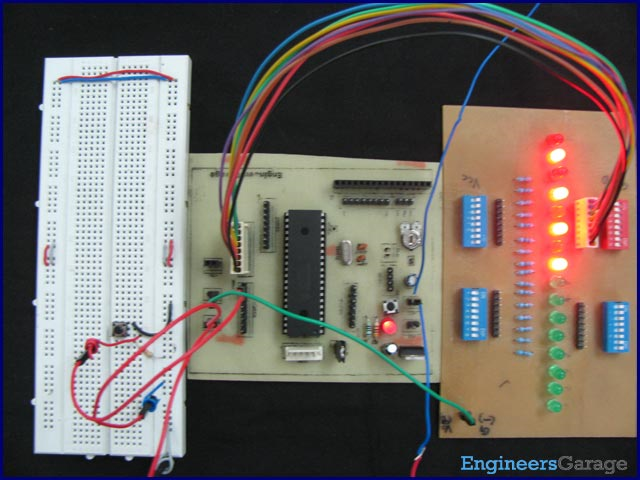
**[How to work with External (Hardware) Interrupts of PIC Microcontroller (PIC18F4550)](http://www.engineersgarage.com/embedded/pic-microcontroller-projects/pic-external-hardware-interrupts-circuit" \o "How to work with External (Hardware) Interrupts of PIC Microcontroller (PIC18F4550))**



[Interrupts](http://www.engineersgarage.com/tutorials/interrupts-8051-interrupt-programming) are special events that require immediate attention. They cause the processor to cease the running task to serve a special task for which the interrupt event had occurred. After the special task is over, the processor resumes performing the original task.

The processor can also serve these events by polling method. But polling is an inefficient technique as compared to interrupts. In the polling method, the processor has to continuously wait for the event signal. Thus it always remains busy using extra resources. To understand the difference between polling and interrupts well, refer introductory paragraphs of [8051 Interrupts](http://www.engineersgarage.com/tutorials/interrupts-8051-interrupt-programming).

This article is based on [PIC18F4550](http://www.engineersgarage.com/electronic-components/pic18f4550-microcontroller) microcontroller’s interrupt system. The configuration and implementation of [PIC](http://www.engineersgarage.com/articles/pic-microcontroller-tutorial) Hardware Interrupts are explained here.

A [PIC microcontroller](http://www.engineersgarage.com/articles/pic-microcontroller-tutorial) consists of both **software and hardware generated interrupts**. The **hardware interrupts** are produced by external hardware at certain pins of the [microcontroller](http://www.engineersgarage.com/microcontroller). The software interrupts, on the other hand, are generated by internal peripherals of the controller. This software interrupt helps the programmer to use more than one internal peripheral in single application and serve them individually when they respond.

Following is an example to illustrate the interrupts better. Suppose a programmer wants to make a real time watch which shows ambient temperature as well. The programmer has to use two internal peripherals of the controller, namely, a [Timer](http://www.engineersgarage.com/tutorials/timers-8051-timer-programming-tutorial) and an ADC channel. Consider this project without using interrupt : the programmer has to take care of both peripherals one by one continuously by polling them. This is not an efficient way of programming.

In Interrupt method, the controller serves the Timer when it overflows and the ADC when the A/D (analog to digital) conversion is done. Along with these, the microcontroller can also perform other tasks, like [displaying some text on LCD](http://www.engineersgarage.com/embedded/pic-microcontroller-projects/lcd-interfacing-text-circuit). Therefore use of interrupt makes the program more efficient and logical.

When an Interrupt occurs in a PIC Microcontroller, the program execution gets transferred to a predefined Interrupt Vector Address from where the processor gets what operations to perform in for a particular interrupt occurrence. The detailed working of interrupts can also be studied from [8051 Interrupts](http://www.engineersgarage.com/tutorials/interrupts-8051-interrupt-programming) or [AVR Interrupts](http://www.engineersgarage.com/embedded/avr-microcontroller-projects/external-hardware-interrupts).

**Interrupts in PIC18F4550:**

[PIC18F4550](http://www.engineersgarage.com/electronic-components/pic18f4550-microcontroller) has following **internal and external** interrupts:

·         Reset, Brown-Out Reset, Watch-dog Reset, Power On Reset

·         External Interrupt 0 (INT0)

·         External Interrupt 1 (INT1)

·         External Interrupt 2 (INT2)

·         Timer 0 Interrupt

·         Timer 1 Interrupt

·         Timer 2 Interrupt

·         Timer 3 Interrupt

·         ADC Interrupt

·         Analog Comparator Interrupt

·         RB Port change Enable Interrupt

·         Streaming Parallel Port Read/Write Interrupt

·         EUSART Receive Interrupt

·         EUSART Transmit Interrupt

·         Master Synchronous Serial Port Interrupt

·         CCP1 Interrupt (Capture, Compare, PWM)

·         Oscillator Fail Interrupt

·         USB Interrupt

·         Data EEPROM/Flash Write Operation Interrupt

·         Bus Collision Interrupt

·         High/Low-Voltage Detect Interrupt

·         CCP2 Interrupt

A total of 10 registers are used to control the interrupt operation in PIC18F4550 which are as follows:

·         RCON

·         INTCON, INTCON2, INTCON3

·         PIR1, PIR2

·         PIE1, PIE2

·         IPR1, IPR2

In this article, the hardware interrupts of PIC18F4550 have been explained. To use the External PIC Interrupts, INTCON registers are required to be configured. The PIE (Peripheral Interrupt Enable) and PIR (Peripheral Interrupt Request) registers are used to configure the Peripheral (Internal) Interrupts. The peripheral interrupts will be covered in subsequent articles.

**Registers for External (Hardware) Interrupts:**

The INTCON registers contain various enable, priority and flag bits for different hardware interrupt operations.

·         Flag bits                     :           indicate that an interrupt event has occurred.

·         Enable bits                :           allow the program execution to be mapped to *Interrupt Vector Address* when Flag bits are set.

·         Priority bits                 :           select high or low priority.

The individual INTCON registers and their configurations have been explained below:

1. **INTCON (Interrupt Control Register)**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit 7** | **Bit 6** | **Bit 5** | **Bit 4** | **Bit 3** | **Bit 2** | **Bit 1** | **Bit 0** |
| GIE/GIEH | PEIE/GIEL | TMR0IE | INT0IE | RBIE | TMR0IF | INT0IF | RBIF |

**INT0IF:** This is External Interrupt 0 (INT0) flag bit.

1 = The INT0 External interrupt occurred (must be cleared in software)

0 = The INT0 External Interrupt did not occur

**INT0IE:** This bit enables/disables the External Interrupt 0.

1 = Enables the External Interrupt 0

0 = Disables the External Interrupt 0

**PEIE/GIEL:** This bit is used to enable/disable all the peripheral interrupts (Internal interrupts) of the controller. But GIE/GIEH bit must be set to high first.

1 = Enables all Peripheral Interrupts

0 = Disables all Peripheral Interrupts

**GIE/GIEH:** This is Global Interrupt Enable bit. This bit is set to high to enable all interrupts of PIC18F4550.

1 = Enables interrupts

0 = Disables all interrupts

2. **INTCON2 (Interrupt Control Register 2):**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit 7** | **Bit 6** | **Bit 5** | **Bit 4** | **Bit 3** | **Bit 2** | **Bit 1** | **Bit 0** |
| RBPU | INTEDG0 | INTEDG1 | INTEDG2 | — | TMR0IP | — | RBIP |

**INTEDG0, INTEDG1, INTEDG2:** These bits are used select the triggering edge of the corresponding interrupt signal on which the controller should respond.

1 = Interrupt on rising edge

0 = Interrupt on falling edge

3. **INTCON3** **(Interrupt Control Register 3):**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit 7** | **Bit 6** | **Bit 5** | **Bit 4** | **Bit 3** | **Bit 2** | **Bit 1** | **Bit 0** |
| INT2IP | INT1IP | — | INT2IE | INT1IE | — | INT2IF | INT1IF |

**INT1IF, INT2IF:** These are External Interrupt 1 and 2 flag bits, respectively.

1 = The INTx External Interrupt occurred (must be cleared in software)

0 = The INTx External Interrupt did not occur

**INT1IE, INT2IE:** These bits enable/disable the External Interrupt 1 and 2, respectively.

1 = Enables the External Interrupt x

0 = Disables the External Interrupt x

**INT1IP, INT2IP:** These bits are used to set priority of the interrupts 1 and 2, respectively.

1 = High priority

0 = Low priority

**Working with an External PIC Interrupt:**

**Objective:** To configure the External Interrupt 0 (INT0) and invert (or toggle) the output at PORTD when interrupt occurs.

The output at PORTD is monitored through a set of 8 LEDs. Their connections with PIC18F4550 are shown in the circuit diagram tab.

**Programming Steps:**

1. Enable the External Interrupt 0 by setting INT0IE bit high (INTCON).

2. Set the interrupt on falling edge by setting the INTEDG0 bit to zero (INTCON2).

3. Enable Global Interrupt by setting GIE bit to high (INTCON).

4. Start a while loop and initialize PORTD with certain value.

5. Write the ISR (Interrupt Service Routine) for this interrupt by using interrupt() as ISR header.

6. Clear the INT0IF bit (INTCON).

7. Write instruction to invert or toggle the value at PORTD.

In this algorithm, the program control will jump to ISR when a falling edge signal appears at INT0 (Pin 33) of [PIC18F4550](http://www.engineersgarage.com/electronic-components/pic18f4550-microcontroller).

**//Program to depict the working of External Interrupt0 of PIC18F4550**  
void main()  
{  
    TRISD=0;                      // Configure PortD as output port  
    INTCON=0x10;                  // Enable INT0  
    INTCON2=0;                    // Set Falling Edge Trigger for INT0  
  
    INTCON.GIE=1;    // Enable The Global Interrupt  
    while(1)  
    {  
        LATD=0x55;       //Set some value at PortD  
    }  
}  
  
void interrupt()              // Interrupt ISR  
{  
    INTCON.INT0IF=0;          // Clear the interrupt 0 flag  
    LATD=~LATD;               // Invert (Toggle) the value at PortD  
    Delay\_ms(1000);           // Delay for 1 sec  
}

