External Interrupts

The purpose of this lab is to get 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 temporary 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 (RETI) 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 or 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 to set 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();                  // Initiailize the ADC with the
                                     // programming of ADCON1
        OSCCON=0x70;                 // Set oscillator to 8 MHz
        TRISB = 0x??                 // Configure the PORTB for the External
```

```
                                        // pins to make sure that all the INTx are
                                        // inputs

        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 occured
        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)         // if software INT0 flag is set
                {
                        INT0_flag = 0;      // clear that software flah
                        printf ("INT0 interrupt pin detected \r\n");
                                            // print a message that INT0 has //
                                            occured
                }
                // take the code for INT0, duplicate it and modify it to handle the
    other two INT1 and INT2 interrupts.
        }
}
```

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 constanstly 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 by setting those DIP switches in the 'OFF' position making the signals staying in the logic '1'. Now we will use two push-buttons SW2 and SW3 that are connected in parallel with those two original DIP switches and 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 access will be performed. If it is set 1, then the normal access will be performed. When done, the flag will be cleared so that on the

next pass no new access will be performed again unless the push-button switch is pressed again in-between.

Use the setup on the program above, copy the steps to initialize the handling of the three INTx signals like the code shown on the sample program above into the code of lab #8. 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 the variables INT0_flag and INT0_flag respectively by those two variables

4) This substitution will allow the pedestrian countdown process to be executed as normal since the PED_Control is only called when either variable is set 1.

5) 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.

6) In addition, in the interrupt service routine ISR(), check in what mode the INTx has occurred. If the mode is in day mode, then proceed as normal. If night more, do not the variable to 1 because no pedestrian access in that mode. Make sure to do this check only after clearing the hardware IF flag.

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-buttton is pressed again.

PART C) Implementation of Emergency Mode

Use the INT2 associated with the push-button SW4 to initiate an emergency request. When the INT2 occurs, it will set a flag variable called EMERGENCY_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 EMERGENCY_REQUEST flag. If 0, then no request is asked. If 1, then the program will reset the EMERGENCY_REQUEST flag to 0 and then call a routine called Do_Emergency() whereas the following steps must be executed:

1) Set the EMERGENCY variable to 1

2) A while loop waiting for the variable EMERGENCY to be cleared to 0 (while (EMERGENCY == 1))

3) Inside the while loop, a check of the variable EMERGENCY_REQUEST must be performed

4) If EMERGENCY_REQUEST is 1, then clear that variable EMERGENCY_REQUEST to 0 and clear the variable EMERGENCY 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 EMERGENCY_REQUEST is 0, then we are still in emergency 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_Emergency() routine.

In addition, to update the status of the 'EMR' and 'EMS' fields on the LCD, add the following lines in the update_misc() routine:

```
   if (EMERGENCY_REQUEST == 0) EmergencyR_Txt[0] = '0'; else
EmergencyR_Txt[0] = '1';
   if (EMERGENCY == 0) EmergencyS_Txt[0] = '0'; else EmergencyS_Txt[0] = '1';
```