Department of Computer Engineering
Digital Hardware Systems
CpE 3201 - Embedded Systems

Practical Activity #3
Interrupts (External and Timer Overflows)

Exercise Objectives:
1. Configure external and timer overflow interrupt sources
2. Use interrupts for input devices and delays
3. Developing embedded systems application using interrupt-based software

Student Outcomes:
At the end of the exercise, students must be able to:
1. learn how to configure the interrupts of the PIC16F877A (RB0/INT and Timer0 Overflow)
2. able to use interrupts for keypad control and timer delays;
3. apply interrupt-based software for I/O control.

Tools Required:
The following will be provided for the students:
1. MPLAB IDE v8.92 + MPLAB XC8 v1.33
2. Proteus v8.11

Delivery Process:
The instructor will conduct a short lecture about the RB0/INT external interrupt and Timer0 overflow interrupt. All inquiries pertaining to the latter must be referred to this manual.

Note:
Images and other contents in this manual are copyright protected. Use of these without consent from the author is unauthorized.

Part I. External Interrupt Source

An external interrupt source is a signal sent to MCU from an I/O device or other MCU. The signal should have a state change for the interrupt to be detected. This state changes are signal edges maybe rising edge, falling edge or both. An interrupt request to the CPU will be generated (if enabled) and an Interrupt Service Routine will be executed. An MCU could have more than one pin assigned as an external interrupt. The PIC16F877A MCU has the RB0/INT external interrupt source. If enabled, will generate an interrupt signal when it encounters a rising edge or falling edge signal transition. The RB0 pin should be configured as input if used as an interrupt source.

RB0/INT External Interrupt

The RB0/INT external interrupt sends interrupt requests from external peripheral devices or data transfer requests to the CPU to generate an external interrupt request. This interrupt is controlled by the OPTION_REG and INTCON registers (see pages 23-24 of the PIC16F87X data sheet for more information). Below are the configurations for each register:

OPTION_REG	bit 6	*INTEDG*	Interrupt Edge Select bit
INTCON	bit 4	*INTE*	RB0/INT External Interrupt Enable bit
INTCON	bit 1	*INTF*	RB0/INT External Interrupt Flag bit
INTCON	bit 7	*GIE*	Global Interrupt Enable bit

To enable/unmask the interrupt, INTE must be set to ‘1’. However, GIE will enable or enable all unmasked interrupt. Therefore to enable RB0/INT external interrupt, both INTE and GIE must be set to ‘1’. When an interrupt is generated, INTF will be set to ‘1’ by the CPU and has to be cleared manually via software.

External Interrupt Example

For this exercise, the DAVBL pin of the 74C922 keypad encoder is connected to RB0/INT pin. When a key is pressed on the keypad, the key number will be displayed on a 7-segment display. The number will stay on the display until another key is pressed. In this example, the MCU response is handled solely by the ISR.

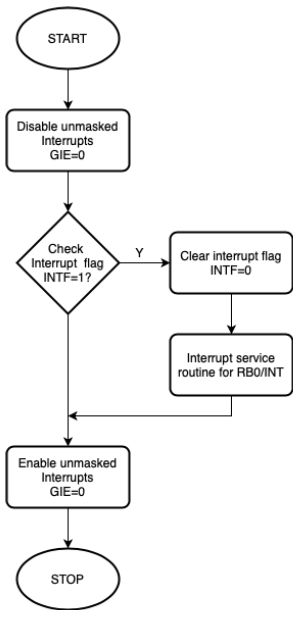
START
Disable unmasked Interrupts GIE=0
Check Interrupt flag INTF=1?
Y
Clear interrupt flag INTF=0
Interrupt service routine for RB0/INT
Enable unmasked Interrupts GIE=0
STOP

Fig. 1. Flowchart of ISR

Therefore no foreground routine will be written. The main() function shall only contain an only a blank infinite loop.

Setting Up the External Interrupt

The configuration for this exercise requires SW1 to trigger on the rising edge due to the way SW1 is biased (active-high). Since SW1 is connected to RB0, the RB0/INT external interrupt will be enabled. The following code snippet sets shows how the external interrupt will be setup:

```
INTEDG = 1;   // interrupt at rising edge

INTE = 1;     // enable RB0/INT external interrupt
INTF = 0;     // clears the interrupt flag
GIE = 1;      // enables all unmasked interrupt
```

There is only a single interrupt vector in PIC16F877A which means that all interrupt sources is pointed only to a single address in memory to handle the interrupt(s). To handle the RB0/INT external interrupt, an interrupt service routine (ISR) function must be written. This function is pointed to by the interrupt vector.

```
void interrupt ISR()
{
       GIE = 0;      // disables all unmasked interrupts to prevent interrupt overlap

       if (INTF)            // check the interrupt flag
       {
              INTF = 0;     // clears the interrupt flag

              /* write the interrupt service routine here
               for RB0/INT external interrupt */

       }

       GIE = 1;      // enable interrupts again
}
```

To prevent interrupt overlap, GIE must set to ‘0’ to disable all unmasked interrupts temporarily. To execute the correct ISR for the interrupt source, the interrupt flags must be checked first. The first interrupt flag to be checked has a higher priority. When INTF is equal to ‘1’, then execute the ISR but the flag must be cleared first. Set GIE back to ‘1’ to enable back all unmasked interrupts.

RB0/INT External Interrupt Exercise (LE3-1)

Open Proteus and create a firmware project for PIC16F877A with the filename “LE3-1.pdsprj” and construct the circuit according the I/O connections below:

Keypad Data	->	PORTD (RD3:RD0)
DAVBL	->	PORTB (RB0)
7-segment display w/ decoder	->	PORTC (RC3:RC0)

Open MPLAB IDE and create a new project via the project wizard with the name “CpE 3201-LE3”. Create a new source file with the filename “LE3-1.c” and add it to the project. Write the program to perform the required function both for the interrupt service routine. Refer to the flowchart in Figure 1. Build the program and debug if necessary. After successfully building the source code, simulate the program in Proteus (“LE3-1.pdsprj”) by loading the generated .hex file to the microcontroller. Observe the output and make sure that the counter will start/stop when the switch is pressed.

Part II. Timer Overflow Interrupt Source

A timer overflow occurs after a counter reach its terminal count. Incrementing a counter on its terminal count will make the value overflow and it automatically resets the counter. The example below is an 8-bit timer about to overflow:

11111111_2 (terminal count) + 1 (increment) = 100000000_2 (overflow)

The result of incrementing the counter above is a 9-bit value in which the lower 8 bits resets to “00000000”. The overflow situation can be used as an interrupt source, generating an interrupt request every time it happens. The PIC16F877A MCU has several counters/timers that can be configured to generate an overflow interrupt, one of which is the Timer0.

The time from counter reset to overflow can be determined by the F_{osc} (oscillator frequency), timer factor such as a prescaler or postscaler (if any). The PIC16F877A has an 8-bit timer module called Timer0 which can generate an interrupt upon overflow.

Timer0 Module

The Timer0 module timer/counter has the following features:

- 8-bit timer/counter
- Readable and writable
- 8-bit software programmable prescaler
- Internal or external clock select
- Interrupt on overflow from FFh to 00h
- Edge select for external clock

Timer mode is selected by clearing bit T0CS (OPTION_REG<5>). In Timer mode, the Timer0 module will increment every instruction cycle (without prescaler). If the TMR0 register is written, the increment is inhibited for the following two instruction cycles. The user can work around this by writing an adjusted value to the TMR0 register (see page 53 of the PIC16F87X data sheet).

Timer0 Overflow Interrupt

The Timer0 overflow interrupt is controlled by the following registers:

OPTION_REG	bit 2-0	*PS2:PS0*	Prescaler Rate Select bits
OPTION_REG	bit 3	*PSA*	Prescaler Assignment bit

OPTION_REG	bit 4	*T0SE*	TMR0 Source Edge Select bit
OPTION_REG	bit 5	*T0CS*	TMR0 Clock Source Select bit
INTCON	bit 2	*TMR0IE*	TMR0 Overflow Interrupt Enable bit
INTCON	bit 5	*TMR0IF*	TMR0 Overflow Interrupt Flag bit
INTCON	bit 7	*GIE*	Global Interrupt Enable bit

To enable/unmask the interrupt, TMR0IE must be set to ‘1’. However, GIE will enable or enable all unmasked interrupt. Therefore to enable Timer0 overflow interrupt, both TMR0IE and GIE must be set to ‘1’. When an interrupt is generated, TMR0IF will be set to ‘1’ by the CPU and has to be cleared manually via software.

Delay Function Using Timer0 Overflow Interrupt

The delay function used before in the previous exercise was just a simple empty loop and it is difficult to measure the actual timeout of a certain number of loops. To have a better and more precise delay, MCU timers shall be utilized. To easily monitor the counter status, an overflow interrupt shall be enabled. To achieve an accurate 1 second delay for example, we can use the timeout of a single overflow. The timeout can be calculated given the oscillator frequency of the MCU and a prescaler value. Therefore the timeout for 1 overflow in seconds with an oscillator frequency of 4MHz and 1:32 prescaler:

$$timeout = \frac{F_{osc}}{4}\ x\ prescaler\ x\ Timer\ Max\ Count \tag{1}$$

$$timeout = \frac{4MHz}{4}\ x\ 32\ x\ 256 = \frac{1}{1x10^6}\ x\ 8192 = 8.196x10^{-3}\ s$$

Equation 1 shows that the oscillator frequency is divided by 4 that is because Timer0 increments at every instruction cycle which requires 4 clock pulses (for the PIC16 architecture). Substituting the values to equation 1, the overflow cycle is about 8.196 ms. Therefore, we can divide the desired delay period with the overflow timeout (equation 1) to come up with the number of overflows required.

$$no.of\ overflows = \frac{delay\ period}{timeout} \tag{2}$$

So for a 1 second delay:

$$no.of\ overflows = \frac{1s}{8.196x10^3 s} = 122.011 \approx 122$$

Therefore the delay function shall count the number of overflows required to come up with the desired period.

Setting Up the Timer0 Overflow Interrupt

Based on the example above for a 1 second delay, we start with setting up the OPTION_REG and INTCON registers:

```
OPTION_REG = 0x04;  // PS2:PS0 - prescaler 1:32
                    // PSA - prescaler assignment to TMR0
                    // T0CS - internal instruction cycle clock
                    // T0SE - not used since clock is internal

T0IE = 1;     // enable Timer0 overflow interrupt
T0IF = 0;     // clears the interrupt flag
GIE = 1;      // enables all unmasked interrupt
```

Since there is only one interrupt vector in PIC16F877A, the ISR of the Timer0 overflow interrupt can be shared with ISRs from other interrupt sources. The following interrupt function shows the combined ISRs of RB0/INT external interrupt and Timer0 overflow interrupt. However, only one ISR will be executed when an interrupt occurs. The interrupt flag that is evaluated first will have the higher priority.

```
void interrupt ISR()
{
       GIE = 0;      // disables all unmasked interrupts to prevent interrupt overlap

       if (INTF)            // check the interrupt flag for RB0/INT
       {
              INTF = 0;            // clears the interrupt flag

              /* write the interrupt service routine here
               for RB0/INT external interrupt */

       }
       else if(T0IF)        // check the interrupt flag for Timer0
       {

              T0IF = 0;            // clears the interrupt flag

              /* write the interrupt service routine here
               for Timer0 overflow interrupt */

              count_flag = 1;      // this is a global variable which will be
                                   // in the main routine (toggle)

       }
       GIE = 1;      // enable interrupts again
}
```

The `count_flag` is a global variable that will be checked in the `delay()` to signify an overflow. Once the required number of overflows for the desired timeout was reached, `delay()` exits.

RB0/INT External Interrupt Exercise (LE3-2)

Modify "LE3-1.pdsprj" Proteus project file by connecting an LED to RA0. Save it as "LE3-2.pdsprj".

Save "LE3-1.c" as "LE3-2". Remove "LE3-1.c" from the source files then add "LE3-2.c". Modify the program by adding the Timer0 overflow interrupt. This time, the foreground routine will blink the LED at an interval of 1 second. Revise the `delay()` so that it will utilize the Timer0 overflows to achieve an accurate delay. Pressing the keypad will still display the key in the 7-segment display as in LE3-1.

Build the new source code. After successfully building, simulate the program in Proteus ("LE3-2.pdsprj") by loading the generated .hex file to the microcontroller. Observe the output and make sure that the counter will start/stop when the switch is pressed and the count interval is 1 second.

Part III. Hands-On Exercise

For this exercise, use the Proteus project file in LE3-1 "LE3-1.pdsprj".

Create a new source file with the filename "LE3-3.c" and add it to the project "CpE 3201-LE3". Write a program that will perform the following:

a. Upon start-up, the display would count-up from 0-9 then recycles at an interval of about 0.8 seconds.
b. When keys 1-9 in the keypad is pressed, the key number will be displayed and continues to count from that number. Counter rolls back to zero after 9.
c. Use the RB0/INT ISR to load the counter with the key pressed. For example if key 5 is pressed, it loads the counter with 5 then continues to count up.
d. Use TMR0 overflow interrupt for the `delay()`. Calculate the number of overflows required for a 0.8 second interval.
e. Foreground routine will **only** handle the counting. The `delay()` must use timer overflows.

Furthermore, please refer to the following instructions.

1. Draw the flowchart for the foreground and interrupt service routines. Use the recommended flowcharting application (draw.io) and annotate the necessary information. Export flowchart as PDF as "LE3-3.pdf".
2. Build the "CpE 3201-LE3" project with the source code "LE3-3.c". After successfully building the source code, simulate the program in Proteus ("LE3-1.pdsprj") by loading the generated .hex file to the microcontroller.
3. Observe the output of the program and make sure it functions as required.
4. Submit the flowchart ("LE3-3.pdf"), Proteus project file ("LE3-1.pdsprj") and the source code ("LE3-3.c") in Canvas.

——————-————-————— End of Practical Activity #3 ————————————————

Assessment

Criteria	Outstanding (4 pts)	Competent (3 pts)	Marginal (2 pts)	Not Acceptable (1 pt)	None (0 pts)
Interrupt Setup	Interrupts properly configured.	-	-	Interrupts not configured.	No project created.
Functionality	The developed solution works without issues	-	The system has issues.	The system did not work properly.	No project created.
External and timer overflow interrupts	External and timebase timer overflow interrupts demonstrated successfully.	-	-	Failed to demonstrate external and timebase timer overflow interrupts.	No project created.

Copyright Information

References

- PIC16F87X Data Sheet, Microchip Technology Inc. 2003.
- CpE 3201 Lecture Notes on Numeric Keypad Interfacing (available in Canvas).
- CpE 3201 Lecture Notes on Interrupts (available in Canvas).

change log:

Date	Version	Author	Changes
October 15, 2018	1.6	Van B. Patiluna	Original laboratory guide for CpE 426N.
February 17, 2021	1.0	Van B. Patiluna	- Revised the activity objectives. - Removed the LKS Expansion Board and other references to MB90F387S MCU. - Changed some of the procedure to reflect the new software tools. - Remove the Mini-Design activity and replaced with Hands-on Exercise.
February 17, 2021	1.1	Van B. Patiluna	- Revised the activities LE3-1, LE3-2 and LE3-3.
February 18, 2021	1.1.1	Van B. Patiluna	- Corrected some factual errors.