Section 8. Interrupts

HIGHLIGHTS

This section of the manual contains the following topics:

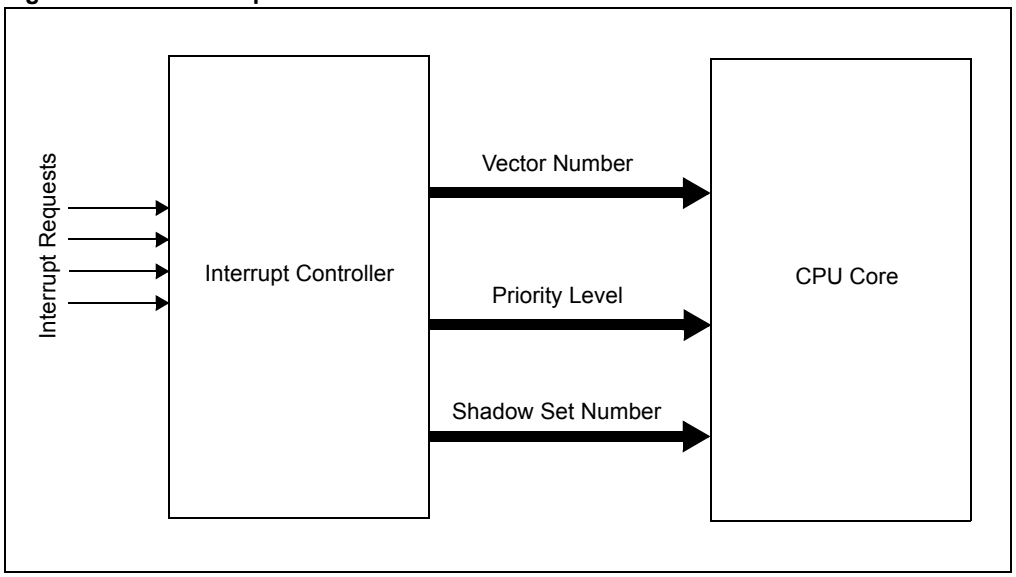
8.1 INTRODUCTION

The PIC32MX generates interrupt requests in response to interrupt events from peripheral modules. The Interrupt 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
- Five external interrupts with edge polarity control
- Interrupt proximity timer
- Module freeze in Debug mode
- Seven user-selectable priority levels for each vector
- Four user-selectable subpriority levels within each priority
- User-configurable shadow set based on priority level (this feature is not available on all devices; refer to the specific device data sheet for availability)
- 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. Refer to **Section 2. "MCU"** (DS61113) for more details.

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 sources, and depending on the device variant, the number of sources may be different. An 'x' used in the names of control/status bits and registers denotes that there are multiple registers, which have the same function, that can define these interrupt sources. Refer to the specific device data sheet for more details.

The Interrupts module consists of the following Special Function Registers (SFRs):

- INTCON: Interrupt Control Register
- INTSTAT: Interrupt Status Register
- TPTMR: Temporal Proximity Timer Register
- IFSx: Interrupt Flag Status Registers
- IECx: Interrupt Enable Control Registers
- IPCx: Interrupt Priority Control Registers

Table 8-1 provides a brief summary of the related Interrupts module registers. Corresponding registers appear after the summary, followed by a detailed description of each register.

Table 8-1: Interrupts Register Summary

Address Offset	Name	Bit Range	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
0x00	INTCON(1,2,3)	31:24	—	—	—	—	—	—	—	—
		23:16	—	—	—	—	—	—	—	SS0
		15:8	—	FRZ	—	MVEC	—	TPC<2:0>		
		7:0	—	—	—	INT4EP	INT3EP	INT2EP	INT1EP	INT0EP
0x10	INTSTAT(1,2,3)	31:24	—	—	—	—	—	—	—	—
		23:16	—	—	—	—	—	—	—	—
		15:8	—	—	—	—	—	RIPL<2:0>		
		7:0	—	—	VEC<5:0>					
0x20	TPTMR(1,2,3)	31:24	TPTMR<31:24>							
		23:16	TPTMR<23:16>							
		15:8	TPTMR<15:8>							
		7:0	TPTMR<7:0>							
0x30-0x50	IFSx(1,2,3)	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
0x60-0x80	IECx(1,2,3)	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
0x90-0x180	IPCx(1,2,3)	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>	

Legend: — = unimplemented, read as '0'. Address offset values are shown in hexadecimal.

Note 1: This register has an associated Clear register at an offset of 0x4 bytes. These registers have the same name with CLR appended to the end of the register name (e.g., INTCONCLR). Writing a '1' to any bit position in the Clear register will clear valid bits in the associated register. Reads from the Clear register should be ignored.

2: This register has an associated Set register at an offset of 0x8 bytes. These registers have the same name with SET appended to the end of the register name (e.g., INTCONSET). Writing a '1' to any bit position in the Set register will set valid bits in the associated register. Reads from the Set register should be ignored.

3: This register has an associated Invert register at an offset of 0xC bytes. These registers have the same name with INV appended to the end of the register name (e.g., INTCONINV). Writing a '1' to any bit position in the Invert register will invert valid bits in the associated register. Reads from the Invert register should be ignored.

Register 8-1: INTCON: Interrupt Control Register(1,2,3)

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/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

Note 1: This register has an associated Clear register (INTCONCLR) at an offset of 0x4 bytes. Writing a ‘1’ to any bit position in the Clear register will clear valid bits in the associated register. Reads from the Clear register should be ignored.

2: This register has an associated Set register (INTCONSET) at an offset of 0x8 bytes. Writing a ‘1’ to any bit position in the Set register will set valid bits in the associated register. Reads from the Set register should be ignored.

3: This register has an associated Invert register (INTCONINV) at an offset of 0xC bytes. Writing a ‘1’ to any bit position in the Invert register will invert valid bits in the associated register. Reads from the Invert register should be ignored.

Register 8-1: INTCON: Interrupt Control Register(1,2,3) (Continued)

bit 10-8 **TPC<2:0>:** Temporal Proximity Control bits
111 = Interrupts of group priority 7 or lower start the TP timer
110 = Interrupts of group priority 6 or lower start the TP timer
101 = Interrupts of group priority 5 or lower start the TP timer
100 = Interrupts of group priority 4 or lower start the TP timer
011 = Interrupts of group priority 3 or lower start the TP timer
010 = Interrupts of group priority 2 or lower start the TP timer
001 = Interrupts 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

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

Note 1: This register has an associated Clear register (INTCONCLR) at an offset of 0x4 bytes. Writing a '1' to any bit position in the Clear register will clear valid bits in the associated register. Reads from the Clear register should be ignored.

2: This register has an associated Set register (INTCONSET) at an offset of 0x8 bytes. Writing a '1' to any bit position in the Set register will set valid bits in the associated register. Reads from the Set register should be ignored.

3: This register has an associated Invert register (INTCONINV) at an offset of 0xC bytes. Writing a '1' to any bit position in the Invert register will invert valid bits in the associated register. Reads from the Invert register should be ignored.

Register 8-2: INTSTAT: Interrupt Status Register(1,2,3)

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-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<2:0>:** 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 7-6 **Reserved:** Write ‘0’; ignore read

bit 5-0 **VEC<5:0>:** 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.

Note 1: This register has an associated Clear register (INTSTATCLR) at an offset of 0x4 bytes. Writing a ‘1’ to any bit position in the Clear register will clear valid bits in the associated register. Reads from the Clear register should be ignored.

2: This register has an associated Set register (INTSTATSET) at an offset of 0x8 bytes. Writing a ‘1’ to any bit position in the Set register will set valid bits in the associated register. Reads from the Set register should be ignored.

3: This register has an associated Invert register (INTSTATINV) at an offset of 0xC bytes. Writing a ‘1’ to any bit position in the Invert register will invert valid bits in the associated register. Reads from the Invert register should be ignored.

Register 8-3: TPTMR: Temporal Proximity Timer Register(1,2,3)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
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<31:0>:** 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.

Note 1: This register has an associated Clear register (TPTMRCLR) at an offset of 0x4 bytes. Writing a ‘1’ to any bit position in the Clear register will clear valid bits in the associated register. Reads from the Clear register should be ignored.

2: This register has an associated Set register (TPTMRSET) at an offset of 0x8 bytes. Writing a ‘1’ to any bit position in the Set register will set valid bits in the associated register. Reads from the Set register should be ignored.

3: This register has an associated Invert register (TPTMRINV) at an offset of 0xC bytes. Writing a ‘1’ to any bit position in the Invert register will invert valid bits in the associated register. Reads from the Invert register should be ignored.

Register 8-4: IFSx: Interrupt Flag Status Register(1,2,3,4)

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 specific device data sheet to learn exact bit definitions.

2: These registers have an associated Clear register (IFSxCLR) at an offset of 0x4 bytes. Writing a ‘1’ to any bit position in the Clear register will clear valid bits in the associated register. Reads from the Clear register should be ignored.

3: These registers have an associated Set register (IFSxSET) at an offset of 0x8 bytes. Writing a ‘1’ to any bit position in the Set register will set valid bits in the associated register. Reads from the Set register should be ignored.

4: These registers have an associated Invert register (IFSxINV) at an offset of 0xC bytes. Writing a ‘1’ to any bit position in the Invert register will invert valid bits in the associated register. Reads from the Invert register should be ignored.

Register 8-5: IECx: Interrupt Enable Control Register(1,2,3,4)

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 IECx register. Refer to the **“Interrupts”** chapter in the specific device data sheet to learn exact bit definitions.

2: These registers have an associated Clear register (IECxCLR) at an offset of 0x4 bytes. Writing a ‘1’ to any bit position in the Clear register will clear valid bits in the associated register. Reads from the Clear register should be ignored.

3: These registers have an associated Set register (IECxSET) at an offset of 0x8 bytes. Writing a ‘1’ to any bit position in the Set register will set valid bits in the associated register. Reads from the Set register should be ignored.

4: These registers have an associated Invert register (IECxINV) at an offset of 0xC bytes. Writing a ‘1’ to any bit position in the Invert register will invert valid bits in the associated register. Reads from the Invert register should be ignored.

Register 8-6: IPCx: Interrupt Priority Control Register(1,2,3,4)

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

Note 1: This register represents a generic definition of the IPCx register. Refer to the **“Interrupts”** chapter in the specific device data sheet to learn exact bit definitions.

2: These registers have an associated Clear register (IPCxCLR) at an offset of 0x4 bytes. Writing a ‘1’ to any bit position in the Clear register will clear valid bits in the associated register. Reads from the Clear register should be ignored.

3: These registers have an associated Set register (IPCxSET) at an offset of 0x8 bytes. Writing a ‘1’ to any bit position in the Set register will set valid bits in the associated register. Reads from the Set register should be ignored.

4: These registers have an associated Invert register (IPCxINV) at an offset of 0xC bytes. Writing a ‘1’ to any bit position in the Invert register will invert valid bits in the associated register. Reads from the Invert register should be ignored.

Register 8-6: IPCx: Interrupt Priority Control Register(1,2,3,4) (Continued)

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

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 IPCx register. Refer to the **"Interrupts"** chapter in the specific device data sheet to learn exact bit definitions.

2: These registers have an associated Clear register (IPCxCLR) at an offset of 0x4 bytes. Writing a '1' to any bit position in the Clear register will clear valid bits in the associated register. Reads from the Clear register should be ignored.

3: These registers have an associated Set register (IPCxSET) at an offset of 0x8 bytes. Writing a '1' to any bit position in the Set register will set valid bits in the associated register. Reads from the Set register should be ignored.

4: These registers have an associated Invert register (IPCxINV) at an offset of 0xC bytes. Writing a '1' to any bit position in the Invert register will invert valid bits in the associated register. Reads from the Invert register should be ignored.

8.3 OPERATION

The interrupt controller is responsible for pre-processing interrupt requests (IRQs) from a number of on-chip peripherals and presenting them in the appropriate order to the processor.

Figure 8-2 depicts the interrupt process within the PIC32MX. The interrupt controller is designed to receive up to 96 IRQs from the processor core, on-chip peripherals capable of generating interrupts, and five external inputs. 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, the user can select a priority level to receive a dedicated shadow register set. 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 the 'E' and 'M' stages 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 the Exception Level (EXL) bit (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 (VEC) bits (INTSTAT<5:0>) and Requested Interrupt Priority (RIPL) bits (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.

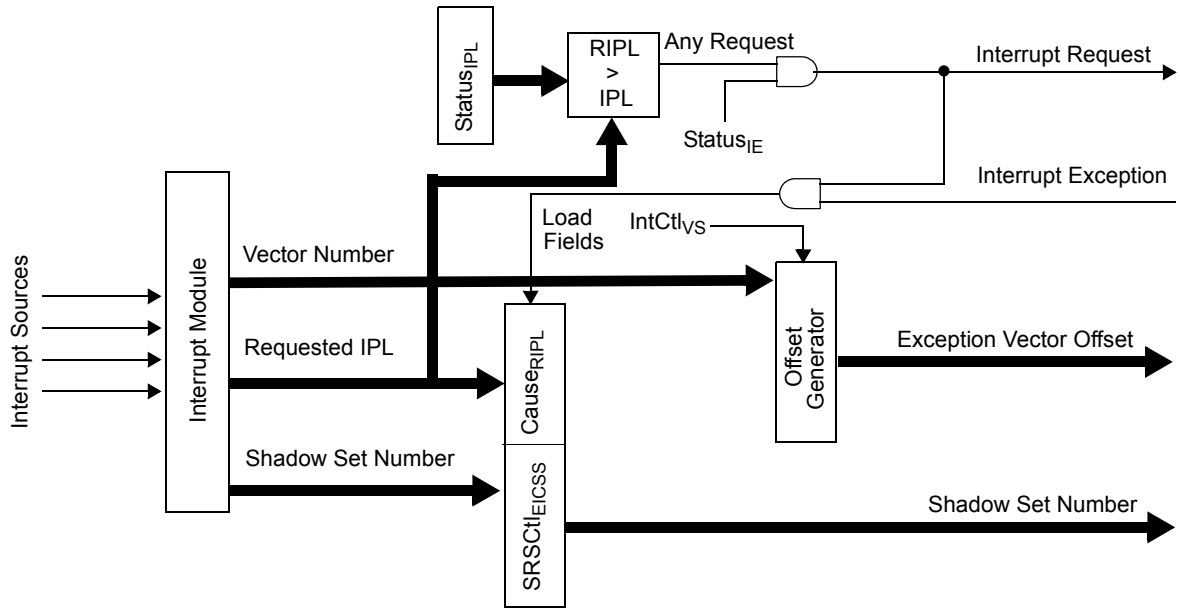
The PIC32MX interrupt controller can be configured to operate in one of two modes:

- 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. "MCU"** (DS61113).

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 the 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 the 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 (Exception Base) 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"** (DS61113) for more details.)

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 "Single Vector Mode"**), all interrupt vectors use the same vector address. When it is configured for Multi-Vectored mode (see **Section 8.5 "Multi-Vector Mode"**), 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"** (DS61113) for more details.) 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 "Single Vector Mode"** and **Section 8.5 "Multi-Vector Mode"**) 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 the 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

$$\textit{Single Vector Address} = \textit{EBase} + 0x200$$

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 the 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 (RIPL) bit of the Cause register (Cause<15:10>) into the Status register's Interrupt Priority (IPL) bits (Status<15:10>) before re-enabling interrupts. (The Cause and Status registers are located in the CPU; refer to **Section 2. "MCU"** (DS61113) for more details.) This action will disable all lower priority interrupts until the completion of the Interrupt Service Routine (ISR).

Note:	The Interrupt Service Routine 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 the 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"** (DS61113) 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 the Stack and Global Data Pointers from one set to another. Most C compilers supporting the PIC32MX family of devices 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, the SS0 (INTCON<16>) bit determines which register set will be used. If the SS0 bit is '1', the interrupt controller will instruct the CPU to use the second register set for all interrupts. If the SS0 bit 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 whether the second shadow set will be used at all.

8.8.2 Register Set Selection in Multi-Vector Mode

When a priority level interrupt matches a shadow set priority, the interrupt controller instructs the CPU to use the shadow set. For all other interrupt priorities, the interrupt controller instructs the CPU to use the primary register set. The interrupt priority that uses the shadow set will not need to perform any context save and restore. This results in increased code throughput and decreases interrupt latency.

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 (IPL) bits (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 (RIPL) bits (Cause<15:10>), Exception Level (EXL) bit and Error Level (ERL) bit 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 must first save the current priority and exception PC counter from Interrupt Priority (IPL) bits (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 (RIPL) bits (Cause<15:10>), Exception Level (EXL) bit and Error Level (ERL) bit 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"** (DS61113) for more details.)

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"** (DS61113) for more details.)

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
transition 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
```

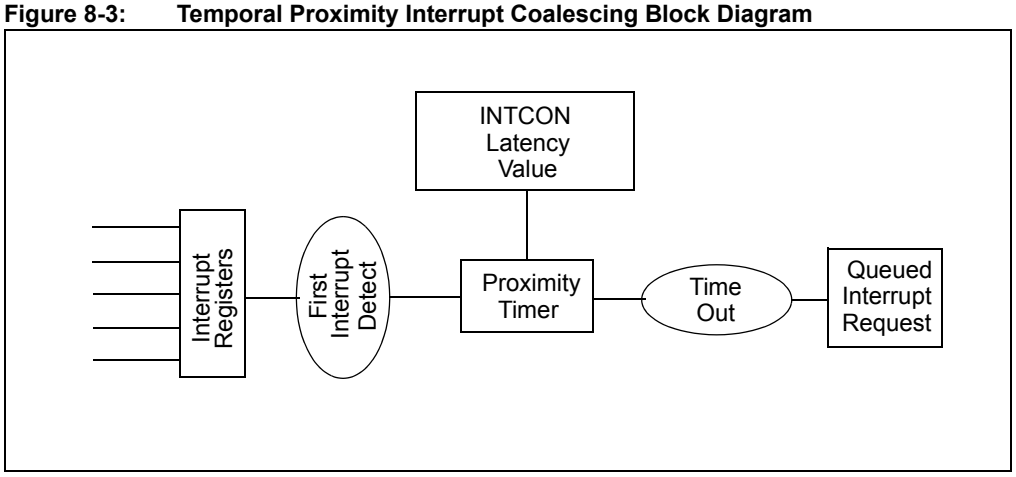
8.11 TEMPORAL PROXIMITY INTERRUPT COALESCING

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 multiple IRQs in a 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:

1. Set the TPC to the preferred priority level. (Setting TPC to zero will disable the proximity timer.)
2. 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 Power-on Reset.

8.12.3 Watchdog Timer Reset

All interrupt controller registers are forced to their reset states upon a Watchdog Timer Reset.

8.13 OPERATION IN POWER-SAVING AND DEBUG MODES

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 processor does not remain in sleep. It transitions to idle and then goes to run, once the TPT times out. 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: 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 and assign a non-zero priority in the IPS register, you will receive interrupt requests.

Question 2: ***When should I clear the interrupt request flag in my interrupt handler?***

Answer: You should clear the interrupt request flag after servicing the interrupt condition. For example, if a UART RX interrupt has occurred, the handler should read the RX buffer, and then clear the UART RX IRQ.

Question 3: ***After the proximity timer has counted down, which interrupt request is serviced?***

Answer: 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”.

Revision E (July 2009)

This revision includes the following updates:

- Minor updates to text and formatting have been implemented throughout the document
- Interrupts Register Summary (Table 8-1):
  - Removed all references to the Clear, Set and Invert registers
  - Added the Address Offset column
  - Added Notes 1, 2 and 3, which describe the Clear, Set and Invert registers
- Added Notes describing the Clear, Set and Invert registers to the following registers:
  - INTCON
  - INTSTAT
  - TPTMR
  - IFSx
  - IPCx
- Updated the note at the beginning of **Section 8.2 “Control Registers”**
- Updated the second sentence of the second paragraph in **Section 8.3 “Operation”** to clarify the IRQ sources
- Updated the first paragraph of **Section 8.8.2 “Register Set Selection in Multi-Vector Mode”**
- Updated the answer to Question 2 in **Section 8.14 “Design Tips”**