ENGR 270 Lab #7 Online – Data Serialization

Objectives

Apply basic serialization concepts used in storage systems, USB, Bluetooth, Local Area Network and other computer interfaces and data transmissions. Learn to setup and use the EUSART subsystem in the PIC18F1220; learn concepts of serial I/O; gain more practice using interrupts.

Preparation

For sending and receiving a single bit of information, pins like RA0 or RB1 are fine; but to send an ASCII character would require at least 7 bits (for example: RA6 RA5 RA4 RA3 RA2 RA1 RA0) which is approximately half of the available I/O pins on the PIC18F1220. To receive an ASCII character would require another 7 pins, for a total of 14 pins (remember, the PIC only has 18 pins total!)

A better approach for transmitting characters is to use *serial* I/O. The basic idea is to take the bits comprising an ASCII character, and send them to or from the PIC *one bit at a time*. This requires only two pins: one for sending serial data, and another for receiving serial data.

One complication to this approach is knowing when to send (or read) each bit of the character. There must be an agreed-upon rate at which a character's bits are sent or received. This rate is called the "baud rate," and is expressed as bits per second (bps). Typical baud rates for serial transmission are 115200, 57600 or 9600 bps (systems in the 1970s used rates as low as 1200, 300 or 110 bps).

To perform serial I/O with a PIC processor, you need to do a number of things:

- setup the EUSART subsystem to enable serial I/O;
- setup the baud rate of the system;
- setup interrupts if you want interrupt-driven I/O (especially useful for receiving characters, since you may not know when a character is going to arrive); and
- read from or write to special SFRs to actually transmit or receive a character.

Experiment 1. Introduction to Serial I/O

- Start up MPLAB
- Select menu "File>New Project" and make the following selection and click Next:
  * Select "Microchip Embedded"/"Standalone Project"
  * Select "PIC18F1220." From list of processors
  * Select "Simulator" from the Tools Menu
  * Select "mpasm: from the compile tool chain
  * Enter project name and folder for your Counter project (call it Echo)

  Note: It is recommended that you create a new project directory for each new lab experiment to avoid the most common problem of including old files in new projects.
  * Click Finish

- Once you have finished the Project Wizard, you should see a window with different sub-windows inside it. In the upper-left area of the window should be a window with the name of your project ("Echo"), followed by "Header Files," "Important Files," and other folder names. Note that if you were previously working on a different project, you may see that project listed here as well. Find the name of the project you're creating now, and *right-click* on the project name, then select "New/AssemblyFile.asm" or “New/pic_8b_simple.asm”

- Enter a "File Name" of your choice. main.asm is always a good choice, but you can choose something more descriptive if you like. Don't enter anything in the Folder: area. You should see a full path under "Created File:" which should end with "\Echo.X\main.asm" **Remember what this full path is: you’ll need to use it in Experiment 1.** Hit "Finish"

  Note: On the upper-middle/right side of the window should be a tab labeled "main.asm" This is where you'll enter your program's source code. You can close the tab labeled “Start Page”)

- Right click on the project name (Echo) and select Set Configuration/Customize…

- In the Categories window, select **Simulator**, then under Option Categories select **Uart1 IO Options** from the dropdown menu. Check the box that says “**Enable Uart1 IO**” and next to Output select **Window** (click **Unlock** first if you cannot change any options). Click **OK**.

- Any code already in the main.asm window can be deleted (select it all and press the delete button, or CTRL-X to cut it)

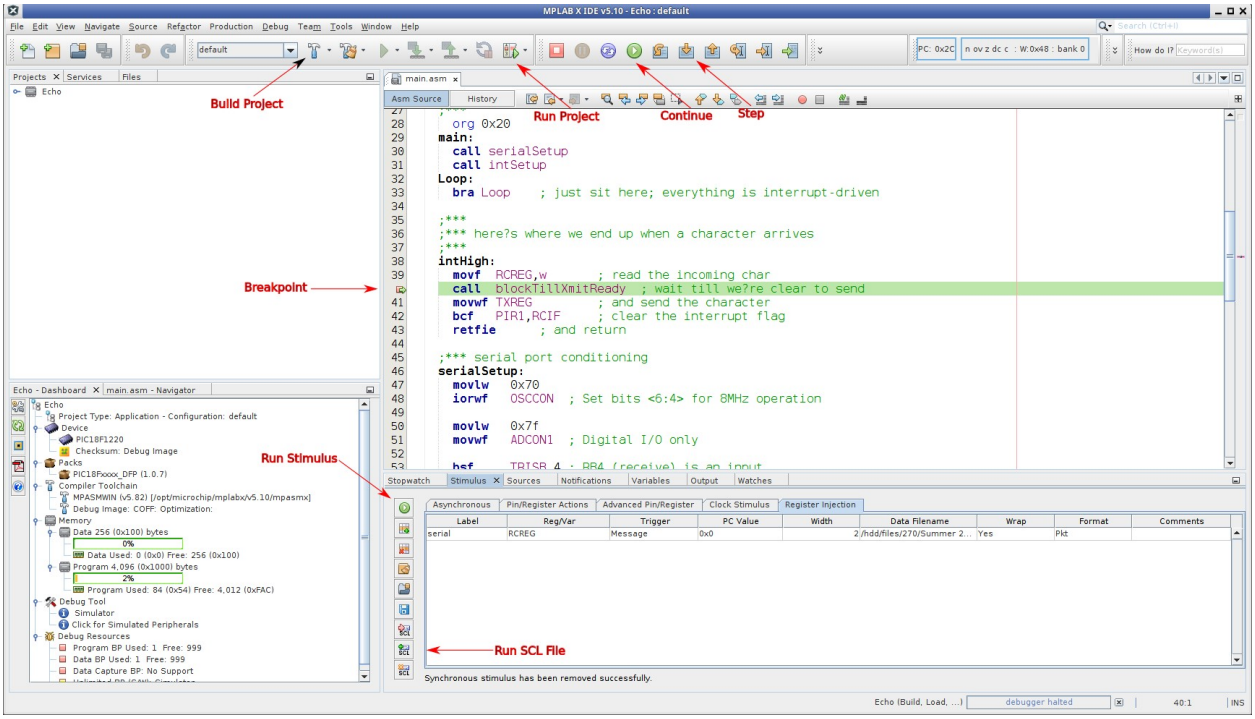
- Copy the code below into your newly created main.asm file. Save your code by selecting File/Save" or by just hitting CTRL-S. Note that some of the indenting may not match what's shown below. You can indent entire sections of text in MPLAB-X by selecting the text and hitting TAB (you can un-indent by hitting SHIFT-TAB).

```
;-------------------------------------------------------------------------------
; File: main.asm
; Desc: Experiment 1 – Simple Echo Test
;  * Read from EUSART and write to EUSART
; Last Update: June, 2020
; Auth: Class
;-------------------------------------------------------------------------------

; Serial port simulation test code

    list    p=18f1220 ;processor type
    radix   hex       ;default radix for data
    ; Disable Watchdog timer, Low V. Prog
    config  WDT=OFF,LVP=OFF,OSC=INTIO2
    include p18f1220.inc    ; include useful symbols

;***
;*** main code starts here
;***
    org 0x00
    bra main

;***
```

```
;*** high-priority interrupt jumps here
;***
    org 0x08
    bra intHigh ; put your code at intHigh

;***
;*** main code begins here
;***
    org 0x20
main:
    call serialSetup
    call intSetup
Loop:
    bra Loop    ; just sit here; everything is interrupt-driven

;***
;*** here's where we end up when a character arrives
;***
intHigh:
    call  blockTillXmitReady  ; wait till we're clear to send
    movf  RCREG,w             ; read the incoming char
    movwf TXREG               ; and send the character
    bcf   PIR1,RCIF           ; clear the interrupt flag
    retfie                    ; and return


;*** serial port conditioning
serialSetup:
    movlw   0x70
    iorwf   OSCCON    ; Set bits <6:4> for 8MHz operation

    movlw   0x7f
    movwf   ADCON1    ; Digital I/O only

    bsf     TRISB,4    ; RB4 (receive) is an input.
    bsf     TRISB,1    ; RB1 (transmit) is initially an input.
                       ; The EUSART module will adjust this as needed
    bsf     TXSTA,TXEN ; Enable transmission
    bsf     TXSTA,BRGH ; Enable high-speed baud

    bsf     RCSTA,CREN ; enable continuous received
    bsf     RCSTA,SPEN ; enable serial port

; baud rate generation
    bsf     BAUDCTL,BRG16 ; enable 16-bit baud rate
    movlw   .207           ; =9600 baud with 8MHz FOSC
    movwf   SPBRG         ; Table 16-3, PIC18F1220/1230 Datasheet
    clrf    SPBRGH        ; high byte=0
    return

;*** interrupt setup
intSetup:
    movlw 0x7f
    movwf ADCON1         ; all digital I/O

    bcf PIR1,RCIF        ; clear any pending interrupt
    bsf IPR1,RCIP        ; high priority
    bsf PIE1,RCIE        ; enable int on character reception
```

```
    bsf RCON,IPEN        ; allow high and low int levels
    bsf INTCON,GIEH      ; enable high ints
    bsf INTCON,GIEL      ; and low ints
    return

;*** useful code for sending a character (DESTROYS W)
; wait until we're clear to transmit
blockTillXmitReady:
    movf    PIR1,w            ; Bit 4 is TXIF (set when clear to send)
    andlw   0x10              ; mask off that bit
    bz      blockTillXmitReady ; not set yet
    return

    end        ; Indicates the end of the program.
```

The following image may be helpful in locating controls in MPLAB:



Step II. Simulate Echo Project

Use the simulation feature of MPLAB-X to test the functionality of your counter by following the steps shown:

a) Click on the line number next to the statement that says "`movwf TXREG`" (around line 39). This should highlight the line in red, indicating you have set a breakpoint at that line
b) Find the **Stimulus** tab (in the lower window), click on that, and then select **Register Injection**.
c) Click under Label and enter the word "Serial"
d) Click under Reg/Var and select **RCREG** from the drop-down menu
e) Under Trigger select **Message**
f) Leave PC Value and Width as they are
g) Click under Data Filename and enter **data.txt**
h) Leave Wrap at **Yes**
i) Leave Format at **Pkt**

The next step will vary depending on your computer setup. Find the directory that contains your project (using the full path you noted in Step I.4 above). Create a file named data.txt containing a single line: "Hello World!"
(**you must include the quotation marks**). You can create this file using notepad, vi, or whatever *plain text editor* you like (avoid wordpad, word, etc.)

Now you're ready to build and simulate your project: select "Debug/Debug Project" to assemble the code and start the debugger.

Note: The lower-right window should have a tab labeled "Echo (Build, Load, ...)" and the bottom of that window should include the messages "BUILD SUCCESSFUL" and "Loading Complete." Any messages in red indicate a likely error in your source code.

Your code should now be running, waiting for serial input to arrive. To simulate serial data being sent to the PIC, click on the Stimulus tab (bottom window), and click the small green circle/white triangle near the upper left corner of that window. That will begin sending data from your data.txt file to the PIC processor. At this point, the simulator should have hit your breakpoint and halted. If not, double check all of the above steps.

If the simulator has halted, check the value in the W register (neat the upper right of the simulation window). You should see something like "n ov z dc c : W:0x48 : bank 0"
That W:0x48 shows you the ASCII value of the first letter of your message ("H")! Now hit the run button to continue the simulation; the simulator should again halt, but now W has a new value (0x65, the letter "e"). Continue clicking the run button, and observe the values of W each time the simulation halts. The pattern will eventually repeat.

Record all the values observed for at least one full cycle.

Step III. Simulate Output

The above lets you inject serial data to the PIC processor. We also want to read data *from* the PIC. Do the following:

1) Click on the Output tab, and look for a new tab labeled "**UART 1 Output**" Click on that. Any output sent from the PIC should appear in this window. **If you don't see this tab, save your project, close it and then re-open it**.
2) Now click the Run button and you should see the next word from the file appearing in the output window. Keep clicking Run, and you should see the entire message ("Hello World!") appearing, and repeating as you continue clicking.

Experiment 2. Change input before echoing

Modify your interrupt code to add 1 to each received character before sending form the PIC. Record the new output

Experiment 3. Switch between lowercase and uppercase

You can manipulate ASCII codes to do things like change between uppercase and lowercase. Use the following rules, based on each character's ASCII code:

1) if >= 0x41 and <= 0x5A, add 0x20 to make lowercase
2) if >=0x61 and <= 0x7A, subtract 0x20 to make uppercase

Update your interrupt code to do this conversion. Try it with different inputs (change the contents of data.txt) and record the input and output.

Report Requirements

This lab and associated report must be completed individually. All reports must be computer printed (formulas and diagrams may be hand drawn) and at minimum:

For each experiment include:

- Clear problem statement in your words.
- Answer to any specific experiment questions (if any)
- Pseudo code which may be written in C-like syntax
- Disassembled code available after successful assembling
- Test plan which describes the input values and expect output/memory values for a successful design.
- Simulator output which shows the stimuli, relevant memory locations values showing validation based on test plan.

For the whole report include:

- A Cover sheet with your name, class, lab and completion date.
- A Lessons Learned section which summarizes your learning from this lab.
- A New Experiment section that has description of a new experiment and the experiment's results. Experiment should be related to material covered in class but not similar to one of the experiments in this lab.