CoE 115 Lab 4: Analog to Digital Conversion

March 2019

Topics/Objectives:

- To understand the general functionality of Analog-to-Digital Converters
- To be able to properly setup the Analog-to-Digital Converter functionality of the PIC microcontroller
- To interface an analog peripheral to the PIC Microcontroller

Pre-lab

- Read the "10-bit High-Speed A/D Converter" section of the PIC24FJ64GB002 Family Reference Manual

A. Overview

The PIC's 10-bit A/D converter allows us to use the PIC microcontroller to read analog voltages as inputs, as opposed to just HIGH and LOW values of digital inputs.

For this exercise, we will be interfacing an analog device as our input and use multiple LEDs to display the converted digital output. We will use AN9 as the analog input pin. For the analog signal we need, set up a voltage divider circuit using a potentiometer and the 3.3V supply. Lastly, we will revisit Lab 1b and apply the ADC functionality accordingly.

B. Setting Up the ADC

Refer to the "10-Bit High-Speed A/D Converter" section of the PIC24FJ64GB002 as your guide for setting up the ADC registers. The PIC ADC Module for this exercise is configured according to the table bellow.

AD1PCFG	**A/D Port Configuration Register** The AD1PCFG register is used to configure which pins are digital inputs and which are analog inputs.	• PCFG<12:0> - Set up AN9 as analog input, all other pins as digital inputs. • PCFG<15:13> - Set to 1.
AD1CON1	**A/D Control Register 1** AD1CON1, like AD1CON2 and AD1CON3, is a collection of multiple control fields to configure certain parts/functionality of the ADC.	• Set up the ADC to discontinue operation if device enters Idle mode • Output the data in integer form. • Use the internal counter as trigger for conversion and automatically begin sampling.
AD1CON2	**A/D Control Register 2** AD1CON2, like AD1CON1 and AD1CON3, is a collection of multiple control fields to configure certain parts/functionality of the ADC.	• Do not scan inputs. • Interrupts should be at the completion of each sample/convert. • Always use MUX A input settings • Buffer is configured as one 16-word buffer.
AD1CON3	**A/D Control Register 3** AD1CON3, like AD1CON1 and AD1CON2, is a collection of multiple control fields to configure certain parts/functionality of the ADC.	• Use system clock • AD1CON3<12:8> - set to 2 * TAD • AD1CON<7:0> - set to 2 * TCY
AD1CHS	**A/D Input Select** This is to set which inputs are the positive or negative inputs to channel 0.	• AD1CHS<4:0> - Channel 0 positive input is AN9 for MUX A. • All other bits 0.
AD1CSSL	**Input Scan Select Register** This is to set which inputs are the positive or negative inputs to channel 0.	• Set to 0

Do not forget to also set up your inputs and outputs using GPIO bits, including AN9.

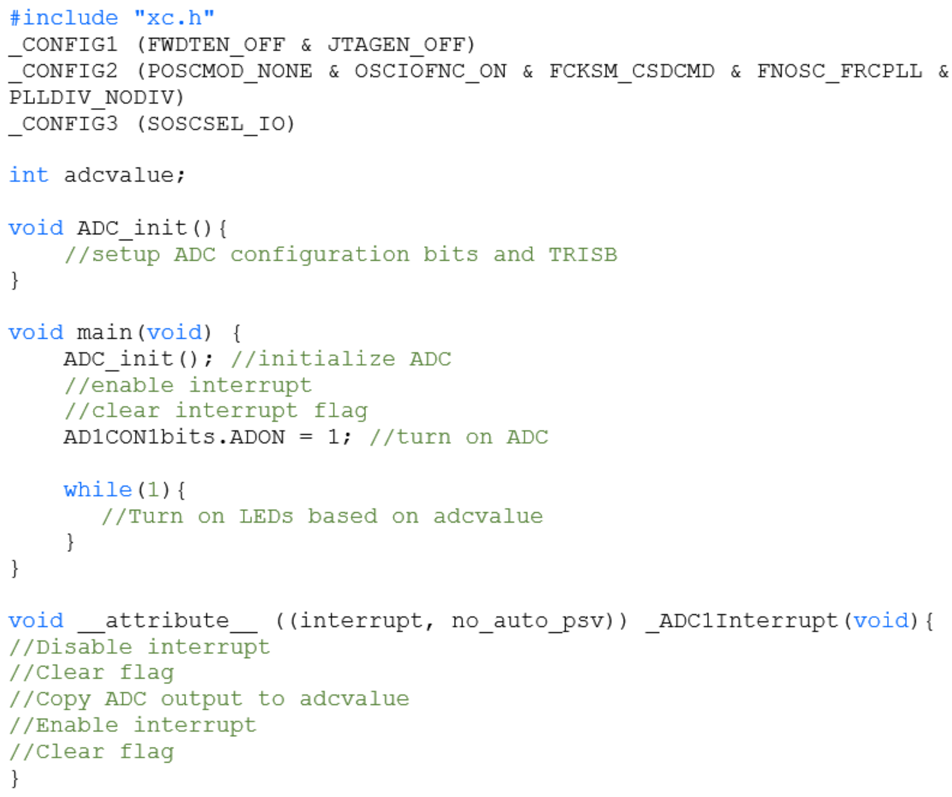
Setting up your ADC this way will set it to continuously convert the value it reads at AN9, and store it in a buffer, ADC1BUF0. We will need to set up a ISR to be able to utilize this value and store it whenever the microcontroller completes a sample/convert. Similar to our Input Change Notification interrupt enable and interrupt flag.

EC0bitsAD1IE	A/D Interrupt Enable
IFS0bits.AD1IF	A/D Interrupt Flag

C. Lab Proper

Part 1:
Line up N LEDs, with N being the last digit of your student number. If the last digit of your student number is less than 5, double that to get N. Therefore, student XXXX-57343 and student XXXX-00746 will both use 6 LEDs. For students with '0' as the last digit, use 6 LEDs.

Write a code that will control the number of LEDs ON (output) depending on the voltage level (use a potentiometer of any value). The middle pin of the potentiometer should be interfaced to AN9 while the two other pins are connected to the output of the 3.3V regulator and ground. Modify the basic code below.



```
#include "xc.h"
_CONFIG1 (FWDTEN_OFF & JTAGEN_OFF)
_CONFIG2 (POSCMOD_NONE & OSCIOFNC_ON & FCKSM_CSDCMD & FNOSC_FRCPLL & PLL96MHZ_OFF &
PLLDIV_NODIV)
_CONFIG3 (SOSCSEL_IO)

int adcvalue;

void ADC_init(){
    //setup ADC configuration bits and TRISB
}

void main(void) {
    ADC_init(); //initialize ADC
    //enable interrupt
    //clear interrupt flag
    AD1CON1bits.ADON = 1; //turn on ADC

    while(1){
       //Turn on LEDs based on adcvalue
    }
}

void __attribute__ ((interrupt, no_auto_psv)) _ADC1Interrupt(void){
//Disable interrupt
//Clear flag
//Copy ADC output to adcvalue
//Enable interrupt
//Clear flag
}
```

Prove to your instructor that the number of LEDs that are actually on is the same as what is expected. [70]

Part 2:
Using the same number of LEDs above, use your code from Lab 1b and control the speed of the change in pattern depending on the analog input. Maximum delay is 1s, minimum delay is 125ms. [30]