THE CITY COLLEGE OF NEW YORK
Department of Electrical Engineering

EE425 Computer Engineering Laboratory – Fall 2021

Experiment 3 – Interrupt Service Routine (ISR)

Objective: This experiment is designed to show the handling capabilities of interrupts of the PIC18F4520 microcontroller. Use the microchip to detect external interrupts using the appropriate controller pins.

Preliminaries:

- This experiment will use the "*P3_template.asm*" that has been provided to you.
- In this experiment you will use the Logic Analyzer embedded in the IDE—please see my MPLAB IDE Tutorial which shows how to set it up and how to use it. Do not proceed any further into this assignment until you know how to use the Logic Analyzer.
- Add the channels **RC2, RC1, RB0, RE2, RB1, RA1, RA2,** and **RA3** to the *Logic Analyzer* window and let the animation run for some time. Your *Logic Analyzer* window should look as in Figure 1:

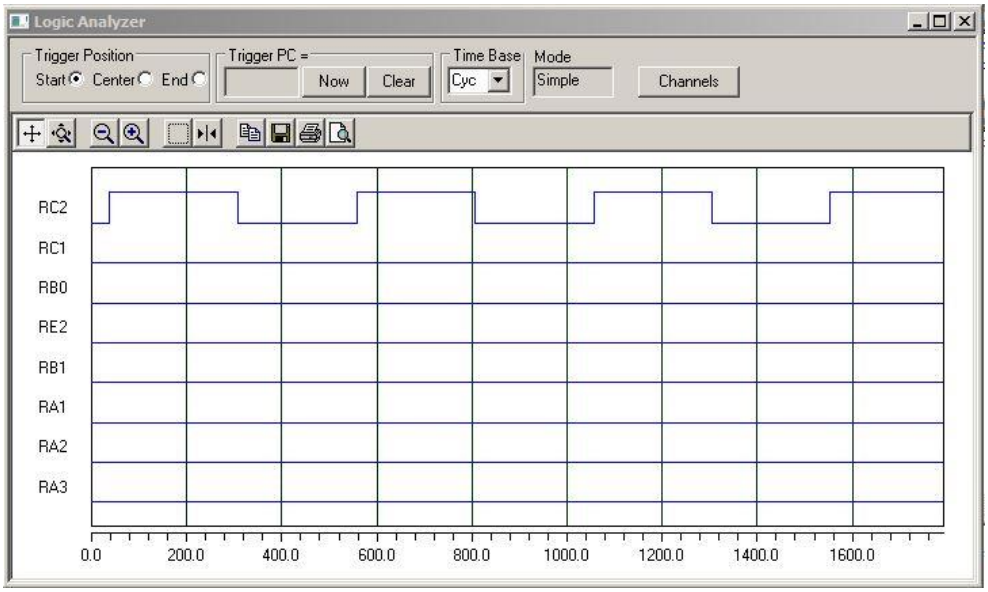

Figure 1. Pulse train at bit RC2.

- Click on **Debugger >> Stimulus>> New Workbook.** This will launch a new window called *Stimulus*. The rows of cells in this new window will be empty, but you can then populate one cell at the time by clicking on it and adding the appropriate signal that corresponds to an external stimulus. For now, you can add pins **RB1**, **RB0**, and **RE2** to so that your *Stimulus* window looks as in Figure 2. You must make sure that the *Action* and *Width* settings for each pin are the same as show in the figure. You should also write the comments as in the right column in order to make testing easier later.

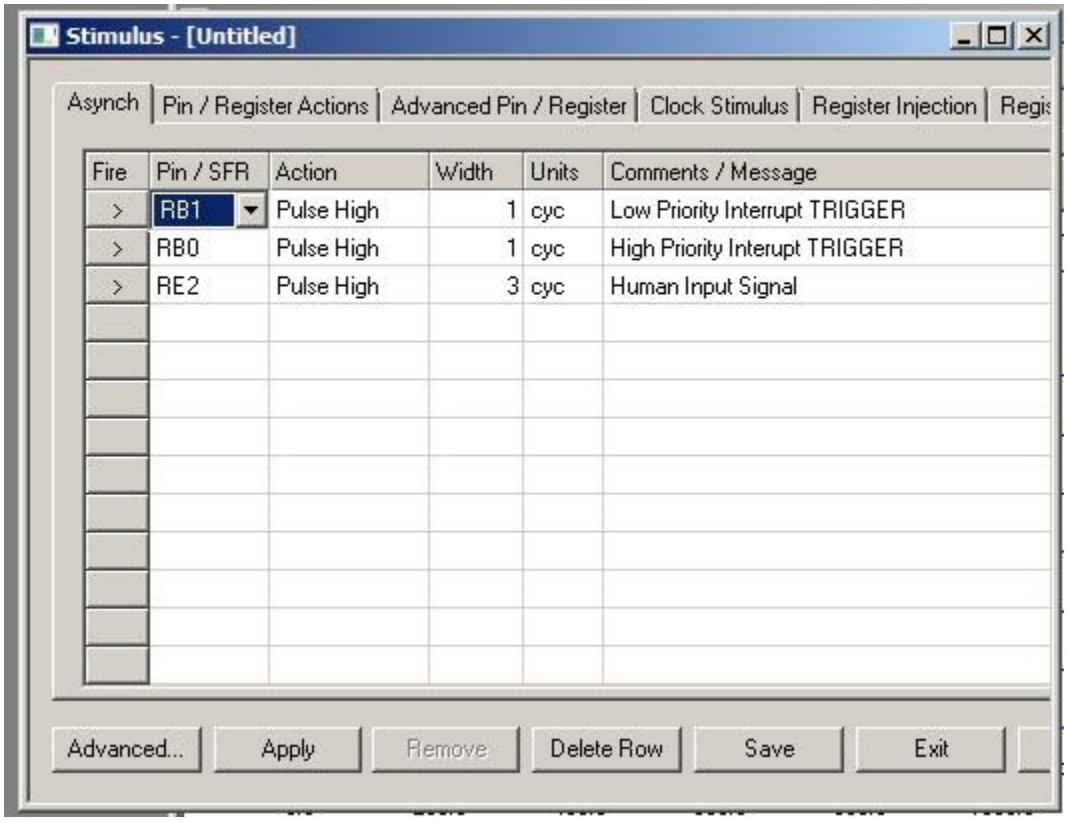

Figure 2. External stimuli signals.

Specific Tasks: The microcontroller should be programmed in a way so that it executes the following tasks (READ CAREFULLY):

Please note that the following tasks must be programmed into a single *.asm* file.

1. Write assembly code to set INT0 (bit **RB0** in PORTB) as your High Priority Interrupt and INT1 (bit **RB1** in PORTB) as your Low Priority Interrupt. These bits will be interpreted as the external, **asynchronous** signals to be used as your interrupt triggering signals in order to execute an interruption of the mainline program. Note that **RB0** and **RB1** will be monitored from the *Logic Analyzer* window shown in Figure 1, but they will be triggered from the *Stimulus* window in Figure 2. The signal on **RE2** is an additional external input signal that will be supplied (or triggered) by the human programmer or program user—we will call this asynchronous signal on **RE2** the *Human Input Signal*. **During the animation of the main program, whenever you want to trigger (or fire) an interrupt event on either RB0, RB1, or RE2, you must click on the corresponding *Fire* button to the left of each signal in the *Stimulus* window, and then see what happens on the *Logic Analyzer* window.**

2. Write assembly code to implement the following interrupt event hierarchy.

    a. As the mainline program, that is, the main program that will be interrupted using the external interrupts, let the bit **RC2** of PORTC output a periodic pulse train with duty cycle=50% and a half-period of 0.1*ms*. This pulse train in the mainline must run indefinitely **unless** an interrupt is triggered. Please note that this pulse train is already being generated in the *P3_template.asm.*

b. When an asynchronous, low priority (LP) interrupt event occurs, that is, when you trigger or fire the interrupt signal **RB1** from the *Stimulus* window, the following actions must take place:
   - The bits **RA1, RA2,** and **RA3** of PORTA must begin stepping through the sequence shown in the table below. The code you write must count up from STEP 1 all the way to STEP 8 in the table. You must program a 0.2ms delay in between steps.

STEP	RA1	RA2	RA3
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

   - All other activities must stop: pulse train from pin **RC2** must be stopped (cleared) so that it is LOW for as long as the LP-ISR is running.
   - Upon returning to the main program, bits **RA1, RA2,** and **RA3** of PORTA must be cleared and the pulse train from pin **RC2** must resume execution.
   - As a check, you must trigger the Low Priority Interrupt signal, **RB1**, from the *Stimulus* window to test the functionality of your LP-ISR code.

c. When an asynchronous, high priority (HP) interrupt event occurs, that is, when you trigger or fire the interrupt signal RB0 from the *Stimulus* window, the following actions must take place:
   - The pulse train at bit **RC2** must be cleared and the bits **RA1, RA2,** and **RA3** of PORTA must be also cleared.
   - The HP-ISR must then enter into an indefinite loop. In order to leave the indefinite loop, write a breaking condition that is dependent upon the Human Input Signal coming from bit **RE2**. That is, the indefinite loop must break **only when RE2 = 1**, otherwise the indefinite loop will continue to run until human input is sensed. As a check, you should trigger the Human Input Signal, **RE2**, from the *Stimulus* window to test the functionality of your breaking condition. This indefinite loop and its breaking condition is designed to wait for immediate user's attention (via external input **RE2**) and only resume execution of the program after immediate input has been provided by the user.
   - As a check, you must trigger the High Priority Interrupt signal, **RB0**, from the *Stimulus* window to test the functionality of your HP-ISR code.

Guidelines:

The following are some steps that you need to consider when dealing with interrupts:
- Configure the registers that control the interrupts.
- Configure the required pins as inputs (external interrupts).
- Code you Interrupt Service Routines (ISR).
- Program your code so that is stores the more important registers of your program before executing the ISR.
- Once an interrupt has been served, clear the corresponding flag, so that the microcontroller can accept new interrupts.
- Refer to Ch. 9 of the textbook and also Ch. 9 in the microcontroller datasheet for more details about interrupts.