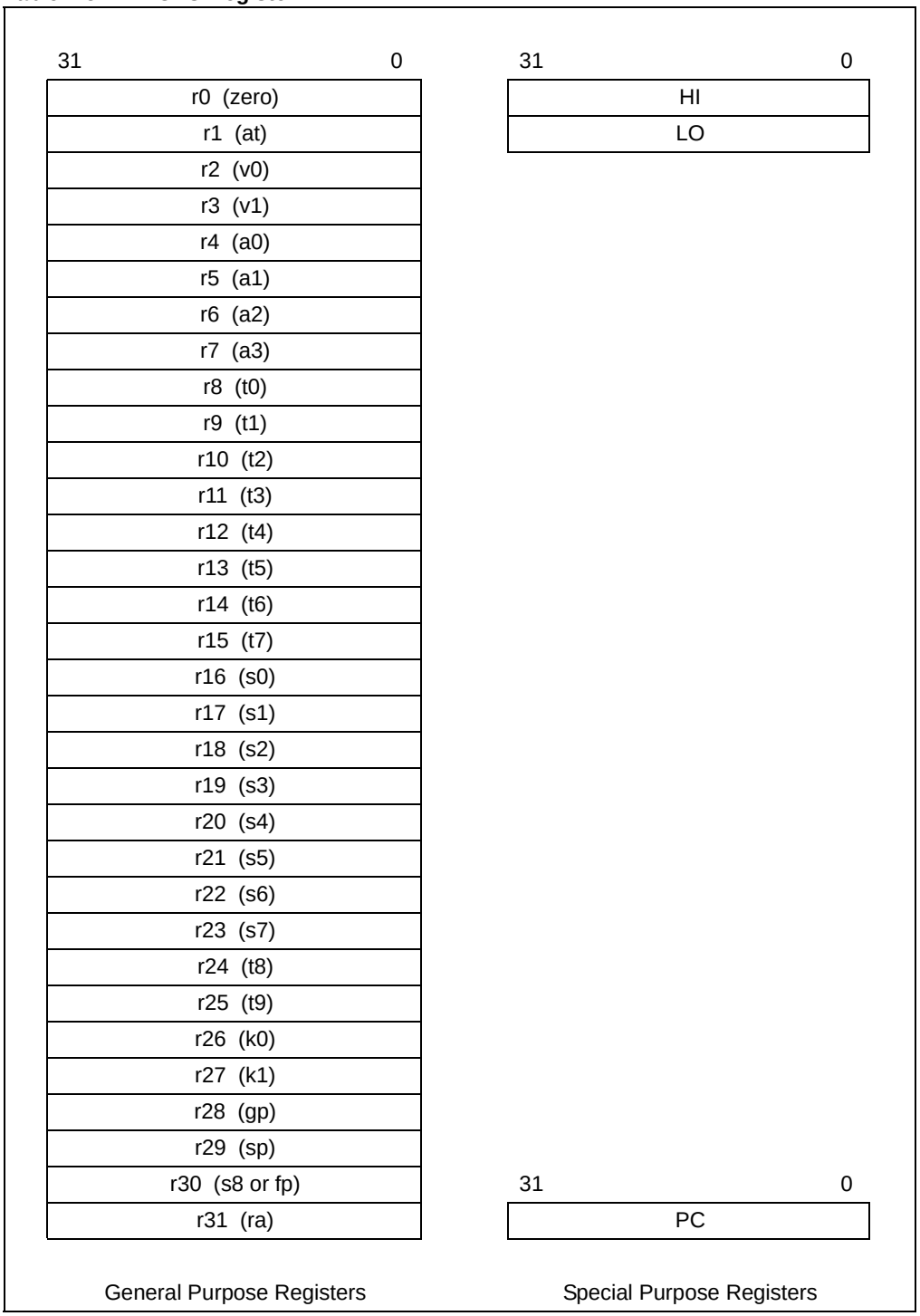


Table 2-6: MIPS16e Register Usage

MIPS16e Register Encoding	32-Bit MIPS Register Encoding	Symbolic Name	Description
0	16	s0	General Purpose Register
1	17	s1	General Purpose Register
2	2	v0	General Purpose Register
3	3	v1	General Purpose Register
4	4	a0	General Purpose Register
5	5	a1	General Purpose Register
6	6	a2	General Purpose Register
7	7	a3	General Purpose Register
N/A	24	t8	MIPS16e Condition Code register; implicitly referenced by the BTEQZ, BTNEZ, CMP, CMPI, SLT, SLTU, SLTI, and SLTIU instructions
N/A	29	sp	Stack Pointer register
N/A	31	ra	Return Address register

Table 2-7: MIPS16e Special Registers

Symbolic Name	Purpose
PC	Program counter. PC-relative Add and Load instructions can access this register as an operand.
HI	Contains high-order word of multiply or divide result.
LO	Contains low-order word of multiply or divide result.

2.11.3 How to implement a stack/MIPS calling conventions

The PIC32MX CPU does not have hardware stacks. Instead, the processor relies on software to provide this functionality. Since the hardware does not perform stack operations itself, a convention must exist for all software within a system to use the same mechanism. For example, a stack can grow either toward lower address, or grow toward higher addresses. If one piece of software assumes that the stack grows toward lower address, and calls a routine that assumes that the stack grows toward higher address, the stack would become corrupted.

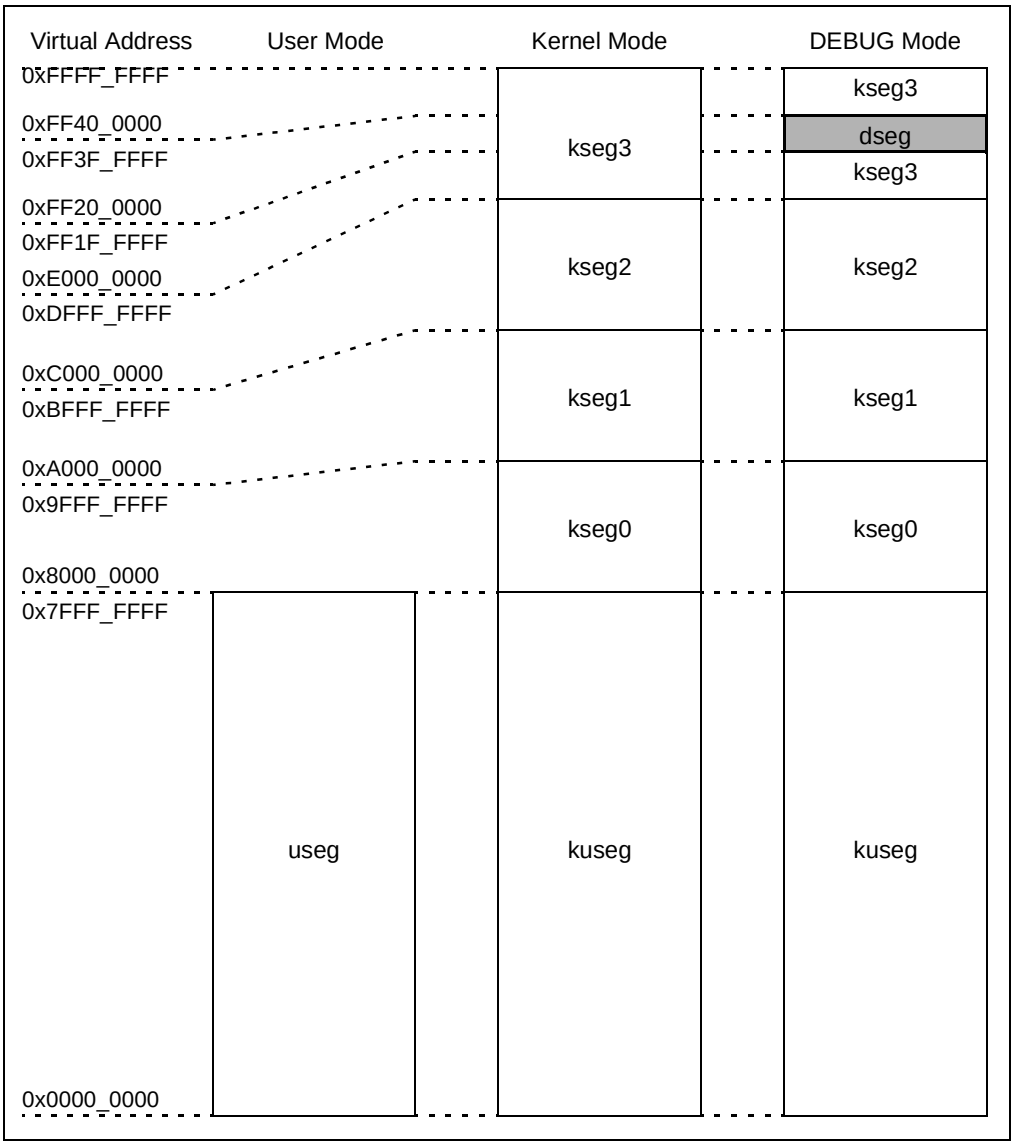
Using a system-wide calling convention prevents this problem from occurring. The Microchip C compiler assumes the stack grows toward lower addresses.

2.11.4 Processor Modes

There are two operational modes and one special mode of execution in the PIC32MX family CPUs; User mode, Kernel mode and DEBUG mode. The processor starts execution in Kernel mode, and if desired, can stay in Kernel mode for normal operation. User mode is an optional mode that allows a system designer to partition code between privileged and un-privileged software. DEBUG mode is normally only used by a debugger or monitor.

One of the main differences between the modes of operation is the memory addresses that software is allowed to access. Peripherals are not accessible in User mode. Figure 2-12 shows the different memory maps for each mode. For more information on the processor's memory map, see **Section 3. "Memory Organization"**.

Figure 2-12: CPU Modes



2.11.4.1 Kernel Mode

In order to access many of the hardware resources, the processor must be operating in Kernel mode. Kernel mode gives software access to the entire address space of the processor as well as access to privileged instructions.

The processor operates in Kernel mode when the DM bit in the DEBUG register is '0' and the STATUS register contains one, or more, of the following values:

UM = 0 ERL = 1 EXL = 1

When a non-debug exception is detected, EXL or ERL will be set and the processor will enter Kernel mode. At the end of the exception handler routine, an Exception Return (ERET) instruction is generally executed. The ERET instruction jumps to the Exception PC (EPC or ErrorPC depending on the exception), clears ERL, and clears EXL if ERL= 0.

If UM = 1 the processor will return to User mode after returning from the exception when ERL and EXL are cleared back to '0'.

2.11.4.2 User Mode

When executing in User mode, software is restricted to use a subset of the processor's resources. In many cases it is desirable to keep application-level code running in User mode where if an error occurs it can be contained and not be allowed to affect the Kernel mode code.

Applications can access Kernel mode functions through controlled interfaces such as the SYSCALL mechanism.

As seen in Figure 2-12, User mode software has access to the USEG memory area.

To operate in User mode, the STATUS register must contain each the following bit values:

UM = 1 EXL = 0 ERL = 0

2.11.4.3 DEBUG Mode

DEBUG mode is a special mode of the processor normally only used by debuggers and system monitors. DEBUG mode is entered through a debug exception and has access to all the Kernel mode resources as well as special hardware resources used to debug applications.

The processor is in DEBUG mode when the DM bit in the DEBUG register is '1'.

DEBUG mode is normally exited by executing a DERET instruction from the debug handler.

2.12 CP0 REGISTERS

The PIC32MX uses a special register interface to communicate status and control information between system software and the CPU. This interface is called Coprocessor 0. The features of the CPU that are visible through Coprocessor 0 are core timer, interrupt and exception control, virtual memory configuration, shadow register set control, processor identification, and debugger control. System software accesses the registers in CP0 using coprocessor instructions such as MFC0 and MTC0. Table 2-8 describes the CP0 registers found on the PIC32MX MCU.

Table 2-8: CP0 Registers

Register Number	Register Name	Function
0-6	Reserved	Reserved in the PIC32MX core
7	HWREna	Enables access via the RDHWR instruction to selected hardware registers in Non-privileged mode
8	BadVAddr	Reports the address for the most recent address-related exception
9	Count	Processor cycle count
10	Reserved	Reserved in the PIC32MX core
11	Compare	Timer interrupt control
12	Status/ IntCtl/ SRSCtl/ SRSMap	Processor status and control; interrupt control; and shadow set control
13	Cause	Cause of last exception
14	EPC	Program counter at last exception
15	PRId/ EBASE/	Processor identification and revision; exception base address
16	Config/ Config1/ Config2/ Config3	Configuration registers
17-22	Reserved	Reserved in the PIC32MX core
23	Debug/ Debug2/	Debug control/exception status and EJTAG trace control
24	DEPC	Program counter at last debug exception
25-29	Reserved	Reserved in the PIC32MX core
30	ErrorEPC	Program counter at last error
31	DeSAVE	Debug handler scratchpad register

2.12.1 HWREna Register (CP0 Register 7, Select 0)

HWREna contains a bit mask that determines which hardware registers are accessible via the RDHWR instruction.

Register 2-1: HWREna: Hardware Accessibility Register; CP0 Register 7, Select 0

r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
—	—	—	—	—	—	—	—
bit 31							bit 24

r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
—	—	—	—	—	—	—	—
bit 23							bit 16

r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
—	—	—	—	—	—	—	—
bit 15							bit 8

r-0	r-0	r-0	r-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	MASK<3:0>			
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-4 **Reserved:** Write ‘0’; returns ‘0’ on read

bit 3-0 **MASK<3:0>:** Bit Mask bits

1 = Access is enabled to corresponding hardware register
0 = Access is disabled
Each bit in this field enables access by the RDHWR instruction to a particular hardware register (which may not be an actual register). See the RDHWR instruction for a list of valid hardware registers.

2.12.2 BadVAddr Register (CP0 Register 8, Select 0)

BadVAddr is a read-only register that captures the most recent virtual address that caused an address error exception. Address errors are caused by executing load, store, or fetch operations from unaligned addresses, and also by trying to access Kernel mode addresses from User mode.

BadVAddr does not capture address information for bus errors, because they are not addressing errors.

Register 2-2: BadVAddr: Bad Virtual Address Register; CP0 Register 8, Select 0

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
BadVAddr<31:24>							
bit 31							bit 24

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
BadVAddr<23:16>							
bit 23							bit 16

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
BadVAddr<15:8>							
bit 15							bit 8

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
BadVAddr<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **BadVAddr<31:0>:** Bad Virtual Address bits
Captures the virtual address that caused the most recent address error exception.

2.12.3 COUNT Register (CP0 Register 9, Select 0)

COUNT acts as a timer, incrementing at a constant rate, whether or not an instruction is executed, retired, or any forward progress is made through the pipeline. The counter increments every other clock, if the DC bit in the CAUSE register is '0'.

COUNT can be written for functional or diagnostic purposes, including at Reset or to synchronize processors.

By writing the CountDM bit in DEBUG register, it is possible to control whether COUNT continues to increment while the processor is in DEBUG mode.

Register 2-3: COUNT: Interval Counter Register; CP0 Register 9, Select 0

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
COUNT<31:24>							
bit 31							bit 24

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
COUNT<23:16>							
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
COUNT<15:8>							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
COUNT<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: ('0', '1', x = Unknown)		

bit 31-0 **COUNT<31:0>:** Interval Counter bits
This value is incremented every other clock cycle.

2.12.4 COMPARE Register (CP0 Register 11, Select 0)

COMPARE acts in conjunction with COUNT to implement a timer and timer interrupt function. COMPARE maintains a stable value and does not change on its own.

When the value of COUNT equals the value of COMPARE, the CPU asserts an interrupt signal to the system interrupt controller. This signal will remain asserted until COMPARE is written.

Register 2-4: COMPARE: Interval Count Compare Register; CP0 Register 11, Select 0

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
COMPARE<31:24>							
bit 31							bit 24

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
COMPARE<23:16>							
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
COMPARE<15:8>							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
COMPARE<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **COMPARE<31:0>:** Interval Count Compare Value bits

2.12.5 STATUS Register (CP0 Register 12, Select 0)

STATUS is a read/write register that contains the operating mode, interrupt enabling, and the diagnostic states of the processor. Fields of this register combine to create operating modes for the processor.

2.12.5.0.1 **Interrupt Enable**

Interrupts are enabled when all of the following conditions are true:

IE = 1 EXL = 0 ERL = 0 DM = 0

If these conditions are met, then the settings of the IPL bits enable the interrupts.

2.12.5.0.2 **Operating Modes**

If the DM bit in the Debug register is '1', then the processor is in DEBUG mode; otherwise, the processor is in either Kernel or User mode.

The following CPU STATUS register bit settings determine User or Kernel mode:

Table 2-9: CPU Status Bits that Determine Processor Mode

User Mode (requires *all* of the following bits and values)	UM = 1	EXL = 0	ERL = 0
Kernal Mode (requires *one* or more of the following bit values)	UM = 0	EXL = 1	ERL = 1

Note:	The STATUS register CU bits <31:28> control coprocessor accessibility. If any coprocessor is unusable, then an instruction that accesses it generates an exception.

Register 2-5: STATUS: Status Register; CP0 Register 12, Select 0

R-0	R-0	R-0	R/W-x	R/W-0(1)	r-x	R/W-x	r-0
CU3	CU2	CU1	CU0	RP	FR	RE	—
bit 31							bit 24

r-0	R/W-1	r-0	R/W-0	R/W-0	r-0	r-0	r-0
—	BEV	Reserved	SR	NMI	—	—	—
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
IPL<15:10>						R<9:8>	
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	UM	—	ERL	EXL	IE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31 **CU3:** Coprocessor 3 Usable bit
Controls access to Coprocessor 3
COP3 is not supported. This bit cannot be written and will read as ‘0’

bit 30 **CU2:** Coprocessor 2 Usable bit
Controls access to Coprocessor 2.
COP2 is not supported. This bit cannot be written and will read as ‘0’

bit 29 **CU1:** Coprocessor 1 Usable bit
Controls access to Coprocessor 1
COP1 is not supported. This bit cannot be written and will read as ‘0’

bit 28 **CU0:** Coprocessor 0 Usable bit
Controls access to Coprocessor 0
0 = access not allowed
1 = access allowed
Coprocessor 0 is always usable when the processor is running in Kernel mode, independent of the state of the CU0 bit.

bit 27 **RP:** Reduced Powerbit
Enables reduced power mode

bit 26 **FR:** FR bit
Reserved on PIC32MX processors

bit 25 **RE:** Used to enable reverse-endian memory references while the processor is running in User mode
0 = User mode uses configured endianness
1 = User mode uses reversed endianness
Neither DEBUG mode nor Kernel mode nor Supervisor mode references are affected by the state of this bit.

bit 24:23 **R<24:23>:** Reserved. Ignored on write and read as ‘0’.

Register 2-5: STATUS: Status Register; CP0 Register 12, Select 0 (Continued)

bit 22	**BEV:** Control bit. Controls the location of exception vectors. 0 = Normal 1 = Bootstrap
bit 21	**Reserved**
bit 20	**SR:** Soft Reset bit Indicates that the entry through the Reset exception vector was due to a Soft Reset. 0 = Not Soft Reset (NMI or Reset) 1 = Soft Reset Software can only write a '0' to this bit to clear it and cannot force a 0-1 transition.
bit 19	**NMI:** Soft Reset bit Indicates that the entry through the reset exception vector was due to an NMI. 0 = Not NMI (Soft Reset or Reset) 1 = NMI Software can only write a '0' to this bit to clear it and cannot force a 0-1 transition.
bit 18	**R:** Reserved. ignored on write and read as '0'.
bit 17	**R:** Reserved. ignored on write and read as '0'.
bit 16	**R:** Reserved. ignored on write and read as '0'.
bit 15-10	**IPL<15:10>:** Interrupt Priority Level bits This field is the encoded (0..63) value of the current IPL. An interrupt will be signaled only if the requested IPL is higher than this value
bit 9-8	**R<9:8>:** Reserved These bits are writable, but have no effect on the interrupt system.
bit 7-5	**R<7:5>:** Reserved. Ignored on write and read as '0'
bit 4	**UM:** This bit denotes the base operating mode of the processor. On the encoding of this bit is: 0 = Base mode in Kernal mode 1 = Base mode is User mode **Note:** The processor can also be in Kernel mode if ERL or EXL is set, regardless of the state of the UM bit.
bit 3	**R:** Reserved. Ignored on write and read as '0'
bit 2	**ERL:** Error Level bit Set by the processor when a Reset, Soft Reset, NMI or Cache Error exception are taken. 0 = Normal level 1 = Error level When ERL is set: - Processor is running in Kernel mode - Interrupts are disabled - ERET instruction will use the return address held in ErrorEPC instead of EPC - Lower 2^{29} bytes of kuseg are treated as an unmapped and uncached region. This allows main memory to be accessed in the presence of cache errors. The operation of the processor is *undefined* if the ERL bit is set while the processor is executing instructions from kuseg.
bit 1	**EXL:** Exception Level bit Set by the processor when any exception other than Reset, Soft Reset, or NMI exceptions is taken. 0 = Normal level 1 = Exception level <u>When EXL is set:</u> - Processor is running in Kernel Mode - Interrupts are disabled EPC, CauseBD and SRSCtl will not be updated if another exception is taken.

Register 2-5: STATUS: Status Register; CP0 Register 12, Select 0 (Continued)

bit 0	**IE:** Interrupt Enable bit Acts as the master enable for software and hardware interrupts: 0 = Interrupts are disabled 1 = Interrupts are enabled This bit may be modified separately via the DI and EI instructions

2.12.6 Intctl: Interrupt Control Register (CP0 Register 12, Select 1)

The Intctl register controls the vector spacing of the PIC32MX architecture.

Register 2-6: Intctl: Interrupt Control Register; CP0 Register 12, Select 1

R-0	R-0	R-0	R-0	R-0	R-0	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	VS<9:8>	
bit 15							bit 8

R/W-0	R/W-0	R/W-0	r-x	r-x	r-x	r-x	r-x
VS<7:5>			—	—	—	—	—
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-29 **R:** Reserved

bit 28-26 **R:** Reserved

bit 25-10 **Reserved:** Write ‘0’; ignore read
Must be written as ‘0’; returns ‘0’ on read.

bit 9-5 **VS<9:5>:** Vector Spacing bits
This field specifies the spacing between each interrupt vector.

Encoding	Spacing Between Vectors (hex)	Spacing Between Vectors (decimal)
16#00	16#000 0x	0
16#01	16#020	32
16#02	16#040	64
16#04	16#080	128
16#08	16#100	256
16#10	16#200	512

All other values are reserved. The operation of the processor is *undefined* if a reserved value is written to this field.

bit 4-0 **Unimplemented:** Read as ‘0’
Must be written as ‘0’; returns ‘0’ on read.

2.12.7 SRSCtl Register (CP0 Register 12, Select 2)

The SRSCtl register controls the operation of GPR shadow sets in the processor.

Table 2-10: Sources for New $SRSCtl_{CSS}$ on an Exception or Interrupt

Exception Type	Condition	$SRSCtl_{CSS}$ Source	Comment
Exception	All	$SRSCtl_{ESS}$	
Non-Vectored Interrupt	$Cause_{IV}$ = 0	$SRSCtl_{ESS}$	Treat as exception
Vectored EIC Interrupt	$Cause_{IV}$ = 1 and $Config3_{VEIC}$ = 1	$SRSCtl_{EICSS}$	Source is external interrupt controller.

Register 2-7: SRSCtl: Register; CP0 Register 12, Select 2

r-x	r-x	R-0	R-0	R-0	R-1	r-x	r-x
—	—	HSS<29:26>				—	—
bit 31							bit 24

r-x	r-x	R-x	R-x	R-x	R-x	r-x	r-x
—	—	EICSS<21:18>				—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	r-x	r-x	R/W-0	R/W-0
ESS<15:12>				—	—	PSS<9:8>	
bit 15							bit 8

R/W-0	R/W-0	r-0	r-0	R-0	R-0	R-0	R-0
PSS<7:6>		0<5:4>		CSS<3:0>			
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-30 **Reserved:** Write ‘0’; ignore read
Must be written as zeros; returns ‘0’ on read.

bit 29-26 **HSS<29:26>:** High Shadow Set bit
This field contains the highest shadow set number that is implemented by this processor. A value of ‘0’ in this field indicates that only the normal GPRs are implemented.
Possible values of this field for the PIC32MX processor are:

0 = One shadow set (normal GPR set) is present
1 = Two shadow sets are present
3 = Four shadow sets are present
2, 3-15 = Reserved

The value in this field also represents the highest value that can be written to the ESS, EICSS, PSS, and CSS fields of this register, or to any of the fields of the SRSMAP register. The operation of the processor is *undefined* if a value larger than the one in this field is written to any of these other fields.

bit 25-22 **Reserved:** Write ‘0’; ignore read
Must be written as ‘0’; returns ‘0’ on read.

Register 2-7: SRSCtl: Register; CP0 Register 12, Select 2 (Continued)

bit 21-18 **EICSS<21:18>:** External Interrupt Controller Shadow Set bits

EIC Interrupt mode shadow set. This field is loaded from the external interrupt controller for each interrupt request and is used in place of the SRSMAP register to select the current shadow set for the interrupt.

bit 17-16 **Reserved:** Write '0'; ignore read

Must be written as '0'; returns '0' on read.

bit 15-12 **ESS<15:12>:** Exception Shadow Set bits

This field specifies the shadow set to use on entry to Kernel mode caused by any exception other than a vectored interrupt.
The operation of the processor is *undefined* if software writes a value into this field that is greater than the value in the HSS field.

bit 11-10 **Reserved:** Write '0'; ignore read

Must be written as '0'; returns '0' on read.

bit 9-6 **PSS<9:6>:** Previous Shadow Set bits

Since GPR shadow registers are implemented, this field is copied from the CSS field when an exception or interrupt occurs. An ERET instruction copies this value back into the CSS field if $Status_{BEV}$ = 0. This field is not updated on any exception which sets $Status_{ERL}$ to 1 (i.e., Reset, Soft Reset, NMI, cache error), an entry into EJTAG DEBUG mode, or any exception or interrupt that occurs with $Status_{EXL}$ = 1, or $Status_{BEV}$ = 1. This field is not updated on an exception that occurs while $Status_{ERL}$ = 1.
The operation of the processor is *undefined* if software writes a value into this field that is greater than the value in the HSS field.

bit 5-4 **Reserved:** Write '0'; ignore read

Must be written as '0'; returns '0' on read.

3-0 **CSS<3:0>:** Current Shadow Set bits

Since GPR shadow registers are implemented, this field is the number of the current GPR set. This field is updated with a new value on any interrupt or exception, and restored from the PSS field on an ERET. Table 2-10 describes the various sources from which the CSS field is updated on an exception or interrupt.
This field is not updated on any exception which sets $Status_{ERL}$ to 1 (i.e., Reset, Soft Reset, NMI, cache error), an entry into EJTAG DEBUG mode, or any exception or interrupt that occurs with $Status_{EXL}$ = 1, or $Status_{BEV}$ = 1. Neither is it updated on an ERET with $Status_{ERL}$ = 1 or $Status_{BEV}$ = 1. This field is not updated on an exception that occurs while $Status_{ERL}$ = 1.
The value of CSS can be changed directly by software only by writing the PSS field and executing an ERET instruction.

2.12.8 SRSMAP: Register (CP0 Register 12, Select 3)

The SRSMAP register contains eight 4-bit fields that provide the mapping from an vector number to the shadow set number to use when servicing such an interrupt. The values from this register are not used for a non-interrupt exception, or a non-vectored interrupt ($Cause_{IV}$ = 0 or $IntCtl_{VS}$ = 0). In such cases, the shadow set number comes from $SRSCtl_{ESS}$.

If $SRSCtl_{HSS}$ is '0', the results of a software read or write of this register are *unpredictable*.

The operation of the processor is *undefined* if a value is written to any field in this register that is greater than the value of $SRSCtl_{HSS}$.

The SRSMAP register contains the shadow register set numbers for vector numbers 7..0. The same shadow set number can be established for multiple interrupt vectors, creating a many-to-one mapping from a vector to a single shadow register set number.

Register 2-8: SRSMAP: Register; CP0 Register 12, Select 3

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SSV7<31:28>				SSV6<27:24>			
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SSV5<23:20>				SSV4<19:16>			
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SSV3<15:12>				SSV2<11:8>			
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SSV1<7:4>				SSV0<3:0>			
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: ('0', '1', x = Unknown)		

bit 31-28 **SSV7<31:28>:** Shadow Set Vector 7 bits
Shadow register set number for Vector Number 7

bit 27-24 **SSV6<27:24>:** Shadow Set Vector 6 bits
Shadow register set number for Vector Number 6

bit 23-20 **SSV5<23:20>:** Shadow Set Vector 5 bits
Shadow register set number for Vector Number 5

bit 19-16 **SSV4<19:16>:** Shadow Set Vector 4 bits
Shadow register set number for Vector Number 4

bit 15-12 **SSV3<15:12>:** Shadow Set Vector 3 bits
Shadow register set number for Vector Number 3

bit 11-8 **SSV2<11:8>:** Shadow Set Vector 2 bits
Shadow register set number for Vector Number 2

bit 7-4 **SSV1<7:4>:** Shadow Set Vector 1 bits
Shadow register set number for Vector Number 1

Register 2-8: SRSMAP: Register; CP0 Register 12, Select 3 (Continued)

bit 3-0 **SSV0<3:0>:** Shadow Set Vector 0 bit
Shadow register set number for Vector Number 0

2.12.9 CAUSE Register (CP0 Register 13, Select 0)

The CAUSE register primarily describes the cause of the most recent exception. In addition, fields also control software interrupt requests and the vector through which interrupts are dispatched. With the exception of the $IP_{1..0}$, DC, IV and WP fields, all fields in the CAUSE register are read-only. $IP_{7..2}$ are interpreted as the Requested Interrupt Priority Level (RIPL).

Table 2-11: Cause Register ExcCode Field

Exception Code Value		Mnemonic	Description
Decimal	Hex		
0	16#00	Int	Interrupt
4	16#04	AdEL	Address error exception (load or instruction fetch)
5	16#05	AdES	Address error exception (store)
6	16#06	IBE	Bus error exception (instruction fetch)
7	16#07	DBE	Bus error exception (data reference: load or store)
8	16#08	Sys	Syscall exception
9	16#09	Bp	Breakpoint exception
10	16#0a	RI	Reserved instruction exception
11	16#0b	CPU	Coprocessor Unusable exception
12	16#0c	Ov	Arithmetic Overflow exception
13	16#0d	Tr	Trap exception
14-18	16#0e-16#12	–	Reserved

Register 2-9: CAUSE: Register; CP0 Register 13, Select 0

R-x	R-x	R-x	R-x	R/W-0	R-0	r-x	r-x
BD	TI	CE<29:28>		DC	R	0<25:24>	
bit 31							bit 24

R/W-x	R/W-0	r-x	r-x	r-x	r-x	r-x	r-x
IV	R	0<21:16>					
bit 23							bit 16

R-x	R-x	R-x	R-x	R-x	R-x	R/W-x	R/W-x
RIPL<15:10>						IP1..IP0<9:8>	
bit 15							bit 8

r-x	R-x	R-x	R-x	R-x	R-x	r-x	r-x
0	EXCCODE<6:2>					0<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31 **BD:** Branch Delay bit

Indicates whether the last exception taken occurred in a branch delay slot:
0 = Not in delay slot
1 = In delay slot
The processor updates BD only if $Status_{EXL}$ was ‘0’ when the exception occurred.

bit 30 **TI:** Timer Interrupt bit

Timer Interrupt. This bit denotes whether a timer interrupt is pending (analogous to the IP bits for other interrupt types):
0 = No timer interrupt is pending
1 = Timer interrupt is pending

bit 29-28 **CE<29:28>:** Coprocessor Exception bits

Coprocessor unit number referenced when a Coprocessor Unusable exception is taken. This field is loaded by hardware on every exception, but is *unpredictable* for all exceptions except for Coprocessor Unusable.

bit 27 **DC:** Disable Count bit

Disable Count register. In some power-sensitive applications, the COUNT register is not used and can be stopped to avoid unnecessary toggling
0 = Enable counting of COUNT register
1 = Disable counting of COUNT register

bit 26 **R:** bit

bit 25-24 **Reserved:** Write ‘0’; ignore read

Must be written as ‘0’; returns ‘0’ on read.

Register 2-9: CAUSE: Register; CP0 Register 13, Select 0 (Continued)

bit 23	**IV:** Interrupt Vector bit Indicates whether an interrupt exception uses the general exception vector or a special interrupt vector 0 = Use the general exception vector (16#180) 1 = Use the special interrupt vector (16#200) If the $Cause_{IV}$ is 1 and $Status_{BEV}$ is 0, the special interrupt vector represents the base of the vectored interrupt table.
bit 22	**R:** bit
bit 21-16	**Reserved:** Write '0'; ignore read Must be written as '0'; returns '0' on read.
bit 15-10	**RIPL<15:10>:** Requested Interrupt Priority Level bits Requested Interrupt Priority Level.\ This field is the encoded (0..63) value of the requested interrupt. A value of '0' indicates that no interrupt is requested.
bit 9-8	**IP1..IP0<9:8>:** Controls the request for software interrupts: 0 = No interrupt requested 1 = Request software interrupt These bits are exported to the system interrupt controller for prioritization in EIC interrupt mode with other interrupt sources
bit 7	**Reserved:** Write '0'; ignore read Must be written as '0'; returns '0' on read.
bit 6-2	**EXCCODE<6:2>:** Exception Code bits Exception code - see Table 2-11
bit 1-0	**Reserved:** Write '0'; ignore read Must be written as '0'; returns '0' on read.

2.12.10 EPC Register (CP0 Register 14, Select 0)

The Exception Program Counter (EPC) is a read/write register that contains the address at which processing resumes after an exception has been serviced. All bits of the EPC register are significant and are writable.

For synchronous (precise) exceptions, the EPC contains one of the following:

- The virtual address of the instruction that was the direct cause of the exception.
- The virtual address of the immediately preceding BRANCH or JUMP instruction, when the exception causing instruction is in a branch delay slot and the Branch Delay bit in the CAUSE register is set.

On new exceptions, the processor does not write to the EPC register when the EXL bit in the STATUS register is set, however, the register can still be written via the MTC0 instruction.

Since the PIC32 family implements MIPS16e ASE, a read of the EPC register (via MFC0) returns the following value in the destination GPR:

$$GPR[rt] \leftarrow ExceptionPC_{31..1} \,||\, ISAMode_{0}$$

That is, the upper 31 bits of the exception PC are combined with the lower bit of the ISAMode field and written to the GPR.

Similarly, a write to the EPC register (via MTC0) takes the value from the GPR and distributes that value to the exception PC and the ISAMode field, as follows

$$ExceptionPC \leftarrow GPR[rt]_{31..1} \,||\, 0$$
$$ISAMode \leftarrow 2\#0 \,||\, GPR[rt]_{0}$$

That is, the upper 31 bits of the GPR are written to the upper 31 bits of the exception PC, and the lower bit of the exception PC is cleared. The upper bit of the ISAMode field is cleared and the lower bit is loaded from the lower bit of the GPR.

Register 2-10: EPC: Register; CP0 Register 14, Select 0

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
EPC<31:24>							
bit 31							bit 24

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
EPC<23:16>							
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
EPC<15:8>							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
EPC<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **EPC<31:0>:** Exception Program Counter bits

2.12.11 PRID Register (CP0 Register 15, Select 0)

The Processor Identification (PRID) register is a 32 bit read-only register that contains information identifying the manufacturer, manufacturer options, processor identification, and revision level of the processor.

Register 2-11: PRID: Register; CP0 Register 15, Select 0

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
R<31:24>							
bit 31							bit 24

R-1	R-1	R-1	R-1	R-1	R-1	R-1	R-1
COMPANY ID<23:16>							
bit 23							bit 16

R-0x87	R-0x87	R-0x87	R-0x87	R-0x87	R-0x87	R-0x87	R-0x87
PROCESSOR ID<15:8>							
bit 15							bit 8

R-Preset	R-Preset	R-Preset	R-Preset	R-Preset	R-Preset	R-Preset	R-Preset
REVISION<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-24 **R<31:24>:** Reserved
Must be ignored on write and read as ‘0’

bit 23-16 **COMPANY ID<23:16>:**
Identifies the company that designed or manufactured the processor. In the PIC32MX this field contains a value of 1 to indicate MIPS Technologies, Inc.

bit 15-8 **PROCESSOR ID<15:8>:**
Identifies the type of processor. This field allows software to distinguish between the various types of MIPS Technologies processors.

bit 7-0 **REVISION<7:0>:**
Specifies the revision number of the processor. This field allows software to distinguish between one revision and another of the same processor type.
This field is broken up into the following three subfields.

bit 7-5 **MAJOR REVISION<7:5>:**
This number is increased on major revisions of the processor core.

bit 4-2 **MINOR REVISION<4:2>:**
This number is increased on each incremental revision of the processor and reset on each new major revision.

bit 1-0 **PATCH LEVEL<1:0>:**
If a patch is made to modify an older revision of the processor, this field will be incremented.

2.12.12 EBASE Register (CP0 Register 15, Select 1)

The EBASE register is a read/write register containing the base address of the exception vectors used when $Status_{BEV}$ equals '0', and a read-only CPU number value that may be used by software to distinguish different processors in a multi-processor system.

The EBASE register provides the ability for software to identify the specific processor within a multi-processor system, and allows the exception vectors for each processor to be different, especially in systems composed of heterogeneous processors. Bits 31..12 of the EBASE register are concatenated with zeros to form the base of the exception vectors when $Status_{BEV}$ is '0'. The exception vector base address comes from the fixed defaults when $Status_{BEV}$ is '1', or for any EJTAG Debug exception. The Reset state of bits 31..12 of the EBASE register initialize the exception base register to 16#8000.0000.

Bits 31..30 of the EBASE Register are fixed with the value 2#10 to force the exception base address to be in the kseg0 or kseg1 unmapped virtual address segments.

If the value of the exception base register is to be changed, this must be done with $Status_{BEV}$ equal '1'. The operation of the processor is *undefined* if the Exception Base field is written with a different value when $Status_{BEV}$ is '0'.

Combining bits 31..20 with the Exception Base field allows the base address of the exception vectors to be placed at any 4 KBbyte page boundary.

Register 2-12: EBASE: Register; CP0 Register 15, Select 1

R-1	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
1	0	EXCEPTION BASE<29:24>					
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
EXCEPTION BASE<23:16>							
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	r-0	r-0	R-0	R-0
EXCEPTION BASE<15:12>				r		CPUNUM<9:8>	
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
CPUNUM<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: ('0', '1', x = Unknown)		

bit 31 **1:** One bit
This bit is ignored on write and returns one on read.

bit 30 **0:** Zero bit
This bit is ignored on write and returns '0' on read.

bit 29-12 **EXCEPTION BASE<29:12>:**
In conjunction with bits 31..30, this field specifies the base address of the exception vectors when $Status_{BEV}$ is '0'.

bit 11-10 **Reserved:**
Must be written as '0'; returns '0' on read.

Register 2-12: EBASE: Register; CP0 Register 15, Select 1 (Continued)

bit 9-0 **CPUNUM<9:0>:**

This field specifies the number of the CPU in a multi-processor system and can be used by software to distinguish a particular processor from the others. In a single processor system, this value is set to '0'.

2.12.13 CONFIG Register (CP0 Register 16, Select 0)

The CONFIG register specifies various configuration and capabilities information. Most of the fields in the CONFIG register are initialized by hardware during the Reset exception process, or are constant.

Table 2-12: Cache Coherency Attributes

C(2:0) Value	Cache Coherency Attribute
2	Uncached
3	Cacheable

Register 2-13: CONFIG: Register; CP0 Register 16, Select 0

R-1	R-0	R-1	R-0	R/W-0	R/W-1	R/W-0	r-0
M	K23<30:28>			KU<27:25>			0
bit 31							bit 24

r-x	R-0	R-0	R-0	r-x	r-x	r-x	R-1
0	UDI	SB	MDU				DS
bit 23							bit 16

R-0	R-0	R-0	R-0	R-0	R-1	R-0	R-1
BE	AT<14:13>		AR<12:10>			MT<9:8>	
bit 15							bit 8

R-1	r-x	r-x	r-x	r-x	R/W-0	R/W-1	R/W-0
MT					K0<2:0>		
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31 **M:**
This bit is hardwired to ‘1’ to indicate the presence of the CONFIG1 register.

bit 30-28 **K23<30:28>:** kseg2 and kseg3 bits
This field controls the cacheability of the kseg2 and kseg3 address segments.
Refer to Table 2-12 for the field encoding.

bit 27-25 **KU<27:25>:** kuseg and useg bits
This field controls the cacheability of the kuseg and useg address segments.
Refer to Table 2-12 for the field encoding.

bit 24-23 **Reserved**: Write ‘0’; ignore read
Must be written as ‘0’. Returns ‘0’ on reads.

bit 22 **UDI:** User Defined bit
This bit indicates that CorExtend User Defined Instructions have been implemented.
0 = No User Defined Instructions are implemented
1 = User Defined Instructions are implemented

Register 2-13: CONFIG: Register; CP0 Register 16, Select 0 (Continued)

bit 21	**SB:** SimpleBE bit Indicates whether SimpleBE Bus mode is enabled. 0 = No reserved byte enables on internal bus interface 1 = Only simple byte enables allowed on internal bus interface
bit 20	**MDU:** Multiply/Divide Unit bit This bit indicates the type of Multiply/Divide Unit present 0 = Fast, high-performance MDU
bit 19-17	**Reserved:** Write '0'; ignore read Must be written as 0. Returns '0' on reads.
bit 16	**DS:** Dual SRAM bit 0 = Unified instruction/data SRAM internal bus interface 1 = Dual instruction/data SRAM internal bus interfaces **Note:** The PIC32MX family currently uses Dual SRAM-style interfaces internally.
bit 15	**BE:** Big Endian bit Indicates the Endian mode in which the processor is running, PIC32MX is always little endian. 0 = Little endian 1 = Big enidan
bit 14-13	**AT<14:13>:** Architecture Type bits Architecture type implemented by the processor. This field is always '00' to indicate the MIPS32 architecture.
bit 12-10	**AR<12:10>:** Architecture Revision Level bits Architecture revision level. This field is always '001' to indicate MIPS32 Release 2. 0: Release 1 1: Release 2 2-7: Reserved
bit 9-7	**MT<9:7>:** MMU Type bits 3: Fixed mapping 0-2, 4-7: Reserved
bit 6-3	**Reserved:** Write '0'; ignore read Must be written as zeros; returns zeros on reads
bit 2-0	**K0<2:0>:** Kseg0 bits Kseg0 coherency algorithm. Refer to XREF Table 2-12 for the field encoding.

2.12.14 CONFIG1 Register (CP0 Register 16, Select 1)

The CONFIG1 register is an adjunct to the CONFIG register and encodes additional information about capabilities present on the core. All fields in the CONFIG1 register are read-only.

Register 2-14: CONFIG1: CONFIG1 Register; CP0 Register 16, Select 1

R-1	R-x	R-x	R-x	R-x	R-x	R-x	R-x
M	MMU Size<30:25>						IS
bit 31							bit 24

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
IS<23:22>			IL<21:19>			IA<18:16>	
bit 23							bit 16

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
DS<15:13>			DL<12:10>			DA<9:8>	
bit 15							bit 8

R-x	R-0	R-0	R-0	R-0	R-1	R-x	R-0
DA	C2	MD	PC	WR	CA	EP	FP
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31 **M:** bit
This bit is hardwired to ‘1’ to indicate the presence of the CONFIG2 register.

bit 30-25 **MMU Size:** bits
This field contains the number of entries in the TLB minus one; since the PIC32MX has no TLB, this field is ‘0’.

bit 24-22 **IS:** Instruction Cache Sets bits
This field contains the number of instruction cache sets per way; since the M4K core does not include caches, this field is always read as ‘0’.

bit 21-19 **IL:** Instruction-Cache Line bits
This field contains the instruction cache line size; since the M4K core does not include caches, this field is always read as ‘0’.

bit 18-16 **IA:** Instruction-Cache Associativity bits
This field contains the level of instruction cache associativity; since the M4K core does not include caches, this field is always read as ‘0’.

bit 15-13 **DS:** Data-Cache Sets bits
This field contains the number of data cache sets per way; since the M4K core does not include caches, this field is always read as ‘0’.

bit 12-10 **DL:** Data-Cache Line bits
This field contains the data cache line size; since the M4K core does not include caches, this field is always read as ‘0’.

bit 9-7 **DA:** Data-Cache Associativity bits
This field contains the type of set associativity for the data cache; since the M4K core does not include caches, this field is always read as ‘0’.

Register 2-14: CONFIG1: CONFIG1 Register; CP0 Register 16, Select 1 (Continued)

bit 6 C2: Coprocessor 2 bit

Coprocessor 2 present.
0 = No coprocessor is attached to the COP2 interface
1 = A coprocessor is attached to the COP2 interface
Since coprocessor 2 is not implemented in the PIC32MX family of microcontrollers, this bit will read '0'.

bit 5 **MD:** MDMX bit

MDMX implemented.
This bit always reads as '0' because MDMX is not supported.

bit 4 **PC:** Performance Counter bit

Performance Counter registers implemented.
Always a '0' since the PIC32MX core does not contain Performance Counters.

bit 3 **WR:** Watch Register bit

Watch registers implemented.
0 = No Watch registers are present
1 = One or more Watch registers are present

Note: The PIC32MX does not implement watch registers, therefore this bit always reads '0'.

bit 2 **CA:** Code Compression Implemented bit

0 = No MIPS16e present
1 = MIPS16e is implemented

bit 1 **EP:** EJTAG Present bit

This bit is always set to indicate that the core implements EJTAG.

bit 0 **FP:** FPU Implemented bit

This bit is always '0' since the core does not contain a floating point unit.

2.12.15 CONFIG2 (CP0 Register 16, Select 2)

The CONFIG2 register is an adjunct to the CONFIG register and is reserved to encode additional capabilities information. CONFIG2 is allocated for showing the configuration of level 2/3 caches. These fields are reset to '0' because L2/L3 caches are not supported by the PIC32MX core. All fields in the CONFIG2 register are read-only.

Register 2-15: CONFIG2: CONFIG2 Register; CP0 Register 16, Select 2

R-1	r-0	r-0	r-0	r-0	r-0	r-0	r-0
M	0	0	0	0	0	0	0
bit 31							bit 24

r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
0	0	0	0	0	0	0	0
bit 23							bit 16

r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
0	0	0	0	0	0	0	0
bit 15							bit 8

r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
0	0	0	0	0	0	0	0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: ('0', '1', x = Unknown)		

bit 31 **M:** bit
This bit is hardwired to '1' to indicate the presence of the CONFIG3 register.

bit 30-0 **Reserved**

2.12.16 CONFIG3 Register (CP0 Register 16, Select 3)

The CONFIG3 register encodes additional capabilities. All fields in the CONFIG3 register are read-only.

Register 2-16: CONFIG3: CONFIG3 Register; CP0 Register 16, Select 3

R-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
M	0	0	0	0	0	0	0
bit 31							bit 24

r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
0	0	0	0	0	0	0	0
bit 23							bit 16

r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
0	0	0	0	0	0	0	0
bit 15							bit 8

r-0	R-1	R-1	R-0	r-0	r-0	R-0	R-0
0	VEIC	VInt	SP	0	0	SM	TL
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31 **M:** Reserved
This bit is reserved to indicate that a CONFIG4 register is present. With the current architectural definition, this bit should always read as a ‘0’.

bit 30-7 **Reserved:** Write ‘0’; ignore read
Must be written as zeros; returns zeros on read.

bit 6 **VEIC:**
Support for an external interrupt controller is implemented.
0 = Support for EIC Interrupt mode is not implemented
1 = Support for EIC Interrupt mode is implemented

Note: PIC32MX internally implements a MIPS “external interrupt controller”, therefore this bit reads ‘1’.

bit 5 **VINT:** Vector Interrupt bit
Vectored interrupts implemented. This bit indicates whether vectored interrupts are implemented.
0 = Vector interrupts are not implemented
1 = Vector interrupts are implemented
On the PIC32MX core, this bit is always a ‘1’ since vectored interrupts are implemented.

bit 4 **SP:** Support Page bit
Small (1 KByte) page support is implemented, and the PAGEGRAIN register exists.
0 = Small page support is not implemented
1 = Small page support is implemented

Note: PIC32MX always reads ‘0’ since PIC32MX does not implement small page support.

bit 3-2 **0:**
Must be written as zeros; returns zeros on read.

Register 2-16: CONFIG3: CONFIG3 Register; CP0 Register 16, Select 3 (Continued)

bit 1 **SM:** SmartMIPS™ bit

SmartMIPS™ ASE implemented. This bit indicates whether the SmartMIPS ASE is implemented. Since SmartMIPS is present on the PIC32MX core, this bit will always be '0'.
0 = SmartMIPS ASE is not implemented
1 = SmartMIPS ASE is implemented

bit 0 **TL:** Trace Logic bit

Trace Logic implemented. This bit indicates whether PC or data trace is implemented.
0 = On-chip trace logic (PDTrace™) is not implemented
1 = On-chip trace logic (PDTrace™) is implemented

Note: PIC32MX does not implement PDTrace™ on-chip trace logic, therefore this bit always reads '0'.

2.12.17 DEBUG Register (CP0 Register 23, Select 0)

The DEBUG register is used to control the debug exception and provide information about the cause of the debug exception and when re-entering at the debug exception vector due to a normal exception in DEBUG mode. The read-only information bits are updated every time the debug exception is taken or when a normal exception is taken when already in DEBUG mode.

Only the DM bit and the EJTAGver field are valid when read from Non-DEBUG mode; the values of all other bits and fields are *unpredictable*. Operation of the processor is *undefined* if the DEBUG register is written from Non-DEBUG mode.

Some of the bits and fields are only updated on debug exceptions and/or exceptions in DEBUG mode, as shown below:

- DSS, DBp, DDBL, DDBS, DIB, DINT are updated on both debug exceptions and on exceptions in Debug modes
- DExcCode is updated on exceptions in DEBUG mode, and is undefined after a debug exception
- Halt and Doze are updated on a debug exception, and are undefined after an exception in DEBUG mode
- DBD is updated on both debug and on exceptions in Debug modes

All bits and fields are undefined when read from normal mode, except EJTAGver and DM.

Register 2-17: DEBUG: Register; CP0 Register 23, Select 0

R-U	R-0	R-0	R/W-0	R-U	R-U	R/W-1	R/W-0
DBD	DM	NODCR	LSNM	DOZE	HALT	COUNTDM	IBUSEP
bit 31							bit 24

R-0	R-0	R/W-0	R/W-0	R-0	R-0	R-0	R-1
MCHECKP	CACHEEP	DBUSEP	IEXI	DDBSIMPR	DDBLIMPR	VER<7:6>	
bit 23							bit 16

R-0	R-U	R-U	R-U	R-U	R-U	R-0	R/W-0
VER	DEXCCODE<14:10					NOSST	SST
bit 15							bit 8

R-0	R-0	R-U	R-U	R-U	R-U	R-U	R-U
R<7:6>		DINT	DIB	DDBS	DDBL	DBP	DSS
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31 **DBD:**
Indicates whether the last debug exception or exception in DEBUG mode, occurred in a branch delay slot:
0 = Not in delay slot
1 = In delay slot

bit 30 **DM:**
Indicates that the processor is operating in DEBUG mode:
0 = Processor is operating in Non-DEBUG mode
1 = Processor is operating in DEBUG mode

bit 29 **NODCR:**
Indicates whether the dseg memory segment is present and the Debug Control Register is accessible:
0 = dseg is present
1 = No dseg present

bit 28 **LSNM:**
Controls access of load/store between dseg and main memory:
0 = Load/stores in dseg address range goes to dseg
1 = Load/stores in dseg address range goes to main memory

bit 27 **DOZE:**
Indicates that the processor was in any kind of Low-Power mode when a debug exception occurred:
0 = Processor not in Low-Power mode when debug exception occurred
1 = Processor in Low-Power mode when debug exception occurred

bit 26 **HALT:**
Indicates that the internal system bus clock was stopped when the debug exception occurred:
0 = Internal system bus clock stopped
1 = Internal system bus clock running

Register 2-17: DEBUG: Register; CP0 Register 23, Select 0 (Continued)

bit 25 **COUNTDM:**

Indicates the Count register behavior in DEBUG mode.
0 = Count register stopped in DEBUG mode
1 = Count register is running in DEBUG mode

bit 24 **IBUSEP:**

Instruction fetch Bus Error exception Pending. Set when an instruction fetch bus error event occurs or if a '1' is written to the bit by software. Cleared when a Bus Error exception on instruction fetch is taken by the processor, and by Reset. If IBUSEP is set when IEXI is cleared, a Bus Error exception on instruction fetch is taken by the processor, and IBUSEP is cleared.

bit 23 **MCHECKP:**

Indicates that an imprecise Machine Check exception is pending. All Machine Check exceptions are precise on the PIC32MX processor so this bit will always read as '0'.

bit 22 **CACHEEP:**

Indicates that an imprecise Cache Error is pending. Cache Errors cannot be taken by the PIC32MX core so this bit will always read as '0'.

bit 21 **DBUSEP:**

Data access Bus Error exception Pending. Covers imprecise bus errors on data access, similar to behavior of IBUSEP for imprecise bus errors on an instruction fetch.

bit 20 **IEXI:**

Imprecise Error eXception Inhibit controls exceptions taken due to imprecise error indications. Set when the processor takes a debug exception or exception in DEBUG mode. Cleared by execution of the DERET instruction; otherwise modifiable by DEBUG mode software. When IEXI is set, the imprecise error exception from a bus error on an instruction fetch or data access, cache error, or machine check is inhibited and deferred until the bit is cleared.

bit 19 **DDBSIMPR:**

Indicates that an imprecise Debug Data Break Store exception was taken. All data breaks are precise on the PIC32MX core, so this bit will always read as '0'.

bit 18 **DDBLIMPR:**

Indicates that an imprecise Debug Data Break Load exception was taken. All data breaks are precise on the PIC32MX core, so this bit will always read as '0'.

bit 17-15 **VER:**

EJTAG version

bit 14-10 **DEXCCODE:**

Indicates the cause of the latest exception in DEBUG mode. The field is encoded as the ExcCode field in the CAUSE register for those normal exceptions that may occur in DEBUG mode.
Value is undefined after a debug exception.

bit 9 **NOSST:**

Indicates whether the single-step feature controllable by the SST bit is available in this implementation:
0 = Single-step feature available
1 = No single-step feature available

bit 8 **SST:**

Controls if debug single step exception is enabled:
0 = No debug single-step exception enabled
1 = Debug single step exception enabled

bit 7-6 **Reserved:**

Must be written as zeros; returns zeros on reads.

bit 5 **DINT:**

Indicates that a debug interrupt exception occurred. Cleared on exception in DEBUG mode.
0 = No debug interrupt exception
1 = Debug interrupt exception

Register 2-17: DEBUG: Register; CP0 Register 23, Select 0 (Continued)

bit 4 **DIB:**

Indicates that a debug instruction break exception occurred. Cleared on exception in DEBUG mode.
0 = No debug instruction exception
1 = Debug instruction exception

bit 3 **DDBS:**

Indicates that a debug data break exception occurred on a store. Cleared on exception in DEBUG mode.
0 = No debug data exception on a store
1 = Debug instruction exception on a store

bit 2 **DDBL:**

Indicates that a debug data break exception occurred on a load. Cleared on exception in DEBUG mode.
0 = No debug data exception on a load
1 = Debug instruction exception on a load

bit 1 **DBP:**

Indicates that a debug software breakpoint exception occurred. Cleared on exception in DEBUG mode.
0 = No debug software breakpoint exception
1 = Debug software breakpoint exception

bit 0 **DSS:**

Indicates that a debug single-step exception occurred. Cleared on exception in DEBUG mode.
0 = No debug single-step exception
1 = Debug single-step exception

2.12.18 DEPC Register (CP0 Register 24, Select 0)

The Debug Exception Program Counter (DEPC) register is a read/write register that contains the address at which processing resumes after a debug exception or DEBUG mode exception has been serviced.

For synchronous (precise) debug and DEBUG mode exceptions, the DEPC contains either:

- The virtual address of the instruction that was the direct cause of the debug exception, or
- The virtual address of the immediately preceding branch or jump instruction, when the debug exception causing instruction is in a branch delay slot, and the Debug Branch Delay (DBD) bit in the Debug register is set.

For asynchronous debug exceptions (debug interrupt), the DEPC contains the virtual address of the instruction where execution should resume after the debug handler code is executed.

Since the PIC32 family implements the MIPS16e ASE, a read of the DEPC register (via MFC0) returns the following value in the destination GPR:

$$GPR[rt] = DebugExceptionPC_{31..1} \;||\; ISAMode_0$$

That is, the upper 31 bits of the debug exception PC are combined with the lower bit of the ISAMode field and written to the GPR.

Similarly, a write to the DEPC register (via MTC0) takes the value from the GPR and distributes that value to the debug exception PC and the ISAMode field, as follows:

$$DebugExceptionPC = GPR[rt]_{31..1} \;||\; 0$$
$$ISAMode = 2\#0 \;||\; GPR[rt]_0$$

That is, the upper 31 bits of the GPR are written to the upper 31 bits of the debug exception PC, and the lower bit of the debug exception PC is cleared. The upper bit of the ISAMode field is cleared and the lower bit is loaded from the lower bit of the GPR.

Register 2-18: DEPC: Debug Exception Program Counter Register; CP0 Register 24, Select 0

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
DEPC<31:24>							
bit 31							bit 24

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
DEPC<23:16>							
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
DEPC<15:8>							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
DEPC<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **DEPC<31:0>:** Debug Exception Program Counter bits

The DEPC register is updated with the virtual address of the instruction that caused the debug exception. If the instruction is in the branch delay slot, then the virtual address of the immediately preceding branch or jump instruction is placed in this register.
Execution of the DERET instruction causes a jump to the address in the DEPC.

2.12.19 ErrorEPC (CP0 Register 30, Select 0)

The ErrorEPC register is a read/write register, similar to the EPC register, except that ErrorEPC is used on error exceptions. All bits of the ErrorEPC register are significant and must be writable. It is also used to store the program counter on Reset, Soft Reset, and nonmaskable interrupt (NMI) exceptions.

The ErrorEPC register contains the virtual address at which instruction processing can resume after servicing an error. This address can be:

- The virtual address of the instruction that caused the exception
- The virtual address of the immediately preceding branch or jump instruction when the error causing instruction is in a branch delay slot

Unlike the EPC register, there is no corresponding branch delay slot indication for the ErrorEPC register.

Since the PIC32 family implements the MIPS16e ASE, a read of the ErrorEPC register (via MFC0) returns the following value in the destination GPR:

$$GPR[rt] = ErrorExceptionPC_{31..1} \;||\; ISAMode_0$$

That is, the upper 31 bits of the error exception PC are combined with the lower bit of the ISAMode field and written to the GPR.

Similarly, a write to the ErrorEPC register (via MTC0) takes the value from the GPR and distributes that value to the error exception PC and the ISAMode field, as follows:

$$ErrprExceptionPC = GPR[rt]_{31..1} \;||\; 0$$
$$ISAMode = 2\#0 \;||\; GPR[rt]_0$$

That is, the upper 31 bits of the GPR are written to the upper 31 bits of the error exception PC, and the lower bit of the error exception PC is cleared. The upper bit of the ISAMode field is cleared and the lower bit is loaded from the lower bit of the GPR.

Register 2-19: ErrorEPC: Error Exception Program Counter Register; CP0 Register 30, Select 0

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
ErrorEPC<31:24>							
bit 31							bit 24

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
ErrorEPC<23:16>							
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
ErrorEPC<15:8>							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
ErrorEPC<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **ErrorEPC<31:0>:** Error Exception Program Counter bits

2.12.20 DeSave Register (CP0 Register 31, Select 0)

The Debug Exception Save (DeSave) register is a read/write register that functions as a simple memory location. This register is used by the debug exception handler to save one of the GPRs that is then used to save the rest of the context to a pre-determined memory area (such as in the EJTAG Probe). This register allows the safe debugging of exception handlers and other types of code where the existence of a valid stack for context saving cannot be assumed.

Register 2-20: DeSave: Debug Exception Save Register; CP0 Register 31, Select 0

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
DESAVE<31:24>							
bit 31							bit 24

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
DESAVE<23:16>							
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
DESAVE<15:8>							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
DESAVE<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **DESAVE<31:0>:** Debug Exception Save bits
Scratch Pad register used by Debug Exception code.

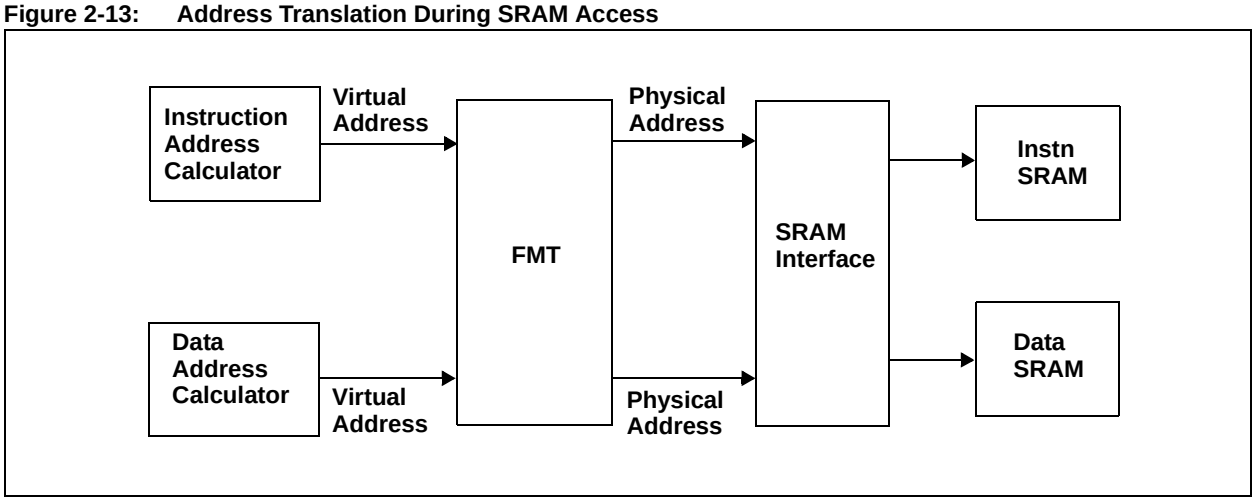
2.13 MIPS16e™ EXECUTION

When the core is operating in MIPS16e mode, instruction fetches only require 16-bits of data to be returned. For improved efficiency, however, the core will fetch 32-bits of instruction data whenever the address is word-aligned. Thus for sequential MIPS16e code, fetches only occur for every other instruction, resulting in better performance and reduced system power.

2.14 MEMORY MODEL

Virtual addresses used by software are converted to physical addresses by the memory management unit (MMU) before being sent to the CPU busses. The PIC32MX CPU uses a fixed mapping for this conversion. For more information regarding the system memory model, see **Section 3. "Memory Organization"**.

Figure 2-13: Address Translation During SRAM Access



2.14.1 Cacheability

The CPU uses the virtual address of an instruction fetch, load or store to determine whether to access the cache or not. Memory accesses within kseg0, or useg/kuseg can be cached, while accesses within kseg1 are non-cacheable. The CPU uses the CCA bits in the CONFIG register to determine the cacheability of a memory segment. A memory access is cacheable if its corresponding CCA = 011_2.

For more information on cache operation, see **Section 4. "Prefetch Cache Module"**.

2.14.1.1 Little Endian Byte Ordering

On CPUs that address memory with byte resolution, there is a convention for multi-byte data items that specify the order of high-order to low-order bytes. Big-endian byte-ordering is where the lowest address has the Most Significant Byte. Little-endian ordering is where the lowest address has the Least Significant Byte of a multi-byte datum. The PIC32MX CPU family supports little-endian byte ordering.

Figure 2-14: Big Endian Byte Ordering

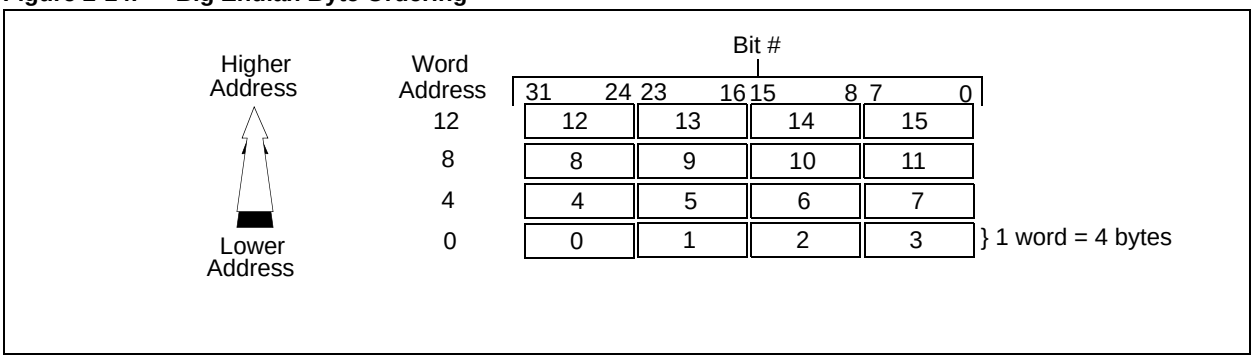
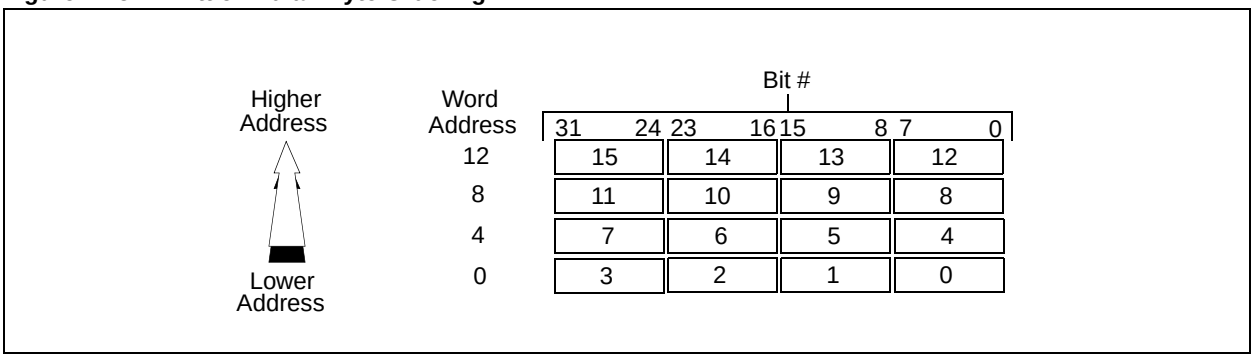


Figure 2-15: Little Endian Byte Ordering



2.15 CPU INSTRUCTIONS, GROUPED BY FUNCTION

CPU instructions are organized into the following functional groups:

- Load and store
- Computational
- Jump and branch
- Miscellaneous
- Coprocessor

Each instruction is 32 bits long.

2.15.1 CPU Load and Store Instructions

MIPS processors use a load/store architecture; all operations are performed on operands held in processor registers and main memory is accessed only through load and store instructions.

2.15.1.1 Types of Loads and Stores

There are several different types of load and store instructions, each designed for a different purpose:

- Transferring variously-sized fields (for example, LB, SW)
- Trading transferred data as signed or unsigned integers (for example, LHU)
- Accessing unaligned fields (for example, LWR, SWL)
- Atomic memory update (read-modify-write: for instance, LL/SC)

2.15.1.2 List of CPU Load and Store Instructions

The following data sizes (as defined in the *AccessLength* field) are transferred by CPU load and store instructions:

- Byte
- Halfword
- Word

Signed and unsigned integers of different sizes are supported by loads that either sign-extend or zero-extend the data loaded into the register.

Unaligned words and doublewords can be loaded or stored in just two instructions by using a pair of special instructions. For loads a LWL instruction is paired with a LWR instruction. The load instructions read the left-side or right-side bytes (left or right side of register) from an aligned word and merge them into the correct bytes of the destination register.

2.15.1.3 Loads and Stores Used for Atomic Updates

The paired instructions, Load Linked and Store Conditional, can be used to perform an atomic read-modify-write of word or doubleword cached memory locations. These instructions are used in carefully coded sequences to provide one of several synchronization primitives, including test-and-set, bit-level locks, semaphores, and sequencers and event counts.

2.15.1.4 Coprocessor Loads and Stores

If a particular coprocessor is not enabled, loads and stores to that processor cannot execute and the attempted load or store causes a Coprocessor Unusable exception. Enabling a coprocessor is a privileged operation provided by the System Control Coprocessor, CP0.

2.15.2 Computational Instructions

Two's complement arithmetic is performed on integers represented in 2s complement notation. These are signed versions of the following operations:

- Add
- Subtract
- Multiply
- Divide

The add and subtract operations labelled "unsigned" are actually modulo arithmetic without overflow detection.

There are also unsigned versions of *multiply* and *divide*, as well as a full complement of *shift* and *logical* operations. Logical operations are not sensitive to the width of the register.

MIPS32 provided 32-bit integers and 32-bit arithmetic.

2.15.2.1 Shift Instructions

The ISA defines two types of shift instructions:

- Those that take a fixed shift amount from a 5-bit field in the instruction word (for instance, SLL, SRL)
- Those that take a shift amount from the low-order bits of a general register (for instance, SRAV, SRLV)

2.15.2.2 Multiply and Divide Instructions

The multiply instruction performs 32-bit by 32-bit multiplication and creates either 64-bit or 32-bit results. Divide instructions divide a 64-bit value by a 32-bit value and create 32-bit results. With one exception, they deliver their results into the HI and LO special registers. The MUL instruction delivers the lower half of the result directly to a GPR.

- Multiply produces a full-width product twice the width of the input operands; the low half is loaded into LO and the high half is loaded into HI.
- Multiply-Add and Multiply-Subtract produce a full-width product twice the width of the input operations and adds or subtracts the product from the concatenated value of HI and LO. The low half of the addition is loaded into LO and the high half is loaded into HI.
- Divide produces a quotient that is loaded into LO and a remainder that is loaded into HI.

The results are accessed by instructions that transfer data between HI/LO and the general registers.

2.15.3 Jump and Branch Instructions

2.15.3.1 Types of Jump and Branch Instructions Defined by the ISA

The architecture defines the following jump and branch instructions:

- PC-relative conditional branch
- PC-region unconditional jump
- Absolute (register) unconditional jump
- A set of procedure calls that record a return link address in a general register.

2.15.3.2 Branch Delays and the Branch Delay Slot

All branches have an architectural delay of one instruction. The instruction immediately following a branch is said to be in the **branch delay slot**. If a branch or jump instruction is placed in the branch delay slot, the operation of both instructions is undefined.

By convention, if an exception or interrupt prevents the completion of an instruction in the branch delay slot, the instruction stream is continued by re-executing the branch instruction. To permit this, branches must be restartable; procedure calls may not use the register in which the return link is stored (usually GPR 31) to determine the branch target address.

2.15.3.3 Branch and Branch Likely

There are two versions of conditional branches; they differ in the manner in which they handle the instruction in the delay slot when the branch is not taken and execution falls through.

- Branch instructions execute the instruction in the delay slot.
- Branch likely instructions do not execute the instruction in the delay slot if the branch is not taken (they are said to nullify the instruction in the delay slot).

  Although the Branch Likely instructions are included in this specification, software is strongly encouraged to avoid the use of the Branch Likely instructions, as they will be removed from a future revision of the MIPS Architecture.

2.15.4 Miscellaneous Instructions

2.15.4.1 Instruction Serialization (SYNC and SYNCI)

In normal operation, the order in which load and store memory accesses appear to a viewer *outside* the executing processor (for instance, in a multiprocessor system) is not specified by the architecture.

The SYNC instruction can be used to create a point in the executing instruction stream at which the relative order of some loads and stores can be determined: loads and stores executed before the SYNC are completed before loads and stores after the SYNC can start.

The SYNCI instruction synchronizes the processor caches with previous writes or other modifications to the instruction stream.

2.15.4.2 Exception Instructions

Exception instructions transfer control to a software exception handler in the kernel. There are two types of exceptions, conditional and unconditional. These are caused by the following instructions: syscall, trap, and break.

Trap instructions, which cause conditional exceptions based upon the result of a comparison

System call and breakpoint instructions, which cause unconditional exceptions

2.15.4.3 Conditional Move Instructions

MIPS32 includes instructions to conditionally move one CPU general register to another, based on the value in a third general register.

2.15.4.4 NOP Instructions

The NOP instruction is actually encoded as an all-zero instruction. MIPS processors special-case this encoding as performing no operation, and optimize execution of the instruction. In addition, SSNOP instruction, takes up one issue cycle on any processor, including super-scalar implementations of the architecture.

2.15.5 Coprocessor Instructions

2.15.5.1 What Coprocessors Do

Coprocessors are alternate execution units, with register files separate from the CPU. In abstraction, the MIPS architecture provides for up to four coprocessor units, numbered 0 to 3. Each level of the ISA defines a number of these coprocessors. Coprocessor 0 is always used for system control and coprocessor 1 and 3 are used for the floating point unit. Coprocessor 2 is reserved for implementation-specific use.

A coprocessor may have two different register sets:

- Coprocessor general registers
- Coprocessor control registers

Each set contains up to 32 registers. Coprocessor computational instructions may use the registers in either set.

2.15.5.2 System Control Coprocessor 0 (CP0)

The system controller for all MIPS processors is implemented as coprocessor 0 (CP0), the **System Control Coprocessor**. It provides the processor control, memory management, and exception handling functions.

2.15.5.3 Coprocessor Load and Store Instructions

Explicit load and store instructions are not defined for CP0; for CP0 only, the move to and from coprocessor instructions must be used to write and read the CP0 registers. The loads and stores for the remaining coprocessors are summarized in “Coprocessor Loads and Stores” on page 60.

2.16 CPU INITIALIZATION

Software is required to initialize the following parts of the device after a Reset event.

2.16.1 General Purpose Registers

The CPU register file powers up in an unknown state with the exception of r0 which is always '0'. Initializing the rest of the register file is not required for proper operation in hardware. Depending on the software environment however, several registers may need to be initialized. Some of these are:

- sp – stack pointer
- gp – global pointer
- fp – frame pointer

2.16.2 Coprocessor 0 State

Miscellaneous CP0 states need to be initialized prior to leaving the boot code. There are various exceptions which are blocked by ERL = 1 or EXL = 1 and which are not cleared by Reset. These can be cleared to avoid taking spurious exceptions when leaving the boot code.

Table 2-13: CP0 Initialization

CP0 Register	Action
CAUSE	WP (Watch Pending), SW0/1 (Software Interrupts) should be cleared.
CONFIG	Typically, the K0, KU and K23 fields should be set to the desired Cache Coherency Algorithm (CCA) value prior to accessing the corresponding memory regions.
COUNT(1)	Should be set to a known value if Timer Interrupts are used.
COMPARE(1)	Should be set to a known value if Timer Interrupts are used. The write to compare will also clear any pending Timer Interrupts (Thus, Count should be set before Compare to avoid any unexpected interrupts).
STATUS	Desired state of the device should be set.
Other CP0 state	Other registers should be written before they are read. Some registers are not explicitly writable, and are only updated as a by-product of instruction execution or a taken exception. Uninitialized bits should be masked off after reading these registers.

Note 1: When the Count register is equal to the Compare register a timer interrupt is signaled. There is a mask bit in the interrupt controller to disable passing this interrupt to the CPU if desired.

2.16.3 Bus Matrix

The BMX should be initialized before switching to User mode or before executing from DRM. The values written to the bus matrix are based on the memory layout of the application to be run.

2.17 EFFECTS OF A RESET

2.17.1 $\overline{\text{MCLR}}$ Reset

The PIC32MX core is not fully initialized by hardware Reset. Only a minimal subset of the processor state is cleared. This is enough to bring the core up while running in unmapped and uncached code space. All other processor state can then be initialized by software. Power-up Reset brings the device into a known state. Soft Reset can be forced by asserting the $\overline{\text{MCLR}}$ pin. This distinction is made for compatibility with other MIPS processors. In practice, both Resets are handled identically with the exception of the setting of StatusSR.

2.17.1.1 Coprocessor 0 State

Much of the hardware initialization occurs in Coprocessor 0.

Table 2-14: Bits Cleared or Set by Reset

Bit Name	Cleared or Set	Value	By	Cleared or Set	Value	By
StatusBEV	Cleared	1	Reset or Soft Reset			
StatusTS	Cleared	0	Reset or Soft Reset			
StatusSR	Cleared	0	Reset	Set	1	Soft Reset
StatusNMI	Cleared	0	Reset or Soft Reset			
StatusERL	Set	1	Reset or Soft Reset			
StatusRP	Cleared	0	Reset or Soft Reset			
Configuration fields related to static inputs	Set	input value	Reset or Soft Reset			
ConfigK0	Set	010 (uncached)	Reset or Soft Reset			
ConfigKU	Set	010 (uncached)	Reset or Soft Reset			
ConfigK23	Set	010 (uncached)	Reset or Soft Reset			
DebugDM	Cleared	0	Reset or Soft Reset(1)			
DebugLSNM	Cleared	0	Reset or Soft Reset			
DebugIBusEP	Cleared	0	Reset or Soft Reset			
DebugIEXI	Cleared	0	Reset or Soft Reset			
DebugSSt	Cleared	0	Reset or Soft Reset			

Note 1: Unless EJTAGBOOT option is used to boot into DEBUG mode.

2.17.1.2 Bus State Machines

All pending bus transactions are aborted and the state machines in the SRAM interface unit are reset when a Reset or Soft Reset exception is taken.

2.17.2 Fetch Address

Upon Reset/SoftReset, unless the EJTAGBOOT option is used, the fetch is directed to VA 0xBFC00000 (PA 0x1FC00000). This address is in KSeg1, which is unmapped and uncached.

2.17.3 WDT Reset

The status of the CPU registers after a WDT event depends on the operational mode of the CPU prior to the WDT event.

If the device was not in Sleep a WDT event will force registers to a Reset value.

2.18 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the PIC32MX device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the CPU of the PIC32MX family include the following:

Title	Application Note #
No related application notes at this time.	N/A

Note: Please visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the PIC32MX family of devices.

2.19 REVISION HISTORY

Revision A (October 2007)

This is the initial released version of this document.

Revision B (April 2008)

Revised status to Preliminary; Revised Section 2.1 (Key Features); Revised Figure 2-1; Revised U-0 to r-x.

Revision C (May 2008)

Revise Figure 2-1; Added Section 2.2.3, Core Timer; Change Reserved bits from “Maintain as” to “Write”.

Section 3. Memory Organization

HIGHLIGHTS

This section of the manual contains the following topics:

3.1 INTRODUCTION

The PIC32MX microcontrollers provide 4 GB of unified virtual memory address space. All memory regions, including program memory, data memory, SFRs, and Configuration registers reside in this address space at their respective unique addresses. The program and data memories can be optionally partitioned into user and kernel memories. In addition, the data memory can be made executable, allowing the PIC32MX to execute from data memory.

Key features of PIC32MX memory organization include the following:

- 32-bit native data width
- Separate User and Kernel mode address spaces
- Flexible program Flash memory partitioning
- Flexible data RAM partitioning for data and program space
- Separate boot Flash memory for protected code
- Robust bus-exception handling to intercept runaway code
- Simple memory mapping with Fixed Mapping Translation (FMT) unit
- Cacheable and non-cacheable address regions

3.2 CONTROL REGISTERS

This section lists the Special Function Registers (SFRs) registers used for setting the RAM and Flash memory partitions for data and code (for both User and Kernel mode).

The following is a list of available SFRs:

- BMXCON: Configuration Register

  BMXCONCLR, BMXCONSET, BMXCONINV: Atomic Bit Manipulation Registers for BMXCON

- BMXxxxBA: Memory Partition Base Address Registers

  BMXxxxBACLR, BMXxxxBASET, BMXxxxBAINV: Atomic Bit Manipulation Registers for BMXxxxBA

- BMXDRMSZ: Data RAM Size Register
- BMXPFMSZ: Program Flash Size Register
- BMXBOOTSZ: Boot Flash Size Register

Table 3-1 provides a brief summary of all Memory Organization-related registers. Corresponding registers appear after the summary, followed by a detailed description of each register.

Table 3-1: Memory Organization SFR Summary

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
BMXCON	31:24	—	—	—	—	—	BMX-CHEDMA	—	—
	23:16	—	—	—	BMXER-RIXI	BMXER-RICD	BMXER-RDMA	BMXER-RDS	BMXERRIS
	15:8	—	—	—	—	—	—	—	—
	7:0	—	BMXWS-DRM	—	—	—	BMXARB		
BMXCONCLR	31:0	Write clears selected bits in BMXCON, read yields undefined value							
BMXCONSET	31:0	Write sets selected bits in BMXCON, read yields undefined value							
BMXCONINV	31:0	Write inverts selected bits in BMXCON, read yields undefined value							
BMXDKPBA	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	BMXDKPBA<15:8>							
	7:0	BMXDKPBA<7:0>							
BMXDKPBACLR	31:0	Write clears selected bits in BMXDKPBA, read yields undefined value							
BMXDKPBASET	31:0	Write sets selected bits in BMXDKPBA, read yields undefined value							
BMXDKPBAINV	31:0	Write inverts selected bits in BMXDKPBA, read yields undefined value							
BMXDUDBA	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	BMXDUDBA<15:8>							
	7:0	BMXDUDBA<7:0>							
BMXDUDBACLR	31:0	Write clears selected bits in BMXDUDBA, read yields undefined value							
BMXDUDBASET	31:0	Write sets selected bits in BMXDUDBA, read yields undefined value							
BMXDUDBAINV	31:0	Write inverts selected bits in BMXDUDBA, read yields undefined value							
BMXDUPBA	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	BMXDUPBA<15:8>							
	7:0	BMXDUPBA<7:0>							
BMXDUPBACLR	31:0	Write clears selected bits in BMXDUPBA, read yields undefined value							
BMXDUPBASET	31:0	Write sets selected bits in BMXDUPBA, read yields undefined value							
BMXDUPBAINV	31:0	Write inverts selected bits in BMXDUPBA, read yields undefined value							

Table 3-1: Memory Organization SFR Summary (Continued)

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
BMXDRMSZ	31:24	BMXDRMSZ<31:24>							
	23:16	BMXDRMSZ<23:16>							
	15:8	BMXDRMSZ<15:8>							
	7:0	BMXDRMSZ<7:0>							
BMXPUPBA	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	BMXPUPBA<19:16>			
	15:8	BMXPUPBA<15:8>							
	7:0	BMXPUPBA<7:0>							
BMXPUPBACLR	31:0	Write clears selected bits in BMXPUPBA, read yields undefined value							
BMXPUPBASET	31:0	Write sets selected bits in BMXPUPBA, read yields undefined value							
BMXPUPBAINV	31:0	Write inverts selected bits in BMXPUPBA, read yields undefined value							
BMXPFMSZ	31:24	BMXPFMSZ<31:24>							
	23:16	BMXPFMSZ<23:16>							
	15:8	BMXPFMSZ<15:8>							
	7:0	BMXPFMSZ<7:0>							
BMXBOOTSZ	31:24	BMXBOOTSZ<31:24>							
	23:16	BMXBOOTSZ<23:16>							
	15:8	BMXBOOTSZ<15:8>							
	7:0	BMXBOOTSZ<7:0>							

Register 3-1: BMXCON: Bus Matrix Configuration Register

r-x	r-x	r-x	r-x	r-x	R/W-0	r-x	r-x
—	—	—	—	—	BMX-CHEDMA	—	—
bit 31							bit 24

r-x	r-x	r-x	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
—	—	—	BMXERRIXI	BMXER-RICD	BMXER-RDMA	BMXER-RDS	BMXERRIS
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

r-x	R/W-1	r-x	r-x	r-x	R/W-0	R/W-0	R/W-0
—	BMXWS-DRM	—	—	—	BMXARB<2:0>		
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-27 **Reserved:** Write ‘0’; ignore read

bit 26 **BMXCHEDMA:** BMX PFM Cacheability for DMA Accesses bit
1 = Enable program Flash memory (data) cacheability for DMA accesses (requires cache to have data caching enabled)
0 = Disable program Flash memory (data) cacheability for DMA accesses (hits are still read from the cache, but misses do not update the cache)

bit 25 - 21 **Reserved:** Write ‘0’; ignore read

bit 20 **BMXERRIXI:** Enable Bus Error from IXI bit
1 = Enable bus error exceptions for unmapped address accesses initiated from IXI shared bus
0 = Disable bus error exceptions for unmapped address accesses initiated from IXI shared bus

bit 19 **BMXERRICD:** Enable Bus Error from ICD Debug Unit bit
1 = Enable bus error exceptions for unmapped address accesses initiated from ICD
0 = Disable bus error exceptions for unmapped address accesses initiated from ICD

bit 18 **BMXERRDMA:** Bus Error from DMA bit
1 = Enable bus error exceptions for unmapped address accesses initiated from DMA
0 = Disable bus error exceptions for unmapped address accesses initiated from DMA

bit 17 **BMXERRDS:** Bus Error from CPU Data Access bit (disabled in DEBUG mode)
1 = Enable bus error exceptions for unmapped address accesses initiated from CPU data access
0 = Disable bus error exceptions for unmapped address accesses initiated from CPU data access

bit 16 **BMXERRIS:** Bus error from CPU Instruction Access bit (disabled in DEBUG mode)
1 = Enable bus error exceptions for unmapped address accesses initiated from CPU instruction access
0 = Disable bus error exceptions for unmapped address accesses initiated from CPU instruction access

bit 15 - 7 **Reserved:** Write ‘0’; ignore read

Register 3-1: BMXCON: Bus Matrix Configuration Register (Continued)

bit 6	**BMXWSDRM:** CPU Instruction or Data Access from Data RAM Wait State bit 1 = Data RAM accesses from CPU have one wait state for address setup 0 = Data RAM accesses from CPU have zero wait states for address setup
bit 5-3	**Reserved:** Write '0'; ignore read
bit 2-0	**BMXARB<2:0>:** Bus Matrix Arbitration Mode bits 111...011 = Reserved (using these Configuration modes will produce undefined behavior) 010 = Arbitration Mode 2 001 = Arbitration Mode 1 000 = Arbitration Mode 0

Register 3-2: BMXCONCLR: BMXCON Clear Register

Write clears selected bits in BMXCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in BMXCON**

A write of '1' in one or more bit positions clears the corresponding bit(s) in BMXCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `BMXCONCLR = 0x00000101` will clear bits 15 and 0 in BMXCON register.

Register 3-3: BMXCONSET: BMXCON Set Register

Write sets selected bits in BMXCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in BMXCON**

A write of '1' in one or more bit positions sets the corresponding bit(s) in BMXCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `BMXCONSET = 0x00000101` will set bits 15 and 0 in BMXCON register.

Register 3-4: BMXCONINV: BMXCON Invert Register

Write inverts selected bits in BMXCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in BMXCON**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in BMXCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `BMXCONINV = 0x00000101` will invert bits 15 and 0 in BMXCON register.

Register 3-5: BMXDKPBA: Data RAM Kernel Program Base Address Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-0
BMXDKPBA<15:8>							
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
BMXDKPBA<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15-11 **BMXDKPBA<15:11>:** DRM Kernel Program Base Address bits
When non-zero, this value selects the relative base address for kernel program space in RAM

bit 10-0 **BMXDKPBA<10:0>:** Read-Only bits
Value is always ‘0’, which forces 2 KB increments

Register 3-6: BMXDKPBACLR: BMXDKPBA Clear Register

Write clears selected bits in BMXDKPBA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in BMXDKPBA**

A write of '1' in one or more bit positions clears the corresponding bit(s) in BMXDKPBA register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `BMXDKPBACLR = 0x00000101` will clear bits 15 and 0 in BMXDKPBA register.

Register 3-7: BMXDKPBASET: BMXDKPBA Set Register

Write sets selected bits in BMXDKPBA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in BMXDKPBA**

A write of '1' in one or more bit positions sets the corresponding bit(s) in BMXDKPBA register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `BMXDKPBASET = 0x00000101` will set bits 15 and 0 in BMXDKPBA register.

Register 3-8: BMXDKPBAINV: BMXDKPBA Invert Register

Write inverts selected bits in BMXDKPBA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in BMXDKPBA**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in BMXDKPBA register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `BMXDKPBAINV = 0x00000101` will invert bits 15 and 0 in BMXDKPBA register.

Register 3-9: BMXDUDBA: Data RAM User Data Base Address Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-0
BMXDUDBA<15:8>							
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
BMXDUDBA<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15-11 **BMXDUDBA<15:11>:** DRM User Data Base Address bits
When non-zero, the value selects the relative base address for User mode data space in RAM

Note: If non-zero, the value must be greater than BMXDKPBA.

bit 10-0 **BMXDUDBA<10:0>:** Read-Only bits
Value is always ‘0’, which forces 2 KB increments

Register 3-10: BMXDUDBACLR: BMXDUDBA Clear Register

Write clears selected bits in BMXDUDBA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in BMXDUDBA**

A write of '1' in one or more bit positions clears the corresponding bit(s) in BMXDUDBA register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `BMXDUDBACLR = 0x00000101` will clear bits 15 and 0 in BMXDUDBA register.

Register 3-11: BMXDUDBASET: BMXDUDBA Set Register

Write sets selected bits in BMXDUDBA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in BMXDUDBA**

A write of '1' in one or more bit positions sets the corresponding bit(s) in BMXDUDBA register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `BMXDUDBASET = 0x00000101` will set bits 15 and 0 in BMXDUDBA register.

Register 3-12: BMXDUDBAINV: BMXDUDBA Invert Register

Write inverts selected bits in BMXDUDBA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in BMXDUDBA**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in BMXDUDBA register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `BMXDUDBAINV = 0x00000101` will invert bits 15 and 0 in BMXDUDBA register.

Register 3-13: BMXDUPBA: Data RAM User Program Base Address Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-0
BMXDUPBA<15:8>							
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
BMXDUPBA<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15-11 **BMXDUPBA<15:11>:** DRM User Program Base Address bits
When non-zero, the value selects the relative base address for User mode program space in RAM

Note: If non-zero, BMXDUPBA must be greater than BMXDUDBA.

bit 10-0 **BMXDUPBA<10:0>:** Read-Only bits
Value is always ‘0’, which forces 2 KB increments

Register 3-14: BMXDUPBACLR: BMXDUPBA Clear Register

Write clears selected bits in BMXDUPBA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in BMXDUPBA**

A write of '1' in one or more bit positions clears the corresponding bit(s) in BMXDUPBA register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `BMXDUPBACLR = 0x00000101` will clear bits 15 and 0 in BMXDUPBA register.

Register 3-15: BMXDUPBASET: BMXDUPBA Set Register

Write sets selected bits in BMXDUPBA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in BMXDUPBA**

A write of '1' in one or more bit positions sets the corresponding bit(s) in BMXDUPBA register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `BMXDUPBASET = 0x00000101` will set bits 15 and 0 in BMXDUPBA register.

Register 3-16: BMXDUPBAINV: BMXDUPBA Invert Register

Write inverts selected bits in BMXDUPBA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in BMXDUPBA**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in BMXDUPBA register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `BMXDUPBAINV = 0x00000101` will invert bits 15 and 0 in BMXDUPBA register.

Register 3-17: BMXDRMSZ: Data RAM Size Register

R	R	R	R	R	R	R	R
BMXDRMSZ<31:24>							
bit 31							bit 24

R	R	R	R	R	R	R	R
BMXDRMSZ<23:16>							
bit 23							bit 16

R	R	R	R	R	R	R	R
BMXDRMSZ<15:8>							
bit 15							bit 8

R	R	R	R	R	R	R	R
BMXDRMSZ<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **BMXDRMSZ:** Data RAM Memory (DRM) Size bits

Static value that indicates the size of the Data RAM in bytes:

.....0x00002000 = device has 8 KB RAM

0x00004000 = device has 16 KB RAM

0x00008000 = device has 32 KB RAM

Register 3-18: BMXPUPBA: Program Flash (PFM) User Program Base Address Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	BMXPUPBA<19:16>			
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-0
BMXPUPBA<15:8>							
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
BMXPUPBA<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-20 Unimplemented: Read as ‘0’

bit 19-11 **BMXPUPBA<19:11>:** Program Flash (PFM) User Program Base Address bits

bit 10-0 **BMXPUPBA<10:0>:** Read-Only bits
Value is always ‘0’, which forces 2 KB increments

Register 3-19: BMXPUPBACLR: BMXPUPBA Clear Register

Write clears selected bits in BMXPUPBA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in BMXPUPBA**

A write of '1' in one or more bit positions clears the corresponding bit(s) in BMXPUPBA register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `BMXPUPBACLR = 0x00000101` will clear bits 15 and 0 in BMXPUPBA register.

Register 3-20: BMXPUPBASET: BMXPUPBA Set Register

Write sets selected bits in BMXPUPBA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in BMXPUPBA**

A write of '1' in one or more bit positions sets the corresponding bit(s) in BMXPUPBA register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `BMXPUPBASET = 0x00000101` will set bits 15 and 0 in BMXPUPBA register.

Register 3-21: BMXPUPBAINV: BMXPUPBA Invert Register

Write inverts selected bits in BMXPUPBA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in BMXPUPBA**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in BMXPUPBA register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `BMXPUPBAINV = 0x00000101` will invert bits 15 and 0 in BMXPUPBA register.

Register 3-22: BMXPFMSZ: Program Flash (PFM) Size Register

R	R	R	R	R	R	R	R
BMXPFMSZ<31:24>							
bit 31							bit 24

R	R	R	R	R	R	R	R
BMXPFMSZ<23:16>							
bit 23							bit 16

R	R	R	R	R	R	R	R
BMXPFMSZ<15:8>							
bit 15							bit 8

R	R	R	R	R	R	R	R
BMXPFMSZ<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **BMXPFMSZ:** Program Flash Memory (PFM) Size bits

Static value that indicates the size of the PFM in bytes:

0x00008000 = device has 32 KB Flash
.....0x00010000 = device has 64 KB Flash
0x00020000 = device has 128 KB Flash
0x00040000 = device has 256 KB Flash
0x00080000 = device has 512 KB Flash

Register 3-23: BMXBOOTSZ: Boot Flash (IFM) Size Register

R	R	R	R	R	R	R	R
BMXBOOTSZ<31:24>							
bit 31							bit 24

R	R	R	R	R	R	R	R
BMXBOOTSZ<23:16>							
bit 23							bit 16

R	R	R	R	R	R	R	R
BMXBOOTSZ<15:8>							
bit 15							bit 8

R	R	R	R	R	R	R	R
BMXBOOTSZ<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

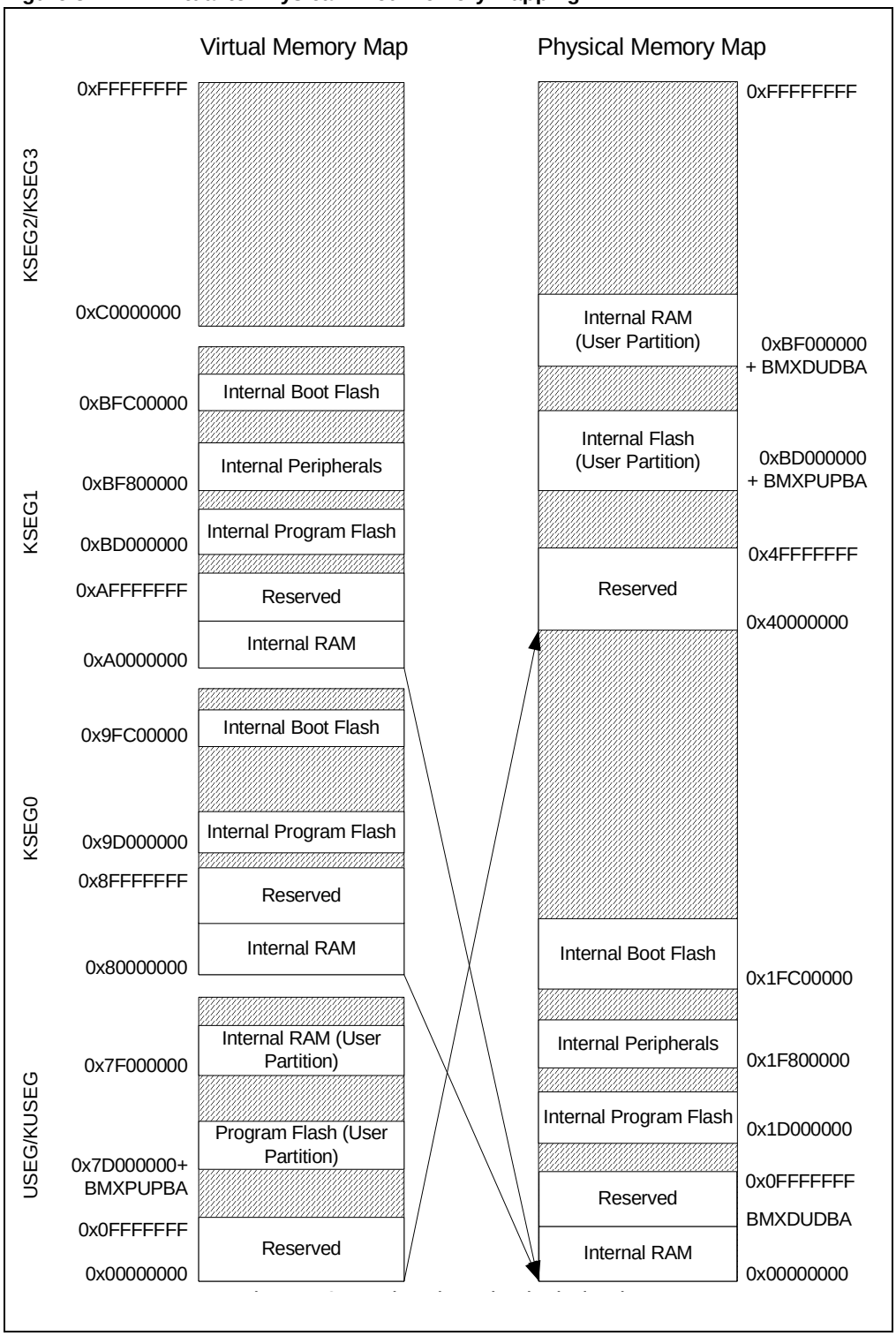
bit 31-0 **BMXBOOTSZ:** Boot Flash Memory (BFM) Size bits

Static value that indicates the size of the Boot PFM in bytes:
0x00003000 = device has 12 KB boot Flash

3.3 PIC32MX MEMORY LAYOUT

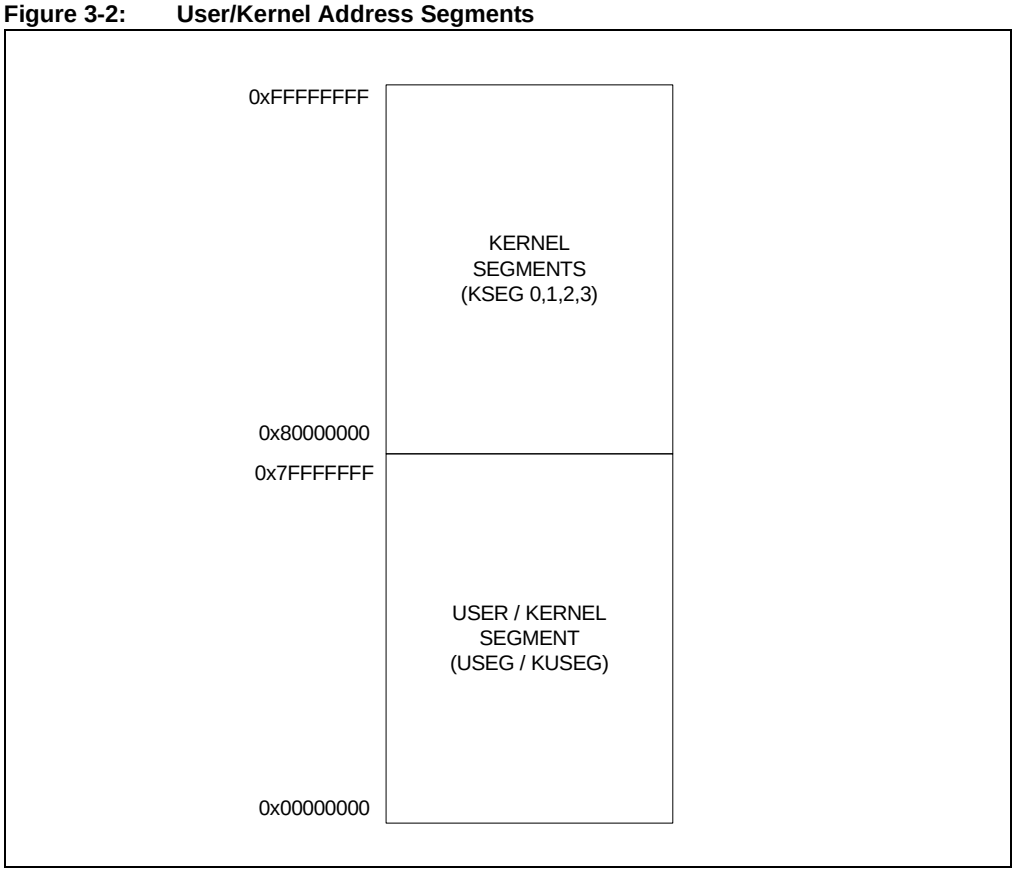
The PIC32MX microcontrollers implement two address spaces: virtual and physical. All hardware resources, such as program memory, data memory and peripherals, are located at their respective physical addresses. Virtual addresses are exclusively used by the CPU to fetch and execute instructions. Physical addresses are used by peripherals, such as DMA and Flash controllers, that access memory independently of the CPU.

Figure 3-1: Virtual to Physical Fixed Memory Mapping



The entire 4 GB virtual address space is divided into two primary regions: user and kernel space. The lower 2 GB of space from the User mode segment is called useg/kuseg. A User mode application must reside and execute in the useg segment. The useg segment is also available to all Kernel mode applications, which is why it is also named kuseg – to indicate that it is available to both User and Kernel modes. When operating in User mode, the bus matrix must be configured to make part of the Flash and data memory available in the useg/kuseg segment. See Section 3.4 for more information.

Figure 3-2: User/Kernel Address Segments



The upper 2 GB of virtual address space forms the kernel only space. The kernel space is divided into four segments of 512 MB each: kseg 0, kseg 1, kseg 2 and kseg 3. Only Kernel mode applications can access kernel space memory. The kernel space includes all peripheral registers. Consequently, only Kernel mode applications can monitor and manipulate peripherals. Only kseg 0 and kseg 1 segments point to real memory resources. Segment kseg 2 is available to the EJTAG probe debugger, as explained in the MIPS documentation (refer to the EJTAG specification). The PIC32MX only uses kseg 0 and kseg 1 segments. The Boot Flash Memory (BFM), Program Flash Memory (PFM), Data RAM Memory (DRM), and peripheral SFRs are accessible from either kseg 0 or kseg 1.

The Fixed Mapping Translation (FMT) unit translates the memory segments into corresponding physical address regions. Figure 3-1 shows the fixed mapping scheme implemented by the PIC32MX core between the virtual and physical address space. A virtual memory segment may also be cached, provided the cache module is available on the device. Please note that the kseg-1 memory segment is not cacheable, while kseg-0 and useg/kuseg are cacheable.

The mapping of the memory segments depend on the CPU error level (set by the ERL bit in the CPU Status register). Error Level is set (ERL = 1) by the CPU on a Reset, Soft Reset, or NMI. In this mode, the processor runs in Kernel mode and useg/kuseg are treated as unmapped and uncached regions, and the mapping in Figure 3-1 does not apply. This mode is provided for compatibility with other MIPS processor cores that use a TLB-based MMU. The C start-up code clears the ERL bit to zero, so that when application software starts up, it sees the proper virtual to physical memory mapping as depicted in Figure 3-1.

Segments kseg 0 and kseg 1 are always translated to physical address 0x0. This translation arrangement allows the CPU to access identical physical addresses from two separate virtual addresses: one from kseg 0 and the other from kseg 1. As a result, the application can choose to execute the same piece of code as either cached or uncached. See **Section 4. "Prefetch Cache Module"** for more information. The on-chip peripherals are visible through kseg 1 segment only (uncached access).

3.4 PIC32MX ADDRESS MAP

The Program Flash Memory is divided into kernel and user partitions. The kernel program Flash space starts at physical address 0x1D000000, whereas the user program Flash space starts at physical address 0xBD000000 + BMXPUDBA register value. Similarly, the internal RAM is also divided into kernel and user partitions. The kernel RAM space starts at physical address 0x00000000, whereas the user RAM space starts at physical address 0xBF000000 + BMXDUDBA register value. By default, the full Flash memory and RAM are mapped to Kernel mode application only.

Please note that the BMXxxxBA register settings must match the memory model of the target software application. If the linked code does not match the register values, the program may not run and may generate bus error exceptions on start-up.

Note:	The Program Flash Memory is not writable through its address map. A write to the PFM address range causes a bus error exception.

3.4.1 Virtual to Physical Address Calculation (and Vice-Versa)

To translate the kernel address (KSEG0 or KSEG1) to a physical address, perform a “Bitwise AND” operation of the virtual address with 0x1FFFFFFF:

Physical Address = Virtual Address and 0x1FFFFFFF

For physical address to KSEG0 virtual address translation, perform a “Bitewise OR” operation of the physical address with 0x80000000:

KSEG0 Virtual Address = Physical Address | 0x80000000

For physical address to KSEG1 virtual address translation, perform a “Bitewise OR” operation of the physical address with 0xA0000000:

KSEG1 Virtual Address = Physical Address | 0xA0000000

To translate from KSEG0 to KSEG1 virtual address, perform a “Bitewise OR” operation of the KSEG0 virtual address with 0x20000000:

KSEG1 Virtual Address = KSEG0 Virtual Address | 0x20000000

Table 3-2: PIC32MX Address Map

		Virtual Addresses		Physical Addresses		Size in Bytes
	Memory Type	**Begin Address**	**End Address**	**Begin Address**	**End Address**	**calculation**
Kernel Address Space	**Boot Flash**	0xBFC00000	0xBFC02FFF	0x1FC00000	0x1FC02FFF	12 KB
	Program Flash(1)	0xBD000000	0xBD000000 + BMXPUPBA - 1	0x1D000000	0x1D00000 + BMXPUPBA - 1	BMXPUPBA
	Program Flash(2)	0x9D000000	0x9D000000 + BMXPUPBA - 1	0x1D000000	0x1D000000 + BMXPUPBA - 1	BMXPUPBA
	RAM (Data)	0x80000000	0x80000000 + BMXDKPBA - 1	0x00000000	BMXDKPBA - 1	BMXDKPBA
	RAM (Prog)	0x80000000 + BMXDKPBA	0x80000000 + BMXDUDBA -1	BMXDKPBA	BMXDUDBA -1	BMXDUDBA - BMXDKPBA
	Peripheral	0xBF800000	0xBF8FFFFF	0x1F800000	0x1F8FFFFF	1 MB
User Address Space	**Program Flash**	0x7D000000 + BMXPUPBA	0x7D000000 + PFM Size - 1	0xBD000000 + BMXPUPBA	0xBD000000 + PFM Size - 1	PFM Size - BMXPUPBA
	RAM (Data)	0x7F000000 + BMXDUDBA	0x7F000000 + BMXDUPBA - 1	0xBF000000 + BMXDUDBA	0xBF000000 + BMXDUPBA - 1	BMXDUPBA - BMXDUDBA
	RAM (Prog)	0x7F000000 + BMXDUPBA	0x7F000000 + RAM Size(3) - 1	0xBF000000 + BMXDUPBA	0xBF000000 + RAM Size(3) - 1	DRM Size - BMXDUPBA

Note 1: Program Flash virtual addresses in the non-cacheable range (KSEG1).

2: Program Flash virtual addresses in the cacheable and prefetchable range (KSEG0).

3: The RAM size varies between PIC32MX device variants.

3.4.2 Program Flash Memory Partitioning

The Program Flash Memory can be partitioned for User and Kernel mode programs as shown in Figure 3-1.

At Reset, the User mode partition does not exist (BMXPUPBA is initialized to 0). The entire Program Flash Memory is mapped to Kernel mode program space starting at virtual address KSEG1: 0xBD000000 (or KSEG0: 0x9D000000). To set up a partition for the User mode program, initialize BMXPUPBA as follows:

BMXPUPBA = BMXPFMSZ – USER_FLASH_PGM_SZ

The USER_FLASH_PGM_SZ is the partition size of the User mode program. BMXPFMSZ is the bus matrix register that holds the total size of Program Flash Memory.

Example:

Assuming the PIC32MX device has 512 Kbytes of Flash memory, the BMXPFMSZ will contain 0x00080000.

To create a user Flash program partition of 20 Kbytes (0x5000):

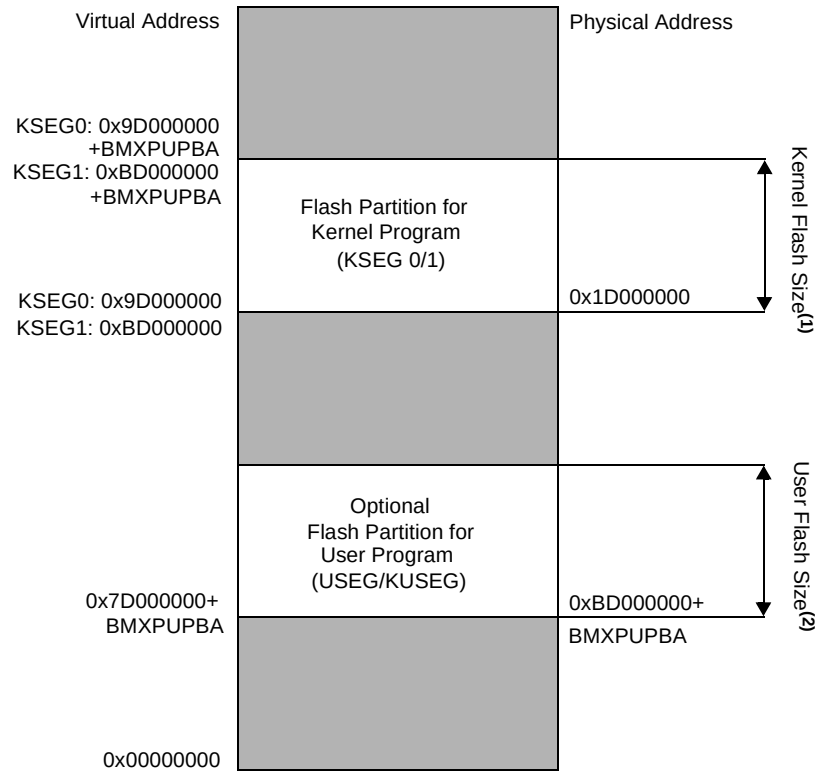
BMXPUPBA = 0x80000 – 0x5000 = 0x7B000

The size of the user Flash will be 20K and the size left for the Kernel Flash will be 512k – 20k = 492K.

The user Flash partition will extend from 0x7D07B000 to 0x7D07FFFF (virtual addresses).

The Kernel mode partition always starts from KSEG1: 0xBD000000 or KSEG0: 0x9D000000. In the above example, the Kernel partition will extend from 0xBD000000 to 0xBD07AFFF (492 Kbytes in size).

Figure 3-3: Flash Partitioning



Note 1: Kernel Flash Size = BMXPUPBA

2: User Flash Size = BMXPFMSZ-BMXPUPBA

3: If BMXPUPBA is '0', then:
K Flash Size = BMXPFMSZ (i.e., all the Flash)
Usr Flash Size = 0

3.4.3 RAM Partitioning

The RAM memory can be divided into 4 partitions. These are:

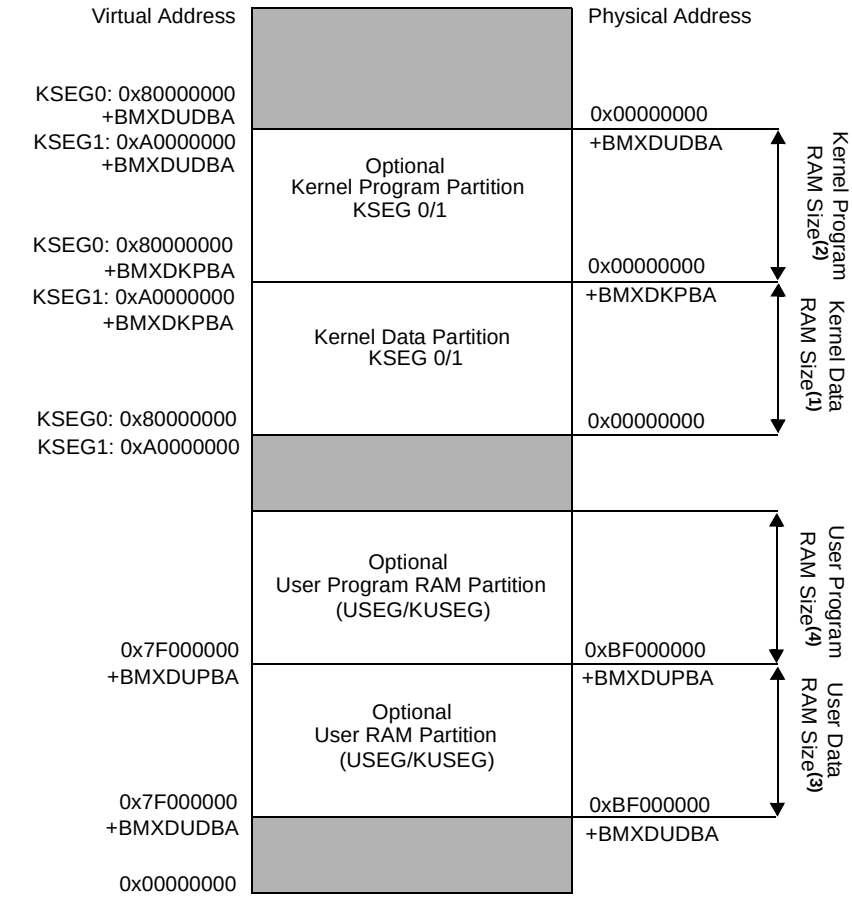
1. Kernel Data
2. Kernel Program
3. User Data
4. User Program

In order to execute from data RAM, a kernel or user program partition must be defined. At Power-on Reset, the entire data RAM is assigned to the kernel data partition. This partition always starts from the base of the data RAM. See Figure 3-4 for details.

Note 1: To properly partition the RAM, you have to program all of the following registers: BMXDKPBA, BMXDUDBA and BMXDUPBA.

2: The size of the available RAM is given by the BMXDRMSZ register.

Figure 3-4: RAM Partitioning



Note 1: Kernel Data RAM Size = BMXDKPBA

2: Kernel Program RAM Size = BMXDUDBA – BMXDKPBA

3: User Data RAM Size = BMXDUPBA – BMXDUDBA

4: User Program RAM Size = DRM Size – BMXDUPBA

5: If BMXDKPBA, BMXDUDBA or BMXDUPBA is '0', then:
Kernel Data RAM Size = BMXDRMSZ (i.e., all RAM)
Kernel Program RAM Size = 0
User Data RAM Size = 0
User Program RAM Size = 0

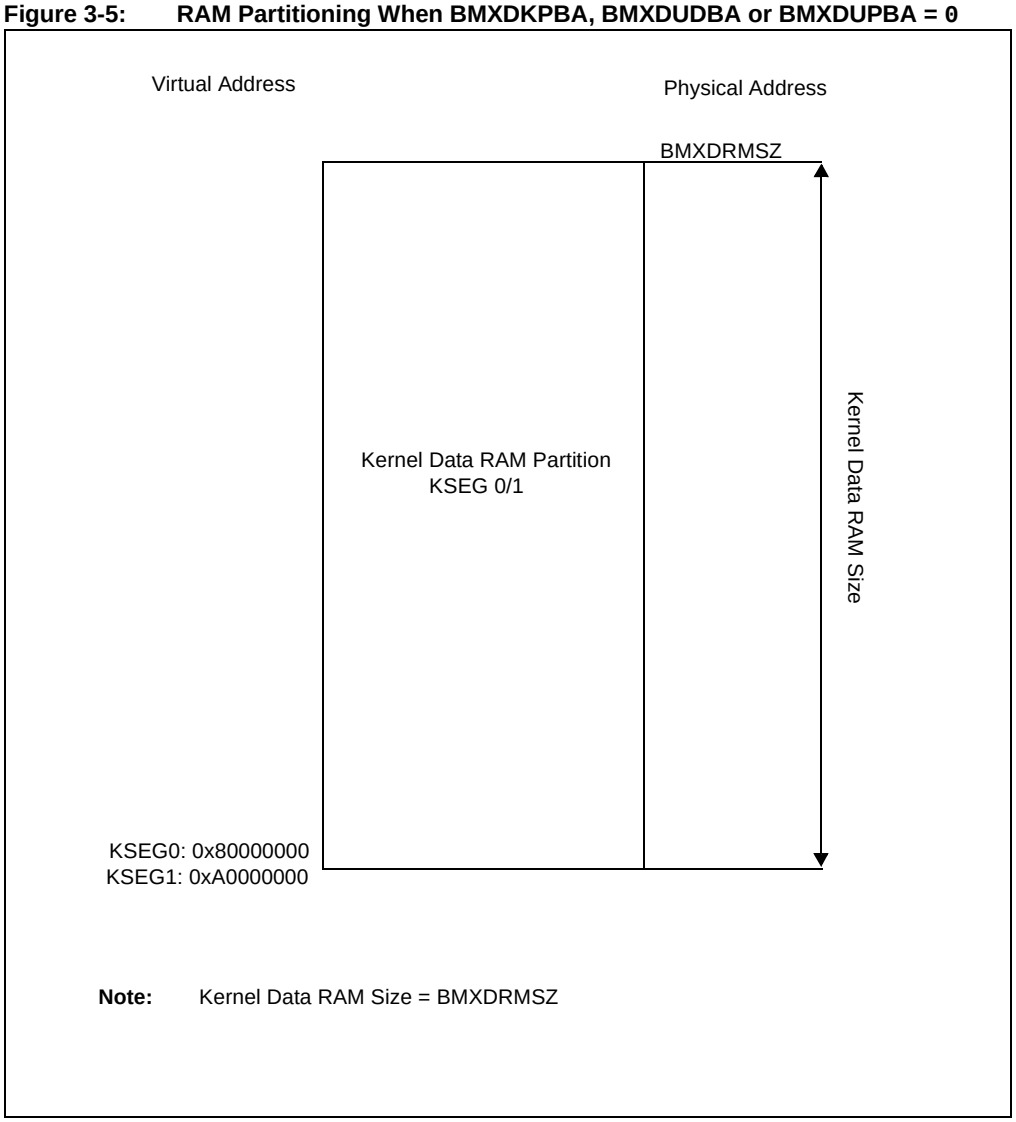
3.4.3.1 Kernel Data RAM Partition

The kernel data RAM partition is located at virtual address KSEG0:0x80000000, KSEG1:0xA0000000. It is always active and cannot be disabled.

Please note that if any of the BMXDKPBA, BMXDUDBA or BMXDUPBA register is '0', then the whole RAM is assigned to kernel data RAM (i.e., the size of the kernel data RAM partition is given by the BMXDRMSZ register value; see Figure 3-5). Otherwise, the size of the kernel data RAM partition is given by the value of the BMXDKPBA register. See Figure 3-6.

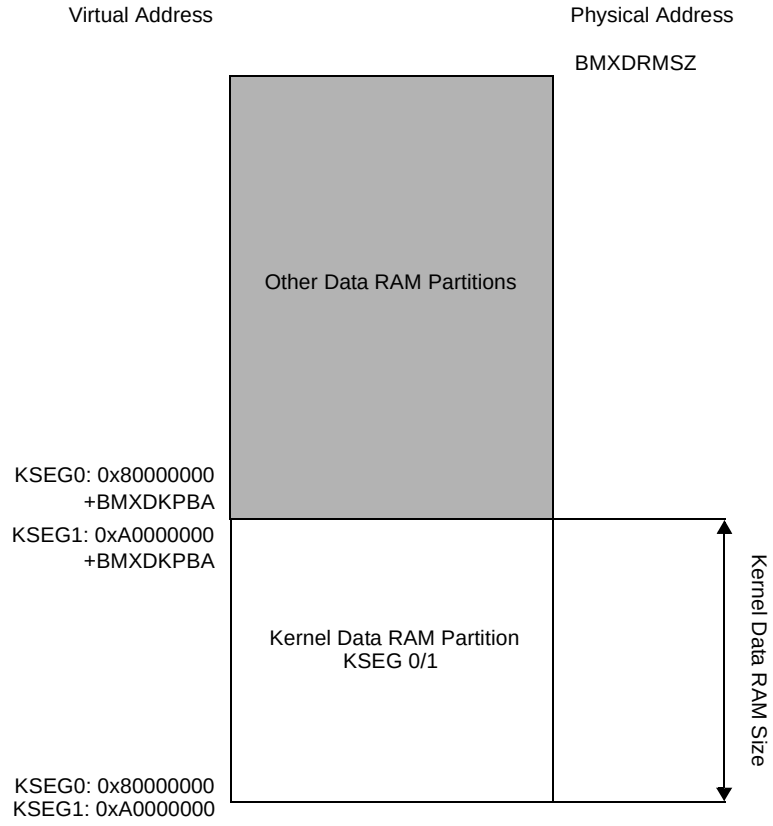
The kernel data RAM partition exists on Reset and takes up all the available RAM, as the BMXDKPBA, BMXDUDBA and BMXDUPBA registers default to zero at any Reset.

Figure 3-5: RAM Partitioning When BMXDKPBA, BMXDUDBA or BMXDUPBA = 0



Note: Kernel Data RAM Size = BMXDRMSZ

Figure 3-6: Kernel Data RAM Partitioning



Note 1: Kernel Data RAM Size = BMXDKPBA.

2: None of the registers BMXDKPBA, BMXDUDBA or BMXDUPBA = 0.

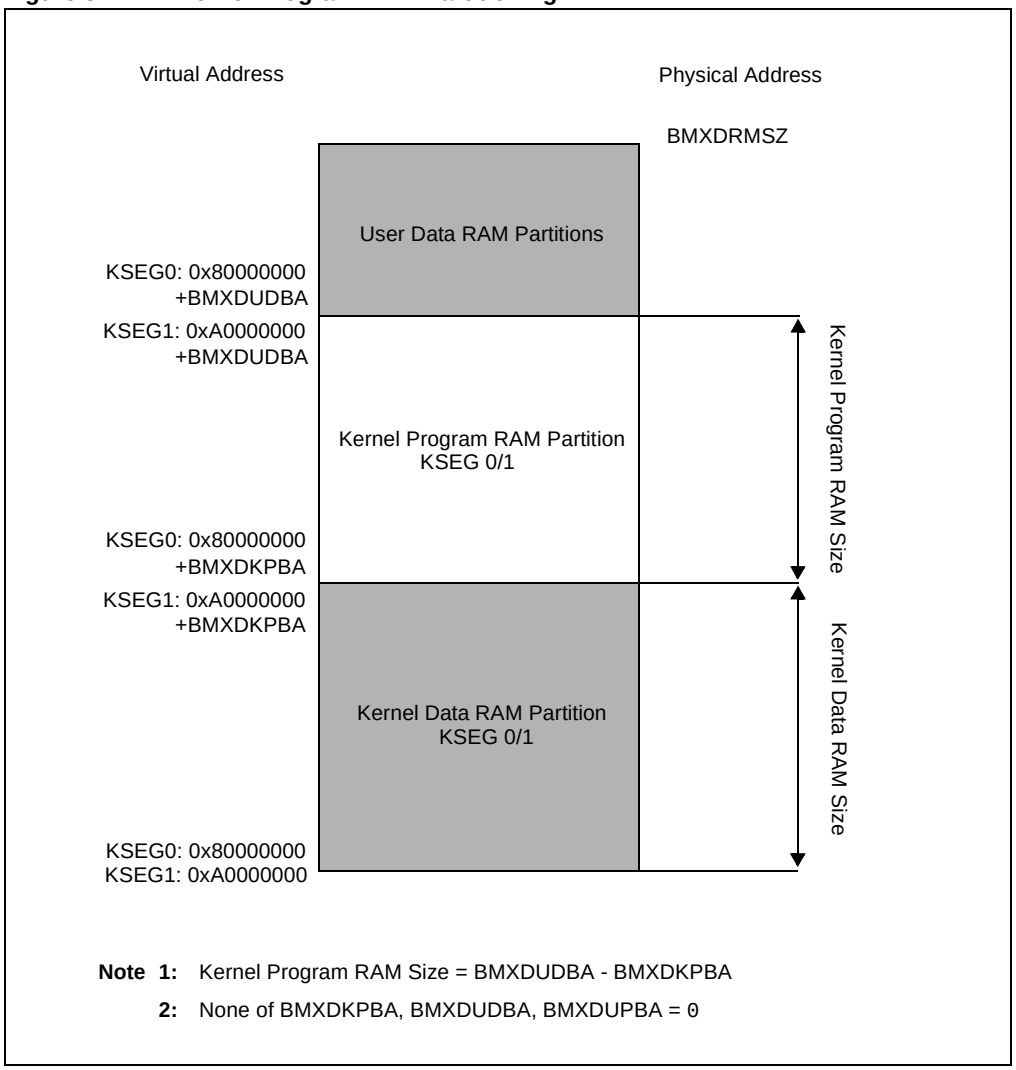
3.4.3.2 Kernel Program RAM Partition

The kernel program RAM partition is required if code needs to be executed from data RAM in Kernel mode.

This partition starts at KSEG0:0x80000000 + BMXDKPBA (KSEG1:0xA0000000 + BMXDKPBA), and its size is given by BMXDUDBA – BMXDKPBA. See Figure 3-7.

The kernel program RAM partition does not exist on Reset, as the BMXDKPBA and BMXDUDBA registers default to zero at Reset.

Figure 3-7: Kernel Program RAM Partitioning



Note 1: Kernel Program RAM Size = BMXDUDBA - BMXDKPBA

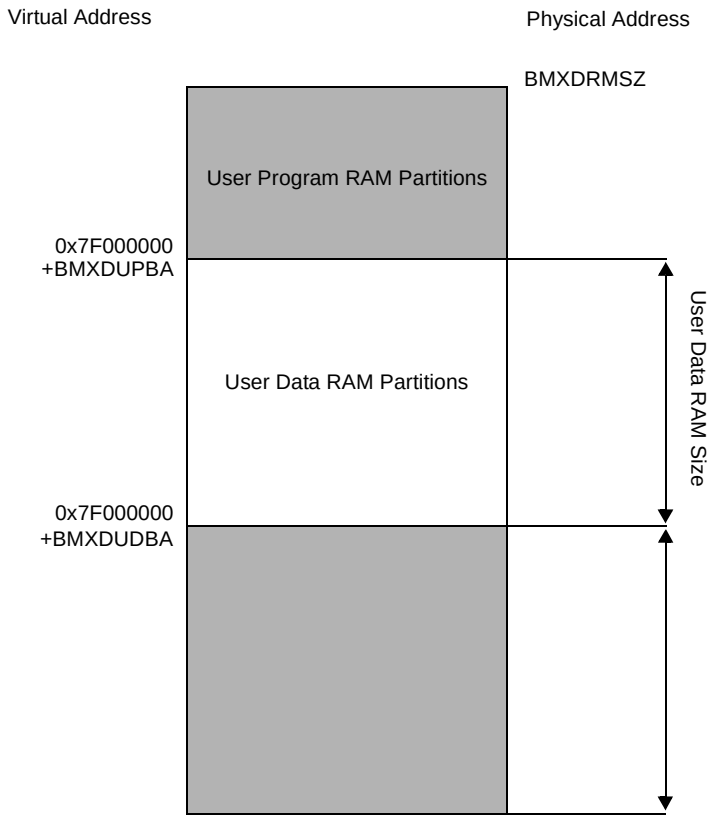
2: None of BMXDKPBA, BMXDUDBA, BMXDUPBA = 0

3.4.3.3 User Data RAM Partition

For User mode applications, a User mode data partition in RAM is required. This partition starts at address 0x7F000000 + BMXDUDBA, and its size is given by BMXDUPBA – BMXDUDBA. See Figure 3-8.

The user data RAM partition does not exist on Reset, as the BMXDUDBA and BMXDUPBA registers default to zero at Reset.

Figure 3-8: User Data RAM Partitioning



Note 1: User Data RAM Size = BMXDUPBA – BMXDUDBA.

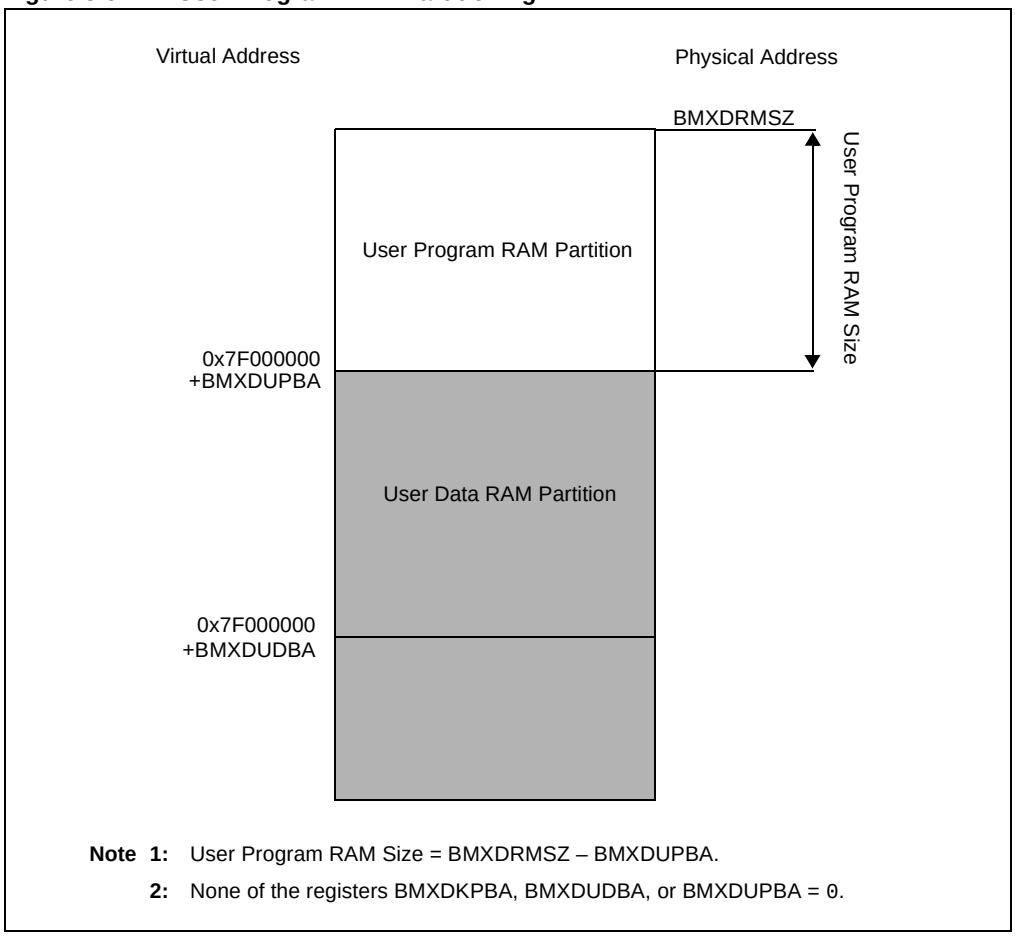
2: None of the registers BMXDKPBA, BMXDUDBA, or BMXDUPBA = 0.

3.4.3.4 User Program RAM Partition

The user program partition in data RAM is required if code needs to be executed from data RAM in User mode. This partition starts at address 0x7F000000 + BMXDUPBA, and its size is given by BMXDRMSZ – BMXDUPBA. See Figure 3-9.

The User Program RAM partition does not exist on Reset, as the BMXDUPBA register defaults to zero at Reset.

Figure 3-9: User Program RAM Partitioning



Note 1: User Program RAM Size = BMXDRMSZ – BMXDUPBA.

2: None of the registers BMXDKPBA, BMXDUDBA, or BMXDUPBA = 0.

3.4.3.5 RAM Partitioning Examples

This section provides the following practical examples of RAM partitioning.

1. RAM Partitioned as Kernel Data
2. RAM Partitioned as Kernel Data and Kernel Program
3. RAM Partitioned as Kernel Data and User Data
4. RAM Partitioned as Kernel Data, Kernel Program and User Data
5. RAM Partitioned as Kernel Data, Kernel Program, User Data and User Program

Example 1. RAM Partitioned as Kernel Data

The entire RAM is partitioned as kernel data RAM after a Reset. No other programming is required. Setting the BMXDKPBA, BMXDUDBA, or BMXDUPBA register to '0' will partition the entire RAM space to a kernel data partition. See Figure 3-5.

Example 2. RAM Partitioned as Kernel Data and Kernel Program

For this example, assume that the available RAM on the PIC32MX device is 32 KB, of which 8 KB kernel data RAM and 24 KB of kernel program RAM are needed. In this example, the user data RAM and user program RAM will have their sizes set to '0'.

Please note that a kernel data RAM partition is always required. See Figure 3-10 for details.

The values of the registers are as follows:

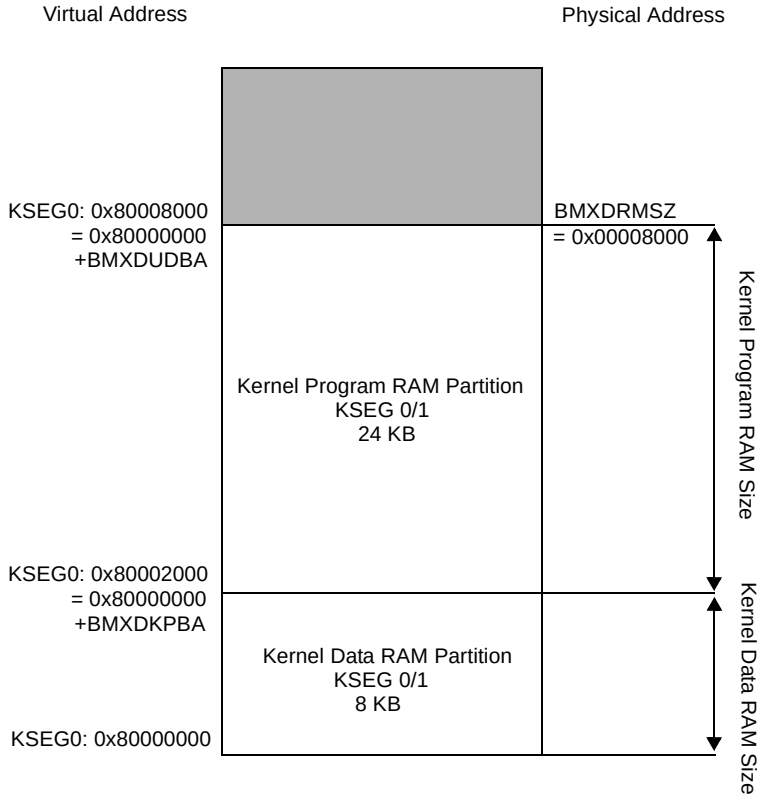
BMXDRMSZ = 0x00008000 (read-only value)

BMXDKPBA = 0x00002000 (i.e., 8 KB kernel data)

BMXDUDBA = 0x00008000 (i.e., 0x6000 kernel program)

BMXDUPBA = 0x00008000 (i.e., user data size = 0, and user program size = 0)

Figure 3-10: RAM Partitioning for 8 KB Kernel Data and 16 KB Kernel Program



Note: Only KSEG0 addresses are shown. For KSEG1 addresses, start at 0xA000000.

Example 3. RAM Partitioned as Kernel Data and User Data

For this example, assume that the available RAM on the PIC32MX device is 32 KB, of which 16 KB of kernel data RAM and 16 KB of user data RAM are needed. In this example, the kernel program RAM and user program RAM will have their sizes set to '0'. See Figure 3-11 for details.

The values of the registers are as follows:

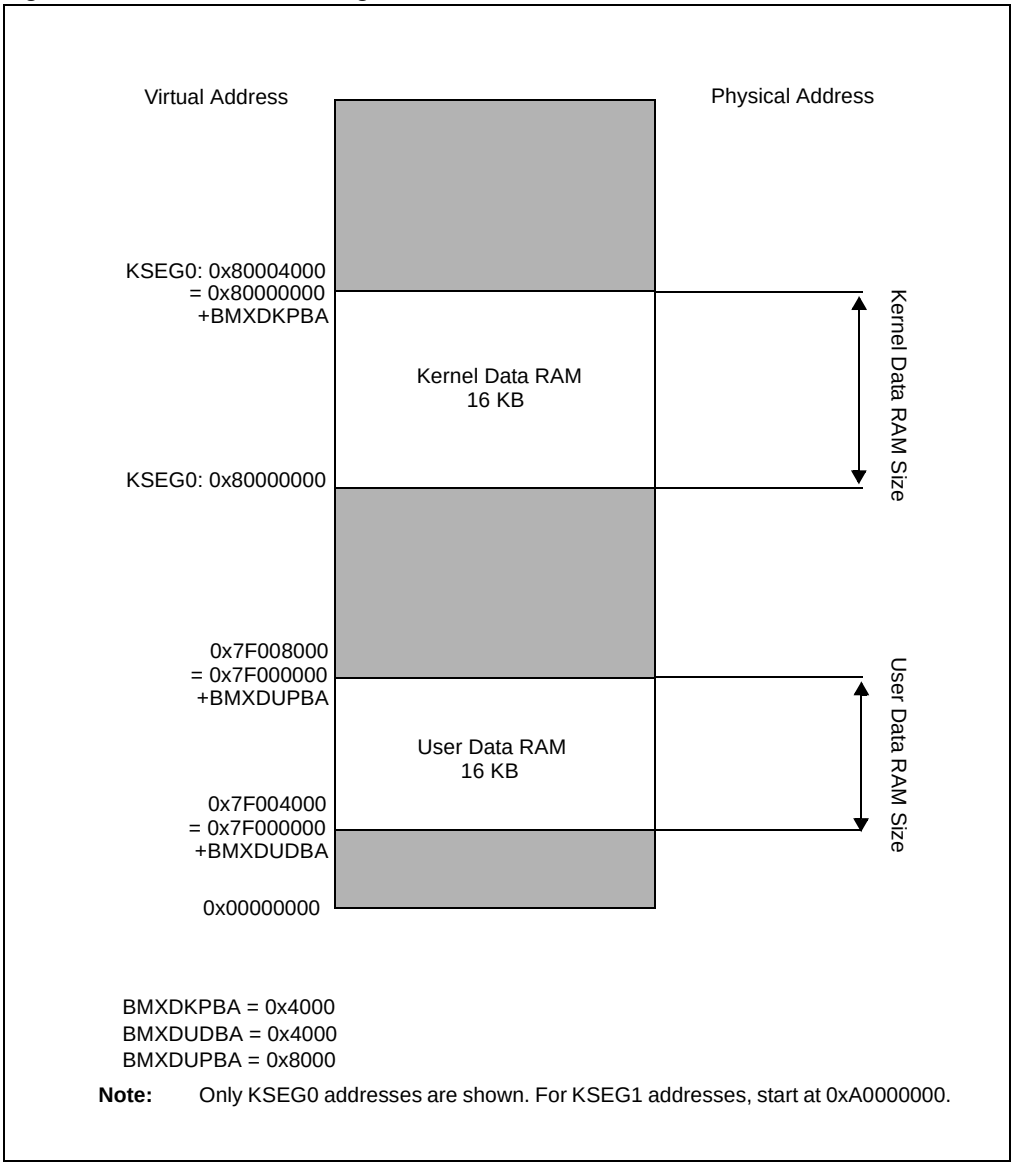
BMXDRMSZ = 0x00008000 (read-only value)

BMXDKPBA = 0x00004000 (i.e., 16 KB kernel data)

BMXDUDBA = 0x00004000 (i.e., 0 kernel program)

BMXDUPBA = 0x00008000 (i.e., user data size = 16 KB, and user program size = 0)

Figure 3-11: RAM Partitioning for 16 KB Kernel Data and 16 KB User Data



Note: Only KSEG0 addresses are shown. For KSEG1 addresses, start at 0xA0000000.

Example 4. RAM Partitioned as Kernel Data, Kernel Program and User Data

For this example, assume that the available RAM on the PIC32MX device is 32 KB, and 4 KB of kernel data RAM, 6 KB of kernel program and 22 KB of user data RAM are needed. In this example, the user program RAM will have its size set to '0'. See Figure 3-12 for details.

The values of the registers are as follows:

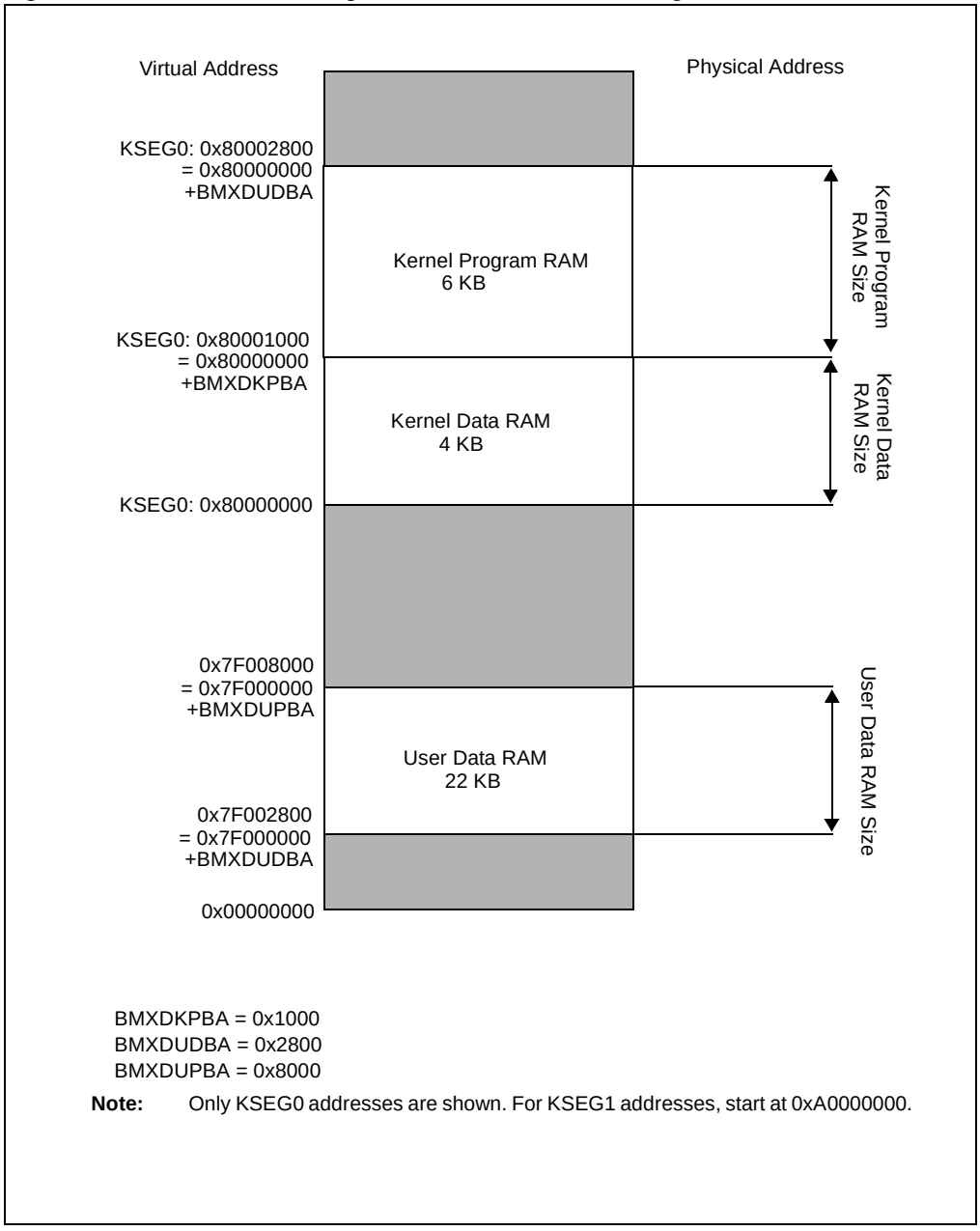
BMXDRMSZ = 0x00008000 (read-only value)

BMXDKPBA = 0x00001000 (i.e., 4 KB kernel data)

BMXDUDBA = 0x00002800 (i.e., 6 KB kernel program)

BMXDUPBA = 0x00008000 (i.e., user data size = 22 KB, and user program size = 0)

Figure 3-12: RAM Partitioning for 4 KB K-Data, 6 KB K-Program and 22 KB U-Data



Note: Only KSEG0 addresses are shown. For KSEG1 addresses, start at 0xA0000000.

Example 5. RAM Partitioned as Kernel Data, Kernel Program, User Data and User Program

For this example, assume that the available RAM on the PIC32MX device is 32 KB, and 6 KB of kernel data RAM, 5 KB of kernel program RAM, 12 KB of user data RAM and 9 KB of user program RAM are needed. See Figure 3-13 for details.

The values of the registers are as follows:

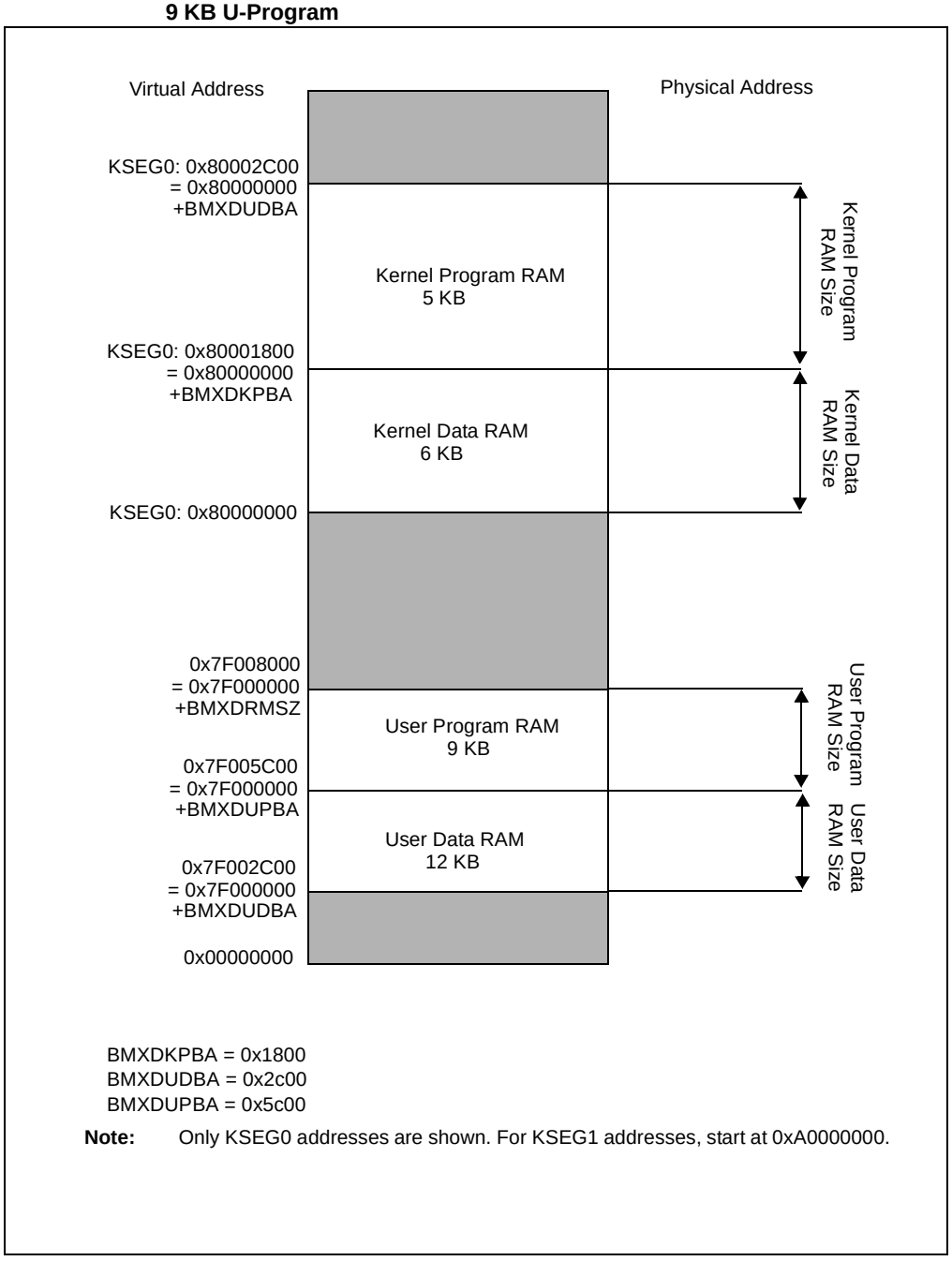
BMXDRMSZ = 0x00008000 (read-only value)

BMXDKPBA = 0x00001800 (i.e., 6 KB kernel data)

BMXDUDBA = 0x00002C00 (i.e., 5 KB kernel program)

BMXDUPBA = 0x00005C00 (i.e., user data size = 12 KB, and user program size = 9 KB)

Figure 3-13: RAM Partitioning for 6 KB K-Data, 5 KB K-Program, 12 KB U-Data and 9 KB U-Program



Note: Only KSEG0 addresses are shown. For KSEG1 addresses, start at 0xA0000000.

3.5 BUS MATRIX

The processor supports two modes of operation, Kernel mode and User mode. The Bus Matrix controls the allocation of memory for each of these modes. It also controls the type of access, program or data, for a given region of address space.

The Bus Matrix connects master devices, generically called initiators, to slave devices, generically called targets. The PIC32MX product family can have up to five initiators and three targets (e.g., Flash, RAM, ...) on the main bus structure.

Of the five possible initiators, the CPU Instruction Bus (CPU IS), CPU Data Bus (CPU DS), In-Circuit Debug (ICD) and DMA Controller (DMA) are the default set of initiators and are always present. The PIC32MX also includes an Initiator Expansion Interface (IXI) to support additional initiators for future expansion.

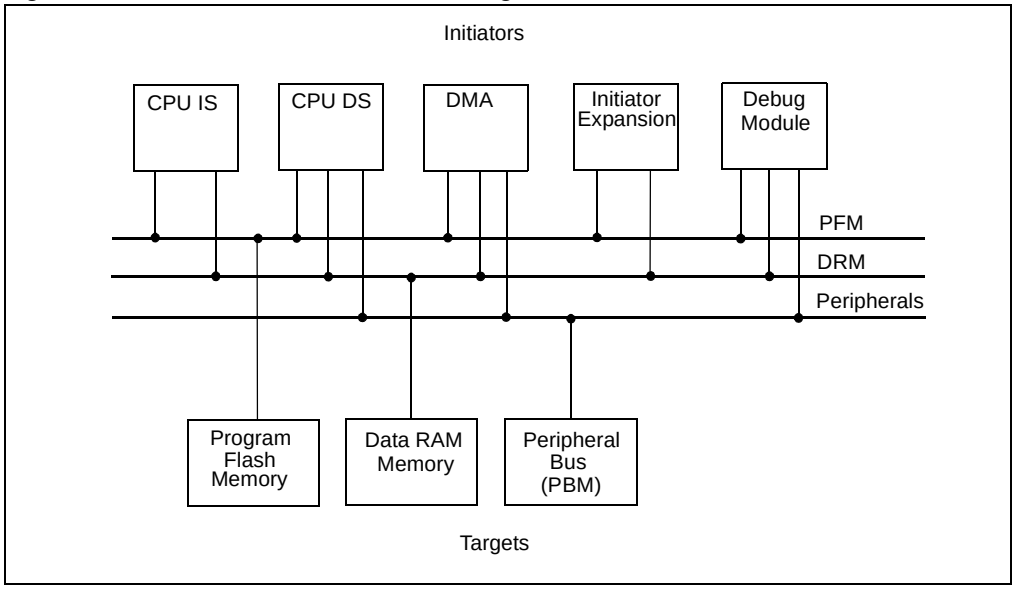
The Bus Matrix decodes a general range of addresses that map to a target. The target (memory or peripherals) may provide additional addresses depending on its functionality.

Table 3-3 shows which initiators can access which targets.

Table 3-3: Initiator Access Map

		Target		
		Flash	**RAM**	**Peripheral Bus**
Initiator	**CPU IS**	Y	Y	N
	CPU DS	Y	Y	Y
	DMA	Y	Y	Y
	IXI	Y	Y	N
	ICD	Y	Y	Y

Figure 3-14: Bus Matrix Initiators and Targets



3.5.1 Initiator Arbitration Modes

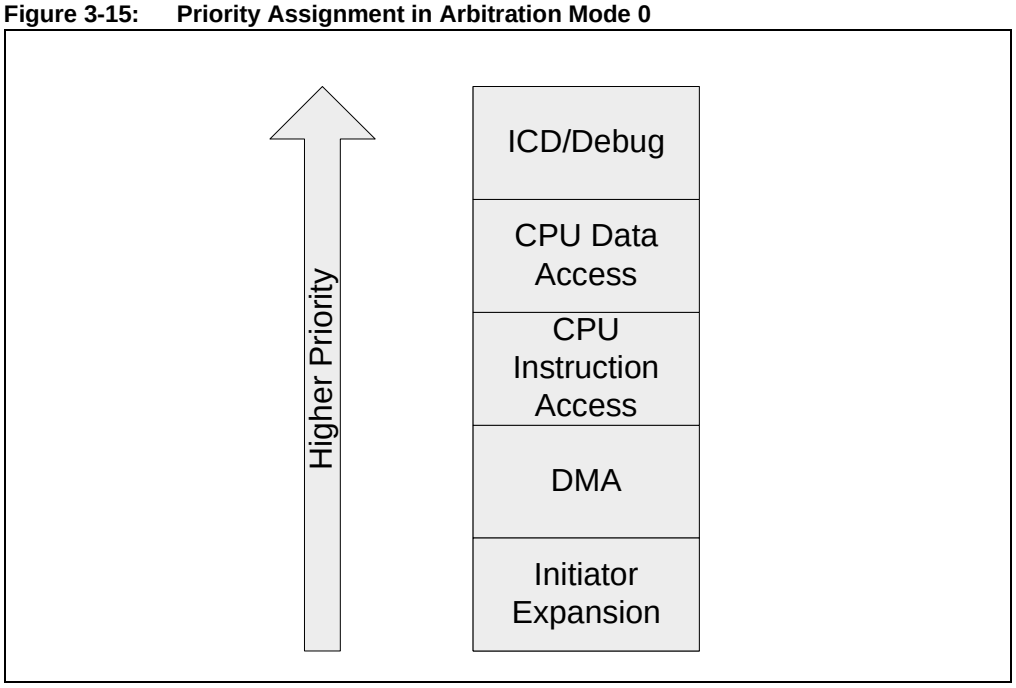
Since there can be more than one initiator attempting to access the same target, an arbitration scheme must be used to control access to the target. The arbitration modes assign priority levels to all the initiators. The initiator with the higher priority level will always win target access over a lower priority initiator.

3.5.1.1 Arbitration Mode 0

The fixed priority scheme in Arbitrition Mode 0 is shown in Figure 3-15. The CPU data and instruction access are given higher priority than DMA access. This mode can starve the DMA, so chose this mode when DMA is not being used.

As shown in Figure 3-15, each initiator is assigned a fixed priority level. Programming the register field BMXARB (BMXCON<2:0>) to '0' selects Mode 0 operation.

Figure 3-15: Priority Assignment in Arbitration Mode 0

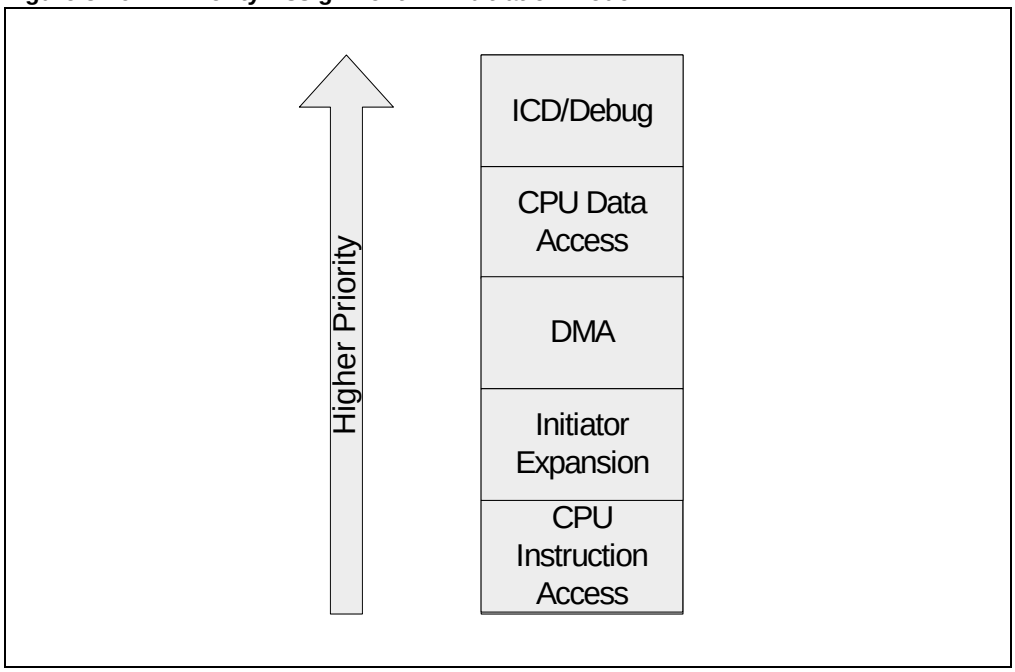


3.5.1.2 Arbitration Mode 1

Arbitrition Mode 1 is a fixed priority scheme like Mode 0; however, the CPU IS is always the lowest priority. Figure 3-16 shows the priority scheme in Mode 1. Mode 1 arbitration is the default mode.

Programming the register field BMXARB (BMXCON<2:0>) to '1' selects Mode 1 operation.

Figure 3-16: Priority Assignment in Arbitration Mode 1

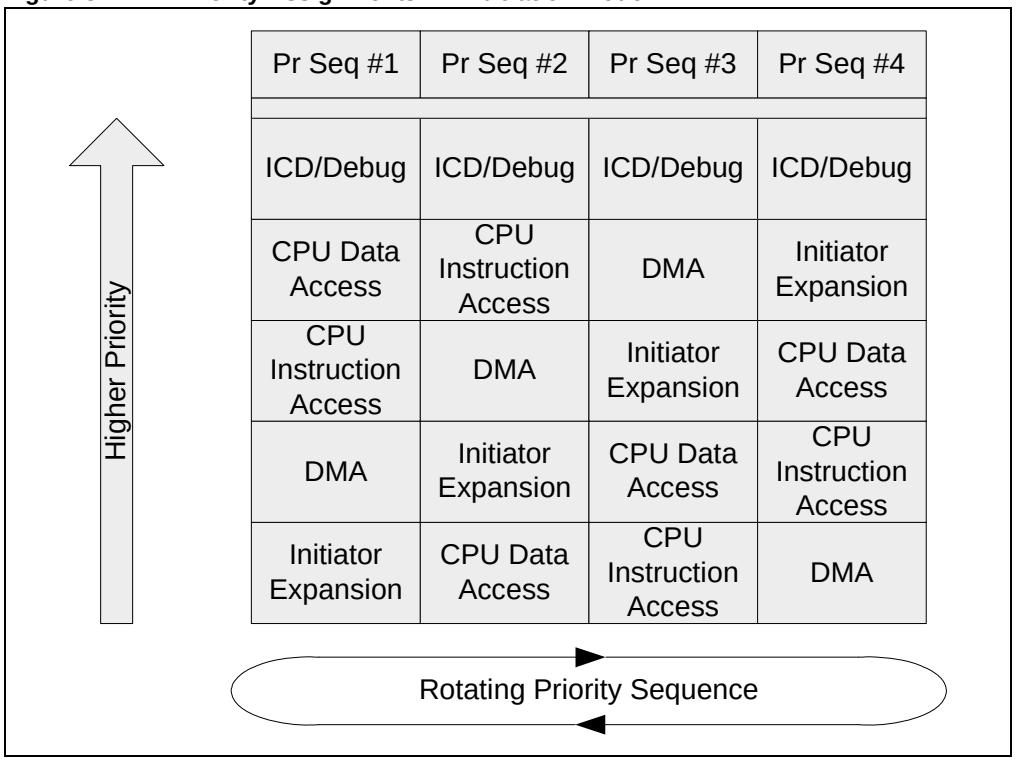


3.5.1.3 Arbitration Mode 2

Mode 2 arbitration supports rotating priority assignments to all initiators. Instead of a fixed priority assignment, each initiator is assigned the highest priority in a rotating fashion. In this mode, the rotating priority is applied with the following exceptions:

1. CPU data is always selected over CPU instruction.
2. ICD is always the highest priority.
3. When the CPU is processing an exception (EXL = 1) or an error (ERL = 1), the arbiter temporarily reverts to Mode 0.

Figure 3-17: Priority Assignments in Arbitration Mode 2



Note that priority sequence #2 is not selected in the rotating priority scheme if there is a pending CPU data access. In this case, once the data access is complete, sequence #2 is selected.

Programming the register field BMXARB (BMXCON<2:0>) to '2' selects Mode 2 operation.

3.5.2 Bus Error Exceptions

The Bus Matrix generates a bus error exception on:

- Any attempt to access unimplemented memory

- Any attempt to access an illegal target

- Any attempt to write to program Flash memory

Bus Error Exceptions may be temporarily disabled by clearing the BMXERRxxx bits in the BMXCON register. This is not recommended.

The Bus Matrix disables bus error exceptions for accesses from CPU IS and CPU DS while in DEBUG mode.

3.5.3 Break Exact Breakpoint Support

The PIC32MX supports break exact breakpoints by inserting one Wait state to data RAM access. This method allows the CPU to stop execution just before the breakpoint address instruction. This is useful in case of breakpointed store instructions. When the Wait state is not used, the break will still occur at the store instruction, however, the DRM location is updated with the store value. If the Wait state is enabled the DRM is not updated with the store value.

3.6 I/O PIN CONTROL

There are no pins associated with this module.

3.7 OPERATION IN POWER-SAVING AND DEBUG MODES

Note:	In this manual, a distinction is made between a power mode as it is used in a specific module, and a power mode as it is used by the device, e.g., Sleep mode of the Comparator and SLEEP mode of the CPU. To indicate which type of power mode is intended, uppercase and lowercase letters (Sleep, Idle, Debug) signify a module power mode, and all uppercase letters (SLEEP, IDLE, DEBUG) signify a device power mode.

3.7.1 Memory Operation on Power-up or Brown-out Reset:

- The contents of data RAM are undefined.
- The BMXxxxBA registers are reset to '0'.
- CPU is switched to Kernel mode.

3.7.2 Memory Operation on Reset:

- The data RAM contents are retained. If the device is code-protected, the RAM contents are cleared.
- The BMX base address registers (BMXxxxBA) are set to '0'.
- CPU is switched to Kernel mode.

3.7.3 Memory Operation on Wake-up from SLEEP or IDLE Mode:

- The RAM contents are retained.
- The BMX base address register (BMXxxxBA) contents are not changed.
- CPU mode is unchanged.

3.8 CODE EXAMPLES

Example 3-1: Create a User Mode Partition of 12K in Program Flash

```
BMXPUPBA = BMXPFMSZ - (12*1024);  // User Mode Flash 12K,
                                  // Kernel Mode Flash 500K (512K-12K)
```

Example 3-2: Create a Kernel Mode Data RAM Partition of 16K; Rest of RAM for Kernel Program

```
BMXDKPBA = 16*1024;
BMXDUDBA = BMXDRMSZ;
BMXDUPBA = BMXDRMSZ;
```

Example 3-3 can be used to create the following partitions in RAM:

Kernel mode data = 12K

Kernel mode program = 6K

User mode data = 8K

User mode program = 6K

Example 3-3: Create RAM Partitions

```
BMXDKPBA = 12*1024;               // Kernel Data Partition of 12K.
                                  // Start offset of Kernel Program Partition
BMXDUDBA = BMXDKPBA + (6*1024);   // Kernel Program Partition of 6K
                                  // Start offset of User Data Partition
BMXDUPBA = BMXDUDBA + (8*1024);   // User Data Partition of 8K
                                  // Start offset of User Program Partition.
                                  // This partition will go up to the size of
                                  // RAM (32K). So the partition size will be
                                  // 6K (32K - 8K - 6K - 12K)
```


3.9 DESIGN TIPS

Question 1: ***At Reset, which mode is the CPU running in?***

Answer: The CPU starts in Kernel mode. The entire RAM is mapped to kernel data segments in KSEG0 and KSEG1. Flash memory is mapped to kernel program segments in KSEG0 and KSEG1. Also ERL = 1, which should be reset to zero (normally in the C start-up code).

Question 2 ***Do I need to initialize the BMX registers?***

Answer: Generally, no. You can leave the BMX registers at their default values, which allows maximum RAM and Flash memories for Kernel mode applications. If you want to run code from RAM or set up User mode partitions, you will need to configure the BMX registers.

Question 3 ***What is the CPU Reset vector address?***

Answer: The CPU Reset address is 0xBFC00000.

Question 4 ***What is a Bus-Error Exception?***

Answer: The bus-error exceptions are generated when the CPU tries to access unimplemented addresses. Also, when the CPU tries to execute a program from RAM without defining a RAM program partition, a bus-error exception is generated.

3.10 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the PIC32MX device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the Memory Organization of the PIC32MX family include the following:

Title	Application Note #
No related application notes at this time.	N/A

Note: Please visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the PIC32MX family of devices.

3.11 REVISION HISTORY

Revision A (August 2007)

This is the initial released version of this document.

Revision B (October 2007)

Updated document to remove Confidential status.

Revision C (April 2008)

Revised status to Preliminary; Revised U-0 to r-x.

Revision D (June 2008)

Change Reserved bits from “Maintain as” to “Write”.

NOTES:

Section 4. Prefetch Cache Module

HIGHLIGHTS

This section of the manual contains the following topics:

4.1 INTRODUCTION

Note:	Prefetch cache is available in select devices only. Refer to the appropriate data sheet for the availability of a prefetch cache module on specific devices.

This section describes the features and operation of the prefetch cache module in the PIC32MX device family. Prefetch cache features increase system performance for most applications.

PFM cache and prefetch cache modules increase performance for applications that execute out of the cacheable Program Flash Memory (PFM) region by implementing the following features:

- Instruction Caching

  The 16-line cache supplies an instruction every clock, for loops up to 256 bytes long.
- Data Caching

  Prefetch cache also allows the allocation of up to 4 cache lines for data storage to provide improved access for Flash-stored constant data.
- Predictive Prefetching

  The prefetch cache module provides instructions once per clock for linear code even without caching by prefetching ahead of the current program counter, hiding the access time of the Flash memory.

4.1.1 Additional Prefetch Cache Module Features

The prefetch cache module also include the following features:

- 16 Fully Associative Lockable Cache Lines
- 16-Byte Cache Lines
- Up to 4 Cache Lines Allocated to Data
- 2 Cache Lines with Address Mask to Hold Repeated Instructions
- Pseudo Least-Recently-Used (LRU) Replacement Policy
- All Cache Lines are Software Writable
- 16-Byte Parallel Memory Fetch
- Predictive Instruction Prefetch Cache

4.2 CACHE OVERVIEW

The prefetch cache module is a performance enhancing module included in some processors of the PIC32MX. When running at high clock rates, Wait states must be inserted into PFM Read transactions to meet the access time of the PFM. Wait states can be hidden to the core by prefetching and storing instructions in a temporary holding area that the CPU can access quickly. Although the data path to the CPU is 32-bits wide, the data path to the Program Memory Flash is 128-bits wide. This wide data path provides the same bandwidth to the CPU as a 32-bit path running at four times the frequency.

There are two main functions that the prefetch cache module performs: caching instructions when they are accessed, and prefetching instructions from the PFM before they are needed.

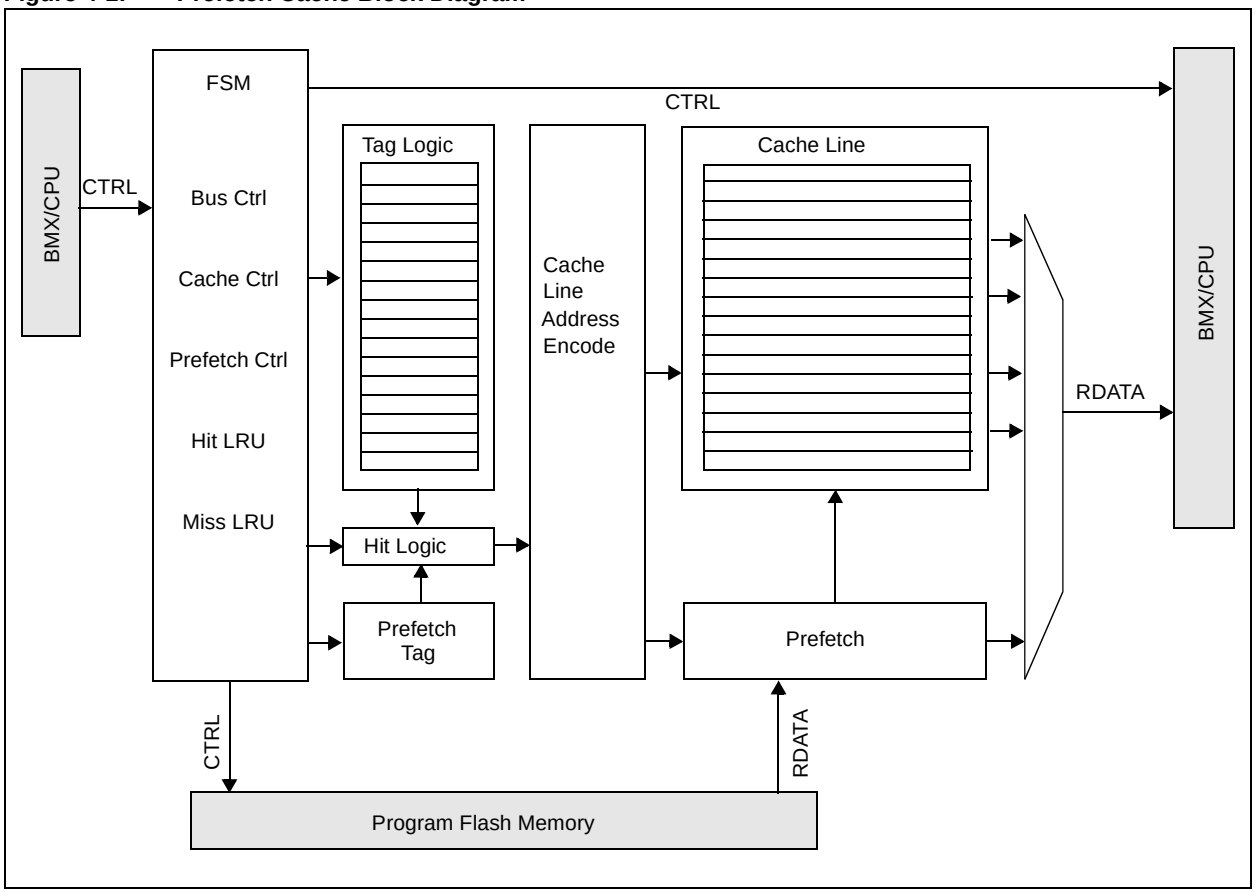
The cache holds a subset of the cacheable memory in temporary holding spaces known as cache lines. Each cache line has a tag describing what it is currently holding, and the address where it is mapped. Normally, the cache lines just hold a copy of what is currently in memory to make data available to the CPU without Wait states.

CPU requested data may or may not be in the cache. A cache-miss occurs if the CPU requests cacheable data that is not in the cache. In this case, a read is performed to the PFM at the correct address, the data is supplied to the cache and to the CPU. A cache-hit occurs if the cache contains the data that the CPU requests. In the case of a cache-hit, data is supplied to the CPU without Wait states.

The second main function of the prefetch cache module is to prefetch cache instructions. The module calculates the address of the next cache line and performs a read of the PFM to get the next 16-byte cache line. This line is placed into a 16-byte-wide prefetch cache buffer in anticipation of executing straight-line code.

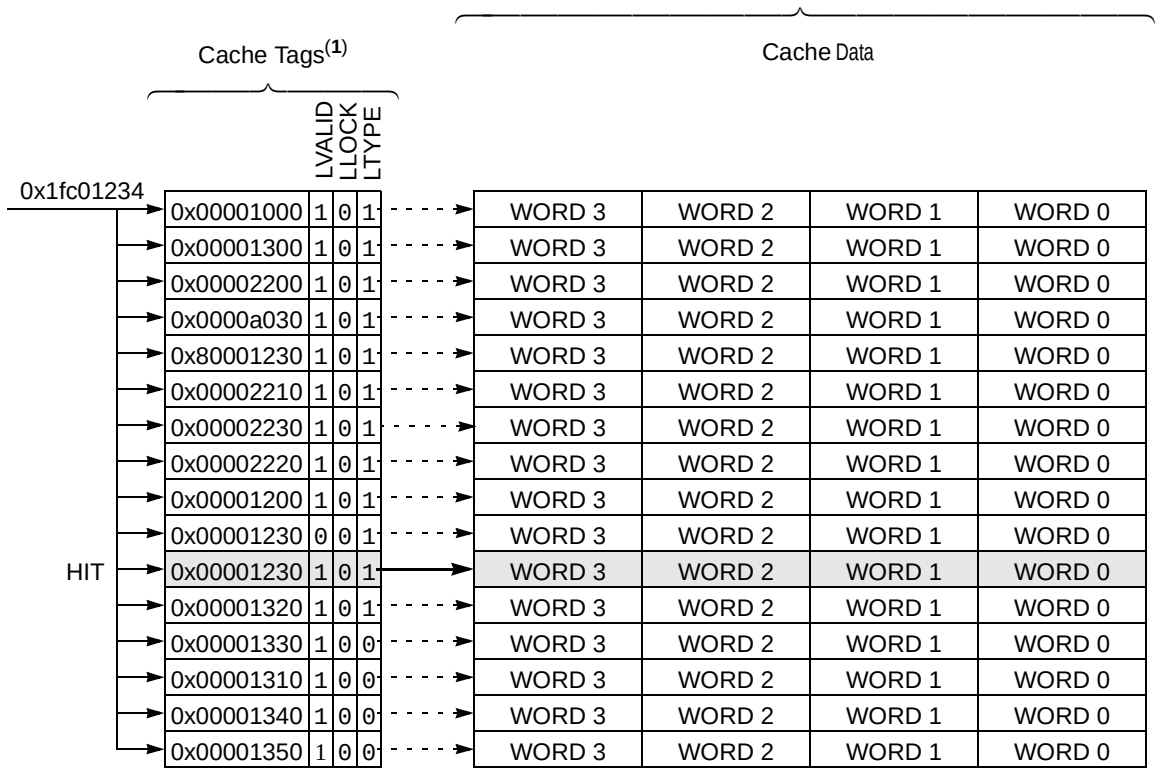
Figure 4-1 shows a block diagram of the prefetch cache module. Logically, the prefetch cache module fits between the Bus Matrix (BMX) module and the PFM module.

Figure 4-1: Prefetch Cache Block Diagram



To illustrate the basic operation of the prefetch cache, Figure 4-2 shows an example of the CPU requesting data from physical address 0x1FC01234. The prefetch cache simultaneously compares this address to all of the tags marked “valid”. Since the shaded entry below has this address, and is marked as valid, this is a cache hit. The proper data word from the data array is then directed to the CPU in a single clock period.

Figure 4-2: Cache Look-up Example(2)



Note 1: Bits 0-3 of the address in the Cache Tags register are always implied ‘0’.

2: Mask Fields are not shown and are assumed to be ‘0’.

4.2.1 Cache Organization

The cache consists of two arrays: tag and data. A data array could consist of program instructions or program data. The cache is physically tagged and address matches are based on the physical address not the virtual address.

Each line in the tag array contains the following information:

- Mask – address mask value
- Tag – tag address to match against
- Valid bit
- Lock bit
- Type – an instruction and/or data type-indicator bit

Each line in the data array contains 16-bytes of program instruction, or program data, depending on the value of the type-indicator bit.

Figure 4-3 shows the organization of a line. Note that the LMASK (CHEMSK<15:5>) and LTYPE (CHETAG<1>) fields are not programmable for every line. The LTAG (CHETAG<23:4>) field only implements the number of bits needed to fully map to the size of the PFM, e.g., if the Flash size is 512 KB, the LTAG (CHETAG<23:4>) field only implements bits 18 through 4.

Figure 4-3: Mask Line

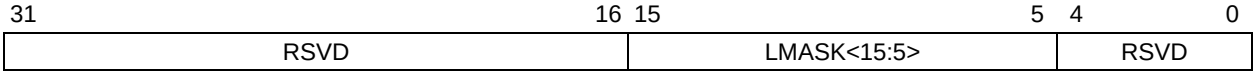


Figure 4-4: Tag Line

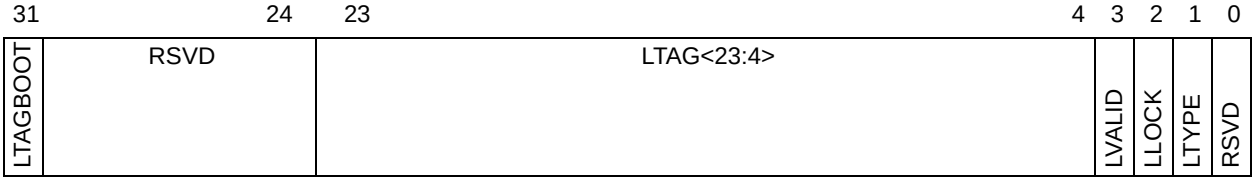
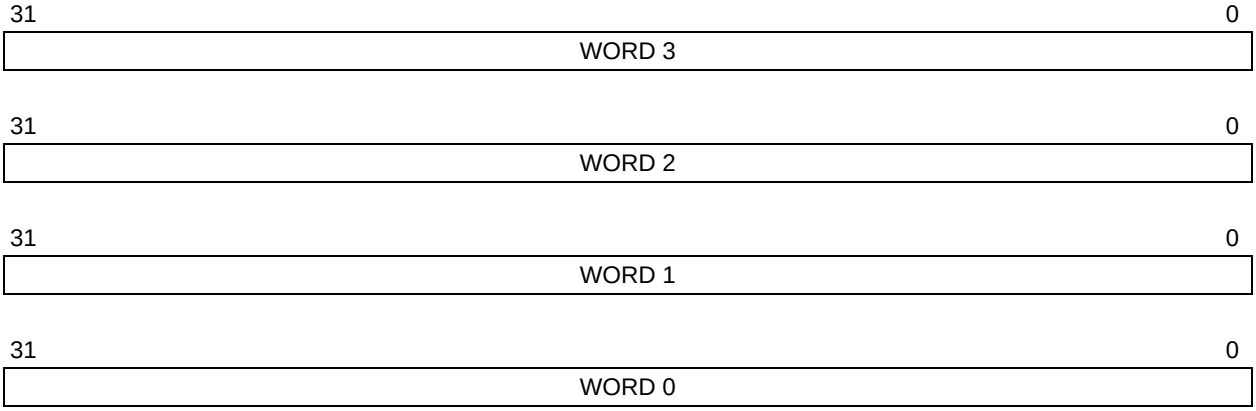


Figure 4-5: Data Line



Cache arrays are shown in Table 4-1. Software can modify values in both the Tag Line and the Data Line of the cache. Configuration register field CHEIDX (CHEACC<3:0>) selects a line for access. That line can then be modified via the CHETAG, CHEMSK, CHEW0, CHEW1, CHEW2, and CHEW3 registers.

Table 4-1: Cache Arrays

Line #	Tag Array					Data Array[2]			
0	000h[1]	TAG	V	L	T[3]	Word 3	Word 2	Word 1	Word 0
1	000h[1]	TAG	V	L	T[3]	Word 3	Word 2	Word 1	Word 0
2	000h[1]	TAG	V	L	T[3]	Word 3	Word 2	Word 1	Word 0
3	000h[1]	TAG	V	L	T[3]	Word 3	Word 2	Word 1	Word 0
4	000h[1]	TAG	V	L	T[3]	Word 3	Word 2	Word 1	Word 0
5	000h[1]	TAG	V	L	T[3]	Word 3	Word 2	Word 1	Word 0
6	000h[1]	TAG	V	L	T[3]	Word 3	Word 2	Word 1	Word 0
7	000h[1]	TAG	V	L	T[3]	Word 3	Word 2	Word 1	Word 0
8	000h[1]	TAG	V	L	T[3]	Word 3	Word 2	Word 1	Word 0
9	000h[1]	TAG	V	L	T[3]	Word 3	Word 2	Word 1	Word 0
A	MASK	TAG	V	L	T[3]	Word 3	Word 2	Word 1	Word 0
B	MASK	TAG	V	L	T[3]	Word 3	Word 2	Word 1	Word 0
C	000h[1]	TAG	V	L	T	Word 3	Word 2	Word 1	Word 0
D	000h[1]	TAG	V	L	T	Word 3	Word 2	Word 1	Word 0
E	000h[1]	TAG	V	L	T	Word 3	Word 2	Word 1	Word 0
F	000h[1]	TAG	V	L	T	Word 3	Word 2	Word 1	Word 0

Note 1: Read-only field.

2: Read zeros when device is code-protected. Read/write otherwise.

3: Type is fixed as instruction.

It is recommended that cache lines be modified while executing from non-cacheable addresses, since the cache controller does not protect against modifying the cache while executing from cacheable address.

Not all fields are writable. The LMASK (CHEMSK<15:5>) field is only writable for lines 10 and 11, and the LTYPE (CHETAG<1>) field is fixed to the “Instruction” setting for lines 0 through 11.

Note that lines allocated for Lock and Data affect the selection of the line to replace on a miss. However, they do not affect the usage order or pseudo LRU value.

4.3 CONTROL REGISTERS

Note: Some devices in the PIC32MX family do not contain a prefetch cache module. For these devices, all prefetch cache register locations are reserved and should not be accessed.

The prefetch cache module contains the following Special Functions Registers (SFRs):

- CHECON: Prefetch Cache Control Register
  Manages configuration of the Prefetch Cache and controls Wait states.
- CHECONCLR, CHECONSET, CHECONINV: Atomic Bit Manipulation Write-only Registers for CHECON
- CHEACC: Prefetch Cache Access Register
  Points to one of the 16 cache lines to access using the CHETAG, CHEMSK, CHEW0, CHEW1, CHEW2, and CHEW3 registers.
- CHEACCCLR, CHEACCSET, CHEACCINV: Atomic Bit Manipulation Write-only Registers for CHEACC
- CHETAG: Prefetch Cache TAG Register
  Contains the address and type of information stored in a cache line.
- CHETAGCLR, CHETAGSET, CHETAGINV: Atomic Bit Manipulation Write-only Registers for CHETAG
- CHEMSK: Prefetch Cache TAG Mask Register
  Provides a mechanism to ignore TAG bits in CHETAG.
- CHEMSKCLR, CHEMSKSET, CHEMSKINV: Atomic Bit Manipulation Write-only Registers for CHEMSK
- CHEW0: Cache Word 0 Register
  Provides Access to the Prefetch Cache Data Array
- CHEW1: Cache Word 1 Register
  Provides Access to the Prefetch Cache Data Array
- CHEW2: Cache Word 2 Register
  Provides Access to the Prefetch Cache Data Array
- CHEW3: Cache Word 3 Register
  Provides Access to the Prefetch Cache Data Array
- CHELRU: Cache LRU Register
- CHEHIT: Cache Hit Statistics Register
- CHEMIS: Cache Miss Statistics Register
- PFABT: Prefetch Cache Abort Statistics Register
  A statistical register that contains the number of aborted Prefetch Cache operations.

The following table provides a brief summary of prefetch cache-related registers. Corresponding registers appear after the summary, followed by a detailed description of each register.

Table 4-2: Prefetch Cache SFRs Summary

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
CHECON	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	CHECOH
	15:8	—	—	—	—	—	—	DCSZ<1:0>	
	7:0	—	—	PREFEN<1:0>		—	PFMWS<2:0>		
CHECONCLR	31:0	Clears selected bits in CHECON, read yields undefined value							
CHECONSET	31:0	Sets selected bits in CHECON, read yields undefined value							
CHECONINV	31:0	Inverts selected bits in CHECON, read yields undefined value							
CHEACC	31:24	CHEWEN	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	—	—	—	—	CHEIDX<3:0>			
CHEACCCLR	31:0	Clears selected bits in CHEACC, read yields undefined value							
CHEACCSET	31:0	Sets selected bits in CHEACC, read yields undefined value							
CHEACCINV	31:0	Inverts selected bits CHEACC, read yields undefined value							
CHETAG	31:24	LTAGBOOT	—	—	—	—	—	—	—
	23:16	LTAG<23:16>							
	15:8	LTAG<15:8>							
	7:0	LTAG<7:4>				LVALID	LLOCK	LTYPE	—
CHETAGCLR	31:0	Clears selected bits in CHETAG, read yields undefined value							
CHETAGSET	31:0	Sets selected bits in CHETAG, read yields undefined value							
CHETAGINV	31:0	Inverts selected bits CHETAG, read yields undefined value							
CHEMSK	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	LMASK<15:8>							
	7:0	LMASK<7:5>			—	—	—	—	—
CHEMSKCLR	31:0	Clears selected bits in CHEMSK, read yields undefined value							
CHEMSKSET	31:0	Sets selected bits in CHEMSK, read yields undefined value							
CHEMSKINV	31:0	Inverts selected bits CHEMSK, read yields undefined value							
CHEW0	31:24	CHEW0<31:24>							
	23:16	CHEW0<23:16>							
	15:8	CHEW0<15:8>							
	7:0	CHEW0<7:0>							
CHEW1	31:24	CHEW1<31:24>							
	23:16	CHEW1<23:16>							
	15:8	CHEW1<15:8>							
	7:0	CHEW1<7:0>							
CHEW2	31:24	CHEW2<31:24>							
	23:16	CHEW2<23:16>							
	15:8	CHEW2<15:8>							
	7:0	CHEW2<7:0>							
CHEW3	31:24	CHEW3<31:24>							
	23:16	CHEW3<23:16>							
	15:8	CHEW3<15:8>							
	7:0	CHEW3<7:0>							

Table 4-2: Prefetch Cache SFRs Summary

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
CHELRU	31:24	—	—	—	—	—	—	—	CHELRU<24>
	23:16	CHELRU<23:16>							
	15:8	CHELRU<15:8>							
	7:0	CHELRU<7:0>>							
CHEHIT	31:24	CHEHIT<31:24>							
	23:16	CHEHIT<23:16>							
	15:8	CHEHIT<15:8>							
	7:0	CHENIT<7:0>							
CHEMIS	31:24	CHEMIS<31:24>							
	23:16	CHEMIS<23:16>							
	15:8	CHEMIS<15:8>							
	7:0	CHEMIS<7:0>							
PFABT	31:24	PFABT<31:24>							
	23:16	PFABT<23:16>							
	15:8	PFABT<15:8>							
	7:0	PFABT<7:0>							

Register 4-1: CHECON: Cache Control Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	R/W-0
—	—	—	—	—	—	—	CHECOH
bit 23							bit 16

r-x	r-x	r-0	r-0	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	DCSZ<1:0>	
bit 15							bit 8

r-x	r-x	R/W-0	R/W-0	r-x	R/W-1	R/W-1	R/W-1
—	—	PREFEN<1:0>		—	PFMWS<2:0>		
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-17 **Reserved:** Write ‘0’; ignore read

bit 16 **CHECOH:** Cache Coherency setting on a PFM Program Cycle bit
1 = Invalidate all data and instruction lines
0 = Invalidate all data lnes and instruction lines that are not locked

bit 15-14 **Reserved:** Write ‘0’; ignore read

bit 13-12 **Reserved:** Must be written with zeros

bit 11-10 **Reserved:** Write ‘0’; ignore read

bit 9-8 **DCSZ<1:0>:** Data Cache Size in Lines bits
11 = Enable data caching with a size of 4 Lines
10 = Enable data caching with a size of 2 Lines
01 = Enable data caching with a size of 1 Line
00 = Disable data caching
Changing this field causes all lines to be re-initialized to the “invalid” state.

bit 7-6 **Reserved:** Write ‘0’; ignore read

bit 5-4 **PREFEN<1:0>:** Predictive Prefetch Cache Enable bits
11 = Enable predictive prefetch cache for both cacheable and non-cacheable regions
10 = Enable predictive prefetch cache for non-cacheable regions only
01 = Enable predictive prefetch cache for cacheable regions only
00 = Disable predictive prefetch cache

bit 3 **Reserved:** Write ‘0’; ignore read

Register 4-1: CHECON: Cache Control Register (Continued)

bit 2-0 **PFMWS<2:0>:** PFM Access Time Defined in terms of SYSLK Wait states bits

111 = Seven Wait states
110 = Six Wait states
101 = Five Wait state
100 = Four Wait states
011 = Three Wait states
010 = Two Wait states
001 = One Wait state
000 = Zero Wait states

Register 4-2: CHECONCLR: CHECON Clear Register

Write clears selected bits in CHECON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in CHECON**

A write of '1' in one or more bit positions clears the corresponding bit(s) in CHECON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `CHECONCLR = 0x00010020` will clear bits 16 and 5 in CHECON register.

Register 4-3: CHECONSET: CHECON Set Register

Write sets selected bits in CHECON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in CHECON**

A write of '1' in one or more bit positions sets the corresponding bit(s) in CHECON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `CHECONSET = 0x00010020` will set bits 16 and 5 in CHECON register.

Register 4-4: CHECONINV: CHECON Invert Register

Write inverts selected bits in CHECON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in CHECON**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in CHECON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `CHECONINV = 0x00010020` will invert bits 16 and 5 in CHECON register.

Register 4-5: CHEACC: Cache Access

R/W-0	r-x	r-x	r-x	r-x	r-x	r-x	r-x
CHEWEN	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

r-x	r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	CHEIDX<3:0>			
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31 **CHEWEN:** Cache Access Enable bits for registers CHETAG, CHEMSK, CHEW0, CHEW1, CHEW2, and CHEW3

1 = The cache line selected by CHEIDX is writeable
0 = The cache line selected by CHEIDX is not writeable

bit 30-4 **Reserved:** Write ‘0’; ignore read

bit 3-0 **CHEIDX<3:0>:** Cache Line Index bits

The value selects the cache line for reading or writing.

Register 4-6: CHEACCCLR: CHEACC Clear Register

Write clears selected bits in CHEACC, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in CHEACC**

A write of '1' in one or more bit positions clears the corresponding bit(s) in CHEACC register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `CHEACCCLR = 0x80000000` will clear bit 31 in CHEACC register.

Register 4-7: CHEACCSET: CHEACC Set Register

Write sets selected bits in CHEACC, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in CHEACC**

A write of '1' in one or more bit positions sets the corresponding bit(s) in CHEACC register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `CHEACCSET = 0x80000000` will clear bit 31 in CHEACC register.

Register 4-8: CHEACCINV: CHEACC Invert Register

Write inverts selected bits in CHEACC, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in CHEACC**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in CHEACC register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `CHEACCINV = 0x80000000` will invert bit 31 in CHEACC register.

Register 4-9: CHETAG(1): Cache TAG Register

R/W-0	r-x	r-x	r-x	r-x	r-x	r-x	r-x
LTAGBOOT	—	—	—	—	—	—	—
bit 31							bit 24

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
LTAG<23:16>							
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
LTAG<15:8>							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-0	R/W-0	R/W-1	r-0
LTAG<7:4>				LVALID	LLOCK	LTYPE	—
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31 **LTAGBOOT:** Line TAG Address Boot
1 = The line is in the 0x1D000000 (physical) area of memory
0 = The line is in the 0x1FC00000 (physical) area of memory

bit 30-24 **Reserved:** Write ‘0’; ignore read

bit 23-4 **LTAG<23:4>:** Line TAG Address bits
LTAG bits are compared against physical address <23:4> to determine a hit. Because its address range and position of Flash in kernel space and user space, the LTAG Flash address is identical for virtual addresses, (system) physical addresses, and Flash physical addresses.

bit 3 **LVALID:** Line Valid bit
1 = The line is valid and is compared to the physical address for hit detection
0 = The line is not valid and is not compared to the physical address for hit detection

bit 2 **LLOCK:** Line Lock bit
1 = The line is locked and will not be replaced
0 = The line is not locked and can be replaced

bit 1 **LTYPE:** Line Type bit
1 = The line caches instruction words
0 = The line caches data words

bit 0 **Reserved:** Write ‘0’; ignore read

Note 1: The TAG and Status of the Line pointed to by CHEIDX (CHEACC<3:0>).

Register 4-10: CHETAGCLR: CHETAG Clear Register

Write clears selected bits in CHETAG, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in CHETAG**

A write of '1' in one or more bit positions clears the corresponding bit(s) in CHETAG register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `CHETAGCLR = 0x0000000C` will clear bits 2 and 3 in CHETAG register.

Register 4-11: CHETAGSET: CHETAG Set Register

Write sets selected bits in CHETAG, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in CHETAG**

A write of '1' in one or more bit positions sets the corresponding bit(s) in CHETAG register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `CHETAGSET = 0x00000004` will set bit 2 in CHETAG register.

Register 4-12: CHETAGINV: CHETAG Invert Register

Write inverts selected bits in CHETAG, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in CHETAG**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in CHETAG register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `CHETAGINV = 0x00000010` will invert bit 4 in CHETAG register.

Register 4-13: CHEMSK(1): Cache TAG Mask Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
LMASK<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	r-x	r-x	r-x	r-x	r-x
LMASK<7:5>			—	—	—	—	—
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15-5 **LMASK<15:5>:** Line Mask bits

1 = Enables mask logic to force a match on the corresponding bit position in LTAG (CHETAG<23:4>) and the physical address.

0 = Only writeable for values of CHEIDX (CHEACC<3:0>) equal to OxOA and OxOB. Disables mask logic.

bit 4-0 **Reserved:** Write ‘0’; ignore read

Note 1: The TAG Mask of the Line pointed to by CHEIDX (CHEACC<3:07>).

Register 4-14: CHEMSKCLR: CHEMSK Clear Register

Write clears selected bits in CHEMSK, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in CHEMSK**

A write of '1' in one or more bit positions clears the corresponding bit(s) in CHEMSK register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `CHEMSKCLR = 0x00008020` will clear bits 15 and 5 in CHEMSK register.

Register 4-15: CHEMSKSET: CHEMSK Set Register

Write sets selected bits in CHEMSK, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in CHEMSK**

A write of '1' in one or more bit positions sets the corresponding bit(s) in CHEMSK register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `CHEMSKSET = 0x00008020` will set bits 15 and 5 in CHEMSK register.

Register 4-16: CHEMSKINV: CHEMSK Invert Register

Write inverts selected bits in CHEMSK, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in CHEMSK**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in CHEMSK register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `CHEMSKINV = 0x00008020` will invert bits 15 and 5 in CHEMSK register.

Register 4-17: CHEW0: Cache Word 0

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHEW0<31:24>							
bit 31							bit 24

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHEW0<23:16>							
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHEW0<15:8>							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHEW0<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **CHEW0<31:0>:** Word 0 of the cache line selected by CHEACC.CHEIDX
Readable only if the device is not code-protected.

Register 4-18: CHEW1: Cache Word 1

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHEW1<31:24>							
bit 31							bit 24

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHEW1<23:16>							
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHEW1<15:8>							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHEW1<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **CHEW1<31:0>:** Word 1 of the cache line selected by CHEACC.CHEIDX
Readable only if the device is not code-protected.

Register 4-19: CHEW2 Cache Word 2

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHEW2<31:24>							
bit 31							bit 24

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHEW2<23:16>							
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHEW2<15:8>							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHEW2<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **CHEW2<31:0>:** Word 2 of the cache line selected by CHEACC.CHEIDX
Readable only if the device is not code-protected.

Register 4-20: CHEW3[(1)]: Cache Word 3

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHEW3<31:24>							
bit 31							bit 24

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHEW3<23:16>							
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHEW3<15:8>							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHEW3<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **CHEW3<31:0>:** Word 3 of the cache line selected by CHEACC.CHEIDX
Readable only if the device is not code-protected.

Note 1: This register is a window into the cache data array and is readable only if the device is not code-protected.

Register 4-21: CHELRU: Cache LRU Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	R-0
—	—	—	—	—	—	—	CHELRU<24>
bit 31							bit 24

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
CHELRU<23-16>							
bit 23							bit 16

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
CHELRU<15-8>							
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
CHELRU<7-0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-25 **Reserved:** Write ‘0’; ignore read

bit 24-0 **CHELRU<24:0>:** Cache Least Recently Used State Encoding bits
CHELRU indicates the Pseudo-LRU state of the cache.

Register 4-22: CHEHIT: Cache Hit Statistics Register

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHEHIT<31:24>							
bit 31							bit 24

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHEHIT<23:16>							
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHEHIT<15:8>							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHEHIT<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **CHEHIT<31:0>:** Cache Hit Count bits

Incremented each time the processor issues an instruction fetch or load that hits the prefetch cache from a cacheable region. Non-cacheable accesses do not modify this value.

Register 4-23: CHEMIS: Cache Miss Statistics Register

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHEMIS<31:24>							
bit 31							bit 24

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHEMIS<23:16>							
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHEMIS<15:8>							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHEMIS<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **CHEMIS<31:0>:** Cache Miss Count bits

Incremented each time the processor issues an instruction fetch from a cacheable region that misses the prefetch cache. Non-cacheable accesses do not modify this value.

Register 4-24: PFABT: Prefetch Cache Abort Statistics Register

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
PFABT<31:24>							
bit 31							bit 24

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
PFABT<23:16>							
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
PFABT<15:8>							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
PFABT<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **PFABT<31:0>:** Prefab Abort Count bits

Incremented each time an automatic prefetch cache is aborted due to a non-sequential instruction fetch, load or store.

4.4 CACHE OPERATION

The cache and prefetch cache module implements a fully associative 16-line cache. Each line consists of 128 bits (16 bytes). The cache and prefetch cache module only request 16-byte aligned instruction data from the PFM. If the CPU requested address is not aligned to a 16-byte boundary, the module will align the address by dropping address bits<3:0>. When configured only as a cache, the module loads multiple instructions into a line on a miss. It uses the pseudo LRU algorithm to select which line receives the new set of instructions. The cache controller uses the Wait states state values from PFMWS (CHECON<2:0>) to determine how long it must wait for a Flash access when it detects a miss. On a hit, the cache returns data in zero Wait states. If the code is 100% linear, the Cache-Only mode will provide instructions back to the CPU with Wait states only on the first instruction of a cache line. For 32-bit linear code, Wait states are seen every four instructions. For 16-bit linear code, Wait states occur only once for every eight instructions executed.

4.5 CACHE CONFIGURATIONS

The CHECON register controls the configurations available for instruction and data caching of PFM. Two parameters control the allocation of cache lines to specific features.

The DCSZ (CHECON<9:8>) field controls the number of lines allocated to program data caching. Table 4-3 shows the cache line relationship for values of DCSZ (CHECON<9:8>). The data caching capability is for read-only data, e.g., constants, parameters, table data, etc., that are not modified.

Table 4-3: Program Data Cache

DCSZ<1:0>	Lines Allocated to Program Data
00	None
01	Cache Line Number 15
10	Cache Lines Number 14 and 15
11	Cache Lines Number 12 through 15

The PREFEN (CHECON<5:4>) field controls predictive prefetching, which allows the cache controller to speculatively fetch the next 16-byte aligned set of instructions.

4.5.1 Line Locking

Each line in the cache can be locked to hold its contents. A line is locked if both LVALID (CHETAG<3>) = 1 and LLOCK (CHETAG<2>) = 1. If LVALID = 0 and LLOCK = 1, the cache controller issues a preload request (see Section **4.5.3 “Preload Behavior”**). Locking cache lines may reduce the performance of general program flow. However, if one or two function calls consume a significant percent of overall processing, locking their addresses can provide improved performance.

Though any number of lines can be locked, the cache works more efficiently when locking either 1 or 4 lines. If locking 4 lines, choose those lines in which the line numbers, when divide by 4, have the same quotient. This locks an entire LRU group which benefits the LRU algorithm. For example, lines 8, 9, A, and B each have a quotient of 2 when divided by 4.

4.5.2 Address Mask

Cache lines 10 and 11 allow masking of the CPU address, and the tag address, to force a match on corresponding bits. The LMASK (CHEMSK<15:5>) field is set up to complement the interrupt vector spacing field in the CPU. This feature allows boot code to lock the first four instructions of a vector in the cache. If all vectors contain identical instructions in their first four locations, then setting the LMASK (CHEMSK<15:5>) to match the vector spacing, and the LTAG (CHETAG<23:4>) to match the vector base address, causes all the vector addresses to hit the cache. The cache responds with zero Wait states and immediately initiates a fetch of the next set of four instructions for the requesting vector if prefetch cache is enabled.

Using LMASK (CHEMSK<15:5>) is restricted to aligned address ranges. Its size allows for a maximum range of 32 KB and a minimum spacing of 32 B. Using the two lines in conjunction provides the ability to have different ranges and different spacing.

Setting up the address mask such that more than one line will match an address causes undefined results. Therefore, it is highly recommended that masking is set up before entering cacheable code.

4.5.3 Preload Behavior

Application code can direct the cache controller to preform a preload of a cache line and lock it with instructions or data from the Flash. The preload function uses the CHEACC.CHEIDX register field to select the cache line into which the load is directed. Setting CHEACC.CHEWEN to '1' enables writes to the CHETAG register.

Writing LVALID (CHETAG<3>) = 0 and LLOCK (CHETAG<2>) = 1 causes a preload request to the cache controller. The controller acknowledges the request in the cycle after the write and, if possible, stops any outstanding Flash access, and stalls any CPU load from the cache or Flash.

When the controller has finished or stalled the previous transaction, it initiates a Flash read to fetch the instructions, or data, requested using the address in LTAG (CHETAG<23:4>). After the programmed number of Wait states, as defined by PFMWS (CHECON<2:0>), the controller updates the data array with the values read from Flash. On the update, it sets LVALID (CHETAG<3>) = 1. The LRU state of the line is not affected.

Once the controller finishes updating the cache, it allows CPU requests to complete. If this request misses the cache, the controller initiates a Flash read, which incurs the full Flash access time.

4.5.4 Bypass Behavior

Processor accesses in which cache coherency attributes indicate uncacheable addresses bypass the cache. In bypass, the module accesses the PFM for every instruction, incurring the Flash access time as defined by PFMWS (CHECON<2:0>).

4.5.5 Predictive Prefetch Cache Behavior

When configured for predictive prefetch cache on cacheable addresses, the module predicts the next line address and returns it into the pseudo LRU line of the cache. If enabled, the prefetch cache function starts predicting based on the first CPU instruction fetch. When the first line is placed in the cache, the module simply increments the address to the next 16-byte aligned address and starts a Flash access. When running linear code (i.e. no jumps), the Flash returns the next set of instructions into the prefetch cache buffer on or before all instructions can be executed from the previous line.

If, at any time during a predicted Flash access, a new CPU address does not match the predicted one, the Flash access will be changed to the correct address. This behavior does not cause the CPU access to take any longer than it does without prediction.

If an access that misses the cache hits the prefetch cache buffer, the instructions are placed in the pseudo LRU line, along with its address tag. The pseudo LRU value is marked as the most recently used line, and other lines are updated accordingly. If an access misses both the cache and the prefetch cache buffer, the access passes to the Flash, and those returning instructions are placed in the pseudo LRU line.

When configured for predictive prefetch cache on non-cacheable addresses, the controller only uses the prefetch cache buffer. The LRU cache line is not updated for hits or fills, so the cache remains intact. For linear code, enabling predictive prefetch cache for non-cacheable addresses allows the CPU to fetch instructions in zero Wait states.

It is not useful to use non-cacheable predictive prefetching when accesses to the Flash are set for zero Wait states. The controller holds prefetched instructions on the output of the Flash for up to 3 clock cycles (while the CPU is fetching from the buffer). This consumes more power, without any benefit, for zero-Wait-state Flash accesses.

Predictive data prefetching is not supported. However, a data access in the middle of a predictive instruction fetch causes the cache controller to stop the Flash access for the instruction fetch, and to start the data load from Flash. The predictive prefetch cache does not resume, but instead, waits for another instruction fetch. At which time, it either fills the buffer because of a miss, or starts a prefetch cache because of a hit.

4.5.6 Cache Replacement Policy

The cache controller uses a pseudo-LRU replacement policy for cache line fills that are caused by a read miss. The policy allows any line in the last quarter of least recently used lines to be replaced. Enabling locking and data caching affect the line to be replaced, but not the actual value of the pseudo-LRU.

4.6 COHERENCY SUPPORT

It is not possible to execute out of cache while programming the Flash memory. The Flash controller stalls the cache during the programming sequence. Therefore, user code that initiates a programming sequence should not be located in a cacheable address region.

During a programming operation, the prefetch cache is flushed by invalidating either all, or some of the cache lines.

If CHECOH (CHECON<16>) is set, every cache line is invalidated and unlocked during a Flash program memory write operation. The cache tags and masks are also cleared for all lines.

If CHECOH is not set, only lines that are not locked are forced invalid. Lines that are locked are retained.

4.7 EFFECTS OF RESET

4.7.1 On Reset

- All cache lines are invalidated
- All cache lines revert to instruction
- All cache lines are unlocked
- The LRU order is sequential, with line 0 being the least recently used
- All mask bits are cleared
- All registers revert to their Reset state

4.7.2 After Reset

- The module operates as per the values in the CHECON register
- The cache obeys the core's cache coherency attributes

4.8 DESIGN TIPS

Even while running at clock frequencies allowing for zero-Wait-state operation, the cache function proves useful as a power-saving technique. Accesses to the Flash memory consume more power than accesses to the cache.

4.9 OPERATION IN POWER-SAVING MODES

Note: In this manual, a distinction is made between a power mode as it is used in a specific module, and a power mode as it is used by the device, e.g., Sleep mode of the Comparator and SLEEP mode of the CPU. To indicate which type of power mode is intended, uppercase and lowercase letters (Sleep, Idle, Debug) signify a module power mode, and all uppercase letters (SLEEP, IDLE, DEBUG) signify a device power mode.

4.9.1 SLEEP Mode

When the device enters SLEEP mode, the prefetch cache is disabled and placed into a low-power state where no clocking occurs in the prefetch cache module.

4.9.2 IDLE Mode

When the device enters IDLE mode, the cache and prefetch cache clock source remains functional and the CPU stops executing code. Any outstanding prefetch cache completes before the module stops its clock via automatic clock gating.

4.9.3 DEBUG Mode

The behavior of the prefetch cache is unaltered by DEBUG mode. Care must be taken to make sure the cache remains coherent during DEBUG mode execution when using software breakpoints. If a debugger places a software break instruction in the cache, the line should be locked before returning control to the application. When a locked software breakpoint is removed, the line should be unlocked and invalidated, causing the original instructions to be reloaded from the PFM upon execution.

4.10 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the PIC32MX device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the prefetch cache module are:

Title	Application Note #
No related application notes at this time.	N/A

Note: Please visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the PIC32MX family of devices.

4.11 REVISION HISTORY

Revision A (October 2007)

This is the initial released version of this document.

Revision B (October 2007)

Updated document to remove Confidential status.

Revision C (April 2008)

Revised status to Preliminary; Revise U-0 to r-x.

Revision D (June 2008)

Change Reserved bits from “Maintain as” to “Write”.

Section 5. Flash Programming

HIGHLIGHTS

This section of the manual contains the following topics:

5.1 INTRODUCTION

This section describes techniques for programming the Flash memory. PIC32MX devices contain internal Flash memory for executing user code. There are three methods by which the user can program this memory:

- Run-Time Self Programming (RTSP) – performed by the user's software
- In-Circuit Serial Programming™ (ICSP™) – performed using a serial data connection to the device, allows much faster programming than RTSP
- EJTAG Programming – performed by an EJTAG-capable programmer, using the EJTAG port of the device

RTSP techniques are described in this chapter. The ICSP and EJTAG methods are described in the PIC32MX Programming Specification document, which can be downloaded from the Microchip web site at www.microchip.com.

5.2 CONTROL REGISTERS

Flash program and erase operations are controlled using the following Nonvolatile Memory (NVM) control registers:

- NVMCON: Nonvolatile Memory Control Register

  NVMCONCLR, NVMCONSET, NVMCONINV: Atomic Bit Manipulation, Write-only Registers for NVMCON

- NVMKEY: Nonvolatile Memory Key Register
- NVMADDR: Nonvolatile Memory Address Register

  NVMADDRCLR, NVMADDRSET, NVMADDRINV: Atomic Bit Manipulation, Write-only Registers for NVMADDR

- NVMDATA: Nonvolatile Memory Data Register
- NVMSRCADDR: Nonvolatile Memory SRAM Source Address Register
- IFSx: Interrupt Flag Status Registers

  IFSxCLR, IFSxSET, IFSxINV: Atomic Bit Manipulation, Write-only Registers for IFSx

- IECx: Interrupt Enable Control Registers

  IECxCLR, IECxSET, IECxINV: Atomic Bit Manipulation, Write-only Registers for IECx

- IPCx: Interrupt Priority Control Registers
- IPCxCLR, IPCxSET, IPCxINV: Atomic Bit Manipulation, Write-only Registers for IPCx

The following table provides a brief summary of all the Flash-programming-related registers. Corresponding registers appear after the summary, followed by a detailed description of each register.

Table 5-1: Flash Controller SFR Summary

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
NVMCON	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	NVMWR	NVMWREN	NVMERR	LVDERR	LVDSTAT	—	—	—
	7:0	—	—	—	—	NVMOP<3:0>			
NVMCONCLR	31:0	Write clears selected bits in NVMCON, read yields undefined value							
NVMCONSET	31:0	Write sets selected bits in NVMCON, read yields undefined value							
NVMCONINV	31:0	Write inverts selected bits in NVMCON, read yields undefined value							
NVMKEY	31:24	NVMKEY<31:24>							
	23:16	NVMKEY<23:16>							
	15:8	NVMKEY<15:8>							
	7:0	NVMKEY<7:0>							
NVMADDR	31:24	NVMADDR<31:24>							
	23:16	NVMADDR<23:16>							
	15:8	NVMADDR<15:8>							
	7:0	NVMADDR<7:0>							
NVMADDRCLR	31:0	Write clears selected bits in NVMADDR, read yields undefined value							
NVMADDRSET	31:0	Write sets selected bits in NVMADDR, read yields undefined value							
NVMADDRINV	31:0	Write inverts selected bits in NVMADDR, read yields undefined value							

Table 5-1: Flash Controller SFR Summary (Continued)

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
NVMDATA	31:24	NVMDATA<31:24>							
	23:16	NVMDATA<23:16>							
	15:8	NVMDATA<15:8>							
	7:0	NVMDATA<7:0>							
NVMSRCADDR	31:24	NVMSRCADDR<31:24>							
	23:16	NVMSRCADDR<23:16>							
	15:8	NVMSRCADDR<15:8>							
	7:0	NVMSRCADDR<7:0>							

Table 5-2: Flash Controller Interrupt SFR Summary

IFS1	31:24	—	—	—	—	—	—	USBIF	FCEIF
	23:16	—	—	—	—	DMA3IF	DMA2IF	DMA1IF	DMA0IF
	15:8	RTCCIF	FSCMIF	I2C2MIF	I2C2SIF	I2C2BIF	U2TXIF	U2RXIF	U2EIF
	7:0	SPI2RXIF	SPI2TXIF	SPI2EIF	CMP2IF	CMP1IF	PMPIF	AD1IF	CNIF
IFS1CLR	31:0	Write clears the selected bits in IFS1, read yields undefined value							
IFS1SET	31:0	Write sets the selected bits in IFS1, read yields undefined value							
IFS1INV	31:0	Write inverts the selected bits in IFS1, read yields undefined value							
IEC1	31:24	—	—	—	—	—	—	USBIE	FCEIE
	23:16	—	—	—	—	DMA3IE	DMA2IE	DMA1IE	DMA0IE
	15:8	RTCCIE	FSCMIE	I2C2MIE	I2C2SIE	I2C2BIE	U2TXIE	U2RXIE	U2EIE
	7:0	SPI2RXIE	SPI2TXIE	SPI2EIE	CMP2IE	CMP1IE	PMPIE	AD1IE	CNIE
IEC1CLR	31:0	Write clears the selected bits in IEC1, read yields undefined value							
IEC1SET	31:0	Write sets the selected bits in IEC1, read yields undefined value							
IEC1INV	31:0	Write inverts the selected bits in IEC1, read yields undefined value							
IPC11	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	—	—	—	FCEIP<2:0>			FCEIS<1:0>	
IPC11CLR	31:0	Write clears the selected bits in IPC11, read yields undefined value							
IPC11SET	31:0	Write sets the selected bits in IPC11, read yields undefined value							
IPC11INV	31:0	Write inverts the selected bits in IPC11, read yields undefined value							

Register 5-1: NVMCON: Programming Control Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R-0	R-0	R-x	r-x	r-x	r-x
NVMWR	NVMWREN	NVMERR	LVDERR	LVDSTAT	—	—	—
bit 15							bit 8

r-x	r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	NVMOP3	NVMOP2	NVMOP1	NVMOP0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15 **NVMWR:** Write Control bit
This bit is writable when NVMWREN = 1 and the unlock sequence is followed
1 = Initiate a Flash operation. Hardware clears this bit when the operation completes.
0 = Flash operation complete or inactive

bit 14 **NVMWREN:** Write Enable bit
1 = Enable writes to NVMWR bit and enables LVD circuit
0 = Disable writes to NVMWR bit and disables LVD circuit
Note: This is the only bit in this register reset by a device Reset.

bit 13 **NVMERR:** Write Error bit
This bit is read-only and is automatically set by hardware
1 = Program or erase sequence did not complete successfully
0 = Program or erase sequence completed normally
Note: Cleared by setting NVMOP==0000b, and initiating a Flash operation (i.e., NVMWR).

bit 12 **LVDERR:** Low Voltage Detect Error Bit (LVD circuit must be enabled)
This bit is read-only and is automatically set by hardware
1 = Low voltage detected (possible data corruption, if NMVERR is set)
0 = Voltage level is acceptable for programming
Note: Cleared by setting NVMOP==0000b, and initiating a Flash operation (i.e., NVMWR).

bit 11 **LVDSTAT:** Low-Voltage Detect Status bit (LVD circuit must be enabled)
This bit is read-only and is automatically set, and cleared, by hardware
1 = Low voltage event active
0 = Low voltage event NOT active
Note: Cleared by setting NVMOP==0000b, and initiating a Flash operation (i.e., NVMWR).

bit 10-4 **Reserved:** Write ‘0’; ignore read

Register 5-1: NVMCON: Programming Control Register (Continued)

bit 3-0 **NVMOP<3:0>:** NVM Operation bits

These bits are writeable when NVMWREN = 1 and the unlock sequence is followed

0111 = Reserved
0110 = No operation
0101 = Program Flash (PFM) erase operation: erases PFM, if all pages are not write-protected
0100 = Page erase operation: erases page selected by NVMADDR, if it is not write-protected
0011 = Row program operation: programs row selected by NVMADDR, if it is not write-protected
0010 = No operation
0001 = Word program operation: programs word selected by NVMADDR, if it is not write-protected
0000 = No operation

Register 5-2: NVMCONCLR: Programming Control Clear Register

Write clears selected bits in NVMCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in NVMCON**

A write of '1' in one or more bit positions clears the corresponding bit(s) in NVMCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `NVMCONCLR = 0x00008001` will clear bits 15 and 0 in NVMCON register.

Register 5-3: NVMCONSET: Programming Control Set Register

Write sets selected bits in NVMCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in NVMCON**

A write of '1' in one or more bit positions sets the corresponding bit(s) in NVMCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `NVMCONSET = 0x00008001` will set bits 15 and 0 in NVMCON register.

Register 5-4: NVMCONINV: Programming Control Invert Register

Write inverts selected bits in NVMCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in NVMCON**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in NVMCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `NVMCONINV = 0x00008001` will invert bits 15 and 0 in NVMCON register.

Register 5-5: NVMKEY: Programming Unlock Register

W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0
NVMKEY<31:24>							
bit 31							bit 24

W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0
NVMKEY<23:16>							
bit 23							bit 16

W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0
NVMKEY<15:8>							
bit 15							bit 8

W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0
NVMKEY<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **NVMKEY<31:0>:** Unlock Register bits

These bits are write-only, and read as ‘0’ on any read

Note: This register is used as part of the unlock sequence to prevent inadvertent writes to the PFM.

Register 5-6: NVMADDR: Flash Address Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
NVMADDR<31:24>							
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
NVMADDR<23:16>							
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
NVMADDR<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	r-x	r-x
NVMADDR<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-3 **NVMADDR<31:0>:** Flash Address bits
Bulk/Chip/PFM Erase:
Address is ignored
Page Erase:
Address identifies the page to erase
Row Program:
Address identifies the row to program
Word Program:
Address identifies the word to program

bit 2-0 **Reserved:** Write ‘0’; ignore read

Register 5-7: NVMADDRCLR: Flash Address Clear Register

Write clears selected bits in NVMADDR, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in NVMADDR**

A write of '1' in one or more bit positions clears the corresponding bit(s) in NVMADDR register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: NVMADDRCLR = 0x00008001 will clear bits 15 and 0 in NVMADDR register.

Register 5-8: NVMADDRSET: Flash Address Set Register

Write sets selected bits in NVMADDR, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in NVMADDR**

A write of '1' in one or more bit positions sets the corresponding bit(s) in NVMADDR register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: NVMADDRSET = 0x00008001 will set bits 15 and 0 in NVMADDR register.

Register 5-9: NVMADDRINV: Flash Address Invert Register

Write inverts selected bits in NVMADDR, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in NVMADDR**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in NVMADDR register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: NVMADDRINV = 0x00008001 will invert bits 15 and 0 in NVMADDR register.

Register 5-10: NVMDATA: Flash Program Data Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
NVMDATA<31:24>							
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
NVMDATA<23:16>							
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
NVMDATA<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
NVMDATA<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **NVMDATA<31:0>:** Flash Programming Data bits

Note: These bits are only reset by a Power-on Reset (POR).

Register 5-11: NVMSRCADDR: Source Data Address Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
NVMSRCADDR<31:24>							
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
NVMSRCADDR<23:16>							
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
NVMSRCADDR<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	r-x	r-x
NVMSRCADDR<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-3 **NVMSRCADDR<31:0>:** Source Data Address bits
The system physical address of the data to be programmed into the Flash when NVMCON.NVMOP is set to perform row programming

bit 2-0 **Reserved:** Write ‘0’; ignore read

Register 5-12: IFS1: Interrupt Flag Status Register 1(1)

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	USBIF	FCEIF
bit 31							bit 24

r-x	r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	DMA3IF	DMA2IF	DMA1IF	DMA0IF
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RTCCIF	FSCMIF	I2C2MIF	I2C2SIF	I2C2BIF	U2TXIF	U2RXIF	U2EIF
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SPI2RXIF	SPI2TXIF	SPI2EIF	CMP2IF	CMP1IF	PMPIF	AD1IF	CNIF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
HC = Hardware clear	HS = Hardware set	C = Clearable by software	
-n = Bit Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 24 **FCEIF:** Flash Control Event Interrupt Flag bit

1 = Interrupt request has occurred
0 = No interrupt request has occurred

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the NVM.

Register 5-13: IEC1: Interrupt Enable Control Register 1(1)

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	USBIE	FCEIE
bit 31							bit 24

r-x	r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	DMA3IE	DMA2IE	DMA1IE	DMA0IE
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RTCCIE	FSCMIE	I2C2MIE	I2C2SIE	I2C2BIE	U2TXIE	U2RXIE	U2EIE
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SPI2RXIE	SPI2TXIE	SPI2EIE	CMP2IE	CMP1IE	PMPIE	AD1IE	CNIE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
HC = Hardware clear	HS = Hardware set	C = Clearable by software	
-n = Bit Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 24 **FCEIE:** Flash Control Event Interrupt Enable bit

1 = Interrupt is enabled
0 = Interrupt is disabled

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the NVM.

Register 5-14: IPC11: Interrupt Priority Control Register 11(1)

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—					
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—					
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	USBIP<2:0>			USBIS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	FCEIP<2:0>			FCEIS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
HC = Hardware clear	HS = Hardware set	C = Clearable by software	
-n = Bit Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 4-2 **FCEIP<2:0>:** Flash Control Event INterrupt Priority bits
- 111 = Interrupt Priority is 7
- 110 = Interrupt Priority is 6
- 101 = Interrupt Priority is 5
- 100 = Interrupt Priority is 4
- 011 = Interrupt Priority is 3
- 010 = Interrupt Priority is 2
- 001 = Interrupt Priority is 1
- 000 = Interrupt is disabled

bit 1-0 **FCEIS<1:0>:** Flash Control Event Interrupt Subpriority bits
- 11 = Interrupt Subpriority is 3
- 10 = Interrupt Subpriority is 2
- 01 = Interrupt Subpriority is 1
- 00 = Interrupt Subpriority is 0

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the NVM.

5.2.1 NVMCON Register

The NVMCON register is the control register for Flash program/erase operations. This register selects whether an erase or program operation can be performed and is used to start the program or erase cycle.

The NVMCON register is shown in Register 5-1. The lower byte of NVMCON configures the type of NVM operation that will be performed. A summary of the NVMCON setup values for various program and erase operations is given in Table 5-3.

Table 5-3: NVMCON Register Values

Operation	NVMCON Value
Page Erase	0x8004
Program Word	0x8001
Program Row	0x8003
NOP	0x8000

5.2.2 NVMADDR Register

The NVM Address register selects the row for Flash memory writes, the address location for word writes, and the page address for Flash memory erase operations.

5.2.3 NVMKEY Register

NVMKEY is a write-only register that is used to prevent accidental writes/erasures of Flash or EEPROM memory. To start a programming or an erase sequence, the following steps must be taken in the exact order shown:

1. Write 0xAA996655 to NVMKEY.
2. Write 0x556699AA to NVMKEY.

After this sequence, only the next transaction on the peripheral bus is allowed to write the NVMCON register. In most cases, the user will simply need to set the NVMWR bit in the NVMCON register to start the program or erase cycle. Interrupts should be disabled during the unlock sequence.

5.2.4 NVMSRCADDR Register

The NVM Source Address register selects the source data buffer address in SRAM for performing row programming operations.

Note:	The address must be word aligned.

5.3 RTSP OPERATION

RTSP allows the user code to modify Flash program memory contents. The device Flash memory is divided into two logical Flash partitions: the Program Flash Memory (PFM), and the Boot Flash Memory (BFM). The last page in Boot Flash Memory contains the DEBUG Page, which is reserved for use by the debugger tool while debugging.

The program Flash array for the PIC32MX device is built up of a series of rows. A row contains 128 32-bit instruction words or 512 bytes. A group of 8 rows compose a page; which, therefore, contains 8 × 512 = 4096 bytes or 1024 instruction words. A page of Flash is the smallest unit of memory that can be erased at a single time. The program Flash array can be programmed in one of two ways:

- Row programming, with 128 instruction words at a time.
- Word programming, with 1 instruction word at a time.

The CPU stalls (waits) until the programming operation is finished. The CPU will not execute any instruction, or respond to interrupts, during this time. If any interrupts occur during the programming cycle, they remain pending until the cycle completes.

Note:	A minimum V_{DD} requirement for Flash erase and write operations is required. Refer to the specific device data sheet for further details.

5.4 LOCK-OUT FEATURE

5.4.1 NVMWREN

A number of mechanisms exist within the device to ensure that inadvertent writes to program Flash do not occur. The NVMWREN (NVMCON<14>) bit should be zero, unless the software intends to write to the program Flash. When NVMWREN = 1, the Flash write control bit NVMWR (NVMCON<15>) is writable and the Flash LVD circuit is enabled.

5.4.2 NVMKEY

In addition to the write protection provided by the NVMWREN bit, an unlock sequence needs to be performed before the NVMCON.NVMWR bit can be set. If the NVMWR (NVMCON<15>) bit is not set on the next peripheral bus transaction (read or write), NVMWR is locked and the unlock sequence must be restarted.

5.4.3 Unlock Sequence

To unlock Flash operations, steps 3 through 8 below must be performed exactly in order. If the sequence is not followed exactly, NVMWR is not set.

1. Suspend or disable all initiators that can access the Peripheral Bus and interrupt the unlock sequence, e.g., DMA and interrupts.
2. Set NVMWREN (NVMCON<14>) to allow writes to NVMWR and set NVMOP<3:0> (NVMCON<3:0>) to the desired operation with a single store instruction.
3. Load 0xAA996655 to CPU register X.
4. Load 0x556699AA to CPU register Y.
5. Load 0x00008000 to CPU register Z.
6. Store CPU register X to NVMKEY.
7. Store CPU register Y to NVMKEY.
8. Store CPU register Z to NVMCONSET.
9. Wait for NVMWR (NVMCON<15>) bit to be clean.
10. Clear the NVMWREN (NVMCON<14>) bit.
11. Check the NVMERR (NVMCON<13>) and LVDERR (NVMCON<12>) bits to ensure that the program/erase sequence completed successfully.

When the NVMWR bit is set, the program/erase sequence starts and the CPU is unable to execute from Flash memory for the duration of the sequence.

Example 5-1: Unlock Example

```
unsigned int NVMUnlock (unsigned int nvmop)
{
    unsigned int status;

    // Suspend or Disable all Interrupts
    asm volatile (“di %0” : “=r” (status));

    // Enable Flash Write/Erase Operations and Select
    // Flash operation to perform
    NVMCON = nvmop;

    // Write Keys
    NVMKEY = 0xAA996655;
    NVMKEY = 0x556699AA;

    // Start the operation using the Set Register
    NVMCONSET = 0x8000;

    // Wait for operation to complete
    while (NVMCON & 0x8000);

    // Restore Interrupts
    if (status & 0x00000001
      asm volatile (“ei”);
    else
      asm volatile (“di”);

    // Return NVMERR and LVDERR Error Status Bits
    return (NVMCON & 0x3000)
}
```

5.5 WORD PROGRAMMING SEQUENCE

The smallest block of data that can be programmed in a single operation is one 32-bit word. The data to be programmed must be written to the NVMDATA register, and the address of the word must be loaded into the NVMADDR register before the programming sequence is initiated. The instruction word at the location pointed to by NVMADDR is then programmed.

A program sequence comprises the following steps:

1. Write 32-bit data to be programmed to the NVMDATA register.
2. Load the NVMADDR register with the address to be programmed.
3. Run the unlock sequence using the Word Program command (see **Section 5.4.3 “Unlock Sequence”**).

The program sequence completes, and the NVMWR (NVMCON<15>) bit is cleared by hardware.

Example 5-2: Word Program Example

```
unsigned int NVMWriteWord (void* address, unsigned int data)
{
    unsigned int res;

    // Load data into NVMDATA register
    NVMDATA = data;

    // Load address to program into NVMADDR register
    NVMADDR = (unsigned int) address;

    // Unlock and Write Word
    res = NVMUnlock (0x4001);

    // Return Result
    return res;

}
```


5.6 ROW PROGRAMMING SEQUENCE

The largest block of data that can be programmed is 1 row, which equates to 512 bytes of data. The row of data must first be loaded into a buffer in SRAM. The NVMADDR register then points to the Flash address where the Flash controller will start programming the row of data.

Note:	The controller ignores the sub-row address bits and always starts programming at the beginning of a row.

A row program sequence comprises the following steps:

1. Write the entire row of data to be programmed into system SRAM. The source address must be word aligned.
2. Set the NVMADDR register with the start address of the Flash row to be programmed.
3. Set the NVMSRCADDR register with the physical source address from step 1.
4. Run the unlock sequence using the Row Program command (see **Section 5.4.3 “Unlock Sequence”**).
5. The program sequence completes, and the NVMWR (NVMCON<15>) bit is cleared by hardware.

Example 5-3: Row Program Example

```
unsigned int NVMWriteRow (void* address, void* data)
{
    unsigned int res;

    // Set NVMADDR to Start Address of row to program
    NVMADDR = (unsigned int) address;

    // Set NVMSRCADDR to the SRAM data buffer Address
    NVMSRCADDR = (unsigned int) data;

    // Unlock and Write Row
    res = NVMUnlock(0x4003);

    // Return Result
    return res;

}
```

5.7 PAGE ERASE SEQUENCE

A page erase performs an erase of a single page of either PFM or BFM, which equates to 4096 bytes. The page to be erased is selected using the NVMADDR register.

Note:	The lower bits of the address are ignored in page selection.

A page of Flash can only be erased if its associated page write protection is not enabled.

- All BFM pages are affected by the Boot write protection Configuration bit.
- PFM pages are affected by the Program Flash write protection Configuration bits.

If in Mission mode, the application must not be executing from the erased page.

A page erase sequence comprises the following steps:

1. Set the NVMADDR register with the address of the page to be erased.
2. Run the unlock sequence using the desired Erase command (see **Section 5.4.3 “Unlock Sequence”**).

The erase sequence completes and the NVMWR (NVMCON<15>) bit is cleared by hardware.

Example 5-4: Page Erase Example

```
unsigned int NVMErasePage(void* address)
{
    unsigned int res;

    // Set NVMADDR to the Start Address of page to erase
    NVMADDR = (unsigned int) address;

    // Unlock and Erase Page
    res = NVMUnlock(0x4004);

    // Return Result
    return res;

}
```


5.8 PROGRAM FLASH MEMORY ERASE SEQUENCE

It is possible to erase the entire PFM area. This mode leaves the Boot Flash intact and is intended to be used by a field upgradeable device.

The Program Flash can be erased if all pages in the Program Flash are not write-protected.

Note:	The application must NOT be executing from the PFM address range.

A PFM erase sequence comprises the following steps:

1. Run the unlock sequence using the Program Flash memory Erase command (see **Section 5.4.3 “Unlock Sequence”**)

The erase sequence completes and the NVMWR (NVMCON<15>) bit is cleared by hardware.

Example 5-5: Program Flash Erase Example

```
unsigned int NVMErasePFM(void)
{
    unsigned int res;

    // Unlock and Erase Program Flash
    res = NVMUnlock(0x4005);

    // Return Result
    return res;

}
```

5.9 OPERATION IN POWER-SAVING AND DEBUG MODES

Note:	In this manual, a distinction is made between a power mode as it is used in a specific module, and a power mode as it is used by the device, e.g., Sleep mode of the Comparator and SLEEP mode of the CPU. To indicate which type of power mode is intended, uppercase and lowercase letters (Sleep, Idle, Debug) signify a module power mode, and all uppercase letters (SLEEP, IDLE, DEBUG) signify a device power mode.

5.9.1 Operation in SLEEP Mode

When the PIC32MX device enters SLEEP mode, the system clock is disabled. The Flash controller does not function in SLEEP mode. If entry into SLEEP mode occurs while an NVM operation is in progress, then the device will not go into sleep until the NVM operation is complete.

5.9.2 Operation in IDLE Mode

IDLE mode has no effect on the Flash controller module when a programming operation is active. The CPU continues to be stalled until the programming operation completes.

5.9.3 Operation in DEBUG Mode

The Flash controller does not provide debug Freeze capability and therefore has no effect on the Flash controller module when a programming operation is active. The CPU continues to be stalled until the programming operation completes. Interrupting the normal programming sequence could cause the device to latch-up. The only exception to this is the NVMKEY unlock sequence, which is suspended when in DEBUG mode, allowing the user to single step through the unlock sequence.

Note:	The FRZ bit is readable and writable only when the CPU is executing in Debug Exception mode. In all other modes, the FRZ bit reads as '0'. If FRZ bit is changed during DEBUG mode, the new value does not take effect until the current Debug Exception mode is exited and re-entered. During the Debug Exception mode, the FRZ bit reads the state of the peripheral when entering DEBUG mode.

5.10 EFFECTS OF VARIOUS RESETS

5.10.1 Device Reset

Only the NVMCON bits for NVMWREN and LVDSTAT are reset on a device Reset. All other SFR bits are only reset by POR. The state of the NUMKEY is reset by a device Reset however.

5.10.2 Power-on Reset

All Flash controller registers are forced to their reset states upon a POR.

5.10.3 Watchdog Timer Reset

All Flash controller registers are unchanged upon a Watchdog Timer Reset

5.11 INTERRUPTS

The Flash Controller has the ability to generate an interrupt reflecting the events that occur during the programming operations. The following interrupt can be generated:

Flash Control Event Interrupt FCEIF (IFS1<24>

The interrupt flag must be cleared in software.The Flash Controller is enabled as a source of interrupt via the following respective Flash Controller interrupt enable bit:

- FCEIE (IE1<24>)

The interrupt priority-level bits and interrupt subpriority-level bits must also be configured:

- FCEIP (IPC11<2:0> and FCEIS (IPC11<1:0>)

Refer to **Section 8. "Interrupts"** in this manual for details.

5.11.1 Interrupt Configuration

The Flash Controller module has the following dedicated interrupt flag bit.

- FCEIF

The Flash Controller module also has the following corresponding interrupt enable/mask bit:

- FCEIE

The bits determine the source of an interrupt and enable or disable an individual interrupt source. Note that all the interrupt sources for a specific Flash Controller module share just one interrupt vector.

Note that the FCEIF bit will be set without regard to the state of the corresponding enable bit, and the IF bit can be polled by software if desired.

The FCEIE bit is used to define the behavior of the Vector Interrupt Controller (VIC) when a corresponding FCEIF bit is set. When the corresponding IE bit is clear the VIC module does not generate a CPU interrupt for the event. If the IE bit is set, the VIC module will generate an interrupt to the CPU when the corresponding IF bit is set (subject to the priority and subpriority as outlined in the following paragraphs).

It is the responsibility of the user's software routine that services a particular interrupt to clear the appropriate Interrupt Flag bit before the service routine is complete.

The priority of the Flash Controller module can be set independently with the FCEIP<2:0> bits. This priority defines the priority group to which the interrupt source is assigned. The priority groups range from a value of 7 (the highest priority), to a value of 0, which does not generate an interrupt. An interrupt being serviced is preempted by an interrupt in a higher priority group.

The subpriority bits allow setting the priority of a interrupt source within a priority group. The values of the subpriority, FCEIS<1:0>, range from 3 (the highest priority), to 0 the lowest priority. An interrupt with the same priority group but having a higher subpriority value, does not preempt a lower subpriority interrupt that is in progress.

The priority group and subpriority bits allow more than one interrupt source to share the same priority and subpriority. If simultaneous interrupts occur in this configuration the natural order of the interrupt sources within a priority/subpriority–group pair determine the interrupt generated. The natural priority is based on the vector numbers of the interrupt sources. The lower the vector number, the higher the natural priority of the interrupt. Any interrupts that are overridden by natural order generate their respective interrupts based on priority, subpriority, and natural order, after the interrupt flag for the current interrupt is cleared.

After an enabled interrupt is generated, the CPU jumps to the vector assigned to that interrupt. The vector number for the interrupt is the same as the natural order number. Then the CPU begins executing code at the vector address. The user's code at this vector address should perform any application specific operations, clear the FCEIF interrupt flag, and then exit. Refer to **Section 8. "Interrupts"** in this manual for the vector address table details and more information on interrupts.

Table 5-4: UART Interrupt Vectors for Various Offsets with EBASE = 0x8000:0000

Interrupt	Vector/ Natural Order	IRQ Number	Vector Address IntCtl.VS = 0x01	Vector Address IntCtl.VS = 0x02	Vector Address IntCtl.VS = 0x04	Vector Address IntCtl.VS = 0x08	Vector Address IntCtl.VS = 0x10
FCE	43	56	80000760	80000CC0	80001780	80002D00	80005800

5.12 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the PIC32MX device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the Flash module include the following:

Title	Application Note #
No related application notes at this time.	N/A

Note: Please visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the PIC32MX family of devices.

5.13 REVISION HISTORY

Revision A (September 2007)

This is the initial released version of this document.

Revision B (October 2007)

Updated document to remove Confidential status.

Revision C (April 2008)

Revised status to Preliminary; Revised U-0 to r-x.

Revision D (June 2008)

Revised Register 5-1, bit 14 NVMWREN; Add footnote 1 to Registers 5-12-5-14; Add note to Section 5.3; Revise Section 5.4.1; Revised Example 5-1; Change Reserved bits “Maintain as” to “Write”.

Section 6. Oscillators

HIGHLIGHTS

This section of the manual contains the following topics:

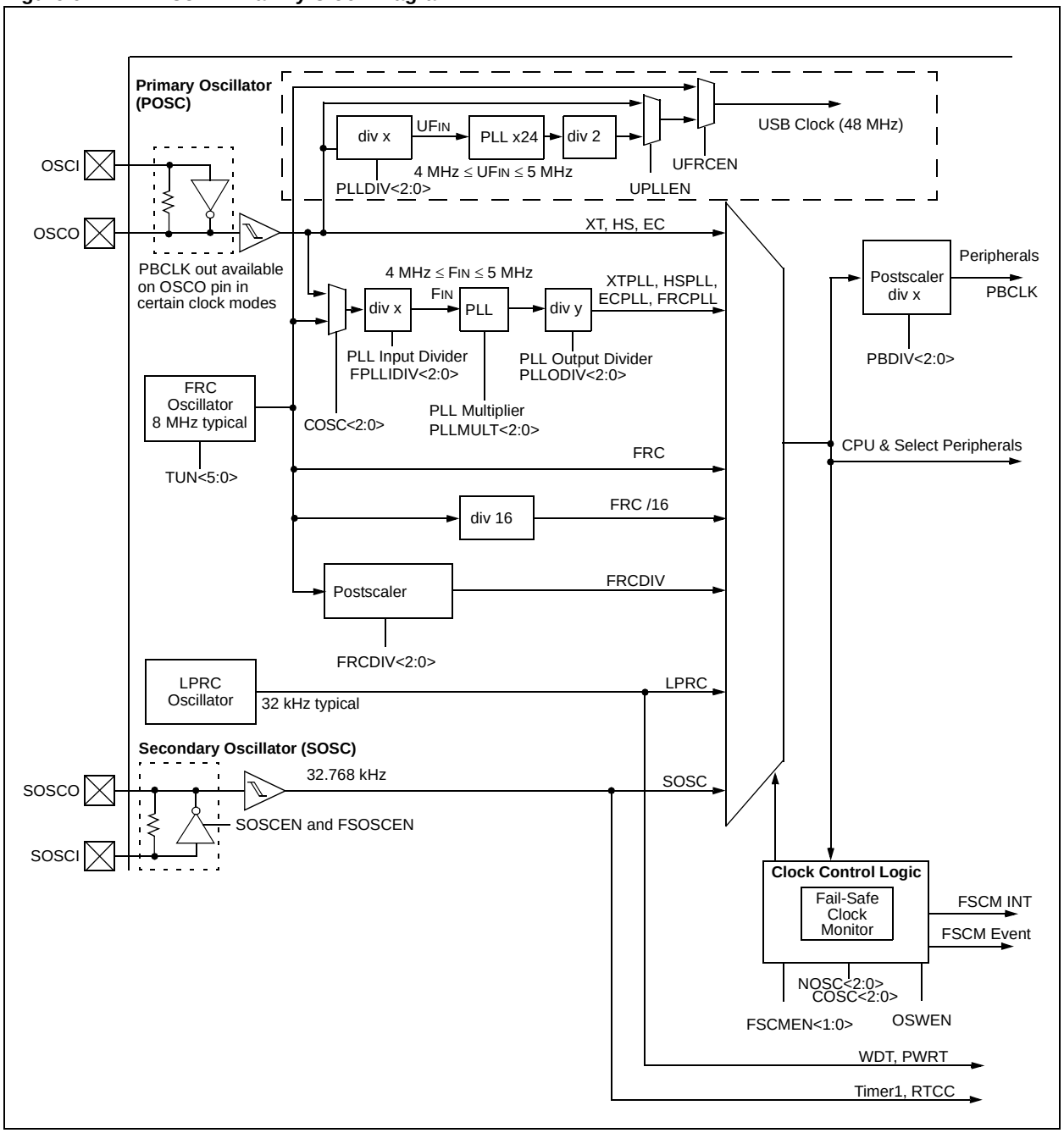
6.1 INTRODUCTION

This section describes the PIC32MX oscillator system and its operation. The PIC32MX oscillator system has the following modules and features:

- A total of four external and internal oscillator options as clock sources
- On-chip PLL with user-selectable input divider, multiplier, and output divider to boost operating frequency on select internal and external oscillator sources
- On-chip user selectable divisor postscaler on select oscillator sources
- Software-controllable switching between various clock sources
- A Fail-Safe Clock Monitor (FSCM) that detects clock failure and permits safe application recovery or shutdown

A simplified diagram of the oscillator system is shown in Figure 6-1.

Figure 6-1: PIC32MX Family Clock Diagram



6.2 CONTROL REGISTERS

The Oscillator module consists of the following Special Function Registers (SFRs):

- OSCCON: Control Register for the Oscillator module

  OSCCONCLR, OSCCONSET, OSCCONINV: Atomic Bit Manipulation Write-only Registers for OSCCON register

- OSCTUN: FRC Tuning Register for the Oscillator module

  OSCTUNCLR, OSCTUNSET, OSCTUNINV: Atomic Bit Manipulation Write-only Registers for OSCTUN register

The Oscillator module also has the following associated bits for interrupt control:

- Interrupt Flag Status bits (IFS1<14>) for Clock Fail FSCMIF in IFS1 Interrupt register
- Interrupt Enable Control bits (IEC1<14>) for Clock Fail FSCMIE in IEC1 Interrupt register
- Interrupt Priority Control bits (FSCMIP<12:10>) for Clock Fail in IPC8 Interrupt register
- Interrupt Subpriority Control bits (FSCMIP<9:8>) for Clock Fail in IPC8 Interrupt register

The following tables provide brief summaries of Oscillator-module-related registers. Corresponding registers appear after the summaries, followed by a detailed description of each register.

Table 6-1: Oscillators SFR Summary

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
OSCCON	31:24	—	—	PLLODIV<2:0>			FRCDIV<2:0>		
	23:16	—	SOSCRDY	—	PBDIV<1:0>		PLLMULT<2:0>		
	15:8	—	COSC<2:0>			—	NOSC<2:0>		
	7:0	CLKLOCK	ULOCK	LOCK	SLPEN	CF	UFRCEN	SOSCEN	OSWEN
OSCCONCLR	31:0	Write clears selected bits in OSCCON, read yields undefined value							
OSCCONSET	31:0	Write sets selected bits in OSCCON, read yields undefined value							
OSCCONINV	31:0	Write inverts selected bits in OSCCON, read yields undefined value							
OSCTUN	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	—	—	TUN<5:0>					
OSCTUNCLR	31:0	Write clears selected bits in OSCTUN, read yields undefined value							
OSCTUNSET	31:0	Write sets selected bits in OSCTUN, read yields undefined value							
OSCTUNINV	31:0	Write inverts selected bits in OSCTUN, read yields undefined value							
WDTCON		—	—	—	—	—	—	—	—
		—	—	—	—	—	—	—	—
	15:8	ON	—	—	—	—	—	—	—
		—	SWDTPS<4:0>					—	WDTCLR
WDTCONCLR	31:0	Write clears selected bits in WDTCON, read yields an undefined value							
WDTCONSET	31:0	Write sets selected bits in WDTCON, read yields an undefined value							
WDTCONINV	31:0	Write inverts selected bits in WDTCON, read yields an undefined value							
IFS1	31:24	—	—	—	—	—	—	USBIF	FCEIF
	23:16	—	—	—	—	DMA3IF	DMA2IF	DMA1IF	DMA0IF
	15:8	RTCCIF	FSCMIF	I2C2MIF	I2C2SIF	I2C2BIF	U2TXIF	U2RXIF	U2EIF
	7:0	SPI2RXIF	SPI2TXIF	SPI2EIF	CMP2IF	CMP1IF	PMPIF	AD1IF	CNIF
IFS1CLR	31:0	Clears the selected bits in IFS1, read yields undefined value							
IFS1SET	31:0	Sets the selected bits in IFS1, read yields undefined value							
IFS1INV	31:0	Inverts the selected bits in IFS1, read yields undefined value							

Table 6-1: Oscillators SFR Summary (Continued)

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
IEC1	31:24	—	—	—	—	—	—	USBIE	FCEIE
	23:16	—	—	—	—	DMA3IE	DMA2IE	DMA1IE	DMA0IE
	15:8	RTCCIE	FSCMIE	I2C2MIE	I2C2SIE	I2C2BIE	U2TXIE	U2RXIE	U2EIE
	7:0	SPI2RXIE	SPI2TXIE	SPI2EIE	CMP2IE	CMP1IE	PMPIE	AD1IE	CNIE
IEC1CLR	31:0	Write clears the selected bits in IE0, read yields undefined value							
IEC1SET	31:0	Write sets the selected bits in IE0, read yields undefined value							
IEC1INV	31:0	Write inverts the selected bits in IE0, read yields undefined value							
IPC8	31:24	—	—	—	DMA0IP<2:0>			DMA0IS<1:0>	
	23:16	—	—	—	RTCCIP<2:0>			RTCCIS<1:0>	
	15:8	—	—	—	FSCMIP<2:0>			FSCMIS<1:0>	
	7:0	—	—	—	I2C2IP<2:0>			I2C2IS<1:0>	
IPC8CLR	31:0	Write clears the selected bits in IPC8, read yields undefined value							
IPC8SET	31:0	Write sets the selected bits in IPC8, read yields undefined value							
IPC8INV	31:0	Write inverts the selected bits in IPC8, read yields undefined value							
DEVCFG1	31:24	—	—	—	—	—	—	—	—
	23:16	FWDTEN	—	—	FWDTPS<4:0>				
	15:8	FCKSM<1:0>		FPBDIV<1:0>		—	OSCIOFNC	POSCMD<1:0>	
	7:0	IESO	—	FSOSCEN	—	—	FNOSC<2:0>		
DEVCFG2	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	FPLLODIV<2:0>		
	15:8	FUPLLEN	—	—	—	—	FUPLLIDIV<2:0>		
	7:0	—	FPLLMULT<2:0>			—	FPLLIDIV<2:0>		

Register 6-1: OSCCON: Oscillator Control Register

r-x	r-x	R/W-x	R/W-x	R/W-x	R/W-0	R/W-0	R/W-1
—	—	PLLODIV<2:0>			FRCDIV<2:0>		
bit 31							bit 24

r-x	R-0	r-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	SOSCRDY	—	PBDIV<1:0>		PLLMULT<2:0>		
bit 23							bit 16

r-x	R-0	R-0	R-0	r-x	R/W-x	R/W-x	R/W-x
—	COSC<2:0>			—	NOSC<2:0>		
bit 15							bit 8

R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-x	R/W-0
CLKLOCK	ULOCK	LOCK	SLPEN	CF	UFRCEN	SOSCEN	OSWEN
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-30 **Reserved:** Write ‘0’; ignore read

bit 29-27 **PLLODIV<2:0>:** Output Divider for PLL

111 = PLL output divided by 256
110 = PLL output divided by 64
101 = PLL output divided by 32
100 = PLL output divided by 16
011 = PLL output divided by 8
010 = PLL output divided by 4
001 = PLL output divided by 2
000 = PLL output divided by 1

Note: On Reset these bits are set to the value of the FPLLODIV configuration bits (DEVCFG2<18:16>)

bit 26-24 **FRCDIV<2:0>:** Fast Internal RC Clock Divider bits

111 = FRC divided by 256
110 = FRC divided by 64
101 = FRC divided by 32
100 = FRC divided by 16
011 = FRC divided by 8
010 = FRC divided by 4
001 = FRC divided by 2 (default setting)
000 = FRC divided by 1

bit 23 **Reserved:** Write ‘0’; ignore read

bit 22 **SOSCRDY:** Secondary Oscillator Ready Indicator bit

1 = Indicates that the Secondary Oscillator is running and is stable
0 = Secondary oscillator is either turned off or is still warming up

bit 21 **Reserved:** Write ‘0’; ignore read

Register 6-1: OSCCON: Oscillator Control Register

bit 20-19 **PBDIV<1:0>:** Peripheral Bus Clock Divisor

11 = PBCLK is SYSCLK divided by 8(default)
10 = PBCLK is SYSCLK divided by 4
01 = PBCLK is SYSCLK divided by 2
00 = PBCLK is SYSCLK divided by 1

Note: On Reset these bits are set to the value of the Configuration bits (DEVCFG1<13:12>).

bit 18-16 **PLLMULT<2:0>:** PLL Multiplier bits

111 = Clock is multiplied by 24
110 = Clock is multiplied by 21
101 = Clock is multiplied by 20
100 = Clock is multiplied by 19
011 = Clock is multiplied by 18
010 = Clock is multiplied by 17
001 = Clock is multiplied by 16
000 = Clock is multiplied by 15

Note: On Reset these bits are set to the value of the FPLLMULT Configuration bits (DEVCFG2<6:4>).

bit 15 **Reserved:** Write '0'; ignore read

bit 14-12 **COSC<2:0>:** Current Oscillator Selection bits

111 = Fast Internal RC Oscillator divided by OSCCON<FRCDIV> bits
110 = Fast Internal RC Oscillator divided by 16
101 = Low-Power Internal RC Oscillator (LPRC)
100 = Secondary Oscillator (SOSC)
011 = Primary Oscillator with PLL module (XTPLL, HSPLL or ECPLL)
010 = Primary Oscillator (XT, HS or EC)
001 = Fast RC Oscillator with PLL module via Postscaler (FRCPLL)
000 = Fast RC Oscillator (FRC)

Note: On Reset these bits are set to the value of the FNOSC Configuration bits (DEVCFG1<2:0>).

bit 11 **Reserved:** Write '0'; ignore read

bit 10-8 **NOSC<2:0>:** New Oscillator Selection bits

111 = Fast Internal RC Oscillator divided by OSCCON<FRCDIV> bits
110 = Fast Internal RC Oscillator divided by 16
101 = Low-Power Internal RC Oscillator (LPRC)
100 = Secondary Oscillator (SOSC)
011 = Primary Oscillator with PLL module (XTPLL, HSPLL or ECPLL)
010 = Primary Oscillator (XT, HS or EC)
001 = Fast Internal RC Oscillator with PLL module via Postscaler (FRCPLL)
000 = Fast Internal RC Oscillator (FRC)

Note: On Reset these bits are set to the value of the FNOSC Configuration bits (DEVCFG1<2:0>).

bit 7 **CLKLOCK:** Clock Selection Lock Enable bit

If FSCM is enabled (FCKSM1 =1):
1 = Clock and PLL selections are locked.
0 = Clock and PLL selections are not locked and may be modified
If FSCM is disabled (FCKSM1 =0):

Note: Clock and PLL selections are never locked and may be modified.

bit 6 **ULOCK:** USB PLL Lock Status bit

1 = Indicates that the USB PLL module is in lock or USB PLL module start-up timer is satisfied
0 = Indicates that the USB PLL module is out of lock or USB PLL module start-up timer is in progress or USB PLL is disabled

bit 5 **LOCK:** PLL Lock Status bit

1 = PLL module is in lock or PLL module start-up timer is satisfied
0 = PLL module is out of lock, PLL start-up timer is running or PLL is disabled

bit 4 **SLPEN:** SLEEP Mode Enable bit

1 = Device will enter SLEEP mode when a WAIT instruction is executed
0 = Device will enter IDLE mode when a WAIT instruction is executed

Register 6-1: OSCCON: Oscillator Control Register

bit 3 **CF:** Clock Fail Detect bit

1 = FSCM (Fail Safe Clock Monitor) has detected a clock failure
0 = No clock failure has been detected

bit 2 **UFRCEN:** USB FRC Clock Enable bit

1 = Enable FRC as the clock source for the USB clock source
0 = Use the primary oscillator or USB PLL as the USB clock source

bit 1 **SOSCEN:** 32.768 kHz Secondary Oscillator (SOSC) Enable bit

1 = Enable Secondary Oscillator
0 = Disable Secondary Oscillator

Note: On Reset this bit is set to the value of the FSOSCEN Configuration bit (DEVCFG1<5>).

bit 0 **OSWEN:** Oscillator Switch Enable bit

1 = Initiate an oscillator switch to selection specified by NOSC2:NOSC0 bits
0 = Oscillator switch is complete

Register 6-2: OSCCONCLR: Oscillator Control Clear Register

Write clears selected bits in OSCCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in OSCCON**

A write of '1' in one or more bit positions clears the corresponding bit(s) in OSCCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `OSCCONCLR = 0x00000101` will clear bits 8 and 0 in OSCCON register.

Register 6-3: OSCCONSET: Oscillator Control Set Register

Write sets selected bits in OSCCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in OSCCON**

A write of '1' in one or more bit positions sets the corresponding bit(s) in OSCCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `OSCCONSET = 0x00000101` will set bits 8 and 0 in OSCCON register.

Register 6-4: OSCCONINV: Oscillator Control Invert Register

Write inverts selected bits in OSCCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in OSCCON**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in OSCCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `OSCCONINV = 0x00000101` will invert bits 8 and 0 in OSCCON register.

Register 6-5: OSCTUN: FRC Tuning Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	TUN<5:0>					
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31:6 **Reserved:** Write ‘0’; ignore read

bit 5-0 **TUN<5:0>:** FRC Oscillator Tuning bits

011111 = Maximum frequency.
011110 =
•
000001 =
000000 = Center frequency. Oscillator runs at calibrated frequency.
111111 =
•
100001 =
100000 = Minimum frequency.

Register 6-6: OSCTUNCLR: FRC Tuning Clear Register

Write clears selected bits in OSCTUN, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in OSCTUN**

A write of '1' in one or more bit positions clears the corresponding bit(s) in OSCTUN register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `OSCTUNCLR = 0x00000021` will clear bits 5 and 0 in OSCTUN register.

Register 6-7: OSCTUNSET: FRC Tuning Set Register

Write sets selected bits in OSCTUN, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in OSCTUN**

A write of '1' in one or more bit positions sets the corresponding bit(s) in OSCTUN register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `OSCTUNSET = 0x00000021` will set bits 5 and 0 in OSCTUN register.

Register 6-8: OSCTUNINV: FRC Tuning Invert Register

Write inverts selected bits in OSCTUN, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in OSCTUN**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in OSCTUN register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `OSCTUNINV = 0x00000021` will invert bits 5 and 0 in OSCTUN register.

Register 6-9: WDTCON: Watchdog Timer Control Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	r-x	r-x	r-x	r-x	r-x	r-x	r-x
ON	—	—	—	—	—	—	—
bit 15							bit 8

r-x	R-x	R-x	R-x	R-x	R-x	r-0	R/W-0
—	WDTPS<4:0>					—	WDTCLR
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 15 **ON:** Watchdog Timer Enable bit

1 = Enables the WDT if it is not enabled by the device configuration
0 = Disable the WDT if it was enabled in software

Note 1: A read of this bit will result in a ‘1’ if the WDT is enabled by the device configuration or by software.

2: The LPRC oscillator will automatically be enabled when this bit is set.

3: When using 1:1 PBCLK divisor, the user’s software should not read/write the peripheral’s SFRs in the SYSCLK cycle immediately following the instruction that clears the module’s ON bit.

Register 6-10: WDTCONCLR: Comparator Control Clear Register

Write clears selected bits in WDTCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clear selected bits in WDTCON**

A write of '1' in one or more bit positions clears the corresponding bit(s) in WDTCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `WDTCONCLR = 0x00008001` clears bits 15 and 0 in WDTCON register.

Register 6-11: WDTCONSET: Comparator Control Set Register

Write sets selected bits in WDTCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Set selected bits in WDTCON**

A write of '1' in one or more bit positions sets the corresponding bit(s) in WDTCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `WDTCONSET = 0x00008001` sets bits 15 and 0 in WDTCON register.

Register 6-12: WDTCONINV: Comparator Control Invert Register

Write inverts selected bits in WDTCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in WDTCON**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in WDTCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `WDTCONINV = 0x00008001` inverts bits 15 and 0 in WDTCON register.

Register 6-13: IFS1: Interrupt Flag Status Register(1)

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	USBIF	FCEIF
bit 31							bit 24

r-x	r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	DMA3IF	DMA2IF	DMA1IF	DMA0IF
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RTCCIF	FSCMIF	I2C2MIF	I2C2SIF	I2C2BIF	U2TXIF	U2RXIF	U2EIF
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SPI2RXIF	SPI2TXIF	SPI2EIF	CMP2IF	CMP1IF	PMPIF	AD1IF	CNIF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 14 **FSCMIF:** Fail-Safe Clock Monitor Interrupt Flag bit
1 = Interrupt request has occured
0 = No interrupt request has a occurred

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the oscillator.

Register 6-14: IEC1: Interrupt Enable Control Register(1)

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	USBIE	FCEIE
bit 31							bit 24

r-x	r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	DMA3IE	DMA2IE	DMA1IE	DMA0IE
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RTCCIE	FSCMIE	I2C2MIE	I2C2SIE	I2C2BIE	U2TXIE	U2RXIE	U2EIE
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SPI2RXIE	SPI2TXIE	SPI2EIE	CMP2IE	CMP1IE	PMPIE	AD1IE	CNIE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 14 **FSCMIE:** Fail-Safe Clock Monitor Interrupt Enable bit

1 = Interrupt is enabled
0 = Interrupt is disabled

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the oscillator.

Register 6-15: IPC8: Interrupt Priority Control Register 8(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	DMA0IP<2:0>			DMA0IS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	RTCCIP<2:0>			RTCCIS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	FSCMIP<2:0>			FSCMIS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	I2C2IP<2:0>			I2C2IS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 12-10 **FSCMIP<2:0>:** Fail-Safe Clock Monitor Interrupt Priority bits

111 = Interrupt priority is 7
110 = Interrupt priority is 6
101 = Interrupt priority is 5
100 = Interrupt priority is 4
011 = Interrupt priority is 3
010 = Interrupt priority is 2
001 = Interrupt priority is 1
000 = Interrupt is disabled

bit 9-8 **FSCMIS<1:0>:** Fail-Safe Clock Monitor Interrupt Subpriority bits

11 = Interrupt subpriority is 3
10 = Interrupt subpriority is 2
01 = Interrupt subpriority is 1
00 = Interrupt subpriority is 0

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the oscillator.

Register 6-16: DEVCFG1 Boot Configuration Register

r-1	r-1	r-1	r-1	r-1	r-1	r-1	r-1
—	—	—	—	—	—	—	—
bit 31							bit 24

R/P-1	R/P-1	r-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1
FWDTEN	—	—	FWDTPS4	FWDTPS3	FWDTPS2	FWDTPS1	FWDTPS0
bit 23							bit 16

R/P-1	R/P-1	R/P-1	R/P-1	r-1	R/P-1	R/P-1	R/P-1
FCKSM<1:0>		FPBDIV<1:0>		—	OSCIOFNC	POSCMD<1:0>	
bit 15							bit 8

R/P-1	r-1	R/P-1	r-1	r-1	R/P-1	R/P-1	R/P-1
IESO	—	FSOSCEN	—	—	FNOSC2	FNOSC1	FNOSC0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Unimplemented:** Maintain ‘1’

bit 15-14 **FCKSM<1:0>:** Fail-safe Clock Monitor (FSCM) and Clock Switch Configuration bits
- 1x = FSCM and Clock Switching are disabled
- 01 = Clock Switching is enabled, FSCM is disabled
- 00 = Clock Switching and FSCM are enabled

bit 10 **OSCIOFNC:** CLKO Enable Configuration bit
- 1 = CLKO output signal active on the OSCO pin; primary oscillator must be disabled or configured for the External Clock mode (EC) for the CLKO to be active (POSCMD<1:0> = 11 or = 00)
- 0 = CLKO output disabled

bit 13-12 **FPBDIV<1:0>:** Peripheral Bus Clock Divisor Default Value bits
- 11 = PBCLK is SYSCLK divided by 8
- 10 = PBCLK is SYSCLK divided by 4
- 01 = PBCLK is SYSCLK divided by 2
- 00 = PBCLK is SYSCLK divided by 1

bit 11 **Unimplemented:** Maintain as ‘1’

bit 9-8 **POSCMD<1:0>:** Primary Oscillator Configuration bits
- 11 = Primary Oscillator Disabled
- 10 = HS mode
- 01 = XT Mode
- 00 = EC Mode

bit 7 **IESO:** Internal External Clock Switch Over Select bit
- 1 = Internal External Clock Switch Over Mode Enabled. Two-Speed Start-up mode.
- 0 = Internal External Clock Switch Over Mode Disabled. Single-Speed Start-up mode.

bit 5 **FSOSCEN:** Secondary Oscillator Enable bit
- 1 = Enable secondary oscillator
- 0 = Disable secondary oscillator

Register 6-16: DEVCFG1 Boot Configuration Register

bit 2-0 **FNOSC<2:0>:** CPU Clock Oscillator Select bits

111 = Fast RC Oscillator with divide-by-N (FRCDIV)
110 = FRC Divided by 16 (FRCDIV16)
101 = Low-Power RC Oscillator (LPRC)
100 = Secondary Oscillator (SOSC)
011 = Primary Oscillator with PLL (XTPLL, HSPLL, or ECPLL)
010 = Primary Oscillator without PLL (XT, HS, or EC)
001 = Fast RC Oscillator with PLL
000 = Fast RC Oscillator (FRC)

Register 6-17: DEVCFG2 Boot Configuration Register

r-1	r-1	r-1	r-1	r-1	r-1	r-1	r-1
—	—	—	—	—	—	—	—
bit 31							bit 24

r-1	r-1	r-1	r-1	r-1	R/P-1	R/P-1	R/P-1
—	—	—	—	—	FPLLODIV<2:0>		
bit 23							bit 16

R/P-1	r-1	r-1	r-1	r-1	R/P-1	R/P-1	R/P-1
FUPLLEN	—	—	—	—	FUPLLIDIV<2:0>		
bit 15							bit 8

U-1	R/P-1	R/P-1	R/P-1	U-1	R/P-1	R/P-1	R/P-1
—	FPLLMULT<2:0>			—	FPLLIDIV<2:0>		
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 18-16 **FPLLODIV<2:0>:** Default postscaler for PLL.
- 111 = PLL output divided by 256
- 110 = PLL output divided by 64
- 101 = PLL output divided by 32
- 100 = PLL output divided by 16
- 011 = PLL output divided by 8
- 010 = PLL output divided by 4
- 001 = PLL output divided by 2
- 000 = PLL output divided by 1 (default setting)

bit 15 **FUPLLEN:** USB PLL Enable bit
- 00 = Enable USB PLL
- 00 = Disable and bypass USB PLL

bit 10-8 **FUPLLIDIV<2:0>:** PLL Input Divider bits
- 000 = 1x divider
- 001 = 2x divider
- 010 = 3x divider
- 011 = 4x divider
- 100 = 5x divider
- 101 = 6x divider
- 110 = 10x divider
- 111 = 12x divider

bit 6-4 **FPLLMULT<2:0>:** Default PLL Multiplier Value bits
- 111 = 24x multiplier
- 110 = 21x multiplier
- 101 = 20x multiplier
- 100 = 19x multiplier
- 011 = 18x multiplier
- 010 = 17x multiplier
- 001 = 16x multiplier
- 000 = 15x multiplier

Register 6-17: DEVCFG2 Boot Configuration Register

bit 2-0 **FPLLIDIV<2:0>:** Default PLL Input Divider Value bits

111 = Divide by 12
110 = Divide by 10
101 = Divide by 6
100 = Divide by 5
011 = Divide by 4
010 = Divide by 3
001 = Divide by 2
000 = Divide by 1

6.3 OPERATION: CLOCK GENERATION AND CLOCK SOURCES

The PIC32MX family has multiple internal clocks that are derived from internal or external clock sources. Some of these clock sources have Phase Locked Loops (PLLs), programmable output divider, or input divider to scale the input frequency to suit the application. The clock source can be changed on the fly by software. The oscillator control register is locked by hardware, it must be unlocked by a series of writes before software can perform a clock switch.

There are three main clocks in the PIC32MX device

- The System clock (SYSCLK) used by CPU and some peripherals
- The Peripheral Bus Clock (PBCLK) used by most peripherals
- The USB Clock (USBCLK) used by USB peripheral

The PIC32MX clocks are derived from one of the following sources:

- Primary Oscillator (POSC) on the OSCI and OSCO pins
- Secondary Oscillator (SOSC) on the SOSCI and SOSCO pins
- Internal Fast RC Oscillator (FRC)
- Internal Low-Power RC Oscillator (LPRC)

Each of the clock sources has unique configurable options, such as a PLL, input divider, and/or output divider, that are detailed in their respective sections.

There are up to four internal clocks depending on the specific device. The clocks are derived from the currently selected oscillator source.

Note:	Clock sources for peripherals that use external clocks, such as the RTC and Timer1, are covered in their respective sections.

6.3.1 System Clock (SYSCLK) Generation

The SYSCLK is primarily used by the CPU and select peripherals such as DMA, Interrupt Controller, and Prefetch Cache. The SYSCLK is derived from one of the four clock sources: POSC, SOSC, FRC, and LPRC. Some of the clock sources have specific clock multipliers and/or divider options. No clock scaling is applied other than the user specified values. The SYSCLK source is selected by the device configuration and can be changed by software during operation. The ability to switch clock sources during operation allows the application to reduce power consumption by reducing the clock speed. Refer to Table for a list of SYSCLK sources.

Table 6-2: Clock Selection Configuration Bit Values

Oscillator Mode	Oscillator Source	POSCMD<1:0>	FNOSC2: FNOSC0	ADIV	Notes
Fast RC Oscillator with Postscaler (FRCDIV)	Internal	xx	111		**1, 2**
Fast RC Oscillator divided by 16 (FRCDIV16)	Internal	xx	110		**1**
Low-Power RC Oscillator (LPRC)	Internal	xx	101		**1**
Secondary (Timer1/RTCC) Oscillator (SOSC)	Secondary	xx	100		**1**
Primary Oscillator (HS) with PLL Module (HSPLL)	Primary	10	011		**3**
Primary Oscillator (XT) with PLL Module (XTPLL)	Primary	01	011		**3**
Primary Oscillator (EC) with PLL Module (ECPLL)	Primary	00	011		**3**
Primary Oscillator (HS)	Primary	10	010		
Primary Oscillator (XT)	Primary	01	010		

Note 1: OSCO pin function as PBCLK out or Digital I/O is determined by the OSCIOFNC Configuration bit. When the pin is not required by the Oscillator mode it may be configured for one of these options.

2: Default Oscillator mode for an unprogrammed (erased) device.

3: When using the PLL modes the input divider must be chosen such that resulting frequency applied to the PLL is in the range of 4 MHz to 5 MHz.

Table 6-2: Clock Selection Configuration Bit Values (Continued)

Oscillator Mode	Oscillator Source	POSCMD<1:0>	FNOSC2: FNOSC0	ADIV	Notes
Primary Oscillator (EC)	Primary	00	010		
Fast RC Oscillator with PLL Module (FRCPLL)	Internal	10	001		1
Fast RC Oscillator (FRC)	Internal	xx	000		1

Note 1: OSCO pin function as PBCLK out or Digital I/O is determined by the OSCIOFNC Configuration bit. When the pin is not required by the Oscillator mode it may be configured for one of these options.

2: Default Oscillator mode for an unprogrammed (erased) device.

3: When using the PLL modes the input divider must be chosen such that resulting frequency applied to the PLL is in the range of 4 MHz to 5 MHz.

6.3.1.1 Primary Oscillator (POSC)

The POSC has six operating modes, as summarized in Table 6-3: High Speed (HS), External Resonator (XT), and the External Clock (EC) mode make up the first three modes. These modes can each be combined with a PLL module to form the last three modes: High Speed PLL (HSPLL), External Resonator PLL (XTPLL), and External Clock (ECPLL). Figures 6-2 through 6-4 show various POSC configurations.

The primary oscillator is connected to the OSCI and OSCO pins of the device family. The primary oscillator can be configured for an external clock input or an external crystal or resonator.

The XT, XTPLL, HS, and HSPLL modes are external crystal or resonator controller oscillator modes. The XT and HS modes are functionally very similar. The primary difference is the gain of the internal inverter of the oscillator circuit (see Figure 6-2). The XT mode is a medium power, medium frequency mode and has medium inverter gain. HS mode is higher power and provides the highest oscillator frequencies and has the highest inverter gain. OSCO provides crystal/resonator feedback in both XT and HS Oscillator modes and hence is not available for use as a input or output in these modes. The XTPLL and HSPLL modes have a Phase Locked Loop (PLL) with user selectable input divider, multiplier, and output divider to provide a wide range of output frequencies. The oscillator circuit will consume more current when the PLL is enabled.

The External Clock modes, EC and ECPLL, allow the system clock to be derived from an external clock source. The EC/ECPLL modes configure the OSCI pin as a high-impedance input that can be driven by a CMOS driver. The external clock can be used to drive the system clock directly (EC) or the ECPLL module with prescale and postscaler can be used to change the input clock frequency (ECPLL). The External Clock mode also disables the internal feedback buffer allowing the OSCO pin to be used for other functions. In the External Clock mode the OSCO pin can be used as an additional device I/O pin (see Figure 6-4) or a PBCLK output pin (see Figure 6-3).

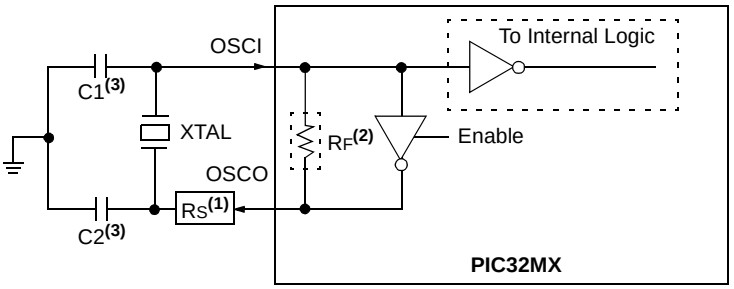
Note: When using the PLL modes the input divider must be chosen such that resulting frequency applied to the PLL is in the range of 4 MHz to 5 MHz.

Table 6-3: Primary Oscillator Operating Modes

Oscillator Mode	Description
HS	10 MHz-40 MHz crystal, high speed crystal
XT	3.5 MHz-10 MHz resonator, crystal or resonator
EC	External clock input
HSPLL	Crystal, PLL enabled
XTPLL	Crystal resonator, PLL enabled
ECPLL	External clock input, PLL enabled

Note: The clock applied to the CPU after applicable prescalers, postscalers, and PLL multipliers must not exceed the maximum allowable processor frequency.

Figure 6-2: Crystal or Ceramic Resonator Operation (XT, XTPLL, HS, or HSPLL Oscillator Mode)



Note 1: A series resistor, Rs, may be required for AT strip cut crystals.

2: The internal feedback resistor, RF, is typically in the range of 2 to 10 MΩ.

3: See **6.8.3.1 "Determining the Best Values for Oscillator Components"**.

Figure 6-3: External Clock Input Operation With Clock-Out (EC, ECPLL Mode)

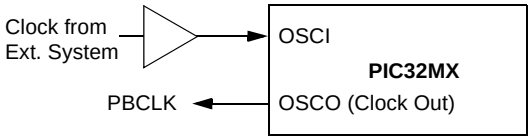
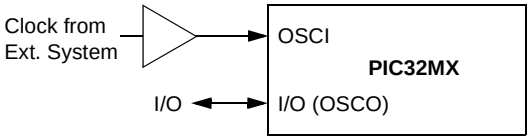


Figure 6-4: External Clock Input Operation with no Clock-Out (EC, ECPLL Mode)



6.3.1.1.1 Primary Oscillator (POSC) Configuration

To configure the POSC the following steps should be performed:

1. Select POSC as the default oscillator in the device Configuration register DEVCFG1 by setting FNOSC<2:0> = '010' without PLL or '011' with PLL
2. Select the desired mode HS, XT, or EC, using POSCMD<1:0> in DEVCFG1.
3. If the PLL is to be used:
   a) Select the appropriate Configuration bits for the PLL input divider to scale the input frequency to be between 4 MHz and 5 MHz using FPLLIDIV<2:0> in DEVCFG2.
   b) Select the desired PLL multiplier ratio using FPLLMULT<2:0>) in DEVCFG2.
   c) At runtime, select the desired PLL output divider using PLLODIV (OSCCON<29:27>) to provide the desired clock frequency. The default value is set by DEVCFG1.

6.3.1.1.2 Oscillator Start-up Timer

In order to ensure that a crystal oscillator (or ceramic resonator) has started and stabilized, an Oscillator Start-up Timer (OST) is provided. The OST is a simple 10-bit counter that counts 1024 TOSC cycles before releasing the oscillator clock to the rest of the system. This time-out period is designated as TOST. The amplitude of the oscillator signal must reach the VIL and VIH thresholds for the oscillator pins before the OST can begin to count cycles.

The TOST interval is required every time the oscillator has to restart (i.e., on POR, BOR and wake-up from SLEEP mode). The Oscillator Start-up Timer is applied to the MS and HS modes for the primary oscillator, as well as the secondary oscillator, see **6.3.1.2 "Secondary Oscillator (SOSC)"**.

6.3.1.1.3 System Clock Phase Locked Loop (PLL)

The system clock PLL provides a user configurable input divider, multiplier, and output divider which can be used with the XT, HS and EC primary oscillator modes and with the Internal Fast RC Oscillator (FRC) mode to create a variety of clock frequencies from a single clock source.

The Input divider, multiplier, and output divider control initial value bits are contained in the in the DEVCFG2 device Configuration register. The multiplier and output divider bits are also contained in the OSCCON register. As part of a device Reset, values from the device configuration register DEVCFG2 are copied to the OSCCON register. This allows the user to preset the input divider to provide the appropriate input frequency to the PLL and set an initial PLL multiplier when programming the device. At runtime the multiplier, divider and output divider can be changed by software to scale the clock frequency to suit the application. The PLL input divider cannot be changed at run time. This is to prevent applying an input frequency outside the specified limits to the PLL.

To configure the PLL the following steps are required:

1. Calculate the PLL input divider, PLL multiplier, and PLL output divider values.
2. Set the PLL input divider and the initial PLL multiplier value in the DEVCFG2 register when programming the part.
3. At runtime the PLL multiplier and PLL output divider can be changed to suit the application.

Combinations of PLL input divider, multiplier and output divider provide a combined multiplier of approximately 0.006 to 24 times the input frequency. For reliable operation the output of the PLL module must not exceed the maximum clock frequency of the device. The PLL input divider value should be chosen to limit the input frequency to the PLL to the range of 4 MHz to 5 MHz.

Due to the time required for the PLL to provide a stable output, a Status bit LOCK (OSCCON<5>) is provided. When the clock input to the PLL is changed, this bit is driven low ('0'). After the PLL has achieved a lock or the PLL start-up timer has expired, the bit is set. The bit will be set upon the expiration of the timer even if the PLL has not achieved a lock.

Table 6-4: Net Multiplier Output for Selected PLL and Output Divider Values

Multiplier	Output Divider	Net Multiplication factor	PLLODIV <2:0>	PLLMULT <2:0>	Multiplier	Postscaler	Net Multiplication factor	PLLODIV <2:0>	PLLMULT <2:0>
15	1	15	‘000’	‘000’	15	16	.938	‘100’	‘000’
16	1	16	‘000’	‘001’	16	16	1	‘100’	‘001’
17	1	17	‘000’	‘010’	17	16	1.063	‘100’	‘010’
18	1	18	‘000’	‘011’	18	16	1.125	‘100’	‘011’
19	1	19	‘000’	‘100’	19	16	1.188	‘100’	‘100’
20	1	20	‘000’	‘101’	20	16	1.250	‘100’	‘101’
21	1	21	‘000’	‘110’	21	16	1.313	‘100’	‘110’
24	1	24	‘000’	‘111’	24	16	1.5	‘100’	‘111’
15	2	7.5	‘001’	‘000’	15	32	.4688	‘101’	‘000’
16	2	8	‘001’	‘001’	16	32	.5	‘101’	‘001’
17	2	8.5	‘001’	‘010’	17	32	.5313	‘101’	‘010’
18	2	9	‘001’	‘011’	18	32	.5625	‘101’	‘011’
19	2	9.5	‘001’	‘100’	19	32	.5938	‘101’	‘100’
20	2	10	‘001’	‘101’	20	32	.6250	‘101’	‘101’
21	2	10.5	‘001’	‘110’	21	32	.6563	‘101’	‘110’
24	2	12	‘001’	‘111’	24	32	.7500	‘101’	‘111’
15	4	3.75	‘010’	‘000’	15	64	.234	‘110’	‘000’
16	4	4	‘010’	‘001’	16	64	.250	‘110’	‘001’
17	4	4.25	‘010’	‘010’	17	64	.266	‘110’	‘010’
18	4	4.5	‘010’	‘011’	18	64	.281	‘110’	‘011’
19	4	4.75	‘010’	‘100’	19	64	.297	‘110’	‘100’
20	4	5	‘010’	‘101’	20	64	.313	‘110’	‘101’
21	4	5.25	‘010’	‘110’	21	64	.328	‘110’	‘110’
24	4	6	‘010’	‘111’	24	64	.375	‘110’	‘111’
15	8	1.875	‘011’	‘000’	15	256	.05859	‘111’	‘000’
16	8	2	‘011’	‘001’	16	256	.06250	‘111’	‘001’
17	8	2.125	‘011’	‘010’	17	256	.06641	‘111’	‘010’
18	8	2.250	‘011’	‘011’	18	256	.07031	‘111’	‘011’
19	8	2.375	‘011’	‘100’	19	256	.07422	‘111’	‘100’
20	8	2.5	‘011’	‘101’	20	256	.07813	‘111’	‘101’
21	8	2.625	‘011’	‘110’	21	256	.08203	‘111’	‘110’
24	8	3	‘011’	‘111’	24	256	.09375	‘111’	‘111’

6.3.1.1.4 USB PLL Lock Status

The ULOCK bit (OSCCON<6>) is a read-only status bit that indicates the lock status of the USB PLL. It is automatically set after the typical time delay for the PLL to achieve lock, also designated as TLOCK. If the PLL does not stabilize properly during start-up, LOCK may not reflect the actual status of PLL lock, nor does it detect when the PLL loses lock during normal operation.

The ULOCK bit is cleared at a Power-on Reset. It remains clear when any clock source not using the PLL is selected.

Refer to the Electrical Characteristics section in the specific device data sheet for further information on the PLL lock interval.

6.3.1.1.5 Primary Oscillator Start-up from SLEEP Mode

To ensure reliable wake-up from SLEEP, care must be taken to properly design the primary oscillator circuit. This is because the load capacitors have both partially charged to some quiescent value and phase differential at wake-up is minimal. Thus, more time is required to achieve stable oscillation. Remember also that low voltage, high temperatures and the lower frequency clock modes also impose limitations on loop gain, which in turn, affects start-up.

Each of the following factors increases the start-up time:

- Low-frequency design (with a Low Gain Clock mode)
- Quiet environment (such as a battery operated device)
- Operating in a shielded box (away from the noisy RF area)
- Low voltage
- High temperature
- Wake-up from SLEEP mode

6.3.1.2 Secondary Oscillator (SOSC)

The Secondary Oscillator (SOSC) is designed specifically for low-power operation with a external 32.768 kHz crystal. The oscillator is located on the SOSCO and SOSCI device pins and serves as a secondary crystal clock source for low-power operation. It can also drive Timer1 and/or the Real-Time Clock/Calendar module for Real-Time Clock applications.

6.3.1.2.1 Enabling the SOSC Oscillator

The SOSC is hardware enabled by the FSOSCEN Configuration bit (DEVCFG1<5>). Once SOSC is enabled, software can control it by modifying SOSCEN bit (OSCCON<1>). Setting SOSCEN enables the oscillator; the SOSCO and SOSCI pins are controlled by the oscillator and cannot be used for port I/O or other functions.

Note:	An unlock sequence is required before a write to OSCCON can occur. Refer to **6.3.5.2 "Oscillator Switching Sequence"** for more information.

The Secondary Oscillator requires a warm-up period before it can be used as a clock source. When the oscillator is enabled, a warm-up counter increments to 1024. When the counter expires the SOSCRDY (OSCCON<22>) is set to '1'. Refer to **6.3.1.1.2 "Oscillator Start-up Timer"**.

6.3.1.2.2 SOSC Continuous Operation

The SOSC is always enabled when SOSCEN (OSCCON<1>) is set. Leaving the oscillator running at all times allows a fast switch to the 32 kHz system clock for lower power operation. Returning to the faster main oscillator will still require an oscillator start-up time if it is a crystal type source and/or uses the PLL (see **6.3.1.1.2 "Oscillator Start-up Timer"**).

In addition, the oscillator will need to remain running at all times for Real-Time Clock applications and may be required for Timer1. Refer to **Section 14. "Timers"** and **Section 29. "Real-Time Clock and Calendar"** for further details.

Example 6-1: Enabling the SOSC

```
SYSKEY = 0x0;                       // ensure OSCCON is locked
SYSKEY = 0xAA996655;                // Write Key1 to SYSKEY
SYSKEY = 0x556699AA;                // Write Key2 to SYSKEY
                                    // OSCCON is now unlocked
                                    // make the desired change
OSCCONSET = 2;                      // enable SOSC
                                    // Relock the SYSKEY
SYSKEY = 0x0;                       // Write any value other than Key1 or Key2
                                    // OSCCON is relocked
```

6.3.1.3 Internal Fast RC Oscillator (FRC)

The FRC oscillator is a fast (8 MHz nominal), user trimmable, internal RC oscillator with user selectable input divider, PLL multiplier, and output divider. See device data sheet for more information about the FRC oscillator.

6.3.1.3.1 FRC Postscaler Mode (FRCDIV)

Users are not limited to the nominal 8 MHz FRC output if they wish to use the fast internal oscillator as a clock source. An additional FRC mode, FRCDIV, implements a selectable output divider that allows the choice of a lower clock frequency from 7 different options, plus the direct 8 MHz output. The output divider is configured using the FRCDIV<2:0> bits (OSCCON<26:24>). Assuming a nominal 8 MHz output, available lower frequency options range from 4 MHz (divide-by-2) to 31 kHz (divide-by-256). The range of frequencies allows users the ability to save power at any time in an application by simply changing the FRCDIV bits. The FRCDIV mode is selected whenever the COSC bits (OSCCON<14:12>) are '111'.

6.3.1.3.2 FRC Oscillator with PLL Mode (FRCPLL)

The output of the FRC may also be combined with a user selectable PLL multiplier and output divider to produce a SYSCLK across a wide range of frequencies. The FRC PLL mode is selected whenever the COSC bits (OSCCON<14:12>) are '001'. In this mode the PLL input divider is forced to '2' to provide a 4 MHz input to the PLL. The desired PLL multiplier and output divider values can be chosen to provide the desired device frequency

6.3.1.3.3 Oscillator Tune Register (OSCTUN)

The FRC Oscillator Tuning register OSCTUN allows the user to fine tune the FRC oscillator over a range of approximately ±12% (typical). Each bit increment or decrement changes the factory calibrated frequency of the FRC oscillator by a fixed amount. Refer to the Electrical Characteristics section of the specific device data sheet for additional information on the available tuning range.

6.3.1.4 Internal Low-Power RC Oscillator (LPRC)

The LPRC oscillator is separate from the FRC. It oscillates at a nominal frequency of 31.25 kHz. The LPRC oscillator is the clock source for the Power-up Timer (PWRT), Watchdog Timer (WDT), Fail Safe Clock Monitor (FSCM) and PLL reference circuits. It may also be used to provide a low-frequency clock source option for the device in those applications where power consumption is critical, and timing accuracy is not required.

6.3.1.4.1 Enabling the LPRC Oscillator

Since it serves the PWRT clock source, the LPRC oscillator is disabled at Power-on Reset whenever the on-board voltage regulator is enabled. After the PWRT expires, the LPRC oscillator will remain on if any one of the following is true:

- The Fail-Safe Clock Monitor is enabled.
- The WDT is enabled.
- The LPRC oscillator is selected as the system clock (COSC2:COSC0 = 100).

If none of the above is true, the LPRC will shut off after the PWRT expires.

6.3.2 Peripheral Bus Clock (PBCLK) Generation

The PBCLK is derived from the System Clock (SYSCLK) divided by PBDIV<1:0> (OSCCON<20:19>). The PBCLK Divisor bits PBDIV<1:0> allow postscalers of 1:1, 1:2, 1:4, and 1:8. Refer to the individual peripheral module section(s) for information regarding which bus a specific peripheral uses.

Notes: When the PBDIV divisor is set to a ratio of '1:1' the SYSCLK and PBCLK are equivalent in frequency. The PBCLK frequency is never greater than the processor clock frequency.

The effect of changing the PBCLK frequency on individual peripherals should be taken into account when selecting or changing the PBDIV value.

Performing back-to-back operations on PBCLK peripheral registers when the PB divisor is not set at 1:1 will cause the CPU to stall for a number of cycles. This stall occurs to prevent an operation from occurring before the pervious one has completed. The length of the stall is determined by the ratio of the CPU and PBCLK and synchronizing time between the two busses.

Changing the PBCLK frequency has no effect on the SYSCLK peripherals operation.

6.3.3 USB Clock (USBCLK) generation

The USBCLK can be derived from 8MHz internal FRC oscillator, 48MHz POSC, or 96MHz PLL from POSC. For normal operation, the USB module requires exact 48MHz clock. When using 96MHz PLL, the output is internally divided to obtain 48MHz clock. The FRC clock source is used to detect USB activity and bring USB module out of SUSPEND mode. Once USB module is out of SUSPEND mode, it starts using any of two 48MHz clock sources. The internal FRC oscillator is not used for normal USB module operation.

6.3.3.0.1 USB Clock Phase Locked Loop (UPLL)

The USB clock PLL provides a user configurable input divider which can be used with the XT, HS and EC primary oscillator modes and with the Internal Fast RC Oscillator (FRC) mode to create a variety of clock frequencies from a clock source. The actual source must be able to provide stable clock as required by the USB specifications.

The UPLL enable and Input divider bits are contained in the in the DEVCFG2 device configuration register. The input to the UPLL must be limited to 4MHz only. Appropriate input divider must be selected to ensure that the UPLL input is 4MHz.

To configure the UPLL the following steps are required:

1. Enable USB PLL by setting UPLLEN bit in DEVCFG2 register.
2. Based on the source clock, calculate the UPLL input divider value such that the PLL input is 4MHz
3. Set the UPLL input divider UPLLIDIV bits in the DEVCFG2 register when programming the part.

6.3.3.0.2 USB PLL Lock Status

The ULOCK bit (OSCCON<6>) is a read-only status bit that indicates the lock status of the USB PLL. It is automatically set after the typical time delay for the PLL to achieve lock, also designated as T_{ULOCK}. If the PLL does not stabilize properly during start-up, ULOCK may not reflect the actual status of PLL lock, nor does it detect when the PLL loses lock during normal operation.

The ULOCK bit is cleared at a Power-on Reset. It remains clear when any clock source not using the PLL is selected.

Refer to the Electrical Characteristics section in the specific device data sheet for further information on the USB PLL lock interval.

6.3.3.0.3 Using Internal FRC Oscillator with USB

The internal 8MHz FRC oscillator is available as a clock source to detect any USB activity during USB SUSPEND mode and bring the module out of the SUSPEND mode. To enable FRC for USB usage, the UFRCEN bit (OSCCON<2>) must be set ‘1’ before putting USB module to SUSPEND mode.

6.3.4 Two Speed Start-up

Two Speed Start-up mode can be used to reduce the device start-up latency when using all external crystal POSC modes including PLL. Two-Speed Start-up uses the FRC clock as the SYSCLK source until the Primary Oscillator (POSC) has stabilized. After the user selected oscillator has stabilized, the clock source will switch to POSC. This allows the CPU to begin running code, at a lower speed, while the oscillator is stabilizing. When the POSC has met the start-up criteria an automatic clock switch occurs to switch to POSC. This mode is enabled by the device configuration bits FCKSM<1:0> (DEVCFG1<15:14>). Two-Speed Start-up operates after a Power-on Reset (POR) or exit from SLEEP. Software can determine the oscillator source currently in use by reading the COSC<2:0> bits in the OSCCON register.

Note:	The Watchdog Timer (WDT), if enabled, will continue to count at the same rate regardless of the SYSCLK frequency. Care must be taken to service the WDT during Two-Speed Start-up, taking into account the change in SYSCLK.

6.3.5 Fail-Safe Clock Monitor Operation

The Fail-Safe Clock Monitor (FSCM) is designed to allow continued device operation if the current oscillator fails. It is intended for use with the Primary Oscillator (POSC) and automatically switches to the FRC oscillator if a POSC failure is detected. The switch to the Fast Internal RC Oscillator (FRC) oscillator allows continued device operation and the ability to retry the POSC or to execute code appropriate for a clock failure.

The FSCM mode is controlled by the FCKSM<1:0> bits in the device configuration DEVCFG1. Any of the POSC modes can be used with FSCM.

When a clock failure is detected with FSCM enabled and the FSCM Interrupt Enable bit FSCMIE (IEC1<14>) set, the clock source will be switched from POSC to FRC. An Oscillator Fail interrupt will be generated, with the CF bit (OSCCON<3>) set. This interrupt has a user settable priority FSCMIP<2:0> (IPC8<12:10>) and subpriority FSCMIS<1:0> (IPC8<9:8>). The clock source will remain FRC until a device Reset or a clock switch is performed. Failure to enable the FSCM interrupt will not inhibit the actual clock switch.

The FSCM module takes the following actions when switching to the FRC oscillator:

1. The COSC bits (OSCCON<14:12>) are loaded with '000'.
2. The CF (OSCCON<3>) bit is set to indicate the clock failure
3. The OSWEN control bit (OSCCON<0>) is cleared to cancel any pending clock switches.

To enable FSCM the following steps should be performed:

1. Enable the FSCM in the Device Configuration register DEVCFG1 by configuring the FCKSM<1:0> bits.

   01 = Clock Switching is enabled, FSCM is disabled
   00 = Clock Switching and FSCM are enabled
2. Select the desired mode HS, XT, or EC using FNOSC<2:0> in DEVCFG1.
3. Select POSC as the default oscillator in the device configuration DEVCFG1 by configuring FNOSC<2:0> = 010 without PLL or '011' with PLL.

If the PLL is to be used:

1. Select the appropriate Configuration bits for the PLL input divider to scale the input frequency to be between 4 MHz and 5 MHz using FPLLIDIV<2:0> (DEVCFG2<2:0>).
2. Select the desired PLL multiplier using FPLLMULT<2:0> (DEVCFG2<6:4>).
3. Select the desired PLL output divider using FPLLODIV<2:0> (DEVCFG2<18:16>).

If a FSCM interrupt is desired when a FSCM event occurs, the following steps should be performed during start-up code:

1. Clear the FSCM interrupt bit FSCMIF (IFS1<14>)
2. Set the Interrupt priority FSCMIP<2:0> (IPC8<12:10>) and subpriority FSCMIS<1:0> (IPC8<9:8>).
3. Set the FSCM Interrupt Enable bit FSCMIE (IEC1<14>)

Note:	The Watchdog Timer, if enabled, will continue to count at the same rate regardless of the SYSCLK frequency. Care must be taken to service the WDT after a Fail-Safe Clock Monitor event, taking into account the change in SYSCLK.

6.3.5.1 FSCM Delay

On a POR, BOR or wake from SLEEP mode event, a nominal delay (TFSCM) may be inserted before the FSCM begins to monitor the system clock source. The purpose of the FSCM delay is to provide time for the oscillator and/or PLL to stabilize when the Power-up Timer (PWRT) is not utilized. The FSCM delay will be generated after the internal System Reset signal, SYSRST, has been released. Refer to **Section 7. "Resets"** for FSCM delay timing information.

The TFSCM interval is applied whenever the FSCM is enabled and the HS, HSPLL, XT, XTPLL, or SOSC Oscillator modes are selected as the system clock.

Note:	Please refer to the Electrical Characteristics section of the specific device data sheet for TFSCM specification values.

6.3.5.2 FSCM and Slow Oscillator Start-up

If the chosen device oscillator has a slow start-up time coming out of POR, BOR or SLEEP mode, it is possible that the FSCM delay will expire before the oscillator has started. In this case, the FSCM will initiate a clock failure trap. As this happens, the COSC bits (OSCCON<14:12>) are loaded with the FRC oscillator selection. This will effectively shut off the original oscillator that was trying to start. Software can detect a clock failure using a Interrupt Service Routine (SFR) or by polling the clock fail interrupt flag FSCMIF (IFS1<14>).

6.3.5.3 FSCM and WDT

The FSCM and the WDT both use the LPRC oscillator as their time base. In the event of a clock failure, the WDT is unaffected and continues to run.

6.3.6 Clock Switching Operation

With few limitations, applications are free to switch between any of the four clock sources (POSC, SOSC, FRC and LPRC) under software control and at any time. To limit the possible side effects that could result from this flexibility, PIC32MX devices have a safeguard lock built into the switch process.

Note:	Primary Oscillator mode has three different submodes (XT, HS and EC) which are determined by the POSCMD Configuration bits in DEVCFG1. While an application can switch to and from Primary Oscillator mode in software, it cannot switch between the different primary submodes without reprogramming the device.

Note:	The device will not permit direct switching between PLL clock sources. The user should not change the PLL multiplier values or postscaler values when running from the affected PLL source. To perform either of the above clock switching functions, the clock switch should be performed in two steps. The clock source should first be switched to a non-PLL source, such as FRC, and then switched to the desired source. This requirement only applies to PLL-based clock sources.

6.3.6.1 Enabling Clock Switching

To enable clock switching, the FCKSM1 Configuration bit (DEVCFG1<15>) must be programmed to '0'. (Refer to **Section 32. "Configuration"** for further details.) If the FCKSM1 Configuration bit is unprogrammed (= 1), the clock switching function and Fail-Safe Clock Monitor function are disabled. This is the default setting.

The NOSC control bits (OSCCON<10:8>) do not control the clock selection when clock switching is disabled. However, the COSC bits (OSCCON<14:12>) will reflect the clock source selected by the FNOSC Configuration bits.

The OSWEN control bit (OSCCON<0>) has no effect when clock switching is disabled. It is held at '0' at all times.

6.3.6.2 Oscillator Switching Sequence

At a minimum, performing a clock switch requires the following sequence:

1. If desired, read the COSC<2:0> bits (OSCCON<14:12>) to determine the current oscillator source.
2. Perform the unlock sequence to allow a write to the OSCCON register. The unlock sequence has critical timing requirements and should be performed with interrupts and DMA disabled.
3. Write the appropriate value to the NOSC<2:0> control bits (OSCCON<10:8>) for the new oscillator source.
4. Set the OSWEN bit (OSCCON<0>) to initiate the oscillator switch.
5. Optionally perform the lock sequence to lock the OSCCON. The lock sequence must be performed separately from any other operation.

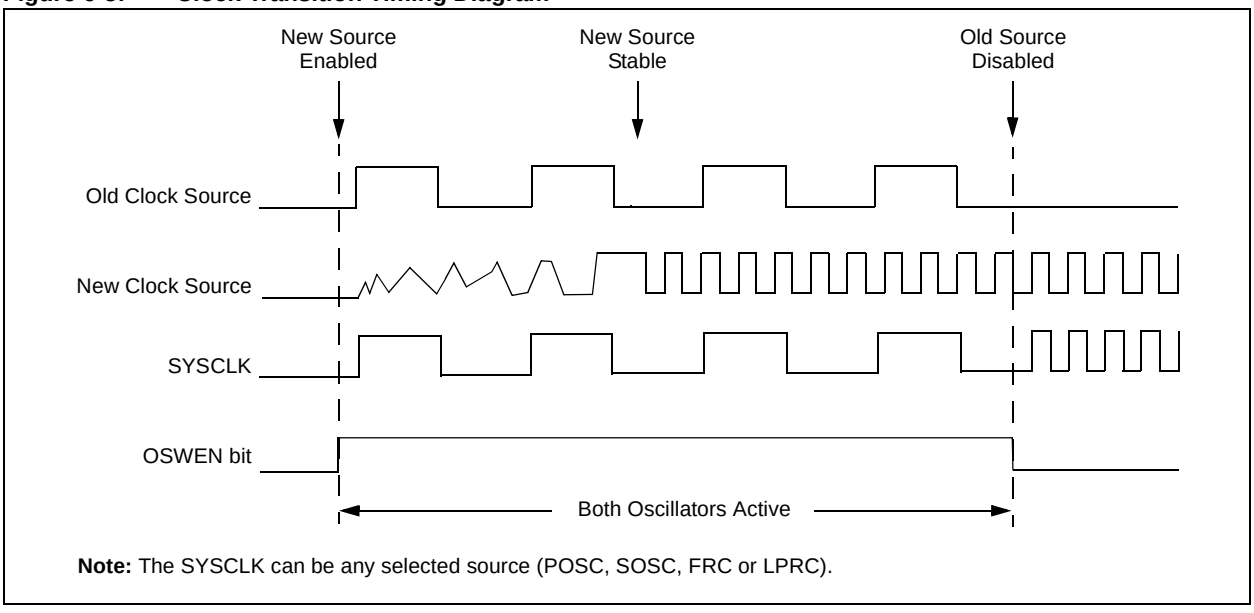
Once the basic sequence is completed, the system clock hardware responds automatically as follows:

1. The clock switching hardware compares the COSC<2:0> Status bits with the new value of the NOSC control bits. If they are the same, then the clock switch is a redundant operation. In this case, the OSWEN bit is cleared automatically and the clock switch is aborted.
2. The new oscillator is turned on by the hardware if it is not currently running. If a crystal oscillator must be turned on, the hardware will wait until the Oscillator Start-up timer (OST) expires. If the new source is using the PLL, then the hardware waits until a PLL lock is detected (LOCK = 1).
3. The hardware clears the OSWEN bit to indicate a successful clock transition. In addition, the NOSC bit values are transferred to the COSC Status bits.
4. The old clock source is turned off at this time if the clock is not being used by any modules.

The timing of the transition between clock sources in shown in Figure 6-5.

Note:	The processor will continue to execute code throughout the clock switching sequence. Timing-sensitive code should not be executed during this time.

Figure 6-5: Clock Transition Timing Diagram



Note: The SYSCLK can be any selected source (POSC, SOSC, FRC or LPRC).

The following is a recommended code sequence for a clock switch:

1. Disable interrupts and DMA prior to the system unlock sequence.
2. Execute the system unlock sequence by writing the Key values of 0xAA996655 and 0x556699AA to the SYSKEY register in two back-to-back assembly or 'C' instructions.
3. Write the new oscillator source value to the NOSC control bits.
4. Set the OSWEN bit in the OSCCON register to initiate the clock switch.
5. Write a non-key value (such as 0x33333333) to the SYSKEY register to perform a lock. Continue to execute code that is not clock-sensitive (optional).
6. Check to see if OSWEN is '0'. If it is, the switch was successful. Loop until the bit is '0'.
7. Re-enable interrupts and DMA.

Notes: There are no timing requirements for the steps other than the initial back-to-back writing of the Key values to perform the unlock sequence.

The unlock sequence unlocks all registers that are secured by the lock function. It is recommended that amount to time is the system is unlock is kept to a minimum. The core sequence for unlocking the OSCCON register and initiating a clock switch is shown in Example 6-2.

6.3.6.3 Clock Switching Considerations

When incorporating clock switching into an application, users should keep certain things in mind when designing their code.

- The SYSLOCK unlock sequence is timing critical. The two Key values must be written back-to-back with no in-between peripheral register access. To prevent unintended peripheral register accesses, it is recommended that all interrupts and DMA transfers are disbaled.
- The system will not relock automatically. The user should perform the relock sequence as soon after the clock switch as is possible.
- The unlock sequence unlocks other registers such as the those related to Real-Time Clock control.
- If the destination clock source is a crystal oscillator, the clock switch time will be dictated by the oscillator start-up time.
- If the new clock source does not start, or is not present, the OSWEN bit remain set.
- A clock switch to a different frequency will affect the clocks to peripherals. Peripherals may require reconfiguration to continue operation at the same rate as they did before the clock switch occurred.
- If the new clock source uses the PLL, a clock switch will not occur until lock has been achieved.
- If the WDT is used, care must be taken to ensure it can be serviced in a timely manner at the new clock rate.

Note: The application should not attempt to switch to a clock with a frequency lower than 100 kHz when the Fail-Safe Clock Monitor is enabled. Clock switching in these instances may generate a false oscillator fail event and result in a switch to the Internal Fast RC oscillator.

Note: The device will not permit direct switching between PLL clock sources. The user should not change the PLL multiplier values or postscaler values when running from the affected PLL source. To perform either of the above clock switching functions, the clock switch should be performed in two steps. The clock source should first be switched to a non-PLL source, such as FRC, and then switched to the desired source. This requirement only applies to PLL-based clock sources.

6.3.6.4 Aborting a Clock Switch

In the event the clock switch did not complete, the clock switch logic can be reset by clearing the OSWEN bit (OSCCON<0>). This will abandon the clock switch process, stop and reset the Oscillator Start-up Timer (OST) (if applicable) and stop the PLL (if applicable).

A clock switch procedure can be aborted at any time. A clock switch that is already in progress can also be aborted by performing a second clock switch.

Example 6-2: Performing a Clock Switch

```
                                    // note: clock switching must be enabled in the device
configuration
SYSKEY = 0x0;                       // write invalid key to force lock
SYSKEY = 0xAA996655;                // Write Key1 to SYSKEY
SYSKEY = 0x556699AA;                // Write Key2 to SYSKEY
                                    // OSCCON is now unlocked
                                    // make the desired change
OSCCONCLR = 7 << 8;                 // clear the clock select bits
OSCCONSET = 7 << 8;                 // set the new clock source to FRC
OSCCONSET = 1;                      // request clock switch
                                    // Relock the SYSKEY
SYSKEY = 0x0;                       // Write any value other than Key1 or Key2
                                    // OSCCON is relocked
```

6.3.6.5 Entering SLEEP Mode During a Clock Switch

If the device enters SLEEP mode during a clock switch operation, the clock switch operation is not aborted. If the clock switch does not complete before entering Sleep mode it will perform the switch when exiting Sleep. The WAIT instruction is then executed normally.

6.3.7 Real-Time Clock Oscillator

To provide accurate timekeeping the Real-Time Clock and Calendar (RTCC) requires a precise time base. To achieve this requirement the Secondary Oscillator (SOSC) is used as the time base for the RTCC. The SOSC uses an external 32.768 kHz crystal connected to the SOSCI and SOSCO pins.

6.3.7.1 SOSC Control

The SOSC can be used by modules other than the RTCC, therefore, the SOSC is controlled by a combination of software and hardware. Setting the SOSCEN bit (OSCCON<1>) to a '1' enables the SOSC. The SOSC is disabled when it is not being used by the CPU module and the SOSCEN bit is '0'. If the SOSC is being used as SYSCLK, such as after a clock switch, it cannot be disabled by writing to the SOSCEN bit. If the SOSC is enabled by the SOSCEN bit, it will continue to operate when the device is in SLEEP. To prevent inadvertent clock changes the OSCCON register is locked. It must be unlocked prior to software enabling or disabling the SOSC.

Notes: If the RTCC is to be used when the CPU clock source is to be switched between SOSC and another clock source the SOSCEN bit should be set to a '1' in software. Failure to set the bit will cause the SOSC to be disabled when the CPU is switched to another clock source.

Due to the start-up time for an external crystal the user should wait for stable SOCSC oscillator output before enabling the RTCC. This typically requires a 32 ms delay between enabling the SOSC and enabling the RTCC. The actual time required will depend on the crystal in use and the application.

There are numerous system and peripheral registers that are protected from inadvertent writes by the SYSREG lock. Performing a lock or unlock affects all registers protected by SYSREG including OSCCON.

6.3.8 Timer1 External Oscillator

The Timer1 module has the ability to use the SOSC as a clock source to increment Timer1. The SOSC is designed to use an external 32.768 kHz crystal connected to the SOSCI and SOSCO pins.

6.3.8.1 SOSC Control

The SOSC can be used by modules other than Timer1, therefore, the SOSC is controlled by a combination of software and hardware. Setting the SOSCEN bit (OSCCON<1>) to a '1' enables the SOSC. The SOSC is disabled when it is not being used by the CPU module and the SOSCEN bit is '0'. If the SOSC is being used as SYSCLK, such as after a clock switch, it cannot be disabled by writing to the SOSCEN bit. If the SOSC is enabled by the SOSCEN bit, it will continue to operate when the device is in SLEEP. To prevent inadvertent clock changes the OSCCON register is locked. It must be unlocked prior to software enabling or disabling the SOSC.

Notes: If the TIMER1 is to be used when the CPU clock source is to be switched between SOSC and another clock source, the SOSCEN bit should be set to a '1' in software. Failure to set the bit will cause the SOSC to be disabled when the CPU is switched to another clock source.

Due to the start-up time for an external crystal the user should wait for stable SOCSC oscillator output before attempting to use Timer1 for accurate measurements. This typically requires a 10 ms delay between enabling the SOSC and use of Timer1. The actual time required will depend on the crystal in use and the application.

There are numerous system and peripheral registers that are protected from inadvertent writes by the SYSREG lock. Performing a lock or unlock affects all registers protected by SYSREG including OSCCON.

6.4 INTERRUPTS

The only interrupt generated by the oscillator module is the Fail-Safe Clock Monitor (FSCM) event interrupt. When the FSCM mode is enabled and the corresponding interrupts have been configured, a FSCM event will generate a interrupt. This interrupt has both priority and subpriorities that must be configured.

6.4.1 Interrupt Operation

The FSCM has a dedicated interrupt bit FSCMIF (IFS1<14>) and a corresponding interrupt enable/mask bit FSCMIE (IEC1<14>). These bits are used to determine the source of an interrupt and to enable or disable an individual interrupt source. The priority level of the FSCM can be set independently of other interrupt sources.

The FSCMIF bit is set when a FSCM detects a POSC clock failure. The FSCMIF bit will then be set without regard to the state of the corresponding FSCMIE bit. The FSCMIF bit can be polled by software if desired.

The FSCMIE bit controls the interrupt generation. If the FSCMIE bit is set, the CPU will be interrupted whenever an FSCM event occurs (subject to the priority and subpriority as outlined below). The FSCMIF bit will be set regardless of interrupt priority.

It is the responsibility of the routine that services a particular interrupt to clear the appropriate Interrupt Flag bit before the service routine is complete.

The priority of the FSCM interrupt can be set independently via the FSCMIP<2:0> bits (IPC8<20:18>). This priority defines the priority group that interrupt source will be assigned to. The priority groups range from a value of 7, the highest priority, to a value of 0, which does not generate an interrupt. An interrupt being serviced will be preempted by an interrupt in a higher priority group.

The subpriority bits allow setting the priority of a interrupt source within a priority group. The values of the subpriority, FSCMIS<1:0> (IPC8<8:9>), range from 3, the highest priority, to 0 the lowest priority. An interrupt with the same priority group but having a higher subpriority value will preempt a lower subpriority interrupt that is in progress.

The priority group and subpriority bits allow more than one interrupt source to share the same priority and subpriority. If simultaneous interrupts occur in this configuration, the natural order of the interrupt sources within a priority/subgroup pair determine the interrupt generated. The natural priority is based on the vector numbers of the interrupt sources. The lower the vector number the higher the natural priority of the interrupt. Any interrupts that were overridden by natural order will then generate their respective interrupts based on priority, subpriority, and natural order after the interrupt flag for the current interrupt is cleared.

After an enabled interrupt is generated, the CPU will jump to the vector assigned to that interrupt (refer to Table 6-5). The vector number for the interrupt is the same as the natural order number. The IRQ number is not always the same as the vector number due to some interrupts sharing a single vector. The CPU will then begin executing code at the vector address. The users code at this vector address should perform an operations required, such as reloading the duty cycle, clear the interrupt flag, and then exit. Refer to **Section 8. "Interrupts"** for the vector address table details and for more information on interrupts.

Table 6-5: FSCM Interrupt Vectors for Various Offsets with EBASE = 0x8000:0000

Interrupt	Vector/Natural Order	IRQ Number	Vector Address IntCtl.VS = 0x01	Vector Address IntCtl.VS = 0x02	Vector Address IntCtl.VS = 0x04	Vector Address IntCtl.VS = 0x08	Vector Address IntCtl.VS = 0x10
FSCM	33	45	8000 0620	8000 0A40	8000 1280	8000 2300	8000 4400

Example 6-3: FSCM Interrupt Configuration

```
                                    // FSCM must be enabled in the device configuration

                                    // Setup the FSCM interrupt
                                    // located in the users start-up code
if ( OSCCON & 0x8000 )              // check for a FSCM during start-up
{
                                    // user handler for a FSCM event occurred during
start-up
}
else
{
                                    // normal start-up
IPC8CLR = 0x1F << 16;               // clear the FSCM priority bits
IPC8SET = 7 << 18;                  // set the FSCM interrupt priority
IPC8SET = 3 << 16;                  // set the FSCM interrupt subpriority
IFS1CLR = 1 << 24;                  // clear the FSCM interrupt bit
IEC1SET = 1 << 24;                  // Enable the FSCM interrupt
}

void __ISR(_FAIL_SAFE_MONITOR_VECTOR, ipl7) FSCM_HANDLER(void)
{
                                    // interrupt handler
                                    // Insert user code here
                                    IFS1CLR = 1 << 3; // Clear the CMP2 interrupt flag
}
```


6.5 INPUT/OUTPUT PINS

The pins used by the POSC and SOSC are shared by other peripherals modules. Table shows the function of these shared pins in the available oscillator modes. When the pins are not used by a oscillator they are available for use as general I/O pins or by use by a peripheral sharing the pin. Refer to **Section 29. "Real-Time Clock and Calendar"** and **Section 9. "Watchdog Timer and Power-up Timer"** for more information.

Table 6-6: Configuration of Pins Associated with the Oscillator Module

Pin Name	Clock Mode	Configuration Bit FIeld(1)	TRIS	Pin Type
OSCI	HS, HSPLL, XT, XTPLL	COSC<2:0>, POSCMD<1:0>	X	OSC
OSCO	HS, HSPLL, XT, XTPLL	COSC<2:0>, POSCMD	X	OSC
OSCI	EC, ECPLL	COSC<2:0>, POSCMD	X	CLOCK IN
OSCO	EC, ECPLL	COSC<2:0>, POSCMD, OSCOFNC	X	PBCLK OUT
OSCO	EC, ECPLL	COSC<2:0>, POSCMD, OSCOFNC	INPUT	INPUT
OSCO	EC, ECPLL	COSC<2:0>, POSCMD, OSCOFNC	OUTPUT	OUTPUT
N/A	FRC, FRCPLL, FRCDIV16, FRCDIV, LPRC	COSC<2:0>	X	GPIO
N/A	FRC, FRCPLL, FRCDIV16, FRCDIV, LPRC	COSC<2:0>	X	GPIO
N/A	FRC, FRCPLL, FRCDIV16, FRCDIV, LPRC	COSC<2:0>	X	GPIO
N/A	FRC, FRCPLL, FRCDIV16, FRCDIV, LPRC	COSC<2:0>	X	GPIO
SOSCI	SOSC	COSC<2:0>	X	OSC
SOSCO	SOSC	COSC<2:0>	X	OSC

Note 1: During device start-up, the Device Oscillator configuration data is copied from device configuration to COSC.

6.5.1 OSCI and OSCO Pin Functions in Non-External Oscillator Modes

When the primary oscillator (POSC) on OSCI and OSCO is not configured as a clock source the OSCI pin is automatically reconfigured as a digital I/O. In this configuration, as well as when the primary oscillator is configured for EC mode (POSCMD1:POSCMD0 = 00), the OSCO pin can also be configured as a digital I/O by programming the OSCIOFCN Configuration bit.

When OSCIOFCN is unprogrammed ('1'), a PBCLK is available on OSCO for testing or synchronization purposes. With OSCIOFCN programmed ('0'), the OSCO pin becomes a general purpose I/O pin. In both of these configurations, the feedback device between OSCI and OSCO is turned off to save current.

6.5.2 SOSCI and SOCI Pin Functions in Non-External Oscillator Modes

When the secondary oscillator (SOSC) on SOSCI and SOSCO pin is not configured as a clock source the pins are automatically reconfigured as a digital I/O.

6.6 OPERATION IN POWER-SAVING MODES

Note: In this manual, a distinction is made between a power mode as it is used in a specific module, and a power mode as it is used by the device, e.g., Sleep mode of the Comparator and SLEEP mode of the CPU. To indicate which type of power mode is intended, uppercase and lowercase letters (Sleep, Idle, Debug) signify a module power mode, and all uppercase letters (SLEEP, IDLE, DEBUG) signify a device power mode.

6.6.1 Oscillator Operation in SLEEP Mode

Clock sources are disabled in SLEEP unless they are being used by a peripheral. The following sub-sections outline the behavior of each of the clock sources in SLEEP.

6.6.1.1 POSC

The Primary Oscillator POSC is always disabled in SLEEP. Start-up delays apply when exiting SLEEP.

6.6.1.2 SOSC

The Secondary Oscillator is disabled in SLEEP unless the SOSCEN bit is set or it is in use by an enabled module that operates in SLEEP. Start-up delays apply when exiting SLEEP if the secondary oscillator is not already running.

6.6.1.3 FRC

The Fast RC (FRC) oscillator is disabled in SLEEP.

6.6.1.4 LPRC

The Low-Power RC oscillator is disabled in SLEEP if the Watchdog Timer (WDT) is disabled.

6.6.2 Oscillator Operation in IDLE Mode

Clock sources are not disabled in IDLE mode. Start-up delays do not apply when exiting Idle mode.

6.6.3 Oscillator Operation in DEBUG Mode

The Oscillator module continues to operate while the device is in DEBUG mode.

Note: There is no FRZ mode for this module.

6.7 EFFECTS OF VARIOUS RESETS

On all forms of Device Reset OSCCON is set to the default value and the COSC<2:0>, PLLIDIV<2:0>, and PLLMULT<2:0>, and UPLLIDIV<2:0> values are forced to the values defined in the DEVCFG1 and DEVCFG2 Device Configuration Registers. The Oscillator source is transferred to the source as defined in the DEVCFG1 register. Oscillator start-up delays will apply.

6.8 DESIGN TIPS

6.8.1 Crystal Oscillators and Ceramic Resonators

In HS and XT modes, a crystal or ceramic resonator is connected to the OSCI and OSCO pins to establish oscillation (Figure 6-2). The PIC32MX oscillator design requires the use of a parallel cut crystal. Using a series cut crystal may give a frequency out of the crystal manufacturer's specifications.

In general, users should select the oscillator option with the lowest possible gain that still meets their specifications. This will result in lower dynamic currents (I_{DD}). The frequency range of each oscillator mode is the recommended frequency cutoff, but the selection of a different gain mode is acceptable as long as a thorough validation is performed (voltage, temperature and component variations, such as resistor, capacitor and internal oscillator circuitry).

6.8.2 Oscillator/Resonator Start-up

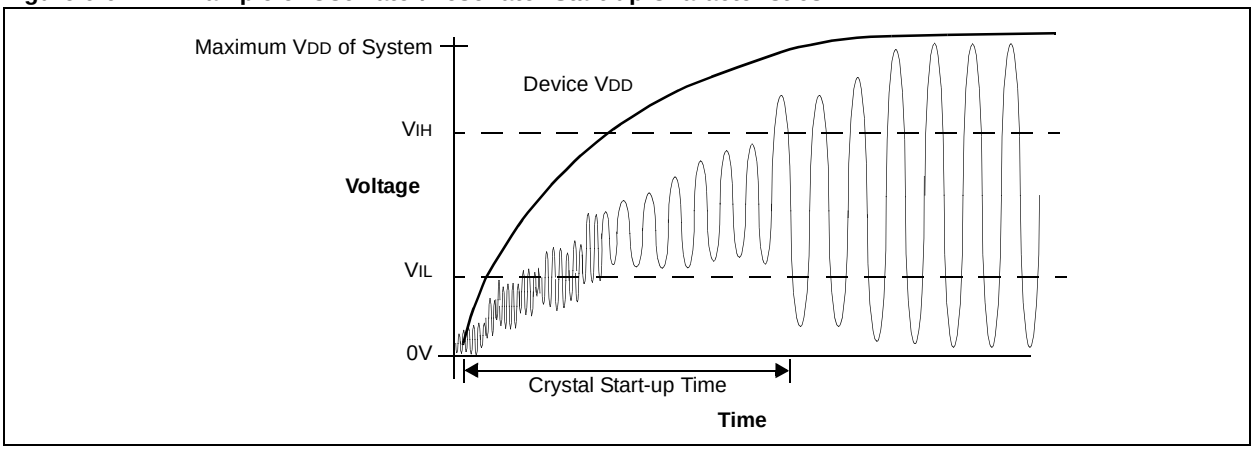
As the device voltage increases from V_{SS}, the oscillator will start its oscillations. The time required for the oscillator to start oscillating depends on many factors, including:

- Crystal/resonator frequency
- Capacitor values used
- Series resistor, if used, and its value and type
- Device V_{DD} rise time
- System temperature
- Oscillator mode selection of device (selects the gain of the internal oscillator inverter)
- Crystal quality
- Oscillator circuit layout
- System noise

The course of a typical crystal or resonator start-up is shown in Figure 6-6. Notice that the time to achieve stable oscillation is not instantaneous.

Refer to the Electrical Characteristics section in the specific device data sheet for further information regarding frequency range for each crystal mode.

Figure 6-6: Example of Oscillator/Resonator Start-up Characteristics



6.8.3 Tuning the Oscillator Circuit

Since Microchip devices have wide operating ranges (frequency, voltage and temperature; depending on the part and version ordered) and external components (crystals, capacitors, etc.) of varying quality and manufacture, validation of operation needs to be performed to ensure that the component selection will comply with the requirements of the application. There are many factors that go into the selection and arrangement of these external components. Depending on the application, these may include any of the following:

- Amplifier gain
- Desired frequency
- Resonant frequency(s) of the crystal
- Temperature of operation
- Supply voltage range
- Start-up time
- Stability
- Crystal life
- Power consumption
- Simplification of the circuit
- Use of standard components
- Component count

6.8.3.1 Determining the Best Values for Oscillator Components

The best method for selecting components is to apply a little knowledge and a lot of trial measurement and testing. Crystals are usually selected by their parallel resonant frequency only; however, other parameters may be important to your design, such as temperature or frequency tolerance. Microchip application note *AN588, "PICmicro® Microcontroller Oscillator Design Guide"* is an excellent reference to learn more about crystal operation and ordering information.

The PIC32MX internal oscillator circuit is a parallel oscillator circuit which requires that a parallel resonant crystal be selected. The load capacitance is usually specified in the 22 pF to 33 pF range. The crystal will oscillate closest to the desired frequency with a load capacitance in this range. It may be necessary to alter these values, as described later, in order to achieve other benefits.

The Clock mode is primarily chosen based on the desired frequency of the crystal oscillator. The main difference between the XT and HS Oscillator modes is the gain of the internal inverter of the oscillator circuit which allows the different frequency ranges. In general, use the oscillator option with the lowest possible gain that still meets specifications. This will result in lower dynamic currents (I_{DD}). The frequency range of each oscillator mode is the recommended frequency cutoff, but the selection of a different gain mode is acceptable as long as a thorough validation is performed (voltage, temperature and component variations, such as resistor, capacitor and internal oscillator circuitry). C1 and C2 should also be initially selected based on the load capacitance, as suggested by the crystal manufacturer, and the tables supplied in the device data sheet. The values given in the device data sheet can only be used as a starting point since the crystal manufacturer, supply voltage, PCB layout and other factors already mentioned may cause your circuit to differ from the one used in the factory characterization process.

Ideally, the capacitance is chosen so that it will oscillate at the highest temperature and the lowest V_{DD} that the circuit will be expected to perform under. High temperature and low V_{DD} both have a limiting effect on the loop gain, such that if the circuit functions at these extremes, the designer can be more assured of proper operation at other temperatures and supply voltage combinations. The output sine wave should not be clipped in the highest gain environment (highest V_{DD} and lowest temperature) and the sine output amplitude should be large enough in the lowest gain environment (lowest V_{DD} and highest temperature) to cover the logic input requirements of the clock as listed in the device data sheet.

A method for improving start-up is to use a value of C2 that is greater than the value of C1. This causes a greater phase shift across the crystal at power-up which speeds oscillator start-up. Besides loading the crystal for proper frequency response, these capacitors can have the effect of lowering loop gain if their value is increased. C2 can be selected to affect the overall gain of the circuit. A higher C2 can lower the gain if the crystal is being overdriven (also, see discussion on Rs). Capacitance values that are too high can store and dump too much current through the crystal, so C1 and C2 should not become excessively large. Unfortunately, measuring the wattage through a crystal is difficult, but if you do not stray too far from the suggested values you should not have to be concerned with this.

A series resistor, Rs, is added to the circuit if, after all other external components are selected to satisfaction, the crystal is still being overdriven. This can be determined by looking at the OSCO pin, which is the driven pin, with an oscilloscope. Connecting the probe to the OSCI pin will load the pin too much and negatively affect performance. Remember that a scope probe adds its own capacitance to the circuit, so this may have to be accounted for in your design (i.e., if the circuit worked best with a C2 of 22 pF and the scope probe was 10 pF, a 33 pF capacitor may actually be called for). The output signal should not be clipping or flattened. Overdriving the crystal can also lead to the circuit jumping to a higher harmonic level, or even, crystal damage.

The OSCO signal should be a clean sine wave that easily spans the input minimum and maximum of the clock input pin. An easy way to set this is to again test the circuit at the minimum temperature and maximum V_{DD} that the design will be expected to perform in, then look at the output. This should be the maximum amplitude of the clock output. If there is clipping, or the sine wave is distorted near V_{DD} and V_{SS}, increasing load capacitors may cause too much current to flow through the crystal or push the value too far from the manufacturer's load specification. To adjust the crystal current, add a trimmer potentiometer between the crystal inverter output pin and C2 and adjust it until the sine wave is clean. The crystal will experience the highest drive currents at the low temperature and high V_{DD} extremes.

The trimmer potentiometer should be adjusted at these limits to prevent overdriving. A series resistor, Rs, of the closest standard value can now be inserted in place of the trimmer. If Rs is too high, perhaps more than 20 kΩ, the input will be too isolated from the output, making the clock more susceptible to noise. If you find a value this high is needed to prevent overdriving the crystal, try increasing C2 to compensate or changing the Oscillator Operating mode. Try to get a combination where Rs is around 10 kΩ or less and load capacitance is not too far from the manufacturer's specification.

6.8.4 FAQs

Question 1: ***When looking at the OSCO pin after power-up with an oscilloscope, there is no clock. What can cause this?***

Answer: There are several possible causes:

1. Entering SLEEP mode with no source for wake-up (such as WDT, MCLR or an interrupt). Verify that the code does not put the device to SLEEP without providing for wake-up. If it is possible, try waking it up with a low pulse on MCLR. Powering up with MCLR held low will also give the crystal oscillator more time to start-up, but the Program Counter will not advance until the MCLR pin is high.
2. The wrong clock mode is selected for the desired frequency. For a blank device, the default oscillator is FRC. Most parts come with the clock selected in the Default mode which will not start oscillation with a crystal or resonator. Verify that the clock mode has been programmed correctly.
3. The proper power-up sequence has not been followed. If a CMOS part is powered through an I/O pin prior to power-up, bad things can happen (latch-up, improper start-up, etc.). It is also possible for brown-out conditions, noisy power lines at start-up and slow V_{DD} rise times to cause problems. Try powering up the device with nothing connected to the I/O, and power-up with a known, good, fast rise power supply. Refer to the power-up information in the specific device data sheet for considerations on brown-out and power-up sequences.
4. The C1 and C2 capacitors attached to the crystal have not been connected properly or are not the correct values. Make sure all connections are correct. The device data sheet values for these components will usually get the oscillator running; however, they just might not be the optimal values for your design.

Question 2: ***Why does my device run at a frequency much higher than the resonant frequency of the crystal?***

Answer: The gain is too high for this oscillator circuit. Refer to **6.8.3.1 "Determining the Best Values for Oscillator Components"** to aid in the selection of C2 (may need to be higher), Rs (may be needed) and Clock mode (wrong mode may be selected). This is especially possible for low-frequency crystals, like the common 32.768 kHz.

Question 3: ***The design runs fine, but the frequency is slightly off. What can be done to adjust this?***

Answer: Changing the value of C1 has some effect on the oscillator frequency. If a series resonant crystal is used, it will resonate at a different frequency than a parallel resonant crystal of the same frequency call-out. Ensure that you are using a parallel resonant crystal.

Question 4: ***What would cause my application to work fine, but then suddenly quit or lose time?***

Answer: Other than the obvious software checks that should be done to investigate losing time, it is possible that the amplitude of the oscillator output is not high enough to reliably trigger the oscillator input. Also, look at the C1 and C2 values and ensure that the device Configuration bits are correct for the desired oscillator mode.

Question 5: ***If I put an oscilloscope probe on an oscillator pin, I don't see what I expect. Why?***

Answer: Remember that an oscilloscope probe has capacitance. Connecting the probe to the oscillator circuitry will modify the oscillator characteristics. Consider using a low capacitance (active) probe.

6.9 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the PIC32MX device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the Oscillator module are:

Title	Application Note #
Crystal Oscillator Basics and Crystal Selection for rfPIC® and PIC® MCU Devices	AN826
Basic PIC® Microcontroller Oscillator Design	AN849
Practical PIC® Microcontroller Oscillator Analysis and Design	AN943
Making Your Oscillator Work	AN949

Note: Please visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the PIC32MX family of devices.

6.10 REVISION HISTORY

Revision A (October 2007)

This is the initial released version of this document.

Revision B (October 2007)

Updated document to remove Confidential status.

Revision C (April 2008)

Revised status to Preliminary; Revise U-0 to r-x; Revise Figure 6-1.

Revision D (May 2008)

Revised Figure 6-1, Table 6-1 (WDTCON); Revised Registers 6-9, 6-13, 6-14, 6-15; Revised Example 6-3; Change Reserved bits from “Maintain as” to “Write”; Added Note to ON bit (WDTCON Register).

Revision E (July 2008)

Revised Figure 6-1; Examples 6-1, 6-2, 6-3.

Section 7. Resets

HIGHLIGHTS

This section of the manual contains the following topics:

7.1 INTRODUCTION

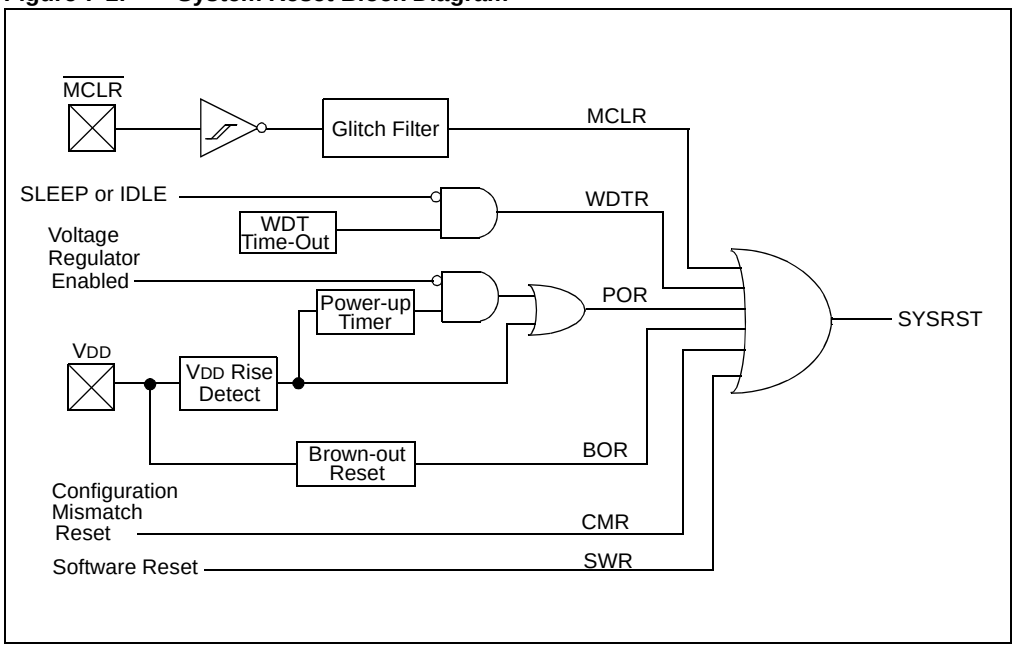
The Resets module combines all Reset sources and controls the system Reset signal SYSRST. The following is a list of device Reset sources:

- POR: Power-on Reset
- $\overline{\text{MCLR}}$: Pin Reset
- SWR: Software Reset
- WDTR: Watchdog Timer Reset
- BOR: Brown-out Reset
- CMR: Configuration Mismatch Reset

A simplified block diagram of the Reset module is shown in Figure 7-1. Any active source of Reset will make the system Reset signal active. Many registers associated with the CPU and peripherals are forced to a known “Reset state”. Most registers are unaffected by a Reset; their status is unknown on POR and unchanged by all other Resets.

Note:	Refer to the specific peripheral or the CPU section of this manual for register Reset states.

Figure 7-1: System Reset Block Diagram



7.2 CONTROL REGISTERS

All types of device Resets will set corresponding Status bits in the RCON register to indicate the type of Reset (see Register 7-1). A Power-on Reset will clear all bits, except for the BOR and POR bits (RCON<1:0>), which are set. The user may set or clear any of the bits at any time during code execution. The RCON bits serve only as Status bits. Setting a particular Reset Status bit in software will not cause a system Reset to occur.

The RCON register also has other bits associated with the Watchdog Timer and device power-saving states. For more information on the function of these bits, refer to **Section 7.4.3 "Using the RCON Status Bits"**.

The RSWRST control register has only one bit, SWRST. This bit is used to force a software Reset condition.

The Resets module consists of the following Special Function Registers (SFRs):

- RCON: Control register for Resets
  RCONCLR, RCONSET, RCONINV: Atomic Bit Manipulation Write-only Registers for RCON
- RSWRST: Data Register for Resets
  RSWRSTCLR, RSWRSTSET, RSWRSTINV: Atomic Bit Manipulation Write-only Registers for RSWRST

The following table summarizes all Resets-related registers. Corresponding registers appear after the summary, followed by a detailed description of each register.

Table 7-1: Reset SFR Summary

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
RCON	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	CMR	VREGS
	7:0	EXTR	SWR	—	WDTO	SLEEP	IDLE	BOR	POR
RCONCLR	31:0	Write clears selected bits in RCON, read yields undefined value							
RCONSET	31:0	Write sets selected bits in RCON, read yields undefined value							
RCONINV	31:0	Write inverts selected bits in RCON, read yields undefined value							
RSWRST	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	—	—	—	—	—	—	—	SWRST
RSWRSTCLR	31:0	Write clears selected bits in RSWRST, read yields undefined value							
RSWRSTSET	31:0	Write sets selected bits in RSWRST, read yields undefined value							
RSWRSTINV	31:0	Write inverts selected bits in RSWRST, read yields undefined value							

Register 7-1: RCON: Reset Control Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-0	R/W-0	R/W-0
—	—	—	—	—	—	CMR	VREGS
bit 15							bit 8

R/W-0	R/W-0	r-x	R/W-0	R/W-0	R/W-0	R/W-1	R/W-1
EXTR	SWR	—	WDTO	SLEEP	IDLE	BOR	POR
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-11 **Reserved:** Write ‘0’; ignore read

bit 10 **Reserved:** Write ‘0’; ignore read

bit 9 **CMR:** Configuration Mismatch Reset Flag bit
1 = Configuration mismatch Reset has occurred
0 = Configuration mismatch Reset has not occurred
Note: This bit is set in hardware, it can only be cleared (= 0) in software.

bit 8 **VREGS:** Voltage Regulator Standby Enable bit
1 = Regulator is enabled and is on during SLEEP mode
0 = Regulator is disabled and is off during SLEEP mode

bit 7 **EXTR:** External Reset ($\overline{\text{MCLR}}$) Pin Flag bit
1 = Master Clear (pin) Reset has occurred
0 = Master Clear (pin) Reset has not occurred
Note: This bit is set in hardware, it can only be cleared (= 0) in software.

bit 6 **SWR:** Software Reset Flag bit
1 = Software Reset was executed
0 = Software Reset as not executed
Note: This bit is set in hardware, it can only be cleared (= 0) in software.

bit 5 **Reserved:** Write ‘0’; ignore read

bit 4 **WDTO:** Watchdog Timer Time-out Flag bit
1 = WDT Time-out has occurred
0 = WDT Time-out has not occurred
Note: This bit is set in hardware, it can only be cleared (= 0) in software.

bit 3 **SLEEP:** Wake From SLEEP Flag bit
1 = Device was in SLEEP mode
0 = Device was not in SLEEP mode
Note: This bit is set in hardware, it can only be cleared (= 0) in software.

bit 2 **IDLE:** Wake From IDLE Flag bit
1 = Device was in IDLE mode
0 = Device was not in IDLE mode
Note: This bit is set in hardware, it can only be cleared (= 0) in software.

Register 7-1: RCON: Reset Control Register

bit 1 **BOR:** Brown-out Reset Flag bit
User software must clear this bit to view next detection.
1 = Brown-out Reset has occurred
0 = Brown-out Reset has not occurred
Note: This bit is set in hardware, it can only be cleared (= 0) in software.

bit 0 **POR:** Power-on Reset Flag bit
User software must clear this bit to view next detection.
1 = Power-on Reset has occurred
0 = Power-on Reset has not occurred
Note: This bit is set in hardware, it can only be cleared (= 0) in software.

Register 7-2: RCONCLR: RCON Clear Register

Write clears selected bits in RCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in RCON**

A write of '1' in one or more bit positions clears the corresponding bit(s) in RCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `RCONCLR = 0x00008001` will clear bits 15 and 5 in RCON register.

Register 7-3: RCONSET: RCON Set Register

Write sets selected bits in RCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in RCON**

A write of '1' in one or more bit positions sets the corresponding bit(s) in RCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `RCONSET = 0x00008001` will set bits 15 and 5 in RCON register.

Register 7-4: RCONINV: RCON Invert Register

Write inverts selected bits in RCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in RCON**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in RCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `RCONINV = 0x00008001` will invert bits 15 and 5 in RCON register.

Register 7-5: RSWRST: Software Reset Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

r-x	r-x	r-x	r-x	r-x	r-x	r-x	W-0
—	—	—	—	—	—	—	SWRST
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-1 **Reserved:** Write ‘0’; ignore read

bit 0 **SWRST:** Software Reset Trigger bit
1 = Enable software Reset event
0 = No effect

Note: The system unlock sequence must be performed before the SWRST bit can be written. See **7.3.4 “Software Reset (SWR)”**.

Register 7-6: RSWRSTCLR: RSWRST Clear Register

Write clears selected bits in RSWRST, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in RSWRST**

A write of '1' in one or more bit positions clears the corresponding bit(s) in RSWRST register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `RSWRSTCLR = 0x00008001` will clear bits 15 and 5 in RSWRST register.

Register 7-7: RSWRSTSET: RSWRST Set Register

Write sets selected bits in RSWRST, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in RSWRST**

A write of '1' in one or more bit positions sets the corresponding bit(s) in RSWRST register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `RSWRSTSET = 0x00008001` will set bits 15 and 5 in RSWRST register.

Register 7-8: RSWRSTINV: RSWRST Invert Register

Write inverts selected bits in RSWRST, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in RSWRST**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in RSWRST register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `RSWRSTINV = 0x00008001` will invert bits 15 and 5 in RSWRST register.

7.3 MODES OF OPERATION

7.3.1 System Reset

The PIC32MX Internal System Reset (SYSRST) can be generated from multiple Reset sources, such as POR (Power-on Reset), BOR (Brown-out Reset), MCLR (Master Clear Reset), WDTO (Watchdog Time-out Reset), SWR (Software Reset) and CMR (Configuration Mis-match Reset). A system Reset is active at first POR and asserted until device configuration settings are loaded and the clock oscillator sources become stable. The system Reset is then de-asserted allowing the CPU to start fetching code after 8 system clock cycles (SYSCLK).

BOR, MCLR and WDTO Resets are asynchronous events and to avoid SFR (Special Function Register) and RAM corruptions, the system Reset is synchronized with the system clock. All other Reset events are synchronous.

7.3.2 Power-on Reset (POR)

A power-on event generates an internal Power-on Reset pulse when a VDD rise is detected above VPOR. The device supply-voltage-characteristics must meet the specified starting-voltage and rise-rate requirements to generate the POR pulse. In particular, VDD must fall below VPOR before a new POR is initiated. For more information on the VPOR and VDD rise-rate specifications, refer to the Electrical Characteristics section of the specific device data sheet for details.

For those PIC32MX variants that have the on-chip voltage regulator enabled, the Power-up Timer (PWRT) is automatically disabled. For those PIC32MX variants that have the on-chip voltage regulator disabled, the core is supplied from an external power supply and the Power-up Timer is automatically enabled and is used to extend the duration of a power-up sequence. The PWRT adds a fixed 64 ms nominal delay at device start-up. Hence, the Power-on delay can either be the on-chip voltage regulator output delay, designated as TPU, or the power-up timer delay, designated as TPWRT.

At this point the POR event has expired, but the device Reset is still asserted while device configuration settings are loaded and the clock oscillator sources are configured and the clock monitoring circuitry waits for the oscillator source to become stable. The clock source used by the PIC32MX device when exiting from Reset is always selected from the FNOSC<2:0> bits in the DEVCFG1 Configuration Word. This additional delay depends on the clock and can include delays for TOSC, TLOCK and TFSCM. For details on the oscillator, PLL and Fail-Safe clock monitoring, refer to **Section 6.3.5 “Fail-Safe Clock Monitor Operation”**.

After these delays expire, the system Reset SYSRST is de-asserted. Before allowing the CPU to start code execution, 8 system clock cycles are required before the synchronized Reset to the CPU core is de-asserted.

The power-on event sets the BOR and POR Status bits (RCON<1:0>).

Refer to the Electrical Characteristics section of the specific device data sheet for more information on the values of the delay parameters.

Note:	When the device exits the Reset condition (begins normal operation), the device operating parameters (voltage, frequency, temperature, etc.) must be within their operating ranges; otherwise, the device will not function correctly. The user must ensure that the delay between the time power is first applied and the time system Reset is released is long enough to get all operating parameters within specification.

7.3.3 $\overline{\text{MCLR}}$ Reset

Whenever the $\overline{\text{MCLR}}$ pin is driven low, the Reset event is synchronized with the system clock SYSCLK before asserting the system Reset SYSRST, provided the input pulse on $\overline{\text{MCLR}}$ is longer than a certain minimum width, as specified in the Electrical Characteristics section of the specific device data sheet for details.

$\overline{\text{MCLR}}$ provides a filter to minimize the effects of noise and to avoid unwanted Reset events. The EXTR Status bit (RCON<7>) is set to indicate the $\overline{\text{MCLR}}$ Reset.

7.3.4 Software Reset (SWR)

The PIC32MX CPU core doesn't provide a specific RESET "instruction"; however, a hardware Reset can be performed in software (Software Reset) by executing a software Reset-command sequence. The software Reset command acts like a $\overline{\text{MCLR}}$ Reset. The software Reset sequence requires the system unlock sequence to be executed before the SWRST bit can be written. Refer to **Section 6.3.6 "Clock Switching Operation"** regarding the system unlock details. A software Reset is performed as follows:

- Write the system unlock sequence
- Set bit SWRST (RSWRST<0>) = 1
- Read the RSWRST register
- Follow with "while(1);" or 4 "NOP" instructions

Writing a '1' to RSWRST register sets bit SWRST, arming the software Reset. The subsequent read of the RSWRST register triggers the software Reset, which should occur on the next clock cycle following the read operation. To ensure no other user code is executed before the Reset event occurs, it is recommended that 4 'NOP' instructions or a "while(1);" statement be placed after the READ instruction.

The SWR Status bit (RCON<6>) is set to indicate the Software Reset.

Example 7-1: Software Reset Command Sequence

```
/*  The following code illustrates a software Reset */

    /* perform a system unlock sequence */
    SYSTEMUnlock();

    /* set SWRST bit to arm reset */
    RSWRSTSET = 1;

    /* read RSWRST register to trigger reset */
    volatile int* p = &RSWRST;
    *p;

    /* prevent any unwanted code execution until reset occurs*/
    while(1);
```


7.3.5 Watchdog Timer Reset

Note:	In this manual, a distinction is made between a power mode as it is used in a specific module, and a power mode as it is used by the device, e.g., Sleep mode of the Comparator and SLEEP mode of the CPU. To indicate which type of power mode is intended, uppercase and lowercase letters (Sleep, Idle, Debug) signify a module power mode, and all uppercase letters (SLEEP, IDLE, DEBUG) signify a device power mode.

A Watchdog Timer (WDT) Reset event is synchronized with the system clock SYSCLK before asserting the system Reset. Note that a WDT Time-out during SLEEP or IDLE mode will wake-up the processor and branch to the PIC32MX Reset vector, but not reset the processor. The only bits affected are WDTO and SLEEP or IDLE in the RCON register. Refer to **Section 9. "Watchdog Timer and Power-up Timer"** in this manual.

7.3.6 Brown-out Reset

PIC32MX family devices have a simple brown-out capability. If the voltage supplied to the regulator is inadequate to maintain a regulated level, the regulator Reset circuitry will generate a BOR event which is synchronized with the system clock SYSCLK before asserting the system Reset. This event is captured by the BOR flag bit (RCON<1>). Refer to the Electrical Characteristics section of the specific device data sheet for further details.

7.3.7 Configuration Mismatch Reset

To maintain the integrity of the stored configuration values, all device Configuration bits are loaded and implemented as a complementary set of bits. As the Configuration Words are being loaded, for each bit loaded as '1', a complementary value of '0', is stored into its corresponding background word location and vice versa. The bit pairs are compared every time the Configuration Words are loaded, including SLEEP mode. During this comparison, if the Configuration bit values are not found opposite to each other, a configuration mismatch event is generated which causes a device Reset.

If a device Reset occurs as a result of a configuration mismatch, the CMR Status bit (RCON<9>) is set.

7.4 EFFECTS OF VARIOUS RESETS

The Reset value for the Reset Control register, RCON, will depend on the type of device Reset, as indicated in Table 7-2.

Table 7-2: Status Bits, Their Significance and the Initialization Condition for RCON Register

Condition	Program Counter	EXTR	SWR	WDTO	SLEEP	IDLE	CMR	BOR	POR
Power-on Reset	BFC0_0000h	0	0	0	0	0	0	1	1
Brown-out Reset	BFC0_0000h	0	0	0	0	0	0	1	u
$\overline{\text{MCLR}}$ Reset during Run Mode	BFC0_0000h	1	u	u	u	u	u	u	u
$\overline{\text{MCLR}}$ Reset during IDLE Mode	BFC0_0000h	1	u	u	u	1(1)	u	u	u
$\overline{\text{MCLR}}$ Reset during SLEEP Mode	BFC0_0000h	1	u	u	1(1)	u	u	u	u
Software Reset Command	BFC0_0000h	u	1	u	u	u	u	u	u
Configuration Word Mismatch Reset	BFC0_0000h	u	u	u	u	u	1	u	u
WDT Time-out Reset during Run Mode	BFC0_0000h	u	u	1	u	u	u	u	u
WDT Time-out Reset during IDLE Mode	BFC0_0000h	u	u	1	u	1(1)	u	u	u
WDT Time-out Reset during SLEEP Mode	BFC0_0000h	u	u	1	1(1)	u	u	u	u
Interrupt Exit from IDLE Mode	Vector	u	u	u	u	1(1))	u	u	u
Interrupt Exit from SLEEP Mode	Vector	u	u	u	1(1)	u	u	u	u

Legend: u = unchanged

Note 1: SLEEP and IDLE bits states defined by previously executed WAIT instruction.

7.4.1 Special Function Register Reset States

Most of the Special Function Registers (SFRs) associated with the PIC32MX CPU and peripherals are reset to a particular value at a device Reset. Reset values are specified in the corresponding section of this manual.

The Reset value for the Reset Control register, RCON, will depend on the type of device Reset.

7.4.2 Configuration Word Register Reset States

All Reset conditions force the configuration settings to be re-loaded. The POR Reset sets all the Configuration Word register locations = 1 before loading the configuration settings. For all other Reset conditions, the Configuration Word register locations are not reset prior to being re-loaded. This difference in behavior accommodates $\overline{\text{MCLR}}$ assertions during DEBUG mode without affecting the state of the DEBUG operations.

Independent of the source of a Reset, the system clock is always re-loaded and is specified by the FNOSC<2:0> value in the DEVCFG1 Configuration Word. When the device is executing code, the user may change the primary system clock source by using the OSCCON register. Refer to **Section 6. "Oscillators"** in this manual for further details.

7.4.3 Using the RCON Status Bits

The user can read the RCON register after any system Reset to determine the cause of the Reset. Table 7-3 provides a summary of the Reset flag bit operation.

Note:	The Status bits in the RCON register should be cleared after they are read so that the next RCON register value after a device Reset will be meaningful.

Table 7-3: Reset Flag Bit Operation

Flag Bit	Set by:	Cleared by:
POR (RCON<0>)	POR	user software
BOR (RCON<1>)	POR, BOR	user software
EXTR (RCON<7>)	$\overline{\text{MCLR}}$ Reset	user software, POR, BOR
SWR (RCON<6>)	Software Reset command	user software, POR, BOR
CMR (RCON<9>)	Configuration mis-match	user software, POR, BOR
WDTO (RCON<4>)	WDT time-out	user software, POR, BOR
SLEEP (RCON<3>)	`WAIT` instruction	user software, POR, BOR
IDLE (RCON<2>)	`WAIT` instruction	user software, POR, BOR

Note: All Reset flag bits may be set or cleared by the user software.

7.4.4 Device Reset to Code Execution Start Time

The delay between the end of a Reset event and when the device actually begins to execute code is determined by two main factors: the type of Reset, and the system clock source coming out of the Reset. The code execution start time for various types of device Resets are summarized in Table 7-4. Individual delays are characterized in the Electrical Characteristics section of the specific device data sheet for details.

Table 7-4: Code Execution Start Time for Various Device Resets

Reset Type	Clock Source	Power-Up Delay(1)(2)(3)	System Clock Delay(4)(5)	FSCM Delay(6)
POR	EC, FRC, FRCDIV, LPRC	(TPU or TPWRT) + TSYSDLY	—	—
	ECPLL, FRCPLL	(TPU or TPWRT) + TSYSDLY	TLOCK	TFSCM
	XT, HS, SOSC	(TPU or TPWRT) + TSYSDLY	TOST	TFSCM
	XTPLL, HSPLL	(TPU or TPWRT) + TSYSDLY	TOST + TLOCK	TFSCM
BOR	EC, FRC, FRCDIV, LPRC	TSYSDLY	—	—
	ECPLL, FRCPLL	TSYSDLY	TLOCK	TFSCM
	XT, HS, SOSC	TSYSDLY	TOST	TFSCM
	XTPLL	TSYSDLY	TOST + TLOCK	TFSCM
$\overline{\text{MCLR}}$, CMR, SWR, WDTO	Any Clock	TSYSDLY	—	—

Note 1: TPU = Power-up Period with on-chip regulator enabled.

2: TPWRT = Power-up Period (POWER-UP TIMER) with on-chip regulator disabled.

3: TSYSDLY = Time required to reload Device Configuration Fuses plus 8 SYSCLK cycles.

4: TOST = Oscillator Start-up Timer.

5: TLOCK = PLL lock time.

6: TFSCM = Fail-Safe Clock Monitor delay.

Note: For parameter specifictions, see Section 30.2 “AC Characteristics and Timing Parameters.”

7.5 DESIGN TIPS

Question 1: ***How can I use the RCON register to determine the source of the device reset?***

Answer: Initialization code after a Reset can examine the RCON register and confirm the source of the Reset. In certain applications, this information can be used to take appropriate action to correct the problem that caused the Reset to occur. All Reset Status bits in the RCON register should be cleared after reading them to ensure the RCON value will provide meaningful results after the next device Reset.

```
int main(void)
{
    //... perform application specific startup tasks

    // next, check the cause of the Reset
    if(RCON & 0x0003)
    {
        // execute a Power-on-Reset handler
        // ...
    }
    else if(RCON & 0x0002)
    {
        // execute a Brown-out-Reset handler
        // ...
    }
    else if(RCON & 0x0080)
    {
        // execute a Master Clear Reset handler
        // ...
    }
    else if(RCON & 0x0040)
    {
        // execute a Software Reset handler
        // ...
    }
    else if (RCON & 0x0200)
    {
        // execute a Configuration Mismatch Reset handler
        // ...
    }
    else if (RCON & 0x0010)
    {
        // execute Watchdog Timeout Reset handler
        // ...
    }

    //... perform other application specific tasks

    while(1);
}
```


7.6 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the PIC32MX device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to Resets are:

Title	Application Note #
No related application notes at this time.	N/A

Note: Please visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the PIC32MX family of devices.

7.7 REVISION HISTORY

Revision A (September 2007)

This is the initial released version of this document.

Revision B (October 2007)

Updated document to remove Confidential status.

Revision C (April 2008)

Revised status to Preliminary; Revised U-0 to r-x.

Revision D (June 2008)

Revised Figure 7-2; Deleted Figure 7-3; Revised Sections 7.3.2, 7.3.3, 7.3.4; Revised Table 7-4; Delete Figure 7.2 and 7.3; Change Reserved bits from “Maintain as” to “Write”.

Revision E (July 2008)

Revised Section 7.3.2, 7.3.3, 7.3.6, 7.4.4.

Section 8. Interrupts

HIGHLIGHTS

This section of the manual contains the following topics:

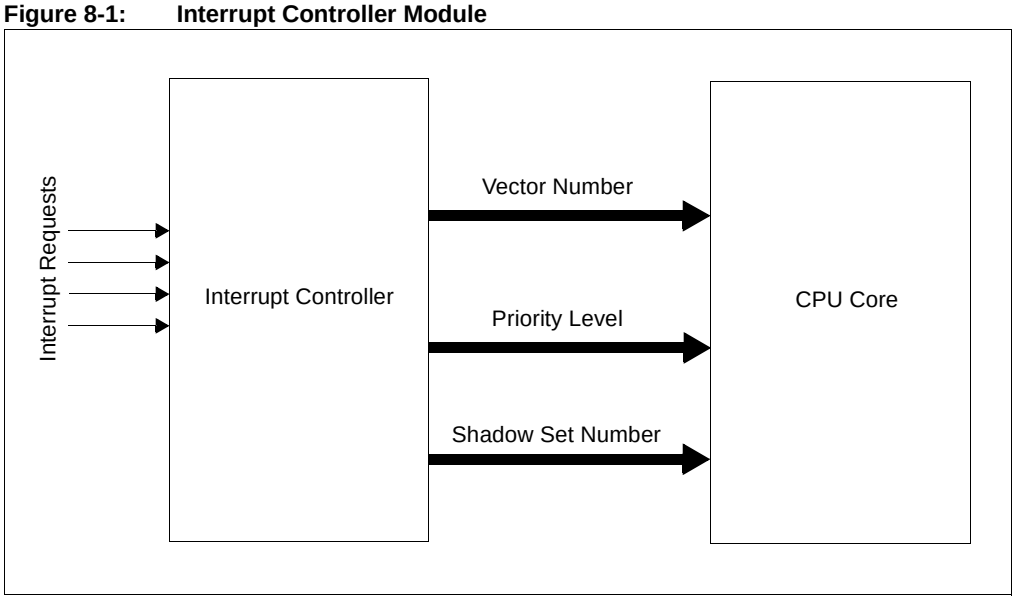
8.1 INTRODUCTION

PIC32MX generates interrupt requests in response to interrupt events from peripheral modules. The interrupts module exists external to the CPU logic and prioritizes the interrupt events before presenting them to the CPU.

The PIC32MX Interrupts module includes the following features:

- Up to 96 interrupt sources
- Up to 64 interrupt vectors
- Single and Multi-Vector mode operations
- 5 External interrupts with edge polarity control
- Interrupt proximity timer
- Module freeze in Debug mode
- 7 user-selectable priority levels for each vector
- 4 user-selectable subpriority levels within each priority
- Dedicated shadow set for highest priority level
- Software can generate any interrupt
- User-configurable interrupt vector table location
- User-configurable interrupt vector spacing

Figure 8-1: Interrupt Controller Module



Note: Several of the registers cited in this section are not in the interrupt controller module. These registers (and bits) are associated with the CPU. Details about them is available in **Section 2. "MCU".**

To avoid confusion, a typographic distinction is made for registers in the CPU. The register names in this section, and all other sections of this manual, are signified by uppercase letters only (except for cases in which variables are used). CPU register names are signified by upper and lowercase letters. For example, INTSTAT is an Interrupts register; whereas, IntCtl is a CPU register.

8.2 CONTROL REGISTERS

Note: Each PIC32MX device variant may have one or more Interrupt channels. An 'x' used in the names of control/Status bits and registers denotes the particular channel. Refer to the specific device data sheets for more details.

The Interrupts module consists of the following Special Function Registers (SFRs).

- INTCON: Interrupt Control Register

  INTCONCLR, INTCONSET, INTCONINV: Atomic Bit Manipulation, Write-only Registers for INTCON

- INTSTAT: Interrupt Status Register

  INTSTATCLR, INTSTATSET, INTSTATINV: Atomic Bit Manipulation, Write-only Registers for INTSTAT

- TPTMR: Temporal Proximity Timer Register

  TPTMRCLR, TPTMRSET, TPTMRNINV: Atomic Bit Manipulation, Write-only Registers for TPTMR

- IFSx: Interrupt Flag Status Registers

  IFSxCLR, IFSxSET, IFSxINV: Atomic Bit Manipulation, Write-only Registers for IFSx

- IECx: Interrupt Enable Control Registers

  IECxCLR, IECxSET, IECxINV: Atomic Bit Manipulation, Write-only Registers for IECx

- IPCx: Interrupt Priority Control Registers

  IPCxCLR, IPCxSET, IPCxINV: Atomic Bit Manipulation, Write-only Registers for IPCx

The following table provides a brief summary of Interrupts-module-related registers. Corresponding registers appear after the summary, followed by a detailed description of each register.

Table 8-1: Interrupt SFR Summary

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
INTCON	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	SS0
	15:8	—	FRZ	—	MVEC	—	TPC<2:0>		
	7:0	—	—	—	INT4EP	INT3EP	INT2EP	INT1EP	INT0EP
INTCONCLR	31:0	Write clears the selected bits in INTCON, read yields undefined value							
INTCONSET	31:0	Write sets the selected bits in INTCON, read yields undefined value							
INTCONINV	31:0	Write inverts the selected bits in INTCON, read yields undefined value							
INTSTAT	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	RIPL<2:0>		
	7:0	—	—	VEC<5:0>					
INTSTATCLR	31:0	Write clears the selected bits in INTSTAT, read yields undefined value							
INTSTATSET	31:0	Write sets the selected bits in INTSTAT, read yields undefined value							
INTSTATINV	31:0	Write inverts the selected bits in INTSTAT, read yields undefined value							
TPTMR	31:24	TPTMR<31:0>							
	23:16								
	15:8								
	7:0								

Table 8-1: Interrupt SFR Summary (Continued)

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
TPTMRCLR	31:0	Write clears the selected bits in TPTMR, read yields undefined value							
TPTMRSET	31:0	Write sets the selected bits in TPTMR, read yields undefined value							
TPTMRINV	31:0	Write inverts the selected bits in TPTMR, read yields undefined value							
IFSx	31:24	IFS31	IFS30	IFS29	IFS28	IFS27	IFS26	IFS25	IFS24
	23:16	IFS23	IFS22	IFS21	IFS20	IFS19	IFS18	IFS17	IFS16
	15:8	IFS15	IFS14	IFS13	IFS12	IFS11	IFS10	IFS09	IFS08
	7:0	IFS07	IFS06	IFS05	IFS04	IFS03	IFS02	IFS01	IFS00
IFSxCLR	31:0	Write clears the selected bits in IFSx, read yields undefined value							
IFSxSET	31:0	Write sets the selected bits in IFSx, read yields undefined value							
IFSxINV	31:0	Write inverts the selected bits in IFSx, read yields undefined value							
IECx	31:24	IEC31	IEC30	IEC29	IEC28	IEC27	IEC26	IEC25	IEC24
	23:16	IEC23	IEC22	IEC21	IEC20	IEC19	IEC18	IEC17	IEC16
	15:8	IEC15	IEC14	IEC13	IEC12	IEC11	IEC10	IEC09	IEC08
	7:0	IEC07	IEC06	IEC05	IEC04	IEC03	IEC02	IEC01	IEC00
IECxCLR	31:0	Write clears the selected bits in IECx, read yields undefined value							
IECxSET	31:0	Write sets the selected bits in IECx, read yields undefined value							
IECxINV	31:0	Write inverts the selected bits in IECx read yields undefined value							
IPCx	31:24	—	—	—	IP03<2:0>			IS03<1:0>	
	23:16	—	—	—	IP02<2:0>			IS02<1:0>	
	15:8	—	—	—	IP01<2:0>			IS01<1:0>	
	7:0	—	—	—	IP00<2:0>			IS00<1:0>	
IPCxCLR	31:0	Write clears the selected bits in IPCx, read yields undefined value							
IPCxSET	31:0	Write sets the selected bits in IPCx, read yields undefined value							
IPCxINV	31:0	Write inverts the selected bits in IPCx, read yields undefined value							

Register 8-1: INTCON: Interrupt Control Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	R/W-0
—	—	—	—	—	—	—	SS0
bit 23							bit 16

r-x	R/W-0	r-x	R/W-0	r-x	R/W-0	R/W-0	R/W-0
—	FRZ	—	MVEC	—	TPC<2:0>		
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	INT4EP	INT3EP	INT2EP	INT1EP	INT0EP
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-17 **Reserved:** Write ‘0’; ignore read

bit 16 **SS0:** Single Vector Shadow Register Set bit
1 = Single vector is presented with a shadow register set
0 = Single vector is not presented with a shadow register set

bit 15 **Reserved:** Write ‘0’; ignore read

bit 14 **FRZ:** Freeze in Debug Exception Mode bit
1 = Freeze operation when CPU is in Debug Exception mode
0 = Continue operation even when CPU is in Debug Exception mode

Note: FRZ is writable in Debug Exception mode only, it is forced to ‘0’ in normal mode.

bit 13 **Reserved:** Write ‘0’; ignore read

bit 12 **MVEC:** Multi Vector Configuration bit
1 = Interrupt controller configured for multi vectored mode
0 = Interrupt controller configured for single vectored mode

bit 11 **Reserved:** Write ‘0’; ignore read

bit 10-8 **TPC:** Temporal Proximity Control bits
111 = Interrupt of group priority 7 or lower start the TP timer
110 = Interrupt of group priority 6 or lower start the TP timer
101 = Interrupt of group priority 5 or lower start the TP timer
100 = Interrupt of group priority 4 or lower start the TP timer
011 = Interrupt of group priority 3 or lower start the TP timer
010 = Interrupt of group priority 2 or lower start the TP timer
001 = Interrupt of group priority 1 start the IP timer
000 = Disables proximity timer

bit 7-5 **Reserved:** Write ‘0’; ignore read

bit 4 **INT4EP:** External Interrupt 4 Edge Polarity Control bit
1 = Rising edge
0 = Falling edge

Register 8-1: INTCON: Interrupt Control Register

bit 3 **INT3EP:** External Interrupt 3 Edge Polarity Control bit
1 = Rising edge
0 = Falling edge

bit 2 **INT2EP:** External Interrupt 2 Edge Polarity Control bit
1 = Rising edge
0 = Falling edge

bit 1 **INT1EP:** External Interrupt 1 Edge Polarity Control bit
1 = Rising edge
0 = Falling edge

bit 0 **INT0EP:** External Interrupt 0 Edge Polarity Control bit
1 = Rising edge
0 = Falling edge

Register 8-2: INTCONCLR: INTCON Clear Register

Write clears selected bits in INTCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in INTCON**

A write of '1' in one or more bit positions clears the corresponding bit(s) in INTCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `INTCONCLR = 0x00000101` will clear bits 8 and 0 in INTCON register.

Register 8-3: INTCONSET: INTCON Set Register

Write sets selected bits in INTCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in INTCON**

A write of '1' in one or more bit positions sets the corresponding bit(s) in INTCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `INTCONSET = 0x00000101` will set bits 8 and 0 in INTCON register.

Register 8-4: INTCONINV: INTCON Invert Register

Write inverts selected bits in INTCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in INTCON**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in INTCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `INTCONINV = 0x00000101` will invert bits 8 and 0 in INTCON register.

Register 8-5: INTSTAT: Interrupt Status Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	R-0	R-0	R-0
—	—	—	—	—	RIPL<2:0>		
bit 15							bit 8

r-x	r-x	R-0	R-0	R-0	R-0	R-0	R-0
—	—	VEC<5:0>					
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-11 **Reserved:** Write ‘0’; ignore read

bit 10-8 **RIPL:** Requested Priority Level bits

000-111 = The priority level of the latest interrupt presented to the CPU
Note: This value should only be used when the interrupt controller is configured for Single Vector mode.

bit 5-0 **VEC:** Interrupt Vector bits

00000-11111 =The interrupt vector that is presented to the CPU
Note: This value should only be used when the interrupt controller is configured for Single Vector mode.

Register 8-6: INTSTATCLR: INTSTAT Clear Register

Write clears selected bits in INTSTAT, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in INTSTAT**

A write of '1' in one or more bit positions clears the corresponding bit(s) in INTSTAT register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `INTSTATCLR = 0x00000101` will clear bits 8 and 0 in INTSTAT register.

Register 8-7: INTSTATSET: INTSTAT Set Register

Write sets selected bits in INTSTAT, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in INTSTAT**

A write of '1' in one or more bit positions sets the corresponding bit(s) in INTSTAT register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `INTSTATSET = 0x00000101` will set bits 8 and 0 in INTSTAT register.

Register 8-8: INTSTATINV: INTSTAT Invert Register

Write inverts selected bits in INTSTAT, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in INTSTAT**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in INTSTAT register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `INTSTATINV = 0x00000101` will invert bits 8 and 0 in INTSTAT register.

Register 8-9: TPTMR: Temporal Proximity Timer Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
TPTMR<31:24>							
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
TPTMR<23:16>							
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
TPTMR<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
TPTMR<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **TPTMR:** Temporal Proximity Timer Reload bits
Used by the Temporal Proximity Timer as a reload value when the Temporal Proximity timer is triggered by an interrupt event.

Register 8-10: TPTMRCLR: TPTMR Clear Register

Write clears selected bits in TPTMR, read yields undefined value
bit 31 ... bit 0

bit 31-0 **Clears selected bits in TPTMR**

A write of '1' in one or more bit positions clears the corresponding bit(s) in TPTMR register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `TPTMRCLR = 0x00000101` will clear bits 8 and 0 in TPTMR register.

Register 8-11: TPTMRSET: TPTMR Set Register

Write sets selected bits in TPTMR, read yields undefined value
bit 31 ... bit 0

bit 31-0 **Sets selected bits in TPTMR**

A write of '1' in one or more bit positions sets the corresponding bit(s) in TPTMR register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `TPTMRSET = 0x00000101` will set bits 8 and 0 in TPTMR register.

Register 8-12: TPTMRINV: TPTMR Invert Register

Write inverts selected bits in TPTMR, read yields undefined value
bit 31 ... bit 0

bit 31-0 **Inverts selected bits in TPTMR**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in TPTMR register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `TPTMRINV = 0x00000101` will toggle bits 8 and 0 in TPTMR register.

Register 8-13: IFSx: Interrupt Flag Status Register(1)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
IFS31	IFS30	IFS29	IFS28	IFS27	IFS26	IFS25	IFS24
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
IFS23	IFS22	IFS21	IFS20	IFS19	IFS18	IFS17	IFS16
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
IFS15	IFS14	IFS13	IFS12	IFS11	IFS10	IFS09	IFS08
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
IFS07	IFS06	IFS05	IFS04	IFS03	IFS02	IFS01	IFS00
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **IFS31-IFS00:** Interrupt Flag Status bits

1 = Interrupt request has occurred

0 = No interrupt request has occurred

Note 1: This register represents a generic definition of the IFSx register. Refer to the **“Interrupts”** chapter in the device data sheet to learn exact bit definitions.

Register 8-14: IFSxCLR: IFSx Clear Register

Write clears selected bits in IFSx, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in IFSx**

A write of '1' in one or more bit positions clears the corresponding bit(s) in IFSx register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `IFSxCLR = 0x00000101` will clear bits 8 and 0 in IFSx register.

Register 8-15: IFSxSET: IFSx Set Register

Write sets selected bits in IFSx, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in IFSx**

A write of '1' in one or more bit positions sets the corresponding bit(s) in IFSx register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `IFSxSET = 0x00000101` will set bits 8 and 0 in IFSx register.

Register 8-16: IFSxINV: IFSx Invert Register

Write inverts selected bits in IFSx, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in IFSx**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in IFSx register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `IFSxINV = 0x00000101` will invert bits 8 and 0 in IFSx register.

Register 8-17: IECx: Interrupt Enable Control Register(1)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
IEC31	IEC30	IEC29	IEC28	IEC27	IEC26	IEC25	IEC24
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
IEC23	IEC22	IEC21	IEC20	IEC19	IEC18	IEC17	IEC16
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
IEC15	IEC14	IEC13	IEC12	IEC11	IEC10	IEC09	IEC08
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
IEC07	IEC06	IEC05	IEC04	IEC03	IEC02	IEC01	IEC00
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **IEC31-IEC00:** Interrupt Enable bits
1 = Interrupt is enabled
0 = Interrupt is disabled

Note 1: This register represents a generic definition of the IFSx register. Refer to the **“Interrupts”** chapter in the device data sheet to learn exact bit definitions.

Register 8-18: IECxCLR: IECx Clear Register

Write clears selected bits in IECx, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in IECx**

A write of '1' in one or more bit positions clears the corresponding bit(s) in IECx register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `IECxCLR = 0x00000101` will clear bits 8 and 0 in IECx register.

Register 8-19: IECxSET: IECx Set Register

Write sets selected bits in IECx, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in IECx**

A write of '1' in one or more bit positions sets the corresponding bit(s) in IECx register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: IECxSET = `0x00000101` will set bits 8 and 0 in IECx register.

Register 8-20: IECxINV: IECx Invert Register

Write inverts selected bits in IECx, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in IECx**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in IECx register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `IECxINV = 0x00000101` will invert bits 8 and 0 in IECx register.

Register 8-21: IPCx: Interrupt Priority Control Register(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	IP03<2:0>			IS03<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	IP02<2:0>			IS02<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	IP01<2:0>			IS01<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	IP00<2:0>			IS00<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-29 **Reserved:** Write ‘0’; ignore read

bit 28-26 **IP03<2:0>:** Interrupt Priority bits
- 111 = Interrupt priority is 7
- 110 = Interrupt priority is 6
- 101 = Interrupt priority is 5
- 100 = Interrupt priority is 4
- 011 = Interrupt priority is 3
- 010 = Interrupt priority is 2
- 001 = Interrupt priority is 1
- 000 = Interrupt is disabled

bit 25-24 **IS03<1:0>:** Interrupt Subpriority bits
- 11 = Interrupt subpriority is 3
- 10 = Interrupt subpriority is 2
- 01 = Interrupt subpriority is 1
- 00 = Interrupt Subpiority is 0

bit 23-21 **Reserved:** Write ‘0’; ignore read

bit 20-18 **IP02<2:0>:** Interrupt Priority bits
- 111 = Interrupt priority is 7
- 110 = Interrupt priority is 6
- 101 = Interrupt priority is 5
- 100 = Interrupt priority is 4
- 011 = Interrupt priority is 3
- 010 = Interrupt priority is 2
- 001 = Interrupt priority is 1
- 000 = Interrupt is disabled

bit 17-16 **IS02<1:0>:** Interrupt Subpriority bits
- 11 = Interrupt subpriority is 3
- 10 = Interrupt subpriority is 2
- 01 = Interrupt subpriority is 1
- 00 = Interrupt subpriority is 0

bit 15-13 **Reserved:** Write ‘0’; ignore read

Register 8-21: IPCx: Interrupt Priority Control Register(1) (Continued)

bit 12-10 **IP01<2:0>:** Interrupt Priority bits
111 = Interrupt priority is 7
110 = Interrupt priority is 6
101 = Interrupt priority is 5
100 = Interrupt priority is 4
011 = Interrupt priority is 3
010 = Interrupt priority is 2
001 = Interrupt priority is 1
000 = Interrupt is disabled

bit 9-8 **IS01<1:0>:** Interrupt Subpriority bits
11 = Interrupt subpriority is 3
10 = Interrupt subpriority is 2
01 = Interrupt subpriority is 1
00 = Interrupt subpriority is 0

bit 7-5 **Reserved:** Write '0'; ignore read

bit 4-2 **IP00<2:0>:** Interrupt Priority bits
111 = Interrupt priority is 7
110 = Interrupt priority is 6
101 = Interrupt priority is 5
100 = Interrupt priority is 4
011 = Interrupt priority is 3
010 = Interrupt priority is 2
001 = Interrupt priority is 1
000 = Interrupt is disabled

bit 1-0 **IS00<1:0>:** Interrupt Subpriority bits
11 = Interrupt subpriority is 3
10 = Interrupt subpriority is 2
01 = Interrupt subpriority is 1
00 = Interrupt subpriority is 0

Note 1: This register represents a generic definition of the IFSx register. Refer to the **"Interrupts"** chapter in the device data sheet to learn exact bit definitions.

Register 8-22: IPCxCLR: IPCx Clear Register

Write clears selected bits in IPCx, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in IPCx**

A write of '1' in one or more bit positions clears the corresponding bit(s) in IPCx register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `IPCxCLR = 0x00000101` will clear bits 8 and 0 in IPCx register.

Register 8-23: IPCxSET: IPCx Set Register

Write sets selected bits in IPCx, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in IPCx**

A write of '1' in one or more bit positions sets the corresponding bit(s) in IPCx register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: IPCxSET = `0x00000101` will set bits 8 and 0 in IPCx register.

Register 8-24: IPCxINV: IPCx Invert Register

Write inverts selected bits in IPCx, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in IPCx**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in IPCx register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `IPCxINV = 0x00000101` will invert bits 8 and 0 in IPCx register.

8.3 OPERATION

The interrupt controller is responsible for pre-processing interrupt requests (IRQ) from a number of on-chip peripherals and presenting them in the appropriate order to the processor.

Figure 8-2 depicts the process within the interrupt controller module. The interrupt controller is designed to receive up to 96 IRQs from the processor core and from on-chip peripherals capable of generating interrupts. All IRQs are sampled on the rising edge of the SYSCLK and latched in associated IFSx registers. A pending IRQ is indicated by the flag bit being equal to '1' in an IFSx register. The pending IRQ will not cause further processing if the corresponding bit in the interrupt enable (IECx) register is clear. The IECx bits act to gate the interrupt flag. If the interrupt is enabled, all IRQs are encoded into a 5-bit-wide vector number. The 5-bit vector results in 0 to 63 unique interrupt vector numbers. Since there are more IRQs than available vector numbers, some IRQs share common vector numbers. Each vector number is assigned an interrupt-priority-level and shadow-set number. The priority level is determined by the IPCx register setting of associated vector. In Multi-Vector mode, all priority-level-7 interrupts use a dedicated register set, while in Single Vector mode, all interrupts may receive a dedicated shadow set. The interrupt controller selects the highest priority IRQ among all pending IRQs and presents the associated vector number, priority-level and shadow-set number to the processor core.

The processor core samples the presented vector information between 'E' and 'M' stage of the pipeline. If the vector's priority level presented to the core is greater than the current priority indicated by the CPU Interrupt Priority bits IPL (Status<15:10>), the interrupt is serviced, otherwise it will remain pending until the current priority is less than the interrupt's priority. When servicing an interrupt, the processor core pushes the program counter into the Exception Program Counter (EPC) register in the CPU and sets Exception Level bit EXL (Status<1>) in the CPU. The EXL bit disables further interrupts until the application explicitly re-enables them by clearing EXL bit. Next, it branches to the vector address calculated from the presented vector number.

The INTSTAT register contains the Interrupt Vector Number bits VEC (INTSTAT<5:0>) and Requested Interrupt Priority bits RIPL (INTSTAT<10:8>) of the current pending interrupt. This may not be the same as the interrupt which caused the core to diverge from normal execution.

The processor returns to the previous state when the ERET (Exception Return) instruction is executed. ERET clears the EXL bit, restores the program counter, and reverts the current shadow set to the previous one.

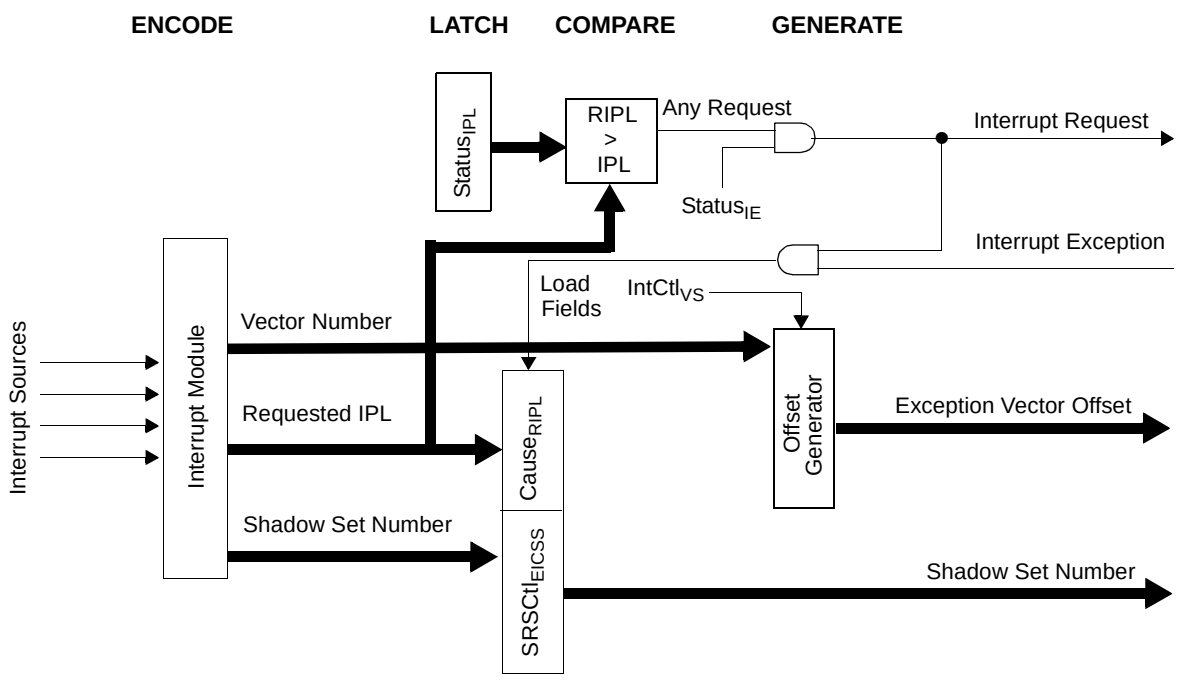
The PIC32MX interrupt controller can be configured to operate in one of two modes:

- Single Vector mode – all interrupt requests will be serviced at one vector address (mode out of reset).
- Multi-Vector mode – interrupt requests will be serviced at the calculated vector address.

Notes: While the user can, during run time, reconfigure the interrupt controller from Single Vector to Multi-Vector mode (or vice versa), such action is strongly discouraged. Changing interrupt controller modes after initialization may result in undefined behavior.

The M4K core supports several different interrupt processing modes. The interrupt controller is designed to work in External Interrupt Controller mode.

Figure 8-2: Interrupt Process



Note: SRSCtl, Cause, Status, and IntCtl registers are CPU registers and are described in **Section 2. "CPU"**.

8.4 SINGLE VECTOR MODE

On any form of Reset, the interrupt controller initializes to Single Vector mode. When the MVEC (INTCON<12>) bit is '0', the interrupt controller operates in Single Vector mode. In this mode, the CPU always vectors to the same address.

Note:	Users familiar with MIPS32 architecture must note that the M4K core in PIC32MX is still operating in External Interrupt Controller (EIC) mode. The PIC32MX achieves Single Vector mode by forcing all IRQs to use a vector number of 0x00. Because the M4K core in PIC32MX always operates in EIC mode, the single vector behavior through "Interrupt Compatibility Mode" as defined by MIPS32 architecture is not recommended.

To configure the CPU in PIC32MX Single Vector mode, the following CPU registers (IntCtl, Cause, and Status) and INTCON register must be configured as follows:

- EBase ≠ 00000
- VS (IntCtl<9:5>) ≠ 00000
- IV (Cause<23>) = 1
- EXL (Status<1>) = 0
- BEV (Status<22>) = 0
- MVEC (INTCON<12>) = 0
- IE (Status<0>) = 1

Example 8-1: Single Vector Mode Initialization

```
/*
   Set the CP0 registers for multi-vector interrupt
   Place EBASE at 0xBD000000

   This code example uses MPLAB C32 intrinsic functions to access CP0 registers.
   Check your compiler documentation to find equivalent functions or use inline assembly
*/
   unsigned int temp;

   asm volatile("di");               // Disable all interrupts

   temp = mips_getsr();              // Get Status
   temp |= 0x00400000;               // Set BEV bit
   mips_setsr(temp);                 // Update Status

   _mips_mtc0(C0_EBASE,  0xBD000000); // Set an EBase value of 0xBD000000
   _mips_mtc0(C0_INTCTL, 0x00000020); // Set the Vector Spacing to non-zero value

   temp = mips_getcr();              // Get Cause
   temp |= 0x00800000;               // Set IV
   mips_setcr(temp);                 // Update Cause

   temp = mips_getsr();              // Get Status
   temp &= 0xFFBFFFFD;               // Clear BEV and EXL
   mips_setsr(temp);                 // Update Status

   INTCONCLR = 0x800;                // Clear MVEC bit

   asm volatile("ie");               // Enable all interrupts
```

8.5 MULTI-VECTOR MODE

When the MVEC (INTCON<12>) bit is '1', the interrupt controller operates in Multi-Vector mode. In this mode, the CPU vectors to the unique address for each vector number. Each vector is located at a specific offset, with respect to a base address specified by the EBase register in the CPU. The individual vector address offset is determined by the vector space that is specified by the VS bits in IntCtl register. (The IntCtl register is located in the CPU; refer to **Section 2. "MCU"** of this manual for more information.)

To configure the CPU in PIC32MX Multi-Vector mode, the following CPU registers (IntCtl, Cause, and Status) and the INTCON register must be configured as follows:

- EBase ≠ 00000
- VS (IntCtl<9:5>) ≠ 00000
- IV (Cause<23>) = 1
- EXL (Status<1>) = 0
- BEV (Status<22>) = 0
- MVEC (INTCON<12>) = 1
- IE (Status<0>) = 1

Example 8-2: Multi-Vector Mode Initialization

```
/*
   Set the CP0 registers for multi-vector interrupt
   Place EBASE at 0xBD000000 and Vector Spacing to 32 bytes

   This code example uses MPLAB C32 intrinsic functions to access CP0 registers.
   Check your compiler documentation to find equivalent functions or use inline assembly
*/
   unsigned int temp;

   asm volatile("di");               // Disable all interrupts

   temp = mips_getsr();              // Get Status
   temp |= 0x00400000;               // Set BEV bit
   mips_setsr(temp);                 // Update Status

   _mips_mtc0(C0_EBASE,  0xBD000000); // Set an EBase value of 0xBD000000
   _mips_mtc0(C0_INTCTL, 0x00000020); // Set the Vector Spacing of 32 bytes
   temp = mips_getcr();              // Get Cause
   temp |= 0x00800000;               // Set IV
   mips_setcr(temp);                 // Update Cause

   temp = mips_getsr();              // Get Status
   temp &= 0xFFBFFFFD;               // Clear BEV and EXL
   mips_setsr(temp);                 // Update Status

   INTCONSET = 0x800;                // Set MVEC bit

   asm volatile("ie");               // Enable all interrupts
```


8.6 INTERRUPT VECTOR ADDRESS CALCULATION

The vector address for a particular interrupt depends on how the interrupt controller is configured. If the interrupt controller is configured for Single Vectored mode (see Section 8.4), all interrupt vectors use the same vector address. When it is configured for Multi-Vectored mode (see Section 8.5), each interrupt vector has a unique vector address.

On all forms of Reset, the processor enters in Bootstrap mode with Control bit BEV (Status<22>) set. (The Status register is located in the CPU; refer to **Section 2. "MCU"** of this manual for more information.) While the processor is in Bootstrap mode, all interrupts are disabled and all general exceptions are redirected to one interrupt vector address, 0xBFC00380. When configuring the interrupt controller to the desired mode of operation, several registers must be set to specific values (See Section 8.4 and Section 8.5) before the BEV bit is cleared.

The vector address of a given interrupt is calculated using Exception Base (EBase<31:12>) register, which provides a 4 KB page-aligned base address value located in the kernel segment (kseg) address space. (EBase is a CPU register.)

8.6.1 Multi-Vector Mode Address Calculation

The Multi-Vector mode address is calculated by using EBase and VS (IntCtl<9:5>) values. (The IntCtl and Status registers are located in the CPU.) The VS bits provide the spacing between adjacent vector addresses. Allowable vector spacing values are 32, 64, 128, 256 and 512 bytes. Modifications to EBase and VS values are only allowed when the BEV (Status<22>) bit is '1' in the CPU. Example 8-3 shows how a multi-vector address is calculated for a given vector.

Note:	The Multi-Vector mode address calculation depends on the interrupt vector number. Each PIC32MX device family may have its own set of vector numbers depending on its feature set. See the respective device data sheet to find out vector numbers associated with each interrupt source.

Example 8-3: Vector Address for Vector Number 16

```
vector address = vector number X (VS << 5) + 0x200 + vector base.

Exception Base is 0xBD000000
Vector Spacing(VS) is 2, which is 64(0x40)
vector address(T4) = 0x10 X 0x40 + 0x200 + 0xBD000000
vector address(T4) = 0xBD000600
```

8.6.2 Single Vector Mode Address Calculation

The Single Vector mode address is calculated by using the Exception Base (EBase<31:12>) register value. In Single Vector mode, the interrupt controller always presents a vector number of '0' The exact formula for Single Vector mode is as follows:

Equation 8-1: Single Vector Mode Address Calculation

```
Single Vector Address = EBase + 0x200
```

8.7 INTERRUPT PRIORITIES

8.7.1 Interrupt Group Priority

The user is able to assign a group priority to each of the interrupt vectors. The groups' priority-level bits are located in IPCx register. Each IPCx register contains group priority bits for four interrupt vectors. The user-selectable priority levels range from 1 (the lowest priority) to 7 (the highest). If an interrupt priority is set to zero, the interrupt vector is disabled for both interrupt and wake-up purposes. Interrupt vectors with a higher priority level preempt lower priority interrupts. The user must move the Requested Interrupt Priority bit of the Cause register RIPL (Cause<15:10>) into the Status register's Interrupt Priority bits IPL (Status<15:10>) before re-enabling interrupts. (The Cause and Status registers are located in the CPU; refer to **Section 2. "MCU"** of this manual for more information.) This action will disable all lower priority interrupts until the completion of the Interrupt Service Routine.

Note:	The Interrupt Service Routine (ISR) must clear the associated interrupt flag in the IFSx register before lowering the interrupt priority level to avoid recursive interrupts.

Example 8-4: Setting Group Priority Level

```
/*
The following code example will set the priority to level 2.
Multi-Vector initialization must be performed (See Example 8-2)
*/
IPC0CLR = 0x0000001C;              // clear the priority level
IPC0SET = 0x00000008;              // set priority level to 2
```

8.7.2 Interrupt Subpriority

The user can assign a subpriority level within each group priority. The subpriority will not cause preemption of an interrupt in the same priority; rather, if two interrupts with the same priority are pending, the interrupt with the highest subpriority will be handled first. The subpriority bits are located in the IPCx register. Each IPCx register contains subpriority bits for four of the interrupt vectors. These bits define the subpriority within the priority level of the vector. The user-selectable subpriority levels range from 0 (the lowest subpriority) to 3 (the highest).

Example 8-5: Setting Subpriority Level

```
/*
The following code example will set the subpriority to level 2.  Multi-Vector
initialization must be performed (See Example 8-2)
*/

IPC0CLR = 0x00000003;              // clear the subpriority level
IPC0SET = 0x00000002;              // set the subpriority to 2
```

8.7.3 Interrupt Natural Priority

When multiple interrupts are assigned to same group priority and subpriority, they are prioritized by their natural priority. The natural priority is a fixed priority scheme, where the highest natural priority starts at the lowest interrupt vector, meaning that interrupt vector 0 is the highest and interrupt vector 63 is the lowest natural priority. See the interrupt vector table in the respective device data sheet to learn the natural priority order of each IRQ.

8.8 INTERRUPTS AND REGISTER SETS

The PIC32MX family of devices employs two register sets, a primary register set for normal program execution and a shadow register set for highest priority interrupt processing. Register set selection is automatically performed by the interrupt controller. The exact method of register set selection varies by the interrupt controller modes of operation.

In Single Vector and Multi-Vector modes of operation, the CSS field in SRSCtl register provides the current number of the register set in use, while the PSS field provides the number of the previous register set. (SRSCtl is a CPU register, refer to **Section 2. "MCU"** of this manual for details.) This information is useful to determine if the Stack and Global Data Pointers should be copied to the new register set, or not. If the current and previous register set are different, the interrupt handler prologue may need to copy Stack and Global Data Pointers from one set to another. Most C compilers supporting PIC32MX automatically generate the necessary interrupt prologue code to handle this operation.

8.8.1 Register Set Selection in Single Vector Mode

In Single Vector mode, SS0 (INTCON<16>) bit determines which register set will be used. If the SS0 is '1', the interrupt controller will instruct the CPU to use the second register set for all interrupts. If the SS0 is '0', the interrupt controller will instruct the CPU to use the first register set. Unlike Multi-Vector mode, there is no linkage between register set and interrupt priority. The application decides if the second shadow set will be used at all.

8.8.2 Register Set Selection in Multi-Vector Mode

When a priority level 7 interrupt is detected in Multi-Vector mode, the interrupt controller instructs the CPU to select the second register set. For interrupts below priority level 7, the interrupt controller instructs the CPU to select the first register set. Because priority level 7 interrupts are uninterruptable, the second register set is dedicated to those interrupts. As a result, priority level 7 interrupts do not need to save and restore General Purpose Register context, resulting in the shortest interrupt latency.

Unlike register set selection in Single Vector mode, the selection of a second register set is automatically linked to priority level 7. The application does not have to set any register to enable a second register set. All interrupts with priority level of 7 are automatically assigned with the second register set.

8.9 INTERRUPT PROCESSING

When the priority of a requested interrupt is greater than the current CPU priority, the interrupt request is taken and the CPU branches to the vector address associated with the requested interrupt. Depending on the priority of the interrupt, the prologue and epilogue of the interrupt handler must perform certain tasks before executing any useful code. The following examples provide recommended prologues and epilogues.

8.9.1 Interrupt Processing in Single Vector Mode

When the interrupt controller is configured in Single Vector mode, all of the interrupt requests are serviced at the same vector address. The interrupt handler routine must generate a prologue and an epilogue to properly configure, save, and restore all of the core registers, along with General Purpose Registers. At a worst case, all of the modifiable General Purpose Registers must be saved and restored by the prologue and epilogue.

8.9.1.1 Single Vector Mode Prologue

When entering the interrupt handler routine, the interrupt controller must first save the current priority and exception PC counter from Interrupt Priority bits IPL (Status<15:10>) and the ErrorEPC register, respectively, on the stack. (Status and ErrorEPC are CPU registers.) If the routine is presented a new register set, the previous register set's stack register must be copied to the current set's stack register. Then the requested priority may be stored in the IPL from the Requested Interrupt Priority bits RIPL (Cause<15:10>), Exception Level bit EXL and Error Level bit ERL in the Status register (Status<1> and Status<2>) are cleared, and the Master Interrupt Enable bit (Status<0>) is set. Finally, the General Purpose Registers will be saved on the stack. (The Cause and Status registers are located in the CPU.)

Example 8-6: Single Vector Interrupt Handler Prologue in Assembly Code

```
    rdpgpr sp, sp
    mfc0   k0, Cause
    mfc0   k1, EPC
    srl    k0, k0, 0xa
    addiu  sp, sp, -76
    sw     k1, 0(sp)
    mfc0   k1, Status
    sw     k1, 4(sp)
    ins    k1, k0, 10, 6
    ins    k1,zero, 1, 4
    mtc0   k1, Status
    sw     s8, 8(sp)
    sw     a0, 12(sp)
    sw     a1, 16(sp)
    sw     a2, 20(sp)
    sw     a3, 24(sp)
    sw     v0, 28(sp)
    sw     v1, 32(sp)
    sw     t0, 36(sp)
    sw     t1, 40(sp)
    sw     t2, 44(sp)
    sw     t3, 48(sp)
    sw     t4, 52(sp)
    sw     t5, 56(sp)
    sw     t6, 60(sp)
    sw     t7, 64(sp)
    sw     t8, 68(sp)
    sw     t9, 72(sp)
    addu    s8, sp, zero

    // start interrupt handler code here
```


8.9.1.2 Single Vector Mode Epilogue

After completing all useful code of the interrupt handler routine, the original state of the Status and EPC registers, along with the General Purpose Registers saved on the stack, must be restored.

Example 8-7: Single Vector Interrupt Handler Epilogue in Assembly Code

```
    // end of interrupt handler code

    addu                                    sp, s8, zero
    lw      t9, 72(sp)
    lw      t8, 68(sp)
    lw      t7, 64(sp)
    lw      t6, 60(sp)
    lw      t5, 56(sp)
    lw      t4, 52(sp)
    lw      t3, 48(sp)
    lw      t2, 44(sp)
    lw      t1, 40(sp)
    lw      t0, 36(sp)
    lw      v1, 32(sp)
    lw      v0, 28(sp)
    lw      a3, 24(sp)
    lw      a2, 20(sp)
    lw      a1, 16(sp)
    lw      a0, 12(sp)
    lw      s8, 8(sp)
    di
    lw      k0, 0(sp)
    mtc0    k0, EPC
    lw      k0, 4(sp)
    mtc0    k0, Status
    eret
```

8.9.2 Interrupt Processing in Multi-Vector Mode

When the interrupt controller is configured in Multi-Vector mode, the interrupt requests are serviced at the calculated vector addresses. The interrupt handler routine must generate a prologue and an epilogue to properly configure, save, and restore all of the core registers, along with General Purpose Registers. At a worst case, all of the modifiable General Purpose Registers must be saved and restored by the prologue and epilogue. If the interrupt priority is set to receive it's own General Purpose Register set, the prologue and epilogue will not need to save or restore any of the modifiable General Purpose Registers, thus providing the lowest latency.

8.9.2.1 Multi-Vector Mode Prologue

When entering the interrupt handler routine, the Interrupt Service Routine (ISR) must first save the current priority and exception PC counter from Interrupt Priority bits IPL (Status<15:10>) and the ErrorEPC register, respectively, on the stack. If the routine is presented a new register set, the previous register set's stack register must be copied to the current set's stack register. Then the requested priority may be stored in the IPL from Requested Interrupt Priority bits RIPL (Cause<15:10>), Exception Level bit EXL and Error Level bit ERL in the Status register (Status<1> and Status<2>) are cleared, and the Master Interrupt Enable bit (Status<0>) is set. If the interrupt handler is not presented a new General Purpose Register set, these resisters will be saved on the stack. (Cause and Status are CPU registers; refer to **Section 2. "MCU"** of this manual for more information.)

Example 8-8: Prologue Without a Dedicated General Purpose Register Set in Assembly Code

```
rdpgpr sp, sp
mfc0   k0, Cause
mfc0   k1, EPC
srl    k0, k0, 0xa
addiu  sp, sp, -76
sw     k1, 0(sp)
mfc0   k1, Status
sw     k1, 4(sp)
ins    k1, k0, 10, 6
ins    k1,zero, 1, 4
mtc0   k1, Status
sw     s8, 8(sp)
sw     a0, 12(sp)
sw     a1, 16(sp)
sw     a2, 20(sp)
sw     a3, 24(sp)
sw     v0, 28(sp)
sw     v1, 32(sp)
sw     t0, 36(sp)
sw     t1, 40(sp)
sw     t2, 44(sp)
sw     t3, 48(sp)
sw     t4, 52(sp)
sw     t5, 56(sp)
sw     t6, 60(sp)
sw     t7, 64(sp)
sw     t8, 68(sp)
sw     t9, 72(sp)
addu    s8, sp, zero

// start interrupt handler code here
```

Example 8-9: Prologue With a Dedicated General Purpose Register Set in Assembly Code

```
rdpgpr sp, sp
mfc0   k0, Cause
mfc0   k1, EPC
srl    k0, k0, 0xa
addiu  sp, sp, -76
sw     k1, 0(sp)
mfc0   k1, Status
sw     k1, 4(sp)
ins    k1, k0, 10, 6
ins    k1,zero, 1, 4
mtc0   k1, Status
addu    s8, sp, zero

// start interrupt handler code here
```


8.9.2.2 Multi-Vector Mode Epilogue

After completing all useful code of the interrupt handler routine, the original state of the Status and ErrorEPC registers, along with the General Purpose Registers saved on the stack, must be restored. (The Status and ErrorEPC registers are located in the CPU; refer to **Section 2. "MCU"** of this manual for more information.)

Example 8-10: Epilogue Without a Dedicated General Purpose Register Set in Assembly Code

```
// end of interrupt handler code

addu  sp, s8, zero
lw     t9, 72(sp)
lw     t8, 68(sp)
lw     t7, 64(sp)
lw     t6, 60(sp)
lw     t5, 56(sp)
lw     t4, 52(sp)
lw     t3, 48(sp)
lw     t2, 44(sp)
lw     t1, 40(sp)
lw     t0, 36(sp)
lw     v1, 32(sp)
lw     v0, 28(sp)
lw     a3, 24(sp)
lw     a2, 20(sp)
lw     a1, 16(sp)
lw     a0, 12(sp)
lw     s8, 8(sp)
di
lw     k0, 0(sp)
mtc0   k0, EPC
lw     k0, 4(sp)
mtc0   k0, Status
eret
```

Example 8-11: Epilogue With a Dedicated General Purpose Register Set in Assembly Code

```
// end of interrupt handler code

addu  sp, s8, zero
di
lw     k0, 0(sp)
mtc0   k0, EPC
lw     k0, 4(sp)
mtc0   k0, Status
eret
```

8.10 EXTERNAL INTERRUPTS

The interrupt controller supports five external interrupt-request signals (INT4-INT0). These inputs are edge sensitive, they require a low-to-high or a high-to-low transition to create an interrupt request. The INTCON register has five bits that select the polarity of the edge detection circuitry: INT4EP (INTCON<4>), INT3EP (INTCON<3>), INT2EP (INTCON<2>), INT1EP (INTCON<1>), and INT0EP (INTCON<0>).

Note:	Changing the external interrupt polarity may trigger an interrupt request. It is recommended that before changing the polarity, the user disables that interrupt, changes the polarity, clears the interrupt flag, and re-enables the interrupt.

Example 8-12: Setting External Interrupt Polarity

```
/*
The following code example will set INT3 to trigger on a high to low transitio
edge. The CPU must be set up for either multi or single vector interrupts to
handle external interrupts
*/
IEC0CLR = 0x00008000;              // disable INT3
INTCONCLR = 0x00000008;            // clear the bit for falling edge trigger
IFS0CLR = 0x00008000;              // clear the interrupt flag
IEC0SET = 0x00008000;              // enable INT3
```

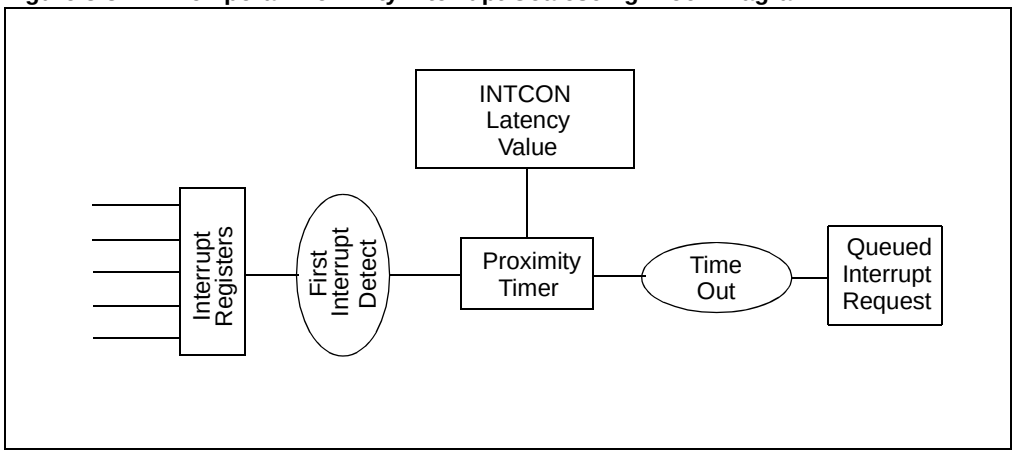

8.11 TEMPORAL PROXIMITY INTERRUPT COALESCING

The PIC32MX CPU responds to interrupt events as if they are all immediately critical because the interrupt controller asserts the interrupt request to the CPU when the interrupt request occurs. The CPU immediately recognizes the interrupt if the current CPU priority is lower than the pending priority. Entering and exiting an ISR consumes clock cycles for saving and restoring context. Events are asynchronous with respect to the main program and have a limited possibility of occurring simultaneously or close together in time. This prevents the ability of a shared ISR to process multiple interrupts at one time.

Temporal Proximity Interrupt uses the interrupt proximity timer, TPTMR, to create a temporal window in which a group of interrupts of the same, or lower, priority will be held off. This provides an opportunity to queue these interrupt requests and process them using tail-chaining single ISR.

Figure 8-3 shows a block diagram of the temporal proximity interrupt coalescing. The interrupt priority group level that triggers the temporal proximity timer is set up in the TPC bits (INTCON<10:8>). TPC selects the interrupt group priority value, and those values below, that will trigger the temporal proximity timer to be reset and loaded with the value in TPTMR. After the timer is loaded with the value in TPTMR, reads to the TPTMR will indicate the current state of the timer. When the timer decrements to zero, the queued interrupt requests are serviced if IPL (Status<15:10>) is less than RIPL (Cause<15:10>).

Figure 8-3: Temporal Proximity Interrupt Coalescing Block Diagram



The user can activate temporal proximity interrupt coalescing by performing the following steps:

- Set the TPC to the preferred priority level. (Setting TPC to zero will disable the proximity timer.)
- Load the preferred 32-bit value to TPTMR

The interrupt proximity timer will trigger when an interrupt request of a priority equal, or lower, matches the TPC value.

Example 8-13: Temporal Proximity Interrupt Coalescing Example

```
/*
The following code example will set the Temporal Proximity Coalescing to
trigger on interrupt priority level of 3 or below and the temporal timer to be
set to 0x12345678.
*/

INTCONCLR = 0x00000700;            // clear TPC
TPTMPCLR  = 0xFFFFFFFF;            // clear the timer
NTCONSET = 0x00000300;             // set TPC->3
TPTMR = 0x12345678;                // set the timer to 0x12345678
```

8.12 EFFECTS OF INTERRUPTS AFTER RESET

8.12.1 Device Reset

All interrupt controller registers are forced to their reset states upon a device Reset.

8.12.2 Power-on Reset

All interrupt controller registers are forced to their reset states upon a device Reset.

8.12.3 Watchdog Timer Reset

All interrupt controller registers are forced to their reset states upon a device Reset.

8.13 OPERATION IN POWER-SAVING AND DEBUG MODES

Note:	In this manual, a distinction is made between a power mode as it is used in a specific module, and a power mode as it is used by the device, e.g., Sleep mode of the Comparator and SLEEP mode of the CPU. To indicate which type of power mode is intended, uppercase and lowercase letters (Sleep, Idle, Debug) signify a module power mode, and all uppercase letters (SLEEP, IDLE, DEBUG) signify a device power mode.

8.13.1 Interrupt Operation in SLEEP Mode

During SLEEP mode, the interrupt controller will only recognize interrupts from peripherals that can operate in SLEEP mode. Peripherals such as RTCC, Change Notice, External Interrupts, ADC, and SPI Slave can continue to operate in SLEEP mode and interrupts from these peripherals can be used to wake up the device. An interrupt with its Interrupt Enable bit set may switch the device to either RUN or IDLE mode, subject to its Interrupt Enable bit status and priority level. An interrupt event with its Interrupt Enable bit cleared or a priority of zero will not be recognized by the interrupt controller and cannot change device status. If the priority of the interrupt request is higher than the current processor priority level, the device will switch to RUN mode and processor will execute the corresponding interrupt request. If the proximity timer is enabled and the pending interrupt priority is less than the temporal proximity priority, the device will remain in SLEEP and the processor will not take the interrupt until after the proximity timer is expired. If the priority of the interrupt request is less than, or equal to, the current processor priority level, the device will switch to IDLE mode and the processor will remain halted.

8.13.2 Interrupt Operation in IDLE Mode

During IDLE mode, interrupt events, with their respective Interrupt Enable bits set, may switch the device to RUN mode subject to its Interrupt Enable bit status and priority level. An interrupt event with its Interrupt Enable bit cleared or a priority of zero will not be recognized by the interrupt controller and cannot change device status. If the priority of the interrupt request is higher than the current CPU priority level, the device will switch to RUN mode and the CPU will execute the corresponding interrupt request. If the proximity timer is enabled and the pending interrupt priority is less than the temporal proximity priority, the device will remain in IDLE and the processor will not take the interrupt until after the proximity time has expired. If the priority of the interrupt request is less than, or equal to, the current CPU priority level, the device will remain in IDLE mode. The corresponding Interrupt Flag bits will remain set and the interrupt request will remain pending.

8.13.3 Interrupt Operation in DEBUG Mode

While the CPU is executing in Debug Exception mode (i.e., the application is halted), all interrupts, regardless of their priority level, are not taken and they will remain pending. Once the CPU exits Debug Exception mode, all pending interrupts will be taken in their order of priority.

Note:	The FRZ bit is readable and writable only when the CPU is executing in Debug Exception mode. In all other modes, the FRZ bit reads as '0'. If FRZ bit is changed during DEBUG mode, the new value does not take effect until the current Debug Exception mode is exited and re-entered. During the Debug Exception mode, the FRZ bit reads the state of the peripheral when entering DEBUG mode.

8.14 DESIGN TIPS

Question 1: ***Can I just enable the interrupt in the IEC registers to start receiving interrupt requests?***

Answer 1: No, you must first enable system interrupts for the core to service any interrupt request. Then, when you enable the interrupt in the IEC register, you will receive interrupt requests.

Question 2: ***When should I clear the interrupt request flag in my interrupt handler?***

Answer 2: You should clear the interrupt request flag as soon as you enter the routine. Handlers that service more than one interrupt request flag can copy the interrupt request flags into a local variable, clear the IFS register, and then service the request.

Question 3: ***After the proximity timer has counted down, which interrupt request is serviced?***

Answer 3: When the proximity timer reaches zero, the interrupt request of the highest priority will be serviced.

8.15 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the PIC32MX device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the Interrupts module are:

Title	Application Note #
No related application notes at this time.	N/A

Note: Visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the PIC32MX family of devices.

8.16 REVISION HISTORY

Revision A (August 2007)

This is the initial released version of this document.

Revision B (October 2007)

Updated document to remove Confidential status.

Revision C (April 2008)

Revised status to Preliminary; Revised U-0 to r-x.

Revision D (June 2008)

Revise Register 8-1, FRZ note; Revise Examples 8-1 and 8-2; Change Reserved bits from “Maintain as” to “Write”.

NOTES:

Section 9. Watchdog Timer and Power-up Timer

HIGHLIGHTS

This section of the manual contains the following topics:

9.1 INTRODUCTION

The PIC32MX Watchdog Timer (WDT) and Power-up Timer (PWRT) modules are described in this section. Refer to Figure 9-1 for a block diagram of the WDT and PWRT.

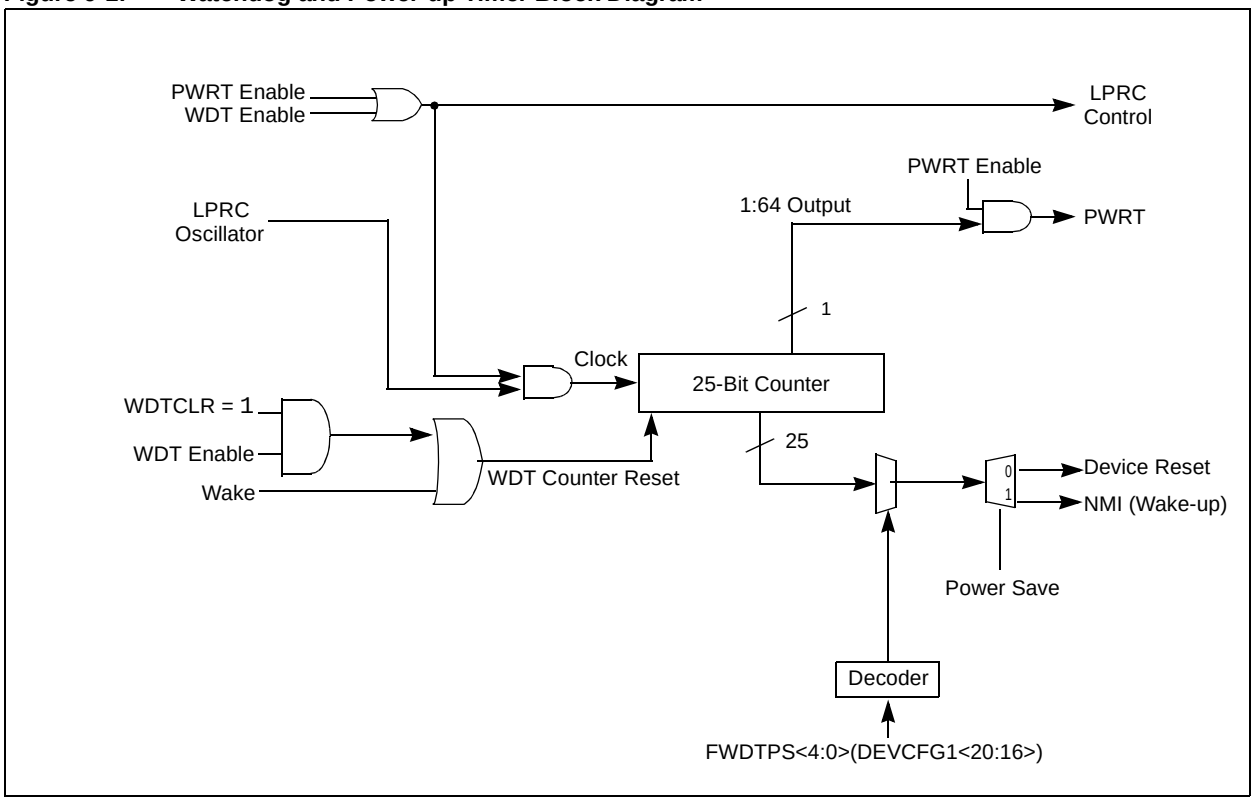
The WDT, when enabled, operates from the internal Low-Power RC (LPRC) oscillator clock source. The WDT can be used to detect system software malfunctions by resetting the device if the WDT is not cleared periodically in software. Various WDT time-out periods can be selected using the WDT postscaler. The WDT can also be used to wake the device from SLEEP or IDLE mode.

The PWRT, when active, holds the device in Reset for a 64 millisecond period after the normal Power-on Reset (POR) start-up period is complete. This allows additional time for the Primary Oscillator (POSC) clock source and the power supply to stabilize. Like the WDT, the PWRT also uses the LPRC as its clock source. Refer to Figure 9-1 for details.

Following are some of the key features of the WDT module:

- Configuration or software controlled
- User configurable time-out period
- Can wake the device from SLEEP or IDLE

Figure 9-1: Watchdog and Power-up Timer Block Diagram



9.2 WATCHDOG TIMER AND POWER-UP TIMER CONTROL REGISTERS

The WDT and PWRT modules consist of the following Special Function Registers (SFRs):

- WDTCON: Watchdog Timer Control Register

  WDTCONCLR, WDTCONSET, WDTCONINV: Atomic Bit Manipulation Registers for WDTCON

- RCON: Resets Control and Status Register

  RCONCLR, RCONSET, RCONINV: Atomic Bit Manipulation Registers for RCON

- DEVCFG1: Device Configuration Register

The following table provides a brief summary of WDT and PWRT-related registers. Corresponding registers appear after the summary, followed by a detailed description of each registers.

Table 9-1: Watchdog Timer and Power-up Timer SFR Summary

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
WDTCON		—	—	—	—	—	—	—	—
		—	—	—	—	—	—	—	—
	15:8	ON	—	—	—	—	—	—	—
	7:0	—	WDTPS<4:0>					—	WDTCLR
WDTCONCLR	31:0	Write clears selected bits in WDTCON, read yields an undefined value							
WDTCONSET	31:0	Write sets selected bits in WDTCON, read yields an undefined value							
WDTCONINV	31:0	Write inverts selected bits in WDTCON, read yields an undefined value							
RCON		—	—	—	—	—	—	—	—
		—	—	—	—	—	—	—	—
	15:8	TRAPR	—	—	—	—	—	CM	VREGS
	7:0	EXTR	SWR	—	WDTO	SLEEP	IDLE	BOR	POR
RCONCLR	31:0	Write clears selected bits in RCON, read yields an undefined value							
RCONSET	31:0	Write sets selected bits in RCON, read yields an undefined value							
RCONINV	31:0	Write inverts selected bits in RCON, read yields an undefined value							
DEVCFG1	31:24	—	—	—	—	—	—	—	—
	23:16	FWDTEN	—	—	FWDTPS<4:0>				
	15:8	FCKSM<1:0>		FPBDIV<1:0>		—	OSCIOFNC	POSCMD<1:0>	
	7:0	IESO	—	FSOSCEN	—	—	FNOSC<2:0>		

Register 9-1: WDTCON: Watchdog Timer Control Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	r-x	r-x	r-x	r-x	R-1	R-1	R-0
ON	—	—	—	—	—	—	—
bit 15							bit 8

r-x	R-x	R-x	R-x	R-x	R-x	r-0	R/W-0
—	WDTPS<4:0>					—	WDTCLR
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15 **ON:** Watchdog Timer Enable bit
1 = Enables the WDT if it is not enabled by the device configuration
0 = Disable the WDT if it was enabled in software

Note 1: A read of this bit will result in a ‘1’ if the WDT is enabled by the device configuration or by software.

2: When using 1:1 PBCLK divisor, the user’s software should not read/write the peripheral’s SFRs in the SYSCLK cycle immediately following the instruction that clears the module’s ON bit.

bit 14-7 **Reserved:** Write ‘0’; ignore read

bit 6-2 **WDTPS<4:0>:** Watchdog Timer Postscaler Value.
On Reset these bits are set to the values of the FWTDPS[4:0] of Configuration bits

bit 1 **reserved:** Write ‘0’; ignore read

bit 0 **WDTCLR:** Watchdog Timer Reset bit
1 = Writing a ‘1’ will clear the WDT.
0 = Software cannot force this bit to a ‘0’

Register 9-2: WDTCONCLR: Comparator Control Clear Register

Write clears selected bits in WDTCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clear selected bits in WDTCON**

A write of '1' in one or more bit positions clears the corresponding bit(s) in WDTCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `WDTCONCLR = 0x00008001` clears bits 15 and 0 in WDTCON register.

Register 9-3: WDTCONSET: Comparator Control Set Register

Write sets selected bits in WDTCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Set selected bits in WDTCON**

A write of '1' in one or more bit positions sets the corresponding bit(s) in WDTCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `WDTCONSET = 0x00008001` sets bits 15 and 0 in WDTCON register.

Register 9-4: WDTCONINV: Comparator Control Invert Register

Write inverts selected bits in WDTCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in WDTCON**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in WDTCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `WDTCONINV = 0x00008001` inverts bits 15 and 0 in WDTCON register.

Register 9-5: RCON: Resets Control Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	r-x	r-x	r-x	r-x	R-0	R/W-0	R/W-0
TRAPR	—	—	—	—	—	CM	VREGS
bit 15							bit 8

R/W-0	R/W-0	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
EXTR	SWR	—	WDTO	SLEEP	IDLE	BOR	POR
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 4 **WDTO:** Watchdog Time-out bit
- 1 = A WDT time out has occurred since either the device was powered up or the WDTO bit was last cleared by software
- 0 = A WDT time out has not occurred since either the WDTO bit was cleared by software or the device was reset

bit 3 **SLEEP:** SLEEP Event bit
- 1 = The device was in SLEEP since either the device was powered up or the SLEEP bit was last cleared by software
- 0 = The device was not in SLEEP since either the SLEEP bit was cleared by software or the device was reset

bit 2 **IDLE:** IDLE Event bit
- 1 = The device has been in IDLE mode since either the device was powered up or the IDLE bit was last cleared by software
- 0 = The device has not been in IDLE mode since either the IDLE bit was cleared by software or the device was reset

Register 9-6: RCONCLR: Comparator Control Clear Register

Write clears selected bits in RCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clear selected bits in RCON**

A write of '1' in one or more bit positions clears the corresponding bit(s) in RCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `RCONCLR = 0x00008001` clears bits 15 and 0 in RCON register.

Register 9-7: RCONSET: Comparator Control Set Register

Write sets selected bits in RCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Set selected bits in RCON**

A write of '1' in one or more bit positions sets the corresponding bit(s) in RCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `RCONSET = 0x00008001` sets bits 15 and 0 in RCON register.

Register 9-8: RCONINV: Comparator Control Invert Register

Write inverts selected bits in RCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in RCON**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in RCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `RCONINV = 0x00008001` inverts bits 15 and 0 in RCON register.

Register 9-9: DEVCFG1 Device Configuration Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

R/P-1	r-1	r-x	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1
FWDTEN	—	—	WDTPS<4:0>				
bit 23							bit 16

R/P-1	R/P-1	R/P-1	R/P-1	r-x	R/P-1	R/P-1	R/P-1
FCKSM<1:0>		FPBDIV<1:0>		—	OSCIOFNC	POSCMD<1:0>	
bit 15							bit 8

R/P-1	r-x	R/P-1	r-x	r-x	R/P-1	R/P-1	R/P-1
IESO	—	FSOSCEN	—	—	FNOSC<2:0>		
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 23 **FWDTEN:** Watchdog Timer Enable bit

1 = WDT is enabled and cannot be disabled by software
0 = WDT is not enabled and can be enabled in software

bit 22 **Reserved:** Write ‘1’; ignore read

bit 20-16 **FWDTPS<4:0>:** Watchdog Timer Postscale Select bits. These bits define the WDT period.

10100 = 1:1048576
10011 = 1:524288
10010 = 1:262144
10001 = 1:131072
10000 = 1:65536
01111 = 1:32768
01110 = 1:16384
01101 = 1:8192
01100 = 1:4096
01011 = 1:2048
01010 = 1:1024
01001 = 1:512
01000 = 1:256
00111 = 1:128
00110 = 1:64
00101 = 1:32
00100 = 1:16
00011 = 1:8
00010 = 1:4
00001 = 1:2
00000 = 1:1
All other combinations not shown result in operation = 10100

9.3 OPERATION

If enabled, the WDT will increment until it overflows or “times out”. A WDT time out will force a device Reset, except during SLEEP or IDLE mode. To prevent a WDT time-out Reset, the user must periodically clear the WDT by setting the WDTCLR (WDTCON<0>) bit. To prevent a device Reset the WDTCLR bit must be periodically set within the selected WDT period.

Table 9-2: Results of a WDT Time-out Event for Available Modes of Device Operation

Device Mode	Device Reset Generated	Non-Maskable Interrupt Generated	WDTO(1) Bit Set	SLEEP(1) Bit Set	IDLE(1) Bit Set	Device Registers Reset
Awake	Yes	No	Yes	No	No	Yes
SLEEP	No	Yes	Yes	Yes	No	No
IDLE	No	Yes	Yes	No	Yes	No

Note 1: Status bits are in the RCON register.

.

Note:	The LPRC oscillator is automatically enabled whenever the WDT is enabled.

9.3.1 Enabling and Disabling the WDT

The WDT is either enabled or disabled by the device configuration, or controlled via software by writing to the WDTCON register.

9.3.2 Device Configuration Controlled WDT

If the FWDTEN device Configuration bit (DEVCFG1<23>) is set, the WDT is always enabled. The WDT ON control bit (WDTCON<15>) will reflect this by reading a ‘1’. In this mode, the ON bit cannot be cleared in software or any form of Reset. To disable the WDT in this mode, the configuration must be rewritten to the device.

Note:	The default state for the WDT on an unprogrammed device is WDT enabled.

9.3.3 Software Controlled WDT

If the FWDTEN device Configuration bit (DEVCFG1<23>) has a value of ‘0’, the WDT can be enabled and disabled by software. In this mode, the ON bit (WDTCON<15>) reflects the status of the WDT under software control. A value of ‘1’ indicates the WDT is enabled and a ‘0’ indicates it is disabled.

The WDT is enabled in software by setting WDT ON control bit. WDT ON control bit is cleared on any device Reset. The bit is not cleared on a wake-up from SLEEP mode or an exit from IDLE mode.

The software WDT option allows the user to enable the WDT for critical code segments, and disable the WDT during non-critical segments, for maximum power savings. The WDT ON control bit can also be used to disable the WDT while the device is awake to eliminate the need for WDT servicing, and then re-enable it before the device is put into IDLE or SLEEP mode to wake-up the device at a later time.

Example 9-1: Sample WDT Initialization and Servicing

```
                        // This code fragment assumes the WDT was not enabled by
                        // the device configuration
                        // The Postscaler value must be set with the device configuration

WDTCONSET = 0x8000;    // Turn on the WDT

main()
{
    WDTCONSET = 0x01;  // Service the WDT

    ... User code goes here ...


}
```


9.3.4 Resetting the WDT Timer

The WDT is cleared by any of the following:

- On any device Reset.
- By a `WDTCONSET = 0x01`, or equivalent instruction, during normal execution. Refer to Example 9-2.
- Exiting from IDLE or SLEEP mode, due to an interrupt.

Note:	The WDT timer is not cleared when the device enters a Power-Saving mode. The WDT should be serviced prior to entering a Power-Saving mode.

Example 9-2: Determining Power-Saving Mode After a Reset

```
OSCCONSET = 0x10;                   // set Power-Saving mode to SLEEP
                                    // OSCCONCLR = 0x10;
                                    // set Power-Saving mode to IDLE

WDTCONSET = 0x8000;                 // Enable WDT

while (1)
{
... user code ...

WDTCONSET = 0x01;                   // service the WDT
asm volatile( “wait” );             // put device is selected Power-Saving mode

                                    // code execution will resume here after wake

... user code ...
}


                                    // The following code fragment is at the top of the
                                    // device start-up code

if ( RCON & 0x18 )
{
                                    // The WDT caused a wake-from-SLEEP
asm volatile( “eret” );             // return from interrupt
}

if ( RCON & 0x14 )
{
                                    // The WDT caused a wake-from-IDLE
asm volatile( “eret” );             // return from interrupt
}

if ( RCON & 0x10 )
{
                                    // The WDT timed-out while the device was awake
}
```

9.3.5 WDT Period Selection

The WDT clock source is the internal LPRC oscillator, which has a nominal frequency of 31.25 kHz. This creates a nominal time-out period for the WDT (TWDT) of 1 millisecond when no postscaler is used.

Note:	The WDT time-out period is directly related to the frequency of the LPRC oscillator. The frequency of the oscillator will vary as a function of device operating voltage and temperature. Please refer to the specific device data sheet for LPRC oscillator clock frequency specifications.

9.3.5.1 WDT Postscalers

The WDT has a 5-bit postscaler to create a wide variety of time-out periods. This postscaler provides 1:1 through 1:1048576 divider ratios. Time-out periods that range between 1 ms and 1048.576 seconds (nominal) can be achieved using the postscaler.

The postscaler settings are selected using the FWDTPS<4:0> Configuration bits in the DEVCFG1 device configuration register. For more information on the WDT Configuration bits, please refer to **Section 32. "Configuration"**.

Equation 9-1: WDT Time-out Period Calculation

$$\text{WDT Period} = 1\text{ ms} \bullet 2^{\text{Prescaler}}$$

The time-out period of the WDT is calculated as follows:

Table 9-3: WDT Time-out Period vs. Postscaler Settings(1, 2)

FWDTPS<4:0>	Postscaler Ratio	Time-out Period
00000	1:1	1 ms
00001	1:2	2 ms
00010	1:4	4 ms
00011	1:8	8 ms
00100	1:16	16 ms
00101	1:32	32 ms
00110	1:64	64 ms
00111	1:128	128 ms
01000	1:256	256 ms
01001	1:512	512 ms
01010	1:1024	1.024 s
01011	1:2048	2.048 s
01100	1:4096	4.096 s
01101	1:8192	8.192 s
01110	1:16384	16.384 s
01111	1:32768	32.768 s
10000	1:65536	65.536 s
10001	1:131072	131.072 s
10010	1:262144	262.144 s
10011	1:524288	524.288 s
10100	1:1045876	1048.576 s

Note 1: All other combinations will result in operation as if the prescaler was set to 10100.

2: The periods listed are based on a 32 kHz (nominal) input clock.

9.3.6 PWRT Timer Operation

The PWRT provides an additional delay between the device POR delay and the beginning of code execution to allow the oscillator to stabilize. Devices that do not have an on-board voltage regulator have the PWRT permanently enabled. Devices that incorporate an on-board voltage regulator automatically enable the PWRT only when the on-board voltage regulator is disabled. The PWRT cannot be enabled or disabled by the device configuration or software.

9.4 INTERRUPT AND RESET GENERATION

The WDT will either cause an Non-Maskable Interrupt (NMI) or a device Reset when it expires. The Power-Saving mode of the device determines which event occurs.

The PWRT does not generate interrupts or Resets.

9.4.1 Watchdog Timer Reset

When the WDT expires and the device is not in SLEEP or IDLE mode, a device Reset is generated. The CPU code execution jumps to the Device Reset Vector and the registers and peripherals are forced to their Reset values.

9.4.2 Watchdog Timer NMI

When the WDT expires in SLEEP or IDLE mode, a NMI is generated. The NMI causes the CPU code execution to jump to the Device Reset Vector. While the NMI shares the same Vector as a device Reset, registers and peripherals are not reset.

To cause a WDT time out in SLEEP mode to act like an interrupt, a return-from-interrupt (`RETFIE`) instruction may be used in the start-up code after the event was determined to be a WDT wake-up. This will cause code execution to continue with the opcode, following the `WAIT` instruction that put the device into Power-Saving mode. Refer to Example 9-2.

9.4.3 Determining Device Status When a WDT Event Has Occurred

To detect a WDT Reset, the WDTO (RCON<4>), SLEEP (RCON<3>), and IDLE (WDTCON<2>) bits must be tested. If the WDTO bit is a ‘1’, the event was due to a WDT time out. The SLEEP and IDLE bits can then be tested to determine whether the WDT event occurred while the device was awake or if it was in SLEEP or IDLE mode. The user should clear the WDTO, SLEEP, and IDLE bits in the Interrupt Service Routine (ISR) to allow software to correctly determine the source of a subsequent WDT event.

9.4.4 Wake From Power-Saving Mode By a Non-WDT Event

When the device is awakened from Power-Saving mode by an interrupt, the WDT is cleared. Practically, this extends the time until the next WDT-generated device Reset occurs, so that an unintended WDT event does not occur too soon after the interrupt that woke the device.

9.5 I/O PINS

The PWRT is disabled when the internal voltage regulator is enabled. A device without an internal voltage regulator will always have the PWRT enabled. A device with an internal voltage regulator will enable the PWRT when the VREG pin is tied to ground (to disable the regulator).

9.6 OPERATION IN DEBUG AND POWER-SAVING MODES

Note:	In this manual, a distinction is made between a power mode as it is used in a specific module, and a power mode as it is used by the device, e.g., Sleep mode of the Comparator and SLEEP mode of the CPU. To indicate which type of power mode is intended, uppercase and lowercase letters (Sleep, Idle, Debug) signify a module power mode, and all uppercase letters (SLEEP, IDLE, DEBUG) signify a device power mode.

9.6.1 WDT Operation in Power-Saving Modes

The WDT can be used to wake the device from SLEEP or IDLE. The WDT continues to operate in Power-Saving mode. A time out can then be used to wake the device. This allows the device to remain in SLEEP mode until the WDT expires or another interrupt wakes the device.

If the device does not re-enter SLEEP or IDLE mode following a wake-up, the WDT must be disabled or periodically serviced to prevent a device Reset.

9.6.2 WDT Operation in SLEEP Mode

The WDT, if enabled, will continue operation in SLEEP mode. The WDT may be used to wake the device from SLEEP. When the WDT times out in SLEEP, a NMI is generated and the WDTO (RCON<4>) bit is set. The NMI vectors execution to the CPU start-up address, but does not reset registers or peripherals. The SLEEP (RCON<3>) status bit will be set indicating the device was in SLEEP. These bits allow the start-up code to determine the cause of the wake-up.

9.6.3 WDT Operation in IDLE Mode

The WDT, if enabled, will continue operation in IDLE mode. The WDT may be used to wake the device from IDLE. When the WDT times out in IDLE, a NMI is generated and the WDTO (RCON<4>) bit is set. The NMI vectors execution to the CPU start-up address, but does not reset registers or peripherals. The IDLE (RCON<2>) status bit will be set indicating the device was in IDLE. These bits allow the start-up code to determine the cause of the wake-up.

9.6.4 Time Delays During Wake-up

The delay between a WDT time-out and the beginning of code execution depends on the Power-Saving mode.

There will be a time delay between the WDT event in SLEEP mode and the beginning of code execution. The duration of this delay consists of the start-up time for the oscillator in use and the PWRT delay, if it is enabled.

Unlike a wake-up from SLEEP mode, there are no time delays associated with wake-up from IDLE mode. The system clock is running during IDLE mode; therefore, no start-up delays are required at wake-up.

9.6.5 WDT Operation in DEBUG Mode

The WDT is always frozen and therefore does not time-out in DEBUG mode.

9.7 EFFECTS OF VARIOUS RESETS

Any form of device Reset will clear the WDT. The Reset will return the WDTCON register to the default value and the WDT will be disabled unless it is enabled by the device configuration.

Note:	After a device Reset, the WDT ON (WDTCON<15>) bit will reflect the state of the FWDTEN (DEVCFG1<23>) bit.

9.8 DESIGN TIPS

Question 1: ***Why does the device reset even though I reset the WDT in my main software loop?***

Answer: Make sure that the timing of the software loop that clears the WDTCLR (WDTCON<0>) bit meets the minimum time-out specification of the WDT (not the typical value) to ensure operation at different voltage and temperatures. Also, make sure that interrupt processing time has been accounted for.

Question 2: ***What should my software do before entering SLEEP or IDLE mode?***

Answer: Make sure that the sources intended to wake the device have their IEC bits set. In addition, make sure that the particular source of interrupt has the ability to wake the device. Some sources do not function when the device is in SLEEP mode.

If the device is to be placed in IDLE mode, make sure that the Stop In Idle (SIDL) control bit for each device peripheral is properly set. These control bits determine whether the peripheral will continue operation in IDLE mode. See the individual peripheral sections of this manual for details.

If the WDT is to be used in SLEEP mode, then the WDT should be serviced before entering sleep to provide a complete WDT interval before the device exits SLEEP mode.

Question 3: ***How do I tell if the WDT or other peripheral woke the device from SLEEP or IDLE mode?***

Answer: Most interrupts have their own unique vector. The vector is determined by the interrupt source. For interrupts that share a vector, the IFS bits for each enabled interrupt source (that shares the vector) can be polled to determine: a.) the source of the interrupt and b.) the source of the wake-up. If the WDT woke the device, the user's start-up code must check for the WDT time-out event, WDTO (RCON<4>), and branch accordingly.

9.9 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the PIC32MX device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the WDT and PWRT modules are:

Title	Application Note #
No related application notes at this time.	N/A

Note: Please visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the PIC32MX family of devices.

9.10 REVISION HISTORY

Revision A (September 2007)

This is the initial released revision of this document.

Revision B (October 2007)

Updated document to remove Confidential status.

Revision C (April 2008)

Revised status to Preliminary; Revised U-0 to r-x;

Revision D (June 2008)

Delete note from Section 9.3; Revise Example 9-2; Change Reserved bits from “Maintain as” to “Write”; Added Note to ON bit (WDTCON Register).

NOTES:

Section 10. Power-Saving Modes

HIGHLIGHTS

This section of the manual contains the following topics:

10.1 INTRODUCTION

This section describes the Power-Saving modes of operation for the PIC32MX device family. The PIC32MX devices have nine Low-Power modes in two categories that allow the user to balance power consumption with device performance. In all of the modes listed below, the device can select the desired Power-Saving mode via software.

10.1.1 CPU Running Modes

In the CPU Running modes, the CPU is running and peripherals can optionally be switched ON or OFF.

- FRC RUN mode: the CPU is clocked from the FRC clock source with or without postscalers.
- LPRC RUN mode: the CPU is clocked from the LPRC clock source.
- SOSC RUN mode: the CPU is clocked from the SOSC clock source.
- Peripheral Bus Scaling mode:
  Peripherals are clocked at programmable fraction of the CPU clock (SYSCLK).

10.1.2 CPU Halted Modes

In the CPU Halted modes, the CPU is halted. Depending on the mode, peripherals can continue to operate or be halted as well.

- POSC IDLE mode: the system clock is derived from the POSC. The system clock source continues to operate.
  Peripherals continue to operate, but can optionally be individually disabled.
- FRC IDLE mode: the system clock is derived from the FRC with or without postscalers.
  Peripherals continue to operate, but can optionally be individually disabled.
- SOSC IDLE mode: the system clock is derived from the SOSC.
  Peripherals continue to operate, but can optionally be individually disabled.
- LPRC IDLE mode: the system clock is derived from the LPRC.
  Peripherals continue to operate, but can optionally be individually disabled. This is the lowest power mode for the device with a clock running.
- SLEEP Mode: the CPU, the system clock source, and any peripherals that operate from the system clock source, are halted.
  Some peripherals can operate in Sleep using specific clock sources. This is the lowest power mode for the device.

10.2 POWER-SAVING MODES CONTROL REGISTERS

Power-Saving modes control consists of the following Special Function Registers (SFRs):

- OSCCON: Control Register for the Oscillators Module
  OSCCONCLR, OSCCONSET, OSCCONINV: Atomic Bit Manipulation Write-only Registers for OSCCON
- WDTCON: Control Register for the Watchdog Timer Module
  WDTCONCLR, WDTCONSET, WDTCONINV: Atomic Bit Manipulation Write-only Registers for WDTCON
- RCON: Control Register for the Resets Module
  RCONCLR, RCONSET, RCONINV: Atomic Bit Manipulation Write-only Registers for RCON

The following table summarizes all Power-Saving-modes-related registers. Corresponding registers appear after the summary, followed by a detailed description of each register.

Table 10-1: Power-Saving Modes SFR Summary

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
OSCCON	31:24	—	—	PLLODIV<2:0>			FRCDIV<2:0>		
	23:16	—	SOSCRDY	—	PBDIV<1:0>		PLLMULT<2:0>		
	15:8	—	COSC<2:0>			—	NOSC<2:0>		
	7:0	CLKLOCK	ULOCK	LOCK	SLPEN	CF	UFRCEN	SOSCEN	OSWEN
OSCCONCLR	31:0	Write clears selected bits in OSCCON, read yields undefined value							
OSCCONSET	31:0	Write sets selected bits in OSCCON, read yields undefined value							
OSCCONINV	31:0	Write inverts selected bits in OSCCON, read yields undefined value							
WDTCON	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	ON	—	—	—	—	—	—	—
	7:0	—	SWDTPS<4:0>					—	WDTCLR
WDTCONCLR	31:0	Write clears selected bits in WDTCON; read yields undefined value							
WDTCONSET	31:0	Write sets selected bits in WDTCON; read yields undefined value							
WDTCONINV	31:0	Write inverts selected bits in WDTCON; read yields undefined value							
RCON	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	CM	VREGS
	7:0	EXTR	SWR	—	WDTO	SLEEP	IDLE	BOR	POR
RCONCLR	31:0	Write clears selected bits in RCON; read yields undefined value							
RCONSET	31:0	Write sets selected bits in RCON; read yields undefined value							
RCONINV	31:0	Write inverts selected bits in RCON; read yields undefined value							

Register 10-1: OSCCON: Oscillator Control Register

r-x	r-x	R/W-x	R/W-x	R/W-x	R/W-0	R/W-0	R/W-1
—	—	PLLODIV<2:0>			FRCDIV<2:0>		
bit 31							bit 24

r-x	R-0	r-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	SOSCRDY	—	PBDIV<1:0>		PLLMULT<2:0>		
bit 23							bit 16

r-x	R-0	R-0	R-0	r-x	R/W-x	R/W-x	R/W-x
—	COSC<2:0>			—	NOSC<2:0>		
bit 15							bit 8

R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-x	R/W-0
CLKLOCK	ULOCK	LOCK	SLPEN	CF	UFRCEN	SOSCEN	OSWEN
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 29-27 **PLLODIV<2:0>:** Output Divider for PLL bits

111 = PLL output divided by 256
110 = PLL output divided by 64
101 = PLL output divided by 32
100 = PLL output divided by 16
011 = PLL output divided by 8
010 = PLL output divided by 4
001 = PLL output divided by 2
000 = PLL output divided by 1

Note: On Reset these bits are set to the value of the FPLLODIV Configuration bits (DEVCFG2<18:16>)

bit 26-24 **FRCDIV<2:0>:** Fast Internal RC Clock Divider bits

111 = FRC divided by 256
110 = FRC divided by 64
101 = FRC divided by 32
100 = FRC divided by 16
011 = FRC divided by 8
010 = FRC divided by 4
001 = FRC divided by 2 (default setting)
000 = FRC divided by 1

bit 23 **Reserved:** Write ‘0’; ignore read

bit 22 **SOSCRDY:** Secondary Oscillator Ready Indicator bit

1 = Indicates that the Secondary Oscillator is running and is stable
0 = Secondary oscillator is either turned off or is still warming up

bit 21 **Unimplemented:** Read as ‘0’

Register 10-1: OSCCON: Oscillator Control Register (Continued)

bit 20-19 **PBDIV<1:0>:** Peripheral Bus Clock Divisor
11 = PBCLK is SYSCLK divided by 8(default)
10 = PBCLK is SYSCLK divided by 4
01 = PBCLK is SYSCLK divided by 2
00 = PBCLK is SYSCLK divided by 1
Note: On Reset these bits are set to the value of the Configuration bits (DEVCFG1<13:12>).

bit 18-16 **PLLMULT<2:0>:** PLL Multiplier bits
111 = Clock is multiplied by 24
110 = Clock is multiplied by 21
101 = Clock is multiplied by 20
100 = Clock is multiplied by 19
011 = Clock is multiplied by 18
010 = Clock is multiplied by 17
001 = Clock is multiplied by 16
000 = Clock is multiplied by 15
Note: On Reset these bits are set to the value of the PLLMULT Configuration bits (DEVCFG2<6:4>)

bit 15 **Reserved:** Write '0'; ignore read

bit 14-12 **COSC<2:0>:** Current Oscillator Selection bits
111 = Fast Internal RC Oscillator divided by OSCCON<FRCDIV> bits
110 = Fast Internal RC Oscillator divided by 16
101 = Low-Power Internal RC Oscillator (LPRC)
100 = Secondary Oscillator (SOSC)
011 = Primary Oscillator with PLL module (XTPLL, HSPLL or ECPLL)
010 = Primary Oscillator (XT, HS or EC)
001 = Fast RC Oscillator with PLL module via Postscaler (FRCPLL)
000 = Fast RC Oscillator (FRC)
Note: On Reset these bits are set to the value of the FNOSC Configuration bits (DEVCFG1<2:0>).

bit 11 **Reserved:** Write '0'; ignore read

bit 10-8 **NOSC<2:0>:** New Oscillator Selection bits
111 = Fast Internal RC Oscillator divided by OSCCON (FRCDIV> bits
110 = Fast Internal RC Oscillator divided by 16
101 = Low Power Internal RC Oscillator (LPRC)
100 = Secondary Oscillator (SOSC)
011 = Primary Oscillator with PLL module (XTPLL, HSPLL or ECPLL)
010 = Primary Oscillator (XT, HS or EC)
001 = Fast Internal RC Oscillator with PLL module via Postscaler (FRCPLL)
000 = Fast Internal RC Oscillator (FRC)
On Reset these bits are set to the value of the FNOSC Configuration bits (DEVCFG1<2:0>).

bit 7 **CLKLOCK:** Clock Selection Lock Enable bit
If FSCM is enabled (FCKSM1 = 1):
1 = Clock and PLL selections are locked
0 = Clock and PLL selections are not locked and may be modified
If FSCM is disabled (FCKSM1 = 0):
Clock and PLL selections are never locked and may be modified

bit 6 **ULOCK:** USB PLL Lock Status bit
1 = Indicates that the USB PLL module is in lock or USB PLL module start-up timer is satisfied
0 = Indicates that the USB PLL module is out of lock or USB PLL module start-up timer is in progress or USB PLL is disabled

bit 5 **LOCK:** PLL Lock Status bit
1 = PLL module is in lock or PLL module start-up timer is satisfied
0 = PLL module is out of lock, PLL start-up timer is running or PLL is disabled

bit 4 **SLPEN:** SLEEP Mode Enable bit
1 = Device will enter SLEEP mode when a WAIT instruction is executed
0 = Device will enter IDLE mode when a WAIT instruction is executed

Register 10-1: OSCCON: Oscillator Control Register (Continued)

bit 3 **CF:** Clock Fail Detect bit
1 = FSCM (Fail Safe Clock Monitor) has detected a clock failure
0 = No clock failure has been detected

bit 2 **UFRCEN:** USB FRC Clock Enable bit
1 = Enable FRC as the clock source for the USB clock source
0 = Use the primary oscillator or USB PLL as the USB clock source

bit 1 **SOSCEN:** 32.768 kHz Secondary Oscillator (SOSC) Enable bit
1 = Enable Secondary Oscillator
0 = Disable Secondary Oscillator
Note: On Reset this bit is set to the value of the FSOSCEN Configuration bit (DEVCFG1<5>).

bit 0 **OSWEN:** Oscillator Switch Enable bit
1 = Initiate an oscillator switch to selection specified by NOSC2:NOSC0 bits
0 = Oscillator switch is complete

Register 10-2: OSCCONCLR: Programming Control Clear Register

Write clears selected bits in OSCCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in OSCCON**

A write of '1' in one or more bit positions clears the corresponding bit(s) in OSCCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: OSCCONCLR = 0x00000001 will clear bit 0 in OSCCON register.

Register 10-3: OSCCONSET: Programming Control Set Register

Write sets selected bits in OSCCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in OSCCON**

A write of '1' in one or more bit positions sets the corresponding bit(s) in OSCCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: OSCCONSET = 0x00000001 will set bit 0 in OSCCON register.

Register 10-4: OSCCONINV: Programming Control Invert Register

Write inverts selected bits in OSCCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in OSCCON**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in OSCCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: OSCCONINV = 0x00000001 will invert bit 0 in OSCCON register.

Register 10-5: WDTCON: WATCHDOG TIMER CONTROL REGISTER

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	r-x	r-x	r-x	r-x	r-x	r-x	r-x
ON	—	—	—	—	—	—	—
bit 15							bit 8

r-x	R-x	R-x	R-x	R-x	R-x	r-0	R/W-0
—	SWDTPS<4:0>					—	WDTCLR
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 15 **ON:** Watchdog Peripheral On bit

1 = Watchdog peripheral is enabled. The status of other bits in the register are not affected by setting this bit. The LPRC oscillator will not be disabled when entering Sleep.

0 = Watchdog peripheral is disabled and not drawing current. SFR modifications are allowed. The status of other bits in this register are not affected by clearing this bit.

Note: When using 1:1 PBCLK divisor, the user’s software should not read/write the peripheral’s SFRs in the SYSCLK cycle immediately following the instruction that clears the module’s ON bit.

Register 10-6: WDTCONCLR: Comparator Control Clear Register

Write clears selected bits in WDTCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clear selected bits in WDTCON**

A write of '1' in one or more bit positions clears the corresponding bit(s) in WDTCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `WDTCONCLR = 0x00008001` clears bits 15 and 0 in WDTCON register.

Register 10-7: WDTCONSET: Comparator Control Set Register

Write sets selected bits in WDTCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Set selected bits in WDTCON**

A write of '1' in one or more bit positions sets the corresponding bit(s) in WDTCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `WDTCONSET = 0x00008001` sets bits 15 and 0 in WDTCON register.

Register 10-8: WDTCONINV: Comparator Control Invert Register

Write inverts selected bits in WDTCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in WDTCON**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in WDTCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `WDTCONINV = 0x00008001` inverts bits 15 and 0 in WDTCON register.

Register 10-9: RCON: RESETS CONTROL REGISTER

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-0	R/W-0	R/W-0
—	—	—	—	—	—	CM	VREGS
bit 15							bit 8

R/W-0	R/W-0	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
EXTR	SWR	—	WDTO	SLEEP	IDLE	BOR	POR
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 3 **SLEEP:** Wake from Sleep bit

1 = The device woke up from SLEEP mode
0 = The device did not wake from SLEEP mode
Note: Must clear this bit to detect future wake ups from SLEEP.

bit 2 **IDLE:** Wake from IDLE bit

1 = The device woke up from IDLE mode
0 = The device did not wake from IDLE mode
Note: Must clear this bit to detect future wake ups from IDLE.

Register 10-10: RCONCLR: RCON Clear Register

Write clears selected bits in RCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in RCON**

A write of '1' in one or more bit positions clears the corresponding bit(s) in RCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `RCONCLR = 0x0000000C` will clear bits 3 and 2 in RCON register.

Register 10-11: RCONSET: RCON Set Register

Write sets selected bits in RCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in RCON**

A write of '1' in one or more bit positions sets the corresponding bit(s) in RCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `RCONSET = 0x0000000C` will set bits 3 and 2 in RCON register.

Register 10-12: RCONINV: RCON Invert Register

Write inverts selected bits in RCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in RCON**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in RCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `RCONINV = 0x0000000C` will invert bits 3 and 2 in RCON register.

10.3 OPERATION OF POWER-SAVING MODES

Note: In this manual, a distinction is made between a power mode as it is used in a specific module, and a power mode as it is used by the device, e.g., Sleep mode of the Comparator and SLEEP mode of the CPU. To indicate which type of power mode is intended, uppercase and lowercase letters (Sleep, Idle, Debug) signify a module power mode, and all uppercase letters (SLEEP, IDLE, DEBUG) signify a device power mode.

The PIC32MX device family has nine Power-Saving modes. The purpose of all the Power-Saving modes is to reduce power consumption by reducing the device clock frequency. To achieve this, multiple low-frequency clock sources can be selected. In addition, the peripherals and CPU can be halted or disabled to further reduce power consumption.

10.3.1 SLEEP Mode

SLEEP mode has the lowest power consumption of the device Power-Saving operating modes. The CPU and most peripherals are halted. Select peripherals can continue to operate in Sleep mode and can be used to wake the device from Sleep. See the individual peripheral module sections for descriptions of behavior in Sleep.

Some of the characteristics of SLEEP mode are as follows:

- The CPU is halted.
- The system clock source is typically shut down. See **10.3.1.1 “Oscillator Shutdown in SLEEP Mode”** for specific information.
- There can be a wake-up delay based on the oscillator selection (refer to Table 10-2).
- The Fail-Safe Clock Monitor (FSCM) does not operate during Sleep mode.
- The BOR circuit, if enabled, remains operative during SLEEP mode.
- The WDT, if enabled, is not automatically cleared prior to entering SLEEP mode.
- Some peripherals can continue to operate in SLEEP mode. These peripherals include I/O pins that detect a change in the input signal, WDT, RTCC, ADC, UART, and peripherals that use an external clock input or the internal LPRC oscillator.
- I/O pins continue to sink or source current in the same manner as they do when the device is not in Sleep.
- The USB module can override the disabling of the POSC or FRC. Refer to the USB section for specific details.
- Modules can be individually disabled by software prior to entering SLEEP in order to further reduce consumption.

The processor will exit, or ‘wake-up’, from SLEEP on one of the following events:

- On any interrupt from an enabled source that is operating in Sleep. The interrupt priority must be greater than the current CPU priority.
- On any form of device Reset.
- On a WDT time-out. See **10.4.2 “Wake-up from SLEEP or IDLE on Watchdog Time-out (NMI)”**.

If the interrupt priority is lower than or equal to current priority, the CPU will remain halted, but the PBCLK will start running and the device will enter in IDLE mode.

Refer Example 10-1 for example code.

Note: There is no FRZ mode for this module.

10.3.1.1 Oscillator Shutdown in SLEEP Mode

The criteria for the device disabling the clock source in SLEEP are: the oscillator type, peripherals using the clock source, and (for select sources) the clock enable bit.

- If the CPU clock source is POSC, it is turned off in SLEEP. See Table 10-2 for applicable delays when waking from SLEEP. The USB module can override the disabling of the POSC or FRC. Refer to the USB section for specific details.
- If the CPU clock source is FRC, it is turned off in SLEEP. See Table 10-2 for applicable delays when waking from SLEEP. The USB module can override the disabling of the POSC or FRC. Refer to the USB section for specific details.
- If the CPU clock source is SOSC, it will be turned off if the SOSCEN bit is not set. See Table 10-2 for applicable delays when waking from SLEEP.
- If the CPU clock source is LPRC, it will be turned off if the clock source is not being used by a peripheral that will be operating SLEEP such as the WDT. See Table 10-2 for applicable delays when waking from SLEEP.

10.3.1.2 Clock Selection on Wake-up from SLEEP

The processor will resume code execution and use the same clock source that was active when SLEEP mode was entered. The device is subject to a start-up delay if a crystal oscillator and/or PLL is used as a clock source when the device exits SLEEP.

10.3.1.3 Delay on Wake-up from SLEEP

The oscillator start-up and Fail-Safe Clock Monitor delays, if enabled associated with waking up from SLEEP mode are shown in Table 10-2.

Table 10-2: Delay Times for Exit from Sleep Mode

Clock Source	Oscillator Delay	FSCM Delay
EC, EXTRC	—	—
EC + PLL	TLOCK	TFSCM
XT + PLL	TOST + TLOCK	TFSCM
XT, HS, XTL	TOST	TFSCM
LP (OFF during Sleep)	TOST	TFSCM
LP (ON during Sleep)	—	—
FRC, LPRC	—	—

Note: Please refer to the “Electrical Specifications” section of the PIC32MX device data sheet for TPOR, TFSCM and TLOCK specification values.

10.3.1.4 Wake-up from SLEEP Mode with Crystal Oscillator or PLL

If the system clock source is derived from a crystal oscillator and/or the PLL, then the Oscillator Start-up Timer (OST) and/or PLL lock times will be applied before the system clock source is made available to the device. As an exception to this rule, no oscillator delays are applied if the system clock source is the POSC oscillator and it was running while in SLEEP mode.

Note: In spite of the various delays applied the crystal oscillator (and PLL) may not be up and running at the end of the Tost, or Tlock delays. For proper operation the user must design the external oscillator circuit such that reliable oscillation will occur within the delay period.

10.3.1.5 Fail-Safe Clock Monitor (FSCM) Delay and SLEEP Mode

The FSCM does not operate while the device is in Sleep. If the FSCM is enabled it will resume operation when the device wakes from Sleep. A delay of TFSCM is applied to allow the oscillator source to stabilize before the FSCM resumes monitoring.

the following conditions are true, a delay of TFSCM will be applied when waking from SLEEP mode:

- The oscillator was shutdown while in SLEEP mode.
- The system clock is derived from a crystal oscillator source and/or the PLL.

In most cases, the TFSCM delay provides time for the OST to expire and the PLL to stabilize before device execution resumes. If the FSCM is enabled, it will begin to monitor the system clock source after the TFSCM delay expires.

10.3.1.6 Slow Oscillator Start-up

When an oscillator starts slowly, the OST and PLL lock times may not have expired before FSCM time out.

If the FSCM is enabled, then the device will detect this condition as a clock failure and a clock fail trap will occur. The device will switch to the FRC oscillator and the user can re-enable the crystal oscillator source in the clock failure Interrupt Service Routine.

If the FSCM is not enabled, then the device will simply not start executing code until the clock is stable. From the user's perspective, the device will appear to be in SLEEP until the oscillator clock has started.

10.3.1.7 USB Peripheral Control of Oscillators in SLEEP Mode

The USB module, when active, will prevent the clock source it is using from being disabled when the device enters sleep. Though the oscillator remains active the CPU and peripherals will remain halted.

Example 10-1: Put Device in SLEEP, then Wake with WDT

```
// Code example to put the Device in sleep and then Wake the device
// with the WDT

OSCCONSET = 0x10;      // set Power-Saving mode to Sleep

WDTCONCLR = 0x0002;    // Disable WDT window mode
WDTCONSET = 0x8000;    // Enable WDT
                       // WDT timeout period is set in the device configuration

while (1)
{
    ... user code ...

    WDTCONSET = 0x01;  // service the WDT
    asm volatile( “wait” );// put device in selected Power-Saving mode

                       // code execution will resume here after wake

    ... user code ...
}

// The following code fragment is at the beginning of the ‘C’ start-up code

if ( RCON & 0x18 )
{
                       // The WDT caused a wake from Sleep
    asm volatile( “eret” );// return from interrupt
}
```


10.3.2 Peripheral Bus Scaling

Most of the peripherals on the device are clocked using the PBCLK. The peripheral bus can be scaled relative to the SYSCLK to minimize the dynamic power consumed by the peripherals. The PBCLK divisor is controlled by PBDIV<1:0> (OSCCON<20:19>), allowing SYSCLK-to-PBCLK ratios of 1:1, 1:2, 1:4, and 1:8. All peripherals using PBCLK are affected when the divisor is changed. Peripherals such as the Interrupt Controller, DMA, Bus Matrix, and Prefetch Cache are clocked directly from SYSCLK, as a result, they are not affected by PBCLK divisor changes.

Most of the peripherals on the device are clocked using the PBCLK. The peripheral bus can be scaled relative to the SYSCLK to minimize the dynamic power consumed by the peripherals. The PBCLK divisor is controlled by PBDIV<1:0> (OSCCON<20:19>), allowing SYSCLK-to-PBCLK ratios of 1:1, 1:2, 1:4, and 1:8. All peripherals using PBCLK are affected when the divisor is changed. Peripherals such as USB, Interrupt Controller, DMA, Bus Matrix, and Prefetch Cache are clocked directly from SYSCLK, as a result, they are not affected by PBCLK divisor changes

Changing the PBCLK divisor affects:

- The CPU to peripheral access latency. The CPU has to wait for next PBCLK edge for a read to complete. In 1:8 mode this results in a latency of one to seven SYSCLKs.
- The power consumption of the peripherals. Power consumption is directly proportional to the frequency at which the peripherals are clocked. The greater the divisor, the lower the power consumed by the peripherals.

To minimize dynamic power the PB divisor should be chosen to run the peripherals at the lowest frequency that provides acceptable system performance. When selecting a PBCLK divider, peripheral clock requirements such as baud rate accuracy should be taken into account. For example, the UART peripheral may not be able to achieve all baud rate values at some PBCLK divider depending on the SYSCLK value.

10.3.2.1 Dynamic Peripheral Bus Scaling

The PBCLK can be scaled dynamically, by software, to save additional power when the device is in a low activity mode. The following issues need to be taken into account when scaling the PBCLK:

- All the peripherals clocked from PBCLK will scale at the same ratio, at the same time. This needs to be accounted in peripherals which need to maintain a constant baud rate, or pulse period even in low-power modes.
- Any communication through a peripheral on the peripheral bus that is in progress when the PBCLK changes may cause a data or protocol error due to a frequency change during transmission or reception.

The following steps are recommended, if the user intends to scale the PBCLK divisor dynamically:

- Disable all communication peripherals whose baud rate will be affected. Care should be taken to ensure that no communication is currently in progress before disabling the peripherals as it may result in protocol errors.
- Update the Baud Rate Generator (BRG) settings for peripherals as required for operation at the new PBCLK frequency.
- Change the peripheral bus ratio to the desired value.
- Enable all communication peripherals whose baud rate were affected.

Note:	Modifying the peripheral baud rate is done by writing to the associated peripheral SFRs. To minimize latency, the peripherals should be modified in the mode where the PBCLK is running at its highest frequency.

Example 10-2: Changing the PB Clock Divisor

```
                                        // Code example to change the PBCLK divisor
                                        // This example is for a device running at 40 MHz
                                        // Make sure that there is no UART send/receive in
progress
... user code ...
U1BRG = 0x81;                           // set baud rate for UART1 for 9600
... user code ...
SYSKEY = 0x0;                           // write invalid key to force lock
SYSKEY = 0xAA996655;                    // Write Key1 to SYSKEY
SYSKEY = 0x556699AA;                    // Write Key2 to SYSKEY
OSCCONCLR = 0x3 << 19;                  // set PB divisor to minimum (1:1)
SYSKEY = 0x0;                           // write invalid key to force lock

... user code ...
                                        // Change Peripheral Clock value
U1BRG = 0x0F;                           // set baud rate for UART1 for 9600 based on

                                        // new PB clock frequency
SYSKEY = 0x0;                           // write invalid key to force lock
SYSKEY = 0xAA996655;                    // Write Key1 to SYSKEY
SYSKEY = 0x556699AA;                    // Write Key2 to SYSKEY
OSCCONSET = 0x3 << 19;                  // set PB divisor to maximum (1:8)
SYSKEY = 0x0;                           // write invalid key to force lock

                                        // Reset Peripheral Clock
SYSKEY = 0x0;                           // write invalid key to force lock
SYSKEY = 0xAA996655;                    // Write Key1 to SYSKEY
SYSKEY = 0x556699AA;                    // Write Key2 to SYSKEY
OSCCONCLR = 0x3 << 19;                  // set PB divisor to minimum (1:1)
SYSKEY = 0x0;                           // write invalid key to force lock

U1BRG = 0x81;                           // restore baud rate for UART1 to 9600 based
                                        // on new PB clock frequency
```


10.3.3 IDLE Modes

In the IDLE modes, the CPU is halted but the System clock (SYSCLK) source is still enabled. This allows peripherals to continue to operate when the CPU is halted. Peripherals can be individually configured to halt when entering IDLE by setting their respective SIDL bit. Latency when exiting Idle mode is very low due to the CPU oscillator source remaining active.

There are four Idle modes of operation: POSC IDLE, FRC IDLE, SOSC IDLE, and LPRC IDLE.

- POSC IDLE mode: The SYSCLK is derived from the POSC. The CPU is halted, but the SYSCLK source continues to operate. Peripherals continue to operate, but can optionally be individually disabled. If the PLL is used, the Multiplier value, PLLMULT<2:0> (OSCCON<18:16>), can also be lowered to reduce power consumption by peripherals.
- FRC IDLE mode: The SYSCLK is derived from the FRC. The CPU is halted. Peripherals continue to operate, but can optionally be individually disabled. If the PLL is used, the Multiplier value, PLLMULT<2:0> (OSCCON<18:16>), can also be lowered to reduce power consumption by peripherals. The FRC clock can be further divided by a postscaler using RCDIV<2:0> (OSCCON<26:24>).
- SOSC IDLE mode: The SYSCLK is derived from the SOSC. The CPU is halted. Peripherals continue to operate, but can optionally be individually disabled.
- LPRC IDLE. The SYSCLK is derived from the LPRC. The CPU is halted. Peripherals continue to operate, but can optionally be individually disabled.

Note:	Changing the PBCLK divider ratio requires recalculation of peripheral timing. For example, assume the UART is configured for 9600 baud with a PB clock ratio of 1:1 and a POSC of 8 MHz. When the PB clock divisor of 1:2 is used, the input frequency to the baud clock is cut in half; therefore, the baud rate is reduced to 1/2 its former value. Due to numeric truncation in calculations (such as the baud rate divisor), the actual baud rate may be a tiny percentage different than expected. For this reason, any timing calculation required for a peripheral should be performed with the new PB clock frequency instead of scaling the previous value based on a change in PB divisor ratio.

Note:	Oscillator start-up and PLL lock delays are applied when switching to a clock source that was disabled and that uses a crystal and/or the PLL. For example, assume the clock source is switched from POSC to LPRC just prior to entering Sleep in order to save power. No oscillator start-up delay would be applied when exiting Idle. However, when switching back to POSC, the appropriate PLL and or Oscillator startup/lock delays would be applied.

The device enters IDLE mode when the SLPEN (OSCCON<4>) bit is clear and a `WAIT` instruction is executed.

The processor will wake or exit from Idle mode on the following events:

- On any interrupt event for which the interrupt source is enabled. The priority of the interrupt event must be greater than the current priority of CPU. If the priority of the interrupt event is lower than or equal to current priority of CPU, the CPU will remain halted and the device will remain in IDLE mode.
- On any source of device Reset.
- On a WDT time-out interrupt. See **10.4.2 “Wake-up from SLEEP or IDLE on Watchdog Time-out (NMI)”** and **Section 9. “Watchdog Timer and Power-up Timer”**.

Example 10-3: Placing Device in IDLE and Waking by ADC Event

```
                          // Code example to put the Device in Idle and then Wake the device
                          // when the ADC completes a conversion
SYSKEY = 0x0;             // write invalid key to force lock
SYSKEY = 0xAA996655;      // Write Key1 to SYSKEY
SYSKEY = 0x556699AA;      // Write Key2 to SYSKEY
OSCCONCLR = 0x10;         // set Power-Saving mode to Idle
SYSKEY = 0x0;             // write invalid key to force lock

asm volatile ( "wait" );  // put device in selected Power-Saving mode
                          // code execution will resume here after wake and the ISR is
                             complete
... user code ...

                          // interrupt handler
void __ISR(27_ADC_VECTOR, ipl7) ADC_HANDLER(void)
{
                          // interrupt handler
unsigned long int result;

result = ADC1BUF0;        // read the result
IFS1CLR = 2;              // Clear ADC conversion interrupt flag
}
```


10.4 INTERRUPTS

There are two sources of interrupts that will wake the device from a Power-Saving mode: Peripheral interrupts and an Non-Maskable Interrupt (NMI) generated by the WDT in Power-Saving mode.

10.4.1 Wake-up from SLEEP or IDLE on Peripheral Interrupt

Any source of interrupt that is individually enabled using the corresponding IE control bit in the IECx register and is operational in the current Power-Saving mode will be able to wake up the processor from SLEEP or IDLE mode. When the device wakes, one of two events will occur based on the interrupt priority:

- If the assigned priority for the interrupt is less than or equal to the current CPU priority, the CPU will remain halted and the device enters or remains in IDLE mode.
- If the assigned priority level for the interrupt source is greater than the current CPU priority, the device will wake-up and the CPU will jump to the corresponding interrupt vector. Upon completion of the ISR, CPU will start executing the next instruction after `WAIT`.

The IDLE Status bit (RCON<2>) is set upon wake-up from IDLE mode. The SLEEP Status bit (RCON<3>) is set upon wake-up from SLEEP mode.

Note:	A peripheral with an interrupt priority setting of Zero cannot wake the device.

Note:	Any applicable oscillator start-up delays are applied before the CPU resumes code execution.

10.4.2 Wake-up from SLEEP or IDLE on Watchdog Time-out (NMI)

When the WDT times out in SLEEP or IDLE mode, an NMI is generated. The NMI causes the CPU code execution to jump to the device Reset vector. Although CPU executes Reset vector, it is not a device Reset – peripherals and most CPU registers do not change their states.

Note:	Any applicable oscillator start-up delays are applied before the CPU resumes code execution.

To detect a wake from a Power-Saving mode caused by WDT expiration, the WDTO (RCON<4>), SLEEP (RCON<3>) and IDLE (RCON<2>) bits must be tested. If the WDTO bit is a '1' the event was due to a WDT time-out. The SLEEP and IDLE bits can then be tested to determine if the WDT event occurred in Sleep or Idle.

To use a WDT time-out during SLEEP mode as a wake-up interrupt, a return from interrupt (`ERET`) instruction must be used in the start-up code after the event was determined to be a WDT wake-up. This will cause code execution to continue from the instruction following the `WAIT` instruction that put the device in Power-Saving mode.

Note:	If a peripheral interrupt and WDT event occur simultaneously, or in close proximity, the NMI may not occur, due to the device being woken-up by the peripheral interrupt. To avoid unexpected WDT Reset in this scenario, the WDT is automatically cleared when the device awakens.

See **Section 9. "Watchdog Timer and Power-up Timer"** for detailed information on the WDT operation.

10.4.3 Interrupts Coincident with Power-Saving Instruction

Any peripheral interrupt that coincides with the execution of a `WAIT` instruction will be held off until entry into SLEEP or IDLE mode has completed. The device will then wake-up from SLEEP or IDLE mode.

10.5 I/O PINS ASSOCIATED WITH POWER-SAVING MODES

No device pins are associated with Power-Saving modes.

10.6 OPERATION IN DEBUG MODE

The user cannot change Clock modes when the debugger is active. Clock source changes due to the Fail-Safe Clock Monitor (FSCM) will still occur when the debugger is active.

10.7 RESETS

The behavior of the device after a Reset is determined by the type of Reset that occurred. For behavior related to the Power-Saving modes, Resets can be categorized into two groups: Power-on Reset (POR) and all other Resets (non POR).

10.7.1 Resets Other than POR During SLEEP or IDLE

The CPU will wake and code execution will begin at the device Reset vector. Any applicable oscillator delays will apply. The IDLE Status bit (RCON<2>) or SLEEP Status bit (RCON<3>) will be set to indicate the device was in a Power-Saving mode prior to the Reset.

10.7.2 POR Reset During SLEEP or IDLE

The CPU will wake and code execution will begin at the device Reset vector. Any applicable oscillator delays will apply. The IDLE Status bit (RCON<2>) or SLEEP Status bit (RCON<3>) will be forced clear. The power-saving state prior to the POR event is lost.

10.8 DESIGN TIPS

Question 1: ***What should my software do before entering SLEEP or IDLE mode?***

Answer: Make sure that the sources intended to wake the device have their IE bits set. In addition, make sure that the particular source of interrupt has the ability to wake the device. Some sources do not function when the device is in SLEEP mode.

If the device is to be placed in Idle mode, make sure that the 'stop-in-idle' control bit for each device peripheral is properly set. These control bits determine whether the peripheral will continue operation in IDLE mode. See the individual peripheral sections of this manual for further details.

Clear the WDT before entering SLEEP. If in Window mode, the WDT can only be cleared within the window period to prevent a device Reset.

Question 2: ***How do I determine which peripheral woke the device from SLEEP or IDLE mode?***

Answer: Most peripherals have a unique interrupt vector. If needed, you can poll the IF bits for each enabled interrupt source to determine the source of wake-up.

10.9 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the PIC32MX device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the Power-Saving modes include the following:

Title	Application Note #
Low-Power Design using PIC® Microcontrollers	AN606

Note: Please visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the PIC32MX family of devices.

10.10 REVISION HISTORY

Revision A (October 2007)

This is the initial released version of this document.

Revision B (October 2007)

Updated document to remove Confidential status.

Revision C (April 2008)

Revised status to Preliminary; Revised U-0 to r-x.

Revision D (July 2008)

Revised Example 10-1 and 10-3; Revised Table 10-1; Revised Register 10-5 and 10-9; Revised Section 10.3.2 (2nd para.); Change Reserved bits from “Maintain as” to “Write”; Added Note to ON bit (WDTCON Register).

Revision E (July 2008)

Revised Examples 10-2, 10-3.

NOTES:

Section 11. Reserved for Future

NOTES:

Section 12. I/O Ports

HIGHLIGHTS

This section of the manual contains the following topics:

12.1 INTRODUCTION

The general purpose I/O pins can be considered the simplest of peripherals. They allow the PIC32MX microcontroller to monitor and control other devices. To add flexibility and functionality to a device, some pins are multiplexed with alternate function(s). These functions depend on which peripheral features are on the device. In general, when a peripheral is functioning, that pin may not be used as a general purpose I/O pin.

Following are some of the key features of this module:

- Individual output pin open-drain enable/disable
- Individual input pin pull-up enable/disable
- Monitor select inputs and generate interrupt on mismatch condition
- Operate during CPU SLEEP and IDLE modes
- Fast bit manipulation using CLR, SET and INV registers

A block diagram of a typical I/O port structure is shown in Figure 12-1. The diagram depicts the many peripheral functions that can be multiplexed onto the I/O pin.

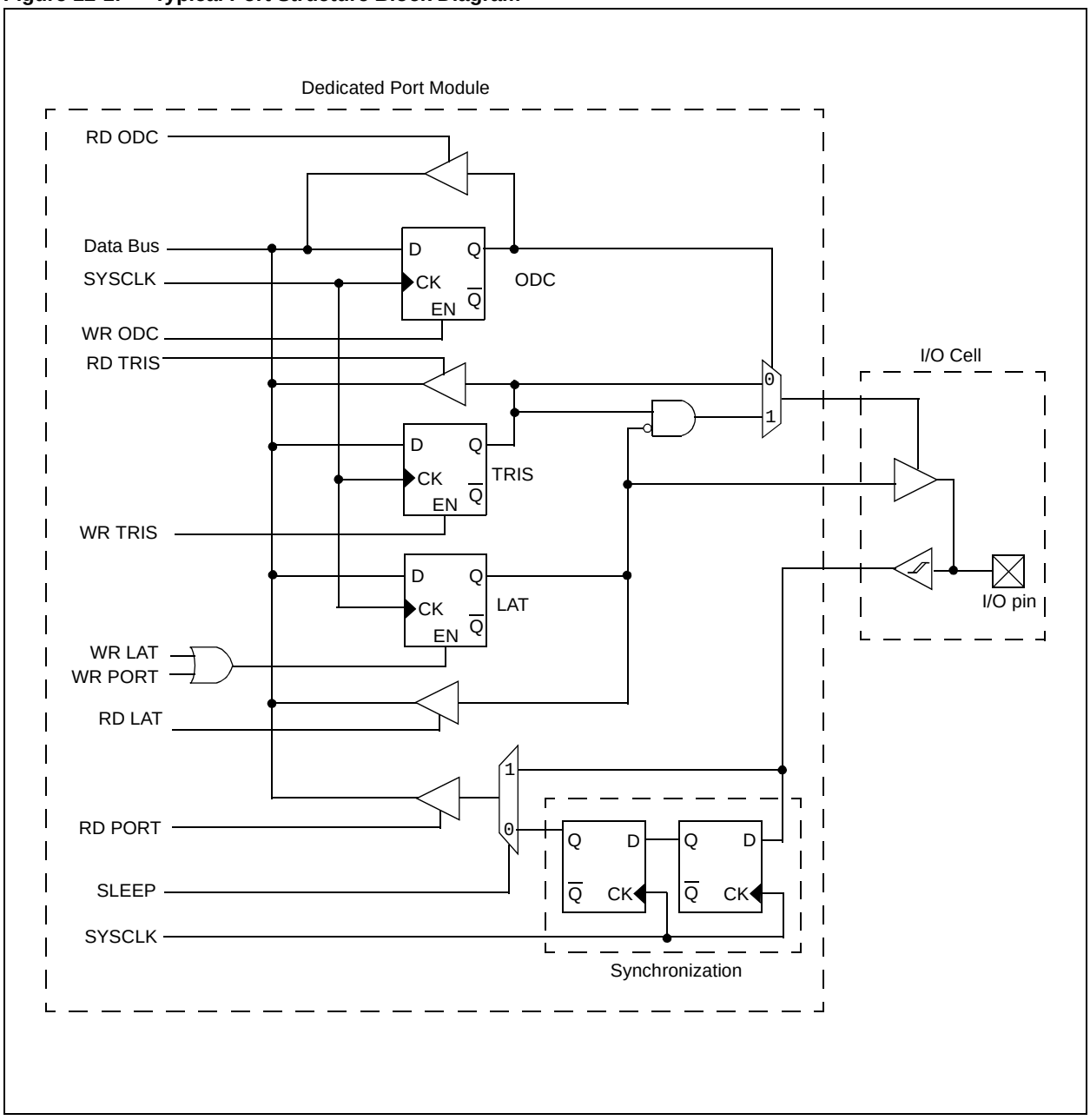
The I/O Ports module consists of the following Special Function Registers (SFRs):

- TRISx: Data Direction register for the module ‘x’
- PORTx: PORT register for the module ‘x’
- LATx: Latch register for the module ‘x’
- ODCx: Open-Drain Control register for the module ‘x’
- CNCON: Interrupt-on-Change Control register
- CNEN: Input Change Notification Interrupt Enable register
- CNPUE: Input Change Notification Pull-up Enable register

The I/O Ports module also has the following associated bits for interrupt control:

- Interrupt Enable Control bits for CN events (CNIE) in INT register IEC1: Interrupt Enable Control Register 1.
- Interrupt Flag Status bits for CN events (CNIF) in INT register IFS1: Interrupt Flag Status Register 1.
- Interrupt Priority Control bits (CNIP<2:0>) in INT register IPC6: Interrupt Priority Control Register 6.

Figure 12-1: Typical Port Structure Block Diagram



12.2 CONTROL REGISTERS

Before reading and writing any I/O port, the desired pin(s) should be properly configured for the application. Each I/O port has three registers directly associated with the operation of the port: TRIS, PORT and LAT. Each I/O port pin has a corresponding bit in these registers. Depending on the PIC32MX device variant, up to seven I/O ports are available. Through out this section, the letter 'x', denotes any or all port module instances. For example "TRISx" would represent TRISA, TRISB, TRISC, etc. Any bit and its associated data and control registers that is not valid for a particular device will be disabled and will read as zeros.

Note:	The total number of ports and available I/O pins will depend on the device variant. In a given device, all of the bits in a port control register might not be available. Refer to the specific device data sheet for further details.

12.2.1 TRIS (tri-state) Registers

TRIS registers configure the data direction flow through port I/O pin(s). The TRIS register bits determine whether a PORT I/O pin is an input or an output.

- A TRIS bit set = `1` configures the corresponding I/O port pin as an input.
- A TRIS bit set = `0` configures the corresponding I/O port pin as an output.
- A read from a TRIS register reads the last value written to the TRIS register.
- All I/O port pins are defined as inputs after a Power-on Reset.

12.2.2 PORT Registers

PORT registers allow I/O pins to be accessed (read).

- A write to a PORT register writes to the corresponding LAT register (PORT data latch). Those I/O port pin(s) configured as outputs are updated.
- A write to a PORT register is the effectively the same as a write to a LAT register.
- A read from a PORT register reads the synchronized signal applied to the port I/O pins.

12.2.3 LAT Registers

LAT registers (PORT data latch) hold data written to port I/O pin(s).

- A write to a LAT register latches data to corresponding port I/O pins. Those I/O port pin(s) configured as outputs are updated.
- A read from LAT register reads the data held in the PORT data latch, not from the port I/O pins.

12.2.4 SET, CLR, INV I/O Port Registers

In addition to the TRIS, PORT, and LAT base registers, each port module is associated with a SET, CLR and INV register which provides atomic bit manipulations and allowing faster I/O pin operations. As the name of the register implies, a value written to a SET, CLR or INV register effectively performs the implied operation, but only on the corresponding base register and only bits specified as '1' are modified. Bits specified as '0' are not modified.

- Writing 0x0001 to TRISASET register sets only bit 0 in base register TRISA
- Writing 0x0020 to PORTDCLR register clears only bit 5 in base register PORTD
- Writing 0x9000 to LATCINV register inverts only bits 15 and 12 in the base register LATC.

Reading SET, CLR and INV registers return an undefined value. To see the affects of a write operation to a SET, CLR or INV register, the base register must be read instead.

The SET, CLR and INV registers are not exclusive to TRIS, PORT and LAT registers. Other I/O port module registers ODC, CNEN and CNPUE also feature these bit manipulation registers.

A typical method to toggle an I/O pin requires a read-modify-write operation performed on a PORT register in software. For example, a read from a PORTx register, mask and modify the desired output bit(s), write the resulting value back to the PORTx register. This method is vulnerable to a read-modify-write issue where the port value may change after it is read and before the modified data can be written back, thus changing the previous state. This method also requires more instructions.

```
PORTA ^= 0x0001;
```

A more efficient and atomic method uses the PORTxINV register. A write to the PORTxINV register effectively performs a read-modify-write operation on the target base register, equivalent to the software operation described above, however, it is done in hardware. To toggle an I/O pin using this method, a "1" is written to the corresponding bit in the PORTxINV register. This operation will read the PORTx register, invert only those bits specified as '1' and write the resulting value to the LATx register, thus toggling the corresponding I/O pin(s) all in a single atomic instruction cycle.

```
PORTAINV = 0x0001;
```


TRISx SET,CLR,INV Register Behavior

- A value written to a TRISxSET register reads the TRISx base register, sets any bit(s) specified as '1', writes the modified value back to the TRISx base register.
- A value written to a TRISxCLR register reads the TRISx base register, clears any bit(s) specified as '1', writes the modified value back to the TRISx base register.
- A value written to a TRISxINV register reads the TRISx base register, inverts any bit(s) specified as '1', writes the modified value back to the TRISx base register.
- Any bit(s), specified as '0', are not modified.

PORTx SET,CLR,INV Register Behavior

- A value written to a PORTxSET register reads the PORTx base register, sets any bit(s) specified as '1', writes the modified value back to the LATx base register. Those I/O port pin(s) configured as outputs are updated.
- A value written to a PORTxCLR register reads the PORTx base register, clears any bit(s) specified as '1', writes the modified value back to the LATx base register. Those I/O port pin(s) configured as outputs are updated.
- A value written to a PORTxINV register reads the PORTx base register, inverts any bit(s) specified as '1', writes the modified value back to the LATx base register. Those I/O port pin(s) configured as outputs are updated.
- Any bit(s), specified as '0', are not modified.

LATx SET,CLR,INV Register Behavior

- A value written to a LATxSET register reads the LATx base register, sets any bit(s) specified as '1', writes the modified value back to the LATx base register. Those I/O port pin(s) configured as outputs are updated.
- A value written to a LATxCLR register reads the LATx base register, clears any bit(s) specified as '1', writes the modified value back to the LATx base register. Those I/O port pin(s) configured as outputs are updated.
- A value written to a LATxINV register reads the LATx base register, inverts any bit(s) specified as '1', writes the modified value back to the LATx base register. Those I/O port pin(s) configured as outputs are updated.

Any bit(s), specified as '0', are not modified.

12.2.5 ODC Registers

Each I/O pin can be individually configured for either normal digital output or open-drain output. This is controlled by the Open-Drain Control register, ODCx, associated with each I/O pin. If the ODC bit for an I/O pin is a '1', then the pin acts as an open-drain output. If the ODC bit for an I/O pin is a '0', then the pin is configured for a normal digital output (ODC bit is valid only for output pins). After a Reset, the status of all the bits of the ODCx register is set to '0'.

The open-drain feature allows the generation of outputs higher than VDD on any desired digital only pins by using external pull-up resistors. The maximum open-drain voltage allowed is the same as the maximum VIH specification. The ODC register setting takes effect in all the I/O modes, allowing the output to behave as an open-drain even if a peripheral is controlling the pin. Although the user could achieve the same effect by manipulating the corresponding LAT and TRIS bits, this procedure will not allow the peripheral to operate in Open-Drain mode (except for the default operation of the I2C™ pins). Since I2C pins are already open-drain pins, the ODCx settings do not affect the I2C pins. Also, the ODCx settings do not affect the JTAG output characteristics as the JTAG scan cells are inserted between the ODCx logic and the I/O.

12.2.6 CN Control Registers

Several I/O pins may be individually configured to generate an interrupt when a change on an input pin is detected. There are three control registers associated with the CN (Change Notice) module. The CNCON control register is used to enable or disable the CN module. The CNEN register contains the CNENx control bits, where 'x' denotes the number of the CN input pin. The CNPUE register contains the CNPUEx control bits. Each CN pin has a pull-up device connected to the pin which can be enabled or disabled using the CNPUEx control bits. The pull-up devices act as a current source that is connected to the pin and eliminate the need for external resistors when push button or keypad devices are connected. Refer to the **"Electrical Characteristics"** section of the specific device data sheet for CN pull-up device current specifications.

The following table provides a brief summary of all I/O ports-related registers. Corresponding registers appear after the summary, followed by a detailed description of each register.

Table 12-1: I/O Ports SFR Summary

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
TRISx	31:0	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	TRISx<15:8>							
	7:0	TRISx<7:0>							
TRISxCLR	31:0	Write clears selected bits in TRISx, read yields undefined value							
TRISxSET	31:0	Write sets selected bits in TRISx, read yields undefined value							
TRISxINV	31:0	Write inverts selected bits in TRISx, read yields undefined value							
PORTx	31:0	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	PORTx<15:8>							
	7:0	PORTx<7:0>							
PORTxCLR	31:0	Write clears selected bits in LATx, read yields undefined value							
PORTxSET	31:0	Write sets selected bits in LATx, read yields undefined value							
PORTxINV	31:0	Write inverts selected bits in LATx, read yields undefined value							
LATx	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	LATx<15:8>							
	7:0	LATx<7:0>							
LATxCLR	31:0	Write clears selected bits in LATx, read yields undefined value							
LATxSET	31:0	Write sets selected bits in LATx, read yields undefined value							
LATxINV	31:0	Write inverts selected bits in LATx, read yields undefined value							
ODCx	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	ODCx<5:8>							
	7:0	ODCx<7:0>							
ODCxCLR	31:0	Write clears selected bits in ODCx, read yields undefined value							
ODCxSET	31:0	Write sets selected bits in ODCx, read yields undefined value							
ODCxINV	31:0	Write inverts selected bits in ODCx, read yields undefined value							

Table 12-1: I/O Ports SFR Summary (Continued)

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
CNCON	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	ON	FRZ	SIDL	—	—	—	—	—
	7:0	—	—	—	—	—	—	—	—
CNCONCLR	31:0	Write clears selected bits in CNCON, read yields undefined value							
CNCONSET	31:0	Write sets selected bits in CNCON, read yields undefined value							
CNCONINV	31:0	Write inverts selected bits in CNCON, read yields undefined value							
CNEN	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	CNEN<21:16>					
	15:8	CNEN<15:8>							
	7:0	CNEN<7:0>							
CNENCLR	31:0	Write clears selected bits in CNEN, read yields undefined value							
CNENSET	31:0	Write sets selected bits in CNEN, read yields undefined value							
CNENINV	31:0	Write inverts selected bits in CNEN, read yields undefined value							
CNPUE	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	CNPUE<21:16>					
	15:8	CNPUE<15:8>							
	7:0	CNPUE<7:0>							
CNPUECLR	31:0	Write clears selected bits in CNPUE read yields undefined value							
CNPUESET	31:0	Write sets selected bits in CNPUE, read yields undefined value							
CNPUEINV	31:0	Write inverts selected bits in CNPUE, read yields undefined value							
IEC1	31:24	—	—	—	—	—	—	USBIE	FCEIE
	23:16	—	—	—	—	DMA3IE	DMA2IE	DMA1IE	DMA0IE
	15:8	RTCCIE	FSCMIE	I2C2MIE	I2C2SIE	I2C2BIE	U2TXIE	U2RXIE	U2EIE
	7:0	SPI2RXIE	SPI2TXIE	SPI2EIE	CMP2IE	CMP1IE	PMPIE	AD1IE	CNIE
IFS1	31:24	—	—	—	—	—	—	USBIF	FCEIF
	23:16	—	—	—	—	DMA3IF	DMA2IF	DMA1IF	DMA0IF
	15:8	RTCCIF	FSCMIF	I2C2MIF	I2C2SIF	I2C2BIF	U2TXIF	U2RXIF	U2EIF
	7:0	SPI2RXIF	SPI2TXIF	SPI2EIF	CMP2IF	CMP1IF	PMPIF	AD1IF	CNIF
IPC6	31:24	—	—	—	AD1IP<2:0>			AD1IS<1:0>	
	23:16	—	—	—	CNIP<2:0>			CNIS<1:0>	
	15:8	—	—	—	I2C1IP<2:0>			I2C1IS<1:0>	
	7:0	—	—	—	U1IP<2:0>			U1IS<1:0>	

Register 12-1: TRISx: TRIS Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TRIS<15:8>							
bit 15							bit 8

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TRIS<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15-0 **TRISx<15:0>:** TRIS Register bits

1 = Corresponding port pin “Input”

0 = Corresponding port pin ‘Output”

Register 12-2: TRISxCLR: TRIS Clear Register

Write clears selected bits in TRISx, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in TRISx**

A write of '1' in one or more bit positions clears the corresponding bit(s) in TRISx register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `TRISxCLR = 0x00008001` will clear bits 15 and 0 in TRISx register.

Register 12-3: TRISxSET: TRIS Set Register

Write sets selected bits in TRISx, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in TRISx**

A write of '1' in one or more bit positions sets the corresponding bit(s) in TRISx register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `TRISxSET = 0x00008001` will set bits 15 and 0 in TRISx register.

Register 12-4: TRISxINV: TRIS Invert Register

Write inverts selected bits in TRISx, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in TRIS**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in TRISx register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `TRISxINV = 0x00008001` will invert bits 15 and 0 in TRISx register.

Register 12-5: PORTx: PORT Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
PORT<15:8>							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
PORT<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15-0 **PORTx<15:0>:** PORT Register bits
Read = Value on port pins
Write = Value written to the LATx register, PORT latch and I/O pins

Register 12-6: PORTxCLR: PORT Clear Register

Write clears selected bits in LATx, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in LATx**

A write of '1' in one or more bit positions clears the corresponding bit(s) in LATx register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `PORTxCLR = 0x00008001` will clear bits 15 and 0 in LATx register.

Register 12-7: PORTxSET: PORT Set Register

Write sets selected bits in LATx, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in LATx**

A writes of '1' in one or more bit positions sets the corresponding bit(s) in LATx register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `PORTxSET = 0x00008001` will set bits 15 and 0 in LATx register.

Register 12-8: PORTxINV: PORT Invert Register

Write inverts selected bits in LATx, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in LATx**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in LATx register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `PORTxINV = 0x00008001` will invert bits 15 and 0 in LATx register.

Register 12-9: LATx: LAT Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-1x	R/W-x	R/W-x
LAT<15:8>							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
LAT<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15-0 **LATx<15:0>:** LAT Register bits
Read = Value on PORT latch, not I/O pins
Write = Value written to PORT latch and I/O pins

Register 12-10: LATxCLR: LAT Clear Register

Write clears selected bits in LATx, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in LATx**

A write of '1' in one or more bit positions clears the corresponding bit(s) in LATx register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `LATxCLR = 0x00008001` will clear bits 15 and 0 in LATx register.

Register 12-11: LATxSET: LAT Set Register

Write sets selected bits in LATx, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in LATx**

A write of '1' in one or more bit positions sets the corresponding bit(s) in LATx register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `LATxSET = 0x00008001` will set bits 15 and 0 in LATx register.

Register 12-12: LATxINV: LAT Invert Register

Write inverts selected bits in LATx, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in LATx**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in LATx register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `LATxINV = 0x00008001` will invert bits 15 and 0 in LATx register.

Register 12-13: ODCx: Open Drain Configuration Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ODCx<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ODCx<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15-0 **ODCx<15:0>:** ODCx Register bits

If a port pin is configured as an output (corresponding TRISx bit = 0)
1 = Port pin open-drain output enabled
0 = Port pin open-drain output disabled
If a port pin is configured as an input, ODCx bits have no effect.

Register 12-14: ODCxCLR: Open Drain Configuration Clear Register

Write clears selected bits in ODCx, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in ODCx**

A write of '1' in one or more bit positions clears the corresponding bit(s) in ODCx register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `ODCxCLR = 0x00008001` will clear bits 15 and 0 in ODCx register.

Register 12-15: ODCxSET: Open Drain Configuration Set Register

Write sets selected bits in ODCx, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in ODCx**

A write of '1' in one or more bit positions sets the corresponding bit(s) in ODCx register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `ODCxSET = 0x00008001` will set bits 15 and 0 in ODCx register.

Register 12-16: ODCxINV: Open Drain Configuration Invert Register

Write inverts selected bits in ODCx, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in ODCx**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in ODCx register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `ODCxINV = 0x00008001` will invert bits 15 and 0 in ODCx register.

Register 12-17: CNCON: Interrupt-On-Change Control Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	r-x	r-x	r-x	r-x	r-x
ON	FRZ	SIDL	—	—	—	—	—
bit 15							bit 8

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15 **ON:** Change Notice Module On bit

1 = CN Module is enabled
0 = CN Module is disabled

Note: When using 1:1 PBCLK divisor, the user’s software should not read/write the peripheral’s SFRs in the SYSCLK cycle immediately following the instruction that clears the module’s ON bit.

bit 14 **FRZ:** Freeze in Debug Exception Mode bit

1 = Freeze operation when CPU is in Debug Exception mode
0 = Continue operation when CPU is in Debug Exception mode

Note: FRZ is writable in Debug Exception mode only, it is forced to ‘0’ in normal mode.

bit 13 **SIDL:** Stop in IDLE Mode bit

1 = Discontinue operation when device enters IDLE mode
0 = Continue operation in IDLE mode

bit 12-0 **Reserved:** Write ‘0’; ignore read

Register 12-18: CNCONCLR: Interrupt-On-Change Control Clear Register

Write clears selected bits in CNCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in CNCON**

A write of '1' in one or more bit positions clears the corresponding bit(s) in CNCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `CNCONCLR = 0x00008000` will clear bit 15 in CNCON register.

Register 12-19: CNCONSET: Interrupt-On-Change Control Set Register

Write sets selected bits in CNCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in CNCON**

A write of '1' in one or more bit positions sets the corresponding bit(s) in CNCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `CNCONSET = 0x00008000` will set bit 15 in CNCON register.

Register 12-20: CNCONINV: Interrupt-On-Change Control Invert Register

Write inverts selected bits in CNCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in CNCON**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in CNCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `CNCONINV = 0x00008000` will invert bit 15 in CNCON register.

Register 12-21: CNEN: Input Change Notification Interrupt Enable Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CNEN21	CNEN20	CNEN19	CNEN18	CNEN17	CNEN16
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CNEN15	CNEN14	CNEN13	CNEN12	CNEN11	CNEN10	CNEN9	CNEN8
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CNEN7	CNEN6	CNEN5	CNEN4	CNEN3	CNEN2	CNEN1	CNEN0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-22 **Reserved:** Write ‘0’; ignore read

bit 21-0 **CNEN<21:0>:** CNEN Register bits

If a port pin is configured as an input (corresponding TRISx bit = 1)
1 = Port pin input change notice enabled
0 = Port pin input change notice disabled
If a port pin is configured as an output, CNENx bits have no effect.

Register 12-22: CNENCLR: Input Change Notification Interrupt Enable Register Clear Register

Write clears selected bits in CNEN, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in CNEN**

A write of '1' in one or more bit positions clears the corresponding bit(s) in CNEN register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `CNENCLR = 0x00008001` will clear bits 15 and 0 in CNEN register.

Register 12-23: CNENSET: Input Change Notification Interrupt Enable Register Set Register

Write sets selected bits in CNEN, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in CNEN**

A write of '1' in one or more bit positions sets the corresponding bit(s) in CNEN register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `CNENSET = 0x00008001` will set bits 15 and 0 in CNEN register.

Register 12-24: CNENINV: Input Change Notification Interrupt Enable Register Invert Register

Write inverts selected bits in CNEN, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in CNEN**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in CNEN register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `CNENINV = 0x00008001` will invert bits 15 and 0 in CNEN register.

Register 12-25: CNPUE: Input Change Notification Pull-up Enable Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CNPUE21	CNPUE20	CNPUE19	CNPUE18	CNPUE17	CNPUE16
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CNPUE15	CNPUE14	CNPUE13	CNPUE12	CNPUE11	CNPUE10	CNPUE9	CNPUE8
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CNPUE7	CNPUE6	CNPUE5	CNPUE4	CNPUE3	CNPUE2	CNPUE1	CNPUE0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-22 **Reserved:** Write ‘0’; ignore read

bit 21-0 **CNPUE<21:0>:** CNPUE Register bits

If a port pin is configured as an input (corresponding TRISx bit = 1)
1 = port pin pull-up enabled
0 = port pin pull-up disabled
If a port pin is configured as an output, the corresponding CNPUEx bit should be disabled.

Register 12-26: CNPUECLR: Interrupt Change Pull-up Enable Clear Register

Write clears selected bits in CNPUE, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in CNPUE**

A write of '1' in one or more bit positions clears the corresponding bit(s) in CNPUE register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `CNPUECLR = 0x00008001` will clear bits 15 and 0 in CNPUE register.

Register 12-27: CNPUESET: Interrupt Change Pull-up Enable Set Register

Write sets selected bits in CNPUE, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in CNPUE**

A write of '1' in one or more bit positions sets the corresponding bit(s) in CNPUE register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `CNPUESET = 0x00008001` will set bits 15 and 0 in CNPUE register.

Register 12-28: CNPUEINV: Interrupt Change Pull-up Enable Invert Register

Write inverts selected bits in CNPUE, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in CNPUE**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in CNPUE register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `CNPUEINV = 0x00008001` will invert bits 15 and 0 in CNPUE register.

Register 12-29: IEC1: Interrupt Enable Control Register 1(1)

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	USBIE	FCEIE
bit 31							bit 24

r-x	r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	DMA3IE	DMA2IE	DMA1IE	DMA0IE
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RTCCIE	FSCMIE	I2C2MIE	I2C2SIE	I2C2BIE	U2TXIE	U2RXIE	U2EIE
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SPI2RXIE	SPI2TXIE	SPI2EIE	CMP2IE	CMP1IE	PMPIE	AD1IE	CNIE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 0 **CNIE:** Change Notice Interrupt Enable bit

1 = Interrupt is enabled

0 = Interrupt is disabled

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the I/O Port Change Notice.

Register 12-30: IFS1: Interrupt Flag Status Control Register 1(1)

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	USBIF	FCEIF
bit 31							bit 24

r-x	r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	DMA3IF	DMA2IF	DMA1IF	DMA0IF
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RTCCIF	FSCMIF	I2C2MIF	I2C2SIF	I2C2BIF	U2TXIF	U2RXIF	U2EIF
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SPI2RXIF	SPI2TXIF	SPI2EIF	CMP2IF	CMP1IF	PMPIF	AD1IF	CNIF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 0 **CNIE:** Change Notice Interrupt Enable bit

1 = Interrupt is enabled

0 = Interrupt is disabled

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the I/O Port Change Notice.

Register 12-31: IPC6: Interrupt Priority Control Register 6(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	AD1IP<2:0>			AD1IS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	CNIP<2:0>			CNIS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	I2C1IP<2:0>			I2C1IS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	U1IP<2:0>			U1IS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 20-18 **CNIP<2:0>:** Change Notice Interrupt Priority bits

111 = Interrupt Priority is 7
110 = Interrupt Priority is 6
101 = Interrupt Priority is 5
100 = Interrupt Priority is 4
011 = Interrupt Priority is 3
010 = Interrupt Priority is 2
001 = Interrupt Priority is 1
000 = Interrupt is disabled

bit 17-16 **CNIS<1:0>:** Change Notice Interrupt Subpriority bits

11 = Interrupt Subpriority is 3
10 = Interrupt Subpriority is 2
01 = Interrupt Subpriority is 1
00 = Interrupt Subpriority is 0

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the I/O Port Change Notice.

12.3 MODES OF OPERATION

12.3.1 Digital Inputs

Pins are configured as digital inputs by setting the corresponding TRIS register bits = 1. When configured as inputs, they are Schmitt Triggers. Several digital pins share functionality with analog inputs and default to the analog inputs at Power-on Reset. Setting the corresponding bit in the AD1PCFG register = 1 enables the pin as a digital pin.

Note:	Refer to the specific device data sheet for further details regarding input buffer types. To use pins that are multiplexed with the 10-bit Analog-to-Digital Converter (A/D) module for digital I/O, the corresponding bits in the AD1PCFG register must be set to '1' – even if the A/D module is turned off.

12.3.2 Analog Inputs

Certain pins can be configured as analog inputs used by the A/D and Comparator modules. Setting the corresponding bits in the AD1PCFG register = 0 enables the pin as an analog input pin, independent of the TRIS register setting for the corresponding pin.

12.3.3 Digital Outputs

Pins are configured as digital outputs by setting the corresponding TRIS register bits = 0. When configured as digital outputs, these pins are CMOS drivers or can be configured as open-drain outputs by setting the corresponding bits in the ODC register.

12.3.4 Analog Outputs

Certain pins can be configured as analog outputs, such as the CVREF output voltage used in the Comparator module. Configuring the Comparator module to provide this output will present the analog output voltage on the pin, independent of the TRIS register setting for the corresponding pin.

Note:	Refer to the specific device data sheet for further details regarding the use of A\D and Comparator modules.

12.3.5 Open-Drain Configuration

In addition to the PORT, LAT and TRIS registers for data control, each port pin configured as a digital output can also select between an active drive output and open-drain output. This is controlled by the Open-Drain Control register, ODCx, associated with each port. From Power-on Reset, when an I/O pin is configured as a digital output, its output is active drive by default. Setting a bit in the ODCx register = 1 configures the corresponding pin as an open-drain output.

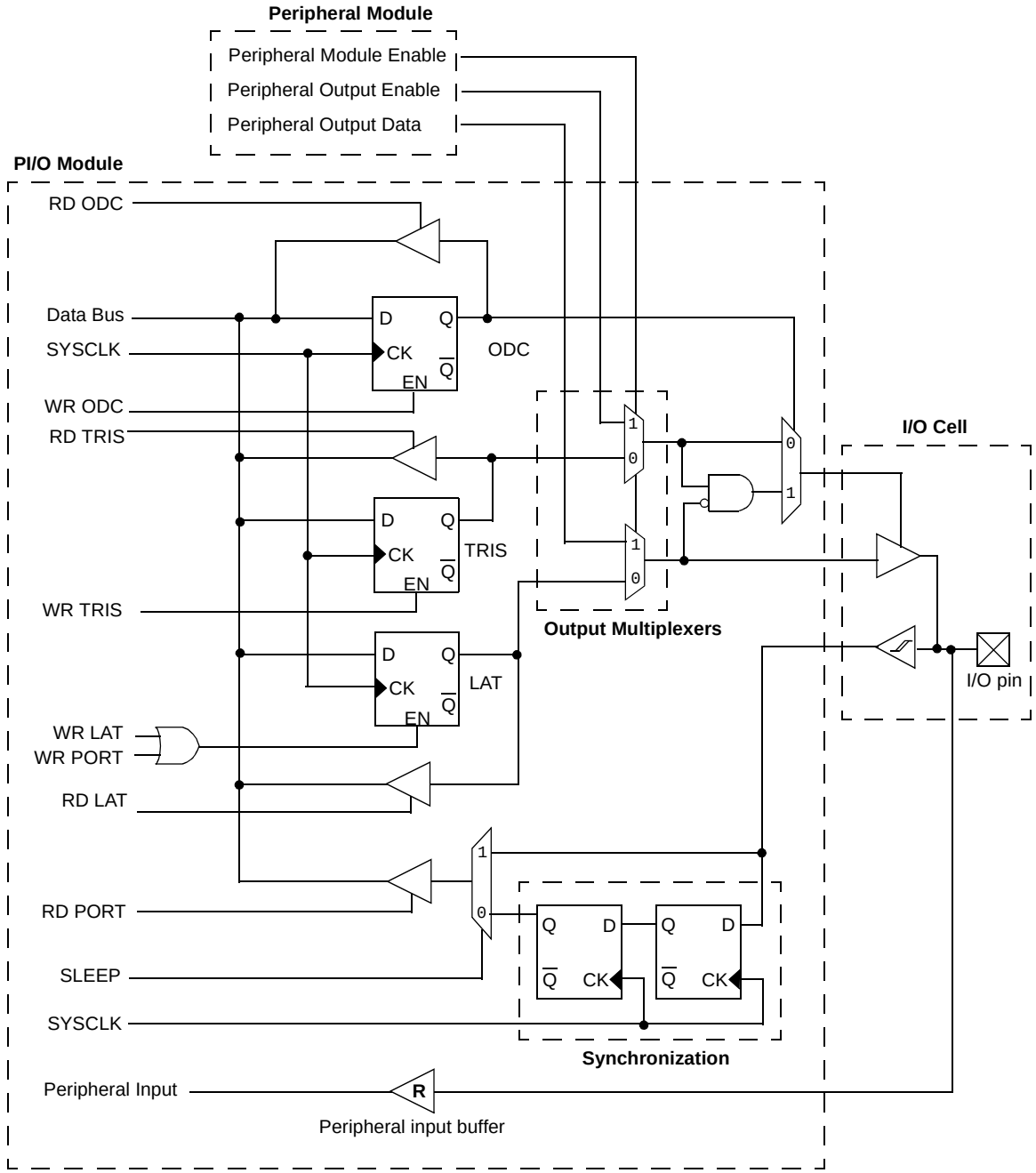
The open-drain feature allows the generation of outputs higher than V_{DD} (e.g., 5V) on any desired digital-only pins by using external pull up resistors. The maximum open-drain voltage allowed is the same as the maximum V_{IH} specification.

12.3.6 Peripheral Multiplexing

Many pins also support one or more peripheral modules. When configured to operate with a peripheral, a pin may not be used for general input or output. In many cases, a pin must still be configured for input or output, although some peripherals override the TRIS configuration. Figure 12-2 shows how ports are shared with other peripherals, and the associated I/O pin to which they are connected. For some PIC32MX devices, multiple peripheral functions may be multiplexed on each I/O pin. The priority of the peripheral function depends on the order of the pin description in the pin diagram of the specific product data sheet.

Note that the output of a pin can be controlled by the TRISx register bit or, in some cases, by the peripheral itself.

Figure 12-2: Block Diagram of a Typical Shared Port Structure



Notes: This block diagram is a general representation of a shared port/peripheral structure for illustration purposes only. The actual structure for any specific port/peripheral combination may be different than what is shown here.

R = Peripheral input buffer types may vary. Refer to the PIC32MX data sheet for peripheral details.

12.3.6.1 Multiplexed Digital Input Peripheral

The following conditions are characteristic of a multiplexed digital input peripheral:

- Peripheral does not control the TRISx register.
  Some peripherals require the pin be configured as an input by setting the corresponding TRISx bit = 1.
- Peripheral input path is independent of I/O input path and uses an input buffer that is dependent on the peripheral.
- PORTx register data input path is not affected and is able to read the pin value.

12.3.6.2 Multiplexing Digital Output Peripheral

The following conditions are characteristic of a multiplexed digital output peripheral:

- Peripheral controls the output data.
  Some peripherals require the pin be configured as an output by setting the corresponding TRISx bit = 0.
- If a peripheral pin has an automatic tri-state feature, e.g., PWM outputs, the peripheral has the ability to tri-state the pin.
- Pin output driver type could be affected by peripheral, e.g., drive strength, slew rate, etc.
- PORTx register output data has no effect.

12.3.6.3 Multiplexing Digital Bidirectional Peripheral

The following conditions are characteristic of a multiplexed digital bidirectional peripheral:

- Peripheral automatically configures the pin as an output, but not as an input.
  Some peripherals require the pin be configured as an input by setting the corresponding TRISx bit = 1.
- Peripherals control output data.
- Pin output driver type could be affected by peripheral (e.g., drive strength, slew rate, etc.).
- PORTx register data input path is not affected and is able to read the pin value.
- PORTx register output data has no effect.

12.3.6.4 Multiplexing Analog Input Peripheral

The following condition is characteristic of a multiplexed analog input peripheral:

All digital port input buffers are disabled and PORTx registers read '0' to prevent crowbar current.

12.3.6.5 Multiplexing Analog Output Peripheral

The following conditions are characteristic of a multiplexed analog output peripheral:

- All digital port input buffers are disabled and PORTx registers read '0' to prevent crowbar current.
- Analog output is driven onto the pin independent of the associated TRISx setting.

Note:	In order to use pins that are multiplexed with the A/D module for digital I/O, the corresponding bits in the AD1PCFG register must be set to '1' – even if the A/D module is turned off.

12.3.6.6 Software Input Pin Control

Some of the functions assigned to an I/O pin may be input functions that do not take control of the pin output driver. An example of one such peripheral is the input capture module. If the I/O pin associated with the input capture is configured as an output, using the appropriate TRIS control bit, the user can manually affect the state of the input capture pin through its corresponding LAT register. This behavior can be useful in some situations, especially for testing purposes, when no external signal is connected to the input pin.

As shown in Figure 12-2, the organization of the peripheral multiplexers will determine if the peripheral input pin can be manipulated in software using the PORT register. The conceptual peripherals shown in this figure disconnect the PORT data from the I/O pin when the peripheral function is enabled.

In general, the following peripherals allow their input pins to be controlled manually through the LAT registers:

- External Interrupt pins
- Input Capture pins
- Timer Clock Input pins
- PWM Fault pins

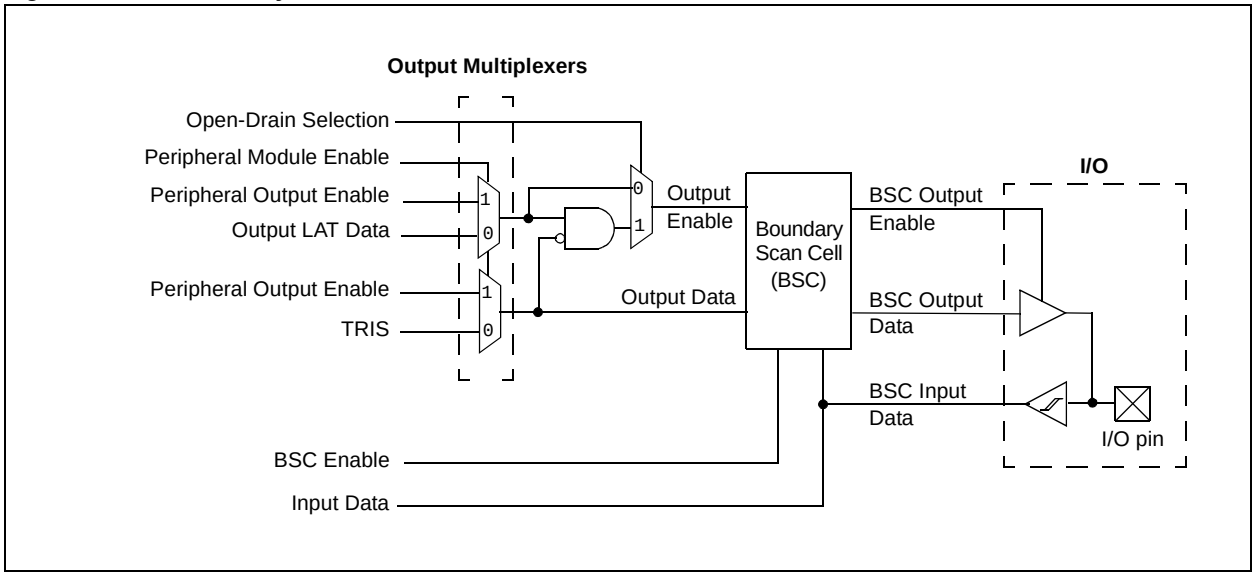
Most serial communication peripherals, when enabled, take full control of the I/O pin so that the input pins associated with the peripheral cannot be affected through the corresponding PORT registers. These peripherals include the following modules:

- SPI
- I^2C
- UART
- UART

12.3.7 Boundary Scan Cell Connections

The PIC32MX device supports JTAG boundary scan. A Boundary Scan Cell (BSC) is inserted between the internal I/O logic circuit and the I/O pin, as shown in Figure 12-3. Most of the I/O pads have boundary scan cells, however, JTAG pads do not. For normal I/O operation, the BSC is disabled and hence bypassed: The output enable input of the BSC is directly connected to the BSC output enable, and the output data input of the BSC is directly connected to the BSC output data. The pads that do not have BSC are the power supply pads (V_{DD}, V_{SS} and V_{CAP}/V_{DDCORE}) and the JTAG pads (T_{CK}, T_{DI}, T_{DO} and TMS).

Figure 12-3: Boundary Scan Cell Connections



12.3.8 Port Descriptions

Refer to the specific device data sheet for a description of the available I/O ports and peripheral multiplexing details.

12.3.9 Change Notification Pins

The Change Notification (CN) pins provide PIC32MX devices the ability to generate interrupt requests to the processor in response to a change of state on selected input pins (corresponding TRISx bits must = 1). Up to 22 input pins may be selected (enabled) for generating CN interrupts. The total number of available CN inputs is dependent on the selected PIC32MX device. Refer to the specific device data sheet for further details.

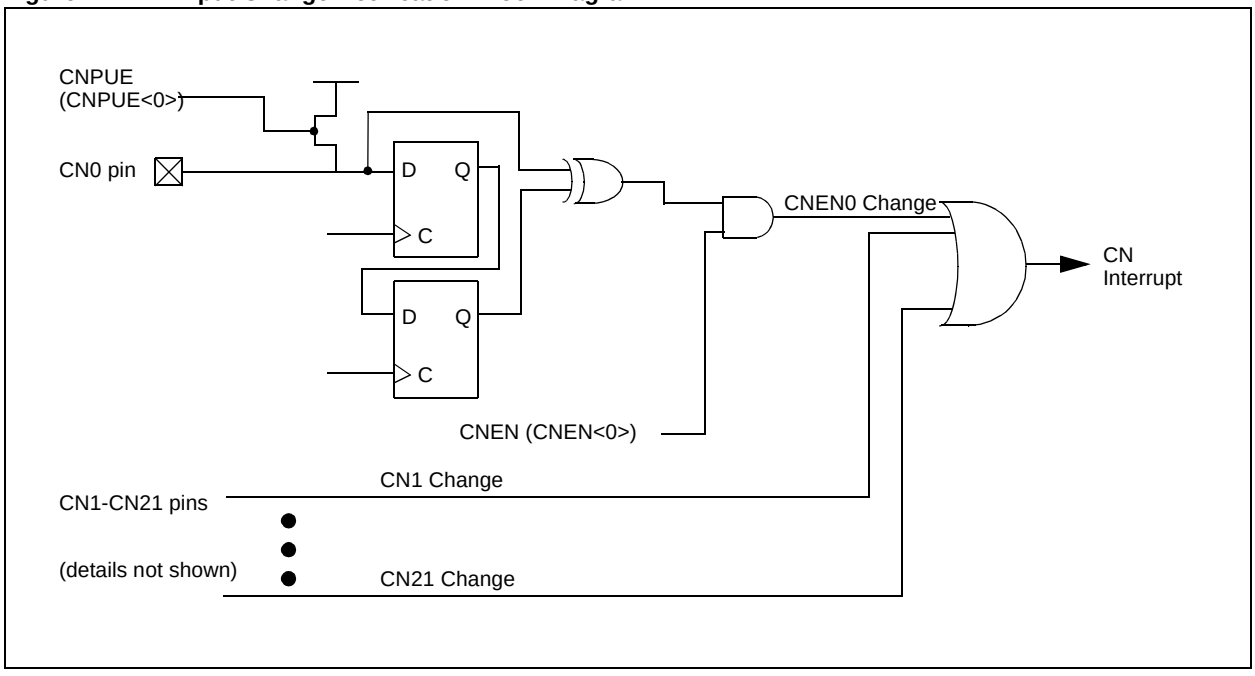
The enabled pin values are compared with the values sampled during the last read operation of the designated PORT register. If the pin value is different from the last value read, a mismatch condition is generated. The mismatch condition can occur on any of the enabled input pins. The mismatches are ORed together to provide a single interrupt-on-change signal. The enabled pins are sampled on every internal system clock cycle, SYSCLK.

Each CN pin has a pull up connected to it. The pull ups act as a current source that is connected to the pin, and eliminate the need for external resistors when push button or keypad devices are connected. The pull ups are enabled separately using the CNPUE register, which contain the control bits for each of the CN pins. Setting any of the CNPUE register bits enables the pull up for the corresponding pins.

Note:	Pull up on CN pins should always be disabled whenever the port pin is configured as a digital output.

Figure 12-4 shows the basic function of the CN hardware.

Figure 12-4: Input Change Notification Block Diagram



12.3.9.1 CN Configuration and Operation

The CN pins are configured as follows:

1. Disable CPU interrupts.
2. Set desired CN I/O pin as input by setting corresponding TRISx register bits = 1.
   Note: If the I/O pin is shared with an analog peripheral, it may be necessary to set the corresponding AD1PCFG register bit = 1 to ensure that the I/O pin is a digital input.
3. Enable CN Module ON (CNCON<15>) = 1.
4. Enable individual CN input pin(s), enable optional pull up(s).
5. Read corresponding PORT registers to clear mismatch condition on CN input pins.
6. Configure the CN interrupt priority, CNIP<2:0>, and subpriority CNIS<1:0>.
7. Clear CN interrupt flag, CNIF = 0.
8. Enable CN interrupt enable, CNIE = 1.
9. Enable CPU interrupts.

When a CN interrupt occurs, the user should read the PORT register associated with the CN pin(s). This will clear the mismatch condition and set up the CN logic to detect the next pin change. The current PORT value can be compared to the PORT read value obtained at the last CN interrupt or during initialization, and used to determine which pin changed.

The CN pins have a minimum input pulse-width specification. Refer to the **"Electrical Characteristics"** section of the data sheet for the specific device to learn more.

12.4 INTERRUPTS

12.4.1 Interrupt Configuration

The CN module has a dedicated interrupt flag bit CNIF and a corresponding interrupt enable/mask bit, CNIE. These bits are used to determine the source of an interrupt and to enable or disable an individual interrupt source.

The CNIE bit is used to define the behavior of the Interrupt Controller when a corresponding CNIF is set. When the CNIE bit is clear, the Interrupt Controller module does not generate a CPU interrupt for the event. If the CNIE bit is set, the Interrupt Controller module will generate an interrupt to the CPU when the corresponding CNIF bit is set (subject to the priority and subpriority as outlined below).

It is the responsibility of the user's software routine that services a particular interrupt to clear the appropriate interrupt flag bit before the service routine is complete.

The priority of the CN module can be set with the CNIP<2:0> bits. This priority defines the priority group to which the interrupt source will be assigned. The priority groups range from a value of 7 (the highest priority), to a value of 0 (which does not generate an interrupt). An interrupt being serviced will be preempted by an interrupt in a higher priority group.

The subpriority bits allow setting the priority of a interrupt source within a priority group. The values of the subpriority CNIS<1:0> range from 3 (the highest priority), to 0 (the lowest priority). An interrupt with the same priority group but having a higher subpriority value will preempt a lower subpriority interrupt that is in progress.

The priority group and subpriority bits allow more than one interrupt source to share the same Priority and subpriority. If simultaneous interrupts occur in this configuration, the natural order of the interrupt sources within a priority/sub-group pair determine the interrupt generated. The natural priority is based on the vector numbers of the interrupt sources. The lower the vector number the higher the natural priority of the interrupt. Any interrupts that were overridden by natural order will then generate their respective interrupts based on priority, subpriority, and natural order after the interrupt flag for the current interrupt is cleared.

After an enabled interrupt is generated, the CPU jumps to the vector assigned to that interrupt. The vector number for the interrupt is the same as the natural order number. The CPU then begins executing code at the vector address. The user's code at this vector address should perform any application specific operations and clear the CNIF interrupt flag, and then exit. Refer to **Section 8. "Interrupts"** for the vector address table details for more information on interrupts.

Table 12-2: Change Notice Interrupt Vector with EBASE = 0x8000:0000

Interrupt	Vector/ Natural Order	IRQ Number	Vector Address IntCtl.VS = 0x01	Vector Address IntCtl.VS = 0x02	Vector Address IntCtl.VS = 0x04	Vector Address IntCtl.VS = 0x08	Vector Address IntCtl.VS = 0x10
CN	23	23	8000 04E0	8000 07C0	8000 0D80	8000 1900	8000 3000

Table 12-3: Example of Priority and Subpriority Assignment

Interrupt	Priority Group	Subpriority	Vector/Natural Order
CN	7	3	23

Example 12-1: Change Notice Configuration Example

```
/*
    The following code example illustrates a Change Notice interrupt configuration for pins
    CN1(PORTC.RC13), CN4(PORTB.RB2) and CN18(PORTF.RF5).
*/

    /* NOTE: disable vector interrupts prior to configuration */

    CNCON = 0x8000;        // Enable Change Notice module
    CNEN =  0x00040012;    // Enable individual CN pins CN1, CN4 and CN18
    CNPUE = 0x00040012;    // Enable weak pull ups for pins CN1, CN4 and CN18

    /* read port(s) to clear mismatch on change notice pins */
    PORTB;
    PORTC;
    PORTF;

    IPC6SET = 0x00140000;  // Set priority level=5
    IPC6SET = 0x00030000;  // Set Subpriority level=3
                           // Could have also done this in single
                           // operation by assigning IPC6SET = 0x00170000

    IFS1CLR  = 0x0001;     // Clear the interrupt flag status bit
    IEC1SET  = 0x0001;     // Enable Change Notice interrupts

    /* re-enable vector interrupts after configuration */
```

Example 12-2: Change Notice ISR Code Example

```
/*
    The following code example demonstrates a simple interrupt service routine for CN
    interrupts. The user's code at this vector can perform any application specific
    operations. The user's code must read the CN corresponding PORT registers to clear the
    mismatch conditions before clearing the CN interrupt status flag. Finally, the CN
    interrupt status flag must be cleared before exiting.
*/
void __ISR(_CHANGE_NOTICE_VECTOR, ipl5 ChangeNoticeHandler(void)
{
    ... perform application specific operations in response to the interrupt

    readB = PORTB          // Read PORTB to clear CN4 mismatch condition
    readC = PORTC          // Read PORTC to clear CN1 mismatch condition
    readF = PORTF          // Read PORTF to clear CN18 mismatch condition
    ...
    IFS1CLR  = 0x0001;     // Be sure to clear the CN interrupt status
                           // flag before exiting the service routine.
{
```

Note: The CN ISR code example shows MPLAB® C32 C compiler specific syntax. Refer to your compiler manual regarding support for ISRs.

12.5 OPERATION IN POWER-SAVING AND DEBUG MODES

Note:	In this manual, a distinction is made between a power mode as it is used in a specific module, and a power mode as it is used by the device, e.g., Sleep mode of the Comparator and SLEEP mode of the CPU. To indicate which type of power mode is intended, uppercase and lowercase letters (Sleep, Idle, Debug) signify a module power mode, and all uppercase letters (SLEEP, IDLE, DEBUG) signify a device power mode.

12.5.1 I/O Port Operation in SLEEP Mode

As the device enters SLEEP mode, the system clock is disabled; however, the CN module continues to operate. If one of the enabled CN pins changes state, the Status bit CNIF (IFS1<0>) will be set. If the CNIE bit (IEC1<0>) is set, and its priority is greater than current CPU priority, the device will wake from SLEEP or IDLE mode and execute the CN Interrupt Service Routine.

If the assigned priority level of the CN interrupt is less than or equal to the current CPU priority level, the CPU will not be awakened and the device will enter IDLE mode.

12.5.2 I/O Port Operation in IDLE Mode

As the device enters IDLE mode, the system clock sources remain functional. The SIDL bit (CNCON<13>) selects whether the module will stop or continue functioning on IDLE.

- If SIDL = 1, the module will continue to sample Input CN I/O pins in IDLE mode, however, synchronization is disabled.
- If SIDL = 0, the module will continue to synchronize and sample Input CN I/O pins in IDLE mode.

12.5.3 I/O Port Operation in DEBUG Mode

The FRZ bit (CNCON<14>) determines whether the CN module will run or stop while the CPU is executing DEBUG exception code (i.e., application is halted) in DEBUG mode.

- If FRZ = 0, the module continues to operate even when application is halted in DEBUG mode.
- If FRZ = 1 and application is halted in DEBUG mode, the module will freeze its operations and make no changes to the state of the CN module. The module will resume its operation after CPU resumes execution.

Note:	The FRZ bit is readable and writable only when the CPU is executing in Debug Exception mode. In all other modes, the FRZ bit reads as '0'. If FRZ bit is changed during DEBUG mode, the new value does not take effect until the current Debug Exception mode is exited and re-entered. During the Debug Exception mode, the FRZ bit reads the state of the peripheral when entering DEBUG mode.

12.6 EFFECTS OF VARIOUS RESETS

12.6.1 Device Reset

All TRIS, LAT, PORT, ODC, CNEN, CNPUE and CNCON registers are forced to their Reset states upon a device Reset.

12.6.2 Power-On Reset

All TRIS, LAT, PORT, ODC, CNEN, CNPUE and CNCON registers are forced to their Reset states upon a Power-on Reset.

12.6.3 Watchdog Reset

All TRIS, LAT, PORT, ODC, CNEN, CNPUE and CNCON registers are unchanged upon a Watchdog Reset.

12.7 I/O Port Application

Example 12-3: Code Example

```
/*
 The following code example illustrates configuring
 RB0, RB1 as analog (default) inputs, RB2 as a digital
 input, RB3 as digital output and RB4 as digital output
 with open-drain enabled using SET, CLR atomic SFR registers.*/

 AD1PCFGCLR = 0x0003;      // RB0, RB1 = analog pins
 TRISBSET = 0x0003;        // RB0, RB1 = inputs

 AD1PCFGSET = 0x000C;      // RB2, RB3 = digital pins
 TRISBSET =  0x0004;       // RB2 = input
 TRISBCLR =  0x0018;       // RB3, RB4 = outputs

 ODCBSET = 0x0010;         // RB4 open-drain enabled

/*
 The following code example illustrates same configuration
 above using Base SFR registers directly.*/

 AD1PCFG = 0x001C;         // RB0, RB1 = analog pins; RB2, RB3, RB4 = digital pins
 TRISB =  0x0007;          // RB0, RB1, RB2 = inputs; RB3, RB4 = outputs

 ODCB = 0x0010;            // RB4 open-drain enabled
```

12.8 I/O PIN CONTROL

Table provides a summary of I/O pin mode settings.

Table 12-4: I/O Pin Configurations

Required Settings for Digital Pin Control							
Mode or Pin Usage	**Pin Type**	**Buffer Type**	**TRIS Bit**	**ODC Bit**	**CNEN Bit**	**CNPUE Bit(1)**	**AD1PCFG Bit**
Input	IN	ST	1	—	—	—	1
CN	IN	ST	1	—	1	1	1
Output	OUT	CMOS	0	0	—	—	1
Open Drain	OUT	OPEN	0	1	—	—	1

Required Settings for Analog Pin Control							
Mode or Pin Usage	**Pin Type**	**Buffer Type**	**TRIS Bit**	**ODC Bit**	**CNEN Bit**	**CNPUE Bit(1)**	**AD1PCFG Bit**
ANx Input	IN	A	1	—	—	—	0
CV Output	OUT	A	—	—	—	—	0

Required Settings for JTAG Pin Control(2)							
Mode or Pin Usage	**Pin Type**	**Buffer Type**	**TRIS Bit**	**ODC Bit**	**CNEN Bit**	**CNPUE Bit(1)**	**AD1PCFG Bit**
TCK	IN	ST	—	—	—	—	—
TDI	IN	ST	—	—	—	—	—
TMS	IN	ST	—	—	—	—	—
TDO	OUT	CMOS	—	—	—	—	—

Required Settings for ICSP Pin Control(3)							
Mode or Pin Usage	**Pin Type**	**Buffer Type**	**TRIS Bit**	**ODC Bit**	**CNEN Bit**	**CNPUE Bit(1)**	**AD1PCFG Bit**
PGC	IN	ST	—	—	—	—	—
	OUT	CMOS	—	—	—	—	—
PGD	IN	ST	—	—	—	—	—
	OUT	CMOS	—	—	—	—	—

Legend: CMOS = CMOS compatible input or output
ST = Schmitt Trigger input with CMOS levels
I = Input
O = Output

Note 1: The CN Enable Pull-up bit is optional.

2: The pin control for the JTAG module is automatically set when JTAG is enabled and the corresponding DEBUGGING mode is selected. No user configuration is required.

3: The pin control for the ICSP™ module is set automatically when entering ICSP mode. No user configuration is required.

12.9 DESIGN TIPS

Question 1: ***How should I configure my unused I/O pins?***

Answer: I/O pins that are not used can be set as outputs (corresponding TRIS bit = 0) and driven low (corresponding LAT bit = 0) in software.

Question 2: ***Is it possible to connect PIC32MX I/O pins to a 5V device?***

Answer: Yes, with limitations. PIC32MX I/O pins are 5V tolerant when configured as an input, which means the pin can tolerate an input up to 5V. When configured as an output, an I/O pin can only drive as high as the voltage supplied to the PIC32MX V_{DD} pin, which is limited to 3.6V. Depending on the 5V device's input pin design, this may not be sufficient to be correctly read as a logic "high" signal. For a detailed discussion on interfacing different logic level families, refer to the "*Microchip 3V Tips 'n Tricks*" (DS41285) guide.

12.10 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the PIC32MX device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the I/O Ports are:

Title	Application Note #
Implementing Wake-up on Key Stroke	AN552

Note: Please visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the PIC32MX family of devices.

12.11 REVISION HISTORY

Revision A (August 2007)

This is the initial released version of this document.

Revision B (October 2007)

Updated document to remove Confidential status.

Revision C (April 2008)

Revised status to Preliminary; Revised U-0 to r-x; Revised Register 12-13; Revised Figure 12-1 and 12-2.

Revision D (May 2008)

Revised Register 12-17, add note to FRZ; Add note to Registers 12-19, 12-30, 12-31; Revised Example 12-1 and 12-2; Change Reserved bits from “Maintain as” to “Write”; Added Note to ON bit (CNCON Register).

NOTES:

Section 13. Parallel Master Port

HIGHLIGHTS

This section of the manual contains the following topics:

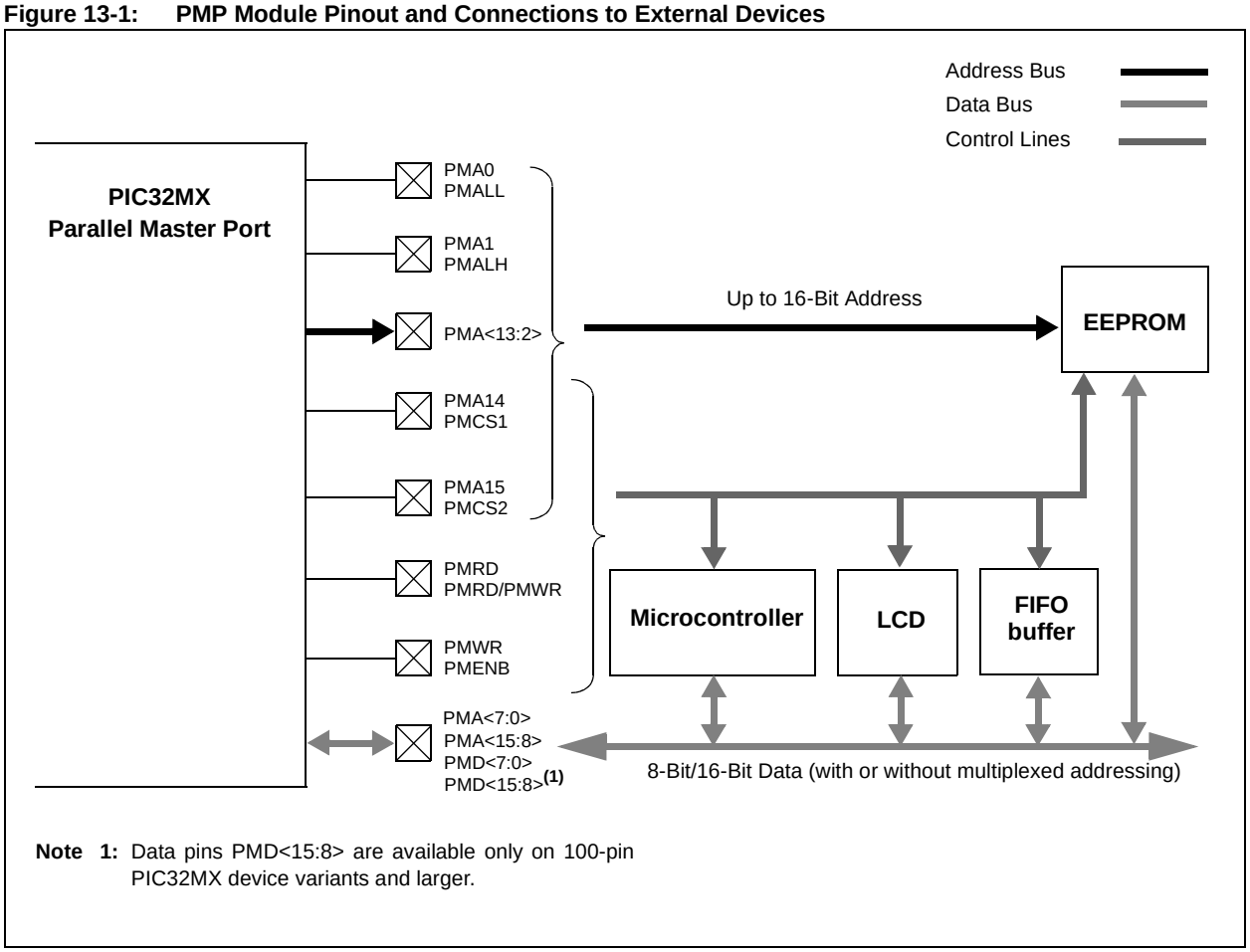
13.1 INTRODUCTION

The Parallel Master Port (PMP) is a parallel 8-bit/16-bit I/O module specifically designed to communicate with a wide variety of parallel devices such as communications peripherals, LCDs, external memory devices, and microcontrollers. Because the interfaces to parallel peripherals vary significantly, the PMP module is highly configurable.

Key features of the PMP module include:

- Up to 16 programmable address lines
- Up to two Chip Select lines
- Programmable strobe options
  - individual read and write strobes, or
  - read/write strobe with enable strobe
- Address auto-increment/auto-decrement
- Programmable address/data multiplexing
- Programmable polarity on control signals
- Legacy parallel slave port support
- Enhanced parallel slave support
  - address support
  - 4-bytes-deep, auto-incrementing buffer
- Programmable Wait states
- Freeze option for in-circuit debugging

Figure 13-1: PMP Module Pinout and Connections to External Devices



Note 1: Data pins PMD<15:8> are available only on 100-pin PIC32MX device variants and larger.

13.2 CONTROL REGISTERS

The PMP module uses these Special Function Registers (SFRs):

- PMCON: Parallel Master Port Control Register

  PMCONCLR, PMCONSET, PMCONINV: Atomic Bit Manipulation, Write-only Registers for PMCON

- PMMODE: Parallel Master Port Mode Control Register

  PMMODECLR, PMMODESET, PMMODEINV: Atomic Bit Manipulation, Write-only Registers for PMMODE

- PMADDR: Parallel Master Port Address Register

  PMADDRCLR, PMADDRSET, PMADDRINV: Atomic Bit Manipulation, Write-only Registers for PMDOUT

- PMDOUT: Parallel Master Port Data Output Register

  PMDOUTCLR, PMDOUTSET, PMDOUTINV: Atomic Bit Manipulation, Write-only Registers for PMDOUT

- PMDIN: Parallel Master Port Data Input Register

  PMDINCLR, PMDINSET, PMDININV: Atomic Bit Manipulation, Write-only Registers for PMDIN

- PMAEN: Parallel Master Port Address Enable Register

  PMAENCLR, PMAENSET, PMAENINV: Atomic Bit Manipulation, Write-only Registers for PMAEN

- PMSTAT: Parallel Master Port Status Register

  PMSTATCLR, PMSTATSET, PMSTATINV: Atomic Bit Manipulation, Write-only Registers for PMSTAT

The PMP module also has the following associated bits for interrupt control:

- Interrupt Enable Control bit (PMPIE) in IEC1: Interrupt Enable Control Register 1
- Interrupt Flag Status bit (PMPIF) in IFS1: Interrupt Flag Status Register 0
- Interrupt Priority Control bits (PMPIP<2:0>) and (PMPIS<1:0>) in IPC7: Interrupt Priority Control Register 7

13.2.1 PMCON Register

PMCON (Register 13-1) contains the bits that control much of the module's basic functionality. A key bit is ON, which is used to reset the module enable or disable the module.

When the module is disabled, all the associated I/O pins revert to their designated I/O function. In addition, any read or write operations active or pending are stopped, and the BUSY bit is cleared. The data within the module registers is retained, including the data in PMSTAT. Thus, the module could be disabled after a reception, and the last received data and status would still be available for processing.

When the module is enabled, all buffer control logic is reset, along with PMSTAT.

All other bits in PMCON control address multiplexing, enable various port control signals, and select control signal polarity. These are discussed in more detail in **Section 13.3.1 "Parallel Master Port Configuration Options"**.

13.2.2 PMMODE Register

PMMODE (Register 13-5) contains bits that control the operational modes of the module. Master/Slave mode selection, as well as configuration options for both modes, are set by this register. It also contains the universal status flag BUSY, used in master modes to indicate that an operation by the module in progress.

Details on the use of the PMMODE bits to configure PMP operation are provided in **Section 13.4 "Slave Modes of Operation"** and **Section 13.3 "Master Modes of Operation"**.

13.2.3 PMADDR Register

PMADDR (Register 13-9) functions as PMADDR in master modes. It contains the address to which outgoing data is to be written, as well as the Chip Select control bits for addressing parallel slave devices. The PMADDR register is not used in any of the Slave modes.

13.2.4 PMDOUT Register

PMDOUT is only used in Slave mode for buffered output data.

13.2.5 PMDIN Register

PMDIN is used by the module in both Master and Slave modes. In Slave mode, this register is used to hold data that is asynchronously clocked in. Its operation is described in **Section 13.4.2 "Buffered Parallel Slave Port Mode"**.

In Master mode, PMDIN is the holding register for both incoming and outgoing data. Its operation in Master mode is described in **Section 13.3.3 "Read Operation"** and **Section 13.3.4 "Write Operation"**.

13.2.6 PMAEN Register

Parallel Master Port Address Enable register (Register 13-21) controls the operation of address and Chip Select pins associated to this module. Setting these bits allocates the corresponding microcontroller pins to the PMP module; clearing the bits allocates the pins to port I/O or other peripheral modules associated with the pin.

13.2.7 PMSTAT Register

The Parallel Port Status register (Register 13-25) contains Status bits associated with buffered operating modes when the port is functioning as a Slave port. This includes overflow, underflow, and full flag bit. These flags are discussed in detail in **Section 13.4.2 "Buffered Parallel Slave Port Mode"**.

13.2.8 PMP SFR Summary

The following table provides a brief summary of all PMP-module-related registers. Corresponding registers appear after the summary, followed by a detailed description of each register.

Table 13-1: PMP SFR Summary

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
PMCON	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	ON	FRZ	SIDL	ADRMUX<1:0>		PMPTTL	PTWREN	PTRDEN
	7:0	CSF<1:0>		ALP	CS2P	CS1P	—	WRSP	RDSP
PMCONCLR	31:0	Write clears selected bits in PMCON, read yields undefined value							
PMCONSET	31:0	Write sets selected bits in PMCON, read yields undefined value							
PMCONINV	31:0	Write inverts selected bits in PMCON, read yields undefined value							
PMMODE	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	BUSY	IRQM<1:0>		INCM<1:0>		MODE16	MODE<1:0>	
	7:0	WAITB<1:0>		WAITM<3:0>				WAITE<1:0>	
PMMODECLR	31:0	Write clears selected bits in PMMODE, read yields undefined value							
PMMODESET	31:0	Write sets selected bits in PMMODE, read yields undefined value							
PMMODEINV	31:0	Write inverts selected bits in PMMODE, read yields undefined value							
PMADDR	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	CS2/A15	CS1/A14	ADDR<13:8>					
	7:0	ADDR<7:0>							
PMADDRCLR	31:0	Write clears selected bits in PMADDR, read yields undefined value							
PMADDRSET	31:0	Write sets selected bits in PMADDR, read yields undefined value							
PMADDRINV	31:0	Write inverts selected bits in PMADDR, read yields undefined value							
PMDOUT	31:24	DATAOUT<31:24>							
	23:16	DATAOUT<23:16>							
	15:8	DATAOUT<15:8>							
	7:0	DATAOUT<7:0>							
PMDOUTCLR	31:0	Write clears selected bits in PMDOUT, read yields undefined value							
PMDOUTSET	31:0	Write sets selected bits in PMDOUT, read yields undefined value							
PMDOUTINV	31:0	Write inverts selected bits in PMDOUT, read yields undefined value							
PMDIN	31:24	DATAIN<31:24>							
	23:16	DATAIN<23:16>							
	15:8	DATAIN<15:8>							
	7:0	DATAIN<7:0>							
PMDINCLR	31:0	Write clears selected bits in PMDIN, read yields undefined value							
PMDINSET	31:0	Write sets selected bits in PMDIN, read yields undefined value							
PMDININV	31:0	Write inverts selected bits in PMDIN, read yields undefined value							
PMAEN	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	PTEN<15:8>							
	7:0	PTEN<7:0>							
PMAENCLR	31:0	Write clears selected bits in PMAEN, read yields undefined							
PMAENSET	31:0	Write sets selected bits in PMAEN, read yields undefined							
PMAENINV	31:0	Write inverts selected bits in PMAEN, read yields undefined							

Table 13-1: PMP SFR Summary (Continued)

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
PMSTAT	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	IBF	IBOV	—	—	IB3F	IB2F	IB1F	IB0F
	7:0	OBE	OBUF	—	—	OB3E	OB2E	OB1E	OB0E
PMSTATCLR	31:0	Write clears selected bits in PMSTAT, read yields undefined value							
PMSTATSET	31:0	Write sets selected bits in PMSTAT, read yields undefined value							
PMSTATINV	31:0	Write inverts selected bits in PMSTAT, read yields undefined value							
IEC1	31:24	—	—	—	—	—	—	USBIE	FCEIE
	23:16	—	—	—	—	DMA3IE	DMA2IE	DMA1IE	DMA0IE
	15:8	RTCCIE	FSCMIE	I2C2MIE	I2C2SIE	I2C2BIE	U2TXIE	U2RXIE	U2EIE
	7:0	SPI2RXIE	SPI2TXIE	SPI2EIE	CMP2IE	CMP1IE	PMPIE	AD1IE	CNIE
IFS1	31:24	—	—	—	—	—	—	USBIF	FCEIF
	23:16	—	—	—	—	DMA3IF	DMA2IF	DMA1IF	DMA0IF
	15:8	RTCCIF	FSCMIF	I2C2MIF	I2C2SIF	I2C2BIF	U2TXIF	U2RXIF	U2EIF
	7:0	SPI2RXIF	SPI2TXIF	SPI2EIF	CMP2IF	CMP1IF	PMPIF	AD1IF	CNIF
IPC7	31:24	—	—	—	SPI2IP<2:0>			SPI2IS<1:0>	
	23:16	—	—	—	CMP2IP<2:0>			CMP2IS<1:0>	
	15:8	—	—	—	CMP1IP<2:0>			CMP1IS<1:0>	
	7:0	—	—	—	PMPIP<2:0>			PMPIS<1:0>	

Register 13-1: PMCON: Parallel Port Control Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ON	FRZ	SIDL	ADRMUX1	ADRMUX0	PMPTTL	PTWREN	PTRDEN
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	r-x	R/W-0	R/W-0
CSF1	CSF0	ALP	CS2P	CS1P	—	WRSP	RDSP
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15 **ON:** Parallel Master Port Enable bit
1 = PMP enabled
0 = PMP disabled, no off-chip access performed

Note: When using 1:1 PBCLK divisor, the user’s software should not read/write the peripheral’s SFRs in the SYSCLK cycle immediately following the instruction that clears the module’s ON bit.

bit 14 **FRZ:** Freeze in Debug Exception Mode bit
1 = Freeze operation when CPU is in Debug Exception mode
0 = Continue operation even when CPU is in Debug Exception mode

Note: FRZ is writable in Debug Exception mode only, it is forced to ‘0’ in normal mode.

bit 13 **SIDL:** Stop in IDLE Mode bit
1 = Discontinue module operation when device enters IDLE mode
0 = Continue module operation in IDLE mode

bit 12-11 **ADRMUX<1:0>:** Address/Data Multiplexing Selection bits
11 = All 16 bits of address are multiplexed on PMD<15:0> pins
10 = All 16 bits of address are multiplexed on PMD<7:0> pins
01 = Lower 8 bits of address are multiplexed on PMD<7:0> pins, upper 8 bits are on PMA<15:8>
00 = Address and data appear on separate pins

bit 10 **PMPTTL**: PMP Module TTL Input Buffer Select bit
1 = PMP module uses TTL input buffers
0 = PMP module uses Schmidt Trigger input buffer

bit 9 **PTWREN:** Write Enable Strobe Port Enable bit
1 = PMWR/PMENB port enabled
0 = PMWR/PMENB port disabled

bit 8 **PTRDEN:** Read/Write Strobe Port Enable bit
1 = PMRD/PMWR port enabled
0 = PMRD/PMWR port disabled

Note 1: These bits have no effect when their corresponding pins are used as address lines.

Register 13-1: PMCON: Parallel Port Control Register (Continued)

bit 7-6 **CSF<1:0>:** Chip Select Function bits(1)

11 = Reserved
10 = PMCS2 and PMCS1 function as Chip Select
01 = PMCS2 functions as Chip Select, PMCS1 functions as address bit 14
00 = PMCS2 and PMCS1 function as address bits 15 and 14

bit 5 **ALP:** Address Latch Polarity bit(1)

1 = Active-high (PMALL and PMALH)
0 = Active-low ($\overline{\text{PMALL}}$ and $\overline{\text{PMALH}}$)

bit 4 **CS2P:** Chip Select 1 Polarity bit(1)

1 = Active-high (PMCS2)
0 = Active-low ($\overline{\text{PMCS2}}$)

bit 3 **CS1P:** Chip Select 0 Polarity bit(1)

1 = Active-high (PMCS1)
0 = Active-low ($\overline{\text{PMCS1}}$)

bit 2 **Reserved:** Write '0'; ignore read

bit 1 **WRSP:** Write Strobe Polarity bit

For Slave Modes and Master mode 2 (PMMODE<9:8> = 00, 01, 10):

1 = Write strobe active-high (PMWR)
0 = Write strobe active-low ($\overline{\text{PMWR}}$)

For Master mode 1 (PMMODE<9:8> = 11):

1 = Enable strobe active-high (PMENB)
0 = Enable strobe active-low ($\overline{\text{PMENB}}$)

bit 0 **RDSP:** Read Strobe Polarity bit

For Slave modes and Master mode 2 (PMMODE<9:8> = 00, 01, 10):

1 = Read Strobe active-high (PMRD)
0 = Read Strobe active-low ($\overline{\text{PMRD}}$)

For Master mode 1 (PMMODE<9:8> = 11):

1 = Read/write strobe active-high (PMRD/$\overline{\text{PMWR}}$)
0 = Read/write strobe active-low ($\overline{\text{PMRD}}$/PMWR)

Note 1: These bits have no effect when their corresponding pins are used as address lines.

Register 13-2: PMCONCLR: PMP Control Clear Register

Write clears selected bits in PMCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in PMCON**

A write of '1' in one or more bit positions clears the corresponding bit(s) in PMCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: PMCONCLR = 0x00008001 will clear bits 15 and 0 in PMCON register.

Register 13-3: PMCONSET: PMP Control Set Register

Write sets selected bits in PMCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in PMCON**

A write of '1' in one or more bit positions sets the corresponding bit(s) in PMCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: PMCONSET = 0x00008001 will set bits 15 and 0 in PMCON register.

Register 13-4: PMCONINV: PMP Control Invert Register

Write inverts selected bits in PMCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in PMCON**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in PMCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: PMCONINV = 0x00008001 will invert bits 15 and 0 in PMCON register.

Register 13-5: PMMODE: Parallel Port Mode Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
BUSY	IRQM<1:0>		INCM<1:0>		MODE16	MODE<1:0>	
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WAITB<1:0>		WAITM<3:0>				WAITE<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15 **BUSY:** Busy bit (Master mode only)
- 1 = Port is busy
- 0 = Port is not busy

bit 14-13 **IRQM<1:0>:** Interrupt Request Mode bits
- 11 = Reserved, do not use
- 10 = Interrupt generated when Read Buffer 3 is read or Write Buffer 3 is written (Buffered PSP mode) or on a read or write operation when PMA<1:0> =11 (Addressable Slave mode only)
- 01 = Interrupt generated at the end of the read/write cycle
- 00 = No Interrupt generated

bit 12-11 **INCM<1:0>:** Increment Mode bits
- 11 = Slave mode read and write buffers auto-increment (PMMODE<1:0> = 00 only)
- 10 = Decrement ADDR<15:0> by 1 every read/write cycle **(2) (4)**
- 01 = Increment ADDR<15:0> by 1 every read/write cycle **(2) (4)**
- 00 = No increment or decrement of address

bit 10 **MODE16:** 8/16-bit Mode bit
- 1 = 16-bit mode: a read or write to the data register invokes a single 16-bit transfer
- 0 = 8-bit mode: a read or write to the data register invokes a single 8-bit transfer

bit 9-8 **MODE<1:0>:** Parallel Port Mode Select bits
- 11 = Master mode 1 (PMCSx, PMRD/PMWR, PMENB, PMA<x:0>, PMD<7:0> and PMD<8:15>**(3)**)
- 10 = Master mode 2 (PMCSx, PMRD, PMWR, PMA<x:0>, PMD<7:0> and PMD<8:15>**(3)**)
- 01 = Enhanced Slave mode, control signals (PMRD, PMWR, PMCS, PMD<7:0>, and PMA<1:0>)
- 00 = Legacy Parallel Slave Port, control signals (PMRD, PMWR, PMCS, and PMD<7:0>)

Note 1: Whenever WAITM<3:0> = 0000, WAITB and WAITE bits are ignored and forced to 1 T_{PBCLK} cycle for a write operation; WAITB = 1 T_{PBCLK} cycle, WAITE = 0 T_{PBCLK} cycles for a read operation.

2: Address bit A15 and A14 are not subject to auto-increment/decrement if configured as Chip Select CS2 and CS1.

3: These pins are active when bit MODE16 = 1 (16-bit mode)

4: The PMPADDR register is always incremented/decremented by 1 regardless of the transfer data width.

Register 13-5: PMMODE: Parallel Port Mode Register (Continued)

bit 7-6 **WAITB1:WAITB0:** Data Setup to Read/Write Strobe Wait States bits(1)

11 =Data wait of 4 TPB; multiplexed address phase of 4 TPB
10 =Data wait of 3 TPB; multiplexed address phase of 3 TPB
01 =Data wait of 2 TPB; multiplexed address phase of 2 TPB
00 = Data wait of 1 TPB; multiplexed address phase of 1 TPB (DEFAULT)

bit 5-2 **WAITM3:WAITM0:** Data Read/Write Strobe Wait States bits

1111 =Wait of 16 TPB
...
0001 =Wait of 2 TPB
0000 = Wait of 1 TPB (DEFAULT)

bit 1-0 **WAITE1:WAITE0:** Data Hold After Read/Write Strobe Wait States bits(1)

11 =Wait of 4 TPB
10 =Wait of 3 TPB
01 =Wait of 2 TPB
00 =Wait of 1 TPB (DEFAULT)

for Read operations:
11 =Wait of 3TPB
10 =Wait of 2TPB
01 =Wait of 1TPB
00 = Wait of 0TPB (DEFAULT)

Note 1: Whenever WAITM<3:0> = 0000, WAITB and WAITE bits are ignored and forced to 1 T_{PBCLK} cycle for a write operation; WAITB = 1 T_{PBCLK} cycle, WAITE = 0 T_{PBCLK} cycles for a read operation.

2: Address bit A15 and A14 are not subject to auto-increment/decrement if configured as Chip Select CS2 and CS1.

3: These pins are active when bit MODE16 = 1 (16-bit mode)

4: The PMPADDR register is always incremented/decremented by 1 regardless of the transfer data width.

Register 13-6: PMMODECLR: PMMODE Clear Register

Write clears selected bits in PMMODE, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in PMMODE**

A write of '1' in one or more bit positions clears the corresponding bit(s) in PMMODE register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `PMMODECLR = 0x00008001` will clear bits 15 and 0 in PMMODE register.

Register 13-7: PMMODESET: PMMODE Set Register

Write sets selected bits in PMMODE, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in PMMODE**

A write of '1' in one or more bit positions sets the corresponding bit(s) in PMMODE register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `PMMODESET = 0x00008001` will set bits 15 and 0 in PMMODE register.

Register 13-8: PMMODEINV: PMMODE Invert Register

Write inverts selected bits in PMMODE, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in PMMODE**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in PMMODE register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `PMMODEINV = 0x00008001` will invert bits 15 and 0 in PMMODE register.

Register 13-9: PMADDR: Parallel Port Address Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CS2	CS1	ADDR<13:8>					
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADDR<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15 **CS2:** Chip Select 2 bit
1 = Chip Select 2 is active
0 = Chip Select 2 is inactive (pin functions as PMA<15>)

bit 14 **CS1:** Chip Select 1 bit
1 = Chip Select 1 is active
0 = Chip Select 1 is inactive (pin functions as PMA<14>)

bit 13-0 **ADDR<13:0>:** Destination Address bits

Register 13-10: PMADDRCLR: PMADDR Clear Register

Write clears selected bits in PMADDR, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in PMADDR**

A write of '1' in one or more bit positions clears the corresponding bit(s) in PMADDR register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: PMADDRCLR = 0x00008001 will clear bits 15 and 0 in PMADDR register.

Register 13-11: PMADDRSET: PMADDR Set Register

Write sets selected bits in PMADDR, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in PMADDR**

A write of '1' in one or more bit positions sets the corresponding bit(s) in PMADDR register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: PMADDRSET = 0x00008001 will set bits 15 and 0 in PMADDR register.

Register 13-12: PMADDRINV: PMADDR Invert Register

Write inverts selected bits in PMADDR, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in PMADDR**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in PMADDR register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: PMADDRINV = 0x00008001 will invert bits 15 and 0 in PMADDR register.

Register 13-13: PMDOUT: Parallel Port Data Output Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
DATAOUT<31:24>							
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
DATAOUT<23:16>							
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
DATAOUT<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
DATAOUT<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **DATAOUT<31:0>:** Output Data Port bits for 8-bit write operations in Slave mode

Register 13-14: PMDOUTCLR: PMDOUT Clear Register

Write clears selected bits in PMDOUT, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in PMDOUT**

A write of '1' in one or more bit positions clears the corresponding bit(s) in PMDOUT register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: PMDOUTCLR = 0x00008001 will clear bits 15 and 0 in PMDOUT register.

Register 13-15: PMDOUTSET: PMDOUT Set Register

Write sets selected bits in PMDOUT, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in PMDOUT**

A write of '1' in one or more bit positions sets the corresponding bit(s) in PMDOUT register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: PMDOUT = 0x00008001 will set bits 15 and 0 in PMDOUT register.

Register 13-16: PMDOUTINV: PMDOUT Invert Register

Write inverts selected bits in PMDOUT, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in PMDOUT**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in PMDOUT register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: PMDOUTINV = 0x00008001 will invert bits 15 and 0 in PMDOUT register.

Register 13-17: PMDIN: Parallel Port Data Input Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
DATAIN<31:24>							
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
DATAIN<23:16>							
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
DATAIN<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
DATAIN<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **DATAIN<31:0>:** Input and Output Data Port bits for 8-bit or 16-bit read/write operations in Master mode

Input Data Port for 8-bit read operations in Slave mode.

Register 13-18: PMDINCLR: PMDIN Clear Register

Write clears selected bits in PMDIN, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in PMDIN**

A write of '1' in one or more bit positions clears the corresponding bit(s) in PMDIN register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: PMDINCLR = 0x00008001 will clear bits 15 and 0 in PMDIN register.

Register 13-19: PMDINSET: PMDIN Set Register

Write sets selected bits in PMDIN, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in PMDIN**

A write of '1' in one or more bit positions sets the corresponding bit(s) in PMDIN register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: PMDINSET = 0x00008001 will set bits 15 and 0 in PMDIN register.

Register 13-20: PMDININV: PMDIN Invert Register

Write inverts selected bits in PMDIN, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in PMDIN**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in PMDIN register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: PMDININV = 0x00008001 will invert bits 15 and 0 in PMDIN register.

Register 13-21: PMAEN: Parallel Port Pin Enable Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PTEN<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PTEN<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15-14 **PTEN<15:14>:** PMCSx Strobe Enable bits
1 = PMA15 and PMA14 function as either PMA<15:14> or PMCS2 and PMCS1(1)
0 = PMA15 and PMA14 function as port I/O

bit 13-2 **PTEN<13:2>:** PMP Address Port Enable bits
1 = PMA<13:2> function as PMP address lines
0 = PMA<13:2> function as port I/O

bit 1-0 **PTEN<1:0>:** PMALH/PMALL Strobe Enable bits
1 = PMA1 and PMA0 function as either PMA<1:0> or PMALH and PMALL(2)
0 = PMA1 and PMA0 pads functions as port I/O

Note 1: The use of these pins as PMA15/PMA14 or CS2/CS1 are selected by bits CSF<1:0> in the PMCON register.

2: The use of these pins as PMA1/PMA0 or PMALH/PMALL depend on the Address/Data Multiplex mode selected by bits ADRMUX<1:0> in the PMCON register.

Register 13-22: PMAENCLR: PMAEN Clear Register

Write clears selected bits in PMAEN, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in PMAEN**

A write of '1' in one or more bit positions clears the corresponding bit(s) in PMAEN register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: PMAENCLR = 0x00008001 will clear bits 15 and 0 in PMAEN register.

Register 13-23: PMAENSET: PMAEN Set Register

Write sets selected bits in PMAEN, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in PMAEN**

A write of '1' in one or more bit positions sets the corresponding bit(s) in PMAEN register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: PMAENSET = 0x00008001 will set bits 15 and 0 in PMAEN register.

Register 13-24: PMAENINV: PMAEN Invert Register

Write inverts selected bits in PMAEN, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in PMAEN**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in PMAEN register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: PMAENINV = 0x00008001 will invert bits 15 and 0 in PMAEN register.

Register 13-25: PMSTAT: Parallel Port Status Register (Slave modes only)

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R-0	R/W-0	r-x	r-x	R-0	R-0	R-0	R-0
IBF	IBOV	—	—	IB3F	IB2F	IB1F	IB0F
bit 15							bit 8

R-1	R/W-0	r-x	r-x	R-1	R-1	R-1	R-1
OBE	OBUF	—	—	OB3E	OB2E	OB1E	OB0E
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15 **IBF:** Input Buffer Full Status bit
1 = All writable input buffer registers are full
0 = Some or all of the writable input buffer registers are empty

bit 14 **IBOV:** Input Buffer Overflow Status bit
1 = A write attempt to a full input byte buffer occurred (must be cleared in software)
0 = No overflow occurred
This bit is set (= 1) in hardware; can only be cleared (= 0) in software.

bit 13-12 **Reserved:** Write ‘0’; ignore read

bit 11-8 **IBnF:** Input Buffer n Status Full bits
1 = Input Buffer contains data that has not been read (reading buffer will clear this bit)
0 = Input Buffer does not contain any unread data

bit 7 **OBE:** Output Buffer Empty Status bit
1 = All readable output buffer registers are empty
0 = Some or all of the readable output buffer registers are full

bit 6 **OBUF:** Output Buffer Underflow Status bit
1 = A read occurred from an empty output byte buffer (must be cleared in software)
0 = No underflow occurred
This bit is set (= 1) in hardware; can only be cleared (= 0) in software.

bit 5-4 **Reserved:** Write ‘0’; ignore read

bit 3-0 **OBnE:** Output Buffer n Status Empty bits
1 = Output buffer is empty (writing data to the buffer will clear this bit)
0 = Output buffer contains data that has not been transmitted

Register 13-26: PMSTATCLR: PMSTAT Clear Register

Write clears selected bits in PMSTAT, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in PMSTAT**

A write of '1' in one or more bit positions clears the corresponding bit(s) in PMSTAT register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `PMSTATCLR = 0x00008001` will clear bits 15 and 0 in PMSTAT register.

Register 13-27: PMSTATSET: PMSTAT Set Register

Write sets selected bits in PMSTAT, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in PMSTAT**

A write of '1' in one or more bit positions sets the corresponding bit(s) in PMSTAT register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `PMSTATSET = 0x00008001` will set bits 15 and 0 in PMSTAT register.

Register 13-28: PMSTATINV: PMSTAT Invert Register

Write inverts selected bits in PMSTAT, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in PMSTAT**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in PMSTAT register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `PMSTATINV = 0x00008001` will invert bits 15 and 0 in PMSTAT register.

Register 13-29: IEC1: Interrupt Enable Control Register 1(1)

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	USBIE	FCEIE
bit 31							bit 24

r-x	r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	DMA3IE	DMA2IE	DMA1IE	DMA0IE
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RTCCIE	FSCMIE	I2C2MIE	I2C2SIE	I2C2BIE	U2TXIE	U2RXIE	U2EIE
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SPI2RXIE	SPI2TXIE	SPI2EIE	CMP2IE	CMP1IE	PMPIE	AD1IE	CNIE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 2 **PMPIE:** PMP Interrupt Enable bit

1 = Interrupt is enabled
0 = Interrupt is disabled

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the PMP.

Register 13-30: IFS1: Interrupt Flag Status Control Register 1(1)

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	USBIF	FCEIF
bit 31							bit 24

r-x	r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	DMA3IF	DMA2IF	DMA1IF	DMA0IF
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RTCCIF	FSCMIF	I2C2MIF	I2C2SIF	I2C2BIF	U2TXIF	U2RXIF	U2EIF
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SPI2RXIF	SPI2TXIF	SPI2EIF	CMP2IF	CMP1IF	PMPIF	AD1IF	CNIF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 2 **PMPIF:** PMP Interrupt Flag bits

1 = Interrupt flag is enabled
0 = Interrupt flag is disabled

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the PMP.

Register 13-31: IPC7: Interrupt Priority Control Register 7(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	SPI2IP<2:0>			SPI2IS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	CMP2IP<2:0>			CMP2IS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	CMP1IP<2:0>			CMP1IS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	PMPIP<2:0>			PMPIS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 4-2 **PMPIP<2:0>:** PMP Interrupt Priority bits
- 111 = Interrupt Priority is 7
- 110 = Interrupt Priority is 6
- 101 = Interrupt Priority is 5
- 100 = Interrupt Priority is 4
- 011 = Interrupt Priority is 3
- 010 = Interrupt Priority is 2
- 001 = Interrupt Priority is 1
- 000 = Interrupt is disabled

bit 1-0 **PMPIS<1:0>:** PMP Interrupt Subpriority bits
- 11 = Interrupt Subpriority is 3
- 10 = Interrupt Subpriority is 2
- 01 = Interrupt Subpriority is 1
- 00 = Interrupt Subpriority is 0

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the PMP.

13.3 MASTER MODES OF OPERATION

In its master modes, the PMP module can provide a 16-bit or 8-bit data bus, up to 16 bits of address, and all the necessary control signals to operate a variety of external parallel devices such as memory devices, peripherals, and slave microcontrollers. The PMP master modes provide a simple interface for reading and writing data, but not executing program instructions from external devices, such as SRAM or Flash memories.

Because there are a number of parallel devices with a variety of control methods, the PMP module is designed for flexibility to accommodate a range of configurations. Some of these features include:

- 8-Bit and 16-Bit Data modes
- Configurable address/data multiplexing
- Up to 2 Chip Select lines
- Up to 16 selectable address lines
- Address auto-increment and auto-decrement
- Selectable polarity on all control lines
- Configurable Wait states at different stages of the read/write cycle

13.3.1 Parallel Master Port Configuration Options

13.3.1.1 8-Bit and 16-Bit Data Modes

The PMP in Master mode supports data widths 8 and 16 bits wide. By default, the data width is 8-bit, MODE16 (PMMODE<10>) bit = 0. To select 16-bit data width, set MODE16 = 1. When configured in 8-bit Data mode, the upper 8 bits of the data bus, PMD<15:8>, are not controlled by the PMP module and are available as general purpose I/O pins.

Note:	Data pins PMD<15:0> are available on 100-pin PIC32MX device variants. For 64-pin device variants, only pins PMD<7:0> are available. Refer to the specific PIC32MX device data sheet for details.

13.3.1.2 Chip Selects

Two Chip Select lines, PMCS1 and PMCS2, are available for the master modes. The two Chip Select lines are multiplexed with the Most Significant bits of the address bus A14 and A15. When a pin is configured as a Chip Select, it is not included in any address auto-increment/decrement. It is possible to enable both PMCS2 and PMCS1 as Chip Selects, or enable only PMCS2 as a Chip Select, allowing PMCS1 to function strictly as address line A14. It is not possible to enable PMCS1 alone. The Chip Select signals are configured using the Chip Select Function bits CSF<1:0> (PMCON <7:6>).

Table 13-2: Chip Select Control

CSF<1:0>	FUNCTION
00	PMCS2 = A15, PMCS1 = A14
01	PMCS2 = Enabled, PMCS1 = A14
10	PMCS2, PMCS1 = Enabled

13.3.1.3 PORT Pin Control

There are several bits available to configure the presence or absence of control and address signals in the module. These bits are PTWREN (PMCON<9>), PTRDEN (PMCON<8>), and PTEN<15:0> (PMAEN<15:0>). They give the user the ability to conserve pins for other functions and allow flexibility to control the external address. When any one of these bits is set, the associated function is present on its associated pin; when clear, the associated pin reverts to its defined I/O port function.

Setting a PTEN bit will enable the associated pin as an address pin and drive the corresponding data contained in the PMADDR register. Clearing any PTEN bit will force the pin to revert to its original I/O function.

For the pins configured as Chip Select (PMCS1 or PMCS2) with the corresponding PTEN bit set, Chip Select pins drive inactive data when a read or write operation is not being performed. The PTEN0 and PTEN1 bits also control the PMALL and PMALH signals. When multiplexing is used, the associated address latch signals should be enabled. Refer to **Section 13.10 "I/O Pin Control"** later in this chapter regarding I/O pin configuration.

13.3.1.4 Read/Write Control

The PMP module supports two distinct read/write signaling methods. In Master mode 1, read and write strobe are combined into a single control line, PMRD/PMWR; a second control line, PMENB, determines when a read or write action is to be taken. In Master mode 2, read and write strobes (PMRD and PMWR) are supplied on separate pins.

13.3.1.5 Control Line Polarity

All control signals (PMRD, PMWR, PMENB, PMALL, PMALH, PMCS2 and PMCS1) can be individually configured for either positive or negative polarity. Configuration is controlled by separate bits in the PMCON register.

Table 13-3: PIN POLARITY CONFIGURATION

CONTROL PIN	PMCON Control Bit	Active-High Select	Active-Low Select
PMRD	RDSP	1	0
PMWR	WRSP	1	0
PMCS2	CS2P	1	0
PMCS1	CS1P	1	0
PMALL	ALP	1	0
PMALH	ALP	1	0

Note that the polarity of control signals that share the same output pin (for example, PMWR and PMENB) are controlled by the same bit; the configuration depends on which master port mode is being used.

13.3.1.6 Auto-Increment/Decrement

While the module is operating in one of the master modes, the INCM<1:0> (PMMODE<12:11>) bits control the behavior of the address value. The address in the PMADDR register can be made to automatically increment or decrement by 1, regardless of the transfer data width, after each read and write operation is completed, and the BUSY bit (PMMODE<15>) goes to '0'.

Table 13-4: ADDRESS INC/DEC CONTROL

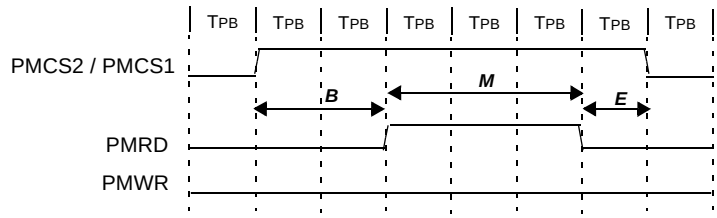
INCM<1:0>	FUNCTION
00	No Increment – No Decrement
01	Increment every R/W cycle
10	Decrement every R/W cycle

If the Chip Select signals are disabled and configured as address bits, the bits will participate in the increment and decrement operations; otherwise, CS2 and CS1 bit values will be unaffected.

13.3.1.7 Wait States

In Master mode, the user can control the duration of the read, write, and address cycles by configuring the module Wait states. One Wait state period is equivalent to one peripheral-bus-clock cycles, T_{PBCLK}. Below is an example of a Master mode 2 Read operation using Wait states

Figure 13-2: Read Operation, Wait States Enabled



Legend:
B = WAITB<1:0> = 01 (2 Wait states)
M = WAITM<3:0> = 0010 (3 Wait states)
E = WAITE<1:0> = 01 (1 Wait state)

Note: If WAITM<3:0> = 0000, M is forced to 1 T_{PBCLK}, WAITB is ignored (B forced to 1 T_{PBCLK}) and WAITE is ignored (E forced to 0 T_{PBCLK}).

Wait states can be added to the beginning, middle, and end of any read or write cycle using the corresponding WAITB, WAITM, and WAITE bits in the PMMODE register.

The WAITB<1:0> (PMMODE<7:6>) bits define the number of wait cycles for the data setup prior to the PMRD/PMWR strobe in Mode 10, or prior to the PMENB strobe in Mode 11. When multiplexing the address and data bus, ADRMUX<1:0> = 01, 10 or 11, WAITB defines the number of wait cycles for which the addressing period is extended.

The WAITM<3:0> (PMMODE<5:2>) bits define the number of wait cycles for the PMRD/PMWR strobe in Mode 10, or for the PMENB strobe in Mode 11. When this Wait state setting is 0000, WAITB and WAITE are ignored. The number of Wait states for the data setup time (WAITB) defaults to one while the number of Wait states for data hold time (WAITE) defaults to one during a write operation and zero during a read operation.

The WAITE<1:0> (PMMODE<1:0>) bits define the number of wait cycles for the data hold time after the PMRD/PMWR strobe in Mode 10, or after the PMENB strobe in Mode 11.

13.3.1.8 Address Multiplexing

Address multiplexing allows some or all address line signals to be generated from the data bus during the address cycle of a read/write operation. This can be a useful option for address lines PMA<15:0> needed as general purpose I/O pins. The user can select to multiplex the lower 8 data bits, upper 8 data bits or full 16 data bits. These multiplexing modes are available in both Master mode 1 and 2. Refer to **Section 13.3.8 "Master Mode Timing"** for the multiplexing mode timing diagrams.

Table 13-5: Address Multiplex Configurations

ADRMUX <1:0>	Address/Data Multiplex Modes
00	Demultiplexed
01	Partially Multiplexed (lower eight data pins PMD[7:0])
10	Fully multiplexed (lower eight data pins PMD[7:0])
11	Fully multiplexed (16 data pins PMD[15:0])

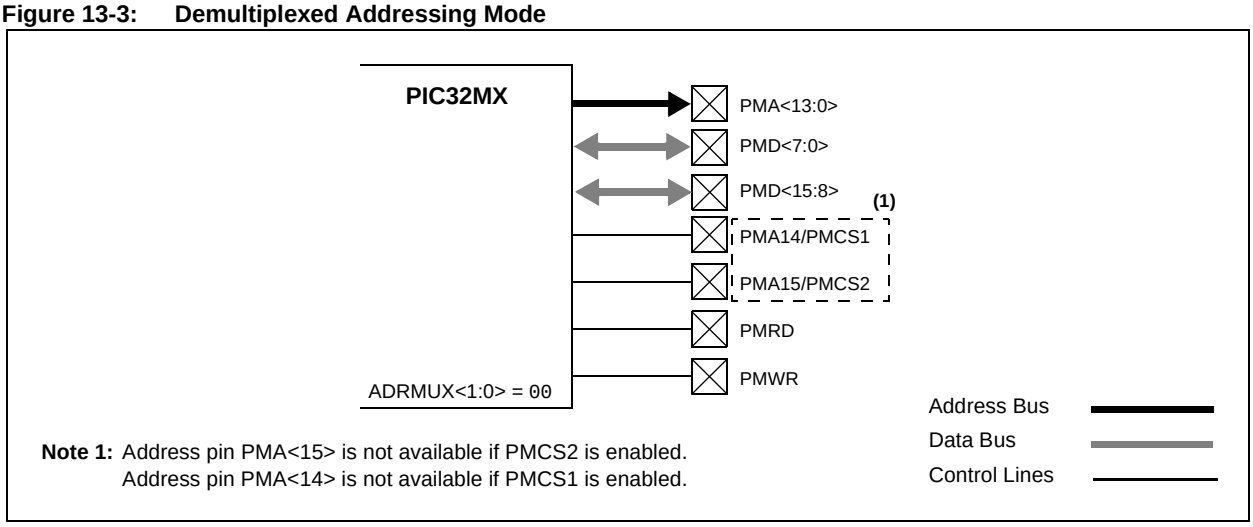
13.3.1.8.1 Demultiplexed Mode

Demultiplexed mode is selected by configuring bits ADRMUX<1:0> = 00, (PMMODE<9:8>). In this mode, address bits are presented on pins PMA<15:0>.

When PMCS2 is enabled, address pin PMA15 is not available. When PMCS1 is enabled, Address pin PMA14 is not available.

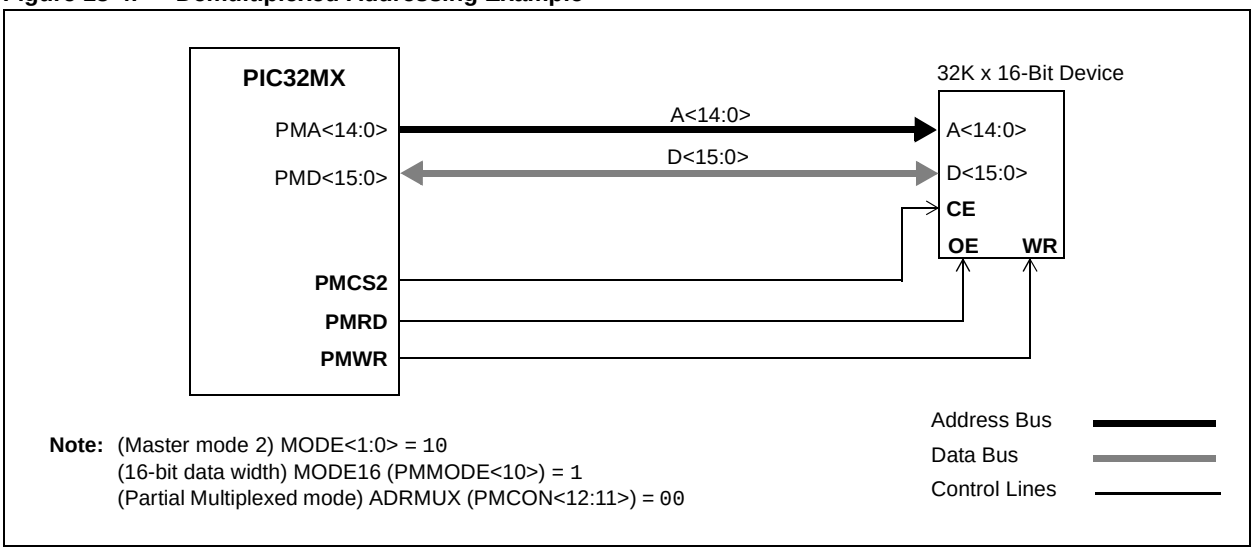
In 16-Bit Data mode, data bits are presented on pins PMD<15:0>. In 8-Bit Data mode, data bits are presented on pins PMD<7:0>.

Figure 13-3: Demultiplexed Addressing Mode



Note 1: Address pin PMA<15> is not available if PMCS2 is enabled.
Address pin PMA<14> is not available if PMCS1 is enabled.

Figure 13-4: Demultiplexed Addressing Example



Note: (Master mode 2) MODE<1:0> = 10
(16-bit data width) MODE16 (PMMODE<10>) = 1
(Partial Multiplexed mode) ADRMUX (PMCON<12:11>) = 00

13.3.1.8.2 Partially Multiplexed Mode

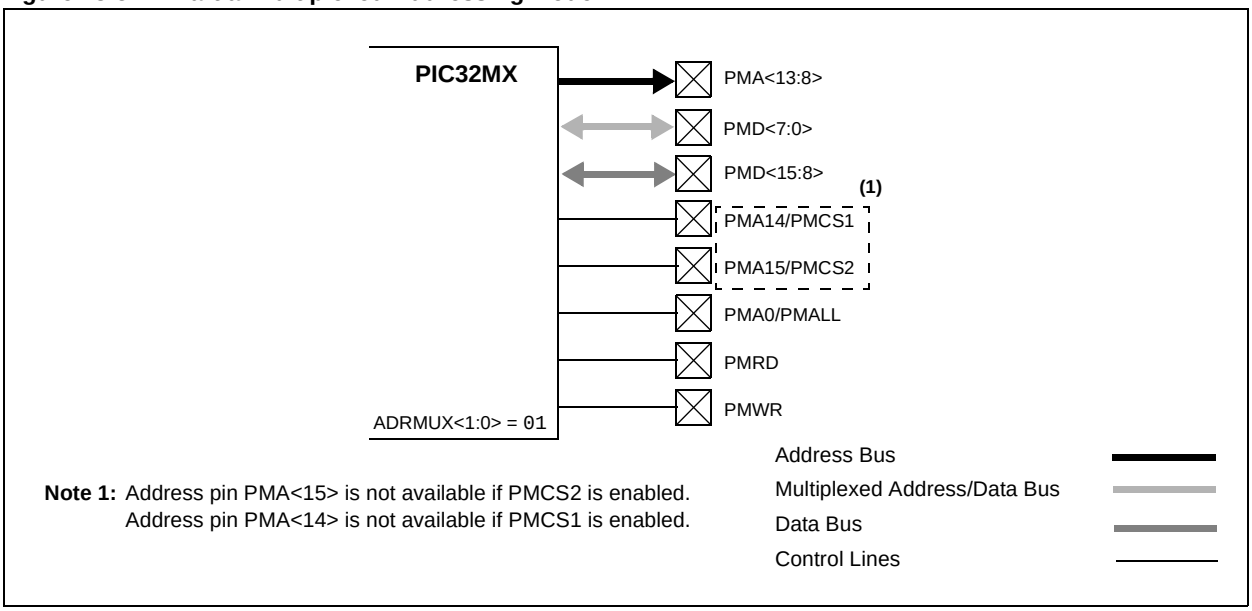
Partially Multiplexed mode (8-Bit data pins) is available in both 8-bit and 16-bit data bus configurations and is selected by setting bits ADRMUX<1:0> = 01. In this mode, the lower eight address bits are multiplexed with the lower eight data bus pins, PMD<7:0>. The upper eight address bits are unaffected and are presented on PMA<15:8>. In this mode, address pins PMA<7:1> are available as general purpose I/O pins.

Address pin PMA15 is not available when PMCS2 is enabled; address pin PMA14 is not available when PMCS1 is enabled.

Address pin PMA<0> is used as an Address Latch enable strobe, PMALL, during which the lower eight bits of the address are presented on the PMD<7:0> pins. Read and write sequences are extended by at least 3 peripheral-bus-clock cycles (T_{PBCLK}).

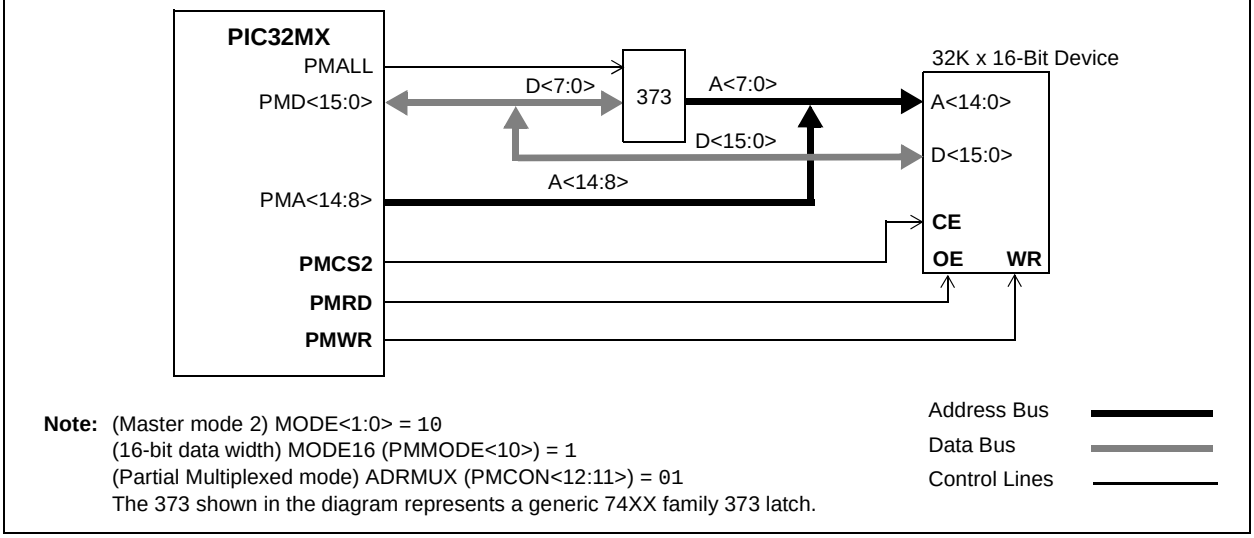
If WAITM<3:0> (PMMODE<5:2>) is non-zero, the PMALL strobe will be extended by WAITB<1:0> (PMMODE<7:6>) Wait states.

Figure 13-5: Partial Multiplexed Addressing Mode



Note 1: Address pin PMA<15> is not available if PMCS2 is enabled.
Address pin PMA<14> is not available if PMCS1 is enabled.

Figure 13-6: Partial Multiplexed Addressing Example



Note: (Master mode 2) MODE<1:0> = 10
(16-bit data width) MODE16 (PMMODE<10>) = 1
(Partial Multiplexed mode) ADRMUX (PMCON<12:11>) = 01
The 373 shown in the diagram represents a generic 74XX family 373 latch.

13.3.1.8.3 Full Multiplexed mode (8-bit data pins)

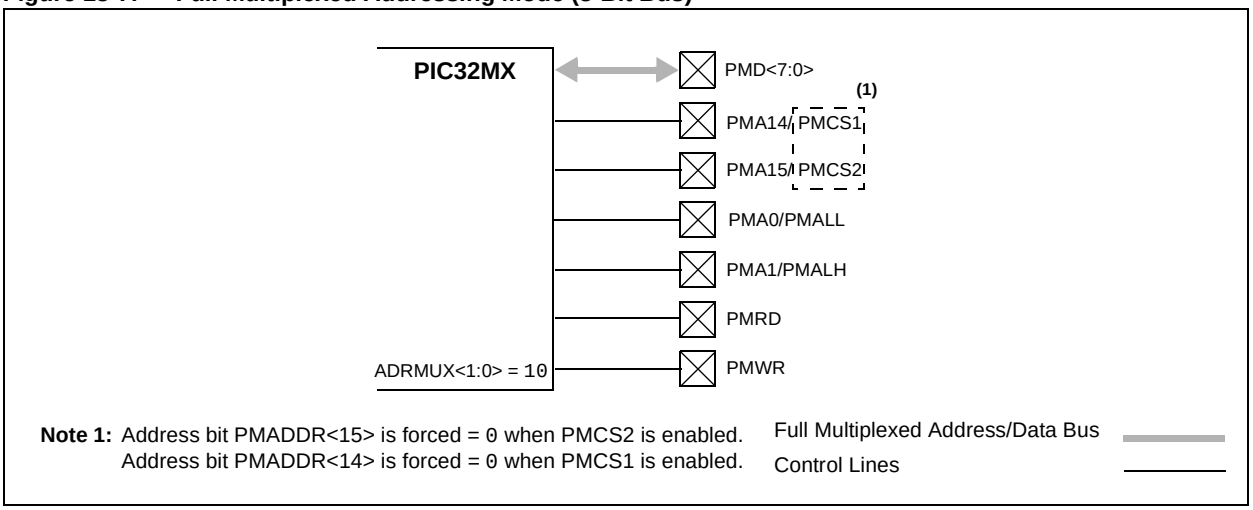
Full multiplexed mode (8-Bit data pins) is available in both 8-bit and 16-bit data bus configurations and is selected by setting bits ADRMUX<1:0> (PMCON<12:11>) = 10. In this mode, the entire 16 bits of the address are multiplexed with the lower eight data bus pins, PMD<7:0>. In this mode, Pins PMA<13:2> available as general purpose I/O pins.

In the event the pins PMCS2/PMA15 or PMCS1/PMA14 are configured as Chip Select pins, the corresponding address bits PMADDR<15> or PMADDR<14> are automatically forced to '0'.

Address pins PMA<0> and PMA<1> are used as an Address Latch enable strobes, PMALL and PMALH, respectively. During the first cycle, the lower eight address bits are presented on the PMD<7:0> pins with the PMALL strobe active. During the second cycle the upper eight address bits are presented on the PMD<7:0> pins with the PMALH strobe active. The read and write sequences are extended by at least 6 peripheral-bus-clock cycles (T_{PBCLK}).

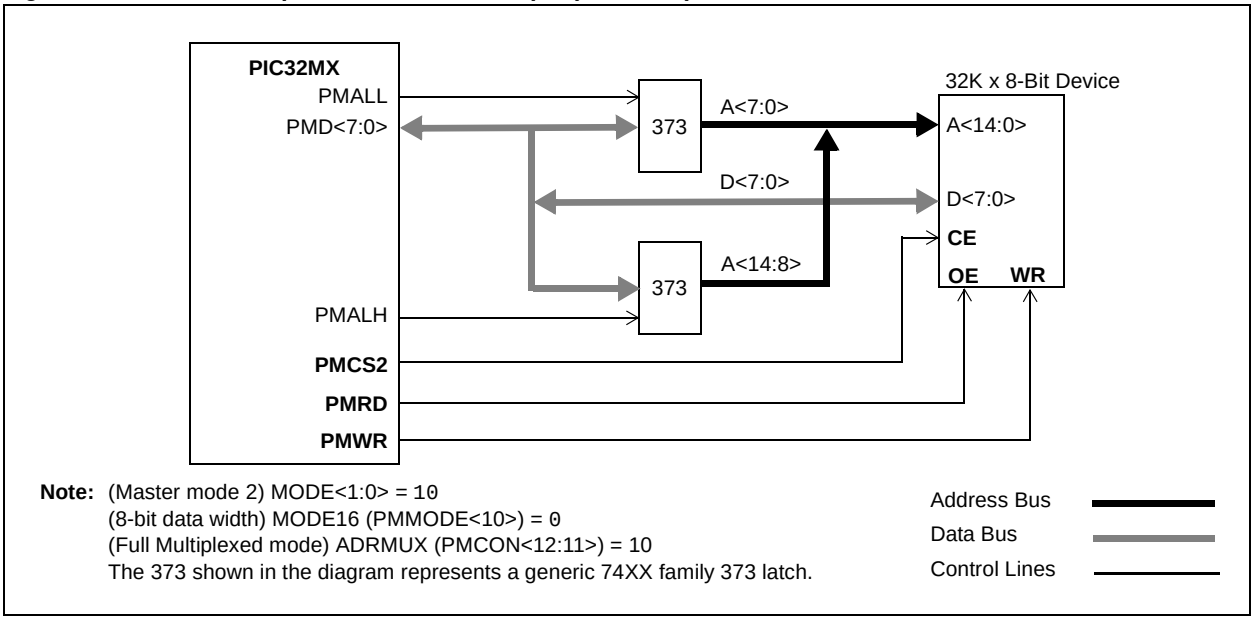
If WAITM<3:0> (PMMODE<5:2>) is non-zero, both PMALL and PMALH strobes will be extended by WAITB<1:0> (PMMODE<7:6>) Wait states.

Figure 13-7: Full Multiplexed Addressing Mode (8-Bit Bus)



Note 1: Address bit PMADDR<15> is forced = 0 when PMCS2 is enabled.
Address bit PMADDR<14> is forced = 0 when PMCS1 is enabled.

Figure 13-8: Full Multiplexed Address Example (8-Bit Bus)



Note: (Master mode 2) MODE<1:0> = 10
(8-bit data width) MODE16 (PMMODE<10>) = 0
(Full Multiplexed mode) ADRMUX (PMCON<12:11>) = 10
The 373 shown in the diagram represents a generic 74XX family 373 latch.

13.3.1.8.4 Full Multiplexed mode (16-bit data pins)

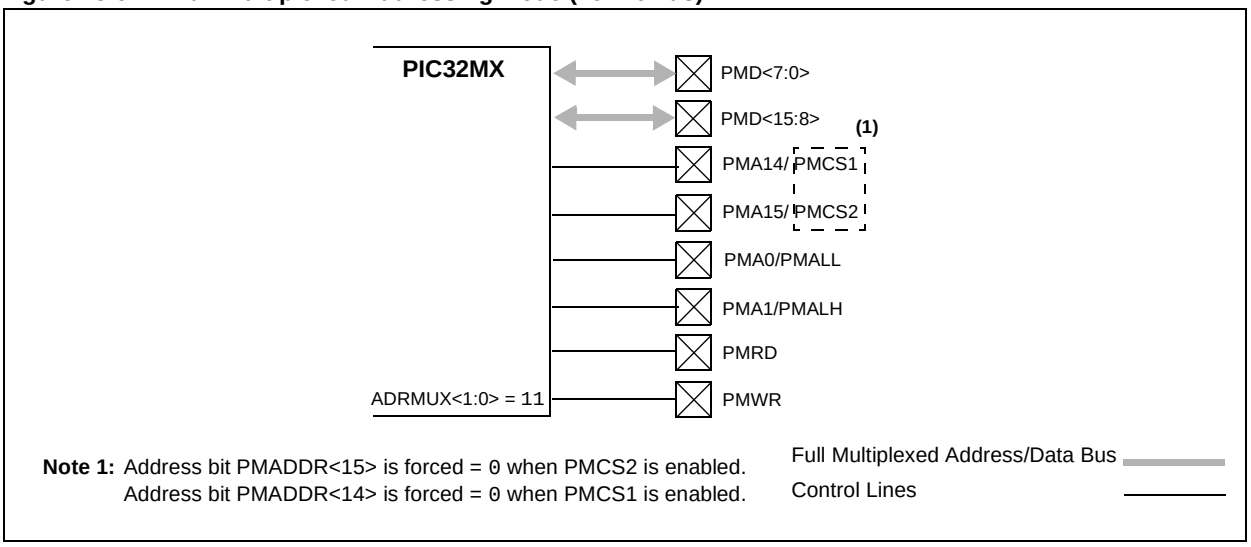
Full Multiplexed mode (16-Bit data pins) is only available in the 16-bit data bus configuration and is selected by configuring bits ADRMUX<1:0> (PMCON<12:11>) = 11. In this mode, the entire 16 bits of the address are multiplexed with all 16 data bus pins, PMD<15:0>.

In the event the pins PMCS2/PMA15 or PMCS1/PMA14 are configured as Chip Select pins, the corresponding address bits PMADDR<15> or PMADDR<14> are automatically forced to '0'.

Address pins PMA<0> and PMA<1> are used as an Address Latch enable strobes, PMALL and PMALH respectively, and at the same time. While the PMALL and PMALH strobes are active, the lower eight address bits are presented on the PMD<7:0> pins and the upper eight address bits are presented on the PMD<15:8> pins. The read and write sequences are extended by at least 3 peripheral-bus-clock cycles (T_{PBCLK}).

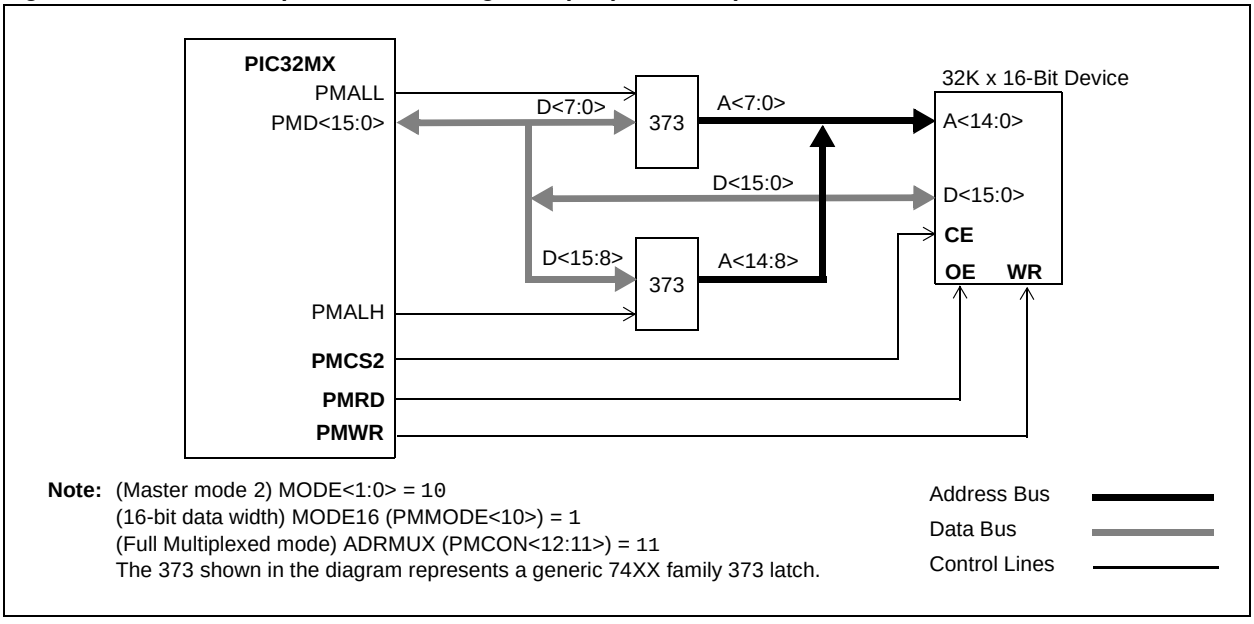
If WAITM<3:0> (PMMODE<5:2>) is non-zero, both PMALL and PMALH strobes will be extended by WAITB<1:0> (PMMODE<7:6>) Wait states.

Figure 13-9: Full Multiplexed Addressing Mode (16-Bit Bus)



Note 1: Address bit PMADDR<15> is forced = 0 when PMCS2 is enabled.
Address bit PMADDR<14> is forced = 0 when PMCS1 is enabled.

Figure 13-10: Full Multiplexed Addressing Example (16-Bit Bus)



Note: (Master mode 2) MODE<1:0> = 10
(16-bit data width) MODE16 (PMMODE<10>) = 1
(Full Multiplexed mode) ADRMUX (PMCON<12:11>) = 11
The 373 shown in the diagram represents a generic 74XX family 373 latch.

13.3.2 Master Port Configuration

The Master mode configuration is determined primarily by the interface requirements to the external device. Address multiplexing, control signal polarity, data width and Wait states typically dictate the specific configuration of the PMP master port.

To use the PMP as a master, the module must be enabled, control bit ON (PMCON<15>) = 1, and the mode must be set to one of two possible master modes. Control bits MODE<1:0> (PMMODE<9:8>) = 10 for Master mode 2 or MODE<1:0> = 11 for Master mode 1.

The following Master mode initialization properly prepares the PMP port for communicating with an external device.

1. If interrupts are used, disable the PMP interrupt by clearing the interrupt enable bit PMPIE (IEC1<2>) = 0.
2. Stop and reset the PMP module by clearing the control bit ON (PMCON<15>) = 0.
3. Configure the desired settings in the PMCON, PMMODE and PMAEN control registers.
4. If interrupts are used:
   a. Clear interrupt flag bit PMPIF (IFS1<2>) = 0.
   b. Configure the PMP interrupt priority bits PMPIP<2:0> (IPC7<4:2>) and interrupt sub-priority bits PMPIS (IPC7<1:0>.
   c. Enable PMP interrupt by setting interrupt enable bit PMPIE = 1.
5. Enable the PMP master port by setting control bit ON = 1.

Note:	It is recommended to wait for any pending read or write operation to be completed before reconfiguring the PMP module.

The following illustrates an example setup for a typical Master mode 2 operation:

- Select Master mode 2 – MODE<1:0> (PMMODE<9:8>) = 10.
- Select 16-bit Data mode – MODE16 (PMMODE<10>) = 0.
- Select partially multiplexed addressing – ADRMUX<1:0> (PMCON<12:11>) = 01.
- Select auto address increment – INCM<1:0> (PMMODE<12:11>) = 01.
- Enable Interrupt Request mode – IRQM<1:0> (PMMODE<14:13>) = 01.
- Enable PMRD strobe – PTRDEN (PMCON<8>) = 1.
- Enable PMWR strobe – PTWREN (PMCON<9>) = 1.
- Enable PMCS2 and PMCS1 Chip Selects – CSF (PMCON<7:6>) = 10.
- Select PMRD active-low pin polarity – RDSP (PMCON<0>) = 0.
- Select PMWR active-low pin polarity – WRSP (PMCON<1>) = 0.
- Select PMCS2, PMCS1 active-low pin polarity – CS2P (PMCON<4>) = 0 and CS1P (PMCON<3>) = 0.
- Select 1 wait cycle for data setup – WAITB<1:0>(PMMODE<7:6>) = 00.
- Select 2 wait cycles to extend PMRD/PMWR – WAITM<3:0>(PMMODE<5:2>) = 0001.
- Select 1 wait cycle for data hold – WAITE<1:0>(PMMODE<1:0>) = 00.
- Enable upper 8 PMA<15:8> address pins – PMAEN<15:8> = 1 (the lower 8 bits can be used as general purpose I/O).

See the example code shown in Example 13-1.

Example 13-1: Initialization for Master Mode 2, Demultiplexed Address, 16-Bit Data

```
/*
    Configuration Example: Master mode 2, 16-bit data, partially multiplexed
    address/data, active-lo polarities.
*/
    IEC1CLR = 0x0004      // Disable PMP interrupt
    PMCON = 0x0000;       // Stop PMP module and clear control register
    PMCONSET = 0x0B80;    // Configure the addressing and polarities
    PMMODE = 0x2A40;      // Configure the mode
    PMAEN = 0xFF00;       // Enable all address and Chip select lines

    IPC7SET = 0x001C;     // Set priority level=7 and
    IPC7SET = 0x0003;     // Set subpriority level=3
                          // Could have also done this in single
                          // operation by assigning IPC7SET = 0x001F

    IEC1SET  = 0x0004;    // Enable PMP interrupts
    PMCONSET = 0x8000;    // Enable PMP module
```

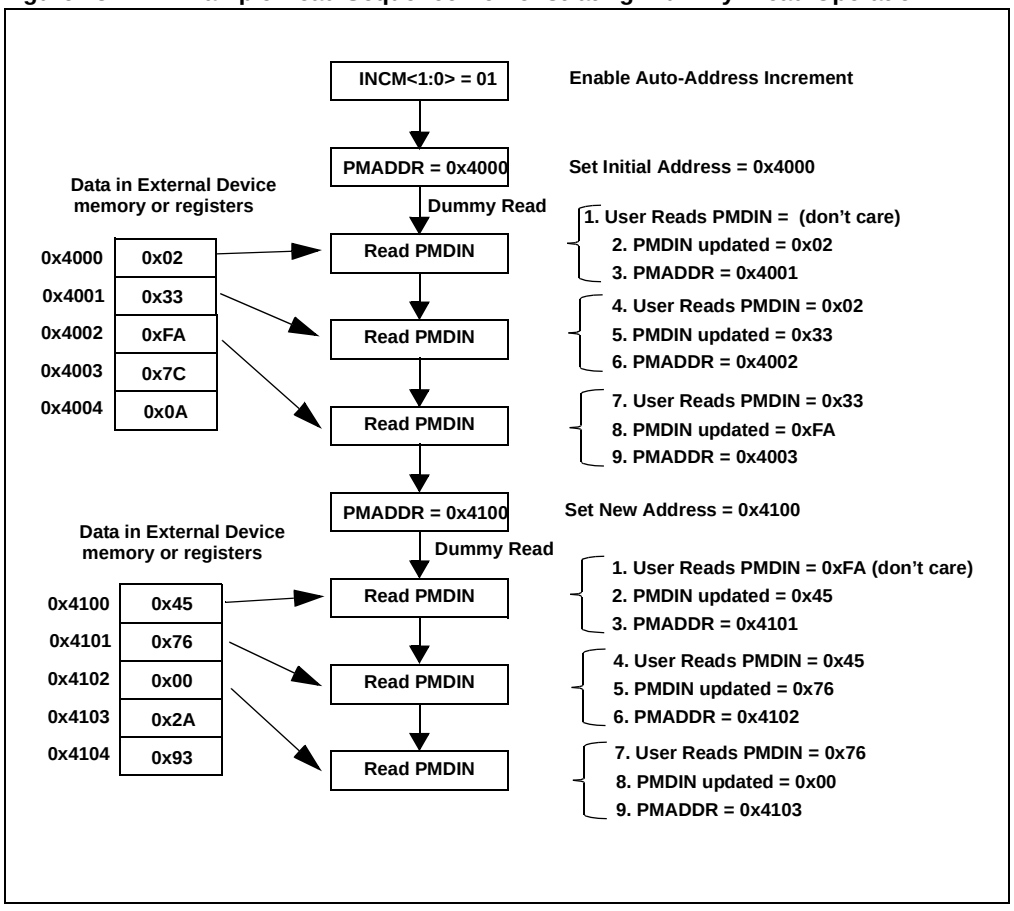
13.3.3 Read Operation

To perform a read on the parallel bus, the user reads the PMDIN register. The effect of reading the PMDIN register retrieves the current value and causes the PMP to activate the Chip Select lines and the address bus. The read line PMRD is strobed in Master mode 2, PMRD/PMWR and PMENB lines in Master mode 1, and the new data is latched into the PMDIN register making it available for the next time the PMDIN register is read.

Note that the read data obtained from the PMDIN register is actually the read value from the previous read operation. Hence, the first user read will be a dummy read to initiate the first bus read and fill the read register. See Figure 13-11 below illustrating this sequence. Also, the requested read value will not be ready until after the BUSY bit is observed low. Thus, in a back-to-back read operation, the data read from the register will be the same for both reads. The next read of the register will yield the new value.

In 16-Bit Data mode, PMMODE<MODE16> = 1, the read from the PMDIN register causes the data bus PMD<15:0> to be read into PMDIN<15:0>. In 8-bit mode, PMMODE<MODE16> = 0, the read from the PMDIN register causes the data bus PMD<7:0> to be read into PMDIN<7:0>. The upper 8 bits, PMD<15:8>, are ignored.

Figure 13-11: Example Read Sequence Demonstrating 'Dummy' Read Operation



13.3.4 Write Operation

To perform a write on the parallel port, the user writes to the PMDIN register (same register as a read operation). This causes the module to first activate the Chip Select lines and the address bus. The write data from the PMDIN register is placed onto the PMD data bus and the write line PMPWR is strobed in Master mode 2, PMRD/PMWR and PMENB lines in Master Mode 1.

In 16-Bit Data mode, PMMODE<MODE16> = 1, the write to the PMDIN register causes PMDIN<15:0> to appear on the data bus, (PMD<15:0>). In 8-bit mode, PMMODE<MODE16> = 0, the write to the PMDIN register causes PMDIN<7:0> to appear on the data bus, PMD<7:0>. The upper 8 bits, PMD<15:8>, are ignored.

13.3.5 Master Mode Interrupts

In PMP master modes, the PMPIF bit is set on every read or write strobe. An interrupt request is generated when bits IRQM<1:0> (PMMODE<14:13>) are set = 01 and PMP interrupts are enabled, PMPIE (IEC1<2>) = 1.

13.3.6 Parallel Master Port Status – The BUSY Bit

In addition to the PMP interrupt, a BUSY bit, (PMMODE<15>), is provided to indicate the status of the module. This bit is only used in Master mode.

While any read or write operation is in progress, the BUSY bit is set for all but the very last peripheral bus cycle of the operation. This is helpful when Wait states are enabled or multiplexed address/data is selected.

While the bit is set, any request by the user to initiate a new operation will be ignored (i.e., writing or reading the PMDIN register will not initiate a read or a write).

Since the system clock, SYSCLK, can operate faster than the peripheral bus clock in certain configurations, or if a large number of Wait states are used, it is possible for the PMP to be in the process of completing a read or write operation when the next CPU instruction is reading or writing to the PMP module. For this reason, it is highly recommended that the BUSY bit be checked prior to any operation that accesses the PMDIN or PMADDR registers. Below is an example of a polling operation of the BUSY bit prior to accessing the PMP module.

In most applications, the PMP's Chip Select pin(s) provide the Chip Select interface and are under the timing control of the PMP module. However, some applications may require the PMP Chip Select pin(s) not be configured as a Chip Select, but as a high order address line, such as PMA<14> or PMA<15>. In this situation, the application's Chip Select function must be provided by an available I/O port pin under software control. In these cases, it is especially important that the user's software poll the BUSY bit to ensure any read or write operation is complete before de-asserting the software controlled Chip Select.

Example 13-2: Example Code: Polling the BUSY Bit Flag

```
/*
    This example reads 256 16-bit words from an external device at address 0x4000 and copies
    the data to a second external device at address 0x8000. The PMP port is operating in
    Master mode 2. Note how the PMP's BUSY bit is polled prior to all operations to the
    PMDOUT, PMDIN or PMADDR register, except where noted.
*/
    unsigned short DataArray<256>;

                                  // Provide the setup code here including large Wait
                                  // states, auto increment.
    ...
    CopyData();                   // A call to the copy function is made.
    ...

void CopyData()
{
    PMADDR = 0x4000;              // Init the PMP address. First time, no need to poll BUSY
                                  // bit.
    while(PMMODE & 0x8000);       // Poll - if busy, wait before reading.
    PMDIN;                        // Read the PMDIN to clear previous data and latch new
                                  // data.

    for(i=0; i<256; i++)
    {
        while(PMMODE & 0x8000);   // Poll - if busy, wait before reading.
        DataArray<i> = PMDIN;     // Read the external device.
    }

    while(PMMODE & 0x8000);       // Poll - if busy, wait before changing PMADDR.
    PMADDR = 0x8000;              // Address of second external device.

    for(i=0; i<256; i++)
    {
        while(PMMODE & 0x8000);   // Poll - if busy, wait before writing.
        DataArray<i> = PMDIN;     // Read the external device.
    }
    return();
}
```


13.3.7 Addressing Considerations

PMCS2 and PMCS1 Chip Select pins share functionality with address lines A15 and A14. It is possible to enable both PMCS2 and PMCS1 as Chip Selects, or enable only PMCS2 as a Chip Select; allowing PMCS1 to function strictly as address line A14. It is not possible to enable only PMCS1.

Note:	Setting both A15 and A14 = 1 when PMCS2 and PMCS1 are enabled as Chip Selects will cause both PMCS2 and PMCS1 to be active during a read or write operation. This may enable two devices simultaneously and should be avoided.

When configured as Chip Selects, a '1' must be written into bit position 15 or 14 of the PMADDR register in order for PMCS2 or PMCS1 to become active during a read or write operation. Failing to write a '1' to PMCS2 or PMCS1 does not prevent address pins PMA<13:0> from being active as the specified address appears; however, no Chip Select signal will be active.

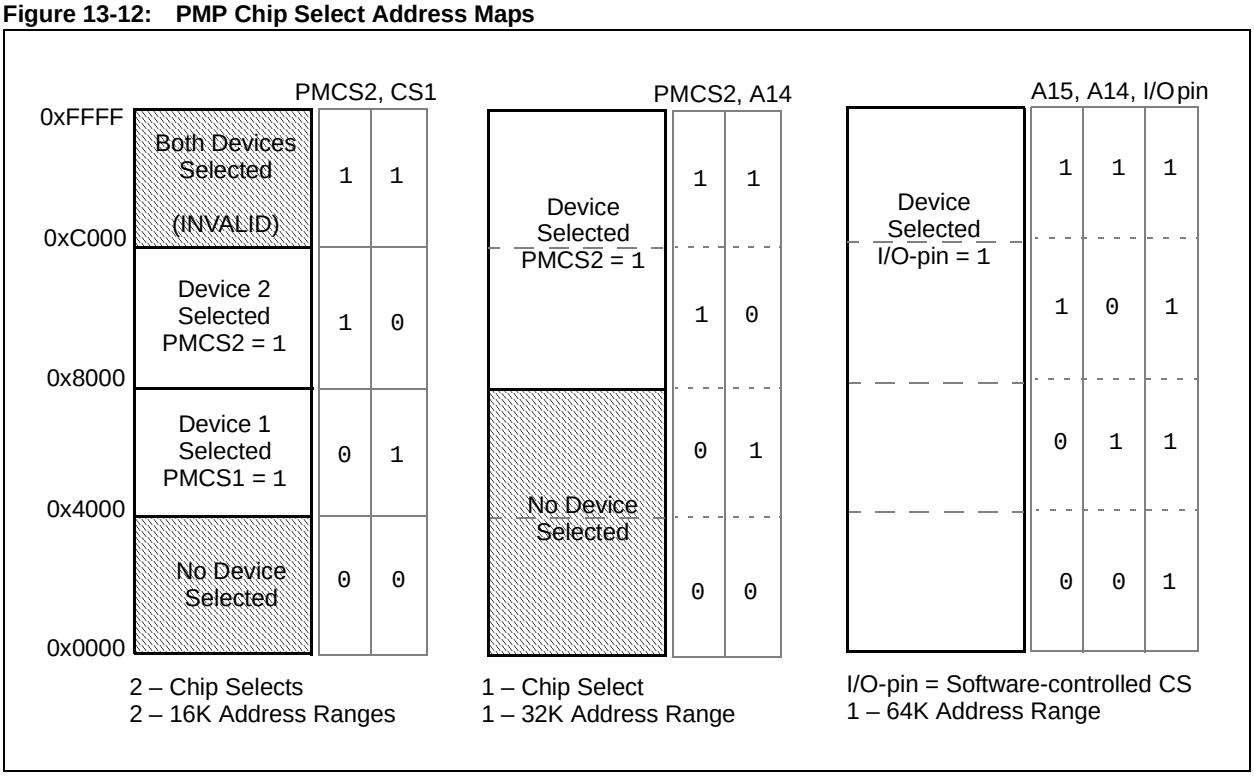
Note:	When using Auto-Increment Address mode, PMCS2 and PMCS1 do not participate and must be controlled by the user's software by writing to '1' to PMADDR<15:14> explicitly.

In Full Multiplexed modes, address bits PMADDR<15:0> are multiplexed with the data bus and in the event address bits PMA15 or PMA14 are configured as Chip Selects, the corresponding PMADDR<15:14> address bits are automatically forced = 0. Disabling one or both PMCS2 and PMCS1 makes these bits available as address bits PMADDR<15:14>.

In any of the Master mode multiplexing schemes, disabling both Chip Select pins PMCS2 and PMCS1 requires the user to provide Chip Select line control through some other I/O pin under software control. See Figure 13-12.

Refer to **Section 13.11 "Design Tips"** for additional information regarding memory banking.

Figure 13-12: PMP Chip Select Address Maps



13.3.8 Master Mode Timing

A PMP Master mode cycle time is defined as the number of PBCLK cycles required by the PMP to perform a read or write operation and is dependent on PBCLK clock speed, PMP address/data multiplexing modes and the number of PMP wait states, if any. Refer to the PIC32MX Family Data sheet for specific setup and hold timing characteristics.

A PMP master mode read or write cycle is initiated by accessing (reading or writing) the PMDIN register. Table 13-6 below provides a summary of read and write PMP cycle times for each multiplex configuration.

The actual data rate of the PMP (the rate which user's code can perform a sequence of read or write operations) will be highly dependent on several factors:

- a user's application code content
- code optimization level
- internal bus activity
- other factors relating to the instruction execution speed.

Note:	During any Master mode read or write operation, the busy flag will always de-assert 1 peripheral bus clock cycle (T_{PBCLK}), before the end of the operation, including Wait states. The user's application must check the status of the busy flag to ensure it is = 0 before initiating the next PMP operation.

Table 13-6: PMP Read/Write Cycle Times

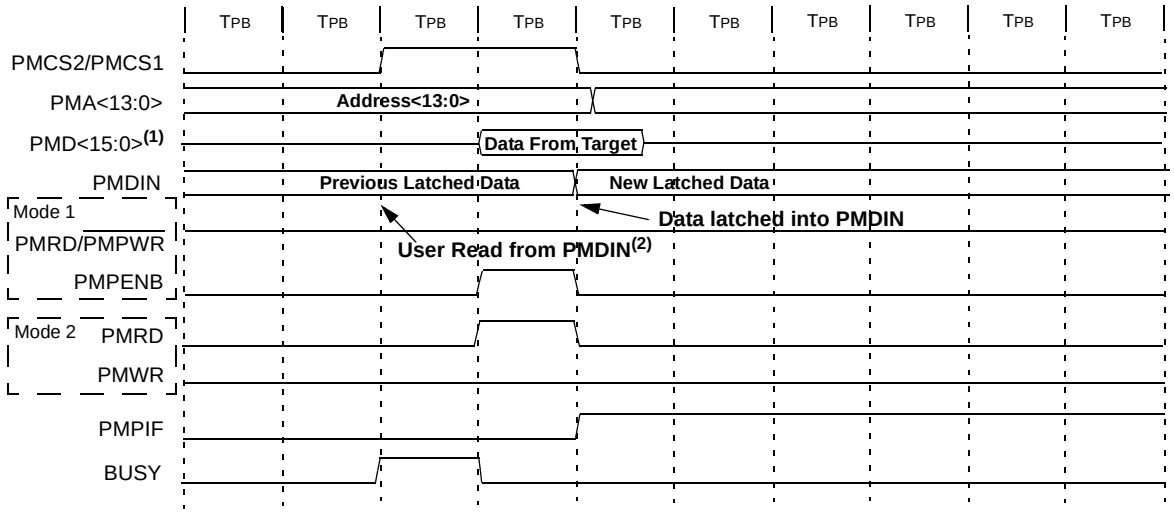
Address/Data Multiplex Configuration	ADRMUX bit settings	PMP Cycle Time (PBCLK cycles)	
		Read	Write
Demultiplexed	00	2	3
Partial Multiplex	01	5	6
Full Multiplexed (8-bit data)	10	8	9
Full Multiplexed (16-bit data)	11	5	6
Note: Wait states are not enabled			

The following timing examples represent the common master mode configuration options. These options vary from 8-Bit to 16-Bit data, non-multiplexed to full multiplexed address, as well as with and without Wait states. For illustration purposes only, all control signal polarities are shown as “active-high”.

13.3.8.1 Demultiplexed Address and Data Timing

This timing diagram illustrates demultiplexed timing (separate address and data bus) for a read operation with no Wait states. A read operation requires 2 T_{PBCLK}, peripheral-bus-clock cycles.

Figure 13-13: 8-Bit, 16-Bit Read Operations, ADRMUX = 00, No Wait States

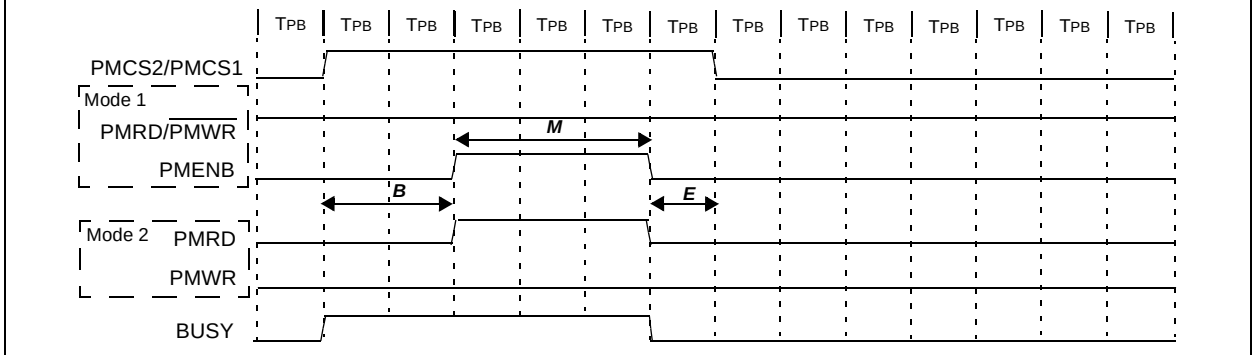


Note 1: In 8-bit mode, PMD<15:8> are not implemented.

2: Read data obtained from the PMDIN register is actually the value from the previous read operation.

In this timing diagram with Wait states, the read operation requires 6 T_{PBCLK}, peripheral-bus-clock cycles.

Figure 13-14: 8-Bit, 16-Bit Read Operations, ADRMUX = 00, Wait States Enabled



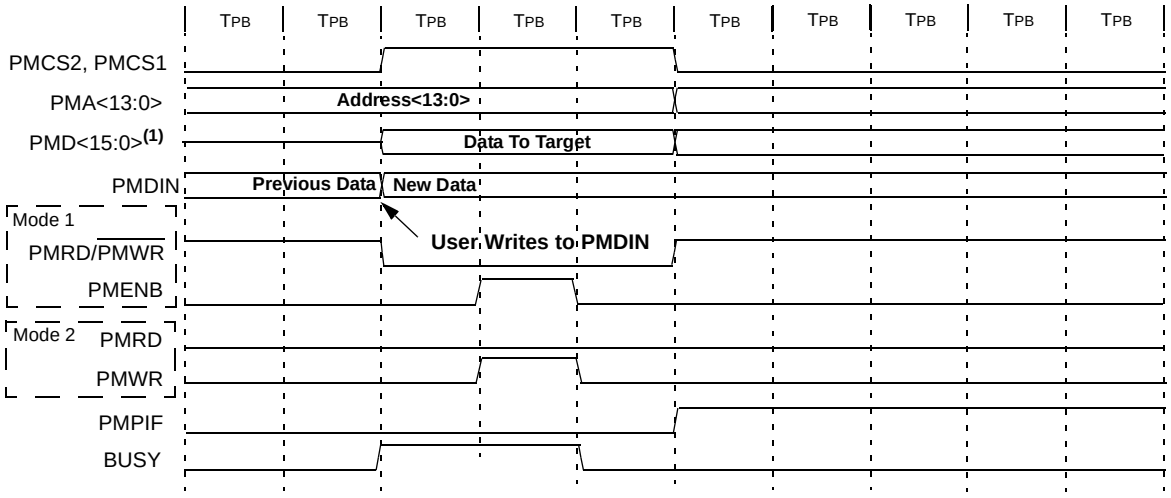
Legend:

B = WAITB<1:0> = 01 (2 Wait states)
M = WAITM<3:0> = 0010 (3 Wait states)
E = WAITE<1:0> = 01 (1 Wait state)

Note: If WAITM<3:0> = 0000, M is forced to 1 T_{PBCLK}, WAITB is ignored (B forced to 1 T_{PBCLK}), and WAITE is ignored (E forced to 0 T_{PBCLK}).

This timing diagram illustrates demultiplexed timing (separate address and data bus) for a write operation with no Wait states. A write operation requires 3 T_{PBCLK}, peripheral-bus-clock cycles.

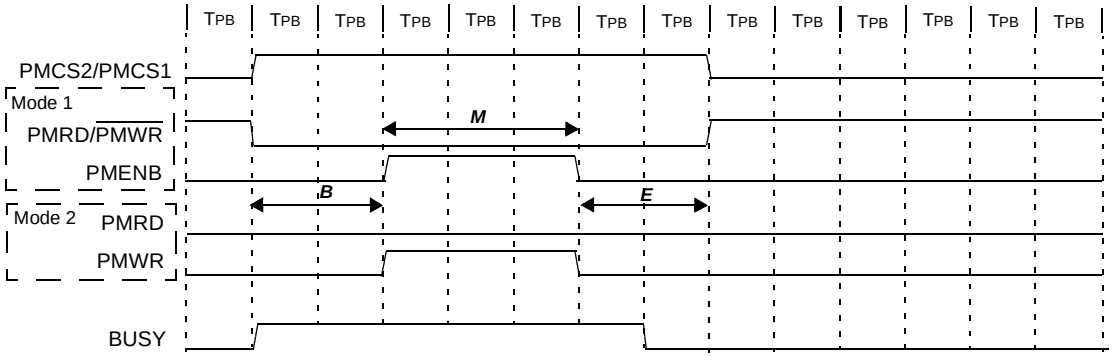
Figure 13-15: 8-Bit, 16-Bit Write Operations, ADRMUX = 00, No Wait States



Note 1: In 8-bit mode, PMD<15:8> are not implemented.

In this timing diagram with Wait states, the write operation requires 7 T_{PBCLK}, peripheral-bus-clock cycles.

Figure 13-16: 8-Bit, 16-Bit Write Operations, ADRMUX = 00, Wait States Enabled



Legend:

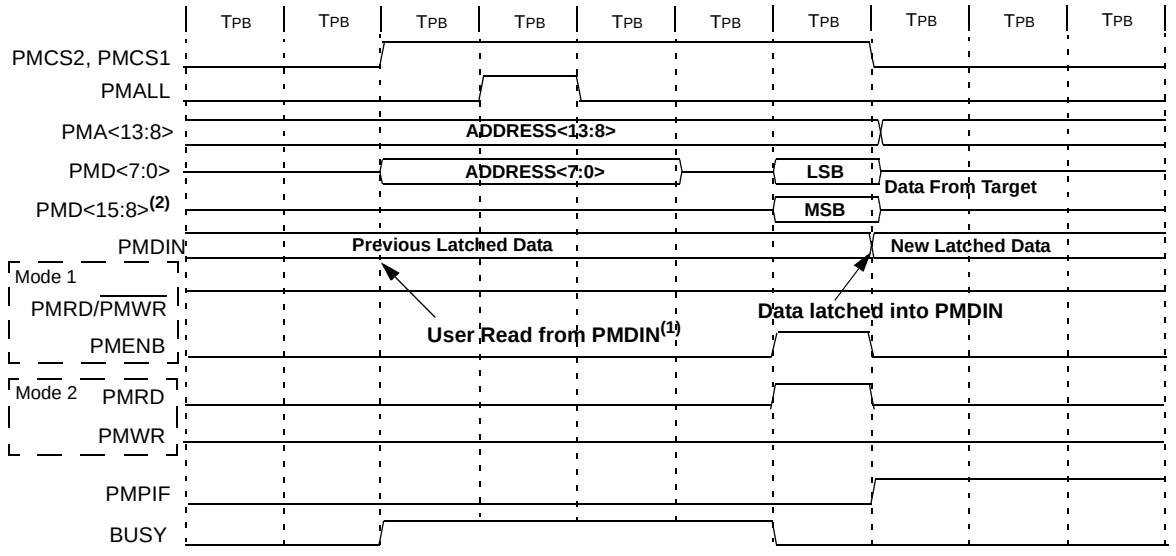
B = WAITB<1:0> = 01 (2 Wait states)
M = WAITM<3:0> = 0010 (3 Wait states)
E = WAITE<1:0> = 01 (2 Wait states)

Note: If WAITM<3:0> = 0000, M is forced to 1 T_{PBCLK}, WAITB is ignored (B forced to 1 T_{PBCLK}), and WAITE is ignored (E forced to 1 T_{PBCLK}).

13.3.8.2 Partially Multiplexed Address and Data Timing

This timing diagram illustrates partially multiplexed timing (address bits <7:0> multiplexed with data bus, PMD<7:0>) for a read operation with no Wait states. A read operation requires 5 T_{PBCLK}, peripheral-bus-clock cycles.

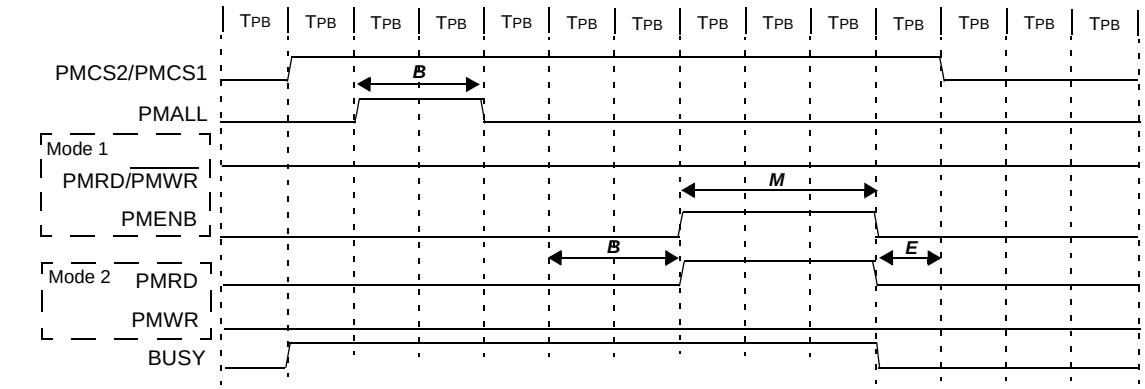
Figure 13-17: 8-Bit, 16-Bit Read Operations, ADRMUX = 01, No Wait States



Note 1: Read data obtained from the PMDIN register is actually the value from the previous read operation.
2: In 8-Bit mode, PMD<15:8> are not implemented.

In this timing diagram with Wait states, the read operation requires 10 T_{PBCLK}, peripheral-bus-clock cycles.

Figure 13-18: 8-Bit, 16-Bit Read Operations, ADRMUX = 01, Wait States Enabled

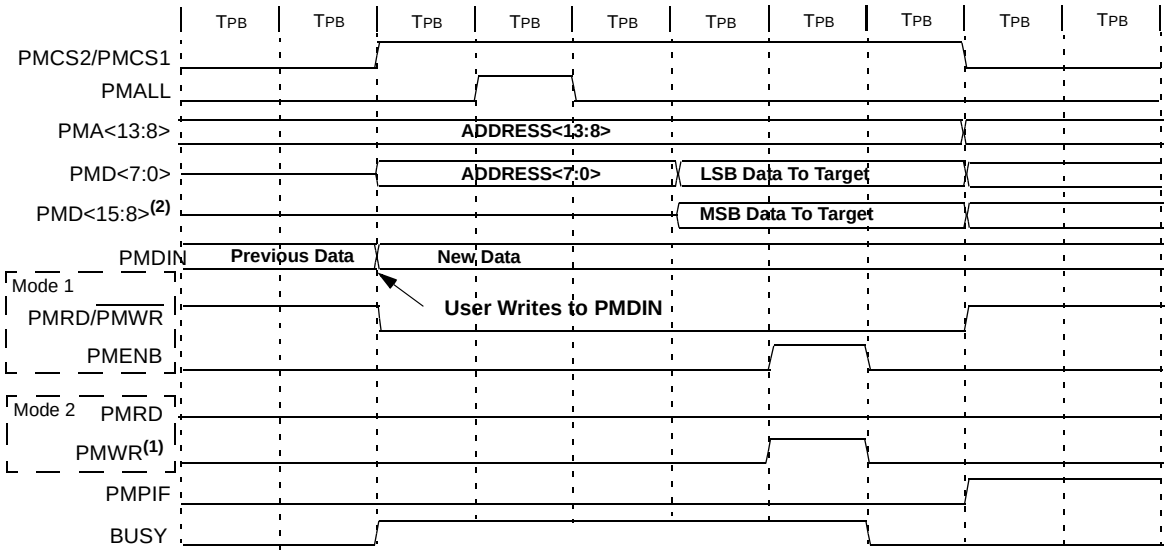


Legend:
B = WAITB<1:0> = 01 (2 Wait states)
M = WAITM<3:0> = 0010 (3 Wait states)
E = WAITE<1:0> = 01 (1 Wait state)

Note: If WAITM<3:0> = 0000, M is forced to 1 T_{PBCLK}, WAITB is ignored (B forced to 1 T_{PBCLK}) and WAITE is ignored (E forced to 0 T_{PBCLK}).

This timing diagram illustrates partially multiplexed timing (address bits <7:0> multiplexed with data bus, PMD<7:0>) for a write operation with no Wait states. A write operation requires 6 T_{PBCLK}, peripheral-bus-clock cycles.

Figure 13-19: 8-Bit, 16-Bit Write Operations, ADRMUX = 01, No Wait States

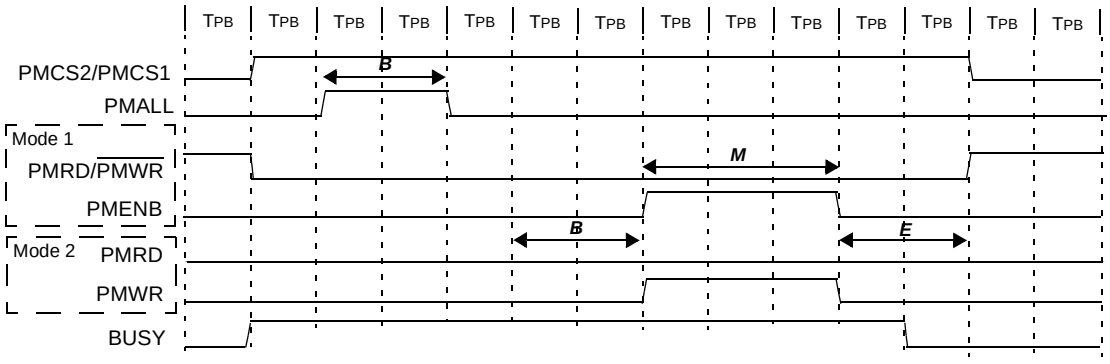


Note 1: During a write operation, there is one TPBCLK hold cycle following the PMWR signal.

2: In 8-bit mode, PMD<15:8> are not implemented.

In this timing diagram with Wait states, the write operation requires 11 T_{PBCLK}, peripheral-bus-clock cycles.

Figure 13-20: 8-Bit, 16-Bit Write Operations, ADRMUX = 01, Wait States Enabled



Legend:

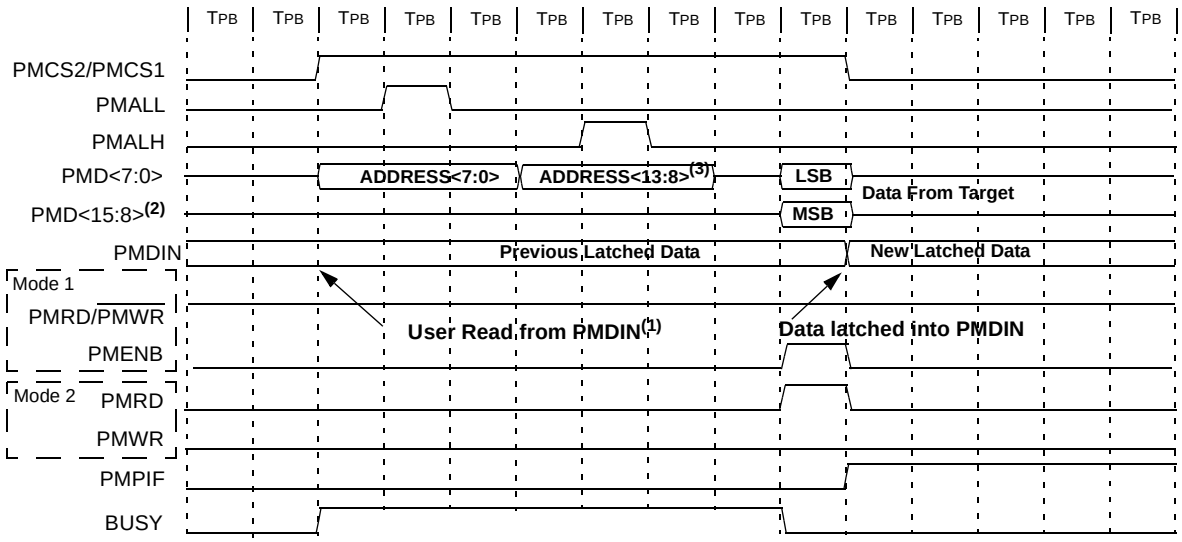
B = WAITB<1:0> = 01 (2 Wait states)
M = WAITM<3:0> = 0010 (3 Wait states)
E = WAITE<1:0> = 01 (1 Wait state)

Note: If WAITM<3:0> = 0000, M is forced to 1 T_{PBCLK}, WAITB is ignored (B forced to 1 T_{PBCLK}) and WAITE is ignored (E forced to 1 T_{PBCLK}).

13.3.8.3 Full Multiplexed (8-Bit Bus) Address and Data Timing

This timing diagram illustrates full multiplexed timing (address bits <15:0> multiplexed with data bus, PMD<7:0>) for a read operation with no Wait states. A read operation requires 8 T_{PBCLK}, peripheral-bus-clock cycles.

Figure 13-21: 8-Bit, 16-Bit Read Operations, ADRMUX = 10, No Wait States



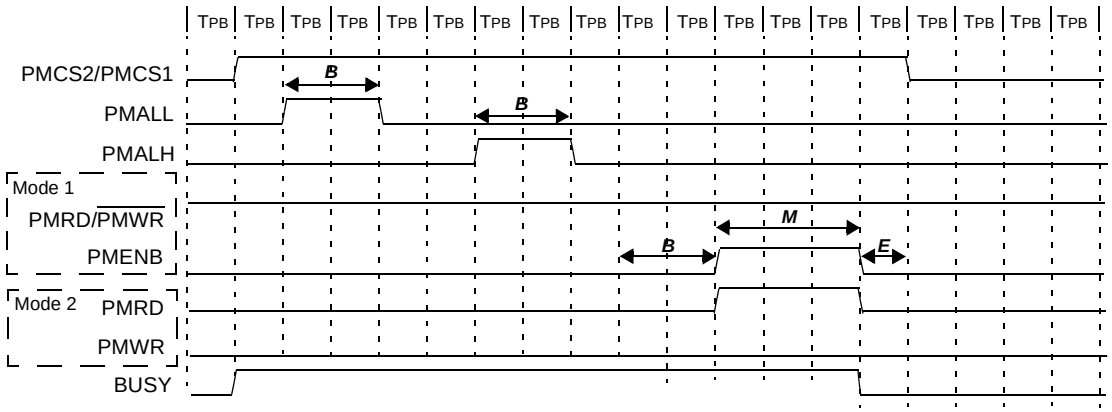
Note 1: Read data obtained from the PMDIN register is actually the value from the previous read operation.

2: In 8-bit mode, PMD<15:8> are not implemented.

3: PMADDR Address bit A15 and A14 are forced to '0' if PMCS2 and/or PMCS1 are enabled as Chip Selects.

In this timing diagram with Wait states, the read operation requires 14 T_{PBCLK}, peripheral-bus-clock cycles.

Figure 13-22: 8-Bit, 16-Bit Read Operation, ADRMUX = 10, Wait States Enabled



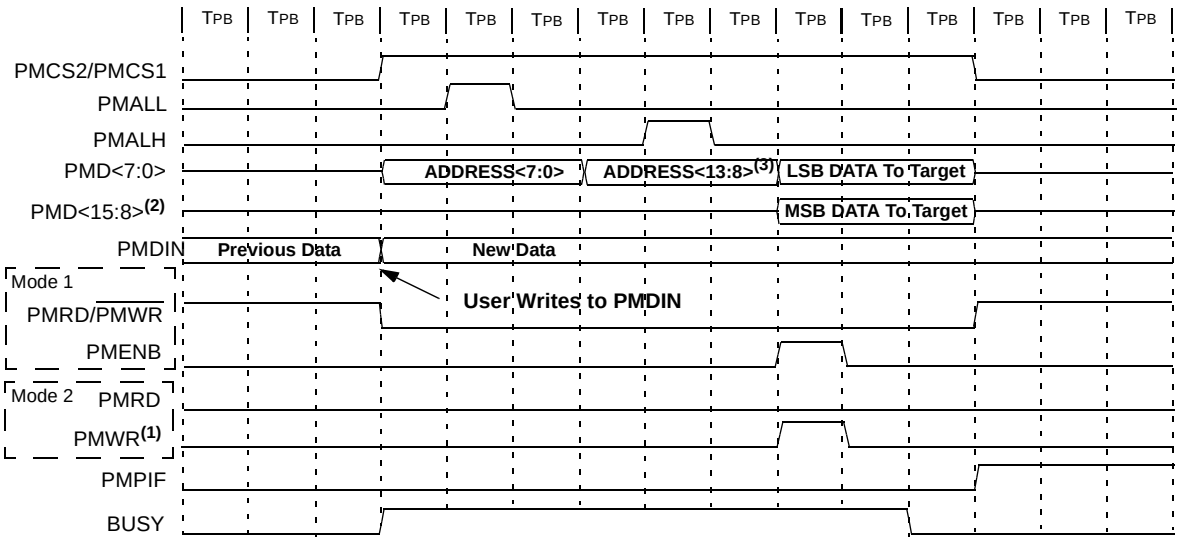
Legend:

B = WAITB<1:0> = 01 (2 Wait states)
M = WAITM<3:0> = 0010 (3 Wait states)
E = WAITE<1:0> = 01 (1 Wait state)

Note: If WAITM<3:0> = 0000, M is forced to 1 T_{PBCLK}, WAITB is ignored (B forced to 1 T_{PBCLK}) and WAITE is ignored (E forced to 0 T_{PBCLK}).

This timing diagram illustrates full multiplexed timing (address bits <15:0> multiplexed with data bus, PMD<7:0>) for a write operation with no Wait states. A write operation requires 9 T_{PBCLK}, peripheral-bus-clock cycles.

Figure 13-23: 8-Bit, 16-Bit Write, ADRMUX = 10, No Wait States



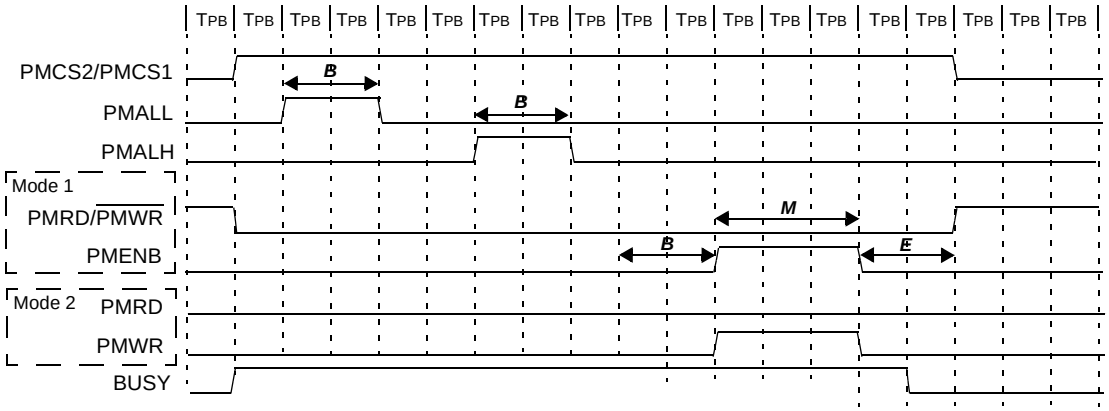
Note 1: During a write operation, there is one TPBCLK hold cycle following the PMWR signal.

2: In 8-bit mode, PMD<15:8> are not implemented.

3: PMADDR Address bit A15 and A14 are forced to '0' if PMCS2 and/or PMCS1 are enabled as Chip Selects.

In this timing diagram with Wait states, the write operation requires 15 T_{PBCLK}, peripheral-bus-clock cycles.

Figure 13-24: 8-Bit, 16-Bit Write Operations, ADRMUX = 10, Wait States Enabled



Legend:

B = WAITB<1:0> = 01 (2 Wait states)

M = WAITM<3:0> = 0010 (3 Wait states)

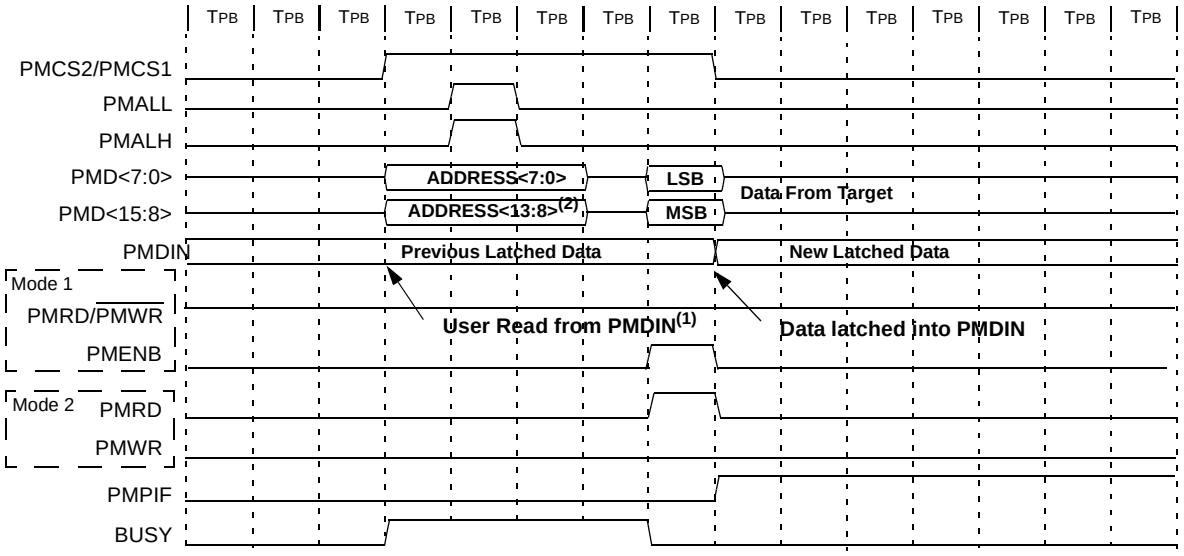
E = WAITE<1:0> = 01 (2 Wait states)

Note: If WAITM<3:0> = 0000, M is forced to 1 T_{PBCLK}, WAITB is ignored (B forced to 1 T_{PBCLK}), and WAITE is ignored (E forced to 1 T_{PBCLK}).

13.3.8.4 Full Multiplexed (16-Bit Bus) Address and Data Timing

This timing diagram illustrates full multiplexed timing (address bits <15:0> multiplexed with data bus, PMD<15:0>) for a read operation with no Wait states. A read operation requires 5 T_{PBCLK}, peripheral-bus-clock cycles.

Figure 13-25: 16-Bit Read Operation, ADRMUX = 11, No Wait States

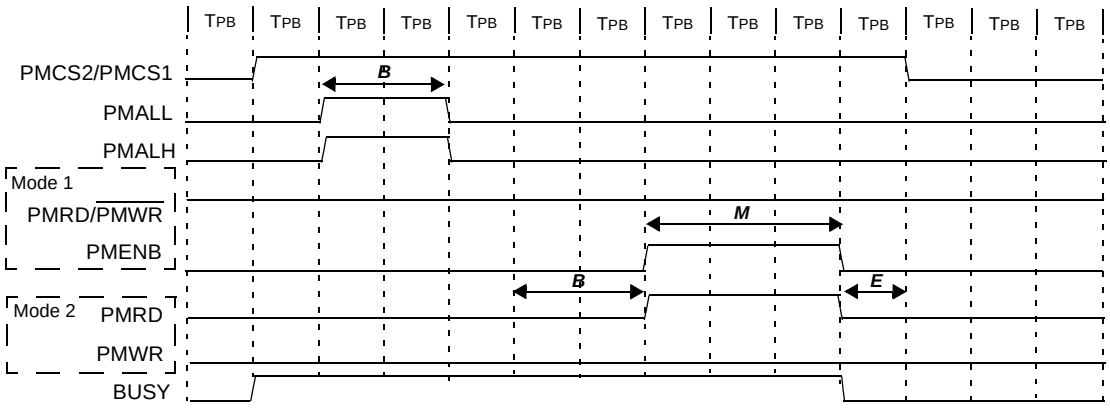


Note 1: Read data obtained from the PMDIN register is actually the value from the previous read operation.

2: PMADDR Address bit A15 and A14 are forced to '0' if PMCS2 and/or PMCS1 are enabled as Chip Selects.

In this timing diagram with Wait states, the read operation requires 10 T_{PBCLK}, peripheral-bus-clock cycles.

Figure 13-26: 16-Bit Read Operation, ADRMUX = 11, Wait States Enabled



Legend:

B = WAITB<1:0> = 01 (2 Wait states)

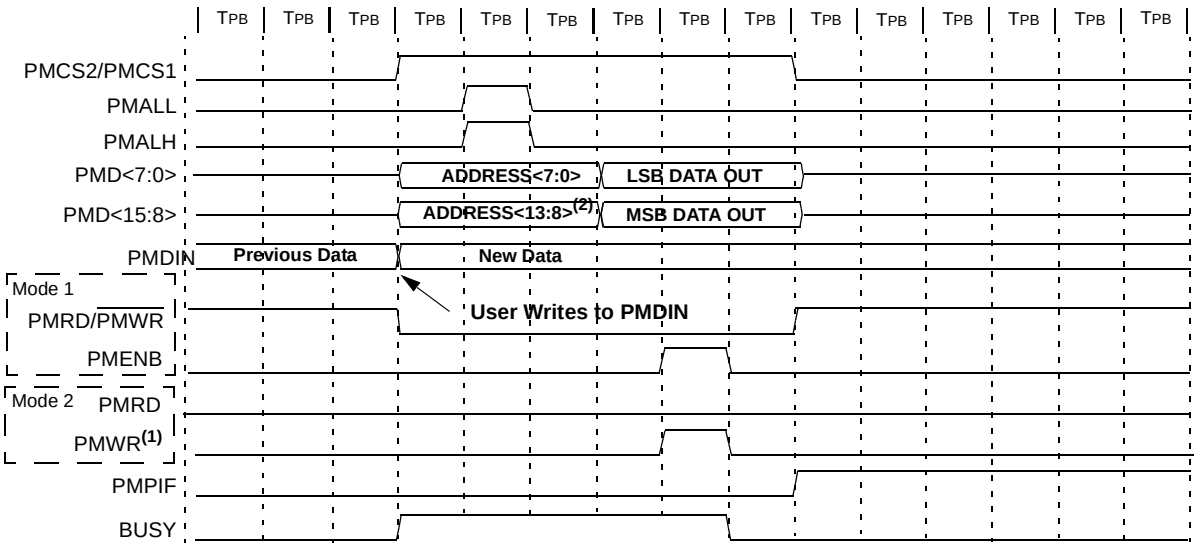
M = WAITM<3:0> = 0010 (3 Wait states)

E = WAITE<1:0> = 01 (1 Wait state)

Note: If WAITM<3:0> = 0000, M is forced to 1 T_{PBCLK}, WAITB is ignored (B forced to 1 T_{PBCLK}) and WAITE is ignored (E forced to 0 T_{PBCLK}).

This timing diagram illustrates full multiplexed timing (address bits <15:0> multiplexed with data bus, PMD<15:0>) for a read operation with no Wait states. A read operation requires 6 T_{PBCLK}, peripheral-bus-clock cycles.

Figure 13-27: 16-Bit Write, ADRMUX = 11, No Wait States

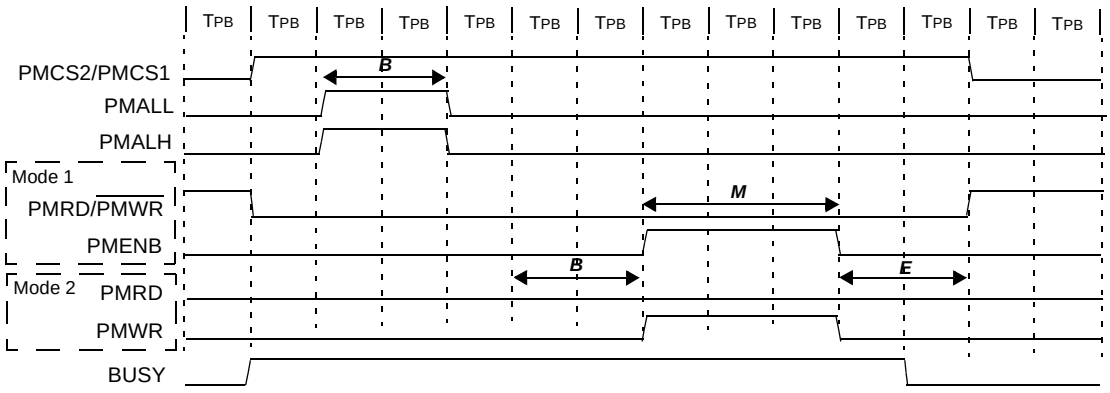


Note 1: During a write operation, there is one T_{PB} hold cycle following the PMWR signal.

2: PMADDR Address bit A15 and A14 are forced to '0' if PMCS2 and/or PMCS1 are enabled as Chip Selects.

In this timing diagram with Wait states, the write operation requires 11 T_{PBCLK}, peripheral-bus-clock cycles.

Figure 13-28: 16-Bit Write Operations, ADRMUX = 11, Wait States Enabled



Legend:

B = WAITB<1:0> = 01 (2 Wait states)

M = WAITM<3:0> = 0010 (3 Wait states)

E = WAITE<1:0> = 01 (2 Wait states)

Note: If WAITM<3:0> = 0000, M is forced to 1 T_{PBCLK}, WAITB is ignored (B forced to 1 T_{PBCLK}), and WAITE is ignored (E forced to 1 T_{PBCLK}).

13.4 SLAVE MODES OF OPERATION

The PMP module provides 8-Bit (byte) legacy PSP (Parallel Slave Port) functionality as well as new Buffered and Addressable slave modes.

Table 13-7: Slave Mode Selection

Slave Mode	PMCON MODE bits <1:0>	PMMODE INCM bits <1:0>
Legacy	00	x = don't care
Buffered	00	'11'
Addressable	01	x = don't care

All slave modes support 8-bit data only and the module control pins are automatically dedicated when any of these modes are selected. The user only need to configure the polarity of the PMCS1, PMRD and PMWR signals.

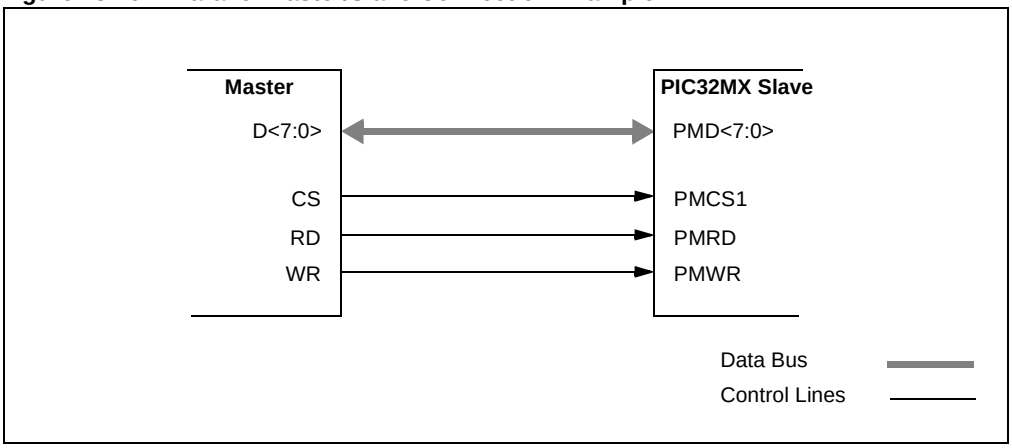
Table 13-8: Slave Mode Pin Polarity Configuration

CONTROL PIN	PMCON Control Bit	Active-High Select	Active-Low Select
PMRD	RDSP	1	0
PMWR	WRSP	1	0
PMCS1	CS1P	1	0

13.4.1 Legacy Slave Port Mode

In 8-bit PMP Legacy Slave mode, the module is configured as a parallel slave port using control bits MODE<1:0> (PMMODE<9:8>) = 00. In this mode, an external device such as another microcontroller or microprocessor can asynchronously read and write data using the 8-bit data bus PMD<7:0>, the read PMRD, write PMWR, and Chip Select PMCS1 inputs.

Figure 13-29: Parallel Master/Slave Connection Example



13.4.1.1 Initialization Steps

The following Slave mode initialization properly prepares the PMP port for communicating with an external device.

1. Clear control bit ON (PMCON<15>) = 0 to disable PMP module.
2. Select the Legacy mode with MODE<1:0> (PMMODE<9:8>) = 00.
3. Select the polarity of the Chip Select pin CS1P (PMCON<3>).
4. Select the polarity of the control pins WRSP and RDSP (PMCON<1:0>).

5. If interrupts are used:
   a. Clear interrupt flag bit PMPIF (IFS1<2>) = 0.
   b. Configure the PMP interrupt priority bits PMPIP<2:0> (IPC7<4:2>) and interrupt sub-priority bits PMPIS (IPC7<1:0>).
   c. Enable PMP interrupt by setting interrupt enable bit PMPIE (IEC1<2>) = 1.
6. Set control bit ON = 1 to enable PMP module.

Example 13-3: Example Code: Legacy Parallel Slave Port Initialization

```
/*
   Example Configuration for Legacy Slave mode
*/
   IEC1CLR  = 0x0004     // Disable PMP interrupt in case it is already enabled
   PMCON    = 0x0000     // Stop and Configure PMCON register for Legacy mode
   PMMODE   = 0x0000     // Configure PMMODE register
   IPC7SET  = 0x001C;    // Set priority level = 7 and
   IPC7SET  = 0x0003;    // Set subpriority level = 3
                         // Could have also done this in single
                         // operation by assigning IPC7SET = 0x001F
   IFS1CLR  = 0x0004;    // Clear the PMP interrupt status flag
   IEC1SET  = 0x0004;    // Enable PMP interrupts
   PMCONSET = 0x8000;    // Enable PMP module
```

13.4.1.2 Write to Slave Port

When Chip Select is active and a write strobe occurs, the data on the bus pins PMD<7:0> is captured into the lower 8 bits of the PMDIN register, PMDIN<7:0>. The PMPIF (interrupt flag bit) is set during the write strobe, however, IB0F (input buffer full flag) bit requires 2 to 3 peripheral-bus-clock cycles to synchronize before it is set and the PMDIN register can be read. The IB0F bit will remain set until the PMDIN register is read by the user. If a write operation occurs while IB0F bit is = 1, the write data will be ignored and an overflow condition will be generated, IB0V = 1.

Refer to timing diagrams in **Section 13.4.4**.

13.4.1.3 Read from Slave Port

When Chip Select is active and a read strobe occurs, the data from the lower 8 bits of the PMDOUT register, PMDOUT<7:0> is presented onto data bus pins PMD<7:0> and read by the master device. The PMPIF (interrupt flag bit) is set during the read strobe, however, OB0E (output buffer empty flag) bit requires 2 to 3 peripheral-bus-clock cycles to synchronize before it is set. The OB0E bit will remain set until the PMDOUT register is written to by the user. If a read operation occurs while OB0E bit is = 1, the read data will be the same as the previous read data and an underflow condition will be generated, OBUF = 1.

Refer to timing diagrams in **Section 13.4.4**.

13.4.1.4 Legacy Mode Interrupt Operation

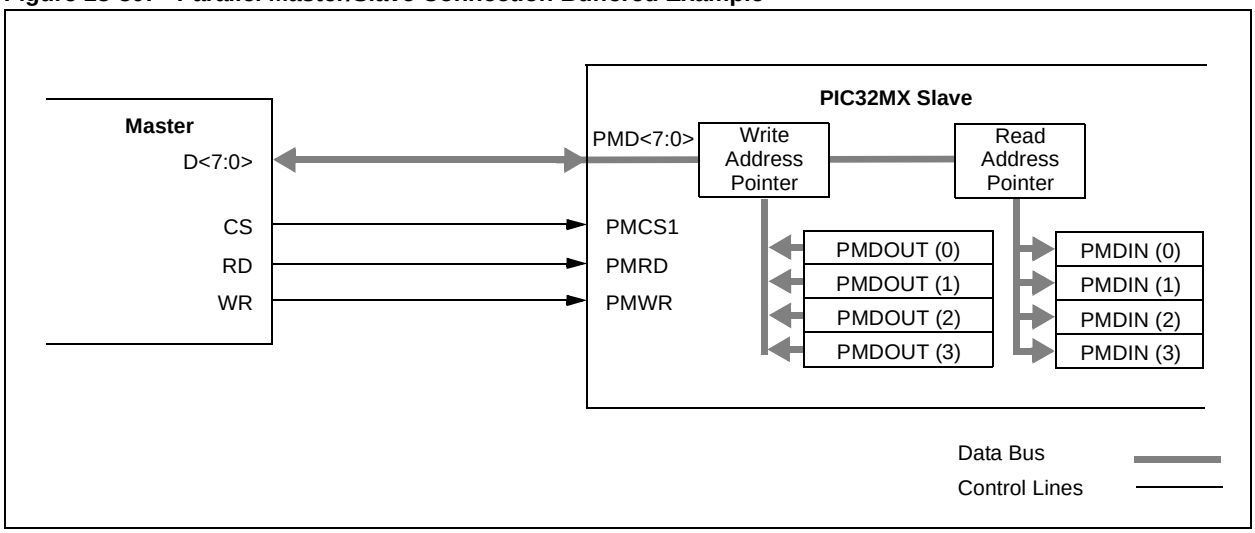
In PMP Legacy Slave mode, the PMPIF bit is set every read or write strobe. If using interrupts, the user's application vectors to an Interrupt Service Routine (ISR) where the IBF and OBE Status bits can be examined to determine if the buffer is full or empty. If not using interrupts, the user's application should wait for PMPIF to be set before polling the IBF and OBE Status bits to determine if the buffer is full or empty.

13.4.2 Buffered Parallel Slave Port Mode

The 8-bit Buffered Parallel Slave Port mode is functionally identical to the Legacy Parallel Slave Port mode with one exception: the implementation of 4-level read and write buffers. Buffered Slave mode is enabled by setting the PMMODE<MODE1:MODE0>) bits = 00, and the PMMODE<INCM1:INCM0> bits = 11.

When the buffered mode is active, the module uses the PMDIN register as write buffers and the PMDOUT register as read buffers. Each register is divided into four 8-bit buffer registers, four read buffers in PMDOUT and four write buffers in PMDIN. Buffers are numbered 0 through 3, starting with the lower byte <7:0> and progressing upward through the high byte <31:24>.

Figure 13-30: Parallel Master/Slave Connection Buffered Example



13.4.2.1 Initialization Steps

The following Buffered Slave mode initialization properly prepares the PMP port for communicating with an external device.

1. Clear control bit ON (PMCON<15>) = 0 to disable PMP module.
2. Select the Legacy mode with MODE<1:0> (PMMODE<9:8>) = 00
3. Select Buffer mode with INCM<1:0> (PMMODE<12:11>) = 11.
4. Select the polarity of the Chip Select CS1P (PMCON<3>).
5. Select the polarity of the control pins with WRSP and RDSP (PMCON<1:0>).
6. If interrupts are used:
   a. Clear interrupt flag bit PMPIF (IFS1<2>).
   b. Configure interrupt priority and subpriority levels in IPC7.
   c. Set interrupt enable bit PMPIE (IEC1<2>).
7. Set control bit ON = 1 to enable PMP module.

Example 13-4: Example Code: Buffered Parallel Slave Port Initialization

```
/*
    Example Configuration for Buffered Slave mode
*/
    IEC1CLR = 0x0004      // Disable PMP interrupt in case it is already enabled
    PMCON = 0x0000        // Stop and Configure PMCON register for Buffered mode
    PMMODE = 0x1800       // Configure PMMODE register
    IPC7SET = 0x001C;     // Set priority level = 7 and
    IPC7SET = 0x0003;     // Set subpriority level = 3
                          // Could have also done this in single operation by assigning
                          // IPC7SET = 0x001F
    IFS1CLR  = 0x0004;    // Clear the PMP interrupt status flag
    IEC1SET  = 0x0004;    // Enable PMP interrupts
    PMCONSET = 0x8000;    // Enable PMP module
```

13.4.2.2 Read from Slave Port

For read operations, the bytes will be sent out sequentially, starting with Buffer 0, PMDOUT<7:0>, and ending with Buffer 3, PMDOUT<31:24>, for every read strobe. The module maintains an internal pointer to keep track of which buffer is to be read.

Each of the buffers has a corresponding read Status bit, OBnE, in the PMSTAT register. This bit is cleared when a buffer contains data that has not been written to the bus, and is set when data is written to the bus. If the current buffer location being read from is empty, a buffer underflow is generated, and the Buffer Overflow flag bit OBUF is set. If all four OBnE Status bits are set, then the Output Buffer Empty flag OBE will also be set.

Refer to timing diagrams in **Section 13.4.4**.

13.4.2.3 Write to Slave Port

For write operations, the data is be stored sequentially, starting with Buffer 0, PMDIN<7:0> and ending with Buffer 3, PMDIN<31:24>. As with read operations, the module maintains an internal pointer to the buffer that is to be written next.

The input buffers have their own write Status bits, IBnf. The bit is set when the buffer contains unread incoming data, and cleared when the data has been read. The flag bit is set on the write strobe. If a write occurs on a buffer when its associated IBnf bit is set, the Buffer Overflow flag IBOV is set; any incoming data in the buffer will be lost. If all 4 IBnf flags are set, the Input Buffer Full flag IBF is set.

Refer to timing diagrams in **Section 13.4.4**.

13.4.2.4 Buffered Mode Interrupt Operation

In Buffered Slave mode, the module can be configured to generate an interrupt on every read or write strobe, IRQM<1:0> (PMMODE<14:13>) = 01. It can be configured to generate an interrupt on a read from Read Buffer 3 or a write to Write Buffer 3, IRQM<1:0> = 10, which is essentially an interrupt every fourth read or write strobe. When interrupting every fourth byte for input data, all input buffer registers should be read to clear the IBnF flags. If these flags are not cleared then there is a risk of hitting an overflow condition.

If using interrupts, the user's application vectors to an Interrupt Service Routine (ISR) where the IBF and OBE Status bits can be examined to determine if the buffer is full or empty. If not using interrupts, the user's application should wait for PMPIF to be set before polling the IBF and OBE Status bits to determine if the buffer is full or empty.

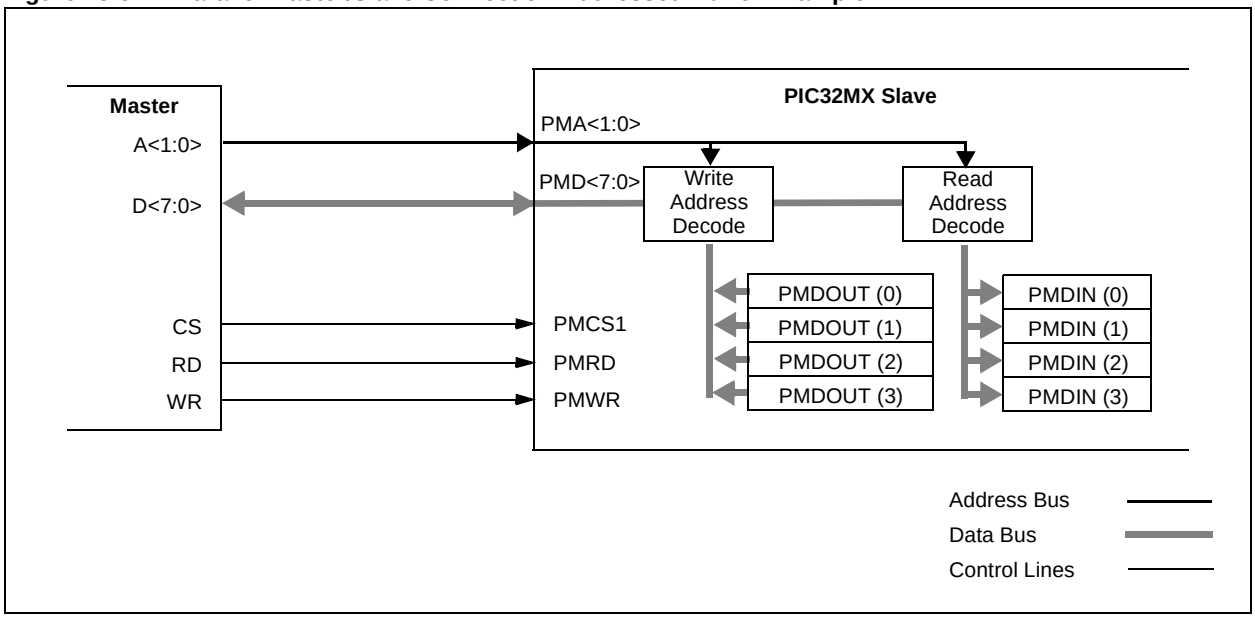
13.4.3 Addressable Buffered Parallel Slave Port Mode

In the 8-bit Addressable Buffered Parallel Slave Port mode the module is configured with two extra inputs, PMA<1:0>. This makes the 4-byte buffer space directly addressable as fixed pairs of read and write buffers. As with Buffered Legacy mode, data is output from register PMDOUT and is input to register PMDIN. Table 20-1 shows the address resolution for the incoming address to the input and output registers.

Table 13-9: Slave Mode Buffer Addresses

PMA<1:0>	Output Register (Buffer)	Input Register (Buffer)
00	PMDOUT<7:0>(0)	PMDIN<7:0> (0)
01	PMDOUT<15:8> (1)	PMDIN<15:8> (1)
10	PMDOUT<23:16> (2)	PMDIN<23:16> (2)
11	PMDOUT<31:24> (3)	PMDIN<31:24> (3)

Figure 13-31: Parallel Master/Slave Connection Addressed Buffer Example



13.4.3.1 Initialization Steps

The following Addressable Buffered Slave mode initialization properly prepares the PMP port for communicating with an external device.

1. Clear control bit ON (PMCON<15>) = 0 to disable PMP module.
2. Select the Legacy mode with MODE<1:0> (PMMODE<9:8) = 00.
3. Select the polarity of the Chip Select CS1P (PMCON<3>).
4. Select the polarity of the control pins with WRSP and RDSP (PMCON<1:0>).
5. If interrupts are used:
   a. Clear interrupt flag bit PMPIF (IFS1<2>).
   b. Configure interrupt priority and subpriority levels in IPC7.
   c. Set interrupt enable bit PMPIE (IEC1<2>).
6. Set control bit ON = 1 to enable PMP module.

Example 13-5: Example Code: Addressable Parallel Slave Port Initialization

```
/*
    Example Configuration for Addressable Slave mode
*/
    IEC1CLR = 0x0004      // Disable PMP interrupt in case it is already enabled
    PMCON = 0x0000        // Stop and Configure PMCON register for Address mode
    PMMODE = 0x0100       // Configure PMMODE register
    IPC7SET = 0x001C;     // Set priority level = 7 and
    IPC7SET = 0x0003;     // Set subpriority level = 3
                          // Could have also done this in single operation by assigning
                          // IPC7SET = 0x001F
    IFS1CLR  = 0x0004;    // Clear the PMP interrupt status flag
    IEC1SET  = 0x0004;    // Enable PMP interrupts
    PMCONSET = 0x8000;    // Enable PMP module
```

13.4.3.2 Read from Slave Port

When Chip Select is active and a read strobe occurs, the data from one of the four output 8-bit buffers is presented onto PMD<7:0>. The byte selected to be read depends on the 2-bit address placed on PMA<1:0>. Table 20-1 shows the corresponding output registers and their associated address. When an output buffer is read, the corresponding OBnE bit is set. The OBE flag bit is set when all the buffers are empty. If any buffer is already empty, OBnE = 1, the next read to that buffer will generate an OBUF event.

Refer to timing diagrams in **Section 13.4.4**.

13.4.3.3 Write to Slave Port

When Chip Select is active and a write strobe occurs (PMCS = 1 and PMWR = 1), the data from PMD<7:0> is captured into one of the four input buffer bytes. The byte selected to be written depends on the 2-bit address placed on ADDR<1:0>. Table 20-1 shows the corresponding input registers and their associated address.

When an input buffer is written, the corresponding IBnF bit is set. The IBF flag bit is set when all the buffers are written. If any buffer is already written, IBnF = 1, the next write strobe to that buffer will generate an IBOV event, and the byte will be discarded.

Refer to timing diagrams in **Section 13.4.4**.

13.4.3.4 Addressable Buffered Mode Interrupt Operation

In Addressable Slave mode, the module can be configured to generate an interrupt on every read or write strobe, IRQM<1:0> (PMMODE<14:13>) = 01. It can also be configured to generate an interrupt on any read from Read Buffer 3 or write to Write Buffer 3, IRQM<1:0> = 10; in other words, an interrupt will occur whenever a read or write occurs when PMA<1:0> is '11'.

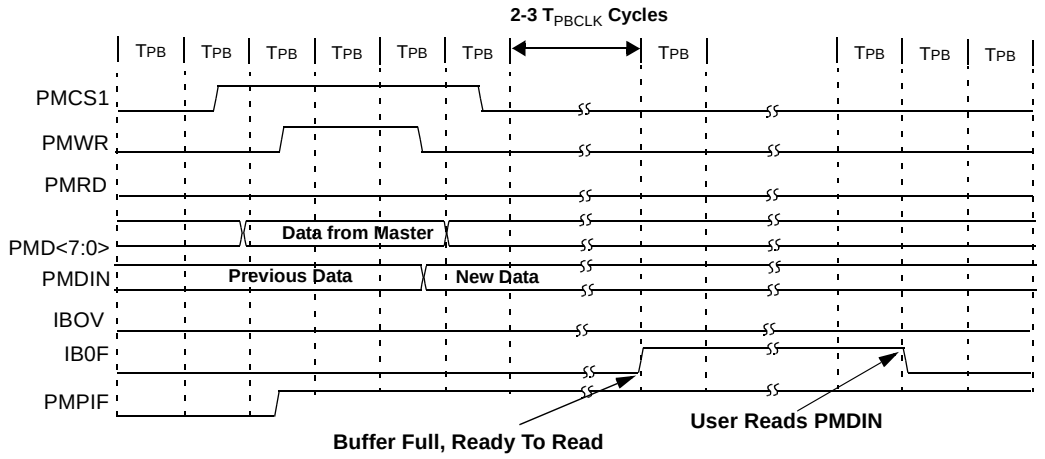
If using interrupts, the user's application vectors to an Interrupt Service Routine (ISR) where the IBF and OBE Status bits can be examined to determine if the buffer is full or empty. If not using interrupts, the user's application should wait for PMPIF to be set before polling the IBF and OBE Status bits to determine if the buffer is full or empty.

13.4.4 Slave Mode Read and Write Timing Diagrams

In all of the slave modes, an external master device is connected to the parallel slave port and is controlling the read and write operations. When an external read or write operation is performed by the external master device, the PMPIF (IFS1<2>) will be set on the active edge of PMRD or PMWR pin.

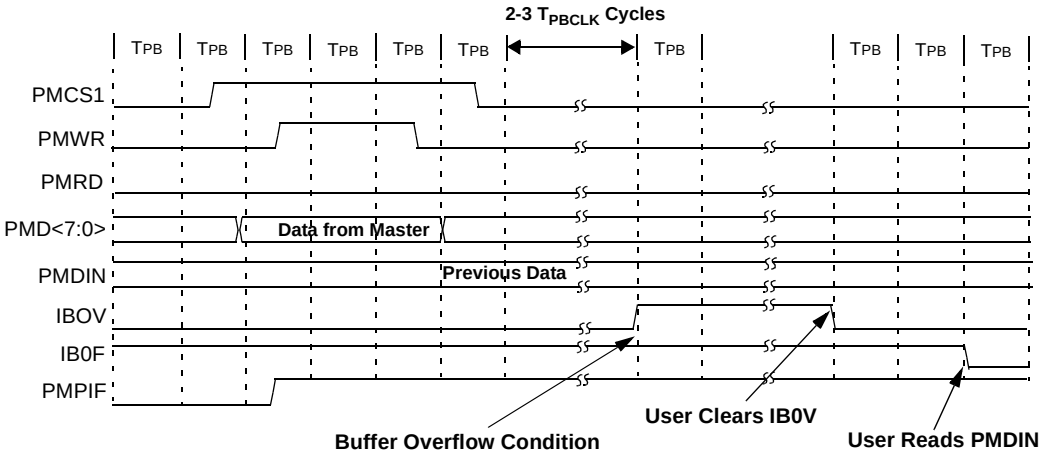
- For any external write operation, the user's application must poll the IBOV or IB0F buffer Status bits to ensure adequate time for the write operation to be completed before accessing the PMDIN register.
- For any external read operation, the user's application must poll the OBUF or OB0E buffer Status bits to ensure adequate time for the read operation to be completed before accessing PMDOUT register.

Figure 13-32: Parallel Slave Port Write Operation



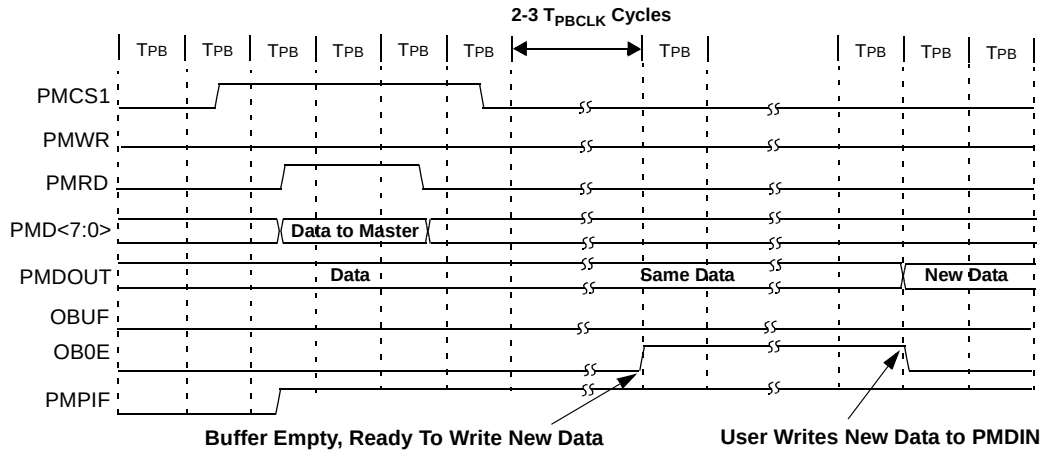
Note: Control signal polarity are configurable and are shown active-high in this example.

Figure 13-33: Parallel Slave Port Write Operation – Buffer Full, Overflow Condition



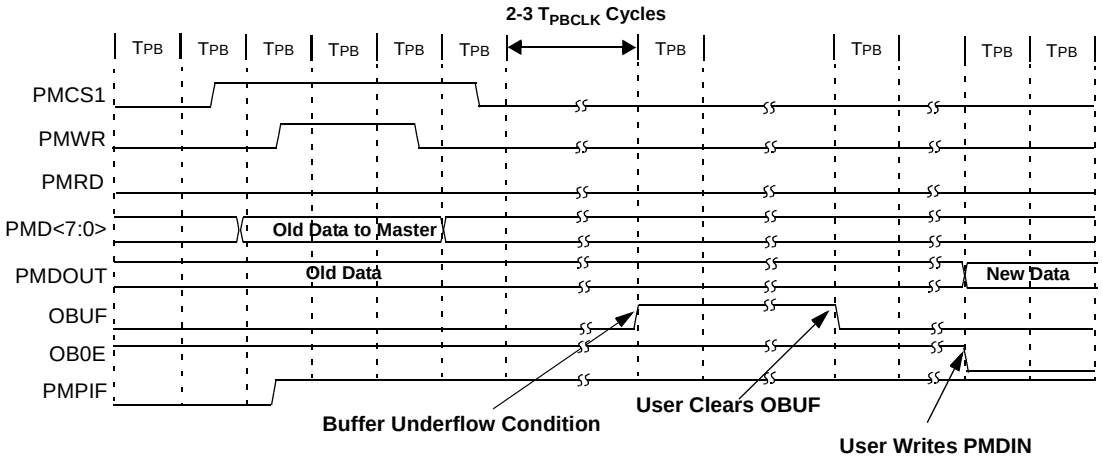
Note: Control signal polarity are configurable and are shown active-high in this example.

Figure 13-34: Parallel Slave Port Read Operation



Note: Control signal polarity are configurable and are shown active-high in this example.

Figure 13-35: Parallel Slave Port Read Operation – Buffer Empty, Underflow Condition



Note: Control signal polarity are configurable and are shown active-high in this example.

13.5 INTERRUPTS

The Parallel Master Port has the ability to generate an interrupt, depending on the selected Operating mode.

- PMP (Master) mode:
  - Interrupt on every completed read or write operation.
- PSP (Legacy Slave) mode:
  - Interrupt on every read and write byte
- PSP (Buffered Slave) mode:
  - Interrupt on every read and write byte
  - Interrupt on read or write byte of Buffer 3 (PMDOUT<31:24>)
- EPSP (Enhanced Addressable Slave) mode:
  - Interrupt on every read and write byte
  - Interrupt on read or write byte of Buffer 3 (PMDOUT<31:24>), PMA<1:0> = 11.

The PMPIF bit must be cleared in software.

The PMP module is enabled as a source of interrupt via the PMP Interrupt Enable bit, PMPIE. The Interrupt Priority level bits (PMPIP<2:0>) and Interrupt Subpriority level bits (PMPIS<1:0>) must also be configured. Refer to **Section Section 1. "Interrupts"** for further details.

13.5.1 Interrupt Configuration

The PMP module has a dedicated interrupt flag bit PMPIF and a corresponding interrupt enable/mask bit PMPIE. These bits are used to determine the source of an interrupt and to enable or disable an individual interrupt source.

The PMPIE bit is used to define the behavior of the Vector Interrupt Controller or Interrupt Controller when the PMPIF is set. When the PMPIE bit is clear, the Interrupt Controller module does not generate a CPU interrupt for the event. If the PMPIE bit is set, the Interrupt Controller module will generate an interrupt to the CPU when the PMPIF bit is set (subject to the priority and subpriority as outlined below).

It is the responsibility of the user's software routine that services a particular interrupt to clear the appropriate Interrupt Flag bit before the service routine is complete.

The priority of PMP module can be set with the PMPIP<2:0> bits. This priority defines the priority group to which the interrupt source will be assigned. The priority groups range from a value of 7, the highest priority, to a value of 0, which does not generate an interrupt. An interrupt being serviced will be preempted by an interrupt in a higher priority group.

The subpriority bits allow setting the priority of a interrupt source within a priority group. The values of the subpriority, PMPIS<1:0>, range from 3, the highest priority, to 0 the lowest priority. An interrupt with the same priority group but having a higher subpriority value will preempt a lower subpriority interrupt that is in progress.

The priority group and subpriority bits allow more than one interrupt source to share the same priority and subpriority. If simultaneous interrupts occur in this configuration the natural order of the interrupt sources within a priority/subgroup pair determine the interrupt generated. The natural priority is based on the vector numbers of the interrupt sources. The lower the vector number the higher the natural priority of the interrupt. Any interrupts that were overridden by natural order will then generate their respective interrupts based on priority, subpriority, and natural order after the interrupt flag for the current interrupt is cleared.

After an enabled interrupt is generated, the CPU will jump to the vector assigned to that interrupt. The vector number for the interrupt is the same as the natural order number. The CPU will then begin executing code at the vector address. The user's code at this vector address should perform any application specific operations and clear the PMPIF interrupt flag, and then exit. Refer to the Interrupt Controller chapter for the vector address table details for more information on interrupts.

Table 13-10: PMP Interrupt Vectors for Various Offsets with EBASE = 0x8000:0000

Channel	Vector/Natural Order	IRQ Number	Vector Address IntCtl.VS = 0x01	Vector Address IntCtl.VS = 0x02	Vector Address IntCtl.VS = 0x04	Vector Address IntCtl.VS = 0x08	Vector Address IntCtl.VS = 0x10
PMP	28	34	8000 0580	8000 0900	8000 1000	8000 1E00	8000 3A00

Table 13-11: Priority and Subpriority Assignment Example

Channel	Priority Group	Subpriority	Vector/Natural Order
PMP	7	3	28

Example 13-6: PMP Module Interrupt Initialization Code Example

```
/*
    The following code example illustrates a PMP interrupt configuration.
    When the PMP interrupt is generated, the cpu will branch to the vector assigned to PMP
    interrupt.
*/

    // Configure PMP for desired mode of operation
    ...
    // Configure the PMP interrupts
    IPC7SET = 0x0014;      // Set priority level = 5
    IPC7SET = 0x0003;      // Set subpriority level = 3
                           // Could have also done this in single
                           // operation by assigning IPC7SET = 0x0017

    IFS1CLR  = 0x0004;     // Clear the PMP interrupt status flag
    IEC1SET  = 0x0004;     // Enable PMP interrupts
    PMCONSET = 0x8000;     // Enable PMP module
```

Example 13-7: PMP ISR Code Example

```
/*
    The following code example demonstrates a simple Interrupt Service Routine for PMP
    interrupts. The user's code at this vector should perform any application specific
    operations and must clear the PMP interrupt status flag before exiting.
*/

void __ISR(_PMP_VECTOR, ipl5) PMP_HANDLER(void)
{
    ... perform application specific operations in response to the interrupt

    IFS1CLR  = 0x0004;     // Be sure to clear the PMP interrupt status
                           // flag before exiting the service routine.
}
```

Note: The PMP ISR code example shows MPLAB® C32 C compiler-specific syntax. Refer to your compiler manual regarding support for ISRs.

13.6 OPERATION IN POWER-SAVING AND DEBUG MODES

Note:	In this manual, a distinction is made between a power mode as it is used in a specific module, and a power mode as it is used by the device, e.g., Sleep mode of the Comparator and SLEEP mode of the CPU. To indicate which type of power mode is intended, uppercase and lowercase letters (Sleep, Idle, Debug) signify a module power mode, and all uppercase letters (SLEEP, IDLE, DEBUG) signify a device power mode.

13.6.1 PMP Operation in SLEEP Mode

When the device enters SLEEP mode, the system clock is disabled. The consequences of SLEEP mode depend on which mode the module is configured in at the time that SLEEP mode is invoked.

13.6.1.1 PMP Operation – SLEEP in Master Mode

If the microcontroller enters SLEEP mode while the module is operating in Master mode, PMP operation will be suspended in its current state until clock execution resumes. As this may cause unexpected control pin timings, users should avoid invoking SLEEP mode when continuous use of the module is needed.

13.6.1.2 PMP Operation – SLEEP in Slave Mode

While the module is inactive but enabled for any Slave mode operation, any read or write operations occurring at that time will be able to complete without the use of the microcontroller clock. Once the operation is completed, the module will issue an interrupt according to the setting of the IRQM bits.

If the PMPIE bit is set, and its priority is greater than current CPU priority, the device will wake from SLEEP or IDLE mode and execute the PMP interrupt service routine.

If the assigned priority level of the PMP interrupt is less than or equal to the current CPU priority level, the CPU will not be awakened and the device will enter the IDLE mode.

13.6.2 PMP Operation in IDLE Mode

When the device enters IDLE mode, the system clock sources remain functional. The PMCON<SIDL> bit selects whether the module will stop or continue functioning on IDLE. If PMCON<SIDL> = 0, the module will continue operation in IDLE mode.

If PMCON<SIDL> = 1, the module will stop communications when the microcontroller enters IDLE mode, in the same manner as it does in SLEEP mode. The current transaction in Slave modes will complete and issue an interrupt, while the current transaction in Master mode will be suspended until normal clocking resumes. As with SLEEP mode, IDLE mode should be avoided when using the module in Master mode if continuous use of the module is required.

Note:	The FRZ bit is readable and writable only when the CPU is executing in Debug Exception mode. In all other modes, the FRZ bit reads as '0'. If FRZ bit is changed during DEBUG mode, the new value does not take effect until the current Debug Exception mode is exited and re-entered. During the Debug Exception mode, the FRZ bit reads the state of the peripheral when entering DEBUG mode.

13.7 EFFECTS OF VARIOUS RESETS

13.7.1 Device Reset

All PMP module registers are forced to their Reset states on a device Reset.

13.7.2 Power-on Reset

All PMP module registers are forced to their Reset states on a POR.

13.7.3 Watchdog Reset

All PMP module registers are forced to their Reset states on a Watchdog Reset.

13.8 PARALLEL MASTER PORT APPLICATIONS

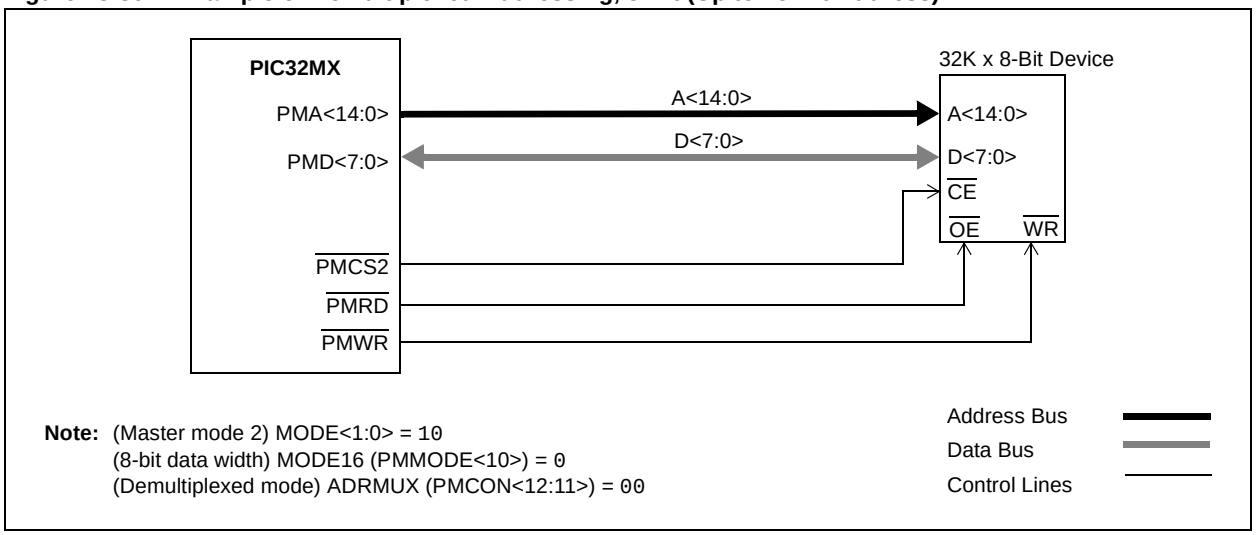
This section illustrates typical interfaces between the PMP module and external devices for each of the module's multiplexing modes. Additionally, there are some potential applications shown for the PMP module.

Note:	Data pins PMD<15:0> are available on 100-pin PIC32MX device variants and larger. For all other device variants, only pins PMD<7:0> are available. Refer to the specific PIC32MX device data sheet for details.

13.8.1 Demultiplexed Memory or Peripheral

Figure 13-36 illustrates the connections to an 8-bit memory or addressable peripheral in Demultiplexed mode. This mode does not require any external latches.

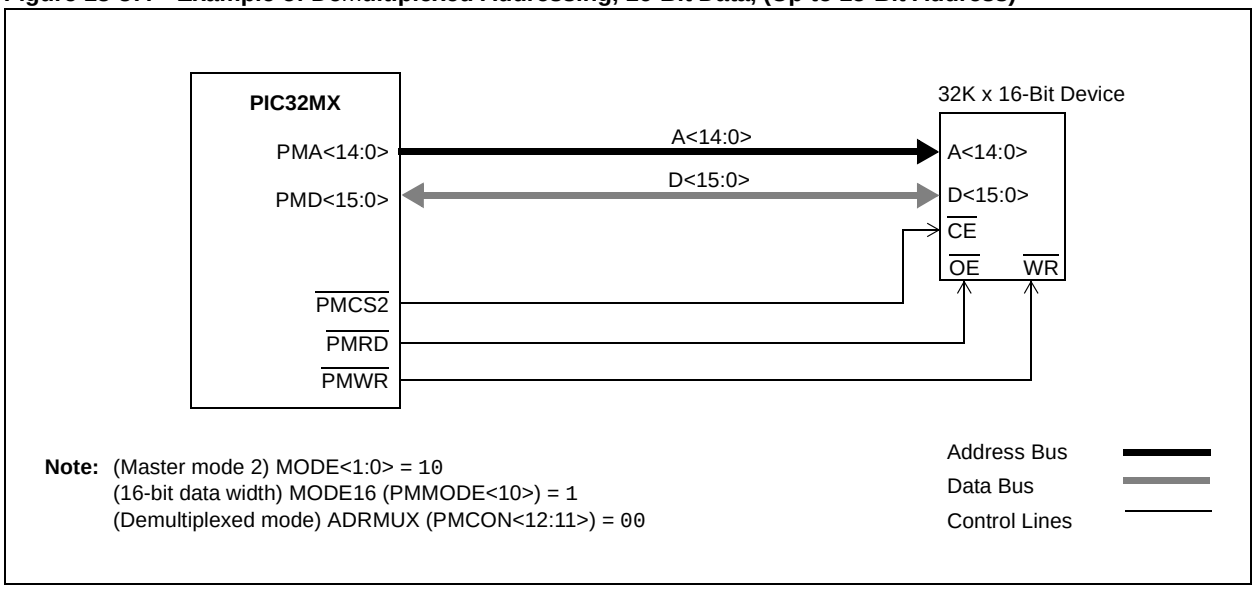
Figure 13-36: Example of Demultiplexed Addressing, 8-Bit (Up to 15-Bit Address)



Note: (Master mode 2) MODE<1:0> = 10
(8-bit data width) MODE16 (PMMODE<10>) = 0
(Demultiplexed mode) ADRMUX (PMCON<12:11>) = 00

Figure 13-37 illustrates the connections to a 16-bit memory or addressable peripheral in Demultiplexed mode. This mode does not require any external latches.

Figure 13-37: Example of Demultiplexed Addressing, 16-Bit Data, (Up to 15-Bit Address)

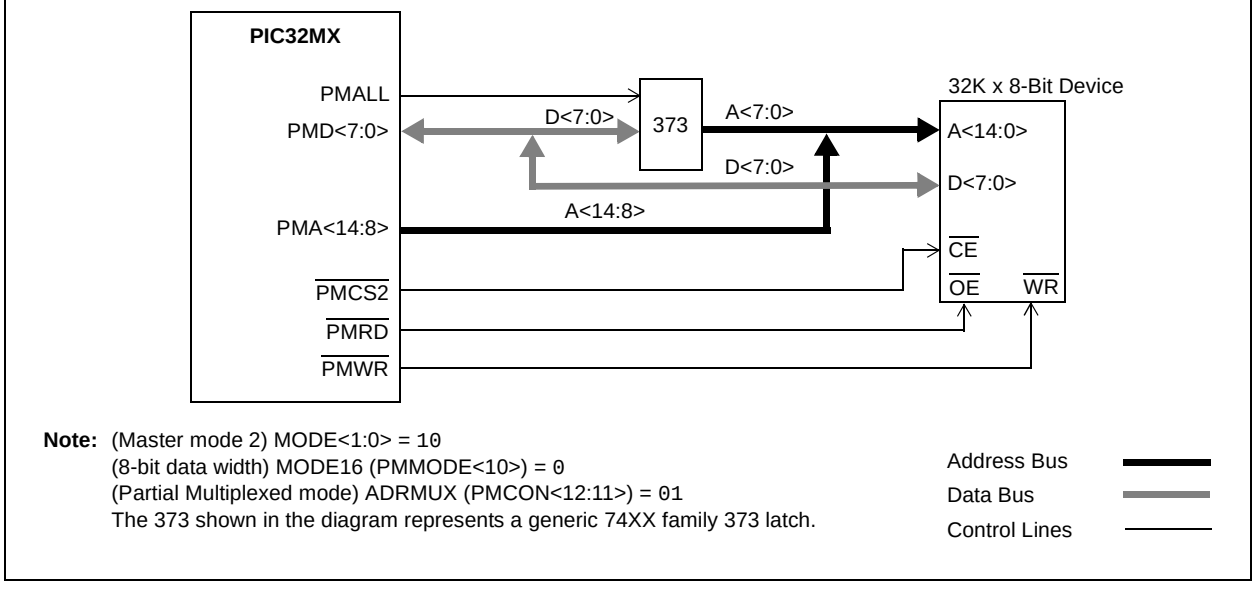


Note: (Master mode 2) MODE<1:0> = 10
(16-bit data width) MODE16 (PMMODE<10>) = 1
(Demultiplexed mode) ADRMUX (PMCON<12:11>) = 00

13.8.2 Partial Multiplexed Memory or Peripheral

Figure 13-38 illustrates the connections to an 8-bit memory or other addressable peripheral in Partial Multiplex mode. In this mode, an external latch is required. Consequently, from the microcontroller perspective, this mode achieves some pin savings over the Demultiplexed mode, however, at the price of performance. The lower 8 bits of address are multiplexed with the PMD<7:0> data bus and require one extra peripheral-bus-clock cycle.

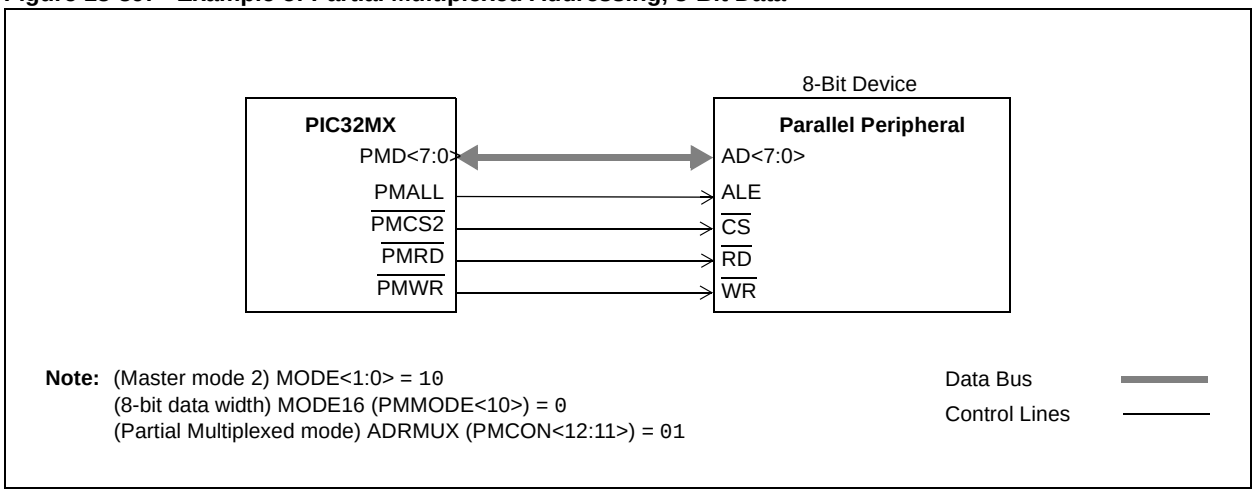
Figure 13-38: Example of Partial Multiplexed Addressing, 8-Bit Data (Up to 15-Bit Address)



Note: (Master mode 2) MODE<1:0> = 10
(8-bit data width) MODE16 (PMMODE<10>) = 0
(Partial Multiplexed mode) ADRMUX (PMCON<12:11>) = 01
The 373 shown in the diagram represents a generic 74XX family 373 latch.

If the peripheral has internal latches as shown in Figure 13-39, then no extra circuitry is required except for the peripheral itself.

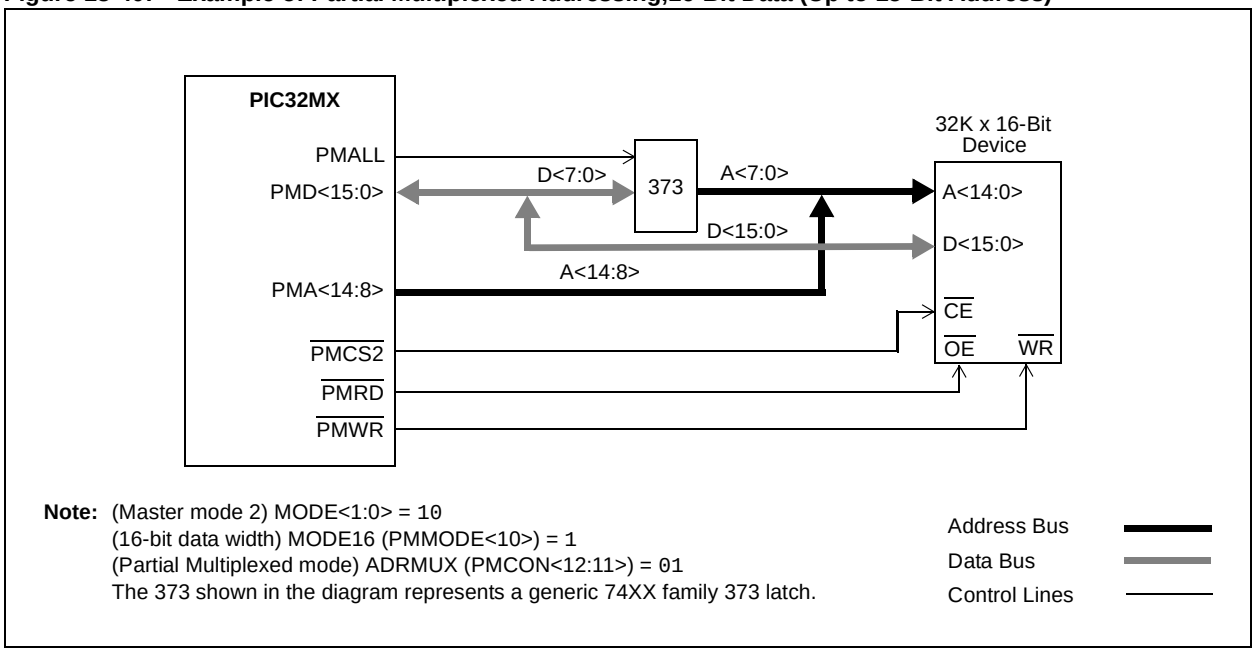
Figure 13-39: Example of Partial Multiplexed Addressing, 8-Bit Data



Note: (Master mode 2) MODE<1:0> = 10
(8-bit data width) MODE16 (PMMODE<10>) = 0
(Partial Multiplexed mode) ADRMUX (PMCON<12:11>) = 01

Figure 13-40 illustrates the connections to a 16-bit memory or other addressable peripheral in Partial Multiplex mode. In this mode, an external latch is required. Consequently, from the microcontroller perspective, this mode achieves some pin savings over the Demultiplexed mode, however, at the price of performance. The lower 8 bits of address are multiplexed with the PMD<7:0> data bus and require one extra peripheral-bus-clock cycle.

Figure 13-40: Example of Partial Multiplexed Addressing,16-Bit Data (Up to 15-Bit Address)

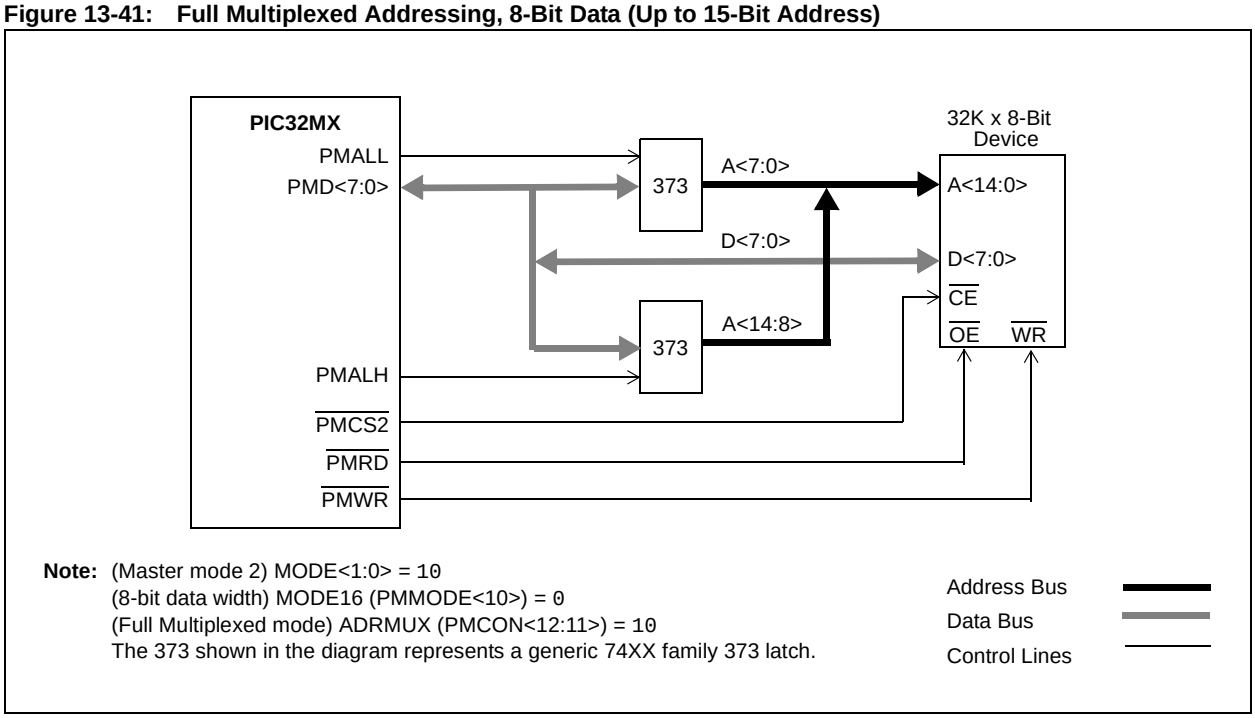


Note: (Master mode 2) MODE<1:0> = 10
(16-bit data width) MODE16 (PMMODE<10>) = 1
(Partial Multiplexed mode) ADRMUX (PMCON<12:11>) = 01
The 373 shown in the diagram represents a generic 74XX family 373 latch.

13.8.3 Full Multiplexed Memory or Peripheral

Figure 13-41 illustrates the connections to a memory or other addressable peripheral in full 8-bit Multiplexed mode, ADRMUX = 10 (PMCON<12:11>). Consequently, from the microcontroller perspective, this mode achieves the best pin saving over the Demultiplexed mode or Partially Multiplexed mode, however, at the price of performance. The lower 8 address bits are multiplexed with the PMD<7:0> data bus followed by the upper 6 or 7 address bits (if CS2, CS1 or both are enabled) and therefore require two extra peripheral-bus-clock cycles.

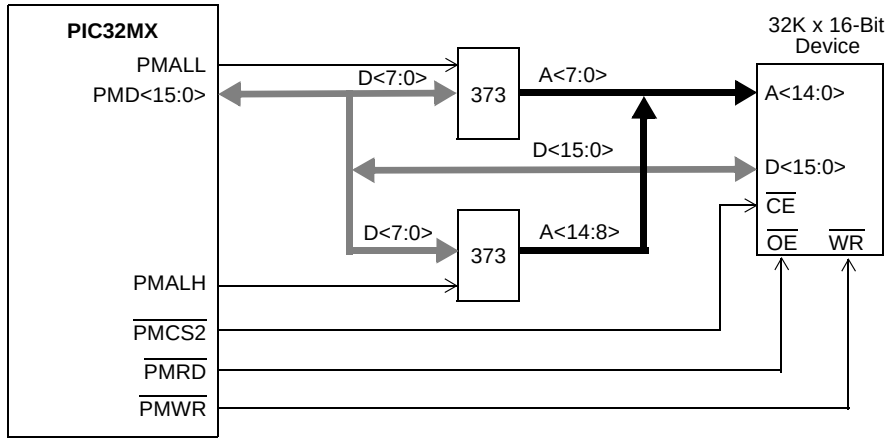
Figure 13-41: Full Multiplexed Addressing, 8-Bit Data (Up to 15-Bit Address)



Note: (Master mode 2) MODE<1:0> = 10
(8-bit data width) MODE16 (PMMODE<10>) = 0
(Full Multiplexed mode) ADRMUX (PMCON<12:11>) = 10
The 373 shown in the diagram represents a generic 74XX family 373 latch.

Figure 13-42 illustrates the connections to a 16-bit memory or other addressable peripheral in full 16-bit Multiplex mode, ADRMUX = 10 (PMCON<12:11>). Consequently, from the microcontroller perspective, this mode achieves the best pin saving over the Demultiplexed mode or Partially Multiplexed mode, however, at the price of performance. The lower 8 address bits are multiplexed with the PMD<7:0> data bus followed by the upper 6 or 7 address bits (if CS2, CS1 or both are enabled) and therefore require two extra peripheral-bus-clock cycles.

Figure 13-42: Full Multiplexed Addressing, 16-Bit Data (Up to 15-Bit Address)

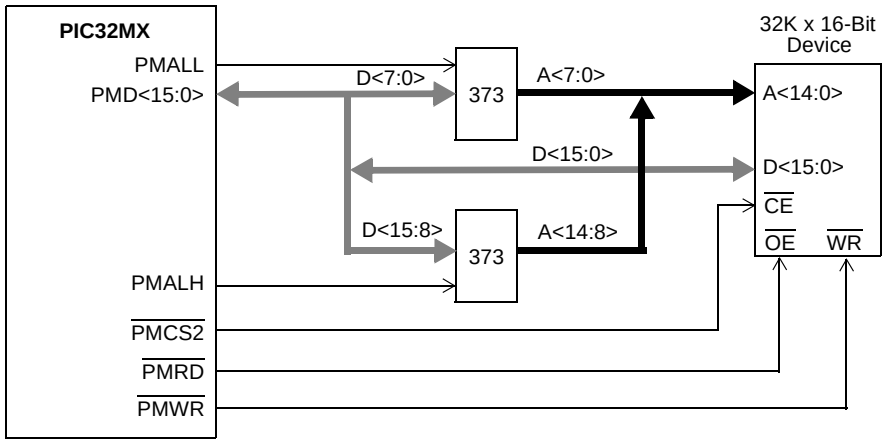


Note: (Master mode 2) MODE<1:0> = 10
(16-bit data width) MODE16 (PMMODE<10>) = 1
(Full Multiplexed mode) ADRMUX (PMCON<12:11>) = 10
The 373 shown in the diagram represents a generic 74XX family 373 latch.

Address Bus
Data Bus
Control Lines

Figure 13-43 illustrates the connections to a 16-bit memory or other addressable peripheral in full 16-bit Multiplex mode, ADRMUX = 11 (PMCON<12:11>). Consequently, from the microcontroller perspective, this mode achieves the best pin saving over the Demultiplexed mode or Partially Multiplexed mode, however, at the price of performance. Compared to the previous Full Multiplex mode, ADRMUX = 10, this mode multiplexes 14 or 15 address bits (if CS2, CS1 or both are enabled) simultaneously with the PMD<15:0> bus and therefore requires only one extra peripheral-bus-clock cycle.

Figure 13-43: Example 2 of Full 16-Bit Multiplexed Addressing, 16-Bit Data (Up to 15-Bit Address)



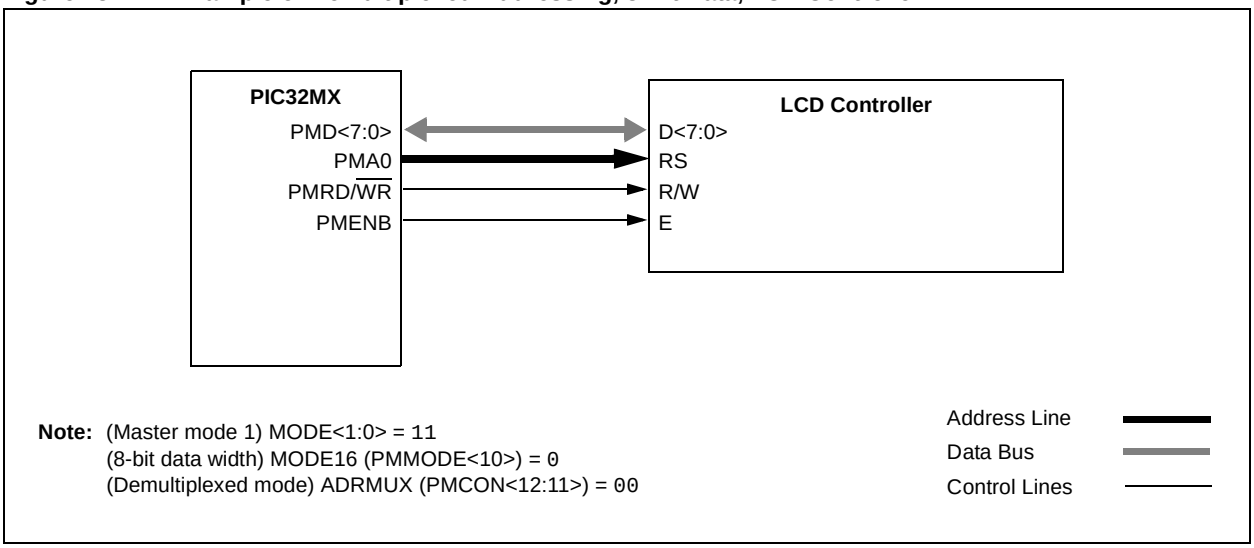
Note: (Master mode 2) MODE<1:0> = 10
(16-bit data width) MODE16 (PMMODE<10>) = 1
(Full Multiplexed mode) ADRMUX (PMCON<12:11>) = 11
The 373 shown in the diagram represents a generic 74XX family 373 latch.

Address Bus
Data Bus
Control Lines

13.8.4 8-Bit LCD Controller Example

The PMP module can be configured to connect to a typical LCD controller interface as shown in Figure 13-44. In this case the PMP module is configured for Master mode 1, MODE<1:0> = 11 (PMMODE<9:8>), and uses active-high control signals since common LCD displays require active-high control.

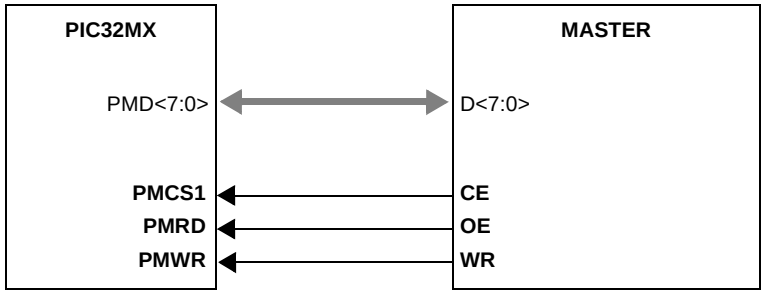
Figure 13-44: Example of Demultiplexed Addressing, 8-Bit Data, LCD Controller



Note: (Master mode 1) MODE<1:0> = 11
(8-bit data width) MODE16 (PMMODE<10>) = 0
(Demultiplexed mode) ADRMUX (PMCON<12:11>) = 00

13.9 PARALLEL SLAVE PORT APPLICATIONS

Figure 13-45: Legacy Mode Slave Port



Note: (Legacy Slave mode) MODE<1:0> (PMMODE<9:8>) = 00

13.10 I/O PIN CONTROL

13.10.1 I/O Pin Resources

When enabling the PMP module for Master mode operations, the PMAEN register must be configured (set = 1) for the corresponding bits of PMA<15:0> I/O pins to be controlled by the PMP module. Those I/O pins not configured for use by the PMP module remain as general purpose I/O pins.

Table 13-12: Required I/O Pin Resources for Master Modes

I/O Pin Name	Demultiplex	Partial Multiplex	Full Multiplex	Functional Description
PMPCS2/PMA15	Yes(2)	Yes(2)	Yes(2)	PMP Chip Select 2/Address A15
PMPCS1/PMA14	Yes(2)	Yes(2)	Yes(2)	PMP Chip Select 1/Address A14
PMA<13:2>	Yes(2)	Yes(3)	No(1)	PMP Address A13..A2
PMA1/PALH	No(1)	No(1)	Yes(4)	PMP Address A1/Address Latch High
PMA0/PALL	No(1)	Yes(3)	Yes(4)	PMP Address A0/Address Latch Low
PMRD/PMWR	Yes	Yes	Yes	PMP Read/Write Control
PMWR/PMENB	Yes	Yes	Yes	PMP Write/Enable Control
PMD<15:0>(6)	Yes(5)	Yes(5)	Yes(5)	PMP Bidirectional Data Bus D15..D0

Note 1: "No" indicates the pin is not required and is available as a general purpose I/O pin when the corresponding PMAEN bit is cleared = 0.

2: Depending on the application, not all PMA<15:0> or CS2, CS1 may be required.

3: When Partial Multiplex mode is selected (ADDRMUX<1:0> = 01), the lower 8 address lines are multiplexed with PMD<7:0>, PMA<0> becomes (ALL) and PMA<7:1> are available as general purpose I/O pins.

4: When Full Multiplex mode is selected (ADDRMUX<1:0> = 10 or 11), all 16 address lines are multiplexed with PMD<15:0>, PMA<0> becomes (ALL), PMA<1> becomes (ALH) and PMA<13:2> are available as general purpose I/O pins.

5: If MODE16 = 0, then only PMD<7:0> are required. PMD<15:8> are available as general purpose I/O pins.

6: Data pins PMD<15:0> are available on 100-pin PIC32MX device variants and larger. For all other device variants, only pins PMD<7:0> are available. Refer to the specific PIC32MX device data sheet for details.

When enabling any of the PMP module for Slave mode operations, the PMPCS1, PMRD, PMWR control pins and PMD<7:0> data pins are automatically enabled and configured. The user is, however, responsible for selecting the appropriate polarity for these control lines.

Table 13-13: Required I/O Pin Resources for Slave Modes

I/O Pin Name	Legacy	Buffered	Enhanced	Functional Description
PMPCS1/PMA14	Yes	Yes	Yes	Chip Select
PMA1/PALH	No(1)	No(1)	Yes	Address A1
PMA0/PALL	No(1)	No(1)	Yes	Address A0
PMRD/PMWR	Yes	Yes	Yes	Read Control
PMWR/PMENB	Yes	Yes	Yes	Write Control
PMD<15:0>	Yes(2)	Yes(2)	Yes(2)	Bidirectional Data Bus D7..D0

Note 1: "No" indicates the pin is not required and is available as a general purpose I/O pin when the corresponding PMAEN bit is cleared = 0.

2: Slave modes use PMD<7:0> only pins. PMD<15:8> are available as general purpose I/O pins. Control bit MODE16 (PMMODE<10>) is ignored.

13.10.2 I/O Pin Configuration

The following table provides a summary of the settings required to enable the I/O pin resources used with this module. The PMAEN register controls the functionality of pins PMA<15:0>. Setting any PMAEN bit = 1 configures the corresponding PMA pin as an address line. Those bits set = 0 remain as general purpose I/O pins.

Table 13-14: I/O Pin Configuration

		Required Settings for Module Pin Control					
I/O Pin Name	**Required(1)**	**Module Control**	**Bit Field**	**TRIS**	**Pin Type**	**Buffer Type**	**Description**
PMPCS2/PMA15	Yes	ON	CSF<1:0>, CS2, PTEN15	—	O	CMOS	PMP Chip Select 2/ Address A15
PMPCS1/PMA14	Yes	ON	CSF<1:0>, CS1 PTEN14	—	O	CMOS	PMP Chip Select 1/ Address A14
PMA<13:2>	Yes	ON	PTEN<13:2>	—	O	CMOS	PMP Address A13 .. A2
PMA1/PALH	Yes	ON	PTEN<1>	—	I,O	CMOS	PMP Address A1/ Address Latch High
PMA0/PALL	Yes	ON	PTEN<0>	—	I,O	CMOS	PMP Address A0/ Address Latch Low
PMRD/PM$\overline{WR}$	Yes	ON	PTRDEN	—	O	CMOS	PMP Read/Write Control
PMWR/PMENB	Yes	ON	PTWREN	—	O	CMOS	PMP Write/Enable Control
PMD<15:0>	Yes	ON	MODE16, ADRMUX<1:0>	—	I,O	CMOS	PMP Bidirectional Data Bus D15 .. D0

Legend: CMOS = CMOS-compatible input or output
ST = Schmitt Trigger input with CMOS levels
I = Input
O = Output

Note 1: Depending on the PMP mode and the user's application, these pins may not be required. If not enabled, these pins can be used for general purpose I/O.

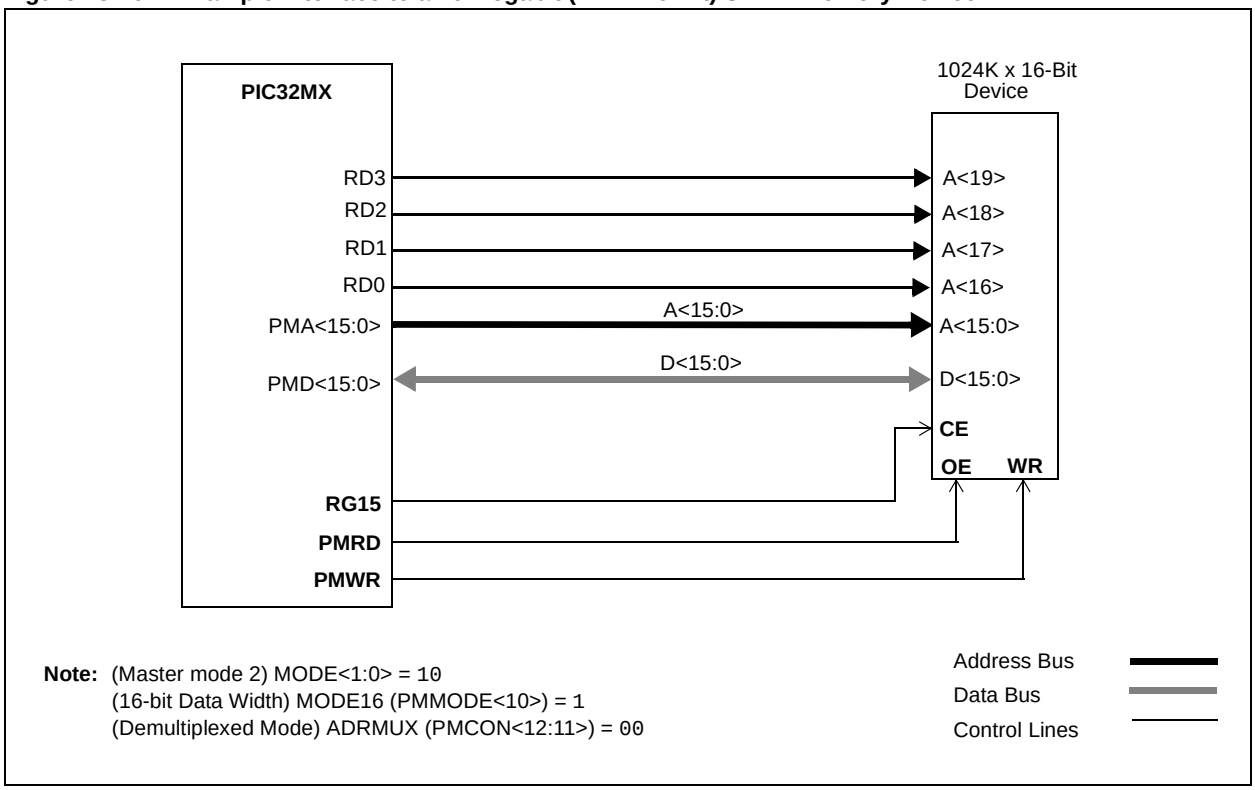
13.11 DESIGN TIPS

Question 1: ***Is it possible for the PMP module to address memory devices larger than 64K?***

Answer: Yes, however not directly under the control of the PMP module. When using the PMCS2 or PMCS1 Chip Select pins, the addressable range is limited to 16K or 32K locations, depending on the Chip Select pin being used. Disabling PMCS2 and PMCS1 as Chip Selects allows these pins to function as address lines PMA15 and PMA14, increasing the range to 64K addressable locations. A dedicated I/O pin is required to function as the Chip Select and the user's software must now control the function of this pin.

To interface to memory devices larger than 64K, use additional available I/O pins as the higher order address lines A16, A17, A18, etc.

Figure 13-46: Example Interface to a 16 Megabit (1 M x 16-Bit) SRAM Memory Device



Note: (Master mode 2) MODE<1:0> = 10
(16-bit Data Width) MODE16 (PMMODE<10>) = 1
(Demultiplexed Mode) ADRMUX (PMCON<12:11>) = 00

Question 2: ***Is it possible to execute code from an external memory device connected to the PMP module?***

Answer: No. Because of the architecture of the PMP module, this is not possible. Only data can be read or written through the PMP.

13.12 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the PIC32MX device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the PMP module are:

Title	Application Note #
No related application notes at this time.	N/A

Note: Please visit the Microchip web site (www.microchip.com) for additional Application Notes and code examples for the PIC32MX family of devices.

13.13 REVISION HISTORY

Revision A (August 2007)

This is the initial released version of this document.

Revision B (October 2007)

Updated document to remove Confidential status.

Revision C (April 2008)

Revised status to Preliminary; Revised U-0 to r-x; Revised Table 13-10; Revised Section 13.3.1.6 and Section 13.3.8; Revised Register 13-5; Revised Figures 13-11, 13-37, 13-40, 13-41, 13-42, 13-43, 13-46; Revised Timing Diagram text for Figures 13-16, 13-18, 13-19.

Revision D (June 2008)

Revised Register 13-1, add note to FRZ; Revised Figures 13-4, 13-6, 13-8, 13-10, 13-36, 13-37, 13-38, 13-45; Revised Table 13-6; Revised Examples 13-6 and 13-7; Change Reserved bits from “Maintain as” to “Write”; Added Note to ON bit (PMCON Register).

NOTES:

Section 14. Timers

HIGHLIGHTS

This section of the manual contains the following topics:

14.1 INTRODUCTION

The PIC32MX device family features two different types of timers, depending on the device variant. Timers are useful for generating accurate time-based periodic interrupt events for software applications or real-time operating systems. Other uses include counting external pulses or accurate timing measurement of external events using the timer's gate feature.

With certain exceptions, all of the timers have the same functional circuitry. All timers are classified into two types to account for their functional differences.

- Type A Timer (16-bit synchronous/asynchronous timer/counter with gate)
- Type B Timer (16-bit, 32-bit synchronous timer/counter with gate and Special Event Trigger)

All Timer modules includes the following common features:

- 16-bit timer/counter
- Software-selectable internal or external clock source
- Programmable interrupt generation and priority
- Gated external pulse counter

Beyond the common features, each timer type offers these additional features:

Type A:

- Asynchronous timer/counter with a built-in oscillator
- Operational during CPU SLEEP mode
- Software selectable prescalers 1:1, 1:8, 1:64 and 1:256

Type B:

- Ability to form a 32-bit timer/counter
- Software prescalers 1:1, 1:2, 1:4, 1:8, 1:16, 1:32, 1:64 and 1:256
- Event trigger capability

The following table presents a summary of timer features. For a specific device variant, refer to the PIC32MX device family data sheet for the available type and number of timers.

Table 14-1: Timer Features

Available Timer Types	Secondary Oscillator	Asynchronous External Clock	Synchronous External Clock	16-Bit Synchronous Timer/Counter	32-Bit(1) Synchronous Timer/Counter	Gated Timer	Special Event Trigger
Type A	Yes	Yes	Yes	Yes	No	Yes	No
Type B	No	No	Yes	Yes	Yes	Yes	Yes

Note 1: 32-bit timer/counter configuration requires an even-numbered timer combined with an adjacent odd-numbered timer, e.g., Timer2 and Timer3, or Timer4 and Timer 5.

14.1.1 Type A Timer

Most PIC32MX devices contain at least one Type A timer; usually, Timer1.

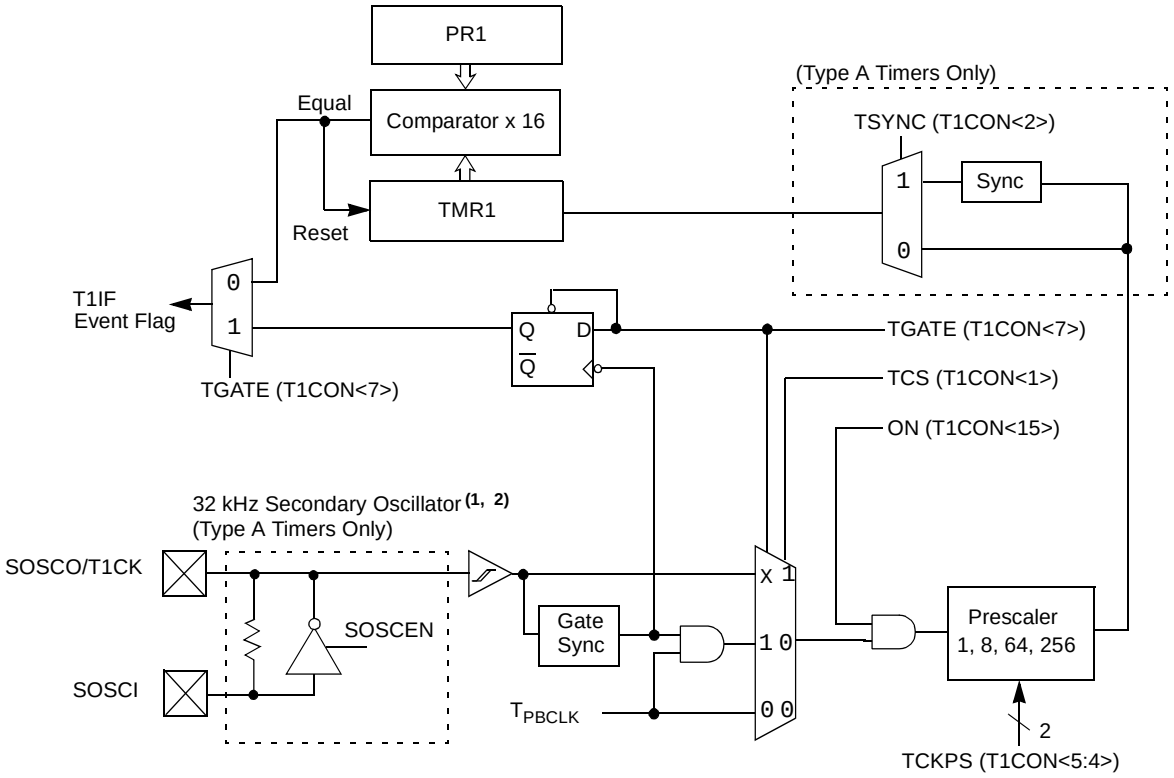
The Type A Timer module is distinct from other types of timers, based on the following features:

- Operable from the external secondary oscillator
- Operable in Asynchronous mode using an external clock source
- Operable during CPU SLEEP mode
- Software selectable prescalers 1:1, 1:8, 1:64 and 1:256

The Type A Timer does not support 32-bit mode.

The unique features of a Type A Timer module allow it to be used for Real-Time Clock (RTC) applications. A block diagram of the Type A Timer module is shown in Figure 14-1.

Figure 14-1: Type A Timer Block Diagram



Note 1: Refer to **Section 6. "Oscillators"** for information on enabling the 32 kHz Secondary Oscillator.

2: The default state of the SOSCEN (OSCCON<1>) during a device Reset is controlled by the FSOSCEN bit in Configuration Word DEVCFG1.

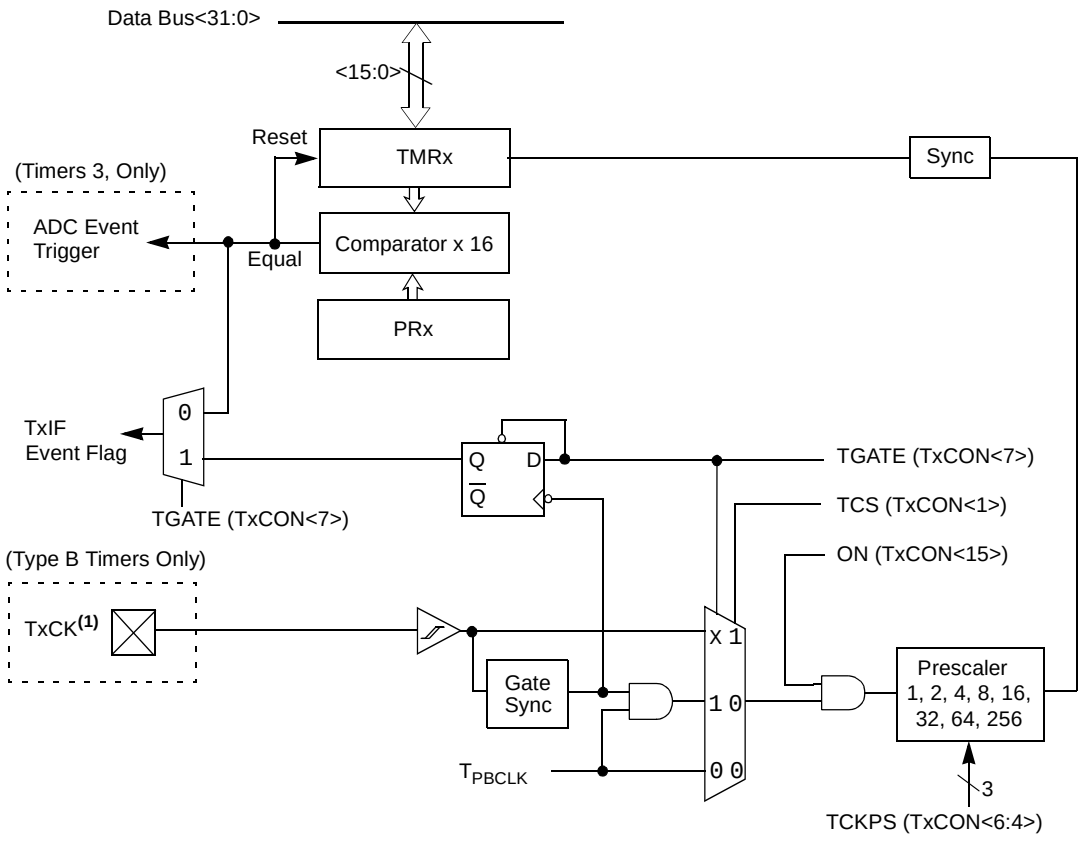
14.1.2 Type B Timer

The Type B timer is distinct from other types of timers, based on the following features:

- Can be combined to form a 32-bit timer
- Software selectable prescalers 1:1, 1:2, 1:4, 1:8, 1:16, 1:32, 1:64 and 1:256
- ADC event trigger capability

A block diagram of Type B timer (16-bit) is shown in Figure 14-2.

Figure 14-2: Type B Timer Block Diagram (16-Bit)

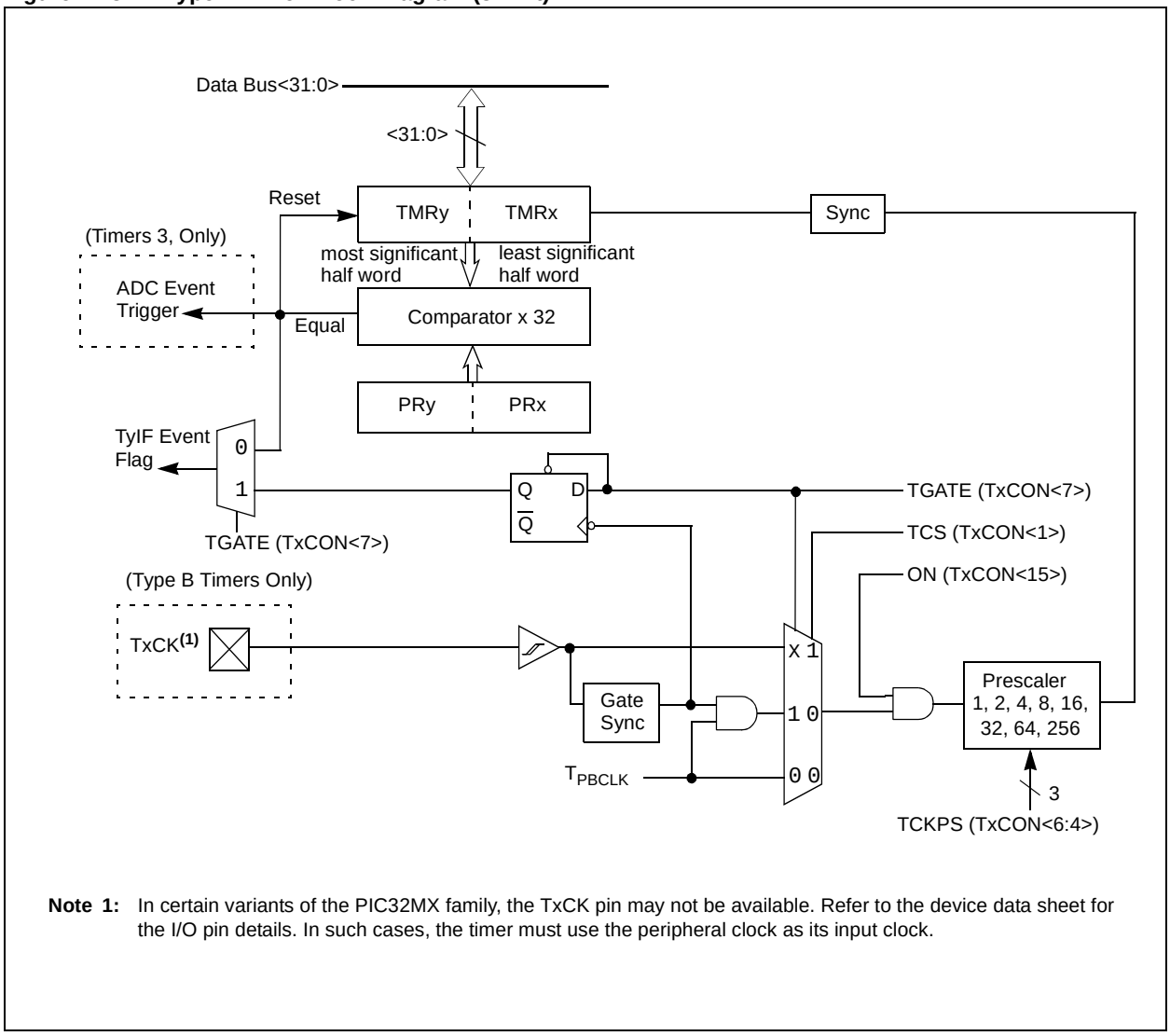


Note 1: In certain variants of the PIC32MX family, the TxCK pin may not be available. Refer to the device data sheet for the I/O pin details. In such cases, the timer must use the peripheral clock as its input clock.

A block diagram of Type B timer (32-bit) is shown in Figure 14-3.

Note: The Timer Configuration bit, T32 (TxCON<3>), must be set to '1' for a 32-bit timer/counter operation. All control bits are respective to the TxCON register. All interrupt bits are respective to the TyCON register.

Figure 14-3: Type B Timer Block Diagram (32-Bit)



Note 1: In certain variants of the PIC32MX family, the TxCK pin may not be available. Refer to the device data sheet for the I/O pin details. In such cases, the timer must use the peripheral clock as its input clock.

14.2 CONTROL REGISTERS

Note: Each PIC32MX device variant may have one or more Timer modules. An 'x' used in the names of pins, control/Status bits, and registers denotes the particular module. Refer to the specific device data sheets for more details.

Each timer module is a 16-bit timer/counter that consists of the following Special Function Registers (SFRs):

- TxCON: 16-Bit Control Register Associated with the Timer
- TxCONCLR, TxCONSET, TxCONINV: Atomic Bit Manipulation Write-only Registers for TxCON
- TMRx: 16-Bit Timer Count Register
- TMRxCLR, TMRxSET, TMRxINV: Atomic Bit Manipulation Write-only Registers for TMRx
- PRx: 16-Bit Period Register Associated with the Timer
- PRxCLR, PRxSET, PRxINV: Atomic Bit Manipulation Write-only Registers for PRx

Each timer module also has the following associated bits for interrupt control:

- TxIE: Interrupt Enable Control Bit – in IEC0 INT Register
- TxIF: Interrupt Flag Status Bit – in IFS0 INT Register
- TxIP<2:0>: Interrupt Priority Control Bits – in IPC1, IPC2, IPC3, IPC4, IPC5 INT Registers
- TxIS<1:0>: Interrupt Subpriority Control Bits – in IPC1, IPC2, IPC3, IPC4, IPC5 INT Registers

The following table summarizes all Timer-related registers. Corresponding registers appear after the summary, followed by a detailed description of each register.

Table 14-2: Timers SFR Summary

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
T1CON	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	7:0	TGATE	—	TCKPS<1:0>		—	TSYNC	TCS	—
T1CONCLR	31:0	Write clears selected bits in T1CON, read yields undefined value							
T1CONSET	31:0	Write sets selected bits in T1CON, read yields undefined value							
T1CONINV	31:0	Write inverts selected bits in T1CON, read yields undefined value							
TxCON	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	ON	FRZ	SIDL	—	—	—	—	—
	7:0	TGATE	TCKPS<2:0>(2)			T32(1)	—	TCS	—
TxCONCLR	31:0	Write clears selected bits in TxCON, read yields undefined value							
TxCONSET	31:0	Write sets selected bits in TxCON, read yields undefined value							
TxCONINV	31:0	Write inverts selected bits in TxCON, read yields undefined value							
TMRx	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	TMRx<15:8>							
	7:0	TMRx<7:0>							
TMRxCLR	31:0	Write clears selected bits in TMRx, read yields undefined value							
TMRxSET	31:0	Write sets selected bits in TMRx, read yields undefined value							
TMRxINV	31:0	Write inverts selected bits in TMRx, read yields undefined value							

Note 1: Bit T32 is available only on even-numbered Type B timers, e.g., Timer2, Timer4.
2: TCKPS<2:0> is available only on even-numbered Type B timers, e.g., Timer2, Timer4 in 32-bit Timer mode.

Table 14-2: Timers SFR Summary (Continued)

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
PRx	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	PRx<15:8>							
	7:0	PRx<7:0>							
PRxCLR	31:0	Write clears selected bits in PRx, read yields undefined value							
PRxSET	31:0	Write sets selected bits in PRx, read yields undefined value							
PRxINV	31:0	Write inverts selected bits in PRx, read yields undefined value							
IEC0	31:24	I2C1MIE	I2C1SIE	I2C1BIE	U1TXIE	U1RXIE	U1EIE	SPI1RXIE	SPI1TXIE
	23:16	SPI1EIE	OC5IE	IC5IE	T5IE	INT4IE	OC4IE	IC4IE	T4IE
	15:8	INT3IE	OC3IE	IC3IE	T3IE	INT2IE	OC2IE	IC2IE	T2IE
	7:0	INT1IE	OC1IE	IC1IE	T1IE	INT0IE	CS1IE	CS0IE	CTIE
IFS0	31:24	I2C1MIF	I2C1SIF	I2C1BIF	U1TXIF	U1RXIF	U1EIF	SPI1RXIF	SPI1TXIF
	23:16	SPI1EIF	OC5IF	IC5IF	T5IF	INT4IF	OC4IF	IC4IF	T4IF
	15:8	INT3IF	OC3IF	IC3IF	T3IF	INT2IF	OC2IF	IC2IF	T2IF
	7:0	INT1IF	OC1IF	IC1IF	T1IF	INT0IF	CS1IF	CS0IF	CTIF
IPC1	31:24	—	—	—	INT1IP<2:0>			INT1IS<1:0>	
	23:16	—	—	—	OC1IP<2:0>			OC1IS<1:0>	
	15:8	—	—	—	IC1IP<2:0>			IC1IS<1:0>	
	7:0	—	—	—	T1IP<2:0>			T1IS<1:0>	
IPC2	31:24	—	—	—	INT2IP<2:0>			INT2IS<1:0>	
	23:16	—	—	—	OC2IP<2:0>			OC2IS<1:0>	
	15:8	—	—	—	IC2IP<2:0>			IC2IS<1:0>	
	7:0	—	—	—	T2IP<2:0>			T2IS<1:0>	
IPC3	31:24	—	—	—	INT3IP<2:0>			INT3IS<1:0>	
	23:16	—	—	—	OC3IP<2:0>			OC3IS<1:0>	
	15:8	—	—	—	IC3IP<2:0>			IC3IS<1:0>	
	7:0	—	—	—	T3IP<2:0>			T3IS<1:0>	
IPC4	31:24	—	—	—	INT4IP<2:0>			INT4IS<1:0>	
	23:16	—	—	—	OC4IP<2:0>			OC4IS<1:0>	
	15:8	—	—	—	IC4IP<2:0>			IC4IS<1:0>	
	7:0	—	—	—	T4IP<2:0>			T4IS<1:0>	
IPC5	31:24	—	—	—	SPI1IP<2:0>			SPI1IS<1:0>	
	23:16	—	—	—	OC5IP<2:0>			OC5IS<1:0>	
	15:8	—	—	—	IC5IP<2:0>			IC5IS<1:0>	
	7:0	—	—	—	T5IP<2:0>			T5IS<1:0>	

Note 1: Bit T32 is available only on even-numbered Type B timers, e.g., Timer2, Timer4.

2: TCKPS<2:0> is available only on even-numbered Type B timers, e.g., Timer2, Timer4 in 32-bit Timer mode.

Register 14-1: T1CON: Type A Timer Control Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R-0	r-x	r-x	r-x
ON	FRZ	SIDL	TWDIS	TWIP	—	—	—
bit 15							bit 8

R/W-0	r-x	R/W-0	R/W-0	r-x	R/W-0	R/W-0	r-x
TGATE	—	TCKPS<1:0>		—	TSYNC	TCS	—
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15 **ON:** Timer On bit

1 = Timer is enabled
0 = Timer is disabled

Note: When using 1:1 PBCLK divisor, the user’s software should not read/write the peripheral’s SFRs in the SYSCLK cycle immediately following the instruction that clears the module’s ON bit.

bit 14 **FRZ:** Freeze in Debug Exception Mode bit

1 = Freeze operation when CPU is in Debug Exception mode
0 = Continue operation even when CPU is in Debug Exception mode

Note: FRZ is writable in Debug Exception mode only, it is forced to ‘0’ in Normal mode.

bit 13 **SIDL:** Stop in IDLE Mode bit

1 = Discontinue operation when device enters IDLE mode
0 = Continue operation even in IDLE mode

bit 12 **TWDIS:** Asynchronous Timer Write Disable bit

1 = Writes to TMR1 are ignored until pending write operation completes
0 = Back to back writes are enabled (Legacy Asynchronous Timer functionality)

bit 11 **TWIP:** Asynchronous Timer Write in Progress bit

In Asynchronous Timer mode:
1 = Asynchronous write to TMR1 register in progress
0 = Asynchronous write to TMR1 register complete

In Synchronous Timer mode:
This bit is read as ‘0’.

bit 10-8 **Reserved:** Write ‘0’; ignore read

Register 14-1: T1CON: Type A Timer Control Register (Continued)

bit 7 **TGATE:** Timer Gated Time Accumulation Enable bit

When TCS = 1:
This bit is ignored and read '0'.

When TCS = 0:
1 = Gated time accumulation is enabled
0 = Gated time accumulation is disabled

bit 6 **Reserved:** Write '0'; ignore read

bit 5-4 **TCKPS<1:0>:** Timer Input Clock Prescale Select bits

11 = 1:256 prescale value
10 = 1:64 prescale value
01 = 1:8 prescale value
00 = 1:1 prescale value

bit 3 **Reserved:** Write '0'; ignore read

bit 2 **TSYNC:** Timer External Clock Input Synchronization Selection bit

When TCS = 1:
1 = External clock input is synchronized
0 = External clock input is not synchronized

When TCS = 0:
This bit is ignored and read '0'.

bit 1 **TCS:** Timer Clock Source Select bit

1 = External clock from TxCKI pin
0 = Internal peripheral clock

bit 0 **Reserved:** Write '0'; ignore read

Register 14-2: T1CONCLR: Timer Control Clear Register

Write clears selected bits in T1CON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in T1CON**

A write of '1' in one or more bit positions clears the corresponding bit(s) in T1CON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `T1CONCLR = 0x00008001` will clear bits 15 and 0 in T1CON register.

Register 14-3: T1CONSET: Timer Control Set Register

Write sets selected bits in T1CON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in T1CON**

A write of '1' in one or more bit positions sets the corresponding bit(s) in T1CON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `T1CONSET = 0x00008001` will set bits 15 and 0 in T1CON register.

Register 14-4: T1CONINV: Timer Control Invert Register

Write inverts selected bits in T1CON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in T1CON**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in T1CON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `T1CONINV = 0x00008001` will invert bits 15 and 0 in T1CON register.

Register 14-5: TxCON: Type B Timer Control Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	r-x	r-x	r-x	r-x	r-x
ON	FRZ	SIDL	—	—	—	—	—
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	r-x	R/W-0	r-x
TGATE	TCKPS<2:0>			T32	—	TCS	—
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15 **ON:** Timer On bit

1 = Module is enabled
0 = Module is disabled

Note: When using 1:1 PBCLK divisor, the user’s software should not read/write the peripheral’s SFRs in the SYSCLK cycle immediately following the instruction that clears the module’s ON bit.

bit 14 **FRZ:** Freeze in Debug Exception Mode bit

1 = Freeze operation when CPU is in Debug Exception mode
0 = Continue operation even when CPU is in Debug Exception mode
Note: FRZ is writable in Debug Exception mode only, it is forced to ‘0’ in Normal mode.

bit 13 **SIDL:** Stop in IDLE Mode bit

1 = Discontinue operation when device enters IDLE mode
0 = Continue operation even in IDLE mode

bit 12-8 **Reserved:** Write ‘0’; ignore read

bit 7 **TGATE:** Timer Gated Time Accumulation Enable bit

When TCS = 1:
This bit is ignored and read ‘0’.

When TCS = 0:
1 = Gated time accumulation is enabled
0 = Gated time accumulation is disabled

Register 14-5: TxCON: Type B Timer Control Register (Continued)

bit 6-4 **TCKPS<2:0>:** Timer Input Clock Prescale Select bits

111 = 1:256 prescale value
110 = 1:64 prescale value
101 = 1:32 prescale value
100 = 1:16 prescale value
011 = 1:8 prescale value
010 = 1:4 prescale value
001 = 1:2 prescale value
000 = 1:1 prescale value

bit 3 **T32:** 32-Bit Timer Mode Select bit

1 = TMRx and TMRy form a 32-bit timer
0 = TMRx and TMRy form separate 16-bit timer

Note: Bit T32 is available only on even-numbered Type B timers: Timer 2, Timer 4, etc.

bit 2 **Reserved:** Write '0'; ignore read

bit 1 **TCS:** Timer Clock Source Select bit

1 = External clock from TxCKI pin
0 = Internal peripheral clock

bit 0 **Reserved:** Write '0'; ignore read

Register 14-6: TxCONCLR: Type B Timer Control Clear Register

Write clears selected bits in TxCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in TxCON**

A write of '1' in one or more bit positions clears the corresponding bit(s) in TxCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `TxCONCLR = 0x00008001` will clear bits 15 and 0 in TxCON register.

Register 14-7: TxCONSET: Type B Timer Control Set Register

Write sets selected bits in TxCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in TxCON**

A write of '1' in one or more bit positions sets the corresponding bit(s) in TxCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `TxCONSET = 0x00008001` will set bits 15 and 0 in TxCON register.

Register 14-8: TxCONINV: Type B Timer Control Invert Register

Write inverts selected bits in TxCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in TxCON**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in TxCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `TxCONINV = 0x00008001` will invert bits 15 and 0 in TxCON register.

Register 14-9: TMRx: Timer Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
TMR<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
TMR<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 15-0 **TMRx<15:0>:** Timer Count Register

16-bit mode:
These bits represent the complete 16-bit timer count.

32-bit mode (Timer Type B only):
Timer2 and Timer4
These bits represent the least significant half word (16 bits) of the 32-bit timer count.
Timer3 and Timer5
These bits represent the most significant half word (16 bits) of the 32-bit timer count.

Register 14-10: TMRxCLR: Timer Clear Register

Write clears selected bits in TMRx, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in TMRx**

A write of '1' in one or more bit positions clears the corresponding bit(s) in TMRx register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `TMRxCLR = 0x00008001` will clear bits 15 and 0 in TMRx register.

Register 14-11: TMRxSET: Timer Set Register

Write sets selected bits in TMRx, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in TMRx**

A write of '1' in one or more bit positions sets the corresponding bit(s) in TMRx register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `TMRxSET = 0x00008001` will set bits 15 and 0 in TMRx register.

Register 14-12: TMRxINV: Timer Invert Register

Write inverts selected bits in TMRx, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in TMRx**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in TMRx register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `TMRxINV = 0x00008001` will invert bits 15 and 0 in TMRx register.

Register 14-13: PRx: Period Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PR<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PR<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 15-0 **PRx<15:0>:** Period Register

16-bit mode:
These bits represent the complete 16-bit period match

32-bit mode (Timer Type B only):
Timer2 and Timer4
These bits represent the least significant half word (16 bits) of the 32-bit period match.
Timer3 and Timer5
These bits represent the most significant half word (16 bits) of the 32-bit period match.

Register 14-14: PRxCLR: Period Clear Register

Write clears selected bits in PRx, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in PRx**

A write of '1' in one or more bit positions clears the corresponding bit(s) in PRx register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `PRxCLR = 0x00008001` **will clear bits 15 and 0 in PRx register.**

Register 14-15: PRxSET: Period Set Register

Write sets selected bits in PRx, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in PRx**

A write of '1' in one or more bit positions sets the corresponding bit(s) in PRx register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `PRxSET = 0x00008001` **will set bits 15 and 0 in PRx register.**

Register 14-16: PRxINV: Period Invert Register

Write inverts selected bits in PRx, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in PRx**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in PRx register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `PRxINV = 0x00008001` **will invert bits 15 and 0 in PRx register.**

Register 14-17: IEC0: Interrupt Enable Control Register(1)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
I2C1MIE	I2C1SIE	I2C1BIE	U1TXIE	U1RXIE	U1EIE	SPI1RXIE	SPI1TXIE
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SPI1EIE	OC5IE	IC5IE	T5IE	INT4IE	OC4IE	IC4IE	T4IE
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INT3IE	OC3IE	IC3IE	T3IE	INT2IE	OC2IE	IC2IE	T2IE
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INT1IE	OC1IE	IC1IE	T1IE	INT0IE	CS1IE	CS0IE	CTIE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 20 **T5IE:** Timer5 Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 16 **T4IE:** Timer4 Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 12 **T3IE:** Timer3 Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 8 **T2IE:** Timer2 Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 4 **T1IE:** Timer1 Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the Timers.

Register 14-18: IFS0: Interrupt Flag Status Register 0(1)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
I2C1MIF	I2C1SIF	I2C1BIF	U1TXIF	U1RXIF	U1EIF	SPI1RXIF	SPI1TXIF
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SPI1EIF	OC5IF	IC5IF	T5IF	INT4IF	OC4IF	IC4IF	T4IF
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INT3IF	OC3IF	IC3IF	T3IF	INT2IF	OC2IF	IC2IF	T2IF
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INT1IF	OC1IF	IC1IF	T1IF	INT0IF	CS1IF	CS0IF	CTIF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 20 **T5IF:** Timer5 Interrupt Request Flag bit
1 = Interrupt request has occurred
0 = No interrupt request has occurred

bit 16 **T4IF:** Timer4 Interrupt Request Flag bit
1 = Interrupt request has occurred
0 = No interrupt request has occurred

bit 12 **T3IF:** Timer3 Interrupt Request Flag bit
1 = Interrupt request has occurred
0 = No interrupt request has occurred

bit 8 **T2IF:** Timer2 Interrupt Request Flag bit
1 = Interrupt request has occurred
0 = No interrupt request has occurred

bit 4 **T1IF:** Timer1 Interrupt Request Flag bit
1 = Interrupt request has occurred
0 = No interrupt request has occurred

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the Timers.

Register 14-19: IPC1: Interrupt Priority Control Register 1(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	INT1IP<2:0>			INT1IS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	OC1IP<2:0>			OC1IS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	IC1IP<2:0>			IC1IS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	T1IP<2:0>			T1IS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 4-2 **T1IP<2:0>:** Timer1 Interrupt Priority bits
- 111 = Interrupt Priority is 7
- 110 = Interrupt Priority is 6
- 101 = Interrupt Priority is 5
- 100 = Interrupt Priority is 4
- 011 = Interrupt Priority is 3
- 010 = Interrupt Priority is 2
- 001 = Interrupt Priority is 1
- 000 = Interrupt is disabled

bit 1-0 **T1IS<1:0>:** Timer1 Interrupt Subpriority bits
- 11 = Interrupt Subpriority is 3
- 10 = Interrupt Subpriority is 2
- 01 = Interrupt Subpriority is 1
- 00 = Interrupt Subpriority is 0

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the Timer1.

Register 14-20: IPC2: Interrupt Priority Control Register 2(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	INT2IP<2:0>			INT2IS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	OC2IP<2:0>			OC2IS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	IC2IP<2:0>			IC2IS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	T2IP<2:0>			T2IS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 4-2 **T2IP<2:0>:** Timer2 Interrupt Priority bits

111 = Interrupt Priority is 7
110 = Interrupt Priority is 6
101 = Interrupt Priority is 5
100 = Interrupt Priority is 4
011 = Interrupt Priority is 3
010 = Interrupt Priority is 2
001 = Interrupt Priority is 1
000 = Interrupt is disabled

bit 1-0 **T2IS<1:0>:** Timer2 Interrupt Subpriority bits

11 = Interrupt Subpriority is 3
10 = Interrupt Subpriority is 2
01 = Interrupt Subpriority is 1
00 = Interrupt Subpriority is 0

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the Timer2.

Register 14-21: IPC3: Interrupt Priority Control Register 3(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	INT3IP<2:0>			INT3IS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	OC3IP<2:0>			OC3IS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	IC3IP<2:0>			IC3IS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	T3IP<2:0>			T3IS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 4-2 **T3IP<2:0>:** Timer3 Interrupt Priority bits

111 = Interrupt Priority is 7
110 = Interrupt Priority is 6
101 = Interrupt Priority is 5
100 = Interrupt Priority is 4
011 = Interrupt Priority is 3
010 = Interrupt Priority is 2
001 = Interrupt Priority is 1
000 = Interrupt is disabled

bit 1-0 **T3IS<1:0>:** Timer3 Interrupt Subpriority bits

11 = Interrupt Subpriority is 3
10 = Interrupt Subpriority is 2
01 = Interrupt Subpriority is 1
00 = Interrupt Subpriority is 0

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the Timer3.

Register 14-22: IPC4: Interrupt Priority Control Register 4(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	INT4IP<2:0>			INT4IS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	OC4IP<2:0>			OC4IS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	IC4IP<2:0>			IC4IS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	T4IP<2:0>			T4IS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 4-2 **T4IP<2:0>:** Timer4 Interrupt Priority bits
- 111 = Interrupt Priority is 7
- 110 = Interrupt Priority is 6
- 101 = Interrupt Priority is 5
- 100 = Interrupt Priority is 4
- 011 = Interrupt Priority is 3
- 010 = Interrupt Priority is 2
- 001 = Interrupt Priority is 1
- 000 = Interrupt is disabled

bit 1-0 **T4IS<1:0>:** Timer4 Interrupt Subpriority bits
- 11 = Interrupt Subpriority is 3
- 10 = Interrupt Subpriority is 2
- 01 = Interrupt Subpriority is 1
- 00 = Interrupt Subpriority is 0

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the Timer4.

Register 14-23: IPC5: Interrupt Priority Control Register 5(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	SPI1IP<2:0>			SPI1IS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	OC5IP<2:0>			OC5IS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	IC5IP<2:0>			IC5IS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	T5IP<2:0>			T5IS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 4-2 **T5IP<2:0>:** Timer5 Interrupt Priority bits
- 111 = Interrupt Priority is 7
- 110 = Interrupt Priority is 6
- 101 = Interrupt Priority is 5
- 100 = Interrupt Priority is 4
- 011 = Interrupt Priority is 3
- 010 = Interrupt Priority is 2
- 001 = Interrupt Priority is 1
- 000 = Interrupt is disabled

bit 1-0 **T5IS<1:0>:** Timer5 Interrupt Subpriority bits
- 11 = Interrupt Subpriority is 3
- 10 = Interrupt Subpriority is 2
- 01 = Interrupt Subpriority is 1
- 00 = Interrupt Subpriority is 0

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the Timer5.

14.3 MODES OF OPERATION

14.3.1 16-Bit Modes

Both Type A and Type B timer modules support the following 16-bit modes:

- 16-bit Synchronous Clock Counter
- 16-bit Synchronous External Clock Counter
- 16-bit Asynchronous External Counter (Type A Timer module only)
- 16-bit Gated Timer

The 16-bit Timer modes are determined by the following bits:

- TCS (TxCON<1>): Timer Clock Source Control bit
- TSYNC (T1CON<2>): Timer Synchronization Control bit (Type A Timer module only)
- TGATE (TxCON<7>): Timer Gate Control bit

14.3.1.1 16-Bit Timer Considerations

The following should be considered when using a 16-bit timer:

- All timer module SFRs can be written to as a byte (8 bits) or as a half word (16 bits).
- All timer module SFRs can be read from as a byte or as a half word.

14.3.2 32-Bit Modes (Type B Timer)

Only Type B timer modules support 32-bit modes of operation. A 32-Bit Timer module is formed by combining an even numbered Type B timer (referred to as TimerX) with a consecutive odd numbered Type B timer (referred to as TimerY). For example, 32-bit timer combinations are Timer2 and Timer3, Timer4 and Timer5, etc. The number of timer pairs depends on the device family variant.

The 32-Bit Timer pairs can operate in the following modes:

- 32-Bit Synchronous Clock Counter
- 32-Bit Synchronous External Clock Counter
- 32-Bit Gated Timer

The 32-Bit Timer modes are determined by the following bits:

- T32 (TxCON<3>): 32-Bit mode Control Bit (TimerX only)
- TCS (TxCON<1>): Timer Clock Source Control Bit
- TGATE (TxCON<7>): Timer Gate Control Bit

Specific behavior in 32-bit Timer mode:

- TimerX is the master timer; TimerY is the slave timer
- TMRx count register is least significant half word (lshw) of the 32-bit timer value
- TMRy count register is most significant half word (mshw) of the 32-bit timer value
- PRx period register is least significant half word of the 32-bit period value
- PRy period register is most significant half word of the 32-bit period value
- TimerX control bits (TxCON) configure the operation for the 32-bit timer pair
- TimerY control bits (TyCON) have no effect
- TimerX interrupt and Status bits are ignored
- TimerY provides the interrupt enable, interrupt flag and interrupt priority control bits

14.3.2.1 32-Bit Timer Considerations

The following should be considered when using a 32-bit timer:

- Ensure that the timer pair is configured for 32-bit mode by setting T32 (TxCON<3>) = 1, before writing any 32-bit value to the TMRxy count registers or PRxy period registers.
- All timer module SFRs can be written to as a byte (8 bits), a half word (16 bits) or a word (32 bits).
- All timer module SFRs can be read from as a byte, a half word or a word.
- TMRx and TMRy count register pairs can be read as well as written as a single 32-bit value.
- PRx and PRy period register pairs can be read as well as written as a single 32-bit value.

14.3.3 16-Bit Synchronous Clock Counter Mode

The Synchronous Clock Counter operation provides the following capabilities:

- Elapsed time measurements
- Time delays
- Periodic timer interrupts

Type A and B timers have the ability to operate in Synchronous Clock Counter mode. In this mode, the input clock source for the timer is the internal peripheral bus clock, PBCLK, and is selected by clearing the clock source control bit TCS, (TxCON<1>) = 0. Type A and B Timers automatically provide synchronization to the peripheral bus clock; therefore, the Type A Timer Synchronous mode control bit TSYNC (T1CON<2>) is ignored in this mode.

Type A and B timers that use a 1:1 clock prescale operate at a timer clock rate which is the same as the PBCLK, and which increments the TMR count register on every rising timer clock edge. The timer continues to increment until the TMR count register matches the PR period register value. The TMR count register resets to 0000h on the next timer clock cycle, then continues to increment and repeat the period match until the timer is disabled. If the PR period register value = 0000h, the TMR count register resets to 0000h on the next timer clock cycle, but does not continue to increment.

Type A and B timers using a clock prescale = N (other than 1:1) operate at a timer clock rate (PBCLK/N) and the TMR count register increments on every Nth timer clock rising edge. For example, if the clock prescale is 1:8, then the timer increments on every 8th timer clock cycle. The timer continues to increment until the TMR count register matches the PR period register value. The TMR count register then resets to 0000h after N more timer clock cycles, then continues to increment and repeat the period match until the timer is disabled. If the PR period register value = 0000h, the TMR count register resets to 0000h on the next Nth timer clock cycle, but will not continue to increment.

Type A timers generate a timer event one-half timer clock cycle (on the falling edge) after the TMR count register matches the PR period register value. Type B timers generate a timer event within 1 PBCLK + 2 SYSCLK system clock cycles after the TMR count register matches the PR period register value. Both Type A and B timer interrupt flag bits, TxIF, are set within 1 PBCLK + 2 SYSCLK cycles of this event and if the timer interrupt enable bit TxIE is set, an interrupt is generated.

14.3.3.1 16-Bit Synchronous Clock Counter Considerations

This section describes items that should be considered when using a 16-bit Synchronous Clock Counter.

The timer period is determined by the value in the PR period register. To initialize the timer period, a user may write to the PR period register directly at any time while the timer is disabled, ON bit = 0, or during a timer match Interrupt Service Routine (ISR) while the timer is enabled, ON bit = 1. In all other cases, writing to the period register while the timer is enabled is not recommended and may allow unintended period matches to occur.

The maximum period that can be loaded is FFFFh.

Writing 0000h to PRx period register allows a TMRx match to occur; however, no interrupt will be generated.

14.3.4 32-Bit Synchronous Clock Counter Mode (Type B Timer)

Only Type B timers have the ability to operate in 32-bit Synchronous Counter mode. To enable 32-bit Synchronous Clock Counter operation, Type B (TimerX) T32 control bit (TxCON<3>) must be set (= 1). In this mode, the input clock source for the timer is the internal peripheral bus clock, PBCLK, and is selected by clearing the clock source control bit TCS, (TxCON<1>) = 0. Type B timers automatically provide synchronization to the peripheral bus clock.

Type B Timers that use a 1:1 clock prescale operate at a timer clock rate which is the same as the PBCLK, and increments the TMRxy count register on every rising timer clock edge. The timer continues to increment until the TMRxy count register matches the PRxy period register value. The TMRxy count register resets to 00000000h on the next timer clock cycle, then continues to increment and repeat the period match until the timer is disabled. If the PR period register value = 00000000h, the TMR count register resets to 00000000h on the next timer clock cycle, but does not continue to increment.

Type B timers using a clock prescale = N (other than 1:1) operate at a timer clock rate (PBCLK/N) and the TMRxy count register increments on every Nth timer clock rising edge. For example, if the clock prescale is 1:8, then the timer increments on every 8th timer clock cycle. The timer continues to increment until the TMRxy count register matches the PRxy period register value. The TMRxy count register resets to 00000000h after N more timer clock cycles, then continues to increment and repeat the period match until the timer is disabled.

Type B timers generate a timer event within 1 PBCLK + 2 SYSCLK system clock cycles after the TMRxy count register matches the PRxy period register value. The Type B timer interrupt flag bit, TyIF, is set within 1 PBCLK + 2 SYSCLK cycles of this event and if the timer interrupt enable bit TyIE is set, an interrupt is generated.

14.3.4.1 32-Bit Synchronous Clock Counter Considerations

This section describes items that should be considered when using the 32-bit Synchronous Clock Counter.

The timer period is determined by the value in the PRxy period register. To initialize the timer period, a user may write to the PRxy period register directly at any time while the timer is disabled, ON bit = 0, or during a timer match Interrupt Service Routine while the timer is enabled, ON bit = 1. In all other cases, writing to the period register while the timer is enabled is not recommended, and may allow unintended period matches to occur.

The maximum period that can be loaded is FFFFFFFFh.

Writing 00000000h to the PRxy period register will allow a TMRxy match to occur; however, no interrupt is generated.

14.3.4.2 16-Bit Synchronous Counter Initialization Steps

Performed the following steps to configure the timer for 16-bit Synchronous Timer mode.

1. Clear control bit ON (TxCON<15> = 0) to disable timer.
2. Clear control bit TCS (TxCON<1> = 0) to select internal PBCLK source.
3. Select desired clock prescale.
4. Load/Clear timer register TMRx.
5. Load period register PRx with desired 16-bit match value.
6. If interrupts are used:
   i. Clear interrupt flag bit TxIF in IFS0 register.
   ii. Configure interrupt priority and subpriority levels in IPCn register.
   iii. Set interrupt enable bit TxIE in IEC0 registers.
7. Set control bit ON (TxCON<15> = 1) to enable the timer.

Example 14-1: 16-Bit Synchronous Clock Counter Example Code

```
T2CON = 0x0;              // Stop Timer and clear control register,
                          // set prescaler at 1:1, internal clock source
TMR2 = 0x0;               // Clear timer register
PR2 = 0xFFFF;             // Load period register
T2CONSET = 0x8000;        // Start Timer
```

14.3.4.3 32-Bit Synchronous Clock Counter Initialization Steps

Performed the following steps to configure the timer for 32-bit Synchronous Clock Counter mode.

1. Clear control bit ON (TxCON<15> = 0) to disable timer.
2. Clear control bit TCS (TxCON<1> = 0) to select internal PBCLK source.
3. Set control bit T32 (TxCON<3> = 1) to select 32-bit operations.
4. Select desired clock prescale.
5. Load/Clear timer register TMRxy.
6. Load period register PRxy with desired 32-bit match value.
7. If interrupts are used:
   i. Clear interrupt flag bit TyIF in IFSn register.
   ii. Configure interrupt priority and subpriority levels in IPCn register.
   iii. Set interrupt enable bit TyIE in IECn registers.
8. Set control bit ON (TxCON<15> = 1) to enable the timer.

Example 14-2: 32-Bit Synchronous Clock Counter Example Code

```
T4CON = 0x0;              // Stop any 16/32-bit Timer4 operation
T5CON = 0x0;              // Stop any 16-bit Timer5 operation
T4CONSET = 0x0038;        // Enable 32-bit mode, prescaler 1:8,
                          // internal peripheral clock source

TMR4= 0x0;                // Clear contents of the TMR4 and TMR5
PR4 = 0xFFFFFFFF;         // Load PR4 and PR5 registers with 32-bit value

T4CONSET = 0x8000;        // Start Timer45
```


14.3.5 16-Bit Synchronous External Clock Counter Mode

The Synchronous External Clock Counter operation provides the following capabilities:

- Counting periodic or non-periodic pulses
- Use external clock as time base for timers

Type A and B timers have the ability to operate in Synchronous External Clock Counter mode. In this mode, the input clock source for the timer is an external clock applied to the TxCK pin and is selected by setting the clock source control bit TCS (TxCON<1>) = 1. Type B timers automatically provide synchronization for the external clock source; however, the Type A timer does not, and requires the external clock synchronization bit TSYNC (T1CON<2>) be set = 1.

Type A and B timers that use a 1:1 clock prescale increment the TMR count register on every rising external clock edge after synchronization. The timer continues to increment until the TMR count register matches the PR period register value. The TMR count register resets to 0000h on the next timer clock cycle, then continues to increment and repeat the period match until the timer is disabled. If the PR period register value = 0000h, the TMR count register resets to 0000h on the next timer clock cycle, but will not continue to increment.

Type A and B timers using a clock prescale = N (other than 1:1) operate at a timer clock rate (external clock/N) and the TMR count register increments on every Nth external clock rising edge after synchronization. For example, if the clock prescale is 1:8, then the timer increments on every 8th external clock cycle. The timer continues to increment until the TMR count register matches the PR period register value. The TMR count register then resets to 0000h after N more external clock cycles, then continues to increment and repeat the period match until the timer is disabled. If the PR period register value = 0000h, the TMR count register resets to 0000h on the next external clock cycle, but does not continue to increment.

Type A timers generate a timer event one-half timer clock cycle (on the falling edge) after the TMR count register matches the PR period register value. Type B timers generate a timer event within 1 PBCLK + 2 SYSCLK system clock cycles after the TMR count register matches the PR period register value. Both Type A and B timer interrupt flag bits, TxIF, are set within 1 PBCLK + 2 SYSCLK cycles of this event and if the timer interrupt enable bit TxIE is set, an interrupt is generated.

14.3.5.1 16-Bit Synchronous External Clock Counter Considerations

This section describes items that should be considered when using the 16-bit Synchronous External Clock Counter.

Type A or Type B timers operating from a synchronized external clock source will not operate in SLEEP mode, since the synchronization circuit is disabled during SLEEP mode.

Type A and B Timers using a clock prescale = N (other than 1:1) require 2 to 3 external clock cycles, after the ON bit = 1, before the TMR count register increments. Refer to **Section 14.3.12 “Timer Latency Considerations”** for more information.

When operating the timer in Synchronous Counter mode, the external input clock must meet certain minimum high time and low time requirements. Refer to the device data sheet **“Electrical Specifications”** section for further details.

14.3.6 32-Bit Synchronous External Clock Counter Mode

The 32-bit Synchronous External Clock counter operation provides the following capabilities:

- Counting large number of periodic or non-periodic pulses
- Use external clock as large time base for timers

Only Type B timers have the ability to operate in 32-bit Synchronous External Clock Counter mode. To enable 32-bit Synchronous External Clock Counter operation, a Type B (TimerX) T32 control bit (TxCON<3>) must be set = 1. In this mode, the input clock source for the timer is an external clock applied to the TxCK pin and is selected by setting the clock source control bit TCS (TxCON<1>) = 1. Type B timers automatically provide synchronization for the external clock source.

Type B timers that use a 1:1 clock prescale increment the TMRxy count register on every rising external clock edge after synchronization. The timer continues to increment until the TMRxy count register matches the PRxy period register value. The TMRxy count register resets to 00000000h on the next timer clock cycle, then continues to increment and repeat the period match until the timer is disabled. If the PRxy period register value = 0000h, the TMR count register resets to 00000000h on the next timer clock cycle, but does not continue to increment.

Type B timers that use a clock prescale = N (other than 1:1) operate at a timer clock rate (external clock/N) and the TMRxy count register increments on every Nth external clock rising edge after sychronization. For example, if the clock prescale is 1:8, then the timer increments on every 8th external clock cycle. The timer continues to increment until the TMRxy count register matches the PRxy period register value. The TMRxy count register resets to 0000h after N more external clock cycles, then continues to increment and repeat the period match until the timer is disabled. If the PRxy period register value = 00000000h, the TMRxy count register resets to 00000000h on the next external clock cycle, but does not continue to increment.

Type B timers generate a timer event within 1 PBCLK + 2 SYSCLK system clock cycles after the TMRxy count register matches the PRxy period register value. The Type B timer interrupt flag bit, TyIF, is set within 1 PBCLK + 2 SYSCLK cycles of this event and if the timer interrupt enable bit TyIE is set, an interrupt is generated.

14.3.6.1 32-Bit Synchronous External Clock Counter Considerations

This section describes items that should be considered when using the 32-bit Synchronous External Clock Counter.

Type B timers operating from a synchronized external clock source will not operate in SLEEP mode, since the synchronization circuit is disabled during SLEEP mode.

Type B timers using a clock prescale = N (other than 1:1) require 2 to 3 external clock cycles, after the ON bit = 1, before the TMR count register increments. Refer to **Section 14.3.12 “Timer Latency Considerations”**.

When operating the timer in Synchronous Counter mode, the external input clock must meet certain minimum high time and low time requirements. Refer to the device data sheet **“Electrical Specifications”** section for further details.

14.3.6.2 16-Bit Synchronous External Counter Initialization Steps

Perform the following steps to configure the timer for 16-bit Synchronous Counter mode:

1. Clear control bit ON (TxCON<15> = 0) to disable timer.
2. Set control bit TCS (TxCON<1> = 1) to select external clock source.
3. If Type A Timer, set control bit TSYNC (T1CON<2> = 1) to enable clock synchronization.
4. Select desired clock prescale.
5. Load/Clear timer register TMRx.
6. If using period match:
   a. Load period register PRx with desired 16-bit match value.
7. If interrupts are used:
   i. Clear interrupt flag bit TxIF in IFS0 register.
   ii. Configure interrupt priority and subpriority levels in IPCn register.
   iii. Set interrupt enable bit TxIE in IEC0 registers.
8. Set control bit ON (TxCON<15> = 1) to enable the timer.

Example 14-3: 16-Bit Synchronous External Counter Example Code

```
T3CON = 0x0;              // Stop Timer and clear control register
T3CONSET = 0x0072;        // Set prescaler at 1:256, external clock source
TMR3 = 0x0;               // Clear timer register
PR3 = 0x3FFF;             // Load period register
T3CONSET = 0x8000;        // Start Timer
```

14.3.6.3 32-Bit Synchronous External Clock Counter Initialization Steps

Perform the following steps to configure the timer for 32-bit Synchronous External Clock Counter mode:

1. Clear control bit ON (TxCON<15> = 0) to disable timer.
2. Set control bits TCS (TxCON<1> = 1) to select external clock source.
3. Set T32 (TxCON<3> = 1) to enable 32-bit operations.
4. Select desired clock prescale.
5. Load/Clear timer register TMRxy.
6. Load period register PRxy with desired 32-bit match value.
7. If interrupts are used:
   i. Clear interrupt flag bit TyIF in IFSn register.
   ii. Configure interrupt priority and subpriority levels in IPCn register.
   iii. Set interrupt enable bit TyIE in IECn registers.
8. Set control bit ON (TxCON<15> = 1) to enable the timer.

Example 14-4: 32-Bit Synchronous External Clock Counter Example Code

```
T4CON = 0x0;              // Stop any 16/32-bit Timer4 operation
T5CON = 0x0;              // Stop any 16-bit Timer5 operation
T4CONSET = 0x006A;        // 32-bit mode, external clock, 1:64 prescale
TMR4 = 0x0;               // Clear contents of the TMR4 and TMR5

PR4 = 0xFFFFFFFF;         // Load PR4 and PR5 registers with 32-bit value

T4CONSET = 0x8000;        // Start 32-bit timer
```

14.3.7 16-Bit Gated Timer Mode

The gate operation starts on a rising edge of the signal applied to the TxCK pin. The TMRx count register increments while the external gate signal remains high. The gate operation terminates on the falling edge of the signal applied to the TxCK pin. The timer interrupt flag, TxIF, is set.

Both Type A and B timers can operate in Gated Timer mode. The timer clock source is the internal peripheral bus clock, PBCLK, and is selected by clearing the TCS control bit = 0, (TxCON<1>). Type A and B Timers automatically provide synchronization to the peripheral bus clock, therefore the Type A Timer Synchronous mode control bit TSYNC (T1CON<2>) is ignored in this mode. In Gated Timer mode, the input clock is gated by the signal applied to the TxCK pin. The Gated Timer mode is enabled by setting the TGATE control bit = 1, (TxCON<7>).

Type A and B timers using a 1:1 clock prescale operate at a timer clock rate the same as the PBCLK and increment the TMR count register on every rising timer clock edge. The timer continues to increment until the TMR count register matches the PR period register value. The TMR count register then resets to 0000h on the next timer clock cycle, then continues to increment and repeat the period match until the falling edge of the gate signal or the timer is disabled. The timer does not generate an interrupt when a timer period match occurs.

Type A and B timers using a clock prescale = N (other than 1:1) operate at a timer clock rate (PBCLK/N) and the TMR count register increments on every Nth timer clock rising edge. For example, if the clock prescale is 1:8, then the timer increments on every 8th timer clock cycle. The timer continues to increment until the TMR count register matches the PR period register value. The TMR count register then resets to 0000h after N more timer clock cycles, and continues to increment and repeat the period match until the falling edge of the gate signal or the timer is disabled. The timer does not generate an interrupt when a timer period match occurs.

On the falling edge of the gate signal, the count operations terminates, a Timer event is generated and the interrupt flag bit TxIF is set 1 PBCLK + 2 SYSCLK system clock cycles after the falling edge of the signal on the gate pin. The TMR count register is not reset to 0000h. The user must reset the TMR count register if it is desired to start from zero on the next rising edge gate input.

The resolution of the timer count is directly related to the timer clock period. When the timer prescaler is 1:1, the timer clock period is one peripheral bus clock cycle T_{PBCLK}. For a timer prescaler of 1:8, the timer clock period is 8 times the peripheral bus clock cycle.

14.3.7.1 Special Gated Timer Mode Considerations

This section describes items that should be considered when using the special Gated Timer mode.

Gated Timer mode is overridden if the clock source bit TCS is set to external clock source, TCS = 1. For Gated Timer operation, the internal clock source must be selected, TCS = 0.

Type A and B timers using a clock prescale = N (other than 1:1) require 2 to 3 timer clock cycles, after the ON bit = 1, before the TMR count register increments. Refer to **Section 14.3.12 “Timer Latency Considerations”** for more information.

Refer to the **“Electrical Specifications”** section in the device data sheet for details on the gate width pulse requirements.

14.3.8 32-Bit Gated Timer Mode

The gate operation starts on a rising edge of the signal applied to the TxCK pin. The TMRx count register increments while the external gate signal remains high. The gate operation terminates on the falling edge of the signal applied to the TxCK pin. The timer interrupt flag, TyIF, is set.

Only Type B timers can operate in 32-bit Gated Timer mode. The timer clock source is the internal peripheral bus clock, PBCLK, and is selected by clearing the TCS control bit = 0, (TxCON<1>). Type B timers automatically provide synchronization to the peripheral bus clock. In 32-bit Gated Timer mode, the input clock is gated by the signal applied to the TxCK pin. The Gated Timer mode is enabled by setting the TGATE control bit (TxCON<7>) = 1.

The gate operation starts on a rising edge of the signal applied to the TxCK pin and the TMRxy count register increments while the external gate signal remains high.

Type B timers using a 1:1 clock prescale operate at a timer clock rate the same as the PBCLK and increment the TMRxy count register on every rising timer clock edge. The timer continues to increment until the TMRxy count register matches the PRxy period register value. The TMRxy count register then resets to 00000000h on the next timer clock cycle, then continues to increment and repeat the period match until the falling edge of the gate signal or the timer is disabled. The timer does not generate an interrupt when a timer period match occurs.

Type B timers using a clock prescale = N (other than 1:1) operate at a timer clock rate (PBCLK/N) and the TMRxy count register increments on every Nth timer clock rising edge. For example, if the clock prescale is 1:8, then the timer increments on every 8th timer clock cycle. The timer continues to increment until the TMRxy count register matches the PRxy period register value. The TMRxy count register then resets to 00000000h after N more timer clock cycles, then continues to increment and repeat the period match until the falling edge of the gate signal or the timer is disabled. The timer does not generate an interrupt when a timer period match occurs.

On the falling edge of the gate signal, the count operations terminate, a timer event is generated, and the interrupt flag bit TyIF is set 1 PBCLK + 2 SYSCLK system clock cycles after the falling edge of the signal on the gate pin. The TMR count register is not reset to 00000000h. The user must reset the TMRxy count register if it is desired to start from zero on the next rising edge gate input.

The resolution of the timer count is directly related to the timer clock period. When the timer prescaler is 1:1, the timer clock period is 1 PBCLK peripheral bus clock cycle. For a timer prescaler of 1:8, the timer clock period is 8 times the peripheral bus clock cycle.

14.3.8.1 32-Bit Gated Timer Mode Considerations

This section describes items that should be considered when using the 32-bit Gated Timer mode.

Gated Timer mode is overridden if the clock source bit TCS is set to external clock source, TCS = 1. For Gated Timer operation, the internal clock source must be selected, TCS = 0.

Refer to the **"Electrical Specifications"** section in the device data sheet for details on the gate width pulse requirements.

14.3.8.2 16-Bit Gated Timer Initialization Steps

Perform the following steps to configure the timer for 16-bit Gated Timer mode:

1. Clear control bit ON (TxCON<15> = 0) to disable timer.
2. Set control bits TCS (TxCON<1> = 0) to select internal PBCLK source.
3. Set control bit TGATE (T1CON<7> = 1) to enable gated Timer mode.
4. Select desired prescaler.
5. Clear timer register TMRx.
6. Load period register PRx with desired 16-bit match value.
7. If interrupts are used:
   i. Clear interrupt flag bit TxIF in IFS0 register.
   ii. Configure interrupt priority and subpriority levels in IPCn register.
   iii. Set Interrupt enable bit TxIE in IEC0 registers.
8. Set control bit ON (TxCON<15> = 1) to enable the timer.

Example 14-5: 16-Bit Gated Timer Example Code

```
T4CON = 0x0;              // Stop Timer and clear control register
T4CON = 0x00E0;           // Gated timer mode, prescaler at 1:64, internal clock source
TMR4 = 0;                 // Clear timer register
PR4 = 0xFFFF;             // Load period register with 16-bit match value
T4CONSET = 0x8000;        // Start Timer
```

14.3.8.3 32-Bit Gated Timer Initialization Steps

Perform the following steps to configure the timer for 32-bit Gated Timer Accumulation mode:

1. Clear control bit ON (TxCON<15> = 0) to disable Timer.
2. Clear control bit TCS (TxCON<1>) = 0 to select internal PBCLK source.
3. Set control bit T32 (TxCON<3>= 1) = 1 to enable 32-bit operations.
4. Set control bit TGATE (TxCON<7> = 1) to enable gated Timer mode.
5. Select desired clock prescale.
6. Load/Clear timer register TMRx.
7. Load period register PRx with desired 32-bit match value.
8. If interrupts are used:
   i. Clear interrupt flag bit TyIF in IFSn register.
   ii. Configure interrupt priority and subpriority levels in IPCn register.
   iii. Set interrupt enable bit TyIE in IECn registers.

Set control bit ON (TxCON<15> = 1) to enable the timer.

Example 14-6: 32-Bit Gated Timer Example Code

```
T2CON = 0x0;              // Stops any 16/32-bit Timer2 operation
T3CON = 0x0;              // Stops any 16-bit Timer3 operation
T2CONSET = 0x00C8;        // 32-bit mode, gate enable, internal clock,
                          // 1:16 prescale
TMR2 = 0x0;               // Clear contents of the TMR2 and TMR3

PR2 = 0xFFFFFFFF;         // Load PR2 and PR3 registers with 32-bit match value

T2CONSET = 0x8000;        // Start 32-bit timer
```


14.3.9 Asynchronous Clock Counter Mode (Type A Timer Only)

The Asynchronous Timer operation provides the following capabilities:

- The timer can operate during SLEEP mode and can generate an interrupt on period register match that will wake-up the processor from SLEEP or IDLE mode.
- The timer can be clocked from the Secondary Oscillator for real-time clock applications.

The Type A timer has the ability to operate in an Asynchronous Counting mode, using an external clock source connected to the T1CK pin, and is selected by setting the clock source control bit TCS (TxCON<1>) = 1. This requires the external clock synchronization be disabled, bit TSYNC (T1CON<2>) = 0. It is also possible to utilize the Secondary Oscillator with a 32 kHz crystal connected to SOSCI/SOSCO pins as an asynchronous clock source. Refer to **Section 14.3.13 "Secondary Oscillator"** for more information.

Type A timer using a 1:1 clock prescale operates at the same clock rate as the applied external clock rate, and increments the TMR count register on every rising timer clock edge. The timer continues to increment until the TMR count register matches the PR period register value. The TMR count register resets to 0000h on the next timer clock cycle, then continues to increment and repeat the period match until the timer is disabled. If the PR period register value = 0000h, the TMR count register resets to 0000h on the next timer clock cycle, but will not continue to increment.

Type A timers generate a timer event when the TMR count register matches the PR period register value. The timer interrupt flag bit, TxIF is set within 1 PBCLK + 2 SYSCLK system clock cycles of this event. If the timer interrupt enable bit is set, TxIE = 1, an interrupt is generated.

14.3.9.1 Asynchronous Mode TMR1 Read and Write Operations

Due to the asynchronous nature of Timer1 operating in this mode, reading and writing to the TMR1 count register requires synchronization between the asynchronous clock source and the internal PBCLK peripheral bus clock. Timer1 features a TWDIS (Timer Write Disable) control bit (T1CON<12>) and a TWIP (TImer Write in Progress) Status bit (T1CON<11>) to provide the user with 2 options for safely writing to the TMR1 count register while Timer1 is enabled. These bits have no affect in Synchronous Clock Counter modes.

Option 1 is the legacy Timer1 Write mode, TWDIS bit = 0. To determine when it is safe to write to the TMR1 count register, it is recommended to poll the TWIP bit. When TWIP = 0, it is safe to perform the next write operation to the TMR1 count register. When TWIP = 1, the previous Write operation to the TMR1 count register is still being synchronized and any additional write operations should wait until TWIP = 0.

Option 2 is the new synchronized Timer1 Write mode, TWDIS bit = 1. A write to the TMR1 count register can be performed at any time. However, if the previous write operation to the TMR1 count register is still being synchronized, any additional write operations are ignored.

When performing a write to the TMR1 count register, 2 to 3 asynchronous external clock cycles are required for the value to be synchronized into the register.

When performing a read from the TMR1 count register, synchronization requires 2 PBCLK cycle delays between the current unsynchronized value in the TMR1 count register and the synchronized value returned by the read operation. In other words, the value read is always 2 PBCLK cycles behind the actual value in the TMR1 count register.

14.3.9.2 Asynchronous Clock Counter Considerations

This section describes items that should be considered when using the Asynchronous Clock Counter.

Regardless of the clock prescale, Type A timers require 2 to 3 timer clock cycles, after the ON bit = 1 before the TMR count register increments. Refer to **Section 14.3.12 "Timer Latency Considerations"** for more information.

The external input clock must meet certain minimum high time and low time requirements when used in the Asynchronous Counter mode. Refer to the device data sheet **"Electrical Specifications"** section for further details.

14.3.9.3 Asynchronous External Clock Counter Initialization Steps

Perform the following steps to configure the Timer for 16-bit Asynchronous Counter mode.

1. Clear control bit ON (T1CON<15> = 0) to disable timer.
2. Set control bit TCS (T1CON<1> = 1) to enable external clock source.
3. Clear control bit TSYNC (T1CON<2> = 0) to disable clock synchronization.
4. Select desired prescaler.
5. Load/Clear timer register TMR1.
6. If using period match:
   i. Load period register PR1 with desired 16-bit match value.
7. If interrupts are used:
   i. Clear interrupt flag bit T1IF in IFS0 register.
   ii. Configure interrupt priority and subpriority levels in IPCn register.
   iii. Set interrupt enable bit T1IE in IEC0 registers.
8. Set control bit ON (T1CON<15> = 1) to enable the timer.

Example 14-7: Example Code: 16-Bit Asynchronous Counter Mode

```
/*
    16-bit asynchronous counter mode example
*/
    T1CON = 0x0;           // Stops the Timer1 and reset control reg.
    T1CON = 0x0042;        // Set prescaler 1:16, external clock, asynch mode

    TMR1 = 0x0;            // Clear timer register
    PR1 = 0x7FFF;          // Load period register
    T1CONSET = 0x8000;     // Start Timer
```


14.3.10 Timer Prescalers

Type A timers provide input clock (peripheral bus clock or external clock) prescale options of 1:1, 1:8, 1:64 and 1:256 selected using TCKPS<1:0> (TxCON<5:4>).

Type B timers provide input clock (peripheral bus clock or external clock) prescale options of 1:1, 1:2, 1:4, 1:8, 1:16, 1:32, 1:64 and 1:256 selected using TCKPS<2:0> (TxCON<6:4>).

The prescaler counter is cleared when any of the following occurs:

- A write to the TMRx register
- Disabling the timer, ON (TxCON<15>) = 0
- Any device Reset, except Power-on Reset

14.3.11 Writing to TxCON, TMR, and PR Registers

A timer module is disabled and powered off when the ON bit (TxCON<15>) = 0, thus providing maximum power savings.

To prevent unpredictable timer behavior, it is recommended that the timer be disabled, ON bit = 0, before writing to any of the TxCON register bits or timer prescaler. Attempting to set ON bit = 1 and write to any TxCON register bits in the same instruction may cause erroneous timer operation.

The PRx period register can be written to while the module is operating. However, to prevent unintended period matches, writing to the PRx period register while the timer is enabled, (ON bit = 1)is not recommended.

The TMRx count register can be written to while the module is operating. The user should be aware of the following when byte writes are performed:

- If the timer is incrementing and the low byte of the timer is written to, the upper byte of the timer is not affected. If 0xFF is written into the low byte of the timer, the next timer count clock after this write will cause the low byte to rollover to 0x00 and generate a carry into the high byte of the timer.
- If the timer is incrementing and the high byte of the timer is written to, the low byte of the timer is not affected. If the low byte of the timer contains 0xFF when the write occurs, the next timer count clock will generate a carry from the timer low byte and this carry will cause the upper byte of the timer to increment.

Additionally, TMR1 count register can be written to while the module is operating. However, see **Section 14.3.9.1 "Reading and Writing TMR1 register"** regarding asynchronous clock operations.

When the TMRx register is written to (a word, half word, or byte) via an instruction, the TMRx register increment is masked and does not occur during that instruction cycle.

A TMR count register is not reset to zero when the module is disabled.

14.3.12 Timer Latency Considerations

This section describes items that should be considered regarding timer latency.

Since both Type A and Type B timers can use the Internal Peripheral Bus Clock (PBCLK) or an external clock (Type A also supports asynchronous clock), there are considerations regarding latencies of operations performed on the timer. These latencies represent the time delay between the moment an operation is executed (read or write) and the moment its first effect begins, as shown in Table 14-3 and Table 14-4.

For Type A and Type B timers, reading and writing the TxCON, TMRx, and PRx registers in any Synchronized Clock mode does not require synchronization of data between the main SYSCLK clock domain and the timer module clock domain. Therefore, the operation is immediate. However, when operating Timer1 in Asynchronous Clock mode, reading the TMR1 count register requires 2 PBCLK cycles for synchronization, while writing to theTMR1 count register requires 2 to 3 timer clock cycles for synchronization.

For example, Timer1 is using an asynchronous clock source and a read operation of TMR1 register is executed. There are 2 PBCLK peripheral bus clocks required to synchronize this data to the TMR1 count register. The effect is a value which is always 2 PBCLK cycles behind the actual TMR1 count.

Additionally, any timer using an external clock source requires 2-3 external clock cycles, after the ON bit (TxCON<15>) has been set (= 1), before the timer starts incrementing.

The interrupt flag latency represents the time delay between the timer event and the moment the timer interrupt flag is active.

Table 14-3: Type A Timer Latencies

Operation	PBCLK Internal clock	Synchronous External clock	Asynchronous External clock
Set ON = 1 (enable timer)	0 PBCLK	2-3 $TMRCLK_{CY}$	2-3 $TMRCLK_{CY}$
Set ON = 0 (disable timer)	0 PBCLK	2-3 $TMRCLK_{CY}$	2-3 $TMRCLK_{CY}$
Read PRx	0 PBCLK	0 PBCLK	0 PBCLK
Write PRx	0 PBCLK	0 PBCLK	0 PBCLK
Read TMRx	0 PBCLK	0 PBCLK	2 PBCLK
Write TMRx	0 PBCLK	0 PBCLK	2-3 $TMRCLK_{CY}$
Interrupt Flag INTF = 1	1 PBCLK + 2 to 3 SYSCLK	1 PBCLK + 2 to 3 SYSCLK	($TMRCLK_{CY}$ / 2) + 2 to 3 SYSCLK

Note: $TMRCLK_{CY}$ = External synchronous or asynchronous timer clock cycles.

Table 14-4: Type B Timer Latencies

Operation	PBCLK Internal clock	Synchronous External clock
Set ON = 1 (enable timer)	0 PBCLK	0 PBCLK
Set ON = 0 (disable timer)	0 PBCLK	0 PBCLK
Read PRx	0 PBCLK	0 PBCLK
Write PRx	0 PBCLK	0 PBCLK
Read TMRx	0 PBCLK	0 PBCLK
Write TMRx	0 PBCLK	0 PBCLK
Interrupt Flag INTF = 1	1 PBCLK + 2 to 3 SYSCLKs	1 PBCLK + 2 to 3 SYSCLKs

14.3.13 Secondary Oscillator

In each device variant, the secondary oscillator is available to the Type A timer module for Real-Time Clock (RTC) applications.

- The secondary oscillator becomes the clock source for the timer when the secondary oscillator is enabled and the timer is configured to use the external clock source.
- The secondary oscillator is enabled when the Configuration Fuse bit FSOSCEN (DEVCFG1<5>) = 0 and by setting the SOSCEN control bit (OSCCON<1>).

Refer to **Section 6. "Oscillators"** for further details.

14.4 INTERRUPTS

A timer has the ability to generate an interrupt on a period match or falling edge of the external gate signal, depending on the operating mode.

The TxIF bit (TyIF bit in 32-bit mode) is set when one of the following conditions is true:

- When the timer count matches the respective period register and the timer module is not operating in Gated Time Accumulation mode.
- When the falling edge of the gate signal is detected when the timer is operating in Gated Time Accumulation mode.

The TxIF bit (TyIF bit in 32-bit mode) must be cleared in software.

A timer is enabled as a source of interrupt via the respective timer interrupt enable bit, TxIE (TyIE for 32-bit mode). The interrupt priority level bits TxIP<2:0> (TyIP<2:0> for 32-bit mode) and interrupt subpriority level bits TxIS<1:0> (TyIS<1:0> for 32-bit mode) also must be configured. Refer to **Section 8. "Interrupts"** in this manual for further details.

Note:	A special case occurs when the period register is loaded with '0' and the timer is enabled. No timer interrupts will be generated for this configuration.

14.4.1 Interrupt Configuration

Each Time Base module has a dedicated interrupt flag bit TxIF and a corresponding interrupt enable/mask bit TxIE. These bits determine the source of an interrupt, and enable or disable an individual interrupt source. Each Timer module can have its own priority level independent of other Timer modules.

The TxIF is set when the timer count matches the respective period register and the timer module is not operating in Gated Time Accumulation mode, or when the falling edge of the gate signal is detected when the timer is operating in Gated Time Accumulation mode. The TxIF bit is set without regard to the state of the corresponding TxIE bit. The TxIF bit can be polled by software if desired.

The TxIE bit is used to define the behavior of the Interrupt Controller when a corresponding TxIF is set. When the TxIE bit is clear, the Interrupt Controller does not generate a CPU interrupt for the event. If the TxIE bit is set, the Interrupt Controller will generate an interrupt to the CPU when the corresponding TxIF bit is set (subject to the priority and subpriority as outlined below).

It is the responsibility of the user's software routine that services a particular interrupt to clear the appropriate Interrupt Flag bit before the service routine is complete.

The priority of each timer module can be set independently with the TxIP<2:0> bits. This priority defines the priority group to which the interrupt source will be assigned. The priority groups range from a value of 7 (the highest priority) to a value of 0 (which does not generate an interrupt). An interrupt being serviced will be preempted by an interrupt in a higher priority group.

The subpriority bits allow setting the priority of a interrupt source within a priority group. The values of the subpriority, TxIS<1:0>, range from 3 (the highest priority), to 0 (the lowest priority). An interrupt with the same priority group, but having a higher subpriority value, will preempt a lower subpriority interrupt that is in progress.

The priority group and subpriority bits allow more than one interrupt source to share the same priority and subpriority. If simultaneous interrupts occur in this configuration, the natural order of the interrupt sources within a priority/subgroup pair determines the interrupt generated. The natural priority is based on the vector numbers of the interrupt sources. The lower the vector number, the higher the natural priority of the interrupt. Any interrupts that were overridden by natural order will then generate their respective interrupts based on priority, subpriority, and natural order after the interrupt flag for the current interrupt is cleared.

After an enabled interrupt is generated, the CPU will jump to the vector assigned to that interrupt. The vector number for the interrupt is the same as the natural order number. The CPU will then begin executing code at the vector address. The user's code at this vector address should perform any application specific operations and clear the TxIF interrupt flag, and then exit. Refer to **Section 8. "Interrupts"** for the vector address table details for more information on interrupts.

Table 14-5: Timer Interrupt Vectors for Various Offsets with EBASE = 0x8000:0000

Interrupt	Vector/Natural Order	IRQ Number	Vector Address IntCtl.VS = 0x01	Vector Address IntCtl.VS = 0x02	Vector Address IntCtl.VS = 0x04	Vector Address IntCtl.VS = 0x08	Vector Address IntCtl.VS = 0x10
Timer1	4	4	8000_0280	8000 0300	8000_0400	8000_0600	8000 0A00
Timer2	8	8	8000_0300	8000_0400	8000_0600	8000_0A00	8000 1200
Timer3	12	12	8000_0380	8000_0500	8000_0800	8000_0E00	8000 1A00
Timer4	16	16	8000_0400	8000_0600	8000_0A00	8000 1200	8000 2200
Timer5	20	20	8000_0480	8000_0700	8000_0C00	8000 1600	8000 2A00

Table 14-6: Example of Priority and Subpriority Assignment

Interrupt	Priority Group	Subpriority	Vector/Natural Order
Timer1	7	3	4
Timer2	7	3	8
Timer3	7	2	12
Timer4	6	1	16
Timer5	0	3	20

Example 14-8: 16-Bit Timer Interrupt Initialization Code Example

```
/*
    The following code example will enable Timer2 interrupts, load the Timer2 Period
    register and start the Timer.

    When a Timer2 period match interrupt occurs, the interrupt service routine must clear
    the Timer2 interrupt status flag in software.
*/
    T2CON = 0x0;           // Stop Timer and clear control register,
                           // prescaler at 1:1,internal clock source

    TMR2 = 0x0;            // Clear timer register
    PR2 = 0xFFFF;          // Load period register

    IPC2SET = 0x0000000C;  // Set priority level=3
    IPC2SET = 0x00000001;  // Set sub-priority level=1
                           // Could have also done this in single
                           // operation by assigning IPC2SET = 0x0000000D

    IFS0CLR = 0x00000100;  // Clear Timer interrupt status flag
    IEC0SET = 0x00000100;  // Enable Timer interrupts

    T2CONSET = 0x8000;     // Start Timer
```

Example 14-9: Timer ISR Code Example

```
/*
    The following code example demonstrates a simple interrupt service routine for Timer
    interrupts. The user's code at this ISR handler should perform any application
    specific operations and must clear the corresponding Timer interrupt status flag
    before exiting.
*/
void __ISR(_Timer_1_Vector,ipl3)Timer1Handler(void)
{
    ... perform application specific operations in response to the interrupt

    IFS0CLR = 0x00000010;  // Be sure to clear the Timer 2 interrupt status
}
```

Note: The Timer ISR code example shows MPLAB® C32 C-compiler specific syntax. Refer to your compiler manual regarding support for ISRs.

Example 14-10: 32-bit Timer Interrupt Initialization Code Example

```
/*
    The following code example will enable Timer5 interrupts, load the Timer4:Timer5 Period
    Register pair and start the 32-bit timer module.

    When a 32-bit period match interrupt occurs, the user must clear the Timer5 interrupt
    status flag in software.
 */

    T4CON = 0x0;           // Stop 16-bit Timer4 and clear control register
    T5CON = 0x0;           // Stop 16-bit Timer5 and clear control register
    T4CONSET = 0x0038;     // Enable 32-bit mode, prescaler at 1:8,
                           // internal clock source

    TMR4 = 0x0;            // Clear contents of the TMR4 and TMR5

    PR4 = 0xFFFFFFFF;      // Load PR4 and PR5 registers with 32-bit value

    IPC5SET = 0x00000004;  // Set priority level=1 and
    IPC5SET = 0x00000001;  // Set sub-priority level=1
                           // Could have also done this in single
                           // operation by assigning IPC5SET = 0x00000005

    IFS0CLR = 0x00100000;  // Clear the Timer5 interrupt status flag
    IEC0SET = 0x00100000;  // Enable Timer5 interrupts

    T4CONSET = 0x8000;     // Start Timer
```


14.5 OPERATION IN POWER-SAVING AND DEBUG MODES

Note: In this manual, a distinction is made between a power mode as it is used in a specific module, and a power mode as it is used by the device, e.g., Sleep mode of the Comparator and SLEEP mode of the CPU. To indicate which type of power mode is intended, uppercase and lowercase letters (Sleep, Idle, Debug) signify a module power mode, and all uppercase letters (SLEEP, IDLE, DEBUG) signify a device power mode.

14.5.1 Timer Operation in SLEEP Mode

As the device enters SLEEP mode, the system clock SYSCLK and peripheral bus clock PBCLK are disabled. For both timer types (A and B) operating in Synchronous mode, the timer module stops operating.

Type A timer module is different from the Type B timer module because it can operate asynchronously from an external clock source. Because of this distinction, the Type A timer module can continue to operate during SLEEP mode.

To operate in SLEEP mode, Type A timer module must be configured as follows:

- Timer1 module is enabled, ON (T1CON<15> = 1) and
- Timer1 clock source is selected as external, TCS (T1CON<1> = 1) and
- TSYNC bit (T1CON<2>) is set to logic '0' (Asynchronous Counter mode enabled).

When all of the preceding conditions are met, Timer1 continues to count and detect period matches when the device is in SLEEP mode. When a match between the timer and the period register occurs, the T1IF Status bit is set. If the T1IE bit is set, and its priority is greater than current CPU priority, the device wakes from SLEEP or IDLE mode and executes the Timer1 Interrupt Service Routine.

If the assigned priority level of the Timer1 interrupt is less than, or equal to, the current CPU priority level, the CPU is not awakened and the device enters IDLE mode.

14.5.2 Timer Operation in IDLE Mode

When the device enters IDLE mode, the system clock sources remain functional and the CPU stops executing code. The timer modules can optionally continue to operate in IDLE mode.

The SIDL bit (TxCON<13>) selects whether the timer module stops in IDLE mode, or continues to operate normally. If TSIDL = 0, the module continues operation in IDLE mode. If SIDL = 1, the module stops in IDLE mode.

14.5.3 Timer Operation in DEBUG Mode

The FRZ bit (TxCON<14>) determines whether the timer module will run or stop while the CPU is executing debug exception code (i.e., the application is halted) in DEBUG mode. When FRZ = 0, the timer module continues to run, even when application is halted in DEBUG mode. When FRZ = 1 and the application is halted in DEBUG mode, the module freezes its operations and makes no changes to the state of the timer module. The module will resume its operation after the CPU resumes execution.

Note: The FRZ bit is readable and writable only when the CPU is executing in DEBUG mode. In all other modes, FRZ reads as '0'. If the FRZ bit is changed during DEBUG mode, the new value does not take effect until the current DEBUG mode is exited and reentered. During DEBUG mode, FRZ reads the last written value, which may or may not be in effect (depending on when the last value was written).

14.6 EFFECTS OF VARIOUS RESETS

14.6.1 Device Reset

All timer registers are forced to their reset states upon a device Reset.

14.6.2 Power-on Reset

All timer registers are forced to their reset states upon a Power-on Reset.

14.6.3 Watchdog Reset

All timer registers are forced to their reset states on a Watchdog Reset.

14.7 PERIPHERALS USING TIMER MODULES

14.7.1 Time Base for Input Capture/Output Compare

The Input Capture and Output Compare peripherals can select one of two timer modules or a combined 32-bit timer as their timer source. Refer to the device data sheet, and to **Section 15. "Input Capture"** and **Section 16. "Output Compare"** in this manual for details.

14.7.2 A/D Special Event Trigger

On each device variant, a Type B Timer3 or Timer5 has the capability to generate a special A/D conversion trigger signal on a period match in both 16-bit and 32-bit modes. The timer module provides a conversion Start signal to the A/D sampling logic.

- If T32 = 0 when a match occurs between the 16-bit timer register (TMRx) and the respective 16-bit period register (PRx), the A/D Special Event Trigger signal is generated.
- If T32 = 1 when a match occurs between the 32-bit timer (TMRx:TMRy) and the 32-bit respective combined period register (PRx:PRy), a A/D Special Event Trigger signal is generated.

The Special Event Trigger signal is always generated by the timer. The trigger source must be selected in the A/D converter control registers. Refer to the device data sheet, and to **Section 17. "10-Bit A/D Converter"** in this manual for additional information.

14.8 I/O PIN CONTROL

Enabling the timer module does not configure the I/O pin direction. When a timer module is enabled and configured for external clock or gate operation, the user must ensure the I/O pin direction is configured as an input by setting the corresponding TRIS control register bit = 1.

On PIC32MX devices, the TxCK pins become the gate inputs when Gated Timer mode is selected, TGATE bit (TxCON<7>) = 1, and internal peripheral bus clock source PBCLK, TCS bit (TxCON<1>) = 0, are selected. The TxCK pins can be external clock inputs for other modes when the external clock source TCS (TxCON<1>) = 1 is selected. If not used as a gate or external clock input, these pins can be general purpose I/O pins.

14.8.1 I/O Pin Resources

A summary of timer/counter modes, and the specific I/O pins required for each mode is provided in Table 14-7. The table illustrates which I/O pin is required for a certain mode of operation.

Refer to Table 14-8 to configure the I/O pins.

Table 14-7: Required I/O Pin Resources

	16/32-Bit Timer Modes			16/32-Bit Counter Modes
I/O Pin Name	**Internal Clock Source(1)**	**External Clock Source**	**Gate For Internal Clock Source**	**External Clock Source**
T1CK	No	Yes	Yes	Yes
T2CK	No	Yes	Yes	Yes
T3CK	No	Yes	Yes	Yes
T4CK	No	Yes	Yes	Yes
T5CK	No	Yes	Yes	Yes

Note 1: “No” indicates the pin is not required and can be used as a general purpose I/O pin.

14.8.2 I/O Pin Configuration

Table 14-8 provides a summary of I/O pin resources associated with the timer modules. The table also shows the settings required to make each I/O pin work with a specific timer module.

Table 14-8: I/O Pin Configuration for Use with Timer Modules

		Required Settings for Module Pin Control					
I/O Pin Name	**Required(1)**	**Module Control**	**Bit Field**	**TRIS**	**Pin Type**	**Buffer Type**	**Description**
T1CK	No	ON	TCS,TGATE	Input	I	ST	Timer 1 External Clock/Gate Input
T2CK	No	ON	TCS,TGATE	Input	I	ST	Timer 2 External Clock/Gate Input
T3CK	No	ON	TCS,TGATE	Input	I	ST	Timer 3 External Clock/Gate Input
T4CK	No	ON	TCS,TGATE	Input	I	ST	Timer 4 External Clock/Gate Input
T5CK	No	ON	TCS,TGATE	Input	I	ST	Timer 5 External Clock/Gate Input

Legend: CMOS = CMOS compatible input or output
ST = Schmitt Trigger input with CMOS levels
I = Input
O = Output

Note 1: These pins are only required for modes that use gated timer or external clock inputs. Otherwise, these pins can be used for general purpose I/O and require the user to set the corresponding TRIS control register bits.

14.9 FREQUENTLY ASKED QUESTIONS

Question 1: ***Can the lower half of the 32-bit timer generate an interrupt?***

Answer: No. When two 16-bit timers are combined in 32-bit mode (TxCON<TGATE> = 1), the interrupt enable bit TxIE, interrupt flag bit TxIF, interrupt priority bit TxIP, and interrupt subpriority bit TxIS associated with the upper timer module are used. The interrupt functions of the lower timer module are disabled.

Question 2: ***If I do not use the TxCK input for my timer mode, is this I/O pin available as a general purpose I/O pin?***

Answer: Yes. If the timer module is configured to use an internal clock source (TxCON<TCS = 0>) and not use the Gated Timer mode (TxCON<TGATE> = 0), then the associated I/O pin is available for general purpose I/O. Note, though, that when the I/O pin is used as a general purpose I/O pin, the user is responsible for configuring the respective TRIS register to input or output.

14.10 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the PIC32MX device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the Timers module are:

Title	Application Note #
No related application notes at this time.	N/A

Note: Please visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the PIC32MX family of devices.

14.11 REVISION HISTORY

Revision A (August 2007)

This is the initial released version of this document.

Revision B (October 2007)

Updated document to remove Confidential status.

Revision C (April 2008)

Revised status to Preliminary; Revised U-0 to r-x; Revised Table 14-2; Revised Register 14-1; Revised Section 14.3.9.1

Revision D (May 2008)

Added note to Registers 14-17, 14-18, 14-19, 14-20, 14-21, 14-22, 14-23; Revised Tables 14-1, 14-5; Revised Examples 14-9, 14-10; Revised Section 14.3.9.1 Title; Revised Section 14.3.11; Change Reserved bits from “Maintain as” to “Write”; Added Note to ON bit (T1CON, TxCON Registers).

NOTES:

Section 15. Input Capture

HIGHLIGHTS

This section of the manual contains the following topics:

15.1 INTRODUCTION

This section describes the Input Capture module and its associated Operational modes. The Input Capture module is used to capture a timer value from one of two selectable time bases on the occurrence of an event on an input pin. The Input Capture features are useful in applications requiring frequency (Time Period) and pulse measurement. Figure 15-1 depicts a simplified block diagram of the Input Capture module.

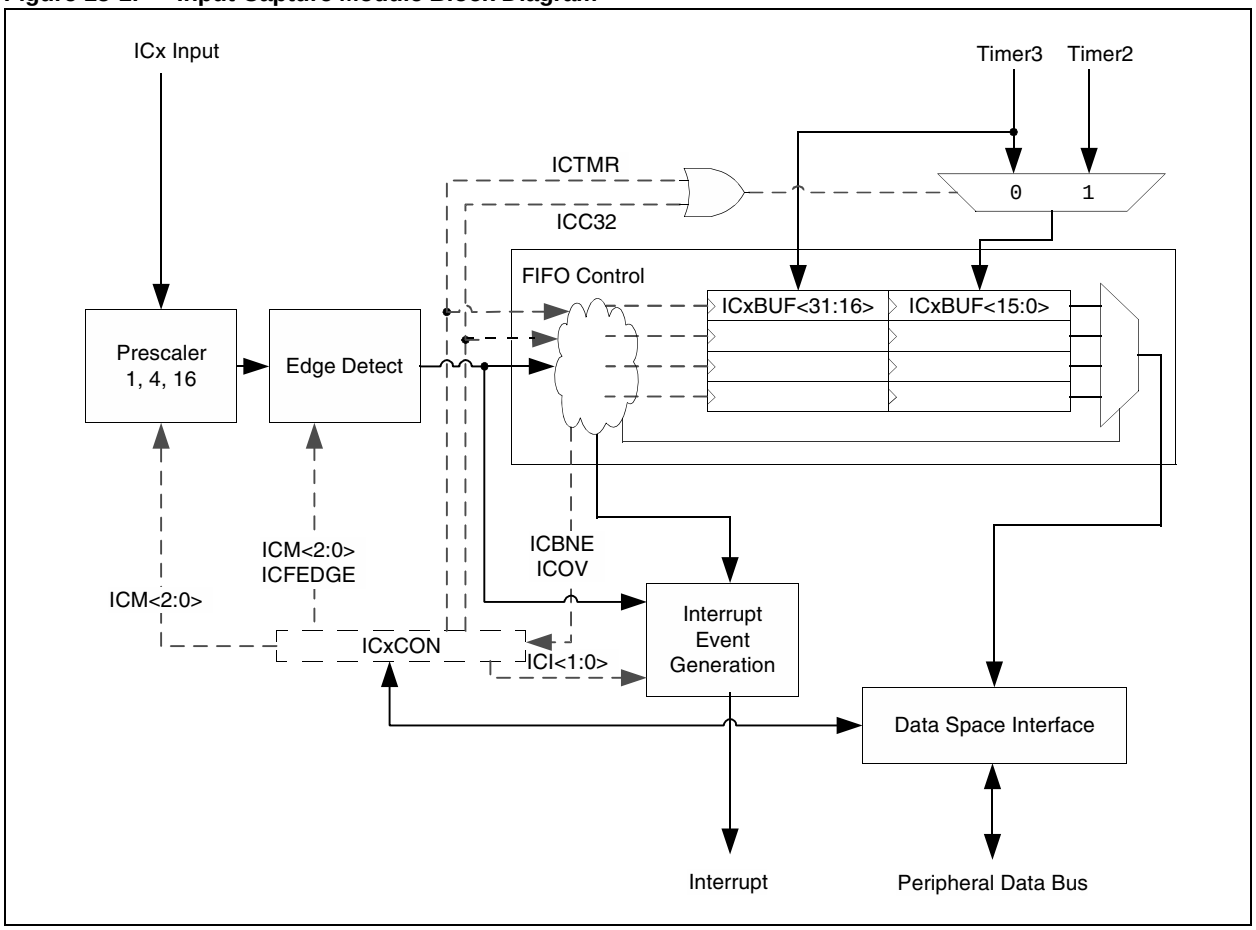
Refer to the specific device data sheet for information on the number of channels available in a particular device. All Input Capture channels are functionally identical. In this section, an 'x' in the pin name or register name denotes the specific Input Capture channel. Timer Y and Timer Z each refer to one of two timer inputs which may be associated with the Input Capture channel.

The Input Capture module has multiple operating modes, which are selected via the ICxCON register. The operating modes include the following:

- Capture timer value on every falling edge of input applied at the ICx pin
- Capture timer value on every rising edge of input applied at the ICx pin
- Capture timer value on every fourth rising edge of input applied at the ICx pin
- Capture timer value on every 16th rising edge of input applied at the ICx pin
- Capture timer value on every rising and falling edge of input applied at the ICx pin
- Capture timer value on the specified edge and every edge thereafter

The Input Capture module has a four-level FIFO buffer. The number of capture events required to generate a CPU interrupt can be selected by the user. An Input Capture Channel can also be configured to generate a CPU interrupt on a rising edge of the capture input when the device is in SLEEP or IDLE mode.

Figure 15-1: Input Capture Module Block Diagram



15.2 INPUT CAPTURE REGISTERS

Note: Each PIC32MX device variant may have one or more Input Capture modules. An 'x' used in the names of pins, control/Status bits and registers denotes the particular module. Refer to the specific device data sheets for more details.

Each capture module available on the PIC32MX devices has the following Special Function Registers (SFRs), where 'x' denotes the module number:

- ICxCON: Input Capture Control Register

  ICxCONCLR, ICxCONSET, ICxCONINV: Atomic Bit Manipulation Write-only Registers for ICxCON
- ICxBUF: Input Capture Buffer Register

Each Input capture module also has the following associated bits for interrupt control:

- Interrupt Enable Control bit (ICxIE)
- Interrupt Flag Status bit (ICxIF)
- Interrupt Priority Control bits (ICxIP)
- Interrupt Subpriority Control bits (ICxIS)

The tables below provide a brief summary of all the Input Capture related registers, and is followed by a detailed description of each register.

Table 15-1: Input Capture SFR Summary

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
ICxCON	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	ON	FRZ	SIDL	—	—	—	ICFEDGE	ICC32
	7:0	ICTMR	ICI<1:0>		ICOV	ICBNE	ICM<2:0>		
ICxCONCLR	31:0	Write clears selected bits in ICxCON, read yields undefined value							
ICxCONSET	31:0	Write sets selected bits in ICxCON, read yields undefined value							
ICxCONINV	31:0	Write inverts selected bits in ICxCON, read yields undefined value							
ICxBUF	31:24	ICxBUF<31:24>							
	23:16	ICxBUF<23:16>							
	15:8	ICxBUF<15:8>							
	7:0	ICxBUF<7:0>							
IFS0	31:24	I2C1MIF	I2C1SIF	I2C1BIF	U1TXIF	U1RXIF	U1EIF	SPI1RXIF	SPI1TXIF
	23:16	SPI1EIF	OC5IF	IC5IF	T5IF	INT4IF	OC4IF	IC4IF	T4IF
	15:8	INT3IF	OC3IF	IC3IF	T3IF	INT2IF	OC2IF	IC2IF	T2IF
	7:0	INT1IF	OC1IF	IC1IF	T1IF	INT0IF	CS1IF	CS0IF	CTIF
IEC0	31:24	I2C1MIE	I2C1SIE	I2C1BIE	U1TXIE	U1RXIE	U1EIE	SPI1RXIE	SPI1TXIE
	23:16	SPI1EIE	OC5IE	IC5IE	T5IE	INT4IE	OC4IE	IC4IE	T4IE
	15:8	INT3IE	OC3IE	IC3IE	T3IE	INT2IE	OC2IE	IC2IE	T2IE
	7:0	INT1IE	OC1IE	IC1IE	T1IE	INT0IE	CS1IE	CS0IE	CTIE
IPC1	31:24	—	—	—	INT1IP<2:0>			INT1IS<1:0>	
	23:16	—	—	—	OC1IP<2:0>			OC1IS<1:0>	
	15:8	—	—	—	IC1IP<2:0>			IC1IS<1:0>	
	7:0	—	—	—	T1IP<2:0>			T1IS<1:0>	
IPC2	31:24	—	—	—	INT2IP<2:0>			INT2IS<1:0>	
	23:16	—	—	—	OC2IP<2:0>			OC2IS<1:0>	
	15:8	—	—	—	IC2IP<2:0>			IC2IS<1:0>	
	7:0	—	—	—	T2IP<2:0>			T2IS<1:0>	

Table 15-1: Input Capture SFR Summary (Continued)

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
IPC3	31:24	—	—	—	INT3IP<2:0>			INT3IS<1:0>	
	23:16	—	—	—	OC3IP<2:0>			OC3IS<1:0>	
	15:8	—	—	—	IC3IP<2:0>			IC3IS<1:0>	
	7:0	—	—	—	T3IP<2:0>			T3IS<1:0>	
IPC4	31:24	—	—	—	INT4IP<2:0>			INT4IS<1:0>	
	23:16	—	—	—	OC4IP<2:0>			OC4IS<1:0>	
	15:8	—	—	—	IC4IP<2:0>			IC4IS<1:0>	
	7:0	—	—	—	T4IP<2:0>			T4IS<1:0>	
IPC5	31:24	—	—	—	SPI1IP<2:0>			SPI1IS<1:0>	
	23:16	—	—	—	OC5IP<2:0>			OC5IS<1:0>	
	15:8	—	—	—	IC5IP<2:0>			IC5IS<1:0>	
	7:0	—	—	—	T5IP<2:0>			T5IS<1:0>	

Register 15-1: ICxCON: Input Capture x Control Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	r-x	r-x	r-x	R/W-0	R/W-0
ON	FRZ	SIDL	—	—	—	ICFEDGE	ICC32
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0
ICTMR	ICI<1:0>		ICOV	ICBNE	ICM<2:0>		
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15 **ON:** ON bit

1 = Module enabled
0 = Disable and reset module, disable clocks, disable interrupt generation, and allow SFR modifications

Note: When using 1:1 PBCLK divisor, the user’s software should not read/write the peripheral’s SFRs in the SYSCLK cycle immediately following the instruction that clears the module’s ON bit.

bit 14 **FRZ:** Freeze in DEBUG Mode Control bit

1 = Freeze module operation when in DEBUG mode
0 = Do not freeze module operation when in DEBUG mode

Note: FRZ is writable in Debug Exception mode only, it is forced to ‘0’ in normal mode.

bit 13 **SIDL:** Stop in IDLE Control bit

1 = Halt in CPU IDLE mode
0 = Continue to operate in CPU IDLE mode

bit 12-10 Unimplemented: Read as ‘0’

bit 9 **ICFEDGE:** First Capture Edge Select bit **(**only used in mode 6, ICxM = 110)

1 = Capture rising edge first
0 = Capture falling edge first

bit 8 **ICC32:** 32-Bit Capture Select bit

1 = 32-Bit timer resource capture
0 = 16-Bit timer resource capture

bit 7 **ICTMR:** Timer Select bit **(**Does not affect timer selection when ICxC32 (ICxCON<8>) is ‘1’)

0 = Timer3 is the counter source for capture
1 = Timer2 is the counter source for capture

bit 6-5 **ICI<1:0>:** Interrupt Control bits

11 = Interrupt on every fourth capture event
10 = Interrupt on every third capture event
01 = Interrupt on every second capture event
00 = Interrupt on every capture event

Register 15-1: ICxCON: Input Capture x Control Register (Continued)

bit 4 **ICOV:** Input Capture Overflow Status Flag bit (read-only)
1 = Input capture overflow occurred
0 = No input capture overflow occurred

bit 3 **ICBNE:** Input Capture Buffer Not Empty Status bit (Read Only)
1 = Input capture buffer is not empty; at least one more capture value can be read
0 = Input capture buffer is empty

bit 2-0 **ICM<2:0>:** Input Capture Mode Select bits
111 = Interrupt Only mode
110 = Simple Capture Event mode – every edge, specified edge first and every edge thereafter
101 = Prescaled Capture Event mode – every 16th rising edge
100 = Prescaled Capture Event mode – every 4th rising edge
011 = Simple Capture Event mode – every rising edge
010 = Simple Capture Event mode – every falling edge
001 = Edge Detect mode – every edge (rising and falling)
000 = Capture Disable mode

Register 15-2: ICxBUF: Input Capture x Buffer Register

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
ICxBUF<31:24>							
bit 31							bit 24

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
ICxBUF<23:16>							
bit 23							bit 16

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
ICxBUF<15:8>							
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
ICxBUF<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **ICxBUF<31:0>:** Buffer Register bits
Value of the current captured input timer count

Register 15-3: IFS0: Interrupt Flag Status Register 0(1)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
I2C1SIF	I2C1BIF	U1TXIF	U1RXIF	U1EIF	SPI1RXIF	SPI1TXIF	SPI1EIF
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CNIF	OC5IF	IC5IF	T5IF	INT4IF	OC4IF	IC4IF	T4IF
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INT3IF	OC3IF	IC3IF	T3IF	INT2IF	OC2IF	IC2IF	T2IF
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INT1IF	OC1IF	IC1IF	T1IF	INT0IF	CS1IF	CS0IF	CTIF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 21 **IC5IF:** Input Capture 5 Interrupt Request Flag bit
1 = Interrupt request has occurred
0 = No interrupt request has a occurred

bit 17 **IC4IF:** Input Capture 4 Interrupt Request Flag bit
1 = Interrupt request has occurred
0 = No interrupt request has a occurred

bit 13 **IC3IF:** Input Capture 3 Interrupt Request Flag bit
1 = Interrupt request has occurred
0 = No interrupt request has a occurred

bit 9 **IC2IF:** Input Capture 2 Interrupt Request Flag bit
1 = Interrupt request has occurred
0 = No interrupt request has a occurred

bit 5 **IC1IF:** Input Capture 1 Interrupt Request Flag bit
1 = Interrupt request has occurred
0 = No interrupt request has a occurred

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the Input Capture module.

Register 15-4: IEC0: Interrupt Enable Control Register 0(1)

I2C1SIE	I2C1BIE	U1TXIE	U1RXIE	U1EIE	SPI1RXIE	SPI1TXIE	SPI1EIE
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CNIE	OC5IE	IC5IE	T5IE	INT4IE	OC4IE	IC4IE	T4IE
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INT3IE	OC3IE	IC3IE	T3IE	INT2IE	OC2IE	IC2IE	T2IE
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INT1IE	OC1IE	IC1IE	T1IE	INT0IE	CS1IE	CS0IE	CTIE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 21 **IC5IE:** Input Capture 5 Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 17 **IC4IE:** Input Capture 4 Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 13 **IC3IE:** Input Capture 3 Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 9 **IC2IE:** Input Capture 2 Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 5 **IC1IE:** Input Capture 1 Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the Input Capture module.

Register 15-5: IPC1: Interrupt Priority Control Register 1(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	INT1IP<2:0>			INT1IS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	OC1IP<2:0>			OC1IS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	IC1IP<2:0>			IC1IS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	T1IP<2:0>			T1IS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 12-10 **IC1IP<2:0>:** Input Capture 1 Interrupt Priority bits
- 111 = Interrupt priority is 7
- 110 = Interrupt priority is 6
- 101 = Interrupt priority is 5
- 100 = Interrupt priority is 4
- 011 = Interrupt priority is 3
- 010 = Interrupt priority is 2
- 001 = Interrupt priority is 1
- 000 = Interrupt is disabled

bit 9-8 **IC1IS<1:0>:** Input Capture 1 Interrupt Subpriority bits
- 11 = Interrupt subpriority is 3
- 10 = Interrupt subpriority is 2
- 01 = Interrupt subpriority is 1
- 00 = Interrupt subpriority is 0

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the Input Capture module.

Register 15-6: IPC2: Interrupt Priority Control Register 2(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	INT2IP<2:0>			INT2IS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	OC2IP<2:0>			OC2IS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	IC2IP<2:0>			IC2IS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	T2IP<2:0>			T2IS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 12-10 **IC2IP<2:0>:** Input Capture 2 Interrupt Priority bits

111 = Interrupt priority is 7
110 = Interrupt priority is 6
101 = Interrupt priority is 5
100 = Interrupt priority is 4
011 = Interrupt priority is 3
010 = Interrupt priority is 2
001 = Interrupt priority is 1
000 = Interrupt is disabled

bit 9-8 **IC2IS<1:0>:** Input Capture 2 Interrupt Subpriority bits

11 = Interrupt subpriority is 3
10 = Interrupt subpriority is 2
01 = Interrupt subpriority is 1
00 = Interrupt subpriority is 0

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the Input Capture module.

Register 15-7: IPC3: Interrupt Priority Control Register 3(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	INT3IP<2:0>			INT3IS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	OC3IP<2:0>			OC3IS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	IC3IP<2:0>			IC3IS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	T3IP<2:0>			T3IS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 12-10 **IC3IP<2:0>:** Input Capture 3 Interrupt Priority bits

111 = Interrupt priority is 7
110 = Interrupt priority is 6
101 = Interrupt priority is 5
100 = Interrupt priority is 4
011 = Interrupt priority is 3
010 = Interrupt priority is 2
001 = Interrupt priority is 1
000 = Interrupt is disabled

bit 9-8 **IC3IS<1:0>:** Input Capture 3 Interrupt Subpriority bits

11 = Interrupt subpriority is 3
10 = Interrupt subpriority is 2
01 = Interrupt subpriority is 1
00 = Interrupt subpriority is 0

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the Input Capture module.

Register 15-8: IPC4: Interrupt Priority Control Register 4(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	INT4IP<2:0>			INT4IS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	OC4IP<2:0>			OC4IS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	IC4IP<2:0>			IC4IS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	T4IP<2:0>			T4IS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 12-10 **IC4IP<2:0>:** Input Capture 4 Interrupt Priority bits

111 = Interrupt priority is 7
110 = Interrupt priority is 6
101 = Interrupt priority is 5
100 = Interrupt priority is 4
011 = Interrupt priority is 3
010 = Interrupt priority is 2
001 = Interrupt priority is 1
000 = Interrupt is disabled

bit 9-8 **IC4IS<1:0>:** Input Capture 4 Interrupt Subpriority bits

11 = Interrupt subpriority is 3
10 = Interrupt subpriority is 2
01 = Interrupt subpriority is 1
00 = Interrupt subpriority is 0

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the Input Capture module.

Register 15-9: IPC5: Interrupt Priority Control Register 5(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	SPI1IP<2:0>			SPI1IS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	OC5IP<2:0>			OC5IS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	IC5IP<2:0>			IC5IS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	T5IP<2:0>			T5IS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 12-10 **IC5IP<2:0>:** Input Capture 5 Interrupt Priority bits

111 = Interrupt priority is 7
110 = Interrupt priority is 6
101 = Interrupt priority is 5
100 = Interrupt priority is 4
011 = Interrupt priority is 3
010 = Interrupt priority is 2
001 = Interrupt priority is 1
000 = Interrupt is disabled

bit 9-8 **IC5IS<1:0>:** Input Capture 5 Interrupt Subpriority bits

11 = Interrupt subpriority is 3
10 = Interrupt subpriority is 2
01 = Interrupt subpriority is 1
00 = Interrupt subpriority is 0

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the Input Capture module.

15.3 TIMER SELECTION

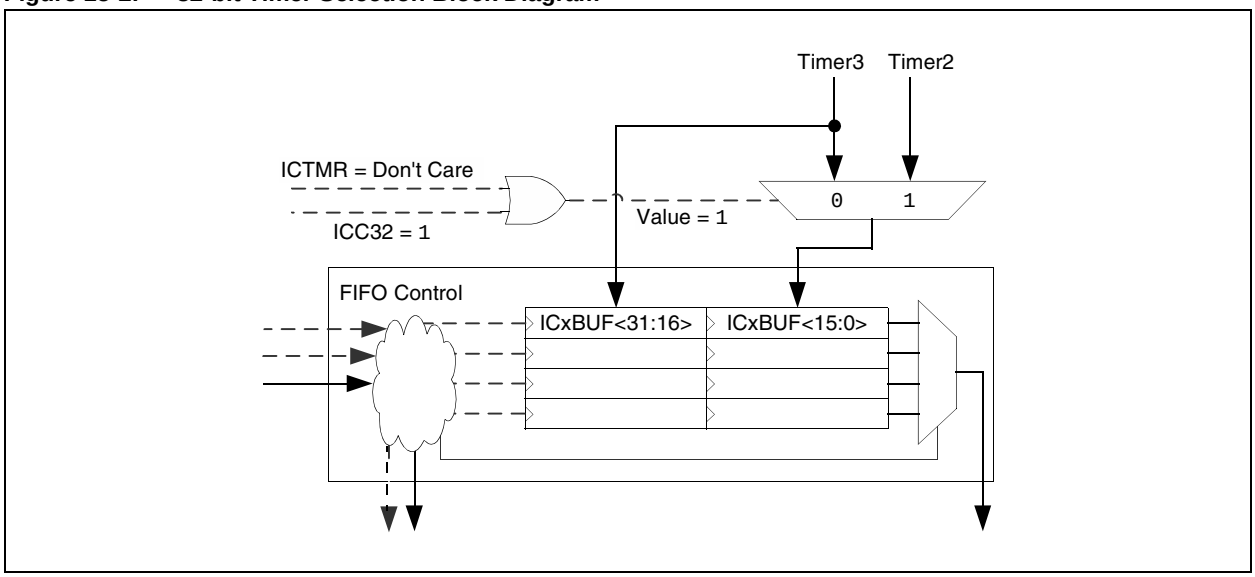
Each PIC32MX device may have one or more input capture channels. Each channel can select between one of two 16-bit timers for the time base or two 16-bit timers together (to form a 32-bit timer). Refer to the device data sheet for the specific timers that can be selected.

For 16-bit Capture mode, setting ICTMR (ICxCON<7>) to '0' selects the Timer3 for capture. Setting ICTMR (ICxCON<7>) to '1' selects the Timer2 for capture.

An input capture channel configured to support 32-bit capture may use a 32-bit timer resource for capture. By setting ICC32 (ICxCON<8>) to '1', a 32-bit timer resource is captured. The 32-bit timer resource is routed into the module using the existing 16-bit timer inputs. The Timer 2 provides the lower 16-bits and the Timer 3 provides the upper 16-bits, as shown in Figure 15-2.

The timers clock can be set up using the internal peripheral clock source or a synchronized external clock source applied at the TxCK pin.

Figure 15-2: 32-bit Timer Selection Block Diagram



15.4 INPUT CAPTURE ENABLE

After configuration, an Input Capture module is enabled by setting the ON bit (ICxCON<15>). When this bit is cleared, the module is reset. Resetting the module has the following effects:

- clears the Overflow Condition Flag
- resets FIFO to the empty state
- resets the event count (for interrupt generation)
- resets the prescaler count

Register reads and writes are allowed regardless of the ON (ICxCON<15>) bit state.

15.5 INPUT CAPTURE EVENT MODES

The input capture module captures the value of the selected time base register when an event occurs at the ICx pin. An input capture channel can be configured in the following modes:

1. Simple Capture Event modes:
   Capture timer value on every falling edge of input at ICx pin
   Capture timer value on every rising edge of input at ICx pin
   Capture timer value on every rising and falling edge of input at ICx pin, starting with a specified edge
2. Prescaled Capture Event modes:
   Capture timer value on every 4th rising edge of input at ICx pin
   Capture timer value on every 16th rising edge of input at ICx pin
3. Edge Detect mode (See **15.5.3 "Edge Detect (Hall Sensor) Mode"**)
4. Interrupt Only mode (See **15.5.4 "Interrupt Only Mode"**)

These Input Capture modes are configured by setting the appropriate Input Capture mode bits ICxM (ICxCON<2:0>).

When the Input Capture Channel is disabled (ICM = 000), the Input Capture logic ignores incoming capture edges and does not generate further capture events or interrupts. The FIFO continues to be operational for reading. Returning the channel to any of the other modes resumes operation. A state change on the capture input while capture is disabled does not cause a capture event on exiting the Capture Disable mode.

Note:	The prescaler logic continues to run when the Input Capture module is in Capture Disable mode.

15.5.1 Simple Capture Events

The capture module can capture a timer count value based on the selected edge (rising, falling or both, defined by mode) of the input applied to the ICx pin. These modes are specified by setting the ICM (ICxCON<2:0>) bits to '010', '011', or '110'. Setting ICM = 011 configures the module to capture the timer value on any rising edge of the capture input. ICM = 010 configures the module to capture the timer on any falling edge of the capture input. Setting ICM = 110 configures the channel to capture the timer on every transition of the capture input, beginning with the edge specified by ICFEDGE (ICxCON<9>). In Simple Capture Event mode, the prescaler is not used. See Figure 15-3, Figure 15-4 and Figure 15-5 for simplified timing diagrams of a simple capture event.

The input capture logic detects and synchronizes the rising or falling edge of the capture pin signal on the peripheral clock. When the rising/falling edge has occurred, the capture module logic will write the current time base value to the capture buffer and signal the interrupt generation logic.

Note:	Since the capture input must be synchronized to the peripheral clock, the module captures the timer count value that is valid 2-3 peripheral clock cycles (TPB) after the capture event.

An input capture interrupt event is generated after one, two, three or four timer count captures, as configured by ICI (ICxCON<6:5>). See **15.7 "Input Capture Interrupts"** for further details.

Since the capture pin is sampled by the peripheral clock, the capture pulse high and low widths must be greater than the peripheral clock period.

Figure 15-3 depicts two capture events when the Input Capture module is in Simple Capture mode configured to capture every rising edge, ICM = 011 (ICxCON<2:0>), with interrupts generated for every event, ICI = 00 (ICxCON<6:5>).

The first capture event occurs when the timer value is 'n'. Due to synchronization delay, timer value 'n + 2' is stored in the capture buffer. The second capture event occurs when the timer value is 'm'. Note that 'm + 3' is stored in the capture buffer due to propagation delay as well as the synchronization delay. Interrupt events are generated on each capture event.

Figure 15-3: Simple Capture Event Timing Diagram, Capture Every Rising Edge

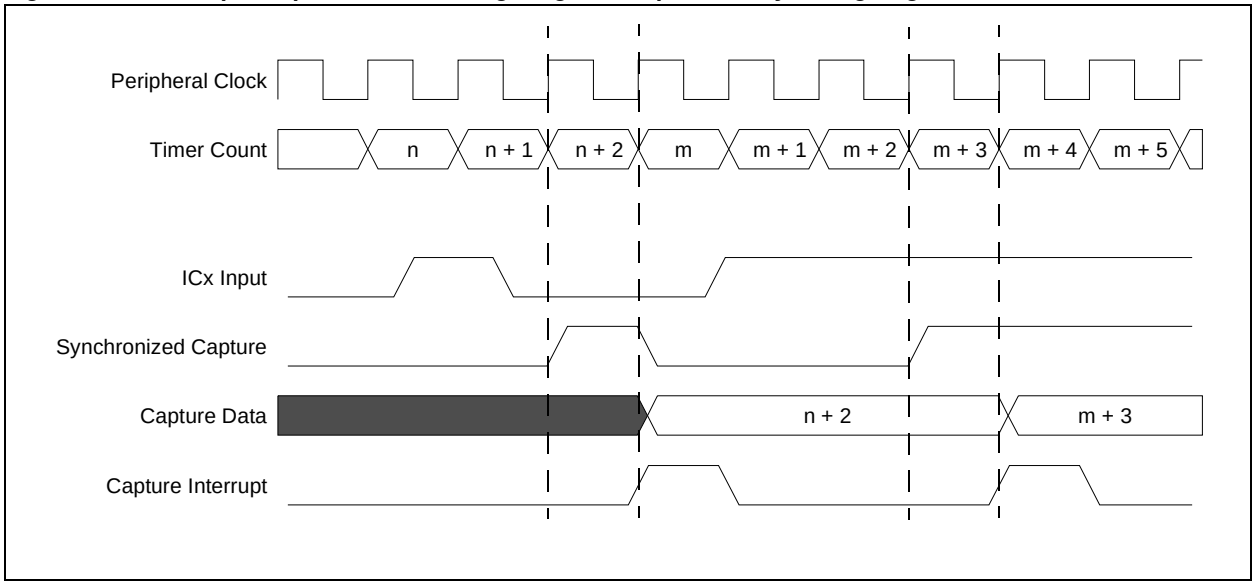


Figure 15-4 depicts a capture event when the Input Capture module is in Simple Capture mode configured to capture every falling edge, ICM = 010 (ICxCON<2:0>), with interrupts generated for every event, ICI = 00 (ICxCON<6:5>). In this example, the timer frequency is slower than the peripheral clock.

The capture event occurs when the timer value is 'n'. Value 'n' is stored in the capture buffer and an interrupt event is generated.

Figure 15-4: Simple Capture Event Timing Diagram, Capture Every Falling Edge

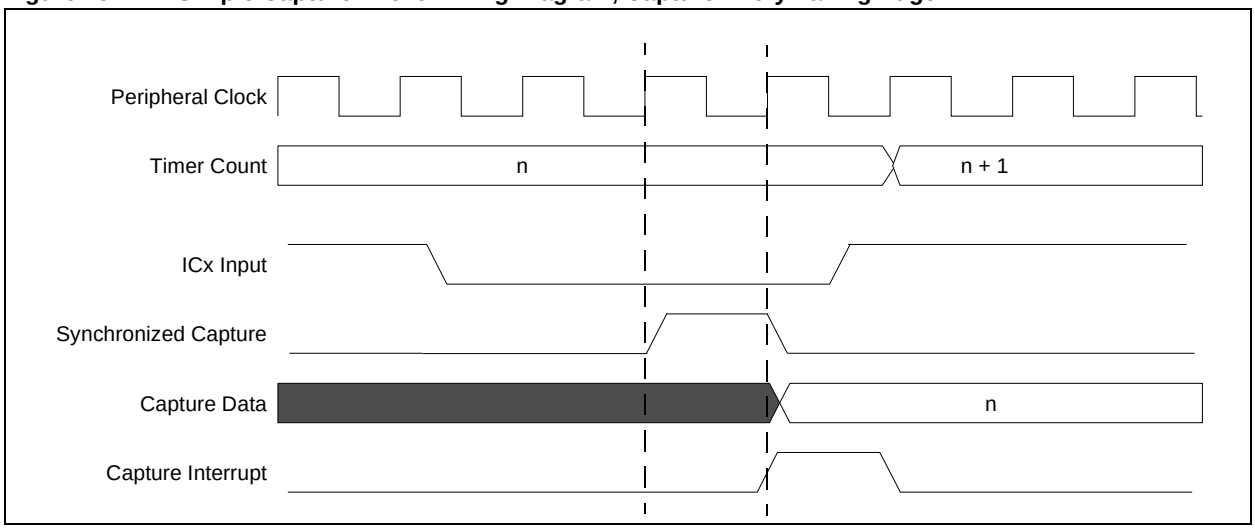
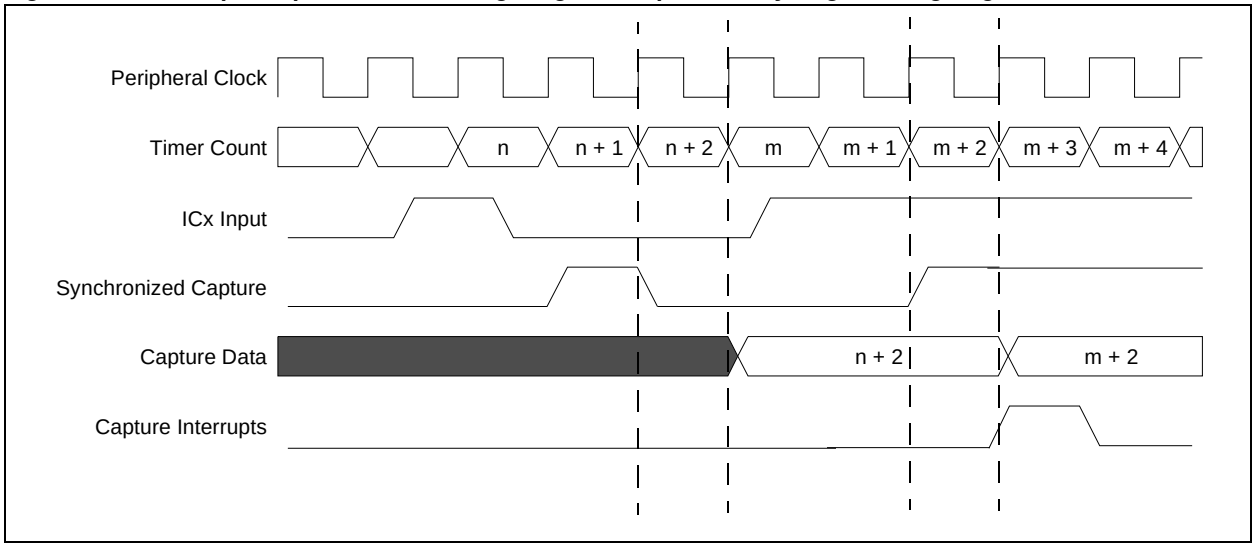


Figure 15-5 depicts a capture event when the Input Capture module is in Simple Capture mode configured to capture every edge, ICM = 011 (ICxCON<2:0>); starting with a falling edge, ICFEDGE = 0 (ICxCON<9>), with interrupts generated for every second event, ICI = 01 (ICxCON<6:5>).

The first falling edge occurs when the timer value is 'n'. Value 'n + 2' is stored in the capture buffer. A subsequent rising edge occurs when the timer value is 'm'. Value 'm + 2' is stored in the capture buffer and an interrupt event is generated.

Figure 15-5: Simple Capture Event Timing Diagram, Capture Every Edge, Falling Edge First



15.5.2 Prescaled Capture Event Mode

In Prescaled Capture Event mode, the Input Capture module triggers a capture event on either every fourth or every sixteenth rising edge. These modes are selected by setting the ICM (ICxCON<2:0>) bits to '100' or '101', respectively.

The capture prescaler counter is incremented on every rising edge on the capture input. When the prescaler counter equals four or sixteen (depending on the mode selected), the counter outputs a "valid" capture event signal. The valid capture event signal is then synchronized to the peripheral clock. The synchronized capture event signal triggers a timer count capture.

Note:	Since the capture input must be synchronized to the peripheral clock, the module captures the timer count value that is valid 2-3 peripheral clock cycles (TPB) after the capture event.

An input capture interrupt is generated after one, two, three or four timer count captures, as configured by ICI. See **15.7 "Input Capture Interrupts"** for further details.

Note:	It is recommended that the user disable the capture module (i.e., clear the ON bit, ICxCON<15>), before switching to Prescaler Capture Event mode. Simply switching to Prescaler Capture Event mode from another active mode does not reset the prescaler and may cause an inadvertent capture event.

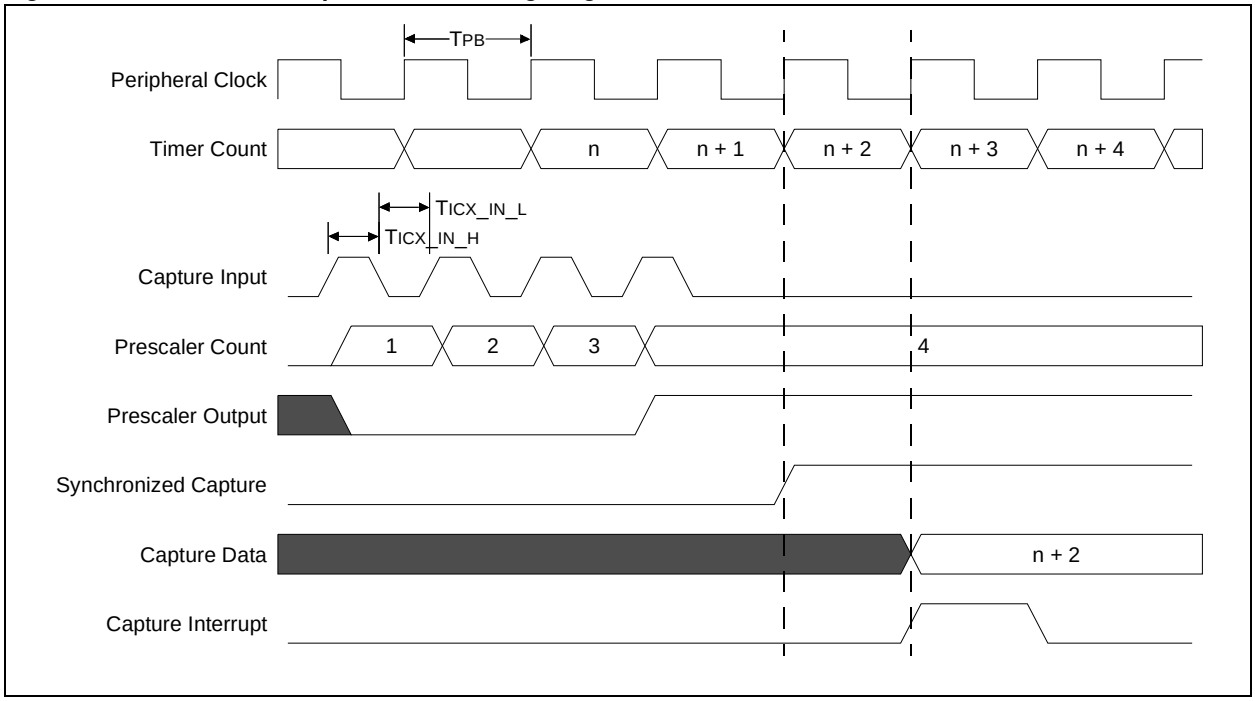
The prescaler counter is cleared when the following events occur:

- The Input Capture module is turned off, i.e., ON (ICxCON<15>) = 0.
- The Input Capture module is reset.

Since the capture pin triggers an internal flip-flop, the input capture pulse high and low widths must be greater than TccL and TccH.

Figure 15-6 depicts a capture event when the Input Capture module is in Prescaler Capture Event mode. The prescaler is configured to capture a timer value for every fourth rising edge on the capture input, ICM = 100 (ICxCON<2:0>), with interrupts generated for every capture event, ICI = 00 (ICxCON<6:5>). The fourth rising edge on the capture input occurs at time 'n'. The prescaler output is synchronized. Due to synchronization delay, timer value 'n + 2' is stored in the capture buffer. An interrupt signal is generated due to the capture event.

Figure 15-6: Prescaler Capture Event Timing Diagram



15.5.3 Edge Detect (Hall Sensor) Mode

In Edge Detect mode, the Input Capture module captures a timer count value on every edge of the capture input. Edge Detection mode is selected by setting the ICM bit to '001'. In this mode, the capture prescaler is not used and the capture overflow bit, ICOV (ICxCON<4>), is not updated. In this mode, the Interrupt Control bits, ICI (ICxCON<6:5>), are ignored and an interrupt event is generated for every timer count capture. See Figure 15-7 for a simplified timing diagram.

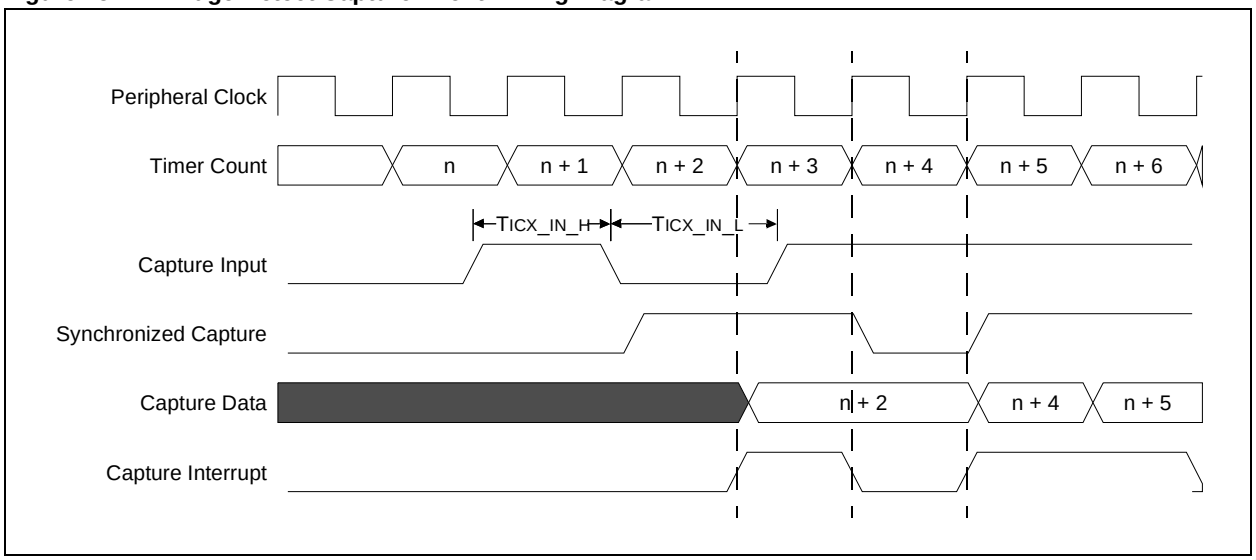
As with the Simple Capture Event mode, the Input Capture logic detects and synchronizes the rising and falling edge of the capture input signal on the peripheral clock. When a rising or falling edge occurs, the capture module writes the time base value to the capture buffer.

Note:	Since the capture input must be synchronized to the peripheral clock, the module captures the timer count value that is valid 2-3 peripheral clock cycles (TPB) after the capture event.

Since the capture pin is sampled by the peripheral clock, the capture pulse high and low widths must be greater than the peripheral clock period.

Figure 15-7 depicts three capture events when the Input Capture module is in Edge Detect mode, ICM = 001 (ICxCON<2:0). Transitions on the capture input occur at times 'n', 'n + 1' and 'n + 3'. Due to synchronization and propagation delay, timer values 'n + 2', 'n + 4' and 'n + 5' are stored in the capture buffer. Interrupt signals are generated due to each capture input transition.

Figure 15-7: Edge Detect Capture Event Timing Diagram



15.5.4 Interrupt Only Mode

When the Input Capture module is set for Interrupt Only mode (ICM = 111) and the device is in SLEEP or IDLE mode, the capture input functions as an interrupt pin. Any rising edge on the capture input triggers an interrupt. No timer values are captured and the FIFO buffer is not updated. When the device leaves SLEEP or IDLE mode, the interrupt signal is deasserted.

In this mode, since no timer values are captured, the Timer Select bit, ICTMR (ICxCON<7>), is ignored and there is no need to configure the timer source. A wake-up interrupt is generated on the first rising edge. Therefore, the Interrupt Control bits, ICI (ICxCON<6:5>), are also ignored. The prescaler is not used in this mode.

Since the capture pin triggers an internal flip-flop, the capture pulse high and low widths must be greater than TccL and TccH.

15.6 CAPTURE BUFFER OPERATION

Each Input Capture module has an associated four-level deep First-In-First-Out (FIFO) buffer. The buffer is accessible to the user via the buffer register (ICxBUF). ICxBUF is written by the Input Capture logic and can only be read by the user. Writes to ICxBUF are ignored.

There are two status flags which provide status on the FIFO buffer:

- ICBNE (ICxCON<3>) – Input Capture Buffer Not Empty
- ICOV (ICxCON<4>) – Input Capture Overrun

When the Input Capture module is disabled, i.e., ON ICxCON<15>) = 0 or Reset, the status flags are cleared and the buffer is cleared to the empty state.

The ICBNE flag is set on the first input capture event and remains set until all capture events have been read from the FIFO. For example, if three capture events have occurred, then three reads of the capture FIFO buffer are required before the ICBNE flag is cleared. If four capture events have occurred, then four reads are required to clear the ICBNE flag.

Each read of the FIFO buffer adjusts the read pointer, allowing the remaining entries to move to the next available top location of the FIFO. In 32-bit Capture mode, make sure you read the upper 16 bit last if you are reading 16 bits at a time. The FIFO read pointer is advanced when you read the MSB.

If the FIFO is full with four capture events and a fifth capture event occurs prior to a read of the FIFO, an overrun condition occurs and the ICOV (ICxCON<4>) bit is set to a logic '1'. In addition, the fifth capture event is not recorded and subsequent capture events do not alter the current FIFO contents until the overrun condition is cleared.

The overflow condition is cleared in any of the following ways:

- Module is disabled, i.e., ON = 0 (ICxCON<15>)
- Capture buffer is read until ICBNE = 0 (ICxCON<3>)
- Device is reset

If the Input Capture module is disabled and at some time reenabled, the FIFO buffer contents are not defined and a read may yield indeterminate results.

If a FIFO read is performed when no capture event has been received, the read yields indeterminate results.

15.7 INPUT CAPTURE INTERRUPTS

The Input Capture module has the ability to generate an interrupt event signal based upon the selected number of capture events. A capture event is defined by the writing of a timer value into the FIFO.

The number of capture events required to trigger an interrupt event is set by control bits ICI (ICxCON<6:5>).

If ICBNE = 0 (ICxCON<3>), then the interrupt count is cleared. This allows the user to synchronize the interrupt count to the FIFO status.

For example, assume that ICI = 01, specifying an interrupt event every 2nd capture event.

1. Turn on module.
2. Interrupt count = 0.
3. Capture event. FIFO contains 1 entry.
4. Interrupt count = 1.
5. Read FIFO. FIFO is empty => interrupt count = 0.
6. Capture event. FIFO contains 1 entry.
7. Interrupt count = 1.
8. Capture event. FIFO contains 2 entries.
9. Interrupt count = 2. Issue interrupt.
10. Interrupt count = 0.
11. Capture event. FIFO contains 3 entries.
12. Interrupt count = 1.
13. Read FIFO 3 times.
14. FIFO is empty => interrupt count = 0.
15. Capture event. FIFO contains 1 entry.
16. Interrupt count = 1.
17. Read FIFO. FIFO is empty => interrupt count = 0.

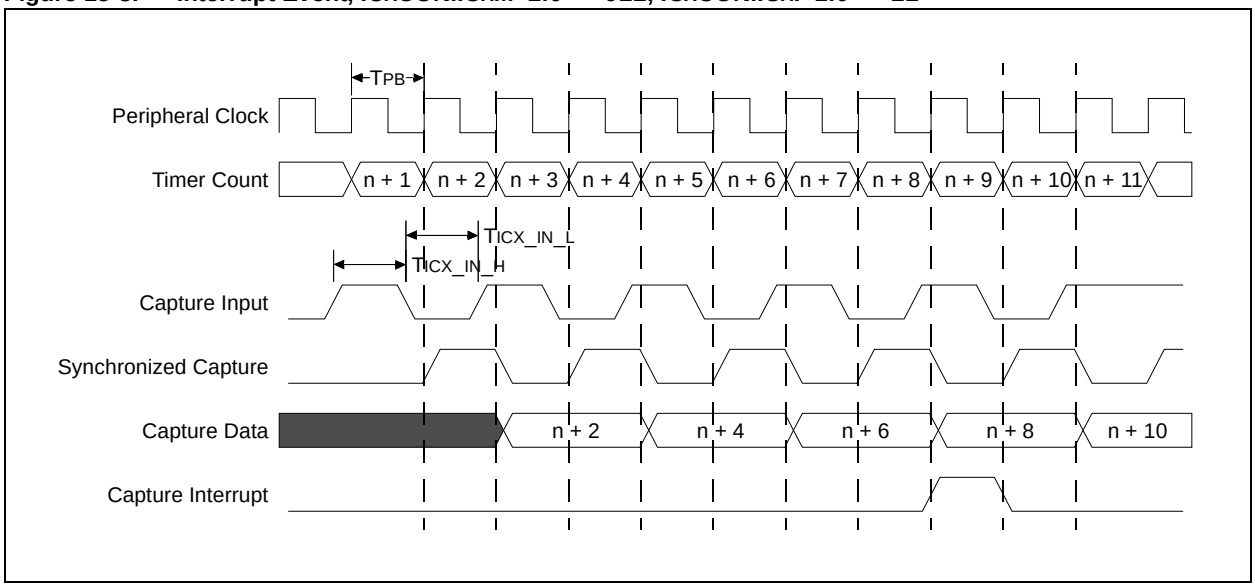
The first capture event is defined as the capture event occurring after a mode change from the OFF mode or after ICBNE = 0.

At overrun, the capture events cease, and therefore, the interrupt events stop – unless ICI = 00 or ICM = 001.

Applications often dictate using the Input Capture pins as auxiliary external interrupt sources. When ICI = 00 or ICM = 001, interrupt events occur regardless of FIFO overrun. There is no need to perform a dummy read on the capture buffer to clear the event and prevent an overflow in order to ensure that future interrupt events are not inhibited. The ICOV (ICxCON<4>) flag is still set for the overflow condition.

Figure 15-8 depicts five capture events when the Input Capture module is configured to capture timer values on every rising edge (ICM = 011) and generate an interrupt for every four captures (ICI = 11). Note that the fourth capture causes the capture of value 'n + 8' and triggers an interrupt event.

Figure 15-8: Interrupt Event, ICxCON.ICxM<2:0> = 011, ICxCON.ICxI<1:0> = 11



15.7.1 Interrupt Control Bits

Each input capture channel has interrupt flag status bits (ICxIF), interrupt enable bits (ICxIE), interrupt priority control bits (ICxIP) and secondary interrupt priority control bits (ICxIS). Refer to **8.2 "Control Registers"** in **Section 1. "Interrupts"** for further information on peripheral interrupts.

15.8 OPERATION IN POWER-SAVING MODES

Note: In this manual, a distinction is made between a power mode as it is used in a specific module, and a power mode as it is used by the device, e.g., Sleep mode of the Comparator and SLEEP mode of the CPU. To indicate which type of power mode is intended, uppercase and lowercase letters (Sleep, Idle, Debug) signify a module power mode, and all uppercase letters (SLEEP, IDLE, DEBUG) signify a device power mode.

15.8.1 Input Capture Operation in SLEEP Mode

When the device enters SLEEP mode, the peripheral clock is disabled. In SLEEP mode, the input capture module can only function as an external interrupt source. This mode is enabled by setting control bits ICM = 111 (ICxCON<2:0>). In this mode, a rising edge on the capture pin will generate a device wake-up from SLEEP condition. If the respective module interrupt bit is enabled and the module's priority is of the required priority level, an interrupt will be generated. See **15.5.4 "Interrupt Only Mode"** for more detail.

If the capture module has been configured for a mode other than ICM = 111 and the device does enter the SLEEP mode, no external pin stimulus, rising or falling, will generate a wake-up from SLEEP event.

15.8.2 Input Capture Operation in IDLE Mode

When the device enters IDLE mode, the peripheral clock sources remain functional and the CPU stops executing code. The SLEEP-In-IDLE control bit, SIDL (ICxCON<13>), determines whether the module will stop in IDLE mode or continue to operate.

If SIDL is '0', the module continues normal operation in IDLE mode. Interrupt only mode (ICM = 111) may generate an interrupt when in IDLE if SIDL is '0'. See **15.5.4 "Interrupt Only Mode"** for further details.

If SIDL is '1', the module stops when the device is in IDLE mode. The module performs the same procedures when stopped in IDLE mode as for SLEEP mode. See **15.5.4 "Interrupt Only Mode"** for further details.

15.8.3 Device Wake-up on SLEEP/IDLE

An input capture event can generate a device wake-up or interrupt, if enabled, when the device is in IDLE or SLEEP mode. See **15.5.4 "Interrupt Only Mode"** for further details.

15.9 INPUT CAPTURE OPERATION IN DEBUG MODE

The FRZ bit (ICxCON<14>) determines whether the Input Capture module will run or stop while the CPU is executing Debug Exception code (i.e., the application is halted) in DEBUG mode. When FRZ is '0', the Timer module continues to run even when the application is halted in DEBUG mode. When FRZ is '1' and the application is halted in DEBUG mode, the module will freeze its operations and make no changes to the state of the Input Capture module. The module will resume its operation after the CPU resumes execution.

Note: The FRZ bit is readable and writable only when the CPU is executing in Debug Exception mode. In all other modes, the FRZ bit reads as '0'. If FRZ bit is changed during DEBUG mode, the new value does not take effect until the current Debug Exception mode is exited and re-entered. During the Debug Exception mode, the FRZ bit reads the state of the peripheral when entering DEBUG mode.

15.9.1 Capture Operation During FREEZE (FRZ = 1)

When frozen, the capture input does not cause changes to the module. The edge detection logic runs during Freeze so that any state changes that occur during Freeze will not be inadvertently detected after leaving Freeze.

Clocks to all of the logic within the Input Capture module, with the exception of the SFR logic and the FIFO read logic, are conditioned on Freeze.

Note:	The prescaler logic is not frozen during DEBUG mode.

When frozen, the emulator is allowed to read the Input Capture FIFO; however, the FIFO status flags as viewed by the user do not change.

15.9.2 Operation of the Capture Buffer in Debug Mode

During DEBUG mode, reads from the capture buffer become circular. Reading ICxBUF adjusts only the DEBUG FIFO pointers; no status flags are affected by reads. This allows the DEBUG software to have visibility of the full contents of the FIFO. To enable this, the hardware contains two sets of ICxBUF FIFO pointers: an operating mode set and a DEBUG mode set.

15.10 I/O PIN CONTROL

When the capture module is enabled, the user must ensure that the I/O pin direction is configured for an input by setting the associated TRIS bit. The pin direction is not set when the capture module is enabled. Furthermore, all other peripherals multiplexed with the input pin must be disabled.

15.11 DESIGN TIPS

Question 1: ***Can the Input Capture module be used to wake the device from SLEEP mode?***

Answer: Yes. When the Input Capture module is configured to ICM = 111 (ICxCON<2:0>) and the respective channel interrupt enable bit is asserted (ICIE = 1; see Interrupt registers IE0-IE2), a rising edge on the capture pin will wake-up the device from SLEEP. (See **15.5.4 “Interrupt Only Mode”** for further details.)

15.12 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the PIC32MX device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the Input Capture module include the following:

Title	Application Note #
Using the CCP Module(s)	AN594
Implementing Ultrasonic Ranging	AN597

Note: Please visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the PIC32MX family of devices.

15.13 REVISION HISTORY

Revision A (October 2007)

This is the initial released version of this document.

Revision B (October 2007)

Updated document to remove Confidential status.

Revision C (April 2008)

Revised status to Preliminary; Revised U-0 to r-x; Removed ‘x’ in bit names.

Revision D (June 2008)

Revised note for Registers 15-1, 15-3, 15-4, 15-5, 15-6, 15-7, 15-8, 15-9; Change Reserved bits from “Maintain as” to “Write”; Added Note to ON bit (ICxCON Register).

NOTES:

Section 16. Output Compare

HIGHLIGHTS

This section of the manual contains the following topics:

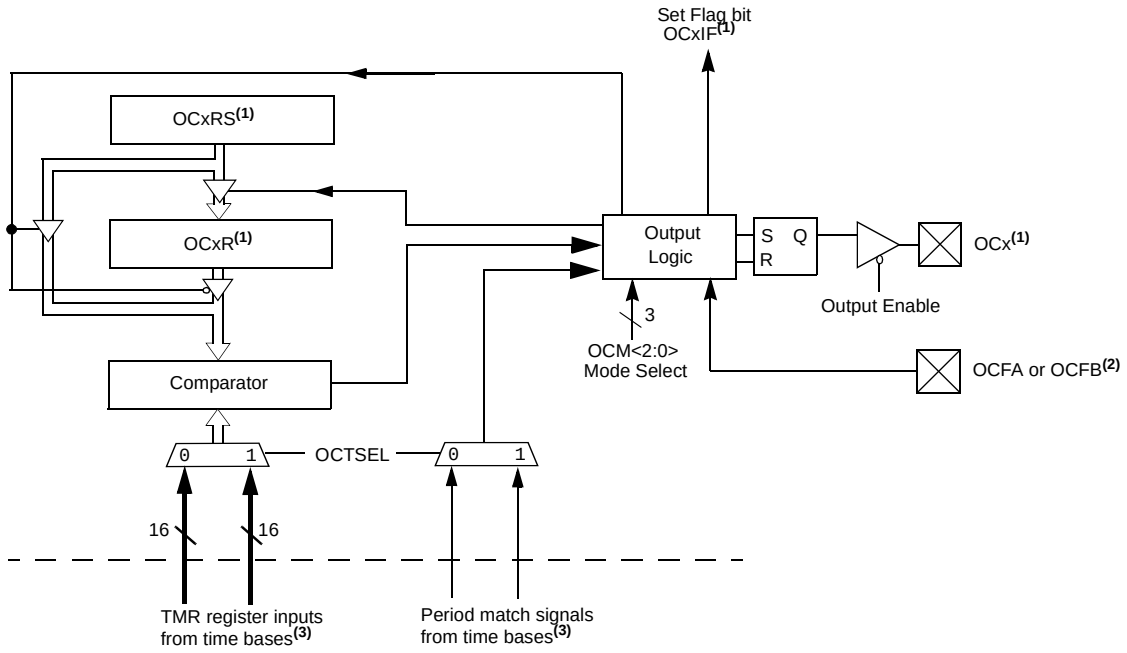
16.1 INTRODUCTION

The Output Compare module is primarily used to generate a single pulse or a train of pulses in response to selected time base events.

The following are some of the key features of the Output Compare module:

- Multiple output compare modules in a device
- Single and Dual Compare modes
- Single and continuous output pulse generation
- Pulse-Width Modulation (PWM) mode
- Programmable interrupt generation on compare event
- Hardware-based PWM Fault detection and automatic output disable
- Programmable selection of 16 or 32-bit time bases
- Can operate from either of two available 16-bit time bases or a single 32-bit time base

Figure 16-1: Output Compare Module Block Diagram



Note 1: Where 'x' is shown, reference is made to the registers associated with the respective output compare channels 1 through 5.

2: OCFA pin controls OC1-OC4 channels. OCFB pin controls OC5 channels.

3: Each output compare channel can use one of two selectable 16-bit time bases or a single 32-bit time base. Refer to the device data sheet for the time bases associated with the module.

16.2 OUTPUT COMPARE REGISTERS

Note: Each PIC32MX device variant may have one or more Output Compare modules. An 'x' used in the names of pins, control/Status bits and registers denotes the particular module. Refer to the specific device data sheets for more details.

Each Output Compare module consists of the following Special Function Registers (SFRs):

- OCxCON: Control register for the OCMP module 'x'
  OCxCONCLR, OCxCONSET, OCxCONINV: Atomic Bit Manipulation Write-only Registers for OCxCON
- OCxR: Data register for the module 'x'
  OCxRCLR, OCxRSET, OCxRINV: Atomic Bit Manipulation Write-only Registers for OCxR
- OCxRS: Secondary data register for the module 'x'
  OCxRSCLR, OCxRSSET, OCxRSINV: Atomic Bit Manipulation Write-only Registers for OCxRS
- T2CON: Time Base Register
  T2CONCLR, T2CONSET, T2CONINV: Atomic Bit Manipulation Write-only Registers for T2CON
- T3CON: Time Base Register
  T3CONCLR, T3CONSET, T3CONINV: Atomic Bit Manipulation Write-only Registers for T3CON
- TMR2: Timer Register
  TTMR2CLR, TMR2SET, TMR2INV: Atomic Bit Manipulation Write-only Registers for TMR2
- TMR3: Timer Register
  TMR3CLR, TMR3SET, TMR3INV: Atomic Bit Manipulation Write-only Registers for TMR3
- PR2: Period 2 Register
  PR2CLR, PR2SET, PR2INV: Atomic Bit Manipulation Write-only Registers for PR2
- PR3: Period 3 Register
  PR3CLR, PR3SET, PR3INV: Atomic Bit Manipulation Write-only Registers for PR3

Each timer module also has the following associated bits for interrupt control:

- OC5IF, OC4IF, OC3IF, OC2IF, OC1IF: Interrupt Flag Status Bits – in IFS0 INT Register
- OC5IE, OC4IE, OC3IE, OC2IE, OC1IE: Interrupt Enable Control Bits – in IEC0 INT Register
- OC1IP<2:0>: Interrupt Priority Control Bits – in IPC1 INT Registers
- OC1IS<1:0>: Interrupt Subpriority Control Bits – in IPC1 INT Registers
- OC2IP<2:0>: Interrupt Priority Control Bits – in IPC2 INT Registers
- OC2IS<1:0>: Interrupt Subpriority Control Bits – in IPC2 INT Registers
- OC3IP<2:0>: Interrupt Priority Control Bits – in IPC3 INT Registers
- OC3IS<1:0>: Interrupt Subpriority Control Bits – in IPC3 INT Registers
- OC4IP<2:0>: Interrupt Priority Control Bits – in IPC4 INT Registers
- OC4IS<1:0>: Interrupt Subpriority Control Bits – in IPC4 INT Registers
- OC5IP<2:0>: Interrupt Priority Control Bits – in IPC5 INT Registers
- OC5IS<1:0>: Interrupt Subpriority Control Bits – in IPC5 INT Registers

The following table summarizes all Output-Compare-related registers. Corresponding registers appear after the summary, followed by a detailed description of each register.

Table 16-1: Output Compare SFR Summary

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
OCxCON	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	ON	FRZ	SIDL	—	—	—	—	—
	7:0	—	—	OC32	OCFLT	OCTSEL	OCM<2:0>		
OCxCONCLR	31:0	Write clears selected bits in OCxCON; read yields an undefined value							
OCxCONSET	31:0	Write sets selected bits in OCxCON; read yields an undefined value							
OCxCONINV	31:0	Write inverts selected bits in OCxCON; read yields an undefined value							
OCxR	31:24	OCxR<31:24>							
	23:16	OCxR<23:16>							
	15:8	OCxR<15:8>							
	7:0	OCxR<7:0>							
OCxRCLR	31:0	Write clears selected bits in OCxR; read yields an undefined value							
OCxRSET	31:0	Write sets selected bits in OCxR; read yields an undefined value							
OCxRINV	31:0	Write inverts selected bits in OCxR; read yields an undefined value							
OCxRS	31:24	OCxRS<31:24>							
	23:16	OCxRS<23:16>							
	15:8	OCxRS<15:8>							
	7:0	OCxRS<7:0>							
OCxRSCLR	31:0	Write clears selected bits in OCxRS; read yields an undefined value							
OCxRSSET	31:0	Write sets selected bits in OCxRS; read yields an undefined value							
OCxRSINV	31:0	Write inverts selected bits in OCxRS; read yields an undefined value							
IFS0	31:24	I2C1MIF	I2C1SIF	I2C1BIF	U1TXIF	U1RXIF	U1EIF	SPI1RXIF	SPI1TXIF
	23:16	SPI1EIF	OC5IF	IC5IF	T5IF	INT4IF	OC4IF	IC4IF	T4IF
	15:8	INT3IF	OC3IF	IC3IF	T3IF	INT2IF	OC2IF	IC3IF	T2IF
	7:0	INT1IF	OC1IF	IC1IF	T1IF	INT0IF	CS1IF	CS0IF	CTIF
IFS0CLR	31:0	Write clears the selected bits in IFS0, read yields undefined value							
IFS0SET	31:0	Write sets the selected bits in IFS0, read yields undefined value							
IFS0INV	31:0	Write inverts the selected bits in IFS0, read yields undefined value							
IEC0	31:24	I2C1MIE	I2C1SIE	I2C1BIE	U1TXIE	U1RXIE	U1EIE	SPI1RXIE	SPI1TXIE
	23:16	SPI1EIE	OC5IE	IC5IE	T5IE	INT4IE	OC4IE	IC4IE	T4IE
	15:8	INT3IE	OC3IE	IC3IE	T3IE	INT2IE	OC2IE	IC2IE	T2IE
	7:0	INT1IE	OC1IE	IC1IE	T1IE	INT0IE	CS1IE	CS0IE	CTIE
IEC0CLR	31:0	Write clears the selected bits in IEC0, read yields undefined value							
IEC0SET	31:0	Write sets the selected bits in IEC0, read yields undefined value							
IEC0INV	31:0	Write inverts the selected bits in IEC0, read yields undefined value							
IPC1	31:24	—	—	—	INT1IP<2:0>			INT1IS<1:0>	
	23:16	—	—	—	OC1IP<2:0>			OC1IS<1:0>	
	15:8	—	—	—	IC1IP<2:0>			IC1IS<1:0>	
	7:0	—	—	—	T1IP<2:0>			T1IS<1:0>	
IPC1CLR	31:0	Write clears the selected bits in IPC1, read yields undefined value							
IPC1SET	31:0	Write sets the selected bits in IPC1, read yields undefined value							
IPC1INV	31:0	Write inverts the selected bits in IPC1, read yields undefined value							

Table 16-1: Output Compare SFR Summary (Continued)

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
IPC2	31:24	—	—	—	INT2IP<2:0>			INT2IS<1:0>	
	23:16	—	—	—	OC2IP<2:0>			OC2IS<1:0>	
	15:8	—	—	—	IC2IP<2:0>			IC2IS<1:0>	
	7:0	—	—	—	T2IP<2:0>			T2IS<1:0>	
IPC2CLR	31:0	Write clears the selected bits in IPC2, read yields undefined value							
IPC2SET	31:0	Write sets the selected bits in IPC2, read yields undefined value							
IPC2INV	31:0	Write inverts the selected bits in IPC2, read yields undefined value							
IPC3	31:24	—	—	—	INT3IP<2:0>			INT3IS<1:0>	
	23:16	—	—	—	OC3IP<2:0>			OC3IS<1:0>	
	15:8	—	—	—	IC3IP<2:0>			IC3IS<1:0>	
	7:0	—	—	—	T3IP<2:0>			T3IS<1:0>	
IPC3CLR	31:0	Write clears the selected bits in IPC3, read yields undefined value							
IPC3SET	31:0	Write sets the selected bits in IPC3, read yields undefined value							
IPC3INV	31:0	Write inverts the selected bits in IPC3, read yields undefined value							
IPC4	31:24	—	—	—	INT4IP<2:0>			INT4IS<1:0>	
	23:16	—	—	—	OC4IP<2:0>			OC4IS<1:0>	
	15:8	—	—	—	IC4IP<2:0>			IC4IS<1:0>	
	7:0	—	—	—	T4IP<2:0>			T4IS<1:0>	
IPC4CLR	31:0	Write clears the selected bits in IPC4, read yields undefined value							
IPC4SET	31:0	Write sets the selected bits in IPC4, read yields undefined value							
IPC4INV	31:0	Write inverts the selected bits in IPC4, read yields undefined value							
IPC5	31:24	—	—	—	SPI1IP<2:0>			SPI1IS<1:0>	
	23:16	—	—	—	OC5IP<2:0>			OC5IS<1:0>	
	15:8	—	—	—	IC5IP<2:0>			IC5IS<1:0>	
	7:0	—	—	—	T5IP<2:0>			T5IS<1:0>	
IPC5CLR	31:0	Write clears the selected bits in IPC5, read yields undefined value							
IPC5SET	31:0	Write sets the selected bits in IPC5, read yields undefined value							
IPC5INV	31:0	Write inverts the selected bits in IPC5, read yields undefined value							
T2CON	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	ON	FRZ	SIDL	—	—	—	—	—
	7:0	TGATE	TCKPS<2:0>			T32	—	TCS	—
T2CONCLR	31:0	Write clears selected bits in T2CON; read yields undefined value							
T2CONSET	31:0	Write sets selected bits in T2CON; read yields undefined value							
T2CONINV	31:0	Write inverts selected bits in T2CON; read yields undefined value							
T3CON	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	ON	FRZ	SIDL	—	—	—	—	—
	7:0	TGATE	TCKPS<2:0>			—	—	TCS	—
T3CONCLR	31:0	Write clears selected bits in T3CON; read yields undefined value							
T3CONSET	31:0	Write sets selected bits in T3CON; read yields undefined value							
T3CONINV	31:0	Write inverts selected bits in T3CON; read yields undefined value							

Table 16-1: Output Compare SFR Summary (Continued)

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
TMR2	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	TMRx<15:8>							
	7:0	TMRx<7:0>							
TMR2CLR	31:0	Write clears selected bits in TMRx, read yields undefined value							
TMR2SET	31:0	Write sets selected bits in TMRx, read yields undefined value							
TMR2INV	31:0	Write inverts selected bits in TMRx, read yields undefined value							
TMR3	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	TMRx<15:8>							
	7:0	TMRx<7:0>							
TMR3CLR	31:0	Write clears selected bits in TMRx, read yields undefined value							
TMR3SET	31:0	Write sets selected bits in TMRx, read yields undefined value							
TMR3INV	31:0	Write inverts selected bits in TMRx, read yields undefined value							
PR2	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	PR2<15:8>							
	7:0	PR2<7:0>							
PR2CLR	31:0	Write clears selected bits in PR2; read yields undefined value							
PR2SET	31:0	Write sets selected bits in PR2; read yields undefined value							
PR2INV	31:0	Write inverts selected bits in PR2; read yields undefined value							
PR3	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	PR3<15:8>							
	7:0	PR3<7:0>							
PR3CLR	31:0	Write clears selected bits in PR3; read yields undefined value							
PR3SET	31:0	Write sets selected bits in PR3; read yields undefined value							
PR3INV	31:0	Write inverts selected bits in PR3; read yields undefined value							

Register 16-1: OCxCON: Output Compare 'x' Control Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	r-x	r-x	r-x	r-x	r-x
ON	FRZ	SIDL	—	—	—	—	—
bit 15							bit 8

r-x	r-x	R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	OC32	OCFLT	OCTSEL	OCM<2:0>		
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: ('0', '1', x = Unknown)		

bit 31-16 **Reserved:** Write '0'; ignore read

bit 15 **ON:** Output Compare Peripheral On bit
- 1 = Output Compare peripheral is enabled.
- 0 = Output Compare peripheral is disabled and not drawing current. SFR modifications are allowed. The status of other bits in this register are not affected by setting or clearing this bit.

Note: When using 1:1 PBCLK divisor, the user's software should not read/write the peripheral's SFRs in the SYSCLK cycle immediately following the instruction that clears the module's ON bit.

bit 14 **FRZ:** Freeze in Debug Exception Mode bit
- 1 = Freeze operation when CPU enters Debug Exception mode
- 0 = Continue operation when CPU enters Debug Exception mode

Note: FRZ is writable in Debug Exception mode only, it is forced to '0' in normal mode.

bit 13 **SIDL:** Stop in IDLE Mode bit
- 1 = Discontinue operation when CPU enters IDLE mode
- 0 = Continue operation in IDLE mode

bit 12-6 **Reserved:** Write '0'; ignore read

bit 5 **OC32:** 32-bit Compare Mode bit
- 1 = OCxR<31:0> and/or OCxRS<31:0> are used for comparisions to the 32-bit timer source
- 0 = OCxR<15:0> and OCxRS<15:0> are used for comparisons to the 16-bit timer source

bit 4 **OCFLT:** PWM Fault Condition Status bit(1)
- 1 = PWM Fault condition has occurred (cleared in HW only)
- 0 = No PWM Fault condition has occurred

Note: This bit is only used when OCM<2:0> = '111'.

bit 3 **OCTSEL:** Output Compare Timer Select bit
- 1 = Timer3 is the clock source for this OCMP module
- 0 = Timer2 is the clock source for this OCMP module

Refer to the device data sheet for specific time bases available to the Output Compare module.

Note 1: Reads as '0' in modes other than PWM mode.

Register 16-1: OCxCON: Output Compare 'x' Control Register (Continued)

bit 2-0 **OCM<2:0>:** Output Compare Mode Select bits
- 111 = PWM mode on OCx; Fault pin enabled
- 110 = PWM mode on OCx; Fault pin disabled
- 101 = Initialize OCx pin low; generate continuous output pulses on OCx pin
- 100 = Initialize OCx pin low; generate single output pulse on OCx pin
- 011 = Compare event toggles OCx pin
- 010 = Initialize OCx pin high; compare event forces OCx pin low
- 001 = Initialize OCx pin low; compare event forces OCx pin high
- 000 = Output compare peripheral is disabled but continues to draw current

Note 1: Reads as '0' in modes other than PWM mode.

Register 16-2: OCxCONCLR: Output Compare 'x' Control Clear Register

R/W-x	
Write clears selected bits in OCxCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in OCxCON**

A write of '1' in one or more bit positions clears the corresponding bit(s) in OCxCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Examples: `OCxCONCLR = 0x00008001` will clear bits 15 and 0 in the OCxCON register.
`localValue = OCxCONCLR` will yield an undefined value.

Register 16-3: OCxCONSET: Output Compare 'x' Control Set Register

R/W-x	
Write sets selected bits in OCxCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in OCxCON**

A write of '1' in one or more bit positions sets the corresponding bit(s) in OCxCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Examples: `OCxCONSET = 0x00008001` will set bits 15 and 0 in the OCxCON register.
`localValue = OCxCONSET` will yield an undefined value.

Register 16-4: OCxCONINV: Output Compare 'x' Control Invert Register

R/W-x	
Write inverts selected bits in OCxCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in OCxCON**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in OCxCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Examples: `OCxCONINV = 0x00008001` will invert bits 15 and 0 in the OCxCON register.
`localValue = OCxCONINV` will yield an undefined value.

Register 16-5: OCxR: Output Compare 'x' Compare Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
OCR<31:24>							
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
OCR<23:16>							
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
OCR<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
OCR<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: ('0', '1', x = Unknown)		

bit 31-16 **OCxR<31:16>:** Upper 16 bits of 32-bit compare value, when OC32 (OCxCON<5>) = 1

bit 15-0 **OCxR<15:0>:** Lower 16 bits of 32-bit compare value or entire 16 bits of 16-bit compare value when OC32 = 0

Register 16-6: OCxRCLR: Output Compare 'x' Compare Clear Register

R/W-x	
Write clears selected bits in OCxR, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in OCxR**

A write of '1' in one or more bit positions clears the corresponding bit(s) in OCxR register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Examples: `OCxRCLR = 0x00008001` will clear bits 15 and 0 in the OCxR register.
`localValue = OCxRCLR` will yield an undefined value.

Register 16-7: OCxRSET: Output Compare 'x' Compare Set Register

R/W-x	
Write sets selected bits in OCxR, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in OCxR**

A write of '1' in one or more bit positions sets the corresponding bit(s) in OCxR register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Examples: `OCxRSET = 0x00008001` will set bits 15 and 0 in the OCxR register.
`localValue = OCxRSET` will yield an undefined value.

Register 16-8: OCxRINV: Output Compare 'x' Compare Invert Register

R/W-x	
Write inverts selected bits in OCxR, read yields undefined value	
bit 31	bit 0

31-0 **Inverts selected bits in OCxR**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in OCxR register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Examples: `OCxRINV = 0x00008001` will invert bits 15 and 0 in the OCxR register.
`localValue = OCxRINV` will yield an undefined value.

Register 16-9: OCxRS: Output Compare x Secondary Compare Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
OCRS<31:24>							
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
OCRS<23:16>							
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
OCRS<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
OCRS<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **OCxRS<31:16>:** Upper 16 bits of 32-bit compare value when OC32 (OCxCON<5>) = 1

bit 15-0 **OCxRS<15:0>:** Lower 16 bits of 32-bit compare value or entire 16 bits of 16-bit compare value when OC32 = 0

Register 16-10: OCxRSCLR: Output Compare 'x' Secondary Compare Clear Register

R/W-x	
Write clears selected bits in OCxRS, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in OCxRS**

A write of '1' in one or more bit positions clears the corresponding bit(s) in OCxRS register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Examples: `OCxRSCLR = 0x00008001` will clear bits 15 and 0 in the OCxRS register.
`localValue = OCxRSCLR` will yield an undefined value.

Register 16-11: OCxRSSET: Output Compare 'x' Secondary Compare Set Register

R/W-x	
Write sets selected bits in OCxRS, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in OCxRS**

A write of '1' in one or more bit positions sets the corresponding bit(s) in OCxRS register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Examples: `OCxRSSET = 0x00008001` will set bits 15 and 0 in the OCxRS register.
`localValue = OCxRSSET` will yield an undefined value.

Register 16-12: OCxRSINV: Output Compare 'x' Secondary Compare Invert Register

R/W-x	
Write inverts selected bits in OCxRS, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in OCxRS**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in OCxRS register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Examples: `OCxRSINV = 0x00008001` will invert bits 15 and 0 in the OCxRS register.
`localValue = OCxRSINV` will yield an undefined value.

Register 16-13: IFS0: Interrupt Flag Status Register 0(1)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
I2C1SIF	I2C1BIF	U1TXIF	U1RXIF	U1EIF	SPI1RXIF	SPI1TXIF	SPI1EIF
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CNIF	OC5IF	IC5IF	T5IF	INT4IF	OC4IF	IC4IF	T4IF
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INT3IF	OC3IF	IC3IF	T3IF	INT2IF	OC2IF	IC2IF	T2IF
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INT1IF	OC1IF	IC1IF	T1IF	INT0IF	CS1IF	CS0IF	CTIF
bit 7							bit 0

Legend:			
R = readable bit	W = writable bit	P = programmable	r = reserved bit
U = unimplemented bit, read as '0'		-n = bit value at POR: ('0', '1', x = unknown)	

bit 22 **OC5IF:** Output Compare 5 Interrupt Request Flag bit
1 = Interrupt request has occurred
0 = No interrupt request has a occurred

bit 18 **OC4IF:** Output Compare 4 Interrupt Request Flag bit
1 = Interrupt request has occurred
0 = No interrupt request has a occurred

bit 14 **OC3IF:** Output Compare 3 Interrupt Request Flag bit
1 = Interrupt request has occurred
0 = No interrupt request has a occurred

bit 10 **OC2IF:** Output Compare 2 Interrupt Request Flag bit
1 = Interrupt request has occurred
0 = No interrupt request has a occurred

bit 6 **OC1IF:** Output Compare 1 Interrupt Request Flag bit
1 = Interrupt request has occurred
0 = No interrupt request has a occurred

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the output compare module.

Register 16-14: IEC0: Interrupt Enable Control Register 0(1)

I2C1SIE	I2C1BIE	U1TXIE	U1RXIE	U1EIE	SPI1RXIE	SPI1TXIE	SPI1EIE
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CNIE	OC5IE	IC5IE	T5IE	INT4IE	OC4IE	IC4IE	T4IE
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INT3IE	OC3IE	IC3IE	T3IE	INT2IE	OC2IE	IC2IE	T2IE
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INT1IE	OC1IE	IC1IE	T1IE	INT0IE	CS1IE	CS0IE	CTIE
bit 7							bit 0

Legend:			
R = readable bit	W = writable bit	P = programmable	r = reserved bit
U = unimplemented bit, read as '0'		-n = bit value at POR: ('0', '1', x = unknown)	

bit 22 **OC5IE:** Output Compare 5 Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 18 **OC4IE:** Output Compare 4 Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 14 **OC3IE:** Output Compare 3 Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 10 **OC2IE:** Output Compare 2 Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 6 **OC1IE:** Output Compare 1 Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the output compare module.

Register 16-15: IPC1: Interrupt Priority Control Register 1(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	INT1IP<2:0>			INT1IS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	OC1IP<2:0>			OC1IS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	IC1IP<2:0>			IC1IS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	T1IP<2:0>			T1IS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 20-18 **OC1IP<2:0>:** Output Compare 1 Interrupt Priority bits
- 111 = Interrupt priority is 7
- 110 = Interrupt priority is 6
- 101 = Interrupt priority is 5
- 100 = Interrupt priority is 4
- 011 = Interrupt priority is 3
- 010 = Interrupt priority is 2
- 001 = Interrupt priority is 1
- 000 = Interrupt is disabled

bit 17-16 **OC1IS<1:0>:** Output Compare 1 Interrupt Subpriority bits
- 11 = Interrupt subpriority is 3
- 10 = Interrupt subpriority is 2
- 01 = Interrupt subpriority is 1
- 00 = Interrupt subpriority is 0

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the output compare module.

Register 16-16: IPC2: Interrupt Priority Control Register 2(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	INT2IP<2:0>			INT2IS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	OC2IP<2:0>			OC2IS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	IC2IP<2:0>			IC2IS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	T2IP<2:0>			T2IS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 20 - 18 **OC2IP<2:0>:** Output Compare 2 Interrupt Priority bits

111 = Interrupt priority is 7
110 = Interrupt priority is 6
101 = Interrupt priority is 5
100 = Interrupt priority is 4
011 = Interrupt priority is 3
010 = Interrupt priority is 2
001 = Interrupt priority is 1
000 = Interrupt is disabled

bit 17-16 **OC2IS<1:0>:** Output Compare 2 Interrupt Subpriority bits

11 = Interrupt subpriority is 3
10 = Interrupt subpriority is 2
01 = Interrupt subpriority is 1
00 = Interrupt subpriority is 0

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the output compare module.

Register 16-17: IPC3: Interrupt Priority Control Register 3(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	INT3IP<2:0>			INT3IS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	OC3IP<2:0>			OC3IS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	IC3IP<2:0>			IC3IS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	T3IP<2:0>			T3IS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 20-18 **OC3IP<2:0>:** Output Compare 3 Interrupt Priority bits
- 111 = Interrupt priority is 7
- 110 = Interrupt priority is 6
- 101 = Interrupt priority is 5
- 100 = Interrupt priority is 4
- 011 = Interrupt priority is 3
- 010 = Interrupt priority is 2
- 001 = Interrupt priority is 1
- 000 = Interrupt is disabled

bit 17-16 **OC3IS<1:0>:** Output Compare 3 Interrupt Subpriority bits
- 11 = Interrupt subpriority is 3
- 10 = Interrupt subpriority is 2
- 01 = Interrupt subpriority is 1
- 00 = Interrupt subpriority is 0

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the output compare module.

Register 16-18: IPC4: Interrupt Priority Control Register 4(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	INT4IP<2:0>			INT4IS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	OC4IP<2:0>			OC4IS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	IC4IP<2:0>			IC4IS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	T4IP<2:0>			T4IS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 20-18 **OC4IP<2:0>:** Output Compare 4 Interrupt Priority bits

111 = Interrupt priority is 7
110 = Interrupt priority is 6
101 = Interrupt priority is 5
100 = Interrupt priority is 4
011 = Interrupt priority is 3
010 = Interrupt priority is 2
001 = Interrupt priority is 1
000 = Interrupt is disabled

bit 17-16 **OC4IS<1:0>:** Output Compare 4 Interrupt Subpriority bits

11 = Interrupt subpriority is 3
10 = Interrupt subpriority is 2
01 = Interrupt subpriority is 1
00 = Interrupt subpriority is 0

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the output compare module.

Register 16-19: IPC5: Interrupt Priority Control Register 5(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	SPI1IP<2:0>			SPI1IS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	OC5IP<2:0>			OC5IS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	IC5IP<2:0>			IC5IS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	T5IP<2:0>			T5IS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 20-18 **OC5IP<2:0>:** Output Compare 5 Interrupt Priority bits
- 111 = Interrupt priority is 7
- 110 = Interrupt priority is 6
- 101 = Interrupt priority is 5
- 100 = Interrupt priority is 4
- 011 = Interrupt priority is 3
- 010 = Interrupt priority is 2
- 001 = Interrupt priority is 1
- 000 = Interrupt is disabled

bit 17-16 **OC5IS<1:0>:** Output Compare 5 Interrupt Subpriority bits
- 11 = Interrupt subpriority is 3
- 10 = Interrupt subpriority is 2
- 01 = Interrupt subpriority is 1
- 00 = Interrupt subpriority is 0

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the output compare module.

Register 16-20: T2CON: Time Base Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	r-x	r-x	r-x	r-x	r-x
ON	FRZ	SIDL	—	—	—	—	—
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	r-x	R/W-0	r-x
TGATE	TCKPS<2:0>			T32	—	TCS	—
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 15 **ON:** TMR2 On bit

1 = Peripheral is enabled
0 = Peripheral is disabled

Note: When using 1:1 PBCLK divisor, the user’s software should not read/write the peripheral’s SFRs in the SYSCLK cycle immediately following the instruction that clears the module’s ON bit.

bit 14 **FRZ:** Freeze in Debug Exception Mode bit

1 = Freeze operation when CPU is in Debug Exception mode
0 = Continue operation even when CPU is in Debug Exception mode

Note: FRZ is writable in Debug Exception mode only, it is forced to ‘0’ in normal mode.

bit 13 **SIDL:** Stop in IDLE Mode bit

1 = Discontinue operation when device enters IDLE mode
0 = Continue operation even in IDLE mode

bit 7 **TGATE:** Timer Gated Time Accumulation Enable bit

When TCS = 1:
This bit is ignored and reads as ‘0’
When TCS = ‘0’:
1 = Gated time accumulation is enabled
0 = Gated time accumulation is disabled

bit 6-4 **TCKPS<2:0>:** Timer Input Clock Prescale Select bits

111 = 1:256 prescale value
110 = 1:64 prescale value
101 = 1:32 prescale value
100 = 1:16 prescale value
011 = 1:8 prescale value
010 = 1:4 prescale value
001 = 1:2 prescale value
000 = 1:1 prescale value

bit 3 **T32:** 32-bit Timer Mode Select bits

1 = TMR2 and TMR3 form a 32-bit timer
0 = TMR2 and TMR3 are separate 16-bit timers

Register 16-20: T2CON: Time Base Register (Continued)

bit 1 **TCS:** TMR2 Clock Source Select bit

1 = External clock from T2CK pin

0 = Internal peripheral clock

Register 16-21: T2CONCLR: Time Base Register

R/W-x	
Write clears selected bits in T2CON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in T2CON**

A write of '1' in one or more bit positions clears the corresponding bit(s) in T2CON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `T2CONCLR = 0x00008000` will clear bit 15 in the T2CON register.

Register 16-22: T2CONSET: Output Compare 'x' Secondary Compare Set Register

R/W-x	
Write sets selected bits in T2CON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in T2CON**

A write of '1' in one or more bit positions sets the corresponding bit(s) in T2CON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `T2CONSET = 0x00008000` will set bit 15 in the T2CON register.

Register 16-23: T2CONINV: Output Compare 'x' Secondary Compare Invert Register

R/W-x	
Write inverts selected bits in T2CON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in T2CON**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in T2CON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `T2CONINV = 0x00008000` will invert bit 15 in the T2CON register.

Register 16-24: TMR2: Timer Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
TMR<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
TMR<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 15-0 **TMR2<15:0>:** Timer Count Register

16-bit mode:
These bits represent the complete 16-bit timer count.

32-bit mode (Timer Type B only):
Timer2 and Timer4
These bits represent the least significant half word (16 bits) of the 32-bit timer count.

Register 16-25: TMR2CLR: Timer Clear Register

Write clears selected bits in TMR2, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in TMR2**

A write of '1' in one or more bit positions clears the corresponding bit(s) in TMR2 register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `TMR2CLR = 0x00008001` will clear bits 15 and 0 in TMR2 register.

Register 16-26: TMR2SET: Timer Set Register

Write sets selected bits in TMR2, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in TMR2**

A write of '1' in one or more bit positions sets the corresponding bit(s) in TMR2 register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `TMR2SET = 0x00008001` will set bits 15 and 0 in TMR2 register.

Register 16-27: TMR2INV: Timer Invert Register

Write inverts selected bits in TMR2, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in TMR2**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in TMR2 register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `TMR2INV = 0x00008001` will invert bits 15 and 0 in TMR2 register.

Register 16-28: PR2: Period Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PR<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PR<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **PR<31:16>:** Unimplemented

bit 15-0 **PR<15:0>:** 16-bit Timer2 period match value. Provides lower half of the 32-bit period match value when Timer2 and Timer3 are configured to form a 32-bit timer.

Register 16-29: PR2CLR: Period 2 Clear Register

R/W-x	
Write clears selected bits in PR2, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in PR2**

A write of '1' in one or more bit positions clears the corresponding bit(s) in PR2 register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `PR2CLR = 0x00008001` will clear bits 15 and 0 in the PR2 register.

Register 16-30: PR2SET: Period 2 Set Register

R/W-x	
Write sets selected bits in PR2, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in PR2**

A write of '1' in one or more bit positions sets the corresponding bit(s) in PR2 register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `PR2SET = 0x00008001` will set bits 15 and 0 in the PR2 register.

Register 16-31: PR2INV: Period 2 Invert Register

R/W-x	
Write inverts selected bits in PR2, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in PR2**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in PR2 register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `PR2INV = 0x00008001` will invert bits 15 and 0 in the PR2 register.

Register 16-32: T3CON: Time Base Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	r-x	r-x	r-x	r-x	r-x
ON	FRZ	SIDL	—	—	—	—	—
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	r-0	r-x	R/W-0	r-x
TGATE	TCKPS<2:0>			—	—	TCS	—
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 15 **ON:** TMR3 On bit

1 = Peripheral is enabled
0 = Peripheral is disabled

Note: When using 1:1 PBCLK divisor, the user’s software should not read/write the peripheral’s SFRs in the SYSCLK cycle immediately following the instruction that clears the module’s ON bit.

bit 14 **FRZ:** Freeze in Debug Exception Mode bit

1 = Freeze operation when CPU is in Debug Exception mode
0 = Continue operation even when CPU is in Debug Exception mode

Note: FRZ is writable in Debug Exception mode only, it is forced to ‘0’ in normal mode.

bit 13 **SIDL:** Stop in IDLE Mode bit

1 = Discontinue operation when device enters IDLE mode
0 = Continue operation even in IDLE mode

bit 7 **TGATE:** Timer Gated Time Accumulation Enable bit

When TCS = 1:
This bit is ignored and reads ‘0’
When TCS = ‘0’:
1 = Gated time accumulation is enabled
0 = Gated time accumulation is disabled

bit 6-4 **TCKPS<2:0>:** Timer Input Clock Prescale Select bits

111 = 1:256 prescale value
110 = 1:64 prescale value
101 = 1:32 prescale value
100 = 1:16 prescale value
011 = 1:8 prescale value
010 = 1:4 prescale value
001 = 1:2 prescale value
000 = 1:1 prescale value

bit 1 **TCS:** TMR3 Clock Source Select bit

1 = External clock from T3CK pin
0 = Internal peripheral clock

Register 16-33: T3CONCLR: Time Base Register

R/W-x	
Write clears selected bits in T3CON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in T3CON**

A write of '1' in one or more bit positions clears the corresponding bit(s) in T3CON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `T3CONCLR = 0x00008000` will clear bit 15 in the T3CON register.

Register 16-34: T3CONSET: Output Compare 'x' Secondary Compare Set Register

R/W-x	
Write sets selected bits in T3CON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in T2CON**

A write of '1' in one or more bit positions sets the corresponding bit(s) in T3CON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `T3CONSET = 0x00008000` will set bit 15 in the T3CON register.

Register 16-35: T3CONINV: Output Compare 'x' Secondary Compare Invert Register

R/W-x	
Write inverts selected bits in T3CON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in T3CON**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in T3CON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `T3CONINV = 0x00008000` will invert bit 15 in the T3CON register.

Register 16-36: TMR3: Timer Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
TMR<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
TMR<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 15-0 **TMR3<15:0>:** Timer Count Register

16-bit mode:
These bits represent the complete 16-bit timer count.

32-bit mode (Timer Type B only):
Timer3 and Timer5
These bits represent the most significant half word (16 bits) of the 32-bit timer count.

Register 16-37: TMR3CLR: Timer Clear Register

Write clears selected bits in TMR3, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in TMR3**

A write of '1' in one or more bit positions clears the corresponding bit(s) in TMR3 register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `TMR3CLR = 0x00008001` will clear bits 15 and 0 in TMR3 register.

Register 16-38: TMR3SET: Timer Set Register

Write sets selected bits in TMR3, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in TMR3**

A write of '1' in one or more bit positions sets the corresponding bit(s) in TMR3 register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `TMR3SET = 0x00008001` will set bits 15 and 0 in TMR3 register.

Register 16-39: TMR3INV: Timer Invert Register

Write inverts selected bits in TMR3, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in TMR3**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in TMR3 register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `TMR3INV = 0x00008001` will invert bits 15 and 0 in TMR3 register.

Register 16-40: PR3: Period 3 Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PR<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PR<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **PR<31:16>:** Unimplemented

bit 15-0 **PR<15:0>:** 16-bit Timer3 period match value. Provides upper half of the 32-bit period match value when Timer 2 and Timer3 are configured to form a 32-bit timer.

Register 16-41: PR3CLR: Period 3 Clear Register

R/W-x	
Write clears selected bits in PR3, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in PR3**

A write of '1' in one or more bit positions clears the corresponding bit(s) in PR3 register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `PR3CLR = 0x8001` will clear bits 15 and 0 in the PR3 register.

Register 16-42: PR3SET: Period 3 Set Register

R/W-x	
Write sets selected bits in PR3, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in PR3**

A write of '1' in one or more bit positions sets the corresponding bit(s) in PR3 register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `PR3SET = 0x00008001` will set bits 15 and 0 in the PR3 register.

Register 16-43: PR3INV: Period 3 Invert Register

R/W-x	
Write inverts selected bits in PR3, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in PR3**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in PR3 register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `PR3INV = 0x00008001` will invert bits 15 and 0 in the PR3 register.

16.3 OPERATION

Each Output Compare module has the following modes of operation:

- Single Compare Match mode
  - With output drive high
  - With output drive low
  - With output drive toggles
- Dual Compare Match mode
  - With single output pulse
  - With continuous output pulses
- Simple Pulse-Width Modulation mode
  - Without fault protection input
  - With fault protection input

Notes: It is required that the user turn off the Output Compare module (i.e., clear OCM<2:0> (OCxCON<2:0>)) before switching to a new mode. Changing modes while the module is in operation may produce unexpected results.

In this section, a reference to any SFRs associated with the selected timer source is indicated by a 'y' suffix. For example, PR2 is the Period register for the selected timer source, while TyCON is the Timer Control register for the selected timer source.

16.3.1 Single Compare Match Mode

When control bits OCM<2:0> (OCxCON<2:0>) are set to '001', '010' or '011', the selected output compare channel is configured for one of three Single Output Compare Match modes. The compare time base must also be enabled.

In the Single Compare mode, the OCxR register is loaded with a value and is compared to the selected incrementing timer register, TMRy. On a compare match event, one of the following events will take place:

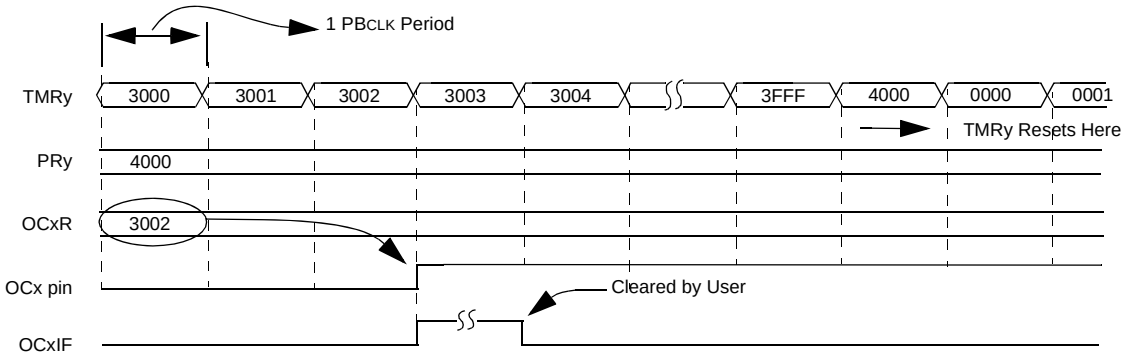
- Compare forces OCx pin high; initial state of pin is low. Interrupt is generated on the single compare match event.
- Compare forces OCx pin low; initial state of pin is high. Interrupt is generated on the single compare match event.
- Compare toggles OCx pin. Toggle event is continuous and an interrupt is generated for each toggle event.

16.3.1.1 Compare Mode Output Driven High

To configure the Output Compare module for this mode, set control bits OCM<2:0> = '001'. The compare time base must also be enabled. Once this Compare mode has been enabled, the output pin, OCx, will be driven low and remain low until a match occurs between the TMRy and OCxR registers. Please note the following key timing events (refer to Figure 16-2):

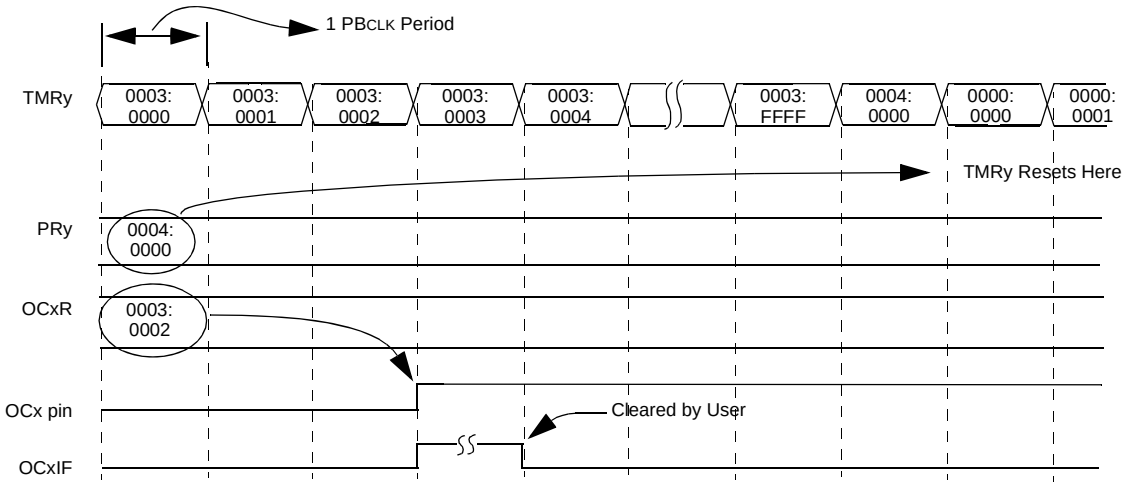
- The OCx pin is driven high one peripheral clock after the compare match occurs between the compare time base and the OCxR register. The OCx pin will remain high until a mode change has been made or the module is disabled.
- The compare time base will count up to the value contained in the associated period register and then reset to 0x0000 on the next PBCLK.
- The respective channel interrupt flag, OCxIF (refer to the IFS0 register for the position of the interrupt flag bit for each of the Output Compare channels), is asserted when the OCx pin is driven high.

Figure 16-2: Single Compare Mode: Set OCx High on Compare Match Event (16-Bit Mode)



Note: An 'x' represents the output compare channel number. A 'y' represents the time base number.

Figure 16-3: Single Compare Mode: Set OCx High on Compare Match Event (32-Bit Mode)



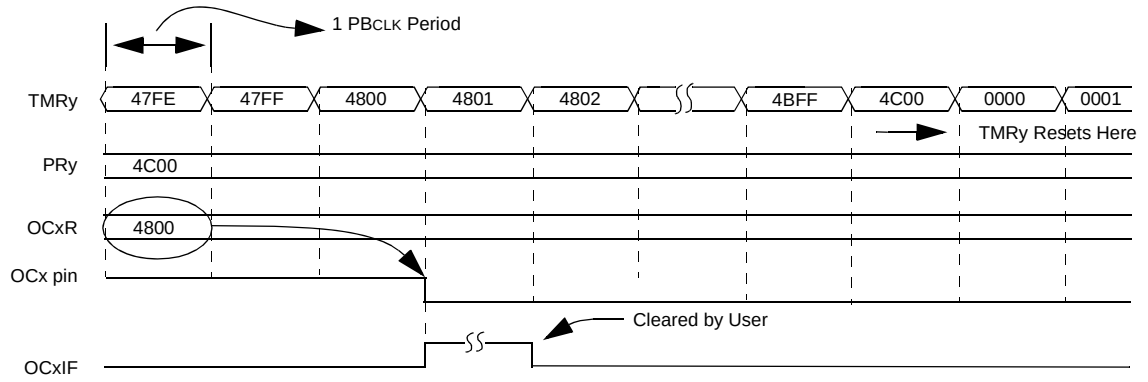
Note: An 'x' represents the output compare channel number. A 'y' represents the time base number.

16.3.1.2 Compare Mode Output Driven Low

To configure the output compare module for this mode, set control bits OCM<2:0> = '010'. The compare time base must also be enabled. Once this Compare mode has been enabled, the output pin, OCx, will be driven high and remain high until a match occurs between the Timer and OCxR registers. Please note the following key timing events (refer to Figure 16-4):

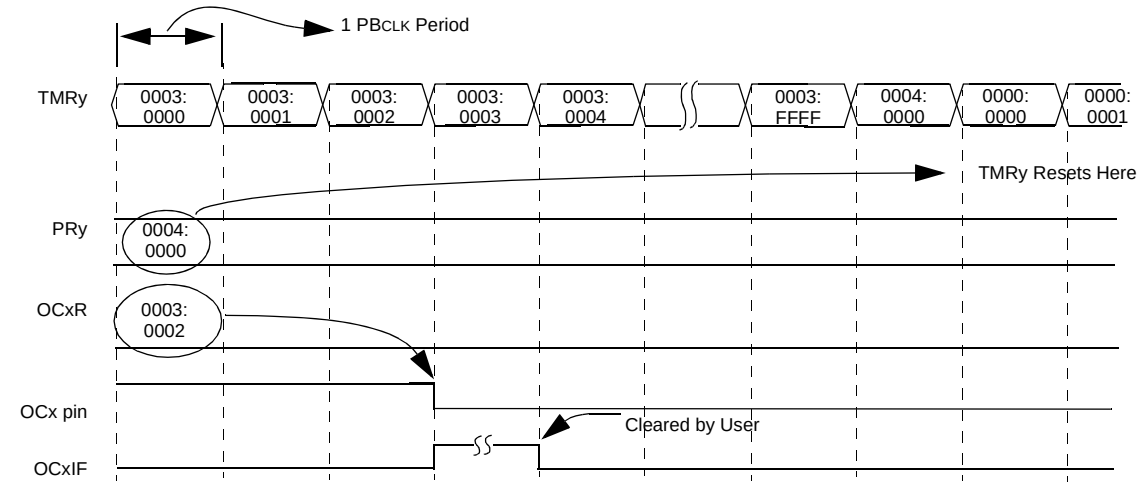
- The OCx pin is driven low one PBCLK after the compare match occurs between the compare time base and the OCxR register. The OCx pin will remain low until a mode change has been made or the module is disabled.
- The compare time base will count up to the value contained in the associated period register and then reset to 0x0000 on the next PBCLK .
- The respective channel interrupt flag, OCxIF, is asserted when the OCx pin is driven low.

Figure 16-4: Single Compare Mode: Force OCx Low on Compare Match Event (16-Bit Mode)



Note: An 'x' represents the output compare channel number. A 'y' represents the time base number.

Figure 16-5: Single Compare Mode: Set OCx Low on Compare Match Event (32-Bit Mode)



Note: An 'x' represents the output compare channel number. A 'y' represents the time base number.

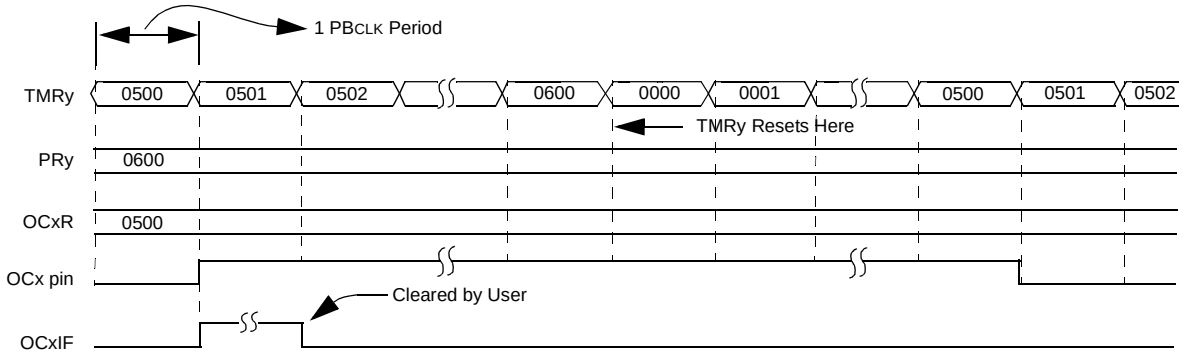
16.3.1.3 Single Compare Mode Toggle Output

To configure the Output Compare module for this mode, set control bits OCM<2:0> = '011'. In addition, Timer2 or Timer3 must be selected and enabled. Once this Compare mode has been enabled, the output pin, OCx, will be initially driven low and then toggle on each and every subsequent match event between the Timer and OCxR registers. Please note the following key timing events (refer to Figure 16-6 and Figure 16-8):

- The OCx pin is toggled one PBCLK after the compare match occurs between the compare time base and the OCxR register. The OCx pin will remain at this new state until the next toggle event, or until a mode change has been made or the module is disabled.
- The compare time base will count up to the contents in the period register and then reset to 0x0000 on the next PBCLK .
- The respective channel interrupt flag, OCxIF, is asserted when the OCx pin is toggled.

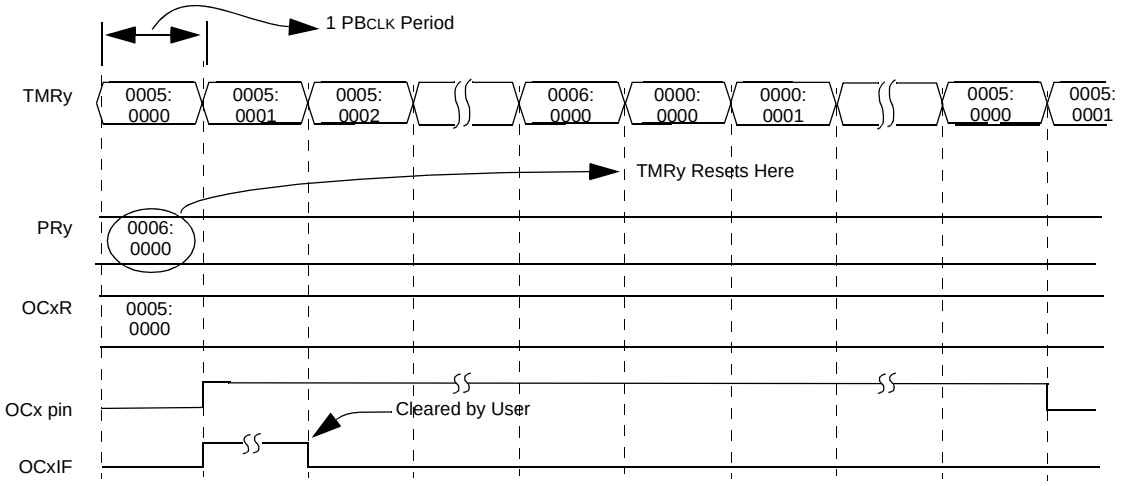
Note:	The internal OCx pin output logic is set to a logic '0' on a device Reset. However, the operational OCx pin state for the Toggle mode can be set by the user software. Example 16-1 shows a code example for defining the desired initial OCx pin state in the Toggle mode of operation.

Figure 16-6: Single Compare Mode: Toggle Output on Compare Match Event (16-Bit Mode)



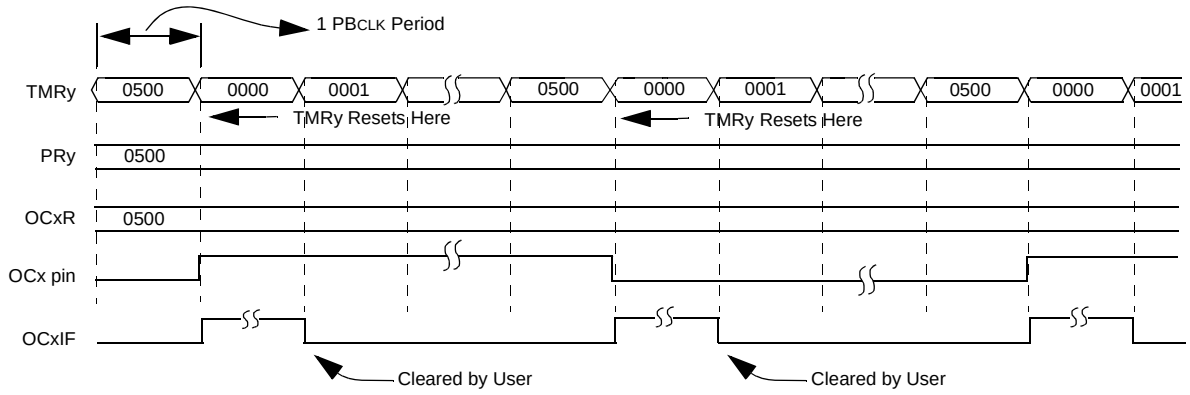
Note: An 'x' represents the output compare channel number. A 'y' represents the time base number.

Figure 16-7: Single Compare Mode: Toggle Output on Compare Match Event (32-Bit Mode)



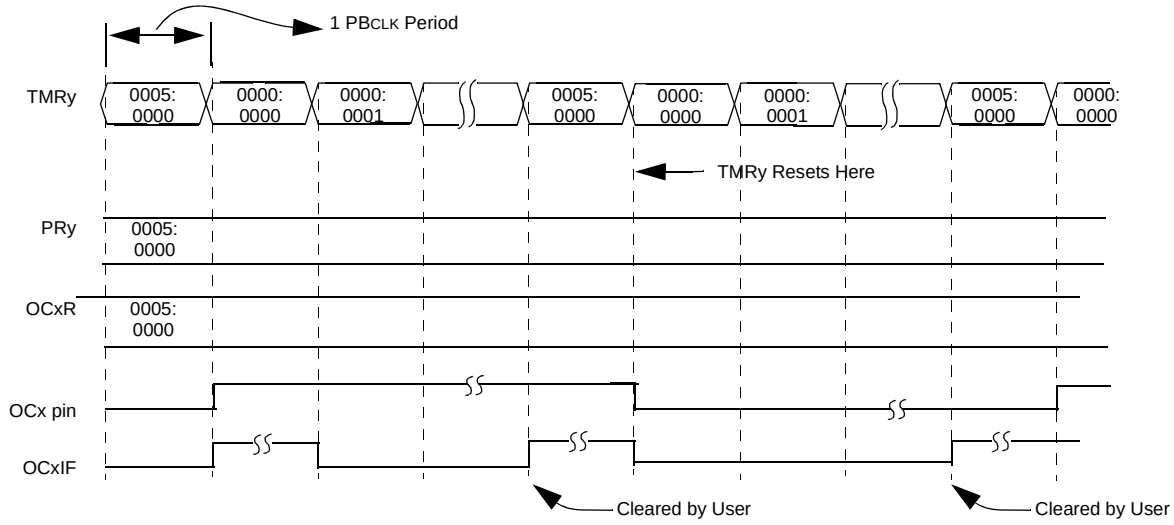
Note: An 'x' represents the output compare channel number. A 'y' represents the time base number.

Figure 16-8: Single Compare Mode: Toggle Output on Compare Match Event (PRy = OCxR, 16-Bit Mode)



Note: An 'x' represents the output compare channel number. A 'y' represents the time base number.

Figure 16-9: Single Compare Mode: Toggle Output on Compare Match Event (PRy = OCxR, 32-Bit Mode)



Note: An 'x' represents the output compare channel number. A 'y' represents the time base number.

Example 16-1: Compare Mode Toggle Mode Pin State Setup (16-Bit Mode)

```
// The following code example illustrates how to define the initial
// OC1 pin state for the output compare toggle mode of operation.

                                    // Toggle mode with initial OC1 pin state set low

OC1CON = 0x0001;                    // Configure module for OC1 pin low, toggle high
OC1CONSET = 0x8000;                 // Enable OC1 module
```


Example 16-2: Compare Mode Toggle Mode Pin State Setup (32-Bit Mode)

```
// The following code example illustrates how to define the initial
// OC1 pin state for the output compare toggle mode of operation.

                                    // Toggle mode with initial OC1 pin state set low

OC1CON = 0x0021;                    // Configure module for OC1 pin low, toggle high,
                                    // 32-bit mode
OC1CONSET = 0x8000;                 // Enable OC1 module
```

Example 16-3 shows example code for the configuration and interrupt service of the Single Compare mode toggle event.

Example 16-3: Compare Mode Toggle Setup and Interrupt Servicing (16-Bit Mode)

```
// The following code example will set the Output Compare 1 module
// for interrupts on the toggle event and select Timer2 as the clock
// source for the compare time base.

T2CON = 0x0010;                     // Configure Timer2 for a prescaler of 2

OC1CON = 0x0000;                    // Turn off OC1 while doing setup.
OC1CON = 0x0003;                    // Configure for compare toggle mode
OC1R = 0x0500;                      // Initialize Compare Register 1
PR2 = 0x0500;                       // Set period

                                    // Configure int
IFS0CLR = 0x0040;                   // Clear the OC1 interrupt flag
IEC0SET = 0x040;                    // Enable OC1 interrupt
IPC1SET = 0x001C0000;               // Set OC1 interrupt priority to 7,
                                    // the highest level
IPC1SET = 0x00030000;               // Set Subpriority to 3, maximum

T2CONSET = 0x8000;                  // Enable Timer 2
OC1CONSET = 0x8000;                 // Enable OC1

// Example code for Output Compare 1 ISR:

void __ISR(_OUTPUT_COMPARE_1_VECTOR, ipl7) OC1_IntHandler (void)
{
// insert user code here
IFS0CLR = 0x0040;                   // Clear the OC1 interrupt flag
}
```

Example 16-4: Compare Mode Toggle Setup and Interrupt Servicing (32-Bit Mode)

```
// The following code example will set the Output Compare 1 module
// for interrupts on the toggle event and select the Timer2/Timer3 pair as
// the 32-bit as the clock source for the compare time base.

T2CON = 0x0018;                       // Configure Timer2 for 32-bit operation
                                      // with a prescaler of 2. The Timer2/Timer3
                                      // pair is accessed via registers associated
                                      // with the Timer2 register

OC1CON = 0x0000;                      // Turn off OC1 while doing setup.
OC1CON = 0x0023;                      // Configure for compare toggle mode
OC1R = 0x00500000;                    // Initialize Compare Register 1
PR2 = 0x00500000;                     // Set period (PR2 is now 32-bits wide)

                                      // configure int
IFS0CLR = 0x00000040;                 // Clear the OC1 interrupt flag
IFS0SET = 0x00000040;                 // Enable OC1 interrupt
IPC1SET = 0x001C0000;                 // Set OC1 interrupt priority to 7,
                                      // the highest level
IPC1SET = 0x00030000;                 // Set Subpriority to 3, maximum

T2CONSET = 0x8000;                    // Enable Timer2
OC1CONSET = 0x8000;                   // Enable OC1

// Example code for Output Compare 1 ISR:

void__ISR (_OUTPUT_COMPARE_1_VECTOR, ipl7) OC1_IntlHandler (void)
{
// insert user code here
IFS0CLR = 0x0040;                     // Clear the OC1 interrupt flag
}
```


16.3.2 Dual Compare Match Mode

When control bits OCM<2:0> = 100 or '101' (OCxCON<2:0>), the selected output compare channel is configured for one of two Dual Compare Match modes:

- Single Output Pulse mode
- Continuous Output Pulse mode

In the Dual Compare mode, the module uses both the OCxR and OCxRS registers for the compare match events. The OCxR register is compared against the incrementing timer count, TMRy, and the leading (rising) edge of the pulse is generated at the OCx pin on a compare match event. The OCxRS register is then compared to the same incrementing timer count, TMRy, and the trailing (falling) edge of the pulse is generated at the OCx pin on a compare match event.

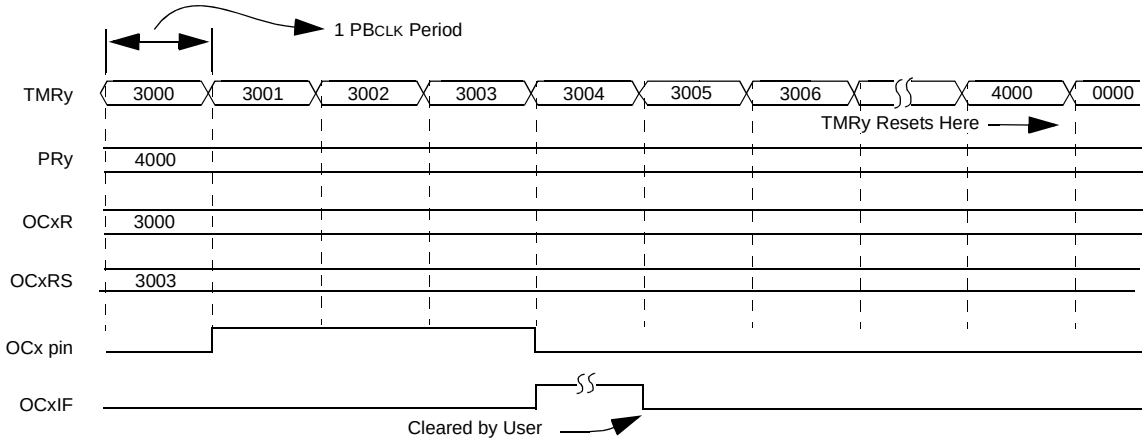
16.3.2.1 Dual Compare Mode: Single Output Pulse

To configure the Output Compare module for the Single Output Pulse mode, set control bits OCM<2:0> = 100. In addition, the compare time base must be selected and enabled. Once this mode has been enabled, the output pin, OCx, will be driven low and remain low until a match occurs between the time base and OCxR registers. Please note the following key timing events (refer to Figure 16-10 and Figure 16-12):

- The OCx pin is driven high one peripheral clock after the compare match occurs between the compare time base and the OCxR register. The OCx pin will remain high until the next match event occurs between the time base and the OCxRS register. At this time, the pin will be driven low. The OCx pin will remain low until a mode change has been made or the module is disabled.
- The compare time base will count up to the value contained in the associated period register and then reset to 0x0000 on the next instruction clock.
- If the time base period register contents are less than the OCxRS register contents, then no falling edge of the pulse is generated. The OCx pin will remain high until OCxRS <= PR2, or a mode change or Reset condition has occurred.
- The respective channel interrupt flag, OCxIF, is asserted when the OCx pin is driven low (falling edge of single pulse).

Figure 16-10 depicts the General Dual Compare mode generating a single output pulse. Figure 16-12 depicts another timing example where OCxRS > PR2. In this example, no falling edge of the pulse is generated because the compare time base resets before counting up to 0x4100.

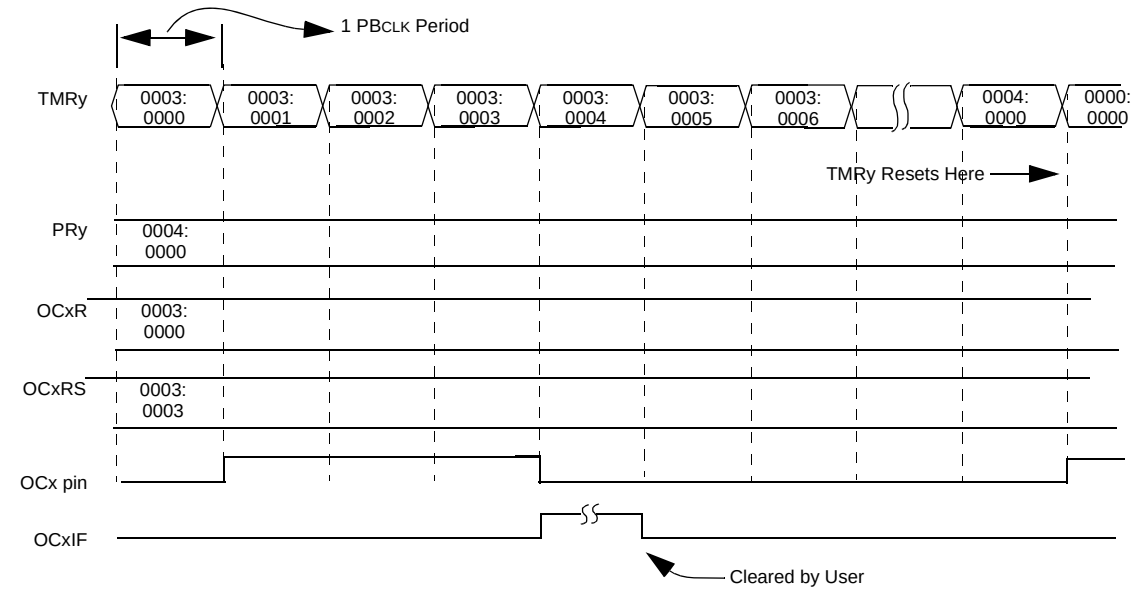
Figure 16-10: Dual Compare Mode (16-Bit Mode)



Note 1: An 'x' represents the output compare channel number. A 'y' represents the time base number.

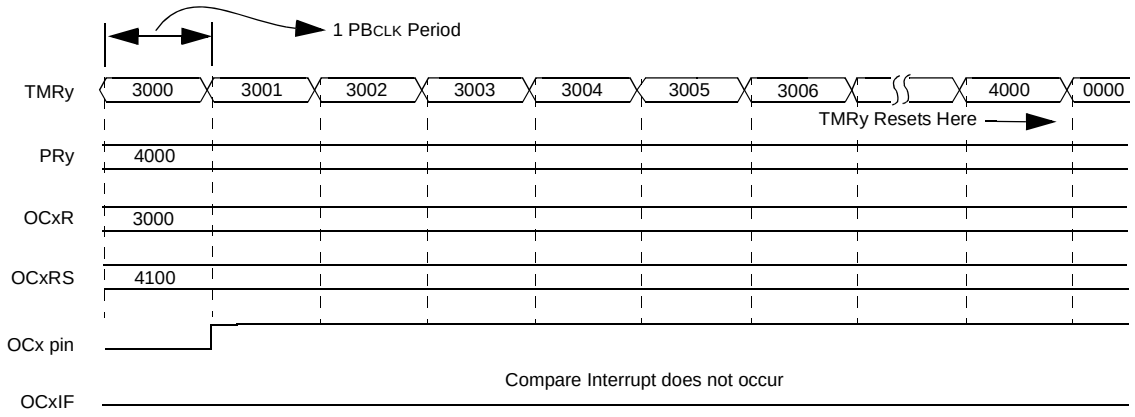
2: OCxR = Compare Register; OCxRS = Secondary Compare Register.

Figure 16-11: Dual Compare Mode (32-Bit Mode)



Note: An 'x' represents the output compare channel number. A 'y' represents the time base number.

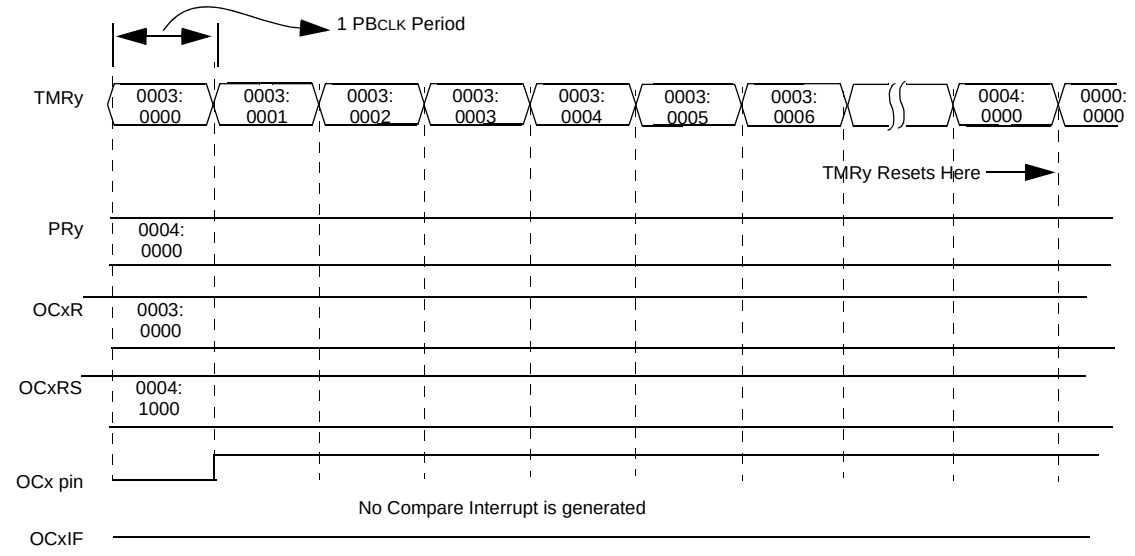
Figure 16-12: Dual Compare Mode: Single Output Pulse (OCxRS > PRy, 16-Bit Mode)



Note 1: An 'x' represents the output compare channel number. A 'y' represents the time base number.

2: OCxR = Compare Register; OCxRS = Secondary Compare Register.

Figure 16-13: Dual Compare Mode: Single Output Pulse (OCxRS > PRy, 32-Bit Mode)



Note: An 'x' represents the output compare channel number. A 'y' represents the time base number.

16.3.2.2 Setup for Single Output Pulse Generation

When control bits OCM<2:0> (OCxCON<2:0>) are set to '100', the selected output compare channel initializes the OCx pin to the low state and generates a single output pulse.

To generate a single output pulse, the following steps are required (these steps assume the timer source is initially turned off, but this is not a requirement for the module operation):

1. Determine the peripheral clock cycle time.
2. Calculate the time to the rising edge of the output pulse relative to the TMRy start value (0x0000).
3. Calculate the time to the falling edge of the pulse based on the desired pulse width and the time to the rising edge of the pulse.
4. Write the values computed in steps 2 and 3 above into the compare register, OCxR, and the secondary compare register, OCxRS, respectively.
5. Set the timer period register, PRy, to value equal to or greater than value in OCxRS, the secondary compare register.
6. Set OCM<2:0> = 100 and the OCTSEL (OCxCON<3>) bit to the desired timer source. The OCx pin state will now be driven low.
7. Set the ON (TyCON<15>) bit to '1', to enable the timer.
8. Upon the first match between TMRy and OCxR, the OCx pin will be driven high.
9. When the incrementing timer, TMRy, matches the secondary compare register, OCxRS, the second and trailing edge (high-to-low) of the pulse is driven onto the OCx pin. No additional pulses are driven onto the OCx pin and it remains at low. As a result of the second compare match event, the OCxIF interrupt flag bit is set, which will result in an interrupt (if it is enabled by setting the OCxIE bit). For further information on peripheral interrupts, refer to **Section 8. "Interrupts"**.
10. To initiate another single pulse output, change the timer and compare register settings, if needed, and then issue a write to set the OCM<2:0> (OCxCON<2:0>) bits to '100'. Disabling and re-enabling of the timer and clearing the TMRy register are not required, but may be advantageous for defining a pulse from a known event time boundary.

The Output Compare module does not have to be disabled after the falling edge of the output pulse. Another pulse can be initiated by rewriting the value of the OCxCON register.

Examples 16-5 and 16-6 show example code for configuration of the single output pulse event.

Example 16-5: Single Output Pulse Setup and Interrupt Servicing (16-Bit Mode)

```
// The following code example will set the Output Compare 1 module
// for interrupts on the single pulse event and select Timer2
// as the clock source for the compare time base.

T2CON = 0x0010;                       // Configure Timer2 for a prescaler of 2

OC1CON = 0x0000;                      // Turn off OC1 while doing setup.
OC1CON = 0x0004;                      // Configure for single pulse mode
OC1R = 0x3000;                        // Initialize primary Compare Register
OC1RS = 0x3003;                       // Initialize secondary Compare Register
PR2 = 0x3003;                         // Set period (PR2 is now 32-bits wide)

                                      // configure int
IFS0CLR = 0x00000040;                 // Clear the OC1 interrupt flag
IFS0SET = 0x00000040;                 // Enable OC1 interrupt
IPC1SET = 0x001C0000;                 // Set OC1 interrupt priority to 7,
                                      // the highest level
IPC1SET = 0x00030000;                 // Set Subpriority to 3, maximum

T2CONSET = 0x8000;                    // Enable Timer2
OC1CONSET = 0x8000;                   // Enable OC1

// Example code for Output Compare 1 ISR:

void__ISR(_OUTPUT_COMPARE_1_VECTOR, ipl7) OC1_IntHandler (void)
{
// insert user code here
IFS0CLR = 0x0040;                     // Clear the OC1 interrupt flag
}
```

Example 16-6: Single Output Pulse Setup and Interrupt Servicing (32-Bit Mode)

```
// The following code example will set the Output Compare 1 module
// for interrupts on the single pulse event and select Timer2
// as the clock source for the compare time base.

T2CON = 0x0018;                     // Configure Timer2 for 32-bit operation
                                    // with a prescaler of 2. The Timer2/Timer3
                                    // pair is accessed via registers associated
                                    // with the Timer2 register

OC1CON = 0x0000;                    // Turn off OC1 while doing setup.
OC1CON = 0x0004;                    // Configure for single pulse mode
OC1R = 0x00203000;                  // Initialize primary Compare Register
OC1RS = 0x00203003;                 // Initialize secondary Compare Register
PR2 = 0x00500000;                   // Set period (PR2 is now 32-bits wide)

                                    // configure int
IFS0CLR = 0x00000040;               // Clear the OC1 interrupt flag
IFS0SET = 0x00000040;               // Enable OC1 interrupt
IPC1SET = 0x001C0000;               // Set OC1 interrupt priority to 7,
                                    // the highest level
IPC1SET = 0x00030000;               // Set Subpriority to 3, maximum

T2CONSET = 0x8000;                  // Enable Timer2
OC1CONSET = 0x8000;                 // Enable OC1

// Example code for Output Compare 1 ISR:

void__ISR(_OUTPUT_COMPARE_1_VECTOR, ipl7) OC1_IntHandler (void)
{
// insert user code here
IFS0CLR = 0x0040;                   // Clear the OC1 interrupt flag
}
```


16.3.2.3 Special Cases for Dual Compare Mode Generating a Single Output Pulse

Depending on the relationship of the OCxR, OCxRS and PRy values, the output compare module has a few unique conditions which should be understood. These special conditions are specified in Table 16-2, along with the resulting behavior of the module.

Table 16-2: Special Cases for Dual Compare Mode Generating a Single Output Pulse

SFR Logical Relationship	Special Conditions	Operation	Output at OCx
PRy >= OCxRS and OCxRS > OCxR	OCxR = 0 Initialize TMRy = 0	In the first iteration of the TMRy counting from 0x0000 up to PRy, the OCx pin remains low; no pulse is generated. After the TMRy resets to zero (on period match), the OCx pin goes high due to match with OCxR. Upon the next TMRy to OCxRS match, the OCx pin goes low and remains there. The OCxIF bit will be set as a result of the second compare. There are two alternative initial conditions to consider: a. Initialize TMRy = 0 and set OCxR >= 1 b. Initialize TMRy = PRy (PRy > 0) and set OCxR = 0	Pulse will be delayed by the value in the PRy register, depending on setup
PRy >= OCxR and OCxR >= OCxRS	OCxR >= 1 and PRy >= 1	TMRy counts up to OCxR and on a compare match event (i.e., TMRy = OCxR), the OCx pin is driven to a high state. TMRy then continues to count and eventually resets on period match (i.e., PRy =TMRy). The timer then restarts from 0x0000 and counts up to OCxRS. On a compare match event (i.e., TMRy = OCxRS), the OCx pin is driven to a low state. The OCxIF bit will be set as a result of the second compare.	Pulse
OCxRS > PRy and PRy >= OCxR	None	Only the rising edge will be generated at the OCx pin. The OCxIF will not be set.	Rising edge/ transition to high
OCxR > PRy	None	Unsupported mode; timer resets prior to match condition.	Remains low

Note 1: In all the cases considered herein, the TMRy register is assumed to be initialized to 0x0000.

2: OCxR = Compare Register, OCxRS = Secondary Compare Register, TMRy = Timery Count and PRy = Timery Period Register.

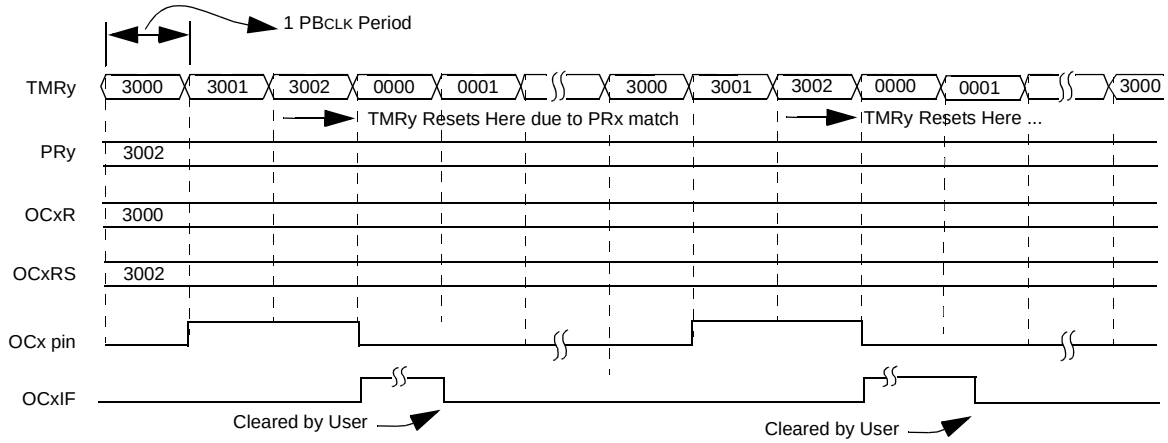
16.3.2.4 Dual Compare Mode: Continuous Output Pulses

To configure the output compare module for this mode, set control bits OCM<2:0> = '101'. In addition, the compare time base must be selected and enabled. Once this mode has been enabled, the output pin, OCx, will be driven low and remain low until a match occurs between the compare time base and OCxR register. Please note the following key timing events (refer to Figure 16-14 and Figure 16-16):

- The OCx pin is driven high one PBCLK after the compare match occurs between the compare time base and OCxR register. The OCx pin will remain high until the next match event occurs between the time base and the OCxRS register, at which time the pin will be driven low. This pulse generation sequence of a low-to-high and high-to-low edge will repeat on the OCx pin without further user intervention.
- Continuous pulses will be generated on the OCx pin until a mode change is made or the module is disabled.
- The compare time base will count up to the value contained in the associated period register and then reset to 0x0000 on the next instruction clock.
- If the compare time base period register value is less than the OCxRS register value, then no falling edge is generated. The OCx pin will remain high until OCxRS <= PRy, a mode change is made, or the device is reset.
- The respective channel interrupt flag, OCxIF, is asserted when the OCx pin is driven low (falling edge of single pulse).

General Dual Compare mode generating a continuous output pulse is illustrated in Figure 16-14. Figure 16-16 depicts another timing example where OCxRS > PRy. In this example, no falling edge of the pulse is generated, because the time base will reset before counting up to the contents of OCxRS.

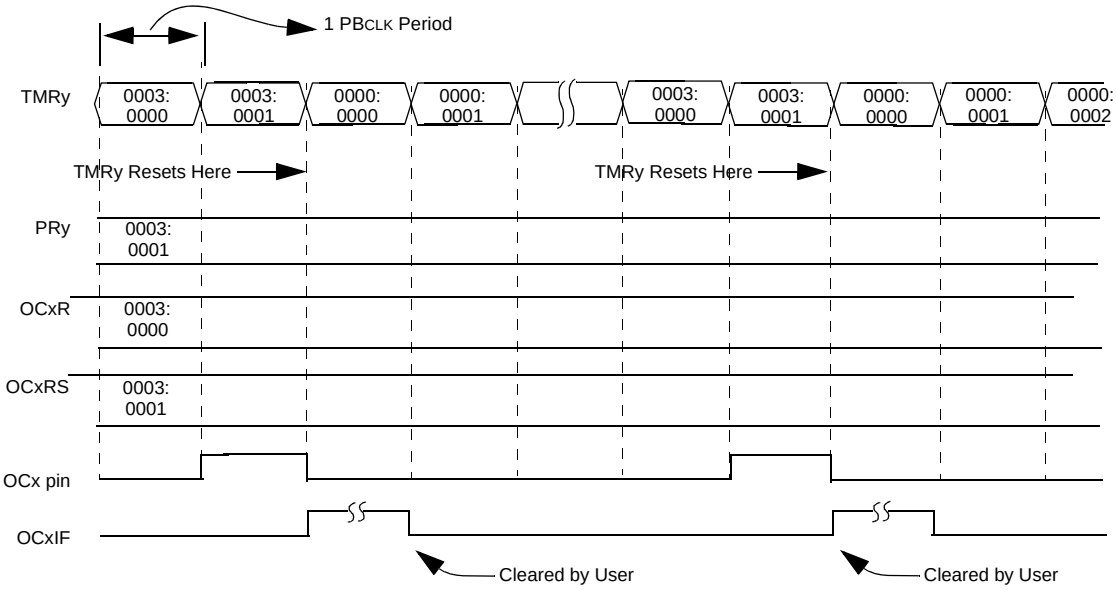
Figure 16-14: Dual Compare Mode: Continuous Output Pulse (PRy = OCxRS, 16-Bit Mode)



Note 1: An 'x' represents the output compare channel number. A 'y' represents the time base number.

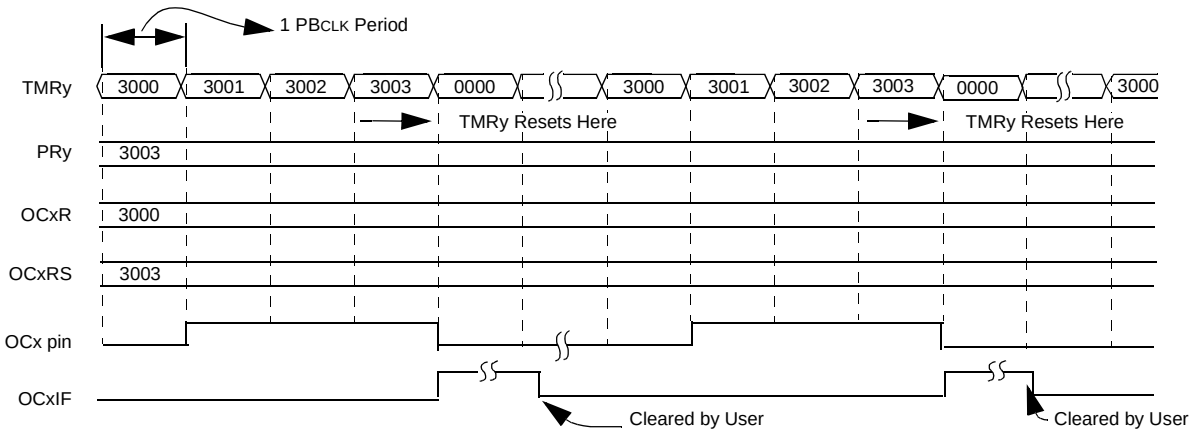
2: OCxR = Compare Register; OCxRS = Secondary Compare Register.

Figure 16-15: Dual Compare Mode: Continuous Output Pulse (PRy = OCxRS, 32-Bit Mode)



Note: An 'x' represents the output compare channel number. A 'y' represents the time base number.

Figure 16-16: Dual Compare Mode: Continuous Output Pulse (PRy = OCxRS, 16-Bit Mode)



Note 1: An 'x' represents the output compare channel number. A 'y' represents the time base number.

2: OCxR = Compare Register; OCxRS = Secondary Compare Register.

16.3.2.5 Setup for Continuous Output Pulse Generation

When control bits OCM<2:0> (OCxCON<2:0>) are set to '101', the selected output compare channel initializes the OCx pin to the low state and generates output pulses on each and every compare match event.

For the user to configure the module for the generation of a continuous stream of output pulses, the following steps are required (these steps assume the timer source is initially turned off, but this is not a requirement for the module operation):

1. Determine the peripheral clock cycle time. Take into account the frequency of the external clock to the timer source (if one is used) and the timer prescaler settings.
2. Calculate the time to the rising edge of the output pulse, relative to the TMRy start value (0x0000).
3. Calculate the time to the falling edge of the pulse, based on the desired pulse width and the time to the rising edge of the pulse.
4. Write the values computed in step 2 and 3 above into the compare register, OCxR, and the secondary compare register, OCxRS, respectively.
5. Set the timer period register, PRy, to a value equal to or greater than the value in OCxRS, the secondary compare register.
6. Set OCM<2:0> = '101' and the OCTSEL (OCxCON<3>) bit to the desired timer source (for 16-bit mode only). The OCx pin state will now be driven low.
7. Enable the compare time base by setting the TON (TyCON<15>) bit to '1'.
8. Upon the first match between TMRy and OCxR, the OCx pin will be driven high.
9. When the compare time base, TMRy, matches the secondary compare register, OCxRS, the second and trailing edge (high-to-low) of the pulse is driven onto the OCx pin.
10. As a result of the second compare match event, the OCxIF interrupt flag bit is set.
11. When the compare time base and the value in its respective period register match, the TMRy register resets to 0x0000 and resumes counting.
12. Steps 8 through 11 are repeated, and a continuous stream of pulses is generated, indefinitely. The OCxIF flag (refer to the IF0 register for the bit position of each channel's interrupt flag) is set on each OCxRS-TMRy compare match event.

Example 16-7 shows example code for configuration of the continuous output pulse event.

Example 16-7: Continuous Output Pulse Setup and Interrupt Servicing (16-Bit Mode)

```
// The following code example will set the Output Compare 1 module
// for interrupts on the continuous pulse event and select Timer2
// as the clock source for the compare time-base.

T2CON = 0x0010;                       // Configure Timer2 for a prescaler of 2

OC1CON = 0x0000;                      // disable OC1 module
OC1CON = 0x0005;                      // Configure OC1 module for Pulse output
OC1R = 0x3000;                        // Initialize Compare Register 1
OC1RS = 0x3003;                       // Initialize Secondary Compare Register 1
PR2 = 0x5000;                         // Set period


                                      // configure int
IFS0CLR = 0x00000040;                 // Clear the OC1 interrupt flag
IFS0SET = 0x00000040;                 // Enable OC1 interrupt
IPC1SET = 0x001C0000;                 // Set OC1 interrupt priority to 7,
                                      // the highest level
IPC1SET = 0x00030000;                 // Set Subpriority to 3, maximum

T2CONSET = 0x8000;                    // Enable Timer2
OC1CONSET = 0x8000;                   // Enable OC1

// Example code for Output Compare 1 ISR:

void__ISR(_OUTPUT_COMPARE_1_VECTOR, ipl7) OC1_IntHandler (void)
{
// insert user code here
IFS0CLR= 0x0040;                      // Clear the OC1 interrupt flag
}
```

Example 16-8: Continuous Output Pulse Setup and Interrupt Servicing (32-Bit Mode)

```
// The following code example will set the Output Compare 1 module
// for interrupts on the continuous pulse event and select Timer2
// as the clock source for the compare time-base.

T2CON = 0x0018;                      // Configure Timer2 for 32-bit operation
                                     // with a prescaler of 2. The Timer2/Timer3
                                     // pair is accessed via registers associated
                                     // with the Timer2 register

OC1CON = 0x0000;                     // disable OC1 module
OC1CON = 0x0005;                     // Configure OC1 module for Pulse output
OC1R = 0x3000;                       // Initialize Compare Register 1
OC1RS = 0x3003;                      // Initialize Secondary Compare Register 1
PR2 = 0x00500000;                    // Set period (PR2 is now 32-bits wide)


                                     // configure int
IFS0CLR = 0x00000040;                // Clear the OC1 interrupt flag
IFS0SET = 0x00000040;                // Enable OC1 interrupt
IPC1SET = 0x001C0000;                // Set OC1 interrupt priority to 7,
                                     // the highest level
IPC1SET = 0x00030000;                // Set Subpriority to 3, maximum

T2CONSET = 0x8000;                   // Enable Timer2
OC1CONSET = 0x8000;                  // Enable OC1

// Example code for Output Compare 1 ISR:

void__ISR(_OUTPUT_COMPARE_1_VECTOR, ipl7) OC1_IntHandler (void)
{
// insert user code here
IFS0CLR = 0x0040;                    // Clear the OC1 interrupt flag
}
```


16.3.2.6 Special Cases for Dual Compare Mode Generating Continuous Output Pulses

Depending on the relationship of the OCxR, OCxRS and PRy values, the output compare module may not provide the expected results. These special cases are specified in Table 16-3, along with the resulting behavior of the module.

Table 16-3: Special Cases for Dual Compare Mode Generating Continuous Output Pulses

SFR Logical Relationship	Special Conditions	Operation	Output at OCx
PRy >= OCxRS and OCxRS > OCxR	OCxR = 0 Initialize TMRy = 0	In the first iteration of the TMRy counting from 0x0000 up to PRy, the OCx pin remains low; no pulse is generated. After the TMRy resets to zero (on period match), the OCx pin goes high. Upon the next TMRy to OCxRS match, the OCx pin goes low. If OCxR = 0 and PRy = OCxRS, the pin will remain low for one clock cycle, then be driven high until the next TMRy to OCxRS match. The OCxIF bit will be set as a result of the second compare. There are two alternative initial conditions to consider: a. Initialize TMRy = 0 and set OCxR >= 1 b. Initialize TMRy = PRy (PRy > 0) and set OCxR = 0	Continuous pulses with the first pulse delayed by the value in the PRy register, depending on setup.
PRy >= OCxR and OCxR >= OCxRS	OCxR >= 1 and PRy >= 1	TMRy counts up to OCxR and on a compare match event (i.e., TMRy = OCxR), the OCx pin is driven to a high state. TMRy then continues to count and eventually resets on period match (i.e., PRy =TMRy). The timer then restarts from 0x0000 and counts up to OCxRS. On a compare match event (i.e., TMRy = OCxR), the OCx pin is driven to a low state. The OCxIF bit will be set as a result of the second compare.	Continuous pulses
OCxRS > PRy and PRy >= OCxR	None	Only one transition will be generated at the OCx pin until the OCxRS register contents have been changed to a value less than or equal to the period register contents (PRy). OCxIF is not set until then.	Rising edge/ transition to high
OCxR > PRy	None	Unsupported mode; Timer resets prior to match condition.	Remains low

Note 1: In all the cases considered herein, the TMRy register is assumed to be initialized to 0x0000.

2: OCxR = Compare Register, OCxRS = Secondary Compare Register, TMRy = Timery Count and PRy = Timery Period Register.

16.3.3 Pulse Width Modulation Mode

When control bits OCM<2:0> (OCxCON<2:0>) are set to '110' or '111', the selected output compare channel is configured for the PWM (Pulse-Width Modulation) mode of operation.

The following two PWM modes are available:

- PWM without Fault Protection Input
- PWM with Fault Protection Input

The OCFA or OCFB Fault input pin is utilized for the second PWM mode. In this mode, an asynchronous logic level '0' on the OCFx pin will cause the selected PWM channel to be shut down. (Refer to **16.3.3.1 "PWM with Fault Protection Input Pin"**.)

In PWM mode, the OCxR register is a read-only slave duty cycle register and OCxRS is a buffer register that is written by the user to update the PWM duty cycle. On every timer to period register match event (end of PWM period), the duty cycle register, OCxR, is loaded with the contents of OCxRS. The TyIF interrupt flag is asserted at each PWM period boundary.

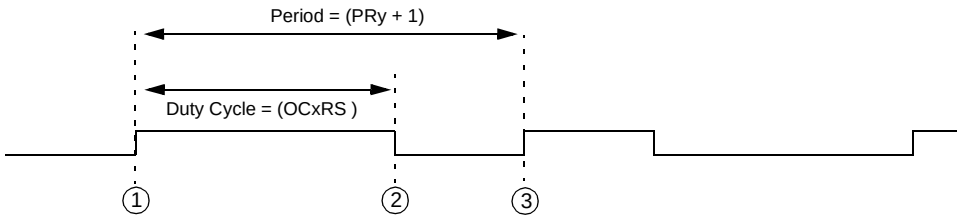
The following steps should be taken when configuring the output compare module for PWM operation:

1. Set the PWM period by writing to the selected timer period register (PRy).
2. Set the PWM duty cycle by writing to the OCxRS register.
3. Write the OxCR register with the initial duty cycle.
4. Enable interrupts, if required, for the timer and output compare modules. The output compare interrupt is required for PWM Fault pin utilization.
5. Configure the Output Compare module for one of two PWM Operation modes by writing to the Output Compare mode bits, OCM<2:0> (OCxCON<2:0>).
6. Set the TMRy prescale value and enable the time base by setting TON (TxCON<15>) = '1'.

Note:	The OCxR register should be initialized before the Output Compare module is first enabled. The OCxR register becomes a read-only duty cycle register when the module is operated in the PWM modes. The value held in OCxR will become the PWM duty cycle for the first PWM period. The contents of the duty cycle buffer register, OCxRS, will not be transferred into OCxR until a time base period match occurs.

An example PWM output waveform is shown in Figure 16-17.

Figure 16-17: PWM Output Waveform



1. Timery is cleared and the new duty cycle value is loaded from OCxRS into OCxR.
2. Timer value equals the value in the OCxR register; OCx Pin is driven low.
3. Timer overflow; value from OCxRS is loaded into OCxR; OCx pin is driven high. TyIF interrupt flag is asserted.

16.3.3.1 PWM with Fault Protection Input Pin

When the Output Compare mode bits, OCM<2:0> (OCxCON<2:0>), are set to '111', the selected output compare channel is configured for the PWM mode of operation. All functions described in **16.3.3 "Pulse Width Modulation Mode"** apply, with the addition of input Fault protection.

Fault protection is provided via the OCFA and OCFB pins. The OCFA pin is associated with the output compare channels 1 through 4, while the OCFB pin is associated with the output compare channel 5.

If a logic '0' is detected on the OCFA/OCFB pin, the selected PWM output pin(s) are placed in the high-impedance state. The user may elect to provide a pull-down or pull-up resistor on the PWM pin to provide for a desired state if a Fault condition occurs. The shutdown of the PWM output is immediate and is not tied to the device clock source. This state will remain until the following conditions are met:

- The external Fault condition has been removed
- The PWM mode is re-enabled by writing to the appropriate mode bits, OCM<2:0> (OCxCON<2:0>)

As a result of the Fault condition, the respective interrupt flag, OCxIF bit, is asserted and an interrupt will be generated, if enabled. Upon detection of the Fault condition, the OCFLT bit (OCxCON<4>) is asserted high (logic '1'). This bit is a read-only bit and will only be cleared once the external Fault condition has been removed and the PWM mode is re-enabled by writing to the appropriate mode bits, OCM<2:0> (OCxCON<2:0>).

Note:	The external Fault pins, if enabled for use, will continue to control the OCx output pins while the device is in SLEEP or IDLE mode.

16.3.3.2 PWM Period

The PWM period is specified by writing to PRy, the Timery period register. The PWM period can be calculated using the following formula:

Equation 16-1: Calculating the PWM Period

$$\text{PWM Period} = [(PR + 1) \bullet T_{PB} \bullet (\text{TMR Prescale Value})]$$

$$\text{PWM Frequency} = 1/[\text{PWM Period}]$$

The PWM period must not exceed the width of the Period Register for the selected mode, 16 bits for 16-bit mode or 32 bits for 32-bit mode. If the calculated period is too large, select a larger prescaler to prevent overflow. To maintain maximum PWM resolution, select the smallest prescaler that does not result in an overflow.

Note:	A PRy value of N will produce a PWM period of N + 1 time base count cycles. For example, a value of 7 written into the PRy register will yield a period consisting of 8 time base cycles.

16.3.3.3 PWM Duty Cycle

The PWM duty cycle is specified by writing to the OCxRS register. The OCxRS register can be written to at any time, but the duty cycle value is not latched into OCxR until a match between PRy and TMRy occurs (i.e., the period is complete). This provides a double buffer for the PWM duty cycle and is essential for glitchless PWM operation. In the PWM mode, OCxR is a read-only register.

Some important boundary parameters of the PWM duty cycle include the following:

- If the duty cycle register OCxR is loaded with 0x0000, the OCx pin will remain low (0% duty cycle).
- If OCxR is greater than PRy (timer period register), the pin will remain high (100% duty cycle).
- If OCxR is equal to PRy, the OCx pin will be low for one time base count value and high for all other count values.

See Figure 16-18 for PWM mode timing details. Table 16-4 through Table 16-9 show example PWM frequencies and resolutions for a device with the Peripheral Bus operating at a variety of frequencies.

Equation 16-2: Calculation for Maximum PWM Resolution

$$\text{Maximum PWM Resolution (bits)} = \frac{\log_{10}\left(\frac{F_{PB}}{F_{PWM} \bullet TMRy \bullet \text{Prescaler bits}}\right)}{\log_{10}(2)}$$

Equation 16-3: PWM Period and Duty Cycle Calculation

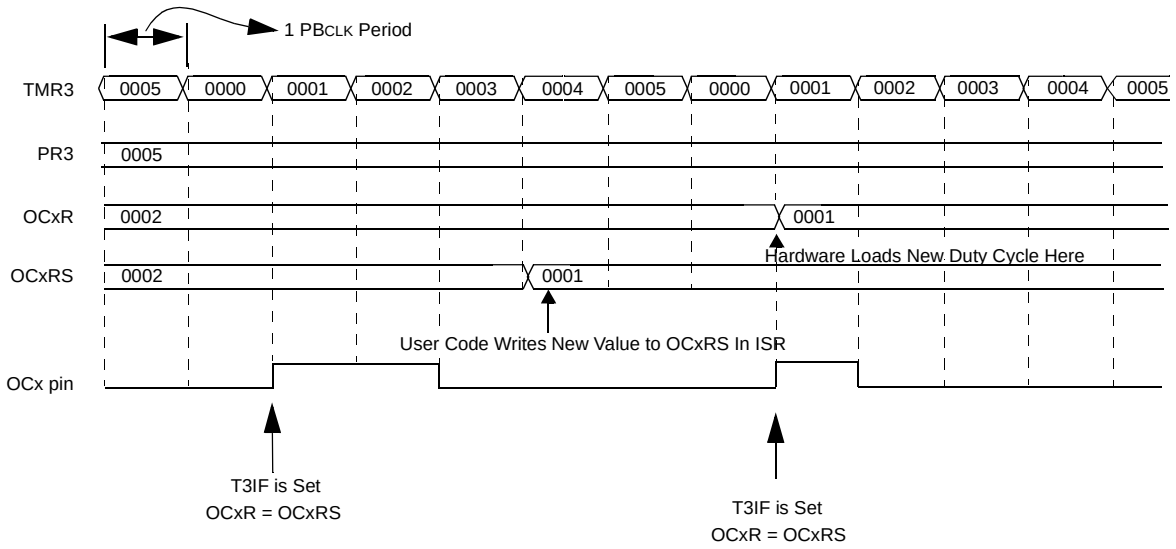
Desired PWM frequency is 52.08 kHz
F_{PB} = 10 MHz
Timer 2 prescale setting: 1:1

$1/52.08\ \text{kHz} = (PR2 + 1) \bullet T_{PB} \bullet (\text{Timer 2 prescale value})$
$19.20\ \mu s = (PR2 + 1) \bullet 0.1\ \mu s \bullet (1)$
$PR2 = 191$

Find the maximum resolution of the duty cycle that can be used with a 52.08 kHz PWM frequency and a 10 MHz Peripheral Bus clock rate.

$1/52.08\ \text{kHz} = 2^{\text{PWM RESOLUTION}} \bullet 1/10\ \text{MHz} \bullet 1$
$19.20\ \mu s = 2^{\text{PWM RESOLUTION}} \bullet 100\ \text{ns} \bullet 1$
$192 = 2^{\text{PWM RESOLUTION}}$
$\log_{10}(192) = (\text{PWM Resolution}) \bullet \log_{10}(2)$
PWM Resolution = 7.6 bits

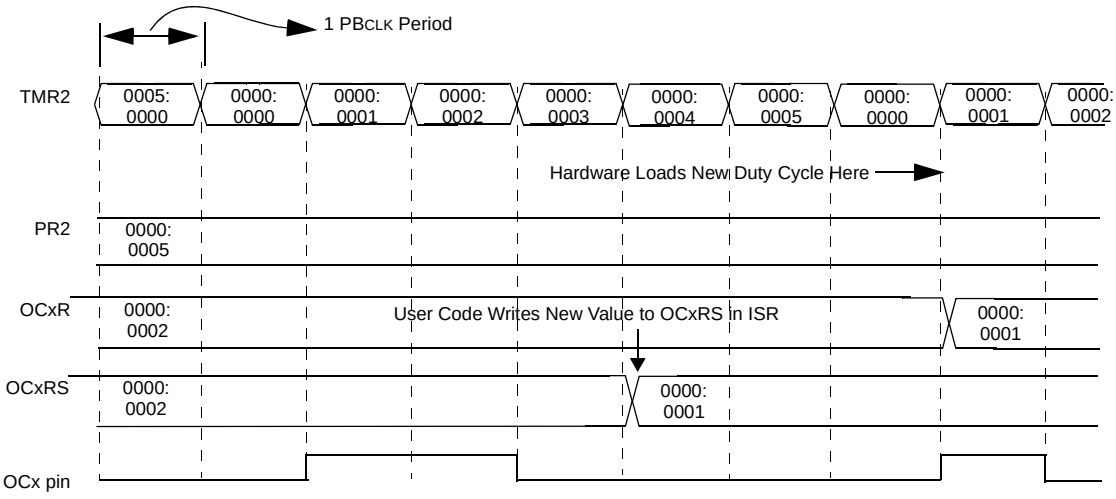
Figure 16-18: PWM Output Timing



Note 1: An 'x' represents the output compare channel number.

2: OCxR = Compare Register; OCxRS = Secondary Compare Register.

Figure 16-19: Dual Compare Mode: Continuous Output Pulse (PR2 = OCxRS, 32-Bit Mode)



Note: An 'x' represents the output compare channel number..

Table 16-4: Example PWM Frequencies and Resolutions with a 10 MHz (16-Bit Mode) Peripheral Bus Clock

PWM Frequency	19 Hz	153 Hz	305 Hz	2.44 kHz	9.77 kHz	78.1 kHz	313 kHz
Timer Prescaler Ratio	8	1	1	1	1	1	1
Period Register Value	0xFFFF	0xFFFF	0x7FFF	0x0FFF	0x03FF	0x007F	0x001F
Resolution (bits)	16	16	15	12	10	7	5

Table 16-5: Example PWM Frequencies and Resolutions with a 30 MHz (16-Bit Mode) Peripheral Bus Clock

PWM Frequency	58 Hz	458 Hz	916 Hz	7.32 kHz	29.3 kHz	234 kHz	938 kHz
Timer Prescaler Ratio	8	1	1	1	1	1	1
Period Register Value	0xFC8E	0xFFDD	0x7FEE	0x1001	0x03FE	0x007F	0x001E
Resolution (bits)	16	16	15	12	10	7	5

Table 16-6: Example PWM Frequencies and Resolutions with a 50 MHz (16-Bit Mode) Peripheral Bus Clock

PWM Frequency	57 Hz	458 Hz	916 Hz	7.32 kHz	29.3 kHz	234 kHz	938 kHz
Timer Prescaler Ratio	64	8	1	1	1	1	1
Period Register Value	0x349C	0x354D	0xD538	0x1AAD	0x06A9	0x00D4	0x0034
Resolution (bits)	13.7	13.7	15.7	12.7	10.7	7.7	5.7

Table 16-7: Example PWM Frequencies and Resolutions with a 50 MHz (16-Bit Mode) Peripheral Bus Clock

PWM Frequency	100 Hz	200 Hz	500 Hz	1 kHz	2 kHz	5 kHz	10 kHz
Timer Prescaler Ratio	8	8	8	1	8	1	1
Period Register Value (hex)	0xF423	0x7A11	0x30D3	0xC34F	0x0C34	0x270F	0x1387
Resolution (bits) (decimal)	15.9	14.9	13.6	15.6	11.6	13.3	12.3

Table 16-8: Example PWM Frequencies and Resolutions with a 50 MHz (16-Bit Mode) Peripheral Bus Clock

PWM Frequency	100 Hz	200 Hz	500 Hz	1 kHz	2 kHz	5 kHz	10 kHz
Timer Prescaler Ratio	8	4	2	1	1	1	1
Period Register Value (hex)	0xF423	0xF423	0xC34F	0x0C34F	0x61A7	0x270F	0x1387
Resolution (bits) (decimal)	15.9	15.9	15.6	15.6	14.6	13.3	12.3

Table 16-9: Example PWM Frequencies and Resolutions with a 50 MHz (32-Bit Mode) Peripheral Bus Clock

PWM Frequency	100 Hz	200 Hz	500 Hz	1 kHz	2 kHz	5 kHz	10 kHz
Timer Prescaler Ratio	1	1	1	1	1	8	1
Period Register Value (hex)	0x0007A11F	0x0003D08F	0x0001869F	0x0000C34F	0x000061A7	0x000004E1	0x00001387
Resolution (bits) (decimal)	18.9	17.9	16.6	15.6	14.6	10.3	12.3

Example 16-9 shows configuration and interrupt service code for the PWM mode of operation.

Example 16-9: PWM Mode Setup and Interrupt Servicing (16-Bit Mode)

```
// The following code example will set the Output Compare 1 module
// for PWM mode with FAULT pin disabled, a 50% duty cycle and a
// PWM frequency of 52.08 kHz at Fosc = 40 MHz. Timer2 is selected as
// the clock for the PWM time base and Timer2 interrupts
// are enabled.

OC1CON = 0x0000;                    // Turn off OC1 while doing setup.
OC1R = 0x0060;                      // Initialize primary Compare Register
OC1RS = 0x0060;                     // Initialize secondary Compare Register
OC1CON = 0x0006;                    // Configure for PWM mode
PR2 = 0x00BF;                       // Set period


                                    // configure int
IFS0CLR = 0x00000040;               // Clear the OC1 interrupt flag
IFS0SET = 0x00000040;               // Enable OC1 interrupt
IPC1SET = 0x001C0000;               // Set OC1 interrupt priority to 7,
                                    // the highest level
IPC1SET = 0x00030000;               // Set Subpriority to 3, maximum

T2CONSET = 0x8000;                  // Enable Timer2
OC1CONSET = 0x8000;                 // Enable OC1

// Example code for Output Compare 1 ISR:

void__ISR(_OUTPUT_COMPARE_1_VECTOR, ipl7) OC1_IntHandler (void)
{
// insert user code here
IFS0CLR = 0x0040;                   // Clear the OC1 interrupt flag
}
```

Example 16-10: PWM Mode Setup and Interrupt Servicing (32-Bit Mode)

```
// The following code example will set the Output Compare 1 module
// for PWM mode with FAULT pin disabled, a 50% duty cycle and a
// PWM frequency of 52.08 kHz at Fosc = 40 MHz. Timer2 is selected as
// the clock for the PWM time base and Timer2 interrupts
// are enabled.

OC1CON = 0x0000;                    // Turn off OC1 while doing setup.
OC1R = 0x00600000;                  // Initialize primary Compare Register
OC1RS = 0x00600000;                 // Initialize secondary Compare Register
OC1CON = 0x0006;                    // Configure for single pulse mode
PR2 = 0x00600000;                   // Set period

                                    // configure int
IFS0CLR = 0x00000040;               // Clear the OC1 interrupt flag
IFS0SET = 0x00000040;               // Enable OC1 interrupt
IPC1SET = 0x001C0000;               // Set OC1 interrupt priority to 7,
                                    // the highest level
IPC1SET = 0x00030000;               // Set Subpriority to 3, maximum

T2CONSET = 0x8000;                  // Enable Timer2
OC1CONSET = 0x8000;                 // Enable OC1

// Example code for Output Compare 1 ISR:

void__ISR(_OUTPUT_COMPARE_1_VECTOR, ipl7) OC1_IntHandler (void)
{
// insert user code here
IFS0CLR = 0x0040;                   // Clear the OC1 interrupt flag
}
```


16.4 INTERRUPTS

Each of the available output compare channels has a dedicated interrupt bit OCxIF, and a corresponding interrupt enable/mask bit OCxIE. These bits are used to determine the source of an interrupt and to enable or disable an individual interrupt source. The priority level of each of the channels can also be set independently of the other channels.

OCxIF is set when an output compare channel detects a predefined match condition that is defined as an event generating an interrupt. The OCxIF bit will then be set without regard to the state of the corresponding OCxIE bit. The OCxIF bit can be polled by software if desired.

The OCxIE bit is used to define the behavior of the Vector Interrupt Controller (VIC) when a corresponding OCxIF is set. When the OCxIE bit is clear, the VIC module does not generate a CPU interrupt for the event. If the OCxIE bit is set, the VIC module will generate an interrupt to the CPU when the corresponding OCxIF bit is set (subject to the priority and subpriority as outlined below).

It is the responsibility of the routine that services a particular interrupt to clear the appropriate interrupt flag bit before the service routine is complete.

The priority of each output compare channel can be set independently via the OCxIP<2:0> bits. This priority defines the priority group that the interrupt source will be assigned to. The priority groups range from a value of 7, the highest priority, to a value of 0, which does not generate an interrupt. An interrupt being serviced will be preempted by an interrupt in a higher priority group.

The subpriority bits allow setting the priority of an interrupt source within a priority group. The values of the subpriority, OCxIS<1:0>, range from 3, the highest priority, to 0, the lowest priority. An interrupt with the same priority group but having a higher subpriority value will preempt a lower subpriority interrupt that is in progress.

The priority group and subpriority bits allow more than one interrupt source to share the same priority and subpriority. If simultaneous interrupts occur in this configuration, the natural order of the interrupt sources within a priority/subpriority group pair determines the interrupt generated. The natural priority is based on the vector numbers of the interrupt sources. The lower the vector number, the higher the natural priority of the interrupt. any interrupts that were overridden by natural order will then generate their respective interrupts (based on priority, subpriority, and natural order) after the interrupt flag for the current interrupt is cleared.

After an enabled interrupt is generated, the CPU will jump to the vector assigned to that interrupt. The vector number for the interrupt is the same as the natural order number. The CPU will then begin executing code at the vector address. The user's code at this vector address should perform any operations required (such as reloading the duty cycle and clearing the interrupt flag), and then exit. Refer to **Section 8. "Interrupts"** for the vector address table details and for more information on interrupts.

16.5 I/O PIN CONTROL

When the output compare module is enabled, the I/O pin direction is controlled by the compare module. The compare module returns the I/O pin control back to the appropriate pin LAT and TRIS control bits when it is disabled.

When the PWM with Fault Protection Input mode is enabled, the OCFx Fault pin must be configured for an input by setting the respective TRIS SFR bit. The OCFx Fault input pin is not automatically configured as an input when the PWM fault mode is selected.

Table 16-10: Pins Associated with Output Compare Modules 1-5

Pin Name	Module Control	Pin Type	Buffer Type	Description
OC1	ON	O	—	Output Compare/PWM Channel 1
OC2	ON	O	—	Output Compare/PWM Channel 2
OC3	ON	O	—	Output Compare/PWM Channel 3
OC4	ON	O	—	Output Compare/PWM Channel 4
OC5	ON	O	—	Output Compare/PWM Channel 5
OCFA	ON	I	ST	PWM Fault Protection A Input (for Channels 1-4)
OCFB	ON	I	ST	PWM Fault Protection B Input (for Channel 5)

Legend: ST = Schmitt Trigger input with CMOS levels
I = Input
O = Output

16.6 OPERATION IN POWER-SAVING AND DEBUG MODES

Note: In this manual, a distinction is made between a power mode as it is used in a specific module, and a power mode as it is used by the device, e.g., Sleep mode of the Comparator and SLEEP mode of the CPU. To indicate which type of power mode is intended, uppercase and lowercase letters (Sleep, Idle, Debug) signify a module power mode, and all uppercase letters (SLEEP, IDLE, DEBUG) signify a device power mode.

16.6.1 Output Compare Operation in SLEEP Mode

When the device enters SLEEP mode, the system clock is disabled. During SLEEP, the Output Compare modules will drive the pin to the same active state as driven prior to entering SLEEP. The module will then halt at this state.

For example, if the pin was high and the CPU entered the SLEEP state, the pin will stay high. Likewise, if the pin was low and the CPU entered the SLEEP state, the pin will stay low. In both cases, when the device wakes up, the Output Compare module will resume operation.

When the module is operating in PWM Fault mode, the asynchronous portions of the fault circuit remain active. If a fault is detected, the compare output enable signal is deasserted and OCFLT (OCxCON<4>) is set. If the corresponding interrupt is enabled, an interrupt will be generated and the device will wake-up from SLEEP.

16.6.2 Output Compare Operation in IDLE Mode

When the device enters IDLE mode, the system clock sources remain functional and the CPU stops executing code. The SIDL bit (OCxCON<13>) selects if the compare module will stop operation when the device enters IDLE mode or whether the module will continue normal operation in IDLE mode.

- If SIDL = 1, the module will discontinue operation in IDLE mode. The module will perform the same procedures when stopped in IDLE mode as it does for SLEEP mode.
- If SIDL = 0, the module will continue operation in IDLE only if the selected time base is set to operate in IDLE mode. The output compare channel(s) will operate during the CPU IDLE mode if the SIDL bit is a logic '0'. Furthermore, the time base must be enabled with the respective SIDL bit set to a logic '0'.

Note: The external Fault pins, if enabled for use, will continue to control the associated OCx output pins while the device is in SLEEP or IDLE mode.

- When the module is operating in PWM Fault mode, the asynchronous portions of the fault circuit remain active. If a fault is detected, the compare output enable signal is deasserted and OCFLT (OCxCON<4>) is set. If the corresponding interrupt is enabled, an interrupt will be generated and the device will wake-up from IDLE.

16.6.3 Output Compare Operation in DEBUG Mode

The FRZ bit (OCxCON<14>) determines whether the Output Compare module will run or stop while the CPU is executing Debug Exception code (i.e., the application is halted) in DEBUG mode. When FRZ = '0', the Output Compare module continues to run even when the application is halted in DEBUG mode. When FRZ = 1 and the application is halted in DEBUG mode, the module will freeze its operations and make no changes to the state of the Output Compare module. The module will resume its operation after the CPU resumes execution.

When the module is operating in PWM Fault mode, the asynchronous portions of the fault circuit remain active. If a fault is detected, the compare output enable signal is deasserted and OCFLT (OCxCON<4>) is set. If the corresponding interrupt is enabled, an interrupt will be generated.

Note: The FRZ bit is readable and writable only when the CPU is executing in Debug Exception mode. In all other modes, the FRZ bit reads as '0'. If FRZ bit is changed during DEBUG mode, the new value does not take effect until the current Debug Exception mode is exited and re-entered. During the Debug Exception mode, the FRZ bit reads the state of the peripheral when entering DEBUG mode.

16.7 EFFECTS OF VARIOUS RESETS

16.7.1 $\overline{\text{MCLR}}$ Reset

Following a $\overline{\text{MCLR}}$ event, the OCxCON, OCxR, and OCxRS registers for each Output Compare module are reset to a value of 0x00000000.

16.7.2 Power-on Reset

Following a Power-on (POR) event, the OCxCON, OCxR, and OCxRS registers for each Output Compare module are reset to a value of 0x00000000.

16.7.3 Watchdog Timer Reset

The status of the OCMP control registers after a Watchdog Timer (WDT) event depends on the operational mode of the CPU prior to the WDT event.

If the device is not in SLEEP, a WDT event will force the OCxCON, OCxR, and OCxRS registers to a Reset value of 0x00000000.

If the device is in SLEEP when a WDT event occurs, the contents of the OCxCON, OCxR and OCxRS register values are not affected.

16.8 OUTPUT COMPARE APPLICATION

This is an example application using the PWM mode of the Output Compare module to control the speed of a DC motor. The speed of the motor is controlled by changing the PWM duty cycle.

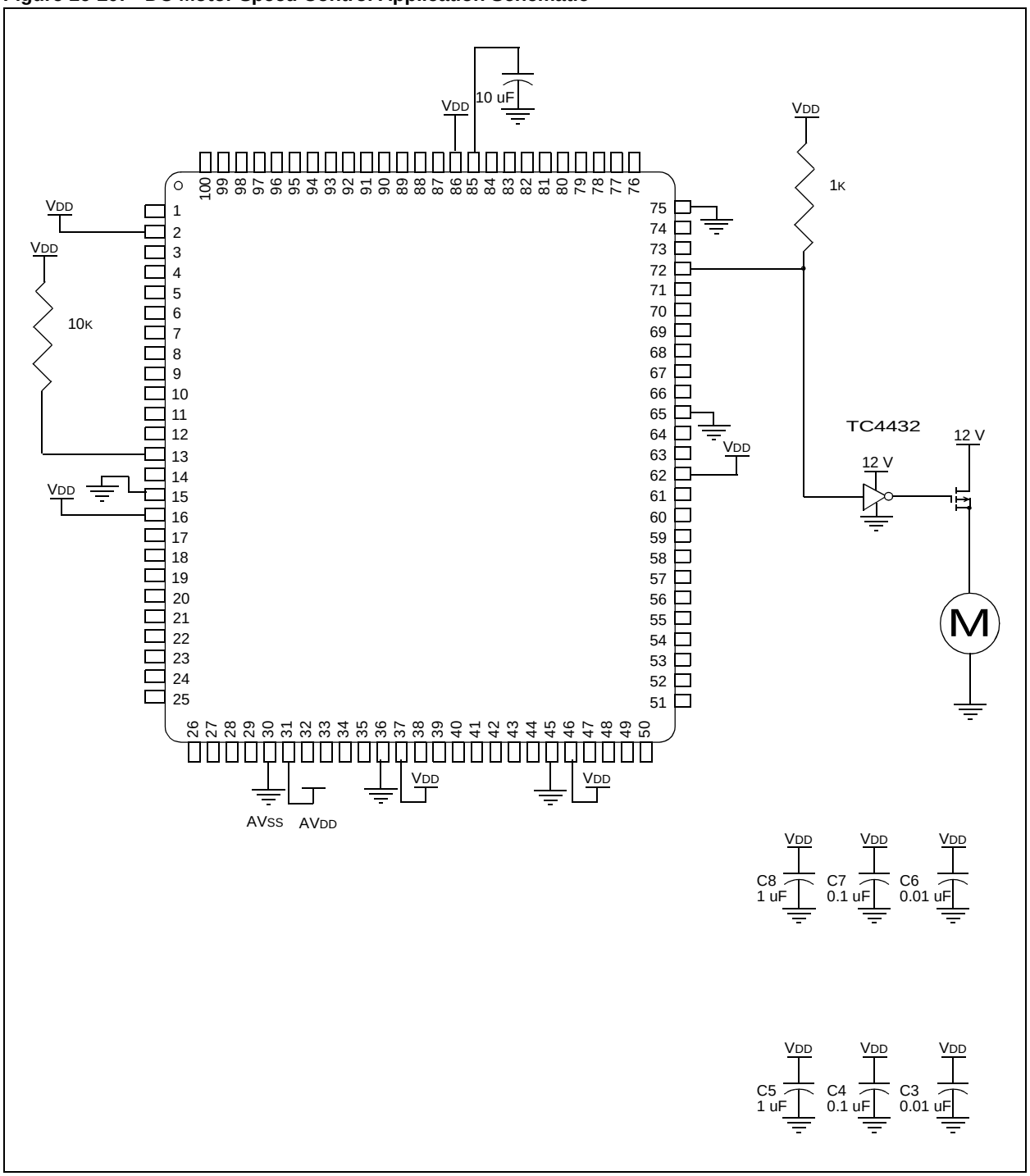
The circuit consists of the following:

- A PIC32MX device to generate the PWM.
- A TC4431 or equivalent MOSFET driver to drive the MOSFET.
- A MOSFET to drive the motor.
- A pull-up resistor is used to pull the input of the MOSFET driver high when the PIC32MX is in Reset. This prevents unwanted motor operation during start-up.
- A DC motor.

Example 16-11: PWM Mode Example Application (16-Bit Mode)

```
// The following code example will set the Output Compare 1 module
// for PWM mode w/o FAULT pin enabled, a 50% duty cycle and a
// PWM frequency of 52.08 kHz at FP = 40 MHz. Timer2 is selected as
// the clock for the PWM time base and Timer2 interrupts
// are enabled. This example ramps the PWM duty cycle from min to max, then
// from max to min and repeats. The rate at which the PWM duty cycle is
// changed can be adjusted by the rate at which the Timer2 overflows.
// The PWM period can be changed by writing a different value to the PR2
// register. If the PR2 value is adjusted the maximum PWM value will also
// have to be adjusted so that it is not greater than the PR2 value.

unsigned int Pwm;                       // variable to store calculated PWM value
unsigned char Mode = 0;                 // variable to determine ramp up or ramp down

OC1CON = 0x0000;                        // Turn off OC1 while doing setup.
OC1R = 0x0000;                          // Initialize primary Compare Register
OC1RS = 0x0000;                         // Initialize secondary Compare Register
OC1CON = 0x0006;                        // Configure for PWM mode
PR2 = 0xFFFF;                           // Set period

                                        // configure int
IFS0CLR = 0x00000040;                   // Clear the OC1 interrupt flag
IFS0SET = 0x00000040;                   // Enable OC1 interrupt
IPC1SET = 0x001C0000;                   // Set OC1 interrupt priority to 7,
                                        // the highest level
IPC1SET = 0x00030000;                   // Set Subpriority to 3, maximum

T2CONSET = 0x8000;                      // Enable Timer2
OC1CONSET = 0x8000;                     // Enable OC1

// Example code for Output Compare 1 ISR:

void__ISR(_OUTPUT_COMPARE_1_VECTOR, ipl7) OC1_IntHandler (void)
{
if ( Mode )
    {
        if ( Pwm < 0xFFFF )             // ramp up mode
        {
            Pwm ++;                     // If the duty cycle is not at max, increase
            OC1RS = Pwm;                // Write new duty cycle
        }
        else
        {
            Mode = 0;                   // PWM is at max, change mode to ramp down
        }
    }                                   // end of ramp up
    else
    {
        if ( !Pwm )                     // ramp down mode
        {
            Pwm --;                     // If the duty cycle is not at min, increase
            OC1RS = Pwm;                // Write new duty cycle
        }
        else
        {
            Mode = 1;                   // PWM is at min, change mode to ramp up
        }
    }                                   // end of ramp down

        // insert user code here
        IFS0CLR = 0x0040;               // Clear the OC1 interrupt flag
}
```

Figure 16-20: DC Motor Speed Control Application Schematic



16.9 DESIGN TIPS

Question 1: ***The Output Compare pin stops functioning even when the SIDL bit is not set. Why?***

Answer: This is most likely to occur when the SIDL bit (TxCON<13>) of the associated timer source is set. Therefore, it is the timer that actually goes into IDLE mode when the PWRSAV instruction is executed.

Question 2: ***Can I use the Output Compare modules with the selected time base configured for 32-bit mode?***

Answer: Yes. The timer can be used in 32-bit mode as a time base for the Output Compare modules by setting the T32 bit (TxCON<3>). For proper operation, the Output Compare module must be configured for 32-bit Compare mode by setting the OC32 bit (OCxCON<5>) for all Output Compare modules using the 32-bit timer as a time base.

16.10 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the PIC32MX device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the Output Compare module are:

Title	Application Note #
No related application notes at this time.	N/A

Note: Please visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the PIC32MX family of devices.

16.11 REVISION HISTORY

Revision A (October 2007)

This is the initial released version of this document.

Revision B (October 2007)

Updated document to remove Confidential status.

Revision C (April 2008)

Revised status to Preliminary; Revised U-0 to r-x.

Revision D (June 2008)

Revised Registers 16-1, 16-20, 16-32; Revised Examples 16-3, 16-4, 16-5, 16-6, 16-7, 16-8, 16-9, 16-10, 16-11; Added TMR1 and TMR2 to Summary Table; Revised Section 16.3, Notes; Change Reserved bits from “Maintain as” to “Write”; Added Note to ON bit (0CxCON, T2CON, T3CON Registers).

NOTES:

Section 17. 10-Bit A/D Converter

HIGHLIGHTS

This section of the manual contains the following topics:

17.1 INTRODUCTION

The PIC32MX 10-bit Analog-to-Digital (A/D) converter (or ADC) includes the following features:

- Successive Approximation Register (SAR) conversion
- Up to 16 analog input pins
- External voltage reference input pins
- One unipolar differential Sample-and-Hold Amplifier (SHA)
- Automatic Channel Scan mode
- Selectable conversion trigger source
- 16 word conversion result buffer
- Selectable Buffer Fill modes
- Eight conversion result format options
- Operation during CPU SLEEP and IDLE modes

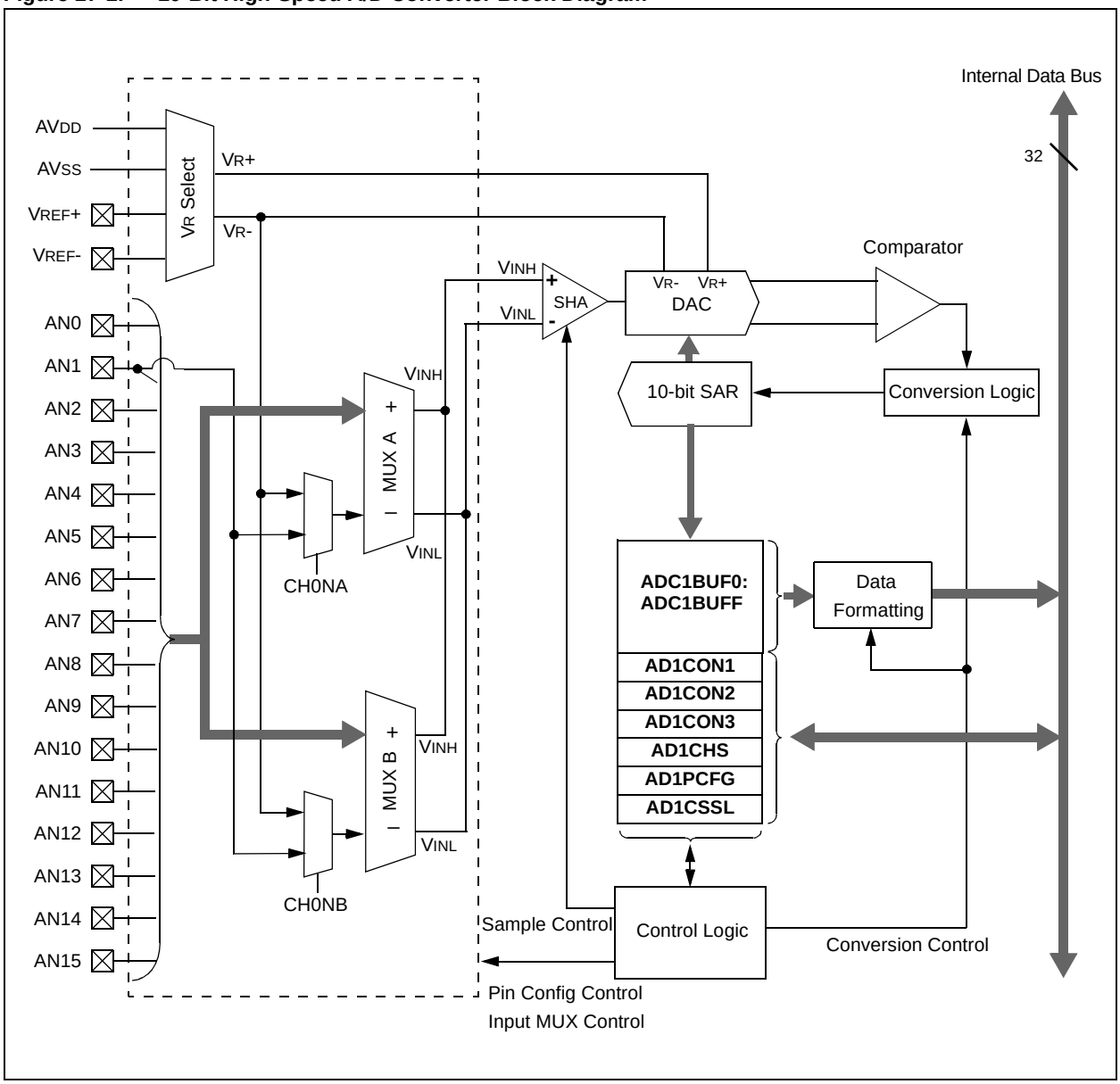
A block diagram of the 10-bit ADC is shown in Figure 17-1. The 10-bit ADC can have up to 16 analog input pins, designated AN0-AN15. In addition, there are two analog input pins for external voltage reference connections. These voltage reference inputs may be shared with other analog input pins and may be common to other analog module references. The actual number of analog input pins and external voltage reference input configuration will depend on the specific PIC32MX device. Refer to the device data sheet for further details.

The analog inputs are connected through two multiplexers (MUXs) to one SHA. The analog input MUXs can be switched between two sets of analog inputs between conversions. Unipolar differential conversions are possible on all channels, other than the pin used as the reference, using a reference input pin (see Figure 17-1).

The Analog Input Scan mode sequentially converts user-specified channels. A Control register specifies which analog input channels will be included in the scanning sequence.

The 10-bit ADC is connected to a 16-word result buffer. Each 10-bit result is converted to one of eight 32-bit output formats when it is read from the result buffer.

Figure 17-1: 10-Bit High-Speed A/D Converter Block Diagram



17.2 CONTROL REGISTERS

The ADC module includes the following Special Function Registers (SFRs):

The AD1CON1, AD1CON2 and AD1CON3 registers control the operation of the ADC module.

- AD1CON1: ADC Control Register 1
  AD1CON1CLR, AD1CON1SET, AD1CON1INV: Atomic Bit Manipulation, Write-only Registers for AD1CON1.
- AD1CON2: ADC Control Register 2
  AD1CON2CLR, AD1CON2SET, AD1CON2INV: Atomic Bit Manipulation, Write-only Registers for AD1CON2.
- AD1CON3: ADC Control Register 3
  AD1CON3CLR, AD1CON3SET, AD1CON3INV: Atomic Bit Manipulation, Write-only Registers for AD1CON3.

The AD1CHS register selects the input pins to be connected to the SHA.

- AD1CHS: ADC Input Channel Select Register
  AD1CHSCLR, AD1CHSSET, AD1CHSINV: Atomic Bit Manipulation, Write-only Registers for AD1CHS.

The AD1PCFG register configures the analog input pins as analog inputs or as digital I/O.

- AD1PCFG: ADC Port Configuration Register
  AD1PCFGCLR, AD1PCFGSET, AD1PCFGINV: Atomic Bit Manipulation, Write-only Registers for AD1PCFG.

The AD1CSSL register selects inputs to be sequentially scanned.

- AD1CSSL: ADC Input Scan Selection Register
  AD1CSSLCLR, AD1CSSLSET, AD1CSSLINV: Atomic Bit Manipulation, Write-only Registers for AD1CSSL.

The ADC module also has the following associated bits for interrupt control:

- Interrupt Request Flag Status bit (AD1IF) in IFS1: Interrupt Flag Status Register 1
- Interrupt Enable Control bit (AD1IE) in IEC1: Interrupt Enable Control Register 1
- Interrupt Priority Control bits (AD1IP<2:0>) and (AD1IS<1:0>) in IPC6: Interrupt Priority Control Register 6

17.2.1 Special Function Registers Associated with the 10-Bit ADC

The following table provides a summary of all ADC-related registers, including their addresses and formats. Corresponding registers appear after the summary, followed by a detailed description of each register. All unimplemented registers and/or bits within a register read as zeros.

Table 17-1: ADC SFR Summary

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
AD1CON1	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	ON	FRZ	SIDL	—	—	FORM2	FORM1	FORM0
	7:0	SSRC2	SSRC1	SSRC0	CLRASAM	—	ASAM	SAMP	DONE
AD1CON1CLR	31:0	Write clears selected bits in AD1CON1, read yields undefined value							
AD1CON1SET	31:0	Write sets selected bits in AD1CON1, read yields undefined value							
AD1CON1INV	31:0	Write inverts selected bits in AD1CON1, read yields undefined value							
AD1CON2	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	VCFG2	VCFG1	VCFG0	OFFCAL	—	CSCNA	—	—
	7:0	BUFS	—	SMPI3	SMPI2	SMPI1	SMPI0	BUFM	ALTS
AD1CON2CLR	31:0	Write clears selected bits in AD1CON2, read yields undefined value							
AD1CON2SET	31:0	Write sets selected bits in AD1CON2, read yields undefined value							
AD1CON2INV	31:0	Write inverts selected bits in AD1CON2, read yields undefined value							
AD1CON3	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	ADRC	—	—	SAMC4	SAMC3	SAMC2	SAMC1	SAMC0
	7:0	ADCS7	ADCS6	ADCS5	ADCS4	ADCS3	ADCS2	ADCS1	ADCS0
AD1CON3CLR	31:0	Write clears selected bits in AD1CON3, read yields undefined value							
AD1CON3SET	31:0	Write sets selected bits in AD1CON3, read yields undefined value							
AD1CON3INV	31:0	Write inverts selected bits in AD1CON3, read yields undefined value							
AD1CHS	31:24	CH0NB	—	—	—	CH0SB3	CH0SB2	CH0SB1	CH0SB0
	23:16	CH0NA	—	—	—	CH0SA3	CH0SA2	CH0SA1	CH0SA0
	15:8	—	—	—	—	—	—	—	—
	7:0	—	—	—	—	—	—	—	—
AD1CHSCLR	31:0	Write clears selected bits in AD1CHS, read yields undefined value							
AD1CHSSET	31:0	Write sets selected bits in AD1CHS, read yields undefined value							
AD1CHS1INV	31:0	Write inverts selected bits in AD1CHS, read yields undefined value							
AD1PCFG	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	PCFG15	PCFG14	PCFG13	PCFG12	PCFG11	PCFG10	PCFG9	PCFG8
	7:0	PCFG7	PCFG6	PCFG5	PCFG4	PCFG3	PCFG2	PCFG1	PCFG0
AD1PCFGCLR	31:0	Write clears selected bits in AD1PCFG, read yields undefined value							
AD1PCFGSET	31:0	Write sets selected bits in AD1PCFG, read yields undefined value							
AD1PCFGINV	31:0	Write inverts selected bits in AD1PCFG, read yields undefined value							
AD1CSSL	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	CSSL15	CSSL14	CSSL13	CSSL12	CSSL11	CSSL10	CSSL9	CSSL8
	7:0	CSSL7	CSSL6	CSSL5	CSSL4	CSSL3	CSSL2	CSSL1	CSSL0
AD1CSSLCLR	31:0	Write clears selected bits in AD1CSSL, read yields undefined value							

Table 17-1: ADC SFR Summary (Continued)

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
AD1CSSLSET	31:0	Write sets selected bits in AD1CSSL, read yields undefined value							
AD1CSSLINV	31:0	Write inverts selected bits in AD1CSSL, read yields undefined value							
ADC1BUF0	31:0	ADC Result Word 0 (ADC1BUF0<31:0>)							
ADC1BUF1	31:0	ADC Result Word 1 (ADC1BUF1<31:0>)							
ADC1BUF2	31:0	ADC Result Word 2 (ADC1BUF2<31:0>)							
ADC1BUF3	31:0	ADC Result Word 3 (ADC1BUF3<31:0>)							
ADC1BUF4	31:0	ADC Result Word 4 (ADC1BUF4<31:0>)							
ADC1BUF5	31:0	ADC Result Word 5 (ADC1BUF5<31:0>)							
ADC1BUF6	31:0	ADC Result Word 6 (ADC1BUF6<31:0>)							
ADC1BUF7	31:0	ADC Result Word 7 (ADC1BUF7<31:0>)							
ADC1BUF8	31:0	ADC Result Word 8 (ADC1BUF8<31:0>)							
ADC1BUF9	31:0	ADC Result Word 9 (ADC1BUF9<31:0>)							
ADC1BUFA	31:0	ADC Result Word A (ADC1BUFA<31:0>)							
ADC1BUFB	31:0	ADC Result Word B (ADC1BUFB<31:0>)							
ADC1BUFC	31:0	ADC Result Word C (ADC1BUFC<31:0>)							
ADC1BUFD	31:0	ADC Result Word D (ADC1BUFD<31:0>)							
ADC1BUFE	31:0	ADC Result Word E (ADC1BUFE<31:0>)							
ADC1BUFF	31:0	ADC Result Word F (ADC1BUFF<31:0>)							
IFS1	31:24	—	—	—	—	—	—	USBIF	FCEIF
	23:16	—	—	—	—	DMA3IF	DMA2IF	DMA1IF	DMA0IF
	15:8	RTCCIF	FSCMIF	I2C2MIF	I2C2SIF	I2C2BIF	U2TXIF	U2RXIF	U2EIF
	7:0	SPI2RXIF	SPI2TXIF	SPI2EIF	CMP2IF	CMP1IF	PMPIF	AD1IF	CNIF
IFS1CLR	31:0	Write clears the selected bits in IFS1, read yields undefined value							
IFS1SET	31:0	Write sets the selected bits in IFS1, read yields undefined value							
IFS1INV	31:0	Write inverts the selected bits in IFS1, read yields undefined value							
IEC1	31:24	—	—	—	—	—	—	USBIE	FCEIE
	23:16	—	—	—	—	DMA3IE	DMA2IE	DMA1IE	DMA0IE
	15:8	RTCCIE	FSCMIE	I2C2MIE	I2C2SIE	I2C2BIE	U2TXIE	U2RXIE	U2EIE
	7:0	SPI2RXIE	SPI2TXIE	SPI2EIE	CMP2IE	CMP1IE	PMPIE	AD1IE	CNIE
IPC6	31:24	—	—	—	AD1IP<2:0>			AD1IS<1:0>	
	23:16	—	—	—	CNIP<2:0>			CNIS<1:0>	
	15:8	—	—	—	I2C1IP<2:0>			I2C1IS<1:0>	
	7:0	—	—	—	U1IP<2:0>			U1IS<1:0>	
IPC6CLR	31:0	Write clears the selected bits in IPC6, read yields undefined value							
IPC6SET	31:0	Write sets the selected bits in IPC6, read yields undefined value							
IPC6INV	31:0	Write inverts the selected bits in IPC6, read yields undefined value							

Register 17-1: AD1CON1: ADC Control Register 1

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	r-x	r-x	R/W-0	R/W-0	R/W-0
ON	FRZ	SIDL	—	—	FORM<2:0>		
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	r-x	R/W-0	R/W-0	R/C-0
SSRC<2:0>			CLRASAM	—	ASAM	SAMP	DONE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15 **ON:** ADC Operating Mode bit
1 = A/D converter module is operating
0 = A/D converter is off
Note: When using 1:1 PBCLK divisor, the user’s software should not read/write the peripheral’s SFRs in the SYSCLK cycle immediately following the instruction that clears the module’s ON bit.

bit 14 **FRZ:** Freeze in Debug Exception Mode bit
1 = Freeze operation when CPU enters Debug Exception mode
0 = Continue operation when CPU enters Debug Exception mode
Note: FRZ is writable in Debug Exception mode only. It reads ‘0’ in Normal mode.

bit 13 **SIDL:** Stop in IDLE Mode bit
1 = Discontinue module operation when device enters IDLE mode
0 = Continue module operation in IDLE mode

bit 12-11 **Reserved:** Write ‘0’; ignore read

bit 10-8 **FORM<2:0>:** Data Output Format bits
011 = Signed Fractional 16-bit (DOUT = 0000 0000 0000 0000 sddd dddd dd00 0000)
010 = Fractional 16-bit (DOUT = 0000 0000 0000 0000 dddd dddd dd00 0000)
001 = Signed Integer 16-bit (DOUT = 0000 0000 0000 0000 ssss sssd dddd dddd)
000 = Integer 16-bit (DOUT = 0000 0000 0000 0000 0000 00dd dddd dddd)
111 = Signed Fractional 32-bit (DOUT = sddd dddd dd00 0000 0000 0000 0000)
110 = Fractional 32-bit (DOUT = dddd dddd dd00 0000 0000 0000 0000 0000)
101 = Signed Integer 32-bit (DOUT = ssss ssss ssss ssss ssss sssd dddd dddd)
100 = Integer 32-bit (DOUT = 0000 0000 0000 0000 0000 00dd dddd dddd)

Register 17-1: AD1CON1: ADC Control Register 1 (Continued)

bit 7-5 **SSRC<2:0>:** Conversion Trigger Source Select bits

111 = Internal counter ends sampling and starts conversion (auto convert)
110 = Reserved
101 = Reserved
100 = Reserved
011 = Reserved
010 = Timer 3 period match ends sampling and starts conversion
001 = Active transition on INT0 pin ends sampling and starts conversion
000 = Clearing SAMP bit ends sampling and starts conversion

bit 4 **CLRASAM:** Stop Conversion Sequence bit (when the first A/D converter interrupt is generated)

1 = Stop conversions when the first ADC interrupt is generated. Hardware clears the ASAM bit when the ADC interrupt is generated.
0 = Normal operation, buffer contents will be overwritten by the next conversion sequence

bit 3 **Reserved:** Write '0'; ignore read

bit 2 **ASAM:** ADC Sample Auto-Start bit

1 = Sampling begins immediately after last conversion completes; SAMP bit is automatically set.
0 = Sampling begins when SAMP bit is set

bit 1 **SAMP:** ADC Sample Enable bit

1 = The ADC SHA is sampling
0 = The ADC sample/hold amplifier is holding
When ASAM = 0, writing '1' to this bit starts sampling.
When SSRC = 000, writing '0' to this bit will end sampling and start conversion.

bit 0 **DONE:** A/D Conversion Status bit

1 = A/D conversion is done
0 = A/D conversion is not done or has not started
Clearing this bit will not affect any operation in progress.

Note: The DONE bit isn't persistent in automatic modes. It is cleared by hardware at the beginning of the next sample.

Register 17-2: AD1CON1CLR: ADC Port Configuration Register

Write clears selected bits in AD1CON1, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in AD1CON1**

A write of '1' in one or more bit positions clears the corresponding bit(s) in AD1CON1 register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `AD1CON1CLR = 0x00008002` will clear bits 15 and 1 in AD1CON1 register.

Register 17-3: AD1CON1SET:ADC Port Configuration Register

Write sets selected bits in AD1CON1, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in AD1CON1**

A write of '1' in one or more bit positions sets the corresponding bit(s) in AD1CON1 register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `AD1CON1SET = 0x00008002` will set bits 15 and 1in AD1CON1 register.

Register 17-4: AD1CON1INV:ADC Port Configuration Register

Write inverts selected bits in AD1CON1, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in AD1CON1**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in AD1CON1 register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `AD1CON1INV = 0x00008002` will invert bits 15 and 1 in AD1CON1 register.

Register 17-5: AD1CON2: ADC Control Register 2

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	r-x	R/W-0	r-x	r-x
VCFG<2:0>			OFFCAL	—	CSCNA	—	—
bit 15							bit 8

R-0	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
BUFS	—	SMPI<3:0>				BUFM	ALTS
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15-13 **VCFG<2:0>:** Voltage Reference Configuration bits

	ADC VR+	**ADC VR-**
000	AVDD	AVSS
001	External VREF+ pin	AVSS
010	AVDD	External VREF- pin
011	External VREF+ pin	External VREF- pin
1xx	AVDD	AVSS

bit 12 **OFFCAL:** Input Offset Calibration Mode Select bit

1 = Enable Offset Calibration mode
VINH and VINL of the SHA are connected to **VR-**
0 = Disable Offset Calibration mode
The inputs to the SHA are controlled by AD1CHS or AD1CSSL

bit 11 **Reserved:** Write ‘0’; ignore read

bit 10 **CSCNA:** Scan Input Selections for CH0+ SHA Input for MUX A Input Multiplexer Setting bit
1 = Scan inputs
0 = Do not scan inputs

bit 9-8 **Reserved:** Write ‘0’; ignore read’

bit 7 **BUFS:** Buffer Fill Status bit

Only valid when BUFM = 1 (ADRES split into 2 x 8-word buffers).
1 = ADC is currently filling buffer 0x8-0xF, user should access data in 0x0-0x7
0 = ADC is currently filling buffer 0x0-0x7, user should access data in 0x8-0xF

bit 6 **Reserved:** Write ‘0’; ignore read

Register 17-5: AD1CON2: ADC Control Register 2 (Continued)

bit 5-2 **SMPI<3:0>:** Sample/Convert Sequences Per Interrupt Selection bits

1111 = Interrupts at the completion of conversion for each 16th sample/convert sequence
1110 = Interrupts at the completion of conversion for each 15th sample/convert sequence
.....
0001 = Interrupts at the completion of conversion for each 2nd sample/convert sequence
0000 = Interrupts at the completion of conversion for each sample/convert sequence

bit 1 **BUFM:** ADC Result Buffer Mode Select bit

1 = Buffer configured as two 8-word buffers, ADC1BUF(7...0), ADC1BUF(15...8)
0 = Buffer configured as one 16-word buffer ADC1BUF(15...0.)

bit 0 **ALTS:** Alternate Input Sample Mode Select bit

1 = Uses MUX A input multiplexer settings for first sample, then alternates between MUX B and MUX A input multiplexer settings for all subsequent samples
0 = Always use MUX A input multiplexer settings

Register 17-6: AD1CON2CLR: ADC Port Configuration Register

Write clears selected bits in AD1CON2, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in AD1CON2**

A write of '1' in one or more bit positions clears the corresponding bit(s) in AD1CON2 register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `AD1CON2CLR = 0x00008001` will clear bits 15 and 0 in AD1CON2 register.

Register 17-7: AD1CON2SET:ADC Port Configuration Register

Write sets selected bits in AD1CON2, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in AD1CON2**

A write of '1' in one or more bit positions sets the corresponding bit(s) in AD1CON2 register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `AD1CON2SET = 0x00008001` will set bits 15 and 0 in AD1CON2 register.

Register 17-8: AD1CON2INV:ADC Port Configuration Register

Write inverts selected bits in AD1CON2, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in AD1CON2**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in AD1CON2 register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `AD1CON2INV = 0x00008001` will invert bits 15 and 0 in AD1CON2 register.

Register 17-9: AD1CON3: ADC Control Register 3

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADRC	—	—	SAMC<4:0>				
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W	R/W-0
ADCS<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15 **ADRC:** ADC Conversion Clock Source bit
1 = ADC internal RC clock
0 = Clock derived from Peripheral Bus Clock (PBclock)

bit 14-13 **Reserved:** Write ‘0’; ignore read

bit 12-8 **SAMC<4:0>:** Auto-Sample Time bits
11111 = 31 TAD
.....
00001 = 1 TAD
00000 = 0 TAD (Not allowed)

bit 7-0 **ADCS<7:0>:** ADC Conversion Clock Select bits
11111111 = TPB • 2 • (ADCS<7:0> + 1) = 512 • TPB = TAD
.....
00000001 = TPB • 2 • (ADCS<7:0> + 1) = 4 • TPB = TAD
00000000 = TPB • 2 • (ADCS<7:0> + 1) = 2 • TPB = TAD

Register 17-10: AD1CON3CLR: ADC Port Configuration Register

Write clears selected bits in AD1CON3, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in AD1CON3**

A write of '1' in one or more bit positions clears the corresponding bit(s) in AD1CON3 register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `AD1CON3CLR = 0x00008001` will clear bits 15 and 0 in AD1CON3 register.

Register 17-11: AD1CON3SET:ADC Port Configuration Register

Write sets selected bits in AD1CON3, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in AD1CON3**

A write of '1' in one or more bit positions sets the corresponding bit(s) in AD1CON3 register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `AD1CON3SET = 0x00008001` will set bits 15 and 0 in AD1CON3 register.

Register 17-12: AD1CON3INV:ADC Port Configuration Register

Write inverts selected bits in AD1CON3, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in AD1CON3**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in AD1CON3 register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `AD1CON3INV = 0x00008001` will invert bits 15 and 0 in AD1CON3 register.

Register 17-13: AD1CHS:ADC Input Select Register

R/W-0	r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0
CH0NB	—	—	—	CH0SB<3:0>			
bit 31							bit 24

R/W-0	r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0
CH0NA	—	—	—	CH0SA<3:0>			
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31 **CH0NB:** Negative Input Select bit for MUX B
1 = Channel 0 negative input is AN1
0 = Channel 0 negative input is VR-

bit 30-28 **Reserved:** Write ‘0’; ignore read

bit 27-24 **CH0SB<3:0>:** Positive Input Select bits for MUX B
1111 = Channel 0 positive input is AN15
1110 = Channel 0 positive input is AN14
1101 = Channel 0 positive input is AN13
...
...
...
0001 = Channel 0 positive input is AN1
0000 = Channel 0 positive input is AN0

bit 23 **CH0NA:** Negative Input Select bit for MUX A Multiplexer Setting(2)
1 = Channel 0 negative input is AN1
0 = Channel 0 negative input is VR-

bit 22-20 **Reserved:** Write ‘0’; ignore read

bit 19-16 **CH0SA<3:0>:** Positive Input Select bits for MUX A Multiplexer Setting
1111 = Channel 0 positive input is AN15
1110 = Channel 0 positive input is AN14
1101 = Channel 0 positive input is AN13
...
...
...
0001 = Channel 0 positive input is AN1
0000 = Channel 0 positive input is AN0

bit 15-0 **Reserved:** Write ‘0’; ignore read

Register 17-14: AD1CHSCLR: ADC Port Configuration Register

Write clears selected bits in AD1CHS, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in AD1CHS**

A write of '1' in one or more bit positions clears the corresponding bit(s) in AD1CHS register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `AD1CHSCLR = 0x80010000` will clear bits 15 and 0 in AD1CHS register.

Register 17-15: AD1CHSSET:ADC Port Configuration Register

Write sets selected bits in AD1CHS, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in AD1CHS**

A write of '1' in one or more bit positions sets the corresponding bit(s) in AD1CHS register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `AD1CHSSET = 0x80010000` will set bits 15 and 0 in AD1CHS register.

Register 17-16: AD1CHSINV:ADC Port Configuration Register

Write inverts selected bits in AD1CHS, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in AD1CHS**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in AD1CHS register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `AD1CHSINV = 0x80010000` will invert bits 15 and 0 in AD1CHS register.

Register 17-17: AD1PCFG:ADC Port Configuration Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PCFG15	PCFG14	PCFG13	PCFG12	PCFG11	PCFG10	PCFG9	PCFG8
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PCFG7	PCFG6	PCFG5	PCFG4	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15-0 **PCFG<15:0>:** Analog Input Pin Configuration Control bits

1 = Analog input pin in Digital mode, port read input enabled, ADC input multiplexer input for this analog input connected to AVss

0 = Analog input pin in Analog mode, digital port read will return as a ‘1’ without regard to the voltage on the pin, ADC samples pin voltage

Note: The AD1PCFG register functionality will vary depending on the number of ADC inputs available on the selected device. Please refer to the specific device data sheet for additional details on this register.

Register 17-18: AD1PCFGCLR: ADC Port Configuration Register

Write clears selected bits in AD1PCFG, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in AD1PCFG**

A write of '1' in one or more bit positions clears the corresponding bit(s) in AD1PCFG register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `AD1PCFGCLR = 0x00008001` will clear bits 15 and 0 in AD1PCFG register.

Register 17-19: AD1PCFGSET:ADC Port Configuration Register

Write sets selected bits in AD1PCFG, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in AD1PCFG**

A write of '1' in one or more bit positions sets the corresponding bit(s) in AD1PCFG register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `AD1PCFGSET = 0x00008001` will set bits 15 and 0 in AD1PCFG register.

Register 17-20: AD1PCFGINV:ADC Port Configuration Register

Write inverts selected bits in AD1PCFG, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in AD1PCFG**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in AD1PCFG register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `AD1PCFGINV = 0x00008001` will invert bits 15 and 0 in AD1PCFG register.

Register 17-21: AD1CSSL: ADC Input Scan Select Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CSSL15	CSSL14	CSSL13	CSSL12	CSSL11	CSSL10	CSSL9	CSSL8
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CSSL7	CSSL6	CSSL5	CSSL4	CSSL3	CSSL2	CSSL1	CSSL0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15-0 **CSSL<15:0>:** ADC Input Pin Scan Selection bits

1 = Select ANx for input scan

0 = Skip ANx for input scan

Note: The AD1CSSL register functionality will vary depending on the number of ADC inputs available on the selected device. Please refer to the specific device data sheet for additional details on this register.

Register 17-22: AD1CSSLCLR: ADC Port Configuration Register

Write clears selected bits in AD1CSSL, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in AD1CSSL**

A write of '1' in one or more bit positions clears the corresponding bit(s) in AD1CSSL register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `AD1CSSLCLR = 0x00008001` will clear bits 15 and 0 in AD1CSSL register.

Register 17-23: AD1CSSLSET:ADC Port Configuration Register

Write sets selected bits in AD1CSSL, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in AD1CSSL**

A write of '1' in one or more bit positions sets the corresponding bit(s) in AD1CSSL register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `AD1CSSLSET = 0x00008001` will set bits 15 and 0 in AD1CSSL register.

Register 17-24: AD1CSSLINV:ADC Port Configuration Register

Write inverts selected bits in AD1CSSL, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in AD1CSSL**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in AD1CSSL register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `AD1CSSLINV = 0x00008001` will invert bits 15 and 0 in AD1CSSL register.

Register 17-25: IFS1: Interrupt Flag Status Register 1(1)

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	USBIF	FCEIF
bit 31							bit 24

r-x	r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	DMA3IF	DMA2IF	DMA1IF	DMA0IF
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RTCCIF	FSCMIF	I2C2MIF	I2C2SIF	I2C2BIF	U2TXIF	U2RXIF	U2EIF
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SPI2RXIF	SPI2TXIF	SPI2EIF	CMP2IF	CMP1IF	PMPIF	AD1IF	CNIF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 1 **AD1IF:** Analog-to-Digital Converter 1 Interrupt Request Flag bit

1 = Interrupt request has occurred
0 = No interrupt request has a occurred

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the ADC.

Register 17-26: IEC1: Interrupt Enable Control Register 1(1)

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	USBIE	FCEIE
bit 31							bit 24

r-x	r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	DMA3IE	DMA2IE	DMA1IE	DMA0IE
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RTCCIE	FSCMIE	I2C2MIE	I2C2SIE	I2C2BIE	U2TXIE	U2RXIE	U2EIE
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SPI2RXIE	SPI2TXIE	SPI2EIE	CMP2IE	CMP1IE	PMPIE	AD1IE	I2C1MIE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 1 **AD1IE:** Analog-to-Digital Converter 1 Interrupt Enable bit

1 = Interrupt is enabled.
0 = Interrupt is disabled.

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the ADC.

Register 17-27: IPC6:Interrupt Priority Control Register 6(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	AD1IP<2:0>			AD1IS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	CNIP<2:0>			CNIS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	I2C1IP<2:0>			I2C1IS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W	R/W-0
—	—	—	U1IP<2:0>			U1IS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 28 - 26 **AD1IP<2:0>:** Analog-to-Digital Converter 1 Interrupt Priority bits
- 111 = Interrupt Priority is 7
- 110 = Interrupt Priority is 6
- 101 = Interrupt Priority is 5
- 100 = Interrupt Priority is 4
- 011 = Interrupt Priority is 3
- 010 = Interrupt Priority is 2
- 001 = Interrupt Priority is 1
- 000 = Interrupt is disabled

bit 25-24 **AD1IS<1:0>:** Analog-to-Digital 1 Subpriority bits
- 11 = Interrupt Subpriority is 3
- 10 = Interrupt Subpriority is 2
- 01 = Interrupt Subpriority is 1
- 00 = Interrupt Subpriority is 0

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the ADC.

17.3 ADC OPERATION, TERMINOLOGY AND CONVERSION SEQUENCE

This section will describe the operation the A/D converter, the steps required to configure the converter, describe special feature of the module, and provide examples of ADC configuration with timing diagrams and charts showing the expected output of the converter.

17.3.1 Overview of Operation

Analog sampling consists of two steps: acquisition and conversion (see Figure 17-2). During acquisition the analog input pin is connected to the Sample and Hold Amplifier (SHA). After the pin has been sampled for a sufficient period, the sample voltage is equivalent to the input, the pin is disconnected from the SHA to provide a stable input voltage for the conversion process. The conversion process then converts the analog sample voltage to a binary representation.

An overview of the ADC is presented in Figure 17-1. The 10-bit A/D converter has a single SHA. The SHA is connected to the analog input pins via the analog input MUXs, MUX A and MUX B. The analog input MUXs are controlled by the AD1CHS register. There are two sets of MUX control bits in the AD1CHS register. These two sets of control bits allow the two different analog input to be independently controlled. The A/D converter can optionally switch between MUX A and MUX B configurations between conversions. The A/D converter can also optionally scan through a series of analog inputs using a single MUX.

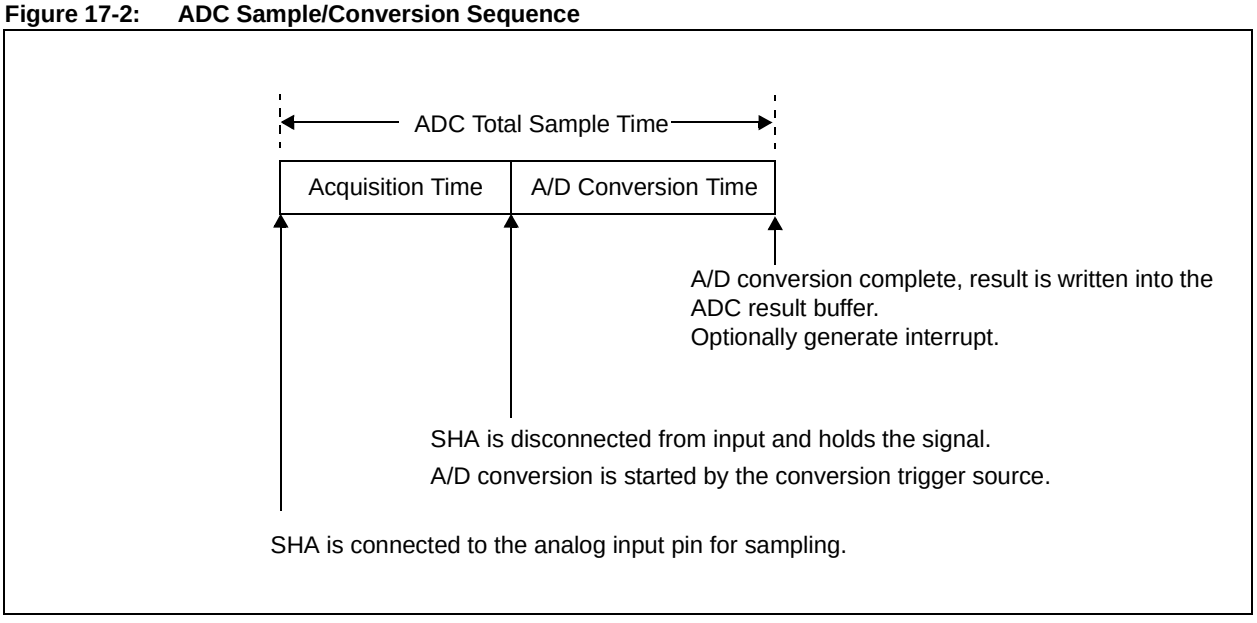
Acquisition time can be controlled manually or automatically. The acquisition time may be started manually by setting the SAMP bit (AD1CON1<1>), and ended manually by clearing the SAMP in the user software. The acquisition time may be started automatically by the A/D converter hardware and ended automatically by a conversion trigger source. The acquisition time is set by the SAMC bits (AD1CON3<12:8>). The SHA has a minimum acquisition period. Refer to the device data sheet for acquisition time specifications

Conversion time is the time required for the A/D converter to convert the voltage held by the SHA. The A/D converter requires one ADC clock cycle (TAD) to convert each bit of the result, plus two additional clock cycles. Therefore a total of 12 TAD cycles are required to perform the complete conversion. When the conversion time is complete, the result is written into one of the 16 ADC result registers (ADC1BUF0...ADC1BUFF).

The sum of the acquisition time and the A/D conversion time provides the total sample time (refer to Figure 17-2). There are multiple input clock options for the A/D converter that are used to create the TAD clock. The user must select an input clock option that does not violate the minimum TAD specification.

The sampling process can be performed once, periodically, or based on a trigger as defined by the module configuration.

Figure 17-2: ADC Sample/Conversion Sequence



The start time for sampling can be controlled in software by setting the SAMP control bit. The start of the sampling time can also be controlled automatically by the hardware. When the A/D converter operates in the Auto-Sample mode, the SHA is reconnected to the analog input pin at the end of the conversion in the sample/convert sequence. The auto-sample function is controlled by the ASAM control bit (AD1CON1<2>).

The conversion trigger source ends the sampling time and begins an A/D conversion or a sample/convert sequence. The conversion trigger source is selected by the control bits SSRC<2:0> (AD1CON1<7:5>). The conversion trigger can be taken from a variety of hardware sources, or can be controlled manually in software by clearing the SAMP control bit. One of the conversion trigger sources is an auto-conversion. The time between auto-conversions is set by a counter and the ADC clock. The Auto-Sample mode and auto-conversion trigger can be used together to provide endless automatic conversions without software intervention.

An interrupt may be generated at the end of each sample sequence or multiple sample sequences as determined by the value of the SMPI<3:0> (AD1CON2<5:2>). The number of sample sequences between interrupts can vary between 1 and 16. The user should note that the A/D conversion buffer holds the results of a single conversion sequence. The next sequence starts filling the buffer from the top even if the number of samples in the previous sequence was less than 16. The total number of conversion results between interrupts is the SMPI value. The total number of conversions between interrupts cannot exceed the physical buffer length.

17.4 ADC MODULE CONFIGURATION

Operation of the ADC module is directed through bit settings in the appropriate registers. The following instructions summarize the actions and the settings. Options and details for each configuration step are provided in subsequent sections.

1. To configure the ADC module, perform the following steps:

   A-1. Configure analog port pins in AD1PCFG<15:0>, as described in **Section 17.4.1 "Configuring Analog Port Pins"**.

   B-1. Select the analog inputs to the ADC MUXs in AD1CHS<32:0>, as described in **Section 17.4.2 "Selecting the Analog Inputs to the ADC MUXs"**.

   C-1. Select the format of the ADC result using FORM<2:0> (AD1CON1<10:8>), as described in **Section 17.4.3 "Selecting the Format of the ADC Result"**.
   C-2. Select the sample clock source using SSRC<2:0> (AD1CON1<7:5>), as described in **Section 17.4.4 "Selecting the Sample Clock Source"**.

   D-1. Select the voltage reference source using VCFG<2:0> (AD1CON2<15:13>), as described in **Section 17.4.7 "Selecting the Voltage Reference Source"**.
   D-2. Select the Scan mode using CSCNA (AD1CON2<10>), as described in **Section 17.4.8 "Selecting the Scan Mode"**.
   D-3. Set the number of conversions per interrupt SMP<3:0> (AD1CON2<5:2>), if interrupts are to be used, as described in **Section 17.4.9 "Setting the Number of Conversions per Interrupt"**.
   D-4. Set Buffer Fill mode using BUFM (AD1CON2<1>), as described in **Section 17.4.10 "Buffer Fill Mode"**.
   D-5. Select the MUX to be connected to the ADC in ALTS AD1CON2<0>, as described in **Section 17.4.11 "Selecting the MUX to be Connected to the ADC (Alternating Sample Mode)"**.

   E-1. Select the ADC clock source using ADRC (AD1CON3<15>), as described in **Section 17.4.12 "Selecting the ADC Conversion Clock Source and Prescaler"**.
   E-2. Select the sample time using SAMC<4:0> (AD1CON3<12:8>), if auto-convert is to be used, as described in **Section 17.4.13 "Acquisition Time Considerations"**.
   E-3. Select the ADC clock prescaler using ADCS<7:0> (AD1CON3<7:0>), as described in **Section 17.4.12 "Selecting the ADC Conversion Clock Source and Prescaler"**.

   F. Turn on ADC module using AD1CON1<15>, as described in **Section 17.4.14 "Turning the ADC On"**.

Note:	Steps A through E, above, can be performed in any order, but Step F must be the final step in every case.

2. To configure ADC interrupt (if required).

   A-1. Clear AD1IF bit (IFS1<1>), as described in **Section 17.7 "Interrupts"**.
   A-2. Select ADC interrupt priority AD1IP<2:0> (IPC<28:26>) and sub priority AD1IS<1:0> (IPC<24:24>), as described in **Section 17.7 "Interrupts"**, if interrupts are to be used.

3. Start the conversion sequence by initiating sampling, as described in **Section 17.4.15 "Initiating Sampling"**.

17.4.1 Configuring Analog Port Pins

The AD1PCFG register and the TRISB register control the operation of the ADC port pins.

AD1PCFG specifies the configuration of device pins to be used as analog inputs. A pin is configured as analog input when the corresponding PCFGn bit (AD1PCFG<n>) = 0. When the bit = 1, the pin is set to digital control. When configured for analog input, the associated port I/O digital input buffer is disabled so it does not consume current. The AD1PCFG register is cleared at Reset, causing the ADC input pins to be configured for analog input by default at Reset.

TRIS registers control the digital function of the port pins. The port pins that are desired as analog inputs must have their corresponding TRIS bit set, specifying the pin as an input. If the I/O pin associated with an ADC input is configured as an output, TRIS bit is cleared, the ports digital output level (V_{OH} or V_{OL}) will be converted. After a device Reset, all TRIS bits are set.

Notes: When reading a PORT register that shares pins with the ADC, any pin configured as an analog input reads as a '0' when the PORT latch is read.

Analog levels on any pin that is defined as a digital input (including the AN15:AN0 pins), but is not configured as an analog input, may cause the input buffer to consume current that is out of the device's specification.

17.4.2 Selecting the Analog Inputs to the ADC MUXs

The AD1CHS register is used to select which analog input pin is connected to MUX A and MUX B. Each MUX has two inputs referred to as the positive and the negative input. The positive input to MUX A is controlled by CH0SA<4:0> and the negative input is controlled by CH0NA. The positive input for MUX B is controlled by CH0SB<4:0> and the negative input is controlled by CH0NB.

The positive input can be selected from any one of the available analog input pins. The negative input can be selected as the ADC negative reference or AN1. The use of AN1 as the negative input allows the ADC to be used in a Unipolar Differential mode. Refer to the device data sheet for AN1 input voltage restrictions when used as a negative reference.

Note: When using Scan mode CH0SA<4:0> may be overridden. Refer to **Section 17.4.8 "Selecting the Scan Mode"** for more information.

17.4.3 Selecting the Format of the ADC Result

The data in the ADC result register can be read as one of eight formats. The format is controlled by FORM<2:0> (AD1CON1<10:8>). The user can select from Integer, Signed Integer, Fractional, or Signed Fractional as a 16-bit or 32-bit result. Figure 17-3 shows how a result is formatted. Table 17-2 and Table 17-3 show examples of results for select results in each of the four formats with 32-bit and 16-bit results.

Note: There is no numeric difference between 32-bit and 16-bit modes. In 32-bit mode, the sign extension is applied to all 32-bits; whereas in 16-bit mode, sign extension is only applied to the lower 16-bits of the result.

Figure 17-3: ADC Output Data Formats, 32-Bit Mode

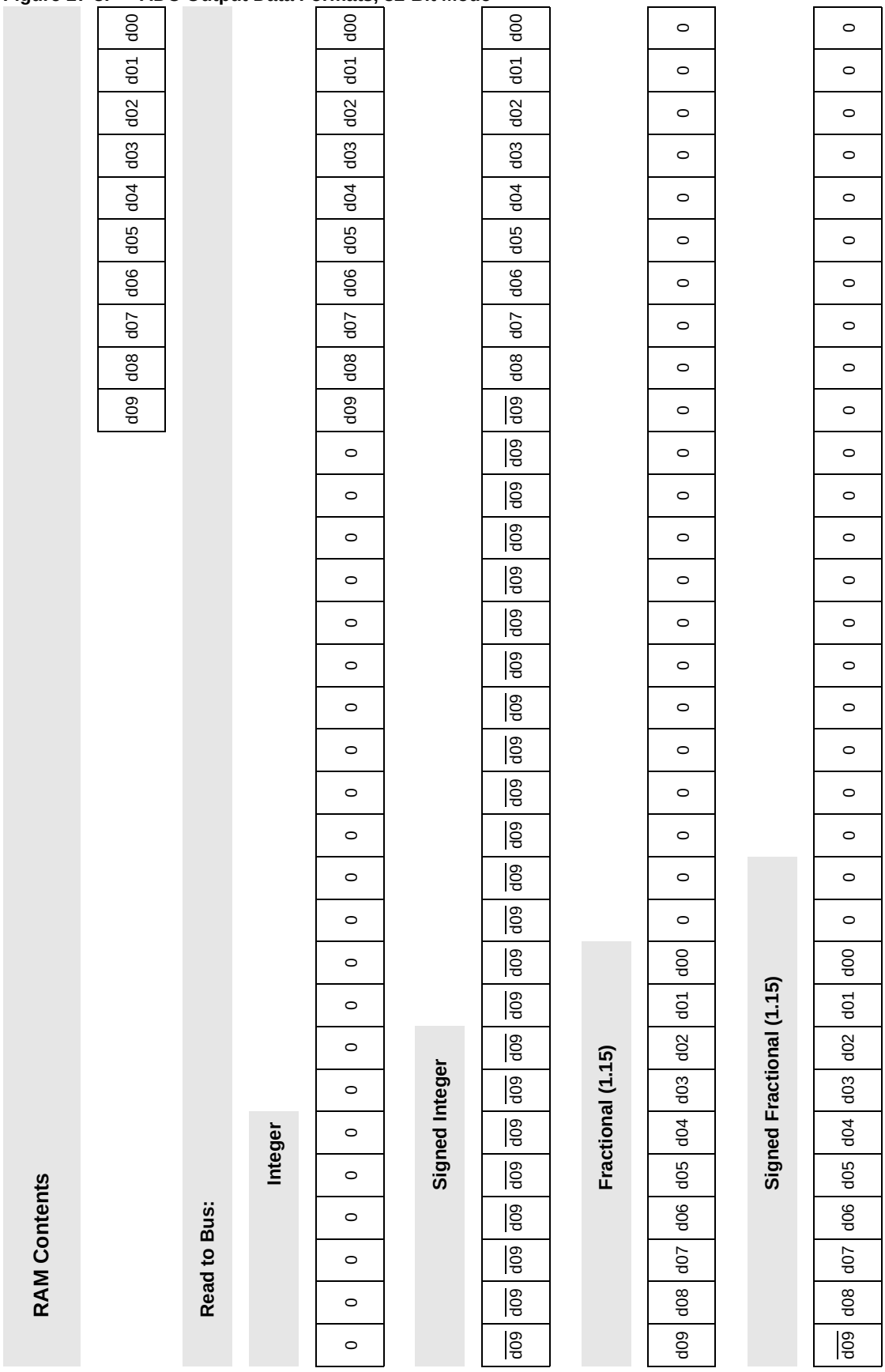


Figure 17-4: ADC Output Data Formats, 16-Bit Mode

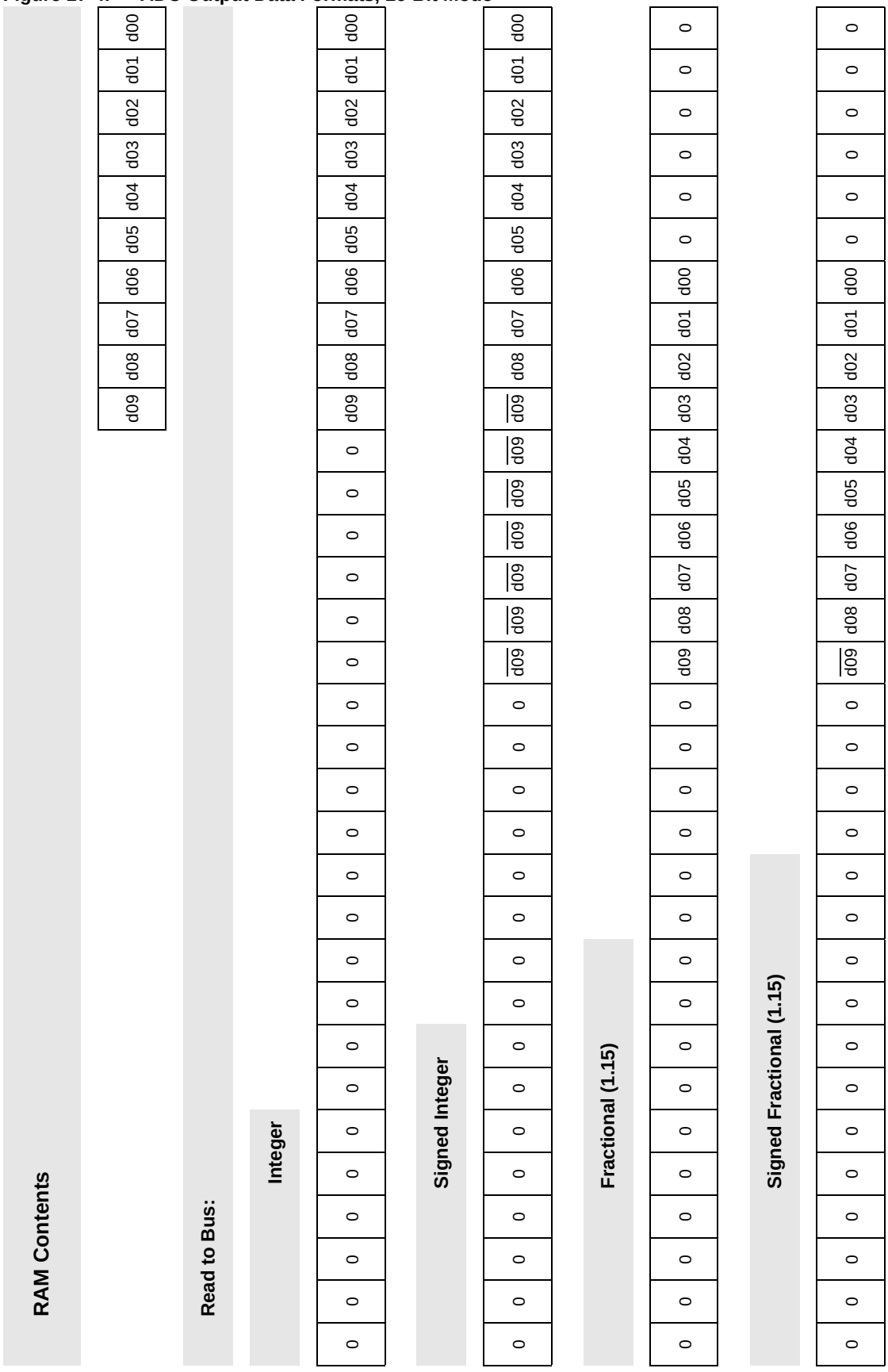


Table 17-2: Numerical Equivalents of Select Result Codes for FORM<2> (AD1CON1 <10>) = 1, 32-Bit Result

VIN/ VR	10-Bit Output Code	32-Bit Integer Format	32-Bit Signed Integer Format	32-Bit Fractional Format	32-Bit Signed Fractional Format
1023/1024	11 1111 1111	0000 0000 0000 0000 0000 0011 1111 1111 = 1023	0000 0000 0000 0000 0000 0001 1111 1111 = 511	1111 1111 1100 0000 0000 0000 0000 0000 = 0.999	0111 1111 1100 0000 0000 0000 0000 0000 = 0.499
1022/1024	11 1111 1110	0000 0000 0000 0000 0000 0011 1111 1110 = 1022	0000 0000 0000 0000 0000 0001 1111 1110 = 510	1111 1111 1000 0000 0000 0000 0000 0000 = 0.998	0111 1111 1000 0000 0000 0000 0000 0000 = 0.498
•••					
513/1024	10 0000 0001	0000 0000 0000 0000 0000 0010 0000 0001 = 513	0000 0000 0000 0000 0000 0000 0000 0001 = 1	1000 0000 0100 0000 0000 0000 0000 0000 = 0.501	0 000 0000 0100 0000 0000 0000 0000 0000 = 0.001
512/1024	10 0000 0000	0000 0000 0000 0000 0000 0010 0000 0000 = 512	0000 0000 0000 0000 0000 0000 0000 0000 = 0	1000 0000 0000 0000 0000 0000 0000 0000 = 0.500	0000 0000 0000 0000 0000 0000 0000 0000 = 0.000
511/1024	01 1111 1111	0000 0000 0000 0000 0000 0001 1111 1111 = 511	1111 1111 1111 1111 1111 1111 1111 1111 = -1	0111 1111 1100 0000 0000 0000 0000 0000 = .499	1111 1111 1100 0000 0000 0000 0000 0000 = -0.001
•••					
1/1024	00 0000 0001	0000 0000 0000 0000 0000 0000 0000 0001 = 1	1111 1111 1111 1111 1111 1110 0000 0001 = -511	0000 0000 0100 0000 0000 0000 0000 0000 = 0.001	1000 0000 0100 0000 0000 0000 0000 0000 = -0.499
0/1024	00 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000 = 0	1111 1111 1111 1111 1111 1110 0000 0000 = -512	0000 0000 0000 0000 0000 0000 0000 0000 = 0.000	1000 0000 0000 0000 0000 0000 0000 0000 = -0.500

Table 17-3: Numerical Equivalents of Select Result Codes for FORM<2> (AD1CON1 <10>) = 0, 16-Bit Result

VIN/ VR	10-bit Output Code	16-bit Integer Format	16-Bit Signed Integer Format	16-Bit Fractional Format	16-Bit Signed Fractional Format
1023/1024	11 1111 1111	0000 0011 1111 1111 = 1023	0000 0001 1111 1111 = 511	1111 1111 1100 0000 = 0.999	0111 1111 1100 0000 = 0.499
1022/1024	11 1111 1110	0000 0011 1111 1110 = 1022	0000 0001 1111 1110 = 510	1111 1111 1000 0000 = 0.998	0111 1111 1000 0000 = 0.498
•••					
513/1024	10 0000 0001	0000 0010 0000 0001 = 513	0000 0000 0000 0001 = 1	1000 0000 0100 0000 = 0.501	0 000 0000 0100 0000 = 0.001
512/1024	10 0000 0000	0000 0010 0000 0000 = 512	0000 0000 0000 0000 = 0	1000 0000 0000 0000 = 0.500	0000 0000 0000 0000 = 0.000
511/1024	01 1111 1111	0000 0001 1111 1111 = 511	1111 1111 1111 1111 = -1	0111 1111 1100 0000 = .499	1111 1111 1100 0000 = -0.001
•••					
1/1024	00 0000 0001	0000 0000 0000 0001 = 1	1111 1110 0000 0001 = -511	0000 0000 0100 0000 = 0.001	1000 0000 0100 0000 = -0.499
0/1024	00 0000 0000	0000 0000 0000 0000 = 0	1111 1110 0000 0000 = -512	0000 0000 0000 0000 = 0.000	1000 0000 0000 0000 = -0.500

17.4.4 Selecting the Sample Clock Source

It is often desirable to synchronize the end of sampling and the start of conversion with some other time event. The ADC module may use one of four sources as a conversion trigger. The selection of the conversion trigger source is controlled by the SSRC<2:0> (AD1CON1<7:5>) bits.

17.4.4.1 Manual Conversion

To configure the ADC to end sampling and start a conversion when SAMP is cleared (= 0), SSRC is set to '000'.

17.4.4.2 Timer Compare Trigger

The ADC is configured for this Trigger mode by setting SSRC<2:0> = 010. When a period match occurs for the 32-bit timer, TMR3/TMR2, or the 16-bit Timer3 a special A/D converter trigger event signal is generated by Timer3. This feature does not exist for the TMR5/TMR4 timer pair or for 16-bit timers other than Timer3. Refer to **Section 14. "Timers"** for more details.

17.4.4.2.1 External INT0 Pin Trigger

To configure the ADC to begin a conversion on an active transition on the INT0 pin, SSRC<2:0> is set to '001'. The INT0 pin may be programmed for either a rising edge input or a falling edge input to trigger the conversion process.

17.4.4.2.2 Auto-Convert

The ADC can be configured to automatically perform conversions at the rate selected by the Auto Sample Time bits SAMC<4:0>. The ADC is configured for this Trigger mode by setting SSRC<2:0> = 111. In this mode, the ADC will perform continuous conversions on the selected channels.

17.4.5 Synchronizing ADC Operations to Internal or External Events

The modes where an external event trigger pulse ends sampling and starts conversion (SSRC2:SSRC0 = 001, 010 or 011) may be used in combination with auto-sampling (ASAM = 1) to cause the ADC to synchronize the sample conversion events to the trigger pulse source. For example, in Figure 17-13 where SSRC = 010 and ASAM = 1, the ADC will always end sampling and start conversions synchronously with the timer compare trigger event. The ADC will have a sample conversion rate that corresponds to the timer comparison event rate. See Example 17-5 for a code example.

17.4.6 Selecting Automatic or Manual Sampling

Sampling can be started manually or automatically when the previous conversion is complete.

17.4.6.1 Manual

Clearing the ASAM (AD1CON1<2>) bit disables the Auto-Sample mode. Acquisition will begin when the SAMP (AD1CON1<1>) bit is set by software. Acquisition will not resume until the SAMP bit is once again set. For an example, see Figure 17-8.

17.4.6.2 Automatic

Setting the ASAM (AD1CON1<2>) bit enables the Auto-Sample mode. In this mode, the sampling will start automatically after the pervious sample has been converted. For an example, see Figure 17-9.

17.4.7 Selecting the Voltage Reference Source

The user can select the voltage reference for the ADC module. The reference can be internal or external.

The VCFG<2:0> control bits (AD1CON2<15:13>) select the voltage reference for A/D conversions. The upper voltage reference (V_R+) and the lower voltage reference (V_R-) may be the internal AV_{DD} and AV_{SS} voltage rails, or the $V_{REF}+$ and $V_{REF}-$ input pins. The external ADC voltage reference may be used to reduce noise in the converter.

The external voltage reference pins may be shared with the AN0 and AN1 inputs on low pin count devices. The A/D converter can still perform conversions on these pins when they are shared with the $V_{REF}+$ and $V_{REF}-$ input pins.

The voltages applied to the external reference pins must meet certain specifications. Refer to the electrical specifications section of the device data sheet for the electrical specifications.

Notes: External references $V_{REF}+$ and $V_{REF}-$ must be selected for high conversion. See the data sheet for further details.

The external $V_{REF}+$ and $V_{REF}-$ pins may be shared with other analog peripherals. Refer the device data sheet for further details.

17.4.8 Selecting the Scan Mode

The ADC module has the ability to scan through a selected vector of inputs. The CSCNA bit (AD1CON2<10>) enables the MUX A input to be scanned across a selected number of analog inputs.

17.4.8.1 Scan Mode Enable

Scan mode is enabled by setting CSCNA (AD1CON2<10>). When Scan mode is enabled the positive input of MUX A is controlled by the contents of the AD1CSSL register. Each bit in the AD1CSSL register corresponds to an analog input. Bit 0 corresponds to AN0, bit 1 corresponds to AN1 and so on. If a particular bit in the AD1CSSL register is '1', the corresponding input is part of the scan sequence. The inputs are always scanned from lower- to higher-numbered inputs, starting at the first selected channel after each interrupt occurs. When Scan mode is enabled the CH0SA<3:0> bits are ignored.

Notes:	If the number of scanned inputs selected is greater than the number of samples taken per interrupt, the higher numbered inputs will not be sampled. The AD1CSSL bits only specify the input of the positive input of the channel. The CH0NA bit selects the input of the negative input of the channel during scanning.

17.4.8.2 Scan Mode Disable

When CSCNA = 0, Scan mode is disabled and the positive input to MUX A is controlled by CH0SA<3:0>.

17.4.8.3 Using Scan and Alternate Modes Together

The Scan and Alternate modes may be combined to allow a vector of inputs to be scanned and a single input to be converted every other sample.

This mode is enabled by setting the CSCNA bit = 1, and setting the ALTS (AD1CON2<0>) bit = 1.

The CSCNA bit enables the scan for MUX A, and the CH0SB<3:0> (AD1CHS<27:24>) and CH0NB (AD1CHS<31>) are used to configure the inputs to MUX B. Scanning only applies to the MUX A input selection. The MUX B input selection, as specified by CH0SB<3:0>, will still select a single input.

The following sequence is an example of 3 scanned channels (MUX A) and a single fixed channel (MUX B):

1. The first input in the scan list is sampled.
2. The input selected by CH0SB<3:0> and CH0NB is sampled.
3. The second input in the scan list is sampled.
4. The input selected by CH0SB<3:0> and CH0NB is sampled.
5. The third input in the scan list is sampled.
6. The input selected by CH0SB<3:0> and CH0NB is sampled.

The process is repeated.

17.4.9 Setting the Number of Conversions per Interrupt

The SMPI<3:0> bits (AD1CON2<5:2>) select how many A/D conversions will take place before a CPU interrupt is generated. This also defines the number of locations that will be written in the result buffer stating with ADC1BUF0 (ADC1BUF0 or ADC1BUF8 for Dual Buffer mode). This can vary from 1 sample to 16 samples (1 to 8 samples for Dual Buffer mode). After the interrupt is generated, the sampling sequence restarts; with the result of the first sample being written to the first buffer location.

For example, if SMPI<3:0> = 0000, the conversion results will always be written to ADC1BUF0. In this example, no other buffer locations would be used.

For example, if SMPI<3:0> = 1110, 15 samples would be converted and stored in buffer locations ADC1BUF0 through ADC1BUFE. An interrupt would be generated after ADC1BUFE was written. The next sample would be written to ADC1BUF0. In this example ADC1BUFF would not be used.

The data in the result registers will be overwritten by the next sampling sequence. The data in the result buffer must be read before the completion of the first sample after the interrupt is generated. The Buffer Fill mode can be used to increase the time between interrupt generation and the overwriting of data. Refer to the Buffer Fill Mode section.

The user cannot program a combination of samples and SMPI bits that results in more than 16 conversions per interrupt when the BUFM bit (AD1CON2<1>) is '1', or more than 8 conversions per interrupt when the BUFM bit (AD1CON2<1>) is '0'. Attempting to create a conversion list with the number of samples greater than 16 will result in the sampling sequence being truncated to 16 samples.

17.4.10 Buffer Fill Mode

The Buffer Fill mode allows the output buffer to be used as a single 16-word buffer or two 8-word buffers.

When BUFM is '0', the complete 16-word buffer is used for all conversion sequences. Conversion results will be written sequentially in the buffer starting at ADC1BUF0 until the number of samples as defined by SMPI<3:0> (AD1CON2<5:2>) is reached. The next conversion result will be written to ADC1BUF0 and the process repeats. If the ADC interrupt is enabled an interrupt will be generated when the number of samples in the buffer equals SMPI<3:0>.

When the BUFM bit (AD1CON2<1>) is '1', the 16-word results buffer (ADRES) will be split into two 8-word groups. Conversion results will be written sequentially into the first buffer starting at ADC1BUF0, BUFS (AD1CON2<7>) will be cleared, until the number of samples as defined by SMPI<3:0> (AD1CON2<5:2>) is reached. The ADC interrupt flag will then be set.

After the ADC interrupt flag is set the following result will be written sequentially to the second buffer starting at ADC1BUF8 The next conversion result will be written to the second buffer starting at ADC1BUF8, BUFS (AD1CON2<7>) will be set, until the number of samples as defined by SMPI<3:0> (AD1CON2<5:2>) is reached. The ADC interrupt flag will then be set.

The process then restarts with BUFS = 0 and results being written to the first buffer.

The decision of which Buffer Fill mode to use will depend upon how much time is available to move the buffer contents after the A/D interrupt and the interrupt latency, as determined by the application. If the processor can unload a full buffer within the time it takes to sample and convert one channel, the BUFM bit can be '0' and up to 16 conversions may be done per interrupt. The processor will have one acquisition-and-conversion period before the first buffer location is overwritten.

If the processor cannot unload the buffer within the sample-and-conversion time the Dual Buffer mode BUFM bit = 1, should be used to prevent overwriting result data. For example, if SMPI<3:0> = 0111, then eight conversions will be written loaded into the first buffer, following which an interrupt will occur. The next eight conversions will be written to the second buffer. Therefore the processor will have the entire time between interrupts to read the eight conversions out of the buffer.

17.4.11 Selecting the MUX to be Connected to the ADC (Alternating Sample Mode)

The ADC has two input MUXs that connect to the SHA. These MUXs are used to select which analog input is to be sampled. Each of the MUXs have a positive and a negative input (see Figure 17-5 and Figure 17-6).

Note:	The number of analog inputs will vary among different devices. Verify the analog input availability with the appropriate device data sheet.

17.4.11.1 Single Input Selection

The user may select one of up to 16 analog inputs, as determined by the number of analog channels on the device, as the positive input of the SHA. The CH0SA<3:0> bits (AD1CHS<19:16>) select the positive analog input.

The user may select either V_{R-} or AN1 as the negative input. The CH0NA bit (AD1CHS<23>) selects the analog input for the negative input of channel 0. Using AN1 as the negative input allows unipolar differential measurements.

The ALTS bit (AD1CON2<0>) must be clear for this mode of operation.

17.4.11.2 Alternating Input Selections

The ALTS bit causes the module to alternate between the two input MUXs.

The inputs specified by CH0SA<3:0> and CH0NA are called the MUX A inputs. The inputs specified by CH0SB<3:0> and CH0NB are called the MUX B inputs, see Figure .

When ALTS is '1', the module will alternate between the MUX A inputs on one sample and the MUX B inputs on the subsequent sample. When ALTS is '0', only the inputs specified by CH0SA<3:0> and CH0NA are selected for sampling.

For example, if ALTS is '1' on the first sample/convert sequence, the inputs specified by CH0SA<3:0> and CH0NA are selected for sampling. On the next sample, the inputs specified by CH0SB<3:0> and CH0NB are selected for sampling. Then the pattern repeats.

17.4.12 Selecting the ADC Conversion Clock Source and Prescaler

The ADC module can use the internal RC oscillator or the PBCLK as the conversion clock source.

When the internal RC oscillator is used as the clock source, ADRC (AD1CON3<15>) = 1, the TAD is the period of the oscillator, no prescaler are used. When using the internal oscillator the ADC can continue to function in SLEEP and in IDLE.

Note:	The ADRC is intended for ADC operation in Sleep it is not calibrated. Applications requiring precise timing of ADC acquisitions should use a stable calibrated clock source for the ADC.

When the PBCLK is used as the conversion clock source, ADRC = 0, the TAD is the period of the PBCLK after the prescaler ADCS<7:0> (AD1CON3<7:0>) is applied.

The A/D converter has a maximum rate at which conversions may be completed. An analog module clock, TAD, controls the conversion timing. The A/D conversion requires 12 clock periods (12 TAD).

The period of the ADC conversion clock is software selected using a 8-bit counter. There are 256 possible options for TAD, specified by the ADCS<7:0> bits (AD1CON3<7:0>).

Equation 17-1 gives the TAD value as a function of the ADCS control bits and the device instruction cycle clock period, TCY.

Equation 17-1: ADC Conversion Clock Period

$$T_{AD} = 2 \bullet (T_{PB}(ADCS + 1)$$

$$ADCS = (T_{AD}/(2 \bullet T_{PB})) - 1$$

For correct A/D conversions, the ADC conversion clock (TAD) must be selected to ensure a minimum TAD time of 83.33 nsec (see Section 17.11.1).

Equation 17-2: Available Sampling Time, Sequential Sampling

$$T_{SMP} = \text{Trigger Pulse Interval } (T_{SEQ}) - \text{Conversion Time } (T_{CONV})$$

$$T_{SMP} = T_{SEQ} - T_{CONV}$$

Note: TSEQ is the trigger pulse interval time.

17.4.13 Acquisition Time Considerations

Different acquisition/conversion sequences provide different times for the sample-and-hold channel to acquire the analog signal. The user must ensure the acquisition time meets the sampling requirements, as outlined in **Section 17.11.3 "ADC Sampling Requirements"**.

When SSRC<2:0> (AD1CON1<7:5>) = 111, the conversion trigger is under ADC clock control. The SAMC<4:0> bits (AD1CON3<12:8>) select the number of TAD clock cycles between the start of acquisition and the start of conversion. This trigger option provides the fastest conversion rates on multiple channels. After the start of acquisition, the module will count a number of TAD clocks specified by the SAMC bits.

17.4.14 Turning the ADC On

When the ON bit (AD1CON1<15>) is '1', the module is in Active mode and is fully powered and functional.

When ON is '0', the module is disabled. The digital and analog portions of the circuit are turned off for maximum current savings.

In order to return to the Active mode from the Off mode, the user must wait for the analog stages to stabilize. For the stabilization time, refer to the Electrical Characteristics section of the device data sheet.

Note:	Writing to ADC control bits other than ON (AD1CON1<15>), SAMP (AD1CON1<1>), and DONE (AD1CON1<0>) is not recommended while the A/D converter is running.

17.4.15 Initiating Sampling

17.4.15.1 Manual Mode

In manual sampling, a acquisition is started by writing a '1' to the SAMP (AD1CON1<1>) bit. Software must manually manage the start and end of the acquisition period by setting SAMP and then clearing SAMP after the desired acquisition period has elapsed.

17.4.15.2 Auto-Sample Mode

In Auto-Sample mode, the sampling process is started by writing a '1' to the ASAM (AD1CON1<2>) bit. In Auto-Sample mode, the acquisition period is defined by ADCS<7:0> (AD1CON3<7:0>). Acquisition is automatically started after a conversion is completed. Auto-Sample mode can be used with any trigger source other than manual.

17.5 MISCELLANEOUS ADC FUNCTIONS

The following section describes bits not covered in the previous section.

17.5.1 Aborting Sampling

Clearing the SAMP (AD1CON1<1>) bit while in Manual Sample mode will terminate sampling, but may also start a conversion if SSRC (AD1CON1<7:5>) = 000.

Clearing the ASAM (AD1CON1<2>) bit while in Auto-Sample mode will not terminate an ongoing acquire/convert sequence, however, sampling will not automatically resume after the current sample is converted.

17.5.2 Aborting a Conversion

Clearing the ON (AD1CON1<15>) bit during a conversion will abort the current conversion. The ADC Result register will NOT be updated with the partially completed A/D conversion sample. That is, the corresponding result buffer location will continue to contain the value of the last completed conversion (or the last value written to the buffer).

17.5.3 Buffer Fill Status

When the conversion result buffer is split using the BUFM control bit, the BUFS Status bit (AD1CON2<7>) indicates which half of the buffer the A/D converter is currently filling. If BUFS = 0, then the A/D converter is filling ADC1BUF0-ADC1BUF7 and the user software should read conversion values from ADC1BUF8-ADC1BUFF. If BUFS = 1, the situation is reversed and the user software should read conversion values from ADC1BUF0-ADC1BUF7.

17.5.4 Offset Calibration

The ADC module provides a method of measuring the internal offset error. After this offset error is measured, it can be subtracted, in software, from the result of a A/D conversion. Use the following steps to perform an offset measurement:

1. Configure the A/D converter in the same manner as it will be used in the application.
2. Set the OFFCAL bit (AD1CON2<12>). This overrides the input selections and connects the sample-and-hold inputs to AVss.
3. If auto-sample is used set the CLRASAM bit (AD1CON1<4>) to stop conversions when the number of samples stated by SMPI is reached.
4. Enable the A/D converter and perform a conversion. The result that is written to the ADC result buffer is the internal offset error.
5. Clear the OFFCAL (AD2CON<12>) bit to return the A/D converter to normal operation.

Note:	Only positive ADC offsets can be measured with this method.

17.5.5 Terminate Conversion Sequence after an Interrupt

The CLRASAM bit provides a method to terminate auto-sample after the first sequence is completed. Setting the CLRASAM and starting an auto-sample sequence will cause the A/D converter to complete one auto-sample sequence (the number of samples as defined by SMPI<3:0> (AD1CON2<5:2>)). Hardware will the clear ASAM (AD1CON1<2>) and set the interrupt flag. This will stop the sampling process to allow inspection of the result buffer without results being overwritten by the next automatic conversion sequence. The CLRASAM must be cleared by software to disable this mode.

Note:	Disabling Interrupts or masking the ADC interrupt has no effect on the operation of the CLRASAM bit.

17.5.6 DONE Bit Operation

The DONE (AD1CON1<0>) bit is set when a conversion sequence is complete.

In Manual mode the DONE bit is persistent. It remains set until it is cleared by software. The DONE bit can be polled to determine when the conversion has completed.

In all automatic sample modes (ASAM = 1) the DONE bit is not persistent. It is set at the end of a conversion sequence and cleared by hardware when the next acquisition is started. Polling the DONE bit is not recommended when operating the ADC in automatic modes. The AD1IF (IFS1<1>) flag is latched after a conversion sequence is completed and can therefore be polled.

Figure 17-5: Simplified 10-Bit High-Speed A/D Converter Block Diagram for Alternate Sample Mode

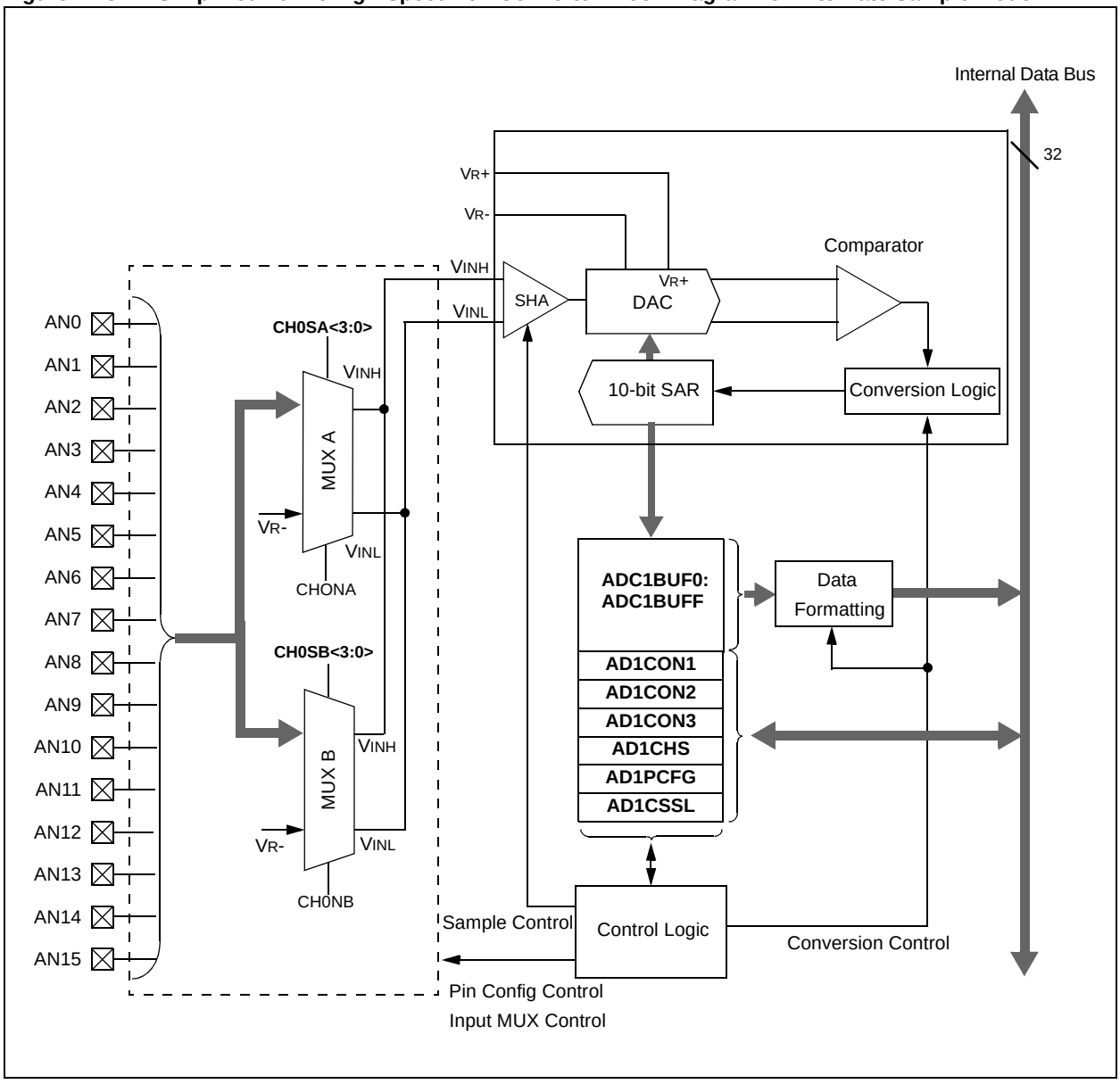


Figure 17-6: Simplified 10-Bit High-Speed A/D Converter Block Diagram for Scan Mode

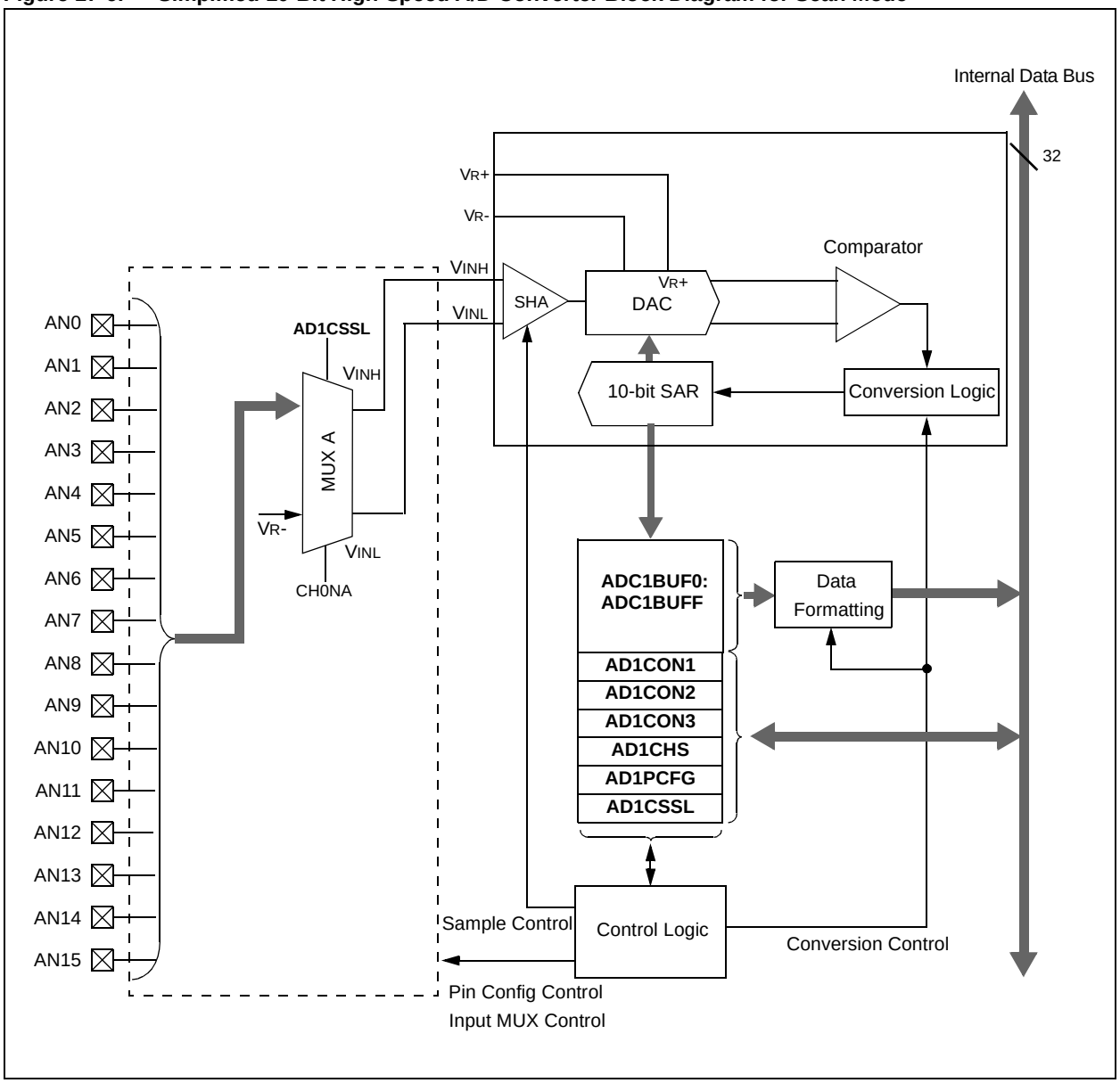
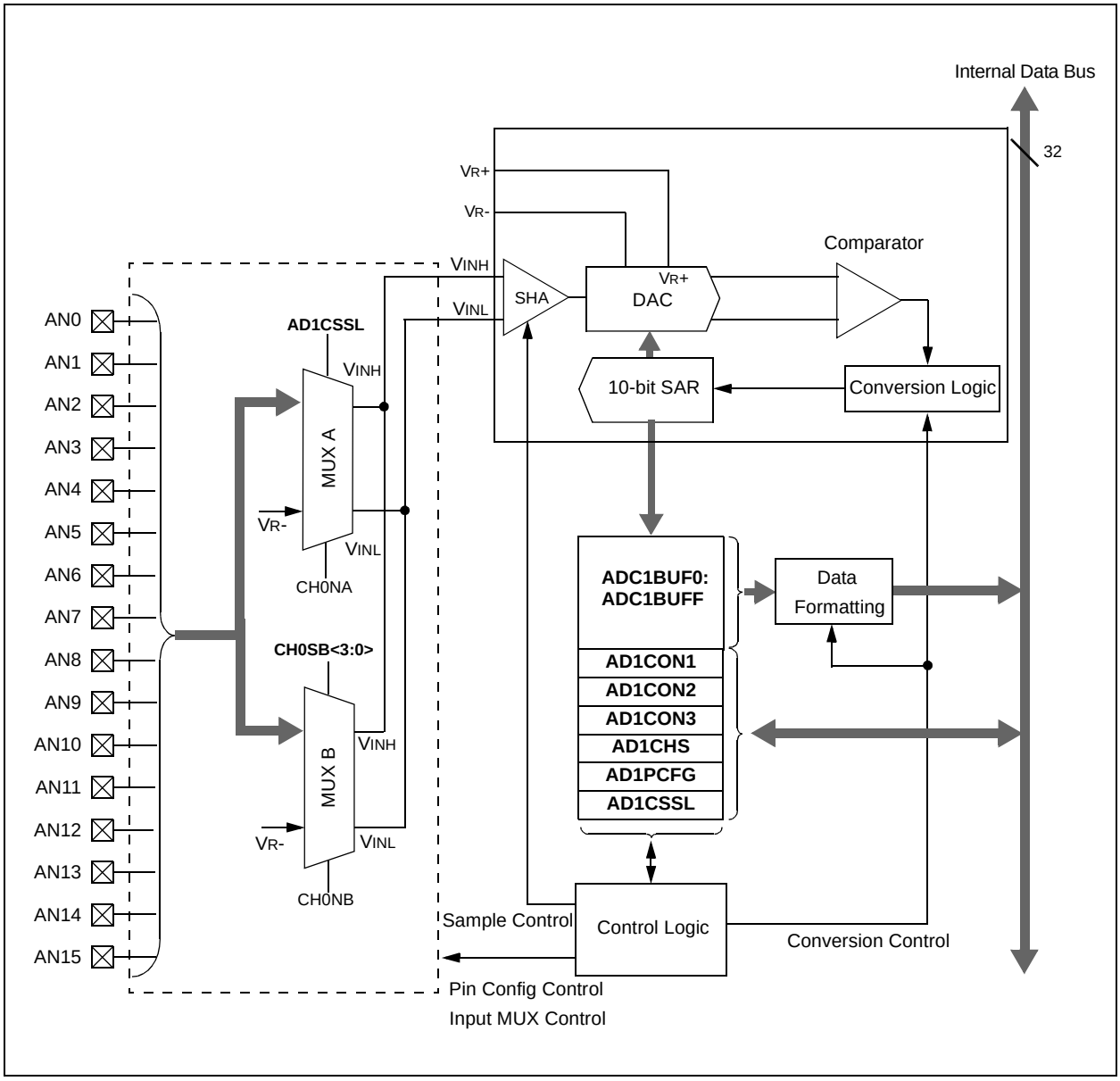


Figure 17-7: Simplified 10-Bit High-Speed A/D Converter Block Diagram for Alternate Sample and Scan Mode



17.5.7 Conversion Sequence Examples

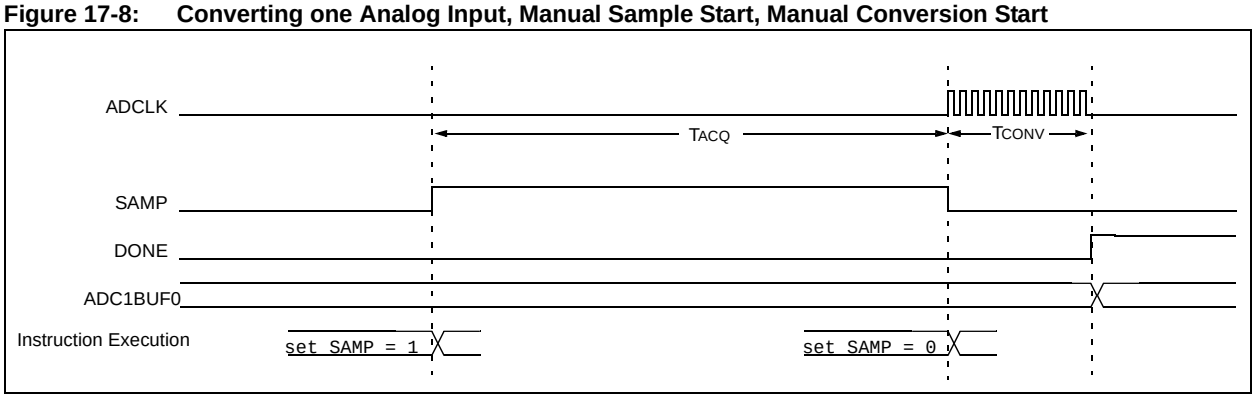
The following configuration examples show the ADC operation in different sampling and buffering configurations. In each example, setting the ASAM bit starts automatic sampling. A conversion trigger ends sampling and starts conversion.

17.5.8 Manual Conversion Control

When SSRC<2:0> = 000, the conversion trigger is under software control. Clearing the SAMP bit (AD1CON1<1>) starts the conversion sequence.

Figure 17-8 is an example where setting the SAMP bit initiates sampling and clearing the SAMP bit terminates sampling and starts conversion. The user software must time the setting and clearing of the SAMP bit to ensure adequate acquisition time of the input signal. See Example 17-1 for a code example.

Figure 17-8: Converting one Analog Input, Manual Sample Start, Manual Conversion Start



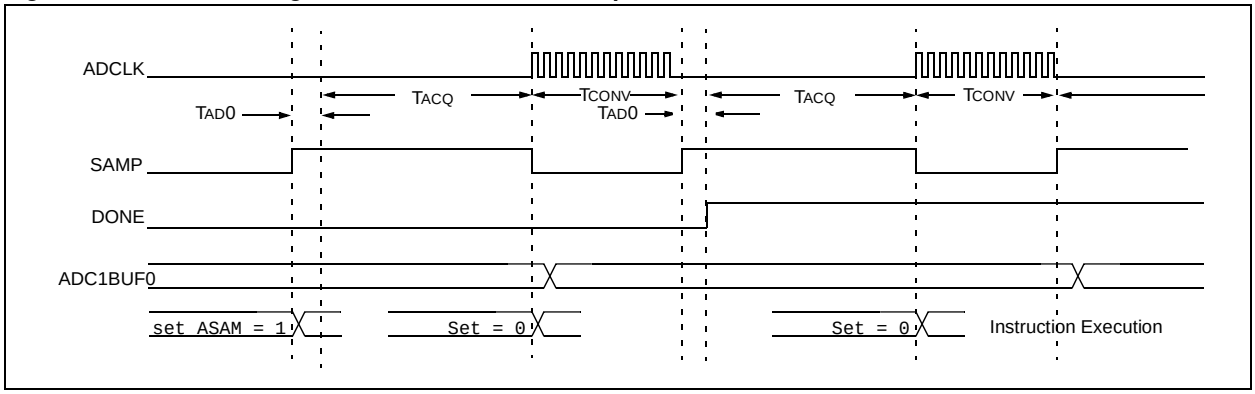
Example 17-1: Converting 1 Channel, Manual Sample Start, Manual Conversion Start Code

```
  AD1PCFG = 0xFFFB;             // PORTB = Digital; RB2 = analog
  AD1CON1 = 0x0000;             // SAMP bit = 0 ends sampling ...
                                // and starts converting
  AD1CHS = 0x00020000;          // Connect RB2/AN2 as CH0 input  ..
                                // in this example RB2/AN2 is the input
  AD1CSSL = 0;
  AD1CON3 = 0x0002;             // Manual Sample, Tad = internal 6 TPB
  AD1CON2 = 0;


  AD1CON1SET = 0x8000;          // turn ADC ON
  while (1)                     // repeat continuously
  {
     AD1CON1SET = 0x0002;       // start sampling ...
     DelayNmSec(100);           //  for 100 mS
     AD1CON1CLR = 0x0002;       // start Converting
     while (!(AD1CON1 & 0x0001));// conversion done?
     ADCValue = ADC1BUF0;       // yes then get ADC value
  }                             // repeat
```


17.5.9 Automatic Acquisition

Figure 17-9 is an example in which setting the ASAM (AD1CON1<2>) bit initiates automatic acquisition, and clearing the SAMP (AD1CON1<1>) bit terminates sampling and starts conversion. After the conversion completes, the module will automatically return to a acquisition state. The SAMP bit is automatically set at the start of the acquisition interval. The user software must time the clearing of the SAMP bit to ensure adequate acquisition time of the input signal, understanding that the time between clearing of the SAMP bit includes the conversion time as well as the acquisition time. See Example 17-2 for a code example.

Figure 17-9: Converting 1 Channel, Automatic Sample Start, Manual Conversion Start



Example 17-2: Converting 1 Channel, Automatic Sample Start, Manual Conversion Start Code

```
AD1PCFG = 0xFF7F;            // all PORTB = Digital but RB7 = analog
AD1CON1 = 0x0004;            // ASAM bit = 1 implies acquisition ..
                             // starts immediately after last
                             // conversion is done
AD1CHS = 0x00070000;         // Connect RB7/AN7 as CH0 input  ..
                             // in this example RB7/AN7 is the input
AD1CSSL = 0;
AD1CON3 = 0x0002;            // Sample time manual, Tad = internal 6 TPB
AD1CON2 = 0;


AD1CON1SET = 0x8000;         // turn ADC ON
while (1)                    // repeat continuously
{
  DelayNmSec(100);           // sample for 100 mS
  AD1CON1SET = 0x0002;       // start Converting
  while (!(AD1CON1 & 0x0001));// conversion done?
  ADCValue = ADC1BUF0;       // yes then get ADC value
}                            // repeat
```

17.5.10 Clocked Conversion Trigger

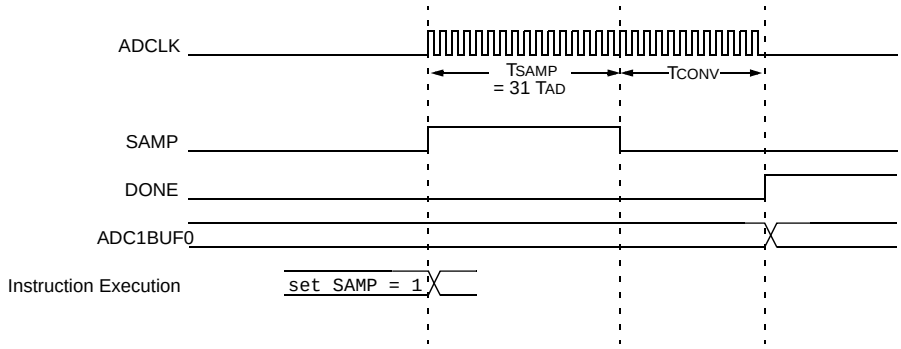
When SSRC<2:0> = 111, the conversion trigger is under ADC clock control. The SAMC bits (AD1CON3<4:0>) select the number of TAD clock cycles between the start of acquisition and the start of conversion. This trigger option provides the fastest conversion rates on multiple channels. After the start of acquisition, the module will count a number of TAD clocks specified by the SAMC bits.

Equation 17-3: Clocked Conversion Trigger Time

$$TSMP = SAMC<4:0> * TAD$$

SAMC must always be programmed for at least one clock cycle. See Example 17-3 for a code example.

Figure 17-10: Converting 1 Channel, Manual Sample Start, TAD Based Conversion Start



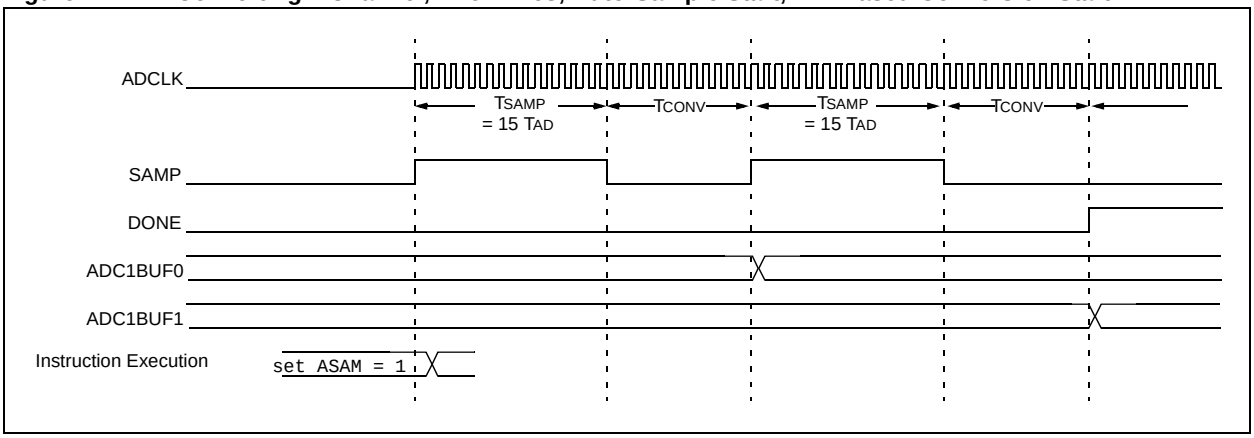
Example 17-3: Converting 1 Channel, Manual Sample Start, TAD Based Conversion Start Code

```
AD1PCFG = 0xEFFF;                   // all PORTB = Digital; RB12 = analog
AD1CON1 = 0x00E0;                   // SSRC bit = 111 implies internal
                                    // counter ends sampling and starts
                                    // converting.
AD1CHS = 0x000C0000;                // Connect RB12/AN12 as CH0 input  ..
                                    // in this example RB12/AN12 is the input
AD1CSSL = 0;
AD1CON3 = 0x1F02;                   // Sample time = 31Tad
AD1CON2 = 0;


AD1CON1SET = 0x8000;                // turn ADC ON
while (1)                           // repeat continuously
{
   AD1CON1CLR = 0x0002;             // start sampling then ...
                                    // after 31Tad go to conversion
   while (!(AD1CON1 & 0x0001));     // conversion done?
   ADCValue = ADC1BUF0;             // yes then get ADC value
}                                   // repeat
```


17.5.11 Free Running Sample Conversion Sequence

As shown in Figure 17-11, using the Auto-Convert Conversion Trigger mode (SSRC = 111) in combination with the Automatic Sampling Start mode (ASAM = 1), allows the ADC module to schedule acquisition/conversion sequences with no intervention by the user or other device resources. This “Clocked” mode allows continuous data collection after module initialization. See Example 17-4 for a code example.

Figure 17-11: Converting 1 Channel, Two Times, Auto-Sample Start, TAD Based Conversion Start



Example 17-4: Converting 1 Channel, Auto-Sample Start, TAD Based Conversion Start Code

```
AD1PCFG = 0xFFFB;                   // all PORTB = Digital; RB2 = analog
AD1CON1 = 0x00E0;                   // SSRC bit = 111 implies internal
                                    // counter ends sampling and starts
                                    // converting.
AD1CHS  = 0x00020000;               // Connect RB2/AN2 as CH0 input  ..
                                    // in this example RB2/AN2 is the input
AD1CSSL = 0;
AD1CON3 = 0x0F00;                   // Sample time = 15Tad
AD1CON2 = 0x0004;                   // Interrupt after every 2 samples



AD1CON1SET = 0x8000;                // turn ADC ON
while (1)                           // repeat continuously
{
   ADCValue = 0;                    // clear value
   ADC16Ptr = &ADC1BUF0;            // initialize ADC1BUF0 pointer
   IFS1CLR = 0x0002;                // clear ADC interrupt flag
   AD1CON1SET = 0x0004;             // auto start sampling
                                    // for 31Tad then go to conversion
   while (!IFS1 & 0x0002);          // conversion done?
   AD1CON1CLR = 0x0004;             // yes then stop sample/convert
   for (count = 0; count < 2; count++)// average the 2 ADC values
   {
      ADCValue = ADCValue + *(ADC16Ptr++);
      ADCValue = ADCValue >> 1;
   }                                // repeat
}
```

17.5.12 Acquisition Time Considerations Using Clocked Conversion Trigger and Automatic Sampling

Different acquisition/conversion sequences provide different available acquisition times for the sample-and-hold channel to acquire the analog signal. The user must ensure the acquisition time exceeds the acquisition requirements, as outlined in **Section 17.11.3 "ADC Sampling Requirements"**.

Assuming that the module is set for automatic sampling and using a clocked conversion trigger, the acquisition interval is determined by the SAMC (AD1CON3<12:8>) bits.

Equation 17-4: Available Sampling Time

$$TSMP = SAMC<4:0> * TAD$$

Figure 17-12: Converting 1 Channel, Manual Sample Start, Conversion Trigger Based Conversion Start

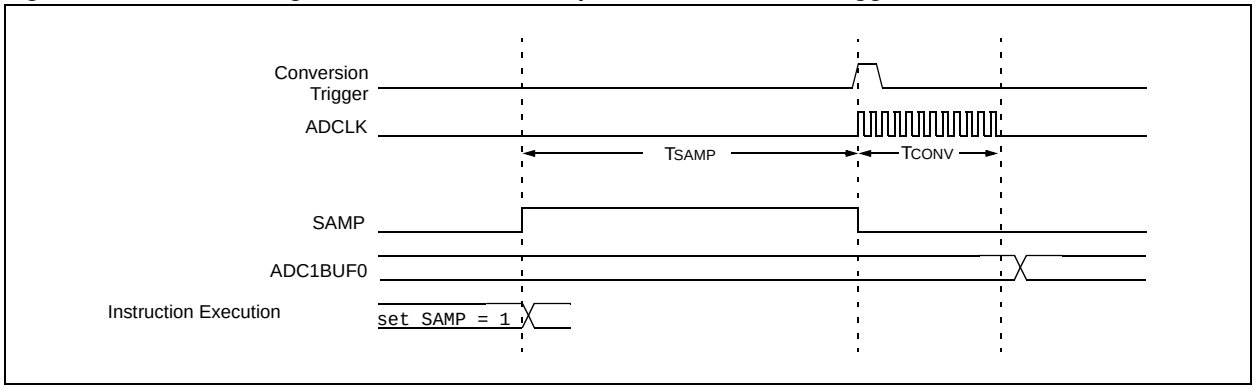
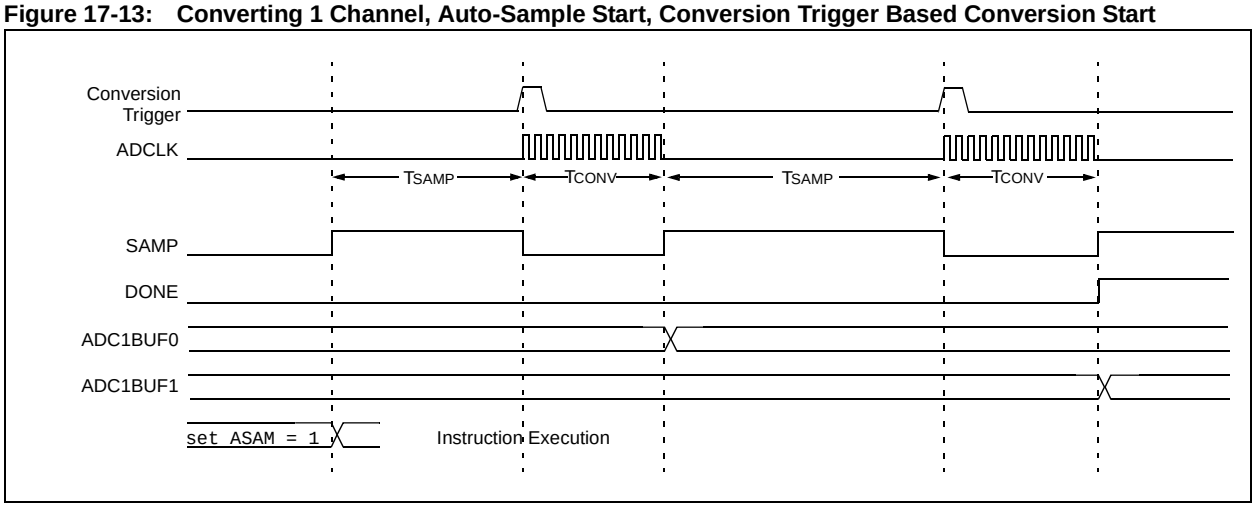


Figure 17-13: Converting 1 Channel, Auto-Sample Start, Conversion Trigger Based Conversion Start



Example 17-5: Converting 1 Channel, Auto-Sample Start, Conversion Trigger Based Conversion Start Code

```
AD1PCFG = 0xFFFB;                   // all PORTB = Digital; RB2 analog
AD1CON1 = 0x0040;                   // SSRC bit = 010 implies GP TMR3
                                    // compare ends sampling and starts
                                    // converting.
AD1CHS = 0x00020000;                // Connect RB2/AN2 as CH0 input  ..
                                    // in this example RB2/AN2 is the input
AD1CSSL = 0;
AD1CON3 = 0x0000;                   // Sample time is TMR3, Tad = internal TPB*2
AD1CON2 = 0x0004;                   // Interrupt after 2 conversions

                                    // set TMR3 to time out every 125 mSecs
TMR3= 0x0000;
PR3= 0x3FFF;
T3CON = 0x8010;

AD1CON1SET = 0x8000;                // turn ADC ON
AD1CON1SET = 0x0004;                // start auto sampling every 125 mSecs
while (1)                           // repeat continuously
{
  while (!IFS1 & 0x0002){};         // conversion done?
  ADCValue = ADC1BUF0;              // yes then get first ADC value
  IFS1CLR = 0x0002;                 // clear ADIF
}                                   // repeat
```

17.5.13 Sampling a Single Channel Multiple Times

Figure 17-14 and Table 17-4 illustrate a basic configuration of the A/D converter. In this case, one ADC input, AN0, will be acquired and converted. The results are stored in the ADC1BUF buffer. This process repeats 15 times until the buffer is full, and then the module generates an interrupt. Then entire process repeats.

With ALTS (AD1CON2<0>) clear, only the MUX A inputs are active. The CH0SA (AD1CHS<19:16>) bits and CH0NA (AD1CHS<23>) bit are specified (AN0-VREF-) as the input to the sample/hold channel. Other input selection bits are not used.

Figure 17-14: Converting One Channel 15 Times 15 Samples Per Interrupt

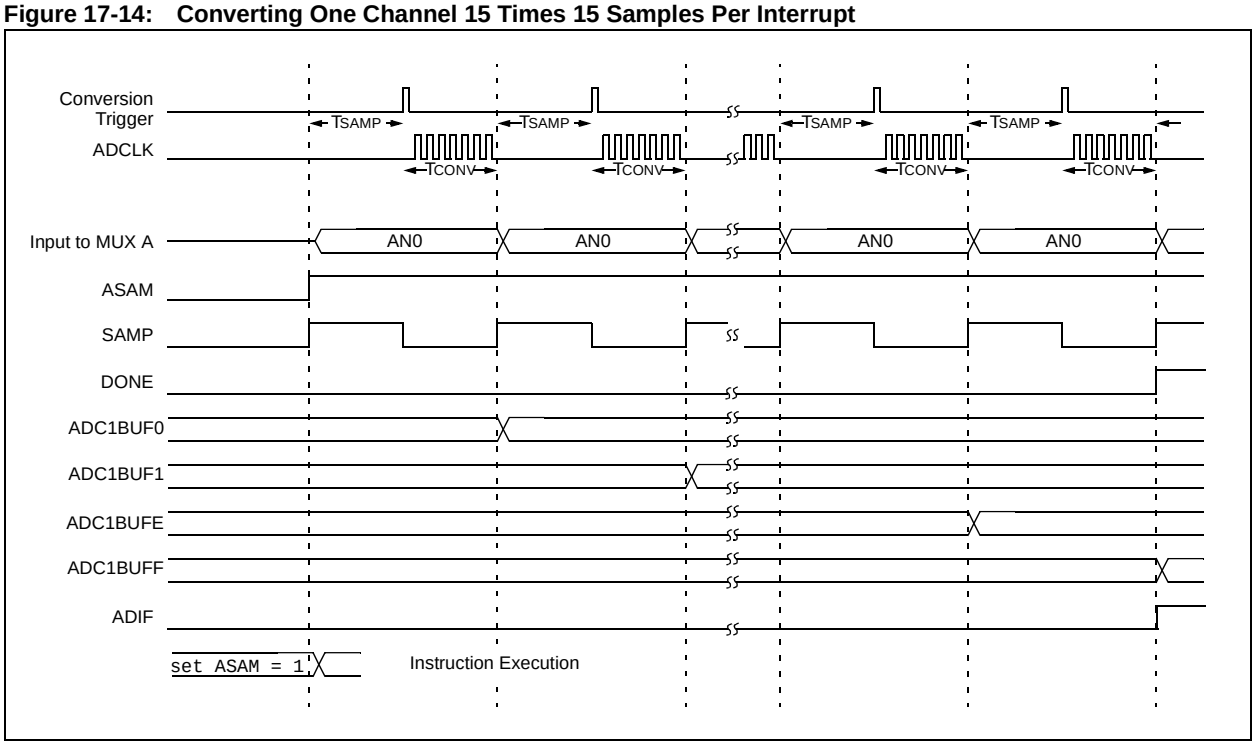


Table 17-4: Converting One Channel 15 Times/Interrupt

CONTROL BITS	OPERATION SEQUENCE
Sequence Select	Sample MUX A Inputs: AN0
SMPI<2:0> = 1111	Convert, Write Buffer 0x0
Interrupt on 15th sample	Sample MUX A Inputs: AN0
—	Convert, Write Buffer 0x1
—	Sample MUX A Inputs: AN0
BUFM = 0	Convert, Write Buffer 0x2
Single 16-word result buffer	Sample MUX A Inputs: AN0
ALTS = 0	Convert, Write Buffer 0x3
Always use MUX A input select	Sample MUX A Inputs: AN0
MUX A Input Select	Convert, Write Buffer 0x4
CH0SA<3:0> = 0000	Sample MUX A Inputs: AN0
Select AN0 for CH0+ input	Convert, Write Buffer 0x5
CH0NA = 0	Sample MUX A Inputs: AN0
Select VR- for CH0- input	Convert, Write Buffer 0x6
CSCNA = 0	Sample MUX A Inputs: AN0
No input scan	Convert, Write Buffer 0x7
CSSL<15:0> = n/a	Sample MUX A Inputs: AN0
Scan input select unused	Convert, Write Buffer 0x8
—	Sample MUX A Inputs: AN0
—	Convert, Write Buffer 0x9
MUX B Input Select	Sample MUX A Inputs: AN0
CH0SB<3:0> = n/a	Convert, Write Buffer 0xA
Mux B positive input unused	Sample MUX A Inputs: AN0
CH0NB = n/a	Convert, Write Buffer 0xB
Mux B negative input unused	Sample MUX A Inputs: AN0
—	Convert, Write Buffer 0xC
—	Sample MUX A Inputs: AN0
	Convert, Write Buffer 0xD
	Sample MUX A Inputs: AN0
	Convert, Write Buffer 0xE
	Interrupt
	Repeat

Buffer Address	Buffer @ 1st Interrupt	Buffer @ 2nd Interrupt
ADC1BUF0	AN0 sample 1	AN0 sample 16
ADC1BUF1	AN0 sample 2	AN0 sample 17
ADC1BUF2	AN0 sample 3	AN0 sample 18
ADC1BUF3	AN0 sample 4	AN0 sample 19
ADC1BUF4	AN0 sample 5	AN0 sample 20
ADC1BUF5	AN0 sample 6	AN0 sample 21
ADC1BUF6	AN0 sample 7	AN0 sample 22
ADC1BUF7	AN0 sample 8	AN0 sample 23
ADC1BUF8	AN0 sample 9	AN0 sample 24
ADC1BUF9	AN0 sample 10	AN0 sample 25
ADC1BUFA	AN0 sample 11	AN0 sample 26
ADC1BUFB	AN0 sample 12	AN0 sample 27
ADC1BUFC	AN0 sample 13	AN0 sample 28
ADC1BUFD	AN0 sample 14	AN0 sample 29
ADC1BUFE	AN0 sample 15	AN0 sample 30
ADC1BUFF		

• • •

17.5.14 Example: A/D Conversions While Scanning Through Analog Inputs

Figure 17-15 and Table 17.5.14.1 illustrate a typical setup where all available analog input channels are sampled and converted. The set CSCNA (AD1CON2<10>) bit specifies scanning of the ADC inputs. Other conditions are similar to the previous example, (see **Section 17.5.13 "Sampling a Single Channel Multiple Times"**).

Initially, the AN0 input is acquired and converted. The result is stored in the ADC1BUF buffer. Then the AN1 input is acquired and converted. This process of scanning the inputs repeats 16 times until the buffer is full and then the module generates an interrupt. Then the entire process repeats.

Figure 17-15: Scanning Through 16 Inputs 16 Samples Per Interrupt

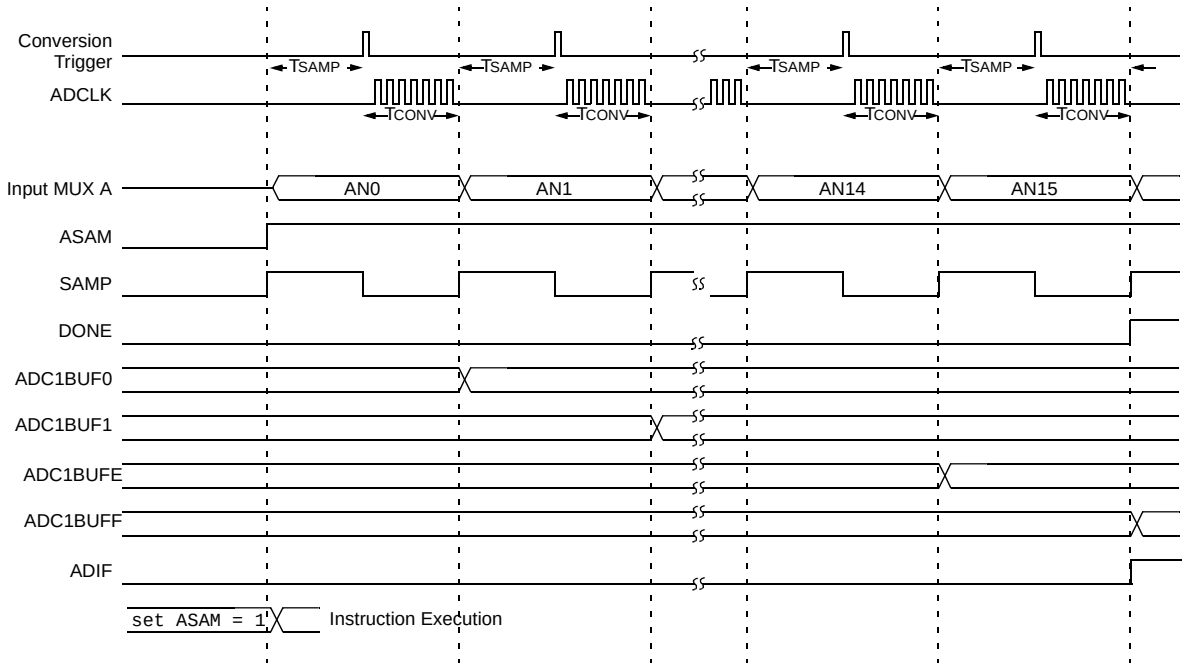


Table 17-5: Scanning Through 16 Inputs/Interrupt

CONTROL BITS	OPERATION SEQUENCE
Sequence Select	Sample MUX A Inputs: AN0
SMPI<2:0> = 1111	Convert, Write Buffer 0x0
Interrupt on 16th sample	Sample MUX A Inputs: AN1
—	Convert, Write Buffer 0x1
—	Sample MUX A Inputs: AN2
BUFM = 0	Convert, Write Buffer 0x2
Single 16-word result buffer	Sample MUX A Inputs: AN3
ALTS = 0	Convert, Write Buffer 0x3
Always use MUX A input select	Sample MUX A Inputs: AN4
MUX A Input Select	Convert, Write Buffer 0x4
CH0SA<3:0> = n/a	Sample MUX A Inputs: AN5
Overridden by CSCNA	Convert, Write Buffer 0x5
CH0NA = 0	Sample MUX A Inputs: AN6
Select VR- for MUX A negative input	Convert, Write Buffer 0x6
CSCNA = 1	Sample MUX A Inputs: AN7
Scan inputs	Convert, Write Buffer 0x7
CSSL<15:0> = 1111 1111 1111 1111	Sample MUX A Inputs: AN8
Scan input select	Convert, Write Buffer 0x8
—	Sample MUX A Inputs: AN9
—	Convert, Write Buffer 0x9
MUX B Input Select	Sample MUX A Inputs: AN10
SB<3:0> = n/a	Convert, Write Buffer 0xA
MUX B positive input unused	Sample MUX A Inputs: AN11
CH0NB = n/a	Convert, Write Buffer 0xB
MUX B negative input unused	Sample MUX A Inputs: AN12
—	Convert, Write Buffer 0xC
—	Sample MUX A Inputs: AN13
	Convert, Write Buffer 0xD
	Sample MUX A Inputs: AN14
	Convert, Write Buffer 0xE
	Sample MUX A Inputs: AN15
	Convert, Write Buffer 0xF
	Interrupt
	Repeat

Buffer Address	Buffer @ 1st Interrupt	Buffer @ 2nd Interrupt	
ADC1BUF0	AN0 sample 1	AN0 sample 17	
ADC1BUF1	AN1 sample 2	AN1 sample 18	
ADC1BUF2	AN2 sample 3	AN2 sample 19	
ADC1BUF3	AN3 sample 4	AN3 sample 20	
ADC1BUF4	AN4 sample 5	AN4 sample 21	
ADC1BUF5	AN5 sample 6	AN5 sample 22	
ADC1BUF6	AN6 sample 7	AN6 sample 23	
ADC1BUF7	AN7 sample 8	AN7 sample 24	• • •
ADC1BUF8	AN8 sample 9	AN8 sample 25	
ADC1BUF9	AN9 sample 10	AN9 sample 26	
ADC1BUFA	AN10 sample 11	AN10 sample 27	
ADC1BUFB	AN11 sample 12	AN11 sample 28	
ADC1BUFC	AN12 sample 13	AN12 sample 29	
ADC1BUFD	AN13 sample 14	AN13 sample 30	
ADC1BUFE	AN14 sample 15	AN14 sample 31	
ADC1BUFF	AN15 sample 16	AN15 sample 32	

17.5.14.1 Example: Using Dual 8-Word Buffers

Figure 17-16 and Table 17.5.14.2 demonstrate using dual 8-word buffers and alternating the buffer fill. Setting the BUFM (AD1CON2<1>) bit enables dual 8-word buffers. The BUFM setting does not affect other operational parameters. First, the conversion sequence starts filling the buffer at ADC1BUF0 (buffer location 0 x 0). After the first interrupt occurs, the buffer begins to fill at ADC1BUF8 (buffer location 0 x 8). The BUFS (AD1CON2<7>) Status bit is alternately set and cleared after each interrupt to show which buffer is being filled. In this example, three analog inputs are sampled and an interrupt occurs after every third sample.

Figure 17-16: Converting Three Inputs, Three Samples Per Interrupt Using Dual 8-Word Buffers

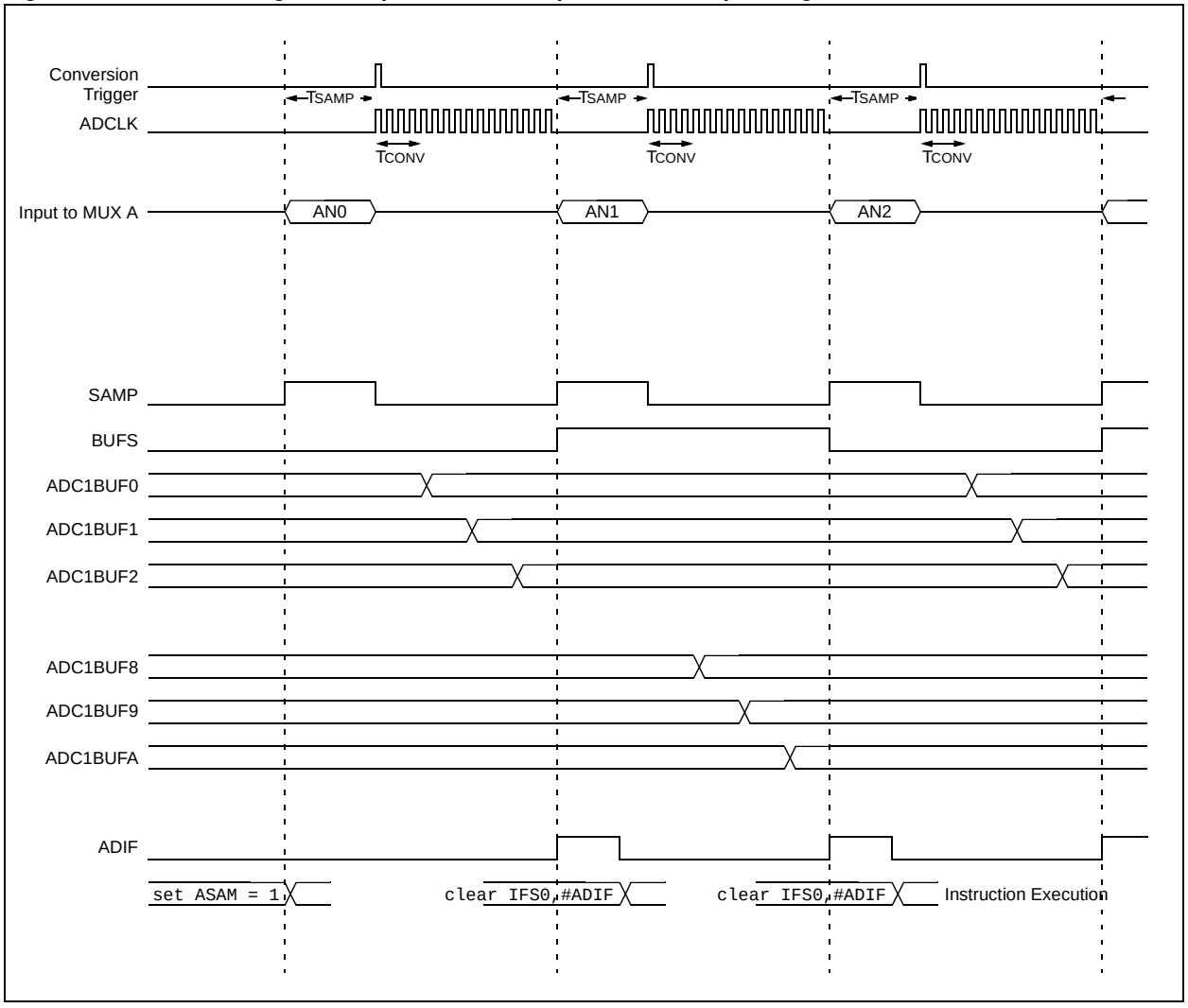


Table 17-6: Converting Three Inputs, Three Samples/Interrupt Using Dual 8-Word Buffers

CONTROL BITS Sequence Select
SMPI<2:0> = 0010 Interrupt after every third sample
—
—
BUFM = 1 Dual 8-word result buffers
ALTS = 0 Always use MUX A
MUX A Input Select
CH0SA<3:0> = n/a MUX A positive input select is not used
CH0NA = 0 Select V_{R-} for MUX A negative input
CSCNA = 1 Enable input scan
CSSL<15:0> = 0x0007 Scan input select scan list consisting of AN0, AN1, and AN2
AD1PCFG = 0X0007 Select Analog Input mode for AN0, AN1, and AN2
—
MUX B Input Select
CH0SB<3:0> = n/a MUX B positive input unused
CH0NB = n/a MUX B negative input unused
—
—

OPERATION SEQUENCE
Sample MUX A Inputs: AN0
Convert AN0, Write Buffer 0x0
Sample MUX A Inputs: AN1
Convert AN1, Write Buffer 0x1
Sample MUX A Inputs: AN2
Convert AN2, Write Buffer 0x2
Interrupt; Change Buffer
Sample MUX A Inputs: AN0
Convert AN0, Write Buffer 0x8
Sample MUX A Inputs: AN1
Convert AN1, Write Buffer 0x9
Sample MUX A Inputs: AN2
Convert AN2, Write Buffer 0xA
Interrupt; Change Buffer
Repeat

Buffer Address	Buffer @ 1st Interrupt	Buffer @ 2nd Interrupt
ADC1BUF0	AN0 sample 1	
ADC1BUF1	AN1 sample 1	
ADC1BUF2	AN2 sample 1	
ADC1BUF3		
ADC1BUF4		
ADC1BUF5		
ADC1BUF6		
ADC1BUF7		
ADC1BUF8		AN0 sample 2
ADC1BUF9		AN1 sample 2
ADC1BUFA		AN2 sample 2
ADC1BUFB		
ADC1BUFC		
ADC1BUFD		
ADC1BUFE		
ADC1BUFF		

• • •

17.5.14.2 Example: Using Alternating MUX A, MUX B Input Selections

Figure 17-17 and Table 17.5.14.3 demonstrate alternating sampling of the inputs assigned to MUX A and MUX B. Setting the ALTS (AD1CON2<0>) bit enables alternating input selections. The first sample uses the MUX A inputs specified by the CH0SA (AD1CHS<19:16>) and CH0NA (AD1CHS<23>) bits. The next sample uses the MUX B inputs specified by the CH0SB (AD1CHS<27:24>) and CH0NB (AD1CHS<31>) bits.

In the following example, one of the MUX B input specifications uses 2 analog inputs as a differential source to the sample/hold.

This example also demonstrates use of the dual 8-word buffers. An interrupt occurs after every 4th sample, which results in filling 4-words into the buffer on each interrupt.

Figure 17-17: Converting Two Analog Inputs by Alternating with Four Samples Per Interrupt

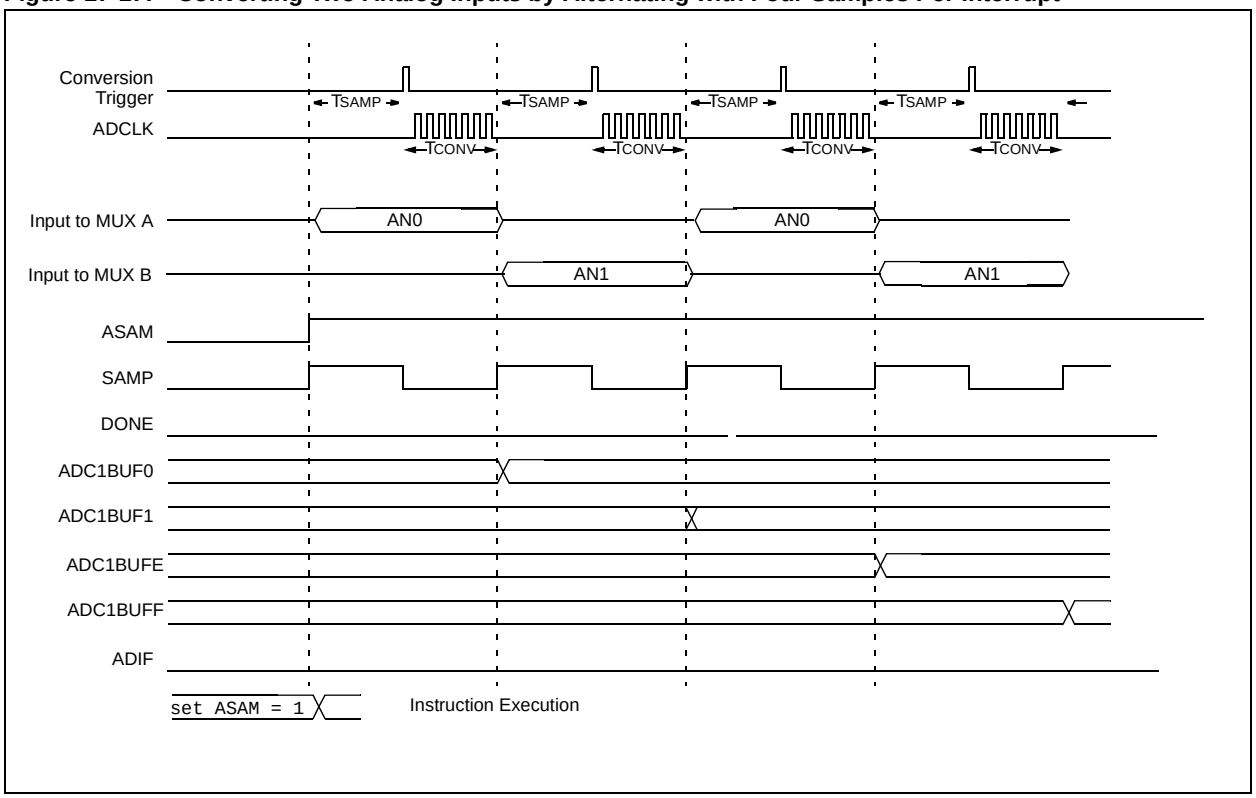


Table 17-7: Converting Two Sets of Inputs Using Alternating Input Selections

CONTROL BITS Sequence Select	OPERATION SEQUENCE
SMPI<2:0> = 0011 Interrupt on 4th sample	Sample MUX A Inputs: AN0 Convert, Write Buffer 0x0 Sample MUX B Inputs: AN1 Convert, Write Buffer 0x1 Sample MUX A Inputs: AN0 Convert, Write Buffer 0x2 Sample MUX B Inputs: AN1 Convert, Write Buffer 0x3 Interrupt; Change Buffer Sample MUX A Inputs: AN0 Convert, Write Buffer 0x8 Sample MUX B Inputs: AN1 Convert, Write Buffer 0x9 Sample MUX A Inputs: AN0 Convert, Write Buffer 0xA Sample MUX B Inputs: AN1 Convert, Write Buffer 0xB Interrupt; Change Buffer **Repeat**
—	
—	
BUFM = 1 Dual 8-word result buffers	
ALTS = 1 Alternate MUX A/B input select	
MUX A Input Select	
CH0SA<3:0> = 0000 Select AN0 for MUX A positive input	
CH0NA = 0 Select VR- for MUX A negative input	
CSCNA = 0 No input scan	
CSSL<15:0> = n/a Scan input select unused	
—	
—	
MUX B Input Select	
CH0SB<3:0> = 0001 Select AN1 for MUX B positive input	
CH0NB = 0 Select VR- for MUX B negative input	
—	
—	

Buffer Address	Buffer @ 1st Interrupt	Buffer @ 2nd Interrupt
ADC1BUF0	AN0 sample 1	
ADC1BUF1	AN1 sample 1	
ADC1BUF2	AN0 sample 2	
ADC1BUF3	AN1 sample 2	
ADC1BUF4		
ADC1BUF5		
ADC1BUF6		
ADC1BUF7		
ADC1BUF8		AN0 sample 3
ADC1BUF9		AN1 sample 3
ADC1BUFA		AN0 sample 4
ADC1BUFB		AN1 sample 4
ADC1BUFC		
ADC1BUFD		
ADC1BUFE		
ADC1BUFF		

• • •

17.5.14.3 Example: Converting Three Analog Inputs Using Alternating Sample Mode and a Scan List

Figure 17-18, Figure 17-19, and Table 17-8 demonstrate sampling by scanning through inputs and alternating between MUX A and MUX B. When the Alternating Sample mode is selected, the first input to be sampled will be the input selected for MUX A, the second sample will be the input selected for MUX B. Then the process repeats. When scanning is combined with Alternating Input mode, the positive input to MUX A is selected by the contents of the AD1CSSL register, not CH0SA. For each sample that MUX A is selected the next item in the scan list is sampled. The positive input to MUX B is selected by CH0SB (AD1CHS<27:24>).

When ASAM (AD1CON1<2>) is clear, sampling will not resume after conversion completion, but will occur when setting the SAMP (AD1CON1<1>) bit.

Figure 17-18: Converting Three Analog Inputs Using Alternating Sample Mode and a Scan List

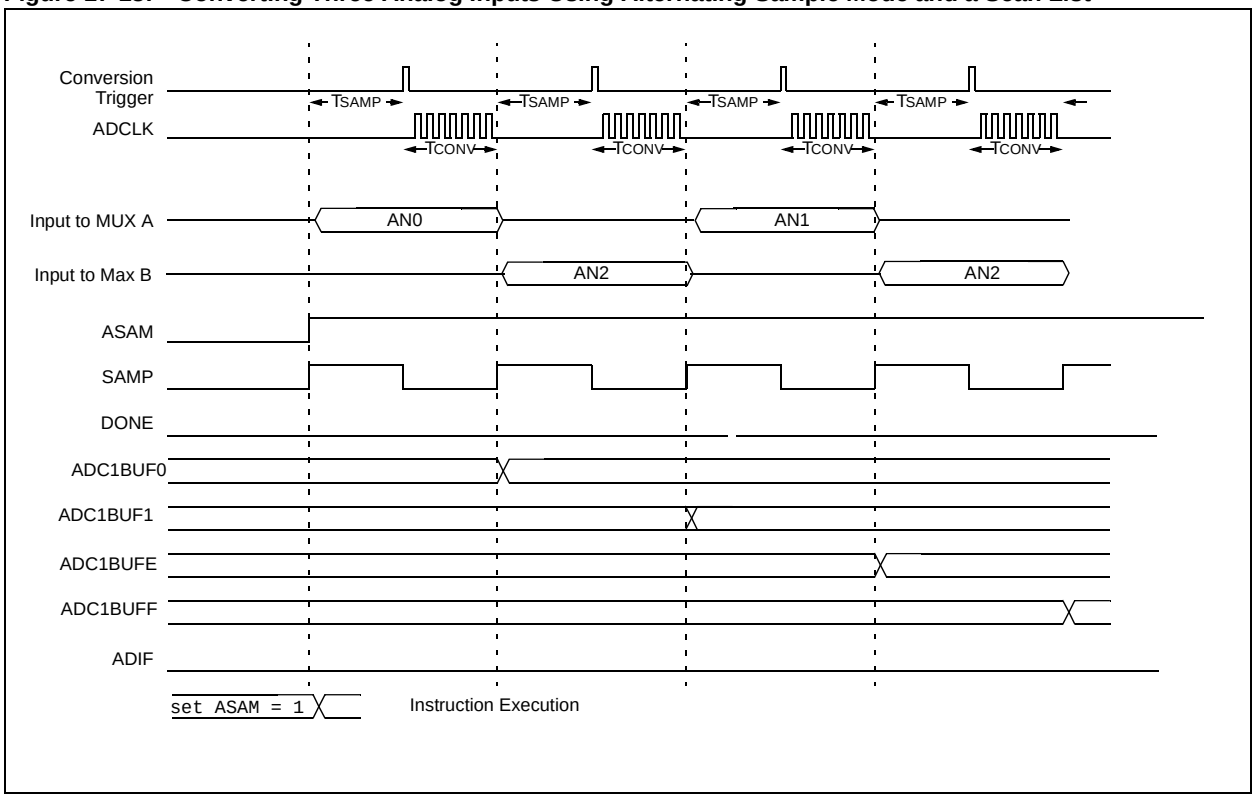


Figure 17-19: 10-Bit High-Speed A/D Converter Block Diagram For Alternating Sample and Scan

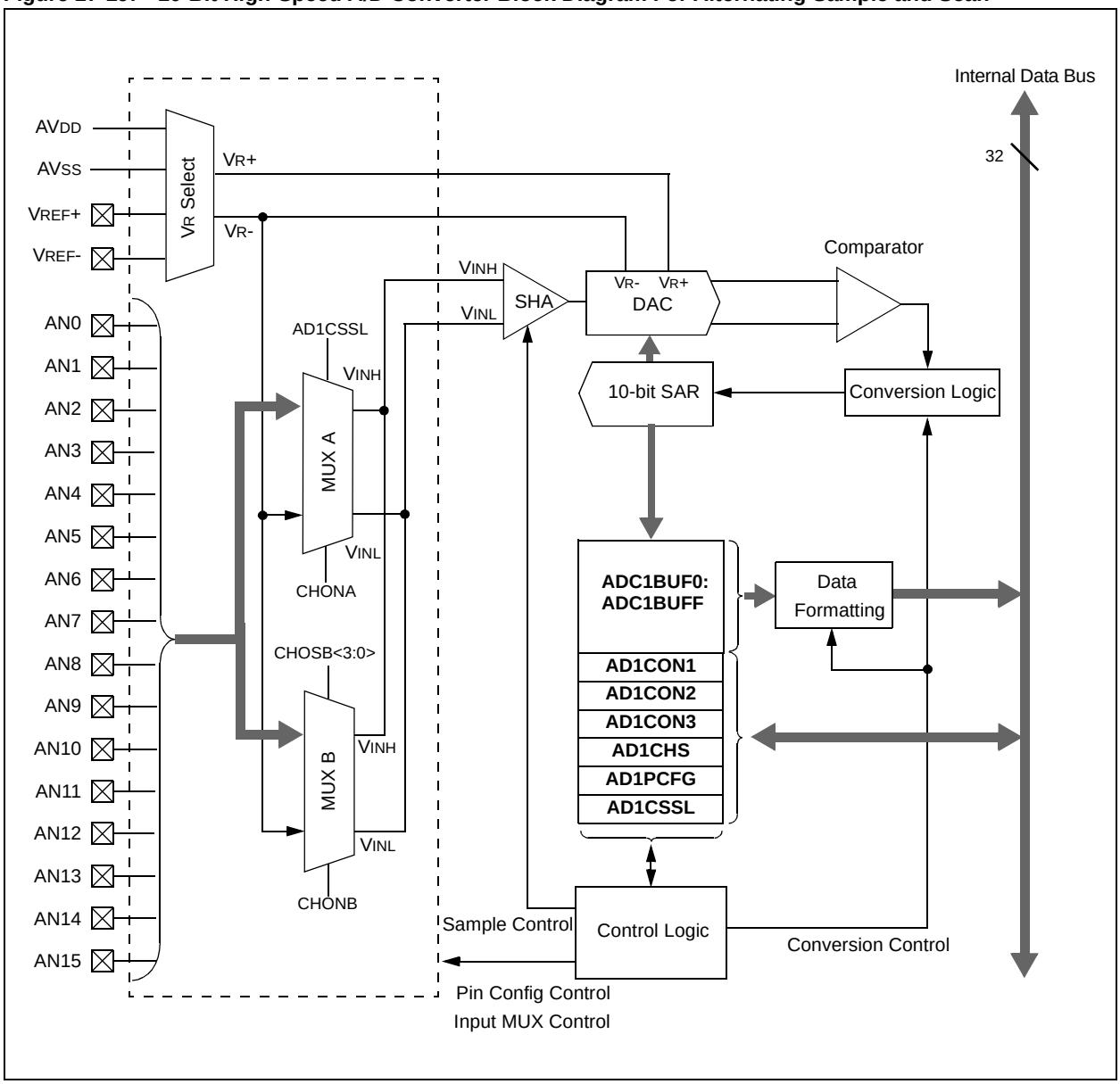


Table 17-8: Sampling Eight Inputs Using Sequential Sampling

CONTROL BITS Sequence Select	OPERATION SEQUENCE
SMPI<2:0> = 0011 Interrupt on 4th sample	Sample: AN0
—	Convert, Write Buffer 0x0
—	Sample: AN2
BUFM = 0 Single 16-word result buffer	Convert, Write Buffer 0x1
ALTS = 1 Alternate MUX A/B input select	Sample: AN1
MUX A Input Select	Convert, Write Buffer 0x2
CH0SA<3:0> = n/a Not used	Sample: AN2
CH0NA = 0 Select V_R- for CH0- input	Convert, Write Buffer 0x3
CSCNA = 1 Enable input scan	
CSSL<15:0> = n/a Scan input select scan list consisting of AN0 and AN1	Interrupt
—	**Repeat**
—	
MUX B Input Select	
CH0SB<3:0> = 0010 Select AN7 for CH0+ input	
CH0NB = 0 Select V_R- for CH0- input	
—	
—	

Buffer Address	Buffer @ 1st Interrupt	Buffer @ 2nd Interrupt
ADC1BUF0	AN0 sample 1	AN0 sample 5
ADC1BUF1	AN2 sample 2	AN2 sample 6
ADC1BUF2	AN1 sample 3	AN1 sample 7
ADC1BUF3	AN2 sample 4	AN2 sample 8
ADC1BUF4		
ADC1BUF5		
ADC1BUF6		
ADC1BUF7		
ADC1BUF8		
ADC1BUF9		
ADC1BUFA		
ADC1BUFB		
ADC1BUFC		
ADC1BUFD		
ADC1BUFE		
ADC1BUFF		

• • •

17.6 INITIALIZATION

A simple initialization code example for the ADC module is provided in Example 17-6.

In this particular configuration, all 16 analog input pins, AN0-AN15, are set up as analog inputs. Operation in IDLE mode is disabled, output data is in unsigned fractional format, and AVDD and AVSS are used for VR+ and VR-. The start of acquisition, as well as start of conversion (conversion trigger), are performed manually in software. The CH0 SHA is used for conversions. Scanning of inputs is disabled, and an interrupt occurs after every acquisition/convert sequence (1 conversion result). The ADC conversion clock is TPB/2.

Since acquisition is started manually by setting the SAMP bit (AD1CON1<1>) after each conversion is complete, the auto-sample time bits, SAMC<4:0> (AD1CON3<12:8>), are ignored. Moreover, since the start of conversion (i.e., end of acquisition) is also triggered manually, the SAMP bit needs to be cleared each time a new sample needs to be converted.

Example 17-6: ADC Initialization Code Example

```
AD1PCFG = 0x0000;             /* Configure ADC port
                                 all input pins are analog */

AD1CON1 = 0x2208;             /* Configure sample clock source and Conversion Trigger mode.
                                 Unsigned Fractional format, Manual conversion trigger,
                                 Manual start of sampling, Simultaneous sampling,
                                 No operation in IDLE mode.  */

AD1CON2 = 0x0000;             /* Configure ADC voltage reference
                                  and buffer fill modes.
                                 VREF from AVDD and AVSS,
                                 Inputs are not scanned,
                                 Interrupt every sample */

AD1CON3 = 0x0000;             /* Configure ADC conversion clock */

AD1CHS = 0x0000;              /* Configure input channels,
                                 CH0+ input is AN0.
                                 CHO- input is VREFL (AVss)

AD1CSSL = 0x0000;             /* No inputs are scanned.
                                 Note: Contents of AD1CSSL are ignored when CSCNA = 0 */

IFS1CLR = 2;                  /*Clear ADC conversion interrupt*/

// Configure ADC interrupt priority bits (AD1IP<2:0>) here, if
// required. (default priority level is 4)

IEC1SET = 2;                  /* Enable ADC conversion interrupt*/

AD1CON1SET = 0x8000;          /* Turn on the ADC module */
AD1CON1SET = 0x0002;          /* Start sampling the input */
DelayNmSec(100);             /* Ensure the correct sampling time has elapsed before
                                 starting a conversion.*/

AD1CON1CLR = 0x0002;          /* End Sampling and start Conversion*/
      :                       /* The DONE bit is set by hardware when the convert sequence
                                 is finished. */
      :                       /* The ADIF bit will be set. */
```

Example 17-7: Converting 1 Channel at 400 ksps, Auto-Sample Start, 2 TAD Sampling Time Code Example

```
AD1PCFG = 0xFFFB;                       //  all PORTB = Digital; RB2 = analog
AD1CON1 = 0x00E0;                       //  SSRC bit = 111 implies internal
                                        //  counter ends sampling and starts
                                        //  converting.
AD1CHS  = 0x00020000;                   //  Connect RB2/AN2 as CH0 input
                                        //  in this example RB2/AN2 is the input
AD1CSSL = 0;
AD1CON3 = 0x0203;                       //  Sample time = 2Tad

AD1CON2 = 0x6004;                       //  Select external VREF+ and VREF- pins
                                        //  Interrupt after every 2 samples
AD1CON1bits.ADON = 1;                   //  turn ADC ON
while (1)                               //  repeat continuously
{
  ADCValue = 0;                             //  clear value
  ADC16Ptr = &ADC1BUF0;                     //  initialize ADC1BUF0 pointer
  IF1bits.AD1IF = 0;                        //  clear ADC interrupt flag
  AD1CON1bits.ASAM = 1;                     //  auto start sampling
                                            //  for 31Tad then go to conversion
  while (!IFS0bits.ADIF);                   //  conversion done?
  AD1CON1bits.ASAM = 0;                     //  yes then stop sample/convert
  for (count = 0; count <2; count++)
  {                                         //  average the two
       ADCValue = ADCValue + *ADC16Ptr++;
       ADCValue = ADCValue >> 1;
  }
}                                           //  repeat
```


17.7 INTERRUPTS

The ADC has a dedicated interrupt bit AD1IF and a corresponding interrupt enable/mask bit AD1IE. These bits are used to determine the source of an interrupt and to enable or disable an individual interrupt source. The priority level of each of the channels can also be set independently of the other channels.

The AD1IF is set when the condition set by the Samples Per Interrupt bit SMPI<3:0> (AD1CON2<5:2>) is met. The AD1IF bit will then be set without regard to the state of the corresponding AD1IE bit. The AD1IF bit can be polled by software if desired.

The AD1IE bit controls the interrupt generation. If the AD1xIE bit is set, the CPU will be interrupted whenever an event defined by SMPI<3:0> occurs and the corresponding AD1IF bit will be set (subject to the priority and sub priority as outlined below).

It is the responsibility of the routine that services a particular interrupt to clear the appropriate Interrupt Flag bit before the service routine is complete.

The priority of the ADC interrupt can be set independently via the AD1IP<2:0> (IPC6<28:26>) bits. This priority defines the priority group that interrupt source will be assigned to. The priority groups range from a value of 7, the highest priority, to a value of 0, which does not generate an interrupt. An interrupt being serviced will be preempted by an interrupt in a higher priority group.

The subpriority bits allow setting the priority of a interrupt source within a priority group. The values of the subpriority, AD1xIS<1:0> (IPC6<25:24>), range from 3, the highest priority, to 0 the lowest priority. An interrupt with the same priority group but having a higher subpriority value will preempt a lower subpriority interrupt that is in progress.

The priority group and subpriority bits allow more than one interrupt source to share the same priority and subpriority. If simultaneous interrupts occur in this configuration the natural order of the interrupt sources within a priority/subgroup pair determine the interrupt generated. The natural priority is based on the vector numbers of the interrupt sources. The lower the vector number the higher the natural priority of the interrupt. Any interrupts that were overridden by natural order will then generate their respective interrupts based on priority, subpriority, and natural order after the interrupt flag for the current interrupt is cleared.

After an enabled interrupt is generated, the CPU will jump to the vector assigned to that interrupt (refer to Table 17-9). The vector number for the interrupt is the same as the natural order number. The IRQ number is not always the same as the vector number due to some interrupts sharing a single vector. The CPU will then begin executing code at the vector address. The users code at this vector address should perform an operations required, such as reloading the duty cycle, clear the interrupt flag, and then exit. Refer to **Section 8. "Interrupts"** for vector address table details and for more information on interrupts.

Table 17-9: ADC Interrupt Vectors for Various Offsets with EBASE = 0x8000:0000

Interrupt	Vector/Natural Order	IRQ Number	Vector Address IntCtl.VS = 0x01	Vector Address IntCtl.VS = 0x02	Vector Address IntCtl.VS = 0x04	Vector Address IntCtl.VS = 0x08	Vector Address IntCtl.VS = 0x10
ADC	27	32	8000 0560	8000 08C0	8000 0F80	8000 1D00	8000 3800

Example 17-8: ADC Interrupt Configuration Code Example

```
IPS6SET = 0x0014;                       //  Set Priority to 5
IPS6SET = 0x0003;                       //  Set Sub Priority to 3
                                        //
IFS1CLR = 0x0002;                       // Ensure the interrupt flag is clear
IEC1SET = 0x0002;                       // Enable ADC interrupts
```

17.8 I/O PIN CONTROL

The pins used for analog input can also be used for digital I/O. Configuring a pin for analog input requires three steps. Any digital peripherals that share the desired pin must be disabled. The pin must be configured as a Digital input, by setting the corresponding TRIS bit to a '1', to disable the output driver. Then the pin must be placed in Analog mode by setting the corresponding bit in the AD1PCFG register.

Table 17-10: Pins Associated with the ADC Module

Pin Name	Module Control	Controlling Bit Field	Pin Type	Buffer Type	TRIS	Description
AN0	ON	AD1PCFG<0>	A	—	Input	Analog input
AN1	ON	AD1PCFG<1>	A	—	Input	Analog input
AN2	ON	AD1PCFG<2>	A	—	Input	Analog input
AN3	ON	AD1PCFG<3>	A	—	Input	Analog input
AN4	ON	AD1PCFG<4>	A	—	Input	Analog input
AN5	ON	AD1PCFG<5>	A	—	Input	Analog input
AN6	ON	AD1PCFG<6>	A	—	Input	Analog input
AN7	ON	AD1PCFG<7>	A	—	Input	Analog input
AN8	ON	AD1PCFG<8>	A	—	Input	Analog input
AN9	ON	AD1PCFG<9>	A	—	Input	Analog input
AN10	ON	AD1PCFG<10>	A	—	Input	Analog input
AN11	ON	AD1PCFG<11>	A	—	Input	Analog input
AN12	ON	AD1PCFG<12>	A	—	Input	Analog input
AN13	ON	AD1PCFG<13>	A	—	Input	Analog input
AN14	ON	AD1PCFG<14>	A	—	Input	Analog input
AN15	ON	AD1PCFG<15>	A	—	Input	Analog input
$V_{REF}+$	ON	AD1CON2<15:13>	P	—	—	Positive voltage reference
$V_{REF}-$	ON	AD1CON2<15:13>	P	—	—	Negative voltage reference

Legend:	ST = Schmitt Trigger input with CMOS levels	A = Analog
	I = Input	P = Power
	O = Output	

17.9 OPERATION DURING SLEEP AND IDLE MODES

Note:	In this manual, a distinction is made between a power mode as it is used in a specific module, and a power mode as it is used by the device, e.g., Sleep mode of the Comparator and SLEEP mode of the CPU. To indicate which type of power mode is intended, uppercase and lowercase letters (Sleep, Idle, Debug) signify a module power mode, and all uppercase letters (SLEEP, IDLE, DEBUG) signify a device power mode.

SLEEP and IDLE modes are useful for minimizing conversion noise because the digital activity of the CPU, buses and other peripherals is minimized.

17.9.1 CPU SLEEP Mode Without RC ADC Clock

When the device enters SLEEP mode, all clock sources to the module are shut down and stay at logic '0'.

If SLEEP occurs in the middle of a conversion, the conversion is aborted unless the ADC is clocked from its internal RC clock generator. The converter will not resume a partially completed conversion on exiting from SLEEP mode.

ADC register contents are not affected by the device entering or leaving SLEEP mode.

17.9.2 CPU SLEEP Mode With RC ADC Clock

The ADC module can operate during SLEEP mode if the ADC clock source is set to the internal ADC RC oscillator (ADRC = 1). This reduces the digital switching noise from the conversion. When the conversion is completed, the DONE bit will be set and the result loaded into the ADC result buffer, ADC1BUF.

If the ADC interrupt is enabled (AD1IE = 1), the device will wake up from SLEEP when the ADC interrupt occurs. Program execution will resume at the ADC Interrupt Service Routine if the ADC interrupt is greater than the current CPU priority. Otherwise, execution will continue from the instruction after the WAIT instruction that placed the device in SLEEP mode.

If the ADC interrupt is not enabled, the ADC module will then be disabled, although the ON bit will remain set.

To minimize the effects of digital noise on the ADC module operation, the user should select a conversion trigger source that ensures the A/D conversion will take place in SLEEP mode. The automatic conversion trigger option can be used for sampling and conversion in SLEEP (SSRC<2:0> = 111). To use the automatic conversion option, the ADC ON bit should be set in the instruction prior to the WAIT instruction.

Note:	For the ADC module to operate in SLEEP mode, the ADC clock source must be set to RC (ADRC = 1).

17.9.3 ADC Operation During CPU IDLE Mode

For the A/D converter, the ADC SIDL bit (AD1CON1<13>) selects if the module will stop on IDLE or continue on IDLE. If ADC SIDL = 0, the module will continue normal operation when the device enters IDLE mode. If the ADC interrupt is enabled (AD1IE = 1), the device will wake up from IDLE mode when the ADC interrupt occurs. Program execution will resume at the ADC Interrupt Service Routine if the ADC interrupt is greater than the current CPU priority. Otherwise, execution will continue from the instruction after the WAIT instruction that placed the device in IDLE mode.

If ADC SIDL = 1, the module will stop in IDLE mode. If the device enters IDLE mode in the middle of a conversion, the conversion is aborted. The converter will not resume a partially completed conversion on exiting from IDLE mode.

17.9.4 Effects of Freeze on ADC Operation

If Freeze mode is entered while the ADC is performing a conversion the result of the conversion will be lost.

While in Freeze mode the ADC registers can be read. Any writes to the ADC register while in Freeze mode will not take effect until the device exits Freeze mode.

Note:	The FRZ bit is readable and writable only when the CPU is executing in Debug Exception mode. In all other modes, the FRZ bit reads as '0'. If FRZ bit is changed during DEBUG mode, the new value does not take effect until the current Debug Exception mode is exited and re-entered. During the Debug Exception mode, the FRZ bit reads the state of the peripheral when entering DEBUG mode.

17.10 EFFECTS OF VARIOUS RESETS

17.10.1 $\overline{\text{MCLR}}$ Reset

Following a $\overline{\text{MCLR}}$ event all the ADC control registers (AD1CON1, AD1CON2, AD1CON3, AD1CHS, AD1PCFG, and AD1CSSL) are reset to a value of 0x00000000. This disables the ADC and sets the analog input pins to Analog Input mode. Any conversion that was in progress will terminate and the result will not be written to the result buffer.

The values in the ADC1BUF registers are initialized during a $\overline{\text{MCLR}}$ Reset. ADC1BUF0...ADC1BUFF will contain 0x00000000.

17.10.2 Power-on Reset

Following a POR event all the ADC control registers (AD1CON1, AD1CON2, AD1CON3, AD1CHS, AD1PCFG, and AD1CSSL) are reset to a value of 0x00000000. This disables the ADC and sets the analog input pins to Analog Input mode.

The values in the ADC1BUF registers are initialized during a Power-on Reset. ADC1BUF0...ADC1BUFF will contain 0x00000000.

17.10.3 Watchdog Timer Reset

Following a Watchdog Timer (WDT) Reset all the ADC control registers (AD1CON1, AD1CON2, AD1CON3, AD1CHS, AD1PCFG, and AD1CSSL) are reset to a value of 0x00000000. This disables the ADC and sets the analog input pins to Analog Input mode. Any conversion that was in progress will terminate and the result will not be written to the result buffer.

The values in the ADC1BUF registers are initialized after a WDT Reset.

ADC1BUF0...ADC1BUFF will contain 0x00000000.

17.11 DESIGN TIPS

Question 1: ***How can I optimize the system performance of the A/D converter?***

Answer: The following tips can be helpful for optimizing performance:

1. Make sure you are meeting all of the timing specifications. If you are turning the module off and on, there is a minimum delay you must wait before taking a sample. If you are changing input channels, there is a minimum delay you must wait for this as well. Also, there is TAD, which is the time selected for each bit conversion. This is selected in AD1CON3 and should be within a certain range as specified in the Electrical Characteristics. If TAD is too short, the result may not be fully converted before the conversion is terminated, and if TAD is made too long, the voltage on the sampling capacitor can decay before the conversion is complete. These timing specifications are provided in the "Electrical Specifications" section of the device data sheets.
2. Often the source impedance of the analog signal is high (greater than 10 kΩ), so the current drawn from the source to charge the sample capacitor can affect accuracy. If the input signal does not change too quickly, try putting a 0.1 μF capacitor on the analog input. This capacitor will charge to the analog voltage being sampled and supply the instantaneous current needed to charge the 4.4 pF internal holding capacitor.
3. Put the device into SLEEP mode before the start of the A/D conversion. The RC clock source selection is required for conversions in SLEEP mode. This technique increases accuracy because digital noise from the CPU and other peripherals is minimized.

Question 2: ***Do you know of a good reference on ADCs?***

Answer: The following handbook can assist with a good understanding of A/D conversions:

Analog Devices, Inc., and Scheingold, D. H., ed. *Analog-Digital Conversion Handbook.* 3rd ed., Englewood Cliffs, NJ: Prentice Hall, 1986. ISBN 0-13-032848-0.

Question 3: ***My combination of channels/sample and samples/interrupt is greater than the size of the buffer. What will happen to the buffer?***

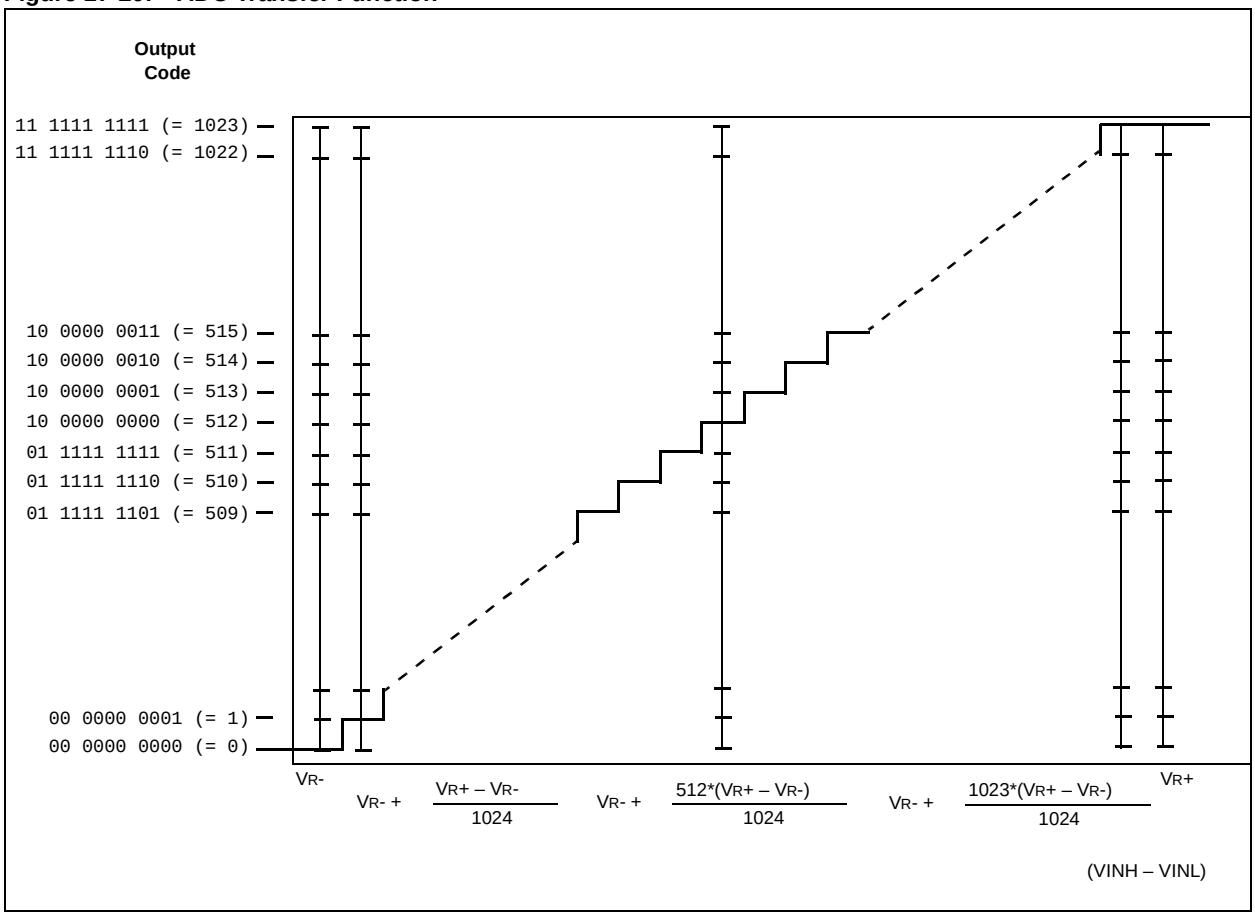
Answer: This configuration is not recommended. The buffer will contain the results of the first 16 samples (or 8, if a Dual Buffer mode is used) in the conversion sequence. The remaining items in the conversion sequence will be ignored.

17.11.1 Transfer Function

The ideal transfer function of the A/D converter is shown in Figure 17-20. The difference of the input voltages, (V_{INH} – V_{INL}), is compared to the reference, (V_{R+} – V_{R-}).

- The first code transition occurs when the input voltage is (V_{R+} – V_{R-L}/2048) or 0.5 LSb.
- The `00 0000 0001` code is centered at (V_{R+} – V_{R-}/1024) or 1.0 LSb.
- The `10 0000 0000` code is centered at (512*(V_{R+} – V_{R-})/1024).
- An input voltage less than (1 × (V_{R+} – V_{R-L})/2048) converts as `00 0000 0000`.
- An input greater than (2045 × (V_{R+} – V_{R-})/2048) converts as `11 1111 1111`.

Figure 17-20: ADC Transfer Function

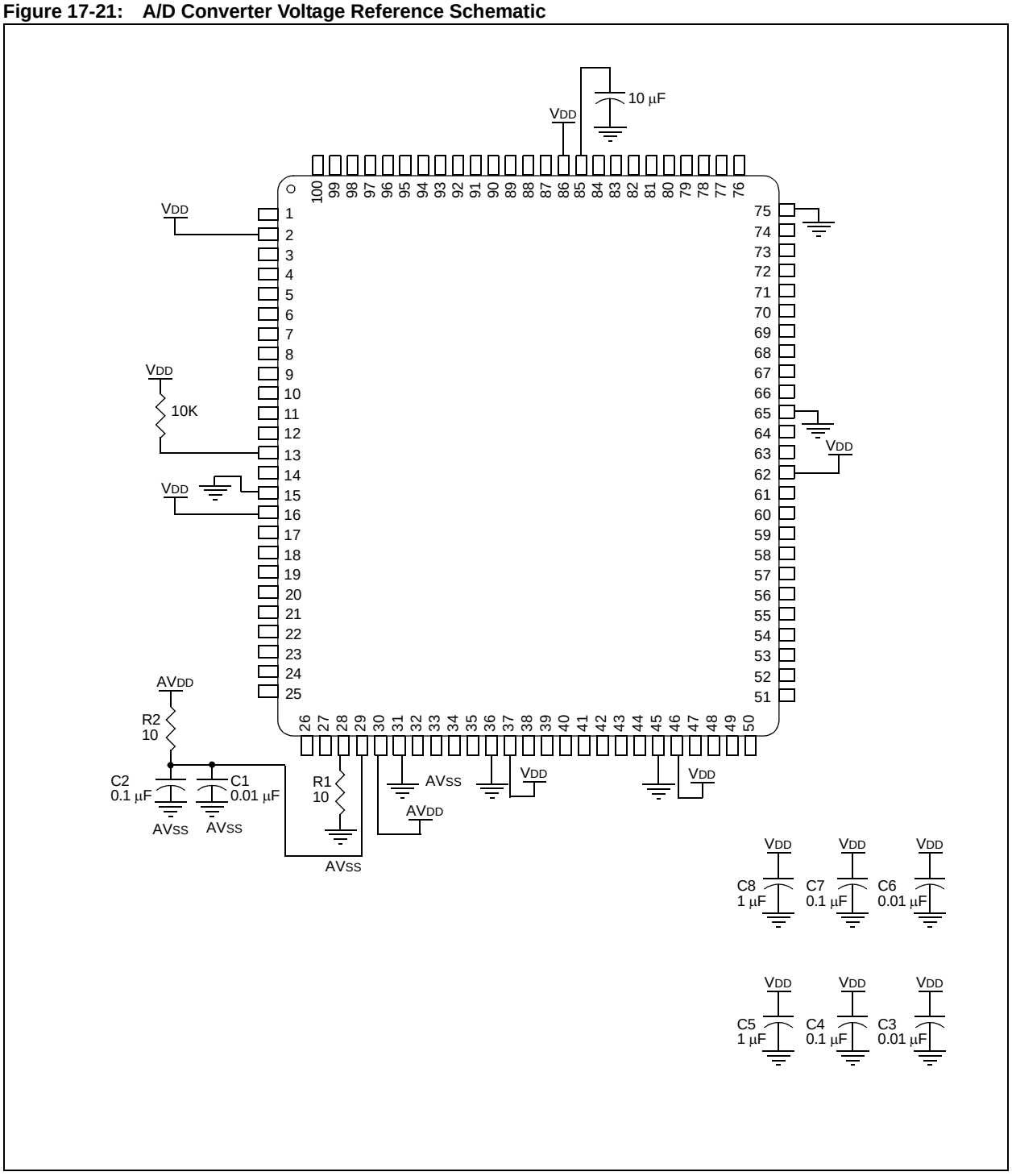


17.11.2 ADC Accuracy/Error

Refer to **Section 17.12 "Related Application Notes"** for a list of documents that discuss ADC accuracy.

The following figure depicts the recommended circuit for the conversion rates above 400 ksps. The PIC32MX is shown as an example.

Figure 17-21: A/D Converter Voltage Reference Schematic



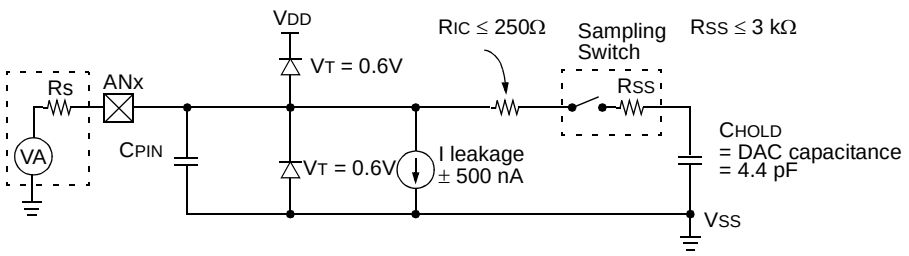
17.11.3 ADC Sampling Requirements

The analog input model of the 10-bit A/D converter is shown in Figure 17-22. The total acquisition time for the A/D conversion is a function of the internal amplifier settling time and the holding capacitor charge time.

For the A/D converter to meet its specified accuracy, the charge holding capacitor (CHOLD) must be allowed to fully charge to the voltage level on the analog input pin. The analog output source impedance (RS), the interconnect impedance (RIC), and the internal sampling switch (RSS) impedance combine to directly affect the time required to charge the CHOLD. The combined impedance of the analog sources must therefore be small enough to fully charge the holding capacitor within the chosen sample time. After the analog input channel is selected (changed), this acquisition function must be completed prior to starting the conversion. The internal holding capacitor will be in a discharged state prior to each sample operation.

At least 1 TAD time period should be allowed between conversions for the acquisition time. For more details, see the device electrical specifications.

Figure 17-22: 10-Bit A/D Converter Analog Input Model



Note: CPIN value depends on device package and is not tested. Effect of CPIN negligible if Rs ≤ 5 kΩ.

Legend:

CPIN = input capacitance

VT = threshold voltage

RSS = sampling switch resistance

RIC = interconnect resistance

RS = source resistance

CHOLD = sample/hold capacitance

I leakage = leakage current at the pin due to various junctions

17.11.4 Connection Considerations

Since the analog inputs employ ESD (Electrostatic Discharge) protection, they have diodes to VDD and VSS. This requires that the analog input must be between VDD and VSS. If the input voltage exceeds this range by greater than 0.3V (either direction), one of the diodes becomes forward biased and it may damage the device if the input current specification is exceeded.

An external RC filter is sometimes added for anti-aliasing of the input signal. The R component should be selected to ensure that the acquisition time requirements are satisfied. Any external components connected (via high-impedance) to an analog input pin (capacitor, Zener diode, etc.) should have very little leakage current at the pin.

17.12 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the PIC32MX device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the 10-bit A/D Converter module are:

Title	Application Note #
Using the Analog-to-Digital (A/D) Converter	AN546
Four Channel Digital Voltmeter with Display and Keyboard	AN557
Understanding A/D Converter Performance Specifications	AN693
Using the dsPIC30F for Sensorless BLDC Control	AN901
Using the dsPIC30F for Vector Control of an ACIM	AN908
Sensored BLDC Motor Control Using the dsPIC30F2010	AN957
An Introduction to AC Induction Motor Control Using the dsPIC30F MCU	AN984

Note: Please visit the Microchip web site (www.microchip.com) for additional Application Notes and code examples for the PIC32MX family of devices.

17.13 REVISION HISTORY

Revision A (October 2007)

This is the initial released version of this document.

Revision B (October 2007)

Updated document to remove Confidential status.

Revision C (April 2008)

Revised status to Preliminary; Revised U-0 to r-x.

Revision D (June 2008)

Revised Register 17-1 note; Revised Registers 17-13, 17-17, 17-21, 17-25, 17-26; Revised Equation 17-1; Added Section 17.5.6; Revised Tables 17-4, 17-5, 17-6, 17-7, 17-8; Delete Section 17.11.5 (500 KSPS Configuration Guideline); Change Reserved bits from “Maintain as” to “Write”; Added Note to ON bit (AD1CON1 Register).

Section 18. Reserved for Future

NOTES:

Section 19. Comparator

HIGHLIGHTS

This section of the manual contains the following topics:

19.1 INTRODUCTION

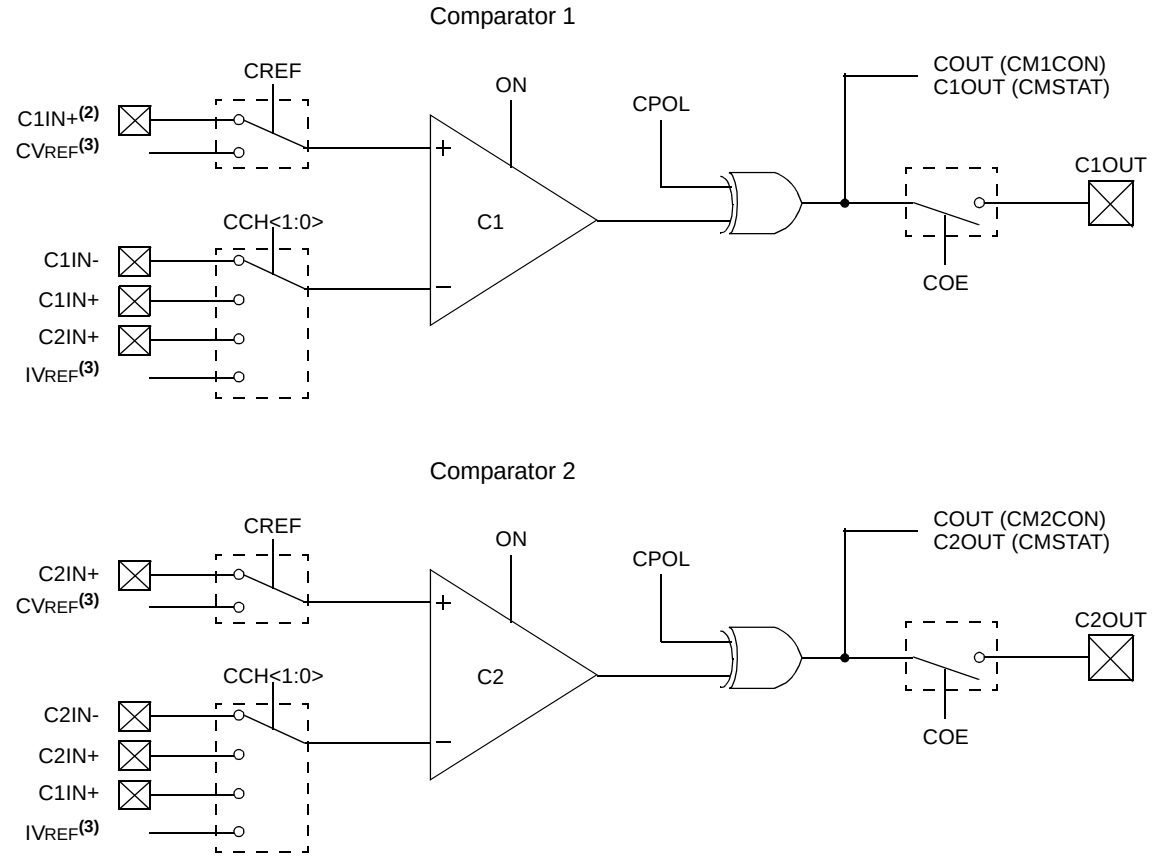
The PIC32MX Analog Comparator module contains one or more comparator(s) that can be configured in a variety of ways.

Following are some of the key features of this module:

- Selectable inputs available include:
  - Analog inputs multiplexed with I/O pins
  - On-Chip Internal Absolute Voltage Reference (IVREF)
  - Comparator Voltage Reference (CVREF)
- Outputs can be inverted
- Selectable interrupt generation

A block diagram of the comparator module is shown in Figure 19-1.

Figure 19-1: Comparator Block Diagram



Note 1: IVref is the internal 1.2V reference.

2: On USB variants, when USB is enabled, this pin is controlled by the USB module and therefore is not available as a comparator input.

3: Internally connected.

19.2 COMPARATOR CONTROL REGISTERS

Note: Each PIC32MX device variant may have one or more Comparator modules. An 'x' used in the names of pins, control/Status bits and registers denotes the particular module. Refer to the specific device data sheets for more details.

A Comparator module consists of the following Special Function Registers (SFRs):

- CMxCON: Comparator Control Register for Module 'x'
- CMxCONCLR, CMxCONSET, CMxCONINV: Atomic Bit Write-only Manipulation Registers for CMxCON
- CMSTAT: Comparator Status Register
- CMSTATCLR, CMSTATSET, CMSTATINV: Atomic Bit Write-only Manipulation Registers for CMSTAT

The Comparator module also has the following interrupt control registers:

- IFS1: Interrupt Flag Status Register
- IFS1CLR, IFS1SET, IFS1INV: Atomic Bit Manipulation Write-only Registers for IFS1
- IEC1: Interrupt Enable Control Register
- IEC1CLR, IEC1SET, IEC1INV: Atomic Bit Manipulation Write-only Registers for IEC1
- IPC7: Interrupt Priority Control Register
- IPC7CLR, IPC7SET, IPC7INV: Atomic Bit Write-only Manipulation Registers for IPC7

The following table provides a brief summary of all Comparator-related registers. Corresponding registers appear after the summary, followed by a detailed description of each register.

Table 19-1: Comparator SFRs Summary

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
CM1CON	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	ON	COE	CPOL	—	—	—	—	COUT
	7:0	EVPOL<1:0>		—	CREF	—	—	CCH<1:0>	
CM1CONCLR	31:0	Write clears selected bits in CM1CON, read yields undefined value							
CM1CONSET	31:0	Write sets selected bits in CM1CON, read yields undefined value							
CM1CONINV	31:0	Write inverts selected bits in CM1CON, read yields undefined value							
CM2CON	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	ON	COE	CPOL	—	—	—	—	COUT
	7:0	EVPOL<1:0>		—	CREF	—	—	CCH<1:0>	
CM2CONCLR	31:0	Write clears selected bits in CM2CON, read yields undefined value							
CM2CONSET	31:0	Write sets selected bits in CM2CON, read yields undefined value							
CM2CONINV	31:0	Write inverts selected bits in CM2CON, read yields undefined value							
CMSTAT	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	FRZ	SIDL	—	—	—	—	—
	7:0	—	—	—	—	—	—	C2OUT	C1OUT
CMSTATCLR	31:0	Write clears selected bits in CMSTAT, read yields undefined value							
CMSTATSET	31:0	Write sets selected bits in CMSTAT, read yields undefined value							
CMSTATINV	31:0	Write inverts selected bits in CMSTAT, read yields undefined value							

Table 19-1: Comparator SFRs Summary (Continued)

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
IFS1	31:24	—	—	—	—	—	—	USBIF	FCEIF
	23:16	—	—	—	—	DMA3IF	DMA2IF	DMA1IF	DMA0IF
	15:8	RTCCIF	FSCMIF	I2C2MIF	I2C2SIF	I2C2BIF	U2TXIF	U2RXIF	U2EIF
	7:0	SPI2RXIF	SPI2TXIF	SPI2EIF	CMP2IF	CMP1IF	PMPIF	AD1IF	CNIF
IFS1CLR	31:0	Clears the selected bits in IFS1, read yields undefined value							
IFS1SET	31:0	Sets the selected bits in IFS1, read yields undefined value							
IFS1INV	31:0	Inverts the selected bits in IFS1, read yields undefined value							
IEC1	31:24	—	—	—	—	—	—	USBIE	FCEIE
	23:16	—	—	—	—	DMA3IE	DMA2IE	DMA1IE	DMA0IE
	15:8	RTCCIE	FSCMIE	I2C2MIE	I2C2SIE	I2C2BIE	U2TXIE	U2RXIE	U2EIE
	7:0	SPI2RXIE	SPI2TXIE	SPI2EIE	CMP2IE	CMP1IE	PMPIE	AD1IE	CNIE
IEC1CLR	31:0	Clears the selected bits in IEC1, read yields undefined value							
IEC1SET	31:0	Sets the selected bits in IEC1, read yields undefined value							
IEC1INV	31:0	Inverts the selected bits in IEC1, read yields undefined							
IPC7	31:24	—	—	—	SPI2IP<2:0>			SP2IS<1:0>	
	23:16	—	—	—	CMP2IP<2:0>			CMP2IS<1:0>	
	15:8	—	—	—	CMP1IP<2:0>			CMP1IS<1:0>	
	7:0	—	—	—	PMPIP<2:0>			PMPIS<1:0>	
IPC7CLR	31:0	Clears the selected bits in IPC7, read yields undefined value							
IPC7SET	31:0	Sets the selected bits in IPC7, read yields undefined value							
IPC7INV	31:0	Inverts the selected bits in IPC7, read yields undefined value							

Register 19-1: CM1CON: Comparator 1 Control Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	r-x	r-x	r-x	r-x	R-0
ON	COE	CPOL	—	—	—	—	COUT
bit 15							bit 8

R/W-1	R/W-1	r-x	R/W-0	r-x	r-x	R/W-1	R/W-1
EVPOL<1:0>		—	CREF	—	—	CCH<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15 **ON:** Comparator ON bit

1 = Module is enabled. Setting this bit does not affect the other bits in this register.
0 = Module is disabled and does not consume current. Clearing this bit does not affect the other bits in this register.

Note: When using 1:1 PBCLK divisor, the user’s software should not read/write the peripheral’s SFRs in the SYSCLK cycle immediately following the instruction that clears the module’s ON bit.

bit 14 **COE:** Comparator Output Enable bit

1 = Comparator output is driven on the output C1OUT pin
0 = Comparator output is not driven on the output C1OUT pin

bit 13 **CPOL:** Comparator Output Inversion bit

1 = Output is inverted
0 = Output is not inverted

Note: Setting this bit will invert the signal to the comparator interrupt generator as well. This will result in an interrupt being generated on the opposite edge from the one selected by EVPOL<1:0>.

bit 12 **Reserved:** Write ‘0’; ignore read

bit 11-9 **Reserved:** Write ‘0’; ignore read

bit 8 **COUT:** Comparator Output bit

1 = Output of the Comparator is a ‘1’
0 = Output of the Comparator is a ‘0’

bit 7-6 **EVPOL<1:0>:** Interrupt Event Polarity Select bits

11 = Comparator interrupt is generated on a low-to-high or high-to-low transition of the comparator output
10 = Comparator interrupt is generated on a high-to-low transition of the comparator output
01 = Comparator interrupt is generated on a low-to-high transition of the comparator output
00 = Comparator interrupt generation is disabled

bit 5 **Reserved:** Write ‘0’; ignore read

Register 19-1: CM1CON: Comparator 1 Control Register (Continued)

bit 4 **CREF:** Comparator 1 Positive Input Configure bit
1 = Comparator non-inverting input is connected to the internal CVREF
0 = Comparator non-inverting input is connected to the C1IN+ pin

bit 3-2 **Reserved:** Write '0'; ignore read

bit 1-0 **CCH<1:0>:** Comparator Negative Input Select bits for Comparator 1
11 = Comparator inverting input is connected to the IVREF
10 = Comparator inverting input is connected to the C2IN+ pin
01 = Comparator inverting input is connected to the C1IN+ pin
00 = Comparator inverting input is connected to the C1IN- pin

Register 19-2: CM1CONCLR: Comparator Control Clear Register

Write clears selected bits in CM1CON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in CM1CON**

A write of '1' in one or more bit positions clears the corresponding bit(s) in CM1CON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `CM1CONCLR = 0x00008001` clears bits 15 and 0 in CM1CON register.

Register 19-3: CM1CONSET: Comparator Control Set Register

Write sets selected bits in CM1CON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in CM1CON**

A write of '1' in one or more bit positions sets the corresponding bit(s) in CM1CON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `CM1CONSET = 0x00008001` sets bits 15 and 0 in CM1CON register.

Register 19-4: CM1CONINV: Comparator Control Invert Register

Write inverts selected bits in CM1CON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in CM1CON**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in CM1CON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `CM1CONINV = 0x00008001` inverts bits 15 and 0 in CM1CON register.

Register 19-5: CM2CON: Comparator 2 Control Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	r-x	r-x	r-x	r-x	R-0
ON	COE	CPOL	—	—	—	—	COUT
bit 15							bit 8

R/W-1	R/W-1	r-x	R/W-0	r-x	r-x	R/W-1	R/W-1
EVPOL<1:0>		—	CREF	—	—	CCH<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15 **ON:** Comparator ON bit
1 = Module is enabled. Setting this bit does not affect the other bits in this register.
0 = Module is disabled and does not consume current. Clearing this bit does not affect the other bits in this register.
Note: When using 1:1 PBCLK divisor, the user’s software should not read/write the peripheral’s SFRs in the SYSCLK cycle immediately following the instruction that clears the module’s ON bit.

bit 14 **COE:** Comparator Output Enable bit
1 = Comparator output is driven on the output C2OUT pin
0 = Comparator output is not driven on the output C2OUT pin

bit 13 **CPOL:** Comparator Output Inversion bit
1 = Output is inverted
0 = Output is not inverted
Note: Setting this bit will invert the signal to the comparator interrupt generator as well. This will result in an interrupt being generated on the opposite edge from the one selected by EVPOL<1:0>.

bit 12 **Reserved:** Write ‘0’; ignore read

bit 11-9 **Reserved:** Write ‘0’; ignore read

bit 8 **COUT:** Comparator Output bit
1 = Output of the Comparator is a ‘1’
0 = Output of the Comparator is a ‘0’

bit 7-6 **EVPOL<1:0>:** Interrupt Event Polarity Select bits
11 = Comparator interrupt is generated on a low-to-high or high-to-low transition of the comparator output
10 = Comparator interrupt is generated on a high-to-low transition of the comparator output
01 = Comparator interrupt is generated on a low-to-high transition of the comparator output
00 = Comparator interrupt generation is disabled

bit 5 **Reserved:** Write ‘0’; ignore read

Register 19-5: CM2CON: Comparator 2 Control Register (Continued)

bit 4 **CREF:** Comparator 1 Positive Input Configure bit
1 = Comparator non-inverting input is connected to the internal CVREF
0 = Comparator non-inverting input is connected to the C2IN+ pin

bit 3-2 **Reserved:** Write '0'; ignore read

bit 1-0 **CCH<1:0>:** Comparator Negative Input Select bits for Comparator 2
11 = Comparator inverting input is connected to the IVREF
10 = Comparator inverting input is connected to the C1IN+ pin
01 = Comparator inverting input is connected to the C2IN+ pin
00 = Comparator inverting input is connected to the C2IN- pin

Register 19-6: CM2CONCLR: Comparator Control Clear Register

Write clears selected bits in CM2CON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in CM2CON**

A write of '1' in one or more bit positions clears corresponding bit(s) in CM2CON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `CM2CONCLR = 0x00008001` clears bits 15 and 0 in CM2CON register.

Register 19-7: CM2CONSET: Comparator Control Set Register

Write sets selected bits in CM2CON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in CM2CON**

A write of '1' in one or more bit positions sets corresponding bit(s) in CM2CON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `CM2CONSET = 0x00008001` sets bits 15 and 0 in CM2CON register.

Register 19-8: CM2CONINV: Comparator Control Invert Register

Write inverts selected bits in CM2CON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in CM2CON**

A write of '1' in one or more bit positions inverts corresponding bit(s) in CM2CON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `CM2CONINV = 0x00008001` inverts bits 15 and 0 in CM2CON register.

Register 19-9: CMSTAT: Comparator Control Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	R/W-0	R/W-0	r-x	r-x	r-x	r-x	r-x
—	FRZ	SIDL	—	—	—	—	—
bit 15							bit 8

r-x	r-x	r-x	r-x	r-x	r-x	R-0	R-0
—	—	—	—	—	—	C2OUT	C1OUT
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-15 **Reserved:** Write ‘0’; ignore read

bit 14 **FRZ:** Freeze Control bit
1 = Freeze operation when CPU enters Debug Exception mode
0 = Continue operation when CPU enters Debug Exception mode
Note: FRZ is writable in Debug Exception mode only. It always reads ‘0’ in Normal mode.

bit 13 **SIDL:** Stop in IDLE Control bit
1 = All Comparator modules are disabled in IDLE mode
0 = All Comparator modules continue to operate in the IDLE mode

bit 12-2 **Reserved:** Write ‘0’; ignore read

bit 1 **C2OUT:** Comparator Output bit
1 = Output of Comparator 2 is a ‘1’
0 = Output of Comparator 2 is a ‘0’

bit 0 **C1OUT:** Comparator Output bit
1 = Output of Comparator 1 is a ‘1’
0 = Output of Comparator 1 is a ‘0’

Register 19-10: CMSTATCLR: Comparator Control Clear Register

Write clears selected bits in CMSTAT, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in CMSTAT**

A write of '1' in one or more bit positions clears corresponding bit(s) in CMSTAT register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `CMSTATCLR = 0x00002000` clears bit 13 in CMSTAT register.

Register 19-11: CMSTATSET: Comparator Control Set Register

Write sets selected bits in CMSTAT, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in CMSTAT**

A write of '1' in one or more bit positions sets corresponding bit(s) in CMSTAT register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `CMSTATSET = 0x00002000` sets bit 13 in CMSTAT register.

Register 19-12: CMSTATINV: Comparator Control Invert Register

Write inverts selected bits in CMSTAT, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in CMSTAT**

A write of '1' in one or more bit positions inverts corresponding bit(s) in CMSTAT register, and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `CMSTATINV = 0x00002000` inverts bit 13 in CMSTAT register.

Register 19-13: IFS1: Interrupt Flag Status Register 1(1)

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	USBIF	FCEIF
bit 31							bit 24

r-0	r-0	r-0	r-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	DMA3IF	DMA2IF	DMA1IF	DMA0IF
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RTCCIF	FSCMIF	I2C2MIF	I2C2SIF	I2C2BIF	U2TXIF	U2RXIF	U2EIF
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SPI2RXIF	SPI2TXIF	SPI2EIF	CMP2IF	CMP1IF	PMPIF	AD1IF	CNIF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 4 **CMP2IF:** Comparator 2 Interrupt Request Flag bit
1 = Interrupt request has occurred
0 = Interrupt request has not occurred

bit 3 **CMP1IF:** Comparator 1 Interrupt Request Flag bit
1 = Interrupt request has occurred
0 = Interrupt request has not occurred

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the comparator.

Register 19-14: IEC1: Interrupt Enable Control Register 1(1)

r-x	r-x	r-x0	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	USBIE	FCEIE
bit 31							bit 24

r-0	r-0	r-0	r-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	DMA3IE	DMA2IE	DMA1IE	DMA0IE
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RTCCIE	FSCMIE	I2C2MIE	I2C2SIE	I2C2BIE	U2TXIE	U2RXIE	U2EIE
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SPI2RXIE	SPI2TXIE	SPI2EIE	CMP2IE	CMP1IE	PMPIE	AD1IE	CNIE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 4 **CMP2IE:** Comparator 2 Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 3 **CMP1IE:** Comparator 1 Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the comparator.

Register 19-15: IPC7: Interrupt Priority Control Register 7(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	SPI2IP<2:0>			SPI2IS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	CMP2IP<2:0>			CMP2IS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	CMP1IP<2:0>			CMP1IS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	PMPIP<2:0>			PMPIS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 20-18 **CMP2IP<2:0>:** Comparator 2 Interrupt Priority bits
- 111 = Interrupt Priority is 7
- 110 = Interrupt Priority is 6
- 101 = Interrupt Priority is 5
- 100 = Interrupt Priority is 4
- 011 = Interrupt Priority is 3
- 010 = Interrupt Priority is 2
- 001 = Interrupt Priority is 1
- 000 = Interrupt is disabled

bit 17-6 **CMP2IS<1:0>:** Comparator 2 Interrupt Subpriority bits
- 11 = Interrupt Subpriority is 3
- 10 = Interrupt Subpriority is 2
- 01 = Interrupt Subpriority is 1
- 00 = Interrupt Subpriority is 0

bit 12-10 **CMP1IP<2:0>:** Comparator 1 Interrupt Priority bits
- 111 = Interrupt Priority is 7
- 110 = Interrupt Priority is 6
- 101 = Interrupt Priority is 5
- 100 = Interrupt Priority is 4
- 011 = Interrupt Priority is 3
- 010 = Interrupt Priority is 2
- 001 = Interrupt Priority is 1
- 000 = Interrupt is disabled

bit 9-8 **CMP1IS<1:0>:** Comparator 1 Interrupt Subpriority bits
- 11 = Interrupt Subpriority is 3
- 10 = Interrupt Subpriority is 2
- 01 = Interrupt Subpriority is 1
- 00 = Interrupt Subpriority is 0

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the comparator.

19.3 COMPARATOR OPERATION

19.3.1 Comparator Configuration

The Comparator module has a flexible input and output configuration to allow the module to be tailored to the needs of the application. The PIC32MX Comparator module has individual control over the enables, output inversion, output on I/O pin and input selections. The VIN+ pin of each comparator can select from an input pin or the CVREF. The VIN- input of the comparator can select from one of 3 input pins or the IVREF. In addition, the module has two individual comparator event generation control bits. These control bits can be used for detecting when the output of an individual comparator changes to a desired state or changes states.

If the Comparator mode is changed, the comparator output level may not be valid for the specified mode change delay (refer to the device data sheet for more information).

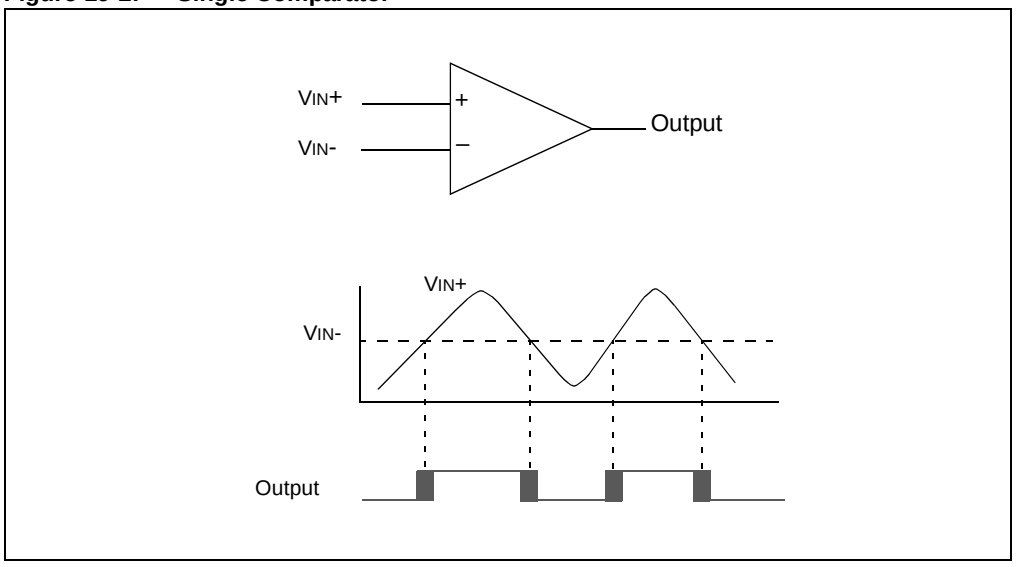
Note:	Comparator interrupts should be disabled during a Comparator mode change; otherwise, a false interrupt may be generated.

A single comparator is shown in the upper portion of Figure 19-2. The lower portion represents the relationship between the analog input levels and the digital output. When the analog input at VIN+ is less than the analog input at VIN-, the output of the comparator is a digital low level. When the analog input at VIN+ is greater than the analog input VIN-, the output of the comparator is a digital high level. The shaded areas of the output of the comparator in the lower portion of Figure 19-2 demonstrates the uncertainty that is due to input offsets and the response time of the comparator.

19.3.2 Comparator Inputs

Depending on the comparator Operating mode, the inputs to the comparators may be from two input pins or a combination of an input pin and one of two internal voltage references. The analog signal present at VIN- is compared to the signal at VIN+ and the digital output of the comparator is set or cleared according to the result of the comparison (see Figure 19-2).

Figure 19-2: Single Comparator



19.3.2.1 External Reference Signal

An external voltage reference may be used with the comparator by using the output of the reference as an input to the comparator. Refer to the device data sheet for input voltage limits.

19.3.2.2 Internal Reference Signals

The CVREF module and the IVREF can be used as inputs to the comparator (see Figure 19-1). The CVREF provides a user-selectable voltage for use as a comparator reference. Refer to **Section 20, "Comparator Voltage Reference"** of this manual for more information on this module. The IVREF has a fixed 1.2V output that does not change with the device supply voltage. Refer to the device data sheet for specific details and accuracy of this reference.

19.3.3 Comparator Response Time

Response time is the minimum amount of time that elapses from the moment a change is made in the input voltage to a comparator to the moment that the output reflects the new level. If the internal reference is changed, the maximum delay of the internal voltage reference must be considered when using the comparator outputs. Otherwise, the maximum delay of the comparators should be used (see the device data sheet for detailed information).

19.3.4 Comparator Outputs

The comparator output is read through the CMSTAT register and the COUT bit (CM2CON<8> or CM1CON<8>). This bit is read-only. The comparator output may also be directed to an I/O pin via the CxOUT bit; however, the COUT bit is still valid when the signal is routed to a pin. For the comparator output to be available on the CxOut pin, the associated TRIS bit for the output pin must be configured as an output. When the COUT signal is routed to a pin the signal is the unsynchronized output of the comparator.

The output of the comparator has a degree of uncertainty. The uncertainty of each of the comparators is related to the input offset voltage and the response time, as stated in the specifications. The lower portion of Figure 19-2 provides a graphical representation of this uncertainty.

The comparator output bit COUT provides the latched sampled value of the comparator's output when the register was read. There are two common methods used to detect a change in the comparator output:

- Software polling
- Interrupt generation

19.3.4.1 Software Polling Method of Comparator Event Detection

Software polling of COUT is performed by periodically reading the COUT bit. This allows the output to be read at uniform time intervals. A change in the comparator output is not detected until the next read of the COUT bit. If the input signal changes at a rate faster than the polling, a brief change in output may not be detected.

19.3.4.2 Interrupt Generation Method of Comparator Event Detection

Interrupt generation is the other method for detecting a change in the comparator output. The Comparator module can be configured to generate an interrupt when the COUT bit changes.

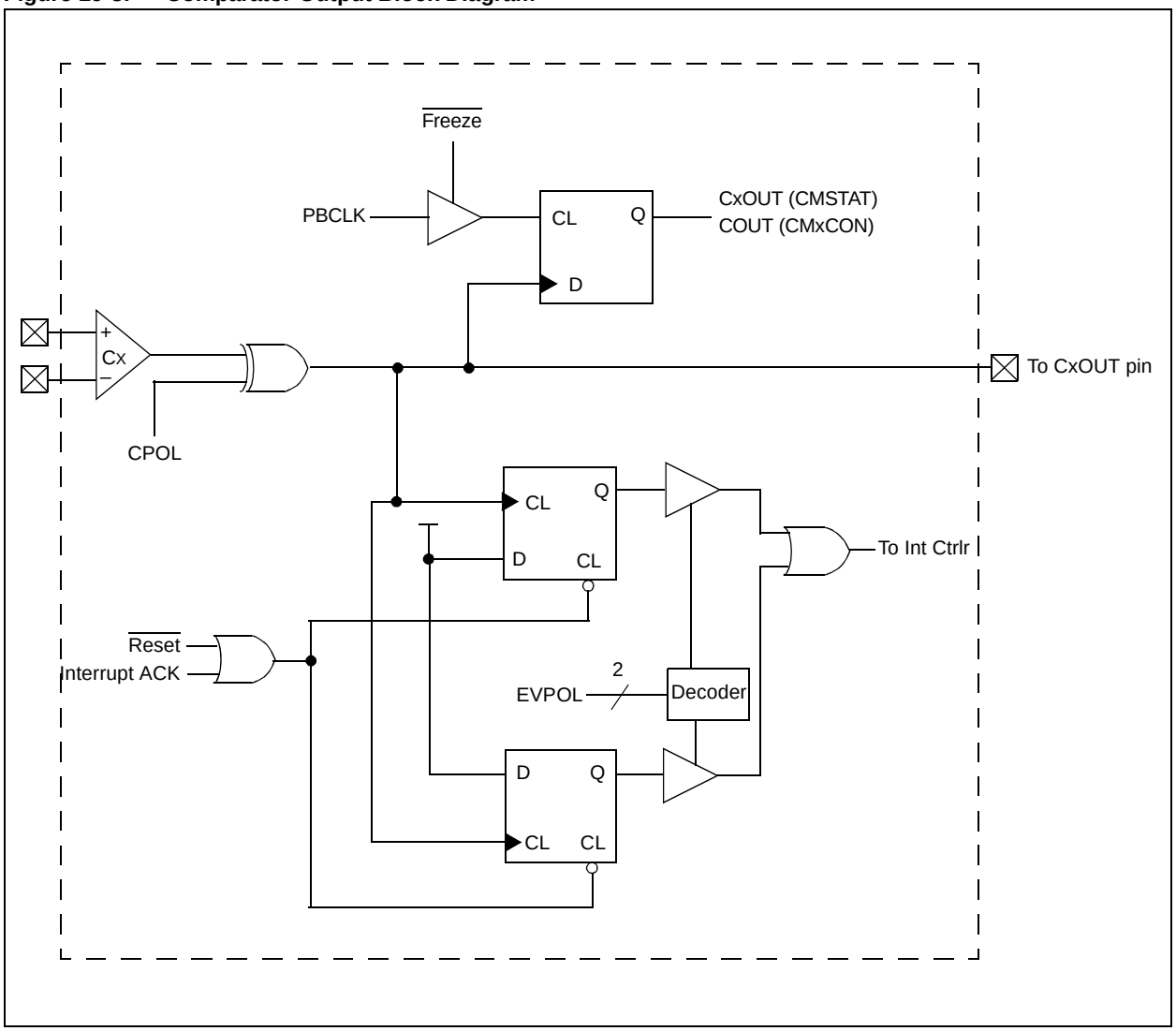
An interrupt will be generated when the comparator's output changes (subject to the interrupt priorities). This method responds more rapidly to changes than the software polling method; however, rapidly changing signals will cause an equally large number of interrupts. This can cause interrupt loading and potentially undetected interrupts due to new interrupts being generated while the previous interrupt is still being serviced or even before the interrupt can be serviced. If the input signal changes rapidly, reading the COUT bit in the Interrupt Service Routine may yield a different result than the one that generated the Interrupt. This is due to the COUT bit representing the value of the comparator output when the bit was read and not the value that caused the interrupt.

Comparator output and interrupt generation is illustrated in Figure 19-3.

19.3.4.3 Changing the Polarity of Comparator Outputs

The polarity of the comparator outputs can be changed using the CPOL bit (CM1CON<13>). CPOL appears below the comparator Cx on the left side of Figure 19-3.

Figure 19-3: Comparator Output Block Diagram



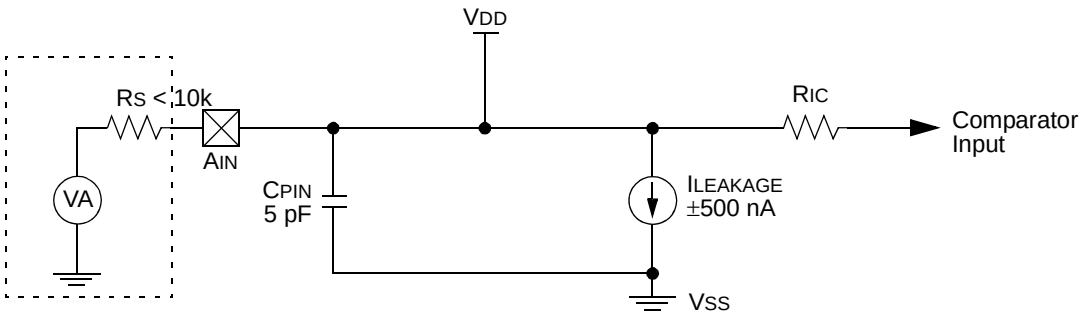
19.3.5 Analog Input Connection Considerations

A simplified circuit for an analog input is shown in Figure 19-4. A maximum source impedance of 10 kΩ is recommended for the analog sources. Any external component connected to an analog input pin, such as a capacitor or a Zener diode, should have very little leakage current. See the device data sheet for input voltage limits. If a pin is to be shared by two or more analog inputs that are to be used simultaneously, the loading effects of all the modules involved must be taken into consideration. This loading may reduce the accuracy of one or more of the modules connected to the common pin. This may also require a lower source impedance than is stated for a single module with exclusive use of a pin in analog mode.

Notes: When reading the PORT register, all pins configured as analog inputs will read as a '0'. Pins configured as digital inputs will convert an analog input according to the Schmitt Trigger input specification.

Analog levels on any pin defined as a digital input may cause the input buffer to consume more current than is specified.

Figure 19-4: Comparator Analog Input Model



Legend:	CPIN	=	Input Capacitance
	ILEAKAGE	=	Leakage Current at the pin due to various junctions
	RIC	=	Interconnect Resistance
	RS	=	Source Impedance
	VA	=	Analog Voltage

19.4 INTERRUPTS

Each of the available comparators has a dedicated interrupt bit, CMPxIF (IFS1<3 or 4>), and a corresponding interrupt enable/mask bit, CMPxIE (IEC1<3 or 4>). These bits are used to determine the source of an interrupt and to enable or disable an individual interrupt source. The priority level of each of the channels can also be set independently of the other channels.

The CMPxIF bit is set when the CMPx channel detects a predefined match condition that is defined as an event generating an interrupt. The CMPxIF bit will then be set without regard to the state of the corresponding CMPxIE bit. The CMPxIF bit can be polled by software if desired.

The CMPxIE bit controls the interrupt generation. If the CMPxIE bit is set, the CPU will be interrupted whenever a comparator interrupt event occurs and the corresponding CMPxIF bit will be set (subject to the priority and subpriority as outlined below).

It is the responsibility of the user's software routine that services a particular interrupt to clear the appropriate interrupt flag bit before the service routine is complete.

The priority of each comparator channel can be set independently via the CMPxIP<2:0> bits. This priority defines the priority group to which the interrupt source will be assigned. The priority groups range from a value of 7 (the highest priority), to a value of 0 (which does not generate an interrupt). An interrupt being serviced will be preempted by an interrupt in a higher priority group.

The subpriority bits allow setting the priority of an interrupt source within a priority group. The values of the subpriority bit OCxIS<1:0> range from 3 (the highest priority), to 0 (the lowest priority). An interrupt within the same priority group but having a higher subpriority value will preempt a lower subpriority interrupt that is in progress.

The priority group and subpriority bits allow more than one interrupt source to share the same priority and subpriority. If simultaneous interrupts occur in this configuration, the natural order of the interrupt sources within a priority/subgroup pair determine the interrupt generated. The natural priority is based on the vector numbers of the interrupt sources. The lower the vector number, the higher the natural priority of the interrupt. Any interrupts that were overridden by natural order will then generate their respective interrupts based on priority, subpriority, and natural order, after the interrupt flag for the current interrupt is cleared.

After an enabled interrupt is generated, the CPU will jump to the vector assigned to that interrupt (refer to Table 19-2). The vector number for the interrupt is the same as the natural order number. The CPU will then begin executing code at the vector address. The user's code at this vector address should perform any application-specific operations required, such as reloading the duty cycle, clear the interrupt flag CMPxIF, and then exit. Refer to the vector address table details in **Section 8. "Interrupts"** for more information on interrupts.

Table 19-2: Typical Comparator Interrupt Vectors for Various Offsets with EBASE = 0x8000:0000

Interrupt	Vector	IRQ Number	Vector Address IntCtl.VS = 0x01	Vector Address IntCtl.VS = 0x02	Vector Address IntCtl.VS = 0x04	Vector Address IntCtl.VS = 0x08	Vector Address IntCtl.VS = 0x10
CMP1	29	35	8000 0660	8000 0AC0	8000 1380	8000 2500	8000 4800
CMP2	30	36	8000 0680	8000 0B00	8000 1400	8000 2600	8000 4A00

Example 19-1: Comparator Initialization with Interrupts Enabled Code Example

```
                          // Configure both comparators to generate an interrupt on any
                          // output transition
CM1CON  = 0xC0D0;         // Initialize Comparator 1
                          // Comparator enabled, output enabled, interrupt on any output
                          // change, inputs: CVref, C1IN-
CM2CON  = 0xA0C2;         // Initialize Comparator 2
                          // Comparator enabled, output enabled, interrupt on any output
                          // change, inputs: C2IN+, C1IN+

                          // Enable interrupts for Comparator modules and set priorities
                          // Set priority to 7 & sub priority to 3
IPC7SET = 0x00000700;     // Set CMP1 interrupt sub priority
IFS1CLR = 0x00000008;     // Clear the CMP1 interrupt flag
IEC1SET = 0x00000008;     // Enable CMP1 interrupt

IPC7SET = 0x00070000;     // Set CMP2 interrupt sub priority
IFS1CLR = 0x000000010;    // Clear the CMP2 interrupt flag
IEC1SET = 0x000000010;    // Enable CMP2 interrupt
```

Example 19-2: Comparator ISR Code Example

```
                          // Insert user code here

void__ISR(_COMPARATOR_2_VECTOR, ipl7) Cmp2_IntHandler (void)
{
                          // Insert user code here
    IFS1CLR = 0x00000010; // Clear the CMP2 interrupt flag
}

void__ISR(_COMPARATOR_1_VECTOR, ipl7) Cmp1_IntHandler (void)
{
                          // Insert code user here
    IFS1CLR = 0x00000008; // Clear the CMP1 interrupt flag
}
```

19.5 I/O PIN CONTROL

The Comparator module shares pins with port input/output control and in some cases with other modules. The following conditions must be provided to configure a pin for use by the comparator:

- Any modules sharing the pin must be disabled
- Comparator must be configured to use the desired pin
- TRIS bit corresponding to the pin must be a '1'
- Comparator must be enabled (refer to Table 19-3)
- Corresponding AD1PCFG bit must be a '0'.

  The comparator controls pin function for the desired comparator via the following bits in the CMxCON register: CREF, CCH<1:0>, and COE. The TRIS bit corresponding to any analog input pin for the comparator must be '1'. This disables the digital input buffer for the pin. When a pin is selected as analog output the digital output driver is disabled. The TRIS bit corresponding to the CxOUT pin must be a '0' if the comparator digital output is to be used.

Table 19-3: Pins Associated with a Comparator

Pin Name	Module Control	Controlling Bit Field	Required TRIS Bit Setting	Pin Type	Buffer Type	Description
C1IN+	ON	CVR_{EF}(1), CCH<1:0>(1), CCH<1:0>(2), AD1PCFG	Input	A, I	—	Analog Input for C1IN+
C1IN-	ON	CCH<1:0>(1), AD1PCFG	Input	A, I	—	Analog Input for C1IN-
C2IN+	ON	CVR_{EF}(2), CCH<1:0>(1), CCH<1:0>(2), AD1PCFG	Input	A, I	—	Analog Input for C2IN+
C2IN-	ON	CCH<1:0>(2), AD1PCFG	Input	A, I	—	Analog Input for C2IN-
C1OUT	ON	COE(1)	Output	D, O	—	Digital Output of the C1
C2OUT	ON	COE(2)	Output	D, O	—	Digital Output of the C2

Legend:
ST = Schmitt Trigger input with CMOS levels, I = Input, O = Output, A = Analog, D = Digital

Note 1: In CM1CON register
2: In CM2CON register

For example, if Comparator 1 is to use two external inputs C1IN+ and C1IN-, with an inverting output to a pin that does not generate an interrupt, the following configuration steps would be performed.

- Configure the TRIS Bits:
  - TRIS = Output – configures the C1IN+ and C1IN- pins as digital outputs to disable the digital input buffer.

    The output driver will be disabled when the pin is selected as an analog input by the module.
  - TRIS = Output – enables the output driver for the C1OUT signal.
- Set the CM1CON bits:
  - CREF (CM1CON<4>) = 0 – selects C1IN+ as a analog input to the comparator.
  - CCH<1:0> (CM1CON<1:0>) = 00 – selects C1IN- as an analog input (C2IN+ and C2IN- are available for use by other modules or general purpose I/O that share the pin).
  - CPOL (CM1CON <13>) = 1 – selects inverted output mode.
  - COE (CM1CON<14>) = 1 – enables the output of the comparator to be available at the C1OUT pin.
  - EVPOL<1:0> (CM1CON<7:6>) = 00 – disables interrupt generation.
  - ON (CM1CON <15>) = 1 – enables the module.

    ON is always set after the preceding bits are set.

19.6 OPERATION IN POWER-SAVING AND DEBUG MODES

Note: In this manual, a distinction is made between a power mode as it is used in a specific module and a power mode as it is used by the device, e.g., Sleep mode of the Comparator and SLEEP mode of the CPU. To indicate which type of power mode is intended, upper- and lower-case letters (Sleep, Idle, Debug) signify a module power mode and all upper-case letters (SLEEP, IDLE, DEBUG) signify a device power mode.

19.6.1 Comparator Operation During IDLE Mode

When a comparator is active and the device is placed in IDLE mode, the comparator remains active and interrupts are generated (if enabled); if SIDL = 1 (CMSTAT<13>), the comparators are disabled in IDLE mode.

19.6.2 Comparator Operation During SLEEP Mode

When a comparator is active and the device is placed in SLEEP mode, the comparator remains active and the interrupt is functional (if enabled). This interrupt will wake up the device from SLEEP mode (when enabled). Each operational comparator will consume additional current, as shown in the comparator specifications. To minimize power consumption while in SLEEP mode, turn off the comparators: ON = 0 (CMxCON<15>), prior to entering SLEEP mode. If the device wakes up from SLEEP mode, the contents of the CMxCON register are not affected. See **Section 10. "Power-Saving Modes"** in this manual for additional information on SLEEP.

19.6.3 Comparator Operation in DEBUG Mode

The FRZ bit (CMSTAT<14>) determines whether the Comparator module will run or stop while the CPU is executing debug exception code (i.e., application is halted) in DEBUG mode. When FRZ = 0, the Comparator module continues to run even when application is halted in DEBUG mode. When FRZ = 1 and application is halted in DEBUG mode, the module will freeze its operations and make no changes to the state of the Comparator module. The module will resume its operation after the CPU resumes execution.

Note: The FRZ bit is readable and writable only when the CPU is executing in Debug Exception mode. In all other modes, the FRZ bit reads as '0'. If FRZ bit is changed during DEBUG mode, the new value does not take effect until the current Debug Exception mode is exited and re-entered. During the Debug Exception mode, the FRZ bit reads the state of the peripheral when entering DEBUG mode.

19.7 EFFECTS OF A RESET

All Resets force the CMxCON registers to its Reset state, causing the comparator modules to be turned off (CMxCON<15> = 0). However, the input pins multiplexed with analog input sources are configured as analog inputs by default on device Reset. The I/O configuration for these pins is determined by the setting of the AD1PCFG register.

19.8 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the PIC32MX device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the Comparator module are:

Title	Application Note #
No related application notes at this time	N/A

Note: Please visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the PIC32MX family of devices.

19.9 REVISION HISTORY

Revision A (October 2007)

This is the initial released version of this document.

Revision B (October 2007)

Updated document to remove Confidential status.

Revision C (April 2008)

Revised status to Preliminary; Revised U-0 to r-x.

Revision D (May 2008)

Revised Figure 19-1; Revised Registers 19-1, 19-5, 19-13, 19-14, 19-15; Revised Example 19-2; Revised Section 19.5, pin names; Change Reserved bits from “Maintain as” to “Write”; Added Note to ON bit (CM1CON/CM2CON Registers).

NOTES:

Section 20. Comparator Voltage Reference

HIGHLIGHTS

This section of the manual contains the following topics:

20.1 INTRODUCTION

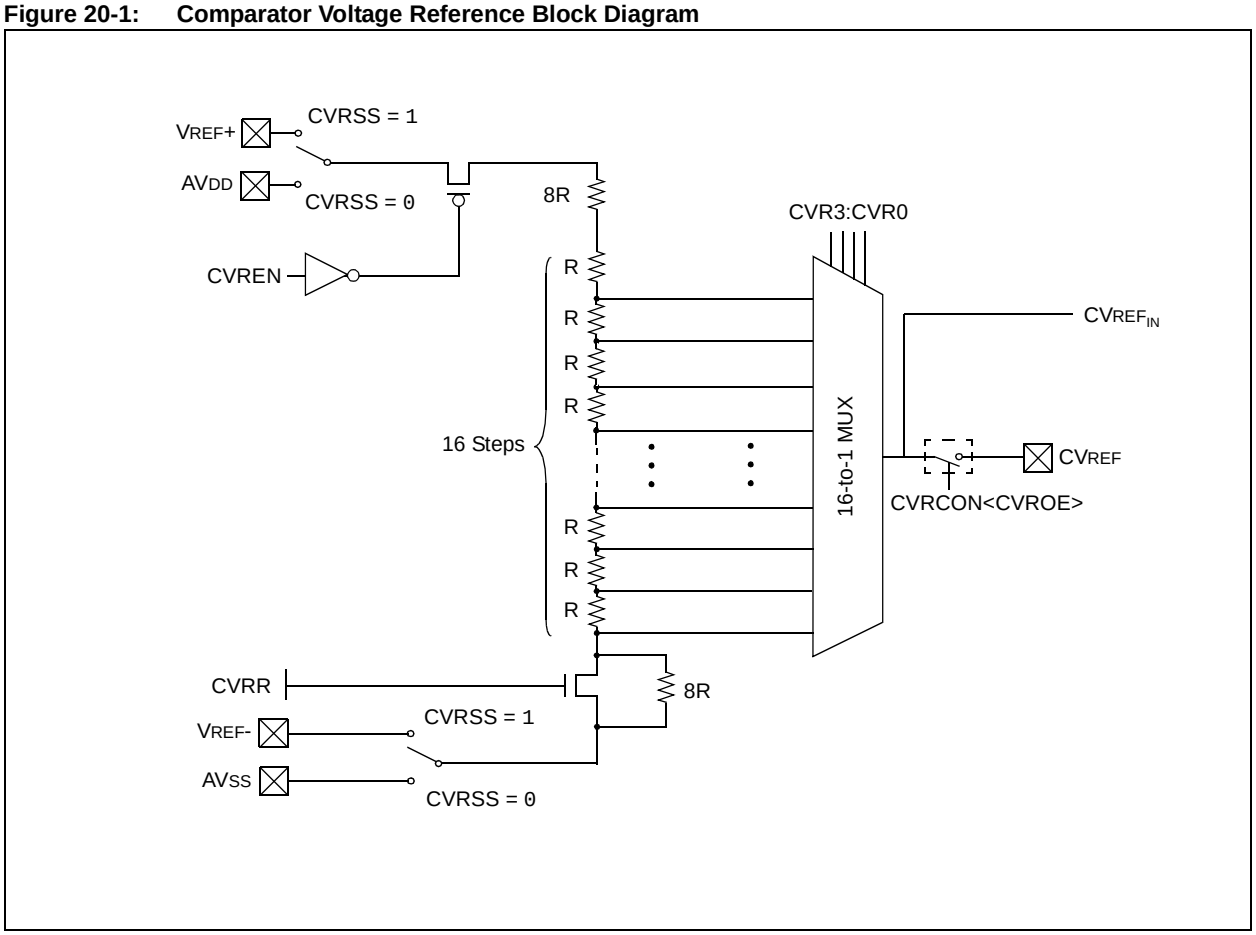
The Comparator Voltage Reference (CVREF) is a 16-tap, resistor ladder network that provides a selectable reference voltage. Although its primary purpose is to provide a reference for the analog comparators, it also may be used independently of them.

A block diagram of the module is shown in Figure 20-1. The resistor ladder is segmented to provide two ranges of voltage reference values and has a power-down function to conserve power when the reference is not being used. The module's supply reference can be provided from either device VDD/VSS or an external voltage reference. The CVREF output is available for the comparators and typically available for pin output. Please see the specific device data sheet for more information.

The Comparator Voltage Reference has the following features:

- High and low range selection
- Sixteen output levels available for each range
- Internally connected to comparators to conserve device pins
- Output can be connected to a pin

Figure 20-1: Comparator Voltage Reference Block Diagram



20.2 COMPARATOR VOLTAGE REFERENCE CONTROL REGISTERS

The CVREF module consists of the following Special Function Registers (SFR):

- CVRCON: Control Register for the module

  CVRCONCLR, CVRCONSET, CVRCONINV: Atomic Bit Manipulation Registers for CVRCON

The following table provides a brief summary of all CVREF-module-related registers. Corresponding registers appear after the summary, followed by a detailed description of each register.

Table 20-1: Comparator Voltage Reference SFR Summary

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
CVRCON	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	ON	—	—	—	—	—	—	—
	7:0	—	CVROE	CVRR	CVRSS	CVR<3:0>			
CVRCONCLR	31:0	Write clears selected bits in CVRCON, read yields undefined							
CVRCONSET	31:0	Write sets selected bits in CVRCON, read yields undefined							
CVRCONINV	31:0	Write inverts selected bits in CVRCON, read yields undefined							

Register 20-1: CVRCON: Comparator Voltage Reference Control Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	r-x	r-x	r-x	r-x	r-x	r-x	r-x
ON	—	—	—	—	—	—	—
bit 15							bit 8

r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	CVROE	CVRR	CVRSS	CVR<3:0>			
bit 7							bit 0

Legend:			
R = readable bit	W = writable bit	P = programmable	r = reserved bit
U = unimplemented bit, read as '0'		-n = bit value at POR: ('0', '1', x = unknown)	

bit 31-16 **Reserved:** Write '0'; ignore read

bit 15 **ON:** CVREF Peripheral On bit

1 = Module is enabled, setting this bit does not affect the other bits in the register.
0 = Module is disabled and does not consume current. Clearing this bit does not affect the other bits in the register.

Note: When using 1:1 PBCLK divisor, the user's software should not read/write the peripheral's SFRs in the SYSCLK cycle immediately following the instruction that clears the module's ON bit.

bit 14-7 **Reserved:** Write '0'; ignore read

bit 6 **CVROE:** CVREF Output Enable bit

1 = Voltage level is output on CVREF pin
0 = Voltage level is disconnected from CVREF pin
Note: CVROE overrides the TRIS bit setting, see **Section 12. "I/O Ports"** for more information.

bit 5 **CVRR:** CVREF Range Selection bit

1 = 0 to 0.67 CVRSRC, with CVRSRC/24 step size
0 = 0.25 CVRSRC to 0.75 CVRSRC, with CVRSRC/32 step size

bit 4 **CVRSS:** CVREF Source Selection bit

1 = Comparator voltage reference source, CVRSRC = (VREF+) – (VREF-)
0 = Comparator voltage reference source, CVRSRC = AVDD – AVSS

bit 3-0 **CVR<3:0>:** CVREF Value Selection 0 ≤ CVR3:CVR0 ≤ 15 bits

When CVRR = 1:
CVREF = (CVR<3:0>/24) • (CVRSRC)

When CVRR = 0:
CVREF = 1/4 • (CVRSRC) + (CVR<3:0>/32) • (CVRSRC)

Register 20-2: CVRCONCLR: CVREF Control Clear Register

Write clears selected bits in CVRCON, read yields undefined
bit 31 bit 0

bit 31-0 **Clear selected bits in CVRCON**

A write of '1' in one or more bit positions clears the corresponding bit(s) in CVRCON register, a write of '0' will not affect the register.(1)

The read operation returns an undefined value and is not recommended.

Example:
`CVRCONCLR = 0x00008001` clears bits 15 and 0 in CVRCON register.

Note 1: This operation will not affect unimplemented or read-only bits.

Register 20-3: CVRCONSET: CVREF Control Set Register

Write sets selected bits in CVRCON, read yields undefined
bit 31 bit 0

bit 31-0 **Set selected bits in CVRCON**

A write of '1' in one or more bit positions sets the corresponding bit(s) in CVRCON register, a write of '0' will not affect the register.(1)

The read operation returns an undefined value and is not recommended.

Example:
`CVRCONSET = 0x00008001` sets bits 15 and 0 in CVRCON register.

Note 1: This operation will not affect unimplemented or read-only bits.

Register 20-4: CVRCONINV: CVREF Control Invert Register

Write inverts selected bits in CVRCON, read yields undefined
bit 31 bit 0

bit 31-0 **Inverts selected bits in CVRCON**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in CVRCON register, a write of '0' will not affect the register.(1)

The read operation returns an undefined value and is not recommended.

Example:
`CVRCONINV = 0x00008001` inverts bits 15 and 0 in CVRCON register.

Note 1: This operation will not affect unimplemented or read-only bits.

20.3 OPERATION

The CVREF module is controlled through the CVRCON register (Register 20-1). The CVREF provides two ranges of output voltage, each with 16 distinct levels. The range to be used is selected by the CVRR bit (CVRCON<5>). The primary difference between the ranges is the size of the steps selected by the CVREF value selection bits, CVR3:CVR0, with one range offering finer resolution and the other offering a wider range of output voltage. The typical output voltages are listed in Table 20-2.

The equations used to calculate the CVREF output are as follows:

If CVRR = 1:
Voltage Reference = ((CVR3:CVR0)/24) x (CVRSRC)

If CVRR = 0:
Voltage Reference = (CVRSRC/4) + ((CVR3:CVR0)/32) x (CVRSRC)

The CVREF Source Voltage (CVRSRC) can come from either VDD and VSS, or the external VREF+ and VREF- pins that are multiplexed with I/O pins. The voltage source is selected by the CVRSS bit (CVRCON<4>). The voltage reference is output to the CVREF pin by setting the CVROE (CVRCON<6>) bit; this will override the corresponding TRIS bit setting.

The settling time of the CVREF must be considered when changing the CVREF output (Refer to the device data sheet).

Table 20-2: Typical Voltage Reference with CVRSRC = 3.3

CVR<3:0>	Voltage Reference	
	CVRR = 0 (CVRCON <5>)	CVRR = 1 (CVRCON <5>
0	0.83V	0.00V
1	0.93V	0.14V
2	1.03V	0.28V
3	1.13V	0.41V
4	1.24V	0.55V
5	1.34V	0.69V
6	1.44V	0.83V
7	1.55V	0.96V
8	1.65V	1.10V
9	1.75V	1.24V
10	1.86V	1.38V
11	1.96V	1.51V
12	2.06V	1.65V
13	2.17V	1.79V
14	2.27V	1.93V
15	2.37V	2.06V

20.3.1 CVREF Output Considerations

The full range of voltage reference cannot be realized due to the construction of the module. The transistors on the top and bottom of the resistor ladder network (Figure 20-1) keep the voltage reference from approaching the reference source rails. The voltage reference is derived from the reference source; therefore, the voltage reference output changes with fluctuations in that source. Refer to the product data sheet for the electrical specifications. Table 20-3 contains the typical output impedances for the CVREF module.

Table 20-3: Typical CVREF Output Impedance in Ohms

CVR<3:0>	Voltage Reference	
	CVRR = 0 (CVRCON <5>)	CVRR = 1 (CVRCON <5>
0	12k	500
1	13k	1.9k
2	13.8k	3.7k
3	14.4k	5.3k
4	15k	6.7k
5	15.4k	7.9k
6	15.8k	9k
7	15.9k	9.9k
8	16k	10.7k
9	15.9k	11.3k
10	15.8k	11.7k
11	15.4k	11.9k
12	15k	12k
13	14.4k	11.9k
14	13.8k	11.7k
15	12.9k	11.3k

20.3.2 Initialization

This initialization sequence shown in Example 20-1 configures the CVREF module for: module enabled, output enabled, high range, and set output for maximum (2.37V).

Example 20-1: Voltage Reference Configuration

```
CVRCON = 0x804F;   //Initialize Voltage Reference Module
                   //enable module, enable output, set
                   // range to high, set output to maximum
```

20.4 INTERRUPTS

There are no Interrupt configuration registers or bits for the CVREF module. The CVREF module does not generate interrupts.

20.5 I/O PIN CONTROL

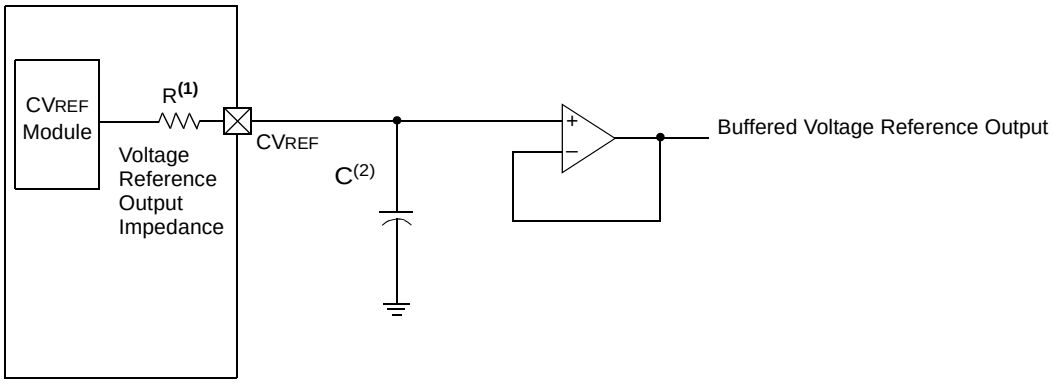
The CVREF module has the ability to output to a pin. When the CVREF module is enabled and CVROE (CVRCON<6>) is '1', the output driver for the CVREF pin is disabled and the CVREF voltage is available at the pin. For proper operation, the TRIS bit corresponding to the CVREF pin must be a '1' when CVREF is to be output to a pin. This disables the digital Input mode for the pin and prevents undesired current draw resulting from applying an analog voltage to a digital input pin. The output buffer has very limited drive capability. An external buffer amplifier is recommended for any application that uses the CVREF voltage externally. An output capacitor may be used to reduce output noise. Use of an output capacitor will increase settling time (see Figure 20-2).

Table 20-4: Pins Associated with a Comparator

Pin Name	Module Control	Controlling Bit Field	Required TRIS bit Setting	Pin Type	Buffer Type	Description
CVREF	ON	CVROE	Input	A, O	—	CVREF Output

Legend: ST = Schmitt Trigger input with CMOS levels, I = Input, O = Output, A = Analog, D = Digital

Figure 20-2: Comparator Voltage Reference Output Buffer Example



Note 1: R is dependent on the comparator voltage reference control bits CVRR (CVRCON<5>) and CVR<3:0> (CVRCON<3:0 Refer to Table 20-3

2: Use of an output capacitor will increase settling time. Capacitor value selection is dependent on the CVR<3:0> and CVRR settings, as well as the frequency to be attenuated.

20.6 OPERATION IN POWER-SAVING AND DEBUG MODES

Note:	In this manual, a distinction is made between a power mode as it is used in a specific module, and a power mode as it is used by the device, e.g., Sleep mode of the Comparator and SLEEP mode of the CPU. To indicate which type of power mode is intended, uppercase and lowercase letters (Sleep, Idle, Debug) signify a module power mode, and all uppercase letters (SLEEP, IDLE, DEBUG) signify a device power mode.

20.6.1 CVREF Operation in SLEEP Mode

The CVREF module continues to operate in SLEEP mode. The CVRCON register is not affected when the device enters or wakes from SLEEP mode. If the CVREF voltage is not used in SLEEP, the module can be disabled by clearing the ON bit CVRCON<15> prior to entering SLEEP to save power.

20.6.2 IDLE

The CVREF module continues to operate in IDLE mode. The CVRCON register is not affected when the device enters or exits IDLE mode. There is no provision to automatically disable the module in IDLE mode. If the CVREF voltage is not used in IDLE, the module can be disabled by clearing the ON bit CVRCON<15> prior to entering IDLE to save power.

20.6.3 DEBUG

The CVREF module continues to operate while the device is in DEBUG mode. The module doesn't support Freeze mode.

Note:	There is no FRZ mode for this module.

20.7 EFFECTS OF RESETS

All Resets disable the voltage reference by forcing all bits in CVRCON to a '0'.

20.8 DESIGN TIPS

Question 1: ***My voltage reference is not what I expect.***

Answer: Any variation of the voltage reference source will translate directly onto the CVREF pin. Also, ensure that you have correctly calculated (specified) the voltage divider which generates the voltage reference. Ensure the TRIS bit for the CVREF pin is a '1' to disable the digital output circuitry, as well.

Question 2: ***I am connecting CVREF into a low-impedance circuit and the voltage reference is not at the expected level.***

Answer: The voltage reference module is not intended to drive large loads. A buffer must be used between the CVREF pin and the load (see Figure 20-2).

20.9 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the PIC32MX device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the Comparator Voltage Reference module are:

Title	Application Note #
No related application notes at this time.	N/A

Note: Please visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the PIC32MX family of devices.

20.10 REVISION HISTORY

Revision A (October 2007)

This is the initial released version of this document.

Revision B (October 2007)

Updated document to remove Confidential status.

Revision C (April 2008)

Revised status to Preliminary; Revised U-0 to r-x.

Revision D (June 2008)

Revised Figure 20-1; Change Reserved bits from “Maintain as” to “Write”; Added Note to ON bit (CVRCON Register).

NOTES:

Section 21. UART

HIGHLIGHTS

This section of the manual contains the following topics:

21.1 INTRODUCTION

The Universal Asynchronous Receiver Transmitter (UART) module is one of the serial I/O modules available in the PIC32MX family of devices. The UART is a full-duplex, asynchronous communication channel that communicates with peripheral devices and personal computers through protocols such as RS-232, RS-485, LIN 1.2 and IrDA®. The module also supports the hardware flow control option, with UxCTS and UxRTS pins, and also includes the IrDA encoder and decoder.

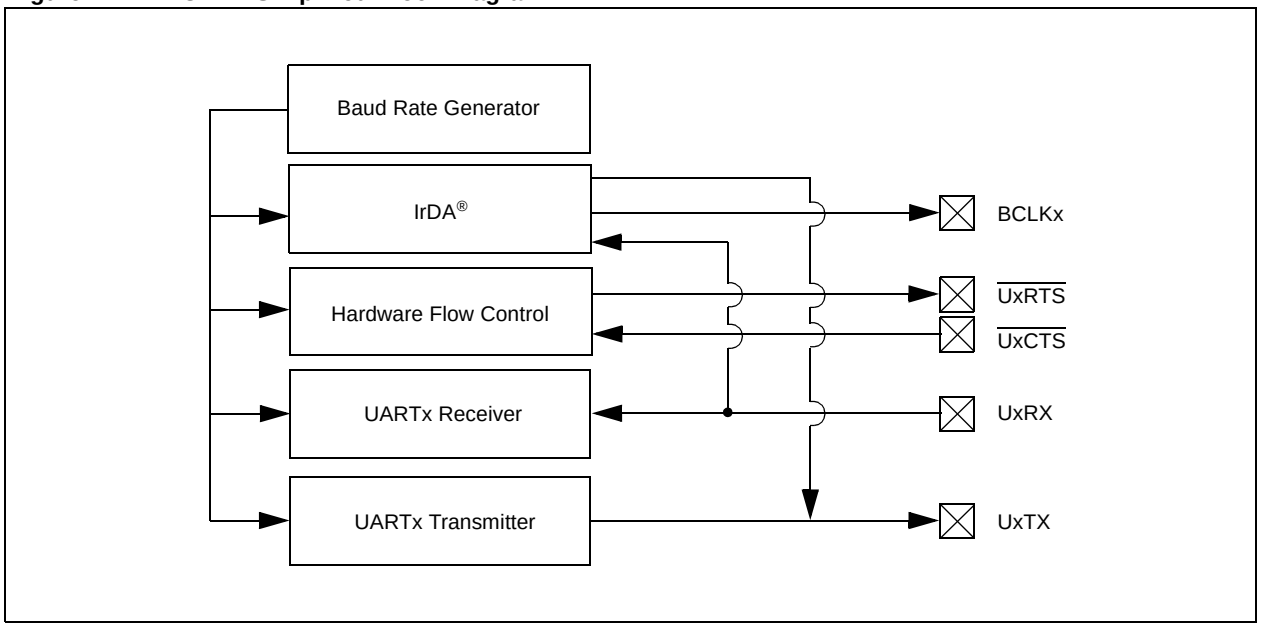
The primary features of the UART module are:

- Full-duplex, 8-bit or 9-bit data transmission
- Even, odd or no parity options (for 8-bit data)
- One or two Stop bits
- Hardware auto-baud feature
- Hardware flow control option
- Fully integrated Baud Rate Generator with 16-bit prescaler
- Baud rates ranging from 76 bps to 20 Mbps at 80 MHz
- 4-level-deep First-In First-Out (FIFO) transmit data buffer
- 4-level-deep FIFO receive data buffer
- Parity, framing and buffer overrun error detection
- Support for interrupt only on address detect (9th bit = 1)
- Separate transmit and receive interrupts
- Loopback mode for diagnostic support
- LIN 1.2 protocol support
- IrDA encoder and decoder with 16x baud clock output for external IrDA encoder/decoder support

A simplified block diagram of the UART is shown in Figure 21-1. The UART module consists of these important hardware elements:

- Baud Rate Generator
- Asynchronous transmitter
- Asynchronous receiver and IrDA encoder/decoder

Figure 21-1: UART Simplified Block Diagram



21.2 CONTROL REGISTERS

Note: Each PIC32MX family device variant may have one or more UART modules. An 'x' used in the names of pins, control/Status bits, and registers denotes the particular module. Refer to the specific device data sheets for more details.

Each UART module consists of the following Special Function Registers (SFRs):

- UxMODE: Control Register for module 'x'
  UxMODECLR, UxMODESET, UxMODEINV: Atomic Bit Manipulation Registers for UxMODE
- UxSTA: Status Register for module 'x'
  UxSTACLR, UxSTASET, UxSTAINV: Atomic Bit Manipulation Registers for UxSTA
- UxTXREG: Transmit Buffer Register for module 'x'
- UxRXREG: Receive Buffer Register for module 'x'
- UxBRG: Baud Rate Generator Register for module 'x'
  UxBRGCLR, UxBRGSET, UxBRGINV: Atomic Bit Manipulation Registers for UxBRG

Each UART module also has the associated bits for interrupt control:

- UxTXIE: Transmit Interrupt Enable Control Bit – in IEC0, IEC1 INT Registers
- UxTXIF: Transmit Interrupt Flag Status Bit – in IFC0, IFC1 INT Registers
- UxRXIE: Receive Interrupt Enable Control Bit – in IEC0, IEC1 INT Registers
- UxRXIF: Receive Interrupt Flag Status Bit – in IFC0, IFC1 INT Registers
- UxEIE: Error Interrupt Enable Control Bit – in IEC0, IEC1 INT Registers
- UxEIF: Error Interrupt Flag Status Bit – in IEC0, IEC1 INT Registers
- UxIP<2:0>Interrupt Priority Control Bits – in IPC6, IPC8 INT Registers
- UxIS<1:0>: Interrupt Subpriority Control Bits – in IPC6, IPC8 INT Registers

The following table summarizes all UART-related registers. Corresponding registers appear after the summary, followed by a detailed description of each register.

Table 21-1: UART SFRs Summary

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
UxMODE	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	ON	FRZ	SIDL	IREN	RTSMD	—	UEN<1:0>	
	7:0	WAKE	LPBACK	ABAUD	RXINV	BRGH	PDSEL<1:0>		STSEL
UxMODECLR	31:0	Write clears selected bits in UxMODE, read yields undefined value							
UxMODESET	31:0	Write sets selected bits in UxMODE, read yields undefined value							
UxMODEINV	31:0	Write inverts selected bits in UxMODE, read yields undefined value							
UxSTA	31:24	—	—	—	—	—	—	—	ADM_EN
	23:16	ADDR<7:0>							
	15:8	UTXISEL0<1:0>		UTXINV	URXEN	UTXBRK	UTXEN	UTXBF	TRMT
	7:0	URXISEL<1:0>		ADDEN	RIDLE	PERR	FERR	OERR	RXDA
UxSTACLR	31:0	Write clears selected bits in UxSTA, read yields undefined value							
UxSTASET	31:0	Write sets selected bits in UxSTA, read yields undefined value							
UxSTAINV	31:0	Write inverts selected bits in UxSTA, read yields undefined value							
UxTXREG	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	UTX8
	7:0	Transmit Register							

Table 21-1: UART SFRs Summary (Continued)

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
UxRXREG	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	RX8
	7:0	Receive Register							
UxBRG	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	BRG <15:8>							
	7:0	BRG<7:0>							
UxBRGCLR	31:0	Write clears selected bits in UxBRG, read yields undefined value							
UxBRGSET	31:0	Write sets selected bits in UxBRG, read yields undefined value							
UxBRGINV	31:0	Write inverts selected bits in UxBRG, read yields undefined value							
IFS0	31:24	I2C1MIF	I2C1SIF	I2C1BIF	U1TXIF	U1RXIF	U1EIF	SPI1RXIF	SPI1TXIF
	23:16	SPI1EIF	OC5IF	IC5IF	T5IF	INT4IF	OC4IF	IC4IF	T4IF
	15:8	INT3IF	OC3IF	IC3IF	T3IF	INT2IF	OC2IF	IC2IF	T2IF
	7:0	INT1IF	OC1IF	IC1IF	T1IF	INT0IF	CS1IF	CS0IF	CTIF
IFS0CLR	31:0	Write clears selected bits in IFSO, read yields undefined value							
IFS0SET	31:0	Write sets the selected bits in IFS0, read yields undefined value							
IFS0INV	31:0	Write inverts the selected bits in IFS, read yields undefined value							
IFS1	31:24	—	—	—	—	—	—	USBIF	FCEIF
	23:16	—	—	—	—	DMA3IF	DMA2IF	DMA1IF	DMA0IF
	15:8	RTCCIF	FSCMIF	I2C2MIF	I2C2SIF	I2C2BIF	U2TXIF	U2RXIF	U2EIF
	7:0	SPI2RXIF	SPI2TXIF	SPI2EIF	CMP2IF	CMP1IF	PMPIF	AD1IF	CNIF
IFS1CLR	31:0	Write clears the selected bits in IFS1, read yields undefined value							
IFS1SET	31:0	Write sets the selected bits in IFS1, read yields undefined value							
IFS1INV	31:0	Write inverts the selected bits in IFS1, read yields undefined value							
IEC0	31:24	I2C1MIE	I2C1SIE	I2C1BIE	U1TXIE	U1RXIE	U1EIE	SPI1RXIE	SPI1TXIE
	23:16	SPI1EIE	OC5IE	IC5IE	T5IE	INT4IE	OC4IE	IC4IE	T4IE
	15:8	INT3IE	OC3IE	IC3IE	T3IE	INT2IE	OC2IE	IC2IE	T2IE
	7:0	INT1IE	OC1IE	IC1IE	T1IE	INT0IE	CS1IE	CS0IE	CTIE
IEC0CLR	31:0	Write clears the selected bits in IEC0, read yields undefined value							
IEC0SET	31:0	Write sets the selected bits in IEC0, read yields undefined value							
IEC0INV	31:0	Write inverts the selected bits in IEC0, read yields undefined value							
IEC1	31:24	—	—	—	—	—	—	USBIE	FCEIE
	23:16	—	—	—	—	DMA3IE	DMA2IE	DMA1IE	DMA0IE
	15:8	RTCCIE	FSCMIE	I2C2MIE	I2C2SIE	I2C2BIE	U2TXIE	U2RXIE	U2EIE
	7:0	SPI2RXIE	SPI2TXIE	SPI2EIE	CMP2IE	CMP1IE	PMPIE	AD1IE	CNIE
IEC1CLR	31:0	Write clears the selected bits in IEC1, read yields undefined value							
IEC1SET	31:0	Write sets the selected bits in IEC1, read yields undefined value							
IEC1INV	31:0	Write inverts the selected bits in IEC1, read yields undefined value							
IPC6	31:24	—	—	—	AD1IP<2:0>			AD1IS<1:0>	
	23:16	—	—	—	CNIP<2:0>			CNIS<1:0>	
	15:8	—	—	—	I2C1IP<2:0>			I2C1IS<1:0>	
	7:0	—	—	—	U1IP<2:0>			U1IS<1:0>	

Table 21-1: UART SFRs Summary (Continued)

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
IPC6CLR	31:0	Write clears the selected bits in IPC6, read yields undefined value							
IPC6SET	31:0	Write sets the selected bits in IPC6, read yields undefined value							
IPC6INV	31:0	Write inverts the selected bits in IPC6, read yields undefined value							
IPC8	31:24	—	—	—	RTCCIP<2:0>			RTCCIS<1:0>	
	23:16	—	—	—	FSCMIP<2:0>			FSCMIS<1:0>	
	15:8	—	—	—	I2C2IP<2:0>			I2C2IS<1:0>	
	7:0	—	—	—	U2IP<2:0>			U2IS<1:0>	
IPC8CLR	31:0	Write clears the selected bits in IPC8, read yields undefined value							
IPC8SET	31:0	Write sets the selected bits in IPC8, read yields undefined value							
IPC8INV	31:0	Write inverts the selected bits in IPC8, read yields undefined value							

Register 21-1: UxMODE: UART 'x' Mode Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	r-x	R/W-0	R/W-0
ON	FRZ	SIDL	IREN	RTSMD	—	UEN<1:0>	
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WAKE	LPBACK	ABAUD	RXINV	BRGH	PDSEL<1:0>		STSEL
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: ('0', '1', x = Unknown)		

bit 31-16 **Reserved:** Write '0'; ignore read

bit 15 **ON:** UARTx Enable bit

1 = UARTx is enabled; UARTx pins are controlled by UARTx as defined by UEN<1:0> and UTXEN control bits

0 = UARTx is disabled, all UARTx pins are controlled by corresponding PORT TRIS and LAT bits; UARTx power consumption is minimal

Note: When using 1:1 PBCLK divisor, the user's software should not read/write the peripheral's SFRs in the SYSCLK cycle immediately following the instruction that clears the module's ON bit.

bit 14 **FRZ:** Freeze in Debug Exception Mode bit

1 = Freeze operation when CPU is in Debug Exception mode

0 = Continue operation when CPU is in Debug Exception mode

Note: FRZ is writable in Debug Exception mode only, it is forced to '0' in Normal mode.

bit 13 **SIDL:** Stop in SLEEP Mode bit

1 = Discontinue operation when device enters in SLEEP mode

0 = Continue operation in SLEEP mode

bit 12 **IREN:** IrDA Encoder and Decoder Enable bit

1 = IrDA is enabled

0 = IrDA is disabled

bit 11 **RTSMD:** Mode Selection for UxRTS Pin bit

1 = UxRTS pin is in Simplex mode

0 = UxRTS pin is in Flow Control mode

bit 10 **Reserved:** Write '0'; ignore read

bit 9-8 **UEN<1:0>:** UARTx Enable bits

11 = UxTX, UxRX, and UxBCLK pins are enabled and used; CTS pin is controlled by PORT latches

10 = UxTX, UxRX, UxCTS, and UxRTS pins are enabled and used

01 = UxTX, UxRX and UxRTS pins are enabled and used; UxCTS pin is controlled by PORT latches

00 = UxTX and UxRX pins are enabled and used; UxCTS and UxRTS/UxBCLK pins are controlled by PORT latches

Register 21-1: UxMODE: UART 'x' Mode Register (Continued)

bit 7 **WAKE:** Enable Wake-up on Start bit Detect During SLEEP Mode bit
1 = Wake-up enabled
0 = Wake-up disabled

bit 6 **LPBACK:** UARTx Loopback Mode Select bit
1 = Enable Loopback mode
0 = Loopback mode is disabled

bit 5 **ABAUD:** Auto-Baud Enable bit
1 = Enable baud rate measurement on the next character – requires reception of Sync character (0x55); cleared by hardware upon completion
0 = Baud rate measurement disabled or completed

bit 4 **RXINV:** Receive Polarity Inversion bit
1 = UxRX IDLE state is '0'
0 = UxRX IDLE state is '1'

bit 3 **BRGH:** High Baud Rate Enable bit
1 = High-Speed mode – 4x baud clock enabled
0 = Standard Speed mode – 16x baud clock enabled

bit 2-1 **PDSEL<1:0>:** Parity and Data Selection bits
11 = 9-bit data, no parity
10 = 8-bit data, odd parity
01 = 8-bit data, even parity
00 = 8-bit data, no parity

bit 0 **STSEL:** Stop Selection bit
1 = 2 Stop bits
0 = 1 Stop bit

Register 21-2: UxMODECLR: UART 'x' Mode Clear Register

Write clears selected bits in UxMODE, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in UxMODE**

A write of '1' in one or more bit positions clears the corresponding bit(s) in UxMODE register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `UxMODECLR = 0x00008001` will clear bits 15 and 0 in UxMODE register.

Register 21-3: UxMODESET: UART 'x' Mode Set Register

Write sets selected bits in UxMODE, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in UxMODE**

A write of '1' in one or more bit positions sets the corresponding bit(s) in UxMODE register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `UxMODE = 0x00008001` will set bits 15 and 0 in UxMODE register.

Register 21-4: UxMODEINV: UART 'x' Mode Invert Register

Write inverts selected bits in UxMODE, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in UxMODE**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in UxMODE register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `UxMODEINV = 0x00008001` will invert bits 15 and 0 in UxMODE register.

Register 21-5: UxSTA: UART 'x' Status and Control Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	R/W-0
—	—	—	—	—	—	—	ADM_EN
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADDR<7:0>							
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-1
UTXISEL0<1:0>		UTXINV	URXEN	UTXBRK	UTXEN	UTXBF	TRMT
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R-1	R-0	R-0	R/W-0	R-0
URXISEL<1:0>		ADDEN	RIDLE	PERR	FERR	OERR	RXDA
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: ('0', '1', x = Unknown)		

bit 31-25 **Reserved:** Write '0'; ignore read

bit 24 **ADM_EN:** Automatic Address Detect Mode Enable bit
1 = Automatic Address Detect mode is enabled
0 = Automatic Address Detect mode is disabled

bit 23-16 **ADDR<7:0>:** Automatic Address Mask bits
When ADM_EN bit is '1', this value defines the address character to use for automatic address detection.

bit 15-14 **UTXISEL0<1:0>:** Tx Interrupt Mode Selection bits
11 = Reserved, do not use
10 = Interrupt is generated when the transmit buffer becomes empty
01 = Interrupt is generated when all characters are transmitted
00 = Interrupt is generated when the transmit buffer contains at least one empty space

bit 13 **UTXINV:** Transmit Polarity Inversion bit
If IrDA mode is disabled (i.e., IREN (UxMODE<12>) is '0')
1 = UxTX IDLE state is '0'
0 = UxTX IDLE state is '1'
If IrDA mode is enabled (i.e., IREN (UxMODE<12>) is '1')
1 = IrDA encoded UxTX IDLE state is '1'
0 = IrDA encoded UxTX IDLE state is '0'

bit 12 **URXEN:** Receiver Enable bit
1 = UARTx receiver is enabled, UxRX pin controlled by UARTx (if ON = 1)
0 = UARTx receiver is disabled, the UxRX pin is ignored by the UARTx module. UxRX pin controlled by port.

bit 11 **UTXBRK:** Transmit Break bit
1 = Send Break on next transmission – Start bit followed by twelve '0' bits, followed by Stop bit; cleared by hardware upon completion
0 = Break transmission is disabled or completed

bit 10 **UTXEN:** Transmit Enable bit
1 = UARTx transmitter enabled, UxTX pin controlled by UARTx (if ON = 1)
0 = UARTx transmitter disabled, any pending transmission is aborted and buffer is reset. UxTX pin controlled by port.

Register 21-5: UxSTA: UART 'x' Status and Control Register (Continued)

bit 9 **UTXBF:** Transmit Buffer Full Status bit (read-only)
1 = Transmit buffer is full
0 = Transmit buffer is not full, at least one more character can be written

bit 8 **TRMT:** Transmit Shift Register is Empty bit (read-only)
1 = Transmit shift register is empty and transmit buffer is empty (the last transmission has completed)
0 = Transmit shift register is not empty, a transmission is in progress or queued in the transmit buffer

bit 7-6 **URXISEL<1:0>:** Receive Interrupt Mode Selection bit
11 = Interrupt flag bit is set when receive buffer is full (i.e., has 4 data characters)
10 = Interrupt flag bit is set when receive buffer is 3/4 full (i.e., has 3 data characters)
0x = Interrupt flag bit is set when a character is received

bit 5 **ADDEN:** Address Character Detect bit (bit 8 of received data = 1)
1 = Address Detect mode enabled. If 9-bit mode is not selected, this control bit has no effect.
0 = Address Detect mode disabled

bit 4 **RIDLE:** Receiver IDLE bit (read-only)
1 = Receiver is IDLE
0 = Data is being received

bit 3 **PERR:** Parity Error Status bit (read-only)
1 = Parity error has been detected for the current character
0 = Parity error has not been detected

bit 2 **FERR:** Framing Error Status bit (read-only)
1 = Framing error has been detected for the current character
0 = Framing error has not been detected

bit 1 **OERR:** Receive Buffer Overrun Error Status bit. This bit is set in hardware, can only be cleared (= 0) in software.
1 = Receive buffer has overflowed
0 = Receive buffer has not overflowed
Note: Clearing a previously set OERR bit resets the receiver buffer and RSR to empty state.

bit 0 **RXDA:** Receive Buffer Data Available bit (read-only)
1 = Receive buffer has data, at least one more character can be read
0 = Receive buffer is empty

Register 21-6: UxSTACLR: UART 'x' Status Clear Register

Write clears selected bits in UxSTA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in UxSTA**

A write of '1' in one or more bit positions clears the corresponding bit(s) in UxSTA register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `UxSTACLR = 0x00008001` will clear bits 15 and 0 in UxSTA register.

Register 21-7: UxSTASET: UART 'x' Status Set Register

Write sets selected bits in UxSTA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in UxSTA**

A write of '1' in one or more bit positions sets the corresponding bit(s) in UxSTA register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `UxSTA = 0x00008001` will set bits 15 and 0 in UxSTA register.

Register 21-8: UxSTAINV: UART 'x' Status Invert Register

Write inverts selected bits in UxSTA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in UxSTA**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in UxSTA register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `UxSTAINV = 0x00008001` will invert bits 15 and 0 in UxSTA register.

Register 21-9: UxTXREG: UART 'x' Transmit Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	R/W-0
—	—	—	—	—	—	—	TX8
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
TX<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: ('0', '1', x = Unknown)		

bit 31-9 **Reserved:** Write '0'; ignore read

bit 8 **TX8:** Data bit 8 of the character to be transmitted (in 9-bit mode)

bit 7-0 **TX<7:0>:** Data bits 7-0 of the character to be transmitted

Register 21-10: UxRXREG: UART 'x' Receive Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	R-0
—	—	—	—	—	—	—	RX8
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
RX<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: ('0', '1', x = Unknown)		

bit 31-9 **Reserved:** Write '0'; ignore read

bit 8 **RX8:** Data bit 8 of the received character (in 9-bit mode)

bit 7-0 **RX<7:0>:** Data bits 7-0 of the received character

Register 21-11: UxBRG: UART 'x' Baud Rate Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
BRG<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
BRG<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: ('0', '1', x = Unknown)		

bit 31-16 **Reserved:** Write '0'; ignore read

bit 15-0 **BRG<15:0>:** Baud Rate Divisor bits

Register 21-12: UxBRGCLR: UART 'x' BRG Clear Register

Write clears selected bits in UxBRG, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in UxBRG**

A write of '1' in one or more bit positions clears the corresponding bit(s) in UxBRG register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `UxBRGCLR = 0x00008001` will clear bits 15 and 0 in UxBRG register.

Register 21-13: UxBRGSET: UART 'x' BRG Set Register

Write sets selected bits in UxBRG, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in UxBRG**

A write of '1' in one or more bit positions sets the corresponding bit(s) in UxBRG register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `UxBRG = 0x00008001` will set bits 15 and 0 in UxBRG register.

Register 21-14: UxBRGINV: UART 'x' BRG Invert Register

Write inverts selected bits in UxBRG, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in UxBRG**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in UxBRG register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `UxBRGINV = 0x00008001` will invert bits 15 and 0 in UxBRG register.

Register 21-15: IFS0: Interrupt Flag Status Register 0(1)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
I2C1MIF	I2C1SIF	I2C1BIF	U1TXIF	U1RXIF	U1EIF	SPI1RXIF	SPI1TXIF
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SPI1EIF	OC5IF	IC5IF	T5IF	INT4IF	OC4IF	IC4IF	T4IF
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INT3IF	OC3IF	IC3IF	T3IF	INT2IF	OC2IF	IC2IF	T2IF
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INT1IF	OC1IF	IC1IF	T1IF	INT0IF	CS1IF	CS0IF	CTIF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 28 **U1TXIF:** UART 1 Transmitter Interrupt Request Flag bit
1 = Interrupt request has occurred
0 = No interrupt request has occurred

bit 27 **U1RXIF:** UART 1 Receiver Interrupt Request Flag bit
1 = Interrupt request has occurred
0 = No interrupt request has occurred

bit 26 **U1EIF:** UART 1 Error Interrupt Request Flag bit
1 = Interrupt request has occurred
0 = No interrupt request has occurred

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the UART.

Register 21-16: IFS1: Interrupt Flag Status Register 1(1)

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	USBIF	FCEIF
bit 31							bit 24

r-0	r-0	r-0	r-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	DMA3IF	DMA2IF	DMA1IF	DMA0IF
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RTCCIF	FSCMIF	I2C2MIF	I2C2SIF	I2C2BIF	U2TXIF	U2RXIF	U2EIF
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SPI2RXIF	SPI2TXIF	SPI2EIF	CMP2IF	CMP1IF	PMPIF	AD1IF	CNIF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 10 **U2TXIF:** UART 2 Transmitter Interrupt Request bit
1 = Interrupt request has occurred
0 = No interrupt request has occurred

bit 9 **U2RXIF:** UART 2 Receiver Interrupt Request Flag bit
1 = Interrupt request has occurred
0 = No interrupt request has occurred

bit 8 **U2EIF:** UART 2 Error Interrupt Request bit
1 = Interrupt request has occurred
0 = No interrupt request has occurred

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the UART.

Register 21-17: IEC0: Interrupt Enable Control Register 0(1)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
I2C1MIE	I2C1SIE	I2C1BIE	U1TXIE	U1RXIE	U1EIE	SPI1RXIE	SPI1TXIE
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SPI1EIE	OC5IE	IC5IE	T5IE	INT4IE	OC4IE	IC4IE	T4IE
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INT3IE	OC3IE	IC3IE	T3IE	INT2IE	OC2IE	IC2IE	T2IE
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INT1IE	OC1IE	IC1IE	T1IE	INT0IE	CS1IE	CS0IE	CTIE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 28 **U1TXIE:** UART 1 Transmitter Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 27 **U1RXIE:** UART 1 Receiver Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 26 **U1EIE:** UART 1 Error Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the UART.

Register 21-18: IEC1: Interrupt Enable Control Register 1(1)

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	USBIE	FCEIE
bit 31							bit 24

r-0	r-0	r-0	r-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	DMA3IE	DMA2IE	DMA1IE	DMA0IE
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RTCCIE	FSCMIE	I2C2MIE	I2C2SIE	I2C2BIE	U2TXIE	U2RXIE	U2EIE
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SPI2RXIE	SPI2TXIE	SPI2EIE	CMP2IE	CMP1IE	PMPIE	AD1IE	CNIE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 10 **U2TXIE:** UART 2 Transmitter Interrupt Request bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 9 **U2RXIE:** UART 2 Receiver Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 8 **U2EIE:** UART 2 Error Interrupt Request bit
1 = Interrupt is enabled
0 = Interrupt is disabled

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the UART.

Register 21-19: IPC6; Interrupt Priority Control Register 6(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	AD1IP<2:0>			AD1IS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	CNIP<2:0>			CNIS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	I2C1IP<2:0>			I2C1IS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	U1IP<2:0>			U1IS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 4-2 **U1IP<2:0>:** UART 1 Interrupt Priority bits

111 = Interrupt Priority is 7
110 = Interrupt Priority is 6
101 = Interrupt Priority is 5
100 = Interrupt Priority is 4
011 = Interrupt Priority is 3
010 = Interrupt Priority is 2
001 = Interrupt Priority is 1
000 = Interrupt is disabled

bit 1-0 **U1IS<1:0>:** UART 1 Interrupt Subpriority bits

11 = Interrupt Subpriority is 3
10 = Interrupt Subpriority is 2
01 = Interrupt Subpriority is 1
00 = Interrupt Subpriority is 0

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the UART.

Register 21-20: IPC8: Interrupt Priority Control Register(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	RTCCIP<2:0>			RTCCIS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	FSCMIP<2:0>			FSCMIS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	I2C2IP<2:0>			I2C2IS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	U2IP<2:0>			U2IS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 4-2 **U2IP<2:0>:** UART 2 Interrupt Priority bits

111 = Interrupt Priority is 7
110 = Interrupt Priority is 6
101 = Interrupt Priority is 5
100 = Interrupt Priority is 4
011 = Interrupt Priority is 3
010 = Interrupt Priority is 2
001 = Interrupt Priority is 1
000 = Interrupt is disabled

bit 1-0 **U2IS<1:0>:** UART 2 Subpriority bits

11 = Interrupt Subpriority is 3
10 = Interrupt Subpriority is 2
01 = Interrupt Subpriority is 1
00 = Interrupt Subpriority is 0

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the UART.

21.3 UART BAUD RATE GENERATOR

The UART module includes a dedicated 16-bit Baud Rate Generator. The UxBRG register controls the period of a free-running 16-bit timer. Equation 21-1 shows the formula for computation of the baud rate with BRGH = 0.

Equation 21-1: UART Baud Rate with BRGH = 0

$$\text{Baud Rate} = \frac{F_{PB}}{16 \bullet (\text{UxBRG} + 1)}$$

$$\text{UxBRG} = \frac{F_{PB}}{16 \bullet \text{Baud Rate}} - 1$$

Note: F_{PB} denotes the PBCLK frequency.

Example 21-1 shows the calculation of the baud rate error for the following conditions:

- F_{PB} = 4 MHz
- Desired Baud Rate = 9600

Example 21-1: Baud Rate Error Calculation (BRGH = 0)

```
Desired Baud Rate     =   FPB/(16 (UxBRG + 1))
Solving for UxBRG value:
    UxBRG             =   ( (FPB/Desired Baud Rate)/16) – 1
    UxBRG             =   ((4000000/9600)/16) – 1
    UxBRG             =   [25.042] = 25
Calculated Baud Rate  =   4000000/(16 (25 + 1))
                      =   9615
Error                 =   (Calculated Baud Rate – Desired Baud Rate)
           Desired Baud Rate
                      =   (9615 – 9600)/9600
                      =   0.16%
```

The maximum possible baud rate (BRGH = 0) is F_{PB}/16 (for UxBRG = 0), and the minimum possible baud rate is F_{PB} /16 * 65536).

Equation 21-2 shows the formula for computation of the baud rate with BRGH = 1.

Equation 21-2: UART Baud Rate with BRGH = 1

$$\text{Baud Rate} = \frac{F_{PB}}{4 \bullet (\text{UxBRG} + 1)}$$

$$\text{UxBRG} = \frac{F_{PB}}{4 \bullet \text{Baud Rate}} - 1$$

Note: F_{PB} denotes the PBCLK frequency.

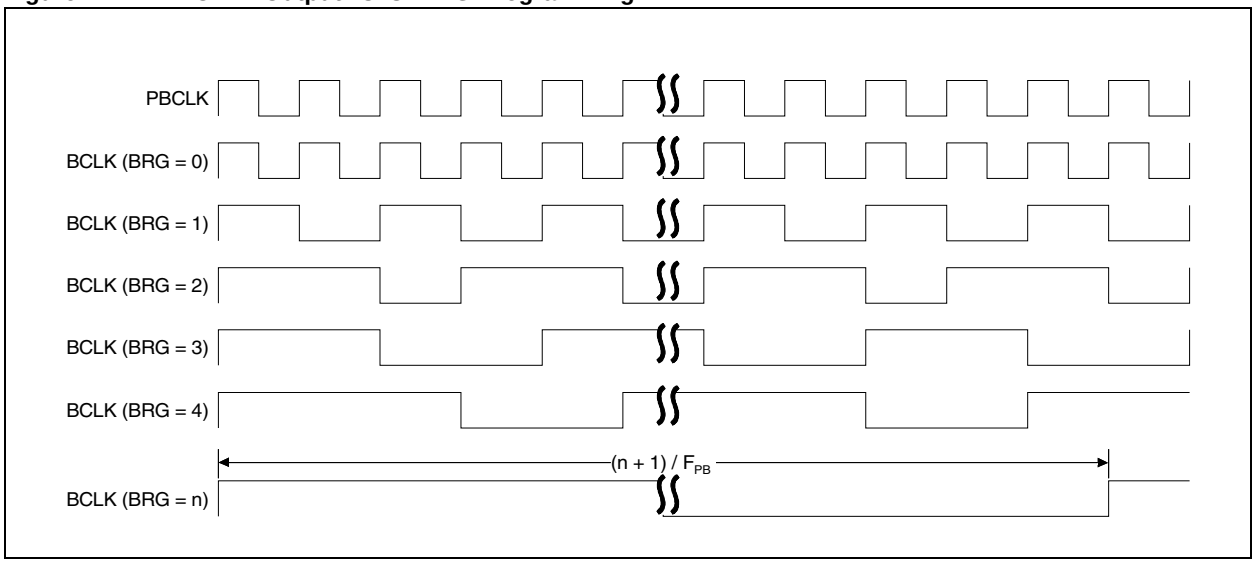
The maximum possible baud rate (BRGH = 1) is F_{PB} /4 (for UxBRG = 0), and the minimum possible baud rate is F_{PB}/(4 * 65536).

Writing a new value to the UxBRG register causes the baud rate counter to reset (clear). This ensures that the BRG does not wait for a timer overflow before it generates the new baud rate.

21.3.1 BCLKx Output

The BCLKx pin outputs the 16x baud clock if the UART and BCLKx output are enabled (UxMODE.UEN<1:0> = 11). This feature is used for external IrDA encoder/decoder support (refer to Figure 21-2). BCLKx output stays low during SLEEP mode. BCLKx is forced as an output as long as UART is kept in this mode (UxMODE.UEN<1:0> = 11), irrespective of PORTx and TRISx latch bits.

Figure 21-2: BCLKx Output vs. UxBRG Programming



21.3.2 Baud Rate Tables

UART baud rates are provided in Table 21-2 for common peripheral bus frequencies (F_{PB}). The minimum and maximum baud rates for each frequency are also provided.

Table 21-2: UART Baud Rates (UxMODE.BRGH = '0', no PLL)

Target Baud Rate	Peripheral Bus Clock: 40 MHz			Peripheral Bus Clock: 33 MHz			Peripheral Bus Clock: 30 MHz		
	Actual Baud Rate	% Error	BRG Value (decimal)	Actual Baud Rate	% Error	BRG Value (decimal)	Actual Baud Rate	% Error	BRG Value (decimal)
110	110.0	0.00%	22726.0	110.0	0.0%	18749.0	110.0	0.0%	17044.0
300	300.0	0.00%	8332.0	300.0	0.0%	6874.0	300.0	0.0%	6249.0
1200	1200.2	0.02%	2082.0	1199.8	0.0%	1718.0	1199.6	0.0%	1562.0
2400	2399.2	-0.03%	1041.0	2401.0	0.0%	858.0	2400.8	0.0%	780.0
9600	9615.4	0.16%	259.0	9593.0	-0.1%	214.0	9615.4	0.2%	194.0
19.2 K	19230.8	0.16%	129.0	19275.7	0.4%	106.0	19132.7	-0.4%	97.0
38.4 K	38461.5	0.16%	64.0	38194.4	-0.5%	53.0	38265.3	-0.4%	48.0
56 K	55555.6	-0.79%	44.0	55743.2	-0.5%	36.0	56818.2	1.5%	32.0
115 K	113636.4	-1.19%	21.0	114583.3	-0.4%	17.0	117187.5	1.9%	15.0
250 K	250000.0	0.00%	9.0	257812.5	3.1%	7.0			
300 K				294642.9	-1.8%	6.0			
500 K	500000.0	0.00%	4.0	515625.0	3.1%	3.0			
Min. Rate	38.1	0.0%	65535	31.5	0.0%	65535	28.6	0.0%	65535
Max. Rate	2500000	0.0%	0	2062500	0.0%	0	1875000	0.0%	0

Target Baud Rate	Peripheral Bus Clock: 25 MHz			Peripheral Bus Clock: 20 MHz			Peripheral Bus Clock: 18.432 MHz		
	Actual Baud Rate	% Error	BRG Value (decimal)	Actual Baud Rate	% Error	BRG Value (decimal)	Actual Baud Rate	% Error	BRG Value (decimal)
110	110.0	0.00%	14204.0	110.0	0.0%	11363.0	110.0	0.0%	10472.0
300	300.0	0.01%	5207.0	300.0	0.0%	4166.0	300.0	0.0%	3839.0
1200	1200.1	0.01%	1301.0	1199.6	0.0%	1041.0	1200.0	0.0%	959.0
2400	2400.2	0.01%	650.0	2399.2	0.0%	520.0	2400.0	0.0%	479.0
9600	9585.9	-0.15%	162.0	9615.4	0.2%	129.0	9600.0	0.0%	119.0
19.2 K	19290.1	0.47%	80.0	19230.8	0.2%	64.0	19200.0	0.0%	59.0
38.4 K	38109.8	-0.76%	40.0	37878.8	-1.4%	32.0	38400.0	0.0%	29.0
56 K	55803.6	-0.35%	27.0	56818.2	1.5%	21.0	54857.1	-2.0%	20.0
115 K	111607.1	-2.95%	13.0	113636.4	-1.2%	10.0	115200.0	0.2%	9.0
250 K				250000.0	0.0%	4.0			
300 K									
500 K									
Min. Rate	23.8	0.0%	65535	19	0.0%	65535	18	0.0%	65535
Max. Rate	1562500	0.0%	0	1250000	0.0%	0	1152000	0.0%	0

Target Baud Rate	Peripheral Bus Clock: 16 MHz			Peripheral Bus Clock: 12 MHz			Peripheral Bus Clock: 10 MHz		
	Actual Baud Rate	% Error	BRG Value (decimal)	Actual Baud Rate	% Error	BRG Value (decimal)	Actual Baud Rate	% Error	BRG Value (decimal)
110	110.0	0.00%	9090.0	110.0	0.0%	6817.0	110.0	0.0%	5681.0
300	300.0	0.01%	3332.0	300.0	0.0%	2499.0	300.0	0.0%	2082.0
1200	1200.5	0.04%	832.0	1200.0	0.0%	624.0	1199.6	0.0%	520.0
2400	2398.1	-0.08%	416.0	2396.2	-0.2%	312.0	2403.8	0.2%	259.0
9600	9615.4	0.16%	103.0	9615.4	0.2%	77.0	9615.4	0.2%	64.0
19.2 K	19230.8	0.16%	51.0	19230.8	0.2%	38.0	18939.4	-1.4%	32.0
38.4 K	38461.5	0.16%	25.0	37500.0	-2.3%	19.0	39062.5	1.7%	15.0
56 K	55555.6	-0.79%	17.0	57692.3	3.0%	12.0	56818.2	1.5%	10.0
115 K	111111.1	-3.38%	8.0			6.0			
250 K	250000.0	0.00%	3.0	250000.0	0.0%	2.0			
300 K									
500 K	500000.0	0.00%	1.0						
Min. Rate	15	0.0%	65535	11	0.0%	65535	10	0.0%	65535
Max. Rate	1000000	0.0%	0	750000	0.0%	0	625000	0.0%	0

Table 21-2: UART Baud Rates (UxMODE.BRGH = '0', no PLL) (Continued)

Target Baud Rate	Peripheral Bus Clock: 8 MHz			Peripheral Bus Clock: 5 MHz			Peripheral Bus Clock: 4 MHz		
	Actual Baud Rate	% Error	BRG Value (decimal)	Actual Baud Rate	% Error	BRG Value (decimal)	Actual Baud Rate	% Error	BRG Value (decimal)
110	110.0	0.01%	4544.0	110.0	0.0%	2840.0	110.0	0.0%	2272.0
300	299.9	-0.02%	1666.0	299.9	0.0%	1041.0	300.1	0.0%	832.0
1200	1199.0	-0.08%	416.0	1201.9	0.2%	259.0	1201.9	0.2%	207.0
2400	2403.8	0.16%	207.0	2403.8	0.2%	129.0	2403.8	0.2%	103.0
9600	9615.4	0.16%	51.0	9469.7	-1.4%	32.0	9615.4	0.2%	25.0
19.2 K	19230.8	0.16%	25.0	19531.3	1.7%	15.0	19230.8	0.2%	12.0
38.4 K	38461.5	0.16%	12.0	39062.5	1.7%	7.0			
56 K	55555.6	-0.79%	8.0						
115 K									
250 K	250000.0	0.00%	1.0						
300 K									
500 K	500000.0	0.00%	0.0						
Min. Rate	8	0.0%	65535	5	0.0%	65535	4	0.0%	65535
Max. Rate	500000	0.0%	0	312500	0.0%	0	250000	0.0%	0

Target Baud Rate	Peripheral Bus Clock: 7.68 MHz			Peripheral Bus Clock: 7.15909 MHz			Peripheral Bus Clock: 5.0688 MHz		
	Actual Baud Rate	% Error	BRG Value (decimal)	Actual Baud Rate	% Error	BRG Value (decimal)	Actual Baud Rate	% Error	BRG Value (decimal)
110	110.0	-0.01%	4363.0	110.0	0.0%	4067.0	110.0	0.0%	2879.0
300	300.0	0.00%	1599.0	300.1	0.0%	1490.0	300.0	0.0%	1055.0
1200	1200.0	0.00%	399.0	1199.6	0.0%	372.0	1200.0	0.0%	263.0
2400	2400.0	0.00%	199.0	2405.6	0.2%	185.0	2400.0	0.0%	131.0
9600	9600.0	0.00%	49.0	9520.1	-0.8%	46.0	9600.0	0.0%	32.0
19.2 K	19200.0	0.00%	24.0	19454.0	1.3%	22.0	18635.3	-2.9%	16.0
38.4 K	36923.1	-3.85%	12.0	37286.9	-2.9%	11.0	39600.0	3.1%	7.0
56 K	53333.3	-4.76%	8.0	55930.4	-0.1%	7.0			
115 K	120000.0	4.35%	3.0	111860.8	-2.7%	3.0			
250 K	240000.0	-4.00%	1.0						
300 K									
500 K									
Min. Rate	7	0.0%	65535	7	0.0%	65535	5	0.0%	65535
Max. Rate	480000	0.0%	0	447443	0.0%	0	316800	0.0%	0

Target Baud Rate	Peripheral Bus Clock: 3.579545 MHz			Peripheral Bus Clock: 3.072 MHz			Peripheral Bus Clock: 1.8432 MHz		
	Actual Baud Rate	% Error	BRG Value (decimal)	Actual Baud Rate	% Error	BRG Value (decimal)	Actual Baud Rate	% Error	BRG Value (decimal)
110	110.0	-0.01%	2033.0	110.0	0.0%	1744.0	110.0	0.0%	1046.0
300	299.9	-0.04%	745.0	300.0	0.0%	639.0	300.0	0.0%	383.0
1200	1202.8	0.23%	185.0	1200.0	0.0%	159.0	1200.0	0.0%	95.0
2400	2405.6	0.23%	92.0	2400.0	0.0%	79.0	2400.0	0.0%	47.0
9600	9727.0	1.32%	22.0	9600.0	0.0%	19.0	9600.0	0.0%	11.0
19.2 K	18643.5	-2.90%	11.0	19200.0	0.0%	9.0	19200.0	0.0%	5.0
38.4 K	37286.9	-2.90%	5.0	38400.0	0.0%	4.0	38400.0	0.0%	2.0
56 K	55930.4	-0.12%	3.0						
115 K	111860.8	-2.73%	1.0						
250 K									
300 K									
500 K									
Min. Rate	3	0.0%	65535	3	0.0%	65535	2	0.0%	65535
Max. Rate	223722	0.0%	0	192000	0.0%	0	115200	0.0%	0

21.4 UART CONFIGURATION

The UART uses standard non-return-to-zero (NRZ) format (one Start bit, eight or nine data bits, and one or two Stop bits). Parity is supported by the hardware, and may be configured by the user as even, odd or no parity. The most common data format is 8 bits, no parity and one Stop bit (denoted as 8, N, 1), which is the default Power-on Reset (POR) setting. The number of data bits and Stop bits, and the parity, are specified in the PDSEL<1:0> (UxMODE<2:1>) and STSEL (UxMODE<0>) bits. An on-chip dedicated 16-bit Baud Rate Generator is used to derive standard baud rate frequencies from the oscillator. The UART transmits and receives the Least Significant bit (LSb) first. The UART's transmitter and receiver are functionally independent, but use the same data format and baud rate.

21.4.1 Enabling the UART

The UART module is enabled by setting the bits ON (UxMODE<15>), URXEN (UxSTA<12>), and UTXEN (UxSTA<10>). Once enabled, the UxTX and UxRX pins are configured as an output and an input, respectively, overriding the TRIS and PORT register bit settings for the corresponding I/O port pins. The UxTX pin is at logic '1' when a transmission is not taking place.

21.4.2 Disabling the UART

The UART module is disabled by clearing the ON bit. This is the default state after any Reset. If the UART is disabled, all UART pins operate as port pins under the control of their corresponding PORT and TRIS bits.

Disabling the UART module resets the buffers to empty states. Any data characters in the buffers are lost, and the baud rate counter is reset.

All error and status flags associated with the UART module are reset when the module is disabled. The RXDA, OERR, FERR, PERR, UTXEN, URXEN, UTXBRK and UTXBF bits in the UxSTA register are cleared, whereas the RIDLE and TRMT bits are set. Other control bits (including ADDEN, RXISEL<1:0> and UTXISEL0), as well as UxMODE and UxBRG registers, are not affected.

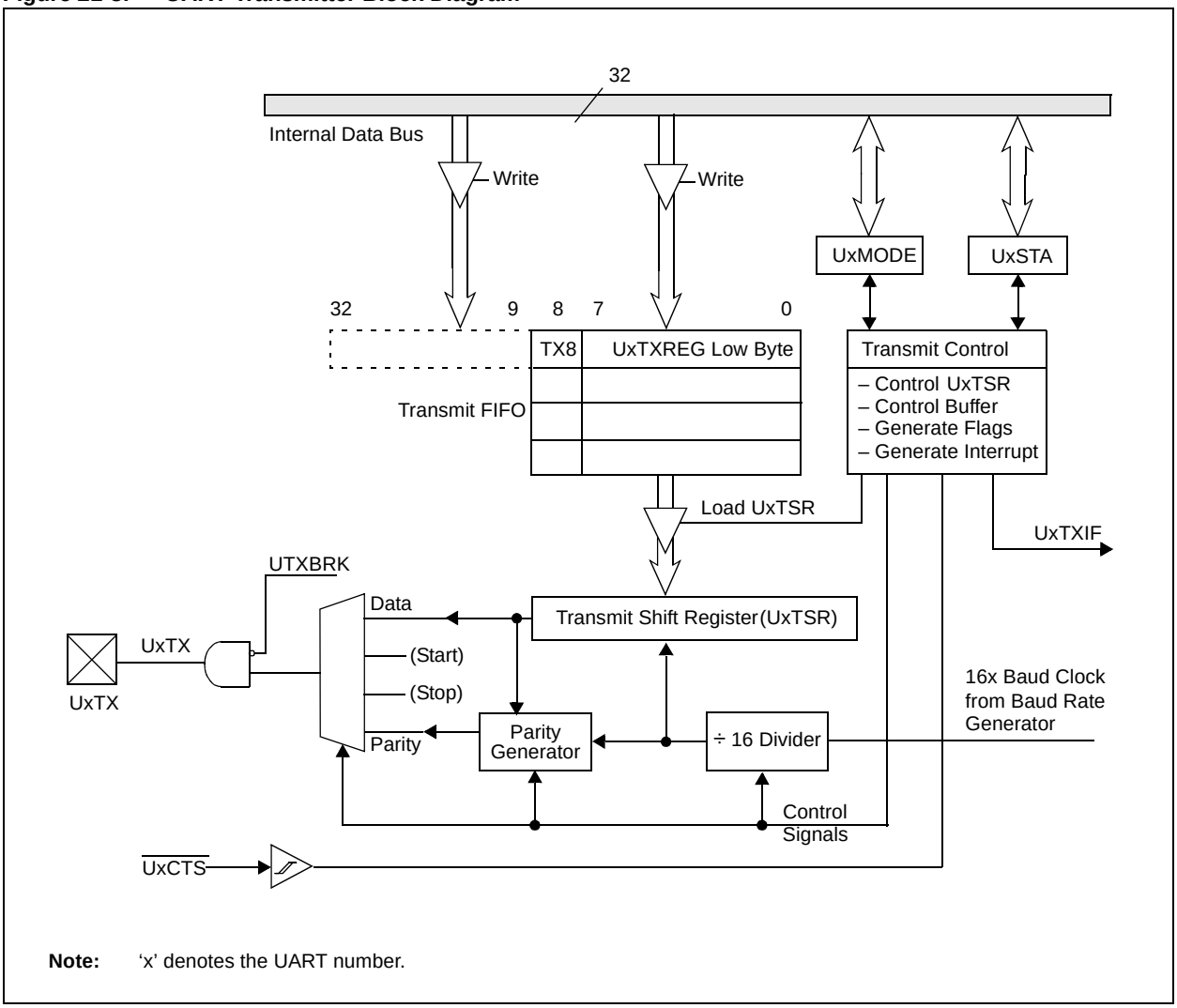
Clearing the ON bit while the UART is active aborts all pending transmissions and receptions and resets the module as defined above. Re-enabling the UART restarts the UART in the same configuration.

21.5 UART TRANSMITTER

Figure 21-3 shows the UART transmitter block diagram. The heart of the transmitter is the Transmit Shift register (UxTSR). UxTSR obtains its data from the transmit FIFO buffer, UxTXREG. The UxTXREG register is loaded with data in software. The UxTSR register is not loaded until the Stop bit is transmitted from the previous load. As soon as the Stop bit is transmitted, the UxTSR is loaded with new data from the UxTXREG register (if available).

Note:	The UxTSR register is not mapped in memory, so it is not available to the user.

Figure 21-3: UART Transmitter Block Diagram



Note: 'x' denotes the UART number.

Transmission is enabled by setting the UTXEN enable bit (UxSTA<10>). The actual transmission will not occur until the UxTXREG register is loaded with data and the Baud Rate Generator UxBRG has produced a shift clock (see Figure 21-3). The transmission can also be started by first loading the UxTXREG register and then setting the UTXEN enable bit. Normally, when transmission is initially started, the UxTSR register is empty, so a transfer to the UxTXREG register results in an immediate transfer to the UxTSR. Clearing the UTXEN bit during a transmission causes the transmission to be aborted and resets the transmitter. As a result, the UxTX pin reverts to a high-impedance state.

To select 9-bit transmission, the PDSEL<1:0> bits, in the UxMODE<2:1>, should be set to '11' and the ninth bit should be written to the UTX8 bit (UxTXREG<8>). A word write should be performed to the UxTXREG so that all nine bits are written at the same time.

Note:	There is no parity in the case of 9-bit data transmission.

21.5.1 Transmit Buffer (UxTXREG)

The transmit buffer is 9 bits wide and 4 levels deep. Together with the Transmit Shift registers (UxTSR), the user effectively has a 5-level-deep buffer. It is organized as FIFO. When the UxTXREG contents are transferred to the UxTSR register, the current buffer location becomes available for new data to be written, and the next buffer location is sourced to the UxTSR register. The UTXBF (UxSTA<9>) Status bit is set whenever the buffer is full. If a user attempts to write to a full buffer, the new data will not be accepted into the FIFO.

The FIFO is reset during any device Reset, but is not affected when the device enters a Power-Saving mode or wakes up from a Power-Saving mode.

21.5.2 Transmit Interrupt

The transmit interrupt flag (UxTXIF) is located in the corresponding interrupt flag status (IFS) register. The UTXISEL0 control bit (UxSTA<15:14>) determines when the UART will generate a transmit interrupt.

1. UTXISEL0<1:0> = 00, the UxTXIF is set when a character is transferred from the transmit buffer to the Transmit Shift register (UxTSR). This implies at least one location is empty in the transmit buffer.
2. UTXISEL0<1:0> = 01, the UxTXIF is set when the last character is shifted out of the Transmit Shift register (UxTSR). This implies that all the transmit operations are completed.
3. UTXISEL0<1:0> = 10, the UxTXIF is set when the character is transferred to the Transmit Shift register (UxTSR) and the transmit buffer is empty.

The UxTXIF bit is set when the module is first enabled. The user should clear the UxTXIF bit in the ISR.

Switching between the two Interrupt modes during operation is possible.

Note:	When the UTXEN bit is set, the UxTXIF flag bit is also set if UTXISEL0<1:0> = 00 (since the transmit buffer is not yet full, i.e.,transmit data can move to the UxTXREG register).

While the UxTXIF flag bit indicates the status of the UxTXREG register, the TRMT bit (UxSTA<8>) shows the status of the UxTSR register. The TRMT Status bit is a read-only bit, which is set when the UxTSR register is empty. No interrupt logic is tied to this bit, so the user has to poll this bit to determine if the UxTSR register is empty.

21.5.3 Setup for UART Transmit

Use the following steps to set up a UART transmission:

1. Initialize the UxBRG register for the appropriate baud rate (refer to **Section 21.3 “UART Baud Rate Generator”**).
2. Set the number of data bits, number of Stop bits, and parity selection by writing to the PDSEL<1:0> (UxMODE<2:1>) and STSEL (UxMODE<0>) bits.
3. If transmit interrupts are desired, set the UxTXIE control bit in the corresponding Interrupt Enable Control register (IEC). Specify the interrupt priority and subpriority for the transmit interrupt using the UxIP<2:0> and UxIS<1:0> control bits in the corresponding Interrupt Priority Control register (IPC). Also, select the Transmit Interrupt mode by writing the UTXISEL0 (UxSTA<15:14>) bits.
4. Enable the UART module by setting the ON (UxMODE<15>) bit.
5. Enable the transmission by setting the UTXEN (UxSTA<10>) bit, which also sets the UxTXIF bit. The UxTXIF bit should be cleared in the software routine that services the UART transmit interrupt. The operation of the UxTXIF bit is controlled by the UTXISEL0 control bits.
6. Load data to the UxTXREG register (starts transmission). If 9-bit transmission is selected, load a word. If 8-bit transmission is used, load a byte. Data can be loaded into the buffer until the TXBF Status bit (UxSTA<9>) is set.

Note:	The UTXEN bit should not be set until the ON bit has been set. Otherwise, UART transmissions will not be enabled.

Figure 21-4: Transmission (8-Bit or 9-Bit Data)

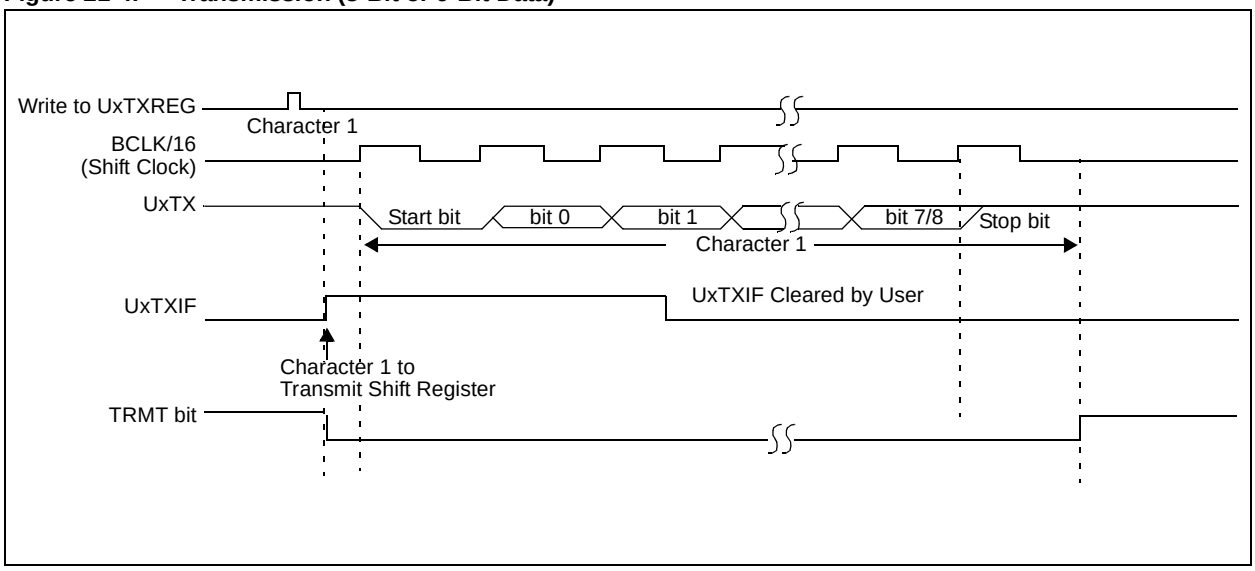
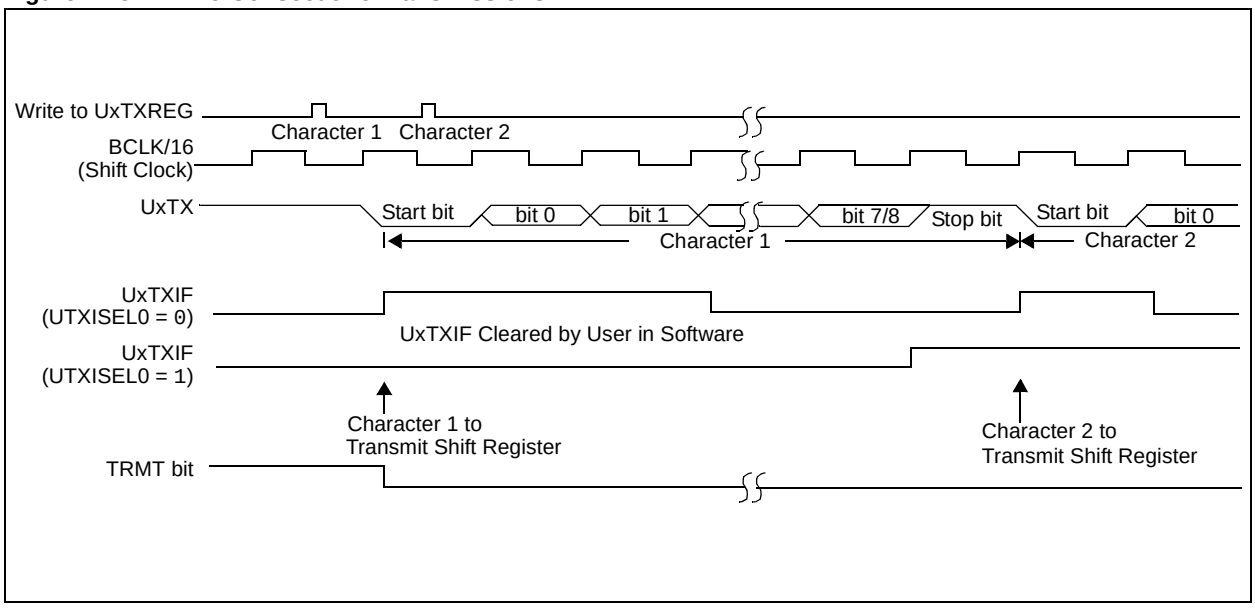


Figure 21-5: Two Consecutive Transmissions



21.5.4 Transmission of Break Characters

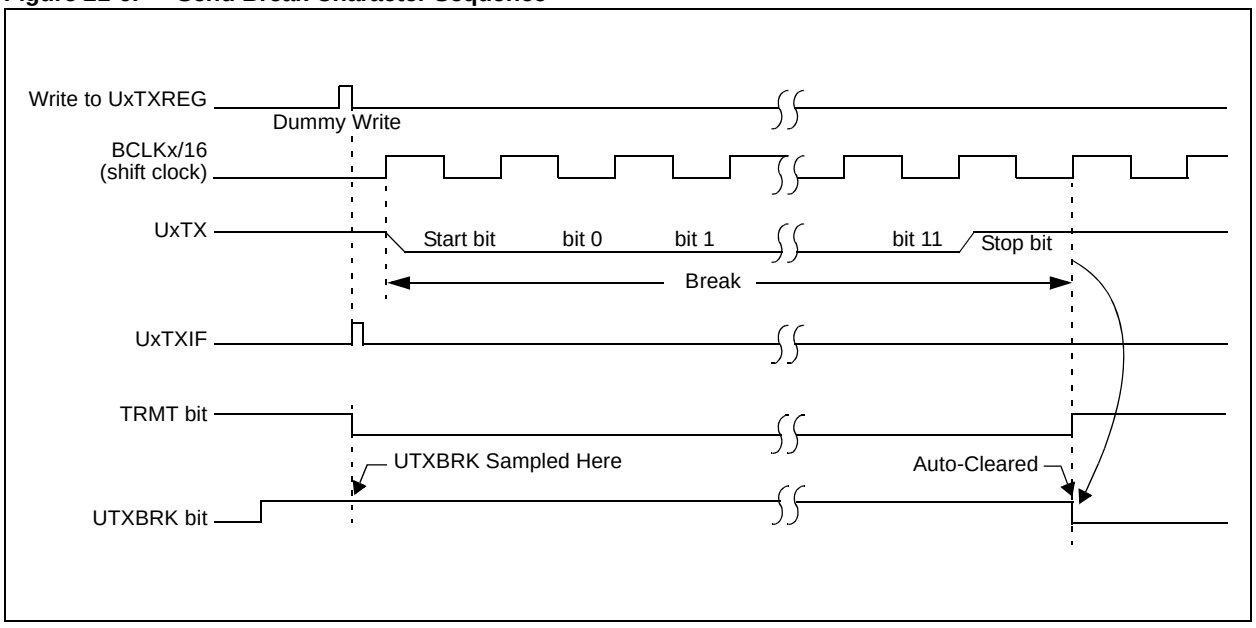
A Break character transmit consists of a Start bit, followed by twelve bits of '0' and a Stop bit. A Frame Break character is sent whenever the UTXBRK and UTXEN bits are set while the Transmit Shift register is loaded with data. A dummy write to the UxTXREG register is necessary to initiate the Break character transmission. Note that the data value written to the UxTXREG for the Break character is ignored. The write merely initiates the proper sequence, so that all zeroes are transmitted.

The UTXBRK bit is automatically reset by hardware after the corresponding Stop bit is sent. This allows the user to preload the transmit FIFO with the next transmit byte following the Break character (typically, the Sync character in the LIN specification).

Note:	The user should wait for the transmitter to be IDLE (TRMT = 1) before setting the UTXBRK. The UTXBRK overrides any other transmitter activity. If the user clears the UTXBRK bit prior to sequence completion, unexpected module behavior can result. Sending a Break character does not generate a transmit interrupt.

The TRMT bit indicates whether the Transmit Shift register is empty or full, just as it does during normal transmission. See Figure 21-6 for the timing of the Break character sequence.

Figure 21-6: Send Break Character Sequence



21.5.5 Break and Sync Transmit Sequence

The following sequence is performed to send a message frame header that is composed of a Break character, followed by an auto-baud Sync byte. This sequence is typical of a LIN bus master.

1. Configure the UART for the desired mode, refer to **Section 21.5.3 "Setup for UART Transmit"**for set up information.
2. Set UTXEN and UTXBRK to set up the Break character.
3. Load the UxTXREG with a dummy character to initiate transmission (value is ignored).
4. Write '0x55' to UxTXREG to load the Sync character into the transmit FIFO.

After the Break is sent, the UTXBRK bit is reset by hardware. The Sync character now transmits.

21.6 UART RECEIVER

The heart of the receiver is the Receive (Serial) Shift register (UxRSR). The data is received on the UxRX pin and is sent to the data recovery block. The data recovery block operates at 16 times the baud rate, whereas the main receive serial shifter operates at the baud rate. After sampling the UxRX pin for the Stop bit, the received data in UxRSR is transferred to the receive FIFO, if it is empty. See Figure 21-7 for a UART receiver block diagram.

Note:	The UxRSR register is not mapped in memory, so it is not available to the user.

Reception is enabled by setting the URXEN bit (UxSTA<12>). The data on the UxRX pin is sampled three times by a majority detect circuit to determine whether a high or a low level is present at the UxRX pin.

21.6.0.1 Receive Buffer (UxRXREG)

The UART receiver has a 4-level deep, 9-bit wide FIFO receive data buffer. UxRXREG is a memory mapped register that provides access to the output of the FIFO. It is possible for four words of data to be received and transferred to the FIFO and a fifth word to begin shifting to the UxRSR register before a buffer overrun occurs.

21.6.0.2 Receiver Error Handling

If the FIFO is full (contains four characters) and a fifth character is fully received into the UxRSR register, the overrun error bit OERR (UxSTA<1>) is set. The word in UxRSR will be kept, but further transfers to the receive FIFO are inhibited as long as the OERR bit is set. The user must clear the OERR bit in software to allow further data to be received.

To keep the data received prior to the overrun, the user should first read all five characters, then clear the OERR bit. If the five characters can be discarded, the user can simply clear the OERR bit. This effectively resets the receive FIFO, and all prior received data is lost.

Note:	The data in the receive FIFO should be read prior to clearing the OERR bit. The FIFO is reset when OERR is cleared, which causes all data in the buffer to be lost.

The framing error bit FERR (UxSTA<2>) is set when a Stop bit is detected as a logic low level.

The parity error bit PERR (UxSTA<3>) is set if a parity error has been detected in the data word at the top of the buffer (i.e., the current word). For example, a parity error occurs if the parity is set as even, but the total number of ones in the data has been detected as odd. The PERR bit is irrelevant in the 9-bit mode. The FERR and PERR bits are buffered along with the corresponding word and should be read before reading the data word.

21.6.0.3 Receive Interrupt

The UART receive interrupt flag (UxRXIF) is located in the corresponding Interrupt Flag Status (IFSx) register. The RXISEL<1:0> (UxSTA<7:6>) control bits determine when the UART receiver generates an interrupt.

1. If RXISEL<1:0> = 00 or 01, an interrupt is generated each time a data word is transferred from the Receive Shift register (UxRSR) to the receive buffer. There may be one or more characters in the receive buffer.
2. If RXISEL<1:0> = 10, an interrupt is generated when a word is transferred from the Receive Shift register (UxRSR) to the receive buffer and as a result, the receive buffer contains three or four characters.
3. If RXISEL<1:0> = 11, an interrupt is generated when a word is transferred from the Receive Shift register (UxRSR) to the receive buffer and as a result, the receive buffer contains four characters, i.e., becomes full.

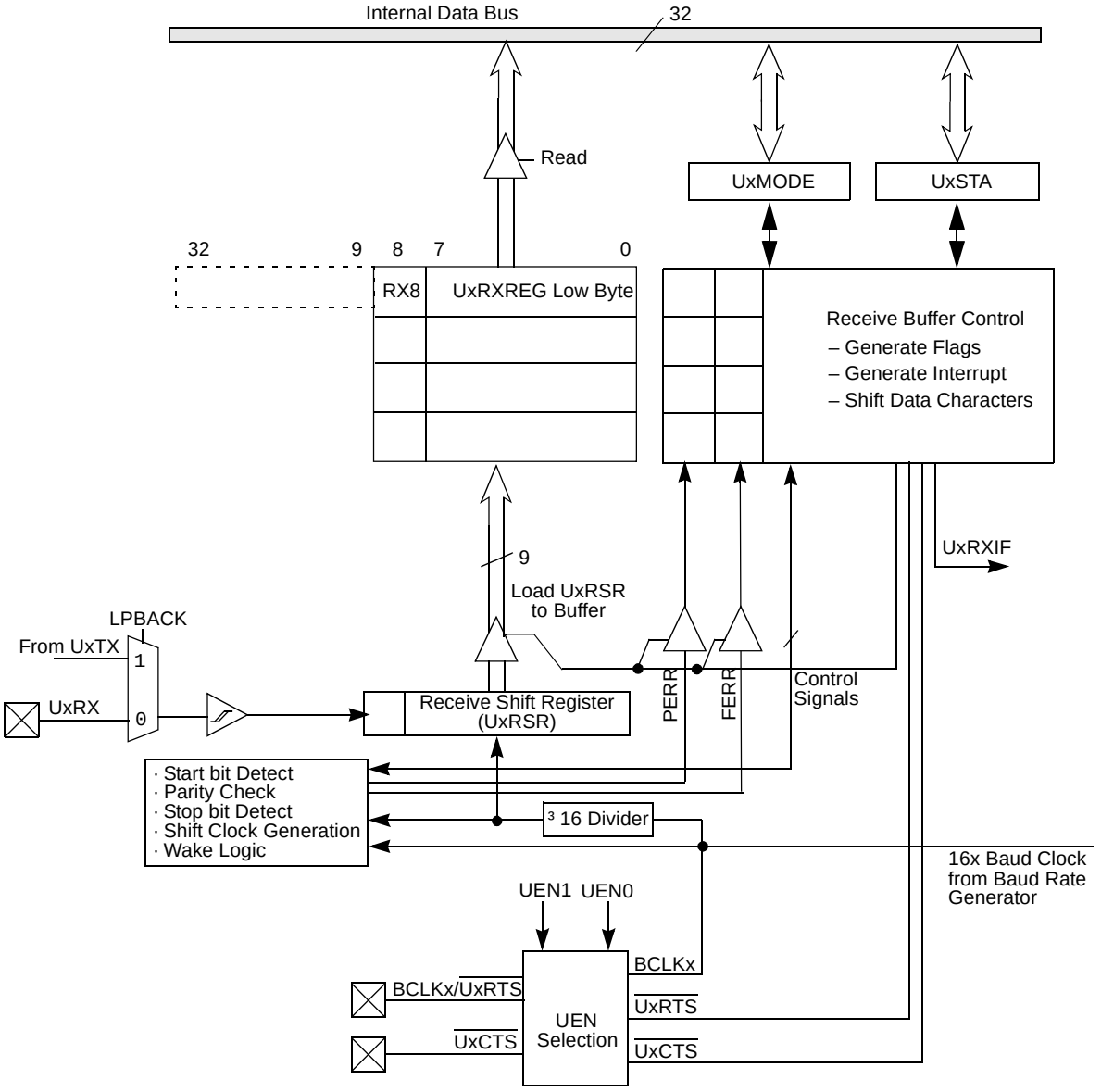
Switching between the three Interrupt modes during operation is possible.

While the RXDA and UxRXIF flag bits indicate the status of the UxRXREG register, the RIDLE bit (UxSTA<4>) shows the status of the UxRSR register. The RIDLE Status bit is a read-only bit that is set when the receiver is IDLE, i.e., the UxRSR register is empty. No interrupt logic is tied to this bit, so the user has to poll this bit to determine whether the UxRSR is IDLE.

The RXDA bit (UxSTA<0>) indicates whether the receive buffer has data or is empty. This bit is set as long as there is at least one character to be read from the receive buffer. RXDA is a read-only bit.

A block diagram of the UART receiver is shown in Figure 21-7.

Figure 21-7: UART Receiver Block Diagram



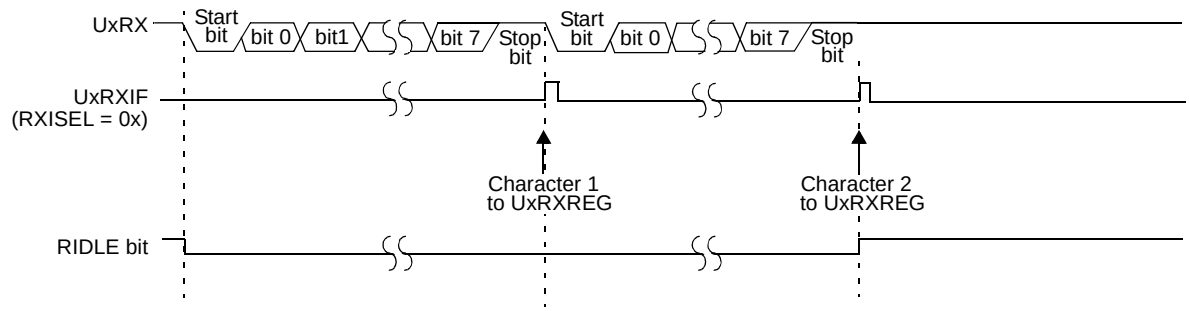
Note: 'x' denotes the UART number.

21.6.0.4 Setup for UART Reception

The following steps are performed to set up a UART reception:

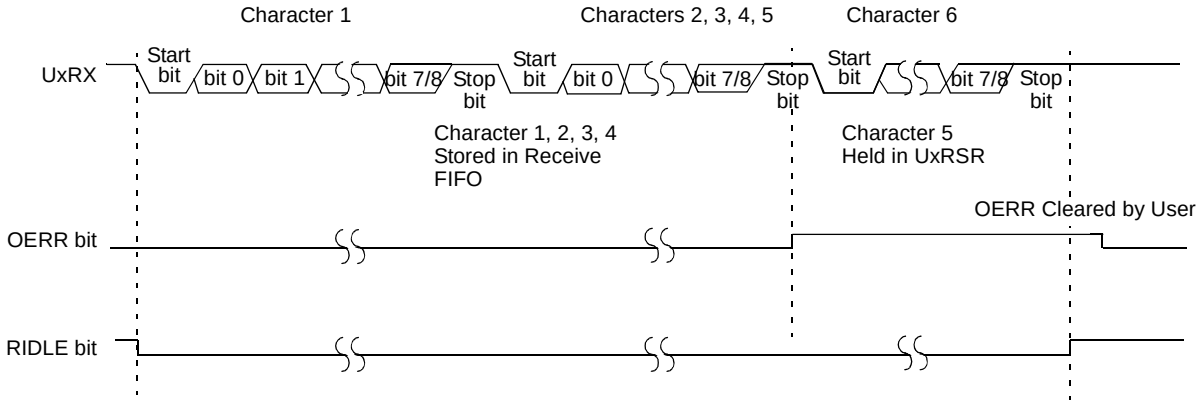
1. Initialize the UxBRG register for the appropriate baud rate (see **Section 21.3 "UART Baud Rate Generator"**).
2. Set the number of data bits, number of Stop bits and parity selection by writing to the PDSEL<1:0> (UxMODE<2:1>) and STSEL (UxMODE<0>) bits.
3. If interrupts are desired, set the UxRXIE bit in the corresponding Interrupt Enable Control (IEC) register. Specify the interrupt priority and subpriority for the interrupt using the UxIP<2:0> and UxIS<1:0> control bits in the corresponding Interrupt Priority Control (IPC) register. Also, select the Receive Interrupt mode by writing to the RXISEL<1:0> (UxSTA<7:6>) bits.
4. Enable the UART receiver by setting the URXEN (UxSTA<12>) bit.
5. Enable the UART module by setting the ON (UxMODE<15>) bit.
6. Receive interrupts are dependent on the RXISEL<1:0> control bit settings. If receive interrupts are not enabled, the user can poll the RXDA bit. The UxRXIF bit should be cleared in the software routine that services the UART receive interrupt.
7. Read data from the receive buffer. If 9-bit transmission has been selected, read a word; otherwise, read a byte. The RXDA Status bit (UxSTA<0>) is set whenever data is available in the buffer.

Figure 21-8: UART Reception



Note: This timing diagram shows 2 characters received on the UxRX input.

Figure 21-9: UART Reception with Receive Overrun



Note: This diagram shows 6 characters received without the user reading the input buffer. The 5th character received is held in the Receive Shift register. An overrun error occurs at the start of the 6th character.

21.7 USING THE UART FOR 9-BIT COMMUNICATION

The UART receiver in 9-bit Data mode is used for communication in a multiprocessor environment. With the ADDEN bit set in 9-bit Data mode, a receiver can ignore the data when the 9th bit of the data is '0'.

21.7.1 Multiprocessor Communications

A typical multi-processor communication protocol differentiates between data bytes and address/control bytes. A common scheme is to use a 9th data bit to identify whether a data byte is address or data information. If the 9th bit is set, the data is processed as address or control information. If the 9th bit is cleared, the received data word is processed as data associated with the previous address/control byte.

The protocol operates in the following sequence:

- The master device transmits a data word with the 9th bit set. The data word contains the address of a slave device and is considered the address word.
- All slave devices in the communication chain receive the address word and check the slave address value.
- The slave device that is specified by the address word receives and processes subsequent data bytes sent by the master device. All other slave devices discard subsequent data bytes until a new address word is received.

21.7.1.1 ADDEN Control Bit

The UART receiver has an Address Detect mode which allows it to ignore data words with the 9th bit cleared. This reduces the interrupt overhead, since data words with the 9th bit cleared are not buffered. This feature is enabled by setting the ADDEN bit (UxSTA<5>).

The UART must be configured for 9-bit data to use the Address Detect mode. The ADDEN bit has no effect when the receiver is configured in 8-bit Data mode.

21.7.1.2 Setup for 9-Bit Transmit Mode

The setup procedure for 9-bit transmission is identical to the 8-bit Transmit modes, except that PDSEL<1:0> (UxMODE<2:1) should be set to '11.' Word writes should be performed to the UxTXREG register (starts transmission). Refer to **Section 21.5.3 "Setup for UART Transmit"** for more information on setting up for transmission.

21.7.1.3 Setup for 9-Bit Reception Using Address Detect Mode

The setup procedure for 9-bit reception is similar to the 8-bit Receive modes, except that PDSEL<1:0> (UxMODE<2:1) should be set to '11' (refer to **Section 21.6.0.4 "Setup for UART Reception"** for more information about setting up for UART reception).

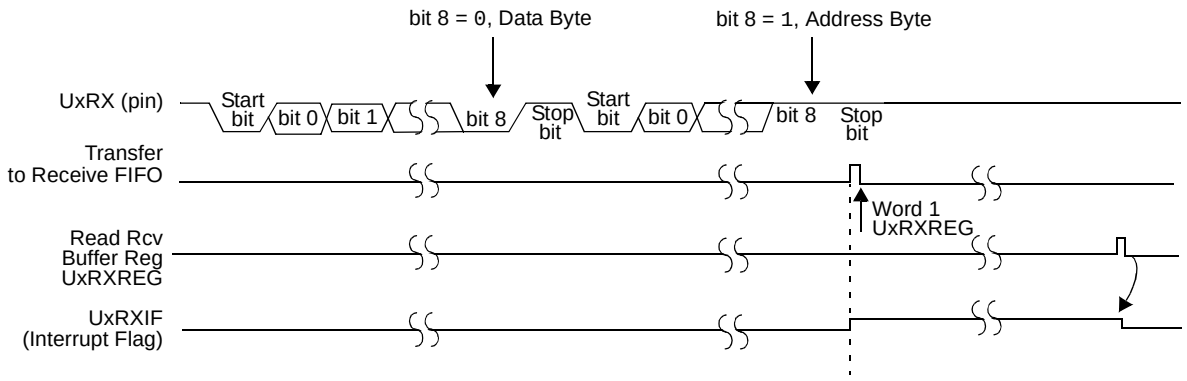
Receive Interrupt mode should be configured by writing to the RXISEL<1:0> (UxSTA<7:6>) bits.

Note:	An interrupt is generated when an Address character is detected and the Address Detect mode is enabled (ADDEN = 1), regardless of how the RXISEL<1:0> control bits are set.

Perform the following steps to use the Address Detect mode:

1. Set PDSEL<1:0> (UxMODE<2:1) to '11' to choose 9-bit mode.
2. Set the ADDEN (UxSTA<5>) bit to enable address detect.
3. Set ADDR (UxSTA<23:16>) to the desired device address character.
4. Set the ADM_EN (UxSTA<24>) bit to enable Address Detect mode.
5. If this device has been addressed, the UxRXREG is discarded, all subsequent characters received that have UxRXREG<8> = 0 are transferred to the UART receive buffer, and interrupts are generated according to RXISEL<1:0>.

Figure 21-10: Reception with Address Detect (ADDEN = 1)



Note: This timing diagram shows a data byte followed by an address byte. The data byte is not read into the UxRXREG (receive buffer) because ADDEN = 1 and bit 8 = 0.

21.8 RECEIVING BREAK CHARACTERS

The wake-up feature is enabled by setting the WAKE bit (UxMODE <7>) = 1. In this mode, the module receives the Start bit, data, and the invalid Stop bit (which sets FERR); but, the receiver waits for a valid Stop bit before looking for the next Start bit. It will not assume that the Break condition on the line is the next Start bit. Break is regarded as a character containing all zeros with the FERR bit set. The Break character is loaded into the buffer. No further reception can occur until a Stop bit is received. The WAKE bit is cleared automatically when the Stop bit is received after the 13-bit Break character. Note that RIDLE goes high when the Stop bit is received.

The receiver counts and expects a certain number of bit times based on the values programmed in the PDSEL<1:0> (UxMODE<2:1>) and STSEL (UxMODE<0>) bits.

If the Break is longer than 13 bit times, the reception is considered complete after the number of bit times specified by the PDSEL and STSEL bits elapses. The RXDA bit is set, FERR is set, zeros are loaded into the receive FIFO and interrupts are generated.

If the wake-up feature is not set, WAKE (UxMODE <7>) = 0, Break reception is not special. The Break is counted as one character loaded into the buffer (all '0' bits) with FERR set.

21.9 INITIALIZATION

An initialization routine for the Transmitter/Receiver in 8-bit mode is shown in Example 21-2. An initialization of the Addressable UART in 9-bit Address Detect mode is shown in Example 21-3. In both examples, the value to load into the UxBRG register is dependent on the desired baud rate and the device frequency.

Note:	UTXEN bit should not be set until the ON bit has been set. Otherwise, UART transmissions is not enabled.

Example 21-2: 8-bit Transmit/Receive (UART1)

```
U1BRG   =   BaudRate;          //Set Baud rate

U1STA   =   0;
U1MODE  =   0x8000;            //Enable Uart for 8-bit data
                               //no parity, 1 STOP bit
U1STASET = 0x1400;             //Enable Transmit and Receive
```

Example 21-3: 8-bit Transmit/Receive (UART1), Address Detect Enabled

```
U1BRG   =   BaudRate;          //Set Baud rate

U1MODE  =   0x8006;            //Enable Uart for 9-bit data
                               //no parity,1 STOP bit
U1STA   =   0x1211420;         //Address detect enabled
                               //Device Address=0x21
                               //Enable Automatic Address Detect Mode
                              //Enable Transmit and Receive
```


21.10 OTHER FEATURES OF THE UART

21.10.1 UART in Loopback Mode

Setting the LPBACK bit enables this special mode in which the UxTX output is internally connected to the UxRX input. When configured for the Loopback mode, the UxRX pin is disconnected from the internal UART receive logic; however, the UxTX pin still functions normally.

Use the following steps to select Loopback mode:

1. Configure UART for the desired mode of operation (see Section 21.5.3).
2. Enable transmission as defined in **Section 21.5 "UART Transmitter"** in this document.
3. Set LPBACK = 1 (UxMODE<6>) to enable Loopback mode.

Table 21-3 shows how the Loopback mode is dependent on the UEN<1:0> bits.

Table 21-3: Loopback Mode Pin Function

UEN<1:0>	Pin Function, LPBACK = 1(1)
00	UxRX input connected to UxTX UxTX pin functions UxRX pin ignored UxCTS/UxRTS unused
01	UxRX input connected to UxTX UxTX pin functions UxRX pin ignored UxRTS pin functions UxCTS unused
10	UxRX input connected to UxTX UxTX pin functions UxRX pin ignored UxRTS pin functions UxCTS input connected to UxRTS UxCTS pin ignored
11	UxRX input connected to UxTX UxTX pin functions UxRX pin ignored BCLKx pin functions UxCTS/UxRTS unused

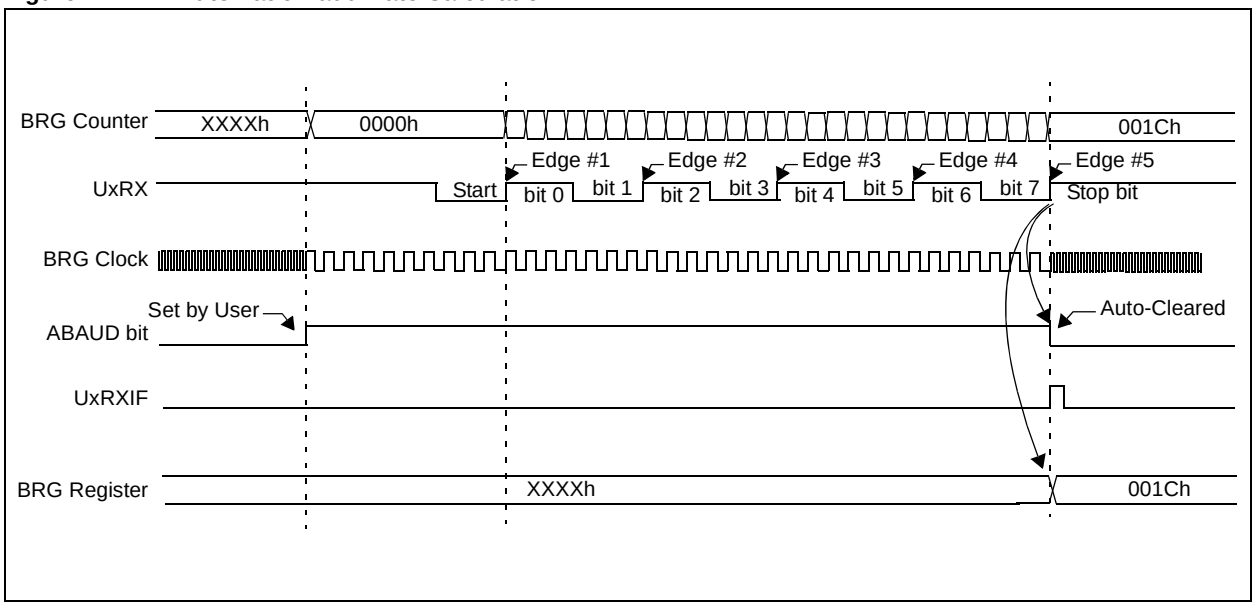
Note 1: LPBACK = 1 should be set only after enabling the other bits associated with the UART module.

21.10.2 Auto-Baud Support

To allow the system to determine the baud rates of the received characters, the ABAUD bit is enabled. The UART begins an automatic baud rate measurement sequence whenever a Start bit is received, and when the Auto-Baud Rate Detect is enabled (ABAUD = 1). The calculation is self-averaging. This feature is active only while the auto-wake-up is disabled (WAKE = 0). In addition, LPBACK must equal '0' for the auto-baud operation. When the ABAUD bit is set, the BRG counter value clears and looks for a Start bit – which, in this case, is defined as a high-to-low transition, followed by a low-to-high transition.

Following the Start bit, the auto-baud expects to receive an ASCII 'U' (55h) to calculate the proper bit rate. The measurement is taken over both the low and the high bit time to minimize any effects caused by asymmetry of the incoming signal. At the end of the Start bit (rising edge), the BRG counter begins counting up using a $F_{PB}/8$ clock. On the 5th UxRX pin rising edge, an accumulated BRG counter value totaling the proper BRG period is transferred to the UxBRG register. The ABAUD bit automatically clears. If the user clears the ABAUD bit prior to sequence completion, unexpected module behavior can result. Refer to Figure 21-1 for the ABD sequence.

Figure 21-11: Automatic Baud Rate Calculation



While the auto-baud sequence is in progress, the UART state machine is held in IDLE. The UxRXIF interrupt is set on the 5th UxRX rising edge, independent of the RXISEL<1:0> settings. The receiver FIFO is not updated.

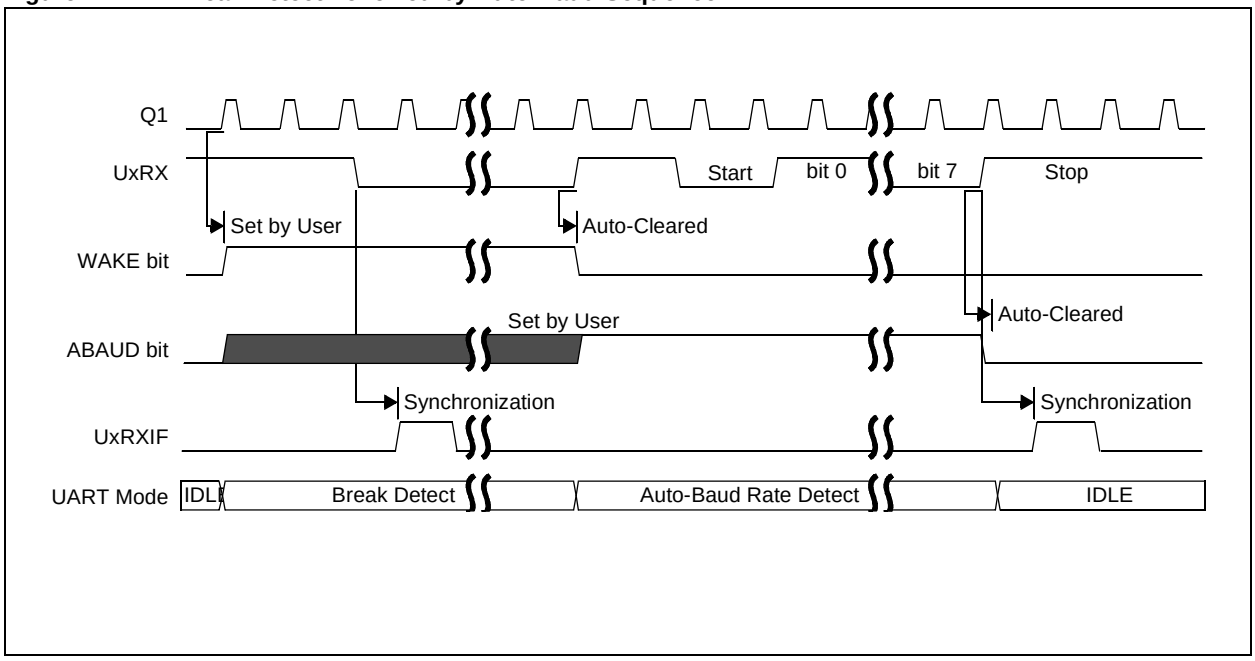
21.10.3 Break Detect Sequence

The user can configure the auto-baud to occur immediately following the Break detect. This is done by setting the ABAUD bit with the WAKE bit set. Figure 21-12 shows a Break detect followed by an auto-baud sequence. The WAKE bit takes priority over the ABAUD bit setting.

Note: If the WAKE bit is set with the ABAUD bit, auto-baud rate detection occurs on the byte following the Break character. The user must ensure that the incoming character baud rate is within the range of the selected UxBRG clock source, considering the baud rate possible with the given clock.

The UART transmitter cannot be used during an auto-baud sequence. Furthermore, the user should ensures that the ABAUD bit is not set while a transmit sequence is already in progress. Otherwise, the UART may exhibit unpredictable behavior.

Figure 21-12: Break Detect Followed by Auto-Baud Sequence



21.11 OPERATION OF $\overline{\text{UxCTS}}$ AND $\overline{\text{UxRTS}}$ CONTROL PINS

$\overline{\text{UxCTS}}$ (Clear to Send) and $\overline{\text{UxRTS}}$ (Request to Send) are the two hardware controlled pins associated with the UART module. These two pins allow the UART to operate in Simplex and Flow Control modes, which are explained in detail in **Section 21.11.2** and **Section 21.11.3**, respectively. They are implemented to control the transmission and reception among the Data Terminal Equipment (DTE).

21.11.1 $\overline{\text{UxCTS}}$ Function

In the UART operation, the $\overline{\text{UxCTS}}$ acts as an input pin that can control the transmission. This pin is controlled by another device (typically a PC). The $\overline{\text{UxCTS}}$ pin is configured using UEN<1:0>. When UEN<1:0> = 10, $\overline{\text{UxCTS}}$ is configured as an input. If $\overline{\text{UxCTS}}$ = 1, then the transmitter will go as far as loading the data in the Transmit Shift register, but will not initiate a transmission. This allows the DTE to control and receive the data accordingly from the controller, per its requirement.

The $\overline{\text{UxCTS}}$ pin is sampled simultaneously with a transmit data change (i.e., at the beginning of the 16 baud clocks). Transmission begins only when the $\overline{\text{UxCTS}}$ is sampled low. The $\overline{\text{UxCTS}}$ is sampled internally with a Q clock, which means that there is a minimum pulse width on $\overline{\text{CTS}}$ of one peripheral clock. However, this cannot be a specification, as the F_{PB} can vary, depending on the clock used.

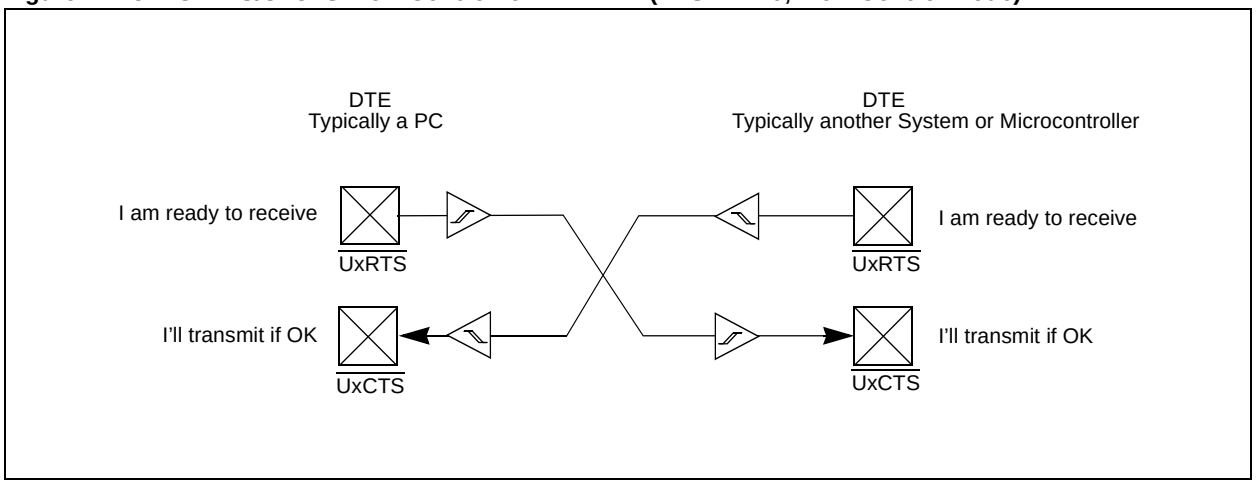
The user can also read the status of the $\overline{\text{UxCTS}}$ by reading the associated port pin.

21.11.2 $\overline{\text{UxRTS}}$ Function in Flow Control Mode

In the Flow Control mode, the $\overline{\text{UxRTS}}$ of one DTE is connected to the $\overline{\text{UxCTS}}$ of the PIC32MX and the $\overline{\text{UxCTS}}$ of the DTE is connected to the $\overline{\text{UxRTS}}$ of the PIC32MX, as shown in Figure 21-13. The $\overline{\text{UxRTS}}$ signal indicates that the device is ready to receive the data. The $\overline{\text{UxRTS}}$ pin is driven as an output whenever UEN<1:0> = 01 or 10. The $\overline{\text{UxRTS}}$ pin is asserted (driven low) whenever the receiver is ready to receive data. When the RTSMD bit = 0 (when the device is in Flow Control mode), the $\overline{\text{UxRTS}}$ pin is driven low whenever the receive buffer is not full or the OERR bit is not set. When the RTSMD bit = 0, the $\overline{\text{UxRTS}}$ pin is driven high whenever the device is not ready to receive (i.e., when the receiver buffer is either full or in the process of shifting).

Since the $\overline{\text{UxRTS}}$ of the DTE is connected to the $\overline{\text{UxCTS}}$ of the PIC32MX, the $\overline{\text{UxRTS}}$ drives the $\overline{\text{UxCTS}}$ low whenever it is ready to receive the data. Transmission of the data begins when the $\overline{\text{UxCTS}}$ goes low, as explained in **Section 21.11.1**.

Figure 21-13: $\overline{\text{UxRTS}}$/$\overline{\text{UxCTS}}$ Flow Control for DTE-DTE (RTSMD = 0, Flow Control Mode)



21.11.3 $\overline{\text{UxRTS}}$ Function in Simplex Mode

In the Simplex mode, the $\overline{\text{UxRTS}}$ of the DCE is connected to the $\overline{\text{UxRTS}}$ of the PIC32MX and the $\overline{\text{UxCTS}}$ of the DCE is connected to the $\overline{\text{UxCTS}}$ of the PIC32MX, respectively, as shown in Figure 21-14. In the Simplex mode, the $\overline{\text{UxRTS}}$ signal indicates that the DTE is ready to transmit. The DCE replies to the $\overline{\text{UxRTS}}$ signal with the valid $\overline{\text{UxCTS}}$ when the DCE is ready to receive the transmission. When the DTE receives a valid $\overline{\text{UxCTS}}$, it begins transmission.

Figure 21-15 shows the Simplex mode is also used in IEEE-485 systems to enable transmitters. When $\overline{\text{UxRTS}}$ indicates that the DTE is ready to transmit, the $\overline{\text{UxRTS}}$ signal enables the driver.

The $\overline{\text{UxRTS}}$ pin is configured as an output and is driven whenever UEN<1:0> = 01 or 10. When RTSMD = 1, the $\overline{\text{UxRTS}}$ is asserted (driven low) whenever the data is available to transmit (TRMT = 0). When RTSMD = 1, $\overline{\text{UxRTS}}$ is deasserted (driven high) when the transmitter is empty (TRMT = 1).

Figure 21-14: $\overline{\text{UxRTS}}$/$\overline{\text{UxCTS}}$ Handshake for DTE-DCE (RTSMD = 1, Simplex Mode)

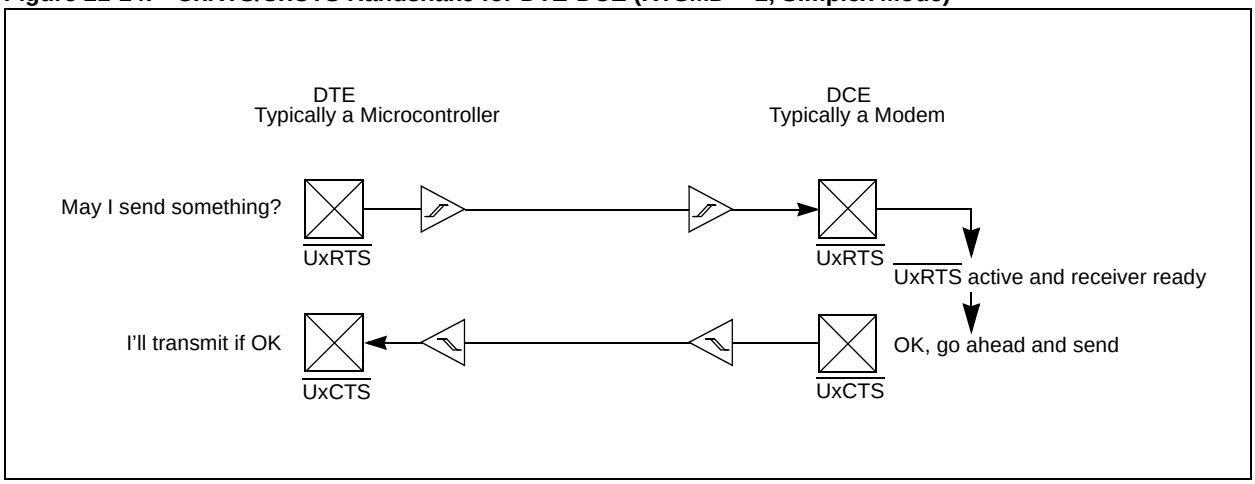
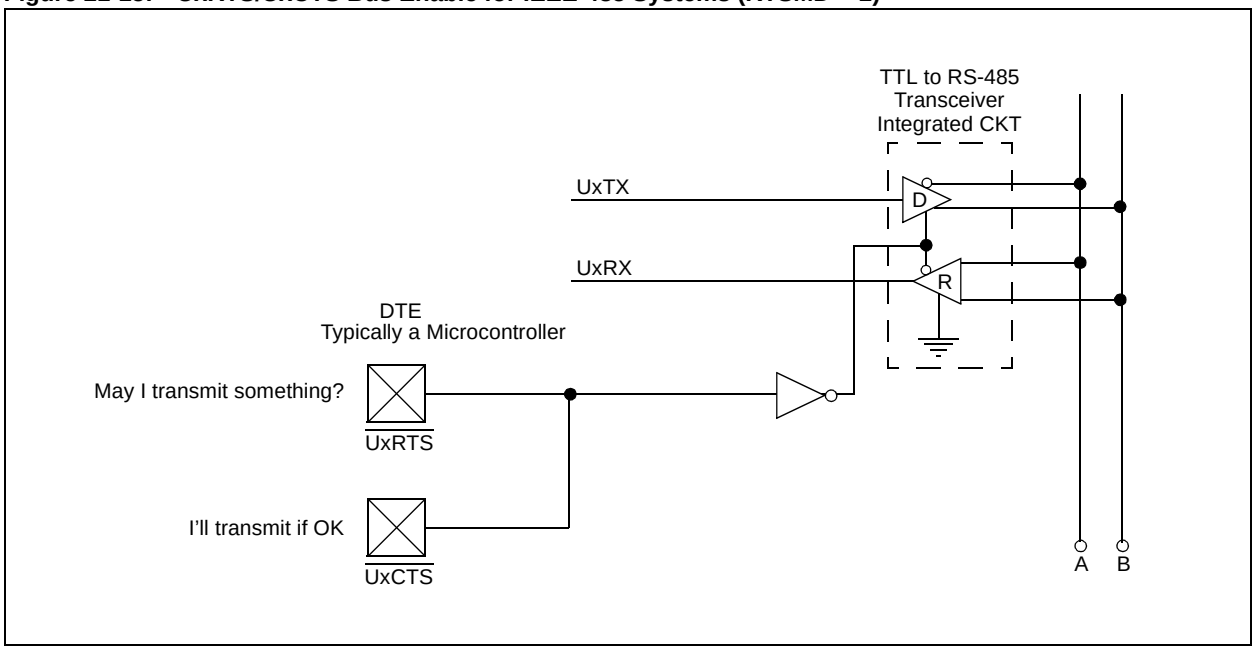


Figure 21-15: $\overline{\text{UxRTS}}$/$\overline{\text{UxCTS}}$ Bus Enable for IEEE-485 Systems (RTSMD = 1)



21.12 INFRARED SUPPORT

The UART module provides the following two types of infrared UART support:

- IrDA clock output to support external IrDA encoder and decoder devices (legacy module support)
- Full implementation of the IrDA encoder and decoder

21.12.1 External IrDA Support – IrDA Clock Output

To support external IrDA encoder and decoder devices, the BCLKx pin can be configured to generate the 16x baud clock. When UEN<1:0> = 11, the BCLKx pin will output the 16x baud clock if the UART module is enabled; it can be used to support the IrDA codec chip.

21.12.2 Built-In IrDA Encoder and Decoder

The UART has full implementation of the IrDA encoder and decoder as part of the UART module. The built-in IrDA encoder and decoder functionality is enabled using the IREN bit (UxMODE<12>). When enabled (IREN = 1), the receive pin UxRX acts as the input from the infrared receiver. The transmit pin UxTX acts as the output to the infrared transmitter.

21.12.2.1 IrDA Encoder Function

The encoder works by taking the serial data from the UART and replacing it in the following manner:

- Transmit bit data of '1' gets encoded as '0' for the entire 16 periods of the 16x baud clock.
- Transmit bit data of '0' gets encoded as '0' for the first 7 periods of the 16x baud clock, as '1' for the next 3 periods and as '0' for the remaining 6 periods.

See Figure 21-16 and Figure 21-18 for details.

21.12.2.2 IrDA Transmit Polarity

The IrDA transmit polarity is selected using the UTXINV bit (UxSTA<13>). This bit only affects the module when the IrDA encoder and decoder are enabled (IREN = 1). The UTXINV bit does not affect the receiver or the module operation for normal transmission and reception. When UTXINV = 0, the IDLE state of the UxTX line is '0' (see Figure 21-16). When UTXINV = 1, the IDLE state of the UxTX line is '1' (see Figure 21-17).

Figure 21-16: IrDA® Encode Scheme

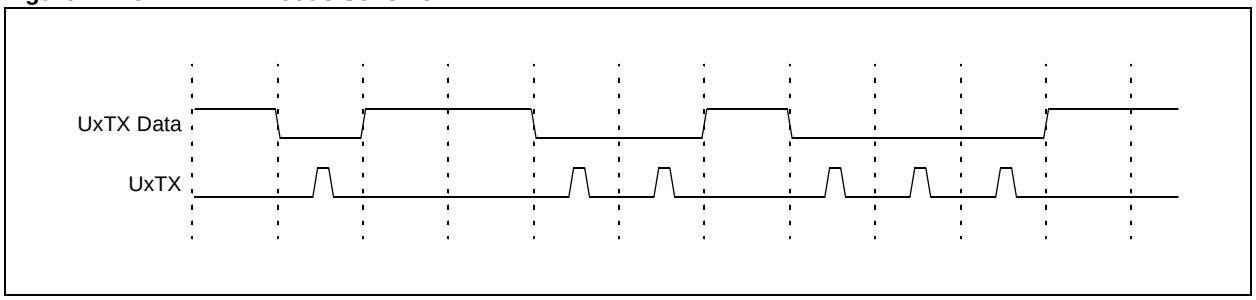


Figure 21-17: IrDA® Encode Scheme for '0' Bit Data

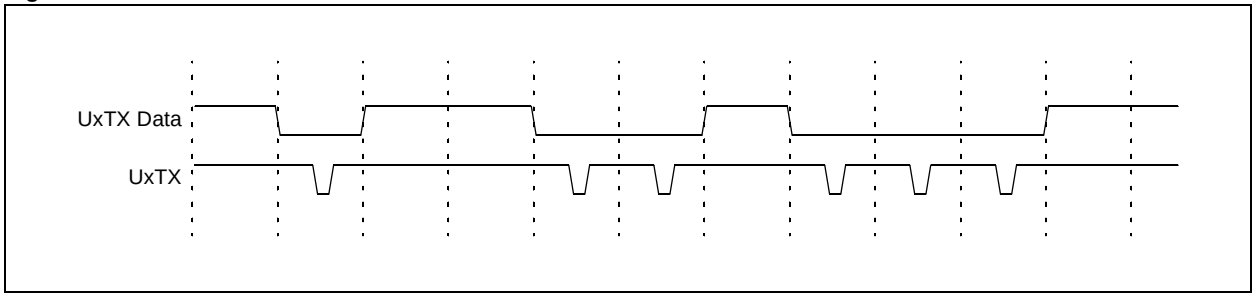
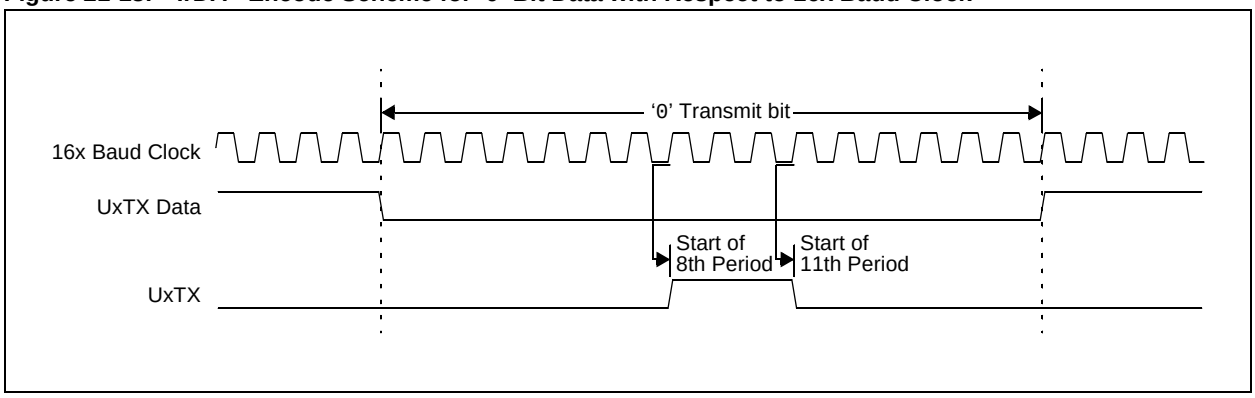


Figure 21-18: IrDA® Encode Scheme for '0' Bit Data with Respect to 16x Baud Clock



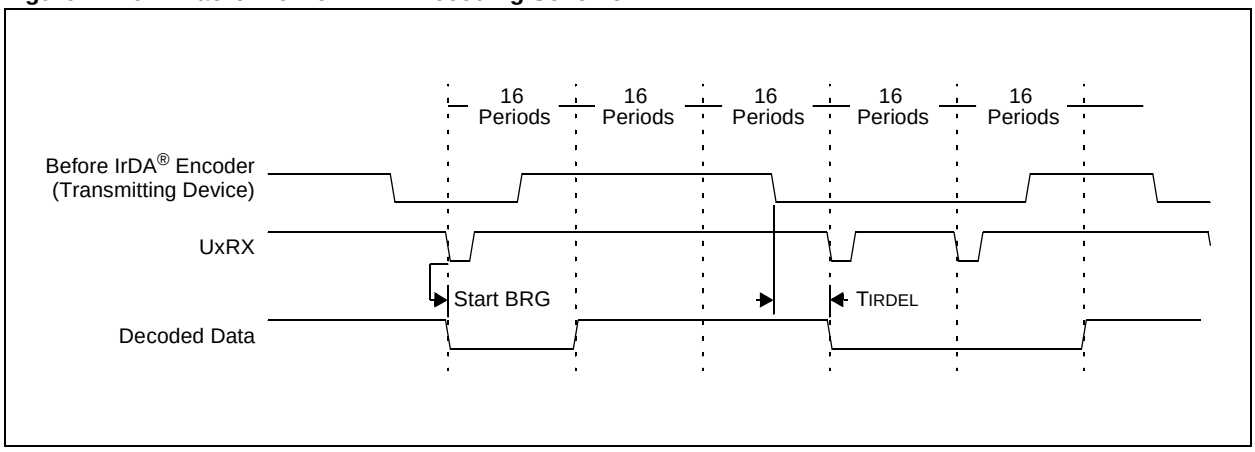
21.12.2.3 IrDA Decoder Function

The decoder works by taking the serial data from the UxRX pin and replacing it with the decoded data stream. The stream is decoded based on falling edge detection of the UxRX input.

Each falling edge of UxRX causes the decoded data to be driven low for 16 periods of the 16x baud clock. If, by the time the 16 periods expire, another falling edge is detected, the decoded data remains low for another 16 periods. If no falling edge is detected, the decoded data is driven high.

Note that the data stream into the device is shifted anywhere from 7 to 8 periods of the 16x baud clock from the actual message source. The one clock uncertainty is due to the clock edge resolution (see Figure 21-19 for details).

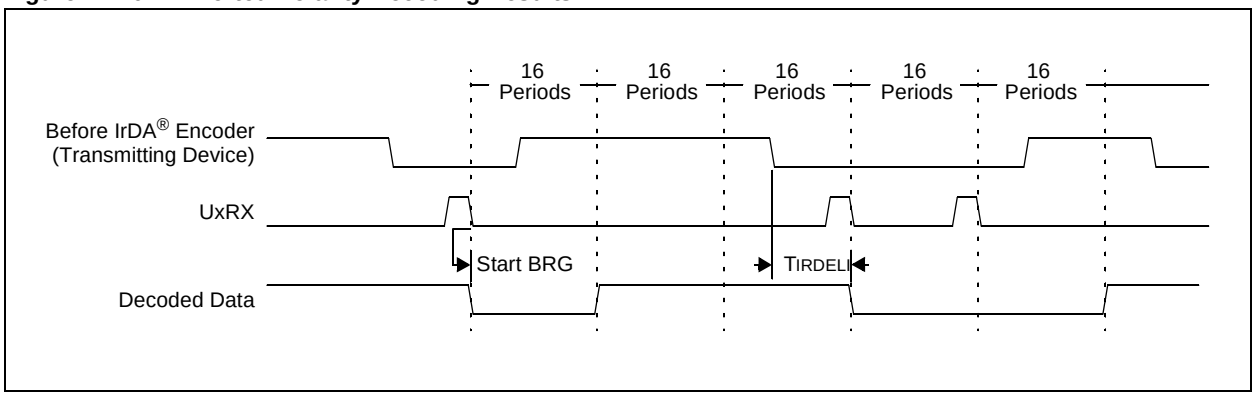
Figure 21-19: Macro View of IrDA® Decoding Scheme



21.12.2.4 IrDA Receive Polarity

The input of the IrDA signal can have an inverted polarity. The same logic is able to decode the signal train, but in this case, the decoded data stream is shifted from 10 to 11 periods of the 16x baud clock from the original message source. Again, the one clock uncertainty is due to the clock edge resolution (see Figure 21-20 for details).

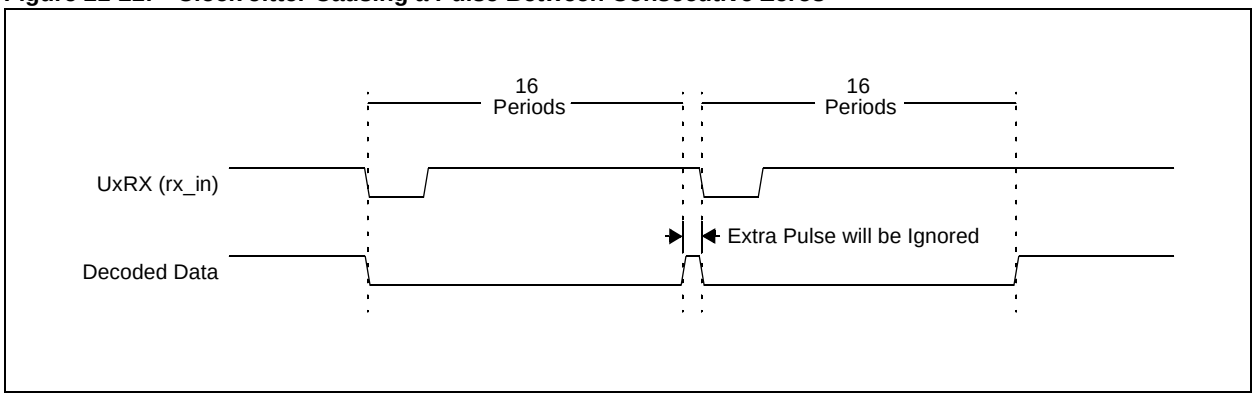
Figure 21-20: Inverted Polarity Decoding Results



21.12.2.5 Clock Jitter

Due to jitter, or slight frequency differences between devices, it is possible for the next falling bit edge to be missed for one of the 16x periods. In that case, a one clock-wide-pulse appears on the decoded data stream. Since the UART performs a majority detect around the bit center, this does not cause erroneous data (see Figure 21-21 for details).

Figure 21-21: Clock Jitter Causing a Pulse Between Consecutive Zeros



21.13 INTERRUPTS

The UART has the ability to generate interrupts reflecting the events that occur during the data communication. The following types of interrupts can be generated:

- Receiver-data-available interrupt, signalled by bits U1RXIF (IFS0<27>) and U2RXIF (IFS1<9>). This event occurs based on the value of RXISEL<1:0> (UxSTA<7:6>) control bits. Refer to **Section 21.6.0.3** for details.
- Transmitter buffer-empty interrupt, signalled by bits U1TXIF (IFS0<28>) and U2TXIF (IFS1<10>). This event occurs based on the value of control bits UTXISEL0<1:0> (UxSTA<15:14>). Refer to **Section 21.5.2** for details.
- UART-error interrupt, signalled by bits U1EIF (IFS0<26>) and U2EIF (IFS1<8>).
  - This event occurs when any of the following error conditions occur:
  - Parity error PERR (UxSTA<3>) is detected
  - Framing Error FERR (UxSTA<2>) is detected
  - Overflow condition for the receive buffer OERR (UxSTA<1>) occurs

All these interrupt flags must be cleared in software.

A UART device is enabled as a source of interrupts via the following respective UART interrupt enable bits:

- U1RXIE (IEC0<27>) and U2RXIE (IEC1<9>)
- U1TXIE (IEC0<28>) and U2TXIE (IEC1<10>)
- U1EIE (IEC0<26>) and U2EIE (IEC1<8>)

The interrupt priority-level bits and interrupt subpriority-level bits must be also be configured:

- U1IP (IPC6<4:2>) and U1IS (IPC6<1:0>)
- U2IP (IPC8<4:2>) and U2IS (IPC8<1:0>)

Refer to **Section 2. “Interrupts”** in this manual for details about priority and subpriority bits.

21.13.1 Interrupt Configuration

Each UART module has the following dedicated interrupt flag bits:

- UxEIF
- UxRXIF
- UxTXIF

Each UART module also has the following corresponding interrupt enable/mask bits:

- UxEIE
- UxRXIE
- UxTXIE

These bits determine the source of an interrupt and enable or disable an individual interrupt source. Note that all the interrupt sources for a specific UART module share just one interrupt vector. Each UART module can have its own priority level, independent of other UART modules.

Note that the UxTXIF, UxRXIF and UxEIF bits will be set without regard to the state of the corresponding enable bits. The IF bits can be polled by software if desired.

The UxEIE, UxTXIE, UxRXIE bits define the behavior of the Vector Interrupt Controller (VIC) when a corresponding UxEIF, UxTXIF or UxRXIF bit is set. When the corresponding IE bit is clear the VIC module does not generate a CPU interrupt for the event. If the IE bit is set, the VIC module will generate an interrupt to the CPU when the corresponding IF bit is set (subject to the priority and subpriority as outlined in the following paragraphs).

It is the responsibility of the user’s software routine that services a particular interrupt to clear the appropriate Interrupt Flag bit before the service routine is complete.

The priority of each UART module can be set independently with the UxIP<2:0> bits. This priority defines the priority group to which the interrupt source is assigned. The priority groups range from a value of 7 (the highest priority), to a value of 0, which does not generate an interrupt. An interrupt being serviced is preempted by an interrupt in a higher priority group.

The subpriority bits allow setting the priority of a interrupt source within a priority group. The values of the subpriority, UxIS<1:0>, range from 3 (the highest priority), to 0 the lowest priority. An interrupt with the same priority group but having a higher subpriority value, does not preempt a lower subpriority interrupt that is in progress.

The priority group and subpriority bits allow more than one interrupt source to share the same priority and subpriority. If simultaneous interrupts occur in this configuration the natural order of the interrupt sources within a priority/subpriority-group pair determine the interrupt generated. The natural priority is based on the vector numbers of the interrupt sources. The lower the vector number, the higher the natural priority of the interrupt. Any interrupts that are overridden by natural order generate their respective interrupts based on priority, subpriority, and natural order, after the interrupt flag for the current interrupt is cleared.

After an enabled interrupt is generated, the CPU jumps to the vector assigned to that interrupt. The vector number for the interrupt is the same as the natural order number. Then the CPU begins executing code at the vector address. The user's code at this vector address should perform any application specific operations, clear the UxEIF, UxTXIF or UxRXIF interrupt flag, and then exit. Refer to **Section 2. "Interrupts"** in this manual for the vector address table details and more information on interrupts.

Table 21-4: UART Interrupt Vectors for Various Offsets with EBASE = 0x8000:0000

Interrupt	Vector/ Natural Order	IRQ Number	Vector Address IntCtl.VS = 0x01	Vector Address IntCtl.VS = 0x02	Vector Address IntCtl.VS = 0x04	Vector Address IntCtl.VS = 0x08	Vector Address IntCtl.VS = 0x10
U1E	24	26	8000 0500	8000 0800	8000 0E00	8000 1A00	8000 3200
U1TX	24	28	8000 0500	8000 0800	8000 0E00	8000 1A00	8000 3200
U1RX	24	27	8000 0500	8000 0800	8000 0E00	8000 1A00	8000 3200
U2E	32	40	8000 0600	8000 0A00	8000 1200	8000 2200	8000 4200
U2TX	32	42	8000 0600	8000 0A00	8000 1200	8000 2200	8000 4200
U2RX	32	41	8000 0600	8000 0A00	8000 1200	8000 2200	8000 4200

21.14 I/O PIN CONTROL

When enabling the UART module ON (UxMODE<15>), the UART module will control the I/O pins as defined by the UEN<1:0> (UxMODE<9:8>) bits, overriding the port TRIS and LATCH register bit settings.

UxTX is forced as an output and UxRX as an input. Additionally, if $\overline{\text{UxCTS}}$ and $\overline{\text{UxRTS}}$ are enabled, the $\overline{\text{UxCTS}}$ is forced as an input and the $\overline{\text{UxRTS}}$/BLCK pin functions as $\overline{\text{UxRTS}}$ output. If BLCK is enabled, then the $\overline{\text{UxRTS}}$/BLCK output drives the 16x baud clock output.

Table 21-5 provides a summary of UART modes and the specific I/O pins required for each mode.

Table 21-5: Required I/O Pin Resources

UxMODE<9:8> Setting		Device Pins			
UEN<1>	UEN<0>	UxTX	UxRX	$\overline{\text{UxCTS}}$	$\overline{\text{UxRTS}}$/BCLK
0	0	Yes	Yes	No	No
0	1	Yes	Yes	No	Yes
1	0	Yes	Yes	Yes	Yes
1	1	Yes	Yes	No	Yes

Note: “No” indicates that the pin is not required and can be used as a general purpose I/O pin.

21.15 UART OPERATION IN POWER-SAVING AND DEBUG MODES

Note: In this manual, a distinction is made between a power mode as it is used in a specific module, and a power mode as it is used by the device, e.g., Sleep mode of the Comparator and SLEEP mode of the CPU. To indicate which type of power mode is intended, uppercase and lowercase letters (Sleep, Idle, Debug) signify a module power mode, and all uppercase letters (SLEEP, IDLE, DEBUG) signify a device power mode.

21.15.1 Operation in SLEEP Mode

When the device enters SLEEP mode, the system clock is disabled. The UART does not function in SLEEP mode. If entry into SLEEP mode occurs while a transmission is in progress, then the transmission is aborted and the UxTX pin is driven to logic '1'. Similarly, if entry into SLEEP mode occurs while a reception is in progress, then the reception is aborted. RTS and BCLK pins are driven to '0'.

The UART can be used optionally to wake the PIC32MX device from SLEEP mode on the detection of a Start bit. If the WAKE bit UxMODE<7> is set before device enters SLEEP mode and the UART receive interrupt is enabled (UxRXIE = 1), then a falling edge on the UxRX pin generates a receive interrupt and device wakes up. The Receive Interrupt Select mode bit (RXISEL) has no effect on this function. The ON bit must be set to generate a wake-up interrupt.

21.15.2 Operation in SLEEP Mode

When the device enters SLEEP mode, the system clock sources remain functional and the CPU stops executing code. The SIDL bit (UxMODE<13>) selects whether the UART module stops operation or continues normal operation when the device enters SLEEP mode.

- If SIDL = 1, the module stops operation in SLEEP mode. The module performs the same procedures when stopped in SLEEP mode (SIDL = 1) as it does for SLEEP mode.
- If SIDL = 0, the module continues operation in SLEEP mode.

21.15.3 Operation in DEBUG Mode

The FRZ bit (UxMODE<14>) determines whether the UART module runs or stops while the CPU is executing DEBUG Exception code (i.e., the application is halted) in DEBUG mode.

Specifically, The FRZ bit affects operation in the following manner:

- If FRZ = 1, the module freezes its operations and make no changes to the state of the UART module when the application is halted in DEBUG mode. The module resumes its operation after the application resumes execution.
- If FRZ = 0, the module continues to run even when application is halted in DEBUG mode.

Note: The FRZ bit is readable and writable only when the CPU is executing in Debug Exception mode. In all other modes, the FRZ bit reads as '0'. If FRZ bit is changed during DEBUG mode, the new value does not take effect until the current Debug Exception mode is exited and re-entered. During the Debug Exception mode, the FRZ bit reads the state of the peripheral when entering DEBUG mode.

21.15.4 Auto-Wake-up on Sync Break Character

The auto-wake-up feature is enabled using the WAKE bit (UxMODE<7>). When WAKE is active, the typical receive sequence on UxRX is disabled. Following the wake-up event, the module generates the UxRXIF interrupt.

Note that the LPBACK bit (UxMODE<6>) must equal '0' for wake-up to operate.

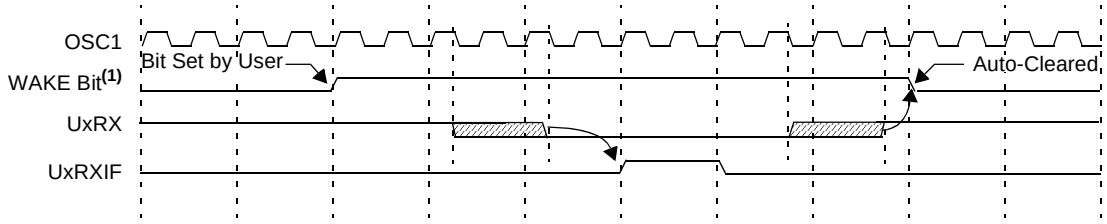
A wake-up event consists of a high-to-low transition on the UxRX line. This coincides with the start of a Sync Break or a Wake-up Signal character for the LIN protocol. When WAKE is active, the UxRX line is monitored independently from the CPU mode. The UxRXIF interrupt is generated synchronously to the Q clocks in Normal User mode; and asynchronously, if the module is disabled due to SLEEP or SLEEP mode. To ensure that no actual data is lost, the WAKE bit should be set just prior to entering the SLEEP mode and while the UART module is in IDLE mode.

The WAKE bit is automatically cleared once a low-to-high transition is observed on the UxRX line following the wake-up event. At this point, the UART module is in IDLE mode and is returned to normal operation. This signals to the user that the Sync Break event is over. If the user clears the WAKE bit prior to sequence completion, unexpected module behavior may result.

The wake-up event causes a receive interrupt by setting the UxRXIF bit. The Receive Interrupt Select mode bits RXISEL<1:0> (UxSTA<7:6>) are ignored for this function. If the UxRXIF interrupt is enabled, it wakes up the device.

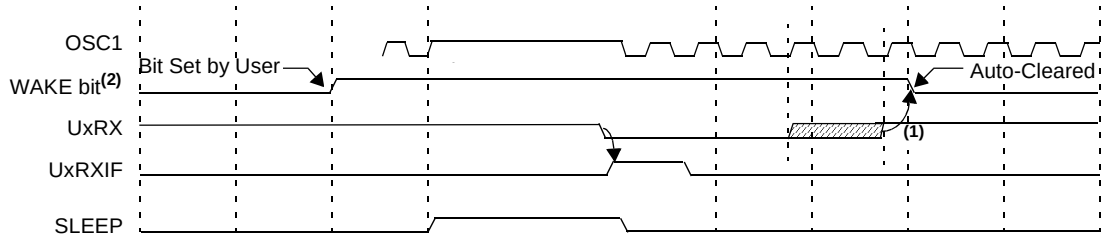
Note:	The Sync Break (or Wake-up Signal) character must be of sufficient length to allow time for the selected oscillator to start and provide proper initialization of the UART. To ensure that the part woke up in time, the user should read the value of the WAKE bit. If it is clear, it is possible that the UART was not ready in time to receive the next character and the module might need to be resynchronized to the bus.

Figure 21-22: Auto-Wake-up Bit (WAKE) Timings During Normal Operation



Note 1: UART state machine is held in IDLE while WAKE bit is active.

Figure 21-23: Auto-Wake-up Bit (WAKE) Timings During SLEEP



Note 1: If the wake-up event requires long oscillator warm-up time, the auto-clear of the WAKE bit can occur while the system clocks are still active. This sequence should not depend on the presence of Q clocks.
2: UART state machine is held in IDLE while WAKE bit is active.

21.16 EFFECTS OF VARIOUS RESETS

21.16.1 Device Reset

All UART registers are forced to their Reset states upon a device Reset.

21.16.2 Power-on Reset

All UART registers are forced to their Reset states upon a Power-on Reset.

21.16.3 Watchdog Reset

All UART registers are unchanged upon a Watchdog Reset.

21.17 DESIGN TIPS

Question 1: ***The data I transmit with the UART is not received correctly. What could cause this?***

Answer: The most common reason for reception errors is that an incorrect value has been calculated for the UART Baud Rate Generator. Ensure the value written to the UxBRG register is correct.

Question 2: ***I am getting framing errors even though the signal on the UART receive pin looks correct. What are the possible causes?***

Answer: Ensure the following control bits have been set up correctly:

- BRGH (UxBRG<15:0) Baud Rate Divisor bits
- PDSEL (UxMODE<1:0>) Parity and Data Selection bits
- STSEL (UxMODE<0>) Stop Selection bit

21.18 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the PIC32MX device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the UART module are:

Title	Application Note #
No related application notes at this time.	N/A

Note: Please visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the PIC32MX device family.

21.19 REVISION HISTORY

Revision A (August 2007)

This is the initial released version of this document.

Revision B (October 2007)

Updated document to remove Confidential status.

Revision C (April 2008)

Revised status to Preliminary; Revised U-0 to r-x; Revised Register 21-1 bit 10; Revised Table 21-1, IEC1; Revised Register 21-16, bit 25; Revised Register 21-18, bit 25; Revised bit names.

Revision D (June 2008)

Revised Section 21.1; Added Footnote number to Registers 21-15-21-20; Change Reserved bits from “Maintain as” to “Write”; Added Note to ON bit (UxMODE Register).

Section 22. Reserved for Future

NOTES:

Section 23. Serial Peripheral Interface

HIGHLIGHTS

This section of the manual contains the following topics:

23.1 INTRODUCTION

The Serial Peripheral Interface (SPI) module is a synchronous serial interface useful for communicating with external peripherals and other microcontroller devices. These peripheral devices may be Serial EEPROMs, shift registers, display drivers, A/D converters, etc. The PIC32MX SPI module is compatible with Motorola® SPI and SIOP interfaces.

Following are some of the key features of this module:

- Master and Slave modes support
- Four different clock formats
- Framed SPI protocol support
- User configurable 8-bit, 16-bit, and 32-bit data width
- Separate SPI shift registers for receive and transmit
- Programmable interrupt event on every 8-bit, 16-bit, and 32-bit data transfer

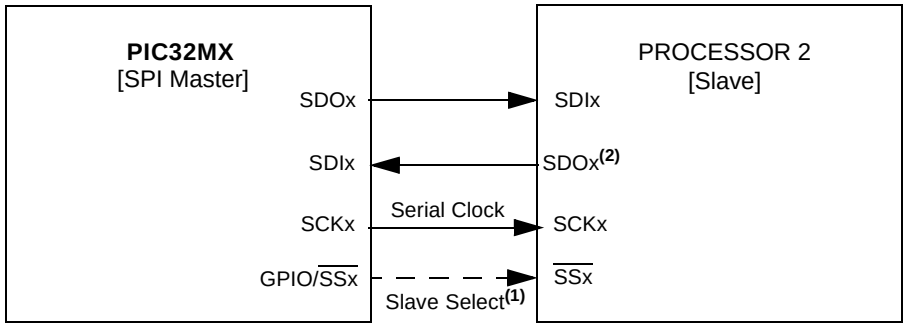
Table 23-1: SPI Features

Available SPI Modes	SPI Master	SPI Slave	Frame Master	Frame Slave	8-Bit, 16-Bit and 32-Bit Modes	Selectable Clock Pulses and Edges	Selectable Frame Sync Pulses and Edges	Slave Select Pulse
Normal Mode	Yes	Yes	—	—	Yes	Yes	—	Yes
Framed Mode	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No

23.1.1 Normal Mode SPI Operation

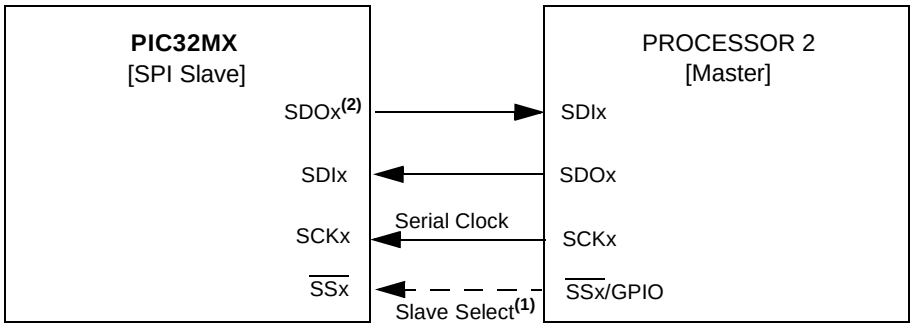
In Normal mode operation, the SPI Master controls the generation of the serial clock. The number of output clock pulses corresponds to the transfer data width: 8, 16, or 32 bits. Figures 23-1 and 23-2 illustrate SPI Master-to-Slave and Slave-to-Master device connections.

Figure 23-1: Typical SPI Master-to-Slave Device Connection Diagram



Note 1: In Normal mode, the usage of the Slave Select pin ($\overline{SSx}$) is optional.
2: Control of the SDO pin can be disabled for Receive-Only modes.

Figure 23-2: Typical SPI Slave-to-Master Device Connection Diagram

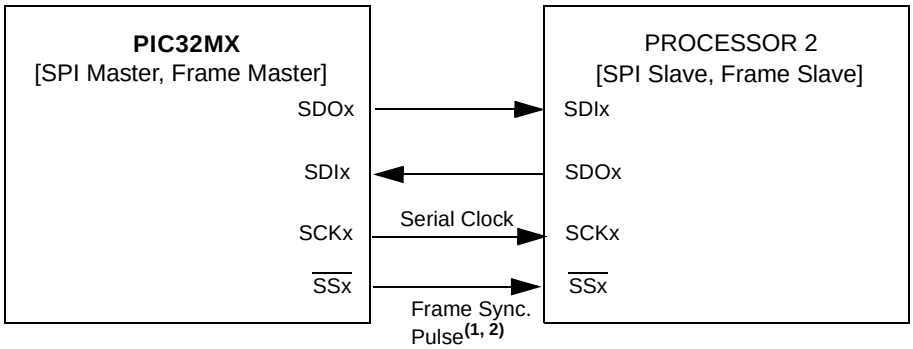


Note 1: In Normal mode, the usage of the Slave Select pin ($\overline{SSx}$) is optional.
2: The control of the SDO pin can be disabled for Receive-Only modes.

23.1.2 Framed Mode SPI Operation

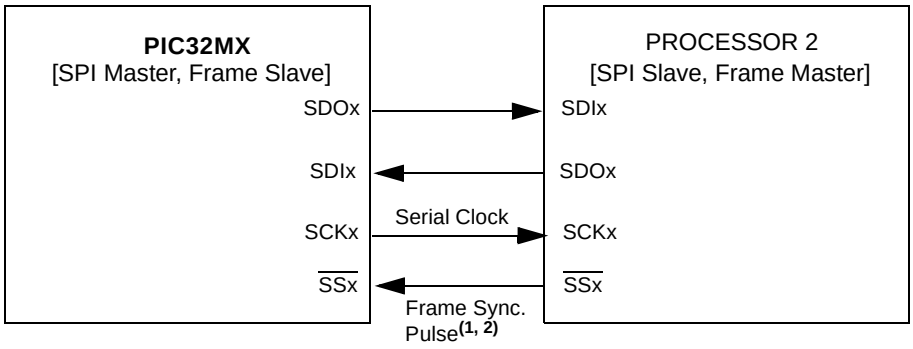
In Framed mode operation, the Frame Master controls the generation of the frame synchronization pulse. The SPI clock is still generated by the SPI Master and is continuously running. Figures 23-3 and 23-4 illustrate SPI Frame Master and Frame Slave device connections.

Figure 23-3: Typical SPI Master, Frame Master Connection Diagram



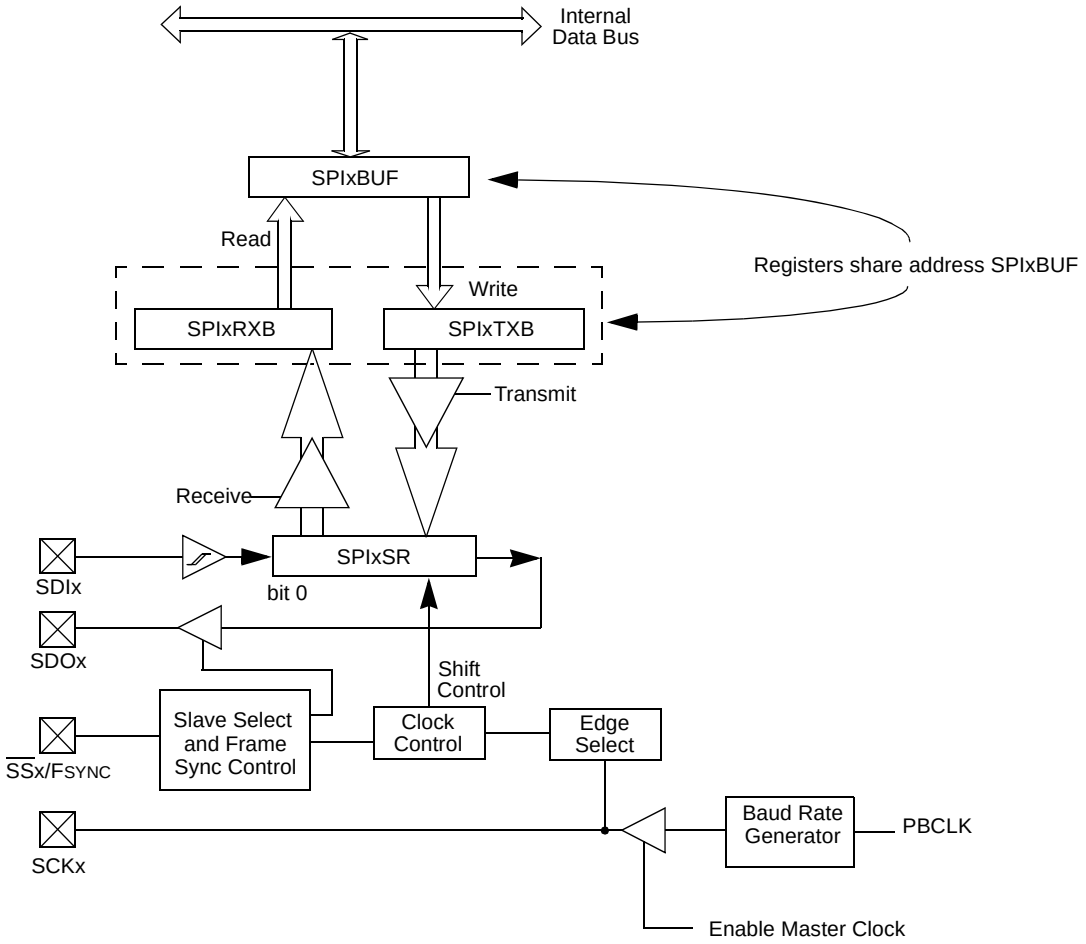
Note 1: In Framed SPI mode, the $\overline{SSx}$ pin is used to transmit/receive the frame synchronization pulse.
2: Framed SPI mode requires the use of all four pins (i.e., using the $\overline{SSx}$ pin is not optional).

Figure 23-4: Typical SPI Master, Frame Slave Connection Diagram



Note 1: In Framed SPI mode, the $\overline{SSx}$ pin is used to transmit/receive the frame synchronization pulse.
2: Framed SPI mode requires the use of all four pins (i.e., using the $\overline{SSx}$ pin is not optional).

Figure 23-5: SPI Module Block Diagram



Note: The SPIxTXB and SPIxRXB registers are accessed via the SPIxBUF register.

23.2 STATUS AND CONTROL REGISTERS

Note: Each PIC32MX device variant may have one or more SPI modules. An 'x' used in the names of pins, control/Status bits, and registers denotes the particular module. Refer to the specific device data sheets for more details.

The SPI module consists of the following Special Function Registers (SFRs):

- SPIxCON: SPI Control Register for the Module 'x'

  SPIxCONCLR, SPIxCONSET, SPIxCONINV: Atomic Bit Manipulation Write-only Registers for SPIxCON

- SPIxSTAT: SPI Status Register for the Module 'x'

  SPIxSTATCLR, SPIxSTATSET, SPIXSTATINV: Atomic Bit Manipulation Write-only Registers for SPIxSTAT

- SPIxBUF: SPI Transmit and Receive Buffer Register for the Module 'x'
- SPIxBRG: SPI Baud Rate Generator Register for the Module 'x'

  SPIxBRGCLR, SPIxBRGSET, SPIxBRGINV: Atomic Bit Manipulation Write-only Registers for SPIxBRG

Each SPI module also has the following associated bits for interrupt control:

- SPIxRXIF, SPIxTXIF, SPIxEIF: Interrupt Flag Status Bits for Receive, Transmit, and Error Events – in IFS0, IFS1 INT Registers
- SPIxRXIE, SPIxTXIE, SPIxEIE: Interrupt Enable Control Bits for Receive, Transmit, and Error Events – in IEC0, IEC1 INT Registers
- SPIxIP<2:0>: Interrupt Priority Control Bits – in IPC6, IPC7 INT Registers
- SPIxIS<1:0>: Interrupt Subpriority Control Bits – in IPC6, IPC7 INT Registers

The following table summarizes all SPI-related registers. Corresponding registers appear after the summary, followed by a detailed description of each register.

Table 23-2: SPI SFR Summary

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
SPIxCON	31:24	FRMEN	FRMSYNC	FRMPOL	—	—	—	—	—
	23:16	—	—	—	—	—	—	SPIFE	—
	15:8	ON	FRZ	SIDL	DISSDO	MODE32	MODE16	SMP	CKE
	7:0	SSEN	CKP	MSTEN	—	—	—	—	—
SPIxCONCLR	31:0	Write clears selected bits in SPIxCON, read yields undefined value							
SPIxCONSET	31:0	Write sets selected bits in SPIxCON, read yields undefined value							
SPIxCONINV	31:0	Write inverts selected bits in SPIxCON, read yields undefined value							
SPIxSTAT	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	SPIBUSY	—	—	—
	7:0	—	SPIROV	—	—	SPITBE	—	—	SPIRBF
SPIxSTATCLR	31:0	Write clears selected bits in SPIxSTAT, read yields undefined value							
SPIxBUF	31:24	DATA<31:24>							
	23:16	DATA<23:16>							
	15:8	DATA<15:8>							
	7:0	DATA<7:0>							
SPIxBRG	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	BRG<8>
	7:0	BRG<7>	BRG<6>	BRG<5>	BRG<4>	BRG<3>	BRG<2>	BRG<1>	BRG<0>
SPIxBRGCLR	31:0	Write clears selected bits in SPIxBRG, read yields undefined value							
SPIxBRGSET	31:0	Write sets selected bits in SPIxBRG, read yields undefined value							
SPIxBRGINV	31:0	Write inverts selected bits in SPIxBRG, read yields undefined value							

Table 23-2: SPI SFR Summary (Continued)

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
IFS0	31:24	I2C1MIF	I2C1SIF	I2C1BIF	U1TXIF	U1RXIF	U1EIF	SPI1RXIF	SPI1TXIF
	23:16	SPI1EIF	OC5IF	IC5IF	T5IF	INT4IF	OC4IF	IC4IF	T4IF
	15:8	INT3IF	OC3IF	IC3IF	T3IF	INT2IF	OC2IF	IC2IF	T2IF
	7:0	INT1IF	OC1IF	IC1IF	T1IF	INT0IF	CS1IF	CS0IF	CTIF
IFS0CLR	31:0	Write clears selected bits in IFS0, read yields undefined value							
IFS0SET	31:0	Write sets selected bits in IFS0, read yields undefined value							
IFS0INV	31:0	Write inverts selected bits in IFS0, read yields undefined value							
IFS1	31:24	—	—	—	—	—	—	USBIF	FCEIF
	23:16	—	—	—	—	DMA3IF	DMA2IF	DMA1IF	DMA0IF
	15:8	RTCCIF	FSCMIF	I2C2MIF	I2C2SIF	I2C2BIF	U2TXIF	U2RXIF	U2EIF
	7:0	SPI2RXIF	SPI2TXIF	SPI2EIF	CMP2IF	CMP1IF	PMPIF	AD1IF	CNIF
IFS1CLR	31:0	Write clears selected bits in IFS1, read yields undefined value							
IFS1SET	31:0	Write sets selected bits in IFS1, read yields undefined value							
IFS1INV	31:0	Write inverts selected bits in IFS1, read yields undefined value							
IEC0	31:24	I2C1MIE	I2C1SIE	I2C1BIE	U1TXIE	U1RXIE	U1EIE	SPI1RXIE	SPI1TXIE
	23:16	SPI1EIE	OC5IE	IC5IE	T5IE	INT4IE	OC4IE	IC4IE	T4IE
	15:8	INT3IE	OC3IE	IC3IE	T3IE	INT2IE	OC2IE	IC2IE	T2IE
	7:0	INT1IE	OC1IE	IC1IE	T1IE	INT0IE	CS1IE	CS0IE	CTIE
IEC0CLR	31:0	Write clears selected bits in IEC0, read yields undefined value							
IEC0SET	31:0	Write sets selected bits in IEC0, read yields undefined value							
IEC0INV	31:0	Write inverts selected bits in IEC0, read yields undefined value							
IEC1	31:24	—	—	—	—	—	—	USBIE	FCEIE
	23:16	—	—	—	—	DMA3IE	DMA2IE	DMA1IE	DMA0IE
	15:8	RTCCIE	FSCMIE	I2C2MIE	I2C2SIE	I2C2BIE	U2TXIE	U2RXIE	U2EIE
	7:0	SPI2RXIE	SPI2TXIE	SPI2EIE	CMP2IE	CMP1IE	PMPIE	AD1IE	CNIE
IEC1CLR	31:0	Write clears selected bits in IEC1, read yields undefined value							
IEC1SET	31:0	Write sets selected bits in IEC1, read yields undefined value							
IEC1INV	31:0	Write inverts selected bits in IEC1, read yields undefined value							
IPC5	31:24	—	—	—	SPI1IP<2:0>			SPI1IS<1:0>	
	23:16	—	—	—	OC5IP<2:0>			OC5IS<1:0>	
	15:8	—	—	—	IC5IP<2:0>			IC5IS<1:0>	
	7:0	—	—	—	T5IP<2:0>			T5IS<1:0>	
IPC5CLR	31:0	Write clears selected bits in IPC5, read yields undefined value							
IPC5SET	31:0	Write sets selected bits in IPC5, read yields undefined value							
IPC5INV	31:0	Write inverts selected bits in IPC5, read yields undefined value							
IPC7	31:24	—	—	—	SPI2IP<2:0>			SPI2IS<1:0>	
	23:16	—	—	—	CMP2IP<2:0>			CMP2IS<1:0>	
	15:8	—	—	—	CMP1IP<2:0>			CMP1IS<1:0>	
	7:0	—	—	—	PMPIP<2:0>			PMPIS<1:0>	
IPC7CLR	31:0	Write clears selected bits in IPC7, read yields undefined value							
IPC7SET	31:0	Write sets selected bits in IPC7, read yields undefined value							
IPC7INV	31:0	Write inverts selected bits in IPC7, read yields undefined value							

Register 23-1: SPIxCON: SPI Control Register

R/W-0	R/W-0	R/W-0	r-x	r-x	r-x	r-x	r-x
FRMEN	FRMSYNC	FRMPOL	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	r-x
—	—	—	—	—	—	SPIFE	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ON	FRZ	SIDL	DISSDO	MODE32	MODE16	SMP	CKE
bit 15							bit 8

R/W-0	R/W-0	R/W-0	r-x	r-x	r-x	r-x	r-x
SSEN	CKP	MSTEN	—	—	—	—	—
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31 **FRMEN:** Framed SPI Support bit
1 = Framed SPI support is enabled ($\overline{SSx}$ pin used as FSYNC input/output)
0 = Framed SPI support is disabled

bit 30 **FRMSYNC:** Frame Sync Pulse Direction Control on $\overline{SSx}$ pin bit (Framed SPI mode only)
1 = Frame sync pulse input (Slave mode)
0 = Frame sync pulse output (Master mode)

bit 29 **FRMPOL:** Frame Sync Polarity bit (Framed SPI mode only)
1 = Frame pulse is active-high
0 = Frame pulse is active-low

bit 28-18 **Reserved:** Write ‘0’; ignore read

bit 17 **SPIFE:** Frame Sync Pulse Edge Select bit (Framed SPI mode only)
1 = Frame synchronization pulse coincides with the first bit clock
0 = Frame synchronization pulse precedes the first bit clock

bit 16 **Reserved:** Write ‘0’; ignore read

bit 15 **ON:** SPI Peripheral On bit
1 = SPI Peripheral is enabled
0 = SPI Peripheral is disabled

Note: When using 1:1 PBCLK divisor, the user’s software should not read/write the peripheral’s SFRs in the SYSCLK cycle immediately following the instruction that clears the module’s ON bit.

bit 14 **FRZ:** Freeze in DEBUG Exception Mode bit
1 = Freeze operation when CPU enters DEBUG Exception mode
0 = Continue operation when CPU enters DEBUG Exception mode
Note: FRZ is writable in DEBUG Exception mode only, it is forced to ‘0’ in Normal mode.

bit 13 **SIDL:** Stop in IDLE Mode bit
1 = Discontinue operation when CPU enters in IDLE mode
0 = Continue operation in IDLE mode

Register 23-1: SPIxCON: SPI Control Register (Continued)

bit 12	**DISSDO:** Disable SDOx pin bit 1 = SDOx pin is not used by the module. Pin is controlled by associated PORT register 0 = SDOx pin is controlled by the module
bit 11-10	**MODE<32,16>:** 32/16-Bit Communication Select bits 1x = 32-bit data width 01 = 16-bit data width 00 = 8-bit data width
bit 9	**SMP:** SPI Data Input Sample Phase bit Master mode (MSTEN = 1): 1 = Input data sampled at end of data output time 0 = Input data sampled at middle of data output time Slave mode (MSTEN = 0): SMP value is ignored when SPI is used in Slave mode. The module always uses SMP = 0.
bit 8	**CKE:** SPI Clock Edge Select bit 1 = Serial output data changes on transition from active clock state to Idle clock state (see CKP bit) 0 = Serial output data changes on transition from Idle clock state to active clock state (see CKP bit) **Note:** The CKE bit is not used in the Framed SPI mode. The user should program this bit to '0' for the Framed SPI mode (FRMEN = 1).
bit 7	**SSEN:** Slave Select Enable (Slave mode) bit 1 = $\overline{SSx}$ pin used for Slave mode 0 = $\overline{SSx}$ pin not used for Slave mode, pin controlled by port function.
bit 6	**CKP:** Clock Polarity Select bit 1 = Idle state for clock is a high level; active state is a low level 0 = Idle state for clock is a low level; active state is a high level
bit 5	**MSTEN:** Master Mode Enable bit 1 = Master mode 0 = Slave mode
bit 4-0	**Reserved:** Write '0'; ignore read

Register 23-2: SPIxCONCLR: SPIxCON Clear Register

Write clears selected bits in SPIxCON, read yields undefined value
bit 31 bit 0

bit 31-0 **Clears selected bits in SPIxCON**

A write of '1' in one or more bit positions clears the corresponding bit(s) in SPIxCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `SPIxCONCLR = 0x00008020` will clear bits 15 and 5 in SPIxCON register.

Register 23-3: SPIxCONSET: SPIxCON Set Register

Write sets selected bits in SPIxCON, read yields undefined value
bit 31 bit 0

bit 31-0 **Sets selected bits in SPIxCON**

A write of '1' in one or more bit positions sets the corresponding bit(s) in SPIxCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `SPIxCONSET = 0x00008020` will set bits 15 and 5 in SPIxCON register.

Register 23-4: SPIxCONINV: SPIxCON Invert Register

Write inverts selected bits in SPIxCON, read yields undefined value
bit 31 bit 0

bit 31-0 **Inverts selected bits in SPIxCON**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in SPIxCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `SPIxCONINV = 0x00008020` will invert bits 15 and 5 in SPIxCON register.

Register 23-5: SPIxSTAT: SPI Status Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	R-0	r-x	r-x	r-x
—	—	—	—	SPIBUSY	—	—	—
bit 15							bit 8

r-x	R/W-0	r-x	r-x	R-1	r-x	r-x	R-0
—	SPIROV	—	—	SPITBE	—	—	SPIRBF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-12 **Reserved:** Write ‘0’; ignore read

bit 11 **SPIBUSY:** SPI Activity Status bit
1 = SPI peripheral is currently busy with some transactions
0 = SPI peripheral is currently idle

bit 10-7 **Reserved:** Write ‘0’; ignore read

bit 6 **SPIROV:** Receive Overflow Flag bit
1 = A new data is completely received and discarded. The user software has not read the previous data in the SPIxBUF register.
0 = No overflow has occurred
This bit is set in hardware; can only be cleared (= 0) in software.

bit 5-4 **Reserved:** Write ‘0’; ignore read

bit 3 **SPITBE:** SPI Transmit Buffer Empty Status bit
1 = Transmit buffer, SPIxTXB is empty
0 = Transmit buffer, SPIxTXB is not empty
Automatically set in hardware when SPI transfers data from SPIxTXB to SPIxSR.
Automatically cleared in hardware when SPIxBUF is written to, loading SPIxTXB.

bit 2 **Reserved:** Write ‘0’; ignore read

bit 1 **Reserved:** Write ‘0’; ignore read

bit 0 **SPIRBF:** SPI Receive Buffer Full Status bit
1 = Receive buffer, SPIxRXB is full
0 = Receive buffer, SPIXRXB is not full
Automatically set in hardware when SPI transfers data from SPIxSR to SPIxRXB.
Automatically cleared in hardware when SPIxBUF is read from, reading SPIxRXB.

Register 23-6: SPIxSTATCLR: SPIxSTAT Clear Register

Write clears selected bits in SPIxSTAT, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in SPIxSTAT**

A write of '1' in one or more bit positions clears the corresponding bit(s) in SPIxSTAT register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `SPIxSTATCLR = 0x00000040` will clear bit 6 in SPIxSTAT register.

Register 23-7: SPIxBUF: SPI Buffer Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
DATA<31:24>							
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
DATA<23:16>							
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
DATA<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
DATA<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **DATA**<31:0>**:** SPI Transmit/Receive Buffer register
Serves as a memory-mapped value of Transmit (SPIxTXB) and Receive (SPIxSR) registers.

When 32-Bit Data mode is enabled (MODE[32,16] (SPIxCON<11:10>) = 1x):
All 32-bits (SPIxBUF<31:0>) of this register are used to form a 32-bit character.

When 16-Bit Data mode is enabled (MODE[32,16] (SPIxCON<11:10>) = 01):
Only lower 16-bits (SPIxBUF<15:0>) of this register are used to form the 16-bit character.

When 8-Bit Data mode is enabled (MODE[32,16] (SPIxCON<11:10>) = 00):
Only lower 8-bits (SPIxBUF<7:0>) of this register are used to form the 8-bit character.

Register 23-8: SPIxBRG: SPI Baud Rate Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	R/W-0
—	—	—	—	—	—	—	BRG<8>
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
BRG<7>	BRG<6>	BRG<5>	BRG<4>	BRG<3>	BRG<2>	BRG<1>	BRG<0>
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-9 **Reserved:** Write ‘0’; ignore read

bit 8-0 **BRG<8:0>:** Baud Rate Divisor bits

Register 23-9: SPIxBRGCLR: SPIxBRG Clear Register

Write clears selected bits in SPIxBRG, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in SPIxBRG**

A write of '1' in one or more bit positions clears the corresponding bit(s) in SPIxBRG register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `SPIxBRGCLR = 0x000001FF` will clear bits 8 through 0 in SPIxBRG register.

Register 23-10: SPIxBRGSET: SPIxBRG Set Register

Write sets selected bits in SPIxBRG, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in SPIxBRG**

A write of '1' in one or more bit positions sets the corresponding bit(s) in SPIxBRG register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `SPIxBRGSET = 0x000001FF` will set bits 8 through 0 in SPIxBRG register.

Register 23-11: SPIxBRGINV: SPIxBRG Invert Register

Write inverts selected bits in SPIxBRG, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in SPIxBRG**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in SPIxBRG register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `SPIxBRGINV = 0x000001FF` will toggle bits 8 through 0 in SPIxBRG register.

Register 23-12: IFS0: Interrupt Flag Status Register 0(1)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
I2C1MIF	I2C1SIF	I2C1BIF	U1TXIF	U1RXIF	U1EIF	SPI1RXIF	SPI1TXIF
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SPI1EIF	OC5IF	IC5IF	T5IF	INT4IF	OC4IF	IC4IF	T4IF
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INT3IF	OC3IF	IC3IF	T3IF	INT2IF	OC2IF	IC2IF	T2IF
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-0
INT1IF	OC1IF	IC1IF	T1IF	INT0IF	CS1IF	CS0IF	CTIF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-26 Interrupt Flag bits for other peripheral devices

bit 25 **SPI1RXIF**: SPI1 Receive Buffer Full Interrupt Flag bit
1 = Receive buffer full interrupt pending
0 = No receive interrupt pending
Set by the hardware when a character is assembled in the SPI1 receive buffer.
Cleared by the software, usually in the ISR.

bit 24 **SPI1TXIF**: SPI1 Transmit Buffer Empty Interrupt Flag bit
1 = Transmit buffer empty interrupt pending
0 = No transmit interrupt pending
Set by the hardware when a character can be written into the SPI1 transmit buffer.
Cleared by the software, usually in the ISR.

bit 23 **SPI1EIF**: SPI1 Error Interrupt Flag bit
1 = SPI1 receive overflow interrupt pending
0 = No receive overflow interrupt pending
Set by the hardware when a character is assembled in the SPI1 receive buffer and the previous character hasn’t been read from the SPI buffer.
Cleared by the software as part of the error processing.

bit 22-0 Interrupt Flag bits for other peripheral devices

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the SPI.

Register 23-13: IFS1: Interrupt Flag Status Register 1(1)

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	USBIF	FCEIF
bit 31							bit 24

r-0	r-0	r-0	r-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	DMA3IF	DMA2IF	DMA1IF	DMA0IF
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RTCCIF	FSCMIF	I2C2MIF	I2C2SIF	I2C2BIF	U2TXIF	U2RXIF	U2EIF
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-0
SPI2RXIF	SPI2TXIF	SPI2EIF	CMP2IF	CMP1IF	PMPIF	AD1IF	CNIF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-26 **Unimplemented:** Read as ‘0’

bit 25-24 Interrupt flags for other peripheral devices

bit 23-20 **Reserved:** Write ‘0’; ignore read

bit 19-8 Interrupt flags for other peripheral devices

bit 7 **SPI2RXIF**: SPI2 Receive buffer full interrupt flag

1 = Receive buffer full interrupt pending
0 = No receive interrupt pending

Set by the hardware when a character is assembled in the SPI2 receive buffer.

Cleared by the software, usually in the ISR.

bit 6 **SPI2TXIF**: SPI2 Transmit buffer empty interrupt flag

1 = Transmit buffer empty interrupt pending
0 = No transmit interrupt pending

Set by the hardware when a character can be written into the SPI2 transmit buffer.

Cleared by the software, usually in the ISR.

bit 5 **SPI2EIF**: SPI2 Error interrupt flag

1 = SPI2 receive overflow interrupt pending
0 = No receive overflow interrupt pending

Set by the hardware when a character is assembled in the SPI2 receive buffer and the previous character hasn’t been read from the SPI buffer.

Cleared by the software as part of the error processing.

bit 4-0 Interrupt flags for other peripheral devices

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the SPI.

Register 23-14: IEC0: Interrupt Enable Control Register 0(1)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
I2C1MIE	I2C1SIE	I2C1BIE	U1TXIE	U1RXIE	U1EIE	SPI1RXIE	SPI1TXIE
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SPI1EIE	OC5IE	IC5IE	T5IE	INT4IE	OC4IE	IC4IE	T4IE
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INT3IE	OC3IE	IC3IE	T3IE	INT2IE	OC2IE	IC2IE	T2IE
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INT1IE	OC1IE	IC1IE	T1IE	INT0IE	CS1IE	CS0IE	CTIE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-26 Interrupt Enable Flag bits for other peripheral devices

bit 25 **SPI1RXIE:** SPI1 Receive Buffer Full Interrupt Enable bit

1 = Receive buffer full interrupt enabled
0 = Receive buffer full interrupt disabled

Set/cleared by the software to enable/disable SPI interrupts when a new character is assembled in the SPI1 receive buffer.

bit 24 **SPI1TXIE:** SPI1 Transmit Buffer Empty Interrupt Enable bit

1 = Transmit buffer empty interrupt enabled
0 = Transmit buffer empty interrupt disabled

Set/cleared by the software to enable/disable SPI interrupts when a new character can be written into the SPI1 transmit buffer.

bit 23 **SPI1EIE:** SPI1 Error Interrupt Enable bit

1 = SPI1 receive overflow interrupt enabled
0 = SPI1 receive overflow interrupt disabled

Set by the software to enable/disable SPI interrupts when a character is assembled in the SPI1 receive buffer and the previous character hasn’t been read from the SPI buffer.

bit 22-0 Interrupt Enable Flag bits for other peripheral devices

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the SPI.

Register 23-15: IEC1: Interrupt Enable Control Register 1(1)

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	USBIE	FCEIE
bit 31							bit 24

r-0	r-0	r-0	r-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	DMA3IE	DMA2IE	DMA1IE	DMA0IE
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RTCCIE	FSCMIE	I2C2MIE	I2C2SIE	I2C2BIE	U2TXIE	U2RXIE	U2EIE
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-0
SPI2RXIE	SPI2TXIE	SPI2EIE	CMP2IE	CMP1IE	PMPIE	AD1IE	CNIE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-26 **Unimplemented:** Read as ‘0’

bit 25-24 Interrupt flags for other peripheral devices

bit 23-20 **Reserved:** Write ‘0’; ignore read

bit 19-8 Interrupt flags for other peripheral devices

bit 7 **SPI2RXIE:** SPI2 Receive Buffer Full Interrupt Enable bit

1 = Receive buffer full interrupt enabled
0 = Receive buffer full interrupt disabled

Set/cleared by the software to enable/disable the interrupt when a character is assembled in the SPI2 receive buffer.

bit 6 **SPI2TXIE:** SPI2 Transmit Buffer Empty Interrupt Enable bit

1 = Transmit buffer empty interrupt enabled
0 = Transmit buffer empty interrupt disabled

Set/cleared by the software to enable/disable interrupts when a character can be written into the SPI2 transmit buffer.

bit 5 **SPI2EIE:** SPI2 Error Interrupt Enable bit

1 = SPI2 receive overflow interrupt enabled
0 = SPI2 receive overflow interrupt disabled
Set/cleared by the software to enable/disable overflow interrupts.

bit 4-0 Interrupt enable bits for other peripheral devices

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the SPI.

Register 23-16: IPC5: Interrupt Priority Control Register 5(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	SPI1IP<2:0>			SPI1IS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	OC5IP<2:0>			OC5IS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	IC5IP<2:0>			IC5IS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	T5IP<2:0>			T5IS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-29 **Reserved:** Write ‘0’; ignore read

bit 28-26 **SPI1IP<2:0>:** SPI1 Interrupt Vector Priority bits
111 = SPI1 interrupts have priority 7 (highest priority)
•
•
•
001 = SPI1 interrupts have priority 1
000 = SPI1 interrupts are disabled

bit 25-24 **SPI1IS<1:0>:** SPI1 Interrupt Vector Subpriority bits
11 = SPI1 interrupts have Subpriority 3 (highest subpriority)
•
•
00 = SPI1 interrupts have Subpriority 0 (lowest subpriority)

bit 23-21 **Reserved:** Write ‘0’; ignore read

bit 20-16 Interrupt Priority Control bits for other peripheral devices

bit 15-13 **Reserved:** Write ‘0’; ignore read

bit 12-8 Interrupt Priority Control bits for other peripheral devices

bit 7-5 **Reserved:** Write ‘0’; ignore read

bit 4-0 Interrupt Priority Control bits for other peripheral devices

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the SPI.

Register 23-17: IPC7: Interrupt Priority Control Register 7(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	SPI2IP<2:0>			SPI2IS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	CMP2IP<2:0>			CMP2IS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	CMP1IP<2:0>			CMP1IS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	PMPIP<2:0>			PMPIS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-29 **Reserved:** Write ‘0’; ignore read

bit 28-26 **SPI2IP<2:0>:** SPI2 Interrupt Vector Priority bits
111 = SPI2 interrupts have priority 7 (highest priority)
•
•
•
001 = SPI2 interrupts have priority 1
000 = SPI2 interrupts are disabled

bit 25-24 **SPI2IS<1:0>:** SPI2 Interrupt Vector Subpriority bits
11 = SPI2 interrupts have Subpriority 3 (highest subpriority)
•
•
00 = SPI2 interrupts have Subpriority 0 (lowest subpriority)

bit 23-21 **Reserved:** Write ‘0’; ignore read

bit 20-16 Interrupt Priority Control bits for other peripheral devices

bit 15-13 **Reserved:** Write ‘0’; ignore read

bit 12-8 Interrupt Priority Control bits for other peripheral devices

bit 7-5 **Reserved:** Write ‘0’; ignore read

bit 4-0 Interrupt Priority Control bits for other peripheral devices

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the SPI.

23.3 MODES OF OPERATION

The SPI module offers the following operating modes:

- 8-Bit, 16-Bit, and 32-bit Data Transmission modes
- 8-Bit, 16-Bit, and 32-bit Data Reception modes
- Master and Slave modes
- Framed SPI modes

23.3.1 8-Bit, 16-Bit, and 32-Bit Operation

The PIC32MX SPI module allows three types of data widths when transmitting and receiving data over an SPI bus. The selection of data width determines the minimum length of SPI data. For example, when the selected data width is 32, all transmission and receptions are performed in 32-bit values. All reads and writes from the CPU are also performed in 32-bit values. Accordingly, the application software should select the appropriate data width to maximize its data throughput.

Two control bits, MODE32 and MODE16 (SPIxCON<11:10>), define the mode of operation. To change the mode of operation on the fly, the SPI module must be idle, i.e., not performing any transactions. If the SPI module is switched off (SPIxCON<15> = 0), the new mode will be available when the module is again switched on.

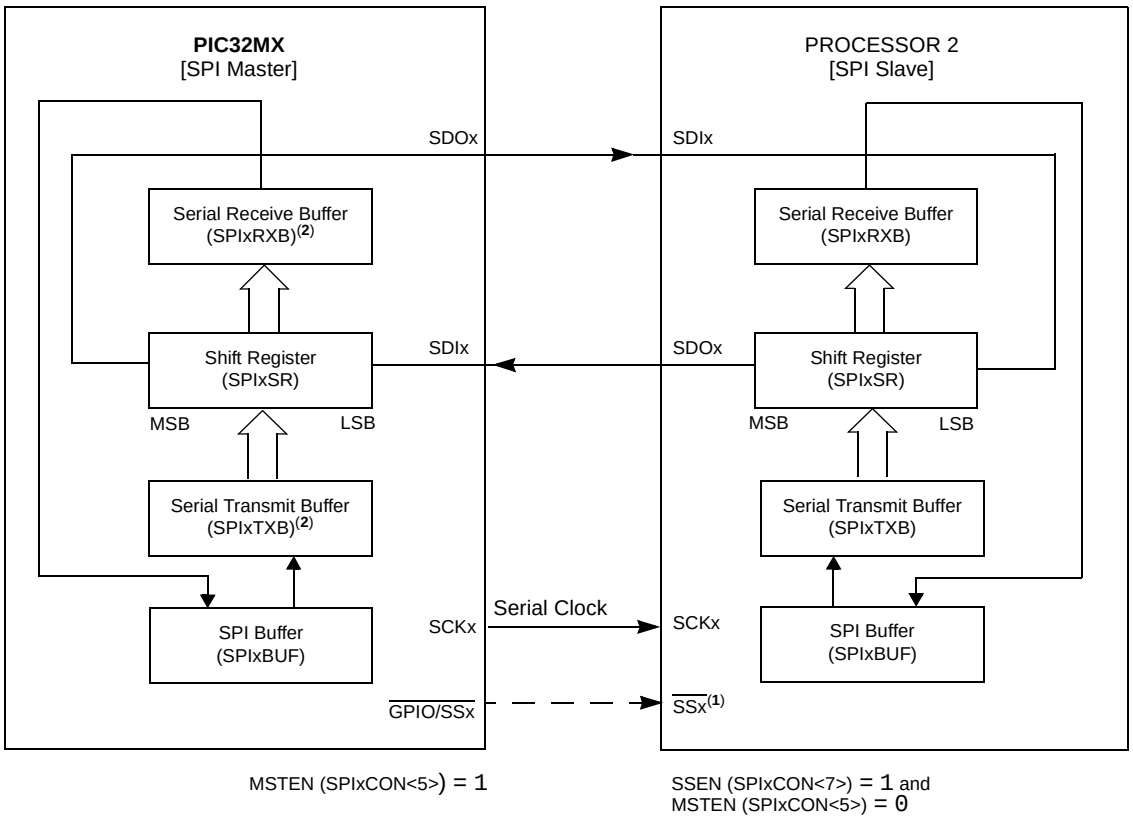
Additionally, the following items should be noted in this context:

- The MODE32 and MODE16 bits should not be changed when a transaction is in progress.
- The first bit to be shifted out from SPIxSR varies with the selected mode of operation:
  - 8-Bit mode, bit 7
  - 16-Bit mode, bit 15
  - 32-Bit mode, bit 31

  In each mode, data is shifted into bit 0 of the SPIxSR.
- The number of clock pulses at the SCKx pin are also dependent on the selected mode of operation:
  - 8-Bit mode, 8 clocks
  - 16-Bit mode, 16 clocks
  - 32-Bit mode, 32 clocks

23.3.2 Master and Slave Modes

Figure 23-6: SPI Master/Slave Connection Diagram



Note 1: Using the $\overline{SSx}$ pin in Slave mode of operation is optional.
2: User must write transmit data to SPIxBUF and read received data from SPIxBUF. The SPIxTXB and SPIxRXB registers are memory mapped to SPIxBUF.

23.3.2.1 Master Mode Operation

Perform the following steps to set up the SPI module for the Master mode of operation:

1. Disable the SPI interrupts in the respective IEC0/1 register.
2. Stop and reset the SPI module by clearing the ON bit.
3. Clear the receive buffer.
4. If SPI interrupts are not going to be used, skip this step and continue to step 5. Otherwise the following additional steps are performed:
   a) Clear the SPIx interrupt flags/events in the respective IFS0/1 register.
   b) Set the SPIx interrupt enable bits in the respective IEC0/1 register.
   c) Write the SPIx interrupt priority and subpriority bits in the respective IPC5/7 register.
5. Write the Baud Rate register, SPIxBRG.
6. Clear the SPIROV bit (SPIxSTAT<6>).
7. Write the desired settings to the SPIxCON register with MSTEN (SPIxCON<5>) = 1.
8. Enable SPI operation by setting the ON bit (SPIxCON<15>).
9. Write the data to be transmitted to the SPIxBUF register. Transmission (and reception) will start as soon as data is written to the SPIxBUF register.

Note:	The SPI device must be turned off prior to changing the mode from Slave to Master.

Note:	When using the Slave Select mode, the $\overline{SSx}$ or another GPIO pin is used to control the slave's $\overline{SSx}$ input. The pin must be controlled in software.

In Master mode, the PBCLK is divided and then used as the serial clock. The division is based on the settings in the SPIxBRG register. The serial clock is output via the SCKx pin to slave devices. Clock pulses are only generated when there is data to be transmitted; except when in Framed mode, when clock is generated continuously. For further information, refer to **23.3.6 "SPI Master Mode Clock Frequency"**.

Bits CKP (SPIxCON<6>) and CKE (SPIxCON<8>) determine on which edge of the clock data transmission occurs.

Note:	The user must turn off the SPI device prior to changing the CKE or CKP bits. Otherwise, the behavior of the device is not guaranteed.

Both data to be transmitted and data that is received are written to, or read from, the SPIxBUF register, respectively.

The following progression describes the SPI module operation in Master mode:

1. Once the module is set up for Master mode operation and enabled, data to be transmitted is written to SPIxBUF register. The SPITBE (SPIxSTAT<3>) bit is cleared.
2. The contents of SPIxTXB are moved to the shift register SPIxSR (see Figure 23-6), and the SPITBE bit is set by the module.
3. A series of 8/16/32 clock pulses shifts 8/16/32 bits of transmit data from SPIxSR to the SDOx pin and simultaneously shifts the data at the SDIx pin into SPIxSR.
4. When the transfer is complete, the following events will occur:
   a) The interrupt flag bit SPIxRXIF is set. SPI interrupts can be enabled by setting the interrupt enable bit SPIxRXIE. The SPIxRXIF flag is not cleared automatically by the hardware.
   b) Also, when the ongoing transmit and receive operation is completed, the contents of SPIxSR are moved to SPIxRXB.
   c) The SPIRBF bit (SPIxSTAT<0>) is set by the module, indicating that the receive buffer is full. Once SPIxBUF is read by the user code, the hardware clears the SPIRBF bit.

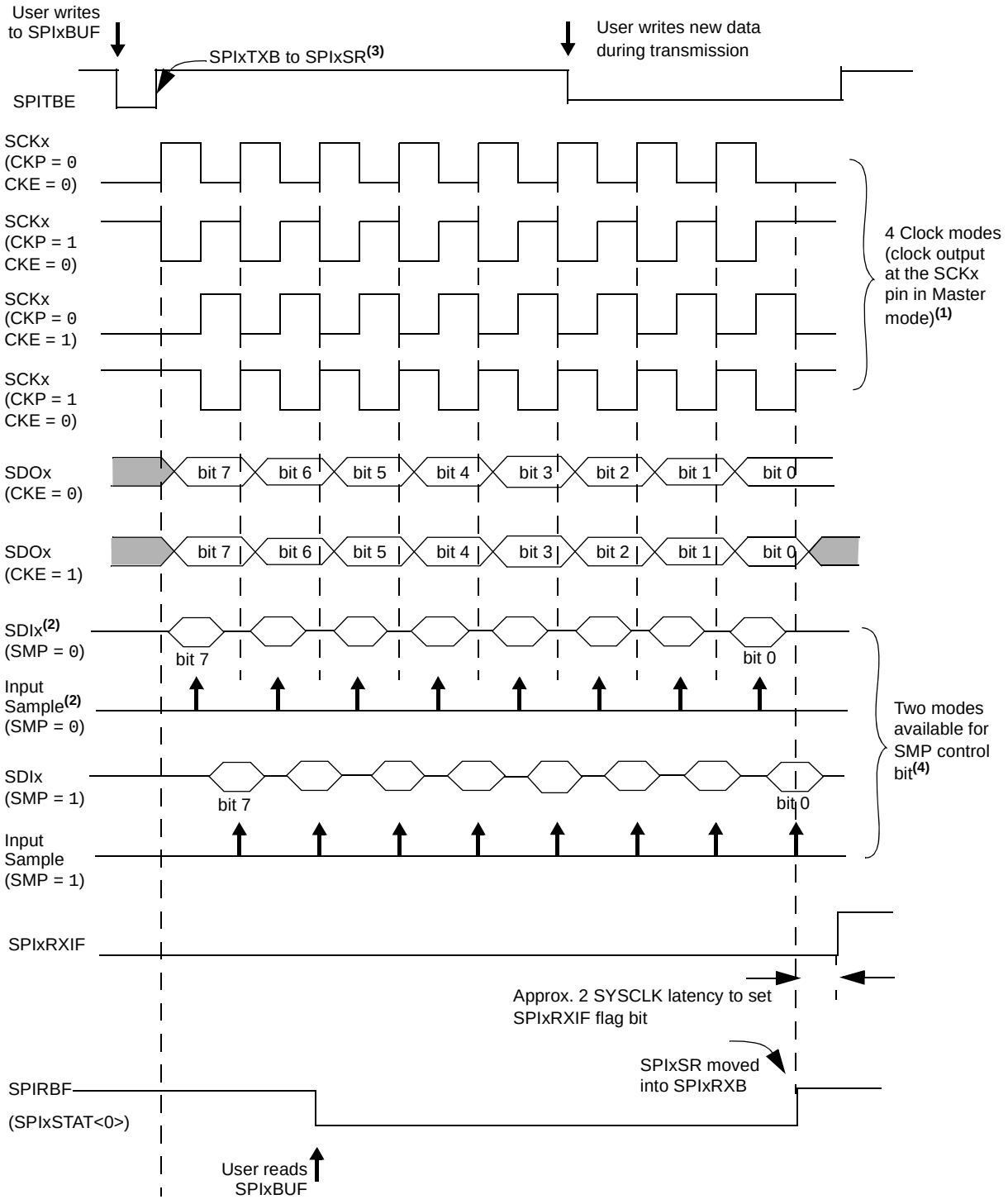
5. If the SPIRBF bit is set (the receive buffer is full) when the SPI module needs to transfer data from SPIxSR to SPIxRXB, the module will set the SPIROV bit (SPIxSTAT<6>) indicating an overflow condition.
6. Data to be transmitted can be written to SPIxBUF by the user software at any time, if the SPITBE (SPIxSTAT<3>) bit is set. The write can occur while SPIxSR is shifting out the previously written data, allowing continuous transmission.

Note: The SPIxSR register cannot be written to directly by the user. All writes to the SPIxSR register are performed through the SPIxBUF register.

Example 23-1: Initialization Code for 16-Bit SPI Master Mode

```
/*
    The following code example will initialize the SPI1 in Master mode.
    It assumes that none of the SPI1 input pins are shared with an analog input. If so, the
    AD1PCFG and corresponding TRIS registers have to be properly configured.
*/
    int rData;

    IEC0CLR=0x03800000;           // disable all interrupts
    SPI1CON = 0;                  // Stops and resets the SPI1.
    rData=SPI1BUF;                // clears the receive buffer
    IFS0CLR=0x03800000;           // clear any existing event
    IPC5CLR=0x1f000000;           // clear the priority
    IPC5SET=0x0d000000;           // Set IPL=3, Subpriority 1
    IEC0SET=0x03800000;           // Enable Rx, Tx and Error interrupts

    SPI1BRG=0x1;                  // use FPB/4 clock frequency
    SPI1STATCLR=0x40;             // clear the Overflow
    SPI1CON=0x8220;               // SPI ON, 8 bits transfer, SMP=1, Master mode

                                  // from now on, the device is ready to transmit and receive
                                     data
    SPI1BUF='A';                  // transmit an A character
```


Figure 23-7: SPI Master Mode Operation in 8-Bit Mode (MODE32 = 0, MODE16 = 0)



Note 1: Four SPI Clock modes are shown here to demonstrate the functionality of bits CKP (SPIxCON<6>) and CKE (SPIxCON<8>). Only one of the four modes can be chosen for operation.

2: The SDI and input samples shown here for two different values of the SMP bit (SPIxCON<9>) are strictly for demonstration purposes. Only one of the two configurations of the SMP bit can be chosen during operation.

3: If there are no pending transmissions, SPIxTXB is transferred to SPIxSR as soon as the user writes to SPIxBUF.

4: Operation for 8-bit mode shown. 16-bit and 32-bit modes are similar.

23.3.2.2 Slave Mode Operation

The following steps are used to set up the SPI module for the Slave mode of operation:

1. If using interrupts, disable the SPI interrupts in the respective IEC0/1 register.
2. Stop and reset the SPI module by clearing the ON bit.
3. Clear the receive buffer.
4. If using interrupts, the following additional steps are performed:
   a) Clear the SPIx interrupt flags/events in the respective IFS0/1 register.
   b) Set the SPIx interrupt enable bits in the respective IEC0/1 register.
   c) Write the SPIx interrupt priority and subpriority bits in the respective IPC5/7 register.
5. Clear the SPIROV bit (SPIxSTAT<6>).
6. Write the desired settings to the SPIxCON register with MSTEN (SPIxCON<5>) = 0.
7. Enable SPI operation by setting the ON bit (SPIxCON<15>).
8. Transmission (and reception) will start as soon as the master provides the serial clock.

Note:	The SPI device must be turned off prior to changing the mode from Master to Slave.

In Slave mode, data is transmitted and received as the external clock pulses appear on the SCKx pin. Bits CKP (SPIxCON<6>) and CKE (SPIxCON<8>) determine on which edge of the clock data transmission occurs.

Both data to be transmitted and data that is received are respectively written into or read from the SPIxBUF register.

The rest of the operation of the module is identical to that in the Master mode.

23.3.2.2.1 Slave Mode Additional Features

The following additional features are provided in the Slave mode:

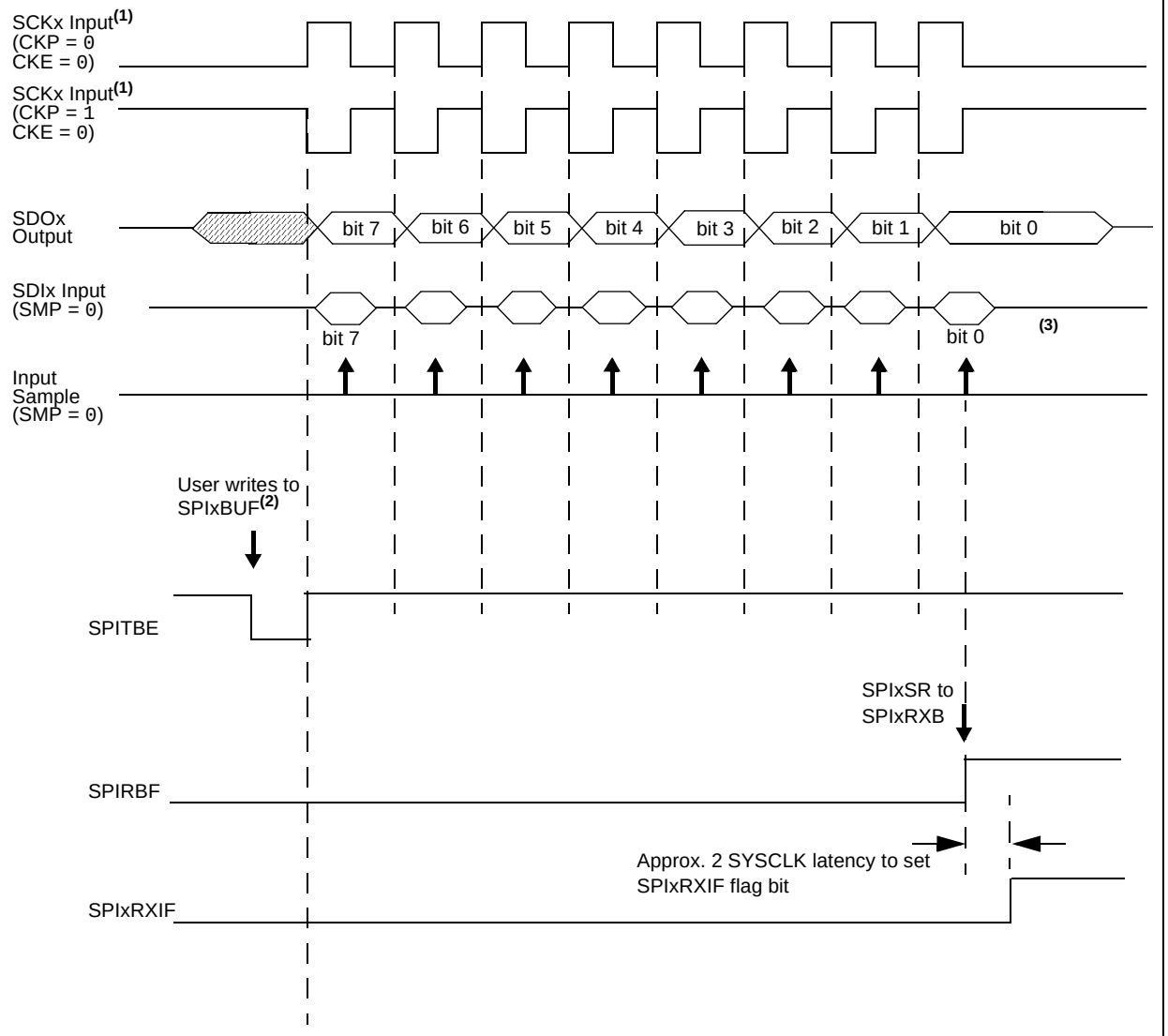
- Slave Select Synchronization
  The $\overline{SSx}$ pin allows a Synchronous Slave mode. If the SSEN bit (SPIxCON<7>) is set, transmission and reception is enabled in Slave mode only if the $\overline{SSx}$ pin is driven to a low state. The port output or other peripheral outputs must not be driven in order to allow the $\overline{SSx}$ pin to function as an input. If the SSEN bit is set and the $\overline{SSx}$ pin is driven high, the SDOx pin is no longer driven and will tri-state even if the module is in the middle of a transmission. An aborted transmission will be retried the next time the $\overline{SSx}$ pin is driven low using the data held in the SPIxTXB register. If the SSEN bit is not set, the $\overline{SSx}$ pin does not affect the module operation in Slave mode.
- SPITBE Status Flag Operation
  The SPITBE bit (SPIxSTAT<3>) has a different function in the Slave mode of operation. The following describes the function of SPITBE for various settings of the Slave mode of operation:
  - If SSEN (SPIxCON<7>) is cleared, the SPITBE is cleared when SPIxBUF is loaded by the user code. It is set when the module transfers SPIxTXB to SPIxSR. This is similar to the SPITBE bit function in Master mode.
  - If SSEN is set, SPITBE is cleared when SPIxBUF is loaded by the user code. However, it is set only when the SPIx module completes data transmission. A transmission will be aborted when the $\overline{SSx}$ pin goes high and may be retried at a later time. So, each data Word is held in SPIxTXB until all bits are transmitted to the receiver.

Note:	Slave Select cannot be used when operating in Frame mode.

Example 23-2: Initialization Code for 16-Bit SPI Slave Mode

```
/*
    The following code example will initialize the SPI1 in Slave mode.
    It assumes that none of the SPI1 input pins are shared with an analog input. If so, the
    AD1PCFG and corresponding TRIS registers have to be properly configured.
*/
    int     rData;

    IEC0CLR=0x03800000;              // disable all interrupts
    SPI1CON = 0;                     // Stops and resets the SPI1.
    rData=SPI1BUF;                   // clears the receive buffer
    IFS0CLR=0x03800000;              // clear any existing event
    IPC5CLR=0x1f000000;              // clear the priority
    IPC5SET=0x0d000000;              // Set IPL=3, Subpriority 1
    IEC0SET=0x03800000;              // Enable Rx, Tx and Error interrupts

    SPI1STATCLR=0x40;                // clear the Overflow
    SPI1CON=0x8000;                  // SPI ON, 8 bits transfer, Slave mode

                                     // from now on, the device is ready to receive and
                                         transmit data
```

Figure 23-8: SPI Slave Mode Operation in 8-Bit Mode with Slave Select Pin Disabled (MODE32 = 0, MODE16 = 0, SSEN = 0)

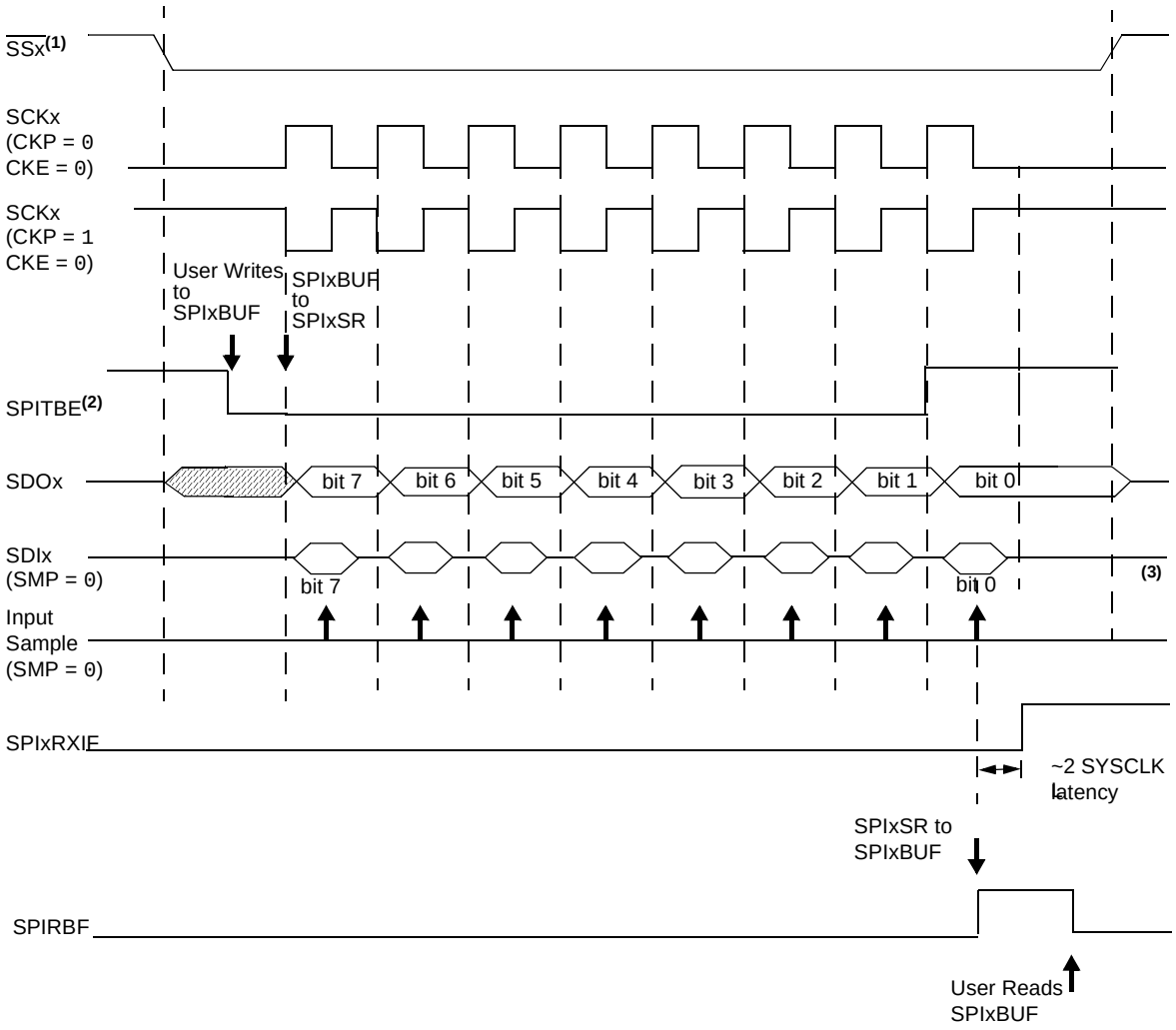


Note 1: Two SPI Clock modes are shown here only to demonstrate the functionality of bits CKP (SPIxCON<6>) and CKE (SPIxCON<8>). Any combination of CKP and CKE bits can be chosen for module operation.

2: If there are no pending transmissions or a transmission is in progress, SPIxBUF is transferred to SPIxSR as soon as the user writes to SPIxBUF.

3: Operation for 8-bit mode is shown. 16-bit and 32-bit modes are similar.

Figure 23-9: SPI Slave Mode Operation in 8-Bit Mode with Slave Select Pin Enabled (MODE32 = 0, MODE16 = 0, SSEN = 1)



Note 1: When the SSEN (SPIxCON<7>) bit is set to '1', the $\overline{SSx}$ pin must be driven low so as to enable transmission and reception in Slave mode.

2: Transmit data is held in SPIxTXB and SPITBE (SPIxSTAT<2>) remains clear until all bits are transmitted.

3: Operation for 8-bit mode is shown. 16-bit and 32-bit modes are similar.

23.3.3 SPI Error Handling

When a new data word has been shifted into shift register SPIxSR and the previous contents of receive register SPIxRXB have not been read by the user software, the SPIROV bit (SPIx-STAT<6>) will be set. The module will not transfer the received data from SPIxSR to the SPIxRXB. Further data reception is disabled until the SPIROV bit is cleared. The SPIROV bit is not cleared automatically by the module and must be cleared by the user software.

23.3.4 SPI Receive-Only Operation

Setting the control bit DISSDO (SPIxCON<12>) disables transmission at the SDOx pin. This allows the SPIx module to be configured for a Receive-Only mode of operation. The SDOx pin will be controlled by the respective port function if the DISSDO bit is set.

The DISSDO function is applicable to all SPI operating modes.

23.3.5 Framed SPI Modes

The module supports a very basic framed SPI protocol while operating in either Master or Slave modes. The following features are provided in the SPI module to support Framed SPI modes:

- The control bit FRMEN (SPIxCON<31>) enables Framed SPI mode and causes the $\overline{SSx}$ pin to be used as a frame synchronization pulse input or output pin. The state of SSEN (SPIxCON<7>) is ignored.
- The control bit FRMSYNC (SPIxCON<30>) determines whether the $\overline{SSx}$ pin is an input or an output, i.e., whether the module receives or generates the frame synchronization pulse.
- The FRMPOL (SPIxCON<29>) determines the frame synchronization pulse polarity for a single SPI clock cycle.

The following Framed SPI modes are supported by the SPI module:

- Frame Master mode

  The SPI module generates the frame synchronization pulse and provides this pulse to other devices at the $\overline{SSx}$ pin.
- Frame Slave mode

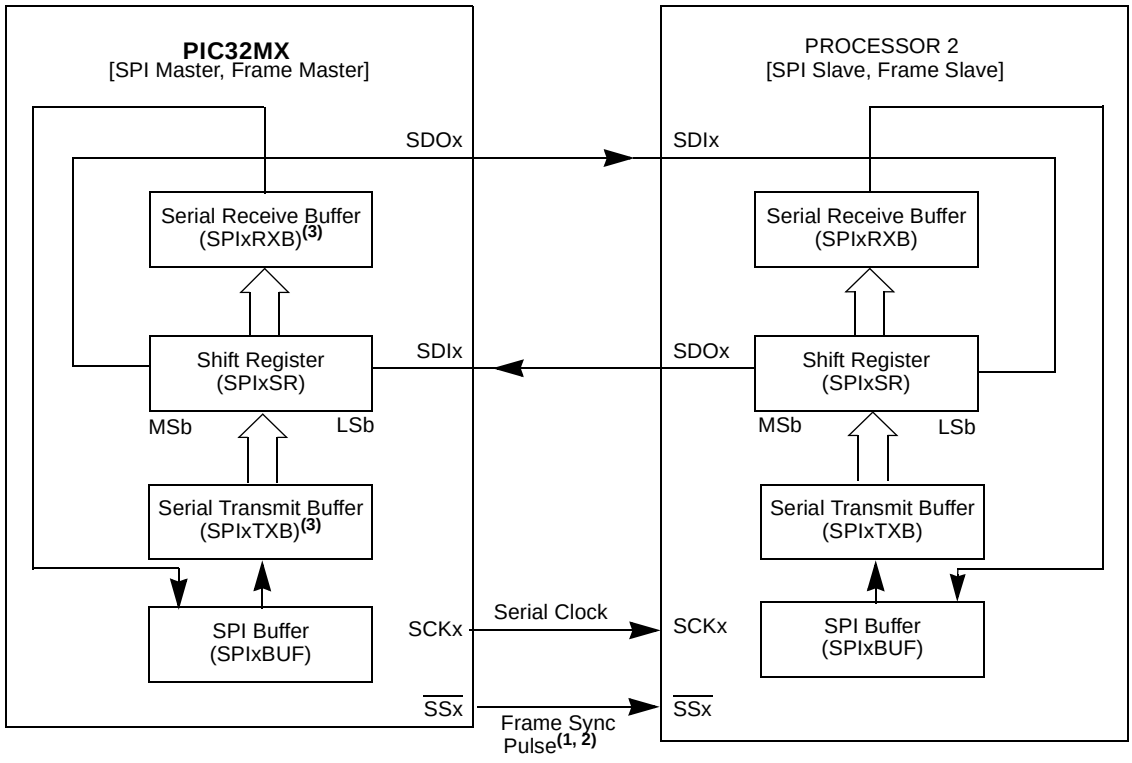
  The SPI module uses a frame synchronization pulse received at the $\overline{SSx}$ pin.

The Framed SPI modes are supported in conjunction with the Master and Slave modes. Thus, the following Framed SPI Configurations are available:

- SPI Master mode and Frame Master mode
- SPI Master mode and Frame Slave mode
- SPI Slave mode and Frame Master mode
- SPI Slave mode and Frame Slave mode

These four modes determine whether or not the SPIx module generates the serial clock and the frame synchronization pulse.

Figure 23-10: SPI Master, Frame Master Connection Diagram



Note 1: In Framed SPI modes, the $\overline{SSx}$ pin is used to transmit/receive the frame synchronization pulse.
2: Framed SPI modes require the use of all four pins, i.e., using the $\overline{SSx}$ pin is not optional.
3: The SPIxTXB and SPIxRXB registers are memory mapped to the SPIxBUF register.

23.3.5.1 SCKx in Framed SPI Modes

When FRMEN (SPIxCON<31>) = 1 and MSTEN (SPIxCON<5>) = 1, the SCKx pin becomes an output and the SPI clock at SCKx becomes a free-running clock.

When FRMEN = 1 and MSTEN = 0, the SCKx pin becomes an input. The source clock provided to the SCKx pin is assumed to be a free-running clock.

The polarity of the clock is selected by bit CKP (SPIxCON<6>). Bit CKE (SPIxCON<8>) is not used for the Framed SPI modes.

When CKP = 0, the frame sync pulse output and the SDOx data output change on the rising edge of the clock pulses at the SCKx pin. Input data is sampled at the SDIx input pin on the falling edge of the serial clock.

When CKP = 1, the frame sync pulse output and the SDOx data output change on the falling edge of the clock pulses at the SCKx pin. Input data is sampled at the SDIx input pin on the rising edge of the serial clock.

23.3.5.2 SPIx Buffers in Framed SPI Modes

When FRMSYNC (SPIxCON<30>) = 0, the SPIx module is in the Frame Master mode of operation. In this mode, the frame sync pulse is initiated by the module when the user software writes the transmit data to SPIxBUF location (thus writing the SPIxTXB register with transmit data). At the end of the frame sync pulse, SPIxTXB is transferred to SPIxSR and data transmission/reception begins.

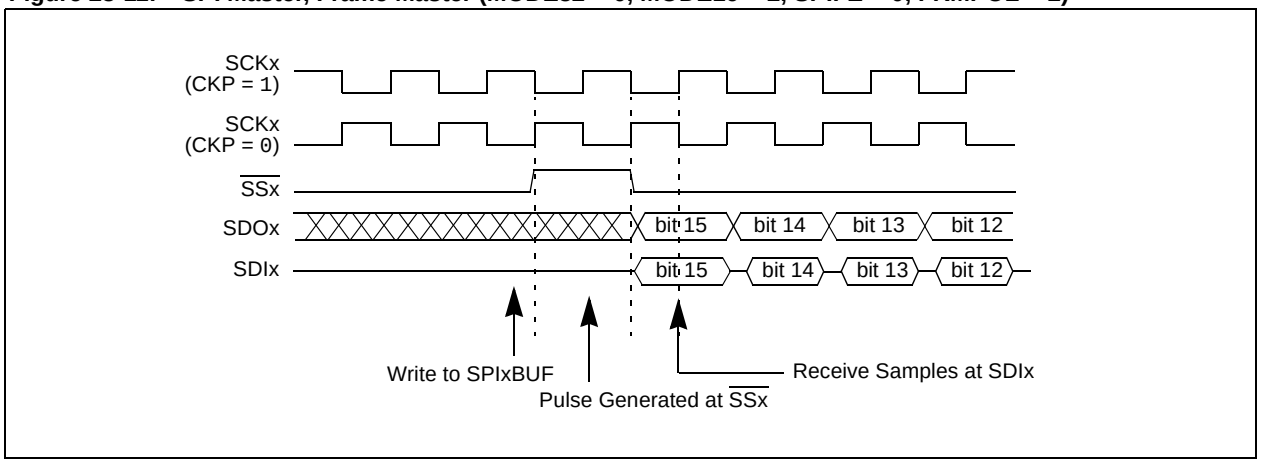
When FRMSYNC = 1, the module is in Frame Slave mode. In this mode, the frame sync pulse is generated by an external source. When the module samples the frame sync pulse, it will transfer the contents of the SPIxTXB register to SPIxSR, and data transmission/ reception begins. The user must make sure that the correct data is loaded into the SPIxBUF for transmission before the frame sync pulse is received.

Note:	Receiving a frame sync pulse will start a transmission, regardless of whether or not data was written to SPIxBUF. If a write was not performed, zeros will be transmitted.

23.3.5.3 SPI Master Mode and Frame Master Mode

This Framed SPI mode is enabled by setting bits MSTEN (SPIxCON<5>) and FRMEN (SPIxCON<31>) to '1', and bit FRMSYNC (SPIxCON<30>) to '0'. In this mode, the serial clock will be output continuously at the SCKx pin, regardless of whether the module is transmitting. When SPIxBUF is written, the $\overline{SSx}$ pin will be driven active, high or low depending on bit FRMPOL (SPIxCON<29>), on the next transmit edge of the SCKx clock. The $\overline{SSx}$ pin will be high for one SCKx clock cycle. The module will start transmitting data on the next transmit edge of the SCKx, as shown in Figure 23-11. A connection diagram indicating signal directions for this operating mode is shown in Figure 23.9.

Figure 23-11: SPI Master, Frame Master (MODE32 = 0, MODE16 = 1, SPIFE = 0, FRMPOL = 1)



23.3.5.4 SPI Master Mode and Frame Slave Mode

This Framed SPI mode is enabled by setting bits MSTEN (SPIxCON<5>), FRMEN (SPIxCON<31>), and bits FRMSYNC (SPIxCON<30>) to '1'. The $\overline{SSx}$ pin is an input, and it is sampled on the sample edge of the SPI clock. When it is sampled active, high or low depending on bit FRMPOL (SPIxCON<29>), data will be transmitted on the subsequent transmit edge of the SPI clock, as shown in Figure 23-12. The interrupt flag SPIxIF is set when the transmission is complete. The user must make sure that the correct data is loaded into SPIxBUF for transmission before the signal is received at the $\overline{SSx}$ pin. A connection diagram indicating signal directions for this operating mode is shown in Figure 23-13.

Figure 23-12: SPI Master, Frame Slave (MODE32 = 0, MODE16 = 1, SPIFE = 0, FRMPOL = 1)

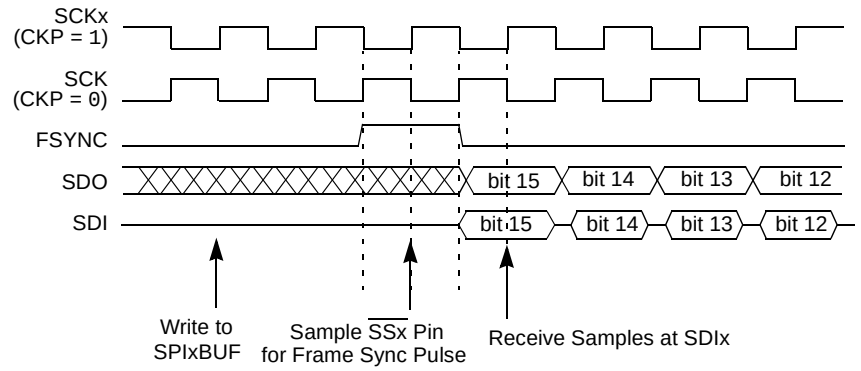
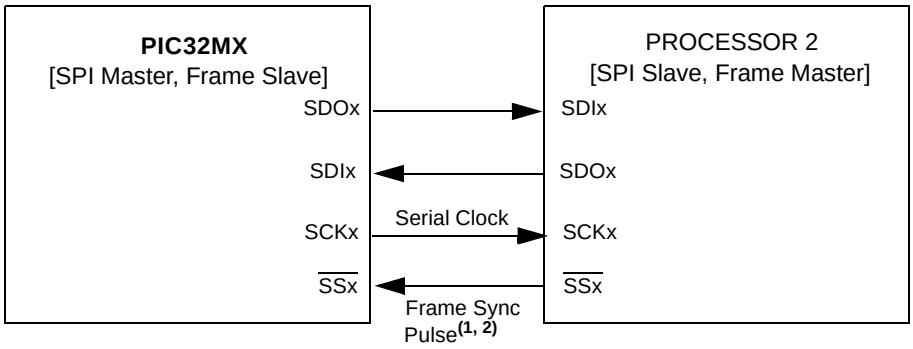


Figure 23-13: SPI Master, Frame Slave Connection Diagram

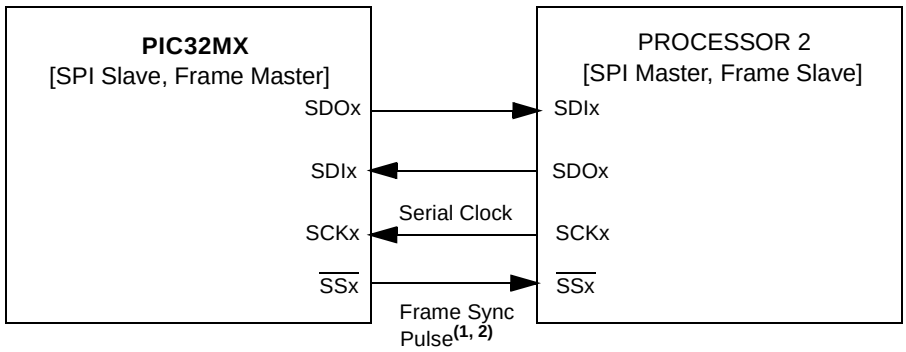


Note 1: In Framed SPI modes, the $\overline{SSx}$ pin is used to transmit/receive the frame synchronization pulse.
2: Framed SPI modes require the use of all four pins (i.e., using the $\overline{SSx}$ pin is not optional).

23.3.5.5 SPI Slave Mode and Frame Master Mode

This Framed SPI mode is enabled by setting bit MSTEN (SPIxCON<5>) to '0', bit FRMEN (SPIxCON<31>) to '1' and bit FRMSYNC (SPIxCON<30>) to '0'. The input SPI clock will be continuous in Slave mode. The $\overline{SSx}$ pin will be an output when bit FRMSYNC is low. Therefore, when SPIBUF is written, the module will drive the $\overline{SSx}$ pin active, high or low depending on bit FRMPOL (SPIxCON<29>), on the next transmit edge of the SPI clock. The $\overline{SSx}$ pin will be driven high for one SPI clock cycle. Data transmission will start on the next SPI clock transmit edge. A connection diagram indicating signal directions for this operating mode is shown in Figure 23-14.

Figure 23-14: SPI Slave, Frame Master Connection Diagram

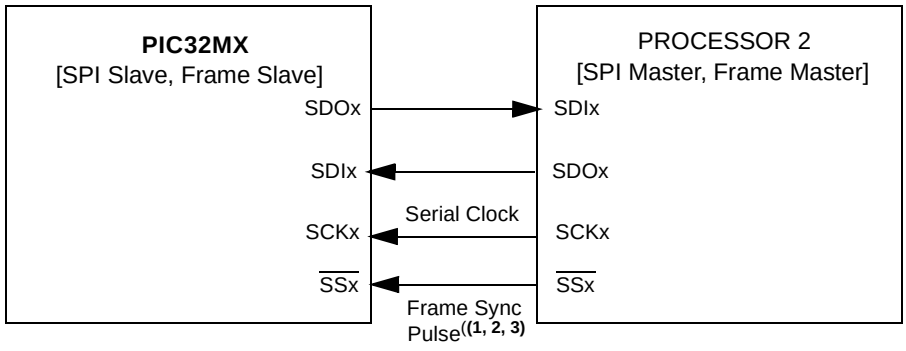


Note 1: In Framed SPI modes, the $\overline{SSx}$ pin is used to transmit/receive the frame sync pulse.
2: Framed SPI modes require the use of all four pins (i.e., using the $\overline{SSx}$ pin is not optional).

23.3.5.6 SPI Slave Mode and Frame Slave Mode

This Framed SPI mode is enabled by setting bits MSTEN (SPIxCON<5>) to '0', FRMEN (SPIxCON<31>) to '1', and FRMSYNC (SPIxCON<30>) to '1'. Therefore, both the SCKx and $\overline{SSx}$ pins will be inputs. The $\overline{SSx}$ pin will be sampled on the sample edge of the SPI clock. When $\overline{SSx}$ is sampled active, high or low depending on bit FRMPOL (SPIxCON<29>), data will be transmitted on the next transmit edge of SCKx. A connection diagram indicating signal directions for this operating mode is shown in Figure 23-15.

Figure 23-15: SPI Slave, Frame Slave Connection Diagram



Note 1: In Framed SPI modes, the $\overline{SSx}$ pin is used to transmit/receive the frame sync pulse.
2: Framed SPI modes require the use of all four pins (i.e., using the $\overline{SSx}$ pin is not optional).
3: Slave Select is not available when using Frame mode as a Slave device.

23.3.6 SPI Master Mode Clock Frequency

The SPI module allows flexibility in baud rate generation through the 9-bit SPIxBRG register. SPIxBRG is readable and writable, and determines the baud rate. The peripheral clock PBCLK provided to the SPI module is a divider function of the CPU core clock. This clock is divided based on the value loaded into SPIxBRG. The SCKx clock obtained by dividing PBCLK is of 50% duty cycle and it is provided to the external devices via the SCKx pin.

Note: The SCKx clock is not free running for nonframed SPI modes. It will only run for 8, 16, or 32 pulses when SPIxBUF is loaded with data. It will however, be continuous for Framed modes.

Equation 23-1 defines the SCKx clock frequency as a function of SPIxBRG settings.

Equation 23-1:

$$F_{SCK} = \frac{F_{PB}}{2 * (SPIxBRG+1)}$$

Therefore, the maximum baud rate possible is $F_{PB}/2$ (SPIXBRG = 0), and the minimum baud rate possible is $F_{PB}/1024$.

Some sample SPI clock frequencies (in kHz) are shown in the table below:

Table 23-3: Sample SCKx Frequencies

SPIxBRG Setting	0	15	31	63	85	127
F_{PB} = 50 MHz	25.00 MHz	1.56 MHz	781.25 kHz	390.63 kHz	290.7 kHz	195.31 kHz
F_{PB} = 40 MHz	20.00 MHz	1.25 MHz	625.00 kHz	312.50 kHz	232.56 kHz	156.25 kHz
F_{PB} = 25 MHz	12.50 MHz	781.25 kHz	390.63 kHz	195.31 kHz	145.35 kHz	97.66 kHz
F_{PB} = 20 MHz	10.00 MHz	625.00 kHz	312.50 kHz	156.25 kHz	116.28 kHz	78.13 kHz
F_{PB} = 10 MHZ	5.00 MHz	312.50 kHz	156.25 kHz	78.13 kHz	58.14 kHz	39.06 kHz

Note: Not all clock rates are supported. For further information, refer to the SPI timing specifications in the specific device data sheet.

23.4 INTERRUPTS

The SPI module has the ability to generate interrupts reflecting the events that occur during the data communication. The following types of interrupts can be generated:

- Receive data available interrupts, signalled by SPI1RXIF (IFS0<25>), SPI2RXIF(IFS1<7>). This event occurs when there is new data assembled in the SPIxBUF receive buffer.
- Transmit buffer empty interrupts, signalled by SPI1TXIF (IFS0<24>), SPI2TXIF (IFS1<6>). This event occurs when there is space available in the SPIxBUF transmit buffer and new data can be written.
- Receive buffer overflow interrupts, signalled by SPI1EIF (IFS0<23>), SPI2EIF(IFS1<5>). This event occurs when there is an overflow condition for the SPIxBUF receive buffer, i.e., new receive data assembled but the previous one not read.

All these interrupt flags must be cleared in software.

To enable the SPI interrupts, use the respective SPI interrupt enable bits:

- SPI1RXIE (IEC0<25>) and SPI2RXIE (IEC1<7>)
- SPI1TXIE (IEC0<24>) and SPI2TXIE (IEC1<6>)
- SPI1FIE (IEC0<23>) and SPI2FIE (IEC1<5>)

The interrupt priority level bits and interrupt subpriority level bits must be also be configured:

- SPI1IP (IPC5<28:26>), SPI1IS (IPC5<25:24>)
- SPI2IP (IPC7<28:26>), SPI2IS (IPC7<25:24>)

Refer to **Section 8. “Interrupts”** for further details.

23.4.1 Interrupt Configuration

Each SPI module has 3 dedicated interrupt flag bits: SPIxEIF, SPIxRXIF, and SPIxTXIF, and corresponding interrupt enable/mask bits SPIxEIE, SPIxRXIE, and SPIxTXIE. These bits are used to determine the source of an interrupt, and to enable or disable an individual interrupt source. Note that all the interrupt sources for a specific SPI module share one interrupt vector. Each SPI module can have its own priority level independent of other SPI modules.

SPIxTXIF is set when the SPI transmit buffer is empty and another character can be written to the SPIxBUF register. SPIxRXIF is set when there is a received character available in SPIxBUF. SPIxEIF is set when a Receive Overflow condition occurs.

Note that bits SPIxTXIF, SPIxRXIF, and SPIxEIF will be set without regard to the state of the corresponding enable bit. IF bits can be polled by software if desired.

Bits SPIxEIE, SPIxTXIE, SPIxRXIE are used to define the behavior of the Interrupt Controller (INT) when a corresponding SPIxEIF, SPIxTXIF, or SPIxRXIF bit is set. When the corresponding IE bit is clear, the INT module does not generate a CPU interrupt for the event. If the IE bit is set, the INT module will generate an interrupt to the CPU when the corresponding IF bit is set (subject to the priority and subpriority as outlined below).

It is the responsibility of the user’s software routine that services a particular interrupt to clear the appropriate interrupt flag bit before the service routine is complete.

The priority of each SPI module can be set independently with the SPIxIP<2:0> bits. This priority defines the priority group to which the interrupt source will be assigned. The priority groups range from a value of 7 (the highest priority), to a value of 0 (which does not generate an interrupt). An interrupt being serviced will be preempted by an interrupt in a higher priority group.

The subpriority bits allow setting the priority of an interrupt source within a priority group. The values of the subpriority SPIxIS<1:0> range from 3 (the highest priority) to 0, the lowest priority. An interrupt within the same priority group but having a higher subpriority value will not preempt a lower subpriority interrupt that is in progress.

The priority group and subpriority bits allow more than one interrupt source to share the same priority and subpriority. If simultaneous interrupts occur in this configuration the natural order of the interrupt sources within a Priority/subpriority group pair determine the interrupt generated. The natural priority is based on the vector numbers of the interrupt sources. The lower the vector number the higher the natural priority of the interrupt. Any interrupts that were overridden by natural order will then generate their respective interrupts based on Priority, subpriority, and natural order, after the interrupt flag for the current interrupt is cleared.

After an enabled interrupt is generated, the CPU will jump to the vector assigned to that (refer to Table 23-4) interrupt. The vector number for the interrupt is the same as the natural order number. The CPU will then begin executing code at the vector address. The user's code at this vector address should perform any application-specific operations required, and clear interrupt flags SPIxEIF, SPIxTXIF, or SPIxRXIF, and then exit. Refer to the vector address table details in the **Section 8. "Interrupts"** for more information on interrupts.

Table 23-4: SPI Interrupt Vectors for Various Offsets with EBASE = 0x8000:0000

Interrupt	Vector/Natural Order	IRQ Number	Vector Address IntCtl.VS = 0x01	Vector Address IntCtl.VS = 0x02	Vector Address IntCtl.VS = 0x04	Vector Address IntCtl.VS = 0x08	Vector Address IntCtl.VS = 0x10
SPI1E	23	23	8000 04E0	8000 07C0	8000 0D80	8000 1900	8000 3000
SPI1TX	23	24	8000 04E0	8000 07C0	8000 0D80	8000 1900	8000 3000
SPI1RX	23	25	8000 04E0	8000 07C0	8000 0D80	8000 1900	8000 3000
SPI2IE	31	37	8000 05C0	8000 0980	8000 1100	8000 2000	8000 3E00
SPI2TX	31	38	8000 05C0	8000 0980	8000 1100	8000 2000	8000 3E00
SPI2RX	31	39	8000 05C0	8000 0980	8000 1100	8000 2000	8000 3E00

Example 23-3: SPI Initialization with Interrupts Enabled Code Example

```
/*
    The following code example illustrates an SPI1 interrupt configuration.
    When the SPI1 interrupt is generated, the cpu will jump to the vector assigned to SPI1
    interrupt.
    It assumes that none of the SPI1 input pins are shared with an analog input. If so, the
    AD1PCFG and corresponding TRIS registers have to be properly configured.
*/

    int rData;

    IEC0CLR=0x03800000;             // disable all SPI interrupts
    SPI1CON = 0;                    // Stops and resets the SPI1.
    rData=SPI1BUF;                  // clears the receive buffer
    IFS0CLR=0x03800000;             // clear any existing event
    IPC5CLR=0x1f000000;             // clear the priority
    IPC5SET=0x0d000000;             // Set IPL=3, Subpriority 1
    IEC0SET=0x03800000;             // Enable Rx, Tx and Error interrupts

    SPI1BRG=0x1;                    // use FPB/4 clock frequency
    SPI1STATCLR=0x40;               // clear the Overflow
    SPI1CON=0x8220;                 // SPI ON, 8 bits transfer, SMP=1, Master mode
```

Example 23-4: SPI1 ISR Code Example

```
/*
    The following code example demonstrates a simple interrupt service routine for SPI1
    interrupts. The user's code at this vector should perform any application specific operation
    and must clear the SPI1 interrupt flags before exiting.
*/

void __ISR(_SPI_1_VECTOR, ipl3)__SPI1Interrupt(void)
{
                              // ... perform application specific operations in response to the
                              // interrupt

    IFS0CLR = 0x03800000;     // Be sure to clear the SPI1 interrupt flags
                              // before exiting the service routine.
}
```

Note: The SPI1 ISR code example shows MPLAB® C32 C compiler specific syntax. Refer to your compiler manual regarding support for ISRs.

23.5 OPERATION IN POWER-SAVING AND DEBUG MODES

Note: In this manual, a distinction is made between a power mode as it is used in a specific module, and a power mode as it is used by the device, e.g., Sleep mode of the Comparator and SLEEP mode of the CPU. To indicate which type of power mode is intended, uppercase and lowercase letters (Sleep, Idle, Debug) signify a module power mode, and all uppercase letters (SLEEP, IDLE, DEBUG) signify a device power mode.

23.5.1 SLEEP Mode

When the device enters SLEEP mode, the system clock is disabled. The exact SPI module operation during SLEEP mode depends on the current mode of operation. The following subsections describe mode-specific behavior.

23.5.1.1 Master Mode in SLEEP Mode

The following items should be noted in SLEEP mode:

- The Baud Rate Generator is stopped and reset.
- On-going transmission and reception sequences are aborted. The module will not resume aborted sequences when SLEEP mode is exited.
- Once in SLEEP mode, the module will not transmit or receive any new data.

Note: To prevent unintentional abort of transmit and receive sequences, wait for the current transmission to be completed before activating SLEEP mode.

23.5.1.2 Slave Mode in SLEEP Mode

In the Slave mode, the SPI module operates from the SCK provided by an external SPI Master. Since the clock pulses at SCKx are externally provided for Slave mode, the module will continue to function in SLEEP mode. It will complete any transactions during the transition into SLEEP. On completion of a transaction, the SPIRBF flag is set. Consequently, bit SPIxRXIF will be set. If SPI interrupts are enabled (SPIxRXIE = 1) and the SPI interrupt priority level is greater than the present CPU priority level, the device will wake from SLEEP mode and the code execution will resume at the SPIx interrupt vector location. If the SPI interrupt priority level is lower than or equal to the present CPU priority level, the CPU will remain in IDLE mode.

The module is not reset on entering SLEEP mode if it is operating as a slave device. Register contents are not affected when the SPIx module is going into or coming out of SLEEP mode.

23.5.2 IDLE Mode

When the device enters IDLE mode, the system clock sources remain functional.

23.5.2.1 Master Mode in IDLE Mode

Bit SIDL (SPIxCON<13>) selects whether the module will stop or continue functioning in IDLE mode.

- If SIDL = 1, the module will discontinue operation in IDLE mode. The module will perform the same procedures when stopped in IDLE mode that it does for SLEEP mode.
- If SIDL = 0, the module will continue operation in IDLE mode.

23.5.2.2 Slave Mode in IDLE Mode

The module will continue operation in IDLE mode irrespective of the SIDL setting. The behavior is identical to the one in SLEEP mode.

23.5.3 DEBUG Mode

Bit FRZ (SPIxCON<14>) determines whether the SPI module will run or stop while the CPU is executing DEBUG exception code (i.e., application is halted) in DEBUG mode. When FRZ = 0, the SPI module continues to run, even when the application is halted in DEBUG mode. When FRZ = 1 and the application is halted in DEBUG mode, the behavior is different from Master-to-Slave mode.

23.5.3.1 Freeze in Master Mode

When FRZ = 1 and the application is halted in DEBUG mode, the module will freeze its operations and make no changes to the state of the SPI module, such that it will continue exactly as it left off. In other words, the transmission/reception **is not aborted** during this halt.

23.5.3.2 Freeze in Slave Mode

In Slave mode with an externally provided SCK, the module will continue to operate, even though it is frozen (FRZ = 1), i.e., the shift register is functional. However, when data is received in the shift register before DEBUG mode is exited, the data that has been received is ignored, i.e., not transferred to SPIxBUF.

23.5.3.3 Operation of SPIxBUF

23.5.3.3.1 Reads During DEBUG Mode

During DEBUG mode, SPIxBUF can be read; but the read operation does not affect any Status bits. For example, if bit SPIRBF (SPIxSTAT<0>) is set when DEBUG mode is entered, it will remain set on EXIT From DEBUG mode, even though the SPIxBUF register was read in DEBUG mode.

23.5.3.3.2 Writes During DEBUG Mode

When FRZ is set, write functionality depends on whether the SPI is in Master or Slave mode.

In Master mode: the write operation will place the data in the buffer, but the transmission will not start until the DEBUG mode is exited.

In Slave mode: the write operation will place the data in the buffer, and the data will be sent out whenever the Master initiates a new transaction, even if the device is still in DEBUG mode.

Note:	The FRZ bit is readable and writable only when the CPU is executing in Debug Exception mode. In all other modes, the FRZ bit reads as '0'. If FRZ bit is changed during DEBUG mode, the new value does not take effect until the current Debug Exception mode is exited and re-entered. During the Debug Exception mode, the FRZ bit reads the state of the peripheral when entering DEBUG mode.

23.6 EFFECTS OF VARIOUS RESETS

23.6.1 Device Reset

All SPI registers are forced to their Reset states upon a device Reset. When the asynchronous Reset input goes active, the SPI logic:

- resets all fields in SPIxCON and SPIxSTAT
- resets the transmit and receive buffers (SPIx-BUF) to the empty state
- resets the Baud Rate Generator

23.6.2 Power-on Reset

All SPI registers are forced to their Reset states when a Power-on Reset occurs.

23.6.3 Watchdog Timer Reset

All SPI registers are forced to their Reset states when a Watchdog Timer Reset occurs.

23.7 PERIPHERALS USING SPI MODULES

There are no other peripherals using the SPI module.

23.8 I/O PIN CONTROL

Enabling the SPI modules will configure the I/O pin direction as defined by the SPI control bits (see Table 23-5). The port TRIS and LATCH registers will be overridden.

Table 23-5: I/O Pin Configuration for Use with SPI Modules

Required Settings for Module Pin Control							
I/O Pin Name	Required	Module Control(3)	Bit Field(3)	TRIS(4)	Pin Type	Buffer Type	Description
SCK1, SCK2	Yes	ON and MSTEN	—	X	O	CMOS	SPI1, SPI2 module Clock Output in Master mode.
SCK1, SCK2	Yes	ON and $\overline{\text{MSTEN}}$	—	X(5)	I	CMOS	SPI1, SPI2 module Clock Input in Slave mode.
SDI1, SDI2	Yes	ON	—	X(5)	I	CMOS	SPI1, SPI2 module Data Receive pin
SDO1, SDO2	Yes(1)	ON	DISSDO	X	O	CMOS	SPI1, SPI2 module Data Transmit pin
$\overline{\text{SS1}}$, $\overline{\text{SS2}}$	Yes(2)	ON and $\overline{\text{FRMEN}}$ and $\overline{\text{MSTEN}}$	SSEN	X(5)	I	CMOS	SPI1, SPI2 module Slave Select Control pin.
$\overline{\text{SS1}}$, $\overline{\text{SS2}}$	Yes	ON and FRMEN and FRMSYNC	—	X(5)	I	CMOS	SPI1, SPI2 Frame Sync Pulse input in Frame mode.
$\overline{\text{SS1}}$, $\overline{\text{SS2}}$	Yes	ON and FRMEN and $\overline{\text{FRMSYNC}}$	—	X	O	CMOS	SPI1,SPI2 Frame Sync Pulse output in Frame mode.

Legend: CMOS = CMOS compatible input or output
ST = Schmitt Trigger input with CMOS levels
I = Input
O = Output

Note 1: The SDO pins are only required when SPI data output is needed. Otherwise, these pins can be used for general purpose IO and require the user to set the corresponding TRIS control register bits.

2: The Slave Select pins are only required when a select signal to the slave device is needed. Otherwise, these pins can be used for general purpose IO and require the user to set the corresponding TRIS control register bits.

3: These bits are contained in the SPIxCON register.

4: The setting of the TRIS bit is irrelevant.

5: If the input pin is shared with an analog input, then the AD1PCFG and corresponding TRIS register has to be properly set to configure this input as digital.

23.9 DESIGN TIPS

Question 1: ***Can I use the SSx pin as an output to a slave device when the PIC32MX SPI module is configured in Master mode?***

Answer: Yes, you can. Notice, however, that the SSx pin is not driven by the SPI Master. You have to drive the bit yourself and pulse it before the SPI transmission takes place. You can use any other I/O pin for that purpose.

Question 2: ***If I do not use the SDO output for my SPI module, is this I/O pin available as a general purpose I/O pin?***

Answer: Yes. If you are not interested in transmitting data, only receiving, you can use the SDO pin as a general I/O pin. This is mainly useful for SPI modules that are configured as SPI slave devices. Note that when used as a general purpose I/O pin, the user is responsible for configuring the respective data direction register (TRIS) for input or output.

23.10 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the PIC32MX device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the SPI module are:

Title	Application Note #
Interfacing Microchip's MCP41XXX/MCP42XXX Digital Potentiometers to a PIC® Microcontroller	AN746
Interfacing Microchip's MCP3201 Analog-to-Digital Converter to the PIC® Microcontroller	AN719

Note: Please visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the PIC32MX family of devices.

23.11 REVISION HISTORY

Revision A (July 2007)

This is the initial released version of this document.

Revision B (October 2007)

Revised Examples 23-1, 23-2, 23-3; Table 23-5.

Revision C (October 2007)

Updated document to remove Confidential status.

Revision D (April 2008)

Revised status to Preliminary; Revised U-0 to r-x.

Revision E (June 2008)

Added Footnote number to Registers 12-12-17; Revised Example 23-4; Revised Figure 23-8; Change Reserved bits from “Maintain as” to “Write”; Added Note to ON bit (SPIxCON Register).

NOTES:

Section 24. Inter-Integrated Circuit

HIGHLIGHTS

This section of the manual contains the following topics:

24.1 OVERVIEW

The Inter-Integrated Circuit (I^2C™) module is a serial interface useful for communicating with other peripheral or microcontroller devices. These peripheral devices may be serial EEPROMs, display drivers, A/D converters, etc.

The I^2C module can operate in any of the following I^2C systems:

- As a slave device
- As a master device in a single master system (slave may also be active)
- As a master/slave device in a multi-master system (bus collision detection and arbitration available)

The I^2C module contains independent I^2C master logic and I^2C slave logic, each generating interrupts based on their events. In multi-master systems, the software is simply partitioned into master controller and slave controller.

When the I^2C master logic is active, the slave logic also remains active, detecting the state of the bus and potentially receiving messages from itself in a single master system or from other masters in a multi-master system. No messages are lost during multi-master bus arbitration.

In a multi-master system, bus collision conflicts with other masters in the system are detected and reported to the application (BCOL Interrupt). The software can terminate, and then restart the message transmission.

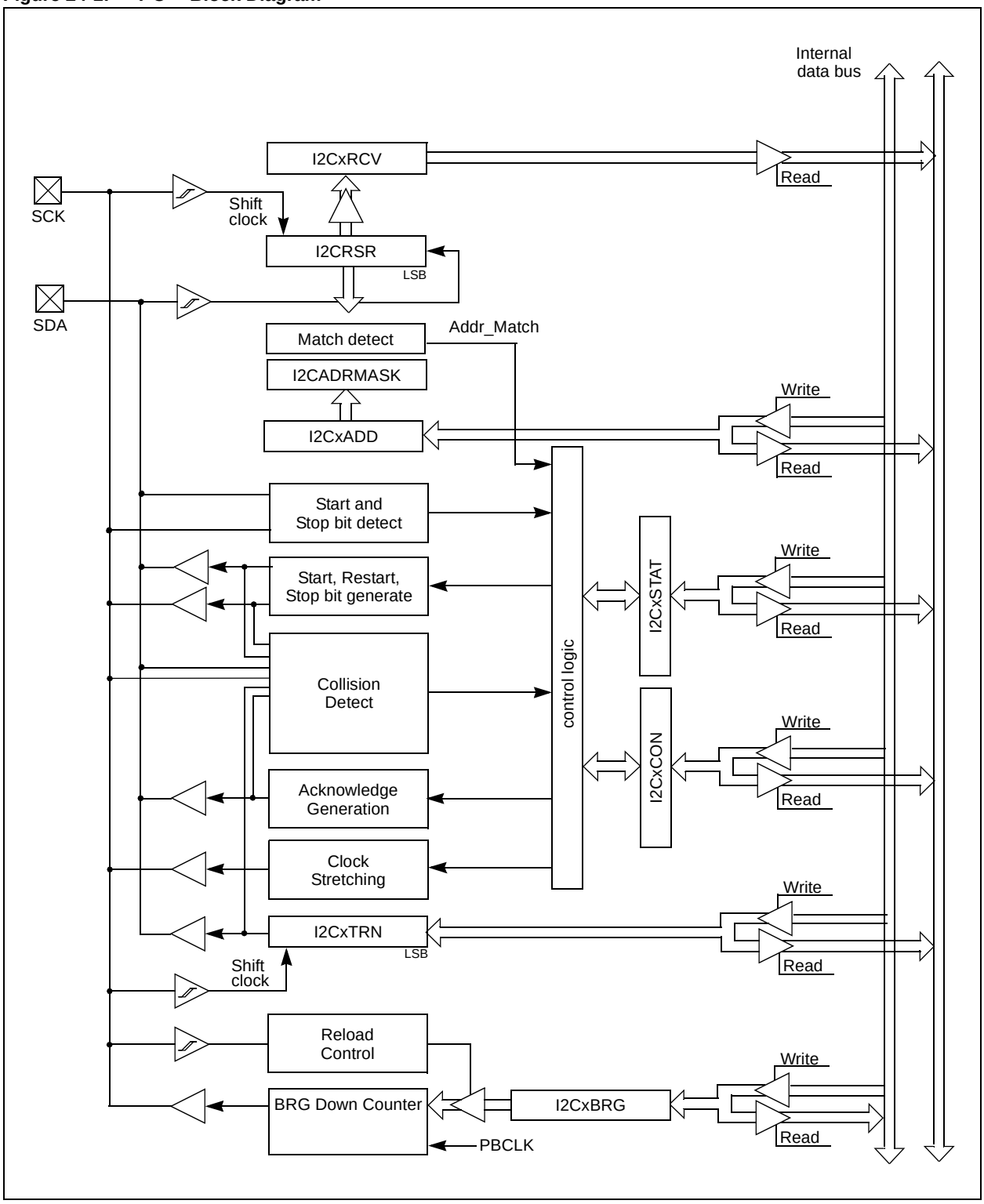
The I^2C module contains a Baud Rate Generator (BRG). The I^2C Baud Rate Generator does not consume other timer resources in the device.

Key features of the I^2C module include the following:

- Independent master and slave logic
- Multi-master support which prevents message losses in arbitration
- Detects 7-bit and 10-bit device addresses with configurable address masking in Slave mode
- Detects general call addresses as defined in the I^2C protocol
- Automatic SCLx clock stretching provides delays for the processor to respond to a slave data request
- Supports 100 kHz and 400 kHz bus specifications
- Supports Strict I^2C Reserved Address Rule

Figure 24-1 shows the I^2C module block diagram.

Figure 24-1: I²C™ Block Diagram



24.2 CONTROL AND STATUS REGISTERS

Note:	Each PIC32MX device variant may have one or more I^2C modules. An 'x' used in the names of pins, control/Status bits, and registers denotes the particular module. Refer to the specific device data sheets for more details.

The PIC32MX I^2C module consists of the following Special Function Registers (SFRs):

- I2CxCON: Control Register for the I^2C Module

  I2CxCONCLR, I2CxCONSET, I2CxCONINV: Atomic Bit Manipulation Write-only Registers for I2CxCON

- I2CxSTAT: Status Register for the I^2C Module

  I2CxSTATCLR, I2CxSTATSET, I2CxSTATINV: Atomic Bit Manipulation Write-only Registers for I2CxSTAT

- I2CxMSK: Address Mask Register (designates which bit positions in I2CxADD can be ignored, which allows for multiple address support)

  I2CxMSKCLR, I2CxMSKSET, I2CxMSKINV: Atomic Bit Manipulation Write-only Registers for I2CxMSK

- I2CxRCV: Receive Buffer Register (read-only)
- I2CxTRN: Transmit Register (read/write)
- I2CxTRNCLR, I2CxTRNSET, I2CxTRNINV: Atomic Bit Manipulation Write-only Registers for I2CxTRN
- I2CxADD: Address Register (holds the slave device address)
- I2CxADDCLR, I2CxADDSET, I2CxADDINV: Atomic Bit Manipulation Write-only Registers for I2CxADD
- I2CxBRG: Baud Rate Generator Reload Register (holds the Baud Rate Generator reload value for the I^2C module Baud Rate Generator)
- I2CxBRGCLR, I2CxBRGSET, I2CxBRGINV: Atomic Bit Manipulation Write-only Registers for I2CxBRG

Each I^2C module also has the following associated bits for interrupt control:

- I2CxMIF: Master Interrupt Flag Status Bits – in IFC0, IFC1 INT Registers
- I2CxSIF: Slave Interrupt Flag Status Bits – in IFS0, IFS1 INT Registers
- I2CxBIF: Bus Collision Interrupt Flag Status Bits – in IFS0, IFS1 INT Registers
- I2CxMIE: Master Interrupt Enable Control Bits – in IEC0, IEC1 INT Registers
- I2CxSIE: Slave Interrupt Enable Control Bits – in IEC0, IEC1 INT Registers
- I2CxBIE: Bus Collision Interrupt Enable Control Bits – in IEC0, IEC1 INT Registers
- I2CxIP<2:0>: Interrupt Priority Control Bits – in IPC6, IPC8 INT Registers
- I2CxIS<1:0>: Interrupt Sub-Priority Control Bits – in IPC6, IPC8 INT Registers

The following table summarizes all I²C-module-related registers. Corresponding registers appear after the summary, followed by a detailed description of each register.

I²C™ SFR Summary

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
I2CxCON	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	ON	FRZ	SIDL	SCLREL	STRICT	A10M	DISSLW	SMEN
	7:0	GCEN	STREN	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN
I2CxCONCLR	31:0	Writes clear selected bits of I2CxCON, reads yield undefined value							
I2CxCONSET	31:0	Writes set selected bits of I2CxCON, reads yield undefined value							
I2CxCONINV	31:0	Writes invert selected bits of I2CxCON, reads yield undefined value							
I2CxSTAT	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	ACKSTAT	TRSTAT	—	—	—	BCL	GCSTAT	ADD10
	7:0	IWCOL	I2COV	D/$\overline{A}$	P	S	R/$\overline{W}$	RBF	TBF
I2CxSTATCLR	31:0	Writes clear selected bits of I2CxSTAT, reads yield undefined value							
I2CxSTATSET	31:0	Writes set selected bits of I2CxSTAT, reads yield undefined value							
I2CxSTATINV	31:0	Writes invert selected bits of I2CxSTAT, reads yield undefined value							
I2CxADD	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	ADD<9:8>	
	7:0	ADD<7:0>							
I2CxADDCLR	31:0	Writes clear selected bits of I2CxADD, reads yield undefined value							
I2CxADDSET	31:0	Writes set selected bits of I2CxADD, reads yield undefined value							
I2CxADDINV	31:0	Writes invert selected bits of I2CxADD, reads yield undefined value							
I2CxMSK	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	MSK<9:8>	
	7:0	MSK<7:0>							
I2CxMSKCLR	31:0	Writes clear selected bits of I2CxMSK, reads yield undefined value							
I2CxMSKSET	31:0	Writes set selected bits of I2CxMSK, reads yield undefined value							
I2CxMSKINV	31:0	Writes invert selected bits of I2CxMSK, reads yield undefined value							
I2CxBRG	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	I2CxBRG<11:8>			
	7:0	I2CxBRG<7:0>							
I2CxBRGCLR	31:0	Writes clear selected bits of I2CxBRG, reads yield undefined value							
I2CxBRGSET	31:0	Writes set selected bits of I2CxBRG, reads yield undefined value							
I2CxBRGINV	31:0	Writes invert selected bits of I2CxBRG, reads yield undefined value							
I2CxTRN	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	I2CTxDATA<7:0>							
I2CxTRNCLR	31:0	Writes clear selected bits of I2CxTRN, reads yield undefined value							
I2CxTRNSET	31:0	Writes set selected bits of I2CxTRN, reads yield undefined value							
I2CxTRNINV	31:0	Writes invert selected bits of I2CxTRN, reads yield undefined value							

I²C™ SFR Summary (Continued)

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
I2CxRCV	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	I2CRXDATA<7:0>							
IFS0	31:24	I2C1MIF	I2C1SIF	I2C1BIF	U1TXIF	U1RXIF	U1EIF	SPI1RXIF	SPI1TXIF
	23:16	SPI1EIF	OC5IF	IC5IF	T5IF	INT4IF	OC4IF	IC4IF	T4IF
	15:8	INT3IF	OC3IF	IC3IF	T3IF	INT2IF	OC2IF	IC2IF	T2IF
	7:0	INT1IF	OC1IF	IC1IF	T1IF	INT0IF	CS1IF	CS0IF	CTIF
IFS1	31:24	—	—	—	—	—	—	USBIF	FCEIF
	23:16	—	—	—	—	DMA3IF	DMA2IF	DMA1IF	DMA0IF
	15:8	RTCCIF	FSCMIF	I2C2MIF	I2C2SIF	I2C2BIF	U2TXIF	U2RXIF	U2EIF
	7:0	SPI2RXIF	SPI2TXIF	SPI2EIF	CMP2IF	CMP1IF	PMPIF	AD1IF	CNIF
IEC0	31:24	I2C1MIE	I2C1SIE	I2C1BIE	U1TXIE	U1RXIE	U1EIE	SPI1RXIE	SPI1TXIE
	23:16	SPI1EIE	OC5IE	IC5IE	T5IE	INT4IE	OC4IE	IC4IE	T4IE
	15:8	INT3IE	OC3IE	IC3IE	T3IE	INT2IE	OC2IE	IC2IE	T2IE
	7:0	INT1IE	OC1IE	IC1IE	T1IE	INT0IE	CS1IE	CS0IE	CTIE
IEC1	31:24	—	—	—	—	—	—	USBIE	FCEIE
	23:16	—	—	—	—	DMA3IE	DMA2IE	DMA1IE	DMA0IE
	15:8	RTCCIE	FSCMIE	I2C2MIE	I2C2SIE	I2C2BIE	U2TXIE	U2RXIE	U2EIE
	7:0	SPI2RXIE	SPI2TXIE	SPI2EIE	CMP2IE	CMP1IE	PMPIE	AD1IE	CNIE
IPC6	31:24	—	—	—	AD1IP<2:0>			AD1IS<1:0>	
	23:16	—	—	—	CNIP<2:0>			CNIS<1:0>	
	15:8	—	—	—	I2C1IP<2:0>			I2C1IS<1:0>	
	7:0	—	—	—	U1IP<2:0>			U1IS<1:0>	
IPC8	31:24	—	—	—	RTCCIP<2:0>			RTCCIS<1:0>	
	23:16	—	—	—	FSCMIP<2:0>			FSCMIS<1:0>	
	15:8	—	—	—	I2C2IP<2:0>			I2C2IS<1:0>	
	7:0	—	—	—	U2IP<2:0>			U2IS<1:0>	

Register 24-1: I2CxCON: I²C Control Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-1	R/W-0	R/W-0	R/W-0	R/W-0
ON	FRZ	SIDL	SCLREL	STRICT	A10M	DISSLW	SMEN
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GCEN	STREN	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15 **ON:** I²C Enable bit
1 = Enables the I²C module and configures the SDA and SCL pins as serial port pins
0 = Disables I²C module; all I²C pins are controlled by PORT functions

Note: When using 1:1 PBCLK divisor, the user’s software should not read/write the peripheral’s SFRs in the SYSCLK cycle immediately following the instruction that clears the module’s ON bit.

bit 14 **FRZ:** Freeze in DEBUG Mode Control bit (Read/Write only in DEBUG mode; otherwise read as ‘0’)
1 = Freeze module operation when in DEBUG mode
0 = Do not freeze module operation when in DEBUG mode

Note: FRZ is writable in Debug Exception mode only, it is forced to ‘0’ in normal mode.

bit 13 **SIDL:** Stop in IDLE Mode bit
1 = Discontinue module operation when device enters IDLE mode
0 = Continue module operation in IDLE mode

bit 12 **SCLREL:** SCL Release Control bit
In I²C Slave mode only
Module Reset and (ON = 0) sets SCLREL = 1
If STREN = 0:
1 = Release clock
0 = Force clock low (clock stretch)

Note: Automatically cleared to ‘0’ at beginning of slave transmission.

If STREN = 1:
1 = Release clock
0 = Holds clock low (clock stretch). User may program this bit to ‘0’ to force a clock stretch at the next SCL low.

Note: Automatically cleared to ‘0’ at beginning of slave transmission; automatically cleared to ‘0’ at end of slave reception.

Register 24-1: I2CxCON: I²C Control Register (Continued)

bit 11 **STRICT:** Strict I²C Reserved Address Rule Enable bit
1 = Strict reserved addressing is enforced. Device doesn't respond to reserved address space or generate addresses in reserved address space.
0 = Strict I²C Reserved Address Rule not enabled

bit 10 **A10M:** 10-bit Slave Address Flag bit
1 = I2CxADD is a 10-bit slave address
0 = I2CADD is a 7-bit slave address

bit 9 **DISSLW:** Slew Rate Control Disable bit
1 = Slew rate control disabled for Standard Speed mode (100 kHz); also disabled for 1 MHz mode
0 = Slew rate control enabled for High Speed mode (400 kHz)

bit 8 **SMEN:** SMBus Input Levels Disable bit
1 = Enable input logic so that thresholds are compliant with SMBus specification
0 = Disable SMBus specific inputs

bit 7 **GCEN:** General Call Enable bit
In I²C Slave mode only
1 = Enable interrupt when a general call address is received in I2CSR. Module is enabled for reception.
0 = General call address disabled.

bit 6 **STREN:** SCL Clock Stretch Enable bit
In I²C Slave mode only; used in conjunction with SCLREL bit.
1 = Enable clock stretching
0 = Disable clock stretching

bit 5 **ACKDT:** Acknowledge Data bit
In I²C Master mode only; applicable during master receive. Value that will be transmitted when the user initiates an Acknowledge sequence at the end of a receive.
1 = A NACK is sent
0 = ACK is sent

bit 4 **ACKEN:** Acknowledge Sequence Enable bit
In I²C Master mode only; applicable during master receive
1 = Initiate Acknowledge sequence on SDA and SCL pins, and transmit ACKDT data bit; cleared by module
0 = Acknowledge sequence idle

bit 3 **RCEN:** Receive Enable bit
In I²C Master mode only
1 = Enables Receive mode for I²C, automatically cleared by module at end of 8-bit receive data byte
0 = Receive sequence not in progress

bit 2 **PEN:** Stop Condition Enable bit
In I²C Master mode only
1 = Initiate Stop condition on SDA and SCL pins; cleared by module
0 = Stop condition idle

bit 1 **RSEN:** Restart Condition Enable bit
In I²C Master mode only
1 = Initiate Restart condition on SDA and SCL pins; cleared by module
0 = Restart condition idle

bit 0 **SEN:** Start Condition Enable bit
In I²C Master mode only
1 = Initiate Start condition on SDA and SCL pins; cleared by module
0 = Start condition idle

Register 24-2: I2CxCONCLR: I²C 'x' Control Clear Register

Write clears selected bits in I2CxCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in I2CxCON**

A write of '1' in one or more bit positions clears the corresponding bit(s) in I2CxCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxCONCLR = 0x00008001` will clear bits 15 and 0 in I2CxCON register.

Register 24-3: I2CxCONSET: I²C 'x' Control Set Register

Write sets selected bits in I2CxCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in I2CxCON**

A write of '1' in one or more bit positions sets the corresponding bit(s) in I2CxCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxCONSET = 0x00008001` will set bits 15 and 0 in I2CxCON register.

Register 24-4: I2CxCONINV: I²C 'x' Control Invert Register

Write inverts selected bits in I2CxCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in I2CxCON**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in I2CxCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxCONINV = 0x00008001` will invert bits 15 and 0 in I2CxCON register.

Register 24-5: I2CxSTAT: I^{2}C Status Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R-0	R-0	r-x	r-x	r-x	R/W-0	R-0	R-0
ACKSTAT	TRSTAT	—	—	—	BCL	GCSTAT	ADD10
bit 15							bit 8

R/W-0	R/W-0	R-0	R/W-0	R/W-0	R-0	R-0	R-0
IWCOL	I2COV	D/$\overline{A}$	P	S	R/$\overline{W}$	RBF	TBF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15 **ACKSTAT:** Acknowledge Status bit
In both I^{2}C Master and Slave modes; applicable to both transmit and receive.
1 = Acknowledge was not received
0 = Acknowledge was received

bit 14 **TRSTAT:** Transmit Status bit
In I^{2}C Master mode only; applicable to Master Transmit mode.
1 = Master transmit is in progress (8 bits + $\overline{ACK}$)
0 = Master transmit is not in progress

bit 13-11 **Reserved:** Write ‘0’; ignore read

bit 10 **BCL:** Master Bus Collision Detect bit
Cleared when the I^{2}C module is disabled (ON = 0).
1 = A bus collision has been detected during a master operation
0 = No collision has been detected

bit 9 **GCSTAT:** General Call Status bit
Cleared after Stop detection.
1 = General call address was received
0 = General call address was not received

bit 8 **ADD10:** 10-bit Address Status bit
Cleared after Stop detection.
1 = 10-bit address was matched
0 = 10-bit address was not matched

bit 7 **IWCOL:** Write Collision Detect bit
1 = An attempt to write the I2CxTRN register collided because the I^{2}C module is busy.
Must be cleared in software.
0 = No collision

Register 24-5: I2CxSTAT: I²C Status Register (Continued)

bit 6 **I2COV:** I²C Receive Overflow Status bit
1 = A byte is received while the I2CxRCV register is still holding the previous byte. I2COV is a "don't care" in Transmit mode. Must be cleared in software.
0 = No overflow

bit 5 **D/$\overline{A}$:** Data/Address bit
Valid only for Slave mode operation.
1 = Indicates that the last byte received or transmitted was data
0 = Indicates that the last byte received or transmitted was address

bit 4 **P:** Stop bit
Updated when Start, Reset or Stop detected; cleared when the I²C module is disabled (ON = 0).
1 = Indicates that a Stop bit has been detected last
0 = Stop bit was not detected last

bit 3 **S:** Start bit
Updated when Start, Reset or Stop detected; cleared when the I²C module is disabled (ON = 0).
1 = Indicates that a start (or restart) bit has been detected last
0 = Start bit was not detected last

bit 2 **R/$\overline{W}$:** Read/Write Information bit
Valid only for Slave mode operation.
1 = Read – indicates data transfer is output from slave
0 = Write – indicates data transfer is input to slave

bit 1 **RBF:** Receive Buffer Full Status bit
1 = Receive complete; I2CxRCV is full
0 = Receive not complete; I2CxRCV is empty

bit 0 **TBF:** Transmit Buffer Full Status bit
1 = Transmit in progress; I2CxTRN is full (8-bits of data)
0 = Transmit complete; I2CxTRN is empty

Register 24-6: I2CxSTATCLR: I²C 'x' Status Clear Register

Write clears selected bits in I2CxSTAT, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in I2CxSTAT**

A write of '1' in one or more bit positions clears the corresponding bit(s) in I2CxSTAT register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxSTATCLR = 0x00008001` will clear bits 15 and 0 in I2CxSTAT register.

Register 24-7: I2CxSTATSET: I²C 'x' Status Set Register

Write sets selected bits in I2CxSTAT, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in I2CxSTAT**

A write of '1' in one or more bit positions sets the corresponding bit(s) in I2CxSTAT register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxSTATSET = 0x00008001` will set bits 15 and 0 in I2CxSTAT register.

Register 24-8: I2CxSTATINV: I²C 'x' Status Invert Register

Write inverts selected bits in I2CxSTAT, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in I2CxSTAT**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in I2CxSTAT register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxSTATINV = 0x00008001` will invert bits 15 and 0 in I2CxSTAT register.

Register 24-9: I2CxADD: I^{2}C Slave Address Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	ADD<9:8>	
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADD<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-10 **Reserved:** Write ‘0’; ignore read

bit 9-0 **ADD<9:0>:** I^{2}C Slave Device Address bits
Either Master or Slave mode

Register 24-10: I2CxADDCLR: I²C 'x' Slave Address Clear Register

Write clears selected bits in I2CxADD, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in I2CxADD**

A write of '1' in one or more bit positions clears the corresponding bit(s) in I2CxADD register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxADDCLR = 0x00008001` will clear bits 15 and 0 in I2CxADD register.

Register 24-11: I2CxADDSET: I²C 'x' Slave Address Set Register

Write sets selected bits in I2CxADD, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in I2CxADD**

A write of '1' in one or more bit positions sets the corresponding bit(s) in I2CxADD register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxADDSET = 0x00008001` will set bits 15 and 0 in I2CxADD register.

Register 24-12: I2CxADDINV: I²C 'x' Slave Address Invert Register

Write inverts selected bits in I2CxADD, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in I2CxADD**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in I2CxADD register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxADDINV = 0x00008001` will invert bits 15 and 0 in I2CxADD register.

Register 24-13: I2CxMSK: I²C Address Mask Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	MSK<9:8>	
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
MSK<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-10 **Reserved:** Write ‘0’; ignore read

bit 9-0 **MSK<9:0>:** I²C Address Mask bits

1 = Forces a “don’t care” in the particular bit position on the incoming address match sequence.
0 = Address bit position must match the incoming I²C address match sequence.

Note: MSK<9:8> and MSK<0> are only used in I²C 10-bit mode.

Register 24-14: I2CxMSKCLR: I²C 'x' Address Mask Clear Register

Write clears selected bits in I2CxMSK, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in I2CxMSK**

A write of '1' in one or more bit positions clears the corresponding bit(s) in I2CxMSK register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxMSKCLR = 0x00008001` will clear bits 15 and 0 in I2CxMSK register.

Register 24-15: I2CxMSKSET: I²C 'x' Address Mask Set Register

Write sets selected bits in I2CxMSK, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in I2CxMSK**

A write of '1' in one or more bit positions sets the corresponding bit(s) in I2CxMSK register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxMSKSET = 0x00008001` will set bits 15 and 0 in I2CxMSK register.

Register 24-16: I2CxMSKINV: I²C 'x' Address Mask Invert Register

Write inverts selected bits in I2CxMSK, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in I2CxMSK**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in I2CxMSK register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxMSKINV = 0x00008001` will invert bits 15 and 0 in I2CxMSK register.

Register 24-17: I2CxBRG: I^{2}C Baud Rate Generator Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	I2CxBRG<11:8>			
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
I2CxBRG<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-12 **Reserved:** Write ‘0’; ignore read

bit 11-0 **I2CxBRG<11:0>:** I^{2}C Baud Rate Generator Value bits
A divider function of the Peripheral Clock.

Register 24-18: I2CxBRGCLR: I²C 'x' Baud Rate Generator Clear Register

Write clears selected bits in I2CxBRG, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in I2CxBRG**

A write of '1' in one or more bit positions clears the corresponding bit(s) in I2CxBRG register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxBRGCLR = 0x00008001` will clear bits 15 and 0 in I2CxBRG register.

Register 24-19: I2CxBRGSET: I²C 'x' Baud Rate Generator Set Register

Write sets selected bits in I2CxBRG, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in I2CxBRG**

A write of '1' in one or more bit positions sets the corresponding bit(s) in I2CxBRG register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxBRGSET = 0x00008001` will set bits 15 and 0 in I2CxBRG register.

Register 24-20: I2CxBRGINV: I²C 'x' Baud Rate Generator Invert Register

Write inverts selected bits in I2CxBRG, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in I2CxBRG**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in I2CxBRG register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxBRGINV = 0x00008001` will invert bits 15 and 0 in I2CxBRG register.

Register 24-21: I2CxTRN: I^{2}C Transmit Data Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
I2CTXDATA<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7-0 **I2CTXDATA<7:0>:** I^{2}C Transmit Data Buffer bits

Register 24-22: I2CxTRNCLR: I²C 'x' Transmit Data Clear Register

Write clears selected bits in I2CxTRN, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in I2CxTRN**

A write of '1' in one or more bit positions clears the corresponding bit(s) in I2CxTRN register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxTRNCLR = 0x00008001` will clear bits 15 and 0 in I2CxTRN register.

Register 24-23: I2CxTRNSET: I²C 'x' Transmit Data Set Register

Write sets selected bits in I2CxTRN, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in I2CxTRN**

A write of '1' in one or more bit positions sets the corresponding bit(s) in I2CxTRN register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxTRNSET = 0x00008001` will set bits 15 and 0 in I2CxTRN register.

Register 24-24: I2CxTRNINV: I²C 'x' Transmit Data Invert Register

Write inverts selected bits in I2CxTRN, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in I2CxTRN**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in I2CxTRN register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxTRNINV = 0x00008001` will invert bits 15 and 0 in I2CxTRN register.

Register 24-25: I2CxRCV: I²C Receive Data Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
I2CRXDATA<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7-0 **I2CRXDATA<7:0>:** I²C Receive Data Buffer bits

Register 24-26: IFS0: Interrupt Flag Status Register 0(1)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
I2C1MIF	I2C1SIF	I2C1BIF	U1TXIF	U1RXIF	U1EIF	SPI1RXIF	SPI1TXIF
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SPI1EIF	OC5IF	IC5IF	T5IF	INT4IF	OC4IF	IC4IF	T4IF
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INT3IF	OC3IF	IC3IF	T3IF	INT2IF	OC2IF	IC2IF	T2IF
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	RW-0	RW-0	RW-0
INT1IF	OC1IF	IC1IF	T1IF	INT0IF	CS1IF	CS0IF	CTIF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31 **I2C1MIF**: I^2C Master Interrupt Flag Status bit
1 = Interrupt request has occurred
0 = No interrupt request has a occurred

bit 30 **I2C1SIF**: I^2C Slave Interrupt Flag Status bit
1 = Interrupt request has occurred
0 = No interrupt request has a occurred

bit 29 **I2C1BIF**: I^2C Bus Collision Interrupt Flag Status bit
1 = Interrupt request has occurred
0 = No interrupt request has a occurred

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the I^2C™.

Register 24-27: IEC0: Interrupt Enable Control Register 0(1)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
I2C1MIE	I2C1SIE	I2C1BIE	U1TXIE	U1RXIE	U1EIE	SPI1RXIE	SPI1TXIE
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SPI1EIE	OC5IE	IC5IE	T5IE	INT4IE	OC4IE	IC4IE	T4IE
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INT3IE	OC3IE	IC3IE	T3IE	INT2IE	OC2IE	IC2IE	T2IE
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INT1IE	OC1IE	IC1IE	T1IE	INT0IE	CS1IE	CS0IE	CTIE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31 **I2C1MIE**: I^{2}C Master Interrupt Enable Control bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 30 **I2C1SIE**: I^{2}C Slave Interrupt Enable Control bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 29 **I2C1BIE**: I^{2}C Bus Collision Interrupt Enable Control bit
1 = Interrupt is enabled
0 = Interrupt is disabled

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the I^{2}C™.

Register 24-28: IPC6: Interrupt Priority Control Register 6(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	AD1IP<2:0>			AD1IS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	CNIP<2:0>			CNIS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	I2C1IP<2:0>			I2C1IS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	U1IP<2:0>			U1IS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 12-10 **I2C1IP<2:0>:** I^2C 1 Interrupt Priority bits
- 111 = Interrupt Priority is 7
- 110 = Interrupt Priority is 6
- 101 = Interrupt Priority is 5
- 100 = Interrupt Priority is 4
- 011 = Interrupt Priority is 3
- 010 = Interrupt Priority is 2
- 001 = Interrupt Priority is 1
- 000 = Interrupt is disabled.

bit 9-8 **I2C1IS<1:0>:** I^2C 1 Subpriority bits
- 11 = Interrupt Subpriority is 3
- 10 = Interrupt Subpriority is 2
- 01 = Interrupt Subpriority is 1
- 00 = Interrupt Subpriority is 0

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the I^2C™.

Register 24-29: IPC8: Interrupt Priority Control Register 8(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	RTCCIP<2:0>			RTCCIS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	FCSMIP<2:0>			FCSMIS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	I2C2IP<2:0>			I2C2IS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	U2IP<2:0>			U2IS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 12-10 **I2C2IP<2:0>:** I^2C 2 Interrupt Priority bits
- 111 = Interrupt Priority is 7
- 110 = Interrupt Priority is 6
- 101 = Interrupt Priority is 5
- 100 = Interrupt Priority is 4
- 011 = Interrupt Priority is 3
- 010 = Interrupt Priority is 2
- 001 = Interrupt Priority is 1
- 000 = Interrupt is disabled

bit 9-8 **I2C2IS<1:0>:** I^2C 2 Subpriority bits
- 11 = Interrupt Subpriority is 3
- 10 = Interrupt Subpriority is 2
- 01 = Interrupt Subpriority is 1
- 00 = Interrupt Subpriority is 0

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the I^2C™.

24.3 I²C™ BUS CHARACTERISTICS

The I²C bus is a two-wire serial interface. Figure 24-2 shows a schematic of an I²C connection between a PIC32MX device and a 24LC256 I²C serial EEPROM, which is a typical example for any I²C interface.

The interface employs a comprehensive protocol to ensure reliable transmission and reception of data. When communicating, one device is the “master” which initiates transfer on the bus and generates the clock signals to permit that transfer, while the other device(s) acts as the “slave” responding to the transfer. The clock line, SCLx, is output from the master and input to the slave, although occasionally the slave drives the SCLx line. The data line, SDAx, may be output and input from both the master and slave.

Because the SDAx and SCLx lines are bidirectional, the output stages of the devices driving the SDAx and SCLx lines must have an open drain in order to perform the wired AND function of the bus. External pull-up resistors are used to ensure a high level when no device is pulling the line down.

In the I²C interface protocol, each device has an address. When a master wishes to initiate a data transfer, it first transmits the address of the device that it wishes to “talk” to. All devices “listen” to see if this is their address. Within this address, bit 0 specifies if the master wishes to read from or write to the slave device. The master and slave are always in opposite modes of operation (transmitter/receiver) during a data transfer. That is, they can be thought of as operating in either of the following two relations:

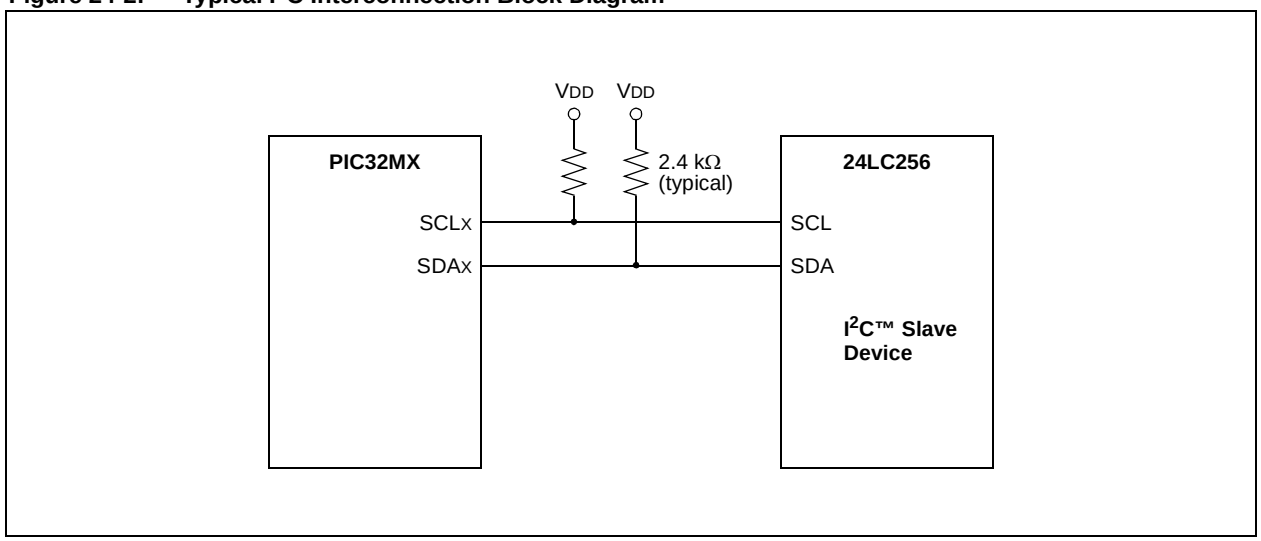
- Master-Transmitter and Slave-Receiver
- Slave-Transmitter and Master-Receiver

In both cases, the master originates the SCLx clock signal.

The following modes and features specified in the V2.1 I²C specifications are not supported:

- HS mode and switching between F/S modes and HS mode
- Start Byte
- CBUS Compatibility
- 2nd byte of General Call Address

Figure 24-2: Typical I²C Interconnection Block Diagram



24.3.1 Bus Protocol

The following I^2C bus protocol has been defined:

- Data transfer may be initiated only when the bus is not busy.
- During data transfer, the data line must remain stable whenever the SCLx clock line is high. Changes in the data line while the SCLx clock line is high will be interpreted as a Start or Stop condition.

Accordingly, the following bus conditions have been defined and are shown in Figure 24-3.

24.3.1.1 Start Data Transfer (S)

After a bus Idle state, a high-to-low transition of the SDAx line while the clock (SCLx) is high determines a Start condition. All data transfers must be preceded by a Start condition.

24.3.1.2 Stop Data Transfer (P)

A low-to-high transition of the SDAx line while the clock (SCLx) is high determines a Stop condition. All data transfers must end with a Stop condition.

24.3.1.3 Repeated Start (R)

After a wait state, a high-to-low transition of the SDAx line while the clock (SCLx) is high determines a Repeated Start condition. Repeated Starts allow a master to change bus direction of addressed slave device without relinquishing control of the bus.

24.3.1.4 Data Valid (D)

The state of the SDAx line represents valid data when, after a Start condition, the SDAx line is stable for the duration of the high period of the clock signal. There is one bit of data per SCLx clock.

24.3.1.5 Acknowledge (A) or Not Acknowledge (N)

All data byte transmissions must be Acknowledged ($\overline{ACK}$) or Not Acknowledged (NACK) by the receiver. The receiver will pull the SDAx line low for an $\overline{ACK}$ or release the SDAx line for a NACK. The Acknowledge is a one-bit period using one SCLx clock.

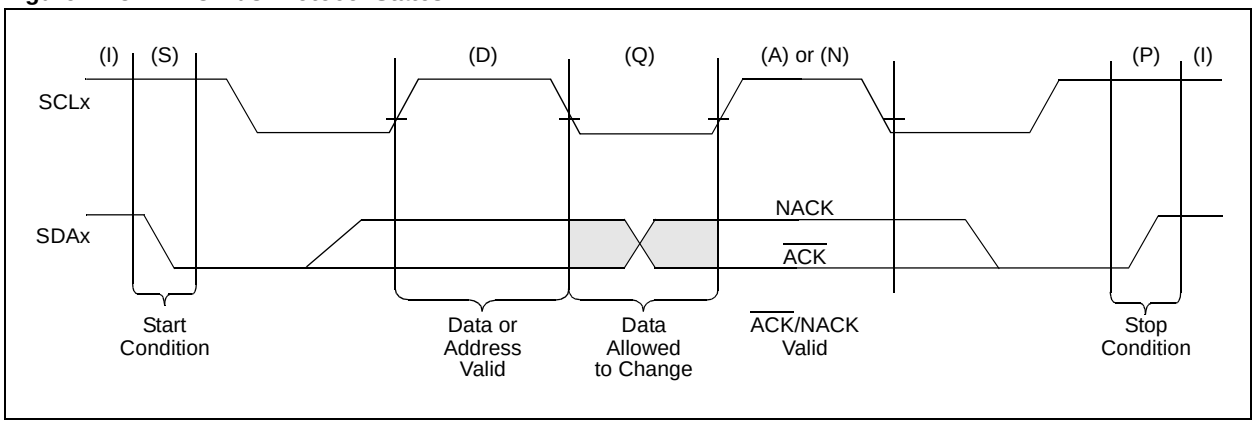
24.3.1.6 Wait/Data Invalid (Q)

The data on the line must be changed during the low period of the clock signal. Devices may also stretch the clock low time by asserting a low on the SCLx line, causing a wait on the bus.

24.3.1.7 Bus Idle (I)

Both data and clock lines remain high at those times after a Stop condition and before a Start condition.

Figure 24-3: I^2C Bus Protocol States

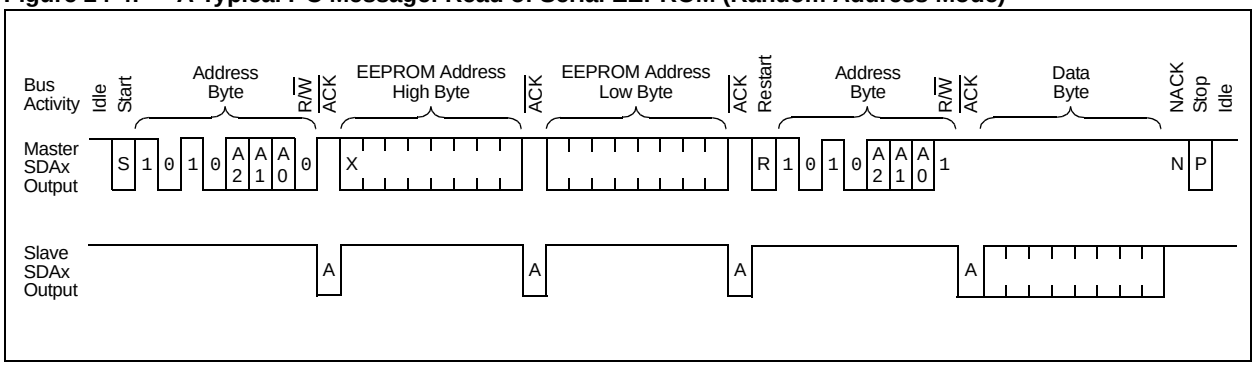


24.3.2 Message Protocol

A typical I^{2}C message is shown in Figure 24-4. In this example, the message will read a specified byte from a 24LC256 I^{2}C serial EEPROM. The PIC32MX device will act as the master and the 24LC256 device will act as the slave.

Figure 24-4 indicates the data as driven by the master device and the data as driven by the slave device, taking into account that the combined SDAx line is a wired AND of the master and slave data. The master device controls and sequences the protocol. The slave device will only drive the bus at specifically determined times.

Figure 24-4: A Typical I^{2}C Message: Read of Serial EEPROM (Random Address Mode)



24.3.2.1 Start Message

Each message is initiated with a “Start” condition and terminated with a “Stop” condition. The number of data bytes transferred between the Start and Stop conditions is determined by the master device. As defined by the system protocol, the bytes of the message may have special meaning, such as “device address byte” or “data byte”.

24.3.2.2 Address Slave

In Figure 24-4, the first byte is the device address byte, that must be the first part of any I^{2}C message. It contains a device address and a R/$\overline{W}$ bit. For additional information on address byte formats, refer to **Appendix A** (check the Microchip web site, www.microchip.com, for availability). Note that R/$\overline{W}$ = 0 for this first address byte, indicating that the master will be a transmitter and the slave will be a receiver.

24.3.2.3 Slave Acknowledge

The receiving device is obliged to generate an Acknowledge signal, $\overline{ACK}$, after the reception of each byte. The master device must generate an extra SCLx clock which is associated with this Acknowledge bit.

24.3.2.4 Master Transmit

The next two bytes, sent by the master to the slave, are data bytes containing the location of the requested EEPROM data byte. The slave must Acknowledge each of the data bytes.

24.3.2.5 Repeated Start

At this point, the slave EEPROM has the address information necessary to return the requested data byte to the master. However, the R/$\overline{W}$ bit from the first device address byte specified master transmission and slave reception. The bus must be turned in the other direction for the slave to send data to the master.

To perform this function without ending the message, the master sends a “Repeated Start”. The Repeated Start is followed with a device address byte containing the same device address as before and with the R/$\overline{W}$ = 1 to indicate slave transmission and master reception.

24.3.2.6 Slave Reply

Now the slave transmits the data byte by driving the SDAx line, while the master continues to originate clocks but releases its SDAx drive.

24.3.2.7 Master Acknowledge

During reads, a master must terminate data requests to the slave by Not Acknowledging (generating a “NACK”) on the last byte of the message. Data is acked for each byte, except for the last byte.

24.3.2.8 Stop Message

The master sends a Stop to terminate the message and return the bus to an Idle state.

24.4 ENABLING I²C™ OPERATION

The module is enabled by setting the ON (I2CxCON<15>) bit.

The I²C module fully implements all master and slave functions. When the module is enabled, the master and slave functions are active simultaneously and will respond according to the software or bus events.

When initially enabled, the module will release the SDAx and SCLx pins, putting the bus into the Idle state. The master functions will remain in the Idle state unless software sets a control bit to initiate a master event. The slave functions will begin to monitor the bus. If the slave logic detects a Start event and a valid address on the bus, the slave logic will begin a slave transaction.

24.4.1 Enabling I²C I/O

Two pins are used for bus operation. These are the SCLx pin, which is the clock, and the SDAx pin, which is the data. When the module is enabled, assuming no other module with higher priority has control, the module will assume control of the SDAx and SCLx pins. The module software need not be concerned with the state of the port I/O of the pins, the module overrides, the port state and direction. At initialization, the pins are tri-state (released).

24.4.2 I²C Interrupts

The I²C module generates three interrupt signals: slave interrupt (I2CxSIF), master interrupt (I2CxMIF) and bus collision interrupt (I2CxBIF). The slave interrupt, master interrupt and bus collision interrupt signals are pulsed high for at least one PBCLK on the falling edge of the 9th clock pulse of the SCL clock. These interrupts will set the corresponding interrupt flag bit and will interrupt the CPU if the corresponding interrupt enable bit is set and the corresponding interrupt priority is high enough.

Master mode operations that generate a master interrupt (I2CxMIF) are as follows:

1. Start Condition
   - 1 BRG (Baud Rate Generator) time after falling edge of SDA
2. Repeated Start Sequence
   - 1 BRG time after falling edge of SDA
3. Stop Condition
   - 1 BRG time after the rising edge of SDA
4. Data transfer byte received
   - 8th falling edge of SCL
   (After receiving eight bits of data from slave)
5. During SEND $\overline{ACK}$ sequence
   - 9th falling edge of SCL
   (After sending $\overline{ACK}$ or NACK to slave)
6. Data transfer byte transmitted
   - 9th falling edge of SCL
   (Regardless of receiving $\overline{ACK}$ from slave)
7. During a slave-detected Stop
   - When slave sets P bit

Slave mode operations that generate a slave interrupt (I2CxSIF) are as follows:

1. Detection of a valid device address (including general call)
   - 9th falling edge of SCL
   (After sending ACK to master. Address must match unless STRICT = 1 or GCEN = 1)
2. Reception of data
   - 9th falling edge of SCL
   (After sending the ACK to master)
3. Request to transmit data
   - 9th falling edge of SCL
   (Regardless of receiving ACK from master)

Bus Collision events that generate an interrupt (I2CxBIF) are as follows:

1. During Start sequence
   - SDA sampled before start condition
2. During Start sequence
   - SCL = 0 before SDA = 0
3. During Start sequence
   - SDA = 0 before BRG time out
4. During a Repeated Start sequence
   - If SDA is sampled 0 when SCL goes high
5. During a Repeated Start sequence
   - If SCL goes low before SDA goes low
6. During a Stop sequence
   - If SDA is sampled low after allowing it to float
7. During a Stop sequence
   - If SCL goes low before SDA goes high

24.4.3 I²C Transmit and Receive Registers

I2CxTRN is the register to which transmit data is written. This register is used when the module operates as a master transmitting data to the slave, or as a slave sending reply data to the master. As the message progresses, the I2CxTRN register shifts out the individual bits. As a result, the I2CxTRN may not be written to unless the bus is Idle.

Data being received by either the master or the slave is shifted into a non-accessible shift register, I2CxRSR. When a complete byte is received, the byte transfers to the I2CxRCV register. In receive operations, the I2CxRSR and I2CxRCV create a double-buffered receiver. This allows reception of the next byte to begin before the current byte of received data is read.

If the module receives another complete byte before the software reads the previous byte from the I2CxRCV register, a receiver overflow occurs and sets the I2COV bit (I2CxSTAT<6>). The byte in the I2CxRSR is lost.

The I2CxADD register holds the slave device address. In 10-Bit Addressing mode, all bits are relevant. In 7-Bit Addressing mode, only I2CxADD<6:0> are relevant. The A10M bit (I2CxCON<10>) specifies the expected mode of the slave address. By using the I2CxMSK register with the I2CxADD register in either Slave Addressing mode, one or more bit positions can be removed from exact address matching, allowing the module in Slave mode to respond to multiple addresses.

24.4.4 I²C Baud Rate Generator

The Baud Rate Generator used for I²C Master mode operation is used to set the SCL clock frequency for 100 kHz, 400 kHz and 1 MHz. The Baud Rate Generator re-load value is contained in the I2CxBRG register. The Baud Rate Generator will automatically begin counting on a write to the I2CxTRN. Once the given operation is complete (i.e., transmission of the last data bit is followed by ACK) the internal clock will automatically stop counting and the SCL pin will remain in its last state.

24.4.5 Baud Rate Generator in I²C Master Mode

In I²C Master mode, the reload value for the BRG is located in the I2CxBRG register. When the BRG is loaded with this value, the BRG counts down to zero and stops until another reload has taken place. In I²C Master mode, the BRG is not reloaded automatically. If Clock Arbitration is taking place, for instance, the BRG will be reloaded when the SCL pin is sampled high (see Figure 24-6). Table 24-1 shows device frequency vs. I2CxBRG setting for standard baud rates.

Note:	I2CxBRG values of 0x0 and 0x1 are expressly forbidden. The user should never program the I2CxBRG with a value of 0x0 or 0x1, as indeterminate results may occur.

To compute the Baud Rate Generator reload value, use the following equation:

Equation 24-1: Baud Rate Generator Reload Value Calculation

$$FSCK = (PBCLK) / ((I2CxBRG+2) * 2)$$

$$I2CBRG = (PBCLK / (2 * FSCK)) - 2$$

Table 24-1: I²C Clock Rate w/BRG

PBCLK	I2CxBRG	Approx. Fsck (2 roll-overs of BRG)
50 MHz	0x03C	400 kHz
50 MHz	0x0F8	100 kHz
40 MHz	0x030	400 kHz
40 MHz	0x0C6	100 kHz
30 MHz	0x023	400 kHz
30 MHz	0x094	100 kHz
20 MHz	0x017	400 kHz
20 MHz	0x062	100 kHz
10 MHz	0x00A	400 kHz
10 MHz	0x030	100 kHz

Figure 24-5: Baud Rate Generator Block Diagram

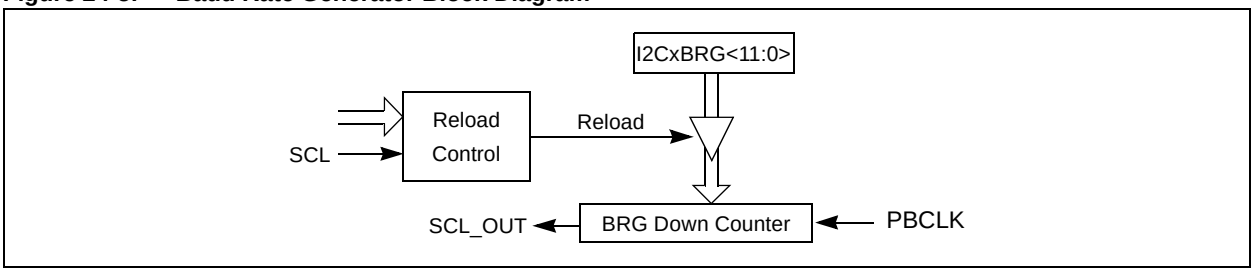
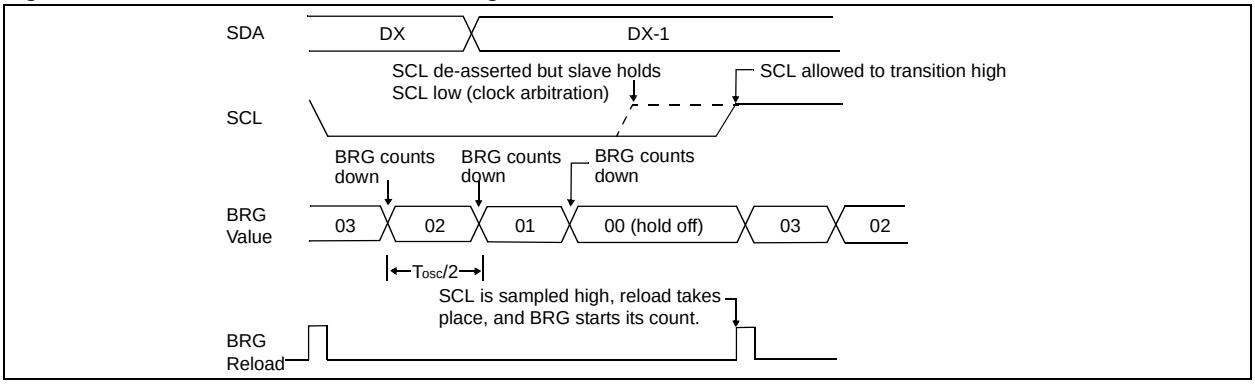


Figure 24-6: Baud Rate Generator Timing With Clock Arbitration



24.5 COMMUNICATING AS A MASTER IN A SINGLE MASTER ENVIRONMENT

The I^2C module's typical operation in a system is using the I^2C to communicate with an I^2C peripheral, such as an I^2C serial memory. In an I^2C system, the master controls the sequence of all data communication on the bus. In this example, the PIC32MX and its I^2C module have the role of the single master in the system. As the single master, it is responsible for generating the SCLx clock and controlling the message protocol.

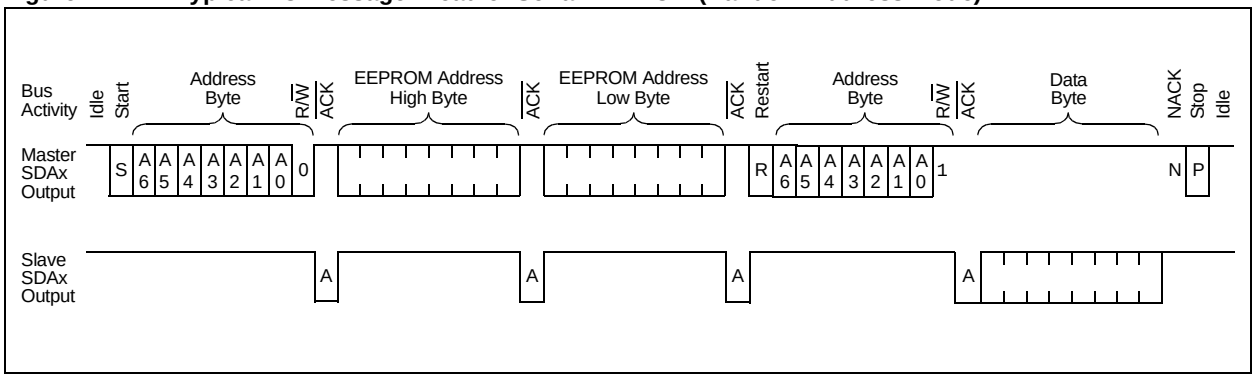
In the I^2C module, the module controls individual portions of the I^2C message protocol; however, sequencing of the components of the protocol to construct a complete message is a software task.

For example, a typical operation in a single master environment may be to read a byte from an I^2C serial EEPROM. This example message is depicted in Figure 24-7.

To accomplish this message, the software will sequence through the following steps.

1. Turn on the module by setting ON bit (I2CxCON<15>) to '1'.
1. Assert a Start condition on SDAx and SCLx.
2. Send the I^2C device address byte to the slave with a write indication.
3. Wait for and verify an Acknowledge from the slave.
4. Send the serial memory address high byte to the slave.
5. Wait for and verify an Acknowledge from the slave.
6. Send the serial memory address low byte to the slave.
7. Wait for and verify an Acknowledge from the slave.
8. Assert a Repeated Start condition on SDAx and SCLx.
9. Send the device address byte to the slave with a read indication.
10. Wait for and verify an Acknowledge from the slave.
11. Enable master reception to receive serial memory data.
12. Generate an $\overline{ACK}$ or NACK condition at the end of a received byte of data.
13. Generate a Stop condition on SDAx and SCLx.

Figure 24-7: Typical I^2C Message: Read of Serial EEPROM (Random Address Mode)



The I^2C module supports Master mode communication with the inclusion of Start and Stop generators, data byte transmission, data byte reception, an Acknowledge generator and a Baud Rate Generator. Generally, the software will write to a control register to start a particular step, then wait for an interrupt or poll status to wait for completion.

Subsequent sections detail each of these operations.

Note:	The I^2C module does not allow queueing of events. For instance, the software is not allowed to initiate a Start condition and then immediately write the I2CxTRN register to initiate transmission before the Start condition is complete. In this case, the I2CxTRN will not be written to and the IWCOL bit will be set, indicating that this write to the I2CxTRN did not occur.

24.5.1 Generating Start Bus Event

To initiate a Start event, the software sets the Start Enable bit, SEN (I2CxCON<0>). Prior to setting the Start bit, the software can check the P Status bit (I2CxSTAT<4>) to ensure that the bus is in an Idle state.

Figure 24-8 shows the timing of the Start condition.

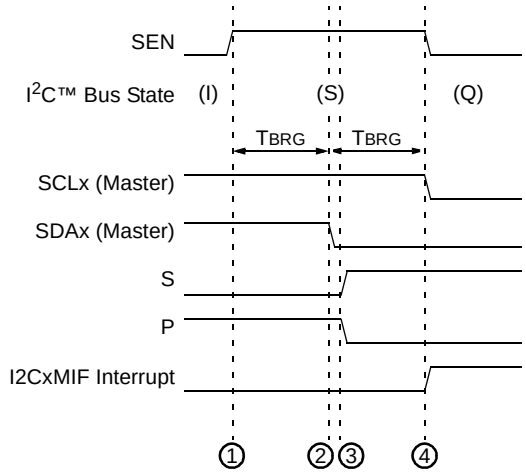
- Slave logic detects the Start condition, sets the S bit (I2CxSTAT<3>) and clears the P bit (I2CxSTAT<4>).
- The SEN bit is automatically cleared at completion of the Start condition.
- I2CxMIF interrupt is generated at completion of the Start condition.
- After the Start condition, the SDAx line and SCLx line are left low (Q state).

24.5.1.1 IWCOL Status Flag

If the software writes the I2CxTRN when a Start sequence is in progress, then IWCOL is set and the contents of the transmit buffer are unchanged (the write doesn't occur).

Note:	Because queueing of events is not allowed, writing to the lower 5 bits of I2CxCON is disabled until the Start condition is complete.

Figure 24-8: Master Start Timing Diagram



1. Writing SEN = 1 initiates a master Start event. Baud Rate Generator starts.
2. Baud Rate Generator times out. Master module drives SDAx low. Baud Rate Generator restarts.
3. Slave module detects Start and sets S = 1 and P = 0.
4. Baud Rate Generator times out. Master module drives SCLx low, generates interrupt and clears SEN.

24.5.2 Sending Data to a Slave Device

Figure 24-9 shows the timing diagram of master to slave transmission. Transmission of a data byte, a 7-bit device address byte or the second byte of a 10-bit address is accomplished by simply writing the appropriate value to the I2CxTRN register. Loading this register will start the following process:

- The software loads the I2CxTRN with the data byte to transmit.
- Writing I2CxTRN sets the buffer full flag bit, TBF (I2CxSTAT<0>).
- The data byte is shifted out the SDAx pin until all 8 bits are transmitted. Each bit of address/data will be shifted out onto the SDAx pin after the falling edge of SCLx.
- On the ninth SCLx clock, the module shifts in the $\overline{ACK}$ bit from the slave device and writes its value into the ACKSTAT bit (I2CxSTAT<15>).
- The module generates the I2CxMIF interrupt at the end of the ninth SCLx clock cycle.

Note that the module does not generate or validate the data bytes. The contents and usage of the bytes are dependent on the state of the message protocol maintained by the software.

24.5.2.1 Sending a 7-Bit Address to the Slave

Sending a 7-bit device address involves sending one byte to the slave. A 7-bit address byte must contain the 7 bits of the I^2C device address and a R/$\overline{W}$ bit that defines if the message will be a write to the slave (master transmission and slave reception) or a read from the slave (slave transmission and master reception).

24.5.2.2 Sending a 10-Bit Address to the Slave

Sending a 10-bit device address involves sending 2 bytes to the slave. The first byte contains 5 bits of the I^2C device address reserved for 10-Bit Addressing modes and 2 bits of the 10-bit address. Because the next byte, which contains the remaining 8 bits of the 10-bit address, must be received by the slave, the R/$\overline{W}$ bit in the first byte must be '0', indicating master transmission and slave reception. If the message data is also directed toward the slave, the master can continue sending the data. However, if the master expects a reply from the slave, a Repeated Start sequence with the R/$\overline{W}$ bit at '1' will change the R/$\overline{W}$ state of the message to a read of the slave.

24.5.2.3 Receiving Acknowledge From the Slave

On the falling edge of the eighth SCLx clock, the TBF bit is cleared and the master will deassert the SDAx pin, allowing the slave to respond with an Acknowledge. The master will then generate a ninth SCLx clock.

This allows the slave device being addressed to respond with an $\overline{ACK}$ bit during the ninth bit time if an address match occurs or data was received properly. A slave sends an Acknowledge when it has recognized its device address (including a general call) or when the slave has properly received its data.

The status of $\overline{ACK}$ is written into the Acknowledge Status bit, ACKSTAT (I2CxSTAT<15>), on the falling edge of the ninth SCLx clock. After the ninth SCLx clock, the module generates the I2CxMIF interrupt and enters an Idle state until the next data byte is loaded into I2CxTRN.

24.5.2.4 ACKSTAT Status Flag

The ACKSTAT bit (I2CxSTAT<15>) is updated in both Master and Slave modes on the 9th SCL clock irrespective of Transmit or Receive modes. ACKSTAT is cleared when acknowledged ($\overline{ACK}$ = 0 i.e.,SDA is 0 on the 9th clock pulse), and is set when not acknowledged ($\overline{ACK}$ = 1, i.e., SDA is 1 on the 9th clock pulse) by the peer.

24.5.2.5 TBF Status Flag

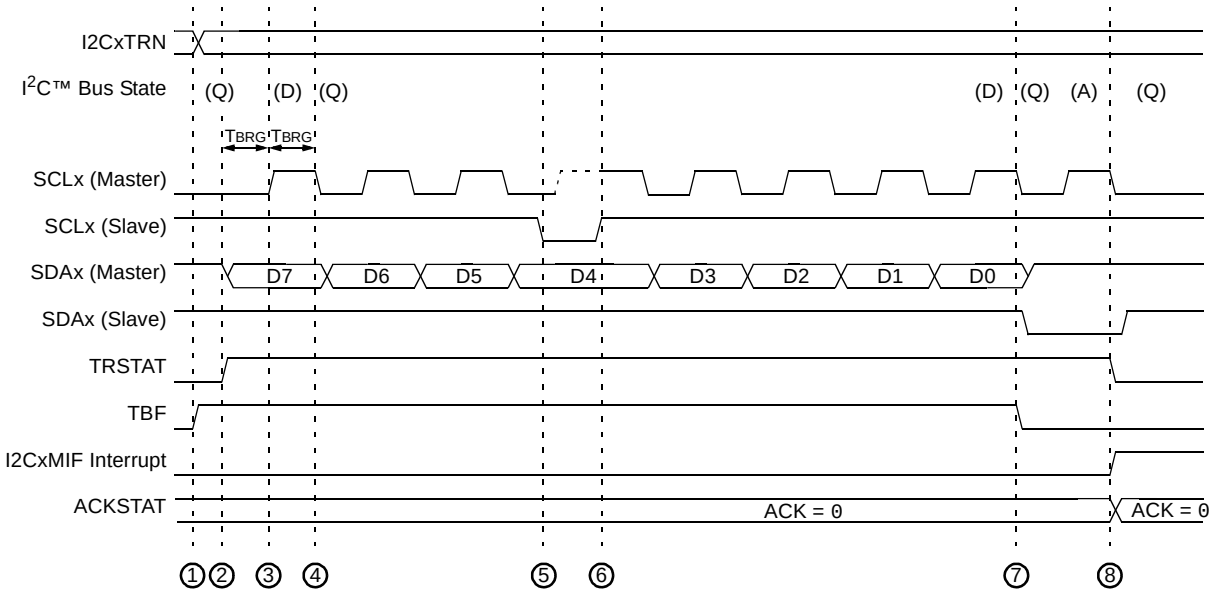
When transmitting, the TBF bit (I2CxSTAT<0>) is set when the CPU writes to I2CXTRN and is cleared when all 8 bits are shifted out.

24.5.2.6 IWCOL Status Flag

If the software writes the I2CxTRN when a transmit is already in progress (i.e., the module is still shifting out a data byte), then IWCOL is set and the contents of the buffer are unchanged (the write doesn't occur). IWCOL must be cleared in software.

Note:	Because queueing of events is not allowed, writing to the lower 5 bits of I2CxCON is disabled until the transmit condition is complete.

Figure 24-9: Master Transmission Timing Diagram



1. Writing the I2CxTRN register will start a master transmission event. TBF bit is set.
2. Baud Rate Generator starts. The MSB of the I2CxTRN drives SDAx. SCLx remains low. TRSTAT bit is set.
3. Baud Rate Generator times out. SCLx released. Baud Rate Generator restarts.
4. Baud Rate Generator times out. SCLx driven low. After SCLx detected low, the next bit of I2CxTRN drives SDAx.
5. While SCLx is low, the slave can also pull SCLx low to initiate a wait (clock stretch).
6. Master has already released SCLx and slave can release to end wait. Baud Rate Generator restarts.
7. At falling edge of 8th SCLx clock, master releases SDAx. TBF bit is cleared. Slave drives $\overline{\text{ACK}}$/NACK.
8. At falling edge of 9th SCLx clock, master generates interrupt. SCLx remains low until next event. Slave releases SDAx. TRSTAT bit is clear.

24.5.3 Receiving Data from a Slave Device

Figure 24-10 shows the timing diagram of master reception. The master can receive data from a slave device after the master has transmitted the slave address with an R/$\overline{W}$ bit value of '1'. This is enabled by setting the Receive Enable bit, RCEN (I2CxCON<3>). The master logic begins to generate clocks, and before each falling edge of the SCLx, the SDAx line is sampled and data is shifted into the I2CxRSR.

Note:	The lower 5 bits of I2CxCON must be '0' before attempting to set the RCEN bit. This ensures the master logic is inactive.

After the falling edge of the eighth SCLx clock, the following events occur:

- The RCEN bit is automatically cleared.
- The contents of the I2CxRSR transfer into the I2CxRCV.
- The RBF flag bit is set.
- The module generates the I2CxMIF interrupt.

When the CPU reads the buffer, the RBF flag bit is automatically cleared. The software can process the data and then do an Acknowledge sequence.

24.5.3.1 RBF Status Flag

When receiving data, the RBF bit is set when a device address or data byte is loaded into I2CxRCV from I2CxRSR. It is cleared when software reads the I2CxRCV register.

24.5.3.2 I2COV Status Flag

If another byte is received in the I2CxRSR while the RBF bit remains set and the previous byte remains in the I2CxRCV register, the I2COV bit is set and the data in the I2CxRSR is lost.

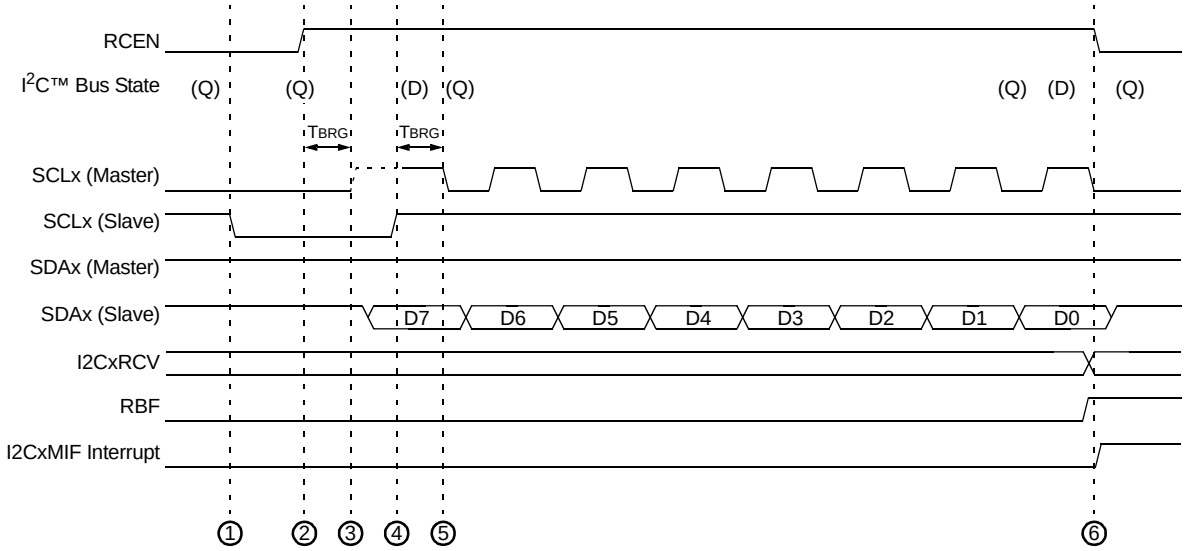
Leaving I2COV set does not inhibit further reception. If RBF is cleared by reading the I2CxRCV and the I2CxRSR receives another byte, that byte will be transferred to the I2CxRCV.

24.5.3.3 IWCOL Status Flag

If the software writes the I2CxTRN when a receive is already in progress (i.e., I2CxRSR is still shifting in a data byte), then the IWCOL bit is set and the contents of the buffer are unchanged (the write doesn't occur).

Note:	Since queueing of events is not allowed, writing to the lower 5 bits of I2CxCON is disabled until the data reception condition is complete.

Figure 24-10: Master Reception Timing Diagram



1. Typically, the slave can pull SCLx low (clock stretch) to request a wait to prepare data response. The slave will drive the MSB of the data response on SDAx when ready.
2. Writing the RCEN bit will start a master reception event. The Baud Rate Generator starts. SCLx remains low.
3. Baud Rate Generator times out. Master attempts to release SCLx.
4. When slave releases SCLx, Baud Rate Generator restarts.
5. Baud Rate Generator times out. MSB of response shifted to I2CxRSR. SCLx driven low for next baud interval.
6. At falling edge of 8th SCLx clock, I2CxRSR transferred to I2CxRCV. Module clears RCEN bit. RBF bit is set. Master generates interrupt.

24.5.4 Acknowledge Generation

Setting the Acknowledge Enable bit, ACKEN (I2CxCON<4>), enables generation of a master Acknowledge sequence.

Note:	The lower 5 bits of I2CxCON must be '0' (master logic inactive) before attempting to set the ACKEN bit.

Figure 24-11 shows an $\overline{ACK}$ sequence and Figure 24-12 shows a NACK sequence. The Acknowledge Data bit, ACKDT (I2CxCON<5>), specifies $\overline{ACK}$ or NACK.

After two baud periods, the ACKEN bit is automatically cleared and the module generates the I2CxMIF interrupt.

24.5.4.1 IWCOL Status Flag

If the software writes the I2CxTRN when an Acknowledge sequence is in progress, then IWCOL is set and the contents of the buffer are unchanged (the write doesn't occur).

Note:	Because queueing of events is not allowed, writing to the lower 5 bits of I2CxCON is disabled until the Acknowledge condition is complete.

Figure 24-11: Master Acknowledge ($\overline{ACK}$) Timing Diagram

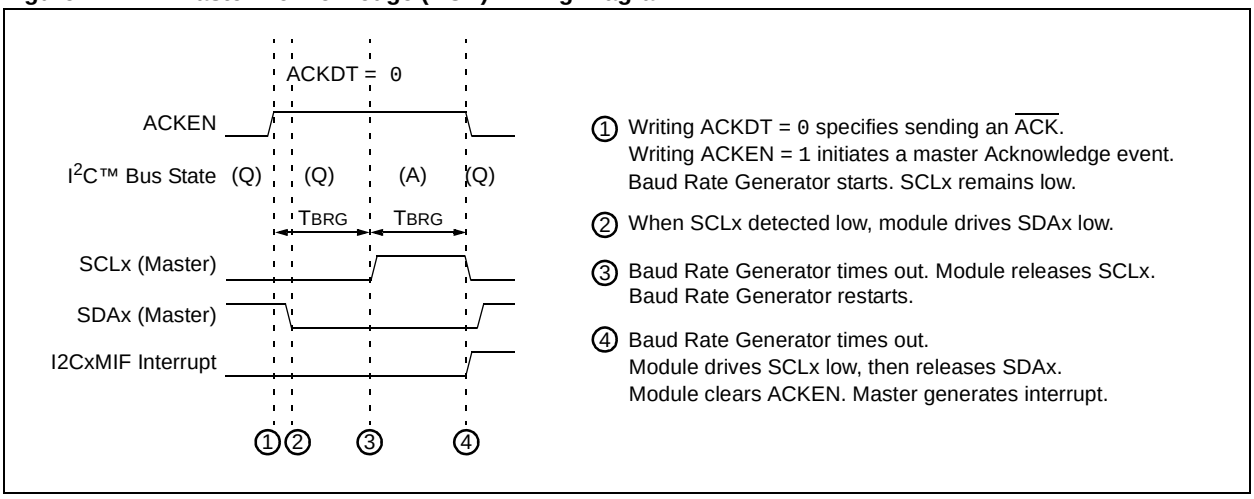
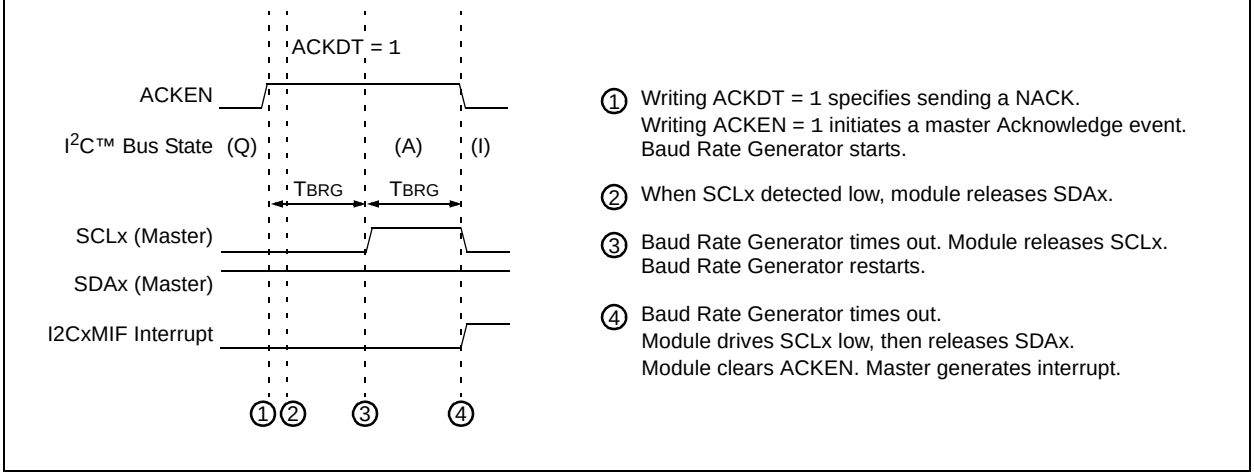


Figure 24-12: Master Not Acknowledge (NACK) Timing Diagram



24.5.5 Generating Stop Bus Event

Setting the Stop Enable bit, PEN (I2CxCON<2>), enables generation of a master Stop sequence.

Note:	The lower 5 bits of I2CxCON must be '0' (master logic inactive) before attempting to set the PEN bit.

When the PEN bit is set, the master generates the Stop sequence as shown in Figure 24-13.

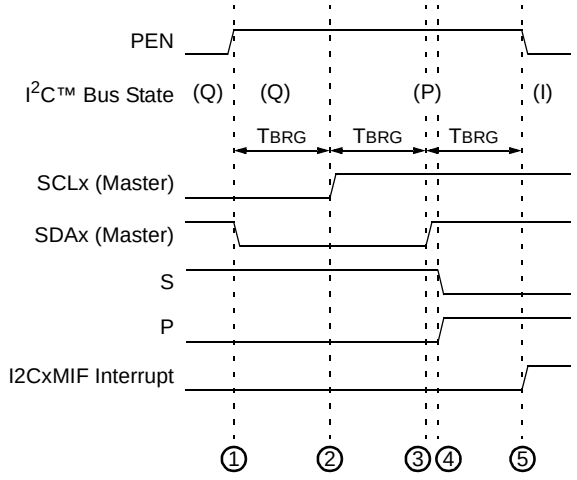
- The slave detects the Stop condition, sets the P bit (I2CxSTAT<4>) and clears the S bit (I2CxSTAT<3>).
- The PEN bit is automatically cleared.
- The module generates the I2CxMIF interrupt.

24.5.5.1 IWCOL Status Flag

If the software writes the I2CxTRN when a Stop sequence is in progress, then the IWCOL bit is set and the contents of the buffer are unchanged (the write doesn't occur).

Note:	Because queueing of events is not allowed, writing to the lower 5 bits of I2CxCON is disabled until the Stop condition is complete.

Figure 24-13: Master Stop Timing Diagram



① Writing PEN = 1 initiates a master Stop event. Baud Rate Generator starts. Module drives SDAx low.

② Baud Rate Generator times out. Module releases SCLx. Baud Rate Generator restarts.

③ Baud Rate Generator times out. Module releases SDAx. Baud Rate Generator restarts.

④ Slave logic detects Stop. Module sets P = 1, S = 0.

⑤ The Baud Rate Generator times out. Module clears PEN. Master generates interrupt.

24.5.6 Generating Repeated Start Bus Event

Setting the Repeated Start Enable bit, RSEN (I2CxCON<1>), enables generation of a master Repeated Start sequence (see Figure 24-14).

Note:	The lower 5 bits of I2CxCON must be '0' (master logic inactive) before attempting to set the RSEN bit.

To generate a Repeated Start condition, software sets the RSEN bit (I2CxCON<1>). The module asserts the SCLx pin low. When the module samples the SCLx pin low, the module releases the SDAx pin for one Baud Rate Generator count (TBRG). When the Baud Rate Generator times out and the module samples SDAx high, the module deasserts the SCLx pin. When the module samples the SCLx pin high, the Baud Rate Generator reloads and begins counting. SDAx and SCLx must be sampled high for one TBRG. This action is then followed by assertion of the SDAx pin low for one TBRG while SCLx is high.

The following is the Repeated Start sequence:

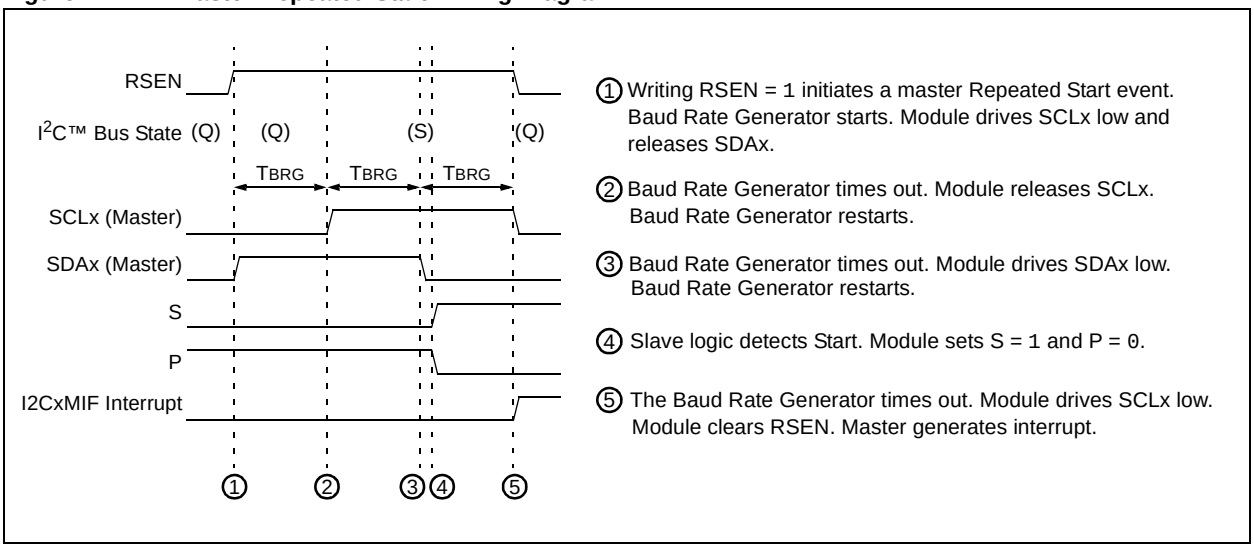
- The slave detects the Start condition, sets the S bit (I2CxSTAT<3>) and clears the P bit (I2CxSTAT<4>).
- The RSEN bit is automatically cleared.
- The module generates the I2CxMIF interrupt.

24.5.6.1 IWCOL Status Flag

If the software writes the I2CxTRN when a Repeated Start sequence is in progress, then IWCOL is set and the contents of the buffer are unchanged (the write doesn't occur).

Note:	Because queueing of events is not allowed, writing of the lower 5 bits of I2CxCON is disabled until the Repeated Start condition is complete.

Figure 24-14: Master Repeated Start Timing Diagram



24.5.7 Building Complete Master Messages

As described at the beginning of **Section 24.5 "Communicating as a Master in a Single Master Environment"**, the software is responsible for constructing messages with the correct message protocol. The module controls individual portions of the I^2C message protocol; however, sequencing of the components of the protocol to construct a complete message is a software task.

The software can use polling or interrupt methods while using the module. The examples shown use interrupts.

The software can use the SEN, RSEN, PEN, RCEN and ACKEN bits (Least Significant 5 bits of the I2CxCON register) and the TRSTAT bit as "state" flags when progressing through a message. For example, Table 24-2 shows some example state numbers associated with bus states.

Table 24-2: Master Message Protocol States

Example State Number	I2CxCON<4:0>	TRSTAT (I2CxSTAT<14>)	State
0	00000	0	Bus Idle or Wait
1	00001	N/A	Sending Start Event
2	00000	1	Master Transmitting
3	00010	N/A	Sending Repeated Start Event
4	00100	N/A	Sending Stop Event
5	01000	N/A	Master Reception
6	10000	N/A	Master Acknowledgement

Note: Example state numbers are for reference only. User software may assign state numbers as desired.

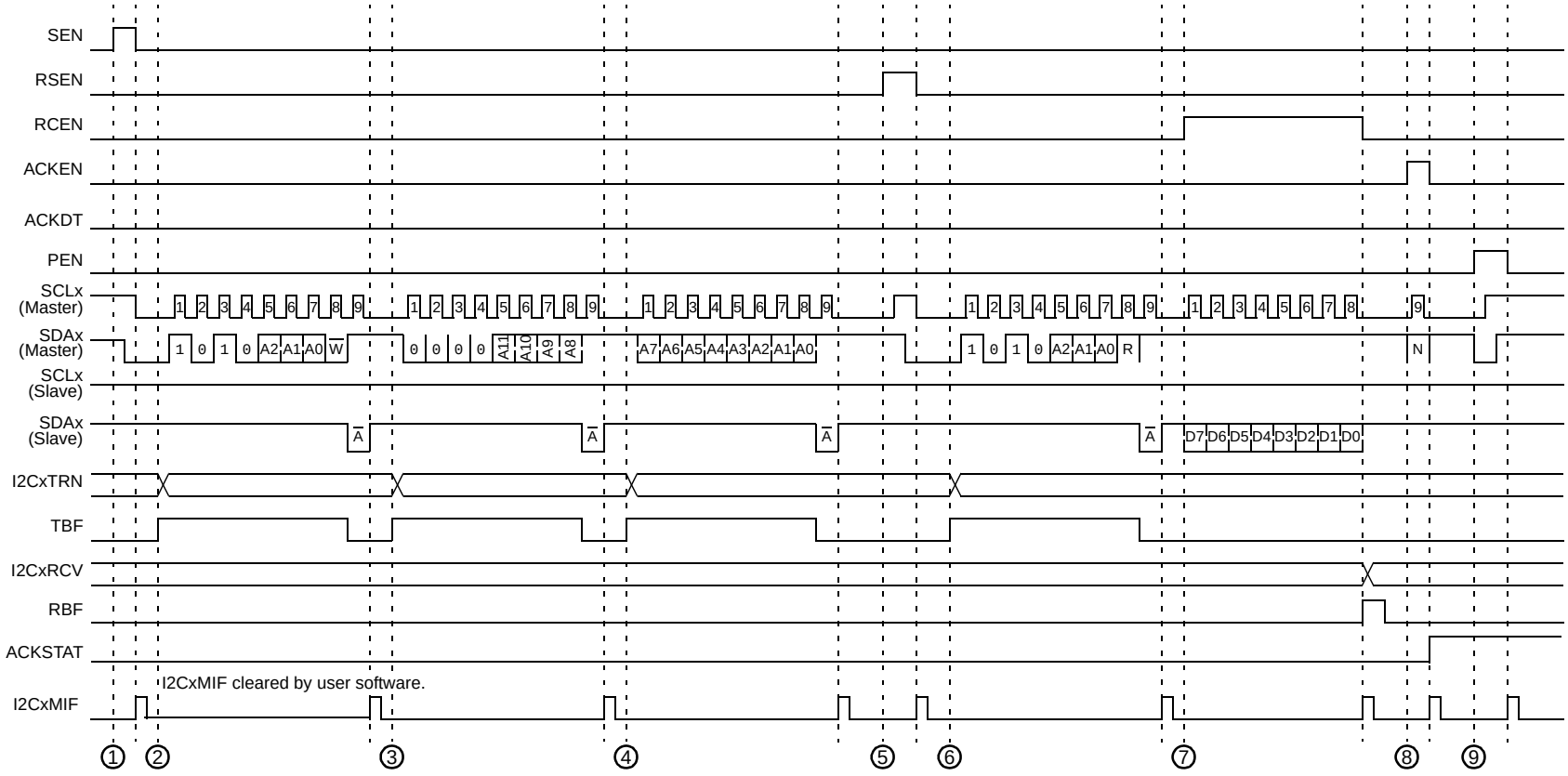
The software will begin a message by issuing a START command. The software will record the state number corresponding to the Start.

As each event completes and generates an interrupt, the interrupt handler may check the state number. So, for a Start state, the interrupt handler will confirm execution of the Start sequence and then start a master transmission event to send the I^2C device address, changing the state number to correspond to the master transmission.

On the next interrupt, the interrupt handler will again check the state, determining that a master transmission just completed. The interrupt handler will confirm successful transmission of the data, then move on to the next event, depending on the contents of the message. In this manner, on each interrupt, the interrupt handler will progress through the message protocol until the complete message is sent.

Figure 24-15 provides a more detailed examination of the same message sequence shown in Figure 24-7. Figure 24-16 shows some simple examples of messages using 7-bit addressing format. Figure 24-17 shows an example of a 10-bit addressing format message sending data to a slave. Figure 24-18 shows an example of a 10-bit addressing format message receiving data from a slave.

Figure 24-15: Master Message (Typical I²C Message: Read of Serial EEPROM)



① Setting the SEN bit starts a Start event.

② Writing the I2CxTRN register starts a master transmission. The data is the serial EEPROM device address byte, with R/$\overline{W}$ clear, indicating a write.

③ Writing the I2CxTRN register starts a master transmission. The data is the first byte of the EEPROM data address.

④ Writing the I2CxTRN register starts a master transmission. The data is the second byte of the EEPROM data address.

⑤ Setting the RSEN bit starts a Repeated Start event.

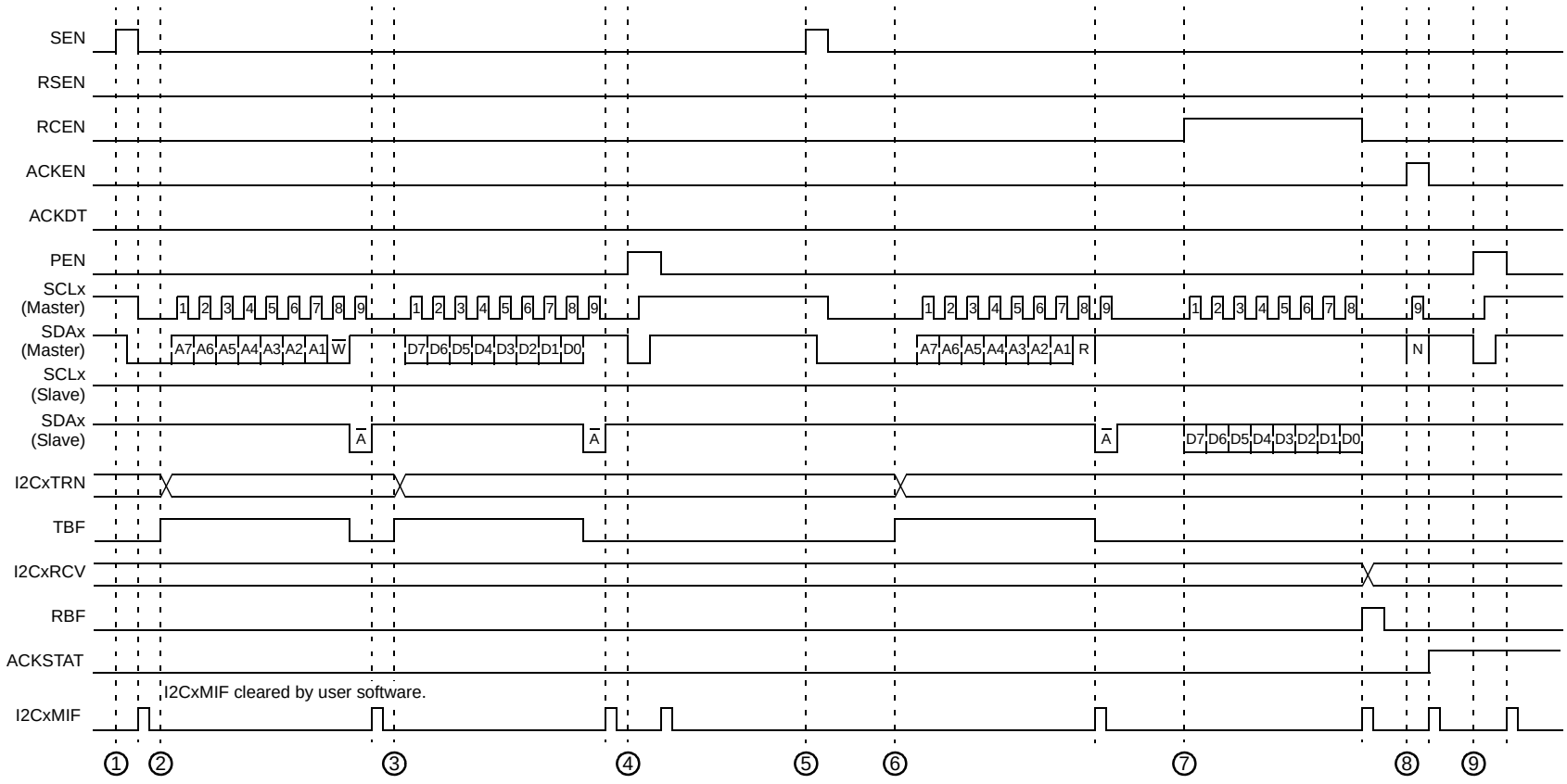
⑥ Writing the I2CxTRN register starts a master transmission. The data is a re-send of the serial EEPROM device address byte, but with R/$\overline{W}$ bit set, indicating a read.

⑦ Setting the RCEN bit starts a master reception. On interrupt, the software reads the I2CxRCV register, which clears the RBF flag.

⑧ Setting the ACKEN bit starts an Acknowledge event. ACKDT = 0 to send NACK.

⑨ Setting the PEN bit starts a master Stop event.

Figure 24-16: Master Message (7-Bit Address: Transmission And Reception)



① Setting the SEN bit starts a Start event.

② Writing the I2CxTRN register starts a master transmission. The data is the address byte with R/$\overline{W}$ bit clear.

③ Writing the I2CxTRN register starts a master transmission. The data is the message byte.

④ Setting the PEN bit starts a master Stop event.

⑤ Setting the SEN bit starts a Start event.

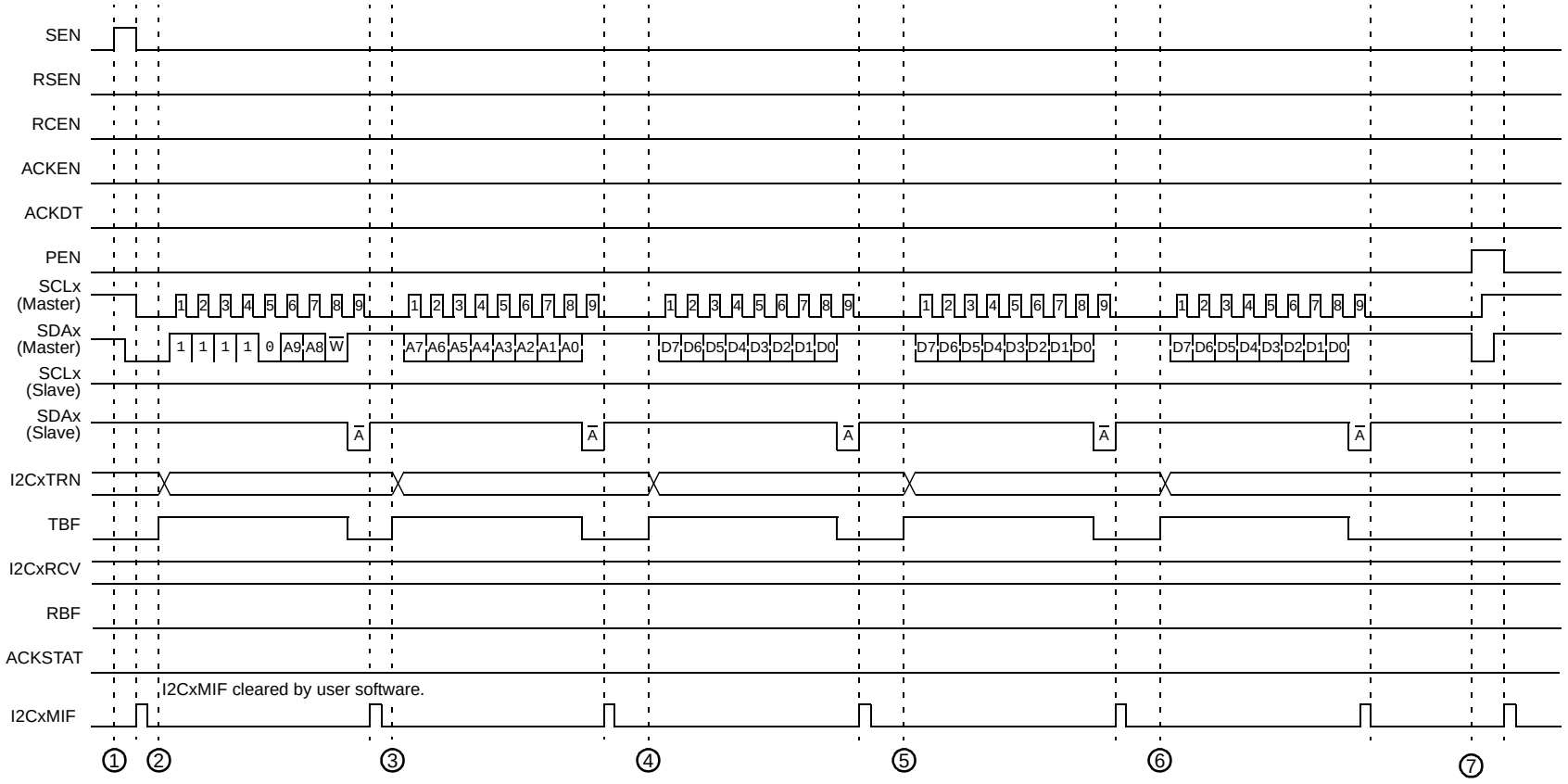
⑥ Writing the I2CxTRN register starts a master transmission. The data is the address byte with R/$\overline{W}$ bit set.

⑦ Setting the RCEN bit starts a master reception.

⑧ Setting the ACKEN bit starts an Acknowledge event. ACKDT = 0 to send NACK.

⑨ Setting the PEN bit starts a master Stop event.

Figure 24-17: Master Message (10-Bit Transmission)



① Setting the SEN bit starts a Start event.

② Writing the I2CxTRN register starts a master transmission. The data is the first byte of the address.

③ Writing the I2CxTRN register starts a master transmission. The data is the second byte of the address.

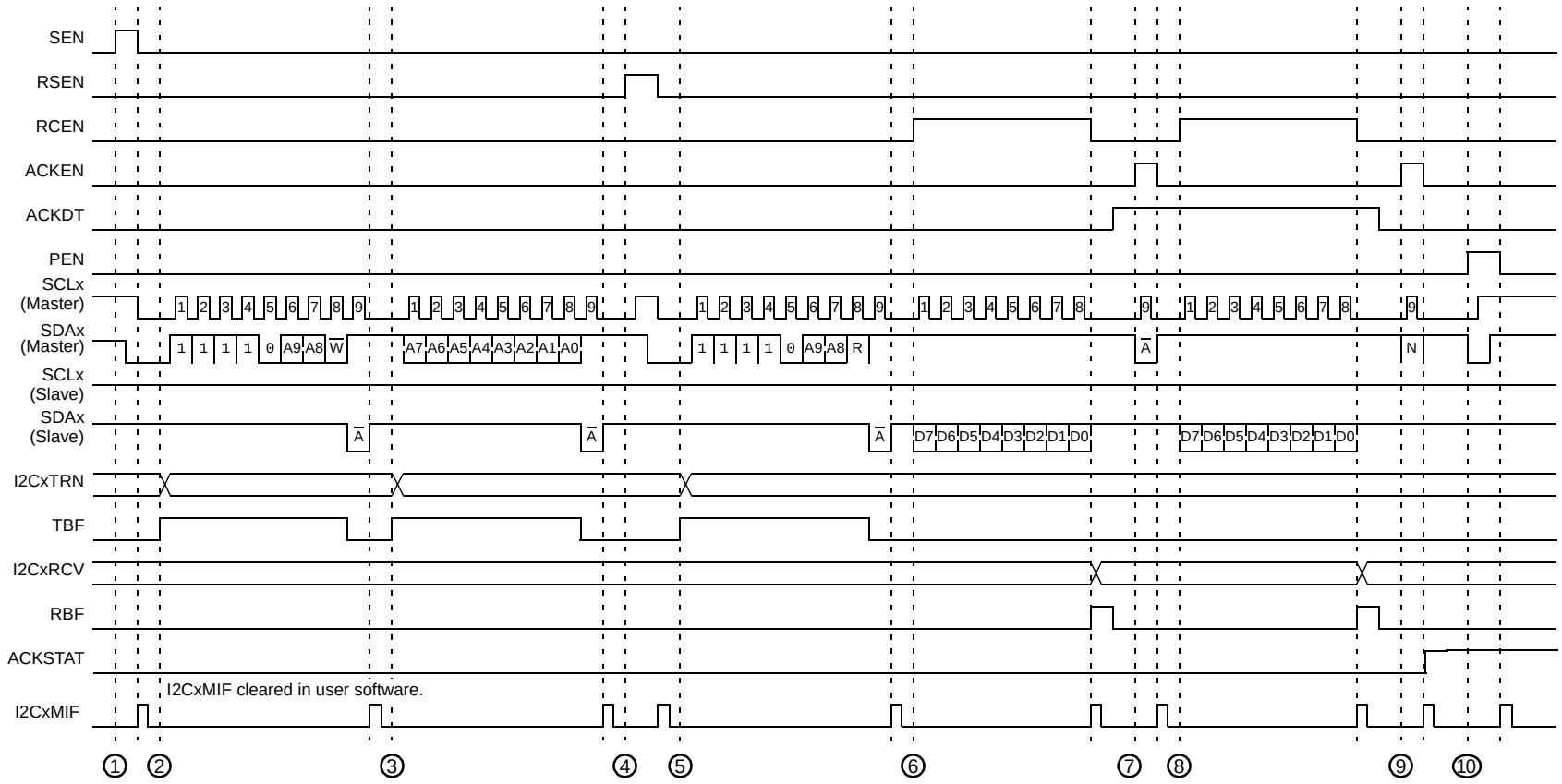
④ Writing the I2CxTRN register starts a master transmission. The data is the first byte of the message data.

⑤ Writing the I2CxTRN register starts a master transmission. The data is the second byte of the message data.

⑥ Writing the I2CxTRN register starts a master transmission. The data is the third byte of the message data.

⑦ Setting the PEN bit starts a master Stop event.

Figure 24-18: Master Message (10-Bit Reception)



① Setting the SEN bit starts a Start event.

② Writing the I2CxTRN register starts a master transmission. The data is the first byte of the address with the R/$\overline{W}$ bit cleared.

③ Writing the I2CxTRN register starts a master transmission. The data is the second byte of the address.

④ Setting the RSEN bit starts a master Restart event.

⑤ Writing the I2CxTRN register starts a master transmission. The data is a re-send of the first byte with the R/$\overline{W}$ bit set.

⑥ Setting the RCEN bit starts a master reception. On interrupt, the software reads the I2CxRCV register, which clears the RBF flag.

⑦ Setting the ACKEN bit starts an Acknowledge event. ACKDT = 1 to send $\overline{ACK}$.

⑧ Setting the RCEN bit starts a master reception.

⑨ Setting the ACKEN bit starts an Acknowledge event. ACKDT = 0 to send NACK.

⑩ Setting the PEN bit starts a master Stop event.

24.6 COMMUNICATING AS A MASTER IN A MULTI-MASTER ENVIRONMENT

The I^2C protocol allows for more than one master to be attached to a system bus. Taking into account that a master can initiate message transactions and generate clocks for the bus, the protocol has methods to account for situations where more than one master is attempting to control the bus. Clock synchronization ensures that multiple nodes can synchronize their SCLx clocks to result in one common clock on the SCLx line. Bus arbitration ensures that if more than one node attempts a message transaction, one node, and only one node, will be successful in completing the message. The other nodes will lose bus arbitration and be left with a bus collision.

24.6.1 Multi-Master Operation

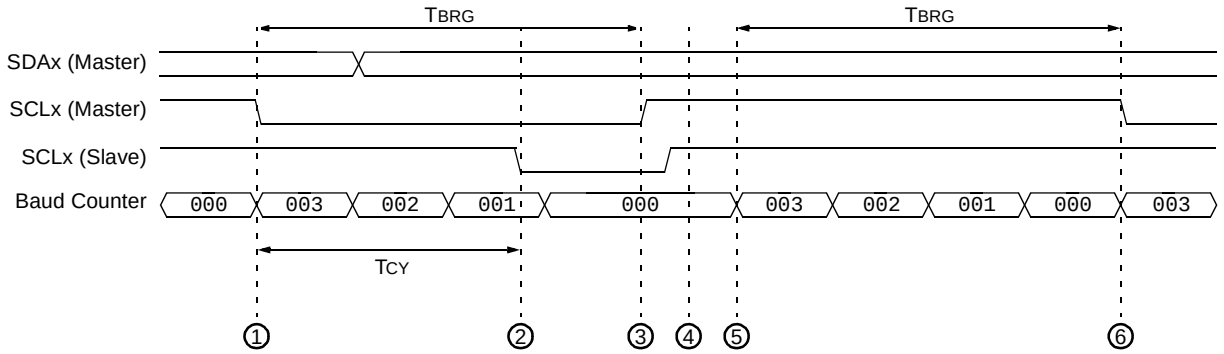
The master module has no special settings to enable multi-master operation. The module performs clock synchronization and bus arbitration at all times. If the module is used in a single master environment, clock synchronization will only occur between the master and slaves, and bus arbitration will not occur.

24.6.2 Master Clock Synchronization

In a multi-master system, different masters may have different baud rates. Clock synchronization will ensure that when these masters are attempting to arbitrate the bus, their clocks will be coordinated.

Clock synchronization occurs when the master deasserts the SCLx pin (SCLx intended to float high). When the SCLx pin is released, the BRG is suspended from counting until the SCLx pin is actually sampled high. When the SCLx pin is sampled high, the BRG is reloaded with the contents of I2CxBR<11:0> and begins counting. This ensures that the SCLx high time will always be at least one BRG rollover count in the event that the clock is held low by an external device, as shown in Figure 24-19.

Figure 24-19: Baud Rate Generator Timing with Clock Synchronization



1. The baud counter decrements twice per TCY. On rollover, the master SCLx will transition.
2. The slave has pulled SCLx low to initiate a wait.
3. At what would be the master baud counter rollover, detecting SCLx low holds counter.
4. Logic samples SCLx once per TCY. Logic detects SCLx high.
5. The baud counter rollover occurs on next cycle.
6. On next rollover, the master SCLx will transition.

24.6.3 Bus Arbitration and Bus Collision

Bus arbitration supports multi-master system operation.

The wired AND nature of the SDAx line permits arbitration. Arbitration takes place when the first master outputs a '1' on SDAx by letting SDAx float high and simultaneously, the second master outputs a '0' on SDAx by pulling SDAx low. The SDAx signal will go low. In this case, the second master has won bus arbitration. The first master has lost bus arbitration and thus, has a bus collision.

For the first master, the expected data on SDAx is a '1', yet the data sampled on SDAx is a '0'. This is the definition of a bus collision.

The first master will set the Bus Collision bit, BCL (I2CxSTAT<10>), and generate a bus collision interrupt. The master module will reset the I^2C port to its Idle state.

In multi-master operation, the SDAx line must be monitored for arbitration to see if the signal level is the expected output level. This check is performed by the master module, with the result placed in the BCL bit.

The states where arbitration can be lost are:

- A Start condition
- A Repeated Start condition
- Address, Data or Acknowledge bit
- A Stop condition

24.6.4 Detecting Bus Collisions and Re-sending Messages

When a bus collision occurs, the module sets the BCL bit and generates a bus collision interrupt. If bus collision occurs during a byte transmission, the transmission is halted, the TBF flag is cleared and the SDAx and SCLx pins are deasserted. If bus collision occurs during a Start, Repeated Start, Stop or Acknowledge condition, the condition is aborted, the respective control bits in the I2CxCON register are cleared and the SDAx and SCLx lines are deasserted.

The software is expecting an interrupt at the completion of the master event. The software can check the BCL bit to determine if the master event completed successfully or a collision occurred. If a collision occurs, the software must abort sending the rest of the pending message and prepare to re-send the entire message sequence, beginning with the Start condition, after the bus returns to an Idle state. The software can monitor the S and P bits to wait for an Idle bus. When the software services the bus collision Interrupt Service Routine and the I^2C bus is free, the software can resume communication by asserting a Start condition.

24.6.5 Bus Collision During a Start Condition

Before issuing a Start command, the software should verify an Idle state of the bus using the S and P Status bits. Two masters may attempt to initiate a message at a similar point in time. Typically, the masters will synchronize clocks and continue arbitration into the message until one loses arbitration. However, certain conditions can cause a bus collision to occur during a Start. In this case, the master that loses arbitration during the Start bit generates a bus collision interrupt.

24.6.6 Bus Collision During a Repeated Start Condition

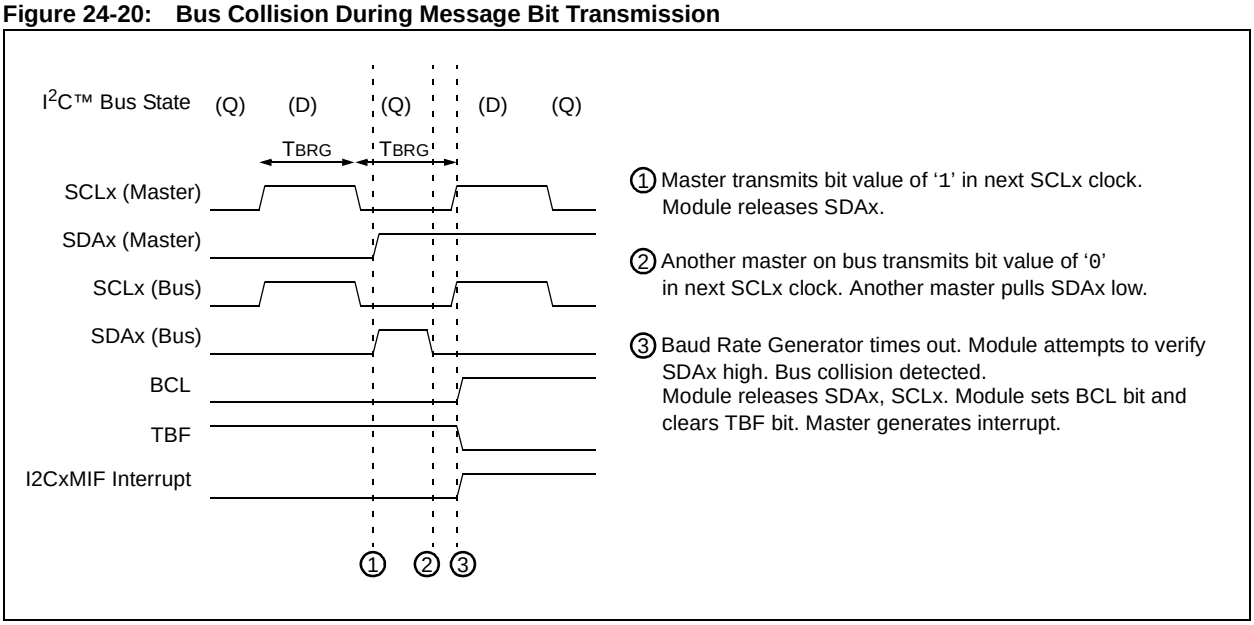
Should two masters not collide throughout an address byte, a bus collision may occur when one master attempts to assert a Repeated Start while another transmits data. In this case, the master generating the Repeated Start will lose arbitration and generate a bus collision interrupt.

24.6.7 Bus Collision During Message Bit Transmission

The most typical case of data collision occurs while the master is attempting to transmit the device address byte, a data byte or an Acknowledge bit.

If the software is properly checking the bus state, it is unlikely that a bus collision will occur on a Start condition. However, because another master can, at a very similar time, check the bus and initiate its own Start condition, it is likely that SDAx arbitration will occur and synchronize the Start of two masters. In this condition, both masters will begin and continue to transmit their messages until one master loses arbitration on a message bit. Remember that the SCLx clock synchronization will keep the two masters synchronized until one loses arbitration. Figure 24-20 shows an example of message bit arbitration.

Figure 24-20: Bus Collision During Message Bit Transmission



24.6.8 Bus Collision During a Stop Condition

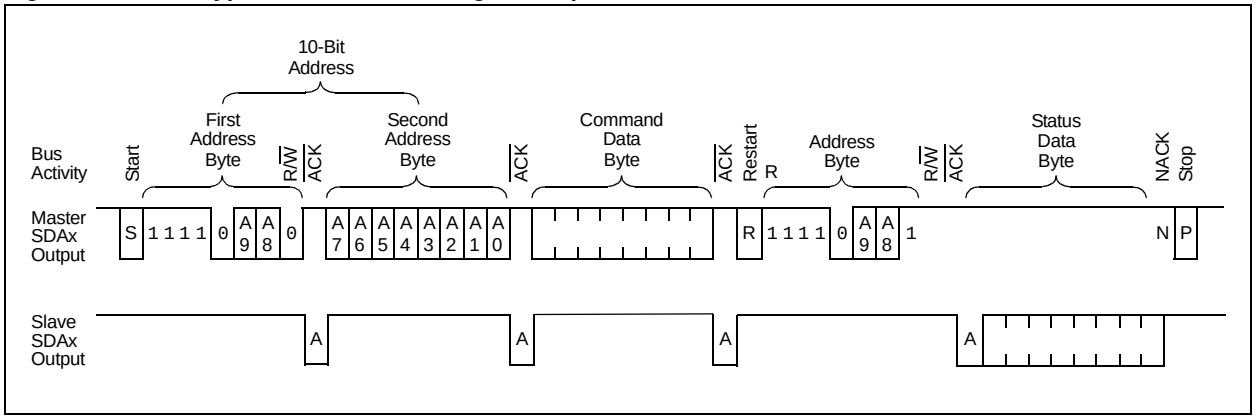
If the master software loses track of the state of the I^{2}C bus, there are conditions which cause a bus collision during a Stop condition. In this case, the master generating the Stop condition will lose arbitration and generate a bus collision interrupt.

24.7 COMMUNICATING AS A SLAVE

In some systems, particularly where multiple processors communicate with each other, the PIC32MX device may communicate as a slave (see Figure 24-21). When the module is enabled, the slave module is active. The slave may not initiate a message, it can only respond to a message sequence initiated by a master. The master requests a response from a particular slave as defined by the device address byte in the I^2C protocol. The slave module replies to the master at the appropriate times as defined by the protocol.

As with the master module, sequencing the components of the protocol for the reply is a software task. However, the slave module detects when the device address matches the address specified by the software for that slave.

Figure 24-21: A Typical Slave I^2C Message: Multiprocessor Command/Status



After a Start condition, the slave module will receive and check the device address. The slave may specify either a 7-bit address or a 10-bit address. When a device address is matched, the module will generate an interrupt to notify the software that its device is selected. Based on the R/$\overline{W}$ bit sent by the master, the slave will either receive or transmit data. If the slave is to receive data, the slave module automatically generates the Acknowledge ($\overline{ACK}$), loads the I2CxRCV register with the received value currently in the I2CxRSR register and notifies the software through an interrupt. If the slave is to transmit data, the software must load the I2CxTRN register.

24.7.1 Sampling Receive Data

All incoming bits are sampled with the rising edge of the clock (SCLx) line.

24.7.2 Detecting Start and Stop Conditions

The slave module will detect Start and Stop conditions on the bus and indicate that status on the S bit (I2CxSTAT<3>) and P bit (I2CxSTAT<4>). The Start (S) and Stop (P) bits are cleared when a Reset occurs or when the module is disabled. After detection of a Start or Repeated Start event, the S bit is set and the P bit is cleared. After detection of a Stop event, the P bit is set and the S bit is clear.

24.7.3 Detecting the Address

Once the module has been enabled, the slave module waits for a Start condition to occur. After a Start, depending on the A10M bit (I2CxCON<10>), the slave will attempt to detect a 7-bit or 10-bit address. The slave module will compare one received byte for a 7-bit address or two received bytes for a 10-bit address. A 7-bit address also contains an R/$\overline{W}$ bit that specifies the direction of data transfer after the address. If R/$\overline{W}$ = 0, a write is specified and the slave will receive data from the master. If R/$\overline{W}$ = 1, a read is specified and the slave will send data to the master. The 10-bit address contains an R/$\overline{W}$ bit; however, by definition, it is always R/$\overline{W}$ = 0 because the slave must receive the second byte of the 10-bit address.

24.7.3.1 Slave Address Masking

The I2CxMSK register masks address bit positions, designating them as “don’t care” bits for both 10-Bit and 7-Bit Addressing modes. When a bit in the I2CxMSK register is set (= 1), it means “don’t care”. The slave module will respond when the bit in the corresponding location of the address is a ‘0’ or ‘1’. For example, in 7-Bit Slave mode with I2CxMSK = 0110000, the module will Acknowledge addresses ‘0010000’ and ‘0100000’ as valid.

24.7.3.2 Limitations of Address Mask

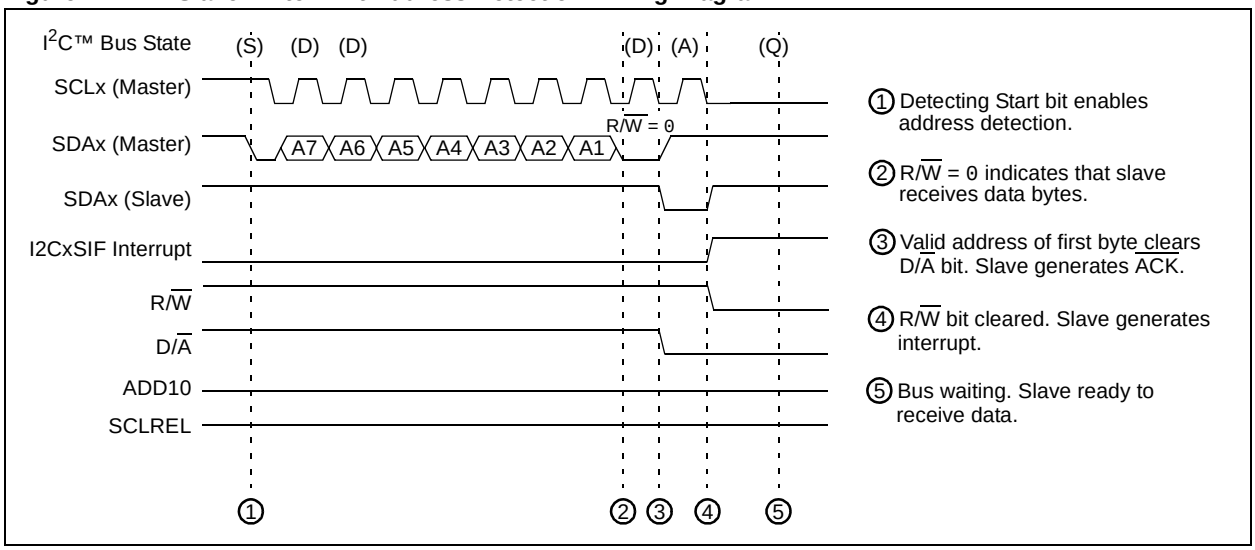
By default, the device will respond or generate addresses in the reserved address space with the address mask enabled (see Table 24-3 for the reserved address spaces). When using the address mask and the STRICT (I2CxCON<11>) bit is cleared, reserved addresses may be acknowledged. If the user wants to enforce the reserved address space, the STRICT bit must be set to a ‘1’. Once the bit is set, the device will not acknowledge reserved addresses regardless of the address mask settings.

24.7.3.3 7-BIT ADDRESS and SLAVE WRITE

Following the Start condition, the module shifts 8 bits into the I2CxRSR register (see Figure 24-22). The value of register I2CxRSR<7:1> is evaluated against that of the I2CxADD<6:0> and I2CxMSK<6:0> registers on the falling edge of the eighth clock (SCLx). If the address is valid (i.e., an exact match between unmasked bit positions), the following events occur:

1. An $\overline{\text{ACK}}$ is generated.
2. The D/$\overline{\text{A}}$ and R/$\overline{\text{W}}$ bits are cleared.
3. The module generates the I2CxSIF interrupt on the falling edge of the ninth SCLx clock.
4. The module will wait for the master to send data.

Figure 24-22: Slave Write 7-Bit Address Detection Timing Diagram



24.7.3.4 7-Bit Address and Slave Read

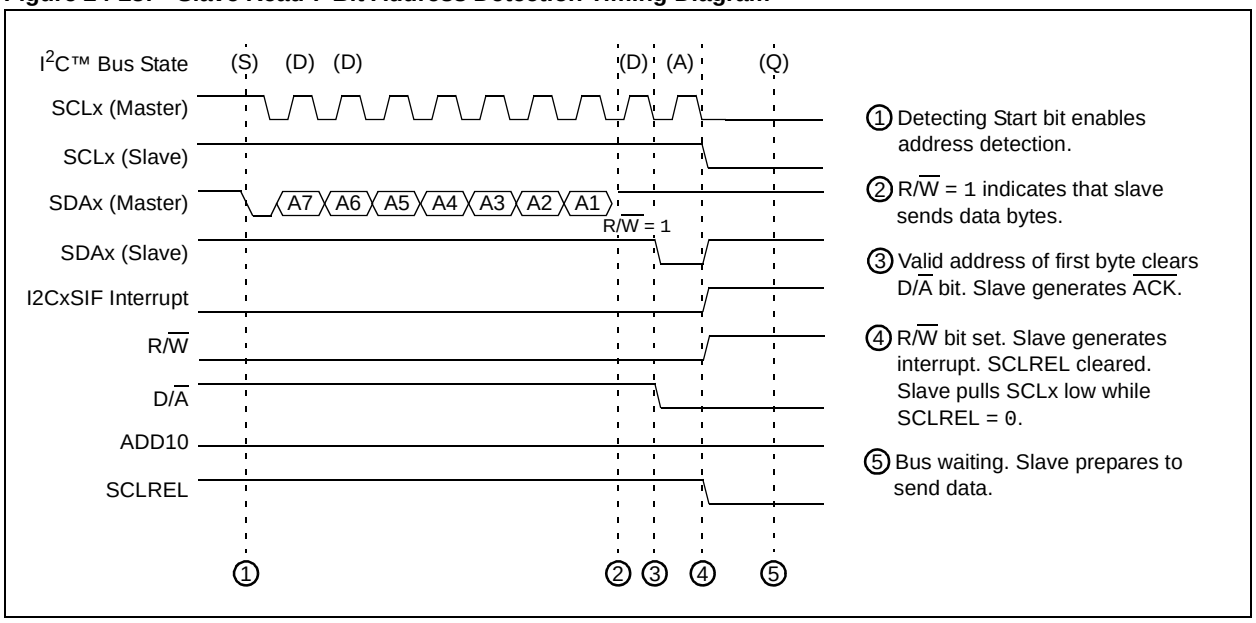
When a slave read is specified by having R/$\overline{\text{W}}$ = 1 in a 7-bit address byte, the process of detecting the device address is similar to that for a slave write (see Figure 24-23). If the addresses match, the following events occur:

1. An $\overline{\text{ACK}}$ is generated.
2. The D/$\overline{\text{A}}$ bit is cleared and the R/$\overline{\text{W}}$ bit is set.
3. The module generates the I2CxSIF interrupt on the falling edge of the ninth SCLx clock.

Since the slave module is expected to reply with data at this point, it is necessary to suspend the operation of the I^2C bus to allow the software to prepare a response. This is done automatically when the module clears the SCLREL bit. With SCLREL low, the slave module will pull down the SCLx clock line, causing a wait on the I^2C bus. The slave module and the I^2C bus will remain in this state until the software writes the I2CxTRN register with the response data and sets the SCLREL bit.

Note:	SCLREL will automatically clear after detection of a slave read address, regardless of the state of the STREN bit.

Figure 24-23: Slave Read 7-Bit Address Detection Timing Diagram



24.7.3.5 10-bit Addressing Mode

Figure 24-24 shows the sequence of address bytes on the bus in 10-bit Address mode. In this mode, the slave must receive two device address bytes (see Figure 24-25). The five Most Significant bits (MSbs) of the first address byte specify a 10-bit address. The R/$\overline{W}$ bit of the address must specify a write, causing the slave device to receive the second address byte. For a 10-bit address, the first byte would equal '11110 A9 A8 0', where 'A9' and 'A8' are the two MSbs of the address.

The I2CxMSK register can mask any bit position in a 10-bit address. The two MSbs of I2CxMSK are used to mask the MSbs of the incoming address received in the first byte. The remaining byte of the register is then used to mask the lower byte of the address received in the second byte.

Following the Start condition, the module shifts eight bits into the I2CxRSR register. The value of the I2CxRSR<2:1> bits are evaluated against the value of the I2CxADD<9:8> and I2CxMSK<9:8> bits, while the value of the I2CxRSR<7:3> bits are compared to '11110'. Address evaluation occurs on the falling edge of the eighth clock (SCLx). For the address to be valid, I2CxRSR<7:3> must equal '11110', while I2CxRSR<2:1> must exactly match any unmasked bits in I2CxADD<9:8>. (If both bits are masked, a match is not needed.) If the address is valid, the following events occur:

1. An $\overline{ACK}$ is generated.
2. The D/$\overline{A}$ and R/$\overline{W}$ bits are cleared.
3. The module generates the I2CxSIF interrupt on the falling edge of the ninth SCLx clock.

The module does generate an interrupt after the reception of the first byte of a 10-bit address; however, this interrupt is of little use.

The module will continue to receive the second byte into I2CxRSR. This time, the I2CxRSR<7:0> bits are evaluated against the I2CADD<7:0> and I2CxMSK<7:0> bits. If the lower byte of the address is valid as previously described, the following events occur:

1. An $\overline{ACK}$ is generated.
2. The ADD10 bit is set.
3. The module generates the I2CxSIF interrupt on the falling edge of the ninth SCLx clock.
4. The module will wait for the master to send data or initiate a Repeated Start condition.

Note:	Following a Repeated Start condition in 10-Bit Addressing mode, the slave module only matches the first 7-bit address, '11110 A9 A8 0'.

Figure 24-24: 10-bit Address Sequence

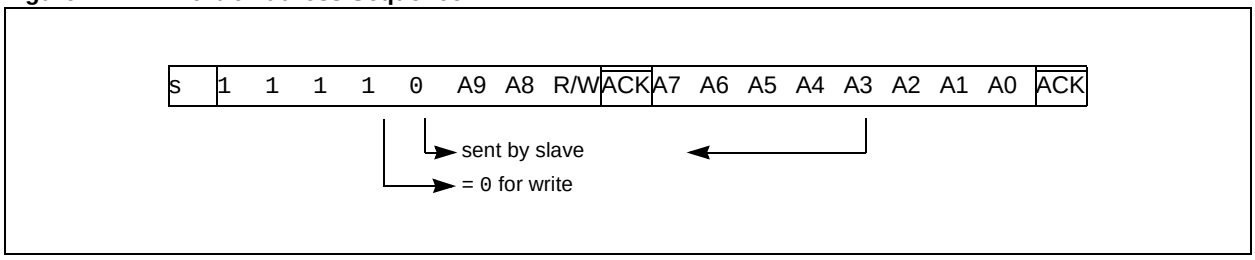
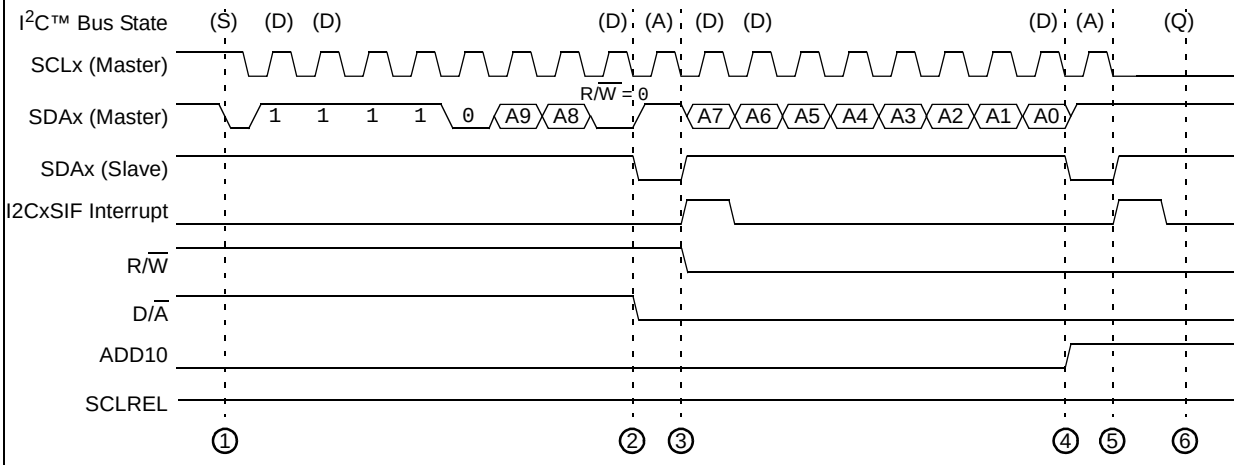


Figure 24-25: 10-Bit Address Detection Timing Diagram



1. Detecting Start bit enables address detection.
2. Address match of first byte clears D/$\overline{A}$ bit and causes slave logic to generate $\overline{ACK}$.
3. Reception of first byte clears R/$\overline{W}$ bit. Slave logic generates interrupt.
4. Address match of first and second byte sets ADD10 and causes slave logic to generate $\overline{ACK}$.
5. Reception of second byte completes 10-bit address. Slave logic generates interrupt.
5. Bus waiting. Slave ready to receive data.

24.7.3.6 General Call Operation

The addressing procedure for the I²C bus is such that the first byte (or first two bytes in case of 10-bit Addressing mode) after a Start condition usually determines which slave device the master is addressing. The exception is the general call address, which can address all devices. When this address is used, all enabled devices should respond with an Acknowledge. The general call address is one of eight addresses reserved for specific purposes by the I²C protocol. It consists of all zeros with R/$\overline{W}$ = 0. The general call is always a slave write operation.

The general call address is recognized when the General Call Enable bit, GCEN (I2CxCON<7>), is set (see Figure 24-26). Following a Start bit detect, eight bits are shifted into the I2CxRSR and the address is compared against the I2CxADD and the general call address.

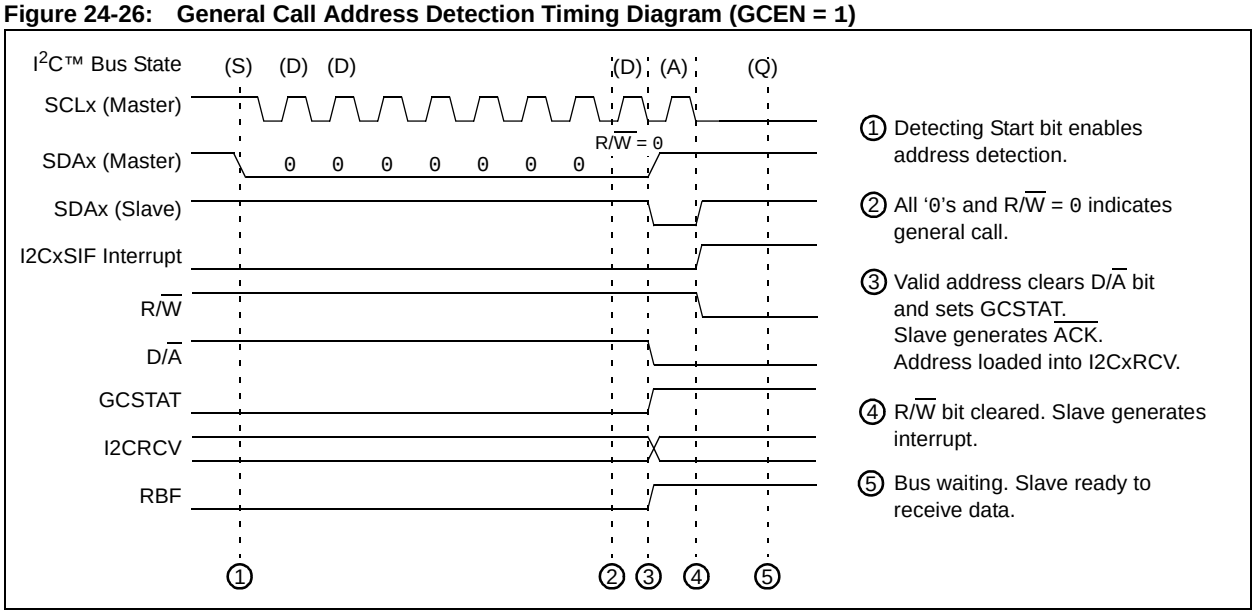
If the general call address matches, the following events occur:

1. An $\overline{ACK}$ is generated.
2. Slave module will set the GCSTAT bit (I2CxSTAT<9>).
3. The D/$\overline{A}$ and R/$\overline{W}$ bits are cleared.
4. The module generates the I2CxSIF interrupt on the falling edge of the ninth SCLx clock.
5. The I2CxRSR is transferred to the I2CxRCV and the RBF flag bit is set (during the eighth bit).
6. The module will wait for the master to send data.

When the interrupt is serviced, the cause for the interrupt can be checked by reading the contents of the GCSTAT bit to determine if the device address was device specific or a general call address.

Note that general call addresses are 7-bit addresses. If configuring the slave module for 10-bit addresses and the A10M and GCEN bits are set, the slave module will continue to detect the 7-bit general call address.

Figure 24-26: General Call Address Detection Timing Diagram (GCEN = 1)



24.7.3.7 STRICT ADDRESS SUPPORT

When the STRICT (I2CxCON<11>) control bit is set, it enables the module to enforce all reserved addressing and will not acknowledge any addresses if they fall within the reserved address table.

24.7.3.8 When an Address is Invalid

If a 7-bit address does not match the contents of I2CxADD<6:0>, the slave module will return to an Idle state and ignore all bus activity until after the Stop condition.

If the first byte of a 10-bit address does not match the contents of I2CxADD<9:8>, the slave module will return to an Idle state and ignore all bus activity until after the Stop condition.

If the first byte of a 10-bit address matches the contents of I2CxADD<9:8> but the second byte of the 10-bit address does not match I2CxADD<7:0>, the slave module will return to an Idle state and ignore all bus activity until after the Stop condition.

24.7.3.9 Addresses Reserved From Masking

Even when enabled, there are several addresses that are excluded in hardware from masking. For these addresses, an Acknowledge will not be issued independent of the mask setting. These addresses are listed in Table .

Table 24-3: Reserved I^2C Bus Addresses(1)

7-Bit Address Mode:		
Slave Address	**R/$\overline{W}$ Bit**	**Description**
0000 000	0	General Call Address(1)
0000 000	1	Start Byte
0000 001	x	CBUS Address
0000 010	x	Reserved
0000 011	x	Reserved
0000 1xx	x	HS Mode Master Code
1111 1xx	x	Reserved
1111 0xx	x	10-Bit Slave Upper Byte(2)

Note 1: Address will be Acknowledged only if GCEN = 1.

2: Match on this address can only occur as the upper byte in the 10-Bit Addressing mode.

24.7.4 Receiving Data From a Master Device

When the R/$\overline{W}$ bit of the device address byte is zero and an address match occurs, the R/$\overline{W}$ bit (I2CxSTAT<2>) is cleared. The slave module enters a state waiting for data to be sent by the master. After the device address byte, the contents of the data byte are defined by the system protocol and are only received by the slave module.

The slave module shifts eight bits into the I2CxRSR register. On the falling edge of the eighth clock (SCLx), the following events occur:

1. The module begins to generate an $\overline{ACK}$ or NACK.
2. The RBF bit is set to indicate received data.
3. The I2CxRSR byte is transferred to the I2CxRCV register for access by the software.
4. The D/$\overline{A}$ bit is set.
5. A slave interrupt is generated. Software may check the status of the I2CxSTAT register to determine the cause of the event and then clear the I2CxSIF flag.
6. The module will wait for the next data byte.

24.7.4.1 Acknowledge Generation

Normally, the slave module will Acknowledge all received bytes by sending an $\overline{ACK}$ on the ninth SCLx clock. If the receive buffer is overrun, the slave module does not generate this $\overline{ACK}$. Overrun is indicated if either (or both):

1. The buffer full bit, RBF (I2CxSTAT<1>), was set before the transfer was received.
2. The overflow bit, I2COV (I2CxSTAT<6>), was set before the transfer was received.

Table 24-4 shows what happens when a data transfer byte is received, given the status of the RBF and I2COV bits. If the RBF bit is already set when the slave module attempts to transfer to the I2CxRCV, the transfer does not occur but the interrupt is generated and the I2COV bit is set. If both the RBF and I2COV bits are set, the slave module acts similarly. The shaded cells show the condition where software did not properly clear the overflow condition.

Reading the I2CxRCV clears the RBF bit. The I2COV is cleared by writing to a '0' through software.

Table 24-4: Data Transfer Received Byte Actions

Status Bits as Data Byte Received		Transfer I2CxRSR to I2CxRCV	Generate ACK	Generate I2CxSIF Interrupt (interrupt occurs if enabled)	Set RBF	Set I2COV
RBF	I2COV					
0	0	Yes	Yes	Yes	Yes	No change
1	0	No	No	Yes	No change	Yes
1	1	No	No	Yes	No change	Yes
0	1	Yes	No	Yes	Yes	No change

Legend: Shaded cells show state where the software did not properly clear the overflow condition.

24.7.4.2 Wait States During Slave Receptions

When the slave module receives a data byte, the master can potentially begin sending the next byte immediately. This allows the software controlling the slave module nine SCLx clock periods to process the previously received byte. If this is not enough time, the slave software may want to generate a bus wait period.

The STREN bit (I2CxCON<6>) enables a bus wait to occur on slave receptions. When STREN = 1 at the falling edge of the 9th SCLx clock of a received byte, the slave module clears the SCLREL bit. Clearing the SCLREL bit causes the slave module to pull the SCLx line low, initiating a wait. The SCLx clock of the master and slave will synchronize, as shown in **Section 24.6.2 "Master Clock Synchronization"**.

When the software is ready to resume reception, the software sets SCLREL. This causes the slave module to release the SCLx line, and the master resumes clocking.

24.7.4.3 Example Messages of Slave Reception

Receiving a slave message is a rather automatic process. The software handling the slave protocol uses the slave interrupt to synchronize to the events.

When the slave detects the valid address, the associated interrupt will notify the software to expect a message. On receive data, as each byte transfers to the I2CxRCV register, an interrupt notifies the software to unload the buffer.

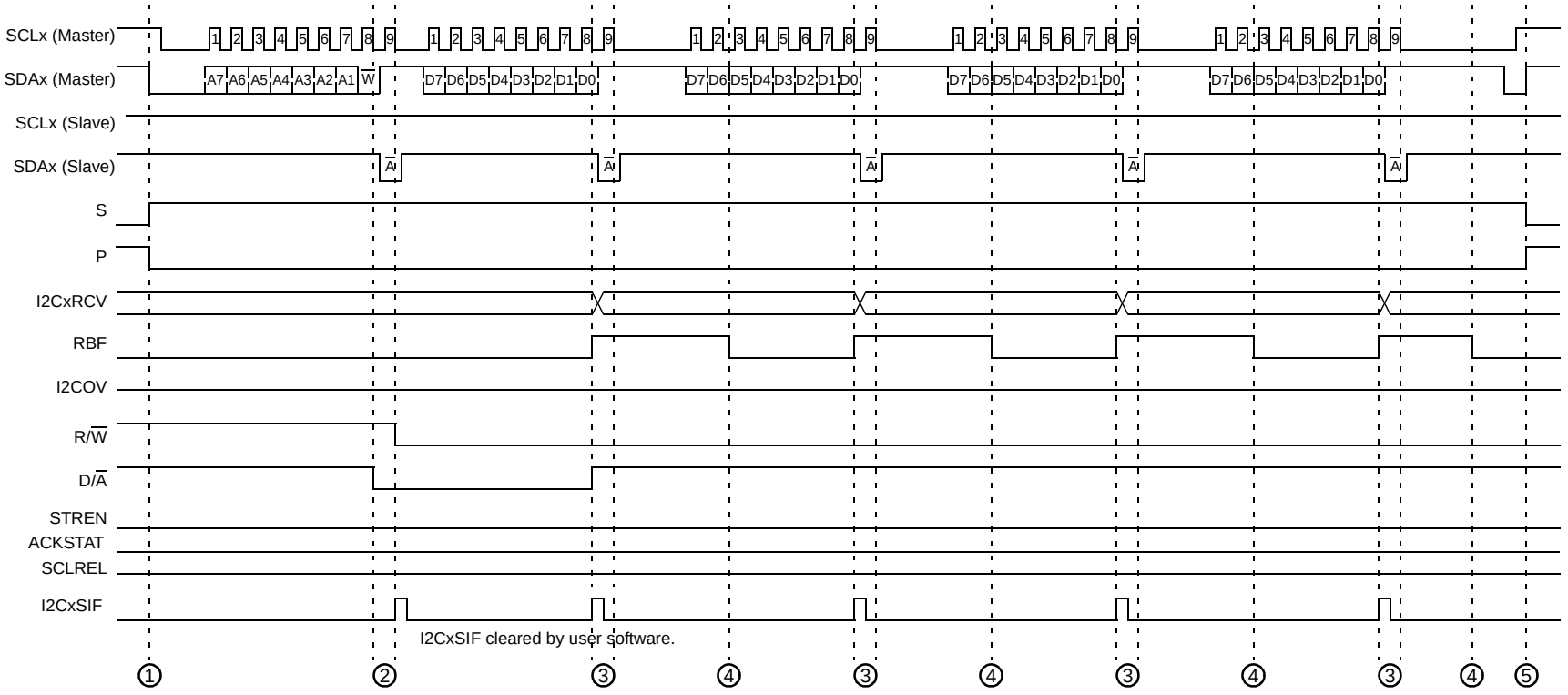
Figure 24-27 shows a simple receive message. Because it is a 7-bit address message, only one interrupt occurs for the address bytes. Then, interrupts occur for each of four data bytes. At an interrupt, the software may monitor the RBF, D/$\overline{A}$ and R/$\overline{W}$ bits to determine the condition of the byte received.

Figure 24-28 shows a similar message using a 10-bit address. In this case, two bytes are required for the address.

Figure 24-29 shows a case where the software does not respond to the received byte and the buffer overruns. On reception of the second byte, the module will automatically NACK the master transmission. Generally, this causes the master to re-send the previous byte. The I2COV bit indicates that the buffer has overrun. The I2CxRCV buffer retains the contents of the first byte. On reception of the third byte, the buffer is still full, and again, the module will NACK the master. After this, the software finally reads the buffer. Reading the buffer will clear the RBF bit, however, the I2COV bit remains set. The software must clear the I2COV bit. The next received byte will be moved to the I2CxRCV buffer and the module will respond with an $\overline{ACK}$.

Figure 24-30 highlights clock stretching while receiving data. Note in the previous examples, STREN = 0, which disables clock stretching on receive messages. In this example, the software sets STREN to enable clock stretching. When STREN = 1, the module will automatically clock stretch after each received data byte, allowing the software more time to move the data from the buffer. Note that if RBF = 1 at the falling edge of the 9th clock, the module will automatically clear the SCLREL bit and pull the SCLx bus line low. As shown with the second received data byte, if the software can read the buffer and clear the RBF before the falling edge of the 9th clock, the clock stretching will not occur. The software can also suspend the bus at any time. By clearing the SCLREL bit, the module will pull the SCLx line low after it detects the bus SCLx low. The SCLx line will remain low, suspending transactions on the bus until the SCLREL bit is set.

Figure 24-27: Slave Message (Write Data to Slave: 7-Bit Address; Address Matches; A10M = 0; GCEN = 0; STRICT = 0)



① Slave recognizes Start event; S and P bits set/clear accordingly.

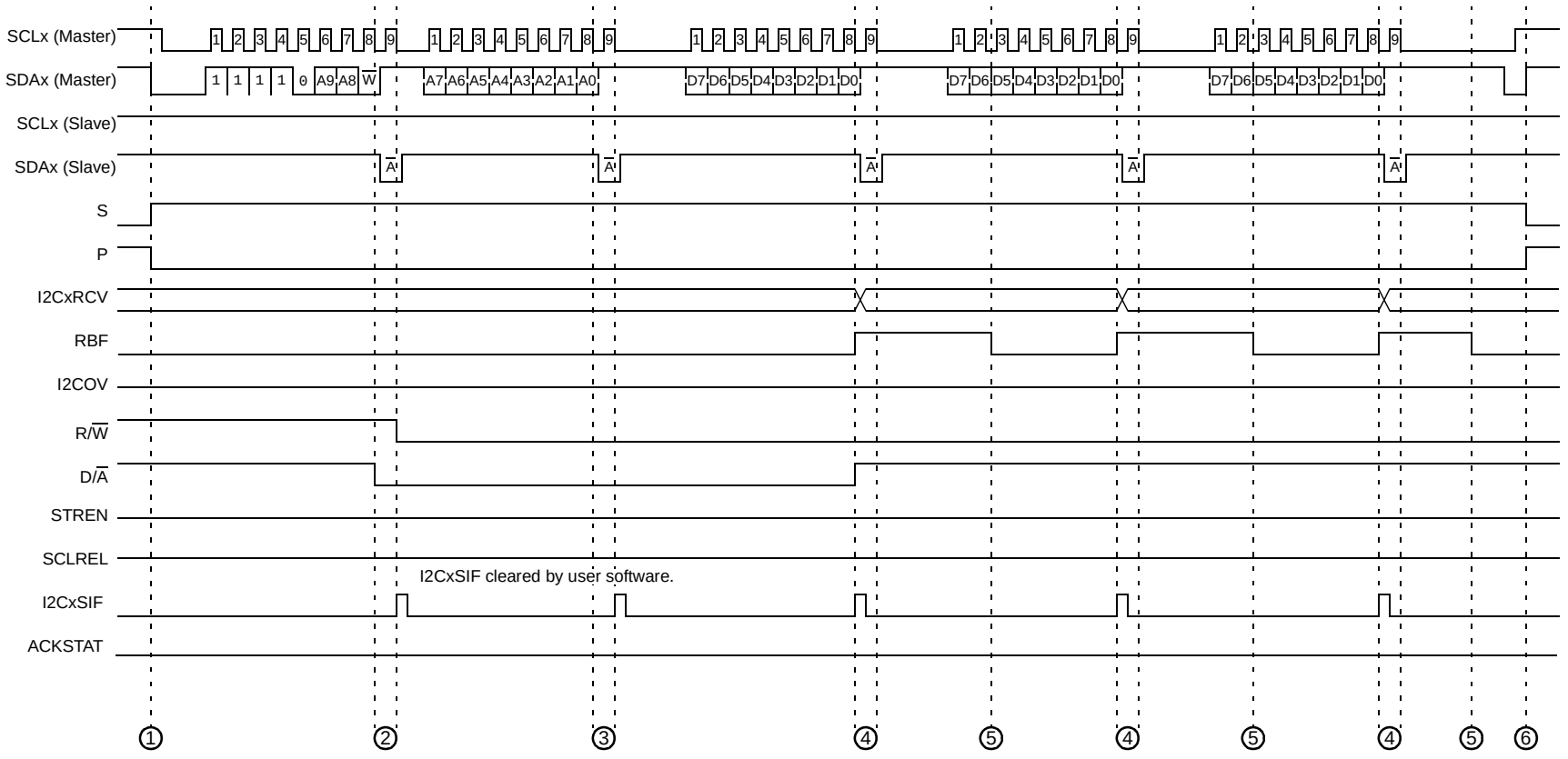
② Slave receives address byte. Address matches. Slave Acknowledges and generates interrupt. Address byte is moved to I2CxRCV register and must be read by user software to prevent buffer overflow.

③ Next received byte is message data. Byte moved to I2CxRCV register sets RBF. Slave generates interrupt. Slave Acknowledges reception.

④ Software reads I2CxRCV register. RBF bit clears.

⑤ Slave recognizes Stop event; S and P bits set/clear accordingly.

Figure 24-28: Slave Message (Write Data to Slave: 10-Bit Address; Address Matches; A10M = 1; GCEN = 0; STRICT = 0)



① Slave recognizes Start event; S and P bits set/clear accordingly.

② Slave receives address byte. High-order address matches. Slave Acknowledges and generates interrupt. Address byte not moved to I2CxRCV register.

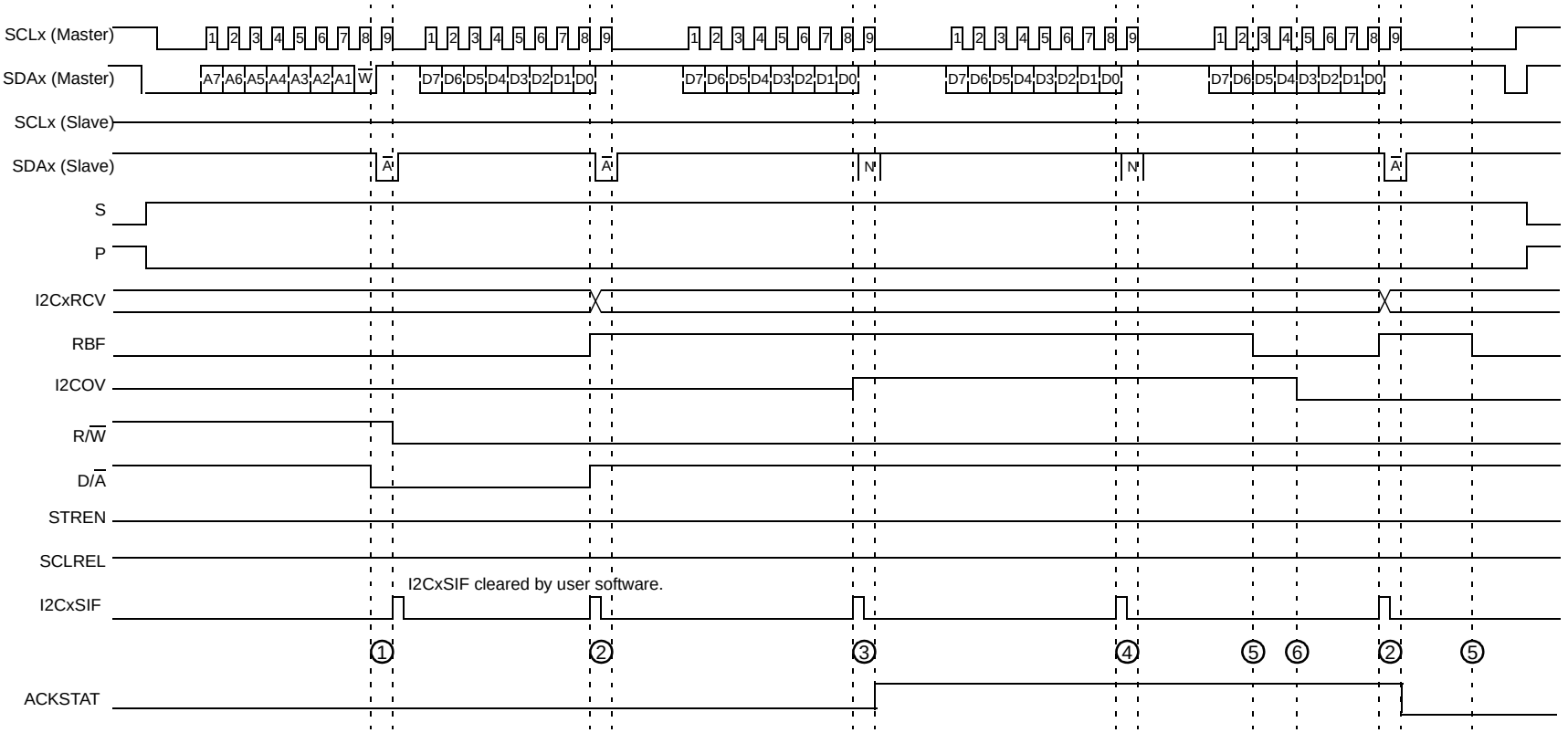
③ Slave receives address byte. Low-order address matches. Slave Acknowledges and generates interrupt. Address byte not moved to I2CxRCV register.

④ Next received byte is message data. Byte moved to I2CxRCV register sets RBF. Slave Acknowledges and generates interrupt.

⑤ Software reads I2CxRCV register. RBF bit clears.

⑥ Slave recognizes Stop event; S and P bits set/clear accordingly.

Figure 24-29: Slave Message (Write Data to Slave: 7-Bit Address; Buffer Overrun; A10M = 0; GCEN = 0; STRICT = 0)



① Slave receives address byte. Address matches. Slave generates interrupt. Address byte not moved to I2CxRCV register.

② Next received byte is message data. Byte moved to I2CxRCV register sets RBF. Slave generates interrupt. Slave Acknowledges reception.

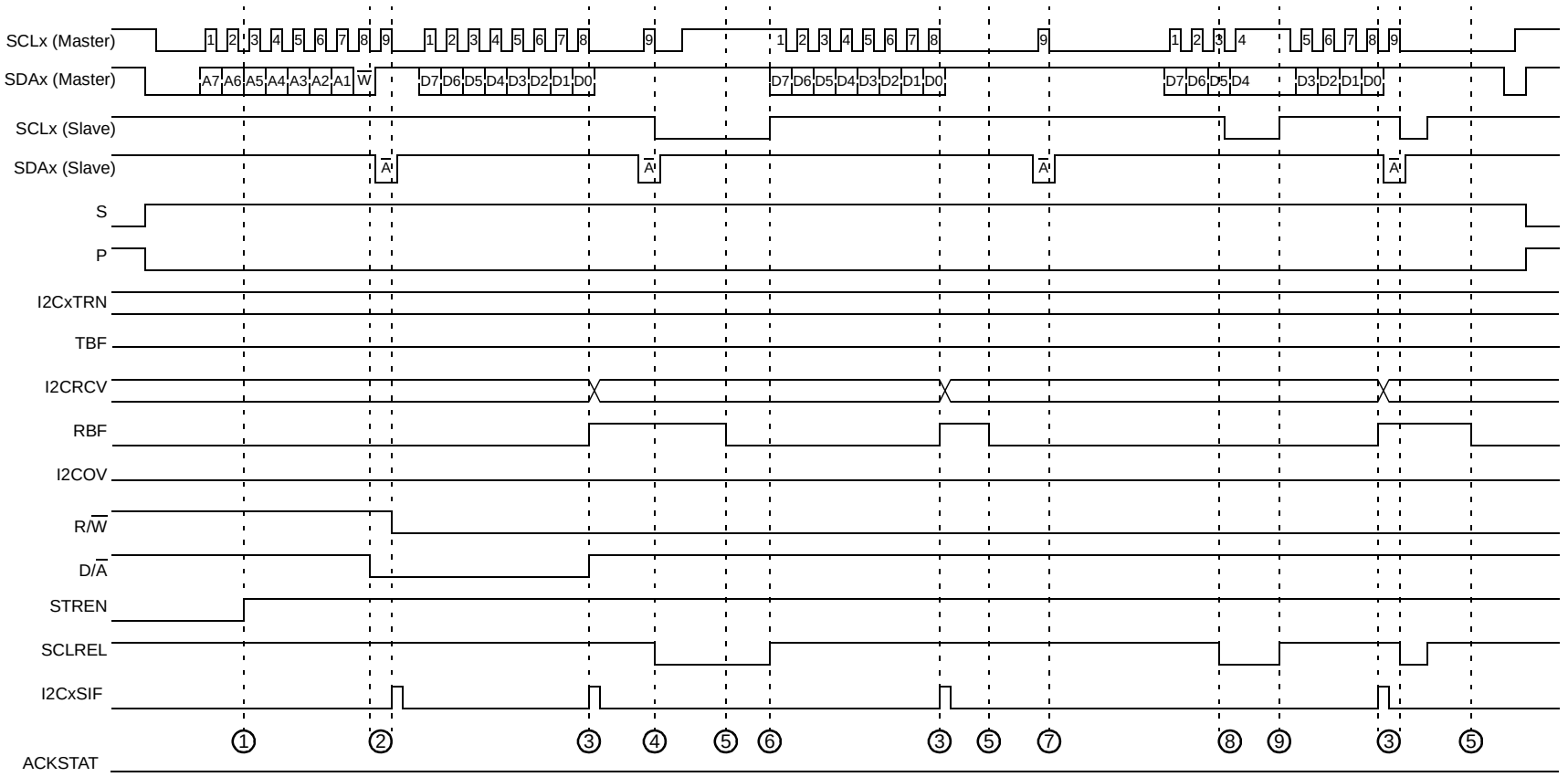
③ Next byte received before I2CxRCV read by software. I2CxRCV register unchanged. I2COV overflow bit set. Slave generates interrupt. Slave sends NACK for reception.

④ Next byte also received before I2CxRCV read by software. I2CxRCV register unchanged. Slave generates interrupt. Slave sends NACK for reception.

⑥ Software reads I2CxRCV register. RBF bit clears.

⑦ Software clears I2COV bit.

Figure 24-30: Slave Message (Write Data to Slave: 7-Bit Address; Clock Stretching Enabled; A10M = 0; GCEN = 0; STRICT = 0)



① Software sets the STREN bit to enable clock stretching.

② Slave receives address byte.

③ Next received byte is message data. Byte moved to I2CxRCV register sets RBF.

④ Because RBF = 1 at 9th clock, automatic clock stretch begins. Slave clears SCLREL bit. Slave pulls SCLx line low to stretch clock.

⑤ Software reads I2CxRCV register. RBF bit clears.

⑥ Software sets SCLREL bit to release clock.

⑦ Slave does not clear SCLREL because RBF = 0 at this time.

⑧ Software may clear SCLREL to cause a clock hold. Module must detect SCLx low before asserting SCLx low.

⑨ Software may set SCLREL to release a clock hold.

24.7.5 Sending Data to a Master Device

When the R/$\overline{W}$ bit of the incoming device address byte is '1' and an address match occurs, the R/$\overline{W}$ bit (I2CxSTAT<2>) is set. At this point, the master device is expecting the slave to respond by sending a byte of data. The contents of the byte are defined by the system protocol and are only transmitted by the slave module.

When the interrupt from the address detection occurs, the software can write a byte to the I2CxTRN register to start the data transmission.

The slave module sets the TBF bit. The eight data bits are shifted out on the falling edge of the SCLx input. This ensures that the SDAx signal is valid during the SCLx high time. When all eight bits have been shifted out, the TBF bit will be cleared.

The slave module detects the Acknowledge from the master-receiver on the rising edge of the ninth SCLx clock.

If the SDAx line is low, indicating an Acknowledge ($\overline{ACK}$), the master is expecting more data and the message is not complete. The module generates a slave interrupt to signal more data is requested.

A slave interrupt is generated on the falling edge of the ninth SCLx clock. Software must check the status of the I2CxSTAT register and clear the I2CxSIF flag.

If the SDAx line is high, indicating a Not Acknowledge (NACK), then the data transfer is complete. The slave module resets and does not generate an interrupt. The slave module will wait for detection of the next Start bit.

24.7.5.1 Wait States During Slave Transmissions

During a slave transmission message, the master expects return data immediately after detection of the valid address with R/$\overline{W}$ = 1. Because of this, the slave module will automatically generate a bus wait whenever the slave returns data.

The automatic wait occurs at the falling edge of the 9th SCLx clock of a valid device address byte or transmitted byte Acknowledged by the master, indicating expectation of more transmit data.

The slave module clears the SCLREL bit. Clearing the SCLREL bit causes the slave module to pull the SCLx line low, initiating a wait. The SCLx clock of the master and slave will synchronize as shown in **Section 24.6.2 "Master Clock Synchronization"**.

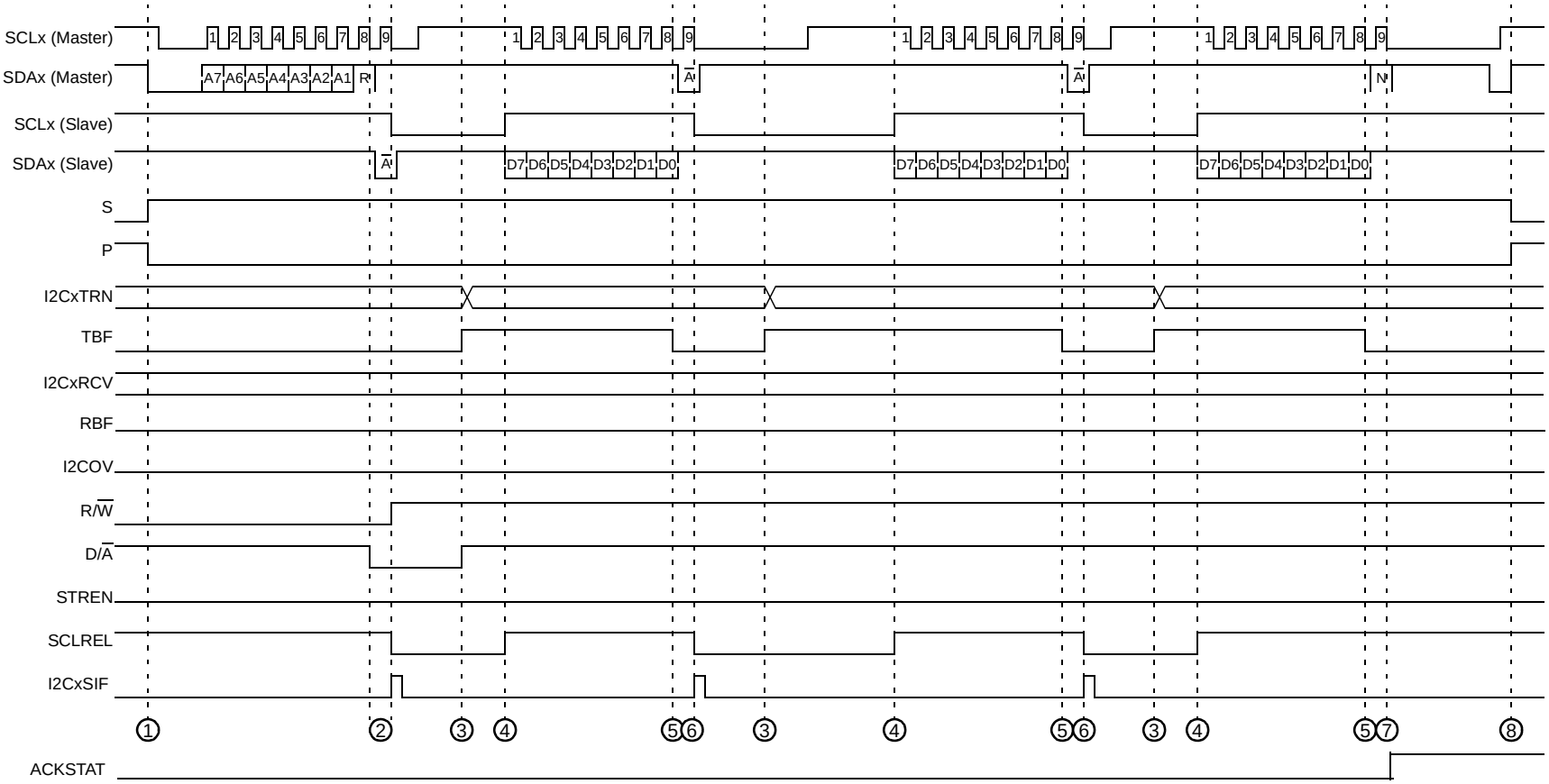
When the software loads the I2CxTRN and is ready to resume transmission, the software sets SCLREL. This causes the slave module to release the SCLx line and the master resumes clocking.

24.7.5.2 Example Messages of Slave Transmission

Slave transmissions for 7-bit address messages are shown in Figure 24-31. When the address matches and the R/$\overline{W}$ bit of the address indicates a slave transmission, the module will automatically initiate clock stretching by clearing the SCLREL bit and generates an interrupt to indicate a response byte is required. The software will write the response byte into the I2CxTRN register. As the transmission completes, the master will respond with an Acknowledge. If the master replies with an $\overline{ACK}$, the master expects more data and the module will again clear the SCLREL bit and generate another interrupt. If the master responds with a NACK, no more data is required and the module will not stretch the clock nor generate an interrupt.

Slave transmissions for 10-bit address messages require the slave to first recognize a 10-bit address. Because the master must send two bytes for the address, the R/$\overline{W}$ bit in the first byte of the address specifies a write. To change the message to a read, the master will send a Repeated Start and repeat the first byte of the address with the R/$\overline{W}$ bit specifying a read. At this point, the slave transmission begins as shown in Figure 24-32.

Figure 24-31: Slave Message (Read Data From Slave: 7-Bit Address)



① Slave recognizes Start event; S and P bits set/clear accordingly.

② Slave receives address byte. Address matches. Slave generates interrupt. Address byte not moved to I2CxRCV register. R/W = 1 to indicate read from slave. SCLREL = 0 to suspend master clock.

③ Software writes I2CxTRN with response data. TBF = 1 indicates that buffer is full. Writing I2CxTRN sets D/A, indicating data byte.

④ Software sets SCLREL to release clock hold. Master resumes clocking and slave transmits data byte.

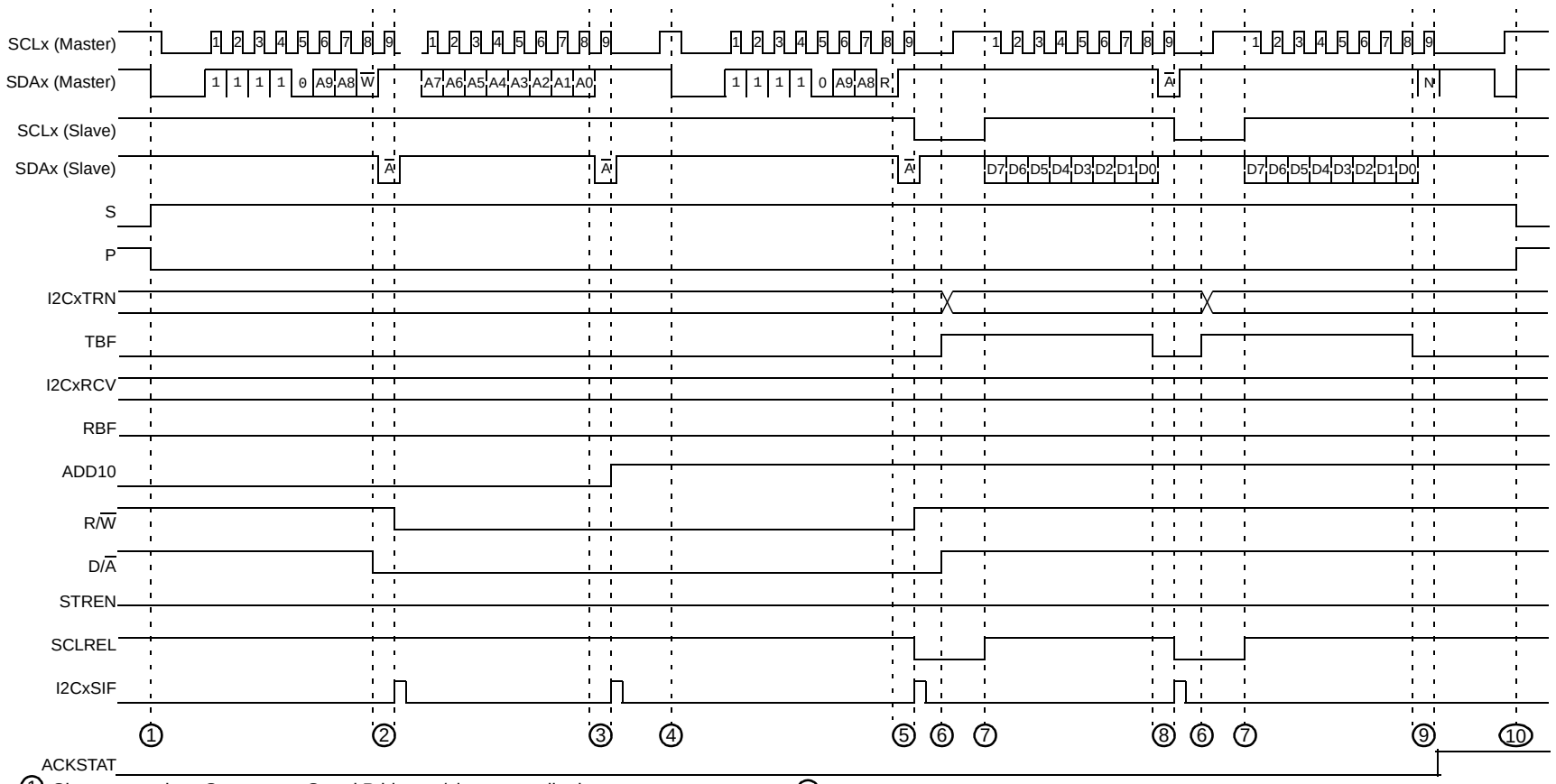
⑤ After last bit, module clears TBF bit, indicating buffer is available for next byte.

⑥ At end of 9th clock, if master sent ACK, module clears SCLREL to suspend clock. Slave generates interrupt.

⑦ At end of 9th clock, if master sent NACK, no more data expected. Module does not suspend clock and will generate an interrupt.

⑧ Slave recognizes Stop event; S and P bits set/clear accordingly.

Figure 24-32: Slave Message (Read Data From Slave: 10-Bit Address)



1. Slave recognizes Start event; S and P bits set/clear accordingly.
2. Slave receives first address byte. Write indicated. Slave Acknowledges and generates interrupt.
3. Slave receives address byte. Address matches. Slave Acknowledges and generates interrupt.
4. Master sends a Repeated Start to redirect the message.
5. Slave receives re-send of first address byte. Read indicated. Slave suspends clock.
6. Software writes I2CxTRN with response data.
7. Software sets SCLREL to release clock hold. Master resumes clocking and slave transmits data byte.
8. At end of 9th clock, if master sent $\overline{\text{ACK}}$, module clears SCLREL to suspend clock. Slave generates interrupt.
9. At end of 9th clock, if master sent NACK, no more data expected. Module does not suspend clock or generate interrupt.
10. Slave recognizes Stop event; S and P bits set/clear accordingly.

24.8 CONNECTION CONSIDERATIONS FOR I²C BUS

Because the I²C bus is a wired AND bus connection, pull-up resistors on the bus are required, shown as RP in Figure 24-33. Series resistors, shown as RS, are optional and used to improve ESD susceptibility. The values of resistors, RP and RS, depend on the following parameters:

- Supply voltage
- Bus capacitance
- Number of connected devices (input current + leakage current)
- Input level selection (I²C or SMBus)

To get accurate SCK clock, the rise time should be as small as possible. The limitation factor is the maximum current sink available on the SCK pad. The following example calculates the Rp min based on 3.3V supply and 6.6 mA sink current at VOLMAX = 0.4V:

Equation 24-2:

$$R_{PMIN} = (V_{DDMAX} - V_{OLMAX})/I_{OL} = (3.3V - 0.4V)/6.6\ mA = 439\Omega$$

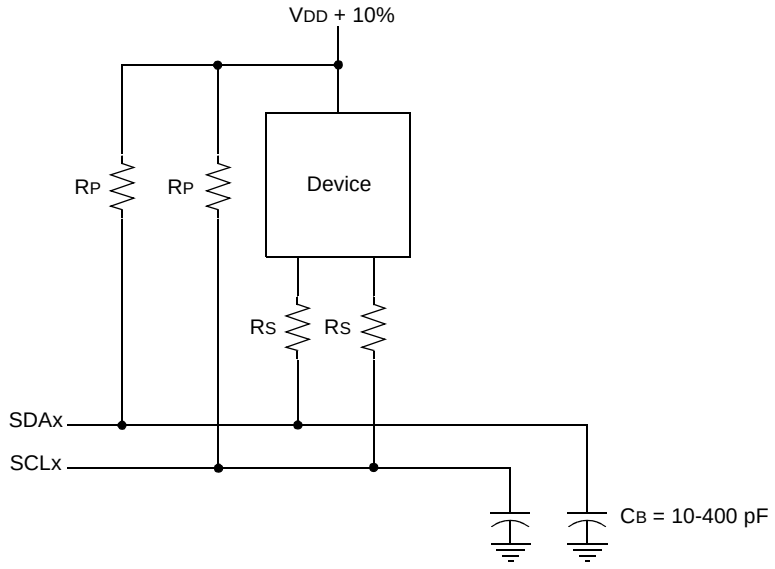
The maximum value for RS is determined by the desired noise margin for the low level. RS cannot drop enough voltage to make the device VOL plus the voltage across RS more than the maximum VIL. Mathematically:

Equation 24-3:

$$R_{SMAX} = (V_{ILMAX} - V_{OLMAX})/I_{OLMAX} = (0.3\ V_{DD} - 0.4)/6.6\ mA = 89\Omega$$

The SCLx clock input must have a minimum high and low time for proper operation. The high and low times of the I²C specification, as well as the requirements of the I²C module, are shown in the Electrical Characteristics section in the device data sheet (DS61143).

Figure 24-33: Sample Device Configuration for I²C Bus



Note: I²C™ devices with input levels related to VDD must have one common supply line to which the pull-up resistor is also connected.

24.8.1 Integrated Signal Conditioning and Slope Control

The SCLx and SDAx pins have an input glitch filter. The I^2C bus requires this filter in both the 100 kHz and 400 kHz systems.

When operating on a 400 kHz bus, the I^2C specification requires a slew rate control of the device pin output. This slew rate control is integrated into the device. If the DISSLW bit (I2CxCON<9>) is cleared, the slew rate control is active. For other bus speeds, the I^2C specification does not require slew rate control and DISSLW should be set.

Some system implementations of I^2C busses require different input levels for V_{ILMAX} and V_{IHMIN}. In a normal I^2C system, V_{ILMAX} is 0.3 V_{DD}; V_{IHMIN} is 0.7 V_{DD}. By contrast, in an SMBus (System Management Bus) system, V_{ILMAX} is set at 0.8V, while V_{IHMIN} is set at 2.1V.

The SMEN bit (I2CxCON<8>) controls the input levels. Setting SMEN (= 1) changes the input levels to SMBus specifications.

24.9 I²C™ OPERATION IN POWER-SAVE MODES AND DEBUG MODES

Note:	In this manual, a distinction is made between a power mode as it is used in a specific module, and a power mode as it is used by the device, e.g., Sleep mode of the Comparator and SLEEP mode of the CPU. To indicate which type of power mode is intended, uppercase and lowercase letters (Sleep, Idle, Debug) signify a module power mode, and all uppercase letters (SLEEP, IDLE, DEBUG) signify a device power mode.

PIC32MX based devices have two Power-Saving modes:

- IDLE mode: core and selected peripherals are shut down
- SLEEP mode: entire device is shut down

24.9.1 SLEEP in Master Mode Operation

When the device enters SLEEP mode, all clock sources to the module are shut down. The Baud Rate Generator stops because the clocks stop. It may have to be reset to prevent partial clock detection.

If SLEEP occurs in the middle of a transmission, and the master state machine is partially into a transmission as the clocks stop, the Master mode transmission is aborted.

There is no automatic way to prevent SLEEP entry if a transmission or reception is pending. The user software must synchronize SLEEP entry with I²C operation to avoid aborted transmissions.

Register contents are not affected by going into SLEEP mode or coming out of SLEEP mode.

24.9.2 SLEEP in Slave Mode Operation

The I²C module can still function in Slave mode operation while the device is in SLEEP.

When operating in Slave mode and the device is put into SLEEP, the master-generated clock will run the slave state machine. This feature provides an interrupt to the device upon reception of the address match in order to wake up the device.

Register contents are not affected by going into SLEEP mode or coming out of SLEEP mode.

It is an error condition to set SLEEP in the middle of a slave data transmit operation; indeterminate results may occur.

24.9.3 IDLE Mode

When the device enters IDLE mode, all PBCLK clock sources remain functional. If the module intends to power down, it disables its own clocks.

For the I²C, the I2CxSIDL bit (I2CxCON<13>) selects whether the module will stop on IDLE or continue on IDLE. If I2CxSIDL = 0, the module will continue operation in IDLE mode. If I2CxSIDL = 1, the module will stop on IDLE.

The I²C module will perform the same procedures for stop on IDLE mode as for SLEEP mode. The module state machines must be reset.

24.9.4 Operation in DEBUG Mode

The FRZ bit (I2CxCON<14>) determines whether the I²C module will run or stop while the CPU is executing Debug Exception code (i.e., the application is halted) in DEBUG mode. When FRZ = 0, the I²C module continues to run even when the application is halted in DEBUG mode. When FRZ = 1 and the application is halted in DEBUG mode, the module will freeze its operations and make no changes to the state of the I²C module. The module will resume its operation after the CPU resumes execution.

Note:	The FRZ bit is readable and writable only when the CPU is executing in Debug Exception mode. In all other modes, the FRZ bit reads as '0'. If FRZ bit is changed during DEBUG mode, the new value does not take effect until the current Debug Exception mode is exited and re-entered. During the Debug Exception mode, the FRZ bit reads the state of the peripheral when entering DEBUG mode.

24.10 EFFECTS OF A RESET

A Reset (POR, WDT, etc.) disables the I^2C module and terminates any active or pending message activity. See the register definitions of I2CxCON and I2CxSTAT for the Reset conditions of those registers.

Note:	In this discussion, 'IDLE' refers to the CPU power-saving state. The lower case 'idle' refers to the time when the I^2C module is not transferring data on the bus.

24.11 PIN CONFIGURATION IN I^2C MODE

In I^2C mode, pin SCL is clock and pin SDA is data. The module will override the data direction bits (TRIS bits) for these pins. The pins that are used for I^2C modes are configured as open drain. Table 24-5 lists the pin usage in different modes.

Table 24-5: Required IO Pin Resources

IO Pin Name	Master Mode	Slave Mode
SDAx	Yes	Yes
SCLx	Yes	Yes

Note: "No" indicates the pin is not required and can be used as a general purpose IO pin.

24.12 DESIGN TIPS

Question 1: ***I'm operating as a bus master and transmitting data. Why do slave and receive interrupts keep occurring at the same time?***

Answer: The master and slave circuits are independent. The slave module will receive events from the bus sent by the master.

Question 2: ***I'm operating as a slave and I write data to the I2CxTRN register. Why isn't the data being transmitted?***

Answer: The slave enters an automatic wait when preparing to transmit. Ensure that you set the SCLREL bit to release the I^2C clock.

Question 3: ***How do I tell what state the master module is in?***

Answer: Looking at the condition of the SEN, RSEN, PEN, RCEN, ACKEN and TRSTAT bits will indicate the state of the master module. If all bits are '0', the module is IDLE.

Question 4: ***Operating as a slave, I receive a byte while STREN = 0. What should the software do if it cannot process the byte before the next one is received?***

Answer: Because STREN was '0', the module did not generate an automatic wait on the received byte. However, the software may, at any time during the message, set STREN and then clear SCLREL. This will cause a wait on the next opportunity to synchronize the SCLx clock.

Question 5: ***My I^2C™ system is a multi-master system. Why are my messages being corrupted when I attempt to send them?***

Answer: In a multi-master system, other masters may cause bus collisions. In the Interrupt Service Routine for the master, check the BCL bit to ensure that the operation completed without a collision. If a collision is detected, the message must be re-sent from the beginning.

Question 6: ***My I^2C™ system is a multi-master system. How can I tell when it is OK to begin a message?***

Answer: Look at the S bit. If S = 0, the bus is Idle.

Question 7: ***I tried to send a Start condition on the bus, then transmit a byte by writing to the I2CxTRN register. The byte did not get transmitted. Why?***

Answer: You must wait for each event on the I^2C bus to complete before starting the next one. In this case, you should poll the SEN bit to determine when the Start event completed or wait for the master I^2C interrupt before data is written to I2CxTRN.

24.13 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the PIC32MX device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the I^2C module include the following:

Title	Application Note #
Use of the SSP Module in the I ^{2}C™ Multi-Master Environment	AN578
Using the PIC® Microcontroller SSP for Slave I^2C™ Communication	AN734
Using the PIC® Microcontroller MSSP Module for Master I^2C™ Communications	AN735
An I^2C™ Network Protocol for Environmental Monitoring	AN736

Note: Please visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the PIC32MX family of devices.

24.14 REVISION HISTORY

Revision A (October 2007)

This is the initial released version of this document.

Revision B (October 2007)

Updated document to remove Confidential status.

Revision C (April 2008)

Revised status to Preliminary; Revised U-0 to r-x.

Revision D (July 2008)

Revised Figure 24-1; Section 24.2 (I2CxMIF); Register 24-1, bits 13 and 14; Revised Register 24-26-24-29; Revised Table 24-1, I2CxCON; Change Reserved bits from “Maintain as” to “Write”; Added Note to ON bit (I2CxCON Register); Delete Section 24.12 (Electrical Characteristics).

NOTES:

Section 25. Reserved for Future

NOTES:

Section 26. Reserved for Future

NOTES:

Section 27. USB On-The-Go

HIGHLIGHTS

This section of the manual contains the following major topics:

27.1 INTRODUCTION

The PIC32MX USB module includes the following features:

- USB Full-Speed Support for Host and Device
- Low-Speed Host Support
- USB On-The-Go (OTG) Support
- Integrated Signaling Resistors
- Integrated Analog Comparators for VBUS Monitoring
- Integrated USB Transceiver
- Transaction Handshaking Performed by Hardware
- Endpoint Buffering Anywhere in System RAM
- Integrated DMA Controller to Access System RAM and Flash

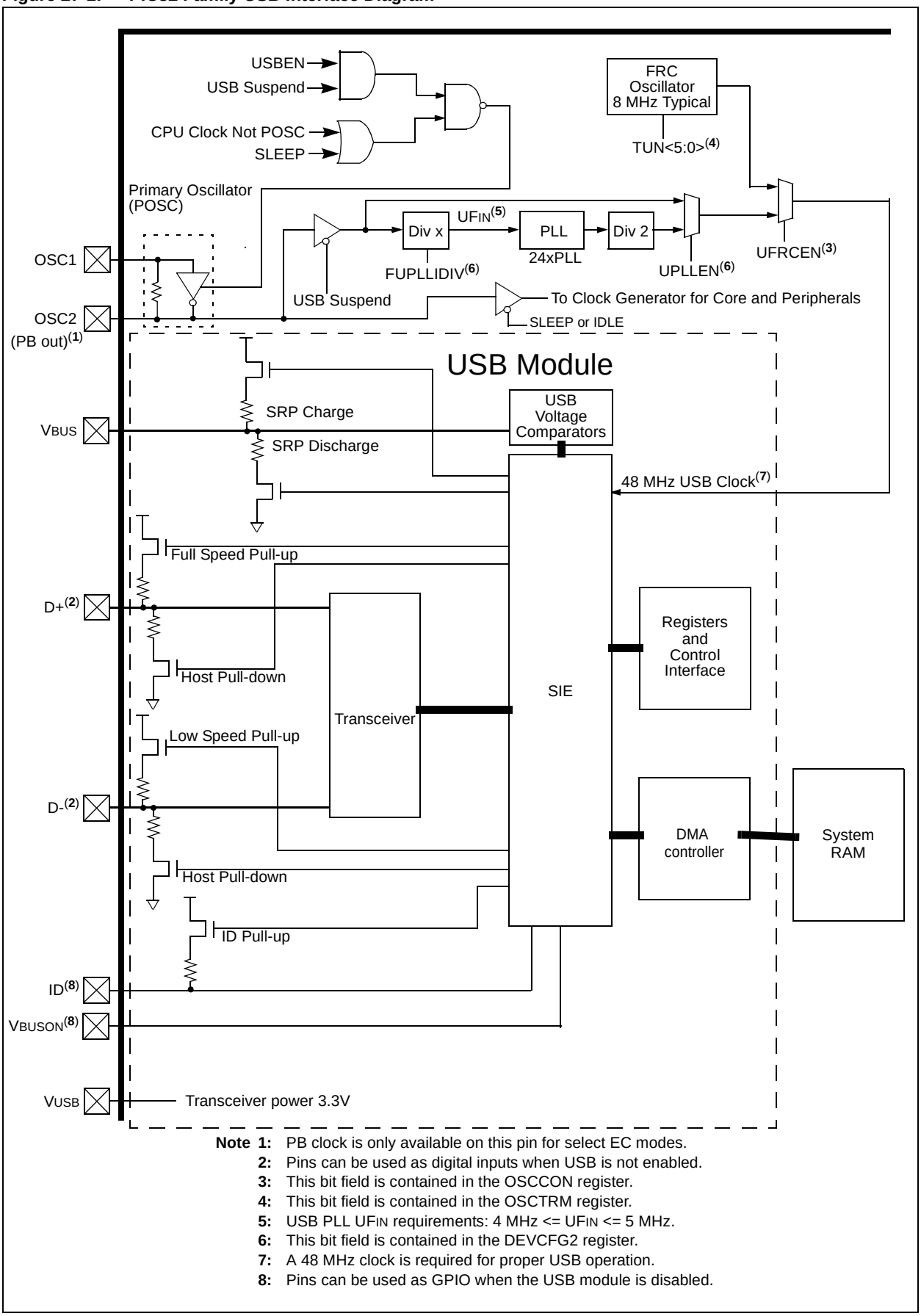
The Universal Serial Bus (USB) module contains analog and digital components to provide a USB 2.0 full-speed and low-speed embedded host, full-speed device, or OTG implementation with a minimum of external components. This module in Host mode is intended for use as an embedded host and therefore does not implement a UHCI or OHCI controller.

The USB module consists of the clock generator, the USB voltage comparators, the transceiver, the Serial Interface Engine (SIE), a dedicated USB DMA controller, pull-up and pull-down resistors, and the register interface. A block diagram of the PIC32MX USB OTG module is presented in Figure 27-1.

The clock generator provides the 48 MHz clock required for USB full speed and low speed communication. The voltage comparators monitor the voltage on the VBUS pin to determine the state of the bus. The transceiver provides the analog translation between the USB bus and the digital logic. The SIE is a state machine that transfers data to and from the endpoint buffers, and generates the hardware protocol for data transfers. The USB DMA controller transfers data between the data buffers in RAM and the SIE. The integrated pull-up and pull-down resistors eliminate the need for external signaling components. The register interface allows the CPU to configure and communicate with the module.

IMPORTANT:	The implementation and use of the USB specifications, as well as other third-party specifications or technologies, may require licensing; including, but not limited to, USB Implementers Forum, Inc. (also referred to as USB-IF). The user is fully responsible for investigating and satisfying any applicable licensing obligations.

Figure 27-1: PIC32 Family USB Interface Diagram



Note 1: PB clock is only available on this pin for select EC modes.
2: Pins can be used as digital inputs when USB is not enabled.
3: This bit field is contained in the OSCCON register.
4: This bit field is contained in the OSCTRM register.
5: USB PLL UFIN requirements: 4 MHz <= UFIN <= 5 MHz.
6: This bit field is contained in the DEVCFG2 register.
7: A 48 MHz clock is required for proper USB operation.
8: Pins can be used as GPIO when the USB module is disabled.

27.2 CONTROL REGISTERS

The USB module includes the following Special Function Registers (SFRs):

- U1OTGIR: USB OTG Interrupt Flags Register
- U1OTGIE: USB OTG Interrupt Enable Register
- U1OTGSTAT: USB Comparator and Pin Status Register
- U1OTGCON: USB Resistor and Pin Control Register
- U1PWRC: USB Power Control Register
- U1IR: USB Pending Interrupt Register
- U1IE: USB Interrupt Enable Register
- U1EIR: USB Pending Error Interrupt Register
- U1EIE: USB Interrupt Enable Register
- U1STAT: USB Status FIFO Register
- U1CON: USB Module Control Register
- U1ADDR: USB Address Register
- U1FRMH and U1FRML: USB Frame Counter Registers
- U1TOK: USB Host Control Register
- U1SOF: USB SOF Counter Register
- U1BDTP1, U1BDTP2, and U1BDTP3: USB Buffer Descriptor Table Pointer Register
- U1CNFG1: USB Debug and Idle Register
- U1EP0-U1EP15: USB Endpoint Control Register

27.2.1 U1OTGIR Register

U1OTGIR (Register 27-1) records changes on the ID, data, and VBUS pins, enabling software to determine which event caused an interrupt. The interrupt bits are cleared by writing a '1' to the corresponding interrupt.

27.2.2 U1OTGIE Register

U1OTGIE (Register 27-2) enables the corresponding interrupt status bits defined in the U1OTGIR register to generate an interrupt.

27.2.3 U1OTGSTAT Register

U1OTGSTAT (Register 27-3) provides access to the status of the VBUS voltage comparators and the debounced status of the ID pin.

27.2.4 U1OTGCON Register

U1OTGCON (Register 27-4) controls the operation of the VBUS pin, and the pull-up and pull-down resistors.

27.2.5 U1PWRC Register

U1PWRC (Register 27-5) controls the power-saving modes, as well as the module enable/disable control.

27.2.6 U1IR Register

U1IR (Register 27-6) contains information on pending interrupts. Once an interrupt bit is set, it can be cleared by writing a '1' to the corresponding bit.

27.2.7 U1IE Register

U1IE (Register 27-7) values provide gating of the various interrupt signals onto the USB interrupt signal. These values do not interact with the USB module. Setting any of these bits enables the corresponding interrupt source in the U1IR register.

27.2.8 U1EIR Register

U1EIR (Register 27-8) contains information on pending error interrupt values. Once an interrupt bit is set, it can be cleared by writing a '1' to the corresponding bit.

27.2.9 U1EIE Register

U1EIE (Register 27-9) values provide gating of the various interrupt signals onto the USB interrupt signal. These values do not interact with the USB module. Setting any of these bits enables the respective interrupt source in the U1EIR register, if UERR is also set in the U1IE register.

27.2.10 U1STAT Register

U1STAT (Register 27-10) is a 16-deep First In, First Out (FIFO) register. It is read-only by the CPU and read/write by the USB module. U1STAT is only valid when the U1IR<TRNIF> bit is set.

27.2.11 U1CON Register

U1CON (Register 27-11) provides miscellaneous control and information about the module.

27.2.12 U1ADDR Register

U1ADDR (Register 27-12) is a read/write register from the CPU side and read-only from the USB module side. Although the register values affect the settings of the USB module, the content of the registers does not change during access.

In Device mode, this address defines the USB device address as assigned by the host during the SETUP phase. The firmware writes the address in response to the SETUP request. The address is automatically reset when a USB bus Reset is detected. In Host mode, the module transmits the address provided in this register with the corresponding token packet. This allows the USB module to uniquely address the connected device.

27.2.13 U1FRMH and U1FRML Registers

U1FRMH and U1FRML (Register 27-13 and Register 27-14) are read-only registers. The frame number is formed by concatenating the two 8-bit registers. The high-order byte is in the U1FRMH register, and the low-order byte is in U1FRML.

27.2.14 U1TOK Register

U1TOK (Register 27-15) is a read/write register required when the module operates as a host. It is used to specify the token type, PID<3:0> (Packet ID), and the endpoint, EP<3:0>, being addressed by the host processor. Writing to this register triggers a host transaction.

27.2.15 U1SOF Register

U1SOF (Register 27-16) threshold is a read/write register that contains the count bits of the Start-of-Frame (SOF) threshold value, and are used in Host mode only.

To prevent colliding a packet data with the SOF token that is sent every 1 ms, the USB module will not send any new transactions within the last U1SOF byte times. The USB module will complete any transactions that are in progress. In Host mode, the SOF interrupt occurs when this threshold is reached, not when the SOF occurs. In Device mode, the interrupt occurs when a SOF is received. Transactions started within the SOF threshold are held by the USB module until after the SOF token is sent.

27.2.16 U1BDTP1, U1BDTP2, and U1BDTP3

These registers (Register 27-17, Register 27-18, and Register 27-19) are read/write registers that define the upper 23 bits of the 32-bit base address of the Buffer Descriptor Table (BDT) in the system memory. The BDT is forced to be 512 byte-aligned. This register allows relocation of the BDT in real time.

27.2.17 U1CNFG1 Register

U1CNFG1 (Register 27-20) is a read/write register that controls the Debug and Idle behavior of the module. The register must be preprogrammed prior to enabling the module.

27.2.18 U1EP0 - U1EP15

These registers control the behavior of the corresponding endpoint.

27.2.19 Associated Registers

The following registers are not part of the USB module but are associated with module operation.

- OSCCON: Oscillator Control Register (Register 27-22)
- IFS1: Interrupt Flag Status Register (Register 27-23)
- IEC1: Interrupt Enable Control Register (Register 27-24)
- DEVCFG2: Device Configuration Control Register (Register 27-25)

27.2.20 Clearing USB OTG Interrupts

Unlike other device-level interrupts, the USB OTG interrupt status flags are not freely writable in software. All USB OTG flag bits are implemented as hardware-set-only bits. These bits can only be cleared in software by writing a '1' to their locations. Writing a '0' to a flag bit has no effect.

Note:	Throughout this section, a bit that can only be cleared by writing a '1' to its location is referred to as "Write '1' to clear bit". In register descriptions, this function is indicated by the descriptor 'K'.

Table 27-1: USB Register Summary

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
U1OTGIR	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	IDIF	T1MSECIF	LSTATEIF	ACTVIF	SESVDIF	SESENDIF	—	VBUSVDIF
U1OTGIE	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	IDIE	T1MSECIE	LSTATEIE	ACTVIE	SESVDIE	SESENDIE	—	VBUSVDIE
U1OTGSTAT	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	ID	—	LSTATE	—	SESVD	SESEND	—	VBUSVD
U1OTGCON	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	DPPULUP	DMPULUP	DPPULDWN	DMPULDWN	VBUSON	OTGEN	VBUSCHG	VBUSDIS
U1PWRC	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	UACTPND	—	—	USLPGRD	—	—	USUSPEND	USBPWR
U1IR	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	STALLIF	ATTACHIF	RESUMEIF	IDLEIF	TRNIF	SOFIF	UERRIF	URSTIF DETACHIF
U1IE	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	STALLIE	ATTACHIE	RESUME	IDLEIE	TRNIE	SOFIE	UERRIE	URSTIE DETACHIE
U1EIR	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	BTSEF	BMXEF	DMAEF	BTOEF	DFN8EF	CRC16EF	CRC5EF EOFEF	PIDEF
U1EIE	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	BTSEE	BMXEE	DMAEE	BTOEE	DFN8EE	CRC16EE	CRC5EE EOFEE	PIDEE
U1STAT	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	ENDPT<3:0>				DIR	PPBI	—	—

Table 27-1: USB Register Summary (Continued)

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
U1CON	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	JSTATE	SE0	PKTDIS TOKBUSY	USBRST	HOSTEN	RESUME	PPBRST	USBEN SOFEN
U1ADDR	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	LSPDEN	DEVADDR<6:0>						
U1FRML	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	FRML<7:0>							
U1FRMH	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	—	—	—	—	—	FRMH<2:8>		
U1TOK	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	PID<3:0>				EP<3:0>			
U1SOF	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	CNT<7:0>							
U1BDTP1	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	BDTPTRL<15:9>							—
U1BDTP2	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	BDTPTRH<23:16>							
U1BDTP3	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	BDTPTRU<31:24>							
U1CNFG1	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	UTEYE	UOEMON	USBFRZ	USBSIDL	—	—	—	—
U1EP0	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	LSPD	RETRYDIS	—	EPCONDIS	EPRXEN	EPTXEN	EPSTALL	EPHSHK

Table 27-1: USB Register Summary (Continued)

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
U1EP1	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	—	—	—	EPCONDIS	EPRXEN	EPTXEN	EPSTALL	EPHSHK
U1EP2	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	—	—	—	EPCONDIS	EPRXEN	EPTXEN	EPSTALL	EPHSHK
U1EP3	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	—	—	—	EPCONDIS	EPRXEN	EPTXEN	EPSTALL	EPHSHK
U1EP4	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	—	—	—	EPCONDIS	EPRXEN	EPTXEN	EPSTALL	EPHSHK
U1EP5	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	—	—	—	EPCONDIS	EPRXEN	EPTXEN	EPSTALL	EPHSHK
U1EP6	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	—	—	—	EPCONDIS	EPRXEN	EPTXEN	EPSTALL	EPHSHK
U1EP7	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	—	—	—	EPCONDIS	EPRXEN	EPTXEN	EPSTALL	EPHSHK
U1EP8	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	—	—	—	EPCONDIS	EPRXEN	EPTXEN	EPSTALL	EPHSHK
U1EP9	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	—	—	—	EPCONDIS	EPRXEN	EPTXEN	EPSTALL	EPHSHK
U1EP10	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	—	—	—	EPCONDIS	EPRXEN	EPTXEN	EPSTALL	EPHSHK
U1EP11	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	—	—	—	EPCONDIS	EPRXEN	EPTXEN	EPSTALL	EPHSHK

Table 27-1: USB Register Summary (Continued)

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
U1EP12	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	—	—	—	EPCONDIS	EPRXEN	EPTXEN	EPSTALL	EPHSHK
U1EP13	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	—	—	—	EPCONDIS	EPRXEN	EPTXEN	EPSTALL	EPHSHK
U1EP14	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	—	—	—	EPCONDIS	EPRXEN	EPTXEN	EPSTALL	EPHSHK
U1EP15	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	—	—	—	EPCONDIS	EPRXEN	EPTXEN	EPSTALL	EPHSHK
OSCCON	31:24	—	SOSCRDY	PLLODIV<2:0>			FRCDIV<2:0>		
	23:16	—	—	—	PBDIV<1:0>		PLLMULT<2:0>		
	15:8	—	COSC<2:0>			—	NOSC<2:0>		
	7:0	CLKLOCK	ULOCK	LOCK	SLPEN	CF	UFRCEN	SOSCEN	OSWEN
IFS1	31:24	—	—	—	—	—	—	USBIF	FCEIF
	23:16	—	—	—	—	DMA3IF	DMA2IF	DMA1IF	DMA0IF
	15:8	RTCCIF	FSCMIF	I2C2MIF	I2C2SIF	I2C2BIF	U2TXIF	U2RXIF	U2EIF
	7:0	SPI2RXIF	SPI2TXIF	SPI2EIF	CMP2IF	CMP1IF	PMPIF	AD1IF	CNIF
IEC1	31:24	—	—	—	—	—	—	USBIE	FCEIE
	23:16	—	—	—	—	DMA3IE	DMA2IE	DMA1IE	DMA0IE
	15:8	RTCCIE	FSCMIE	I2C2MIE	I2C2SIE	I2C2BIE	U2TXIE	U2RXIE	U2EIE
	7:0	SPI2RXIE	SPI2TXIE	SPI2EIE	CMP2IE	CMP1IE	PMPIE	AD1IE	CNIE
DEVCFG2	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	FPLLODIV<2:0>		
	15:8	FUPLLEN	—	—	—	—	FUPLLIDIV<2:0>		
	7:0	—	FPLLMULT<2:0>				FPLLIDIV<2:0>		

Register 27-1: U1OTGIR: USB OTG Interrupt Status Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R/W/K-0	R/W/K-0	R/W/K-0	R/W/K-0	R/W/K-0	R/W/K-0	r-x	R/W/K-0
IDIF	T1MSECIF	LSTATEIF	ACTVIF	SESVDIF	SESENDIF	—	VBUSVDIF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	K = Write ‘1’ to clear	-n = Bit Value at POR: (‘0’, ‘1’, x = unknown)	

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7 **IDIF:** ID State Change Indicator bit
Write a ‘1’ to this bit to clear the interrupt.
1 = Change in ID state detected
0 = No change in ID state detected

bit 6 **T1MSECIF:** 1 Millisecond Timer bit
Write a ‘1’ to this bit to clear the interrupt.
1 = 1 millisecond timer has expired
0 = 1 millisecond timer has not expired

bit 5 **LSTATEIF:** Line State Stable Indicator bit
Write a ‘1’ to this bit to clear the interrupt.
1 = USB line state has been stable for 1 ms, but different from last time
0 = USB line state has not been stable for 1 ms

bit 4 **ACTVIF:** Bus Activity Indicator bit
Write a ‘1’ to this bit to clear the interrupt.
1 = Activity on the D+, D-, ID, or VBUS pins has caused the device to wake-up..
0 = Activity has not been detected.

bit 3 **SESVDIF:** Session Valid Change Indicator bit
Write a ‘1’ to this bit to clear the interrupt.
1 = VBUS voltage has dropped below the session end level
0 = VBUS voltage has not dropped below the session end level

bit 2 **SESENDIF:** B-Device VBUS Change Indicator bit
Write a ‘1’ to this bit to clear the interrupt.
1 = A change on the session end input was detected
0 = No change on the session end input was detected

bit 1 **Reserved:** Write ‘0’; ignore read

Register 27-1: U1OTGIR: USB OTG Interrupt Status Register (Continued)

bit 0 **VBUSVDIF:** A-Device VBUS Change Indicator bit

Write a '1' to this bit to clear the interrupt.

1 = Change on the session valid input detected

0 = No change on the session valid input detected

Register 27-2: U1OTGIE: USB OTG Interrupt Enable Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
IDIE	T1MSECIE	LSTATEIE	ACTVIE	SESVDIE	SESENDIE	—	VBUSVDIE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7 **IDIE:** ID Interrupt Enable bit
1 = ID interrupt enabled
0 = ID interrupt disabled

bit 6 **T1MSECIE:** 1 Millisecond Timer Interrupt Enable bit
1 = 1 millisecond timer interrupt enabled
0 = 1 millisecond timer interrupt disabled

bit 5 **LSTATEIE:** Line State Interrupt Enable bit
1 = Line state interrupt enabled
0 = Line state interrupt disabled

bit 4 **ACTVIE:** Bus Activity Interrupt Enable bit
1 = ACTIVITY interrupt enabled
0 = ACTIVITY interrupt disabled

bit 3 **SESVDIE:** Session Valid Interrupt Enable bit
1 = Session valid interrupt enabled
0 = Session valid interrupt disabled

bit 2 **SESENDIE:** B-Session End Interrupt Enable bit
1 = B-session end interrupt enabled
0 = B-session end interrupt disabled

bit 1 **Reserved:** Write ‘0’; ignore read

bit 0 **VBUSVDIE:** A-VBUS Valid Interrupt Enable bit
1 = A-VBUS valid interrupt enabled
0 = A-VBUS valid interrupt disabled

Register 27-3: U1OTGSTAT: USB OTG Status Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R-0	r-x	R-0	r-x	R-0	R-0	r-x	R-0
ID	—	LSTATE	—	SESVD	SESEND	—	VBUSVD
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7 **ID:** ID Pin State Indicator bit
1 = No cable is attached or a type B cable has been plugged into the USB receptacle
0 = A type A OTG cable has been plugged into the USB receptacle

bit 6 **Reserved:** Write ‘0’; ignore read

bit 5 **LSTATE:** Line State Stable Indicator bit
1 = USB line state (U1CON<SE0> and U1CON<JSTATE>) has been stable for the previous 1 ms
0 = USB line state (U1CON<SE0> and U1CON<JSTATE>) has not been stable for the previous 1 ms

bit 4 **Reserved:** Write ‘0’; ignore read

bit 3 **SESVD:** Session Valid Indicator bit
1 = VBUS voltage is above Session Valid on the A or B device
0 = VBUS voltage is below Session Valid on the A or B device

bit 2 **SESEND:** B-Session End Indicator bit
1 = VBUS voltage is below Session Valid on the B device
0 = VBUS voltage is above Session Valid on the B device

bit 1 **Reserved:** Write ‘0’; ignore read

bit 0 **VBUSVD:** A-VBUS Valid Indicator bit
1 = VBUS voltage is above Session Valid on the A device
0 = VBUS voltage is below Session Valid on the A device

Register 27-4: U1OTGCON: USB OTG Control Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
DPPULUP	DMPULUP	DPPULDWN	DMPULDWN	VBUSON	OTGEN	VBUSCHG	VBUSDIS
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7 **DPPULUP:** D+ Pull-Up Enable bit
1 = D+ data line pull-up resistor is enabled
0 = D+ data line pull-up resistor is disabled

bit 6 **DMPULUP:** D- Pull-Up Enable bit
1 = D- data line pull-up resistor is enabled
0 = D- data line pull-up resistor is disabled

bit 5 **DPPULDWN:** D+ Pull-Down Enable bit
1 = D+ data line pull-down resistor is enabled
0 = D+ data line pull-down resistor is disabled

bit 4 **DMPULDWN:** D- Pull-Down Enable bit
1 = D- data line pull-down resistor is enabled
0 = D- data line pull-down resistor is disabled

bit 3 **VBUSON:** VBUS Power-on bit
1 = VBUS line is powered
0 = VBUS line is not powered

bit 2 **OTGEN:** OTG Functionality Enable bit
1 = DPPULUP, DMPULUP, DPPULDWN, and DMPULDWN bits are under software control
0 = DPPULUP, DMPULUP, DPPULDWN, and DMPULDWN bits are under USB hardware control

bit 1 **VBUSCHG:** VBUS Charge Enable bit
1 = VBUS line is charged through a pull-up resistor
0 = VBUS line is not charged through a resistor

bit 0 **VBUSDIS:** VBUS Discharge Enable bit
1 = VBUS line is discharged through a pull-down resistor
0 = VBUS line is not discharged through a resistor

Register 27-5: U1PWRC: USB Power Control Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R-0	r-x	r-x	R/W-0	r-x	r-x	R/W-0	R/W-0
UACTPND	—	—	USLPGRD	—	—	USUSPEND	USBPWR
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7 **UACTPND:** USB Activity Pending bit
1 = USB bus activity has been detected; but an interrupt is pending, it has not been generated yet
0 = An interrupt is not pending

bit 6-5 **Reserved:** Write ‘0’; ignore read

bit 4 **USLPGRD:** USB Sleep Entry Guard bit
1 = Sleep entry is blocked if USB bus activity is detected or if a notification is pending
0 = USB module does not block Sleep entry

bit 3-2 **Reserved:** Write ‘0’; ignore read

bit 1 **USUSPEND:** USB Suspend Mode bit
1 = USB module is placed in Suspend mode
(The 48 MHz USB clock will be gated off. The transceiver is placed in a low-power state.)
0 = USB module operates normally.

bit 0 **USBPWR:** USB Operation Enable bit
1 = USB module is turned on
0 = USB module is disabled
(Outputs held inactive, device pins not used by USB, analog features are shut down to reduce power consumption.)

Register 27-6: U1IR: USB Interrupt Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R/W/K-0	R/W/K-0	R/W/K-0	R/W/K-0	R/W/K-0	R/W/K-0	R/K-0	R/W/K-0
STALLIF	ATTACH	RESUMEIF	IDLEIF	TRNIF	SOFIF	UERRIF	URSTIF(5) / DETACHIF(6)
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	K = Write '1' to clear	-n = Bit Value at POR: (0, 1, x = unknown)	

bit 31-8 **Reserved:** Write '0'; ignore read

bit 7 **STALLIF:** STALL Handshake Interrupt bit
Write a '1' to this bit to clear the interrupt.
1 = In Host mode a STALL handshake was received during the handshake phase of the transaction
In Device mode a STALL handshake was transmitted during the handshake phase of the transaction
0 = STALL handshake has not been sent

bit 6 **ATTACHIF:** Peripheral Attach Interrupt bit(1)
Write a '1' to this bit to clear the interrupt.
1 = Peripheral attachment was detected by the USB module
0 = Peripheral attachment was not detected

bit 5 **RESUMEIF:** Resume Interrupt bit(2)
Write a '1' to this bit to clear the interrupt.
1 = K-State is observed on the D+ or D- pin for 2.5 µs
0 = K-State is not observed

bit 4 **IDLEIF:** Idle Detect Interrupt bit
Write a '1' to this bit to clear the interrupt.
1 = Idle condition detected (constant Idle state of 3 ms or more)
0 = No Idle condition detected

Note 1: This bit is valid only if the HOSTEN bit is set (see Register 27-11), there is no activity on the USB for 2.5 µs, and the current bus state is not SE0.
2: When not in Suspend mode, this interrupt should be disabled.
3: Clearing this bit will cause the STAT FIFO to advance.
4: Only error conditions enabled through the U1EIE register will set this bit.
5: Device mode.
6: Host mode.

Register 27-6: U1IR: USB Interrupt Register (Continued)

bit 3 **TRNIF:** Token Processing Complete Interrupt bit(3)

Write a '1' to this bit to clear the interrupt.

1 = Processing of current token is complete; a read of the U1STAT register will provide endpoint information

0 = Processing of current token not complete

bit 2 **SOFIF:** SOF Token Interrupt bit

Write a '1' to this bit to clear the interrupt.

1 = SOF token received by the peripheral or the SOF threshold reached by the host

0 = SOF token was not received nor threshold reached

bit 1 **UERRIF:** USB Error Condition Interrupt bit(4)

Write a '1' to this bit to clear the interrupt.

1 = Unmasked error condition has occurred

0 = Unmasked error condition has not occurred

bit 0 **URSTIF:** USB Reset Interrupt bit (Device mode)

1 = Valid USB Reset has occurred

0 = No USB Reset has occurred

DETACHIF: USB Detach Interrupt bit (Host mode)

1 = Peripheral detachment was detected by the USB module

0 = Peripheral detachment was not detected

Note 1: This bit is valid only if the HOSTEN bit is set (see Register 27-11), there is no activity on the USB for 2.5 µs, and the current bus state is not SE0.

2: When not in Suspend mode, this interrupt should be disabled.

3: Clearing this bit will cause the STAT FIFO to advance.

4: Only error conditions enabled through the U1EIE register will set this bit.

5: Device mode.

6: Host mode.

Register 27-7: U1IE: USB Interrupt Enable Register(1)

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
STALLIE	ATTACHIE	RESUMEIE	IDLEIE	TRNIE	SOFIE	UERRIE	URSTIE(2) DETACHIE(3)
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7 **STALLIE:** STALL Handshake Interrupt Enable bit
1 = STALL interrupt enabled
0 = STALL interrupt disabled

bit 6 **ATTACHIE:** ATTACH Interrupt Enable bit
1 = ATTACH interrupt enabled
0 = ATTACH interrupt disabled

bit 5 **RESUMEIE:** RESUME Interrupt Enable bit
1 = RESUME interrupt enabled
0 = RESUME interrupt disabled

bit 4 **IDLEIE:** Idle Detect Interrupt Enable bit
1 = IDLE interrupt enabled
0 = IDLE interrupt disabled

bit 3 **TRNIE:** Token Processing Complete Interrupt Enable bit
1 = TRNIF interrupt enabled
0 = TRNIF interrupt disabled

bit 2 **SOFIE:** SOF Token Interrupt Enable bit
1 = SOFIF interrupt enabled
0 = SOFIF interrupt disabled

bit 1 **UERRIE:** USB Error Interrupt Enable bit
1 = USB Error interrupt enabled
0 = USB Error interrupt disabled

Note 1: For an interrupt to propagate to the USBIF (IFS1<25>), the UERRIE bit (U1IE<1>) must be set.
2: Device mode.
3: Host mode.

Register 27-7: U1IE: USB Interrupt Enable Register(1) (Continued)

bit 0 **URSTIE:** USB Reset Interrupt Enable bit (Device Mode)

1 = URSTIF interrupt enabled
0 = URSTIF interrupt disabled

DETACHIE: USB Detach Interrupt Enable bit (Host Mode)

1 = DATTCHIF interrupt enabled
0 = DATTCHIF interrupt disabled

Note 1: For an interrupt to propagate to the USBIF (IFS1<25>), the UERRIE bit (U1IE<1>) must be set.

2: Device mode.

3: Host mode.

Register 27-8: U1EIR: USB Error Interrupt Status Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R/W/K-0	R/W/K-0	R/W/K-0	R/W/K-0	R/W/K-0	R/W/K-0	R/W-0	R/W-0
BTSEF	BMXEF	DMAEF	BTOEF	DFN8EF	CRC16EF	CRC5EF(4) / EOFEF(5)	PIDEF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	K = Write ‘1’ to clear	-n = Bit Value at POR: (‘0’, ‘1’, x = unknown)	

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7 **BTSEF:** Bit Stuff Error Flag bit
Write a ‘1’ to this bit to clear the interrupt.
1 = Packet rejected due to bit stuff error
0 = Packet accepted

bit 6 **BMXEF:** Bus Matrix Error Flag bit
Write a ‘1’ to this bit to clear the interrupt.
1 = The base address, of the BDT, or the address of an individual buffer pointed to by a BDT entry, is invalid.
0 = No address error

bit 5 **DMAEF:** DMA Error Flag bit(1)
Write a ‘1’ to this bit to clear the interrupt.
1 = USB DMA error condition detected
0 = No DMA error

bit 4 **BTOEF:** Bus Turnaround Time-Out Error Flag bit(2)
Write a ‘1’ to this bit to clear the interrupt.
1 = Bus turnaround time-out has occurred
0 = No bus turnaround time-out

Note 1: This type of error occurs when the module’s request for the DMA bus is not granted in time to service the module’s demand for memory, resulting in an overflow or underflow condition, and/or the allocated buffer size is not sufficient to store the received data packet causing it to be truncated.

2: This type of error occurs when more than 16 bit-times of Idle from the previous (End-of-Packet) EOP has elapsed.

3: This type of error occurs when the module is transmitting or receiving data and the SOF counter has reached zero.

4: Device mode.

5: Host mode.

Register 27-8: U1EIR: USB Error Interrupt Status Register (Continued)

bit 3 **DFN8EF:** Data Field Size Error Flag bit
Write a '1' to this bit to clear the interrupt.
1 = Data field received is not an integral number of bytes
0 = Data field received is an integral number of bytes

bit 2 **CRC16EF:** CRC16 Failure Flag bit
Write a '1' to this bit to clear the interrupt.
1 = Data packet rejected due to CRC16 error
0 = Data packet accepted

bit 1 **CRC5EF:** CRC5 Host Error Flag bit(3) (Device Mode)
Write a '1' to this bit to clear the interrupt.
1 = Token packet rejected due to CRC5 error
0 = Token packet accepted

EOFEF: EOF Error Flag bit (Host Mode)
1 = EOF error condition detected
0 = No EOF error condition

bit 0 **PIDEF:** PID Check Failure Flag bit
1 = PID check failed
0 = PID check passed

Note 1: This type of error occurs when the module's request for the DMA bus is not granted in time to service the module's demand for memory, resulting in an overflow or underflow condition, and/or the allocated buffer size is not sufficient to store the received data packet causing it to be truncated.

2: This type of error occurs when more than 16 bit-times of Idle from the previous (End-of-Packet) EOP has elapsed.

3: This type of error occurs when the module is transmitting or receiving data and the SOF counter has reached zero.

4: Device mode.

5: Host mode.

Register 27-9: U1EIE: USB Error Interrupt Enable Register(1)

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
BTSEE	BMXEE	DMAEE	BTOEE	DFN8EE	CRC16EE	CRC5EE(2) / EOFEE(3)	PIDEE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7 **BTSEE:** Bit Stuff Error Interrupt Enable bit
1 = BTSEF interrupt enabled
0 = BTSEF interrupt disabled

bit 6 **BMXEE:** Bus Matrix Error Interrupt Enable bit
1 = BMXEF interrupt enabled
0 = BMXEF interrupt disabled

bit 5 **DMAEE:** DMA Error Interrupt Enable bit
1 = DMAEF interrupt enabled
0 = DMAEF interrupt disabled

bit 4 **BTOEE:** Bus Turnaround Time-out Error Interrupt Enable bit
1 = BTOEF interrupt enabled
0 = BTOEF interrupt disabled

bit 3 **DFN8EE:** Data Field Size Error Interrupt Enable bit
1 = DFN8EF interrupt enabled
0 = DFN8EF interrupt disabled

bit 2 **CRC16EE:** CRC16 Failure Interrupt Enable bit
1 = CRC16EF interrupt enabled
0 = CRC16EF interrupt disabled

Note 1: For an interrupt to propagate USBIF (IFS1<25>), the UERRIE bit (U1IE<1>) must be set.
2: Device mode.
3: Host mode.

Register 27-9: U1EIE: USB Error Interrupt Enable Register(1) (Continued)

bit 1 **CRC5EE:** CRC5 Host Error Interrupt Enable bit (Device Mode)
1 = CRC5EF interrupt enabled
0 = CRC5EF interrupt disabled
EOFEE: EOF Error Interrupt Enable bit (Host Mode)
1 = EOF interrupt enabled
0 = EOF interrupt disabled

bit 0 **PIDEE:** PID Check Failure Interrupt Enable bit
1 = PIDEF interrupt enabled
0 = PIDEF interrupt disabled

Note 1: For an interrupt to propagate USBIF (IFS1<25>), the UERRIE bit (U1IE<1>) must be set.
2: Device mode.
3: Host mode.

Register 27-10: U1STAT: USB Status Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R-X	R-X	R-X	R-X	R-X	R-X	r-x	r-x
ENDPT<3:0>				DIR	PPBI	—	—
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7-4 **ENDPT<3:0>:** Encoded Number of Last Endpoint Activity bits
(Represents the number of the BDT, updated by the last USB transfer.)
1111 = Endpoint 15
1110 = Endpoint 14
....
0001 = Endpoint 1
0000 = Endpoint 0

bit 3 **DIR:** Last BD Direction Indicator bit
1 = Last transaction was a transmit transfer (TX)
0 = Last transaction was a receive transfer (RX)

bit 2 **PPBI:** Ping-Pong BD Pointer Indicator bit
1 = The last transaction was to the ODD BD bank
0 = The last transaction was to the EVEN BD bank

bit 1-0 **Reserved:** Write ‘0’; ignore read

Note: The U1STAT register is a window into a 4-byte FIFO maintained by the USB module. U1STAT value is only valid when U1IR<TRNIF> is active. Clearing the U1IR<TRNIF> bit advances the FIFO. Data in register is invalid when U1IR<TRNIF> = 0.

Register 27-11: U1CON: USB Control Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R-x	R-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
JSTATE	SE0	PKTDIS(4) / TOKBUSY(5)	USBRST	HOSTEN	RESUME	PPBRST	USBEN(4) / SOFEN(5)
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7 **JSTATE:** Live Differential Receiver JSTATE flag bit
1 = JSTATE detected on the USB
0 =No JSTATE detected

bit 6 **SE0:** Live Single-Ended Zero flag bit
1 = Single Ended Zero detected on the USB
0 = No Single Ended Zero detected

bit 5 **PKTDIS:** Packet Transfer Disable bit (Device mode)
1 = Token and packet processing disabled (set upon SETUP token received)
0 = Token and packet processing enabled
TOKBUSY: Token Busy Indicator bit(1) (Host mode)
1 = Token being executed by the USB module
0 = No token being executed

bit 4 **USBRST:** Module Reset bit (Host mode only)
1 = USB reset generated
0 = USB reset terminated

bit 3 **HOSTEN:** Host Mode Enable bit(2)
1 = USB host capability enabled
0 = USB host capability disabled

Note 1: Software is required to check this bit before issuing another token command to the U1TOK register, see Register 27-15.

2: All host control logic is reset any time that the value of this bit is toggled.

3: Software must set RESUME for 10 ms if the part is a function, or for 25 ms if the part is a host, and then clear it to enable remote wake-up. In Host mode, the USB module will append a low-speed EOP to the RESUME signaling when this bit is cleared.

4: Device mode.

5: Host mode.

Register 27-11: U1CON: USB Control Register (Continued)

bit 2 **RESUME:** RESUME Signaling Enable bit(3)
1 = RESUME signaling activated
0 = RESUME signaling disabled

bit 1 **PPBRST:** Ping-Pong Buffers Reset bit
1 = Reset all Even/Odd buffer pointers to the EVEN BD banks
0 = Even/Odd buffer pointers not being Reset

bit 0 **USBEN:** USB Module Enable bit (Device Mode)
1 = USB module and supporting circuitry enabled
0 = USB module and supporting circuitry disabled

SOFEN: SOF Enable bit (Host Mode)
1 = SOF token sent every 1 ms
0 = SOF token disabled

Note 1: Software is required to check this bit before issuing another token command to the U1TOK register, see Register 27-15.

2: All host control logic is reset any time that the value of this bit is toggled.

3: Software must set RESUME for 10 ms if the part is a function, or for 25 ms if the part is a host, and then clear it to enable remote wake-up. In Host mode, the USB module will append a low-speed EOP to the RESUME signaling when this bit is cleared.

4: Device mode.

5: Host mode.

Register 27-12: U1ADDR: USB Address Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
LSPDEN	DEVADDR<6:0>						
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7 **LSPDEN:** Low Speed Enable Indicator bit
1 = Next token command to be executed at Low Speed
0 = Next token command to be executed at Full Speed

bit 6-0 **DEVADDR<6:0>:** 7-bit USB Device Address bits

Register 27-13: U1FRML: USB Frame Number Low Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
FRML<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7-0 **FRML<7:0>:** The 11-bit Frame Number Lower bits
The register bits are updated with the current frame number whenever a SOF TOKEN is received.

Register 27-14: U1FRMH: USB Frame Number High Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

r-x	r-x	r-x	r-x	r-x	R-0	R-0	R-0
—	—	—	—	—	FRMH<10:8>		
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-3 **Reserved:** Write ‘0’; ignore read

bit 2-0 **FRMH<10:8>:** The Upper 3 Bits of the Frame Numbers
The register bits are updated with the current frame number whenever a SOF TOKEN is received.

Register 27-15: U1TOK: USB Token Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PID<3:0>				EP<3:0>			
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7-4 **PID<3:0>:** Token Type Indicator bits(1)

0001 = OUT (TX) token type transaction
1001 = IN (RX) token type transaction
1101 = SETUP (TX) token type transaction
Note: All other values are reserved and must not be used.

bit 3-0 **EP<3:0>:** Token Command Endpoint Address bits

The four bit value must specify a valid endpoint.

Note 1: All other values are reserved and must not be used.

Register 27-16: U1SOF: USB SOF Threshold Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	r-x
CNT<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7-0 **CNT<7:0>:** SOF Threshold Value bits

Typical values of the threshold are:
0100 1010= 64-byte packet
0010 1010= 32-byte packet
0001 1010= 16-byte packet
0001 0010= 8-byte packet

Register 27-17: U1BDTP1: USB BDT Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	r-x
BDTPTRL<15:9>							—
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7-1 **BDTPTRL<15:9>:** BDT Base Address bits

This 7-bit value provides address bits 15 through 9 of the BDT base address, which defines the BDT’s starting location in the system memory.

The 32-bit BDT base address is 512-byte aligned.

bit 0 **Reserved:** Write ‘0’; ignore read

Register 27-18: U1BDTP2: USB BDT PAGE 2 Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
BDTPTRH<23:16>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7-0 **BDTPTRH<23:16>:** BDT Base Address bits

This 8-bit value provides address bits 23 through 16 of the BDT base address, which defines the BDT’s starting location in the system memory.

The 32-bit BDT base address is 512-byte aligned.

Register 27-19: U1BDTP3: USB BDT PAGE 3 Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
BDTPTRU<31:24>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7-0 **BDTPTRU<31:24>:** BDT Base Address bits

This 8-bit value provides address bits 31 through 24 of the BDT base address, which defines the BDT’s starting location in the system memory.

The 32-bit BDT base address is 512-byte aligned.

Register 27-20: U1CNFG1: USB Configuration 1 Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	r-x	r-x	r-x	r-x
UTEYE	UOEMON	USBFRZ	USBSIDL	—	—	—	—
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7 **UTEYE:** USB Eye-Pattern Test Enable bit
1 = Eye-Pattern Test enabled
0 = Eye-Pattern Test disabled

bit 6 **UOEMON:** USB $\overline{OE}$ Monitor Enable bit
1 = OE signal active; it indicates intervals during which the D+/D- lines are driving
0 = OE signal inactive

bit 5 **USBFRZ:** Freeze in DEBUG Mode bit
1 = When emulator is in DEBUG mode, module freezes operation
0 = When emulator is in DEBUG mode, module continues operation

bit 4 **USBSIDL:** Stop in IDLE Mode bit
1 = Discontinue module operation when device enters IDLE mode
0 = Continue module operation in IDLE mode

bit 3-0 **Reserved:** Write ‘0’; ignore read

Register 27-21: U1EP0-U1EP15: USB Endpoint Control Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R/W-0	R/W-0	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
LSPD	RETRYDIS	—	EPCONDIS	EPRXEN	EPTXEN	EPSTALL	EPHSHK
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7 **LSPD:** Low-Speed Direct Connection Enable bit (Host mode and U1EP0 only)
1 = Direct connection to a low-speed device enabled
0 = Direct connection to a low-speed device disabled; hub required with PRE_PID

bit 6 **RETRYDIS:** Retry Disable bit (Host mode and U1EP0 only)
1 = Retry NAK’d transactions disabled
0 = Retry NAK’d transactions enabled; retry done in hardware

bit 5 **Reserved:** Write ‘0’; ignore read

bit 4 **EPCONDIS:** Bidirectional Endpoint Control bit
If EPTXEN = 1 and EPRXEN = 1:
1 = Disable Endpoint n from Control transfers; only TX and RX transfers allowed
0 = Enable Endpoint n for Control (SETUP) transfers; TX and RX transfers also allowed
Otherwise, this bit is ignored.

bit 3 **EPRXEN:** Endpoint Receive Enable bit
1 = Endpoint n receive enabled
0 = Endpoint n receive disabled

bit 2 **EPTXEN:** Endpoint Transmit Enable bit
1 = Endpoint n transmit enabled
0 = Endpoint n transmit disabled

bit 1 **EPSTALL:** Endpoint Stall Status bit
1 = Endpoint n was stalled
0 = Endpoint n was not stalled

bit 0 **EPHSHK:** Endpoint Handshake Enable bit
1 = Endpoint Handshake enabled
0 = Endpoint Handshake disabled (typically used for isochronous endpoints)

Register 27-22: OSCCON: Oscillator Control Register(1)

r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-1
—	—	PLLODIV<2:0>			FRCDIV<2:0>		
bit 31							bit 24

r-x	r-x	r-x	R/W-1	R/W-1	R/W-P	R/W-P	R/W-P
—	—	—	PBDIV<1:0>		PLLMULT<2:0>		
bit 23							bit 16

r-x	R-0	R-0	R-0	r-x	R/W-x	R/W-x	R/W-x
—	COSC<2:0>			—	NOSC<2:0>		
bit 15							bit 8

R/SO-0	R-0	R-0	R/W-0	R/C-0	R/W-0	R/W-x	R/W-0
CLKLOCK	ULOCK	LOCK	SLPEN	CF	UFRCEN	SOSCEN	OSWEN
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 5 **ULOCK:** USB PLL Lock Status bit

1 = PLL module is in lock or PLL module start-up timer is satisfied
0 = PLL module is out of lock, PLL start-up timer is running or PLL is disabled

bit 3 **UFRCEN:** USB FRC Clock Source Enable bit

1 = FRC is enabled and is the clock source for the USB module (USB transfers cannot complete using this clock source)
0 = FRC is not used as the USB module clock source

Note 1: Shaded bit names in this register control other PIC32MX peripherals and are not related to USB.

Register 27-23: IFS1: Interrupt Flag Status Register 1(1)

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	USBIF	FCEIF
bit 31							bit 24

r-x	r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	DMA3IF	DMA2IF	DMA1IF	DMA0IF
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RTCCIF	FSCMIF	I2C2MIF	I2C2SIF	I2C2BIF	U2TXIF	U2RXIF	U2EIF
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SPI2RXIF	SPI2TXIF	SPI2EIF	CMP2IF	CMP1IF	PMPIF	AD1IF	CNIF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 25 **USBIF:** Global USB Interrupt Request Flag
1 = Interrupt request has occurred
0 = No interrupt request has occurred

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to USB.

Register 27-24: IEC1: Interrupt Enable Control Register 1(1)

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	USBIE	FCEIE
bit 31							bit 24

r-x	r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	DMA3IE	DMA2IE	DMA1IE	DMA0IE
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RTCCIE	FSCMIE	I2C2MIE	I2C2SIE	I2C2BIE	U2TXIE	U2RXIE	U2EIE
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SPI2RXIE	SPI2TXIE	SPI2EIE	CMP2IE	CMP1IE	PMPIE	AD1IE	CNIE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-26 **Reserved:** Write ‘0’; ignore read

bit 25 **USBIE:** USB Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 24-0 **Reserved:** Write ‘0’; ignore read

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to USB.

Register 27-25: DEVCFG2: Boot Configuration Register(1)

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	R-P	R-P	R-P
—	—	—	—	—	FPLLODIV<2:0>		
bit 23							bit 16

R-P	r-x	r-x	r-x	r-x	R-P	R-P	R-P
FUPLLEN	—	—	—	—	FUPLLIDIV<2:0>		
bit 15							bit 8

r-x	R-P	R-P	R-P	r-x	R-P	R-P	R-P
—	FPLLMULT<2:0>			—	FPLLIDIV<2:0>		
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 15 **FUPLLEN:** USB Phase-Lock Loop (PLL) Status bit
1 = Enable USB PLL
0 = Disable USB PLL

bit 10-8 **FUPLLIDIV:** USB PLL Prescaler Value bits
111 = 12x divider
110 = 10x divider
101 = 6x divider
100 = 5x divider
011 = 4x divider
010 = 3x divider
001 = 2x divider
000 = 1x divider

Note 1: Shaded bit names in this register control other PIC32MX peripherals and are not related to USB.

27.3 OPERATION

This section contains a brief overview of USB operation, followed by PIC32MX USB module implementation specifics, and module initialization requirements.

Note: A good understanding of USB can be gained from documents that are available on the USB implementers web site. In particular, refer to "*Universal Serial Bus Specification, Revision 2.0*" (http://www.usb.org/developers/docs).

27.3.1 USB 2.0 Operation Overview

USB is an asynchronous serial interface with a tiered star configuration. USB is implemented as a master/slave configuration. On a given bus, there can be multiple (up to 127) slaves (devices), but there is only one master (host).

27.3.2 Modes of Operation

The following USB implementation modes are described in this overview:

- Host mode
  - USB Standard Host mode – the USB implementation that is typically used for a personal computer
  - Embedded Host mode – the USB implementation that is typically used for a microcontroller
- Device mode – the USB implementation that is typically used for a peripheral such as a thumb drive, keyboard, or mouse
- OTG Dual Role mode – the USB implementation in which an application may dynamically switch its role as either host or device

27.3.2.1 Host Mode

The host is the master in a USB system and is responsible for identifying all devices connected to it (enumeration), initiating all transfers, allocating bus bandwidth and supplying power to any bus-powered USB devices connected directly to it.

27.3.2.1.1 USB Standard Host

In USB Standard Host mode, the following features and requirements are relevant:

- Large variety of devices are supported
- Supports all USB transfer types
- USB hubs are supported (allows connection of multiple devices simultaneously)
- Device drivers can be updated to support new devices
- Type 'A' receptacle is used for each port
- Each port must be able to deliver a minimum of 100 mA for a configured or unconfigured device, and optionally, up to 500 mA for a configured device
- Full-speed and low-speed protocols must be supported (high-speed can be supported)

Note: This mode is not supported by PIC32.

27.3.2.1.2 Embedded Host

In Embedded Host mode, the following features and requirements are relevant:

- Only supports a specific list of devices, referred to as a Targeted Peripheral List (TPL)
- Only required to support those transfer types that are required by devices in the TPL
- USB hub support is optional
- Device drivers are not required to be updatable
- Type 'A' receptacle is used for each port
- Only those speeds required by devices in the TLP must be supported
- Each port must be able to deliver a minimum of 100 mA for a configured or unconfigured device, and optionally, up to 500 mA for a configured device

27.3.2.2 Device Mode

USB devices accept commands and data from the host and respond to requests for data. USB devices perform peripheral functions, e.g., a mouse or other I/O, or data storage.

The following characteristics generally describe a USB device:

- Functionality may be class- or vendor-specific
- Draws 100 mA or less from the bus before configuration
- Can draw up to 500 mA from the bus after successful negotiation with the host
- Can support low-speed, full-speed, or high-speed protocol (high-speed support requires implementation of full-speed protocol to enumerate)
- Supports control and data transfers as required for implementation
- Optionally supports Session Request Protocol (SRP)
- Can be bus-powered or self-powered

27.3.2.3 OTG Dual Role

The OTG dual role device supports both USB host and device functionality. OTG dual role devices use a micro-AB receptacle. This allows a micro-A or a micro-B plug to be attached. Both the micro-A and micro-B plugs have an additional pin, the ID pin, to signify which plug type was connected. The plug type connected to the receptacle, micro-A or micro-B, determines the default role of the device, host or USB device. An OTG device will perform the role of a host when a micro-A plug is detected. When a micro-B plug is detected, the role of a USB device is performed.

When an OTG device is directly connected to another OTG device using an OTG cable (micro-A to micro-B), Host Negotiation Protocol (HNP) can be used to swap the roles of host and USB device between the two without disconnecting and reconnecting the cable. To differentiate between the two OTG devices, the term “A-device” refers to the device connected to the micro-A plug and “B-device” refers to the device connected to the micro-B plug.

27.3.2.3.1 A-Device, the Default Host

In OTG dual role, operating as a host, the following features and requirements describe an A-device:

- Supports the devices on the TPL (class support is not allowed)
- Required to support those transaction types that are required by devices in the TPL
- USB hub support is optional
- Device drivers are not required to be updatable
- A single micro-AB receptacle is used
- Full-speed protocol must be supported (high-speed and/or low-speed protocol can be supported)
- USB port must be able to deliver a minimum of 8 mA for a configured or unconfigured device, and optionally, up to 500 mA for a configured device
- Supports HNP; the host can switch roles to become a device
- Supports at least one form of SRP
- A-device supplies V_{BUS} power when the bus is powered, even if the roles are swapped using HNP

27.3.2.3.2 B-Device, the Default Device

In OTG dual role, operating as a USB device, the following features and requirements describe a B-Device:

- Class- or vendor-specific functionality
- Draws 8 mA or less before configuration
- Is typically self-powered, due to low-current requirements, but can draw up to 500 mA after successful negotiation with the host
- A single micro-AB receptacle is used
- Must support full-speed protocol (support of low-speed and/or high-speed protocol is optional
- Supports control transfers, and supports data transfers as they are required for implementation
- Supports both forms of SRP – VBUS pulsing and data-line pulsing
- Supports HNP
- B-device does not supply VBUS power, even if the roles are swapped using HNP

Note:	Dual-role devices that do not full OTG functionality are possible using multiple USB receptacles, however there may be special requirements if these devices are to be made USB compliant, refer to the USB IF (implementers forum) for details..

27.3.2.4 Protocol

USB communication requires the use of specific protocols. The following subsections provide an overview of communication via USB.

27.3.2.4.1 Bus Transfers

Communication on the USB bus occurs through transfers between a host and a device. Each transfer type has unique features. An embedded or OTG host can implement only the control and the data transfer(s) it will use.

The following four transfer types are possible on the bus:

- Control

  Control transfer is used to identify a device during enumeration and to control it during operation. A percentage of the USB bandwidth is ensured to be available to control transfers. The data is verified by a cyclic redundancy check (CRC) and reception by the target is verified.
- Interrupt

  Interrupt transfer is a scheduled transfer of data in which the host allocates time slots for the transfers as required by the device's configuration. This time slot allocation results in the device being polled in a periodic manner. The data is verified by a CRC and reception by the target is verified.
- Isochronous

  Isochronous transfer is a scheduled transfer of data in which the host allocates time slots for the transactions as required by the device's configuration. Reception of the data is not verified, but the data integrity is verified by the device using a CRC. This transfer type is typically used for audio and video.
- Bulk

  Bulk transfer is used to move large amounts of data where the time of the transaction is not ensured. Time for this type of transfer is allocated from time that has not been allocated to the other three transfer types. The data is verified by a CRC and reception is verified.

The following transfer speeds are defined in the USB 2.0 specification:

- 480 Mbps – high speed
- 12 Mbps – full speed
- 1.5 Mbps – low speed

PIC32MX OTG devices support full-speed operation in Host and Device modes, and support low-speed operation in Host mode.

Information contrasting the timeliness, data integrity, data size, and speed of each transfer, or transaction, type is shown in Table 27-2.

Table 27-2: Transaction Types (Full-Speed Operation)

Transaction Type	Timeliness Ensured	Data Integrity Ensured	Maximum Packet Size	Maximum Throughput(1)
Control	Yes	Yes	64	.83 MB/s
Interrupt	Yes	Yes	64	1.22 MB/s
Isochronous	Yes	No	1023	1.28 MB/s
Bulk	No	Yes	64	1.22 MB/s

Note 1: These numbers reflect the theoretical maximum data throughput, including protocol overhead, on an otherwise empty bus. The bit stuffing overhead required by the Non-Return to Zero Inverted (NRZI) encoding is not included in the calculations.

27.3.2.4.2 Bandwidth Allocation

Control transfers, or transactions, are required to be at least 10% of the available bandwidth within a given frame. The remaining 90% is available for allocation to Interrupt and Isochronous transfers. Bulk transfers are allocated from any bandwidth not allocated to control, interrupt, or isochronous transfers. Bulk transfers are not assured bandwidth. In practice, they have the greatest bandwidth, since frames are rarely fully allocated.

27.3.2.4.3 Endpoints and USB Descriptors

All data transferred on the bus is sent or received through endpoints. USB supports devices with up to 16 endpoints. Each endpoint can have transmit (TX) and/or receive (RX) functionality. Each endpoint uses one transaction type. Endpoint 0 is the default control transfer endpoint.

27.3.2.5 Physical Bus Interface

27.3.2.5.1 Bus Speed Selection

The USB specification defines full-speed operation as 12 Mb/s and low speed operation as 1.5 Mb/s. A data line pull-up resistor is used to identify a device as full speed or low speed. For full-speed operation, the D+ line is pulled up; for low-speed operation, the D- line is pulled up.

27.3.2.5.2 VBUS Control

VBUS is the 5V USB power supplied by the host, or a hub, to operate bus-powered devices. The need for VBUS control depends on the role of the application. If VBUS power must be enabled and disabled, the control must be managed by firmware.

The following list details the VBUS requirements:

- Standard host typically supplies power to the bus at all times.
- Host may switch off VBUS to save power
- USB device never powers the bus – VBUS pulsing may be supported as part of the SRP.
- OTG A-device supplies power to the bus, and typically turns off VBUS to conserve power.
- OTG B-device can pulse VBUS for SRP.

Note:	Refer to the specific device data sheet for VBUS electrical parameters.

27.3.3 PIC32MX Implementation Specifics

This section details how the USB specification requirements are implemented in the PIC32MX USB module.

27.3.3.1 Bus Speed

The PIC32MX USB module supports the following speeds:

- Full-speed operation as a host and a device
- Low-speed operation as a host

27.3.3.2 Endpoints and Descriptors

All USB endpoints are implemented as buffers in RAM. The CPU and USB module have access to the buffers. To arbitrate access to these buffers between the USB module and CPU, a semaphore flag system is used. Each endpoint can be configured for TX and/or RX, and each has an ODD and an EVEN buffer.

Use of the Buffer Descriptor Table (BDT) allows the buffers to be located anywhere in RAM, and provides status flags and control bits. The BDT contains the address of each endpoint data buffer, as well as information about each buffer (see Figure 27-2, Figure 27-3 and Figure 27-4). Each BDT entry is called a Buffer Descriptor (BD) and is 8 bytes long. Four descriptor entries are used for each endpoint. All endpoints, ranging from endpoint 0 to the highest endpoint in use, must have four descriptor entries. Even if all of the buffers for an endpoint are not used, four descriptor entries are required for each endpoint.

The USB module calculates a buffer's location in RAM using the BDT Pointer registers. The base of the BDT is held in registers U1BDTP1 through U1BDTP3. The address of the desired buffer is found by using the endpoint number, the type (RX/TX) and the ODD/EVEN bit to index into the BDT. The address held by this entry is the address of the desired data buffer. Refer to **Section 27.3.2.3.1 "A-Device, the Default Host"**.

Note:	The contents of the U1BDTP1-U1BDTP3 registers provide the upper 23 bits of the 32-bit address; therefore, the BTD must be aligned to a 512-byte boundary (see Figure 27-2). This address must be the physical (not virtual) memory address.

Each of the 16 endpoints owns two descriptor pairs: two for packets to transmit, and two for packets received. Each pair manages two buffers, an EVEN and an ODD, requiring a maximum of 64 descriptors (16 * 2 * 2).

Having EVEN and ODD buffers for each direction allows the CPU to access data in one buffer while the USB module transfers data to or from the other buffer. The USB module alternates between buffers, clearing the UOWN bit in the buffer descriptor automatically when the transaction for that buffer is complete (see **Section 27.3.2.3 "OTG Dual Role"**). The use of alternating buffers maximizes data throughput by allowing CPU data access in parallel with data transfer. This technique is referred to as ping-pong buffering. Figure 27-2 illustrates how the endpoints are mapped in the BDT.

27.3.3.2.1 Endpoint Control

Each endpoint is controlled by an Endpoint Control register, U1EPn, that configures the transfer direction, the handshake, and the stalling properties of the endpoint. The Endpoint Control register also allows support of control transfers.

27.3.3.2.2 Host Endpoints

Note:	In Host mode, Endpoint 0 has additional bits for auto-retry and hub support.

The host performs all transactions through a single endpoint (Endpoint 0). All other endpoints should be disabled and other endpoint buffers are not be used.

27.3.3.2.3 Device Endpoints

Endpoint 0 must be implemented for a USB device to be enumerated and controlled. Devices typically implement additional endpoints to transfer data.

27.3.3.3 Buffer Management

The buffers are shared between the CPU and the USB module, and are implemented in system memory. So, a simple semaphore mechanism is used to distinguish current ownership of the BD, and associated buffers, in memory. This semaphore mechanism is implemented by the UOWN bit in each BD.

The USB module clears the UOWN bit automatically when the transaction for that buffer is complete. When the UOWN bit is clear, the descriptor is owned by the CPU – which may modify the descriptor and buffer as necessary.

Software must configure the BDT entry for the next transaction, then set the UOWN bit to return control to the USB module.

A BD is only valid if the corresponding endpoint has been enabled in the U1EPn register. The BDT is implemented in data memory, and the BDs are not modified when the USB module is reset. Initialize the BDs prior to enabling them through the U1EPn. At a minimum, the UOWN bits must be cleared prior to being enabled.

In Host mode, BDT initialization is required before the U1TOK register is written, which triggers a transfer.

Figure 27-2: BDT Address Generation

BDTBA<22:0>	ENDPOINT<3:0>	DIR	PPBI	FSOTG
31:9	8:5	4	3	2:0

bit 31:9 **BDTBA<22:0>:** BDT Base Address bits
The 23-bit value is made up of the contents of the U1BDTP3, U1BDTP2, and U1BDTP1 registers.

bit 8:5 **ENDPOINT<3:0>:** Transfer Endpoint Number bits
0000 = Endpoint 0
0001 = Endpoint 1
....
1110 = Endpoint 14
1111 = Endpoint 15

bit 4 **DIR:** Transfer Direction bit
1 = Transmit: SETUP/OUT for host, IN for function
0 = Receive: IN for host, SETUP/OUT for function

bit 3 **PPBI:** Ping-Pong Pointer bit
1 = ODD buffer
0 = EVEN buffer

bit 2:0 **Manipulated by the USB module**

27.3.3.3.1 Buffer Descriptor Format

The buffer descriptor is used in the following formats:

- Control
- Status.

Buffer descriptor control format, in which software writes the descriptor and hands it to hardware, is shown in Figure 27-3.

Figure 27-3: USB Buffer Descriptor Control Format: Software -> Hardware

Address Offset +0

31					26	25								16	15						8	7	6	5	4	3	2	1	0
—						BYTE COUNT<9:0>									—							UOWN	DATA0/1	KEEP	NINC	DTS	BSTALL	—	

Address Offset +4

31																												0
BUFFER ADDRESS<31:0>																												

Address Offset +0

bit 25-16 **BYTE_COUNT<9:0>:** Byte Count bits

Byte count represents the number of bytes to be transmitted or the maximum number of bytes to be received during a transfer.

bit 7 **UOWN:** USB Own bit

1 = USB module owns the BD and its corresponding buffer
CPU must not modify the BD or the buffer.
0 = CPU owns the BD and its corresponding buffer
USB module ignores all other fields in the BD.

USBFRZ is writable in Debug Exception mode only, it is forced to '0' in normal mode.

Note: This bit can be programmed by either the CPU or the USB module, and it must be initialized by the user to the desired value prior to enabling the USB module.

bit 6 **DATA0/1:** Data Toggle Packet bit

1 = Transmit a Data 1 packet or Check received PID = DATA1, if DTS = 1
0 = Transmit a Data 0 packet or Check received PID = DATA1, if DTS = 1

bit 5 **KEEP:** BD Keep Enable bit

1 = USB will keep the BD indefinitely once UOWN is set
U1STAT FIFO will not be updated and TRNIF bit will not be set at the end of each transaction.
0 = USB will hand back the BD once a token has been processed

bit 4 **NINC:** DMA Address Increment Disable bit

1 = DMA address increment disabled
0 = DMA address increment enabled

bit 3 **DTS:** Data Toggle Synchronization Enable bit

1 = Data Toggle Synchronization is enabled – data packets with incorrect sync value will be ignored
0 = No Data Toggle Synchronization is performed

Note: Expected value of DATA PID (DATA0/DATA1) specified in the DATA0/1 field.

bit 2 **BSTALL:** Buffer Stall Enable bit

1 = Buffer STALL enabled
STALL handshake issued if a token is received that would use the BD in the given location (UOWN bit remains set, BD value is unchanged).
Corresponding EPSTALL bit will get set on any STALL handshake.

0 = Buffer STALL disabled

Address Offset +4

bit 31-0 **BUFFER_ADDRESS:** Buffer Address bits(3)

Starting point address of the endpoint packet data buffer.

Note: The individual buffer addresses in the BDT must be physical memory addresses.

Buffer descriptor status format, in which hardware writes the descriptor and hands it back to software, is shown in Figure 27-4.

Figure 27-4: USB Buffer Descriptor Status Format: Hardware -> Software

Address Offset +0

31					26	25								16	15						8	7	6	5	4	3	2	1	0
—						BYTE COUNT<9:0>									—							UOWN	DATA0/1	PID<3:0>				—		

Address Offset +4

31																													0
BUFFER ADDRESS<31:0>																													

Address Offset +0

bit 25-16 **BYTE_COUNT<9:0>:** Byte Count bits

Byte count reflects the actual number of bytes received or transmitted.

bit 7 **UOWN:** USB Own bit

1 = USB module owns the BD and its corresponding buffer
CPU must not modify the BD or the buffer.
0 = CPU owns the BD and its corresponding buffer

Note: This bit can be programmed by either the CPU or the USB module, and it must be initialized by the user to the desired value prior to enabling the USB module.

bit 6 **DATA0/1:** Data Toggle Packet bit

1 = Data 1 packet received
0 = Data 0 packet received

Note: This bit is unchanged on an outgoing packet.

bit 5-2 **PID<3:0>:** Packet Identifier bits

The current token PID when a transfer completes.

The values written back are the token PID values from the USB specification: 0x1 for an OUT token, 0x9 for an IN token or 0xd for a SETUP token.

In Host mode, this field is used to report the last returned PID or a transfer status indication.

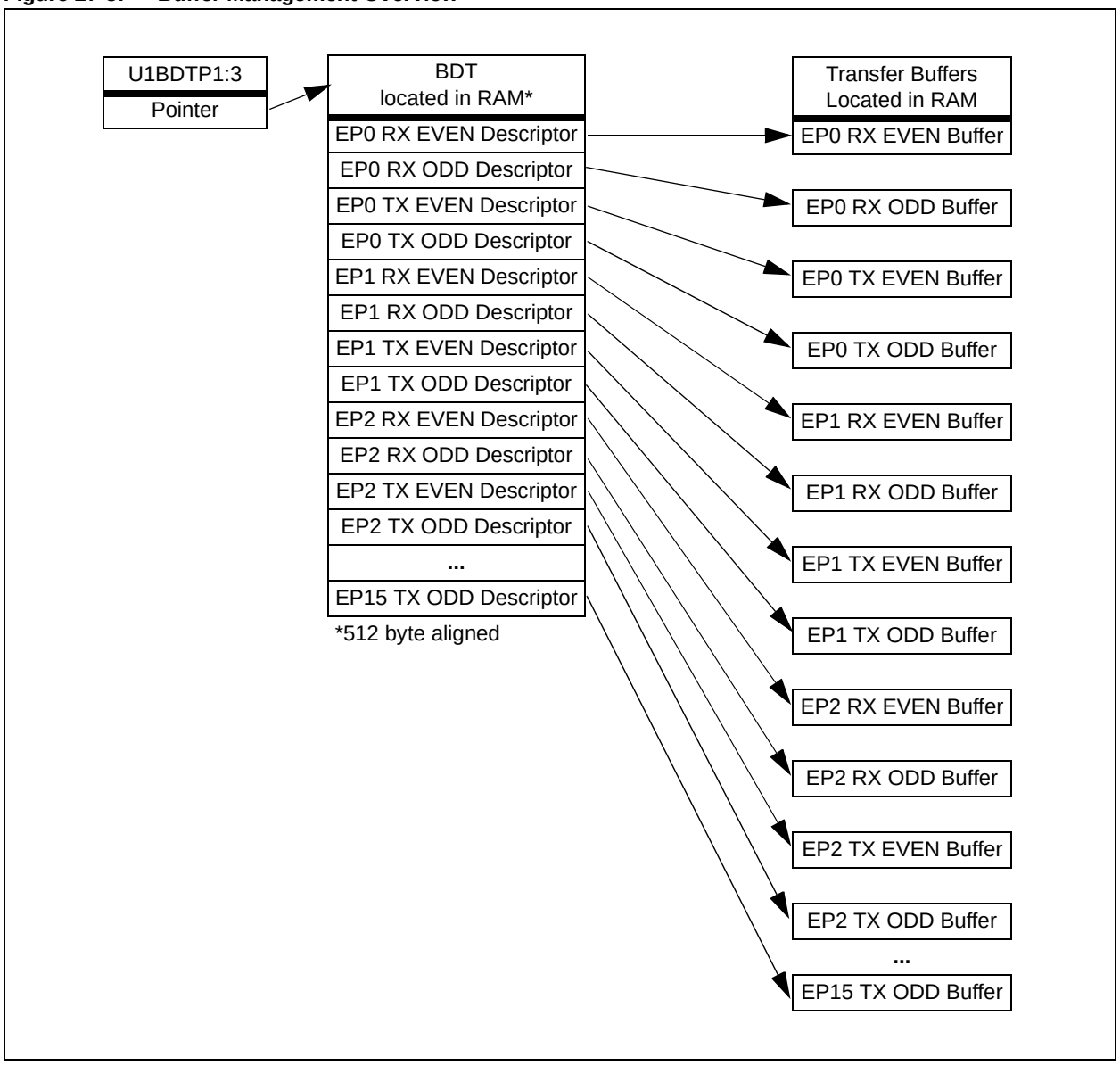
The possible values returned are: 0x3 DATA0, 0xb DATA1, 0x2 ACK, 0xe STALL, 0xa NAK, 0x0 Bus Time-out, 0xf Data Error.

Address Offset +4

bit 31-0 **BUFFER_ADDRESS:** Buffer Address bits

Starting point address of the endpoint packet data buffer.

Figure 27-5: Buffer Management Overview



27.3.3.4 Buffer Descriptor Configuration

The UOWN, DTSEN and BSTALL bits in each BDT entry control the data transfer for the associated buffer and endpoint.

Setting the DTSEN bit enables the USB module to perform data toggle synchronization. When DTS is enabled: if a packet arrives with an incorrect DTS, it will be ignored, the buffer remains unchanged, and the packet will be NAK'd (Negatively Acknowledged).

Setting the BSTALL bit causes the USB to issue a STALL handshake if a token is received by the SIE that would use the BD in this location – the corresponding EPSTALL bit is set and a STALLIF interrupt is generated. When the BSTALL bit is set, the BD is not consumed by the USB module (the UOWN bit remains set and the rest of the BD values are unchanged). If a SETUP token is sent to the stalled endpoint, the module automatically clears the corresponding BSTALL bit.

The byte count represents the total number of bytes that are transmitted or received. Valid byte counts range from 0 to 1023. For all endpoint transfers, the byte count is updated by the USB module, with the actual number of bytes transmitted or received, after the transfer is completed. If number of bytes received exceeds the corresponding byte count value written by the firmware, the overflow bit is set and the data is truncated to fit the size of the buffer (as given in the BTD).

27.3.4 Hardware Interface

27.3.4.1 Power Supply Requirements

Power supply requirements for USB implementation vary with the type of application, and are outlined below.

- Device:

  Operation as a device requires a power supply for the PIC32MX and the USB transceiver, see Figure 27-6 for an overview of USB implementation as a device.
- Embedded Host:

  Operation as a host requires a power supply for the PIC32MX, the USB transceiver, and a 5V nominal supply for the USB VBUS. The power supply must be able to deliver 100 mA, or up to 500 mA, depending on the requirements of the devices in the TPL. The application dictates whether the VBUS power supply can be disabled or disconnected from the bus by the PIC32MX application. Figure 27-7 presents an overview of USB implementation as a host.
- OTG Dual Role:

  Operation as an OTG dual role requires a power supply for the PIC32MX, the USB transceiver, and a switchable 5V nominal supply for the USB VBUS. An overview of USB implementation as OTG is presented in Figure 27-8.

  When acting as an A-device, power must be supplied to VBUS. The power supply must be able to deliver 8 mA, 100 mA, or up to 500 mA, depending on the requirements of the devices in the TPL.

  When acting as a B-device, power must not be supplied to VBUS. VBUS pulsing can be performed by the USB module or by a capable power supply.

27.3.4.2 VBUS REGULATOR Interface

The VBUSON output can be used to control an off-chip 5V VBUS regulator. The VBUSON pin is controlled by the VBUSON bit (U1OTGCON<3>). VBUSON appears in Figure 27-7 and Figure 27-8.

Figure 27-6: Overview of USB Implementation as a Device

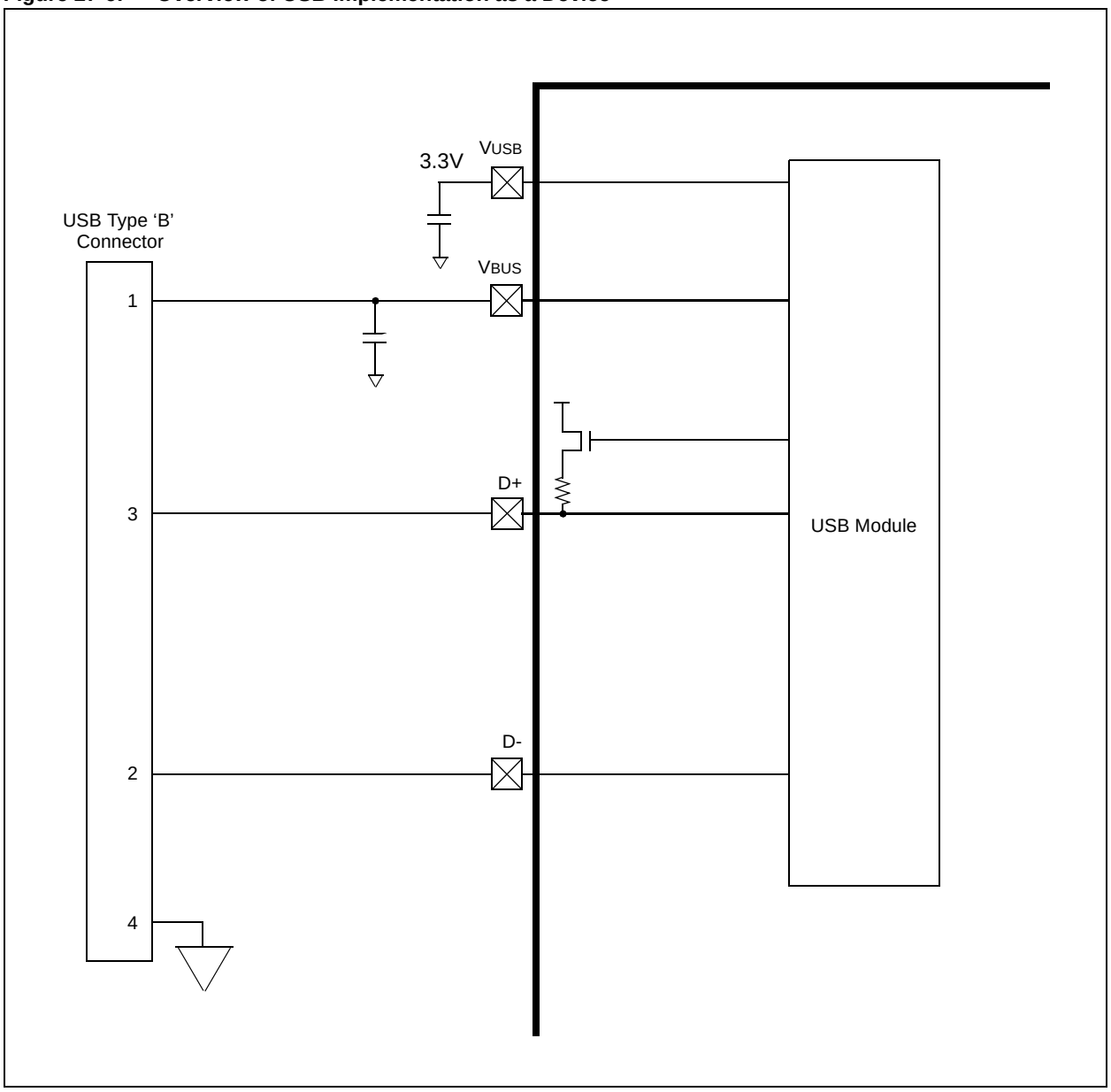


Figure 27-7: Overview of USB Implementation as a Host

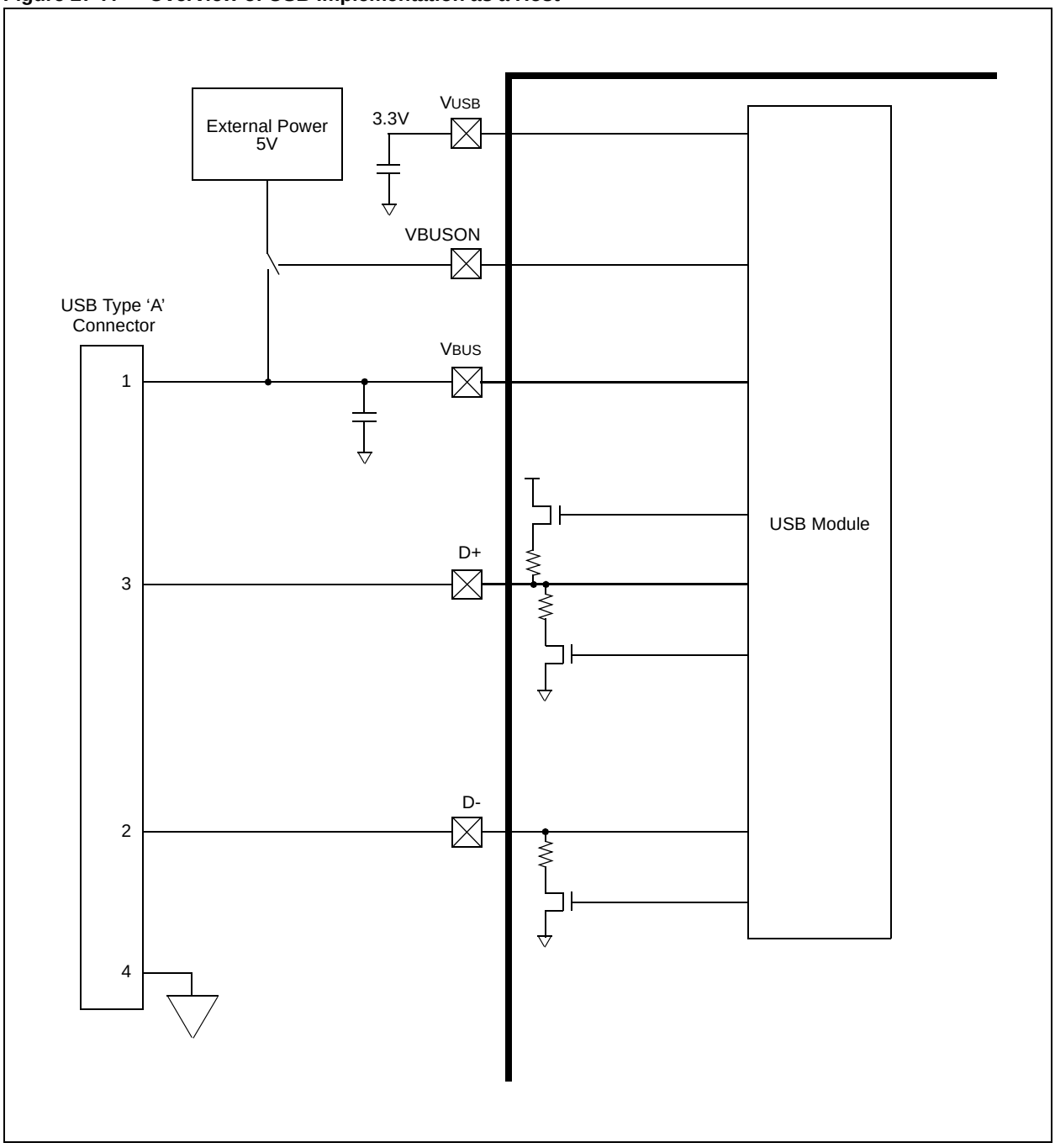
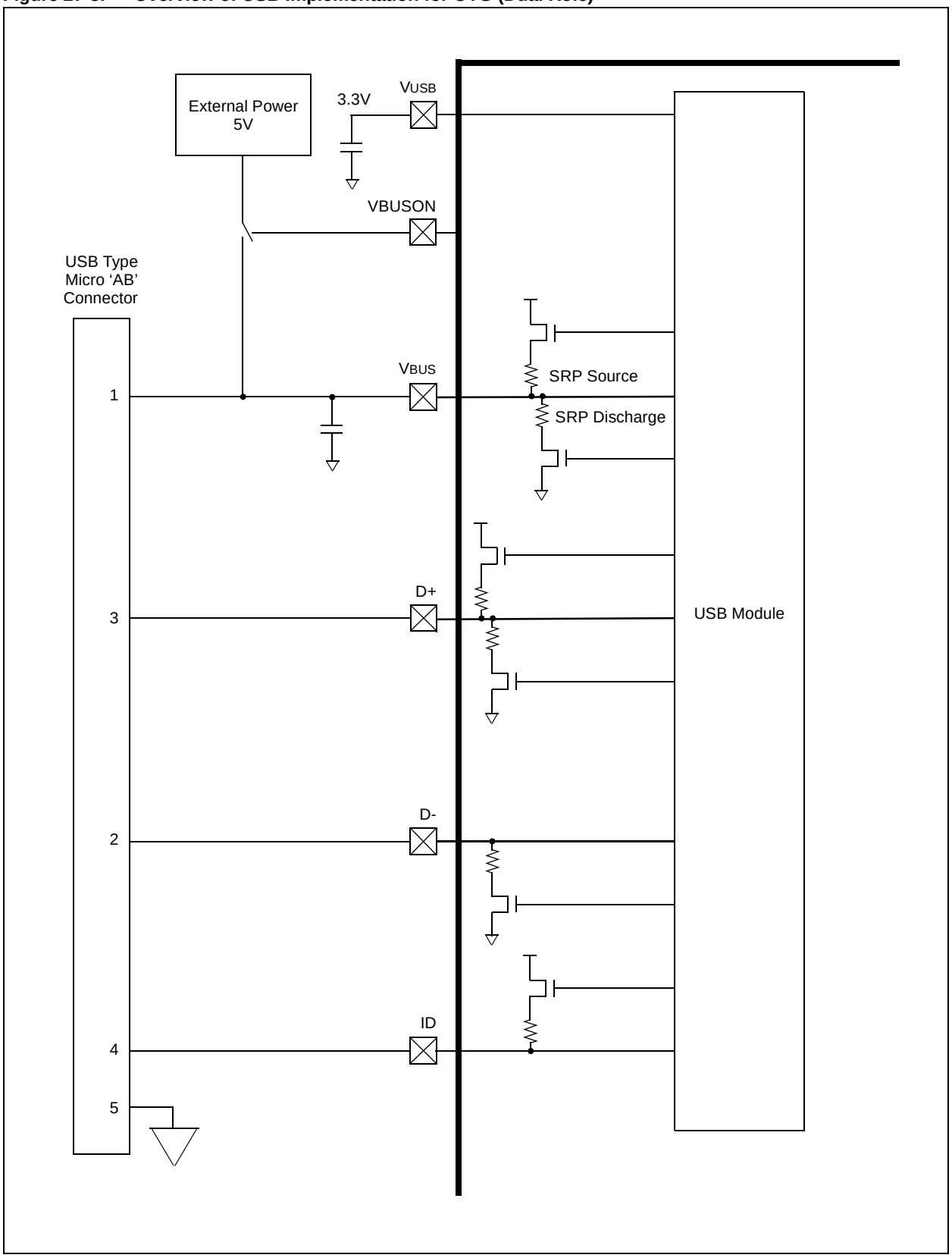


Figure 27-8: Overview of USB Implementation for OTG (Dual Role)



27.3.5 Module Initialization

This section describes the steps that must be taken to properly initialize the OTG USB module.

27.3.5.1 Enabling the USB Hardware

In order to use the USB peripheral, software must set the USBPWR bit (U1PWRC<0>) to '1'. This may be done in start-up boot sequence.

USBPWR is used to initiate the following actions:

- Start the USB clock
- Allow the USB interrupt to be activated
- Select USB as the owner of the necessary IO pins
- Enable the USB transceiver
- Enable the USB comparators

The USB module and internal registers are reset when USBPWR is cleared. Consequently, the appropriate initialization process must be performed whenever the USB module is enabled, as described in the following subsections. Otherwise, any configuration packet sent to the USB module will be NAK'd, by hardware, until the module is configured.

27.3.5.2 Initializing the BDT

All descriptors for a given endpoint and direction must be initialized prior to enabling the endpoint (for that direction). After a reset, all endpoints are disabled and start with the EVEN buffer for transmit and receive directions.

Transmit descriptors must be written with the UOWN bit cleared to '0' (owned by software). All other transmit descriptor setup may be performed anytime prior to setting the UOWN bit to '1'.

Receive descriptors must be fully initialized to receive data. This means that memory must be reserved for received packet data. The pointer to that memory (Physical Address), and the size reserved in bytes, must be written to the descriptor. The receive descriptor UOWN bit should be initialized to '1' (owned by Hardware). The DTS and STALL bits should also be configured appropriately.

If a transaction is received and the descriptor's UOWN bit is '0' (owned by software), the USB module returns a NAK handshake to the host. Usually, this causes the host to retry the transaction.

27.3.5.3 USB Enable/Mode Bits

USB mode of operation is controlled by the following enable bits: OTGEN (U1OTGCON<2>), HOSTEN (U1CON<3>), and USBEN/SOFEN (U1CON<0>).

- OTGEN:

  OTGEN selects whether the PIC32MX is to act as an OTG part (OTGEN = 1) or not. OTG devices support SRP and HNP in hardware with Firmware management and have direct control over the data-line pull-up and pull-down resistors.
- HOSTEN:

  HOSTEN controls whether the part is acting in the role of USB Host (HOSTEN = 1) or USB Device (HOSTEN = 0). Note that this role may change dynamically in an OTG application.
- USBEN/SOFEN:

  USBEN controls the connection to USB when the USB module is not configured as a host.

  If the USB module is configured as a host, SOFEN controls whether the host is active on the USB link and sends SOF tokens every 1 ms.

Note:	The other USB module control registers should be properly initialized before enabling USB via these bits.

27.3.6 Device Operation

All communication on the USB is initiated by the host. Therefore, in device mode, when USB is enabled USBEN = 1 (U1CON<0>), endpoint 0 must be ready to receive control transfers. Initialization of the remaining endpoints, descriptors, and buffers can be delayed until the host selects a configuration for the device. Refer to Chapter 9 of the "*Universal Serial Bus Specification, Revision 2.0*" for more information on this subject.

The following steps are performed to respond to a USB transaction:

1. Software pre-initializes the appropriate BDs, and sets the UOWN bits to '1' to be ready for a transaction.
2. Hardware receives a TOKEN PID (IN, OUT, SETUP) from the USB host, and checks the appropriate BD.
3. If the transaction will be transmitted (IN), the module reads packet data from data memory.
4. Hardware receives a DATA PID (DATA0/1), and sends or receives the packet data.
5. If a transaction is received (SETUP, OUT), the module writes packet data to data memory.
6. The module issues, or waits for, a handshake PID (ACK, NAK, STALL), unless the endpoint is setup as an isochronous endpoint (EPHSHK bit UEPMx<0> is cleared).
7. The module updates the BD, and writes the UOWN bit to '0' (SW owned).
8. The module updates the U1STAT register, and sets the TRNIF interrupt.
9. Software reads the U1STAT register, and determines the endpoint and direction for the transaction.
10. Software reads the appropriate BD, completes all necessary processing, and clears the TRNIF interrupt.

Note:	For transmitted (IN) transactions (host reading data from the device), the read data must be ready when the Host begins USB signaling. Otherwise, the USB module will send a NAK handshake if UOWN is '0'.

27.3.6.1 Receiving an IN Token in Device Mode

Perform the following steps when an IN token is received in Device mode:

1. Attach to a USB host and enumerate as described in Chapter 9 of the USB 2.0 specification.
2. Populate the data buffer with the data to send to the host.
3. In the appropriate (EVEN or ODD) transmit buffer descriptor for the desired endpoint:
   a. Set up the control bit field (BDnSTAT) with the correct data toggle (DATA0/1) value and the byte count of the data buffer.
   b. Set up the address bit field (BDnADR) with the starting address of the data buffer.
   c. Set the UOWN bit field to '1'.
4. When the USB module receives an IN token, it automatically transmits the data in the buffer. Upon completion, the module updates the status bit field (BDnSTAT), and sets the transfer complete interrupt (U1IR<TRNIF>).

27.3.6.2 Receiving an OUT Token in Device Mode

Perform the following steps when an OUT token is received in Device mode:

1. Attach to a USB host and enumerate as described in Chapter 9 of the USB 2.0 specification.
2. Create a data buffer with the amount of data you are expecting from the host.
3. In the appropriate (EVEN or ODD) transmit buffer descriptor for the desired endpoint:
   a. Set up the status bit field (BDnSTAT) with the correct data toggle (DATA0/1) value and the byte count of the data buffer.
   b. Set up the address bit field (BDnADR) with the starting address of the data buffer.
   c. Set the UOWN bit of the status bit field to '1'.
4. When the USB module receives an OUT token, it will automatically transfer the data the host sent into the buffer. Upon completion, the module updates the status bit field (BDnSTAT), and sets the transfer complete interrupt (U1IR<TRNIF>).

27.4 HOST MODE OPERATION

In Host mode, only endpoint 0 is used (all other endpoints should be disabled). Since the host initiates all transfers, the BD does not require immediate initialization. However, the BDs must be configured before a transfer is initiated – which is done by writing to the U1TOK register.

The following sections describe how to perform common Host mode tasks. In Host mode, USB transfers are invoked explicitly by the host software. The host software is responsible for initiating the setup, data, and status stages of all control transfers. The acknowledge (ACK or NAK) is generated automatically by the hardware, based on the CRC. Host software is also responsible for scheduling packets so that they do not violate USB protocol. All transfers are performed using the Endpoint 0 Control register (U1EP0) and BDs.

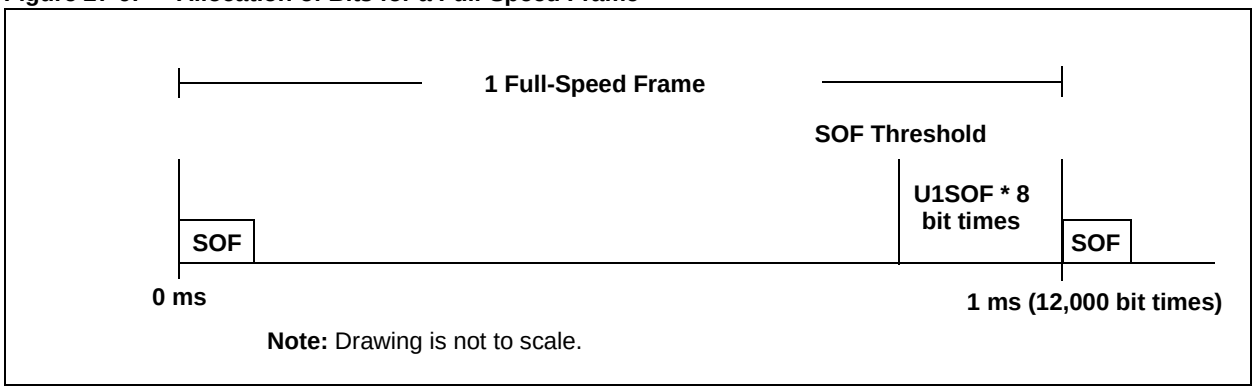
27.4.1 Configuring the SOF Threshold

The module counts down the number of bits that could be transmitted within the current USB full-speed frame. Since 12,000 bits can be transmitted during the 1 ms frame time, a counter (not visible to software) is loaded with the value ‘12,000’ at the start of each frame. The counter decrements once for each bit time in the frame. When the counter reaches zero, the next frame’s SOF packet is transmitted, see Figure 27-9.

The SOF threshold register (U1SOF) is used to ensure that no new tokens are started too close to the end of a frame. This prevents a conflict with the next frame’s SOF packet. When the counter reaches the threshold value of the U1SOF register (the value in the U1SOF register is in terms of bytes), no new tokens are started until after the SOF has been transmitted. Thus, the USB module attempts to ensure that the USB link is idle when the SOF token needs to be transmitted.

This implies that the value programmed into the U1SOF register must reserve enough time to ensure the completion of the worst-case transaction. Typically, the worst-case transaction is an IN token followed by a maximum-sized data packet from the target, followed by the response from the host. If the host is targeting a low-speed device that is bridging through a full-speed hub, the transaction will also include the special PRE token packets.

Figure 27-9: Allocation of Bits for a Full-Speed Frame



Note: Drawing is not to scale.

Table 27-3 and Table 27-4 show examples of calculating worst-case bit times.

Note 1: While the U1SOF register value is described in terms of bytes, these examples show the result in terms of bits.

2: In the second table, the IN, DATA, and HANDSHAKE packets are transmitted at low speed (8 times slower than full speed).

3: These calculations do not take the possibility that the packet data needs to be bit-stuffed for NRZI encoding into account.

.

Table 27-3: Example of SOF Threshold Calculation: Full Speed

Packet	Fields	Bits
IN	SYNC, PID, ADDR, ENDP, CRC5, EOP	35
Turnaround(1)		8
DATA	SYNC, PID, DATA(2), CRC16, EOP	547
Turnaround		2
HANDSHAKE	SYNC, PID, EOP	19
Inter-packet		2
Total		613

Note 1: Inter-packet delay of 2. An additional 5.5 bit times of latency is added to represent a worst-case propagation delay through 5 hubs.

2: Using 64-bytes maximum packet size for this example calculation.

Table 27-4: Example of SOF Threshold Calculation: Low Speed Via Hub

Packet	Fields	Bits	FS Bits
PRE	SYNC, PID	16	16
Hub setup		4	4
IN	SYNC, PID, ADDR, ENDP, CRC5, EOP	35	280
Turnaround(1)		8	8
DATA	SYNC, PID, DATA(2), CRC16, EOP	99	792
Turnaround		2	2
PRE	SYNC, PID	16	16
HANDSHAKE	SYNC, PID, EOP	19	152
Inter-packet		2	2
Total			1272

Note 1: Inter-packet delay of 2. An additional 5.5 bit times of latency is added to represent a worst-case propagation delay through 5 hubs.

2: Packets limited to 8-bytes maximum in Low-Speed mode.

Note: Refer to **Section 5.11.3 “Calculating Bus Transaction Times”** in the USB 2.0 specification for details on calculating bus transaction time.

27.4.2 Enabling Host Mode and Discovering a Connected Device

To enable Host mode, perform the following steps:

1. Enable Host mode (U1CON<HOSTEN> = 1).
   This enables the D+ and D- pull-down resistors, and disables the D+ and D- pull-up resistors. To reduce noise on the bus, disable the SOF packet generation by writing the SOF enable bit to '0' (U1CON<SOFEN> = 0).
2. Enable the device attach interrupt (U1IE<ATTACHIE> = 1).
3. Wait for the device attach interrupt (U1IR<ATTACHIF>).
   This is signaled by the USB device changing the state of D+ or D- from '0' to '1' (SE0 to JSTATE). After it occurs, wait for the device power to stabilize (10 ms is minimum, 100 ms is recommended).
4. Check the state of the JSTATE and SE0 bits in the control register U1CON.
   If U1CON<JSTATE> is '0', the connecting device is low speed; otherwise, the device is full speed.
5. If the connecting device is low speed, set the low-speed enable bit in the address register (U1ADDR<LSPDEN>= 1), and the low-speed bit in the Endpoint 0 Control register (U1EP0<LSPD> = 1). But, if the device is full speed, clear these bits.
6. Reset the USB device by sending the Reset signaling for at least 50 ms (U1CON<USBRST> = 1). After 50 ms, terminate the Reset (U1CON<USBRST> = 0).
7. Enable SOF packet generation to keep the connected device from going into Suspend (U1CON<SOFEN> = 1).
8. Wait 10 ms for the device to recover from Reset.
9. Perform enumeration as described in Chapter 9 of the USB 2.0 specification.

27.4.2.1 Host Transactions

When acting as a host, a transaction consists of the following:

1. Software configures the appropriate BD (Endpoint n, DIR, PPBI), and sets the UOWN bit to '1' (HW owned).
2. Software checks the state of TOKBUSY (U1CON<5>) to verify that any previous transaction has completed
3. Software writes the address of the target device in the U1ADDR register.
4. Software writes the endpoint number and the desired TOKEN PID (IN, OUT, or SETUP) to the U1TOK register.
5. Hardware reads the BD to determine the appropriate action, and to obtain the pointer to data memory.
6. Hardware issues the correct TOKEN PID (IN, OUT, SETUP) on the USB link.
7. If the transaction is a transmit transaction (OUT, SETUP), the USB module reads the packet data out of data memory. Then the module follows with the desired DATA PID (DATA0/DATA1) and packet data.
8. If the transaction is a receive transaction (IN), the USB module waits to receive the DATA PID and packet data. Hardware writes the packet data to memory.
9. Hardware issues or waits for a Handshake PID (ACK, NAK, or STALL), unless the end-point is set up as an Isochronous Endpoint (EPHSHK bit U1EPx<0> is cleared).
10. Hardware updates the BD, and writes the UOWN bit to '0' (SW owned).
11. Hardware updates the U1STAT register, and sets the TRNIF (U1IR<3>) interrupt.
12. Hardware reads the next BD (EVEN or ODD) to see whether it is owned by the USB module. If it is, hardware begins the next transaction.
13. Software should read the U1STAT register, and then clear the TRNIF interrupt.

If Software does not set the UOWN bit to '1' in the appropriate BD prior to writing the U1TOK register, the module will read the descriptor and do nothing.

27.4.3 Completing a Control Transaction to a Connected Device

Complete all of the following steps to discover a connected device:

1. Set up the Endpoint Control register for bidirectional control transfers, U1EP0<4:0> = 0x0D.
2. Place an 8-byte of the device setup packet in the appropriate memory buffer. See Chapter 9 of the USB 2.0 specification for information on the device framework command set.
3. Initialize the current (EVEN or ODD) TX EP0 BD to transfer the 8 byte device framework command (for example, a `GET DEVICE DESCRIPTOR` command).
   a. Set the BD status (BD0STAT) to 0x8008 – UOWN bit set, byte count of 8.
   b. Set the BD data buffer address (BD0ADR) to the starting address of the 8-byte memory buffer containing the command, if it is not already initialized.
4. Set the USB address of the target device in the address register U1ADDR<6:0>. After a USB bus Reset, the device USB address will be zero. After enumeration, it must be set to another value, between 1 and 127, by the host software.
5. Write the token register with a SETUP command to Endpoint 0, the target device's default control pipe (U1TOK = 0xD0). This will initiate a SETUP token on the bus followed by a data packet. The device handshake will be returned in the PID field of BD0STAT after the packets complete. When the module updates BD0STAT, a transfer done interrupt will be asserted (U1IR<TRNIF>). This completes the setup stage of the setup transfer as described in Chapter 9 of the USB specification.
6. To initiate the data stage of the setup transaction (for example, get the data for the `GET DEVICE DESCRIPTOR` command), set up a buffer in memory to store the received data.
7. Initialize the current (EVEN or ODD) RX or TX (RX for IN, TX for OUT) EP0 BD to transfer the data.
   a. Set the BD status (BD0STAT) UOWN bit to '1', data toggle (DTS) to DATA1 and byte count to the length of the data buffer.
   b. Set the BD data buffer address (BD0ADR) to the starting address of the data buffer if it is not already initialized.
8. Write the Token register with the appropriate IN or OUT token to Endpoint 0, the target device's default control pipe) for example, an IN token for a `GET DEVICE DESCRIPTOR` command (U1TOK = 0x90). This will initiate an IN token on the bus followed by a data packet from the device to the host. When the data packet completes, the BD0STAT is written and a transfer done interrupt will be asserted (U1IR<TRNIF>). For control transfers with a single packet data phase, this completes the data phase of the setup transaction. If more data needs to be transferred, return to step 8.
9. To initiate the status stage of the setup transaction, set up a buffer in memory to receive or send the zero length status phase data packet.
10. Initialize the current (EVEN or ODD) TX EP0 BD to transfer the status data.
    a. Set the BD status (BD0STAT) to 0x8000 – UOWN bit to '1', data toggle (DTS) to DATA0 and byte count to '0'.
    b. Set the BDT buffer address field to the start address of the data buffer.
11. Write the Token register with the appropriate IN or OUT token to Endpoint 0, the target device's default control pipe) for example, an OUT token for a `GET DEVICE DESCRIPTOR` command (U1TOK = 0x10). This will initiate a token on the bus, followed by a zero length data packet from the host to the device. When the data packet completes, the BD is updated with the handshake from the device, and a transfer done interrupt will be asserted (U1IR<TRNIF>). This completes the status phase of the setup transaction.

Note: Some devices can only effectively respond to one transaction per frame.

27.4.4 Data Transfer with a Target Device

Complete all of the following steps to discover and configure a connected device.

1. Write the EP0 Control register (U1EP0) to enable transmit and receive transfers as appropriate with handshaking enabled (unless isochronous transfers are to be used). If the target device is a low-speed device, also set the Low-Speed Enable bit (U1EP0<LSPDEN>). If you want the hardware to automatically retry indefinitely if the target device asserts a NAK on the transfer, clear the Retry Disable bit (U1EP0<RETRYDIS>).

Note:	Use of automatic indefinite retries can lead to a deadlock condition if the device never responds.

2. Set up the current Buffer Descriptor (EVEN or ODD) in the appropriate direction to transfer the desired number of bytes.
3. Set the address of the target device in the address register (U1ADDR<6:0>).
4. Write the Token register (U1TOK) with an IN or OUT token as appropriate for the desired endpoint. This triggers the module's transmit state machines to begin transmitting the token and the data.
5. Wait for the transfer done interrupt (U1IR<TRNIF>). This will indicate that the BD has been released back to the microprocessor and the transfer has completed. If the retry disable bit is set, the handshake (ACK, NAK, STALL or ERROR (0xf)) will be returned in the BD PID field. If a stall interrupt occurs, then the pending packet must be dequeued and the error condition in the target device cleared. If a detach interrupt occurs (SE0 for more than 2.5 μs), then the target has detached (U1IR<DETACHIF>).
6. Once the transfer done interrupt (U1IR<TRNIF>) occurs, the BD can be examined and the next data packet queued by returning to step 2.

Note:	USB speed, transceiver and pull-ups should only be configured during the module set-up phase. It is not recommended to change these settings while the module is enabled.

27.4.4.1 USB Link States

Three possible link states are described in the following subsections:

- Reset
- Idle and Suspend
- Resume Signalling

27.4.4.1.1 Reset

As a host, software is required to drive Reset signaling. It may do this by setting USBRST (U1CON<4>). As per the USB specification, the host must drive the Reset for at least 50 ms. (This does not have to be continuous Reset signaling. Refer to the USB 2.0 specification for more information.) Following Reset, the host must not initiate any downstream traffic for another 10 ms.

As a device, the USB module will assert the URSTIF (U1IR<0>) interrupt when it has detected Reset signaling for 2.5 μs. Software must perform any Reset initialization processing at this time. This includes setting the Address register to 0x00 and enabling Endpoint 0. The URSTIF interrupt will not be set again until the Reset signaling has gone away and then has been detected again for 2.5 μs.

27.4.4.1.2 Idle and Suspend

The Idle state of the USB is a constant J state. When the USB has been Idle for 3 ms, a device should go into Suspend state. During active operation, the USB host will send a SOF token every 1 ms, preventing a device from going into Suspend state.

Once the USB link is in the Suspend state, a USB host or device must drive resume signaling prior to initiating any bus activity. (The USB link may also be disconnected.)

As a USB host, software should consider the link in Suspend state as soon as software clears the SOFEN (U1CON<0>).

As a USB device, hardware will set the IDLEIF (U1IR<4>) interrupt when it detects a constant Idle on the bus for 3 ms. Software should consider the link in Suspend state when the IDLEIF interrupt is set.

When a Suspend condition has been detected, the software may wish to place the USB hardware in a Suspend mode by setting USUSPEND (U1PWRC<1>). The hardware Suspend mode gates the USB module's 48 MHz clock and places the USB transceiver in a Low-Power mode.

Additionally, the user may put the PIC32MX into Sleep mode while the link is suspended.

27.4.4.1.3 Driving Resume Signaling

If software wants to wake the USB from Suspend state, it may do so by setting RESUME (U1CON<2>). This will cause the hardware to generate the proper resume signaling (including finishing with a low-speed EOP if a host).

A USB device should not drive resume signaling unless the Idle state has persisted for at least 5 ms. The USB host also must have enabled the function for remote wake-up.

Software must set RESUME for 1-15 ms if a USB device, or >20 ms if a USB host, then clear it to enable remote wake-up. For more information on RESUME signaling, see Section 7.1.7.7, 11.9 and 11.4.4 in the USB 2.0 specification.

Writing RESUME will automatically clear the special hardware Suspend (low-power) state.

If the part is acting as a USB host, software should, at minimum, set the SOFEN (U1CON<0>) after driving its resume signaling. Otherwise, the USB link would return right back to the Suspend state. Also, software must not initiate any downstream traffic for 10 ms following the end of resume signaling.

27.4.4.1.4 Receiving Resume Signaling

When the USB logic detects resume signaling on the USB bus for 2.5 μs, hardware will set the RESUMEIF (U1IR<5>) interrupt.

A device receiving resume signaling must prepare itself to receive normal USB activity. A host receiving resume signaling must immediately start driving resume signaling of its own. The special hardware Suspend (low-power) state is automatically cleared upon receiving any activity on the USB link.

Reception of any activity on the USB link (this may be due to resume signaling or a link disconnect) while the PIC32MX is in Sleep mode will cause the ACTVIF (U1OTGIR<4>) interrupt to be set. This will cause wake-up from Sleep.

27.4.4.1.5 SRP Support

SRP support is not required by non-OTG applications. SRP may only be initiated at full speed. Refer to the On-The-Go Supplement specification for more information regarding SRP.

An OTG A-device or embedded host may decide to power-down the VBUS supply when it is not using the USB link. Software may do this by clearing VBUSON (U1OTGCON<3>). When the VBUS supply is powered down, the A-device is said to have ended a USB session.

Note:	When the A-device powers down the VBUS supply, the B-device must disconnect its pull-up resistor unless signalling a desire to become host during HNP negotiation. Refer to **Section 27.4.4.1.6 "HNP"**.

An OTG A-device or embedded host may repower the VBUS supply at any time to initiate a new session. An OTG B-device may also request that the OTG A-device repower the VBUS supply to initiate a new session. This is the purpose of the SRP.

Prior to requesting a new session, the B-device must first check that the previous session has definitely ended. To do this, the B-device must check that:

1. VBUS supply is below the session end voltage.
2. Both D+ and D- have been low for at least 2 ms.

The B-device will be notified of condition 1 by the SESENDIF (U1OTGIR<2>) interrupt.

Software can use the LSTATEIF (U1OTGIR<5>) bit and the 1 ms timer to identify condition 2.

The B-device may aid in achieving condition 1 by discharging the VBUS supply through a resistor. Software may do this by setting VBUSDIS (U1OTGCON<0>).

The B-device then proceeds by pulsing the D+ data line. Software should do this by setting DPPULUP (U1OTGCON<7>). The data line should be held high for 5-10 ms.

After these initial conditions are met, the B-device may begin requesting the new session. It begins by pulsing the VBUS supply. Software should do this by setting VBUSCHG (U1OTGCON<1>).

When an A-device detects SRP signaling (either via the ATTACHIF (U1IR<6>) interrupt or via the SESVDIF (U1OTGIR<3>) interrupt), the A-device must restore the VBUS supply by setting VBUSON (U1OTGCON<3>).

The B-device should not monitor the state of the VBUS supply while performing VBUS supply pulsing. Afterwards, if the B-device does detect that the VBUS supply has been restored (via the SESVDIF (U1OTGIR<3>) interrupt), it must reconnect to the USB link by pulling up D+. The A-device must complete the SRP by enabling VBUS and driving reset signalling.

27.4.4.1.6 HNP

An OTG application with a micro-AB receptacle must support HNP. HNP allows an OTG B-device to temporarily become the USB host. The A-device must first enable HNP in the B-device. HNP may only be initiated at full-speed. Refer to the On-The-Go supplement for more information regarding HNP.

After being enabled for HNP by the A-device, the B-device can request to become the host any time that the USB link is in Suspend state by simply indicating a disconnect. Software may accomplish this by clearing the DPPULUP bit (U1OTGCON<7>).

When the A-device detects the disconnect condition (via the URSTIF (U1IR<0>) interrupt), the A-device may allow the B-device to take over as host. The A-device does this by signaling connect as a full-speed device. Software may accomplish this by disabling host operation, HOSTEN = 0 (U1CON<3>), and connecting as a device (DPPULUP = 1). If the A-device instead responds with resume signaling, the A-device will remain as host.

When the B-device detects the connect condition (via ATTACHIF (U1IR<6>), the B-device becomes host. The B-device drives Reset signaling prior to using the bus.

When the B-device has finished in its role as host, it stops all bus activity and turns on its D+ pull-up resistor by disabling host operations (HOSTEN = 0) and reconnecting as a device (DPPULUP = 1).

Then the A-device detects a Suspend condition (Idle for 3 ms), the A-device turns off its D+ pull-up. Alternatively the A-device may also power-down the VBUS supply to end the session.

When the A-device detects the connect condition (via ATTACHIF), the A-device resumes host operation, and drives Reset signaling.

27.4.4.2 Clock Requirements

For proper USB operation, the USB module must be clocked with a 48 MHz clock. This clock source is used to generate the timing for USB transfers; it is the clock source for the SIE. The control registers are clocked at the same speed as the CPU (refer to Figure 27-1).

The USB module clock is derived from the Primary Oscillator (POSC) for USB operation. A USB PLL and input prescalers are provided to allow 48 MHz clock generation from a wide variety of input frequencies. The USB PLL allows the CPU and the USB module to operate at different frequencies while both use the POSC as a clock source. To prevent buffer overruns and timing issues, the CPU core must be clocked at a minimum of 16 MHz.

The USB module can also use the on-board Fast RC oscillator (FRC) as a clock source. When using this clock source, the USB module will not meet the USB timing requirements. The FRC clock source is intended to allow the USB module to detect a USB wake-up and report it to the interrupt controller when operating in low-power modes. The USB module must be running from the Primary oscillator before beginning USB transmissions

27.5 INTERRUPTS

The USB module uses interrupts to signal USB events such as a change in status, data received and buffer empty events, to the CPU. Software must be able to respond to these interrupts in a timely manner.

27.5.1 Interrupt Control

Each interrupt source in the USB module has an interrupt flag bit and a corresponding enable bit. In addition, the UERRIF bit (U1IR<1>) is a logical OR of all the enabled error flags and is read-only. The UERRIF bit can be used to poll the USB module for events while in an Interrupt Service Routine (ISR).

27.5.2 USB Module Interrupt Request Generation

The USB module can generate interrupt requests from a variety of events. To interface these interrupts to the CPU, the USB interrupts are combined such that any enabled USB interrupt will cause a generic USB interrupt (if the USB interrupt is enabled) to the interrupt controller, see Figure 27-11. The USB ISR must then determine which USB event(s) caused the CPU interrupt and service them appropriately. There are two layers of interrupt registers in the USB module. The top level of bits consists of overall USB status interrupts in the U1OTGIR and U1IR registers. The U1OTGIR and U1IR bits are individually enabled through the corresponding bits in the U1OTGIE and U1IE registers. In addition, the USB Error Condition bit (UERRIF) passes through any interrupt conditions in the U1EIR register enabled via the U1EIE register bits.

27.5.3 Interrupt Timing

Interrupts for transfers are generated at the end of the transfer. Figure 27-10 shows some typical event sequences that can generate a USB interrupt and when that interrupt is generated. There is no mechanism by which software can manually set an interrupt bit.

The values in the Interrupt Enable registers (U1IE, U1EIE, U1OTGIE) only affect the propagation of an interrupt condition to the CPU's interrupt controller. Even though an interrupt is not enabled, interrupt flag bits can still be polled and serviced.

27.5.4 Interrupt Servicing

Once an interrupt bit has been set by the USB module (in U1IR, U1EIR or U1OTGIR), it must be cleared by software by writing a '1' to the appropriate bit position to clear the interrupt. The USB Interrupt, USBIF (IFS1<25>), must be cleared before the end of the ISR.

Figure 27-10: Typical Events for USB Interrupts

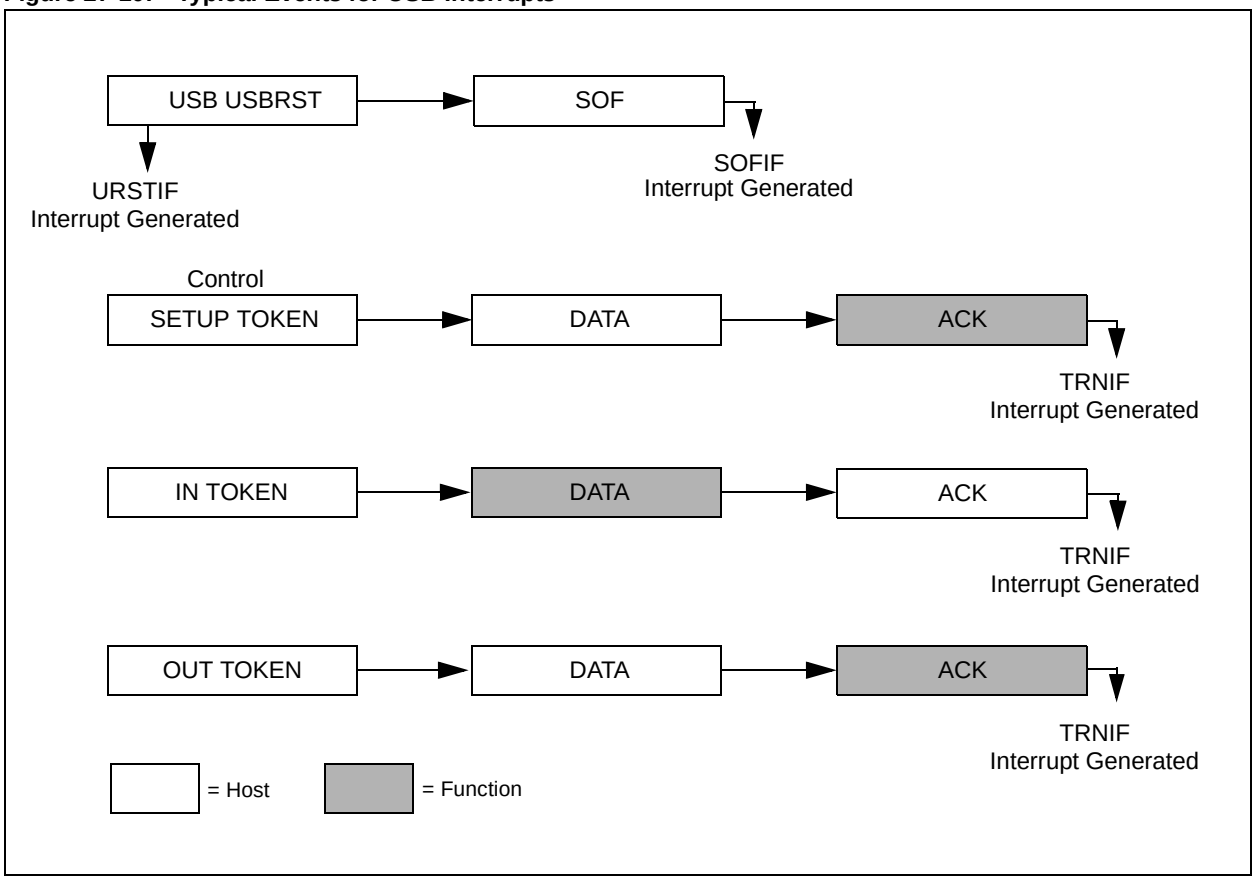
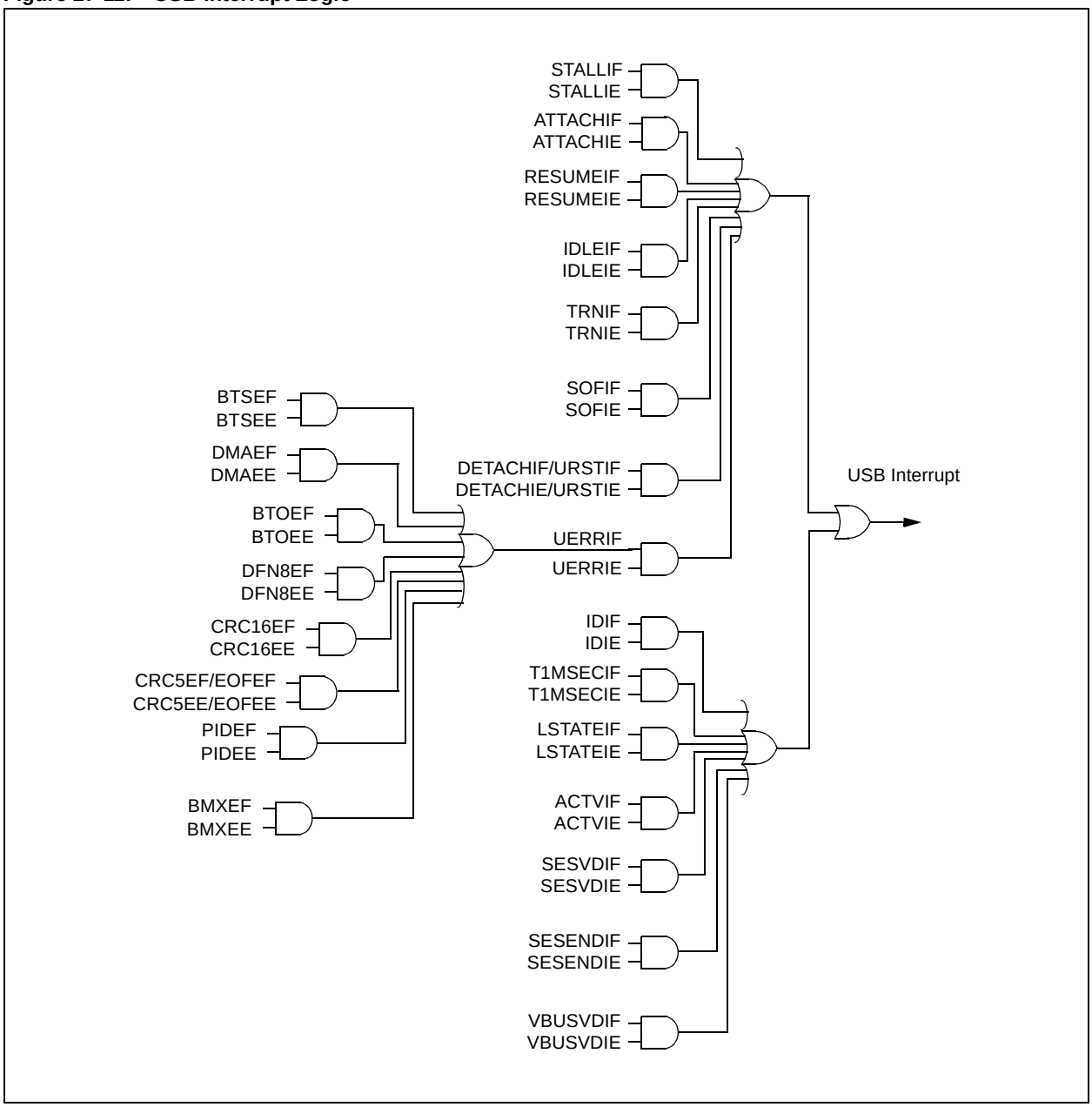


Figure 27-11: USB Interrupt Logic



27.6 I/O PINS

Table 27-5 summarizes the use of pins relating to the USB module.

Table 27-5: Pins Associated with the USB Module

Mode	Pin Name	Module Control	Controlling Bit Field(1)	Required TRIS Bit Setting	Pin Type	Description
Embedded Host						
	D+	USBEN	—	—	U	Data line +
	D-	USBEN	—	—	U	Data line -
	VBUS	USBEN	—	—	P	Input for USB power, connects to OTG comparators
	VBUSON	USBEN	VBUSON	—	D, O	Output to control supply for VBUS
	VUSB	—	—	—	P	Power in for USB transceiver
	ID	USBEN	—	—	R	Reserved
Device						
	D+	USBEN	—	—	U	Data line +
	D-	USBEN	—	—	U	Data line -
	VBUS	USBEN	—	—	P	Input for USB power, connects to OTG comparators
	VBUSON	—	—	—	R	Reserved
	VUSB	—	—	—	P	Power in for USB transceiver
	ID	—	—	—	R	Reserved
OTG						
	D+	USBEN	—	—	U	Data line +
	D-	USBEN	—	—	U	Data line -
	VBUS	USBEN	VBUSCHG, VBUSDIS	—	A, I/O, P	Analog input for USB power, connects to OTG comparators
	VBUSON	USBEN	VBUSCHG, VBUSDIS VBUSON	—	D, O	Output to control supply for VBUS
	VUSB	—	—	—	P	Power in for USB transceiver
	ID	USBEN	—	—	D, I	OTG mode host/device select input

Legend:	I = Input	O = Output	A = Analog	D = Digital
	U = USB	P = Power	R = Reserved	

Note 1: All pins are subject to the device pin priority control. See the specific device data sheet for further information.

Table 27-5: Pins Associated with the USB Module (Continued)

Mode	Pin Name	Module Control	Controlling Bit Field(1)	Required TRIS Bit Setting	Pin Type	Description
USB Disabled						
	D+	USBEN	—	1	D, I	General purpose digital input
	D-	USBEN	—	1	D, I	General purpose digital input
	VBUS	USBEN	—	—	R	Reserved
	VBUSON	USBEN	—	0	D, O	General purpose digital input
	VBUSON	USBEN	—	1	D, I	General purpose digital output
	VUSB	USBEN	—	—	R	Reserved
	ID	USBEN	—	1	D, I	General purpose digital input
	ID	USBEN	—	0	D, O	General purpose digital output

Legend:	I = Input	O = Output	A = Analog	D = Digital
	U = USB	P = Power	R = Reserved	

Note 1: All pins are subject to the device pin priority control. See the specific device data sheet for further information.

27.7 OPERATION IN DEBUG AND POWER-SAVING MODES

Note: In this manual, a distinction is made between a power mode as it is used in a specific module, and a power mode as it is used by the device, e.g., Sleep mode of the Comparator and SLEEP mode of the CPU. To indicate which type of power mode is intended, uppercase and lowercase letters (Sleep, Idle, Debug) signify a module power mode, and all uppercase letters (SLEEP, IDLE, DEBUG) signify a device power mode.

27.7.1 Operation in SLEEP

Use of SLEEP mode is only recommended in two cases:

- USB module is disabled
- USB module is in a Suspend state

Placing the USB module in SLEEP mode while the bus is active can result in violating USB protocol.

When the device enters SLEEP mode, the clock to the USB module is maintained. The effect on the CPU clock source is dependent on the USB and CPU clock configuration.

- If the CPU and USB were using the Primary Oscillator (POSC) source, the CPU is disconnected from the clock source when entering SLEEP and the oscillator is left in Enabled state for the USB module.
- If the CPU was using a different clock source, that clock source is disabled on entering SLEEP, and the USB clock source is left Enabled.

To further reduce power consumption, the USB module can be placed in Suspend mode prior to placing the CPU in SLEEP. The effect on the CPU clock source is dependent on the USB and CPU clock configuration.

- If the CPU and USB were using the Primary Oscillator (POSC) source, the oscillator is disabled when the CPU enters SLEEP.
- If the CPU was not sharing POSC with the USB module, POSC will be disabled when the USB module enters Suspend. The CPU clock source will be disabled when the CPU enters SLEEP.

27.7.1.1 Bus Activity Coincident with Entering SLEEP Mode

Software is unable to predict bus activity therefore even when software has determined that the USB link is in a state safe for entering SLEEP, bus activity can still occur, potentially placing USB in a non-safe link state. The USLPGRD (U1PWRC<4>) and UACTPND (U1PWRC<7>) bits can be used to prevent this. Before entering the sensitive code region, software can set the GUARD bit so that hardware will prevent the device from entering SLEEP mode (by generating a wake-up event) if activity is detected or if there is a notification pending. UACTPND should be polled to ensure no interrupt is pending before attempting to enter SLEEP.

27.7.2 Operation in IDLE Mode

When the device enters IDLE mode, the behavior of the USB module is determined by the USBSIDL bit.

27.7.2.1 IDLE operation with USBSIDL Cleared

When the bit is clear, the clock to the CPU is gated off but the clock to the USB module is maintained when in IDLE mode. The USB module can therefore continue operation while the CPU is idled. When enabled USB interrupts are generated they will bring the CPU out of IDLE.

27.7.2.2 IDLE operation with USBSIDL Set

When the USBSIDL bit is set, the clock to the CPU and the clock to the USB module are both gated off. In this mode the USB module does not continue normal operation and has lower power consumption. Any USB activity can be used to generate an interrupt to bring the CPU out of IDLE.

To further increase power savings, the CPU clock source and USB clock sources can be switched to FRC be fore entering IDLE mode. This will cause the POSC module to power down. When the POSC module is reenabled, start-up delays will apply. This mode of operation should only be used when the bus is idle.

27.7.3 Operation in DEBUG Modes

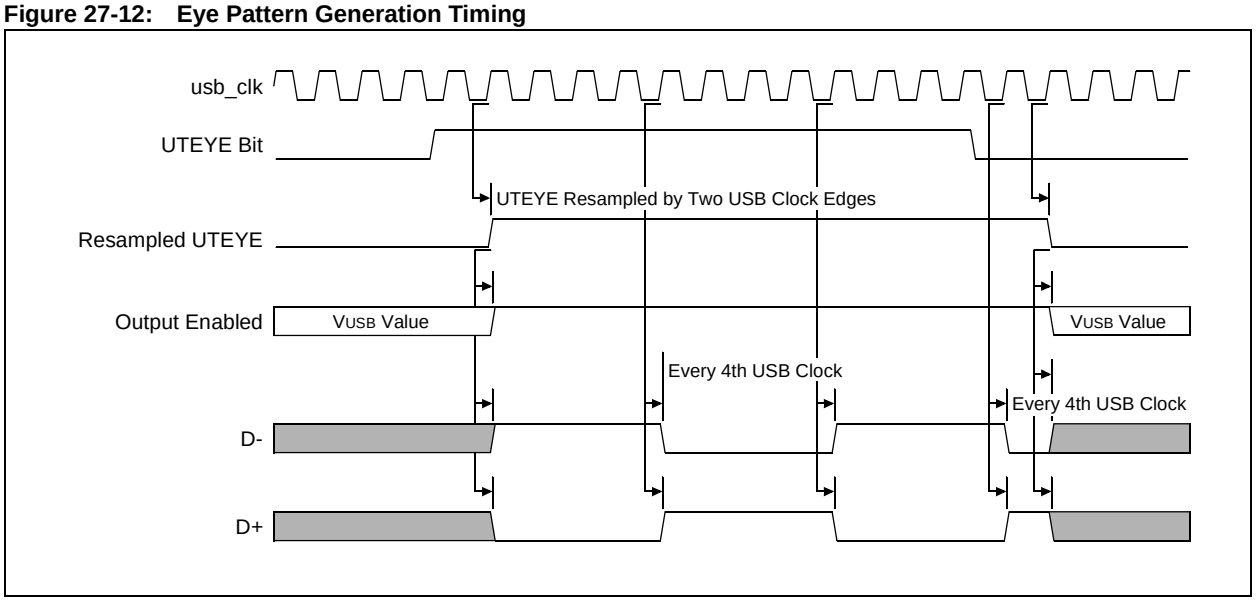
27.7.3.1 Eye Pattern

To assist with USB hardware debugging and testing, an eye pattern test generator is incorporated into the module. This pattern is generated by the module when the UTEYE bit (U1CNFG1<7>) is set. The USB module must be enabled, USBPWR (PWRC<0> = 1), the USB 48 MHz clock must be enabled, SUSPEND (U1PWRC<1>) = 0, and the module is not in Freeze mode.

Once the UTEYE bit is set, the module will start transmitting a **J-K-J-K** bit sequence. The bit sequence will be repeated indefinitely while the Eye Pattern Test mode is enabled (see Figure 27-12).

Note:	The UTEYE bit should never be set while the module is connected to an actual USB system. The mode is intended for board verification to aid with USB certification tests.

Figure 27-12: Eye Pattern Generation Timing



27.7.3.2 USB $\overline{OE}$ Monitor

The USB $\overline{OE}$ monitor indicates whether the USB is listening to the bus or actively driving the bus. This debug feature is enabled when the U1CNFG1<UOEMON> = 1.

The $\overline{OE}$ Monitoring is useful for initial system debugging as well as scope triggering during eye pattern generation tests.

27.8 EFFECTS OF A RESET

All forms of Reset force the USB module registers to the default state.

Note:	The USB module cannot ensure the state of the BDT, nor that of the packet data buffers contained in RAM, following a Reset.

27.8.1 Device Reset ($\overline{\text{MCLR}}$)

A device Reset forces all USB module registers to their Reset state. This turns the USB module off.

27.8.2 Power-on Reset (POR)

A POR Reset forces all USB module registers to their Reset state. This turns the USB module off.

27.8.3 Watchdog Timer Reset (WDT)

A WDT Reset forces all USB module registers to their Reset state. This turns the USB module off.

27.9 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the PIC32MX device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the USB OTG module are:

Title	Application Note #
USB Device Stack for PIC32 Programmer's Guide	AN1170
USB Mass Storage Class on an Embedded Device	AN1169
USB HID Class on an Embedded Device	AN1163
USB CDC Class on an Embedded Device	AN1164
USB Generic Function on an Embedded Device	AN1166
USB Embedded Host Stack Programmer's Guide	AN1141

Note: Please visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the PIC32MX family of devices.

27.10 REVISION HISTORY

Revision A (February 2008)

This is the initial released version of this document.

Revision B (April 2008)

Revised status to Preliminary; Revised U-0 to r-x; Revised Figure 27-1; Revised Table 27-5.

Revision C (July 2008)

Revised Registers 27-23 (IFS1) and 27-24 (IEC1); Revised Figures 27-3 and 27-4; Change Reserved bits from “Maintain as” to “Write”.

NOTES:

Section 28. Reserved for Future

NOTES:

Section 29. Real-Time Clock and Calendar

HIGHLIGHTS

This section of the manual contains the following topics:

29.1 INTRODUCTION

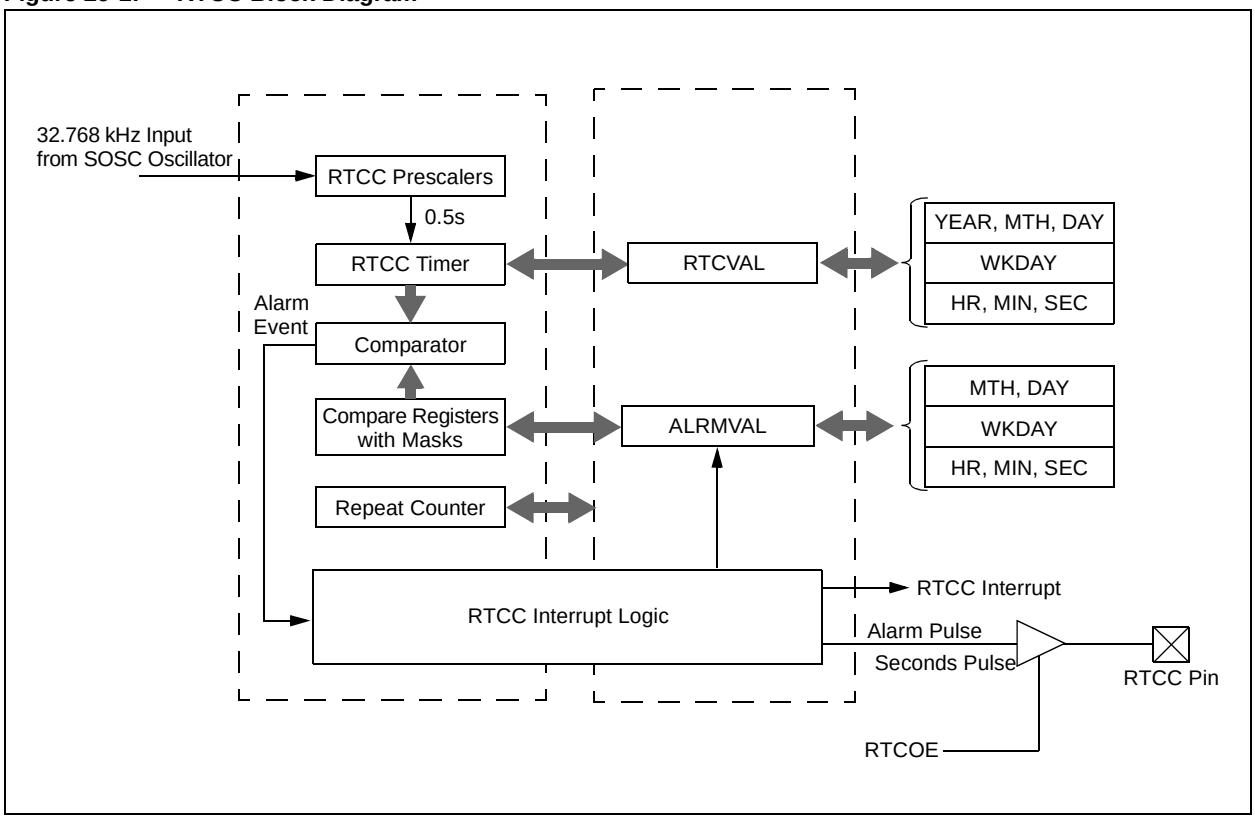
This section discusses the Real-Time Clock and Calendar (RTCC) hardware module, available on PIC32MX devices, and its operation. Listed below are some of the key features of this module:

- Time: Hours, Minutes and Seconds
- 24-hour format (military time)
- Visibility of one-half second period
- Provides calendar: Weekday, Date, Month and Year
- Alarm configurable for half a second, one second, 10 seconds, one minute, 10 minutes, one hour, one day, one week, one month, one year
- Alarm repeat with decrementing counter
- Alarm with indefinite repeat: chime
- Year Range: 2000 to 2099
- Leap Year Correction
- BCD format for smaller firmware overhead
- Optimized for long term battery operation
- Fractional second synchronization
- User calibration of the clock crystal frequency with auto-adjust
- Calibration range: ±0.66 seconds error per month
- Calibrates up to 260 ppm of crystal error
- Requirements: external 32.768 kHz Clock Crystal
- Alarm Pulse or Seconds Clock Output on the RTCC pin

This module provides real-time clock and calendar functions. RTCC is intended for applications where accurate time must be maintained for extended periods of time with minimum-to-no intervention from the CPU. The module is optimized for low-power usage in order to provide extended battery lifetime while keeping track of time.

The RTCC module is a 100-year clock and calendar with automatic leap year detection. The range of the clock is from 00:00:00 (midnight) on January 1, 2000 to 23:59:59 on December 31, 2099. The hours are available in 24-hour (military time) format. The clock provides a granularity of one second with half-second visibility to the user.

Figure 29-1: RTCC Block Diagram



29.2 STATUS AND CONTROL REGISTERS

The RTCC module registers includes the following Special Function Registers (SFRs):

The RTCCON and RTCALRM registers control the operation of the RTCC module.

- RTCCON: Control Register for the RTCC Module
  RTCCONCLR, RTCCONSET, RTCCONINV: Atomic Bit Manipulation, Write-only Registers for RTCCON
- RTCALRM: Control Register for the Alarm Functions of the RTCC Module
  RTCALRMCLR, RTCALRMSET, RTCALRMINV: Atomic Bit Manipulation, Write-only Registers for RTCALRM
- RTCTIME: RTCC Time Register, including Hour, Minutes and Seconds Fields.
  RTCTIMECLR, RTCTIMESET, RTCTIMEINV: Atomic Bit Manipulation, Write-only Registers for RTCTIME
- RTCDATE: RTCC Date Register, including Year, Month, Day and Weekday Fields.
  RTCDATECLR, RTCDATESET, RTCDATEINV: Atomic Bit Manipulation, Write-only Registers for RTCDATE
- ALRMTIME: RTCC Alarm Time Register, including Alarm Hour, Minutes and Seconds Fields
  ALRMTIMECLR, ALRMTIMESET, ALRMTIMEINV: Atomic Bit Manipulation, Write-only Registers for ALRMTIME
- ALRMDATE: RTCC Alarm Date Register, including Alarm Month, Day and Weekday Fields
  ALRMDATECLR, ALRMDATESET,ALRMDATEINV: Atomic Bit Manipulation, Write-only Registers for ALRMDATE
- IFS1: INT Controller Register signalling an Active RTCC Interrupt
  IFS1CLR, IFS1SET, IFS1INV: Atomic Bit Manipulation, Write-only Registers for IFS1
- IEC1: INT Controller Register enabling the RTCC Interrupt
  IEC1CLR, IEC1SET, IEC1INV: Atomic Bit Manipulation, Write-only Registers for IEC1
- IPC8: INT Controller Register for programming the RTCC Interrupt Priority and Subpriority
  IPC8CLR, IPC8SET, IPC8INV: Atomic Bit Manipulation, Write-only Registers for IPC8

The following table summarizes all RTCC-related registers. Corresponding registers appear after the summary, followed by a detailed description of each register.

Table 29-1: RTCC SFR Summary

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
RTCCON	31:24	—	—	—	—	—	—	CAL9:8>	
	23:16	CAL<7:0>							
	15:8	ON	FRZ	SIDL	—	—	—	—	—
	7:0	RTSECSEL	RTCCLKON	—	—	RTCWREN	RTCSYNC	HALFSEC	RTCOE
RTCCONCLR	31:0	Write clears selected bits in RTCCON, read yields undefined value							
RTCCONSET	31:0	Write sets selected bits in RTCCON, read yields undefined value							
RTCCONINV	31:0	Write inverts selected bits in RTCCON, read yields undefined value							
RTCALRM	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	ALRMEN	CHIME	PIV	ALRMSYNC	AMASK<3:0>			
	7:0	ARPT<7:0>							
RTCALRMCLR	31:0	Write clears selected bits in RTCALRM, read yields undefined value							
RTCALRMSET	31:0	Write sets selected bits in RTCALRM, read yields undefined value							
RTCALRMINV	31:0	Write inverts selected bits in RTCALRM, read yields undefined value							
RTCTIME	31:24	HR10<3:0>				HR01<3:0>			
	23:16	MIN10<3:0>				MIN01<3:0>			
	15:8	SEC10<3:0>				SEC01<3:0>			
	7:0	—	—	—	—	—	—	—	—
RTCTIMECLR	31:0	Write clears selected bits in RTCTIME, read yields undefined value							
RTCTIMESET	31:0	Write sets selected bits in RTCTIME, read yields undefined value							
RTCTIMEINV	31:0	Write inverts selected bits in RTCTIME, read yields undefined value							
RTCDATE	31:24	YEAR10<3:0>				YEAR01<3:0>			
	23:16	MONTH10<3:0>				MONTH01<3:0>			
	15:8	DAY10<3:0>				DAY01<3:0>			
	7:0	—	—	—	—	WDAY01<3:0>			
RTCDATECLR	31:0	Write clears selected bits in RTCDATE, read yields undefined value							
RTCDATESET	31:0	Write sets selected bits in RTCDATE, read yields undefined value							
RTCDATEINV	31:0	Write inverts selected bits in RTCDATE, read yields undefined value							
ALRMTIME	31:24	HR10<3:0>				HR01<3:0>			
	23:16	MIN10<3:0>				MIN01<3:0>			
	15:8	SEC10<3:0>				SEC01<3:0>			
	7:0	—	—	—	—	—	—	—	—
ALRMTIMECLR	31:0	Write clears selected bits in ALRMTIME, read yields undefined value							
ALRMTIMESET	31:0	Write sets selected bits in ALRMTIME, read yields undefined value							
ALRMTIMEINV	31:0	Write inverts selected bits in ALRMTIME, read yields undefined value							
ALRMDATE	31:24	—	—	—	—	—	—	—	—
	23:16	MONTH10<3:0>				MONTH01<3:0>			
	15:8	DAY10<3:0>				DAY01<3:0>			
	7:0	—	—	—	—	WDAY01<3:0>			
ALRMDATECLR	31:0	Write clears selected bits in ALRMDATE, read yields undefined value							
ALRMDATESET	31:0	Write sets selected bits in ALRMDATE, read yields undefined value							
ALRMDATEINV	31:0	Write inverts selected bits in ALRMDATE, read yields undefined value							

Table 29-1: RTCC SFR Summary (Continued)

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
IFS1	31:24	—	—	—	—	—	—	USBIF	FCEIF
	23:16	—	—	—	—	DMA3IF	DMA2IF	DMA1IF	DMA0IF
	15:8	RTCCIF	FSCMIF	I2C2MIF	I2C2SIF	I2C2BIF	U2TXIF	U2RXIF	U2EIF
	7:0	SPI2RXIF	SPI2TXIF	SPI2EIF	CMP2IF	CMP1IF	PMPIF	AD1IF	CNIF
IFS1CLR	31:0	Write clears selected bits in IFS1, read yields undefined value							
IFS1SET	31:0	Write sets selected bits in IFS1, read yields undefined value							
IFS1INV	31:0	Write inverts selected bits in IFS1, read yields undefined value							
IEC1	31:24	—	—	—	—	—	—	USBIE	FCEIE
	23:16	—	—	—	—	DMA3IE	DMA2IE	DMA1IE	DMA0IE
	15:8	RTCCIE	FSCMIE	I2C2MIE	I2C2SIE	I2C2BIE	U2TXIE	U2RXIE	U2EIE
	7:0	SPI2RXIE	SPI2TXIE	SPI2EIE	CMP2IE	CMP1IE	PMPIE	AD1IE	CNIE
IEC1CLR	31:0	Write clears selected bits in IEC1, read yields undefined value							
IEC1SET	31:0	Write sets selected bits in IEC1, read yields undefined value							
IEC1INV	31:0	Write inverts selected bits in IEC1, read yields undefined value							
IPC8	31:24	—	—	—	RTCCIP<2:0>			RTCCIS<1:0>	
	23:16	—	—	—	FSCMIP<2:0>			FSCMIS<1:0>	
	15:8	—	—	—	I2C2IP<2:0>			I2C2IS<1:0>	
	7:0	—	—	—	U2IP<2:0>			U2IS<1:0>	
IPC8CLR	31:0	Write clears the selected bits in IPC8, read yields undefined value							
IPC8SET	31:0	Write sets the selected bits in IPC8, read yields undefined value							
IPC8INV	31:0	Write inverts the selected bits in IPC8, read yields undefined value							

Register 29-1: RTCCON: RTC Control Register(1)

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	CAL<9:8>	
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CAL<7:0>							
bit 23							bit 16

R/W-0	R/W-0	R/W-0	r-x	r-x	r-x	r-x	r-x
ON	FRZ	SIDL	—	—	—	—	—
bit 15							bit 8

R/W-0	R-0	r-x	r-x	R/W-0	R-0	R-0	R/W-0
RTSECSEL	RTCCLKON	—	—	RTCWREN	RTCSYNC	HALFSEC	RTCOE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-26 **Reserved:** Write ‘0’; ignore read

bit 25-16 **CAL<9:0>:** RTC Drift Calibration bits, contains a signed 10-bit integer value
0111111111= Maximum positive adjustment, adds 511 RTC clock pulses every one minute
...
0000000001= Minimum positive adjustment, adds 1 RTC clock pulse every one minute
0000000000= No adjustment
1111111111= Minimum negative adjustment, subtracts 1 RTC clock pulse every one minute
...
1000000000= Minimum negative adjustment, subtracts 512 clock pulses every one minute

bit 15 **ON:** RTCC On bit
1 = RTCC module is enabled
0 = RTCC module is disabled
Note 1: The ON bit is only writable when RTCWREN = 1.
2: When using 1:1 PBCLK divisor, the user’s software should not read/write the peripheral’s SFRs in the SYSCLK cycle immediately following the instruction that clears the module’s ON bit.

bit 14 **FRZ:** Freeze in DEBUG Mode bit
1 = When emulator is in DEBUG mode, module freezes operation
0 = When emulator is in DEBUG mode, module continues operation
Note: FRZ is writable in Debug Exception mode only, it is forced to ‘0’ in normal mode.

bit 13 **SIDL:** Stop in IDLE Mode bit
1 = Disables the PBCLK to the RTCC when CPU enters in IDLE mode
0 = Continue normal operation in IDLE mode

bit 12-8 **Reserved:** Write ‘0’; ignore read

bit 7 **RTSECSEL:** RTCC Seconds Clock Output Select
1 = RTCC Seconds Clock is selected for the RTCC pin
0 = RTCC Alarm Pulse is selected for the RTCC pin
Note: Requires RTCOE == 1 (RTCCON<0>) for the output to be active.

Register 29-1: RTCCON: RTC Control Register(1) (Continued)

bit 6 **RTCCLKON:** Status of the RTCC clock enable

1 = RTCC Clock is actively running
0 = RTCC Clock is not running

bit 5-4 **Reserved:** Write '0'; ignore read

bit 3 **RTCWREN:** RTC Value Registers Write Enable bit

1 = RTC Value registers can be written to by the user
0 = RTC Value registers are locked out from being written to by the user

Note: The RTCWREN bit can be set only when write sequence enabled. The register can be written to a '0' at any time.

bit 2 **RTCSYNC:** RTCC Value Registers Read Synchronization bit

1 = RTC Value registers can change while reading, due to a roll-over ripple that results in an invalid data read
If the register is read twice and results in the same data, the data can be assumed to be valid
0 = RTC Value registers can be read without concern about a roll-over ripple

bit 1 **HALFSEC:** Half-Second Status bit

1 = Second half period of a second
0 = First half period of a second

Note: This bit is read-only. It is cleared to '0' on a write to the SECONDS register.

bit 0 **RTCOE:** RTCC Output Enable bit

1 = RTCC clock output enabled – clock presented onto an I/O
0 = RTCC clock output disabled

Note: This bit is ANDed with ON (RTCCON<15>) to produce the effective RTCC output enable.

Note 1: This register is only reset by Power-on Reset (POR).

Register 29-2: RTCCONCLR: RTCCON Clear Register

Write clears selected bits in RTCCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in RTCCON**

A write of '1' in one or more bit positions clears the corresponding bit(s) in RTCCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `RTCCONCLR = 0x00008001` clears bits 15 and 0 in RTCCON register.

Register 29-3: RTCCONSET: RTCCON Set Register

Write sets selected bits in RTCCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in RTCCON**

A write of '1' in one or more bit positions sets the corresponding bit(s) in RTCCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `RTCCONSET = 0x00008001` sets bits 15 and 0 in RTCCON register.

Register 29-4: RTCCONINV: RTCCON Invert Register

Write inverts selected bits in RTCCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in RTCCON**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in RTCCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `RTCCONINV = 0x00008001` inverts bits 15 and 0 in RTCCON register.

Register 29-5: RTCALRM: RTC ALARM Control Register(1)

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
ALRMEN	CHIME	PIV	ALRMSYNC	AMASK<3:0>			
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ARPT<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15 **ALRMEN:** Alarm Enable bit

1 = Alarm is enabled
0 = Alarm is disabled

Note: Hardware clears ALRMEN anytime the alarm event occurs, when ARPT<7:0> = 00 and CHIME = 0.
This field should not be written when RTCC ON = 1 (RTCCON<15>) and ALRMSYNC = 1.

bit 14 **CHIME:** Chime Enable bit

1 = Chime is enabled – ARPT<7:0> is allowed to roll over from 00 to FF
0 = Chime is disabled – ARPT<7:0> stops once it reaches 00

Note: This field should not be written when RTCC ON = 1 (RTCCON<15>) and ALRMSYNC = 1.

bit 13 **PIV:** Alarm Pulse Initial Value bit

When ALRMEN = 0, PIV is writable and determines the initial value of the Alarm Pulse.
When ALRMEN = 1, PIV is read-only and returns the state of the Alarm Pulse.

Note: This field should not be written when RTCC ON = 1 (RTCCON<15>) and ALRMSYNC = 1.

bit 12 **ALRMSYNC:** Alarm Sync bit

1 = ARPT<7:0> and ALRMEN may change as a result of a half second rollover during a read. The ARPT must be read repeatedly until the same value is read twice. This must be done since multiple bits may be changing, which are then synchronized to the PB clock domain
0 = ARPT<7:0> and ALRMEN can be read without concerns of rollover because prescaler is > 32 RTC clock away from a half second rollover.

Note: This assumes a CPU read will execute in less than 32 PBCLKs.

Register 29-5: RTCALRM: RTC ALARM Control Register(1) (Continued)

bit 11-8 **AMASK<3:0>:** Alarm Mask Configuration bits

0000 = Every half second
0001 = Every second
0010 = Every 10 seconds
0011 = Every minute
0100 = Every 10 minutes
0101 = Every hour
0110 = Once a day
0111 = Once a week
1000 = Once a month
1001 = Once a year (except when configured for February 29th, once every 4 years)
1010 = Reserved – do not use
1011 = Reserved – do not use
11XX = Reserved – do not use

Note: This field should not be written when RTCC ON = 1 (RTCCON<15>) and ALRMSYNC = 1.

bit 7-0 **ARPT<7:0>:** Alarm Repeat Counter Value bits

11111111 = Alarm will trigger 256 times
...
00000000 = Alarm will trigger 1 time

The counter decrements on any alarm event. The counter only rolls over from 00 to FF if CHIME = 1.

Note: This field should not be written when RTCC ON = 1 (RTCCON<15>) and ALRMSYNC = 1.

Note 1: This register is only reset by POR.

Register 29-6: RTCALRMCLR: RTCALRM Clear Register

Write clears selected bits in RTCALRM, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in RTCALRM**

A write of '1' in one or more bit positions clears the corresponding bit(s) in RTCALRM register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `RTCALRMCLR = 0x0000c000` clears bits 15 and 14 in RTCALRM register.

Register 29-7: RTCALRMSET: RTCALRM Set Register

Write sets selected bits in RTCALRM, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in RTCALRM**

A write of '1' in one or more bit positions sets the corresponding bit(s) in RTCALRM register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `RTCALRMSET = 0x0000c000` sets bits 15 and 14 in RTCALRM register.

Register 29-8: RTCALRMINV: RTCALRM Invert Register

Write inverts selected bits in RTCALRM, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in RTCALRM**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in RTCALRM register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `RTCALRMINV = 0x0000c000` inverts bits 15 and 14 in RTCALRM register.

Register 29-9: RTCTIME: RTC Time Value Register(1)

R-0	R-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
HR10<3:0>				HR01<3:0>			
bit 31							bit 24

R-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
MIN10<3:0>				MIN01<3:0>			
bit 23							bit 16

R-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
SEC10<3:0>				SEC01<3:0>			
bit 15							bit 8

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-28 **HR10<3:0>:** Binary-Coded Decimal Value of Hours bits, 10 digits; contains a value from 0 to 2
Note: HR10<3:2> bits are always read ‘0’.

bit 27-24 **HR01<3:0>:** Binary-Coded Decimal Value of Hours bits, 1 digit; contains a value from 0 to 9

bit 23-20 **MIN10<3:0>:** Binary-Coded Decimal Value of Minutes bits, 10 digits; contains a value from 0 to 5
Note: MIN10<3> bit is always read ‘0’.

bit 19-16 **MIN01<3:0>:** Binary-Coded Decimal Value of Minutes bits, 1 digit; contains a value from 0 to 9

bit 15-12 **SEC10<3:0>:** Binary-Coded Decimal Value of Seconds bits, 10 digits; contains a value from 0 to 5
Note: SEC10<3> bit is always read ‘0’.

bit 11-8 **SEC01<3:0>:** Binary-Coded Decimal Value of Seconds bits, 1 digit; contains a value from 0 to 9

bit 7-0 **Reserved:** Write ‘0’; ignore read

Note 1: This register is only writable when RTCWREN = 1 (RTCCON<3>).

Register 29-10: RTCTIMECLR: RTCTIME Clear Register

Write clears selected bits in RTCTIME, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in RTCTIME**

A write of '1' in one or more bit positions clears the corresponding bit(s) in RTCTIME register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `RTCTIMECLR = 0x0000ff00` clears bits 15:8 in RTCTIME register.

Register 29-11: RTCTIMESET: RTCTIME Set Register

Write sets selected bits in RTCTIME, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in RTCTIME**

A write of '1' in one or more bit positions sets the corresponding bit(s) in RTCTIME register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `RTCTIMESET = 0x00005900` sets seconds to 59 in RTCTIME register.

Register 29-12: RTCTIMEINV: RTCTIME Invert Register

Write inverts selected bits in RTCTIME, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in RTCTIME**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in RTCTIME register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `RTCTIMEINV = 0x00000300` inverts bits 9 and 8 in RTCTIME register.

Register 29-13: RTCDATE: RTC Date Value Register(1)

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
YEAR10<3:0>				YEAR01<3:0>			
bit 31							bit 24

R-0	R-0	R-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
MONTH10<3:0>				MONTH01<3:0>			
bit 23							bit 16

R-0	R-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
DAY10<3:0>				DAY01<3:0>			
bit 15							bit 8

r-x	r-x	r-x	r-x	R-0	R/W-x	R/W-x	R/W-x
—	—	—	—	WDAY01<3:0>			
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-28 **YEAR10<3:0>:** Binary-Coded Decimal Value of Years bits, 10 digits

bit 27-24 **YEAR01<3:0>:** Binary-Coded Decimal Value of Years bits, 1 digit

bit 23-20 **MONTH10<3:0>:** Binary-Coded Decimal Value of Months bits, 10 digits; contains a value from 0 to 1
Note: MONTH10<3:1> bits are always read ‘0’.

bit 19-16 **MONTH01<3:0>:** Binary-Coded Decimal Value of Months bits, 1 digit; contains a value from 0 to 9

bit 15-12 **DAY10<3:0>:** Binary-Coded Decimal Value of Days bits, 10 digits; contains a value from 0 to 3
Note: DAY10<3:2> bits are always read ‘0’.

bit 11-8 **DAY01<3:0>:** Binary-Coded Decimal Value of Days bits, 1 digit; contains a value from 0 to 9

bit 7-4 **Reserved:** Write ‘0’; ignore read

bit 3-0 **WDAY01<3:0>:** Binary-Coded Decimal Value of Weekdays bits,1 digit; contains a value from 0 to 6
Note: WDAY01<3> bit is always read ‘0’.

Note 1: This register is only writable when RTCWREN = 1 (RTCCON<3>).

Register 29-14: RTCDATECLR: RTCDATE Clear Register

Write clears selected bits in RTCDATE, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in RTCDATE**

A write of '1' in one or more bit positions clears the corresponding bit(s) in RTCDATE register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `RTCDATECLR = 0x00000007` will clear bits 2:0 in RTCDATE register.

Register 29-15: RTCDATESET: RTCDATE Set Register

Write sets selected bits in RTCDATE, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in RTCDATE**

A write of '1' in one or more bit positions sets the corresponding bit(s) in RTCDATE register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `RTCDATESET = 0x00000003` sets weekday to 3 in RTCDATE register.

Register 29-16: RTCDATEINV: RTCDATE Invert Register

Write inverts selected bits in RTCDATE, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in RTCDATE**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in RTCDATE register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `RTCDATEINV = 0x00000003` inverts bits 1 and 0 in RTCDATE register.

Register 29-17: ALRMTIME: Alarm Time Value Register

R-0	R-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
HR10<3:0>				HR01<3:0>			
bit 31							bit 24

R-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
MIN10<3:0>				MIN01<3:0>			
bit 23							bit 16

R-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
SEC10<3:0>				SEC01<3:0>			
bit 15							bit 8

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-28 **HR10<3:0>:** Binary Coded Decimal value of hours bits, ‘10’ digit; contains a value from 0 to 2
Note: HR10<3:2> bits are always read ‘0’.

bit 27-24 **HR01<3:0>:** Binary Coded Decimal value of hours bits, ‘1’ digit; contains a value from 0 to 9

bit 23-20 **MIN10<3:0>:** Binary Coded Decimal value of minutes bits, ‘10’ digit; contains a value from 0 to 5
Note: MIN10<3> bit is always read ‘0’.

bit 19-16 **MIN01<3:0>:** Binary Coded Decimal value of minutes bits, ‘1’ digit; contains a value from 0 to 9

bit 15-12 **SEC10<3:0>:** Binary Coded Decimal value of seconds bits, ‘10’ digit; contains a value from 0 to 5
Note: SEC10<3> bit is always read ‘0’.

bit 11-8 **SEC01<3:0>:** Binary Coded Decimal value of seconds bits, ‘1’ digit; contains a value from 0 to 9

bit 7-0 **Reserved:** Write ‘0’; ignore read

Register 29-18: ALRMTIMECLR: ALRMTIME Clear Register

Write clears selected bits in ALRMTIME, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in ALRMTIME**

A write of '1' in one or more bit positions clears the corresponding bit(s) in ALRMTIME register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `ALRMTIMECLR = 0x0000ff00` clears bits 15:8 in ALRMTIME register.

Register 29-19: ALRMTIMESET: ALRMTIME Set Register

Write sets selected bits in ALRMTIME, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in ALRMTIME**

A write of '1' in one or more bit positions sets the corresponding bit(s) in ALRMTIME register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `ALRMTIMESET = 0x00330000` sets alarm minutes to 33 in ALRMTIME register.

Register 29-20: ALRMTIMEINV: ALRMTIME Invert Register

Write inverts selected bits in ALRMTIME, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in ALRMTIME**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in ALRMTIME register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `ALRMTIMEINV = 0x00000300` inverts bits 9 and 8 in ALRMTIME register.

Register 29-21: ALRMDATE: Alarm Date Value Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

R-0	R-0	R-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
MONTH10<3:0>				MONTH01<3:0>			
bit 23							bit 16

R-0	R-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
DAY10<3:0>				DAY01<3:0>			
bit 15							bit 8

r-x	r-x	r-x	r-x	R-0	R/W-x	R/W-x	R/W-x
—	—	—	—	WDAY01<3:0>			
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-24 **Reserved:** Write ‘0’; ignore read

bit 23-20 **MONTH10<3:0>:** Binary Coded Decimal value of months bits, ‘10’ digit; contains a value from 0 to 1
Note: MONTH10<3:1> bits are always read ‘0’.

bit 19-16 **MONTH01<3:0>:** Binary Coded Decimal value of months bits, ‘1’ digit; contains a value from 0 to 9

bit 15-12 **DAY10<3:0>:** Binary Coded Decimal value of days bits, ‘10’ digit; contains a value from 0 to 3
Note: DAY10<3:2> bits are always read ‘0’.

bit 11-8 **DAY01<3:0>:** Binary Coded Decimal value of days bits, ‘1’ digit; contains a value from 0 to 9

bit 7-4 **Reserved:** Write ‘0’; ignore read

bit 3-0 **WDAY01<3:0>:** Binary Coded Decimal value of weekdays bits, ‘1’ digit; contains a value from 0 to 6
Note: WDAY01<3> bit is always read ‘0’.

Register 29-22: ALRMDATECLR: ALRMDATE Clear Register

Write clears selected bits in ALRMDATE, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in ALRMDATE**

A write of '1' in one or more bit positions clears the corresponding bit(s) in ALRMDATE register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `ALRMDATECLR = 0x00000007` clears bits 2:0 in ALRMDATE register.

Register 29-23: ALRMDATESET: ALRMDATE Set Register

Write sets selected bits in ALRMDATE, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in ALRMDATE**

A write of '1' in one or more bit positions sets the corresponding bit(s) in ALRMDATE register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `ALRMDATESET = 0x00003100` sets alarm day to 31 in ALRMDATE register.

Register 29-24: ALRMDATEINV: ALRMDATE Invert Register

Write inverts selected bits in ALRMDATE, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in ALRMDATE**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in ALRMDATE register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `ALRMDATEINV = 0x00000003` inverts bits 1 and 0 in ALRMDATE register.

Register 29-25: IFS1: Interrupt Flag Status Register 1(1)

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	USBIF	FCEIF
bit 31							bit 24

r-0	r-0	r-0	r-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	DMA3IF	DMA2IF	DMA1IF	DMA0IF
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RTCCIF	FSCMIF	I2C2MIF	I2C2SIF	I2C2BIF	U2TXIF	U2RXIF	U2EIF
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-0
SPI2RXIF	SPI2TXIF	SPI2EIF	CMP2IF	CMP1IF	PMPIF	AD1IF	CNIF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-26 **Reserved:** Write ‘0’; ignore read

bit 25-24 Interrupt flags for other peripheral devices

bit 23-20 **Reserved:** Write ‘0’; ignore read

bit 19-16 Interrupt flags for other peripheral devices

bit 15 **RTCCIF**: RTCC Interrupt Flag bit
1 = RTCC interrupt pending
0 = No RTCC interrupt pending
Set by the hardware when an event is generated by the RTCC.
Cleared by the software, usually in the ISR.

bit 14-0 Interrupt flags for other peripheral devices

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the RTCC.

Register 29-26: IEC1: Interrupt Enable Control Register 1(1)

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	USBIE	FCEIE
bit 31							bit 24

r-0	r-0	r-0	r-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	DMA3IE	DMA2IE	DMA1IE	DMA0IE
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RTCCIE	FSCMIE	I2C2MIE	I2C2SIE	I2C2BIE	U2TXIE	U2RXIE	U2EIE
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-0
SPI2RXIE	SPI2TXIE	SPI2EIE	CMP2IE	CMP1IE	PMPIE	AD1IE	CNIE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-26 Unimplemented: Read as ‘0’

bit 25-24 Interrupt flags for other peripheral devices

bit 23-20 **Reserved:** Write ‘0’; ignore read

bit 19-16 Interrupt flags for other peripheral devices

bit 15 **RTCCIE:** RTCC interrupt enable.
1 = RTCC interrupt enabled
0 = RTCC interrupt disabled
Set/cleared by the software to enable/disable the interrupt when an event is generated by the RTCC.

bit 14-0 Interrupt enable bits for other peripheral devices

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the RTCC.

Register 29-27: IPC8: Interrupt Priority Control Register 8(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	RTCCIP<2:0>			RTCCIS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	FCSMIP<2:0>			FCSMIS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	I2C2IP<2:0>			I2C2IS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	U2IP<2:0>			U2IS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-29 **Reserved:** Write ‘0’; ignore read

bit 28-26 **RTCCIP<2:0>:** RTCC Interrupt Vector Priority bits
111 = RTCC interrupts have priority 7 (highest priority)
•
•
•
001 = RTCC interrupts have priority 1
000 = RTCC interrupts are disabled

bit 25-24 **RTCCIS<1:0>:** RTCC Interrupt Vector Subpriority bits
11 = RTCC interrupts have subpriority 3 (highest subpriority)
•
•
00 = RTCC interrupts have subpriority 0 (lowest subpriority)

bit 23-21 **Reserved:** Write ‘0’; ignore read

bit 20-16 **Interrupt priority control for other peripheral devices**

bit 15-13 **Reserved:** Write ‘0’; ignore read

bit 12-8 **Interrupt priority control for other peripheral devices**

bit 7-5 **Reserved:** Write ‘0’; ignore read

bit 4-0 **Interrupt priority control for other peripheral devices**

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the RTCC.

29.3 MODES OF OPERATION

The RTCC module offers the following operating modes:

- Real-Time Clock and Calendar (RTCC) function
- Alarm function

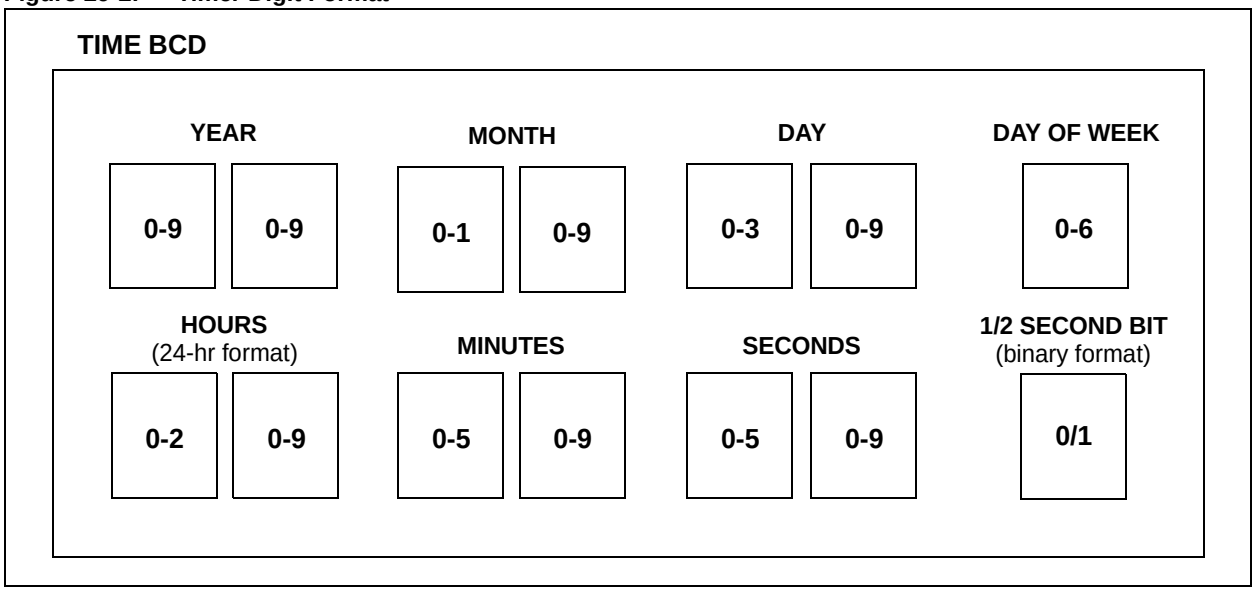
29.3.1 RTCC Operation

The RTCC is a 100-year clock and calendar with automatic leap year detection. The range of the clock is from 00:00:00 (midnight) on January 1, 2000, to 23:59:59 on December 31, 2099. The hours use the 24-hour time format (military time) with no hardware provisions for regular time format (AM/PM).

The RTCC provides a programming granularity of 1 second but has visibility of the half-second field.

The register interface for the RTCC values (RTCTIME and RTCDATE) is implemented using the Binary Coded Decimal (BCD) format. This simplifies the firmware, when using the module, as each of the digit values is contained within its own 4-bit value (see Figure 29-2).

Figure 29-2: Timer Digit Format



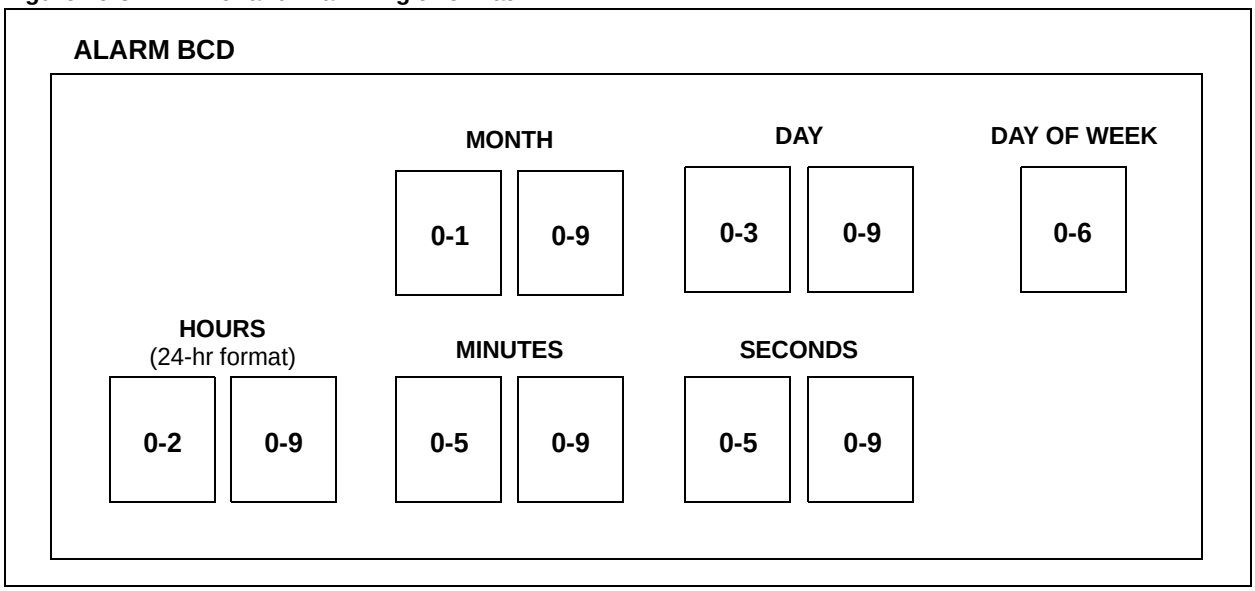
29.3.2 Alarm Operation

The module provides an alarm function configurable anywhere from a half-second to one year. However, only the half-second alarm has the half-second resolution. The module can be configured to repeat the alarm at pre-configured intervals, after the alarm is enabled. The indefinite repetition of the alarm is provided through the Chime feature.

The module provides an interrupt at every alarm pulse event. In addition to the alarm interrupt, an alarm pulse output is provided that operates at half the frequency of the alarm (the alarm pulse toggles at every alarm match). This output is completely synchronous with the RTCC clock and can be used to provide a trigger clock to other devices. The initial value of this output pin is controlled by PIV (RTCALRM<13>). See Register 29-5 for more information.

The register interface for the Alarm values (ALRMTIME and ALRMDATE) is implemented using the BCD format. This simplifies the firmware, when using the module, as each of the digit values is contained within its own 4-bit value (see Figure 29-3).

Figure 29-3: Timer and Alarm Digit Format

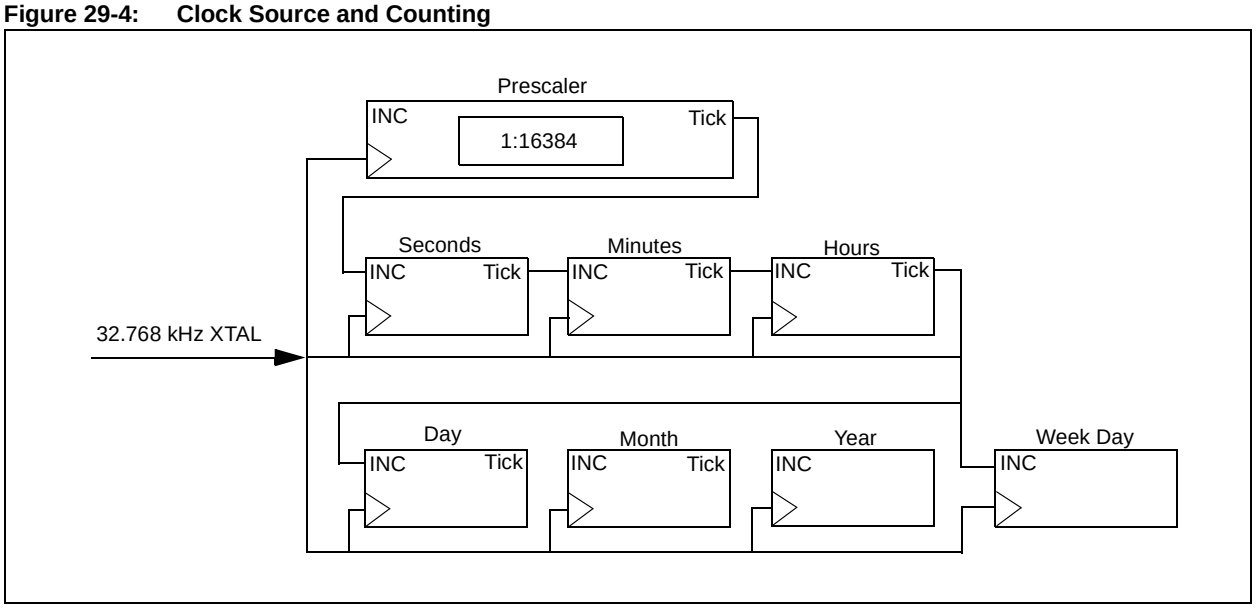


29.3.3 Clock Source

The RTCC module is intended to be clocked by an external real-time clock crystal that is oscillating at 32.768 kHz. Calibration of the crystal can be accomplished through this module, yielding an accuracy of +/-0.66 seconds per month (see **Section 29.3.10 "Calibration"** for further details).

To allow the RTCC to be clocked by an external 32.768 kHz crystal, the SOSCEN bit (OSCCON<1>) must be set (see **Section 6. "Oscillators"**, Register 6-1). This is the only bit outside of the RTCC module with which the user must be concerned for enabling the RTCC. The Status bit SOSCRDY (OSCCON<22>) can be used to check that the secondary oscillator is running.

Figure 29-4: Clock Source and Counting



29.3.4 Digit Carry Rules

This section explains which timer values are affected when there is a rollover.

- Time of Day – from 23:59:59 to 00:00:00, with a carry to the Day field
- Day – the carry from the day field to the month field is dependent on the current month (Refer to Table 29-3 for the day to month rollover schedule.)
- Month – from 12/31 to 01/01, with a carry to the Year field
- Day of Week – from 6 to 0, without a carry (refer to Table 29-2)
- Year – from 99 to 00, without a carry (this surpasses the intended use of the RTCC)

Considering that the following values are in BCD format, the carry to the upper BCD digit will occur at a count of 10, and not a count of 16 (SECONDS, MINUTES, HOURS, WEEKDAY, DAYS, MONTHS).

Table 29-2: Day of Week Schedule

Day of Week	
Sunday	0
Monday	1
Tuesday	2
Wednesday	3
Thursday	4
Friday	5
Saturday	6

Table 29-3: Day to Month Rollover Schedule

Month	Maximum Day Field
01 (January)	31
02 (February)	28 or 29(1)
03 (March)	31
04 (April)	30
05 (May)	31
06 (June)	30
07 (July)	31
08 (August)	31
09 (September)	30
10 (October)	31
11 (November)	30
12 (December)	31

Note 1: See **Section 29.3.5 “Leap Year”**.

29.3.5 Leap Year

Since the year range on the RTCC module is 2000 to 2099, the leap calculation is determined by any year divisible by 4 in the above range. The only month to be affected in a leap year is February. (February has 29 days in a leap year, but only 28 days in all other years.)

29.3.6 RTCC General Functionality

All timer registers containing a time value of seconds, or greater, are writable. The user can configure the current time by simply writing to these registers the desired year, month, day, hour, minutes and seconds. The timer will then use the newly written values to proceed with the count from the desired starting point.

Note that if the RTCC is enabled having ON = 1 (RTCCON<15>), the timer will continue incrementing even while the registers are being adjusted. However, any time the SECONDS register (RTCTIME<15:8>) is written, the Prescaler is reset to '0'. This provides a known Prescaler value after timer adjustments.

If an update (CPU write) of the timer register occurs, it is the user's responsibility to ensure that when ON = 1 (RTCCON<15>), a timer increment will not occur to the registers that are being updated. This can be done by observing the value of the RTCSYNC bit (RTCCON<2>), or the preceding digits from which a carry can occur, or by only updating the registers immediately following the seconds pulse (or alarm interrupt). Note that the corresponding counters are clocked based on their defined intervals, i.e., the DAYS register is clocked once a day, the MONTHS register is only clocked once a month, etc. This leaves large windows of time in which registers can be safely updated.

The timer also provides visibility into the half-second field of the counter. However, this value is read-only and can only be reset by writing to the SECONDS register (RTCTIME<15:8>).

29.3.7 Safety Window for Register Reads and Writes

The RTCSYNC bit (RTCCON<2>) indicates a time window during which: an update to the RTCC time registers (RTCTIME, RTCDATE) is not imminent, and the registers can be safely read and written. When RTCSYNC = 0, the registers can be safely accessed by the CPU. When RTCSYNC = 1, the user must employ a firmware solution to assure that the data read did not fall on an update boundary, resulting in an invalid or partial read.

The RTCSYNC bit is set 32 RTCC clock edges before an update is about to occur. It is cleared one clock later, after the update occurs (thus RTCSYNC is asserted for a total of 33 clocks). At worst case, when the CPU core uses the 32.768 kHz oscillator as its clock and the PBCLK is SYSCLK/8, there are at least 22 CPU clocks in which to execute instructions after a read of the RTCSYNC = 0 (some clock cycles might be lost due to synchronization and bus delays).

Note that, independent of the RTCSYNC value – the user can, by reading and comparing a timer register value twice, ensure in code that the register read did not span an RTCC clock update.

Writes to the Time and Date registers should not be performed when RTCSYNC = 1. This restriction exists for two reasons:

1. A write could cause a timing violation in the Alarm Match logic, leading to an invalid alarm event and a corruption of the ARPT register. This event can occur during the low time of an RTCC clock, following a rollover event.
2. A write during a rollover event, when the RTCC clock is high, will be ignored by H/W.

Example 29-1: Updating the RTCC Time and Date

```
/*
    The following code example will update the RTCC time and date.
*/

    assume the secondary oscillator is enabled and ready, i.e. OSCCON<1>=1, OSCCON<22>=1, and
    RTCC write is enabled i.e. RTCWREN (RTCCON<3>) =1;

    unsigned long time=0x04153300;// set time to 04 hr, 15 min, 33 sec
    unsigned long date=0x06102705;// set date to Friday 27 Oct 2006

    RTCCONCLR=0x8000;             // turn off the RTCC
    while(RTCCON&0x40);           // wait for clock to be turned off
    RTCTIME=time;                 // safe to update the time
    RTCDATE=date;                 // update the date
    RTCCONSET=0x8000;             // turn on the RTCC
    while(!(RTCCON&0x40));        // wait for clock to be turned on

                                  // can disable the RTCC write
```

Example 29-2: Updating the RTCC Time Using the RTCSYNC Window

```
/*
    The following code example will update the RTCC time and date.
*/

    assume RTCC write is enabled i.e. RTCWREN (RTCCON<3>) =1;

    unsigned long time=0x04153300;// set time to 04 hr, 15 min, 33 sec
    unsigned long date=0x06102705;// set date to Friday 27 Oct 2006

    asm volatile (“di”);          // disable interrupts, critical section follows
    while((RTCCON&0x4)!=0);       // wait for not RTCSYNC
    RTCTIME=time;                 // safe to update the time
    RTCDATE=date;                 // update the date
    asm volatile (“ei”);          // restore interrupts, critical section ended

                                  // can disable the RTCC write
```

29.3.8 Synchronization

The module provides a single RTCSYNC bit (RTCCON<2>) that the user must use to determine when it is safe to read and update the time and date registers. In addition, the module provides synchronization for Reset conditions (i.e., a write to the seconds register), and for ON (RTCCON<15>).

29.3.8.1 RTCSYNC Bit Generation

The RTCSYNC bit is a read-only bit that is set when ON = 1 and the RTCC Prescaler counter equals 0x7FE0 (32 clocks away from a one-second roll-over). Logic clears the RTCSYNC bit for any of the following conditions:

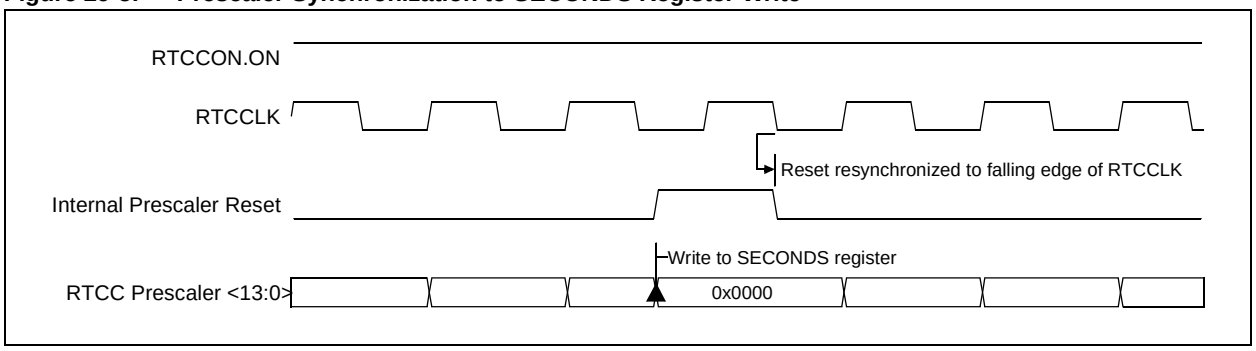
- POR
- Whenever ON = 0
- On a write to the SECONDS (RTCTIME<15:8>) register
- On the rising edge of the RTCC clock, when prescaler is 0x0000

See Figure 29-6 for the RTCSYNC bit timings.

29.3.8.2 Prescaler Reset Synchronization

A write to the SECONDS register (RTCTIME<15:8>) asynchronously resets the RTCC Prescaler (including the HALFSEC register). The Reset remains active until a falling edge of RTCCLK is detected (see Figure 29-5).

Figure 29-5: Prescaler Synchronization to SECONDS Register Write



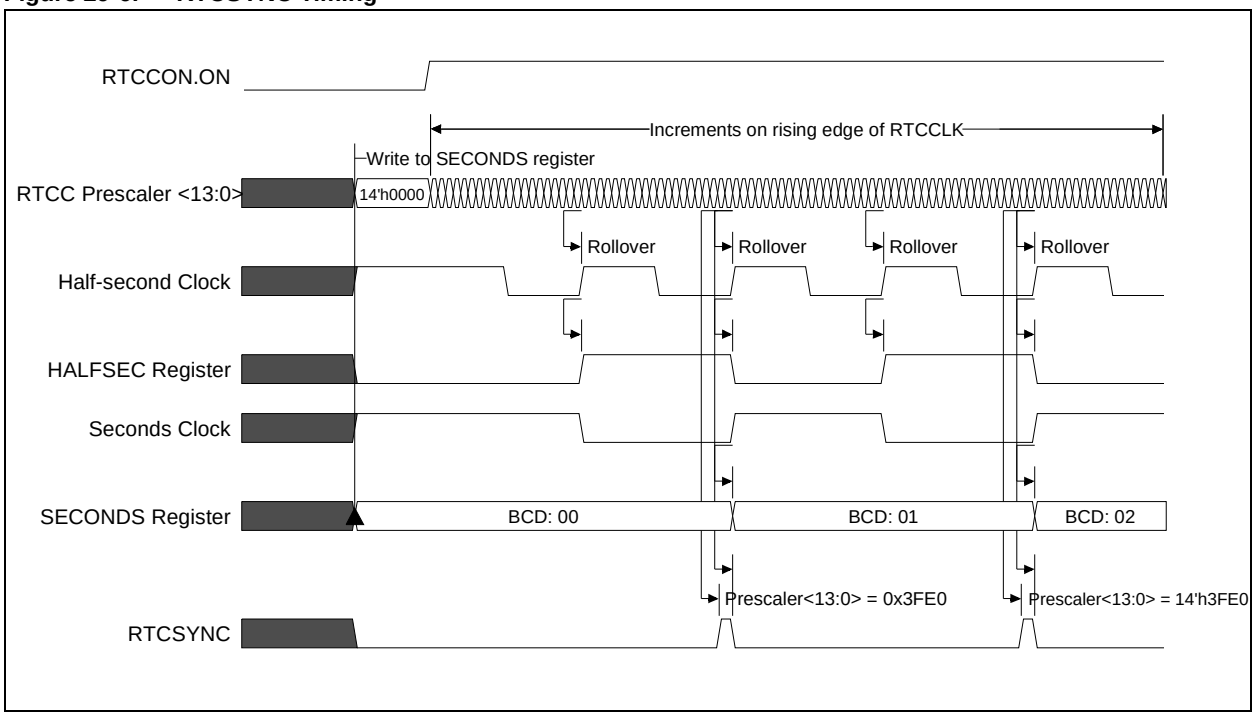
29.3.8.3 Gating Off the RTCC Clock

There are two conditions for which the internal RTCC clock is held off:

- ON (RTCCON<15>) = 0
- The device is in the DEBUG mode and FRZ (RTCCON<14>) = 1.

Stopping the RTCC clock does not affect reading and writing registers from the peripheral bus interface.

Figure 29-6: RTCSYNC Timing



29.3.9 Write Lock

In order to perform a write to any of the RTCC timer registers, the RTCWREN bit (RTCCON<3>) must be set. Setting of the RTCWREN bit is only allowed once the device level unlocking sequence has been executed. The unlocking sequence is as follows:

1. Load 0xAA996655 to CPU register X.
2. Load 0x556699AA to CPU register Y.
3. Load 0x00000008 to CPU register Z (the RTCWREN bit number).
4. Suspend or disable all Initiators that can access the Peripheral Bus and interrupt the unlock sequence. (i.e., DMA and Interrupts).
5. Store CPU register X to SYSKEY.
6. Store CPU register Y to SYSKEY.
7. Store CPU register Z to RTCCONSET.
8. Re-enable DMA and interrupts.

Note that steps 5 through 7 must be followed exactly to unlock RTCC write operations. If the sequence is not followed exactly, the RTCWREN bit will not be set.

Refer to Figure 29-3 for an assembly language implementation of the Write Unlock operation.

Example 29-3: Write Unlock Sequence

```
# assume interrupts are disabled
# assume the DMA controller is suspended
# assume the device is locked

#starting critical sequence
SYSKEY = 0xaa996655;                 // write first unlock key to SYSKEY
SYSKEY = 0x556699aa;                 // write second unlock key to SYSKEY
RTCCONSET = 0x8;                     // set RTCWREN in RTCCONSET
#end critical sequence

# re-enable interrupts
# re-enable the DMA controller
```

Note: To avoid accidental writes to the RTCC time values, it is recommended that the RTCWREN bit (RTCCON<3>) is kept clear at any other time. For RTCWREN to be set, there is only 1 instruction cycle time window allowed between the key1, key2 sequence and the setting of RTCWREN. Therefore, it is recommended that the code example in Example 29-3 be followed.

29.3.10 Calibration

The real-time crystal input can be calibrated using the periodic auto-adjust feature. When properly calibrated, the RTCC can provide an error of less than 0.66 seconds per month. Calibration has the ability to eliminate an error of up to 260 ppm.

The calibration is accomplished by finding the number of error clock pulses and writing this value into the CAL field of the RTCCON register (RTCCON<9:0>). This 10-bit signed value will be either added or subtracted from the RTCC timer once every minute. Refer to the steps below for RTCC calibration:

1. Using another timer resource on the device, the user must find the error of the 32.768 kHz crystal.
2. Once the error is known, it must be converted to the number of error clock pulses per minute.

Equation 29-1: Formula Box:

(Ideal Frequency (32,758) – Measured Frequency) * 60 = Error Clocks per Minute

3. a) If the oscillator is *faster* than ideal (negative result from step 2), the CAL register value needs to be negative. This causes the specified number of clock pulses to be subtracted from the timer counter once every minute.
   b) If the oscillator is *slower* than ideal (positive result from step 2), the CAL register value needs to be positive. This causes the specified number of clock pulses to be added to the timer counter once every minute.
4. Load the CAL register (RTCCON<9:0>) with the correct value.

Writes to the CAL register should only occur when the timer is turned off, or immediately after the rising edge of the seconds pulse (except when SECONDS (RTCTIME<15:8>) field is 00, due to the possibility of the auto-adjust event).

Notes:	It is up to the user to include in the error value the initial error of the crystal, drift due to temperature, and drift due to crystal aging. A write to the SECONDS register resets the state of calibration (not its value). If an adjustment just occurred, it will occur again because of the minute roll-over.

Example 29-4: Updating the RTCC Calibration Value

```
/*
    The following code example will update the RTCC calibration.
*/



    int cal=0x3FD;                // 10 bits adjustment, -3 in value

    if(RTCCON&0x8000)
    {                             // RTCC is ON
        unsigned intt0, t1;
        do
        {
            t0=RTCTIME;
            t1=RTCTIME;
        }while(t0!=t1);           // read valid time value
        if((t0&0xFF)==00)
        {                         // we're at second 00, wait auto-adjust to be performed
            while(!(RTCCON&0x2)); // wait until second half...
        }
    }

    RTCCONCLR=0x03FF0000;         // clear the calibration
    RTCCONSET=cal;
```


29.4 ALARM

The RTCC module provides an alarm function with the following features:

- Configurable from a half-second to one year
- Enabled using the ALRMEN bit (RTCALRM<15>)
- One-time alarm, repeat alarms, and indefinite repetition of the alarm available

29.4.1 Configuring the Alarm

The alarm feature is enabled using the ALRMEN bit.

The interval selection is made based on the settings of the alarm mask, AMASK (RTCALRM<11:8>). The AMASK bits determine which and how many digits of the alarm must match the clock value for the alarm to occur (see Figure 29-7).

Note:	Once the timer value reaches the alarm setting, one RTCC clock period will elapse prior to setting the alarm interrupt. The result is that, for a short period, the user will see the timer value at the alarm setting without the interrupt having gone off.

29.4.1.1 Configuring the One-Time Alarm

When the alarm is issued, with ARPT (RTCALRM<7:0>) = 0 and CHIME (RTCALRM<14>) = 0, the ALRMEN bit automatically clears.

Example 29-5: Configuring the RTCC for a One-Time One-Per-Day Alarm

```
/*
    The following code example will update the RTCC one-time alarm.
    Assumes the interrupts are disabled.
*/


    unsigned long alTime=0x16153300;// set time to 04 hr, 15 min, 33 sec
    unsigned long alDate=0x06102705;// set date to Friday 27 Oct 2006

                                    // turn off the alarm, chime and alarm repeats; clear
                                    // the alarm mask

    while(RTCALRM&0x1000);          // wait ALRMSYNC to be off
    RTCALRMCLR=0xCFFF;              // clear ALRMEN, CHIME, AMASK and ARPT;
    ALRMTIME=alTime;
    ALRMDATE=alDate;                // update the alarm time and date

    RTCALRMSET=0x8000|0x00000600;   // re-enable the alarm, set alarm mask at once per day
```

29.4.1.2 Configuring the Repeat Alarm

In addition to providing a one-time alarm, the module can be configured to repeat the alarm at a pre-configured interval. The ARPT register contains the number of times the alarm repeats after the alarm is enabled. When ARPT = 0 and CHIME = 0, the repeat function is disabled and only a single alarm pulse will be produced. The alarm can be generated up to 256 times by setting ARPT = 0xFF.

Each time, after the alarm is issued, the ARPT register is decremented by one. Once the register reaches 0, the alarm will be generated one last time; after which point, ALRMEN bit is cleared automatically and the alarm will turn off.

Example 29-6: Configuring the RTCC for a Ten-Times One-Per-Hour Alarm

```
/*
    The following code example will update the RTCC repeat alarm.
    Assumes the interrupts are disabled.
*/


    unsigned long alTime=0x23352300;  // set time to 23hr, 35 min, 23 sec
    unsigned long alDate=0x06111301;  // set date to Monday 13 Nov 2006

                                      // turn off the alarm, chime and alarm repeats; clear
                                      // the alarm mask
    while(RTCALRM&0x1000);            // wait ALRMSYNC to be off
    RTCALRMCLR=0xCFFF;                // clear the ALRMEN, CHIME, AMASK and ARPT;
    ALRMTIME=alTime;
    ALRMDATE=alDate;                  // update the alarm time and date
    RTCALRMSET=0x8000|0x0509;         // re-enable the alarm, set alarm mask at once per hour
                                      // for 10 times repeat
```

29.4.1.3 Configuring the indefinite alarm

To provide an indefinite repetition of the alarm, the Chime feature can be enabled using the CHIME (RTCALRM<14>) bit. When CHIME = 1, rather than disabling the alarm when the last repeat has been performed, the ARPT rolls over from 0x00 to 0xFF and continues counting indefinitely.

Example 29-7: Configuring the RTCC for Indefinite One-Per-Day Alarm

```
/*
    The following code example will update the RTCC indefinite alarm.
    Assumes the interrupts are disabled.
*/


    unsigned long alTime=0x23352300;  // set time to 23hr, 35 min, 23 sec
    unsigned long alDate=0x06111301;  // set date to Monday 13 Nov 2006

                                      // turn off the alarm, chime and alarm repeats; clear
                                      // the alarm mask
    while(RTCALRM&0x1000);            // wait ALRMSYNC to be off
    RTCALRMCLR=0xCFFF;                // clear ALRMEN, CHIME, AMASK, ARPT;
    ALRMTIME=alTime;
    ALRMDATE=alDate;                  // update the alarm time and date
    RTCALRMSET=0xC600;                // re-enable the alarm, set alarm mask at once per
                                      // hour, enable CHIME
```


Figure 29-7: Alarm Mask Settings

Alarm Mask Setting AMASK<3:0>	Day of the Week	Month	Day	Hours	Minutes	Seconds
0000 – Every half second 0001 – Every second						
0010 – Every 10 seconds						s
0011 – Every minute						s s
0100 – Every 10 minutes					m	s s
0101 – Every hour					m m	s s
0110 – Every day				h h	m m	s s
0111 – Every week	d			h h	m m	s s
1000 – Every month			d d	h h	m m	s s
1001 – Every year(1)		m m	d d	h h	m m	s s

Note 1: Annually, except when configured for February 29.

29.4.2 Alarm Interrupt

The alarm event is generated when the RTCC timer matches the alarm registers. The match must only occur on the unmasked portion of the time/date registers, according to the AMASK (RTCALRM<11:8>) bit settings (see Figure 29-7).

At every alarm event, an interrupt is generated. In addition, an alarm pulse output is provided that operates at half the frequency of the alarm. This output is completely synchronous to the RTCC clock and can be used as a trigger clock to other peripherals. This output is available on the RTCC pin. The output pulse is a clock with a 50% duty cycle and a frequency half that of the alarm event (see Figure 29-8). The alarm must be enabled for the pulse to be active, ALRMEN (RTCALRM<15>) = 1. The initial value of the alarm pulse at the RTCC output pin is programmable using the PIV bit (RTCALRM<13>).

The RTCC pin is also capable of outputting the seconds clock. The user can select between the alarm pulse – generated by the RTCC module, or the seconds clock output. The RTSECSEL bit (RTCCON<7>) selects between these two outputs. When RTSECSEL = 0, the alarm pulse is selected. When RTSECSEL = 1, the seconds clock is selected (see Figure 29-9).

Note:	Changing any of the alarm time, date and alarm registers, other then the RTCOE (RTCCON<0>), while the alarm is enabled (ALRMEN = 1), can result in a false alarm event leading to a false alarm interrupt. To avoid a false alarm event and to perform a safe write to the alarm registers, the timer and alarm values should only be changed while the RTCC is disabled (RTCCON<15>=0), or when ALRMSYNC (RTCALRM<12>) = 0.

Figure 29-8: Alarm Event Generation

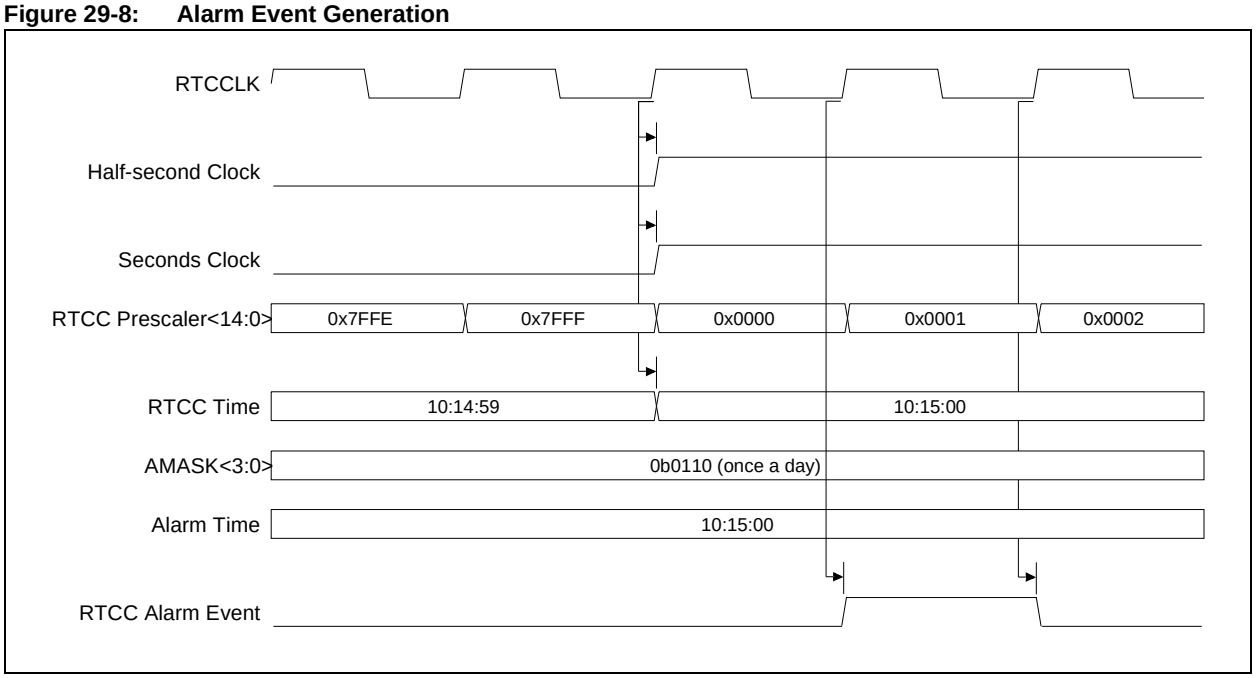
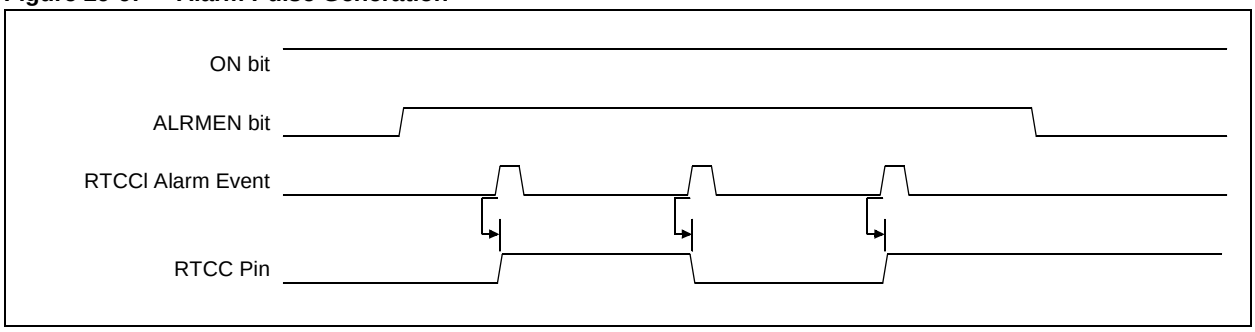


Figure 29-9: Alarm Pulse Generation



29.5 INTERRUPTS

The RTCC device has the ability to generate interrupts reflecting the alarm event that occurs when the RTCC timer matches the alarm registers. The match occurs on the unmasked portion of the time/date registers according to the AMASK (RTCALRM<11:8>) bit settings.

At every alarm event, an interrupt can be generated:

- The alarm interrupt is signalled by RTCCIF (IFS1<15>). This interrupt flag must be cleared in software.

In order to enable the RTCC interrupts, use the respective RTCC interrupt enable bit:

- RTCCIE (IEC1<15>).

The interrupt priority level bit and interrupt subpriority level bit must be also be configured:

- RTCCIP (IPC8<28:26>), RTCCIS (IPC8<25:24>)

Refer to **Section 8. "Interrupts"** for further details.

29.5.1 Interrupt Configuration

The RTCC has one dedicated interrupt flag bit RTCCIF and a corresponding interrupt enable/mask bit RTCCIE. RTCCIE is used to enable or disable the RTCC interrupt source. There is one specific RTCC interrupt vector.

The RTCCIF is set when the RTCC alarm registers match the RTCC time registers.

Note that the RTCCIF bit will be set without regard to the state of the corresponding enable bit. The IF bit can be polled by software if desired.

The RTCCIE bit is used to define the behavior of the Interrupt Controller (INT) when the corresponding RTCCIF bit is set. When the RTCCIE bit is clear, the INT module does not generate a CPU interrupt for the event. If the RTCCIE bit is set, the INT module will generate an interrupt to the CPU when the RTCCIF bit is set (subject to the priority and subpriority as outlined below).

It is the responsibility of the user's software routine that services a particular interrupt to clear the appropriate interrupt flag bit before the service routine is complete.

The priority of the RTCC peripheral can be set with the RTCCIP<2:0> bits. This priority defines the priority group to which the interrupt source will be assigned. The priority groups range from a value of 7 (the highest priority) to a value of 0 (which does not generate an interrupt). An interrupt being serviced will be preempted by an interrupt in a higher priority group.

The subpriority bits allow setting the priority of an interrupt source within a priority group. The values of the subpriority RTCCIS<1:0> range from 3 (the highest priority) to 0, the lowest priority. An interrupt within the same priority group, but having a higher subpriority value, will not preempt a lower subpriority interrupt that is in progress.

The priority group and subpriority bits allow more than one interrupt source to share the same priority and subpriority. If simultaneous interrupts occur in this configuration, the natural order of the interrupt sources within a priority/subpriority group pair determine the interrupt generated. The natural priority is based on the vector numbers of the interrupt sources. The lower the vector number the higher the natural priority of the interrupt. Any interrupts that were overridden by natural order will then generate their respective interrupts based on priority, subpriority, and natural order after the interrupt flag for the current interrupt is cleared.

After an enabled interrupt is generated, the CPU will jump to the vector assigned to that interrupt. The vector number for the interrupt is the same as the natural order number. The CPU will then begin executing code at the vector address. The user's code at this vector address should perform any application specific operations and clear the RTCCIF (IFS1<15>) interrupt flag, and then exit. Refer to the vector address table details in the **Section 8. "Interrupts"** for more information on interrupts.

Table 29-4: RTCC Interrupt Vector for Various Offsets with EBASE = 0x8000:0000

Interrupt	Vector/Natural Order	IRQ Number	Vector Address IntCtl.VS = 0x01	Vector Address IntCtl.VS = 0x02	Vector Address IntCtl.VS = 0x04	Vector Address IntCtl.VS = 0x08	Vector Address IntCtl.VS = 0x10
RTCC Alarm	35	47	8000 0660	8000 0AC0	8000 1380	8000 2500	8000 4800

Example 29-8: RTCC Initialization with Interrupts Enabled Code Example

```
/*
    The following code example illustrates an RTCC initialization with interrupts enabled.
    When the RTCC alarm interrupt is generated, the cpu will jump to the vector assigned to
    RTCC interrupt.
*/
                                  // assume RTCC write is enabled i.e. RTCWREN (RTCCON<3>) =1;
    IEC1CLR=0x00008000;           // disable RTCC interrupts

    RTCCONCLR=0x8000;             // turn off the RTCC
    while(RTCCON&0x40);           // wait for clock to be turned off

    IFS1CLR=0x00008000;           // clear RTCC existing event
    IPC8CLR=0x1f000000;           // clear the priority
    IPC8SET=0x0d000000;           // Set IPL=3, subpriority 1
    IEC1SET=0x00008000;           // Enable RTCC interrupts

    RTCTIME=0x16153300;           // safe to update time to 16 hr, 15 min, 33 sec
    RTCDATE=0x06102705;           // update the date to Friday 27 Oct 2006

    RTCALRMCLR=0xCFFF;            // clear ALRMEN, CHIME, AMASK and ARPT;
    ALRMTIME=0x16154300;          // set alarm time to 16 hr, 15 min, 43 sec
    ALRMDATE=0x06102705;          // set alarm date to Friday 27 Oct 2006

    RTCALRMSET=0x8000|0x00000600; // re-enable the alarm, set alarm mask at once per day

    RTCCONSET=0x8000;             // turn on the RTCC
    while(!(RTCCON&0x40));        // wait for clock to be turned on
```

Example 29-9: RTCC ISR Code Example

```
/*
    The following code example demonstrates a simple interrupt service routine for RTCC
    interrupts. The user's code at this vector should perform any application specific
    operations and must clear the RTCC interrupt flag before exiting.
*/

void__ISR(_RTCC_VECTOR, ipl3) __RTCCInterrupt(void)
{
                                      // ... perform application specific operations
                                      // in response to the interrupt

    IFS1CLR=0x00008000;               // be sure to clear RTCC interrupt flag
                                      // before exiting the service routine.
}
```

Note: The RTCC ISR code example shows MPLAB® C32 C compiler specific syntax. Refer to your compiler manual regarding support for ISRs.

29.6 OPERATION IN POWER-SAVING AND DEBUG MODES

Note: In this manual, a distinction is made between a power mode as it is used in a specific module, and a power mode as it is used by the device, e.g., sleep mode of the Comparator and SLEEP mode of the CPU. To indicate which type of power mode is intended, uppercase and lowercase letters (Sleep, Idle, Debug) signify a module power mode, and all uppercase letters (SLEEP, IDLE, DEBUG) signify a device power mode.

29.6.1 RTCC Operation in SLEEP Mode

When the device enters SLEEP mode, the system clock is disabled. The RTCC and alarm continue to operate while in SLEEP mode. The operation of the alarm is not affected by SLEEP. An alarm event can wake-up the CPU if the alarm interrupt has a higher priority than the current CPU IPL.

29.6.2 RTCC Operation in IDLE Mode

When the device enters IDLE mode, the system clock sources remain functional. The RTCC and alarm continue to operate while in IDLE mode. The operation of the alarm is not affected by IDLE. An alarm event can wake-up the CPU if the alarm interrupt has a higher priority than the current CPU IPL.

Bit SIDL (RTCCON<13>) selects the behavior of the module IDLE mode.

- If SIDL = 1, the PBCLK to the RTCC will be disabled. The PBCLK is the clock source for the AMASK (RTCALRM<11:8), CHIME (RTCALRM<14>), ALRMTIME, ALRMDATE and all of the synchronizers that provide the read data for RTCTIME, and some other bits like ALRMSYNC (RTCALRM<12>), ALRMEN (RTCALRM<15>) and RTCSYNC (RTCCON<2>). Thus, the SIDL functionality can be used to reduce the RTCC power consumption without affecting the functionality of the RTCC.
- If SIDL = 0, the module will continue normal operation in IDLE mode.

29.6.3 RTCC Operation in DEBUG Mode

The FRZ bit (RTCCON<14>) determines whether the RTCC module will run or stop while the CPU is executing Debug Exception code (i.e., the application is halted) in DEBUG mode. When FRZ = 0, the RTCC module continues to run even when the application is halted in DEBUG mode. When FRZ = 1 and the application is halted in DEBUG mode, the module will freeze its operations and make no changes to the state of the RTCC module. The Prescaler and RTCC timers will not increment. If a Configuration register normally causes the state of the module to change on a read, that functionality is disabled during Freeze. The module will resume its operation after the CPU resumes execution.

Note: The FRZ bit is readable and writable only when the CPU is executing in Debug Exception mode. In all other modes, the FRZ bit reads as '0'. If FRZ bit is changed during DEBUG mode, the new value does not take effect until the current Debug Exception mode is exited and re-entered. During the Debug Exception mode, the FRZ bit reads the state of the peripheral when entering DEBUG mode.

29.7 EFFECTS OF VARIOUS RESETS

29.7.1 Device Reset

When a device Reset occurs, the RTCALRM register is forced to its Reset state, causing the alarm to be disabled (if enabled prior to the Reset). Note, however, that if the RTCC is enabled, it will continue to operate when a device Reset occurs.

29.7.2 Power-on Reset

The RTCTIME and RTCDATE registers are not affected by the POR. A POR forces the device to its inactive state. Once the device exits the POR state, the clock registers should be reloaded with the desired values.

The timer prescaler values can only be reset by writing to the SECONDS register (RTCTIME<15:8>). No device Reset can affect the prescalers.

29.7.3 Watchdog Timer Reset

The Watchdog Timer Reset is equivalent to the device Reset.

29.7.4 Effects of the ON Bit

When ON bit (RTCCON<15>) = 0, the bits RTCSYNC (RTCCON<2>), HALFSEC (RTCCON<1>), and ALRMSYNC (RTCALRM<4>) are asynchronously reset and held in Reset. Also, the RTCC pin output is determined by the RTCOE bit (RTCCON<0>), gated by the ON bit.

29.8 PERIPHERALS USING RTCC MODULE

There are no other peripheral modules using the RTCC device.

29.9 I/O PIN CONTROL

Enabling the RTCC modules configures the I/O pin direction. When the RTCC module is enabled, configured and the output enabled, the I/O pin direction is properly configured as a digital output.

Table 29-5: I/O Pin Configuration for use with RTCC Module

Required Settings for Module Pin Control							
IO Pin Name	**Required**	**Module Control**	**Bit Field**	**TRIS(4)**	**Pin Type**	**Buffer Type**	**Description**
RTCC	Yes(1)	ON and RTCOE(2)	RTSECSEL = 1	X	O	CMOS	RTCC Seconds Clock
RTCC	Yes(1)	ON and RTCOE(2)	RTSECSEL = 0 and ALRMEN and PIV(3)	X	O	CMOS	RTCC Alarm Pulse

Legend: CMOS = CMOS compatible input or output
ST = Schmitt Trigger input with CMOS levels
I = Input
O = Output

Note 1: The RTCC pin is only required when Seconds Clock or Alarm Pulse output is needed. Otherwise, this pin can be used for general purpose IO and require the user to set the corresponding TRIS control register bit.

2: The ON (RTCCON<15>) and RTCOE (RTCCON<0>) bits are always required to validate the output function of the RTCC pin, either Seconds Clock or Alarm Pulse.

3: When RTSECSEL (RTCCON<7>) = 0, the RTCC pin output is the Alarm Pulse. If the ALRMEN (RTCALRM<15>) = 0, PIV (RTCALRM<13>) selects the value at the RTCC pin. When the ALRMEN = 1 the RTCC pin reflects the state of the Alarm Pulse.

4: The setting of the TRIS bit is irrelevant.

29.10 DESIGN TIPS

Question 1: ***If I do not use the RTCC output for my RTCC module, is this I/O pin available as a general purpose I/O pin?***

Answer: Yes. If you are not interested in outputting the Seconds Clock or the Alarm Pulse, you can use the RTCC pin as a general I/O pin as long as RTCC output is disabled (RTCCON<0>=0). Note that when used as a general purpose I/O pin, the user is responsible for configuring the respective data direction register (TRIS) for input or output.

Question 2: ***How do I make sure that when reading the RTCC time value I get the proper value, not affected by rollover (seconds to minutes, minutes to hours)?***

Answer: The easiest way to read the proper current time is to perform a double read of the RTCTIME register. The right time value is obtained when two consecutive readings are identical. The same is true when reading the RTCDATE register.

Question 3: ***Is the week of the day automatically calculated by the RTCC device when I set in a specific date, like 18 Jan 2006?***

Answer: No, the device doesn't perform this calculation automatically. When writing the RTCDATE register, you have to provide a valid value for the WDAY01 field (RTCDATE<3:0>), i.e., 3 for Wednesday, 18 Jan. However, from that point on, the RTCC device takes care of updating the day of the week field properly.

Question 4: ***Does the device perform the leap years calculation automatically or do I have to perform some corrections in the RTCC date?***

Answer: The RTCC device automatically performs leap year detection. No updates are necessary in the year range 2000 to 2099. However, when you program the date to the RTCDATE register you must enter a valid date. For example, don't program the RTCC with the date, 29 Feb 2007.

Question 5: ***Can I freely write to the RTCTIME and RTCDATE registers to update the current time or date?***

Answer: In short, no, you cannot write directly to the RTCTIME and RTCDATE registers. Actually, if the RTCC is disabled (RTCCON<15> = 0), you could update the time and date values at any time. However, if the RTCC is ON, further precautions must be taken. There is a safe window when the writes to the Time and Date registers are safely performed. That window is indicated by the RTCSYNC bit (RTCCON<2>). Any update to RTCTIME or RTCDATE registers should occur when RTCSYNC = 0. (The hardware actually ignores a write to the TIME and DATE registers that occurs during a rollover event when the RTCC clock is high, so the write operation could go undetected). Even more, if the alarm is enabled and the AMASK is half-second (RTCALRM<11:8>) any update to the RTCTIME and RTCDATE should occur when the ALRMSYNC is 0 (RTCALRM<12>). This ensures a proper functioning of the alarm trigger mechanism, without spurious alarm events being generated.

Notes:	You have to have the RTCWREN (RTCCON<3>) set in order to be able to update the RTCTIME and RTCDATE registers. Normally you should disable interrupts when trying to update the RTCTIME. If not, you cannot be sure that the write operation to the RTCTIME register occurs when RTCSYNC/ALRMSYNC is deasserted. Another way to perform the update of the system time and date is to turn off the RTCC, perform the write operations and then turn RTCC on again.

Question 6: ***Can I write to the ALRMTIME and ALRMDATE registers freely to update the alarm time or date?***

Answer: In short, no, you cannot update the Alarm Time and Date registers directly. The following steps are necessary:

- If the RTCC is disabled, i.e., ON = 0 (RTCCON<15>), you can perform the write to the ALRMTIME and ALRMDATE at any time.

Else, the write can occur only when ALRMSYNC = 0 (RTCALRM<12>).

Notes:	Normally you should disable interrupts when trying to update the alarm time and date. If not, you cannot be sure the write operation to the ALRMTIME/ALRMDATE registers occurs when ALRMSYNC is deasserted. Another way to perform the update of the alarm time and date is to turn off the RTCC, perform the write operations and then turn RTCC on again. Note that this approach has an impact on the timing accuracy, since the RTCC will not count while it is stopped. The exact same approach applies to the writable fields of the RTCALRM register: CHIME (RTCALRM<14>), AMASK (RTCALRM<11:8>), ALRMEN (RTCALRM<15>), ARPT (RTCALRM<7:0>) and PIV (RTCALRM<13>). If the RTCC is ON, the write should occur only when ALRMSYNC = 0.

Question 7: ***Can I toggle freely the RTCWREN bit in the RTCCON register?***

Answer: You can always clear the RTCWREN bit (RTCCON<3>). However, in order to enable the write to the RTCCTIME and RTCCDATE register, a proper sequence of operations must be performed. Refer to the **Section 29.3.9 "Write Lock"** for details.

29.11 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the PIC32MX device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the Real Time Clock and Calendar (RTCC)) module are:

Title	Application Note #
No related application notes at this time	N/A

Note: Visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the PIC32MX family of devices.

29.12 REVISION HISTORY

Revision A (October 2007)

This is the initial released version of this document.

Revision B (October 2007)

Updated document to remove Confidential status.

Revision C (April 2008)

Revised status to Preliminary; Revised U-0 to r-x.

Revision D (June 2008)

Revised Registers 29-1, bit 14; Revised Registers 29-26, 29-27, Footnote; Revised Examples 29-1 and 29-9; Change Reserved bits from “Maintain as” to “Write”; Added Note to ON bit (RTCCON Register).

Section 30. Reserved for Future

NOTES:

Section 31. DMA Controller Module

HIGHLIGHTS

This section of the manual contains the following topics:

31.1 INTRODUCTION

The Direct Memory Access (DMA) controller is a bus master module that is useful for data transfers between different peripherals without intervention from the CPU. The source and destination of a DMA transfer can be any of the memory-mapped modules included in the PIC32MX, e.g., memory, or one of the Peripheral Bus (PBUS) devices: SPI, UART, I^2C™, etc.

Following are some of the key features of this module:

- Up to four identical channels, including the following:
  - Auto-Increment Source and Destination Address registers
  - Source and Destination Pointers
- Automatic Word-Size Detection, featuring the following:
  - Transfer granularity down to byte level
  - Bytes need not be word-aligned at source and destination
- Fixed Priority Channel Arbitration
- Flexible DMA Channel Operating modes, including the following:
  - Manual (software) or automatic (interrupt) DMA requests
  - One-Shot or Auto-Repeat Block Transfer modes
  - Channel-to-channel chaining
- Flexible DMA Requests, featuring the following:
  - A DMA request can be selected from any of the peripheral interrupt sources
  - Each channel can select any interrupt as its DMA request source
  - A DMA transfer abort can be selected from any of the peripheral interrupt sources
  - Automatic transfer termination upon a data pattern match
- Multiple DMA Channel Status Interrupts, supplying the following:
  - DMA channel block transfer complete
  - Source empty or half empty
  - Destination full or half full
  - DMA transfer aborted due to an external event
  - Invalid DMA address generated
- DMA Debug Support Features, including the following:
  - Most recent address accessed by a DMA channel
  - Most recent DMA channel to transfer data
- CRC Generation Module, featuring the following:
  - CRC module can be assigned to any of the available channels
  - CRC module is highly configurable

Table 31-1: DMA Controller Features

Unaligned Transfers	Different Source and Destination Sizes	Memory-to-Memory Transfers	Memory-to-Peripheral Transfers	Channel Auto-Enable	Events Start/Stop	Pattern Match Detection	Channel Chaining	CRC Calculation
Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

31.1.1 DMA Operation

- A DMA channel transfers data from a source to a destination without CPU intervention. The source and destination start addresses define the start address of the source and destination, respectively.
- Both the source and destination have independently configurable sizes and the number of the transferred bytes is independent of the source and destination sizes.
- A transfer is initiated either by software or by an interrupt request. The user can select any interrupt on the device to start a DMA transfer.
- Upon transfer initiation, the DMA controller will perform a cell transfer and the channel remains enabled until a block transfer is complete. When a channel is disabled, further transfers will be prohibited until the channel is re-enabled.
- The DMA channel uses separate pointers to keep track of the current word locations at the source and destination.
- Interrupts can be generated when the Source/Destination Pointer is half of the source/destination size, or when the source/destination counter reaches the end of the source/destination.
- A DMA transfer can be aborted by the software, by a pattern match or by an interrupt event. The transfer will also stop when an address error is detected.

Figure 31-1: Typical DMA Source to Destination Transfer Diagram

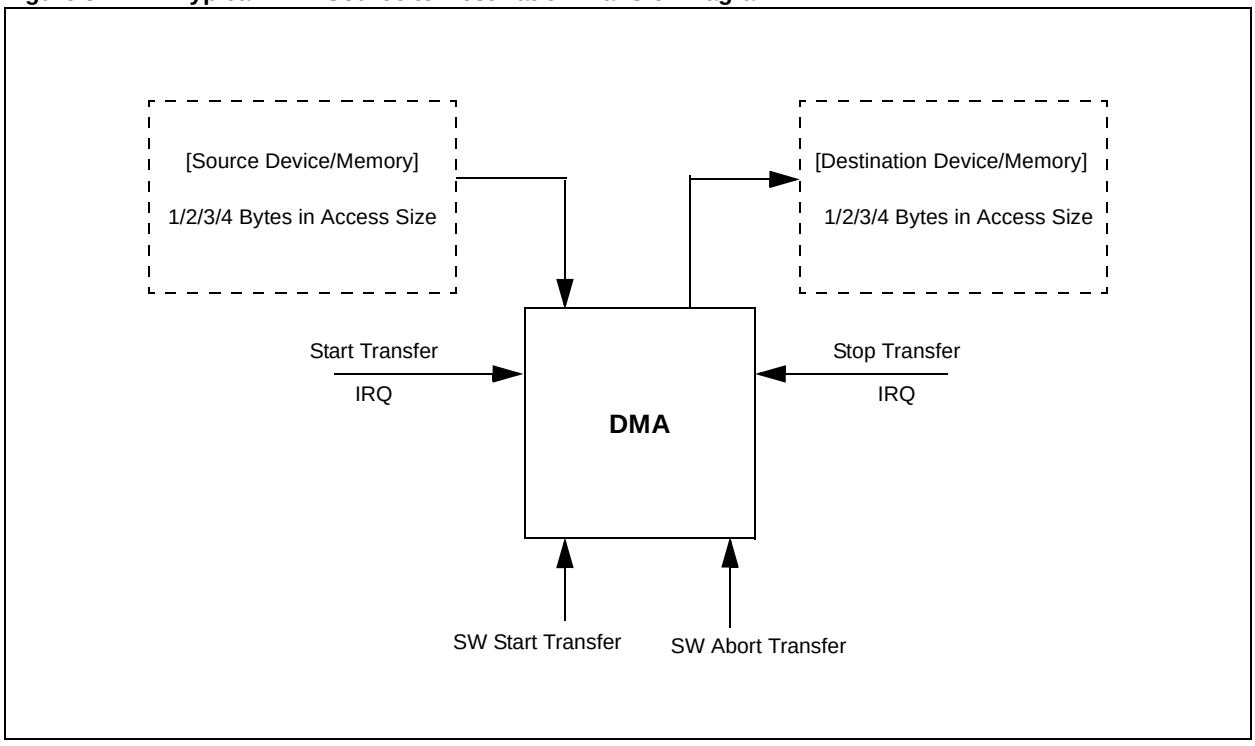


Figure 31-2: DMA Module Block Diagram

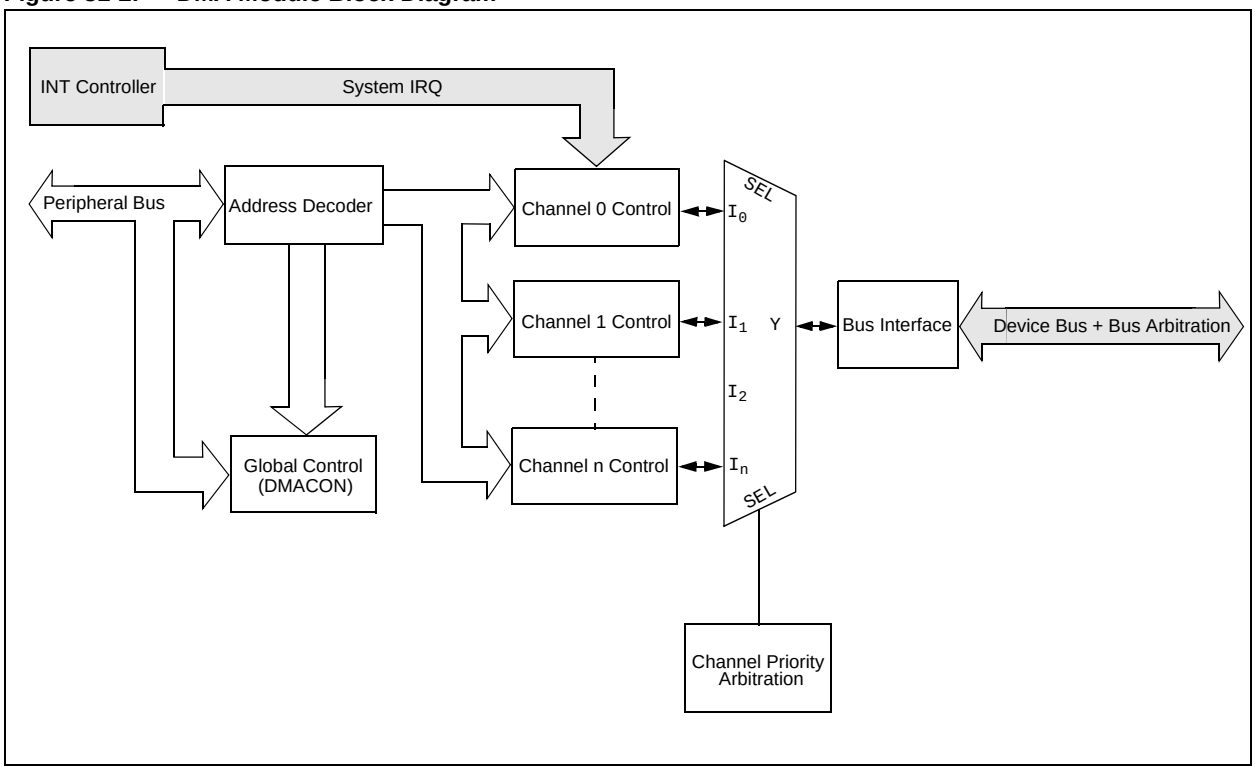
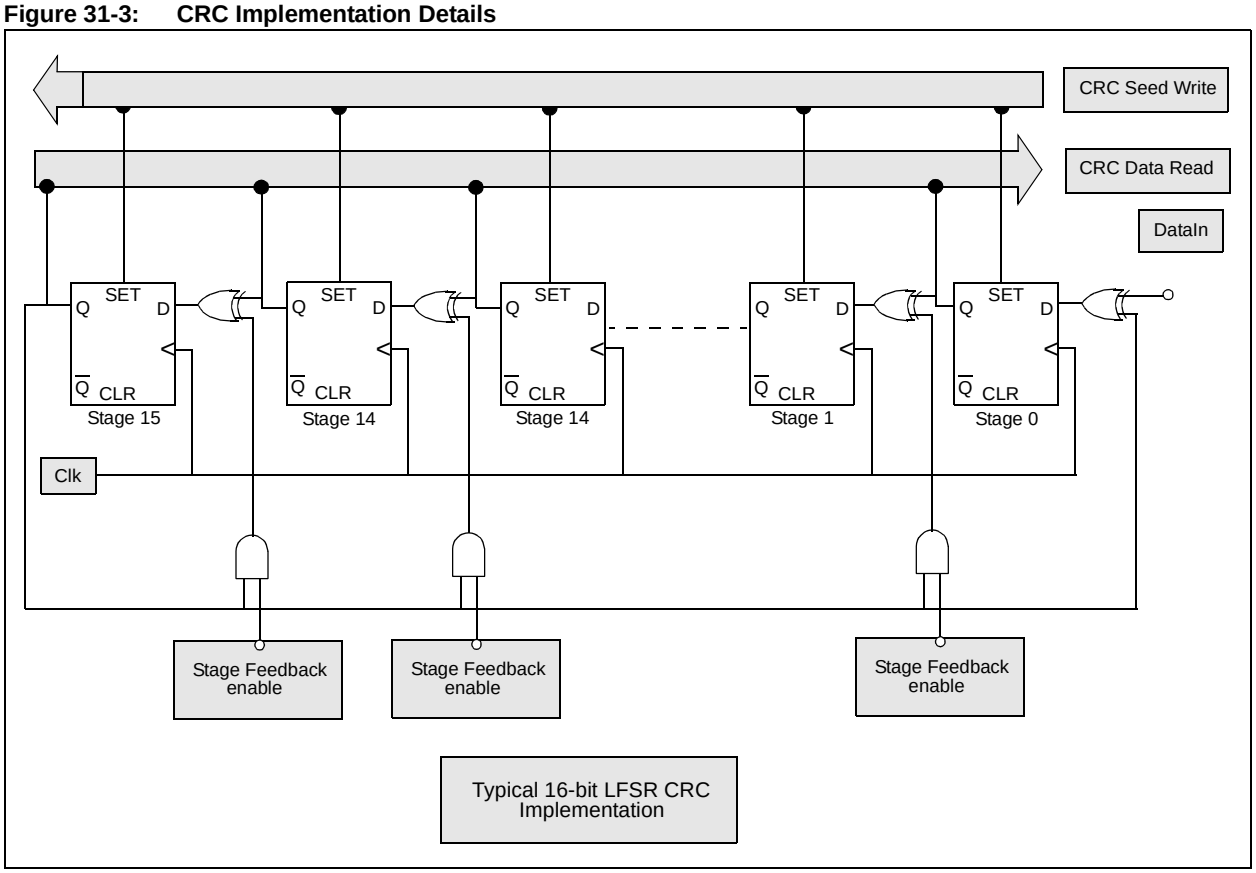


Figure 31-3: CRC Implementation Details



31.2 STATUS AND CONTROL REGISTERS

Note:	Each PIC32MX device variant may have one or more DMA channels. An 'x' used in the names of control/Status bits and registers denotes the particular channel. Refer to the specific device data sheets for more details.

The DMA module consists of the following Special Function Registers (SFRs):

- DMACON: Control Register for the DMA Controller
  DMACONCLR, DMACONSET, DMACONINV: Atomic Bit Manipulation, Write-only Registers for DMACON
- DMASTAT: Status Register for the DMA Module
- DMAADDR: DMA Address Register
- DCRCCON: DMA CRC Control Register
  DCRCCONCLR, DCRCCONSET, DCRCCONINV: Atomic Bit Manipulation, Write-only Registers for DCRCCON
- DCRCDATA: DMA CRC Data Register – The initial value of the CRC generator
  DCRCDATACLR, DCRCDATASET, DCRCDATAINV: Atomic Bit Manipulation, Write-only Registers for DCRCDATA
- DCRCXOR: DMA CRC XOR Enable Register – Provides a description of the generator polynomial for CRC calculation
  DCRCXORCLR, DCRCXORSET, DCRCXORINV: Atomic Bit Manipulation, Write-only Registers for DCRCXOR
- DCHxCON: Channel x Control Register
  DCHxCONCLR, DCHxCONSET, DCHxCONINV: Atomic Bit Manipulation, Write-only Registers for DCHXCON
- DCHxECON: Channel x Event Control Register
  DCHxECONCLR, DCHxECONSET, DCHxECONINV: Atomic Bit Manipulation, Write-only Registers for DCHXCON
- DCHxINT: Channel x Interrupt Control Register
  DCHxINTCLR, DCHxINTSET, DCHxINTINV: Atomic Bit Manipulation, Write-only Registers for DCHXINT
- DCHxSSA: Channel x Source Start Address Register
  DCHxSSACLR, DCHxSSASET, DCHxSSAINV: Atomic Bit Manipulation, Write-only Registers for DCHxSSA
- DCHxDSA: Channel x Destination Start Address Register
  DCHxDSACLR, DCHxDSASET, DCHxDSAINV: Atomic Bit Manipulation, Write-only Registers for DCHxDSA
- DCHxSSIZ: Channel x Source Size Register
  DCHxSSIZCLR, DCHxSSIZSET, DCHxSSIZINV: Atomic Bit Manipulation, Write-only Registers for DCHxSSIZ
- DCHxDSIZ: Channel x Destination Size Register
  DCHxDSIZCLR, DCHxDSIZSET, DCHxDSIZINV: Atomic Bit Manipulation, Write-only Registers for DCHxDSIZ
- DCHxSPTR: Channel x Source Pointer Register
- DCHxDPTR: Channel x Destination Pointer Register
- DCHxCSIZ: Channel x Cell-Size Register
  DCHxCSIZCLR, DCHxCSIZSET, DCHxCSIZINV: Atomic Bit Manipulation, Write-only Registers for DCHxCSIZ
- DCHxCPTR: Channel x Cell Pointer Register
- DCHxDAT: Channel x Pattern Data Register
  DCHxDATCLR, DCHxDATSET, DCHxDATINV: Atomic Bit Manipulation, Write-only Registers for DCHxDAT

The following table provides a brief summary of DMA-module-related registers. Corresponding registers appear after the summary, followed by a detailed description of each register.

Table 31-2: DMA SFRs Summary

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
DMACON	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	ON	FRZ	SIDL	SUSPEND	—	—	—	—
	7:0	—	—	—	—	—	—	—	—
DMACONCLR	31:0	Write clears selected bits in DMACON, read yields undefined value							
DMACONSET	31:0	Write sets selected bits in DMACON, read yields undefined value							
DMACONINV	31:0	Write inverts selected bits in DMACON, read yields undefined value							
DMASTAT	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	—	—	—	—	RDWR	—	DMACH<1:0>	
DMAADDR	31:24	DMAADDR<31-24>							
	23:16	DMAADDR<23-16>							
	15:8	DMAADDR<15-8>							
	7:0	DMAADDR<7-0>							
DCRCCON	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	PLEN<3:0>			
	7:0	CRCEN	CRCAPP	—	—	—	—	CRCCH<1:0>	
DCRCCONCLR	31:0	Write clears selected bits in DCRCCON, read yields undefined value							
DCRCCONSET	31:0	Write sets selected bits in DCRCCON, read yields undefined value							
DCRCCONINV	31:0	Write inverts selected bits in DCRCCON, read yields undefined value							
DCRCDATA	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	DCRCDATA<15:8>							
	7:0	DCRCDATA<7:0>							
DCRCDATACLR	31:0	Write clears selected bits in DCRCDATA, read yields undefined value							
DCRCDATASET	31:0	Write sets selected bits in DCRCDATA, read yields undefined value							
DCRCDATAINV	31:0	Write inverts selected bits in DCRCDATA, read yields undefined value							
DCRCXOR	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	DCRCXOR<15:8>							
	7:0	DCRCXOR<7:0>							
DCRCXORCLR	31:0	Write clears selected bits in DCRCXOR, read yields undefined value							
DCRCXORSET	31:0	Write sets selected bits in DCRCXOR, read yields undefined value							
DCRCXORINV	31:0	Write inverts selected bits in DCRCXOR, read yields undefined value							
DCHxCON	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	CHCHNS
	7:0	CHEN	CHAED	CHCHN	CHAEN	—	CHEDET	CHPRI<1:0>	

Table 31-2: DMA SFRs Summary (Continued)

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
DCHxCONCLR	31:0	Write clears selected bits in DCHxCON, read yields undefined value							
DCHxCONSET	31:0	Write sets selected bits in DCHxCON, read yields undefined value							
DCHxCONINV	31:0	Write inverts selected bits in DCHxCON, read yields undefined value							
DCHxECON	31:24	—	—	—	—	—	—	—	—
	23:16	CHAIRQ<7:0>							
	15:8	CHSIRQ<7:0>							
	7:0	CFORCE	CABORT	PATEN	SIRQEN	AIRQEN	—	—	—
DCHxECONCLR	31:0	Write clears selected bits in DCHxECON, read yields undefined value							
DCHxECONSET	31:0	Write sets selected bits in DCHxECON, read yields undefined value							
DCHxECONINV	31:0	Write inverts selected bits in DCHxECON, read yields undefined value							
DCHxINT	31:24	—	—	—	—	—	—	—	—
	23:16	CHSDIE	CHSHIE	CHDDIE	CHDHIE	CHBCIE	CHCCIE	CHTAIE	CHERIE
	15:8	—	—	—	—	—	—	—	—
	7:0	CHSDIF	CHSHIF	CHDDIF	CHDHIF	CHBCIF	CHCCIF	CHTAIF	CHERIF
DCHxINTCLR	31:0	Write clears selected bits in DCHxINT, read yields undefined value							
DCHxINTSET	31:0	Write sets selected bits in DCHxINT, read yields undefined value							
DCHxINTINV	31:0	Write inverts selected bits in DCHxINT, read yields undefined value							
DCHxSSA	31:24	CHSSA<31:24>							
	23:16	CHSSA<23:16>							
	15:8	CHSSA<15:8>							
	7:0	CHSSA<7:0>							
DCHxSSACLR	31:0	Write clears selected bits in DCHxSSA, read yields undefined value							
DCHxSSASET	31:0	Write sets selected bits in DCHxSSA, read yields undefined value							
DCHxSSAINV	31:0	Write inverts selected bits in DCHxSSA, read yields undefined value							
DCHxDSA	31:24	CHDSA<31:24>							
	23:16	CHDSA<23:16>							
	15:8	CHDSA<15:8>							
	7:0	CHDSA<7:0>							
DCHxDSACLR	31:0	Write clears selected bits in DCHxDSA, read yields undefined value							
DCHxDSASET	31:0	Write sets selected bits in DCHxDSA, read yields undefined value							
DCHxDSAINV	31:0	Write inverts selected bits in DCHxDSA, read yields undefined value							
DCHxSSIZ	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	CHSSIZ<7:0>							
DCHxSSIZCLR	31:0	Write clears selected bits in DCHxSSIZ, read yields undefined value							
DCHxSSIZSET	31:0	Write sets selected bits in DCHxSSIZ, read yields undefined value							
DCHxSSIZINV	31:0	Write inverts selected bits in DCHxSSIZ, read yields undefined value							
DCHxDSIZ	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	CHDSIZ<7:0>							
DCHxDSIZCLR	31:0	Write clears selected bits in DCHxDSIZ, read yields undefined value							
DCHxDSIZSET	31:0	Write sets selected bits in DCHxDSIZ, read yields undefined value							
DCHxDSIZINV	31:0	Write inverts selected bits in DCHxDSIZ, read yields undefined value							

Table 31-2: DMA SFRs Summary (Continued)

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
DCHxSPTR	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	CHSPTR<7:0>							
DCHxDPTR	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	CHDPTR<7:0>							
DCHxCSIZ	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	CHCSIZ<7:0>							
DCHxCSIZCLR	31:0	Write clears selected bits in DCHxCSIZ, read yields undefined value							
DCHxCSIZSET	31:0	Write sets selected bits in DCHxCSIZ, read yields undefined value							
DCHxCSIZINV	31:0	Write inverts selected bits in DCHxCSIZ, read yields undefined value							
DCHxCPTR	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	CHCPTR<7:0>							
DCHxDAT	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	CHPDAT<7:0>							
DCHxDATCLR	31:0	Write clears selected bits in DCHxDAT, read yields undefined value							
DCHxDATSET	31:0	Write sets selected bits in DCHxDAT, read yields undefined value							
DCHxDATINV	31:0	Write inverts selected bits in DCHxDAT, read yields undefined value							
IFS1CLR	31:0	Write clears selected bits in IFS1, read yields undefined value							
IFS1SET	31:0	Write sets selected bits in IFS1, read yields undefined value							
IFS1INV	31:0	Write inverts selected bits in IFS1, read yields undefined value							
IEC1	31:24	—	—	—	—	—	—	USBIE	FCEIE
	23:16	—	—	—	—	DMA3IE	DMA2IE	DMA1IE	DMA0IE
	15:8	RTCCIE	FSCMIE	I2C2MIE	I2C2SIE	I2C2BIE	U2TXIE	U2RXIE	U2EIE
	7:0	SPI2RXIE	SPI2TXIE	SPI2EIE	CMP2IE	CMP1IE	PMPIE	AD1IE	CNIE
IEC1CLR	31:0	Write clears selected bits in IEC1, read yields undefined value							
IEC1SET	31:0	Write sets selected bits in IEC1, read yields undefined value							
IEC1INV	31:0	Write inverts selected bits in IEC1, read yields undefined value							
IPC9	31:24	—	—	—	DMA3IP<2:0>			DMA3IS<1:0>	
	23:16	—	—	—	DMA2IP<2:0>			DMA2IS<1:0>	
	15:8	—	—	—	DMA1IP<2:0>			DMA1IS<1:0>	
	7:0	—	—	—	DMA0IP<2:0>			DMA0IS<1:0>	

Table 31-2: DMA SFRs Summary (Continued)

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
IPC9CLR	31:0	Write clears selected bits in IPC9, read yields undefined value							
IPC9SET	31:0	Write sets selected bits in IPC9, read yields undefined value							
IPC9INV	31:0	Write inverts selected bits in IPC9, read yields undefined value							
IFS1	31:24	—	—	—	—	—	—	USBIF	FCEIF
	23:16	—	—	—	—	DMA3IF	DMA2IF	DMA1IF	DMA0IF
	15:8	RTCCIF	FSCMIF	I2C2MIF	I2C2SIF	I2C2BIF	U2TXIF	U2RXIF	U2EIF
	7:0	SPI2RXIF	SPI2TXIF	SPI2EIF	CMP2IF	CMP1IF	PMPIF	AD1IF	CNIF
IFS1CLR	31:0	Write clears selected bits in IFS1, read yields undefined value							
IFS1SET	31:0	Write sets selected bits in IFS1, read yields undefined value							
IFS1INV	31:0	Write inverts selected bits in IFS1, read yields undefined value							
IEC1	31:24	—	—	—	—	—	—	USBIE	FCEIE
	23:16	—	—	—	—	DMA3IE	DMA2IE	DMA1IE	DMA0IE
	15:8	RTCCIE	FSCMIE	I2C2MIE	I2C2SIE	I2C2BIE	U2TXIE	U2RXIE	U2EIE
	7:0	SPI2RXIE	SPI2TXIE	SPI2EIE	CMP2IE	CMP1IE	PMPIE	AD1IE	CNIE
IEC1CLR	31:0	Write clears selected bits in IEC1, read yields undefined value							
IEC1SET	31:0	Write sets selected bits in IEC1, read yields undefined value							
IEC1INV	31:0	Write inverts selected bits in IEC1, read yields undefined value							
IPC9	31:24	—	—	—	DMA3IP<2:0>			DMA3IS<1:0>	
	23:16	—	—	—	DMA2IP<2:0>			DMA2IS<1:0>	
	15:8	—	—	—	DMA1IP<2:0>			DMA1IS<1:0>	
	7:0	—	—	—	DMA0IP<2:0>			DMA0IS<1:0>	
IPC9CLR	31:0	Write clears selected bits in IPC9, read yields undefined value							
IPC9SET	31:0	Write sets selected bits in IPC9, read yields undefined value							
IPC9INV	31:0	Write inverts selected bits in IPC9, read yields undefined value							

Register 31-1: DMACON: DMA Controller Control Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	r-x	r-x	r-x	r-x
ON	FRZ	SIDL	SUSPEND	—	—	—	—
bit 15							bit 8

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15 **ON:** DMA On bit

1 = DMA module is enabled
0 = DMA module is disabled

Note: When using 1:1 PBCLK divisor, the user’s software should not read/write the peripheral’s SFRs in the SYSCLK cycle immediately following the instruction that clears the module’s ON bit.

bit 14 **FRZ:** DMA Freeze bit

1 = DMA is frozen during DEBUG mode
0 = DMA continues to run during DEBUG mode
Note: FRZ is writable in DEBUG Exception mode only, it is forced to ‘0’ in Normal mode.

bit 13 **SIDL:** Stop in IDLE Mode bit

1 = DMA transfers are frozen during SLEEP
0 = DMA transfers continue during SLEEP

bit 12 **SUSPEND:** DMA Suspend bit

1 = DMA transfers are suspended to allow CPU uninterrupted access to data bus
0 = DMA operates normally

bit 11-0 **Reserved:** Write ‘0’; ignore read

Register 31-2: DMACONCLR: DMACON Clear Register

Write clears selected bits in DMACON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in DMACON**

A write of '1' in one or more bit positions clears the corresponding bit(s) in DMACON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DMACONCLR = 0x00008000` will clear bit 15 in DMACON register.

Register 31-3: DMACONSET: DMACON Set Register

Write sets selected bits in DMACON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in DMACON**

A write of '1' in one or more bit positions sets the corresponding bit(s) in DMACON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DMACONSET = 0x00008000` will set bit 15 in DMACON register.

Register 31-4: DMACONINV: DMACON Invert Register

Write inverts selected bits in DMACON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in DMACON**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in DMACON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DMACONINV = 0x00009000` will invert bits 15 and 12 in DMACON register.

Register 31-5: DMASTAT: DMA Status Register (1)

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

r-x	r-x	r-x	r-x	R-0	r-x	R-0	R-0
—	—	—	—	RDWR	—	DMACH<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-4 **Reserved:** Write ‘0’; ignore read

bit 3 **RDWR:** Read/Write Status bit
1 = Last DMA bus access was a read
0 = Last DMA bus access was a write

bit 2 **Reserved:** Write ‘0’; ignore read

bit 1-0 **DMACH<1:0>:** DMA Channel bits

Note 1: This register contains the value of the most recent active DMA channel.

Register 31-6: DMAADDR: DMA Address Register(1)

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
DMAADDR<31:24>							
bit 31							bit 24

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
DMAADDR<23:16>							
bit 23							bit 16

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
DMAADDR<15:8>							
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
DMAADDR<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **DMAADDR<31:0>:** DMA Module Address bits

Note 1: This register contains the address of the most recent DMA access.

Register 31-7: DCRCCON: DMA CRC Control Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	PLEN<3:0>			
bit 15							bit 8

R/W-0	R/W-0	r-x	r-x	r-x	r-x	R/W-0	R/W-0
CRCEN	CRCAPP	—	—	—	—	CRCCH<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-12 **Reserved:** Write ‘0’; ignore read

bit 11-8 **PLEN<3:0>:** Polynomial Length bits

Denotes the length of the polynomial –1.

bit 7 **CRCEN:** CRC Enable bit

1 = CRC module is enabled and channel transfers are routed through the CRC module
0 = CRC module is disabled and channel transfers proceed normally

bit 6 **CRCAPP:** CRC Append Mode bit

1 = Data read will be passed to the CRC, to be included in the CRC calculation, but is not written to the destination register
When a block transfer completes, the calculated CRC will be written to the location given by DCHxDSA.

0 = Channel behaves normally, with the CRC being calculated as data is transferred from the source to the destination

bit 5-2 **Reserved:** Write ‘0’; ignore read

bit 1-0 **CRCCH<1:0>:** CRC Channel Select bits

11 = CRC is assigned to Channel 3
10 = CRC is assigned to Channel 2
01 = CRC is assigned to Channel 1
00 = CRC is assigned to Channel 0

Register 31-8: DCRCCONCLR: DCRCCON Clear Register

Write clears selected bits in DCRCCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clear selected bits in DCRCCON**

A write of '1' in one or more bit positions clears the corresponding bit(s) in DCRCCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCRCCONCLR = 0x00000101` will clear bits 8 and 0 in DCRCCON register.

Register 31-9: DCRCCONSET: DCRCCON Set Register

Write sets selected bits in DCRCCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Set selected bits in DCRCCON**

A write of '1' in one or more bit positions sets the corresponding bit(s) in DCRCCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCRCCONSET = 0x00000101` will set bits 8 and 0 in DCRCCON register.

Register 31-10: DCRCCONINV: DCRCCON Invert Register

Write inverts selected bits in DCRCCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in DCRCCON**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in DCRCCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCRCCONINV = 0x00000101` will invert bits 8 and 0 in DCRCCON register.

Register 31-11: DCRCDATA: DMA CRC Data Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
DCRCDATA<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
DCRCDATA<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15-0 **DCRCDATA<15:0>:** CRC Data Register bits

Writing to this register will seed the CRC generator. Reading from this register will return the current value of the CRC. Bits > PLEN will return ‘0’ on any read.

Register 31-12: DCRCDATACLR: DCRCDATA Clear Register

Write clears selected bits in DCRCDATA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clear selected bits in DCRCDATA**

A write of '1' in one or more bit positions clears the corresponding bit(s) in DCRCDATA register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCRCDATACLR = 0x000000FF` will clear bits 7 through 0 in DCRCDATA register.

Register 31-13: DCRCDATASET: DCRCDATA Set Register

Write sets selected bits in DCRCDATA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Set selected bits in DCRCDATA**

A write of '1' in one or more bit positions sets the corresponding bit(s) in DCRCDATA register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCRCDATASET = 0x0000FFFF` will set bits 15 through 0 in DCRCDATA register.

Register 31-14: DCRCDATAINV: DCRCDATA Invert Register

Write inverts selected bits in DCRCDATA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in DCRCDATA**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in DCRCDATA register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCRCDATAINV = 0x00000101` will invert bits 8 and 0 in DCRCDATA register.

Register 31-15: DCRCXOR: DMA CRCXOR Enable Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
DCRCXOR<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
DCRCXOR<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15-0 **DCRCXOR<15:0>:** CRC XOR Register bits

1 = Enable the XOR input to the Shift register

0 = Disable the XOR input to the Shift register; data is shifted directly in from the previous stage in the register

Register 31-16: DCRCXORCLR: DCRCXOR Clear Register

Write clears selected bits in DCRCXOR, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clear selected bits in DCRCXOR**

A write of '1' in one or more bit positions clears the corresponding bit(s) in DCRCXOR register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCRCXORCLR = 0x00000101` will clear bits 8 and 0 in DCRCXOR register.

Register 31-17: DCRCXORSET: DCRCXOR Set Register

Write sets selected bits in DCRCXOR, read yields undefined value	
bit 31	bit 0

bit 31-0 **Set selected bits in DCRCXOR**

A write of '1' in one or more bit positions sets the corresponding bit(s) in DCRCXOR register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCRCXORSET = 0x00000101` will set bits 8 and 0 in DCRCXOR register.

Register 31-18: DCRCXORINV: DCRCXOR Invert Register

Write inverts selected bits in DCRCXOR, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in DCRCXOR**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in DCRCXOR register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCRCXORINV = 0x000000FF` will invert bits 7 through 0 in DCRCXOR register.

Register 31-19: DCHxCON: DMA Channel x Control Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	R/W-0
—	—	—	—	—	—	—	CHCHNS
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	r-x	R-0	R/W-0	R/W-0
CHEN	CHAED	CHCHN	CHAEN	—	CHEDET	CHPRI<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-9 **Reserved:** Write ‘0’; ignore read

bit 8 **CHCHNS:** Chain Channel Selection bit
1 = Chain to channel lower in natural priority (CH1 will be enabled by CH2 transfer complete)
0 = Chain to channel higher in natural priority (CH1 will be enabled by CH0 transfer complete)
Note: The chain selection bit takes effect when chaining is enabled, i.e., CHCHN = 1.

bit 7 **CHEN:** Channel Enable bit
1 = Channel is enabled
0 = Channel is disabled

bit 6 **CHAED:** Channel Allow Events If Disabled bit
1 = Channel start/abort events will be registered, even if the channel is disabled
0 = Channel start/abort events will be ignored if the channel is disabled

bit 5 **CHCHN:** Channel Chain Enable bit
1 = Allow channel to be chained to channel higher in natural priority
0 = Do not chain to channel higher in natural priority

bit 4 **CHAEN:** Channel Automatic Enable bit
1 = Channel is continuously enabled, and not automatically disabled after a block transfer is complete
0 = Channel is disabled on block transfer complete

bit 3 **Reserved:** Write ‘0’; ignore read

bit 2 **CHEDET:** Channel Event Detected bit
1 = An event has been detected
0 = No events have been detected

bit 1-0 **CHPRI<1:0>:** Channel Priority bits
11 = Channel has priority 3 (highest)
10 = Channel has priority 2
01 = Channel has priority 1
00 = Channel has priority 0

Register 31-20: DCHxCONCLR: DCHxCON Clear Register

Write clears selected bits in DCHxCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clear selected bits in DCHxCON**

A write of '1' in one or more bit positions clears the corresponding bit(s) in DCHxCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCHxCONCLR = 0x00000081` will clear bits 7 and 0 in DCHxCON register.

Register 31-21: DCHxCONSET: DCHxCON Set Register

Write sets selected bits in DCHxCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Set selected bits in DCHxCON**

A write of '1' in one or more bit positions sets the corresponding bit(s) in DCHxCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCHxCONSET = 0x00000081` will set bits 7 and 0 in DCHxCON register.

Register 31-22: DCHxCONINV: DCHxCON Invert Register

Write inverts selected bits in DCHxCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in DCHxCON**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in DCHxCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCHxCONINV = 0x00000081` will invert bits 7 and 0 in DCHxCON register.

Register 31-23: DCHxECON: DMA Channel x Event Control Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
CHAIRQ<7:0>							
bit 23							bit 16

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
CHSIRQ<7:0>							
bit 15							bit 8

S-0	S-0	R/W-0	R/W-0	R/W-0	r-x	r-x	r-x
CFORCE	CABORT	PATEN	SIRQEN	AIRQEN	—	—	—
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-24 **Reserved:** Write ‘0’; ignore read

bit 23-16 **CHAIRQ<7:0>:** IRQ that will abort Channel Transfer bits

11111111 = Interrupt 255 will abort any transfers in progress and set CHAIF flag

• • •

00000001 = Interrupt 1 will abort any transfers in progress and set CHAIF flag
00000000 = Interrupt 0 will abort any transfers in progress and set CHAIF flag

bit 15-8 **CHSIRQ<7:0>:** IRQ that will Start Channel Transfer bits

11111111 = Interrupt 255 will initiate a DMA transfer

• • •

00000001 = Interrupt 1 will initiate a DMA transfer
00000000 = Interrupt 0 will initiate a DMA transfer

bit 7 **CFORCE:** DMA Forced Transfer bit

1 = A DMA transfer is forced to begin when this bit is written to a ‘1’
0 = This bit always reads ‘0’

bit 6 **CABORT:** DMA Abort Transfer bit

1 = A DMA transfer is aborted when this bit is written to a ‘1’
0 = This bit always reads ‘0’

bit 5 **PATEN:** Channel Pattern Match Abort Enable bit

1 = Abort transfer and clear CHEN on pattern match
0 = Pattern match is disabled

bit 4 **SIRQEN:** Channel Start IRQ Enable bit

1 = Start channel cell transfer if an interrupt matching CHSIRQ occurs
0 = Interrupt number CHSIRQ is ignored and does not start a transfer

bit 3 **AIRQEN:** Channel Abort IRQ Enable bit

1 = Channel transfer is aborted if an interrupt matching CHAIRQ occurs
0 = Interrupt number CHAIRQ is ignored and does not terminate a transfer

bit 2-0 **Reserved:** Write ‘0’; ignore read

Register 31-24: DCHxECONCLR: DCHxECON Clear Register

Write clears selected bits in DCHxECON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in DCHxECON**

A write of '1' in one or more bit positions clears the corresponding bit(s) in DCHxECON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCHxECONCLR = 0x00000088` will clear bits 7 and 3 in DCHxECON register.

Register 31-25: DCHxECONSET: DCHxECON Set Register

Write sets selected bits in DCHxECON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in DCHxECON**

A write of '1' in one or more bit positions sets the corresponding bit(s) in DCHxECON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCHxECONSET = 0x00000088` will set bits 7 and 3 in DCHxECON register.

Register 31-26: DCHxECONINV: DCHxECON Invert Register

Write inverts selected bits in DCHxECON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in DCHxECON**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in DCHxECON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCHxECONINV = 0x00000088` will invert bits 7 and 3 in DCHxECON register.

Register 31-27: DCHxINT: DMA Channel x Interrupt Control Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CHSDIE	CHSHIE	CHDDIE	CHDHIE	CHBCIE	CHCCIE	CHTAIE	CHERIE
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CHSDIF	CHSHIF	CHDDIF	CHDHIF	CHBCIF	CHCCIF	CHTAIF	CHERIF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-24 **Reserved:** Write‘0’; ignore read

bit 23 **CHSDIE:** Channel Source Done Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 22 **CHSHIE:** Channel Source Half Empty Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 21 **CHDDIE:** Channel Destination Done Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 20 **CHDHIE:** Channel Destination Half Full Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 19 **CHBCIE:** Channel Block Transfer Complete Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 18 **CHCCIE:** Channel Cell Transfer Complete Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 17 **CHTAIE:** Channel Transfer Abort Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 16 **CHERIE:** Channel Address Error Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 15-8 **Reserved:** Write ‘0’; ignore read

bit 7 **CHSDIF:** Channel Source Done Interrupt Flag bit
1 = Channel Source Pointer has reached end of source (CHSPTR == CHSSIZ)
0 = No interrupt is pending

Register 31-27: DCHxINT: DMA Channel x Interrupt Control Register

bit 6 **CHSHIF:** Channel Source Half Empty Interrupt Flag bit

1 = Channel Source Pointer has reached midpoint of source (CHSPTR == CHSSIZ/2)
0 = No interrupt is pending

bit 5 **CHDDIF:** Channel Destination Done Interrupt Flag bit

1 = Channel Destination Pointer has reached end of destination (CHDPTR == CHDSIZ)
0 = No interrupt is pending

bit 4 **CHDHIF:** Channel Destination Half Full Interrupt Flag bit

1 = Channel Destination Pointer has reached midpoint of destination (CHDPTR == CHDSIZ/2)
0 = No interrupt is pending

bit 3 **CHBCIF:** Channel Block Transfer Complete Interrupt Flag bit

1 = A block transfer has been completed (the larger of CHSSIZ/CHDSIZ bytes has been transferred), or a pattern match event occurs
0 = No interrupt is pending

bit 2 **CHCCIF:** Channel Cell Transfer Complete Interrupt Flag bit

1 = A cell transfer has been completed (CHCSIZ bytes have been transferred)
0 = No interrupt is pending

bit 1 **CHTAIF:** Channel Transfer Abort Interrupt Flag bit

1 = An interrupt matching CHAIRQ has been detected and the DMA transfer has been aborted
0 = No interrupt is pending

bit 0 **CHERIF:** Channel Address Error Interrupt Flag bit

1 = A channel address error has been detected
Either the source or the destination address is invalid.
0 = No interrupt is pending

Register 31-28: DCHxINTCLR: DCHxINT Clear Register

Write clears selected bits in DCHxINT, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in DCHxINT**

A write of '1' in one or more bit positions clears the corresponding bit(s) in DCHxINT register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCHxINTCLR = 0x00000081` will clear bits 7 and 0 in DCHxINT register.

Register 31-29: DCHxINTSET: DCHxINT Set Register

Write sets selected bits in DCHxINT, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in DCHxINT**

A write of '1' in one or more bit positions sets the corresponding bit(s) in DCHxINT register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCHxINTSET = 0x00000081` will set bits 7 and 0 in DCHxINT register.

Register 31-30: DCHxINTINV: DCHxINT Invert Register

Write inverts selected bits in DCHxINT, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in DCHxINT**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in DCHxINT register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCHxINTINV = 0x00000081` will invert bits 7 and 0 in DCHxINT register.

Register 31-31: DCHxSSA: DMA Channel x Source Start Address Register

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHSSA<31:24>							
bit 31							bit 24

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHSSA<23:16>							
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHSSA<15:8>							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHSSA<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **CHSSA<31:0>** Channel Source Start Address bits

Channel source start address.

Note: This must be the physical address of the source.

Register 31-32: DCHxSSACLR: DCHxSSA Clear Register

Write clears selected bits in DCHxSSA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in DCHxSSA**

A write of '1' in one or more bit positions clears the corresponding bit(s) in DCHxSSA register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCHxSSACLR = 0x000000ff` will clear bits 7 through 0 in DCHxSSA register.

Register 31-33: DCHxSSASET: DCHxSSA Set Register

Write sets selected bits in DCHxSSA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in DCHxSSA**

A write of '1' in one or more bit positions sets the corresponding bit(s) in DCHxSSA register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCHxSSASET = 0x000000ff` will set bits 7 through 0 in DCHxSSA register.

Register 31-34: DCHxSSAINV: DCHxSSA Invert Register

Write inverts selected bits in DCHxSSA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in DCHxSSA**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in DCHxSSA register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCHxSSAINV = 0x000000ff` will invert bits 7 through 0 in DCHxSSA register.

Register 31-35: DCHxDSA: DMA Channel x Destination Start Address Register

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHDSA<31:24>							
bit 31							bit 24

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHDSA<23:16>							
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHDSA<15:8>							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHDSA<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **CHDSA<31:0>:** Channel Destination Start Address bits

Channel destination start address.

Note: This must be the physical address of the destination.

Register 31-36: DCHxDSACLR: DCHxDSA Clear Register

Write clears selected bits in DCHxDSA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in DCHxDSA**

A write of '1' in one or more bit positions clears the corresponding bit(s) in DCHxDSA register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCHxDSACLR = 0x000000ff` will clear bits 7 through 0 in DCHxDSA register.

Register 31-37: DCHxDSASET: DCHxDSA Set Register

Write sets selected bits in DCHxDSA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in DCHxDSA**

A write of '1' in one or more bit positions sets the corresponding bit(s) in DCHxDSA register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCHxDSASET = 0x000000ff` will set bits 7 through 0 in DCHxDSA register.

Register 31-38: DCHxDSAINV: DCHxDSA Invert Register

Write inverts selected bits in DCHxDSA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in DCHxDSA**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in DCHxDSA register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCHxDSAINV = 0x000000ff` will invert bits 7 through 0 in DCHxDSA register.

Register 31-39: DCHxSSIZ: DMA Channel x Source Size Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHSSIZ<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7-0 **CHSSIZ<7:0>:** Channel Source Size bits

255 = 255 byte source size

• • •

2 = 2 byte source size
1 = 1 byte source size
0 = 256 byte source size

Register 31-40: DCHxSSIZCLR: DCHxSSIZ Clear Register

Write clears selected bits in DCHxSSIZ, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in DCHxSSIZ**

A write of '1' in one or more bit positions clears the corresponding bit(s) in DCHxSSIZ register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCHxSSIZCLR = 0x000000ff` will clear bits 7 through 0 in DCHxSSIZ register.

Register 31-41: DCHxSSIZSET: DCHxSSIZ Set Register

Write sets selected bits in DCHxSSIZ, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in DCHxSSIZ**

A write of '1' in one or more bit positions sets the corresponding bit(s) in DCHxSSIZ register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCHxSSIZSET = 0x000000ff` will set bits 7 through 0 in DCHxSSIZ register.

Register 31-42: DCHxSSIZINV: DCHxSSIZ Invert Register

Write inverts selected bits in DCHxSSIZ, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in DCHxSSIZ**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in DCHxSSIZ register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCHxSSIZINV = 0x000000ff` will invert bits 7 through 0 in DCHxSSIZ register.

Register 31-43: DCHxDSIZ: DMA Channel x Destination Size Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHDSIZ<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7-0 **CHDSIZ<7:0>:** Channel Destination Size bits

255 = 255 byte destination size

• • •

2 = 2 byte destination size
1 = 1 byte destination size
0 = 256 byte destination size

Register 31-44: DCHxDSIZCLR: DCHxDSIZ Clear Register

Write clears selected bits in DCHxDSIZ, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in DCHxDSIZ**

A write of '1' in one or more bit positions clears the corresponding bit(s) in DCHxDSIZ register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCHxDSIZCLR = 0x000000ff` will clear bits 7 through 0 in DCHxDSIZ register.

Register 31-45: DCHxDSIZSET: DCHxDSIZ Set Register

Write sets selected bits in DCHxDSIZ, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in DCHxDSIZ**

A write of '1' in one or more bit positions sets the corresponding bit(s) in DCHxDSIZ register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCHxDSIZSET = 0x000000ff` will set bits 7 through 0 in DCHxDSIZ register.

Register 31-46: DCHxDSIZINV: DCHxDSIZ Invert Register

Write inverts selected bits in DCHxDSIZ, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in DCHxDSIZ**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in DCHxDSIZ register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCHxDSIZINV = 0x000000ff` will invert bits 7 through 0 in DCHxDSIZ register.

Register 31-47: DCHxSPTR: DMA Channel x Source Pointer Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
CHSPTR<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7-0 **CHSPTR<7:0>:** Channel Source Pointer bits

255 = Points to 255th byte of the source

• • •

1 = Points to 1st byte of the source
0 = Points to 0th byte of the source

Note: This is reset on pattern detect, when in Pattern Detect mode.

Register 31-48: DCHxDPTR: Channel x Destination Pointer Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
CHDPTR<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7-0 **CHDPTR<7:0>:** Channel Destination Pointer bits

255 = Points to 255th byte of the destination

• • •

1 = Points to 1st byte of the destination
0 = Points to 0th byte of the destination

Register 31-49: DCHxCSIZ: DMA Channel x Cell-Size Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHCSIZ<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7-0 **CHCSIZ<7:0>:** Channel Cell-Size bits

255 = 255 bytes transferred on an event

• • •

2 = 2 bytes transferred on an event
1 = 1 byte transferred on an event
0 = 256 bytes transferred on an event

Register 31-50: DCHxCSIZCLR: DCHxCSIZ Clear Register

Write clears selected bits in DCHxCSIZ, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in DCHxCSIZ**

A write of '1' in one or more bit positions clears the corresponding bit(s) in DCHxCSIZ register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCHxCSIZCLR = 0x000000ff` will clear bits 7 through 0 in DCHxCSIZ register.

Register 31-51: DCHxCSIZSET: DCHxCSIZ Set Register

Write sets selected bits in DCHxCSIZ, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in DCHxCSIZ**

A write of '1' in one or more bit positions sets the corresponding bit(s) in DCHxCSIZ register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCHxCSIZSET = 0x000000ff` will set bits 7 through 0 in DCHxCSIZ register.

Register 31-52: DCHxCSIZINV: DCHxCSIZ Invert Register

Write inverts selected bits in DCHxCSIZ, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in DCHxCSIZ**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in DCHxCSIZ register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCHxCSIZINV = 0x000000ff` will invert bits 7 through 0 in DCHxCSIZ register.

Register 31-53: DCHxCPTR: DMA Channel x Cell Pointer Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
CHCPTR<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7-0 **CHCPTR<7:0>:** Channel Cell Progress Pointer bits

255 = 255 Bytes have been transferred since the last event

• • •

1 = 1 Bytes have been transferred since the last event
0 = 0 Bytes have been transferred since the last event

Note: This is reset on pattern detect, when in Pattern Detect mode

Register 31-54: DCHxDAT: DMA Channel x Pattern Data Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
CHPDAT<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7-0 **CHPDAT<7:0>:** Channel Data Register bits

Pattern Terminate mode:
Data to be matched must be stored in this register to allow terminate on match.

All other modes:
Unused.

Register 31-55: DCHxDATCLR: DCHxDAT Clear Register

Write clears selected bits in DCHxDAT, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in DCHxDAT**

A write of '1' in one or more bit positions clears the corresponding bit(s) in DCHxDAT register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCHxDATCLR = 0x00000020` will clear bit 5 in DCHxDAT register.

Register 31-56: DCHxDATSET: DCHxDAT Set Register

Write sets selected bits in DCHxDAT, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in DCHxDAT**

A write of '1' in one or more bit positions sets the corresponding bit(s) in DCHxDAT register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCHxDATSET = 0x00000020` will set bit 5 in DCHxDAT register.

Register 31-57: DCHxDATINV: DCHxDAT Invert Register

Write inverts selected bits in DCHxDAT, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in DCHxDAT**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in DCHxDAT register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `DCHxDATINV = 0x00000020` will invert bits 5 in DCHxDAT register.

Register 31-58: IFS1: Interrupt Flag Status Register 1(1)

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	USBIF	FCEIF
bit 31							bit 24

r-0	r-0	r-0	r-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	DMA3IF	DMA2IF	DMA1IF	DMA0IF
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RTCCIF	FSCMIF	I2C2MIF	I2C2SIF	I2C2BIF	U2TXIF	U2RXIF	U2EIF
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-0
SPI2RXIF	SPI2TXIF	SPI2EIF	CMP2IF	CMP1IF	PMPIF	AD1IF	CNIF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-26 **Unimplemented:** Read as ‘0’

bit 25-24 Interrupt Flags for Other Peripheral Devices

bit 23-20 **Reserved:** Write ‘0’; ignore read

bit 19-16 **DMAxIF:** DMA Channel x Interrupt Request Flag bits

1 = Interrupt request has occurred on channel x
0 = No interrupt request has a occurred on channel x

bit 15-0 Interrupt Flags for Other Peripheral Devices

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the DMA.

Register 31-59: IEC1: Interrupt Enable Control Register 1(1)

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	USBIE	FCEIE
bit 31							bit 24

r-0	r-0	r-0	r-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	DMA3IE	DMA2IE	DMA1IE	DMA0IE
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RTCCIE	FSCMIE	I2C2MIE	I2C2SIE	I2C2BIE	U2TXIE	U2RXIE	U2EIE
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-0
SPI2RXIE	SPI2TXIE	SPI2EIE	CMP2IE	CMP1IE	PMPIE	AD1IE	CNIE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-26 **Unimplemented:** Read as ‘0’

bit 25-24 Interrupt Flags for Other Peripheral Devices

bit 23-20 Reserved: Write ‘0’; ignore read

bit 19-16 **DMAxIE:** DMA Interrupt Enable bits for Channel x

1 = Interrupt is enabled for DMA channel x
0 = Interrupt is disabled for DMA channel x

bit 15-0 Interrupt Enable Bits for Other Peripheral Devices

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the DMA.

Register 31-60: IPC9: Interrupt Priority Control Register 9(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	DMA3IP<2:0>			DMA3IS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	DMA2IP<2:0>			DMA2IS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	DMA1IP<2:0>			DMA1IS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	DMA0IP<2:0>			DMA0IS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-29 **Reserved:** Write ‘0’; ignore read

bit 28-26 **DMA3IP<2:0>:** DMA Channel 3 Interrupt Priority

111 = Interrupt Priority is 7
110 = Interrupt Priority is 6
101 = Interrupt Priority is 5
100 = Interrupt Priority is 4
011 = Interrupt Priority is 3
010 = Interrupt Priority is 2
001 = Interrupt Priority is 1
000 = Interrupt is disabled

bit 25-24 **DMA3IS<1:0>:** DMA Channel 3 Subpriority

11 = Interrupt Subpriority is 3
10 = Interrupt Subpriority is 2
01 = Interrupt Subpriority is 1
00 = Interrupt Subpriority is 0

bit 23-21 **Unimplemented:** Read as ‘0’

bit 20-18 **DMA2IP<2:0>:** DMA Channel 2 Interrupt Priority

111 = Interrupt Priority is 7
110 = Interrupt Priority is 6
101 = Interrupt Priority is 5
100 = Interrupt Priority is 4
011 = Interrupt Priority is 3
010 = Interrupt Priority is 2
001 = Interrupt Priority is 1
000 = Interrupt is disabled

bit 17-16 **DMA2IS<1:0>:** DMA Channel 2 Subpriority

11 = Interrupt Subpriority is 3
10 = Interrupt Subpriority is 2
01 = Interrupt Subpriority is 1
00 = Interrupt Subpriority is 0

bit 15-13 **Reserved:** Write ‘0’; ignore read

Register 31-60: IPC9: Interrupt Priority Control Register 9(1) (Continued)

bit 12-10 **DMA1IP<2:0>:** DMA Channel 1 Interrupt Priority bits

111 = Interrupt Priority is 7
110 = Interrupt Priority is 6
101 = Interrupt Priority is 5
100 = Interrupt Priority is 4
011 = Interrupt Priority is 3
010 = Interrupt Priority is 2
001 = Interrupt Priority is 1
000 = Interrupt is disabled

bit 9-8 **DMA1IS<1:0>:** DMA Channel 1 Subpriority bits

11 = Interrupt Subpriority is 3
10 = Interrupt Subpriority is 2
01 = Interrupt Subpriority is 1
00 = Interrupt Subpriority is 0

bit 7-5 **Reserved:** Write '0'; ignore read

bit 4-2 **DMA0IP<2:0>:** DMA Channel 0 Interrupt Priority bits

111 = Interrupt Priority is 7
110 = Interrupt Priority is 6
101 = Interrupt Priority is 5
100 = Interrupt Priority is 4
011 = Interrupt Priority is 3
010 = Interrupt Priority is 2
001 = Interrupt Priority is 1
000 = Interrupt is disabled

bit 1-0 **DMA0IS<1:0>:** DMA Channel 0 Subpriority bits

11 = Interrupt Subpriority is 3
10 = Interrupt Subpriority is 2
01 = Interrupt Subpriority is 1
00 = Interrupt Subpriority is 0

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the DMA.

31.3 MODES OF OPERATION

The DMA module offers the following operating modes:

- Basic Transfer mode
- Pattern Match Termination mode
- Channel Chaining Mode
- Channel Auto-Enable mode
- CRC Calculation mode

Note that these operation modes are not mutually exclusive but can be simultaneously operational. For example, the DMA controller can perform CRC calculation using chained channels and terminating the transfer upon a pattern match.

31.3.1 DMA Controller Terminology

Event: Any system event that can initiate or abort a DMA transfer.

Transaction: A single word transfer (up to 4 bytes), comprised of read and write operations.

Cell Transfer: The number of bytes transferred when a DMA channel has a transfer initiated before waiting for another event (given by the DCHxCSIZ register). A cell transfer is comprised of one or more transactions.

Block Transfer: Defined as the number of bytes transferred when a channel is enabled. The number of bytes is the larger of either DCHxSSIZ or DCHxDSIZ. A block transfer is comprised of one or more cell transfers.

31.3.2 Basic Transfer Mode Operation

A DMA channel will transfer data from a source register to a destination register without CPU intervention. The Channel Source Start Address register (DCHxSSA) defines the physical start address of the source. The Channel Destination Start Address register (DCHxDSA) defines the physical start address of the destination. Both the source and destination are independently configurable using the DCHxSSIZ and DCHxDSIZ registers.

A cell transfer is initiated in one of two ways:

- Software can initiate a transfer by setting the channel CFORCE (DCHxECON<7>) bit.
- Interrupt event occurs on the device that matches the CHSIRQ interrupt and SIRQEN = 1 (DCHxECON<4>). The user can select any interrupt on the device to start a DMA transfer.

A DMA transfer will transfer DCHxCSIZ (cell transfer) bytes when a transfer is initiated (an event occurs). The channel remains enabled until the DMA channel has transferred the larger of DCHxSSIZ and DCHxDSIZ (i.e., block transfer is complete). If DCHxCSIZ is greater than the larger of DCHxSSIZ and DCHxDSIZ, then the larger of DCHxSSIZ and DCHxDSIZ bytes will be transferred. When the channel is disabled, further transfers will be prohibited until the channel is re-enabled (CHEN is set to ‘1’).

Each channel keeps track of the number of words transferred from the source and destination using the pointers DCHxSPTR and DCHxDPTR. Interrupts are generated when the source or Destination Pointer is half of the size (DCHxSSIZ/2 or DCHxDSIZ/2), or when the source or destination counter reaches the end. These interrupts are CHSHIF (DCHxINT<6>), CHDHIF (DCHxINT<4>), CHSDIF (DCHxINT<7>), or CHDDIF (DCHxINT<5>), respectively.

A DMA transfer request can be reset by the following:

- Writing the CABORT bit (DCHxECON<6>)
- Pattern match occurs if pattern match is enabled as described in **Section 31.3.4 "Pattern Match Termination Mode Operation"**, provided that Channel Auto-Enable mode bit CHAEN (DCHxCON<4>), is not set
- Interrupt event occurs on the device that matches the CHAIRQ <7:0> bits (DCHxE-CON<23:16>) interrupt if enabled by AIRQEN (DCHxECON<3>)
- Detection of an address error
- Completion of a cell transfer
- A block transfer completes provided that Channel Auto-Enable mode (CHAEN) is not set

When a channel abort interrupt occurs, the Channel Transfer Abort Interrupt Flag CHTAIF (DCHxINT<1>) bit is set. This allows the user to detect and recover from an aborted DMA transfer

When a transfer is aborted, any transaction currently underway will be completed.

The Source and Destination Pointers are updated as a transfer progresses. These pointers are read-only. The pointers are reset under the following conditions:

- If the channel source address (DCHxSSA) is updated, the Source Pointer (DCHxSPTR) will be reset.
- Similar updates to the destination address (DCHxDSA) will cause the Destination Pointer (DCHxDPTR) to be reset.
- A channel transfer is aborted by writing the CABORT (DCHxECON<6>) bit.

Note: Refer to the Table 31-5 for detailed information about the channel event behavior.

Example 31-1: DMA Channel Initialization for Basic Transfer Mode Code Example

```
/*
The following code example illustrates the DMA channel 0 configuration for a data transfer.
*/
       IEC1CLR=0x00010000;           // disable DMA channel 0 interrupts
       IFS1CLR=0x00010000;           // clear existing DMA channel 0 interrupt flag

       DMACONSET=0x00008000;         // enable the DMA controller
       DCH0CON=0x3;                  // channel off, pri 3, no chaining

       CH0ECON=0;                    // no start or stop irq's, no pattern match

                                     // program the transfer
       DCH0SSA=0x1d010000;           // transfer source physical address
       DCH0DSA=0x1d020000;           // transfer destination physical address
       DCH0SSIZ=0;                   // source size 256 bytes
       DCH0DSIZ=0;                   // destination size 256 bytes
       DCH0CSIZ=0;                   // 256 bytes transferred per event

       DCH0INTCLR=0x00ff00ff;        // clear existing events, disable all interrupts
       DCH0CONSET=0x80;              // turn channel on

                                     // initiate a transfer
       DCH0ECONSET=0x00000080;       // set CFORCE to 1

                                     // do something else

                                     // poll to see that the transfer was done

       while(TRUE)
       {
              register int pollCnt;  // use a poll counter.
                                     // polling continuously the DMA controller in a tight
                                     // loop would affect the performance of the DMA transfer

              int dmaFlags=DCH0INT;
              if( (dmaFlags&0xb)
              {                      // one of CHERIF (DCHxINT<0>), CHTAIF (DCHxINT<1>)
                                     // or CHBCIF (DCHxINT<3>) flags set
                     break;          // transfer completed
              }
              pollCnt=100;           // use an adjusted value here
              while(pollCnt--);      // wait before reading again the DMA controller
       }

                                     // check the transfer completion result
```

31.3.2.1 Interrupt and Pointer Updates

The Source and Destination Pointers are updated after every transaction. Interrupts will also be set or cleared at this time. If a pointer passes the halfway point during a transaction, the interrupt will be updated accordingly.

Pointers are reset when any of the following occurs:

- On any device Reset
- When the DMA is turned off (ON bit (DMACON<15>) is '0')
- A block transfer completes, regardless of the state of CHAEN (DCHxCON<4>)
- A pattern match terminates a transfer, regardless of the state of CHAEN (DCHxCON<4>)
- The CABORT (DCHxECON<6>) flag is written
- Source or destination start addresses are updated

31.3.3 Pattern Match Termination Mode Operation

Pattern Match Termination mode allows the user to end a transfer if a byte of data written during a transaction matches a specific pattern, as defined by the DCHxDAT register. A pattern match is treated the same way as a block transfer complete, where the CHBCIF bit (DCHxINT<3>) is set and the CHEN bit (DCHxCON<7>) is cleared.

This feature is useful in applications where a variable data size is required and eases the set up of the DMA channel. UART is a good example of where this can be effectively used.

Assuming a system has a series of messages that are routinely transmitted to an external host and it has a maximum message size of 86 characters, the user would set the following parameters on the channel:

- DCHxSSIZ to 87 bytes:

  If something unexpected occurs the CPU program will be interrupted when the buffer overflows and can take the appropriate action.
- DCHxDSIZ set to 1 byte.
- The destination address is set to the UART TXREG.
- The DCHxDAT is set to 0x00, which will stop the transfer on a NULL character in any byte lane.
- The CHSIRQ (DCHxECON<15:8>) is set to the UART “transmit buffer empty” IRQ.
- The SIRQEN (DCHxECON<4>) is set to enable the channel to respond to the start interrupt event.
- The start address is set to the start address of the message to be transferred.
- The channel is enabled, CHEN = 1 (DCHxCON<7>).
- The user will then force a cell transfer through CFORCE (DCHxECON<7>) and the first byte transmission by the UART.
- Each time a byte is transmitted by the UART, the transmit buffer empty interrupt will initiate the following byte transfer from the source to the UART.
- When the DMA channel detects a NULL character in any of the byte lanes of the channel, the transaction will be completed and the channel disabled.

Pattern matching is independent of the byte lane of the source data. If ANY byte in the source buffer matches DCHxDAT, a pattern match is detected. The transaction will be completed and the data read from the source will be written to the destination.

Example 31-2: DMA Channel Initialization in Pattern Match Transfer Mode Code Example

```
/*
The following code example illustrates the DMA channel 0 configuration for data transfer with
pattern match enabled. Transfer from the UART1 a <CR> ended string, at most 256 characters
long
*/

       IEC1CLR=0x00010000;           // disable DMA channel 0 interrupts
       IFS1CLR=0x00010000;           // clear any existing DMA channel 0 interrupt flag

       DMACONSET=0x00008000;         // enable the DMA controller
       DCH0CON=0x03;                 // channel off, priority 3, no chaining

       DCH0ECON=(27 <<8)| 0x30;      // start irq is UART1 RX, pattern match enabled
       DCH0DAT='\r';                 // pattern value, carriage return

                                     // program the transfer
       DCH0SSA=VirtToPhys(&U1RXREG); // transfer source physical address
       DCH0DSA=0x1d020000;           // transfer destination physical address
       DCH0SSIZ=1;                   // source size is 1 byte
       DCH0DSIZ=0;                   // dst size at most 256 bytes
       DCH0CSIZ=1;                   // one byte per UART transfer request

       DCH0INTCLR=0x00ff00ff;        // clear existing events, disable all interrupts
       DCH0INTSET=0x00090000;        // enable Block Complete and error interrupts

       IPC9CLR=0x0000001f;           // clear the DMA channel 0 priority and sub-priority
       IPC9SET=0x00000016;           // set IPL 5, sub-priority 2
       IEC1SET=0x00010000;           // enable DMA channel 0 interrupt

       DCH0CONSET=0x80;              // turn channel on


                                     // wait for the UART1 RX interrupt to initiate a
                                     // transfer

                                     // do something else

                                     // will get an interrupt when the transfer is done
                                     // or when an address error occurred
```


31.3.4 Channel Chaining Mode Operation

Channel chaining is an enhancement to the DMA channel operation. A channel (slave channel) can be chained to an adjacent channel (master channel). The slave channel will be enabled when a block transfer of the master channel completes, i.e., CHBCIF (DCHxINT<3>), is set.

At this point, any event on the slave channel will initiate a cell transfer. If the channel has an event pending, a cell transfer will begin immediately.

The master channel will set its interrupt flags normally, CHBCIF (CHxINT<3>) and has no knowledge of the “chain” status of the slave channel. The master channel is still able to cause interrupts at the end of a DMA transfer if one of the CHSDIE/CHDDIE/CHBCIE (DCHxINT<23,21,19>) bits is set.

In the channels natural priority order, channel 0 has the highest priority and channel 4 the lowest. The channel higher or lower in natural priority, that can enable a specific channel, is selected by CHCHNS (DCHxCON<8>), provided that channel chaining is enabled, CHCHN = 1 (DCHxCON<5>).

A feature of the DMA module is the ability to allow events while the channel is disabled using CHAED (DCHxCON<6>). This bit is particularly useful in Chained mode, in which the slave channel needs to be ready to start a transfer as soon as the channel is enabled by the master channel.

The following examples demonstrate situations in which chaining may be useful:

1. Transferring data in one peripheral (e.g., from UART1, DMA channel 0, at 9600 baud, to SRAM) to another peripheral (e.g., from SRAM to UART2, DMA channel 1, at 19200 baud).

   In this example, CHAED will be set in both channels; with UART2 setting the event detect, CHEDET (DCHxCON<2>), on channel 1 when the last byte has been transmitted. As soon as channel 0 completes a transfer, channel 1 is enabled and the data is transferred immediately.

2. A/D converter transfers data to one buffer (connected to channel 0).

   When the destination buffer 0 is full (block transfer completes), channel 1 is enabled and further conversions are transferred to buffer 1. In this case, CHAED will not be enabled. If it were, the last word transferred by channel 0 would be transferred a second time by channel 1 (because the A/D converter interrupt event would have set the event detect flag CHEDET in both channels).

Example 31-3: DMA Channel Initialization in Chaining Mode Code Example

```
/*
The following code example illustrates the DMA channel 0 configuration for data transfer with
pattern match enabled. DMA channel 0 transfer from the UART1 to a RAM buffer while DMA channel 1
transfers data from the RAM buffer to UART2. Transferred strings are at most 256 characters
long. Transfer on UART2 will start as soon as the UART1 transfer is completed.
*/

  unsigned char myBuff<256>;// transfer buffer

 IEC1CLR=0x00010000;        // disable DMA channel 0 interrupts
 IFS1CLR=0x00010000;        // clear any existing DMA channel 0 interrupt flag

 DMACONSET=0x00008000;      // enable the DMA controller

 DCH0CON=0x3;               // channel 0 off, priority 3, no chaining
 DCH1CON=0x62;              // channel 1 off, priority 2
                            // chain to higher priority
                            // (ch      0), enable events detection while disabled

 DCH0ECON=(27 <<8)| 0x30;   // start irq is UART1 RX, pattern enabled
 DCH1ECON=(42 <<8)| 0x30;   // start irq is UART1 TX, pattern enabled

 DCH0DAT=DCH1DAT='\r';      // pattern value, carriage return

                            // program channel 0 transfer
 DCH0SSA=VirtToPhys(&U1RXREG); // transfer source physical address
 DCH0DSA=VirtToPhys(myBuff); // transfer destination physical address
 DCH0SSIZ=1;                // source size is 1 byte
 DCH0DSIZ=0;                // dst size at most 256 bytes
 DCH0CSIZ=1;                // one byte per UART transfer request

                            // program channel 1 transfer
 DCH1SSA=VirtToPhys(myBuff); // transfer source physical address
 DCH1DSA=VirtToPhys(&U2TXREG); // transfer destination physical address
 DCH1SSIZ=0;                // source size at most 256 bytes
 DCH1DSIZ=0;                // dst size is 1 byte
 DCH1CSIZ=1;                // one byte per UART transfer request

 DCH0INTCLR=0x00ff00ff;     // DMA0: clear events, disable interrupts
 DCH1INTCLR=0x00ff00ff;     // DMA1: clear events, disable interrupts
 DCH1INTSET=0x00090000;     // DMA1: enable Block Complete and error interrupts


 IPC9CLR=0x00001f1f;        // clear the DMA channels 0 and 1 priority and
                            // sub-priority
 IPC9SET=0x00000b16;        // set IPL 5, sub-priority 2 for DMA channel 0
                            // set IPL 2, sub-priority 3 for DMA channel 1
 IEC1SET=0x00020000;        // enable DMA channel 1 interrupt

 DCH0CONSET=0x80;           // turn channel on

                            // do something else

                            // the UART1 RX interrupts will initiate the DMA channel 0 transfer
                            // once this transfer is complete, the DMA channel 1 will start
                            // upon DMA channel 1 transfer completion will get an interrupt

 while(!intCh1Ocurred);     // poll DMA channel 1 interrupt
```


31.3.5 Channel Auto-Enable Mode Operation

The channel auto-enable can be used to keep a channel active, even if a block transfer completes or pattern match occurs. This prevents the user from having to re-enable the channel each time a block transfer completes. To use this mode the user will configure the channel, setting the CHAEN (DCHxCON<4>) bit before enabling the channel, i.e., setting the CHEN bit (DCHxCON<7>). The channel will behave as normal except that normal termination of a transfer will not result in the channel being disabled.

Normal block transfer completion is defined as:

- Block Transfer Complete
- Pattern Match Detect

As before, the Channel Pointers will be reset. This mode is useful for applications that do repeated pattern matching.

Note:	CHAEN prevents the channel from being automatically disabled once it has been enabled. The channel will still have to be enabled by the software.

31.3.6 Suspending Transfers

The user can immediately suspend the DMA module by writing the SUSPEND bit (DMACON<12>). This will immediately suspend the DMA controller from any further bus transactions. Individual channels may be suspended using the CHEN bit. If a DMA transfer is in progress and the CHEN bit is cleared, the current transaction will be completed and further transactions on the channel will be suspended. Clearing the enable bit (CHEN) will not affect the Channel Pointers or the transaction counters. While a channel is suspended, the user can elect to continue to receive events (abort interrupts, etc.) by setting CHAED (DCHxCON<6>).

31.3.7 Resetting the Channel

The channel logic will be reset on any device Reset. The channel is also reset when the channel flag bit CABORT (DCHxECON<6>) is written. This will turn off channel flag bit CHEN = 0, clear the Source and Destination Pointers, and reset the event detector. When the CABORT bit is set, the current transaction in progress (if any) will complete before the channel is reset, but any remaining transactions will be aborted.

The user should modify the channel registers only while the channel is disabled (CHEN = 0). Modifying the Source and Destination registers will reset the corresponding pointer registers (DCHxSPTR or DCHxDPTR).

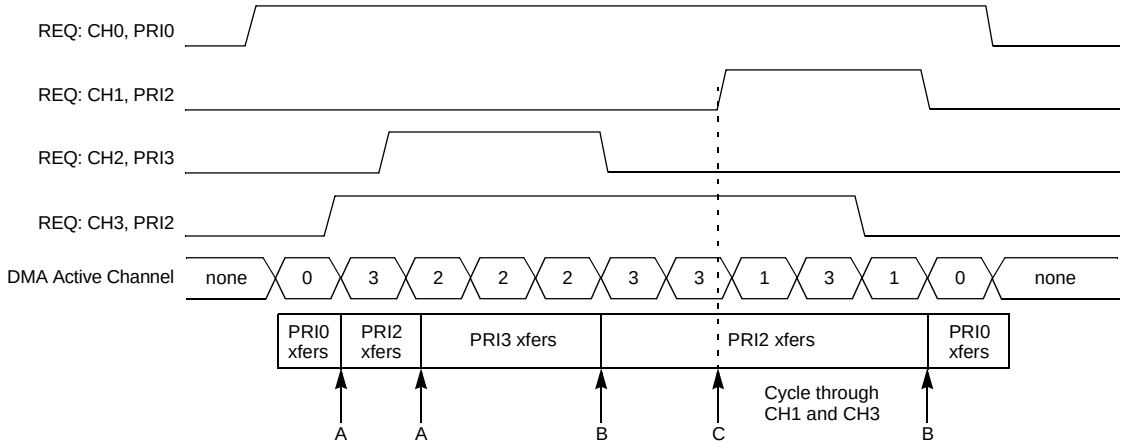
Note:	The channel size must be changed while the channel is disabled.

31.3.8 Channel Priority and Selection

The DMA controller has a natural priority associated with each of the channels. Channel 0 has the highest natural priority. Each channel has two priority bits, CHPRI<1:0> (DCHxCON<1:0>). These bits identify the channel's priority. When multiple channels have transfers pending, the next channel to transmit data will be selected as follows:

- Channels with the highest priority will complete all cell transfers before moving onto channels with a lower priority (see behavior, "PRI3 xfers", in Figure 31-4).
- If multiple channels have the same priority (identical CHPRI), the controller will cycle through all channels at that priority. Each channel with a cell transfer in progress at the highest priority will be allowed a single transaction of the active cell transfer before the controller allows a single transaction by the next channel at that priority level (see behavior, "PRI2 xsfers" between markers "C" and "B", in Figure 31-4).
- If a channel with a higher priority requests a transfer while another channel of lower priority has a transaction in process, the transaction will complete before moving to the channel with the higher priority (see events at markers "A" in Figure 31-4).

Figure 31-4: Channel Priority Behavior



Transition Legend:

A – Higher priority transfer request; suspend current and transfer next.

B – All highest priority transfers complete; drop to channels at lower priority.

C – Cycle through all channels at the current priority.

31.3.9 Byte Alignment

The byte alignment feature of the DMA controller relieves the user from aligning the source and destination addresses.The read portion of a transaction will read the maximum number of bytes that are available to be read in a given word. For example, if the Source Pointer is N > 4 bytes from the source size, 4 bytes will be read if the Source Pointer points to byte 0, 3 bytes if the Source Pointer points to byte 1, etc. If the number of bytes remaining in the source is N < 4, only the first N bytes are read. This is important when the read includes registers that are updated on a read.

The Source Pointer and Destination Pointers are updated after every write, with the number of bytes that have been written. The user should note that in cases where a transfer is aborted, before a transaction is complete, the Source Pointer will not necessarily reflect the reads that have taken place.

An example of this behavior is given in Table 31-3. Example 1 demonstrates a simple transfer of 9 bytes between two large buffers, in which CHxSSA = 0x1000, CHxSSIZ = 100, CHxDSA = 0x43F9, CHxDSIZ = 100, and CHxXSIZ = 9.

Table 31-3: Source and Destination Pointer Updates – Example 1

Transaction	Operation	Source Pointer	Destination Pointer	Transfer Count/Size	Read Address	Write Address	Read Data(1)	Write Data(2)
1	Read	9	11	0/9	1009	xxxx	33_22_11_XX	XX_XX_XX_XX
1	Write1	9	11	0/9	1009	440A	33_22_11_XX	22_11_XX_XX
1	Ptr Update(3)	B	13	2/9	1009	440A	33_22_11_XX	XX_XX_XX_XX
1	Write2	B	13	2/9	1009	440C	33_22_11_XX	XX_XX_XX_33
1	Ptr Update(3)	C	14	3/9	1009	440C	33_22_11_XX	XX_XX_XX_XX
2	Read	C	14	3/9	100C	440C	77_66_55_44	XX_XX_XX_XX
2	Write1	C	14	3/9	100C	440D	77_66_55_44	66_55_44_XX
2	Ptr Update(3)	F	17	6/9	100C	440D	77_66_55_44	XX_XX_XX_XX
2	Write2	F	17	6/9	100C	4410	77_66_55_44	XX_XX_XX_77
2	Ptr Update(3)	10	18	7/9	100C	4410	77_66_55_44	XX_XX_XX_XX
3	Read	10	18	7/9	1010	4410	XX_XX_99_88	XX_XX_XX_XX
3	Write1	10	18	7/9	1010	4411	XX_XX_XX_88	XX_99_88_XX
3	Ptr Update(3)	12	1A	9/9	1010	4411	XX_XX_XX_88	XX_XX_XX_XX

Note 1: XX indicates that data read is discarded.

2: XX indicates that data that is NOT written.

3: Interrupts are updated when the pointers are updated as required.

Another example of this behavior is given in Table 31-4. Example 2 demonstrates worst-case bus utilization, i.e., unaligned buffers with destination buffer wrapping, in which CHxSSA = 0x1000, CHxSSIZ = 100, CHxDSA = 0x4402, CHxDSIZ = 4, and CHxXSIZ = 8.

Table 31-4: Source and Destination Pointer Updates – Example 2

Transaction	Operation	Source Pointer	Destination Pointer	Transfer Count/Size	Read Address	Write Address	Read Data(1)	Write Data(2)
1	Read	9	0	0/8	1009	xxxx	33_22_11_XX	XX_XX_XX_XX
1	Write1	9	0	0/8	1009	4402	33_22_11_XX	22_11_XX_XX
1	Ptr Update(3)	B	2	2/8	1009	4402	33_22_11_XX	XX_XX_XX_XX
1	Write2	B	2	2/8	1009	4404	33_22_11_XX	XX_XX_XX_33
1	Ptr Update(3)	C	3	3/8	1009	4404	33_22_11_XX	XX_XX_XX_XX
2	Read	C	3	3/8	100C	4404	77_66_55_44	XX_XX_XX_XX
2	Write1	C	3	3/8	100C	4405	77_66_55_44	XX_XX_44_XX
2	Ptr Update(3)	D	0	4/8	100C	4405	77_66_55_44	XX_XX_XX_XX
2	Write2	D	0	4/8	100C	4402	77_66_55_44	66_55_XX_XX
2	Ptr Update(3)	F	2	6/8	100C	4402	77_66_55_44	XX_XX_XX_XX
3	Write3	F	2	6/8	100F	4404	77_66_55_44	XX_XX_XX_77
3	Ptr Update(3)	10	3	7/8	100F	4404	77_66_55_44	XX_XX_XX_XX
3	Read	10	18	7/8	1010	4404	BB_AA_99_88	XX_XX_XX_XX
3	Write1	10	18	7/8	1010	4405	BB_AA_99_88	XX_XX_88_XX
3	Ptr Update(3)	11	1A	8/8	1010	4405	77_66_55_44	XX_XX_XX_XX

Note 1: XX indicates that data read is discarded.

2: XX indicates that data that is NOT written.

3: Interrupts are updated when the pointers are updated as required.

31.3.10 Channel Transfer Behavior

Once a channel has been enabled, CHEN = 1 (DCHxCON<7>), any event that starts a cell transfer will transfer the CHCSIZ<7:0> (DCHxCSIZ<7:0>) bytes of data. This will require one or more transactions. Once the cell transfer is complete the channel will return to an inactive state, and will wait for another channel start event to occur before starting another cell transfer.

When the larger of CHSSIZ<7:0> (DCHxSSIZ<7:0>) or CHDSIZ<7:0> (DCHxDSIZ<7:0>) bytes are transferred, a block transfer completes, the channel transfer will be halted and the channel will be disabled (i.e., CHEN set to '0' by hardware, and pointers are reset).

31.3.11 Channel Enable

Each channel has an enable bit CHEN, which can be used to enable or disable the channel in question. When this bit is set, the channel transfer requests are serviced by the DMA controller.

When CHEN is clear, the state of the channel is preserved (this allows the channel to be suspended once a transfer has begun).

CHEN will be cleared by hardware under the following conditions:

- A block transfer is complete, the pointer to the larger of the source or destination matches the size (only if CHAEN (DCHxCON<4>) is clear).
- A pattern match occurs in Pattern Match mode (only if CHAEN is clear).
- An abort interrupt occurs.
- The user writes the CABORT (DCHxECON<6>) flag.

31.3.12 Channel IRQ Detection

The DMA Controller maintains its own flags for detecting the start and abort IRQ in the system and is completely independent of the INT Controller and IES/IFS flags. The corresponding IRQ does not have to be enabled before a transfer can take place, nor cleared at the end of a DMA transfer.

Once the start or abort IRQ system events are triggered, they will be detected automatically by the DMA controller internal logic, without the need for user intervention.

31.3.13 Channel Event Transfer Initiation

A given channel transfer can be initiated by:

- Writing the CFORCE bit (DCHxECON<7>).
- An interrupt occurs that matches the value of CHSIRQ<7:0> (DCHxECON<15:8>) if it is enabled by SIRQEN (DCHxECON<4>).

Channel events are registered if the channel is enabled (CHEN = 1), or if "Allow Event If Disabled" is set, i.e., CHAED = 1 (DCHxCON<6>).

31.3.14 Channel Event Transfer Termination

Channel transfer is terminated in any of the following cases:

- A transfer is aborted as described in **Section 31.3.17 "Channel Abort"**.
- A cell transfer (CHCSIZ<7:0> bytes (DCHxCSIZ<7:0> transferred) completes.
- The DMA has transferred the larger of CHSSIZ<7:0> or CHDSIZ bytes (block transfer complete), the channel is disabled in hardware and must be re-enabled by user software before the channel will respond to channel events.
- A pattern match occurs if enabled.
- An abort interrupt, CHAIRQ<7:0> (DCHxECON<23:16>), occurs if abort interrupts are enabled by AIRQEN (DCHxECON<3>).
- An address error occurs.

An example of how to use the abort interrupt would be a transfer from a UART channel to the memory. While the UART Receive Data Available interrupt can be used to start the transfer, the UART Error interrupt can abort the transfer. This way, whenever an error occurs on the communication channel (a framing/parity error or even an overrun), the transfer is stopped and the user code gets control in an ISR (if the abort interrupt is enabled for the DMA controller).

A summary of the status flags affected by channel transfer initiation or termination is provided in Table 31-5. Channel abort events are allowed if the channel is enabled, CHEN = 1, or if the user elects to allow events while the channel is disabled, CHAED = 1.

Table 31-5: Channel Event Behavior

Event	Description and Function	Registers Affected
Events Initiating Transfers		
System Interrupt Matching CHSIRQ<7:0>(1,2)	The channel event detect will be set.	CHEDET = 1
Channel Chain Event	This will enable the channel if not already set. If an event detect is pending, a channel transfer will begin immediately.	CHEN = 1
User Writes the CFORCE Bit(1)	The channel event detect will be set.	CHEDET = 1
Events Terminating Transfers		
System Interrupt Matching CHAIRQ<7:0>(1,2)	The channel event detect will be reset and the channel turned off. The abort interrupt flag is set.	CHEDET = 0 CHEN = 0 CHAIF = 1
Pattern Match(1)	This occurs when any byte of data written in a transaction matches the data in CHPDAT. The channel event detect is reset. The channel is turned off if CHAEN = 0. This event is treated as a completed block transfer. Pointers are reset.	CHEDET = 0 CHEN = 0 CHBCIF = 1 CHSPTR = 0 CHDPTR = 0 CHCPTR = 0
Cell Transfer is Complete	This occurs when CHCSIZ<7:0> bytes have been transferred. The transfer event detect is reset and the channel remains enabled pending the next event.	CHEDET = 0 CHCCIF = 1
Block Transfer is Complete	The channel event detect is reset. The channel is turned off if CHAEN = 0. This event is treated as a completed transfer. Pointers are reset.	CHEDET = 0 CHEN = 0 CHBCIF = 1 CHSPTR = 0 CHDPTR = 0 CHCPTR = 0
User Writes the CABORT bit	The channel is turned off and the channel event detect is reset. The pointers are reset.	CHEDET = 0 CHEN = 0 CHSPTR = 0 CHDPTR = 0 CHCPTR = 0
Address Error is Detected	The channel is turned off and the event detect is reset. The address error interrupt flag is set.	CHEDET = 0 CHEN = 0 CHERIF = 1

Note 1: Events are allowed only when the channel is enabled, or the user allows events while disabled (CHEN = 1 or CHAED = 1).

2: The DMA Controller maintains its own flags for detecting start and abort IRQs in the system, and is completely independent of the INT Controller IES/IFS flags. Once the start or abort IRQ system events are triggered, they will be detected automatically by the DMA controller internal logic, without the need for user intervention.

31.3.15 Channel Abort Interrupt

A channel can elect to abort a cell transfer if an interrupt event occurs. The interrupt is selected by the channel's abort IRQ, CHAIRQ<7:0> (DCHxECON<23:16>). Any one of the device interrupt events can cause a channel abort. An abort only occurs if enabled by AIRQEN (DCHxECON<3>).

If this occurs (often a timer time-out or a module error flag), the channel's status flags will indicate the external abort event on the channel in question by setting its CHTAIF bit (DCHxINT<1>). The Source and Destination Pointers are not reset, allowing the user to recover from the error.

31.3.16 Channel Abort

A channel transfer can be aborted by the user by writing the CABORT bit (DCHxECON<6>). When a transfer is aborted, the current bus transaction will be completed and any transactions that remain will be aborted. The CHEN (DCHxCON<7>) bit will be cleared. When the user writes the CABORT bit the Source and Destination Pointers are reset.

31.3.17 Address Error

If the address (either source or destination) occurring during a transfer is an illegal address, the channel's address error interrupt flag CHERIF (DCHxINT<0>) will be set. The channel will be disabled, i.e., CHEN will be reset by hardware.

The channel status is unaffected to aid in the debug of the problem.

31.3.18 DMA Suspend

DMA transactions are suspended immediately if the SUSPEND bit (DMACON<12>) is set. The current read or write will be completed. If the suspend comes during the read portion of the transaction, the transaction will be suspended and the write will be put on hold. If the suspend comes during the write portion of the transaction, the write will complete and the pointers updated as normal. Any transactions that were in process will continue where they left off when the SUSPEND bit is cleared.

Example 31-4: DMA Controller Suspension

```
/*
The following code example will suspend the DMA Controller.
*/
 DMACONSET=0x00001000;       // suspend the DMA controller

 while(!(DMACON&0x1000));    // wait for the transfer to be actually suspended

                             // let the CPU have complete control of the bus

 DMACONCLR=0x00001000;       // clear the suspend mode and let the DMA operate normally

                             // from now on, the CPU and DMA controller share the bus access
```

31.3.19 CRC Calculation Mode Operation

The DMA module has one integrated CRC generation module shared by all channels. The CRC module is a highly configurable, 16-bit CRC generator. The CRC module can be assigned to any available DMA channel by setting the CRCCH bits (DCRCCON<1:0>) appropriately. The CRC is enabled by setting the CRCEN bit (DCRCCON<7>).

The CRC generator will take 1 system clock to process each byte of data read from the source. This implies that if 32-bits of data are read from the source, the CRC generation will take 4 system clocks to process the data.

The implementation of the CRC module is software configurable. The terms of the polynomial and its length can be programmed using the DCRCXOR<15:1> bits and the PLEN <3:0> (DCRCCON<11:8>) bits, respectively.

For example, consider the CRC polynomial:

$x^{16} + x^{12} + x^5 + 1$

To program this polynomial into the CRC generator, the CRC register bits should be set as shown in the following table:

Table 31-6: Example CRC Setup

Bit Name	Bit Value
PLEN<3:0>	'b1111'
DCRCXOR<15:1>	'b0001 0000 0010 000'

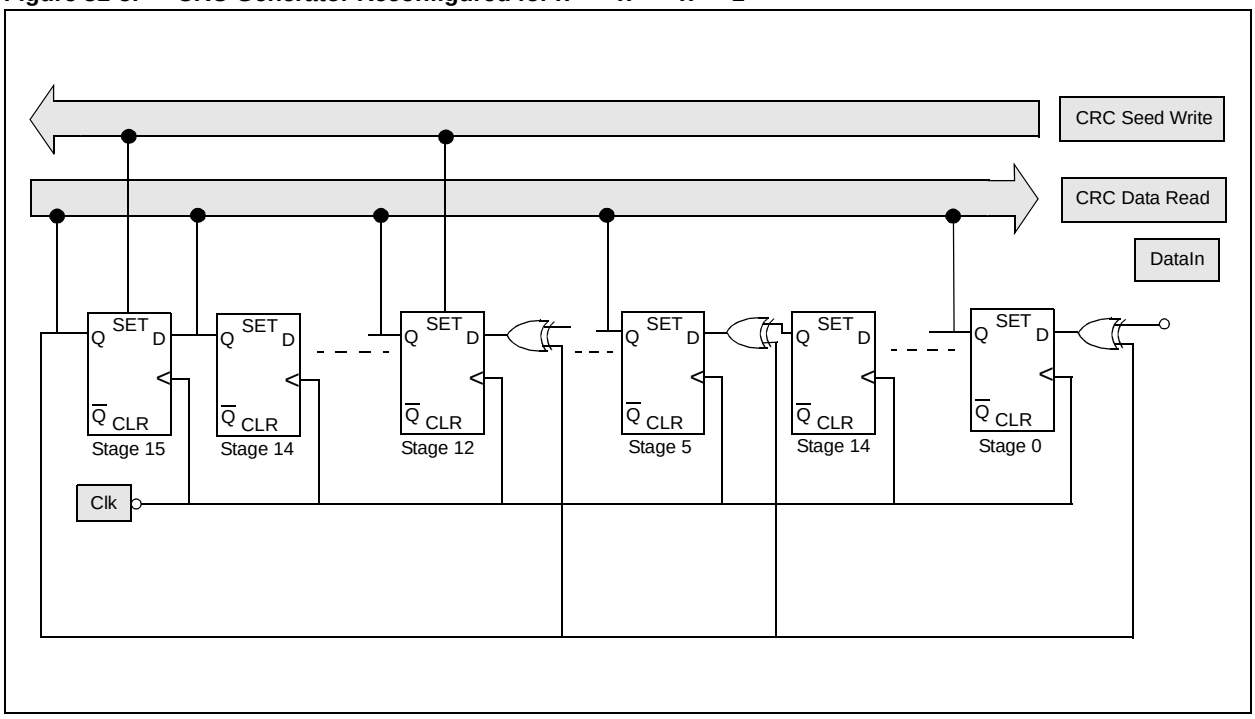
In Table 31-6, note that for the value of DCRCXOR<15:1>, the 12^{th} bit and the 5^{th} bit are set to '1', as required by the equation. The 0 bit required by the equation is always XOR'd. For a 16-bit polynomial, the 16^{th} bit is also always assumed to be XORed; therefore, the DCRCXOR<15:1> bits do not have the 0 bit or the 16^{th} bit.

The circuit shown in Figure 31-5 is the topology of a standard CRC generator.

The CRC module modifies the behavior of the DMA channel associated with the CRC module. The behavior of the channel is selected by the CRCAPP bit (DCRCCON<6>). The two modes are:

- Background Mode: CRC is calculated in the background, with normal DMA behavior maintained.
- Append Mode: Data read from the source is not written to the destination, but the CRC data is accumulated in the CRC data register. The accumulated CRC is written to the location given by DCHxDSA when a block transfer completes.

Figure 31-5: CRC Generator Reconfigured for $x^{16} + x^{12} + x^5 + 1$



31.3.19.1 Seeding the CRC Generator

The CRC generator can be seeded by writing to the DCRCDATA register before enabling the channel that will use the CRC module.

31.3.19.2 Reading the CRC

The CRC can be read as it progresses by reading the DCRCDATA register at any time during the CRC generation.

31.3.19.3 CRC Polynomial Length

The PLEN<3:0> bits (DCRCCON<11:8>) in the CRC generator are used to select which bit is used as the feedback point of the CRC. For example, if PLEN<3:0> = `0x0110`, then bit 6 (PLEN + 1 n) of the Shift register be fed into the XOR gates of all bits set in the CRCXOR register.

31.3.19.4 CRC Feedback Points

The CRC XOR feedback points are specified using the DCRCXOR register. Setting the Nth bit in the DCRCXOR register will enable the input to the Nth bit of the CRC Shift register to be XOR'd with the (PLEN + 1)th bit of the CRC Shift register. Bit 0 of the CRC generator is always XOR'd.

31.3.19.5 Data Order

As data is read from the Source register, the data is fed into the CRC generator, MSb, first.

31.3.19.6 CRC Background Mode (CRCAPP = 0)

In CRC Background mode, the behavior of the DMA channel is maintained when data read from the channel source passed to the CRC module as it is written back to the destination. In this mode, the calculated CRC is left in the DCRCDATA register at the end of the block transfer.

This mode can be used to calculate a CRC as data is moved from source to destination. For example, CRC Background mode can be used is to calculate a CRC as data is transmitted to, or received from, the UART module. When the data transfer is complete, the user can read the calculated CRC and either append it to the transmitted data or verify the received CRC data.

The following usage notes apply to CRC Background mode:

- This mode is very useful for calculating a CRC-on-the-fly.

  Potentially ties up CRC module for extended periods of time. For example, the source device (UART, etc.) will interrupt relatively infrequently, tying CRC module up for an extended period of time when receiving a lengthy data string.
- This mode is also useful for implementations that have high bus usages.

  The CRC is calculated as data is transferred, eliminating the need for a separate bus access to transfer the data to the CRC module.

Example 31-5: DMA CRC Calculation in Background Mode Code Example

```
/*
The following code example illustrates a DMA calculation using the CRC background mode. Data is
transferred from a 200 bytes Flash buffer to a RAM buffer and the CRC is calculated while the
transfer takes place. */

 unsigned int blockCrc;           // CRC of the flash block

 IEC1CLR=0x00010000;              // disable DMA channel 0 interrupts
 IFS1CLR=0x00010000;              // clear any existing DMA channel 0 interrupt flag


 DMACONSET=0x00008000;            // enable the DMA controller

 DCRCDATA=0xffff;                 // seed the CRC generator
 DCRCXOR=0x1021;                  // Use the standard CCITT CRC 16 polynomial: X^16+X^12+X^5+1
 DCRCCON=0x0f80;                  // CRC enabled, polynomial length 16, background mode
                                  // CRC attached to the DMA channel 0.


 DCH0CON=0x03;                    // channel off, priority 3, no chaining
 DCH0ECON=0;                      // no start irqs, no match enabled

                                  // program channel transfer
 DCH0SSA=VirtToPhys(flashBuff);   // transfer source physical address
 DCH0DSA=VirtToPhys(ramBuff);     // transfer destination physical address
 DCH0SSIZ=200;                    // source size
 DCH0DSIZ=200;                    // dst size
 DCHOCSIZ=200;                    // 200 bytes per event

 DCH0INTCLR=0x00ff00ff;           // DMA0: clear events, disable interrupts

 DCH0CONSET=0x80;                 // channel 0 on

                                  // initiate a transfer
 DCH0ECONSET=0x00000080;          // set CFORCE to 1

                                  // do something else while the transfer takes place

                                  // poll to see that the transfer was done
 BOOL error=FALSE;
 while(TRUE)
 {
       register int pollCnt;      // don't poll in a tight loop
       int dmaFlags=DCH0INT;
       if( (dmaFlags& 0x3)
       {                          // CHERIF (DCHxINT<0>) or CHTAIF (DCHxINT<1> set
              error=TRUE;         // error or aborted...
              break;
              }
                 else if (dmaFlags&0x8)
              {                   // CHBCIF (DCHxINT<3>) set
              break;              // transfer completed normally
       }
       pollCnt=100;               // use an adjusted value here
       while(pollCnt--);          // wait before polling again
 }

 if(!error)
 {
       blockCrc=DCRDATA;          // read the CRC of the transferred flash block
 }
 else
 {
                                  // process error
 }
```

31.3.19.7 CRC Append Mode (CRCAPP = 1)

When a channel is enabled as a CRC enabled channel, all data read from the source will be fed into the CRC generation module. No data is written to the destination address in CRC Append mode until a block transfer completes or a pattern match occurs. On completion, the CRC value will be written to the address given by the destination register (DCHxDSA).

This mode is best used when multiple peripherals are required to use the CRC generator. In this case, the input data is accumulated in a buffer on the device. Once the buffer is complete, the CRC is generated and can be used appropriately.

Additional Usage Notes:

- Only the source is used when considering whether a block transfer is complete. The destination address (DCHxDSA) is only used as the location to which the generated CRC can be written.
- The destination size (DCHxDSIZ) can have a maximum size of 4.
  - If DCHxDSIZ is greater than 4, only 4 bytes are written and the channel is disabled.
  - If DCHxDSIZ is less than 4, only DCHxDSIZ bytes of the CRC are written to the destination address.
  - The high bytes (bits <31:16>) are written as zeros, if more than 16 bits of the CRC are written.
  - PLEN<3:0> (DCRCCON<11:8>) bits have no effect on the number of CRC bits that will be written to the destination register.
- All other Transfer Abort modes will NOT cause a write back of the calculated CRC.
  - No CRC written back on an abort IRQ, user abort, bus error, etc.

Example 31-6: CRC Calculation in Append Mode Code Example

```
/*
The following code example illustrates a DMA calculation using the CRC append mode. The CRC of
a 256 bytes flash buffer is calculated without performing any data transfer. As soon as the CRC
calculation is completed the CRC value of the flash buffer is available in a local variable for
further use. */

 unsigned int blockCrc;           // CRC of the flash block

 IEC1CLR=0x00010000;              // disable DMA channel 0 interrupts
 IFS1CLR=0x00010000;              // clear any existing DMA channel 0 interrupt flag


 DMACONSET=0x00008000;            // enable the DMA controller

 DCRCDATA=0xffff;                 // seed the CRC generator
 DCRCXOR=0x1021;                  // Use the standard CCITT CRC 16 polynomial: X^16+X^12+X^5+1
 DCRCCON=0x0fc0;                  // CRC enabled, polynomial length 16, append mode
                                  // CRC attached to the DMA channel 0.


 DCH0CON=0x03;                    // channel off, priority 3, no chaining
 DCH0ECON=0;                      // no start irqs, no match enabled

                                  // program channel transfer
 DCH0SSA=VirtToPhys(flashBuff);   // transfer source physical address
 DCH0DSA=VirtToPhys(&blockCrc);   // transfer destination physical address
 DCH0SSIZ=0;                      // source size
 DCH0DSIZ=0;                      // dst size
 DCHOCSIZ=0;                      // 256 bytes transferred per event

 DCH0INTCLR=0x00ff00ff;           // DMA0: clear events, disable interrupts
 DCH1INTCLR=0x00ff00ff;           // DMA1: clear events, disable interrupts

 DCH0CONSET=0x80;                 // channel 0 on

                                  // initiate a transfer
 DCH0ECONSET=0x00000080;          // set CFORCE to 1

                                  // do something else while the CRC calculation takes place

                                  // poll to see that the transfer was done
 BOOL error=FALSE;
 while(TRUE)
 {
       register int pollCnt;      // don't poll in a tight loop
       int dmaFlags=DCH0INT;
       if( (dmaFlags& 0x3)
       {                          // CHERIF (DCHxINT<0>) or CHTAIF (DCHxINT<1> set
              error=TRUE;         // error or aborted...
              break;
              }
                     else if (dmaFlags&0x8)
              {                   // CHBCIF (DCHxINT<3>) set
                     break;       // transfer completed normally
       }
       pollCnt=100;               // use an adjusted value here
       while(pollCnt--);          // wait before polling again
 }

 if(error)
 {
                                  // process error
 }


                                  // the block CRC is available in the blockCrc variable
```

31.4 INTERRUPTS

The DMA device has the ability to generate interrupts reflecting the events that occur during the channel's data transfer:

- Error interrupts, signalled by each channel's CHERIF bit (DCHxINT<0>) and enabled using the CHERIE bit (DCHxINT<16>). This event occurs when there is an address error occurred during the channel transfer operation.
- Abort interrupts, signalled by each channel's CHTAIF bit (DCHxINT<1>) and enabled using the CHTAIE bit (DCHxINT<17>). This event occurs when a DMA channel transfer gets aborted because of a system event (interrupt) matching the CHAIRQ<7:0> (DCHxECON<23:16>) when the abort interrupt request is enabled, AIRQEN = 1 (DCHxECON<3>).
- Block complete interrupts, signalled by each channel's CHBCIF bit (DCHxINT<3>) and enabled using the CHBCIE bit (DCHxINT<19>). This event occurs when a DMA channel block transfer is completed.
- Cell complete interrupts, signalled by each channel's CHCCIF bit (DCHxINT<2>) and enabled using the CHCCIE bit (DCHxINT<18>). This event occurs when a DMA channel cell transfer is completed.
- Source Address Pointer activity interrupts: either when the Channel Source Pointer reached the end of the source, signalled by the CHSDIF bit (DCHxINT<7>) and enabled by CHSDIE bit (DCHxINT<23>), or when the Channel Source Pointer reached midpoint of the source, signalled by the CHSHIF bit (DCHxINT<6>) and enabled by the CHSHIE bit (DCHxINT<22>).
- Destination Address Pointer activity interrupts: either when the Channel Destination Pointer reached the end of the destination, signalled by the CHDDIF bit (DCHxINT<5>) and enabled by the CHDDIE bit (DCHxINT<21>), or when the Channel Destination Pointer reached midpoint of the destination, signalled by the CHDHIF bit (DCHxINT<4>) and enabled by the CHDHIE bit (DCHxINT<20>).

All the interrupts belonging to a DMA channel map to the corresponding channel interrupt vector. The corresponding DMA channels interrupt flags are:

- DMA0IF (IFS1<16>)
- DMA1IF (IFS1<17>)
- DMA2IF (IFS1<18>)
- DMA3IF (IFS1<19>)

All these interrupt flags must be cleared in software.

A DMA channel is enabled as a source of interrupts via the respective DMA interrupt enable bits:

- DMA0IE (IEC1<16>)
- DMA1IE (IEC1<17>)
- DMA2IE (IEC1<18>)
- DMA3IE (IEC1<19>)

The interrupt-priority-level bits and interrupt-subpriority-level bits must be also be configured:

- DMA0IP<2:0> (IPC9<4:2>), DMA0IS<1:0> (IPC9<1:0>).
- DMA1IP<2:0> (IPC9<12:10>), DMA1IS<1:0> (IPC9<9:8>).
- DMA2IP<2:0> (IPC9<20:18>), DMA2IS<1:0> (IPC9<17:16>).
- DMA3IP<2:0> (IPC9<28:26>), DMA3IS<1:0> (IPC9<25:24>).

31.4.1 Interrupt Configuration

Each DMA channel internally has multiple interrupt flags (CHSDIF, CHSHIF, CHDDIF, CHDHIF, CHBCIF, CHCCIF, CHTAIF, CHERIF) and corresponding enable interrupt control bits (CHSDIE, CHSHIE, CHDDIE, CHDHIE, CHBCIE, CHCCIE, CHTAIE, CHERIE).

However, for the interrupt controller, there is just one dedicated interrupt flag bit per channel: DMA0IF, DMA1IF, DMA2IF, DMA3IF (IFS1<19:16>) and corresponding interrupt enable/mask bits: DMA0IE, DMA1IE, DMA2IE, DMA3IE (IEC1<19:16>).

So, note that all the interrupt conditions for a specific DMA channel share just one interrupt vector. Each DMA channel can have its own priority level independent of other DMA channels.

Note that the DMA0IF-DMA3IF bits will be set without regard to the state of the corresponding enable bits DMA0IE-DMA3IE. The DMA0IF-DMA3IF bits can be polled by software if desired.

The DMA0IE-DMA7IE bits are used to define the behavior of the Vector Interrupt Controller or INT when a corresponding DMA0IF-DMA3IF bit is set. When the corresponding DMAxIE bit is clear, the INT module does not generate a CPU interrupt for the event. If the DMAxIE bit is set, the INT module will generate an interrupt to the CPU when the corresponding DMAxIF bit is set (subject to the priority and subpriority as follows).

It is the responsibility of the user's software routine that services a particular interrupt to clear the appropriate interrupt flag bit before the service routine is complete.

The priority of each DMA channel can be set independently with the IPC9 bits. These priorities define the priority group to which the interrupt source will be assigned. The priority groups range from a value of 7 (the highest priority), to a value of 0, which does not generate an interrupt. An interrupt being serviced will be preempted by an interrupt in a higher priority group.

The subpriority bits allow setting the priority of an interrupt source within a priority group. The values of the subpriority range from 3 (the highest priority), to 0 the lowest priority. An interrupt with the same priority group but having a higher subpriority value will not preempt a lower subpriority interrupt that is in progress.

The priority group and subpriority bits allow more than one interrupt source to share the same priority and subpriority. If simultaneous interrupts occur in this configuration, the natural order of the interrupt sources within a Priority/subpriority group pair determine the interrupt generated. The natural priority is based on the vector numbers of the interrupt sources. The lower the vector number the higher the natural priority of the interrupt. Any interrupts that were overridden by natural order will then generate their respective interrupts based on Priority, subpriority, and natural order after the interrupt flag for the current interrupt is cleared.

After an enabled interrupt is generated, the CPU will jump to the vector assigned to that interrupt. The vector number for the interrupt is the same as the natural order number. The CPU will then begin executing code at the vector address. The user's code at this vector address should perform any application-specific operations and clear the DMAxIF interrupt flags, and then exit. Refer to the vector address table details in **Section 8. "Interrupts"**, for more information on interrupts.

Table 31-7: DMA Interrupt Vectors for Various Offsets with EBASE = 0x8000:0000

Interrupt	Vector/Natural Order	IRQ Number	Vector Address IntCtl.VS = 0x01	Vector Address IntCtl.VS = 0x02	Vector Address IntCtl.VS = 0x04	Vector Address IntCtl.VS = 0x08	Vector Address IntCtl.VS = 0x10
DMA0	36	48	8000 0680	8000 0B00	8000 1400	8000 2600	8000 4A00
DMA1	37	49	8000 06A0	8000 0B40	8000 1480	8000 2700	8000 4C00
DMA2	38	50	8000 06C0	8000 0B80	8000 1500	8000 2800	8000 4E00
DMA3	39	51	8000 06E0	8000 0BC0	8000 1580	8000 2900	8000 5000

Example 31-7: DMA Channel Initialization with Interrupts Enabled Code Example

```
/*
The following code example illustrates a DMA channel 0 interrupt configuration.
When the DMA channel 0 interrupt is generated, the cpu will jump to the vector assigned to
DMA0 interrupt.
*/

       IEC1CLR=0x00010000;         // disable DMA channel 0 interrupts
       IFS1CLR=0x00010000;         // clear any existing DMA channel 0 interrupt flag

       DMACONSET=0x00008000;       // enable the DMA controller
       DCH0CON=0x03;               // channel off, priority 3, no chaining

       DCH0ECON=0;                 // no start or stop irq's, no pattern match

                                   // program the transfer
       DCH0SSA=0x1d010000;         // transfer source physical address
       DCH0DSA=0x1d020000;         // transfer destination physical address
       DCH0SSIZ=0;                 // source size 256 bytes
       DCH0DSIZ=0;                 // destination size 256 bytes
       DCH0CSIZ=0;                 // 256 bytes transferred pe event

       DCH0INTCLR=0x00ff00ff;      // clear existing events, disable all interrupts
       DCH0INTSET=0x00090000;      // enable Block Complete and error interrupts

       IPC9CLR=0x0000001f;         // clear the DMA channel 0 priority and sub-priority
       IPC9SET=0x00000016;         // set IPL 5, sub-priority 2
       IEC1SET=0x00010000;         // enable DMA channel 0 interrupt

       DCH0CONSET=0x80;            // turn channel on
                                   // initiate a transfer
       DCH0ECONSET=0x00000080;     // set CFORCE to 1

                                   // do something else

                                   // will get an interrupt when the block transfer is done
                                   // or when error occurred
```

Example 31-8: DMA Channel 0 ISR Code Example

```
/*
The following code example demonstrates a simple Interrupt Service Routine for DMA channel 0
interrupts. The user's code at this vector should perform any application specific operations
and must clear the DMA0 interrupt flags before exiting.
*/

void __ISR(_DMA_0_VECTOR, ipl5) __DMA0Interrupt(void)
{
       int dmaFlags=DCH0INT&0xff;     // read the interrupt flags

/*
perform application specific operations in response to any interrupt flag set
*/

       DCH0INTCLR=0x000000ff;         // clear the DMA channel interrupt flags
       IFS1CLR = 0x00010000;          // Be sure to clear the DMA0 interrupt flags
                                      // before exiting the service routine.
}
```

Note: The DMA ISR code example shows MPLAB® C32 C compiler specific syntax. Refer to your compiler manual regarding support for ISRs.

31.5 OPERATION IN POWER-SAVING AND DEBUG MODES

Note: In this manual, a distinction is made between a power mode as it is used in a specific module, and a power mode as it is used by the device, e.g., Sleep mode of the Comparator and SLEEP mode of the CPU. To indicate which type of power mode is intended, uppercase and lowercase letters (Sleep, Idle, Debug) signify a module power mode, and all uppercase letters (SLEEP, IDLE, DEBUG) signify a device power mode.

31.5.1 DMA Operation in IDLE Mode

When the device enters IDLE mode, the system clock sources remain functional. The SIDL bit (DMACON<13>) selects whether the module will stop or continue functioning on IDLE.

- If SIDL = 0, the module will continue operation in IDLE mode and will have the clocks turned on.
- If SIDL = 1, the module will discontinue operation in IDLE mode. The DMA module will turn off the clocks so that the power consumption is more efficient.
- Note that the DMA cannot be used by a peripheral which has its SIDL bit set to '1'.

31.5.2 DMA Operation in SLEEP Mode

When the device enters SLEEP mode, the system clock is disabled. No DMA activity can occur in this mode.

31.5.3 DMA Operation in DEBUG Mode

The FRZ bit (DMACON<14>) determines whether the DMA module will run or stop while the CPU is executing debug exception code (i.e., application is halted) in DEBUG mode. When FRZ = 0, the DMA module continues to run even when application is halted in DEBUG mode. When FRZ = 1 and the application is halted in DEBUG mode, the module will freeze its operations and make no changes to the state of the DMA module. The module will resume its operation after CPU resumes execution.

Note: The FRZ bit is readable and writable only when the CPU is executing in Debug Exception mode. In all other modes, the FRZ bit reads as '0'. If FRZ bit is changed during DEBUG mode, the new value does not take effect until the current Debug Exception mode is exited and re-entered. During the Debug Exception mode, the FRZ bit reads the state of the peripheral when entering DEBUG mode.

31.6 EFFECTS OF VARIOUS RESETS

31.6.1 Device Reset

All DMA registers are forced to their Reset states upon a device Reset. When the asynchronous Reset input goes active, the DMA logic:

- Resets all fields in DMACON, DMASTAT, DMAADDR, DCRCCON, DCRCDATA, DCRCXOR
- Sets the appropriate values in each channel's register fields: DCHxCON, DCHxECON, DCHxINT, DCHxSSIZ, DCHxDSIZ, DCHxSPTR, DCHxDPTR, DCHxCSIZ, DCHxCPTR, DCHxDAT
- Registers DCHxSSA and DCHxDSA have random values after Reset
- Aborts any on-going data transfers

31.6.2 Power-On Reset

All DMA registers are forced to their Reset states upon a Power-On Reset.

31.6.3 Watchdog Timer Reset

All DMA registers are forced to their Reset states upon a Watchdog Timer Reset.

31.7 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the PIC32MX device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the Direct Memory Access Controller (DMA) module are:

Title	Application Note #
No related application notes at this time	N/A

Note: Visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the PIC32MX family of devices.

31.8 REVISION HISTORY

Revision A (October 2007)

This is the initial released version of this document.

Revision B (October 2007)

Updated document to remove Confidential status.

Revision C (April 2008)

Revised status to Preliminary; Revised U-0 to r-x; Revised Table 31-1; Revised Table 31-2 (DCHxCON, bit 3), deleted Note 1; Revised Registers 31-19, 31-39, 31-43, 31-47, 31-48, 31-49, 31-53; Revise Sections 31.3, 31.3.2; Revised Examples 31-1, 31-3, 31-4, 31-6, 31-7, 31-8; Delete Example 31-2 and renumber examples; Delete Section 31.3.3 and renumber sections; Revised Section 31.3.20.7.

Revision D (June 2008)

Revised Registers 31-58-31-60, Footnote; Revised Example 31-8; Change Reserved bits “Maintain as” to “Write”; Added Note to ON bit (DMACON Register).

Section 32. Configuration

HIGHLIGHTS

This section of the manual contains the following topics:

32.1 INTRODUCTION

A PIC32MX device includes several nonvolatile (programmable) Configuration Words that define the device's behavior.

Device Configuration features may vary according to PIC32MX family variants; however, the following Configuration features are common:

- System Clock Oscillator mode and Phase-Locked Loop (PLL)
- Secondary oscillator enable/disable
- Watchdog Timer (WDT) enable/disable and postscaler
- Boot Flash and Program Flash write-protect regions
- User ID
- Debug mode

The PIC32MX Configuration Words are located in Boot Flash memory and are programmed when the PIC32MX Boot Flash region is programmed.

System clock oscillator and PLL bits provide a large selection of flexible clock source options and PLL prescaler/postscalers.

The secondary oscillator bit enables or disables a low-power secondary oscillator that can serve as a clock source for several peripherals, such as RTCC, Timer1, and CPU.

WDT and postscaler bits allow the user to permanently disable or enable the Watchdog timer. When enabled, a postscaler can be selected to provide a wide range of Watchdog Time-out periods. A Windowed mode Watchdog feature is also available.

Boot Flash and Program Flash write-protected bits provide write protection to all of Boot Flash memory and selected regions of Program Flash memory.

User ID bits are available for programming application-specific or product-specific identification information, such as product ID or serial numbers.

Debug mode bits provide a selection of debugging modes and channels.

32.2 CONFIGURATION WORDS

Following are the device Configuration Words:

- DEVCFGx: Device Configuration Words
- DEVID: Device ID

The following table summarizes the device Configuration Words. Corresponding Configuration Words appear after the summary, followed by a detailed description of each Configuration Word.

Table 32-1: Configuration Word Summary

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
DEVCFG3	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	USERID15	USERID14	USERID13	USERID12	USERID11	USERID10	USERID9	USERID8
	7:0	USERID7	USERID6	USERID5	USERID4	USERID3	USERID2	USERID1	USERID0
DEVCFG2	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	FPLLODIV<2:0>		
	15:8	FUPLLEN(1)	—	—	—	—	FUPLLIDIV<2:0>(1)		
	7:0	—	FPLLMULT<2:0>			—	FPLLIDIV<2:0>		
DEVCFG1	31:24	—	—	—	—	—	—	—	—
	23:16	FWDTEN	—	—	WDTPS<4:0>				
	15:8	FCKSM<1:0>		FPBDIV<1:0>		—	OSCIOFNC	POSCMD<1:0>	
	7:0	IESO	—	FSOSCEN	—	—	FNOSC<2:0>		
DEVCFG0	31:24	—	—	—	CP	—	—	—	BWP
	23:16	—	—	—	—	PWP19	PWP18	PWP17	PWP16
	15:8	PWP15	PWP14	PWP13	PWP12	—	—	—	—
	7:0	—	—	—	—	ICESEL	—	DEBUG<1:0>	
DEVID	31:24	VER11	VER10	VER9	VER8	VER7	VER6	VER5	VER4
	23:16	VER3	VER2	VER1	VER0	DEV7	DEV6	DEV5	DEV4
	15:8	DEV3	DEV2	DEV1	DEV0	MANID11	MANID10	MANID9	MANID8
	7:0	MANID7	MANID6	MANID5	MANID4	MANID3	MANID2	MANID1	1

Note 1: FUPLLEN and FUPLLIDIV bits are available in USB devices.

Register 32-1: DEVCFG0: Device Configuration Word 0

r-0	r-1	r-1	R/P-1	r-1	r-1	r-1	R/P-1
—	—	—	CP	—	—	—	BWP
bit 31							bit 24

r-1	r-1	r-1	r-1	R/P-1	R/P-1	R/P-1	R/P-1
—	—	—	—	PWP19	PWP18	PWP17	PWP16
bit 23							bit 16

R/P-1	R/P-1	R/P-1	R/P-1	r-1	r-1	r-1	r-1
PWP15	PWP14	PWP13	PWP12	—	—	—	—
bit 15							bit 8

r-1	r-1	r-1	r-1	R/P-1	r-1	R/P-1	R/P-1
—	—	—	—	ICESEL	—	DEBUG<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Default unprogrammed bit value: (‘0’, ‘1’, x = Unknown)		

bit 31 **Reserved:** Write ‘0’; ignore read

bit 30-29 **Reserved:** Write ‘1’; ignore read

bit 28 **CP:** Code-Protect bit

Prevents Boot and Program Flash memory from being read or modified by an external programming device.

1 = Protection disabled
0 = Protection enabled
Refer to **Section 32.3.2 “Device Code Protection”** for more information.

bit 27-25 **Reserved:** Write ‘1’; ignore read

bit 24 **BWP:** Boot Flash Write-protect bit

Prevents Boot Flash memory from being modified during code execution.

1 = Boot Flash is writable
0 = Boot Flash is not writable
Refer to **Section 32.3.3 “Program Write Protection (PWP)”** for more information.

bit 23-20 **Reserved:** Write ‘1’; ignore read

bit 19-12 **PWP<19:12>:** Program Flash Write-protect bits

Prevents selected Program Flash memory blocks from being modified during code execution.

These bits represent the one’s complement of write-protected Program Flash memory region.

Refer to section **Section 32.3.3 “Program Write Protection (PWP)”**

bit 11-4 **Reserved:** Write ‘1’; ignore read

bit 3 **ICESEL:** In-Circuit Emulator/Debugger Communication Channel Select bit

1 = In-Circuit Emulator used EMUC2/EMUD2 pins; In-Circuit Debugger used PGC2/PGD2 pins
0 = In-Circuit Emulator used EMUC1/EMUD1 pins; In-Circuit Debugger used PGC1/PGD1 pins

bit 2 **Reserved:** Write ‘1’; ignore read

bit 1-0 **DEBUG<1:0>:** Background Debugger Enable bits (forced to ‘11’ if code-protect is enabled)

11 = Debugger is disabled
10 = Debugger is enabled
01 = Reserved (same as ‘11’ setting)
00 = Reserved (same as ‘11’ setting)

Register 32-2: DEVCFG1: Device Configuration Word 1

r-1	r-1	r-1	r-1	r-1	r-1	r-1	r-1
—	—	—	—	—	—	—	—
bit 31							bit 24

R/P-1	r-1	r-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1
FWDTEN	—	—	WDTPS<4:0>				
bit 23							bit 16

R/P-1	R/P-1	R/P-1	R/P-1	r-1	R/P-1	R/P-1	R/P-1
FCKSM<1:0>		FPBDIV<1:0>		—	OSCIOFNC	POSCMD<1:0>	
bit 15							bit 8

R/P-1	r-1	R/P-1	r-1	r-1	R/P-1	R/P-1	R/P-1
IESO	—	FSOSCEN	—	—	FNOSC<2:0>		
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Default unprogrammed bit value: (‘0’, ‘1’, x = Unknown)		

bit 31-24 **Reserved:** Write ‘1’; ignore read

bit 23 **FWDTEN:** WDT Enable bit

1 = WDT is enabled and cannot be disabled by software
0 = WDT is not enabled. It can be enabled in software

bit 22-21 **Reserved:** Write ‘1’; ignore read

bit 20-16 **WDTPS<4:0>:** WDT Postscale Select bits

10100 = 1:1048576
10011 = 1:524288
10010 = 1:262144
10001 = 1:131072
10000 = 1:65536
01111 = 1:32768
01110 = 1:16384
01101 = 1:8192
01100 = 1:4096
01011 = 1:2048
01010 = 1:1024
01001 = 1:512
01000 = 1:256
00111 = 1:128
00110 = 1:64
00101 = 1:32
00100 = 1:16
00011 = 1:8
00010 = 1:4
00001 = 1:2
00000 = 1:1

All other combinations not shown result in operation = 10100

Register 32-2: DEVCFG1: Device Configuration Word 1

bit 15-14 **FCKSM<1:0>:** Clock Switching and Monitor Selection Configuration bits

1x = Clock switching is disabled, fail-safe clock monitor is disabled
01 = Clock switching is enabled, fail-safe clock monitor is disabled
00 = Clock switching is enabled, fail-safe clock monitor is enabled

bit 13-12 **FPBDIV<1:0>:** Peripheral Bus Clock Divisor Default Value bits

11 = PBCLK is SYSCLK divided by 8
10 = PBCLK is SYSCLK divided by 4
01 = PBCLK is SYSCLK divided by 2
00 = PBCLK is SYSCLK divided by 1

bit 11 **Reserved:** Write '1'; ignore read

bit 10 **OSCIOFNC:** CLKO Enable Configuration bit

1 = CLKO output signal active on the OSCO pin; primary oscillator must be disabled or configured for the External Clock mode (EC) for the CLKO to be active (POSCMD<1:0> = 11 OR = 00)
0 = CLKO output disabled

bit 9-8 **POSCMD<1:0>:** Primary Oscillator Configuration bits

11 = Primary oscillator disabled
10 = HS Oscillator mode selected
01 = XT Oscillator mode selected
00 = External Clock mode selected

bit 7 **IESO:** Internal External Switch Over bit

1 = Internal External Switch Over mode enabled (Two-Speed Start-up enabled)
0 = Internal External Switch Over mode disabled (Two-Speed Start-up disabled)

bit 6 **Reserved:** Write '1'; ignore read

bit 5 **FSOSCEN:** Secondary Oscillator Enable bit

1 = Enable Secondary Oscillator
0 = Disable Secondary Oscillator

bit 4-3 **Reserved:** Write '1'; ignore

bit 2-0 **FNOSC<2:0>:** Oscillator Selection bits

111 = Fast RC Oscillator with divide-by-N (FRCDIV)
110 = Reserved; do not use
101 = Low-Power RC (LPRC) Oscillator
100 = Secondary Oscillator (SOSC)
011 = Primary Oscillator with PLL Module (XT + PLL, HS + PLL, EC + PLL)
010 = Primary Oscillator (XT, HS, EC)
001 = Fast RC Oscillator with divide-by-N with PLL Module (FRCDIV + PLL)
000 = Fast RC (FRC) Oscillator

Register 32-3: DEVCFG2: Device Configuration Word 2

r-1	r-1	r-1	r-1	r-1	r-1	r-1	r-1
—	—	—	—	—	—	—	—
bit 31							bit 24

r-1	r-1	r-1	r-1	r-1	R/P-1	R/P-1	R/P-1
—	—	—	—	—	FPLLODIV<2:0>		
bit 23							bit 16

R/P-1	r-1	r-1	r-1	r-1	R/P-1	R/P-1	R/P-1
FUPLLEN	—	—	—	—	FUPLLIDIV<2:0>		
bit 15							bit 8

r-1	R/P-1	R/P-1	R/P-1	r-1	R/P-1	R/P-1	R/P-1
—	FPLLMULT<2:0>			—	FPLLIDIV<2:0>		
bit 7							bit 0

Legend:			
R = readable bit	W = writable bit	P = programmable	r = reserved bit
U = unimplemented bit, read as ‘0’		-n = bit value at POR: (‘0’, ‘1’, x = unknown)	

bit 31-19 **Reserved:** Write ‘1’; ignore read

bit 18-16 **FPLLODIV<2:0>:** Default postscaler for PLL
111 = PLL output divided by 256
110 = PLL output divided by 64
101 = PLL output divided by 32
100 = PLL output divided by 16
011 = PLL output divided by 8
010 = PLL output divided by 4
001 = PLL output divided by 2
000 = PLL output divided by 1 (default setting)

bit 15 **FUPLLEN:** USB PLL Enable bit
1= Enable USB PLL
0 = Disable and bypass USB PLL

bit 14-11 **Reserved:** Write ‘1’; ignore read

bit 10-8 **FUPLLIDIV<2:0>:** PLL Input Divider bits
111 = 12x divider
110 = 10x divider
101 = 6x divider
100 = 5x divider
011 = 4x divider
010 = 3x divider
001 = 2x divider
000 = 1x divider

bit 7 **Reserved:** Write ‘1’; ignore read

Register 32-3: DEVCFG2: Device Configuration Word 2

bit 6-4 **FPLLMULT<2:0>:** Initial PLL Multiplier Value

111 = 24x Multiplier
110 = 21x Multiplier
101 = 20x Multiplier
100 = 19x Multiplier
011 = 18x Multiplier
010 = 17x Multiplier
001 = 16x Multiplier
000 = 15x Multiplier

bit 3 **Reserved:** Write '1'; ignore read

bit 2-0 **FPLLIDIV<2:0>:** PLL Input Divider Value

111 = Divide by 12
110 = Divide by 10
101 = Divide by 6
100 = Divide by 5
011 = Divide by 4
010 = Divide by 3
001 = Divide by 2
000 = Divide by 1

Register 32-4: DEVCFG3: Device Configuration Word 3

r-1	r-1	r-1	r-1	r-1	r-1	r-1	r-1
—	—	—	—	—	—	—	—
bit 31							bit 24

r-1	r-1	r-1	r-1	r-1	r-1	r-1	r-1
—	—	—	—	—	—	—	—
bit 23							bit 16

R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1
USERID15	USERID14	USERID13	USERID12	USERID11	USERID10	USERID9	USERID8
bit 15							bit 8

R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1
USERID7	USERID6	USERID5	USERID4	USERID3	USERID2	USERID1	USERID0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Default unprogrammed bit value: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘1’; ignore read

bit 15-0 **USERID:** A 16-bit value that is user defined and is readable via ICSP™ and JTAG

Register 32-5: DEVID: Device ID

R	R	R	R	R-0	R-0	R-0	R-0
VER11	VER10	VER9	VER8	VER7	VER6	VER5	VER4
bit 31							bit 24

R-1	R-0	R-0	R-1	R	R	R	R
VER3	VER2	VER1	VER0	DEV7	DEV6	DEV5	DEV4
bit 23							bit 16

R	R	R	R	R-0	R-0	R-0	R-0
DEV3	DEV2	DEV1	DEV0	MANID11	MANID10	MANID9	MANID8
bit 15							bit 8

R-0	R-1	R-0	R-1	R-0	R-0	R-1	R-1
MANID7	MANID6	MANID5	MANID4	MANID3	MANID2	MANID1	1
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-20 **VER<11:0>:** Device Variant Revision bits

bit 19-12 **DEV<7:0>:** Device ID bits
Refer to PIC32MX data sheet for variant device ID definitions.

bit 11-1 **MANID<11:1>:** JEDEC manufacturer’s identification code for Microchip Technology Inc.

bit 0 **Fixed Value:** Read as ‘1’

32.3 MODES OF OPERATION

32.3.1 Configuration Bits

In PIC32MX devices, the Configuration Words select various device Configurations. These Configuration Words are implemented as volatile memory registers and are automatically loaded from the nonvolatile programmed Configuration data mapped in the last four Words (32-bit x 4 Words) of Boot Flash memory, DEVCFG0-DEVCFG3. These are the four locations an external programming device programs with the appropriate Configuration data, see Table 32-2.

Table 32-2: Boot Flash Configuration Locations

Configuration Word	Virtual Address
DEVCFG0	0xBFC0_2FFC
DEVCFG1	0xBFC0_2FF8
DEVCFG2	0xBFC0_2FF4
DEVCFG3	0xBFC0_2FF0

On Power-on Reset (POR) or any Reset, the Configuration Words are copied from Boot Flash memory to their corresponding Configuration registers. A Configuration bit can only be programmed = 0, (an erased state = 1).

During programming, a Configuration Word can be programmed a maximum of two times before a page erase must be performed. For example, during device programming, a user can program the Configuration Word DEVCFG1 with desired data, and perform a verification or other integrity check; then, program DEVCFG1 again—this time programming any remaining unprogrammed bits = 0.

Note:	Configuration Word DEVCFG0 can only be programmed a single time before a page erase must be preformed. Each time the Boot Flash memory region is erased, bit DEVCFG0<31> is automatically programmed = 0 leaving only one additional programming operation available DEVCFG0.

After programming the Configuration Words, the user should reset the device to ensure the Configuration registers are reloaded with the new programmed data.

Configuration Register Protection

To ensure the 128-bit data integrity of each Configuration Word, a comparison is continuously made between each Configuration bit and its stored complement. If a mismatch is detected, a Configuration Mismatch Reset is generated causing a device Reset.

32.3.2 Device Code Protection

The PIC32MX features a single device code protection bit CP that when programmed = 0, protects Boot Flash and Program Flash from being read or modified by an external programming device. When code protection is enabled, only the device ID word locations are available to be read by an external programmer.

Boot Flash and Program Flash memory are not protected from self-programming during program execution when code protection is enabled. Section 32.3.3 provides more information.

32.3.3 Program Write Protection (PWP)

In addition to a device code protection bit, the PIC32MX also features write protection bits to prevent Boot Flash and Program Flash memory regions from being written during code execution.

Boot Flash memory is write-protected with a single Configuration bit, BWP (DEVCFG0<24>), when programmed = 0.

Using Configuration bits PWP<19:12> (DEVCFG0<19:12>), Program Flash memory can be write-protected entirely, or in blocks of memory starting from address 0xBD00_0000. The PWP bits represent the one's complement of a protected Flash memory region. For example, programming PWP bits = 0xFF selects a region of size '0' to be write-protected, effectively disabling the Program Flash write protection. Programming PWP bits = 0xFE selects the first block of Flash memory to be write-protected. When enabled, the selected memory range is inclusive starting from the beginning of Program Flash memory (0xBD00_0000).

The following table, Table 32-3, illustrates selectable write-protected memory regions for a device variant supporting a 4096 Byte (1024 Word) block size. Depending on the PIC32MX family variant, this memory block size may vary. Refer to the specific PIC32MX family variant data sheet for details.

Table 32-3: Flash Program Memory Write-Protect Ranges (4096 Byte/Block)

PWP Bit Value	Range Size (K-bytes)	Write-Protected Memory Ranges(1)
0xFF	0	disabled
0xFE	4	0xBD00_0FFF
0xFD	8	0xBD00_1FFF
0xFC	12	0xBD00_2FFF
0xFB	16	0xBD00_3FFF
0xFA	20	0xBD00_4FFF
0xF9	24	0xBD00_5FFF
0xF8	28	0xBD00_6FFF
0xF7	32	0xBD00_7FFF
0xF6	36	0xBD00_8FFF
0xF5	40	0xBD00_9FFF
0xF4	44	0xBD00_AFFF
0xF3	48	0xBD00_BFFF
0xF2	52	0xBD00_CFFF
0xF1	56	0xBD00_DFFF
0xF0	60	0xBD00_EFFF
0xEF	64	0xBD00_FFFF
...		
0x7F	512	0xBD07_FFFF

Note 1: Write-protected memory range is inclusive from 0xBD00_0000.

32.4 EFFECTS OF VARIOUS RESETS

On POR (Power-on Reset), BOR (Brown-out Timer Reset), $\overline{\text{MCLR}}$ (External Reset), CM (Configuration-Mismatch Reset), WDTR (Watchdog Timer Reset) or SWR (Software Reset), the Configuration Words are reloaded from their corresponding Boot Flash memory Configuration Words.

32.5 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the PIC32MX device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to Configuration Words are:

Title	Application Note #
No related application notes at this time.	N/A

Note: Please visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the PIC32MX family of devices.

32.6 REVISION HISTORY

Revision A (August 2007)

This is the initial released revision of this document.

Revision B (October 2007)

Updated document to remove Confidential status.

Revision C (April 2008)

Revised status to Preliminary; Revised U-0 to r-x; Revised Section 32.3.2; Revised Table 32-1; Revised Configuration Word DEVID Register; Revised Configuration Word DEVCFG2 Register

Revision D (June 2008)

Revised Register 31-1 (DEVCFG0); Change Reserved bits from “Maintain as” to “Write”.

NOTES:

Section 33. Programming and Diagnostics

HIGHLIGHTS

This section of the manual contains the following topics:

33.1 INTRODUCTION

PIC32MX devices provide a complete range of programming and diagnostic features that can increase the flexibility of any application using them. These features allow system designers to include:

- Simplified field programmability using two-wire In-Circuit Serial Programming™ (ICSP™) interfaces
- Debugging using ICSP
- Programming and debugging capabilities using the EJTAG extension of JTAG
- JTAG Boundary scan testing for device and board diagnostics

PIC32MX devices incorporate two programming and diagnostic modules, and a Trace Controller, that provide a range of functions to the application developer. They are summarized in Table 33-1.

Figure 33-1: Block Diagram of Programming, Debugging, and Trace Ports

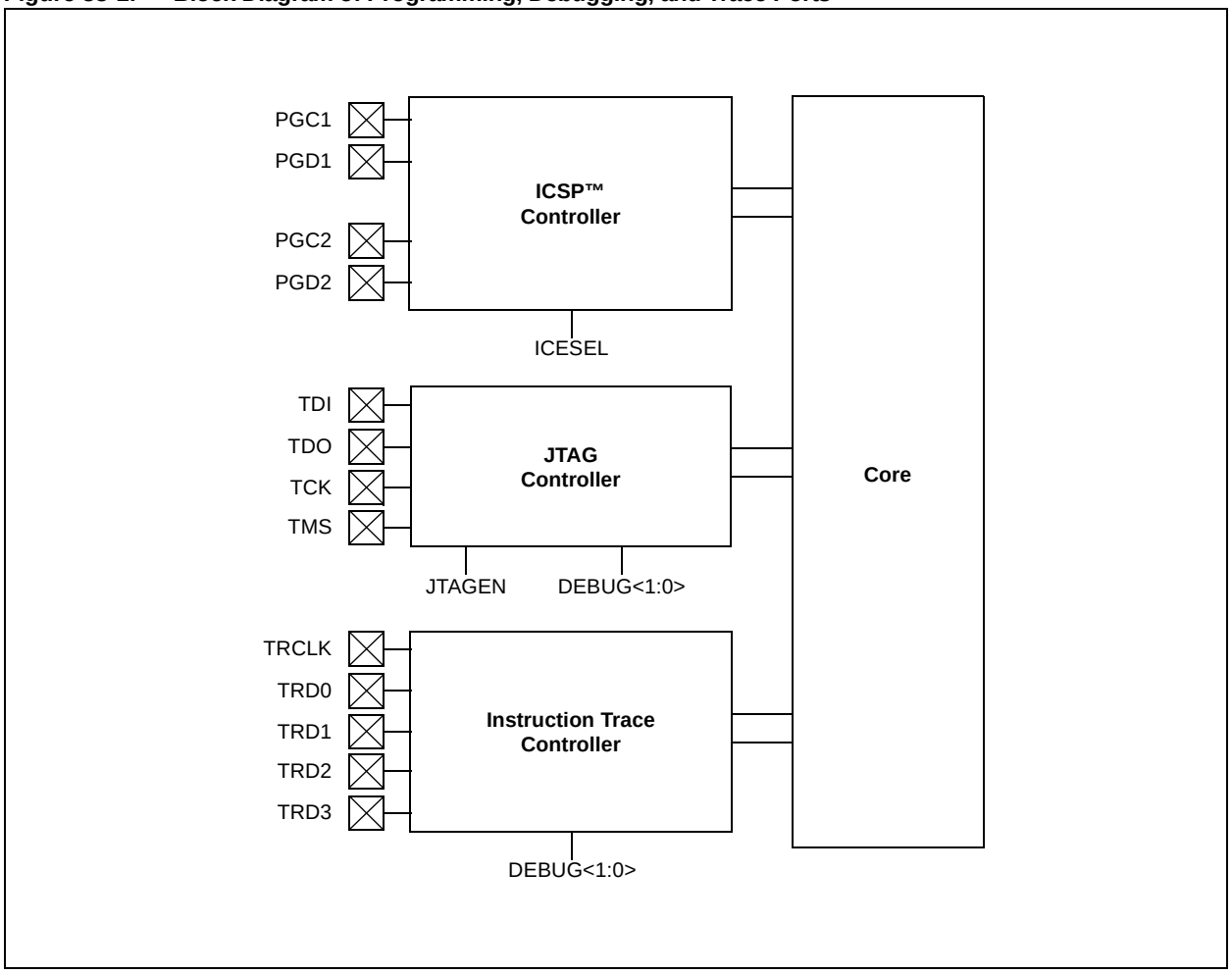


Table 33-1: Comparison of PIC32MX Programming and Diagnostic Features

Functions	Pins Used	Interface
Boundary Scan	TDI, TDO, TMS and TCK pins	JTAG
Programming and Debugging	TDI, TDO, TMS and TCK pins	EJTAG
Programming and Debugging	PGCx and PGDx pins	ICSP™

33.2 CONTROL REGISTERS

The Programming and Diagnostics module consists of the following Special Function Registers (SFRs):

- DDPCON: Control Register for the Diagnostic Module

  DDPCONCLR, DDPCONSET, DDPCONINV: Atomic Bit Manipulation Write-only Registers for DDPCON

- DEVCFG0: Device Configuration Register

The following table summarizes all Programming and Diagnostics-related registers. Corresponding registers appear after the summary, followed by a detailed description of each register.

Table 33-2: Programming and Diagnostics SFR Summary

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
DDPCON	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	—	DDPU1	DDPU2	DDPSPI1	JTAGEN	TROEN	—	—
DEVCFG0	31:24	—	—	—	CP	—	—	—	BWP
	23:16	—	—	—	—	PWP19	PWP18	PWP17	PWP16
	15:8	PWP15	PWP14	PWP13	PWP12	—	—	—	—
	7:0	—	—	—	—	ICESEL	—	DEBUG1	DEBUG0

Register 33-1: DDPCON: Debug Data Port Control Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-1	R/W-0	r-x	r-x
DDPUSB	DDPU1	DDPU2	DDPSPI1	JTAGEN	TROEN	—	—
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7 **DDPUSB:** Debug Data Port Enable for USB bit
- 1 = USB peripheral ignores USBFRZ (U1CNFG1<5>) setting
- 0 = USB peripheral follows USBFRZ setting

bit 6 **DDPU1:** Debug Data Port Enable for UART1 bit
- 1 = UART1 peripheral ignores FRZ (U1MODE<14>) setting
- 0 = UART1 peripheral follows FRZ setting

bit 5 **DDPU2:** Debug Data Port Enable for UART2 bit
- 1 = UART2 peripheral ignores FRZ (U2MODE<14) setting
- 0 = UART2 peripheral follows FRZ setting

bit 4 **DDPSPI1:** Debug Data Port Enable for SPI1 bit
- 1 = SPI1 peripheral ignores FRZ (SPI1CON<14>) setting
- 0 = SPI1 peripheral follows FRZ setting

bit 3 **JTAGEN:** JTAG Port Enable bit
- 1 = Enable JTAG Port
- 0 = Disable JTAG Port

bit 2 **TROEN:** Trace Output Enable bit
- 1 = Enable Trace Port
- 0 = Disable Trace Port

bit **Reserved:** Write ‘1’; ignore read

Register 33-2: DEVCFG0: Device Configuration Register

r-1	r-1	r-1	R/P-1	r-1	r-1	r-1	R/P-1
—	—	—	CP	—	—	—	BWP
bit 31							bit 24

r-1	r-1	r-1	r-1	R/P-1	R/P-1	R/P-1	R/P-1
—	—	—	—	PWP19	PWP18	PWP17	PWP16
bit 23							bit 16

R/P-1	R/P-1	R/P-1	R/P-1	r-1	r-1	r-1	r-1
PWP15	PWP14	PWP13	PWP12	—	—	—	—
bit 15							bit 8

r-1	r-1	r-1	r-1	R/P-1	r-1	R/P-1	R/P-1
—	—	—	—	ICESEL	—	DEBUG1	DEBUG0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 3 **ICESEL:** ICE Debugger Port Select bit
1 = ICE Debugger uses PGC2/PGD2
0 = ICE Debugger uses PGC1/PGD1

bit 1-0 **DEBUG<1:0>:** Background Debugger Enable bits (forced to ‘11’ if Code-Protect is enabled)
11 = ICE Debugger Disabled
10 = ICE Debugger Enabled
01 = Reserved (same as ‘11’ setting)
00 = Reserved (same as ‘11’ setting)

33.3 OPERATION

The PIC32MX family of devices has multiple Programming and Debugging options including:

- In-Circuit Programming via ICSP
- In-Circuit Programming EJTAG
- Debugging via ICSP
- Debugging via EJTAG
- Special Debug modes for select communication peripherals
- Boundary Scan

33.3.1 Device Programming Options

Note:	Following sections provide a brief overview of each programming options. For more detailed information, refer to PIC32MX Programming Specification.

Note:	For all device programming options, a minimum VDD requirement for Flash erase and programming operations is required. Refer to the specific device data sheet for further details.

33.3.1.1 In-Circuit Serial Programming

ICSP is Microchip's proprietary solution to providing microcontroller programming in the target application. ICSP is also the most direct method to program the device, whether the controller is embedded in a system or loaded into a device programmer.

33.3.1.1.1 ICSP Interface

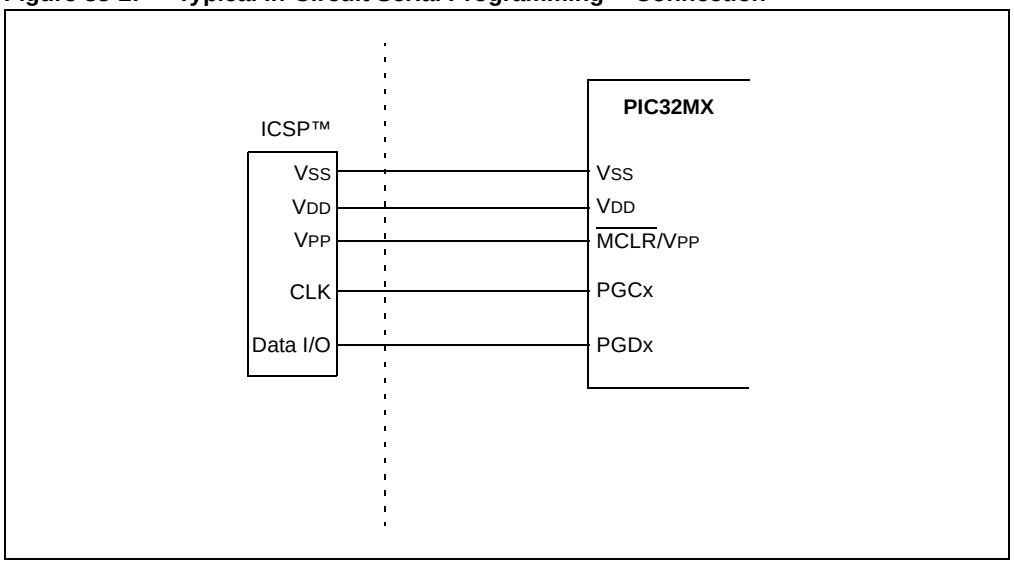
ICSP uses two pins as the core of its interface. The programming data line (PGD) functions as both an input and an output, allowing programming data to be read in and device information to be read out on command. The programming clock line (PGC) is used to clock in data and control the overall process.

Most PIC32MX devices have more than one pair of PGC and PGD pins; these are multiplexed with other I/O or peripheral functions. Individual ICSP pin pairs are indicated by number (e.g., PGC1/PGD1, etc.), and are generically referred to as 'PGCx' and 'PGDx'. The multiple PGCx/PGDx pairs provide additional flexibility in system design by allowing users to incorporate ICSP on the pair of pins that is least constrained by the circuit design. All PGCx and PGDx pins are functionally tied together and behave identically, and any one pair can be used for successful device programming. The only limitation is that both pins from the same pair must be used.

In addition to the PGCx and PGDx pins, ICSP requires that all voltage supply (including voltage regulator pin ENVREG) and ground pins on the device must be connected. The $\overline{\text{MCLR}}$ pin, which is used with PGCx to enter and control the programming process, must also be connected to the programmer.

A typical In-Circuit Serial Programming connection is shown in Figure 33-2.

Figure 33-2: Typical In-Circuit Serial Programming™ Connection



33.3.1.1.2 ICSP Operation

ICSP uses a combination of internal hardware and external control to program the target device. Programming data and instructions are provided on PGD. ICSP uses a special set of commands to control the overall process, combined with standard PIC32MX instructions to execute the actual writing of the program memory. PGD also returns data to the external programmer when responding to queries.

Control of the programming process is achieved by manipulating PGC and $\overline{\text{MCLR}}$. Entry into and exit from Programming mode involves applying (or removing) voltage to $\overline{\text{MCLR}}$ while supplying a code sequence to PGD and a clock to PGC. Any one of the PGCx/PGDx pairs can be used to enter programming.

The internal process is regulated by a state machine built into the PIC32MX core logic; however, overall control of the process must be provided by the external programming device. Microchip programming devices, such as the MPLAB® PM 3 (used with MPLAB IDE software), include the necessary hardware and algorithms to manage the programming process for PIC32MX. Users who are interested in a more detailed description, or who are considering designing their own programming interface for PIC32MX devices, should consult the appropriate PIC32MX device programming specification.

33.3.1.2 Enhanced In-Circuit Serial Programming

The Enhanced In-Circuit Serial Programming (ICSP) protocol is an extension of the original ICSP. It uses the same physical interface as the original, but changes the location and execution of programming control to a software application written to the PIC32MX device. Use of Enhanced ICSP results in significant decrease in overall programming time.

ICSP uses a simple state machine to control each step of the programming process; however, that state machine is controlled by an external programmer. In contrast, Enhanced ICSP uses an on-board bootloader, known as the program executive, to manage the programming process. While overall device programming is still controlled by an external programmer, the program executive manages most of the tasks that must be directly controlled by the programmer in standard ICSP.

The program executive implements its own command set, wider in range than the original ICSP, that can directly erase, program and verify the device program memory. This avoids the need to repeatedly run ICSP command sequences to perform simple tasks. As a result, Enhanced ICSP is capable of programming or reprogramming a device faster than the original ICSP.

The program executive is not preprogrammed into PIC32MX devices. If Enhanced ICSP is needed, the user must use standard ICSP to program the executive to the executive memory space in RAM. This can be done directly by the user, or automatically, using a compatible Microchip programming system. After the Programming Executive is written the device can be programmed using EICSP

For additional information on EICSP and the program executive, refer to the appropriate PIC32MX device programming specification.

33.3.1.3 EJTAG Device Programming Using the JTAG Interface

The JTAG interface can also be used to program PIC32MX family devices in their target applications. Using EJTAG with the JTAG interface allows application designers to include a dedicated test and programming port into their applications, with a single 4-pin interface, without imposing the circuit constraints that the ICSP interface may require.

33.3.1.4 Enhanced EJTAG Programming Using the JTAG Interface

Enhanced EJTAG programming uses the standard JTAG interface but uses a Programming Executive written to RAM. Use of the Programming Executive with the JTAG interface provides a significant improvement in programming speed.

33.3.2 Debugging

33.3.2.1 ICSP and In-Circuit Debugging

ICSP also provides a hardware channel for the In-Circuit Debugger (ICD) which allows externally controlled debugging of software. Using the appropriate hardware interface and software environment, users can force the device to single-step through its code, track the actual content of multiple registers and set software breakpoints.

To use ICD, an external system that supports ICD must load a debugger executive program into the microcontroller. This is automatically handled by many debugger tools, such as the MPLAB IDE. For PIC32MX devices, the program is loaded into the last page of the Boot Flash memory space. When not debugging, the application is free to use the last page of Boot Flash Memory.

PIC32MX ICSP supports standard debugging functions including memory and register viewing and modification. Breakpoints can be set and the program execution may be stopped or started. In addition to these functions registers or memory contents can be viewed and modified while the CPU is running.

In contrast with programming, only one of the ICSP ports may be used for ICD. If more than one ICSP port is implemented a Configuration bit determines which port is available. Depending on the particular PIC32MX device, there may be two or more ICSP ports that can be selected for this function. The active ICSP debugger port is selected by the ICESEL Configuration bit(s). For information on specific devices, refer to the appropriate device data sheet.

33.3.2.2 EJTAG Debugging

The industry standard EJTAG interface allows Third Party EJTAG tools to be used for debugging. Using the EJTAG interface, memory and registers can be viewed and modified. Breakpoints can be set and the program execution may be stopped, started or single-stepped.

33.3.3 Special Debug Modes for Select Communications Peripherals

To aid in debugging applications certain I/O peripherals have a user controllable bit to override the Freeze function in the peripheral. This allows the module to continue to send any data buffered within the peripheral even when a debugger attempts to halt the peripheral. The Debug mode control bits for these peripherals are contained in the DDPCON register.

33.3.4 JTAG Boundary Scan

As the complexity and density of board designs increases, testing electrical connections between the components on fully assembled circuit boards poses many challenges. To address these challenges, the Joint Test Action Group (JTAG) developed a method for boundary scan testing that was later standardized as IEEE 1149.1-2001, *"IEEE Standard Test Access Port and Boundary Scan Architecture"*. Since its adoption, many microcontroller manufacturers have added device programming to the capabilities of the test port.

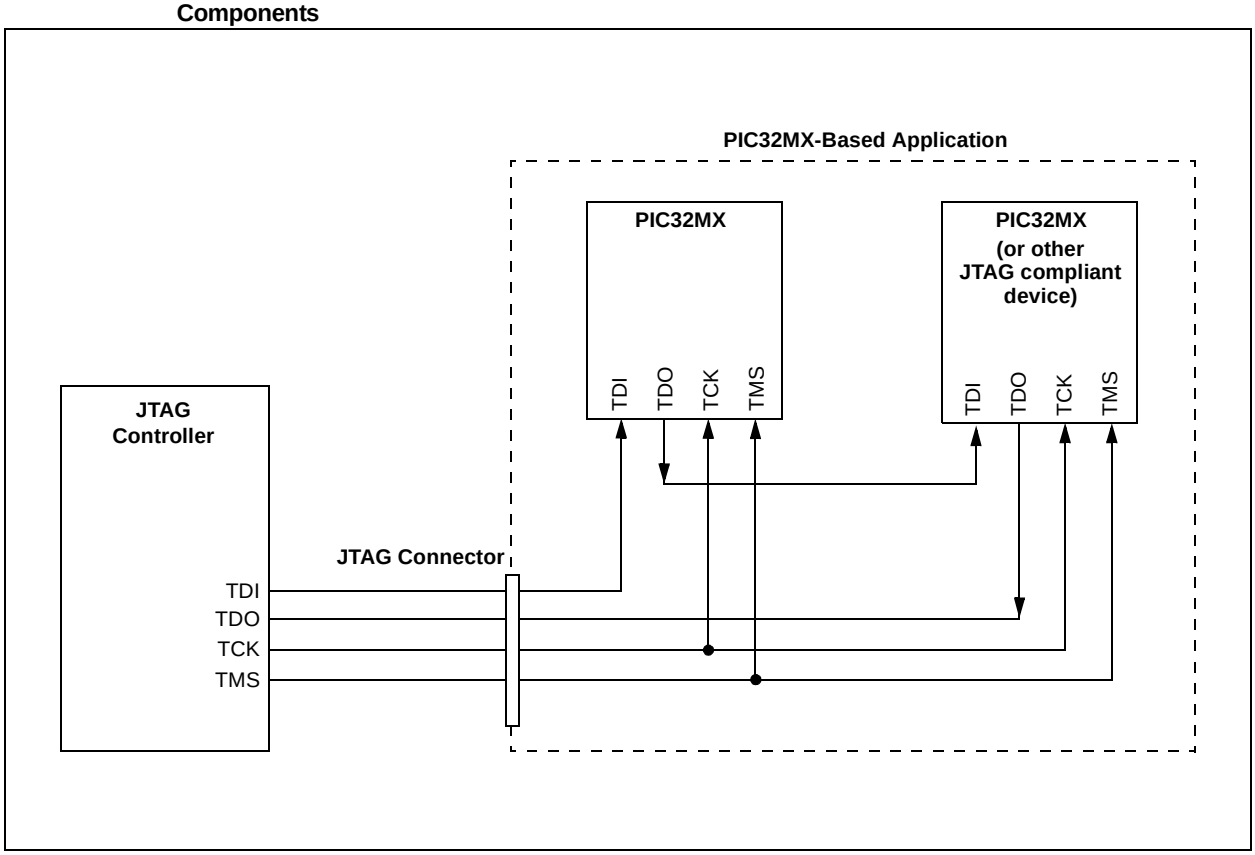
The JTAG boundary scan method is the process of adding a Shift register stage adjacent to each of the component's I/O pins. This permits signals at the component boundaries to be controlled and observed, using a defined set of scan test principles. An external tester or controller provides instructions and reads the results in a serial fashion. The external device also provides common clock and control signals. Depending on the implementation, access to all test signals is provided through a standardized 4-pin interface.

In system-level applications, individual JTAG enabled components are connected through their individual testing interfaces (in addition to their more standard application-specific connections). Devices are connected in a series or daisy-chained fashion, with the test output of one device connected exclusively to the test input of the next device in the chain. Instructions in the JTAG boundary scan protocol allow the testing of any one device in the chain, or any combination of devices, without testing the entire chain. In this method, connections between components, as well as connections at the boundary of the application, may be tested.

A typical application incorporating the JTAG boundary scan interface is shown in Figure 33-3. In this example, a PIC32MX microcontroller is daisy-chained to a second JTAG compliant device. Note that the TDI line from the external tester supplies data to the TDI pin of the first device in the chain (in this case, the microcontroller). The resulting test data for this two-device chain is provided from the TDO pin of the second device to the TDO line of the tester.

This section describes the JTAG module and its general use. Users interested in using the JTAG interface for device programming should refer to the appropriate PIC32MX device programming specification for more information.

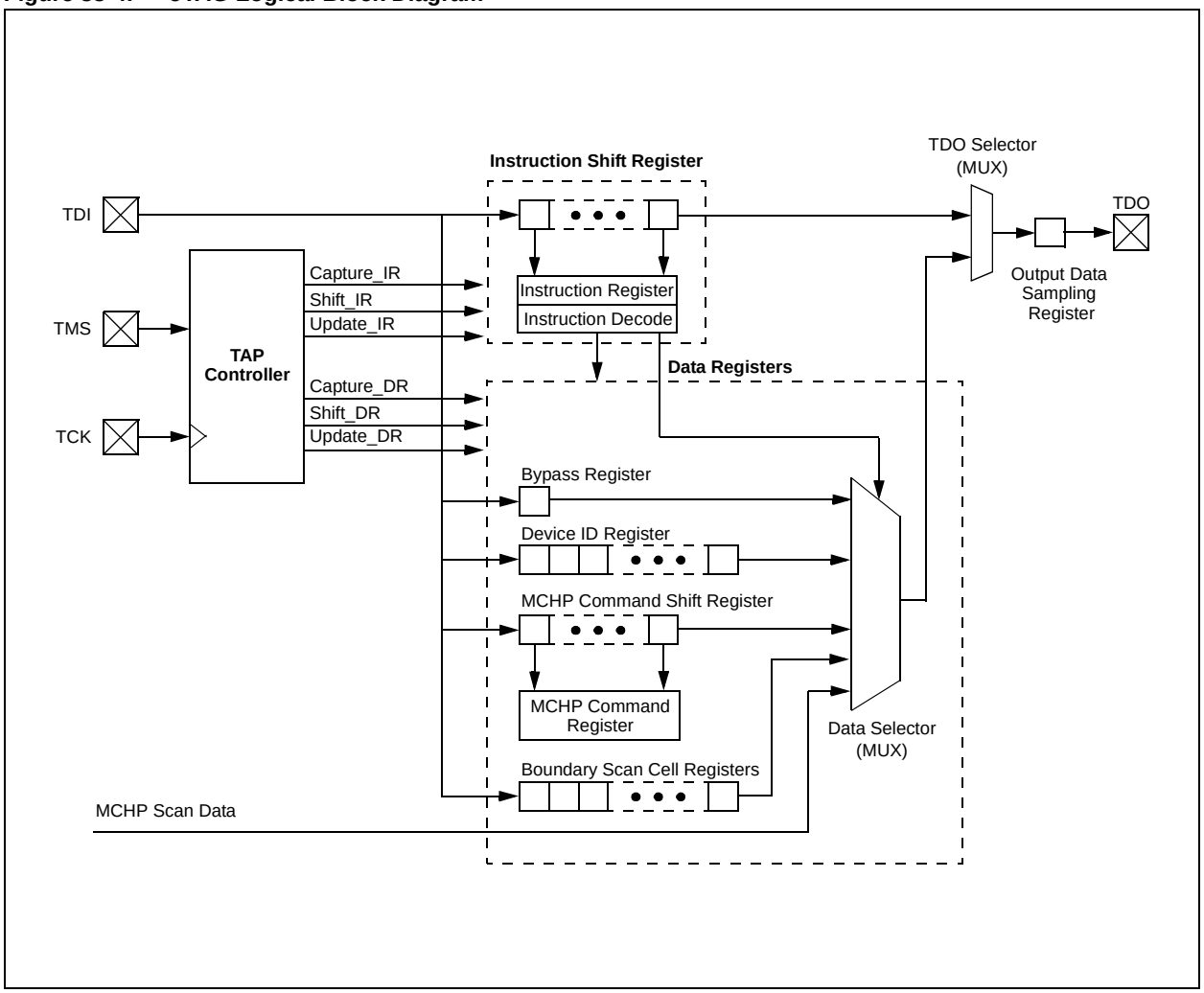
Figure 33-3: Overview of PIC32MX-based JTAG Compliant Application Showing Daisy-Chaining of Components



In PIC32MX family devices, the hardware for the JTAG boundary scan is implemented as a peripheral module (i.e., outside of the CPU core) with additional integrated logic in all I/O ports. A logical block diagram of the JTAG module is shown in Figure 33-4. It consists of the following key elements:

- TAP Interface Pins (TDI, TMS, TCK and TDO)
- TAP Controller
- Instruction Shift register and Instruction Register (IR)
- Data Registers (DR)

Figure 33-4: JTAG Logical Block Diagram



33.3.4.1 Test Access Port (TAP) and TAP Controller

The Test Access Port (TAP) on the PIC32MX device family is a general purpose port that provides test access to many built-in support functions and test logic defined in IEEE Standard 1149.1. The TAP is enabled by the JTAGEN bit in the DDPCON register. The TAP is enabled, JTAGEN = 1, by default when the device exits Power-on-Reset (POR) or any device Reset. Once enabled, the designated I/O pins become dedicated TAP pins. See the appropriate PIC32MX device data sheet for details about enabling the JTAG module and identifying JTAG control pins.

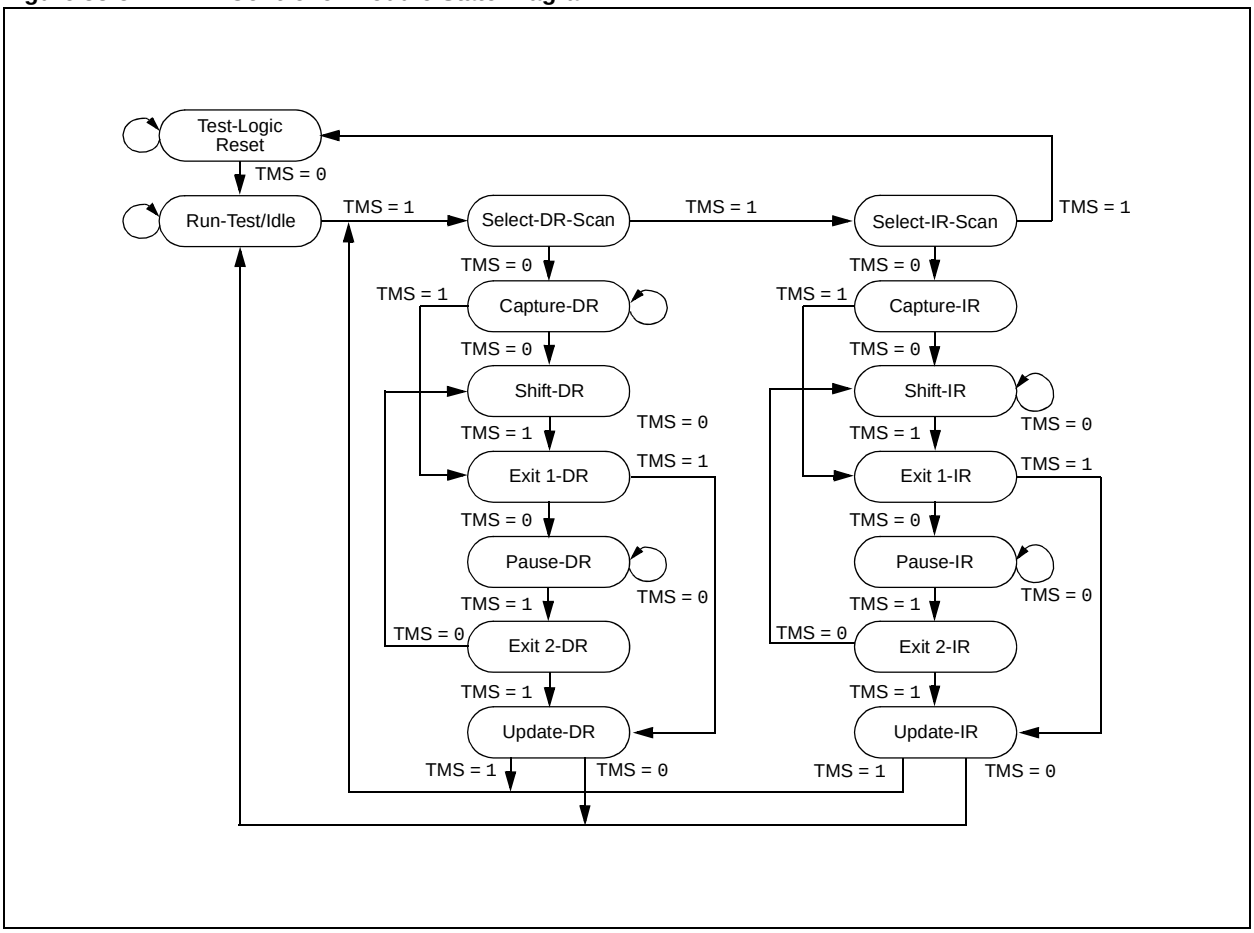
The PIC32MX implements a 4-pin JTAG interface with these pins:

- TCK (Test Clock Input): Provides the clock for test logic.
- TMS (Test Mode Select Input): Used by the TAP to control test operations.
- TDI (Test Data Input): Serial input for test instructions and data.
- TDO (Test Data Output): Serial output for test instructions and data.

To minimize I/O loss due to JTAG, the optional TAP Reset input pin, specified in the standard, is not implemented on PIC32MX devices. For convenience, a “soft” TAP Reset has been included in the TAP controller, using the TMS and TCK pins. To force a port Reset, apply a logic high to the TMS pin for at least 5 rising edges of TCK. Note that device Resets (including POR) do not automatically result in a TAP Reset; this must be done by the external JTAG controller using the soft TAP Reset.

The TAP controller on the PIC32MX family devices is a synchronous finite state machine that implements the standard 16 states for JTAG. Figure 33-5 shows all the module states of the TAP controller. All Boundary Scan Test (BST) instructions and test results are communicated through the TAP via the TDI pin in a serial format, Least Significant bit first.

Figure 33-5: TAP Controller Module State Diagram



By manipulating the state of TMS and the clock pulses on TCK, the TAP controller can be moved through all of the defined module states to capture, shift and update various instruction and/or data registers. Figure 33-5 shows the state changes on TMS as the controller cycles through its state machine. Figure 33-6 shows the timing of TMS and TCK while transitioning the controller through the appropriate module states for shifting in an instruction. In this example, the sequence shown demonstrates how an instruction is read by the TAP controller.

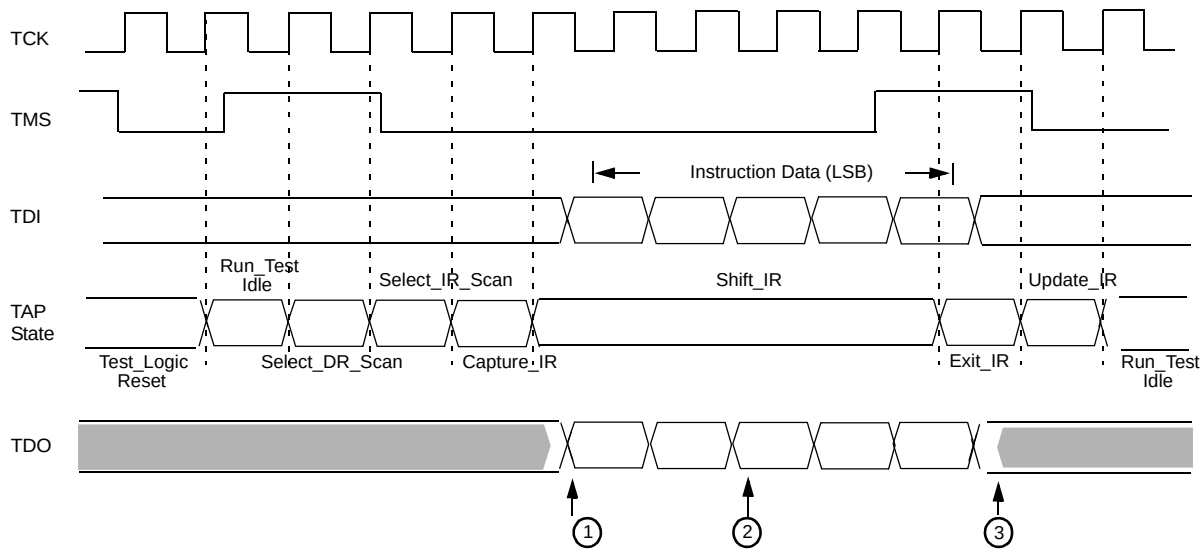
All TAP controller states are entered on the rising edge of the TCK pin. In this example, the TAP controller starts in the Test-Logic Reset state. Since the state of the TAP controller is dependent on the previous instruction, and therefore could be unknown, it is good programing practice to begin in the Test-Logic Reset state.

When TMS is asserted low on the next rising edge of TCK, the TAP controller will move into the Run-Test/Idle state. On the next two rising edges of TCK, TMS is high; this moves the TAP controller to the Select-IR-Scan state.

On the next two rising edges of TCK, TMS is held low; this moves the TAP controller into the Shift-IR state. An instruction is shifted in to the Instruction Shift register via the TDI on the next four rising edges of TCK. After the TAP controller enters this state, the TDO pin goes from a high-impedance state to active. The controller shifts out the initial state of the Instruction Register (IR) on the TDO pin, on the falling edges of TCK, and continues to shift out the contents of the Instruction Register while in the Shift-IR state. The TDO returns to the high-impedance state on the first falling edge of TCK upon exiting the shift state.

On the next three rising edges of TCK, the TAP controller exits the Shift_IR state, updates the Instruction Register and then moves back to the Run-Test/Idle state. Data, or another instruction, can now be shifted in to the appropriate Data or Instruction Register.

Figure 33-6: TAP State Transitions for Shifting in an Instruction



Note 1: TDO pin is always in a high-impedance state, until the first falling edge of TCK, in either the Shift_IR or Shift_DR states.

2: TDO is no longer high-impedance; the initial state of the Instruction Register (IR) is shifted out on the falling edge of TCK.

3: TDO returns to high-impedance again on the first falling edge of TCK in the Exit_IR state.

33.3.4.2 JTAG Registers

The JTAG module uses a number of registers of various sizes as part of its operation. In terms of bit count, most of the JTAG registers are single-bit register cells, integrated into the I/O ports. Regardless of their location within the module, none of the JTAG registers are located within the device data memory space, and cannot be directly accessed by the user in normal operating modes.

33.3.4.2.1 Instruction Shift Register and Instruction Register

The Instruction Shift register is a 5-bit shift register used for selecting the actions to be performed and/or what data registers to be accessed. Instructions are shifted in, Least Significant bit first, and then decoded.

A list and description of implemented instructions is given in **Section 33.3.4.4 "JTAG Instructions"**.

33.3.4.2.2 Data Registers

Once an instruction is shifted in and updated into the Instruction Register, the TAP controller places certain data registers between the TDI and TDO pins. Additional data values can then be shifted into these data registers as needed.

The PIC32MX device family supports three data registers:

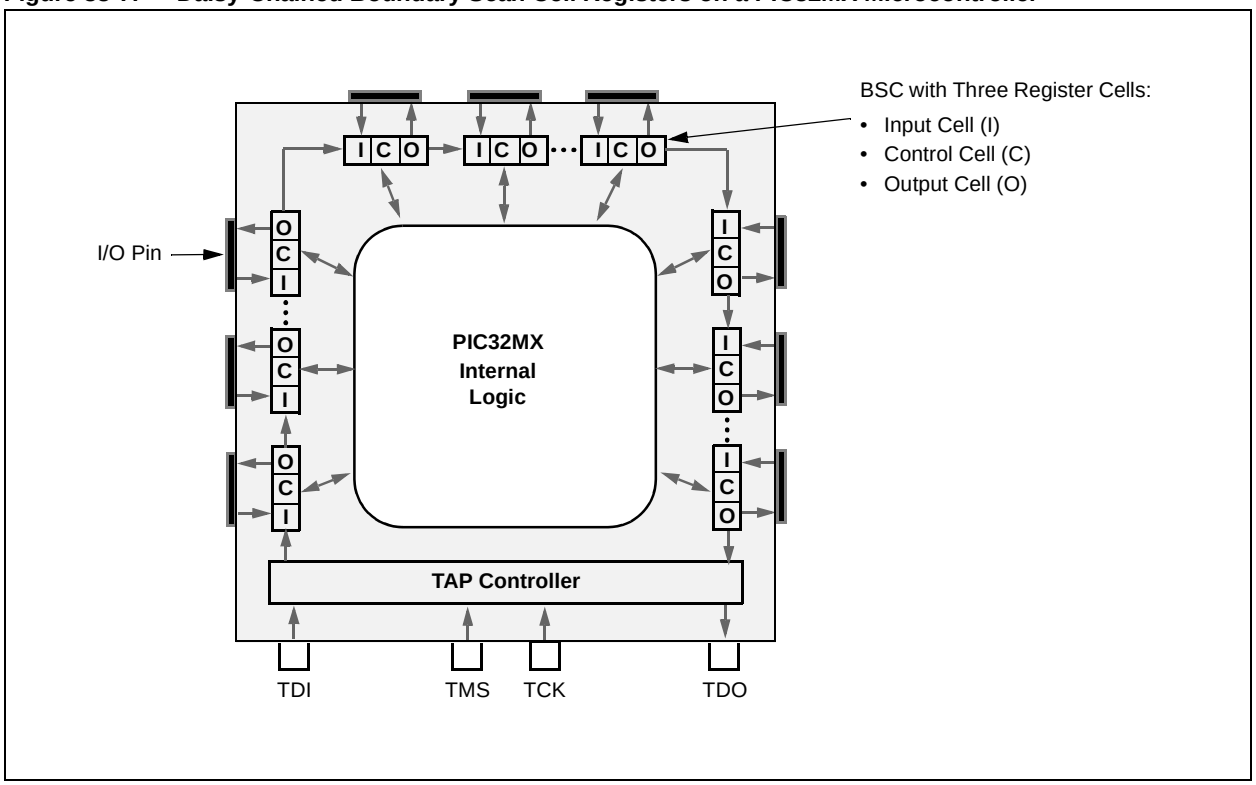
- BYPASS Register: A single-bit register which allows the boundary scan test data to pass through the selected device to adjacent devices. The BYPASS register is placed between the TDI and TDO pins when the BYPASS instruction is active.
- Device ID Register: A 32-bit part identifier. It consists of an 11-bit manufacturer ID assigned by the IEEE (29h for Microchip Technology), device part number and device revision identifier. When the IDCODE instruction is active, the device ID register is placed between the TDI and TDO pins. The device data ID is then shifted out on to the TDO pin, on the next 32 falling edges of TCK, after the TAP controller is in the Shift_DR.
- MCHP Command Shift Register: An 8-bit shift register that is placed between the TDI and TDO pins when the MCHP_CMD instruction is active. This shift register is used to shift in Microchip commands.

33.3.4.3 Boundary Scan Register (BSR)

The BSR is a large shift register that is comprised of all the I/O Boundary Scan Cells (BSCs), daisy-chained together (Figure 33-7). Each I/O pin has one BSC, each containing 3 BSC registers: an input cell, an output cell and a control cell. When the SAMPLE/PRELOAD or EXTEST instructions are active, the BSR is placed between the TDI and TDO pins, with the TDI pin as the input and the TDO pin as the output.

The size of the BSR depends on the number of I/O pins on the device. For example, the 100-pin PIC32MX general purpose parts have 82 I/O pins. With 3 BSC registers for each of the 82 I/Os, this yields a Boundary Scan register length of 244 bits. This is due to the MCLR pin being an input-only BSR cell. Information on the I/O port pin count of other PIC32MX devices can be found in their specific device data sheets.

Figure 33-7: Daisy-Chained Boundary Scan Cell Registers on a PIC32MX Microcontroller



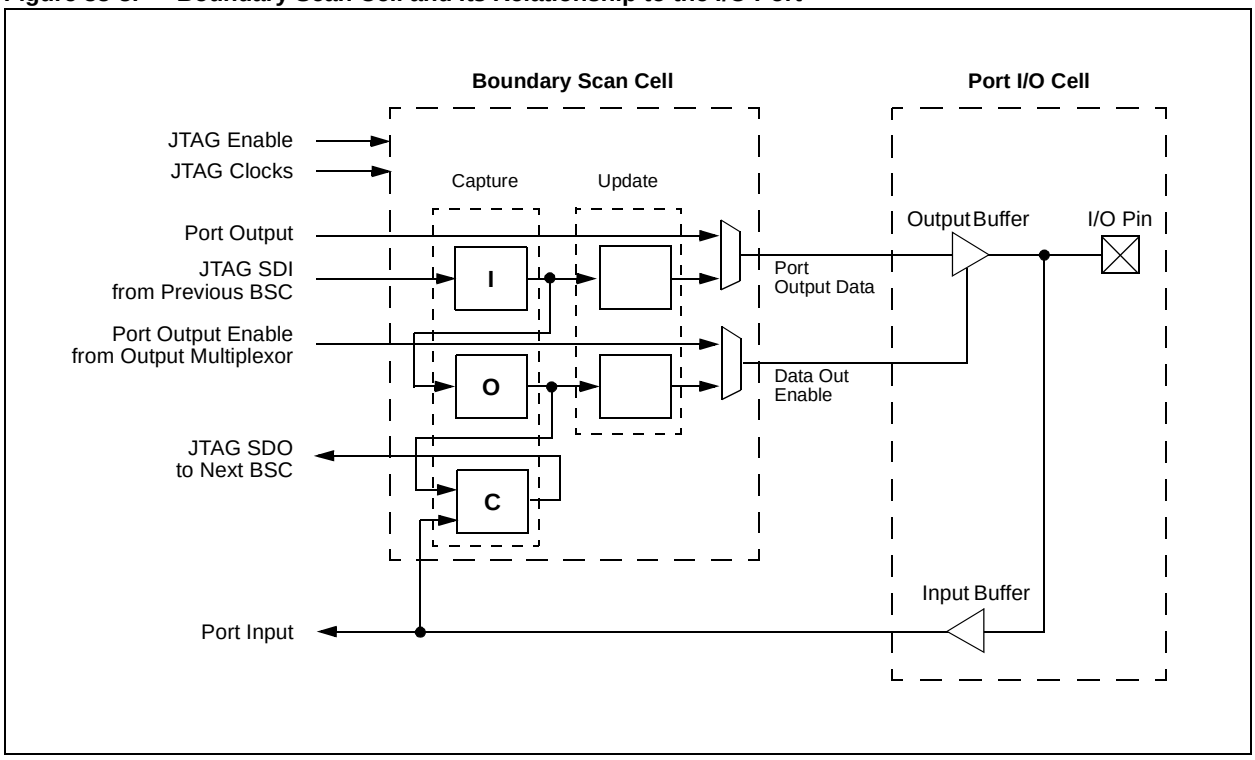
33.3.4.3.3 Boundary Scan Cell (BSC)

The function of the BSC is to capture and override I/O input or output data values when JTAG is active. The BSC consists of three Single-Bit Capture register cells and two Single-Bit Holding register cells. The capture cells are daisy-chained to capture the port's input, output and control (output-enable) data, as well as pass JTAG data along the Boundary Scan register. Command signals from the TAP controller determine if the port of JTAG data is captured, and how and when it is clocked out of the BSC.

The first register either captures internal data destined to the output driver, or provides serially scanned in data for the output driver. The second register captures internal output-enable control from the output driver and also provides serially scanned in output-enable values. The third register captures the input data from the I/O's input buffer.

Figure 33-8 shows a typical BSC and its relationship to the I/O port's structure.

Figure 33-8: Boundary Scan Cell and Its Relationship to the I/O Port



33.3.4.4 JTAG Instructions

PIC32MX family devices support the mandatory instruction set specified by IEEE 1149.1, as well as several optional public instructions defined in the specification. These devices also implement instructions that are specific to Microchip devices.

The mandatory JTAG instructions are:

- BYPASS (0x1F): Used for bypassing a device in a test chain; this allows the testing of off-chip circuitry and board-level interconnections.
- SAMPLE/PRELOAD (0x02): Captures the I/O states of the component, providing a snapshot of its operation.
- EXTEST (0x06): Allows the external circuitry and interconnections to be tested, by either forcing various test patterns on the output pins, or capturing test results from the input pins.

Microchip has implemented optional JTAG instructions and manufacturer-specific JTAG commands in PIC32MX devices. Please refer to Figure 33-3, 33-4, 33-5 and 33-6.

Table 33-3: JTAG Commands

OPCODE	Name	Device Integration
0x1F	Bypass	Bypasses device in test chain
0x00	HIGHZ	Places device in a high-impedance state, all pins are forced to inputs
0x01	ID Code	Shifts out the devices ID code
0x02	Sample/Preload	Samples all pins or loads a specific value into output latch
0x06	EXTEST	Boundry Scan

Table 33-4: Microchip TAP IR Commands

OPCODE	Name	Device Integration
0x01	MTAP_IDCODE	Shifts out the devices ID code
0x07	MTAP_COMMAND	Configure Microchip TAP controller for DR commands
0x04	MTAP_SW_MTAP	Select Microchip TAP controller
0x05	MTAP_SW_ETAP	Select EJTAG TAP controller

Table 33-5: Microchip TAP 8-bit DR Commands

OPCODE	Name	Device Integration
0x00	MCHP_STATUS	Perform NOP and return Status
0xD1	MCHP_ASERT_RST	Request Assert Device Reset
0xD0	MCHP_DE_ASSERT_RST	Request De-Assert Device Reset
0xFC	MCHP_ERASE	Perform a Chip Erase
0xFE	MCHP_FLASH_ENABLE	Enables fetches and loads to the Flash from the CPU
0xFD	MCHP_FLASH_DISABLE	Disables fetches and loads to the Flash from the CPU
0xFF	MCHP_READ_CONFIG	Forces device to reread the configuration settings and initialize accordingly

Table 33-6: EJTAG Commands

OPCODE	Name	Device Integration	Data Length for the Following DR
0x00		Not Used	
0x01	IDCODE	Selects the Devices ID Code register	32 bits
0x02		Not Used	
0x03	IMPCODE	Selects Implementation Register	
0x04(2)	MTAP_SW_MTAP	Select Microchip TAP controller	
0x05(2)	MTAP_SW_ETAP	Select EJTAG TAP controller	
0x06-0x07		Not Used	
0x08	ADDRESS	Selects the Address Register	32 bits
0x09	DATA	Selects the Data Register	32 bits
0x0A	CONTROL	Selects the EJTAG control register	32 bits
0x0B	ALL	Selects the Address, Data, EJTAG control register	96 bits
0x0C	EJTAGBOOT	Forces the CPU to take a Debug Exception after boot	1 bit
0x0D	NORMALBOOT	Makes the CPU execute the reset handler after a boot	1 bit
0x0E	FASTDATA	Selects the Data and Fast Data Registers	1 bit
0x0F-0x1B		Reserved	
0x1C-0xFE		Not Used	
0xFF		Select the Bypass Register	

Note 1: For complete information about EJTAG commands and protocol, refer to EJTAG Specification available on MIPS Technologies web site www.mips.com.

2: Not EJTAG commands but are recognized by the Microchip implementation.

33.3.5 Boundary Scan Testing (BST)

Boundary Scan Testing (BST) is the method of controlling and observing the boundary pins of the JTAG compliant device, like those of the PIC32MX family, utilizing software control. BST can be used to test connectivity between devices by daisy-chaining JTAG compliant devices to form a single scan chain. Several scan chains can exist on a PCB to form multiple scan chains. These multiple scan chains can then be driven simultaneously to test many components in parallel. Scan chains can contain both JTAG compliant devices and non-JTAG compliant devices.

A key advantage of BST is that it can be implemented without physical test probes; all that is needed is a 4-wire interface and an appropriate test platform. Since JTAG boundary scan has been available for many years, many software tools exist for testing scan chains without the need for extensive physical probing. The main drawback to BST is that it can only evaluate digital signals and circuit continuity; it cannot measure input or output voltage levels or currents.

33.3.5.1 Related JTAG Files

To implement BST, all JTAG test tools will require a Boundary Scan Description Language (BSDL) file. BSDL is a subset of VHDL (VHSIC Hardware Description Language), and is described as part of IEEE Std. 1149.1. The device-specific BSDL file describes how the standard is implemented on a particular device and how it operates.

The BSDL file for a particular device includes the following:

- The pinout and package configuration for the particular device
- The physical location of the TAP pins
- The Device ID register and the device ID
- The length of the Instruction Register
- The supported BST instructions and their binary codes
- The length and structure of the Boundary Scan register
- The boundary scan cell definition

The name for each BSDL file is the device name and silicon revision—for example, PIC32MX320F128L_A2.BSD is the BSDL file for PIC32MX320F128L, silicon revision A2.

33.4 INTERRUPTS

Programming and Debugging operations are not performed during code execution and are therefore not affected by interrupts. Trace Operations will report the change in code execution when a interrupt occurs but the Trace Controller is not affected by interrupts.

33.5 I/O PINS

In order to interface the numerous Programming and Debugging option available and still provide peripherals access to the pins, the pins are multiplexed with peripherals. Table 33-7 describes the function of the Programming and Debug related pins.

Table 33-7: Programming and Debugging Pin Functions

Pin Name	Function				Description
	Program Mode	Debug Mode	Trace Mode	Boundary Scan Mode	
$\overline{MCLR}$	$\overline{MCLR}$	$\overline{MCLR}$	$\overline{MCLR}$	$\overline{MCLR}$	Master Clear, used to enter ICSP™ mode and to override JTAGEN (DDPCON<3>)
PGC1	PGC1/ Alternate	PGC1/ Alternate	PGC1/ Alternate	Alternate	ICSP Clock, determined by ICESEL Configuration bit (DEVCFG0<3>)
PGD1	PGD1/ Alternate	PGD1/ Alternate	PGD1/ Alternate	Alternate	ICSP Data, determined by ICESEL (DEVCFG0<3>) and DEBUG Configuration bits (DEVCFG0<1:0>)
PGC2	PGC2/ Alternate	PGC2/ Alternate	PGC2/ Alternate	Alternate	Alternate ICSP Clock, determined by ICESEL (DEVCFG0<3>) and DEBUG Configuration bits (DEVCFG0<1:0>)
PGD2	PGD2/ Alternate	PGD2/ Alternate	PGD2/ Alternate	Alternate	Alternate ICSP Data, determined by ICESEL (DEVCFG0<3>) and DEBUG Configuration bits (DEVCFG0<1:0>)
TCK	TCK	TCK	TCK	TCK	JTAG Clock, determined by JTAGEN control bit (DDPCON<3>)
TDO	TDO	TDO	TDO	TDO	JTAG Data Out, determined by JTAGEN control bit (DDPCON<3>)
TDI	TDI	TDI	TDI	TDI	JTAG Data in, determined by JTAGEN control bit (DDPCON<3>)
TMS	TMS	TMS	TMS	TMS	JTAG Test Mode Select, determined by JTAGEN control bit (DDPCON<3>)
TRCLK	Alternate	Alternate	TRCLK	Alternate	Trace Clock, determined by TROEN control bit (DDPCON<2>)
TRD0	Alternate	Alternate	TRD0	Alternate	Trace Data, determined by TROEN control bit (DDPCON<2>)
TRD1	Alternate	Alternate	TRD1	Alternate	Trace Data, determined by TROEN control bit (DDPCON<2>)
TRD2	Alternate	Alternate	TRD2	Alternate	Trace Data, determined by TROEN control bit (DDPCON<2>)
TRD3	Alternate	Alternate	TRD3	Alternate	Trace Data, determined by TROEN control bit (DDPCON<2>)

33.6 OPERATION IN POWER-SAVING MODES

The PIC32MX must be awake for all programming and debugging operations.

33.7 EFFECTS OF RESETS

33.7.1 Device Reset

A device Reset ($\overline{MCLR}$) while in ICSP mode will force the ICSP to exit. An $\overline{MCLR}$ will force an exit from EJTAG mode.

33.7.2 Watchdog Timer Reset

A Watchdog Timer (WDT) Reset during Erase will not abort the Erase cycle. The WDT event flag will be set to show that a WDT Reset has occurred.

A WDT Reset during an EJTAG session will reset the TAP controller to the Microchip TAP controller.

A WDT Reset during Programming will abort the programming sequence.

33.8 APPLICATION IDEAS

For implementation of ICSP programming, refer the device Programming Specification.

33.9 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the PIC32MX device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the programming and diagnostics are:

Title	Application Note #
No related application notes at this time.	N/A

Note: Please visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the PIC32MX family of devices.

33.10 REVISION HISTORY

Revision A (September 2007)

This is the initial released version of this document.

Revision B (October 2007)

Updated document to remove Confidential status.

Revision C (April 2008)

Revised status to Preliminary; Revised U-0 to r-x.

Revision D (June 2008)

Add Note to Section 33.3.1; Revised Section 33.3.2.1; Revised Table 33-7; Change Reserved bits from “Maintain as” to “Write”.

NOTES:

Section 34. Reserved for Future

NOTES:

Section 35. Reserved for Future

NOTES:

INDEX

S

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://support.microchip.com
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo
Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

Toronto
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Hong Kong SAR
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-4182-8400
Fax: 91-80-4182-8422

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-572-9526
Fax: 886-3-572-6459

Taiwan - Kaohsiung
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan - Taipei
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820

01/02/08