3. INTERUPTS

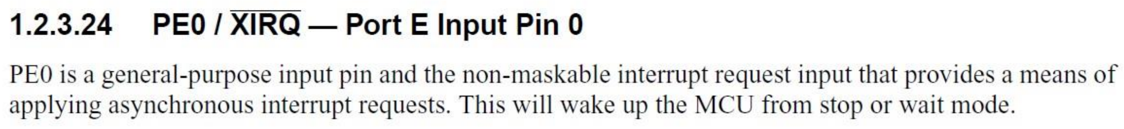
An interrupt is a hardware event that requires the MCU to stop the normal execution of the program to perform a service related to such event. It can be generated by some internal (on chip) peripheral such as a timer, or by an external one, such as a button press. The process of servicing an interrupt is done through a special function known as Interrupt Service Routine (ISR).

Interrupts can be used to do the following:

- Coordinating I/O tasks and preventing the MCU from being halted during such process. Without an interrupt, the system needs to be constantly checking the status of the I/O device, which refers to the method known as *polling*.
- Performing time-critical applications. Many events require the MCU to act immediately. Interrupts provide a mechanism to force the program to divert from normal execution and take immediate action.
- Providing a way of exiting and application when an error occurs. There are many interrupts that can be triggered when a certain fault flag is activated.
- Perform routine tasks. There are many embedded system applications that require the MCU to perform some routine work at a constant time interval or a time interval that cannot exceed certain value. Examples of these are: keeping track or real time, periodic data acquisition, or task switching in a multithreaded operating system (RTOS).

3.1 Interrupt Masking

Some interrupts may not be needed or desired. In this case, they should be disabled. However, there are certain interrupts that cannot be disabled, they are known as non-maskable interrupts. For instance:

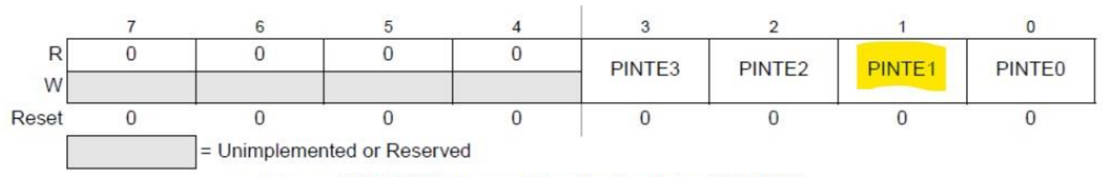


1.2.3.24 PE0 / $\overline{\text{XIRQ}}$ — Port E Input Pin 0

PE0 is a general-purpose input pin and the non-maskable interrupt request input that provides a means of applying asynchronous interrupt requests. This will wake up the MCU from stop or wait mode.

In any case, to enable interrupts globally (global interrupt masking) we use the *Enableinterrupts* macro and to disable them, se use *DisableInterrupts.*

Aside from this masking, maskable interrupts can be enabled or disabled by setting or clearing a specific BIT in the corresponding register. For instance, to enable the Interrupt for PIT1 we set the BIT1 in this register (See PIT Notes).



	7	6	5	4	3	2	1	0
R	0	0	0	0	PINTE3	PINTE2	PINTE1	PINTE0
W	▨	▨	▨	▨				
Reset	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 13-7. PIT Interrupt Enable Register (PITINTE)

3.2 Interrupt Priority

When using multiple interrupts, the interrupt with higher priority always gets serviced before another one with lower priority. In the case of this micro (9S12), they are services according to the number in the interrupt vector. The lower the number, the higher the priority

3.3 Interrupt Service Routine (ISR)

When an interrupt gets called, the system needs to stop the main program execution and service the ISR, then go back to the main program. This is accomplished by saving the program counter (PC) and the MCU status before executing the ISR, then restoring the saved program counter (PC) and MCU status.

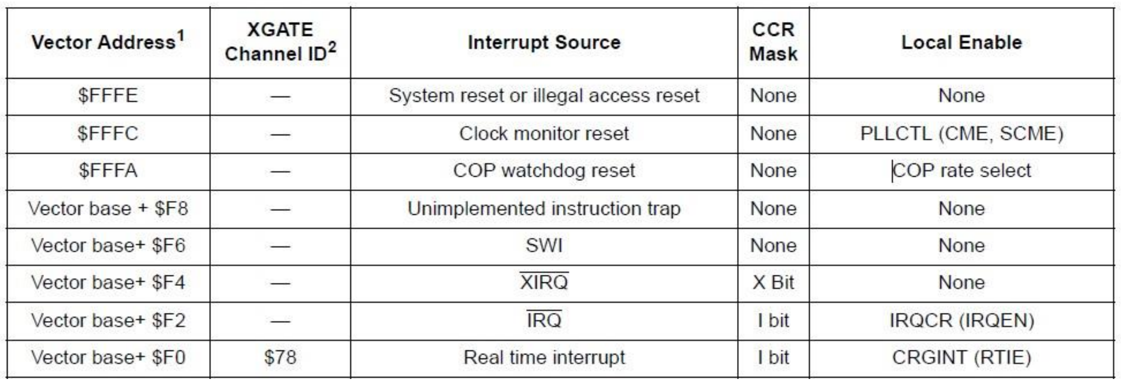
ISR Example:

```
interrupt VectorNumber_Vrti void Vrti_ISR(void)
{
  CRGFLG = CRGFLG_RTIF_MASK; //clear flag;
//Do something
}
```

3.4 Interrupt Vector

As mentioned before, interrupt vectors are saved in an interrupt vector table. The address of each vector in such a table is known as the vector address. The detailed information about this address mapping can be found in the micro datasheet chapter 1:

Table 1-12. Interrupt Vector Locations (Sheet 1 of 3)



Vector Address[1]	XGATE Channel ID[2]	Interrupt Source	CCR Mask	Local Enable
$FFFE	—	System reset or illegal access reset	None	None
$FFFC	—	Clock monitor reset	None	PLLCTL (CME, SCME)
$FFFA	—	COP watchdog reset	None	COP rate select
Vector base + $F8	—	Unimplemented instruction trap	None	None
Vector base+ $F6	—	SWI	None	None
Vector base+ $F4	—	$\overline{XIRQ}$	X Bit	None
Vector base+ $F2	—	$\overline{IRQ}$	I bit	IRQCR (IRQEN)
Vector base+ $F0	$78	Real time interrupt	I bit	CRGINT (RTIE)

The vector addresses are also listed in the derivative file.