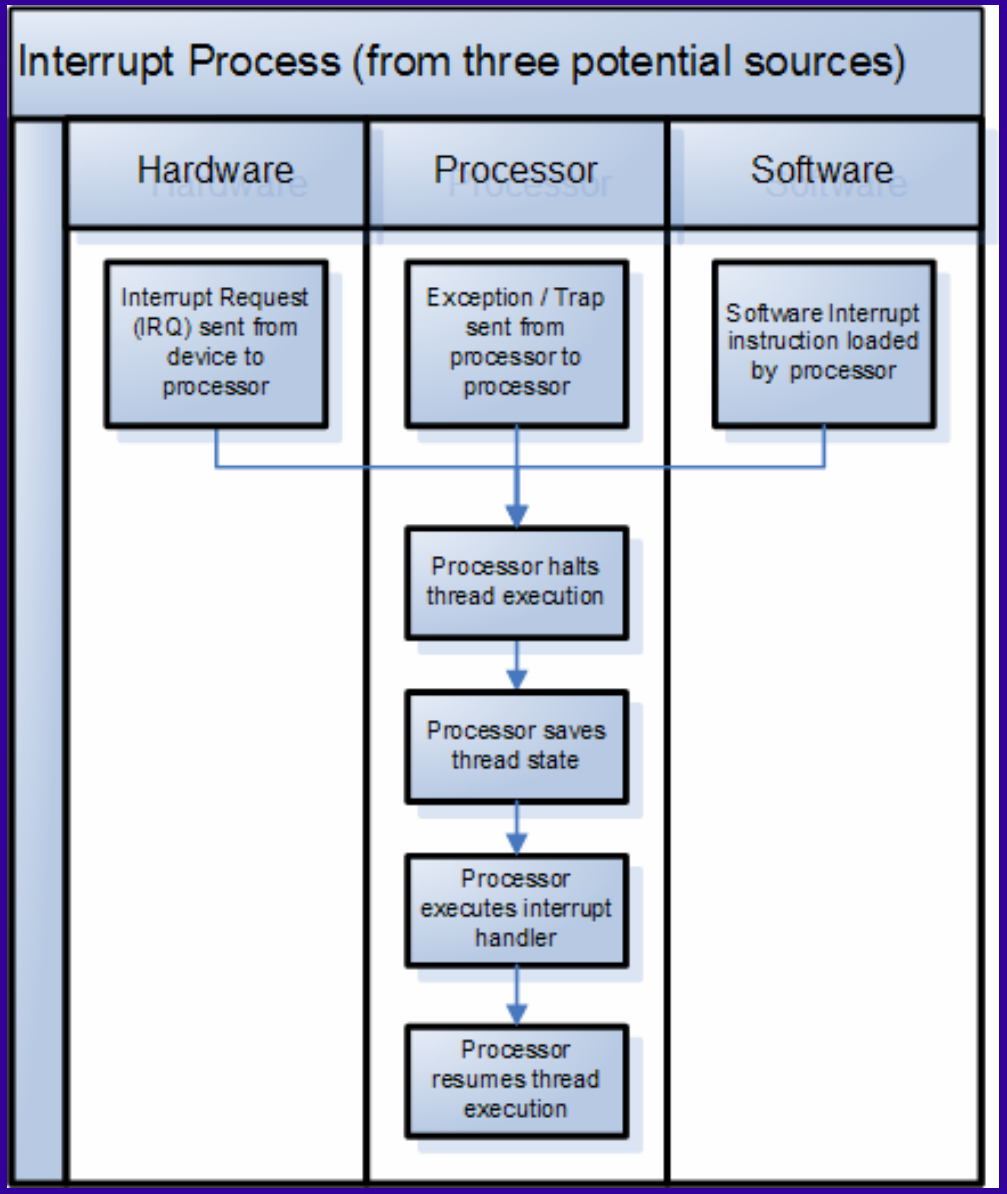
Micro Processor & Controller

Interrupts & PIE

Interrupt - Summary

In System Programming, an **interrupt** is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. An interrupt alerts the processor to a high-priority condition requiring the interruption of the current code the processor is executing. The processor responds by suspending its current activities, saving its state, and executing a function called an *interrupt handler* (or an interrupt service routine, ISR) to deal with the event. This interruption is temporary, and, after the interrupt handler finishes, the processor resumes normal activities. There are two types of interrupts: hardware interrupts and software interrupts.

Interrupt - Source



C28 Interrupts

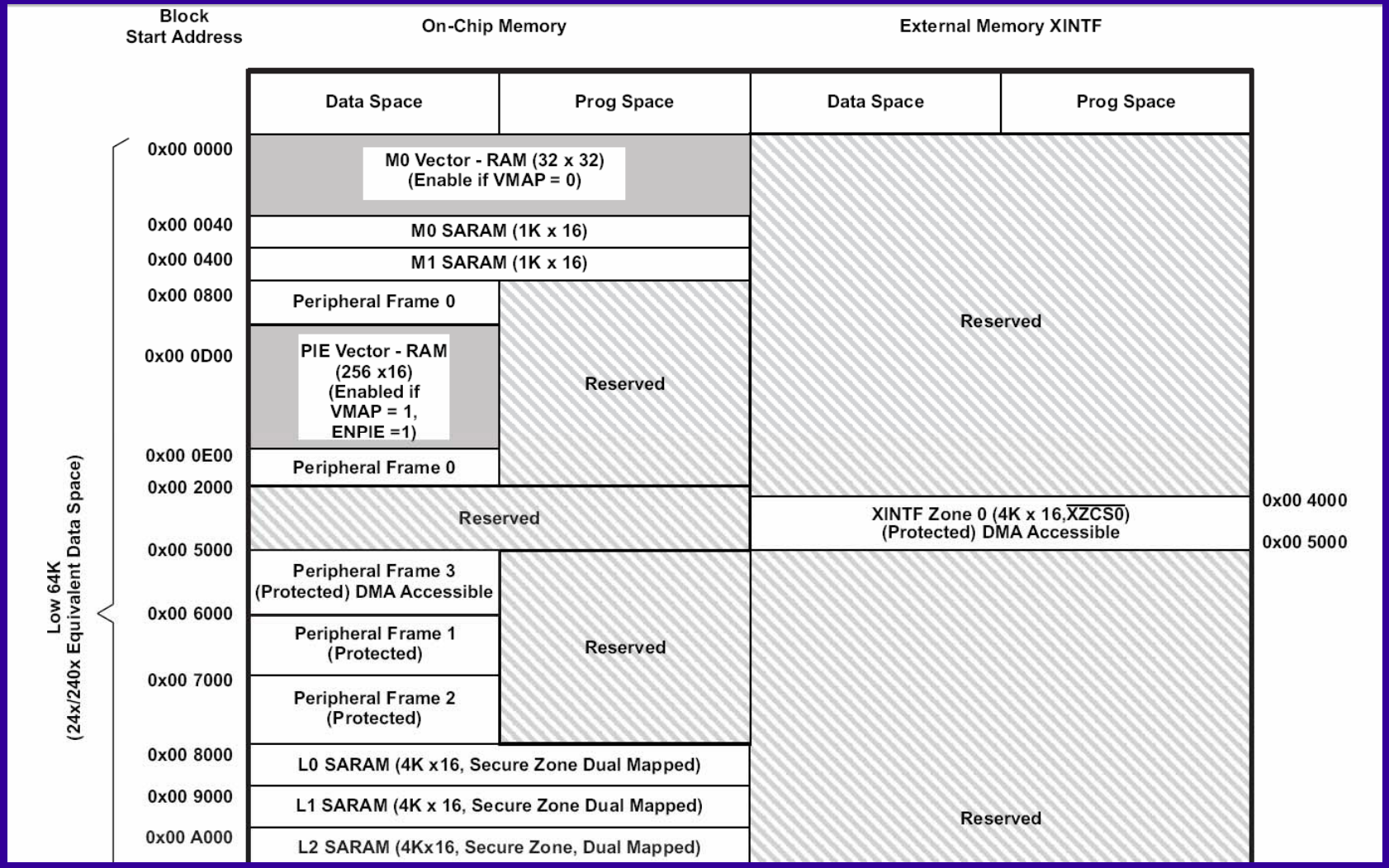
Overview of the PIE Controller

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.

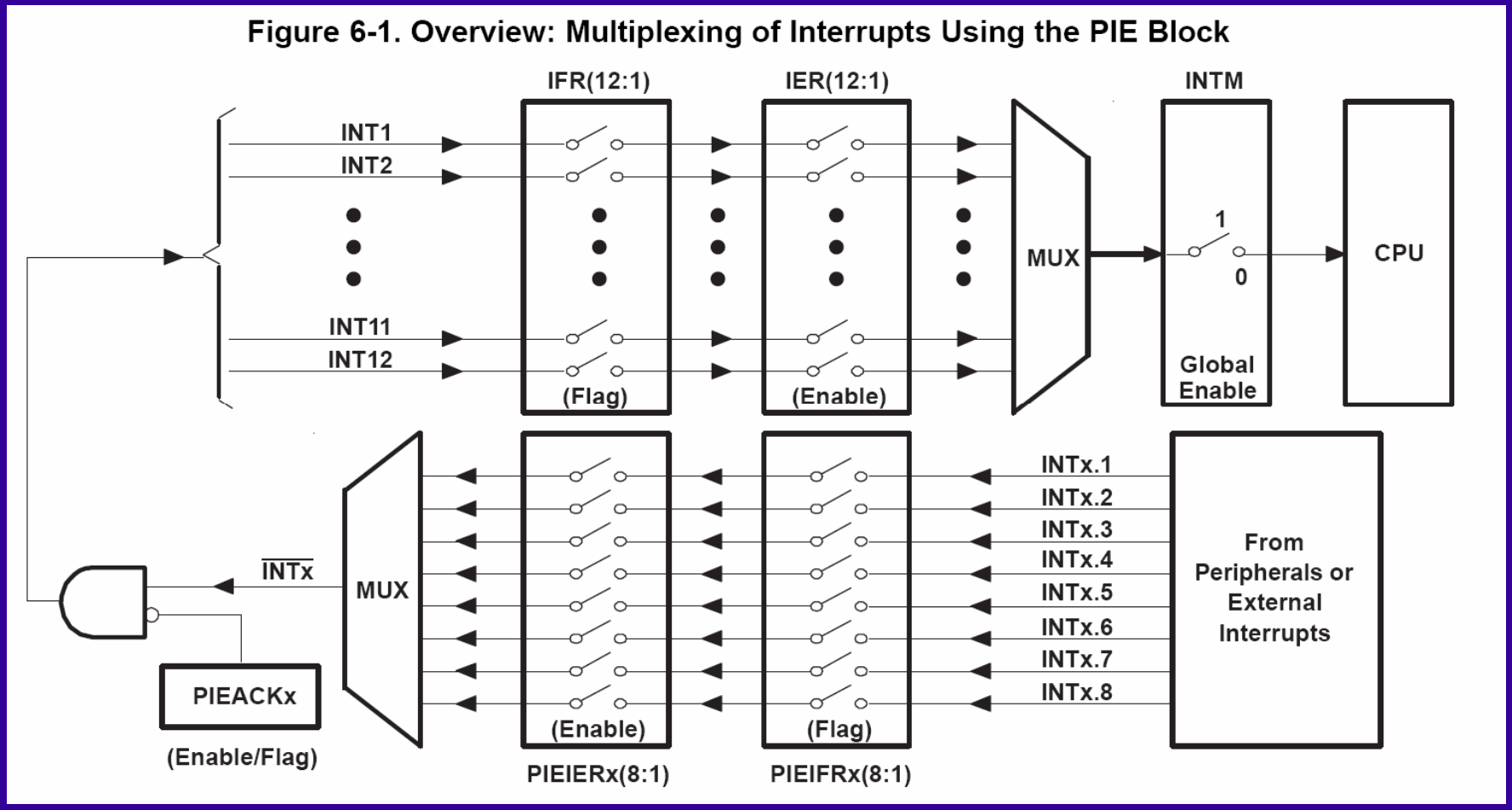
The PIE vector table is used to store the address (vector) of each interrupt service routine (ISR) within the system. There is one vector per interrupt source including all MUXed and nonMUXed interrupts. You populate the vector table during device initialization and you can update it during operation.

Int1 – In12 ➔ PIE

C28 Interrupts Memory Vectors



Peripheral Interrupt Enable (PIE)



12 x 8 = 96 Interrupts

PIE Vector Table

Table 6-4. 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*	*ADCINT*	*XINT2*	*XINT1*	Reserved	*SEQ2INT*	*SEQ1INT*
	(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	*EPWM6_ TZINT*	*EPWM5_TZINT*	*EPWM4_TZINT*	*EPWM3_TZINT*	*EPWM2_TZINT*	*EPWM1_TZINT*
	-	-	(ePWM6)	(ePWM5)	(ePWM4)	(ePWM3)	(ePWM2)	(ePWM1)
	0xD5E	0xD5C	0xD5A	0xD58	0xD56	0xD54	0xD52	0xD50
INT3.y	Reserved	Reserved	*EPWM6_ INT*	*EPWM5_INT*	*EPWM4_INT*	*EPWM3_INT*	*EPWM2_INT*	*EPWM1_INT*
	-	-	(ePWM6)	(ePWM5)	(ePWM4)	(ePWM3)	(ePWM2)	(ePWM1)
	0xD6E	0xD6C	0xD6A	0xD68	0xD66	0xD64	0xD62	0xD60
INT4.y	Reserved	Reserved	*ECAP6_INT*	*ECAP5_INT*	*ECAP4_INT*	*ECAP3_INT*	*ECAP2_INT*	*ECAP1_INT*
	-	-	(eCAP6)	(eCAP5)	(eCAP4)	(eCAP3)	(eCAP2)	(eCAP1)
	0xD7E	0xD7C	0xD7A	0xD78	0xD76	0xD74	0xD72	0xD70
INT5.y	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	*EQEP2_INT*	*EQEP1_INT*
	-	-	-	-	-	-	(eQEP2)	(eQEP1)
	0xD8E	0xD8C	0xD8A	0xD88	0xD86	0xD84	0xD82	0xD80
INT6.y	Reserved	Reserved	*MXINTA*	*MRINTA*	*MXINTB*	*MRINTB*	*SPITXINTA*	*SPIRXINTA*
			(McBSP-A)	(McBSP-A)	(McBSP-B)	(McBSP-B)	(SPI-A)	(SPI-A)
	0xD9E	0xD9C	0xD9A	0xD98	0xD96	0xD94	0xD92	0xD90
INT7.y	Reserved	Reserved	*DINTCH6*	*DINTCH5*	*DINTCH4*	*DINTCH3*	*DINTCH2*	*DINTCH1*
			(DMA6)	(DMA5)	(DMA4)	(DMA3)	(DMA2)	(DMA1)
	0xDAE	0xDAC	0xDAA	0xDA8	0xDA6	0xDA4	0xDA2	0xDA0
INT8.y	Reserved	Reserved	*SCITXINTC*	*SCIRXINTC*	Reserved	Reserved	*I2CINT2A*	*I2CINT1A*
	-	-	(SCI-C)	(SCI-C)	-	-	(I2C-A)	(I2C-A)
	0xDBE	0xDBC	0xDBA	0xDB8	0xDB6	0xDB4	0xDB2	0xDB0
INT9.y	*ECAN1INTB*	*ECAN0INTB*	*ECAN1INTA*	*ECAN0INTA*	*SCITXINTB*	*SCIRXINTB*	*SCITXINTA*	*SCIRXINTA*
	(CAN-B)	(CAN-B)	(CAN-A)	(CAN-A)	(SCI-B)	(SCI-B)	(SCI-A)	(SCI-A)
	0xDCE	0xDCC	0xDCA	0xDC8	0xDC6	0xDC4	0xDC2	0xDC0
INT10.y	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
	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	*LUF*	*LVF*	Reserved	*XINT7*	*XINT6*	*XINT5*	*XINT4*	*XINT3*
	(FPU)	(FPU)		Ext. Int. 7	Ext. Int. 6	Ext. Int. 5	Ext. Int. 4	Ext. Int. 3
	0xDFE	0xDFC	0xDFA	0xDF8	0xDF6	0xDF4	0xDF2	0xDF0

Example – Timer0 Int

```
//###########################################################################
//
// FILE:    Example_2833xLedBlink.c
//
// TITLE:   DSP2833x eZdsp LED Blink Getting Started Program.
//
// ASSUMPTIONS:
//
//    This program requires the DSP2833x header files.
//
//// DESCRIPTION:
//
//    This example configures CPU Timer0 for a 500 msec period, and toggles the
//    LED.
//
//       Watch Variables:
//          CpuTimer0.InterruptCount
//
//       Monitor the LED blink on (for 500 msec) and off (for 500 msec) on the 2833x eZdsp.
//
//###########################################################################
// $TI Release: 2833x/2823x Header Files and Peripheral Examples V133 $
// $Release Date: June 8, 2012 $
//###########################################################################


#include "DSP28x_Project.h"     // Device Headerfile and Examples Include File

// Prototype statements for functions found within this file.
interrupt void cpu_timer0_isr(void);

void main(void)
{

// Step 1. Initialize System Control:
// PLL, WatchDog, enable Peripheral Clocks
// This example function is found in the DSP2833x_SysCtrl.c file.
   InitSysCtrl();

// Step 2. Initalize GPIO:
// This example function is found in the DSP2833x_Gpio.c file and
// illustrates how to set the GPIO to it's default state.
// InitGpio();  // Skipped for this example


// Step 3. Clear all interrupts and initialize PIE vector table:
// Disable CPU interrupts
   DINT;

// Initialize the PIE control registers to their default state.
// The default state is all PIE interrupts disabled and flags
// are cleared.
// This function is found in the DSP2833x_PieCtrl.c file.
   InitPieCtrl();

// Disable CPU interrupts and clear all CPU interrupt flags:
   IER = 0x0000;
```