ENGR 270 Lab #6 Online – Analog to Digital Converter

Objectives

Apply Analog to Digital Converter (ADC) and Pulse Width Modulation (PWM) concepts to develop a control system (sensing and driving). Learn to use the Graphical PIC Simulator to observe the behavior of your PIC code.

Preparation

Complete the following steps before starting to work on the experiments in this lab:

1) Complete Lab 5 and associated report.
2) Read Chapter 4 sections in the textbook on A/D conversion, and review the sections on timers and PWM generation.
3) Download the Graphical PIC Simulator from
   https://gitlab.com/nickmacias/PICSim/-/raw/master/PICSim.jar
   You should be able to run it by just clicking on the icon.
   Notes:
   1) In Linux, you may need to change the permission to be executable.
   2) In Windows, you need to remove ".zip" from the name of downloaded file. Also make sure that Windows file extension is enabled.

   The first time you run it, you'll have to tell it where the MPLAB debugger is (there are instructions provided when you run it).

   You can access the sources code at the following link if you interested:
   https://gitlab.com/nickmacias/PICSim

Experiment 1. Reading Analog Inputs

The PIC18F1220 has an Analog to Digital Converter (ADC), which can be used to read an analog voltage from any of 7 pins (AN0-AN6). The following code may be useful in working with this subsystem.

```
;--------------------------------------------------------------------------------
; File: main.asm
; Desc: Demonstrate A/D Converter usage
;  * Read from AN0, display MSB on PORTB
; Last Update: June, 2020
; Auth: Class
;--------------------------------------------------------------------------------

    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
    org     0x00

;* setup A/D subsystem
    movlw 0x7f
    movwf ADCON1        ; set all pins to be digital
    setf  TRISA         ; PORTA is all inputs
    clrf  TRISB         ; set PORTB for output
    bcf   ADCON1,PCFG0  ; AN0 is an ANALOG input
    bsf   ADCON0,ADON   ; Turn on A/D subsystem

;* normally, you need to choose the acquisition time and
;* A/D clockrate carefully, to ensure conversion happens
;* correctly. For simulation, this isn't critical though.
    clrf  ADCON2        ; Left-justified results. ACQT=000; ADC=000

main:
;* Start the AD converter, wait for it to finish, and read the result
    bsf   ADCON0,1   ; this starts the conversion
adWait:
    movf  ADCON0,w  ; lsb=0 when conversion is done
    andlw 0x02      ; mask everything but lsb
    bnz   adWait    ; if still set, keep waiting

;* conversion is finished - result is in ADRESH
    movff ADRESH,PORTB
    bra   main      ; repeat forever

    end
```

Enter the above code into MPLAB, assemble it, and try it in the debugger. If you single step your code, you should see bit 1 of ADCON0 be set (at main), and it should remain high for several cycles, and then change to 0. At that point, your code should copy ADRESH to PORTB, and loop back to main.

Unfortunately, it's difficult to simulate analog input in the MPLAB debugger, so ADRESH will always be 0. To test your code more thoroughly, you will use the GUI you downloaded above. Run it by clicking on the "PICSim.jar" icon, then click the Load File button, and find the .hex file for your project. This is normally located in:

(your project directory) / projectname / dist / default / production / filename.hex

Once you open the file, the GUI will immediately start running the code. You'll see the PC (program counter) displayed at the top, and it should be changing rapidly. This means your code is executing.

If you've correctly configured AN0 to be an analog input, you should see a slider labeled AN0: 0.000 V Moving the slider adjusts the voltage being supplied to pin AN0. As you change the voltage level, the LEDs labeled RB should show you the digital version of the analog voltage. This confirms that your code is working correctly.

Experiment 2. Low Voltage Detector

One useful application for an A/D converter is to monitor a power line for a low-voltage condition. Write a program that continually monitors AN0, and checks to see if the voltage is below 1.25 volts. Your code should set RB7 HIGH if the voltage falls below 1.25 volts; otherwise RB7 should be LOW.

You can develop and test your code in MPLAB's debugger by manually loading values into ADRESH. Once you believe your code works, load it into the Graphical PIC Simulator and observe its behavior as you change the AN0 voltage.

Experiment 3. Motor Control

Recall that even though the PIC only outputs two voltage levels (0V and 5V), we can use PWM to control the speed of a motor. Use the following code fragment to set up the PWM subsystem:

```
; setup PWM
    movlw 0x0c
    movwf CCP1CON       ; PWM on P1A (pin 18, RB3)
    clrf TMR2

    movlw 0x05
    movwf T2CON         ; timer enabled, pre-scalar=4
    bcf   PIR1,TMR2IF    ; clear interrupt flag

    movlw .255
    movwf PR2           ; period register (period=255)

waitToStart:
    btfss PIR1,TMR2IF   ; wait for interrupt
    bra   waitToStart
    bcf   TRISB,3       ; make RB3 an output (but we already did this)
```

After executing this code, the PIC will generate a PWM signal on pin RB3. The duty cycle is controlled by loading a value into CCPR1L, from 0 (0% duty cycle) to 254 (close to 100% duty cycle).
Note that there is a bug in MPLAB, where, if CCPR1l = PR2, the duty cycle drops to 50%.

Write a program which continually reads an analog voltage from AN0, converts it to a digital value (0-255), and loads that into CCPR1L. Run this in the Graphical PIC Simulator. Check the "PWM" box to display a simulated motor connected to pin RB3. Observe the effect of changing AN0 on the motor's speed. Record and discuss all observations.

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.