

**CALIFORNIA STATE POLYTECHNIC UNIVERSITY, POMONA  
COLLEGE OF ENGINEERING**

**ECE 3301L  
Spring 2022  
Session 3**

**Microcontroller Lab**

**Felix Pinai**

**Experiment 9  
External Interrupts**

The purpose of this lab is to get the student to be familiar with the concept of system interrupt and in particular the external interrupts.

**PART A) External Interrupts**

A hardware interrupt is a signal generated by a special function device to indicate to the CPU that an event has happened and the CPU needs to temporarily leave what it is working on so it can go to process the needs of that event. There are some procedures to ensure that the CPU remembers the location where it leaves before it goes away to process the interrupt. The new location where the CPU jumps to take care of the interrupt is called an Interrupt Service Routine (ISR). The CPU will enter the ISR routine when the interrupt occurs and when the task in that routine is completed, the CPU will perform a Return from Interrupt (RETFIE) in order to return to the main program at the location where the CPU was interrupted.

The advantage for an interrupt is that the CPU does not have to constantly wait for a hardware event to happen. It just needs to perform any task that it needs to take care of sitting idle doing nothing and then when an interrupt happens, the CPU just jumps to the ISR to handle that interrupt.

To allow interrupts to be able to be processed by the CPU, each hardware device that is capable to generate an interrupt has an 'Interrupt Enable' (IE) bit associated with it. When set to 1, the IE will allow an interrupt to happen. In addition, an 'Interrupt Flag' (IF) bit is set when an interrupt event happens. If the IE is on and when IF is also set, then the interrupt has occurred. When an interrupt is processed by the CPU, the IF has to be cleared to allow the next interrupt event to happen. If IF is not cleared, no further interrupt can be generated.

In addition, a 'Global Interrupt Enable' (GIE) bit is a general control bit that will allow any interrupt to be generated to the CPU when that bit is set to 1. This GIE has to be turned on in order for interrupts to happen.

Refer to chapter 10 'Interrupts' of the PIC18F460 datasheets to see all the control bits for the interrupt functions of the hardware devices.

External interrupt is a type of interrupt when an input changes from a logic level to the other level. The edge transitioning from a selectable low-to-high or high-to-low transition will in fact generate an interrupt. There are three pins on the PIC18F4620 that can be used for such operation. They are called by the name INTx where x can be 0, 1 or 2. Here are the registers associated with these pins:

```

INTCONbits.GIE=1;                // Set the Global Interrupt Enable

1) Interrupt Enable bits:
    INTCONbits.INT0IE ;           // INT0 IE is in INTCON
    INTCON3bits.INT1IE;          // INT1 IE is in INTCON3
    INTCON3bits.INT2IE;          // INT2 IE is in INTCON3

2) Interrupt Flag bits:
    INTCONbits.INT0IF ;           // INT0 IF is in INTCON
    INTCON3bits.INT1IF;          // INT1 IF is in INTCON3
    INTCON3bits.INT2IF;          // INT2 IF is in INTCON3

3) Interrupt Edge Select bits:
    INTCON2bits.INTEDG0 ;         // INT0 EDGE is in INTCON2
    INTCON2bits.INTEDG1;         // INT1 EDGE is in INTCON2
    INTCON2bits.INTEDG2;         // INT2 EDGE is in INTCON2

```

The INTxIE must be set to logic 1 to enable the associated INTx signal to generate an interrupt when it transitions with the type of edge specified by the INTEDGx bit (see below).

The INTxIF bit is set to 1 when the associated INTx signal has made the edge transition that would cause a subsequent interrupt. This bit needs to be cleared once the interrupt has been processed in order to allow the next interrupt to occur again. Leaving that bit to logic 1 will prevent further interrupt.

A INTEDGx bit when set to 0 indicates that an interrupt is to be generated when the logic transition goes from high-to-low (falling edge) and the other way when it is set to 1.

Here is an example to test the external interrupts:

```

int INT0_flag, INT1_flag, INT2_flag = 0;

void Do_Init()                // Initialize the ports
{
    Init_Uart();               // Initialize the uart
    Init_ADC();                // Initialize the ADC with the
                                // programming of ADCON1

    OSCCON=0x70;               // Set oscillator to 8 MHz

```

```

    TRISB = 0x??;           // Configure the PORTB to make sure
                             // that all the INTx pins are
                             // inputs
    RBPU = 0x00;           // Enable pull-up for Port B
    INTCONbits.INT0IF = 0;  // Clear INT0IF
    INTCON3bits.INT1IF = 0; // Clear INT1IF
    INTCON3bits.INT2IF = 0; // Clear INT2IF

    INTCON2bits.INTEDG0=0;  // INT0 EDGE falling
    INTCON2bits.INTEDG1=0;  // INT1 EDGE falling
    INTCON2bits.INTEDG2=1;  // INT2 EDGE rising

    INTCONbits.INT0IE =1;   // Set INT0 IE
    INTCON3bits.INT1IE=1;   // Set INT1 IE
    INTCON3bits.INT2IE=1;   // Set INT2 IE

    INTCONbits.GIE=1;       // Set the Global Interrupt Enable
}

void interrupt high_priority chkisr()
{
    if (INTCONbits.INT0IF == 1) INT0_ISR();    // check if INT0
                                                // has occurred
    if (INTCON3bits.INT1IF == 1) INT1_ISR();
    if (INTCON3bits.INT2IF == 1) INT2_ISR();
}

void INT0_ISR()
{
    INTCONbits.INT0IF=0;           // Clear the interrupt flag
    INT0_flag = 1;                 // set software INT0_flag
}

void INT1_ISR()
{
    INTCON3bits.INT1IF=0;          // Clear the interrupt flag
    INT1_flag = 1;                 // set software INT1_flag
}

void INT2_ISR()
{
    INTCON3bits.INT2IF=0;          // Clear the interrupt flag
    INT2_flag = 1;                 // set software INT2_flag
}

```

```

void main()
{
    Do_Init();                // Initialization
    while (1)
    {                          // Do nothing,

        if (INT0_flag == 1)
        {
            INT0_flag = 0;    // clear the flag
            printf ("INT0 interrupt pin detected \r\n");
                                // print a message that INT0 has
                                // occurred
        }
        if (INT1_flag == 1)
        {
            INT1_flag = 0;    // clear the flag
            printf ("INT1 interrupt pin detected \r\n");
                                // print a message that INT1 has
                                // occurred
        }
        if (INT2_flag == 1)
        {
            INT2_flag = 0;    // clear the flag
            printf ("INT2 interrupt pin detected \r\n");
                                // print a message that INT2 has
                                // occurred
        }
    }
}

```

Each time when either push-button SW2, SW3 or SW4 (see schematics) is pressed down, the interrupt INT0, INT1 or INT2 will be generated. The associated INT0\_ISR(), INT1\_ISR() or INT2\_ISR() will be executed to service the interrupt. In each routine, the associated hardware INTxIF will be cleared to allow the next interrupt to be generated. Next, the associated variable INT0\_flag, INT1\_flag or INT2\_flag will be to 1.

In the main program, the three variables INT0\_flag, INT1\_flag, INT2\_flag will be constantly monitored. When either one flag is detected to be 1, that variable will be cleared and a message will be printed to show the corresponding push-button switch has been pressed.

Complete the above program and run it. Push each button associated with the INT line and see on the screen that a message showing that the INT pin has been pressed.

## **PART B) Use of external interrupts in the Traffic Controller Design**

We will now use the external interrupts to trigger the requests for pedestrian accesses on the North-South and East-West directions. We will not use the DIP switches for the Pedestrian requests as in Lab 8. We will use instead the two push-buttons SW2 and SW3 that are hooked to the signals INT0 and INT1 (PORT B bit 0 and PORTB Bit 1 respectively).

When a push-button is pressed, this will set a flag to indicate that a request for the pedestrian access. When it is the time to do the handling of a pedestrian access, the flag will be checked. If it is not set to 1, then no pedestrian access will be performed. If it is set 1, then the pedestrian process will be started. When done, the flag will be cleared so that on the next pass no new pedestrian access will be performed unless the push-button switch is pressed in-between.

Use the setup on the program above, do the following:

- copy the code to initialize the INTCON registers (the code in bold)
- copy the 'void interrupt high\_priority chkisr()' routine
- copy the INT0\_ISR(), INT1\_ISR() and INT2\_ISR() routines

Next, take care of the following steps:

- 1) Remove the definitions for the signals:
  - a. NS\_PED\_SW
  - b. EW\_PED\_SW

We no longer use these switches to detect the pedestrian requests

- 2) Create two variables with the same name:
  - a. char NS\_PED\_SW
  - b. char EW\_PED\_SW
- 3) Replace in the INT0\_ISR() and INT1\_ISR() the variables INT0\_flag and INT1\_flag respectively by those two variables
- 4) This substitution will allow each pedestrian request to activated when the INT0 or INT1 push-button switch is pressed down.
- 5) Use the main routine from lab 8 (don't use the main routine from the example above)
- 6) When the program runs properly, the NSP and EWP fields on the TFT panel will show a '1' when the INT0 or INT1 switch is pressed. The pedestrian process will be handled as usual. However, make sure to add one extra step in the

PED\_control routine **to clear the NS\_PED\_SW or EW\_PED\_SW variable** to 0 so that the pedestrian countdown does not get executed again unless the associated push-button is pressed in-between.

- 7) In addition, in the interrupt service routine ISR(), add code to check what mode the INTx has occurred. If the mode is in day mode, then proceed as normal to set the variable NS\_PED\_SW or EW\_PED\_SW. However, if the mode is in night more, do not set the variable to 1 because no pedestrian access is allowed in that mode.
- 8) When switching from day mode to night mode, make sure to clear both NS\_PED\_SW and EW\_PED\_SW variables to prevent any pedestrian process to be handled in night mode.

If the implementation is done properly, the 'NSP' and 'EWP' display on the screen should have a '0' shown most of the time. When a push-button either at INT0 or INT1 is pressed, then a '1' will show up accordingly and stays at '1' until the pedestrian process in the proper direction is completed. The '1' should be changed to '0' and stays at '0' until the next time the push-button is pressed again.

In addition, make sure the following:

- 1) Any pedestrian request set in day mode should be cleared when the transition goes from day mode to night more.
- 2) Make sure that no request is allowed in night mode

## **PART C) Implementation of Flashing Mode**

Use the INT2 associated with the push-button SW4 to initiate an RED light flashing request. When the INT2 occurs, it will set a flag variable called FLASHING\_REQUEST to logic '1'. When the traffic controller has finished to execute a sequence either in Day mode or in Night mode (depending on the Photo-resistor), the program does return back to the main program. The next task is to check the logic of this FLASHING\_REQUEST flag. If 0, then no request is asked. If 1, then the program will reset the FLASHING\_REQUEST flag to 0 and then call a routine called Do\_Flashing() whereas the following steps must be executed:

- 1) Set the FLASHING variable to 1
- 2) A while loop waiting for the variable FLASHING to be cleared to 0 (while (FLASHING == 1))
- 3) Inside the while loop, a check of the variable FLASHING\_REQUEST must be performed

- 4) If FLASHING \_REQUEST is 1, then clear that variable FLASHING \_REQUEST to 0 and clear the variable FLASHING to 0. This will force the exit out of the while loop. This is to detect when the user presses on the INT2 switch to request to exit out of the emergency mode.
- 5) If FLASHING \_REQUEST is 0, then we are still in flashing mode. Therefore, we need to set all the four directions with the RED color, then do WAIT\_ONE\_SECOND(), clear all the four directions with the OFF color, and do another WAIT\_ONE\_SECOND(). This will create the effect of flashing the RED color in all directions.

That completes the implementation of the Do\_Flashing() routine.

In addition, to update the status of the 'FR' and 'FS' fields on the LCD, add the following lines in the update\_LCD\_misc() routine:

```
if (FLASHING _REQUEST == 0) FlashingR_Txt[0] = '0'; else FlashingR_Txt[0] = '1';  
if (FLASHING == 0) FlashingS_Txt[0] = '0'; else FlashingS_Txt[0] = '1';
```