6 Peripheral Interrupt Expansion (PIE)

The peripheral interrupt expansion (PIE) block multiplexes numerous interrupt sources into a smaller set of interrupt inputs. The PIE block can support 96 individual interrupts that are grouped into blocks of eight. Each group is fed into one of 12 core interrupt lines (INT1 to INT12). Each of the 96 interrupts is supported by its own vector stored in a dedicated RAM block that you can modify. The CPU, upon servicing the interrupt, automatically fetches the appropriate interrupt vector. It takes nine CPU clock cycles to fetch the vector and save critical CPU registers. Therefore, the CPU can respond quickly to interrupt events. Prioritization of interrupts is controlled in hardware and software. Each individual interrupt can be enabled/disabled within the PIE block.

6.1 Overview of the PIE Controller

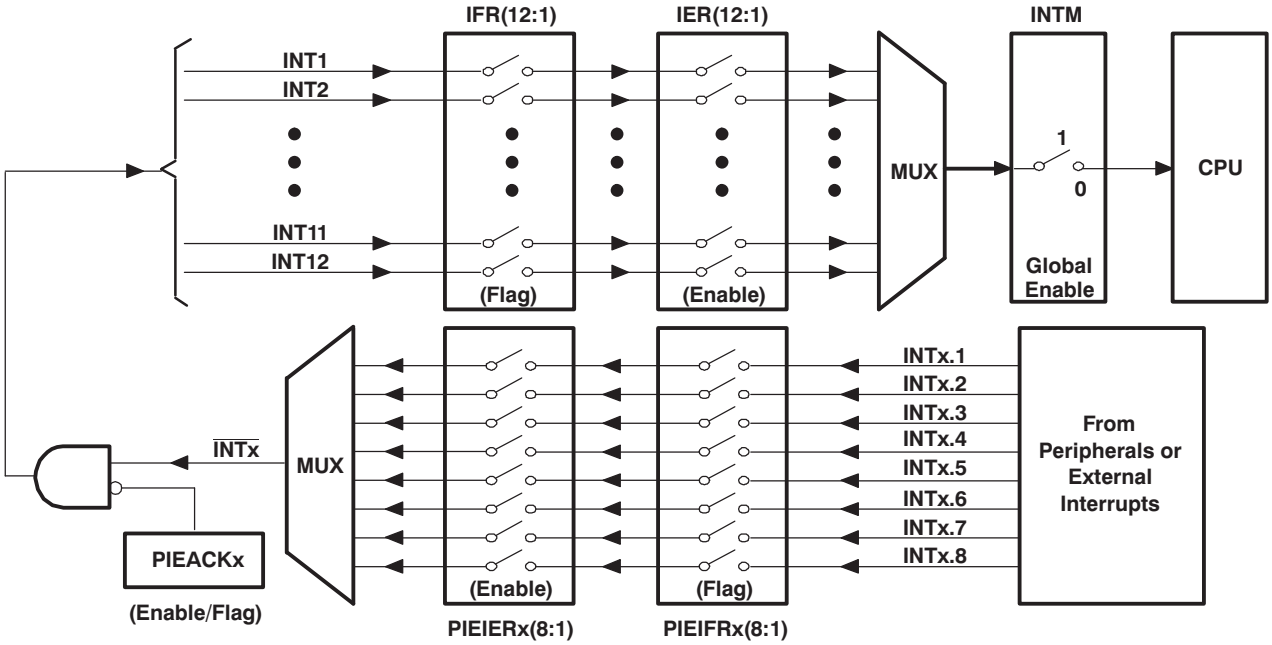
The 28x CPU supports one nonmaskable interrupt (NMI) and 16 maskable prioritized interrupt requests (INT1-INT14, RTOSINT, and DLOGINT) at the CPU level. The 28x devices have many peripherals and each peripheral is capable of generating one or more interrupts in response to many events at the peripheral level. Because the CPU does not have sufficient capacity to handle all peripheral interrupt requests at the CPU level, a centralized peripheral interrupt expansion (PIE) controller is required to arbitrate the interrupt requests from various sources such as peripherals and other external pins.

The PIE vector table is used to store the address (vector) of each interrupt service routine (ISR) within the system. There is one vector per interrupt source including all MUXed and nonMUXed interrupts. You populate the vector table during device initialization and you can update it during operation.

6.1.1 Interrupt Operation Sequence

Figure 76 shows an overview of the interrupt operation sequence for all multiplexed PIE interrupts. Interrupt sources that are not multiplexed are fed directly to the CPU.

Figure 76. Overview: Multiplexing of Interrupts Using the PIE Block



- **Peripheral Level**

  An interrupt-generating event occurs in a peripheral. The interrupt flag (IF) bit corresponding to that event is set in a register for that particular peripheral.

  If the corresponding interrupt enable (IE) bit is set, the peripheral generates an interrupt request to the PIE controller. If the interrupt is not enabled at the peripheral level, then the IF remains set until cleared by software. If the interrupt is enabled at a later time, and the interrupt flag is still set, the interrupt request is asserted to the PIE.

Interrupt flags within the peripheral registers must be manually cleared. See the peripheral reference guide for a specific peripheral for more information.

- **PIE Level**

  The PIE block multiplexes eight peripheral and external pin interrupts into one CPU interrupt. These interrupts are divided into 12 groups: PIE group 1 - PIE group 12. The interrupts within a group are multiplexed into one CPU interrupt. For example, PIE group 1 is multiplexed into CPU interrupt 1 (INT1) while PIE group 12 is multiplexed into CPU interrupt 12 (INT12). Interrupt sources connected to the remaining CPU interrupts are not multiplexed. For the nonmultiplexed interrupts, the PIE passes the request directly to the CPU.

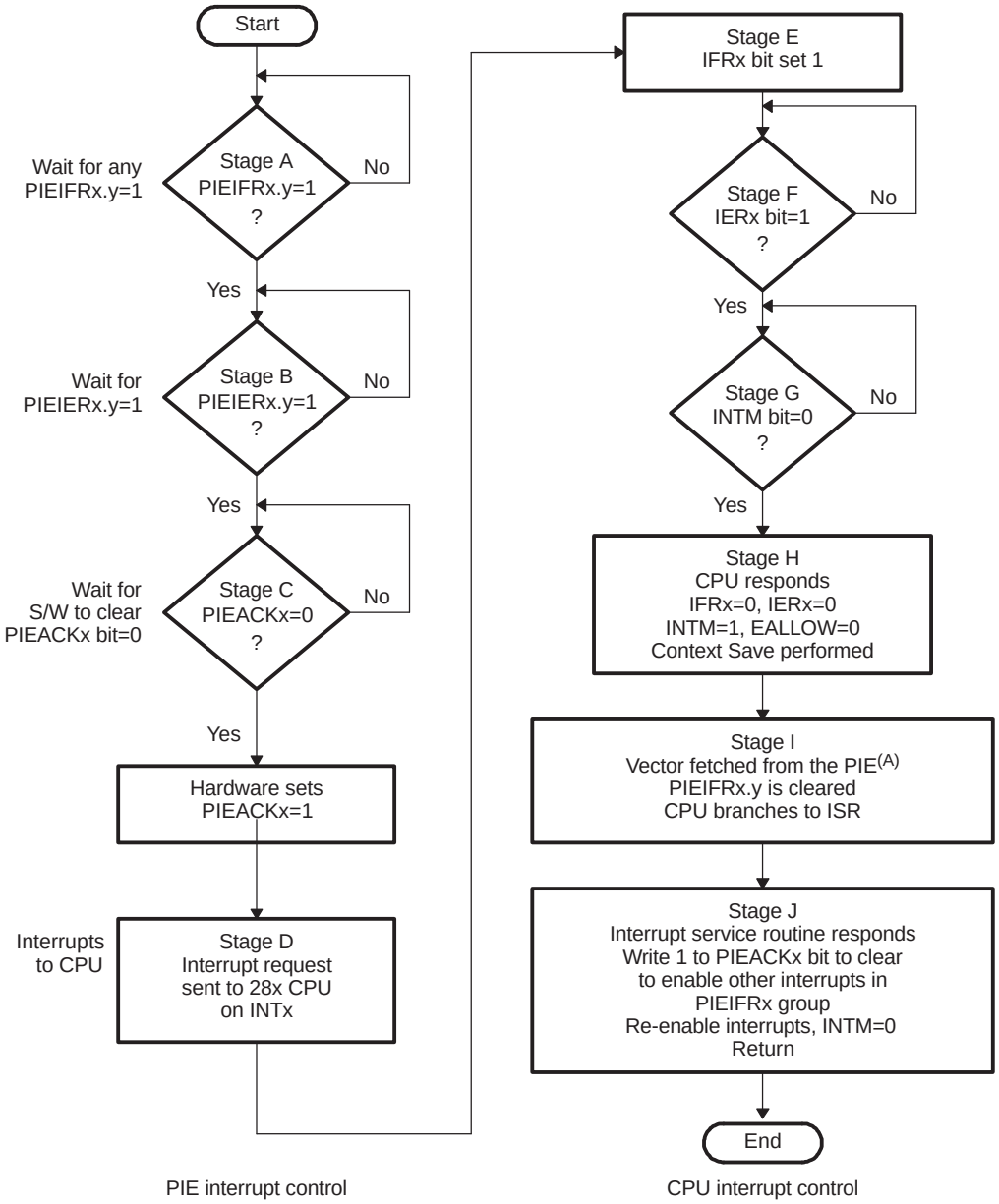
  For multiplexed interrupt sources, each interrupt group in the PIE block has an associated flag register (PIEIFRx) and enable (PIEIERx) register (x = PIE group 1 - PIE group 12). Each bit, referred to as y, corresponds to one of the 8 MUXed interrupts within the group. Thus PIEIFRx.y and PIEIERx.y correspond to interrupt y (y = 1-8) in PIE group x (x = 1-12). In addition, there is one acknowledge bit (PIEACK) for every PIE interrupt group referred to as PIEACKx (x = 1-12). Figure 77 illustrates the behavior of the PIE hardware under various PIEIFR and PIEIER register conditions.

  Once the request is made to the PIE controller, the corresponding PIE interrupt flag (PIEIFRx.y) bit is set. If the PIE interrupt enable (PIEIERx.y) bit is also set for the given interrupt then the PIE checks the corresponding PIEACKx bit to determine if the CPU is ready for an interrupt from that group. If the PIEACKx bit is clear for that group, then the PIE sends the interrupt request to the CPU. If PIEACKx is set, then the PIE waits until it is cleared to send the request for INTx. See Section 6.3 for details.

- **CPU Level**

  Once the request is sent to the CPU, the CPU level interrupt flag (IFR) bit corresponding to INTx is set. After a flag has been latched in the IFR, the corresponding interrupt is not serviced until it is appropriately enabled in the CPU interrupt enable (IER) register or the debug interrupt enable register (DBGIER) and the global interrupt mask (INTM) bit.

Figure 77. Typical PIE/CPU Interrupt Response - INTx.y



A For multiplexed interrupts, the PIE responds with the highest priority interrupt that is both flagged and enabled. If there is no interrupt both flagged and enabled, then the highest priority interrupt within the group (INTx.1 where x is the PIE group) is used. See Section Section 6.3.3 for details.

As shown in Table 105, the requirements for enabling the maskable interrupt at the CPU level depends on the interrupt handling process being used. In the standard process, which happens most of the time, the DBGIER register is not used. When the 28x is in real-time emulation mode and the CPU is halted, a different process is used. In this special case, the DBGIER is used and the INTM bit is ignored. If the DSP is in real-time mode and the CPU is running, the standard interrupt-handling process applies.

Table 105. Enabling Interrupt

Interrupt Handling Process	Interrupt Enabled If...
Standard	INTM = 0 and bit in IER is 1
DSP in real-time mode and halted	Bit in IER is 1 and DBGIER is 1

The CPU then prepares to service the interrupt. This preparation process is described in detail in *TMS320x28x DSP CPU and Instruction Set Reference Guide* (literature number SPRU430). In preparation, the corresponding CPU IFR and IER bits are cleared, EALLOW and LOOP are cleared, INTM and DBGM are set, the pipeline is flushed and the return address is stored, and the automatic context save is performed. The vector of the ISR is then fetched from the PIE module. If the interrupt request comes from a multiplexed interrupt, the PIE module uses the group PIEIERx and PIEIFRx registers to decode which interrupt needs to be serviced. This decode process is described in detail in Section Section 6.3.3.

The address for the interrupt service routine that is executed is fetched directly from the PIE interrupt vector table. There is one 32-bit vector for each of the possible 96 interrupts within the PIE. Interrupt flags within the PIE module (PIEIFRx.y) are automatically cleared when the interrupt vector is fetched. The PIE acknowledge bit for a given interrupt group, however, must be cleared manually when ready to receive more interrupts from the PIE group.

6.2 Vector Table Mapping

On 28xx devices, the interrupt vector table can be mapped to four distinct locations in memory. In practice only the PIE vector table mapping is used.

This vector mapping is controlled by the following mode bits/signals:

VMAP: VMAP is found in Status Register 1 ST1 (bit 3). A device reset sets this bit to 1. The state of this bit can be modified by writing to ST1 or by SETC/CLRC VMAP instructions. For normal operation leave this bit set.

M0M1MAP: M0M1MAP is found in Status Register 1 ST1 (bit 11). A device reset sets this bit to 1. The state of this bit can be modified by writing to ST1 or by SETC/CLRC M0M1MAP instructions. For normal 28xx device operation, this bit should remain set. M0M1MAP = 0 is reserved for TI testing only.

ENPIE: ENPIE is found in PIECTRL Register (bit 0). The default value of this bit, on reset, is set to 0 (PIE disabled). The state of this bit can be modified after reset by writing to the PIECTRL register (address 0x0000 0CE0).

Using these bits and signals the possible vector table mappings are shown in Table 106.

Table 106. Interrupt Vector Table Mapping

Vector MAPS	Vectors Fetched From	Address Range	VMAP	M0M1MAP	ENPIE
M1 Vector(1)	M1 SARAM Block	0x000000 - 0x00003F	0	0	X
M0 Vector(1)	M0 SARAM Block	0x000000 - 0x00003F	0	1	X
BROM Vector	Boot ROM Block	0x3FFFC0 - 0x3FFFFF	1	X	0
PIE Vector	PIE Block	0x000D00 - 0x000DFF	1	X	1

(1) Vector map M0 and M1 Vector is a reserved mode only. On the 28x devices these are used as SARAM.

The M1 and M0 vector table mapping are reserved for TI testing only. When using other vector mappings, the M0 and M1 memory blocks are treated as SARAM blocks and can be used freely without any restrictions.

After a device reset operation, the vector table is mapped as shown in Table 107.

Table 107. Vector Table Mapping After Reset Operation

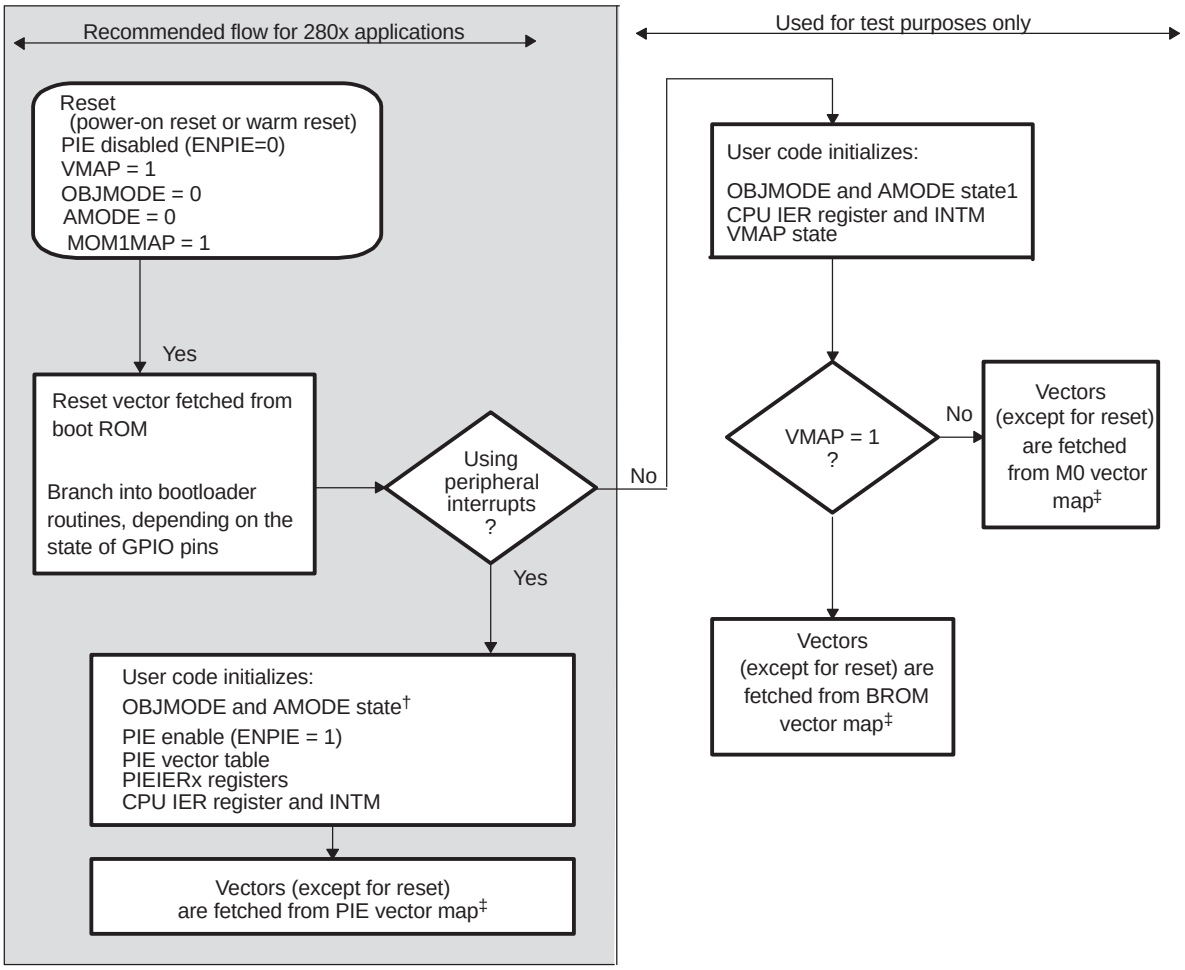
Vector MAPS	Reset Fetched From	Address Range	VMAP (1)	M0M1MAP (1)	ENPIE (1)
BROM Vector (2)	Boot ROM Block	0x3FFFC0 - 0x3FFFFF	1	1	0

(1) On the 28x devices, the VMAP and M0M1MAP modes are set to 1 on reset. The ENPIE mode is forced to 0 on reset.
(2) The reset vector is always fetched from the boot ROM.

After the reset and boot is complete, the PIE vector table should be initialized by the user's code. Then the application enables the PIE vector table. From that point on the interrupt vectors are fetched from the PIE vector table. Note: when a reset occurs, the reset vector is always fetched from the vector table as shown in Table 107. After a reset the PIE vector table is always disabled.

Figure 78 illustrates the process by which the vector table mapping is selected.

Figure 78. Reset Flow Diagram



A The compatibility operating mode of the 28x CPU is determined by a combination of the OBJMODE and AMODE bits in Status Register 1 (ST1):

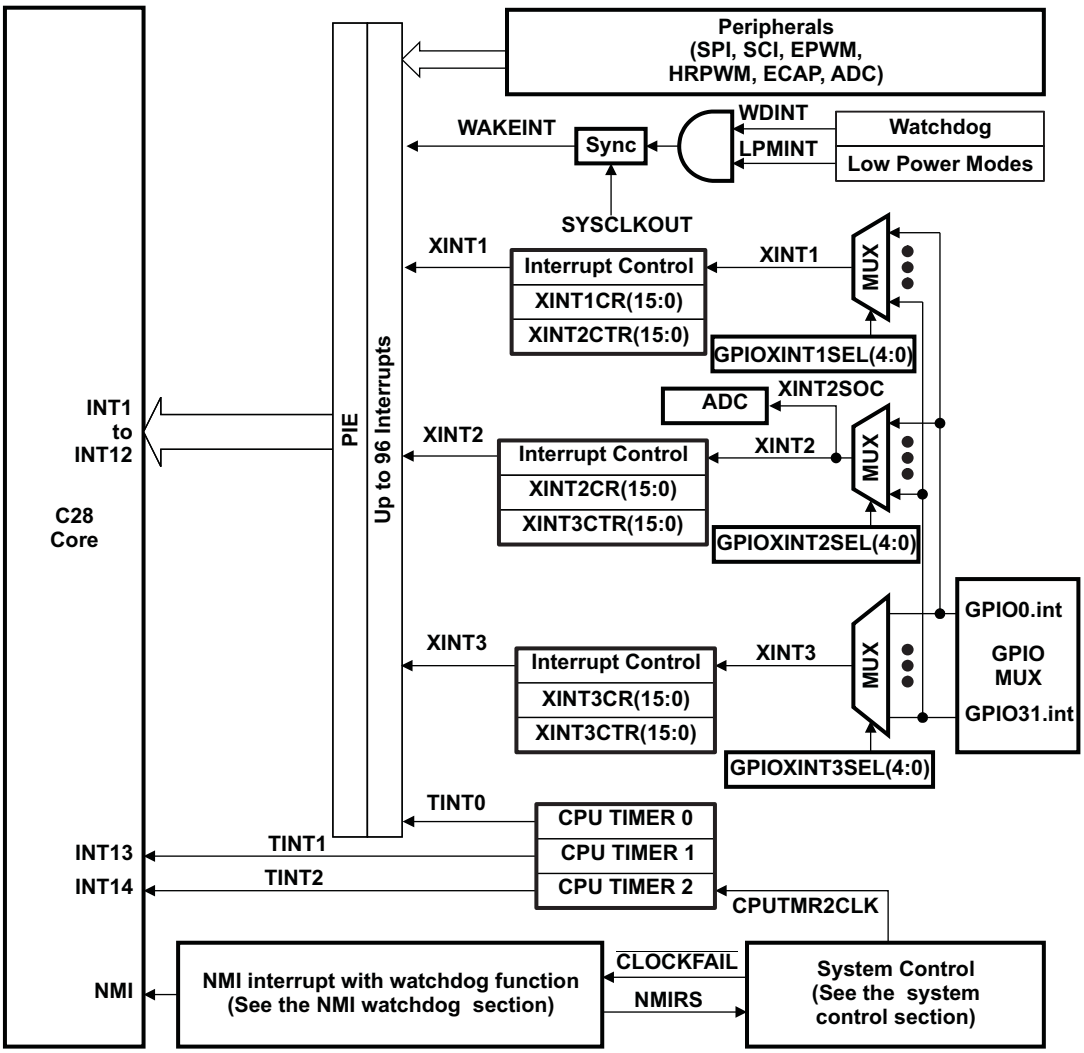
Operating Mode	OBJMODE	AMODE	
C28x Mode	1	0	
24x/240xA Source-Compatible	1	1	
C27x Object-Compatible	0	0	(Default at reset)

B The reset vector is always fetched from the boot ROM.

6.3 Interrupt Sources

Figure 79 shows how the various interrupt sources are multiplexed within the devices. This multiplexing (MUX) scheme may not be exactly the same on all 28x devices. See the data manual of your particular device for details.

Figure 79. PIE Interrupt Sources and External Interrupts XINT1/XINT2/XINT3



6.3.1 Procedure for Handling Multiplexed Interrupts

The PIE module multiplexes eight peripheral and external pin interrupts into one CPU interrupt. These interrupts are divided into 12 groups: PIE group 1 - PIE group 12. Each group has an associated enable PIEIER and flag PIEIFR register. These registers are used to control the flow of interrupts to the CPU. The PIE module also uses the PIEIER and PIEIFR registers to decode to which interrupt service routine the CPU should branch.

There are three main rules that should be followed when clearing bits within the PIEIFR and the PIEIER registers:

Rule 1: Never clear a PIEIFR bit by software

An incoming interrupt may be lost while a write or a read-modify-write operation to the PIEIFR register takes place. To clear a PIEIFR bit, the pending interrupt must be serviced. If you want to clear the PIEIFR bit without executing the normal service routine, then use the following procedure:

1. Set the EALLOW bit to allow modification to the PIE vector table.
2. Modify the PIE vector table so that the vector for the peripheral's service routine points to a temporary ISR. This temporary ISR will only perform a return from interrupt (IRET) operation.
3. Enable the interrupt so that the interrupt will be serviced by the temporary ISR.
4. After the temporary interrupt routine is serviced, the PIEIFR bit will be clear
5. Modify the PIE vector table to re-map the peripheral's service routine to the proper service routine.
6. Clear the EALLOW bit.

Rule 2: Procedure for software-prioritizing interrupts

Use the method found in the *C2833x C/C++ Header Files and Peripheral Examples in C* (literature number SPRC530).

(a) Use the CPU IER register as a global priority and the individual PIEIER registers for group priorities. In this case the PIEIER register is only modified within an interrupt. In addition, only the PIEIER for the same group as the interrupt being serviced is modified. This modification is done while the PIEACK bit holds additional interrupts back from the CPU.

(b) Never disable a PIEIER bit for a group when servicing an interrupt from an unrelated group.

Rule 3: Disabling interrupts using PIEIER

If the PIEIER registers are used to enable and then later disable an interrupt then the procedure described in Section 6.3.2 must be followed.

6.3.2 Procedures for Enabling And Disabling Multiplexed Peripheral Interrupts

The proper procedure for enabling or disabling an interrupt is by using the peripheral interrupt enable/disable flags. The primary purpose of the PIEIER and CPU IER registers is for software prioritization of interrupts within the same PIE interrupt group. The software package *C280x C/C++ Header Files and Peripheral Examples in C* (literature number SPRC191) includes an example that illustrates this method of software prioritizing interrupts.

Should bits within the PIEIER registers need to be cleared outside of this context, one of the following two procedures should be followed. The first method preserves the associated PIE flag register so that interrupts are not lost. The second method clears the associated PIE flag register.

Method 1: Use the PIEIERx register to disable the interrupt and preserve the associated PIEIFRx flags.

To clear bits within a PIEIERx register while preserving the associated flags in the PIEIFRx register, the following procedure should be followed:

Step a. Disable global interrupts (INTM = 1).
Step b. Clear the PIEIERx.y bit to disable the interrupt for a given peripheral. This can be done for one or more peripherals within the same group.
Step c. Wait 5 cycles. This delay is required to be sure that any interrupt that was incoming to the CPU has been flagged within the CPU IFR register.
Step d. Clear the CPU IFRx bit for the peripheral group. This is a safe operation on the CPU IFR register.
Step e. Clear the PIEACKx bit for the peripheral group.
Step f. Enable global interrupts (INTM = 0).

Method 2: Use the PIEIERx register to disable the interrupt and clear the associated PIEIFRx flags.

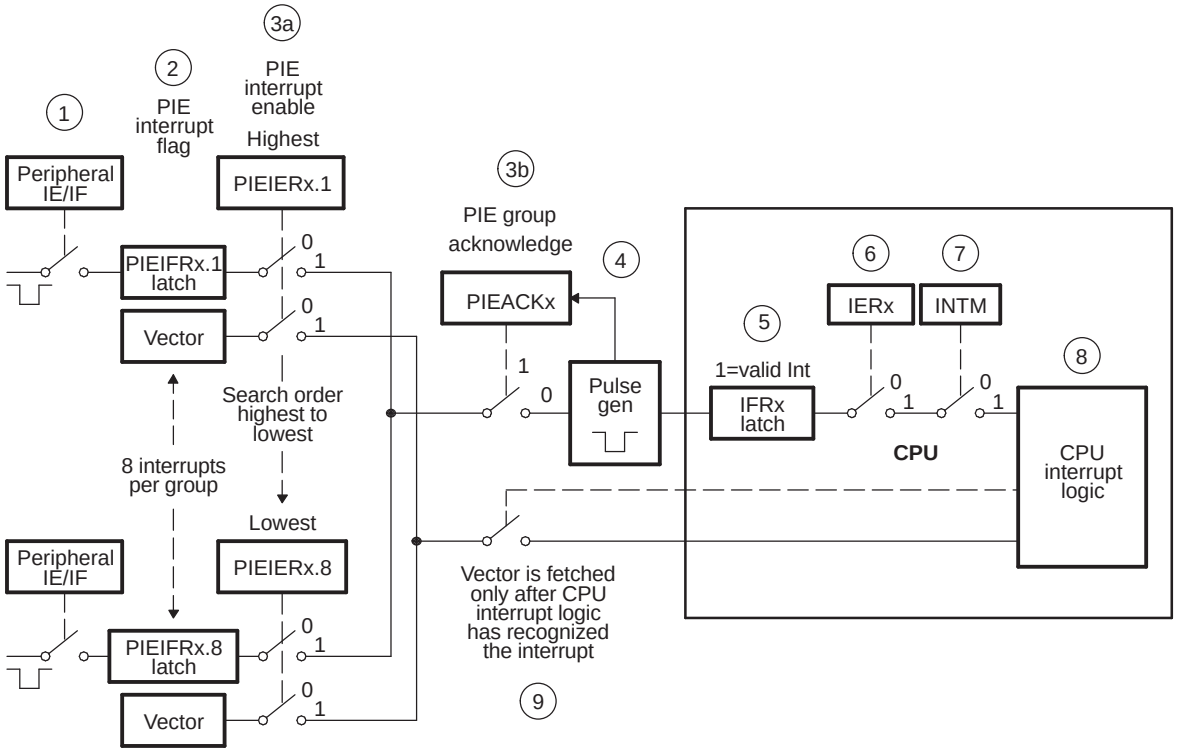
To perform a software reset of a peripheral interrupt and clear the associated flag in the PIEIFRx register and CPU IFR register, the following procedure should be followed:

Step 1. Disable global interrupts (INTM = 1).
Step 2. Set the EALLOW bit.
Step 3. Modify the PIE vector table to temporarily map the vector of the specific peripheral interrupt to a empty interrupt service routine (ISR). This empty ISR will only perform a return from interrupt (IRET) instruction. This is the safe way to clear a single PIEIFRx.y bit without losing any interrupts from other peripherals within the group.
Step 4. Disable the peripheral interrupt at the peripheral register.
Step 5. Enable global interrupts (INTM = 0).
Step 6. Wait for any pending interrupt from the peripheral to be serviced by the empty ISR routine.
Step 7. Disable global interrupts (INTM = 1).
Step 8. Modify the PIE vector table to map the peripheral vector back to its original ISR.
Step 9. Clear the EALLOW bit.
Step 10. Disable the PIEIER bit for given peripheral.
Step 11. Clear the IFR bit for given peripheral group (this is safe operation on CPU IFR register).
Step 12. Clear the PIEACK bit for the PIE group.
Step 13. Enable global interrupts.

6.3.3 Flow of a Multiplexed Interrupt Request From a Peripheral to the CPU

Figure 80 shows the flow with the steps shown in circled numbers. Following the diagram, the steps are described.

Figure 80. Multiplexed Interrupt Request Flow Diagram



Step 1. Any peripheral or external interrupt within the PIE group generates an interrupt. If interrupts are enabled within the peripheral module then the interrupt request is sent to the PIE module.

Step 2. The PIE module recognizes that interrupt y within PIE group x (INTx.y) has asserted an interrupt and the appropriate PIE interrupt flag bit is latched: PIEIFRx.y = 1.

Step 3. For the interrupt request to be sent from the PIE to the CPU, both of the following conditions must be true:

(a) The proper enable bit must be set (PIEIERx.y = 1) and
(b) The PIEACKx bit for the group must be clear.

Step 4. If both conditions in 3a and 3b are true, then an interrupt request is sent to the CPU and the acknowledge bit is again set (PIEACKx = 1). The PIEACKx bit will remain set until you clear it to indicate that additional interrupts from the group can be sent from the PIE to the CPU.

Step 5. The CPU interrupt flag bit is set (CPU IFRx = 1) to indicate a pending interrupt x at the CPU level.

Step 6. If the CPU interrupt is enabled (CPU IER bit x = 1, or DBGIER bit x = 1) AND the global interrupt mask is clear (INTM = 0) then the CPU will service the INTx.

Step 7. The CPU recognizes the interrupt and performs the automatic context save, clears the IER bit, sets INTM, and clears EALLOW. All of the steps that the CPU takes in order to prepare to service the interrupt are documented in the *TM S320C28x DSP CPU and Instruction Set Reference Guide* (literature number SPRU430).

Step 8. The CPU will then request the appropriate vector from the PIE.

Step 9. For multiplexed interrupts, the PIE module uses the current value in the PIEIERx and PIEIFRx registers to decode which vector address should be used. There are two possible cases:

(a) The vector for the highest priority interrupt within the group that is both enabled in the

PIEIERx register, and flagged as pending in the PIEIFRx is fetched and used as the branch address. In this manner if an even higher priority enabled interrupt was flagged after Step 7, it will be serviced first.

(b) If no flagged interrupts within the group are enabled, then the PIE will respond with the vector for the highest priority interrupt within that group. That is the branch address used for INTx.1. This behavior corresponds to the 28x TRAP or INT instructions.

NOTE: Because the PIEIERx register is used to determine which vector will be used for the branch, you must take care when clearing bits within the PIEIERx register. The proper procedure for clearing bits within a PIEIERx register is described in Section 6.3.2. Failure to follow these steps can result in changes occurring to the PIEIERx register after an interrupt has been passed to the CPU at Step 5 in Figure 6-5. In this case, the PIE will respond as if a TRAP or INT instruction was executed unless there are other interrupts both pending and enabled.

At this point, the PIEIFRx.y bit is cleared and the CPU branches to the vector of the interrupt fetched from the PIE.

6.3.4 The PIE Vector Table

The PIE vector table (see Table 109) consists of a 256 x 16 SARAM block that can also be used as RAM (in data space only) if the PIE block is not in use. The PIE vector table contents are undefined on reset. The CPU fixes interrupt priority for INT1 to INT12. The PIE controls priority for each group of eight interrupts. For example, if INT1.1 should occur simultaneously with INT8.1, both interrupts are presented to the CPU simultaneously by the PIE block, and the CPU services INT1.1 first. If INT1.1 should occur simultaneously with INT1.8, then INT1.1 is sent to the CPU first and then INT1.8 follows. Interrupt prioritization is performed during the vector fetch portion of the interrupt processing.

When the PIE is enabled, a TRAP #1 through TRAP #12 or an INTR INT1 to INTR INT12 instruction transfers program control to the interrupt service routine corresponding to the first vector within the PIE group. For example: TRAP #1 fetches the vector from INT1.1, TRAP #2 fetches the vector from INT2.1 and so forth. Similarly an OR IFR, #16-bit operation causes the vector to be fetched from INTR1.1 to INTR12.1 locations, if the respective interrupt flag is set. All other TRAP, INTR, OR IFR,#16-bit operations fetch the vector from the respective table location. The vector table is EALLOW protected.

Out of the 96 possible MUXed interrupts in Table 108, 43 interrupts are currently used. The remaining interrupts are reserved for future devices. These reserved interrupts can be used as software interrupts if they are enabled at the PIEIFRx level, provided none of the interrupts within the group is being used by a peripheral. Otherwise, interrupts coming from peripherals may be lost by accidentally clearing their flags when modifying the PIEIFR.

To summarize, there are two safe cases when the reserved interrupts can be used as software interrupts:

1. No peripheral within the group is asserting interrupts.
2. No peripheral interrupts are assigned to the group. For example, PIE group 11 and 12 do not have any peripherals attached to them.

The interrupt grouping for peripherals and external interrupts connected to the PIE module is shown in Table 108. Each row in the table shows the 8 interrupts multiplexed into a particular CPU interrupt. The entire PIE vector table, including both MUXed and non-MUXed interrupts, is shown in Table 109.

Table 108. PIE MUXed Peripheral Interrupt Vector Table

	INTx.8	INTx.7	INTx.6	INTx.5	INTx.4	INTx.3	INTx.2	INTx.1
INT1.y	WAKEINT	TINT0	ADCINT9	XINT2	XINT1	Reserved	ADCINT2	ADCINT1
	(LPM/WD)	(TIMER 0)	(ADC)	Ext. int. 2	Ext. int. 1	-	(ADC)	(ADC)
	0xD4E	0xD4C	0xD4A	0xD48	0xD46	0xD44	0xD42	0xD40
INT2.y	Reserved	Reserved	Reserved	Reserved	EPWM4_TZINT	EPWM3_TZINT	EPWM2_TZINT	EPWM1_TZINT
	-	-	-	-	(ePWM4)	(ePWM3)	(ePWM2)	(ePWM1)
	0xD5E	0xD5C	0xD5A	0xD58	0xD56	0xD54	0xD52	0xD50
INT3.y	Reserved	Reserved	Reserved	Reserved	EPWM4_INT	EPWM3_INT	EPWM2_INT	EPWM1_INT
	-	-	-	-	(ePWM4)	(ePWM3)	(ePWM2)	(ePWM1)
	0xD6E	0xD6C	0xD6A	0xD68	0xD66	0xD64	0xD62	0xD60

Table 108. PIE MUXed Peripheral Interrupt Vector Table (continued)

	INTx.8	INTx.7	INTx.6	INTx.5	INTx.4	INTx.3	INTx.2	INTx.1
INT4.y	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	ECAP1_INT
	-	-	-	-	-	-	-	(eCAP1)
	0xD7E	0xD7C	0xD7A	0xD78	0xD76	0xD74	0xD72	0xD70
INT5.y	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
	-	-	-	-	-	-	-	-
	0xD8E	0xD8C	0xD8A	0xD88	0xD86	0xD84	0xD82	0xD80
INT6.y	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	SPITXINTA	SPIRXINTA
	-	-	-	-	-	-	(SPI-A)	(SPI-A)
	0xD9E	0xD9C	0xD9A	0xD98	0xD96	0xD94	0xD92	0xD90
INT7.y	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
	-	-	-	-	-	-	-	-
	0xDAE	0xDAC	0xDAA	0xDA8	0xDA6	0xDA4	0xDA2	0xDA0
INT8.y	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	I2CINT2A	I2CINT1A
	-	-	-	-	-	-	(I^2C-A)	(I^2C-A)
	0xDBE	0xDBC	0xDBA	0xDB8	0xDB6	0xDB4	0xDB2	0xDB0
INT9.y	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	SCITXINTA	SCIRXINTA
	-	-	-	-	-	-	(SCI-A)	(SCI-A)
	0xDCE	0xDCC	0xDCA	0xDC8	0xDC6	0xDC4	0xDC2	0xDC0
INT10.y	ADCINT8	ADCINT7	ADCINT6	ADCINT5	ADCINT4	ADCINT3	ADCINT2	ADCINT1
	(ADC)	(ADC)	(ADC)	(ADC)	(ADC)	(ADC)	(ADC)	(ADC)
	0xDDE	0xDDC	0xDDA	0xDD8	0xDD6	0xDD4	0xDD2	0xDD0
INT11.y	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
	-	-	-	-	-	-	-	-
	0xDEE	0xDEC	0xDEA	0xDE8	0xDE6	0xDE4	0xDE2	0xDE0
INT12.y	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	XINT3
	-	-	-	-	-	-	-	Ext. Int. 3
	0xDFE	0xDFC	0xDFA	0xDF8	0xDF6	0xDF4	0xDF2	0xDF0

Table 109. PIE Vector Table

Name	VECTOR ID	Address (1)	Size (x16)	Description (2)	CPU Priority	PIE Group Priority
Reset	0	0x0000 0D00	2	Reset is always fetched from location 0x003F FFC0 in Boot ROM.	1 (highest)	-
INT1	1	0x0000 0D02	2	Not used. See PIE Group 1	5	-
INT2	2	0x0000 0D04	2	Not used. See PIE Group 2	6	-
INT3	3	0x0000 0D06	2	Not used. See PIE Group 3	7	-
INT4	4	0x0000 0D08	2	Not used. See PIE Group 4	8	-
INT5	5	0x0000 0D0A	2	Not used. See PIE Group 5	9	-
INT6	6	0x0000 0D0C	2	Not used. See PIE Group 6	10	-
INT7	7	0x0000 0D0E	2	Not used. See PIE Group 7	11	-
INT8	8	0x0000 0D10	2	Not used. See PIE Group 8	12	-
INT9	9	0x0000 0D12	2	Not used. See PIE Group 9	13	-
INT10	10	0x0000 0D14	2	Not used. See PIE Group 10	14	-
INT11	11	0x0000 0D16	2	Not used. See PIE Group 11	15	-
INT12	12	0x0000 0D18	2	Not used. See PIE Group 12	16	-
INT13	13	0x0000 0D1A	2	External Interrupt 13 (XINT13) or CPU-Timer1	17	-
INT14	14	0x0000 0D1C	2	CPU-Timer2 (for TI/RTOS use)	18	-
DATALOG	15	0x0000 0D1E	2	CPU Data Logging Interrupt	19 (lowest)	-
RTOSINT	16	0x0000 0D20	2	CPU Real-Time OS Interrupt	4	-

(1) Reset is always fetched from location 0x003F FFC0 in Boot ROM.
(2) All the locations within the PIE vector table are EALLOW protected.

Table 109. PIE Vector Table (continued)

Name	VECTOR ID	Address [1]	Size (x16)	Description [2]		CPU Priority	PIE Group Priority
EMUINT	17	0x0000 0D22	2	CPU Emulation Interrupt		2	-
NMI	18	0x0000 0D24	2	External Non-Maskable Interrupt		3	-
ILLEGAL	19	0x0000 0D26	2	Illegal Operation		-	-
USER1	20	0x0000 0D28	2	User-Defined Trap		-	-
USER2	21	0x0000 0D2A	2	User Defined Trap		-	-
USER3	22	0x0000 0D2C	2	User Defined Trap		-	-
USER4	23	0x0000 0D2E	2	User Defined Trap		-	-
USER5	24	0x0000 0D30	2	User Defined Trap		-	-
USER6	25	0x0000 0D32	2	User Defined Trap		-	-
USER7	26	0x0000 0D34	2	User Defined Trap		-	-
USER8	27	0x0000 0D36	2	User Defined Trap		-	-
USER9	28	0x0000 0D38	2	User Defined Trap		-	-
USER10	29	0x0000 0D3A	2	User Defined Trap		-	-
USER11	30	0x0000 0D3C	2	User Defined Trap		-	-
USER12	31	0x0000 0D3E	2	User Defined Trap		-	-
PIE Group 1 Vectors - MUXed into CPU INT1							
INT1.1	32	0x0000 0D40	2	ADCINT1	(ADC)	5	1 (highest)
INT1.2	33	0x0000 0D42	2	ADCINT2	(ADC)	5	2
INT1.3	34	0x0000 0D44	2	Reserved		5	3
INT1.4	35	0x0000 0D46	2	XINT1		5	4
INT1.5	36	0x0000 0D48	2	XINT2		5	5
INT1.6	37	0x0000 0D4A	2	ADCINT9	(ADC)	5	6
INT1.7	38	0x0000 0D4C	2	TINT0	(CPU-Timer0)	5	7
INT1.8	39	0x0000 0D4E	2	WAKEINT	(LPM/WD)	5	8 (lowest)
PIE Group 2 Vectors - MUXed into CPU INT2							
INT2.1	40	0x0000 0D50	2	EPWM1_TZINT	(EPWM1)	6	1 (highest)
INT2.2	41	0x0000 0D52	2	EPWM2_TZINT	(EPWM2)	6	2
INT2.3	42	0x0000 0D54	2	EPWM3_TZINT	(EPWM3)	6	3
INT2.4	43	0x0000 0D56	2	EPWM4_TZINT	(EPWM4)	6	4
INT2.5	44	0x0000 0D58	2	Reserved		6	5
INT2.6	45	0x0000 0D5A	2	Reserved		6	6
INT2.7	46	0x0000 0D5C	2	Reserved		6	7
INT2.8	47	0x0000 0D5E	2	Reserved		6	8 (lowest)
PIE Group 3 Vectors - MUXed into CPU INT3							
INT3.1	48	0x0000 0D60	2	EPWM1_INT	(EPWM1)	7	1 (highest)
INT3.2	49	0x0000 0D62	2	EPWM2_INT	(EPWM2)	7	2
INT3.3	50	0x0000 0D64	2	EPWM3_INT	(EPWM3)	7	3
INT3.4	51	0x0000 0D66	2	EPWM4_INT	(EPWM4)	7	4
INT3.5	52	0x0000 0D68	2	Reserved		7	5
INT3.6	53	0x0000 0D6A	2	Reserved		7	6
INT3.7	54	0x0000 0D6C	2	Reserved		7	7
INT3.8	55	0x0000 0D6E	2	Reserved	-	7	8 (lowest)
PIE Group 4 Vectors - MUXed into CPU INT4							
INT4.1	56	0x0000 0D70	2	ECAP1_INT	(ECAP1)	8	1 (highest)
INT4.2	57	0x0000 0D72	2	Reserved	-	8	2
INT4.3	58	0x0000 0D74	2	Reserved	-	8	3

Table 109. PIE Vector Table (continued)

Name	VECTOR ID	Address [1]	Size (x16)	Description [2]		CPU Priority	PIE Group Priority
INT4.4	59	0x0000 0D76	2	Reserved	-	8	4
INT4.5	60	0x0000 0D78	2	Reserved	-	8	5
INT4.6	61	0x0000 0D7A	2	Reserved	-	8	6
INT4.7	62	0x0000 0D7C	2	Reserved	-	8	7
INT4.8	63	0x0000 0D7E	2	Reserved	-	8	8 (lowest)
PIE Group 5 Vectors - MUXed into CPU INT5							
INT5.1	64	0x0000 0D80	2	EQEP1_INT	(EQEP1)	9	1 (highest)
INT5.2	65	0x0000 0D82	2	Reserved	(EQEP2)	9	2
INT5.3	66	0x0000 0D84	2	Reserved		9	3
INT5.4	67	0x0000 0D86	2	Reserved	-	9	4
INT5.5	68	0x0000 0D88	2	Reserved	-	9	5
INT5.6	69	0x0000 0D8A	2	Reserved	-	9	6
INT5.7	70	0x0000 0D8C	2	Reserved	-	9	7
INT5.8	71	0x0000 0D8E	2	Reserved	-	9	8 (lowest)
PIE Group 6 Vectors - MUXed into CPU INT6							
INT6.1	72	0x0000 0D90	2	SPIRXINTA	(SPI-A)	10	1 (highest)
INT6.2	73	0x0000 0D92	2	SPITXINTA	(SPI-A)	10	2
INT6.3	74	0x0000 0D94	2	Reserved		10	3
INT6.4	75	0x0000 0D96	2	Reserved		10	4
INT6.5	76	0x0000 0D98	2	Reserved		10	5
INT6.6	77	0x0000 0D9A	2	Reserved		10	6
INT6.7	78	0x0000 0D9C	2	Reserved		10	7
INT6.8	79	0x0000 0D9E	2	Reserved		10	8 (lowest)
PIE Group 7 Vectors - MUXed into CPU INT7							
INT7.1	80	0x0000 0DA0	2	Reserved	-	11	1 (highest)
INT7.2	81	0x0000 0DA2	2	Reserved	-	11	2
INT7.3	82	0x0000 0DA4	2	Reserved	-	11	3
INT7.4	83	0x0000 0DA6	2	Reserved	-	11	4
INT7.5	84	0x0000 0DA8	2	Reserved	-	11	5
INT7.6	85	0x0000 0DAA	2	Reserved	-	11	6
INT7.7	86	0x0000 0DAC	2	Reserved	-	11	7
INT7.8	87	0x0000 0DAE	2	Reserved	-	11	8 (lowest)
PIE Group 8 Vectors - MUXed into CPU INT8							
INT8.1	88	0x0000 0DB0	2	I2CINT1A	(I^2C-A)	12	1 (highest)
INT8.2	89	0x0000 0DB2	2	I2CINT2A	(I^2C-A)	12	2
INT8.3	90	0x0000 0DB4	2	Reserved	-	12	3
INT8.4	91	0x0000 0DB6	2	Reserved	-	12	4
INT8.5	92	0x0000 0DB8	2	Reserved	-	12	5
INT8.6	93	0x0000 0DBA	2	Reserved	-	12	6
INT8.7	94	0x0000 0DBC	2	Reserved	-	12	7
INT8.8	95	0x0000 0DBE	2	Reserved	-	12	8 (lowest)
PIE Group 9 Vectors - MUXed into CPU INT9							
INT9.1	96	0x0000 0DC0	2	SCIRXINTA	(SCI-A)	13	1 (highest)
INT9.2	97	0x0000 0DC2	2	SCITXINTA	(SCI-A)	13	2
INT9.3	98	0x0000 0DC4	2	Reserved		13	3
INT9.4	99	0x0000 0DC6	2	Reserved		13	4
INT9.5	100	0x0000 0DC8	2	Reserved		13	5

Table 109. PIE Vector Table (continued)

Name	VECTOR ID	Address [1]	Size (x16)	Description [2]		CPU Priority	PIE Group Priority
INT9.6	101	0x0000 0DCA	2	Reserved		13	6
INT9.7	102	0x0000 0DCC	2	Reserved	-	13	7
INT9.8	103	0x0000 0DCE	2	Reserved	-	13	8 (lowest)
PIE Group 10 Vectors - MUXed into CPU INT10							
INT10.1	104	0x0000 0DD0	2	ADCINT1	(ADC)	14	1 (highest)
INT10.2	105	0x0000 0DD2	2	ADCINT2	(ADC)	14	2
INT10.3	106	0x0000 0DD4	2	ADCINT3	(ADC)	14	3
INT10.4	107	0x0000 0DD6	2	ADCINT4	(ADC)	14	4
INT10.5	108	0x0000 0DD8	2	ADCINT5	(ADC)	14	5
INT10.6	109	0x0000 0DDA	2	ADCINT6	(ADC)	14	6
INT10.7	110	0x0000 0DDC	2	ADCINT7	(ADC)	14	7
INT10.8	111	0x0000 0DDE	2	ADCINT8	(ADC)	14	8 (lowest)
PIE Group 11 Vectors - MUXed into CPU INT11							
INT11.1	112	0x0000 0DE0	2	Reserved	-	15	1 (highest)
INT11.2	113	0x0000 0DE2	2	Reserved	-	15	2
INT11.3	114	0x0000 0DE4	2	Reserved	-	15	3
INT11.4	115	0x0000 0DE6	2	Reserved	-	15	4
INT11.5	116	0x0000 0DE8	2	Reserved	-	15	5
INT11.6	117	0x0000 0DEA	2	Reserved	-	15	6
INT11.7	118	0x0000 0DEC	2	Reserved	-	15	7
INT11.8	119	0x0000 0DEE	2	Reserved	-	15	8 (lowest)
PIE Group 12 Vectors - Muxed into CPU INT12							
INT12.1	120	0x0000 0DF0	2	XINT3	-	16	1 (highest)
INT12.2	121	0x0000 0DF2	2	Reserved	-	16	2
INT12.3	122	0x0000 0DF4	2	Reserved	-	16	3
INT12.4	123	0x0000 0DF6	2	Reserved	-	16	4
INT12.5	124	0x0000 0DF8	2	Reserved	-	16	5
INT12.6	125	0x0000 0DFA	2	Reserved	-	16	6
INT12.7	126	0x0000 0DFC	2	Reserved	-	16	7
INT12.8	127	0x0000 0DFE	2	Reserved	-	16	8 (lowest)

6.4 PIE Configuration Registers

The registers controlling the functionality of the PIE block are shown in Table 110.

Table 110. PIE Configuration and Control Registers

Name	Address	Size (x16)	Description
PIECTRL	0x0000 - 0CE0	1	PIE, Control Register
PIEACK	0x0000 - 0CE1	1	PIE, Acknowledge Register
PIEIER1	0x0000 - 0CE2	1	PIE, INT1 Group Enable Register
PIEIFR1	0x0000 - 0CE3	1	PIE, INT1 Group Flag Register
PIEIER2	0x0000 - 0CE4	1	PIE, INT2 Group Enable Register
PIEIFR2	0x0000 - 0CE5	1	PIE, INT2 Group Flag Register
PIEIER3	0x0000 - 0CE6	1	PIE, INT3 Group Enable Register
PIEIFR3	0x0000 - 0CE7	1	PIE, INT3 Group Flag Register
PIEIER4	0x0000 - 0CE8	1	PIE, INT4 Group Enable Register
PIEIFR4	0x0000 - 0CE9	1	PIE, INT4 Group Flag Register
PIEIER5	0x0000 - 0CEA	1	PIE, INT5 Group Enable Register
PIEIFR5	0x0000 - 0CEB	1	PIE, INT5 Group Flag Register
PIEIER6	0x0000 - 0CEC	1	PIE, INT6 Group Enable Register
PIEIFR6	0x0000 - 0CED	1	PIE, INT6 Group Flag Register
PIEIER7	0x0000 - 0CEE	1	PIE, INT7 Group Enable Register
PIEIFR7	0x0000 - 0CEF	1	PIE, INT7 Group Flag Register
PIEIER8	0x0000 - 0CF0	1	PIE, INT8 Group Enable Register
PIEIFR8	0x0000 - 0CF1	1	PIE, INT8 Group Flag Register
PIEIER9	0x0000 - 0CF2	1	PIE, INT9 Group Enable Register
PIEIFR9	0x0000 - 0CF3	1	PIE, INT9 Group Flag Register
PIEIER10	0x0000 - 0CF4	1	PIE, INT10 Group Enable Register
PIEIFR10	0x0000 - 0CF5	1	PIE, INT10 Group Flag Register
PIEIER11	0x0000 - 0CF6	1	PIE, INT11 Group Enable Register
PIEIFR11	0x0000 - 0CF7	1	PIE, INT11 Group Flag Register
PIEIER12	0x0000 - 0CF8	1	PIE, INT12 Group Enable Register
PIEIFR12	0x0000 - 0CF9	1	PIE, INT12 Group Flag Register

6.5 PIE Interrupt Registers

Figure 81. PIECTRL Register (Address 0xCE0)

15	1	0
PIEVECT		ENPIE
R-0		R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 111. PIECTRL Register Address Field Descriptions

Bits	Field	Value	Description
15-1	PIEVECT		These bits indicate the address within the PIE vector table from which the vector was fetched. The least significant bit of the address is ignored and only bits 1 to 15 of the address is shown. You can read the vector value to determine which interrupt generated the vector fetch. **For Example:** If PIECTRL = 0x0D27 then the vector from address 0x0D26 (illegal operation) was fetched.
0	ENPIE		Enable vector fetching from PIE vector table. **Note:** The reset vector is never fetched from the PIE, even when it is enabled. This vector is always fetched from boot ROM.
		0	If this bit is set to 0, the PIE block is disabled and vectors are fetched from the CPU vector table in boot ROM. All PIE block registers (PIEACK, PIEIFR, PIEIER) can be accessed even when the PIE block is disabled.
		1	When ENPIE is set to 1, all vectors, except for reset, are fetched from the PIE vector table. The reset vector is always fetched from the boot ROM.

Figure 82. PIE Interrupt Acknowledge Register (PIEACK) Register (Address 0xCE1)

15 12	11 0
Reserved	PIEACK
R-0	R/W1C-1

LEGEND: R/W1C = Read/Write 1 to clear; R = Read only; -*n* = value after reset

Table 112. PIE Interrupt Acknowledge Register (PIEACK) Field Descriptions

Bits	Field	Value	Description
15-12	Reserved		Reserved
11-0	PIEACK		Each bit in PIEACK refers to a specific PIE group. Bit 0 refers to interrupts in PIE group 1 that are MUXed into $\overline{\text{INT1}}$ up to Bit 11, which refers to PIE group 12 which is MUXed into CPU $\overline{\text{INT12}}$
		bit x = 0 [1]	If a bit reads as a 0, it indicates that the PIE can send an interrupt from the respective group to the CPU. Writes of 0 are ignored.
		bit x = 1	Reading a 1 indicates if an interrupt from the respective group has been sent to the CPU and all other interrupts from the group are currently blocked. Writing a 1 to the respective interrupt bit clears the bit and enables the PIE block to drive a pulse into the CPU interrupt input if an interrupt is pending for that group.

(1) bit x = PIEACK bit 0 - PIEACK bit 11. Bit 0 refers to CPU $\overline{\text{INT1}}$ up to Bit 11, which refers to CPU $\overline{\text{INT12}}$

6.5.1 PIE Interrupt Flag Registers

There are twelve PIEIFR registers, one for each CPU interrupt used by the PIE module (INT1-INT12).

Figure 83. PIEIFRx Register (x = 1 to 12)

15							8
Reserved							
R-0							
7	6	5	4	3	2	1	0
INTx.8	INTx.7	INTx.6	INTx.5	INTx.4	INTx.3	INTx.2	INTx.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

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 113. PIEIFRx Register Field Descriptions

Bits	Field	Description
15-8	Reserved	Reserved
7	INTx.8	These register bits indicate whether an interrupt is currently active. They behave very much like the CPU interrupt flag register. When an interrupt is active, the respective register bit is set. The bit is cleared when the interrupt is serviced or by writing a 0 to the register bit. This register can also be read to determine which interrupts are active or pending. x = 1 to 12. INTx means CPU INT1 to INT12
6	INTx.7	
5	INTx.6	
4	INTx.5	The PIEIFR register bit is cleared during the interrupt vector fetch portion of the interrupt processing.
3	INTx.4	Hardware has priority over CPU accesses to the PIEIFR registers.
2	INTx.3	
1	INTx.2	
0	INTx.1	

NOTE: Never clear a PIEIFR bit. An interrupt may be lost during the read-modify-write operation. See Section Section 6.3.1 for a method to clear flagged interrupts.

6.5.2 PIE Interrupt Enable Registers

There are twelve PIEIER registers, one for each CPU interrupt used by the PIE module (INT1-INT12).

Figure 84. PIEIERx Register (x = 1 to 12)

15							8
Reserved							
R-0							
7	6	5	4	3	2	1	0
INTx.8	INTx.7	INTx.6	INTx.5	INTx.4	INTx.3	INTx.2	INTx.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

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 114. PIEIERx Register (x = 1 to 12) Field Descriptions

Bits	Field	Description
15-8	Reserved	Reserved

Table 114. PIEIERx Register (x = 1 to 12) Field Descriptions (continued)

Bits	Field	Description
7	INTx.8	These register bits individually enable an interrupt within a group and behave very much like the core interrupt enable register. Setting a bit to 1 enables the servicing of the respective interrupt. Setting a bit to 0 disables the servicing of the interrupt. x = 1 to 12. INTx means CPU INT1 to INT12
6	INTx.7	
5	INTx.6	
4	INTx.5	
3	INTx.4	
2	INTx.3	
1	INTx.2	
0	INTx.1	

NOTE: Care must be taken when clearing PIEIER bits during normal operation. See Section Section 6.3.2 for the proper procedure for handling these bits.

6.5.3 CPU Interrupt Flag Register (IFR)

The CPU interrupt flag register (IFR), is a 16-bit, CPU register and is used to identify and clear pending interrupts. The IFR contains flag bits for all the maskable interrupts at the CPU level (INT1-INT14, DLOGINT and RTOSINT). When the PIE is enabled, the PIE module multiplexes interrupt sources for INT1-INT12.

When a maskable interrupt is requested, the flag bit in the corresponding peripheral control register is set to 1. If the corresponding mask bit is also 1, the interrupt request is sent to the CPU, setting the corresponding flag in the IFR. This indicates that the interrupt is pending or waiting for acknowledgment.

To identify pending interrupts, use the PUSH IFR instruction and then test the value on the stack. Use the OR IFR instruction to set IFR bits and use the AND IFR instruction to manually clear pending interrupts. All pending interrupts are cleared with the AND IFR #0 instruction or by a hardware reset.

The following events also clear an IFR flag:

- The CPU acknowledges the interrupt.
- The 28x device is reset.

NOTE:

1. To clear a CPU IFR bit, you must write a zero to it, not a one.
2. When a maskable interrupt is acknowledged, only the IFR bit is cleared automatically. The flag bit in the corresponding peripheral control register is not cleared. If an application requires that the control register flag be cleared, the bit must be cleared by software.
3. When an interrupt is requested by an INTR instruction and the corresponding IFR bit is set, the CPU does not clear the bit automatically. If an application requires that the IFR bit be cleared, the bit must be cleared by software.
4. IMR and IFR registers pertain to core-level interrupts. All peripherals have their own interrupt mask and flag bits in their respective control/configuration registers. Note that several peripheral interrupts are grouped under one core-level interrupt.

Figure 85. Interrupt Flag Register (IFR) — CPU Register

15	14	13	12	11	10	9	8
RTOSINT	DLOGINT	INT14	INT13	INT12	INT11	INT10	INT9
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
7	6	5	4	3	2	1	0
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 115. Interrupt Flag Register (IFR) — CPU Register Field Descriptions

Bits	Field	Value	Description
15	RTOSINT		Real-time operating system flag. RTOSINT is the flag for RTOS interrupts.
		0	No RTOS interrupt is pending
		1	At least one RTOS interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
14	DLOGINT		Data logging interrupt fag. DLOGINT is the flag for data logging interrupts.
		0	No DLOGINT is pending
		1	At least one DLOGINT interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
13	INT14		Interrupt 14 flag. INT14 is the flag for interrupts connected to CPU interrupt level INT14.
		0	No INT14 interrupt is pending
		1	At least one INT14 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
12	INT13		Interrupt 13 flag. INT13 is the flag for interrupts connected to CPU interrupt level INT13I.
		0	No INT13 interrupt is pending
		1	At least one INT13 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
11	INT12		Interrupt 12 flag. INT12 is the flag for interrupts connected to CPU interrupt level INT12.
		0	No INT12 interrupt is pending
		1	At least one INT12 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
10	INT11		Interrupt 11 flag. INT11 is the flag for interrupts connected to CPU interrupt level INT11.
		0	No INT11 interrupt is pending
		1	At least one INT11 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
9	INT10		Interrupt 10 flag. INT10 is the flag for interrupts connected to CPU interrupt level INT10.
		0	No INT10 interrupt is pending
		1	At least one INT6 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
8	INT9		Interrupt 9 flag. INT9 is the flag for interrupts connected to CPU interrupt level INT6.
		0	No INT9 interrupt is pending
		1	At least one INT9 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
7	INT8		Interrupt 8 flag. INT8 is the flag for interrupts connected to CPU interrupt level INT6.
		0	No INT8 interrupt is pending
		1	At least one INT8 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
6	INT7		Interrupt 7 flag. INT7 is the flag for interrupts connected to CPU interrupt level INT7.
		0	No INT7 interrupt is pending
		1	At least one INT7 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request

Table 115. Interrupt Flag Register (IFR) — CPU Register Field Descriptions (continued)

Bits	Field	Value	Description
5	INT6		Interrupt 6 flag. INT6 is the flag for interrupts connected to CPU interrupt level INT6.
		0	No INT6 interrupt is pending
		1	At least one INT6 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
4	INT5		Interrupt 5 flag. INT5 is the flag for interrupts connected to CPU interrupt level INT5.
		0	No INT5 interrupt is pending
		1	At least one INT5 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
3	INT4		Interrupt 4 flag. INT4 is the flag for interrupts connected to CPU interrupt level INT4.
		0	No INT4 interrupt is pending
		1	At least one INT4 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
2	INT3		Interrupt 3 flag. INT3 is the flag for interrupts connected to CPU interrupt level INT3.
		0	No INT3 interrupt is pending
		1	At least one INT3 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
1	INT2		Interrupt 2 flag. INT2 is the flag for interrupts connected to CPU interrupt level INT2.
		0	No INT2 interrupt is pending
		1	At least one INT2 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
0	INT1		Interrupt 1 flag. INT1 is the flag for interrupts connected to CPU interrupt level INT1.
		0	No INT1 interrupt is pending
		1	At least one INT1 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request

6.5.4 Interrupt Enable Register (IER) and Debug Interrupt Enable Register (DBGIER)

The IER is a 16-bit CPU register. The IER contains enable bits for all the maskable CPU interrupt levels (INT1-INT14, RTOSINT and DLOGINT). Neither NMI nor XRS is included in the IER; thus, IER has no effect on these interrupts.

You can read the IER to identify enabled or disabled interrupt levels, and you can write to the IER to enable or disable interrupt levels. To enable an interrupt level, set its corresponding IER bit to one using the OR IER instruction. To disable an interrupt level, set its corresponding IER bit to zero using the AND IER instruction. When an interrupt is disabled, it is not acknowledged, regardless of the value of the INTM bit. When an interrupt is enabled, it is acknowledged if the corresponding IFR bit is one and the INTM bit is zero.

When using the OR IER and AND IER instructions to modify IER bits make sure they do not modify the state of bit 15 (RTOSINT) unless a real-time operating system is present.

When a hardware interrupt is serviced or an INTR instruction is executed, the corresponding IER bit is cleared automatically. When an interrupt is requested by the TRAP instruction the IER bit is not cleared automatically. In the case of the TRAP instruction if the bit needs to be cleared it must be done by the interrupt service routine.

At reset, all the IER bits are cleared to 0, disabling all maskable CPU level interrupts.

The IER register is shown in Figure 86, and descriptions of the bits follow the figure.

Figure 86. Interrupt Enable Register (IER) — CPU Register

15	14	13	12	11	10	9	8
RTOSINT	DLOGINT	INT14	INT13	INT12	INT11	INT10	INT9
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
7	6	5	4	3	2	1	0
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 116. Interrupt Enable Register (IER) — CPU Register Field Descriptions

Bits	Field	Value	Description
15	RTOSINT		Real-time operating system interrupt enable. RTOSINT enables or disables the CPU RTOS interrupt.
		0	Level INT6 is disabled
		1	Level INT6 is enabled
14	DLOGINT		Data logging interrupt enable. DLOGINT enables or disables the CPU data logging interrupt.
		0	Level INT6 is disabled
		1	Level INT6 is enabled
13	INT14		Interrupt 14 enable. INT14 enables or disables CPU interrupt level INT14.
		0	Level INT14 is disabled
		1	Level INT14 is enabled
12	INT13		Interrupt 13 enable. INT13 enables or disables CPU interrupt level INT13.
		0	Level INT13 is disabled
		1	Level INT13 is enabled
11	INT12		Interrupt 12 enable. INT12 enables or disables CPU interrupt level INT12.
		0	Level INT12 is disabled
		1	Level INT12 is enabled
10	INT11		Interrupt 11 enable. INT11 enables or disables CPU interrupt level INT11.
		0	Level INT11 is disabled
		1	Level INT11 is enabled
9	INT10		Interrupt 10 enable. INT10 enables or disables CPU interrupt level INT10.
		0	Level INT10 is disabled
		1	Level INT10 is enabled
8	INT9		Interrupt 9 enable. INT9 enables or disables CPU interrupt level INT9.
		0	Level INT9 is disabled
		1	Level INT9 is enabled
7	INT8		Interrupt 8 enable. INT8 enables or disables CPU interrupt level INT8.
		0	Level INT8 is disabled
		1	Level INT8 is enabled
6	INT7		Interrupt 7 enable. INT7 enables or disables CPU interrupt level INT7.
		0	Level INT7 is disabled
		1	Level INT7 is enabled
5	INT6		Interrupt 6 enable. INT6 enables or disables CPU interrupt level INT6.
		0	Level INT6 is disabled
		1	Level INT6 is enabled
4	INT5		Interrupt 5 enable.INT5 enables or disables CPU interrupt level INT5.
		0	Level INT5 is disabled
		1	Level INT5 is enabled

Table 116. Interrupt Enable Register (IER) — CPU Register Field Descriptions (continued)

Bits	Field	Value	Description
3	INT4		Interrupt 4 enable.INT4 enables or disables CPU interrupt level INT4.
		0	Level INT4 is disabled
		1	Level INT4 is enabled
2	INT3		Interrupt 3 enable.INT3 enables or disables CPU interrupt level INT3.
		0	Level INT3 is disabled
		1	Level INT3 is enabled
1	INT2		Interrupt 2 enable.INT2 enables or disables CPU interrupt level INT2.
		0	Level INT2 is disabled
		1	Level INT2 is enabled
0	INT1		Interrupt 1 enable.INT1 enables or disables CPU interrupt level INT1.
		0	Level INT1 is disabled
		1	Level INT1 is enabled

The Debug Interrupt Enable Register (DBGIER) is used only when the CPU is halted in real-time emulation mode. An interrupt enabled in the DBGIER is defined as a time-critical interrupt. When the CPU is halted in real-time mode, the only interrupts that are serviced are time-critical interrupts that are also enabled in the IER. If the CPU is running in real-time emulation mode, the standard interrupt-handling process is used and the DBGIER is ignored.

As with the IER, you can read the DBGIER to identify enabled or disabled interrupts and write to the DBGIER to enable or disable interrupts. To enable an interrupt, set its corresponding bit to 1. To disable an interrupt, set its corresponding bit to 0. Use the PUSH DBGIER instruction to read from the DBGIER and POP DBGIER to write to the DBGIER register. At reset, all the DBGIER bits are set to 0.

Figure 87. Debug Interrupt Enable Register (DBGIER) — CPU Register

15	14	13	12	11	10	9	8
RTOSINT	DLOGINT	INT14	INT13	INT12	INT11	INT10	INT9
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
7	6	5	4	3	2	1	0
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 117. Debug Interrupt Enable Register (DBGIER) — CPU Register Field Descriptions

Bits	Field	Value	Description
15	RTOSINT		Real-time operating system interrupt enable. RTOSINT enables or disables the CPU RTOS interrupt.
		0	Level INT6 is disabled
		1	Level INT6 is enabled
14	DLOGINT	.	Data logging interrupt enable. DLOGINT enables or disables the CPU data logging interrupt
		0	Level INT6 is disabled
		1	Level INT6 is enabled
13	INT14	.	Interrupt 14 enable. INT14 enables or disables CPU interrupt level INT14
		0	Level INT14 is disabled
		1	Level INT14 is enabled
12	INT13		Interrupt 13 enable. INT13 enables or disables CPU interrupt level INT13.
		0	Level INT13 is disabled
		1	Level INT13 is enabled

Table 117. Debug Interrupt Enable Register (DBGIER) — CPU Register Field Descriptions (continued)

Bits	Field	Value	Description
11	INT12		Interrupt 12 enable. INT12 enables or disables CPU interrupt level INT12.
		0	Level INT12 is disabled
		1	Level INT12 is enabled
10	INT11		Interrupt 11 enable. INT11 enables or disables CPU interrupt level INT11.
		0	Level INT11 is disabled
		1	Level INT11 is enabled
9	INT10		Interrupt 10 enable. INT10 enables or disables CPU interrupt level INT10.
		0	Level INT10 is disabled
		1	Level INT10 is enabled
8	INT9		Interrupt 9 enable. INT9 enables or disables CPU interrupt level INT9.
		0	Level INT9 is disabled
		1	Level INT9 is enabled
7	INT8		Interrupt 8 enable. INT8 enables or disables CPU interrupt level INT8.
		0	Level INT8 is disabled
		1	Level INT8 is enabled
6	INT7		Interrupt 7 enable. INT7 enables or disables CPU interrupt level INT77.
		0	Level INT7 is disabled
		1	Level INT7 is enabled
5	INT6		Interrupt 6 enable. INT6 enables or disables CPU interrupt level INT6.
		0	Level INT6 is disabled
		1	Level INT6 is enabled
4	INT5		Interrupt 5 enable.INT5 enables or disables CPU interrupt level INT5.
		0	Level INT5 is disabled
		1	Level INT5 is enabled
3	INT4		Interrupt 4 enable.INT4 enables or disables CPU interrupt level INT4.
		0	Level INT4 is disabled
		1	Level INT4 is enabled
2	INT3		Interrupt 3 enable.INT3 enables or disables CPU interrupt level INT3.
		0	Level INT3 is disabled
		1	Level INT3 is enabled
1	INT2		Interrupt 2 enable.INT2 enables or disables CPU interrupt level INT2.
		0	Level INT2 is disabled
		1	Level INT2 is enabled
0	INT1		Interrupt 1 enable.INT1 enables or disables CPU interrupt level INT1.
		0	Level INT1 is disabled
		1	Level INT1 is enabled

6.6 External Interrupt Control Registers

Three external interrupts, XINT1 –XINT3 are supported. Each of these external interrupts can be selected for negative or positive edge triggered and can also be enabled or disabled. The masked interrupts also contain a 16-bit free running up counter that is reset to zero when a valid interrupt edge is detected. This counter can be used to accurately time stamp the interrupt.

Table 118. Interrupt Control and Counter Registers (not EALLOW Protected)

Name	Address Range	Size (x16)	Description
XINT1CR	0x0000 7070	1	XINT1 configuration register
XINT2CR	0x0000 7071	1	XINT2 configuration register
XINT3CR	0x0000 7072	1	XINT3 configuration register
reserved	0x0000 7073 - 0x0000 7077	5	
XINT1CTR	0x0000 7078	1	XINT1 counter register
XINT2CTR	0x0000 7079	1	XINT2 counter register
XINT3CTR	0x0000 707A	1	XINT3 counter register
reserved	0x0000 707B - 0x0000 707E	5	

XINT1CR through XINT3CR are identical except for the interrupt number; therefore, Figure 88 and Table 119 represent registers for external interrupts 1 through 3 as XINT*n*CR where *n* = the interrupt number.

Figure 88. External Interrupt *n* Control Register (XINT*n*CR)

15	4	3	2	1	0
Reserved		Polarity		Reserved	Enable
R-0		R/W-0		R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 119. External Interrupt *n* Control Register (XINT*n*CR) Field Descriptions

Bits	Field	Value	Description
15-4	Reserved		Reads return zero; writes have no effect.
3-2	Polarity		This read/write bit determines whether interrupts are generated on the rising edge or the falling edge of a signal on the pin.
		00	Interrupt generated on a falling edge (high-to-low transition)
		01	Interrupt generated on a rising edge (low-to-high transition)
		10	Interrupt is generated on a falling edge (high-to-low transition)
		11	Interrupt generated on both a falling edge and a rising edge (high-to-low and low-to-high transition)
1	Reserved		Reads return zero; writes have no effect
0	Enable		This read/write bit enables or disables external interrupt XINT*n*.
		0	Disable interrupt
		1	Enable interrupt

For XINT1/XINT2/XINT3, there is also a 16-bit counter that is reset to 0x000 whenever an interrupt edge is detected. These counters can be used to accurately time stamp an occurrence of the interrupt. XINT1CTR through XINT3CTR are identical except for the interrupt number; therefore, Figure 89 and Table 120 represent registers for the external interrupts as XINTnCTR, where n = the interrupt number.

Figure 89. External Interrupt n Counter (XINTnCTR) (Address 7078h)

15	0
INTCTR[15-8]	
R-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 120. External Interrupt n Counter (XINTnCTR) Field Descriptions

Bits	Field	Description
15-0	INTCTR	This is a free running 16-bit up-counter that is clocked at the SYSCLKOUT rate. The counter value is reset to 0x0000 when a valid interrupt edge is detected and then continues counting until the next valid interrupt edge is detected. When the interrupt is disabled, the counter stops. The counter is a free-running counter and wraps around to zero when the max value is reached. The counter is a read only register and can only be reset to zero by a valid interrupt edge or by reset.