EE425 Homework #3 Anthony Ramos

The assembly file for this assignment file can be found here

Introduction

The objective of this experiment is to introduce the concept on interrupts using the PIC18F4520 microcontroller. An interrupt at its core is an asynchronous signal that disrupts the main program to handle other tasks (usually ones that require a fast response). Within interrupts there exists a hierarchy which will be explained in detail later. This results in two types of priority: Low priority interrupts (LPIs) and high priority interrupts (HPIs). Interrupts are managed by the Interrupt Service Routine (ISR) which are similar to a regular subroutine but specifically designed for interrupts.

In terms of the PIC18F4520 microcontroller and this experiment, three interrupts will be utilized to interrupt the `Mainline` program whose only purpose is to generate a pulse train at the `RC2` pin. The purpose of each interrupt will be explained in more detail later. For now, we need to understand how interrupts are initialized in the assembly language.

Setting Up Interrupts

The PIC18F4520 microcontroller has a number of interrupts which are all set to high-priority level by default. We will make use of two external interrupts `INT0` and `INT1` to serve as a high and a low priority respectively. Generally speaking, these interrupts consist of three bits which allow for operation control. It should be noted that interrupt `INT0` does not have a priority bit that can be manipulated because it reserved to be used as only a High Priority interrupt.

Table 1: Summary of interrupt sources

Bit	Description	Function
IF	Interrupt Flag Bit	Indicate if an interrupt request is present
IE	Interrupt Enable Bit	Redirect (i.e. branch) the program execution to the interrupt vector address
IP	Interrupt Priority Bit	Select Priority (High or Low)

Prior to setting up an interrupt however, we must set some global settings. First, we must set the `IPEN`, `GIEH`, and `GIEL` bits which allow the use of interrupts, enable low and high interrupts respectively. Upon setting those bits, the High and Low Priority interrupts can be set up.

To set up the High Priority interrupt, we must set the enable bit and clear the flag bit. Again, we do not need to set the priority bit because `INT0` is strictly reserved for use as a High priority interrupt. In terms of setting the Low Priority interrupt, we must clear the interrupt priority to indicate that `INT1` will be used a Low Priority interrupt. The flag bit must also be cleared.

```
; Global interrupt setup
bsf RCON, IPEN ; Intialize two-level interrupt priority
bsf INTCON, GIEH ; Enable High Priority Interrupts
bsf INTCON, GIEL ; Enable Low Priority Interrupts
bsf INTCON, RBIE ; Enable the RB port change interrupt
; INT0/RB0 (HPI Setup)
bcf INTCON, INT0IF ; Clear INT0 Flag bit
bsf INTCON, INT0IE ; Enable INT0 external interrupt
; INT1/RB1 (LPI Setup)
bcf INTCON3, INT1IF ; Clear INT1 Flag bit
bsf INTCON3, INT1IE ; Enable INT1 external interrupt
bcf INTCON3, INT1IP ; Clear INT1 priority bit (i.e set Low priority)
```

Figure 1: Setup of interrupts

With the interrupts set up, we must define the behavior of both the LPSIR and HPSIR.

The LPSIR

The purpose of the LPSIR is to generate a sequence of pulse trains at pins `RA1`, `RA2`, and `RA3`. The required sequence is outlined below:

Table 2: Defined pulse train sequences for pins RA1 to RA3

STEP	RA3	RA2	RA1
1	on	on	on
2	on	on	off
3	on	off	on
4	on	off	off
5	off	on	on
6	off	on	off
7	off	off	on
8	off	off	off

While these pulse trains are running, the pulse train at pin `RC2` must remain logically low (i.e. cleared) until the `Mainline` program resumes (i.e. when the LPISR terminates). This can be achieved by utilizing the `bcf` instruction to clear the `RC2` bit. In addition, the subroutine `PulseTrainGen`, which was utilized in the previous experiment, is used to generate the required pulse trains. Its behavior is given below:

```
PulseTrainGen
    bsf PORTA,RA3 ;ON
    bsf PORTA,RA2 ;ON
    bsf PORTA,RA1 ;ON
    rcall LoopTime ;0.1ms delay
    rcall LoopTime ;0.1ms delay
    bsf PORTA,RA3 ;ON
    bsf PORTA,RA2 ;ON
    bcf PORTA,RA1 ;OFF
    rcall LoopTime ;0.1ms delay
    rcall LoopTime ;0.1ms delay
    bsf PORTA,RA3 ;ON
    bcf PORTA,RA2 ;OFF
    bsf PORTA,RA1 ;ON
    rcall LoopTime ;0.1ms delay
    rcall LoopTime ;0.1ms delay
    bsf PORTA,RA3 ;ON
    bcf PORTA,RA2 ;OFF
    bcf PORTA,RA1 ;OFF
    rcall LoopTime ;0.1ms delay
    rcall LoopTime ;0.1ms delay
    bcf PORTA,RA3 ;OFF
    bsf PORTA,RA2 ;ON
    bsf PORTA,RA1 ;ON
    rcall LoopTime ;0.1ms delay
    rcall LoopTime ;0.1ms delay
    bcf PORTA,RA3 ;OFF
    bsf PORTA,RA2 ;ON
    bcf PORTA,RA1 ;OFF
    rcall LoopTime ;0.1ms delay
    rcall LoopTime ;0.1ms delay
    bcf PORTA,RA3 ;OFF
    bcf PORTA,RA2 ;OFF
    bsf PORTA,RA1 ;ON
    rcall LoopTime ;0.1ms delay
    rcall LoopTime ;0.1ms delay
    bcf PORTA,RA3 ;OFF
    bcf PORTA,RA2 ;OFF
    bcf PORTA,RA1 ;OFF
    rcall LoopTime ;0.1ms delay
    rcall LoopTime ;0.1ms delay

    return
```

Figure 2: `PulseTrainGen` Subroutine

In this subroutine, we utilize the `bsf` and `bcf` instructions to set or clear the necessary bits so that it follows the sequences given above. In addition, we utilize two `LoopTime` subroutines to introduce a 0.2ms delay between each sequence. When the subroutine ends, the LPISR terminates as well and the `Mainline` program resumes. We also must clear the interrupt flag (`IF` bit) so that other interrupts within the programs can occur. It is important to note that besides normal termination, a Low Priority interrupt can only be stopped by a High Priority interrupt.

```
;;;;;;; LPISR ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
LPISR
    movff STATUS, STATUS_TEMP          ;save STATUS and W
    movf W,WREG_TEMP


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ;Write your code here for the following tasks
    bcf PORTC, RC2 ; Clear pulse train from Mainline
    ; Initiate counting bits
    ; You MUST do this using a separate SUBROUTINE,
    ; and inside that subroutine you may create
    ; yet another subroutine which counts LoopTime (0.1ms)
    ; Clear all counting bits from LPISR
    rcall PulseTrainGen
    bcf INTCON3, INT1IF; Clear LP Interrupt FLAG
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;



    movf WREG_TEMP,W                   ; restore STATUS and W
    movff STATUS_TEMP,STATUS
retfie
```

Figure 3a: LPISR

The HPSIR

When the HPI is trigged, the pulse train at pins RC2, RA1, RA2, and RA2 must all be cleared. At the same time, pin RC1 will triggered to a logical high and remain high. If the LPISR running at the time the HPI is triggered, then the LPISR will be immediately terminated. The HPI interrupt is set to run indefinitely and can only be ended when an external signal called the Human-Input Signal (HIS) is initiated via the RE2 pin. The HIS serves as break condition which will trigger pin RC1 to a logical low and resume the Mainline program.

```
;;;;;;; HPISR ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
HPISR
    bsf PORTC,RC1;Set Signal that we are entering HPISR - DO NOT MODIFY!!


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ;Write your code here for the following tasks
    bcf PORTC, RC2  ;Clear pulse train from RC2
    ;Clear all counting bits from LPISR
    bcf PORTA, RA1
    bcf PORTA, RA2
    bcf PORTA, RA3
    ; Loop to check for human input
Loop
    btfss PORTE,RE2 ; Check if Human Input Signal (HIS) is 1 (If so, skip next instruction)
    bra Loop


    bcf INTCON3, INT1IF ; Clear LP Interrupt FLAG
    bcf INTCON, INT0IF  ; Clear HP Interrupt FLAG
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;




    bcf PORTC,RC1;Set Signal that we are Leaving HPISR -  DO NOT MODIFY!!

    MOVLF  B'11111111',TMR0H ;DO NOT MODIFY
    MOVLF  B'00000000',TMR0L ;DO NOT MODIFY

retfie FAST
```

Figure 3b: HPISR

Simulation

We will simulate the program to verify its correctness. The Logic Analyzer tool will be utilized to monitor the behavior of RC1, RC2, RB0, RB1, RE2, RA1, RA2, RA3 signals. We will also use a new tool called a Stimulus that will used to trigger the interrupts. A summary of the pins monitored is given below.

Table 3: Summary of pins

PIN	TYPE
RB0	External Interrupt
RB1	External Interrupt
RE2	External Signal
RC1	Output
RC2	Output
RA1	Output
RA2	Output
RA3	Output

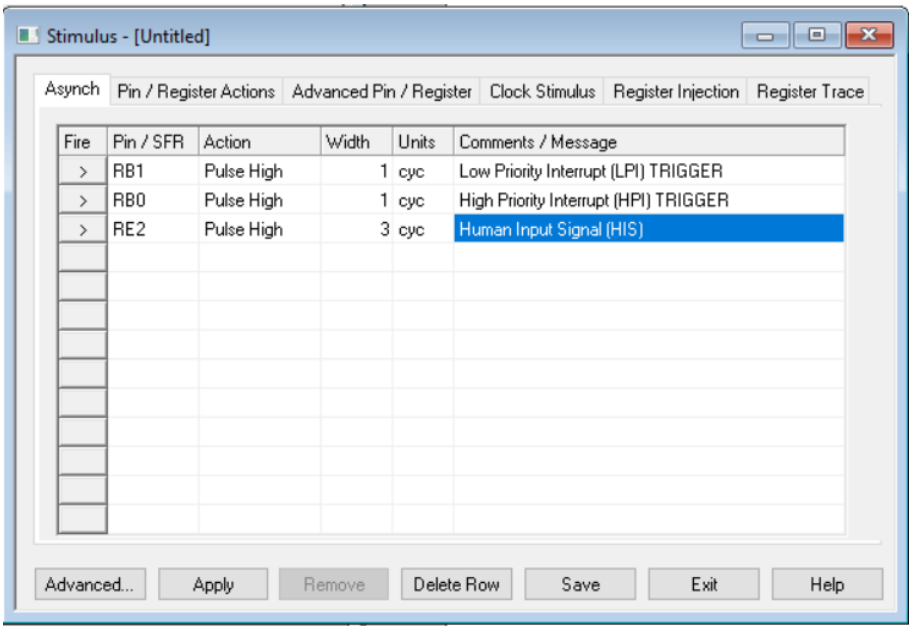

Figure 4: Stimulus window for triggering interrupts

We begin the simulation and as mentioned earlier, the program initially begins by outputting a pulse train at the `RC2` pin continuously. In the Stimulus window we trigger the LPI and obtain the following result:

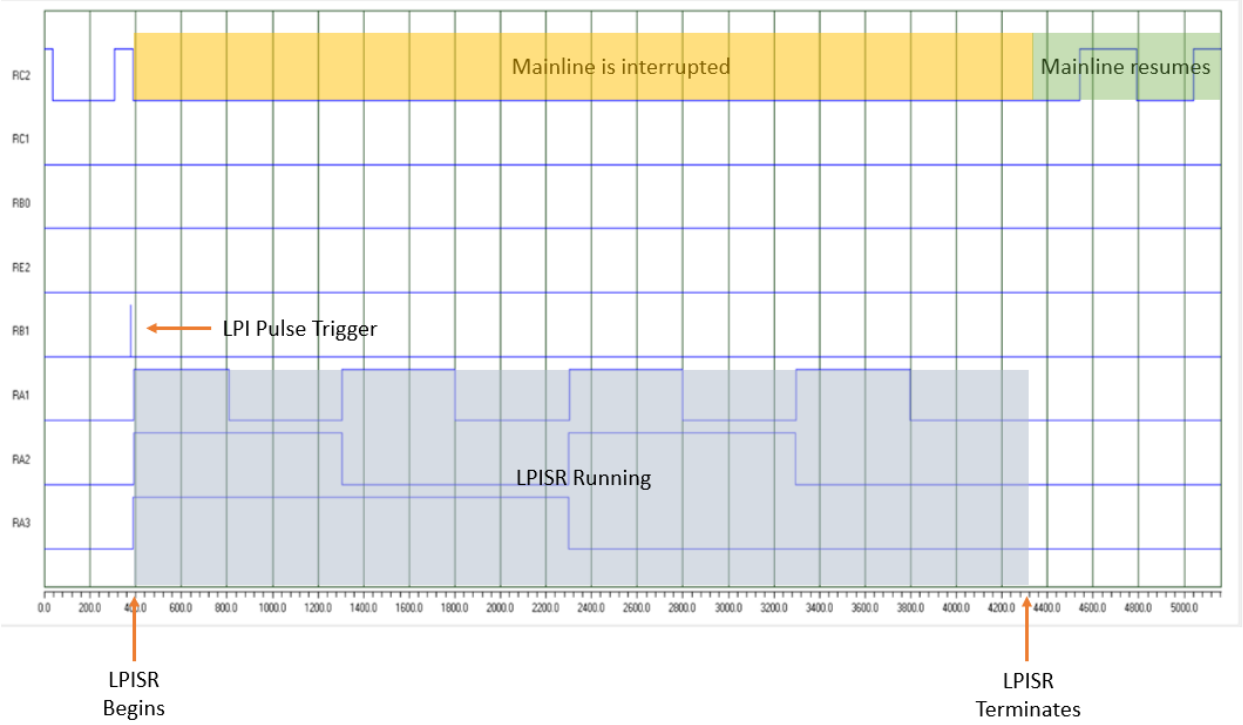

Figure 5: Waveforms when LPI is triggered

When the LPI is triggered (indicated by the pulse in `RB1`), the `RC2` pin is toggled to a logical low (i.e. cleared) and the pulse trains at pins `RA1` to `RA3` begin. When the pulse train sequences end, the LPISR ends and the `Mainline` program resumes. Next, we test the correctness of the HPISR. Consider the following waveforms:

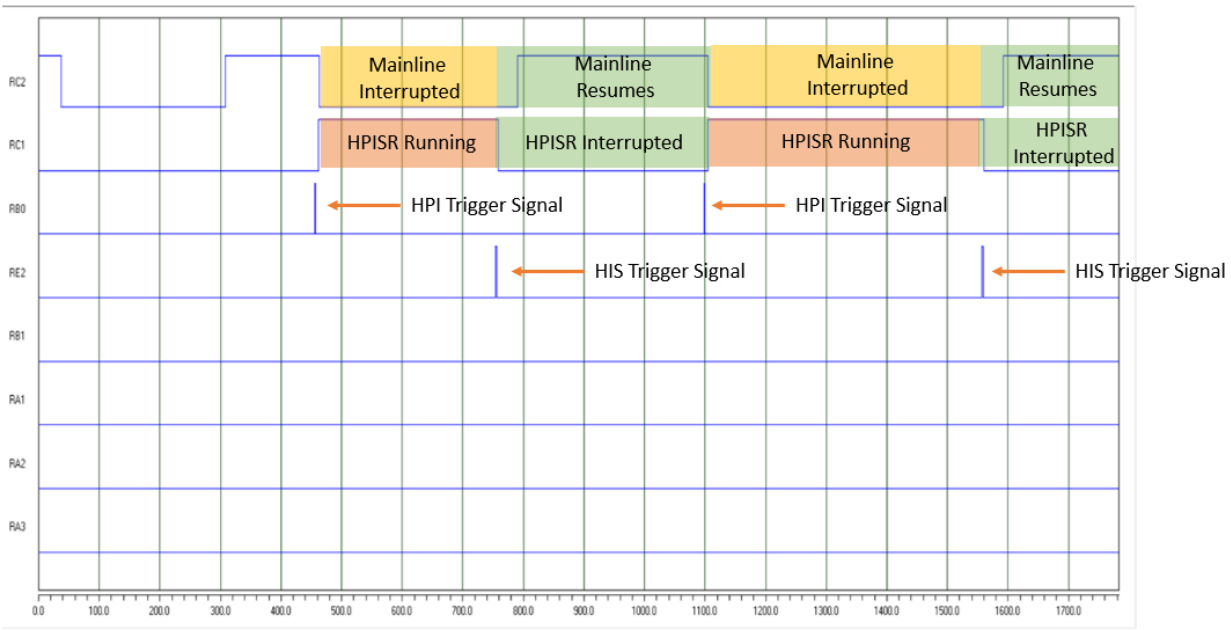

Figure 6: Waveforms when HPI and HIS are triggered

When the HPI is triggered (indicated by the pulse in `RB0`), the `RC2` pin is toggled to a logical low (i.e. cleared) and pin `RC1` is toggled to a logical high. As pin `RC1` remains high the HPISR continuously checks for the HIS. If the HIS is not triggered, pin `RC1` remains logically high indefinitely. If the HIS is triggered (indicated by the pulse in `RE2`), the HPISR is interrupted, pin `RC1` is cleared, and the `Mainline` program resumes. This is the expected behavior because since both signals are of high priority, they can interrupt one another at any time. Having now verified both the LPISR and HPISR, we will conduct one last simulation.

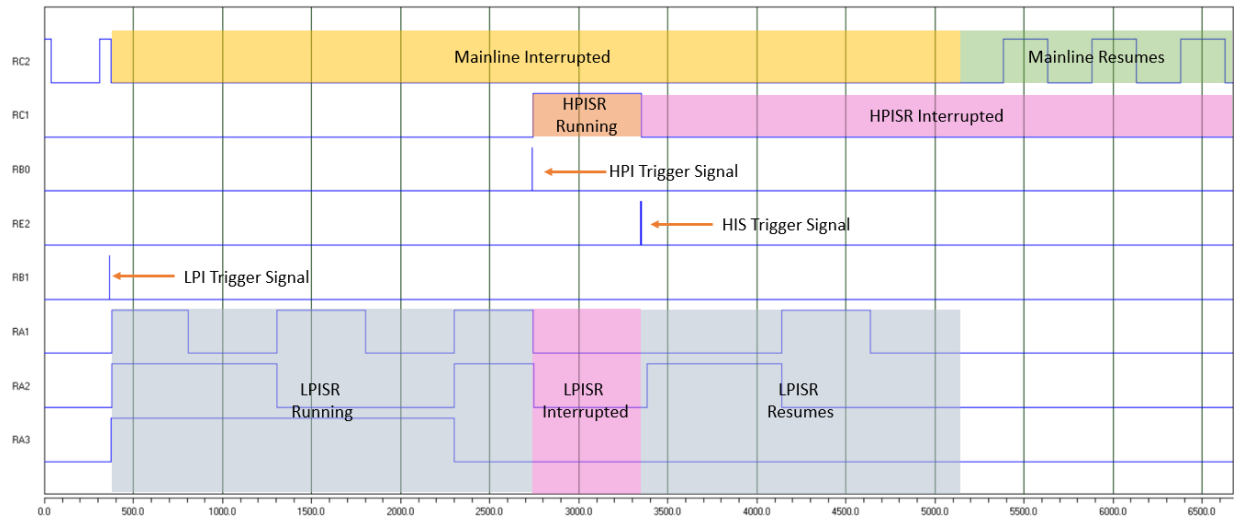

Figure 7: Waveforms when LPI, HPI, and HIS are triggered

In this simulation run, the `Mainline` program initially begins when the LPI signal is triggered. The LPISR begins, pin `RC2` is cleared, and pulse trains at pins `RA1` to `RA3` run for some time until the HPI signal is triggered. When triggered, the LPISR is interrupted and the HPISR begins with setting pin `RC1` to a logical high and runs indefinitely for some time. The HIS is triggered and the HPISR is interrupted, the `RC1` pin is cleared, and in our case, the control flows from the HPISR back to the LPISR. The LPISR resumes and when it terminates, the `Mainline` program resumes. Live simulations for each test can be found here.

Conclusion

It can be confidently stated that the behavior of the program is correct given the waveforms produced. The waveforms themselves follow the expected interrupt behavior. That is, a HPI always interrupts a LPI, a HPI can be only be interrupted by another HPI, and new events are lost if a LPI occurs during another LPI or HPI. Overall, we learned the basic function of an interrupt, how to set and use interrupts on the PIC18F4520 microcontroller, and understood why interrupts may be necessary to use at times for controlling programs.