EE 2361 - Introduction to Microcontrollers

Laboratory #6

A/D Conversion - Interfacing with the LCD Display

Background

Analog to Digital Conversion (ADC) is an important component of many embedded systems. Sensors convert physical properties of the environment such as temperature, humidity, pressure to analog voltages to be read by microcontrollers and other devices. The rate at which we sample the analog voltage and convert it to a digital value is called the *sampling rate*. The sampling rate is dictated by application requirements and the noise in the data on one hand, and by the constraints of the hardware on the other hand. A few examples of these factors are listed below:

- Application requirement examples:
  - Need to sample the temperature and pressure every second (to decide on opening/closing valves, and maintaining a safe pressure inside a vat). The samples have to have a resolution of 8-bits or higher.
  - Need to sample the audio input at the CD quality of 44.1 KHz (to record on an SD card)
- Noise example:
  - Need to record the weight every minute, but due to the presence of vibrations in the environment that lead to noise in the data, we need to sample the sensor value 100 times, and record the average.
- Hardware constraints:
  - The sampling of the voltage and converting it to a 10-bit digital value takes at least 2μs, so we cannot possibly sample at a rate higher than 1/2μs = 500k samples per second (500 kHz).
  - Calculating the running average of 100 samples with floating point operations takes at least 44ms so even if the hardware can sample analog voltages at a higher rate, we cannot sample and compute at a rate higher than 1/44ms.

Summary of the Lab

This lab is divided into two parts. In the first part, you will develop a digital voltmeter, reading the voltage of an analog source over time, placing the values in a circular buffer, and showing the most recent value on an LCD display. We will try two analog sources: a potentiometer, and a force-sensitive resistor (FSR).

I part two, you will calculate the running average and standard deviation of the samples. You will characterize your code and report the effective sampling rate imposed by calculation speed constraints (in other words, since you cannot calculate the running average quickly, you have to slow down the sampling rate). You will also characterize the behavior of the FSR to determine if it can detect the weight of a penny. In other words, you will build a digital scale.

Lab 6 Reading + Videos

- PIC24F Family Reference Manual, Section 17, 10-bit A/D converter.
- Force-sensitive resistor tutorial from Adafruit.
- EE2361 Lecture Video on Intro to A/D.
- EE2361 Lecture Video on Circular Buffers.

Part 1: Digital Voltmeter

In this part we will build a rather fancy "digital voltmeter." It will sample voltages and report them via a serial interface on the LCD display that you used in your previous labs. Your job is to build an application which will sample the analog input from AN0 K=16 times a second and place the 10-bit digital values in a circular buffer of size B=1024.

Every 100ms, the application will convert the most recent digital value to its corresponding floating-point voltage value. The voltage value is output on the LCD display as "x.xxxx V". Note we are showing voltages on the LCD display far less frequently than we acquire them. As a result, in Part 1 you will only output a subset of sampled voltages. As a side note, even if we wanted to update the display as fast as possible, sending an 8-character string to the LCD display would take longer than acquiring an analog sample, so you will still need to drop many samples even if you want to refresh the display faster than 100ms. We will revisit this point in Part 2 of the lab.

To provide the input to the AN0 pin, connect a potentiometer between power and ground, with the "sweeper" connected to the AN0. You should set up the A/D unit to use Vdd and GND as reference voltages. We suggest using Timer 3 as the trigger for the A/D conversion (i.e., SSRC<2:0> = 010), and using an interrupt service routine (ISR) for the A/D conversion.

There are two tasks that you have to perform, and you are expected to characterize the timing of each task. We suggest using interrupts for data acquisition and polling in main to handle communication to the LCD display.

- Getting K=16 analog samples per second, and placing them in the circular buffer. In the deliverables for this part, you will be asked to analyze the effect of K on the overall performance, assuming data acquisition is the only task you have to handle. K could be as high as 512.

- Sending the most recent voltage value to the LCD display. The result you display must always represent the latest values of the voltage you could get. You are not allowed to grab 16 samples and then wait around, not sampling, until you can deliver them. In other words, what we want, at any time the the value is printed, is that it represents the "latest and most accurate" value we could get.

Because the LCD display communicates over a serial bus, there is going to be a limit on how fast you can update values given you have to send ~16 characters.

The following code snippet shows an easy way of converting floating point numbers to strings. It may not be the fastest way, but perhaps the easiest way of implementing it.

```
#include <stdio.h>    // for sprintf
...
int main(void) {
        int adValue;
        char adStr[20];
        adValue = ADC1BUF0;
        sprintf(adStr, "%6.4f V", (3.3/1024)*adValue);  // “x.xxxx V”
                // 6.4 in the format string “%6.4f” means 6 placeholders for the whole
                // floating point number, 4 of which are for the fractional part.
}
```

You have to make sure your code handles these two tasks correctly and you should analyze and explain the effect of each on the performance of the whole program.

After testing your code with the potentiometer, test it with the FSR. Remember: please read through the Adafruit tutorial on the force-sensitive resistor (FSR) before doing this part. Pay attention to the schematic provided in the tutorial and the value of the pull-down resistor. The link was provided in the “Lab 6 Reading + Videos Section”.

Place the FSR’s sensor area on a flat surface such as a table or a hardcover book. Either plug it directly in your breadboard and tilt the breadboard so that the FSR can rest flat on the table / book, or use a connector and long wires to connect it to the breadboard. Place weights on it (e.g., multiple coins) to read the voltage. Since the FSR has a non-linear behavior, you would be better off starting with a base weight of ~30g and add additional weights (e.g., pennies) on top of that to be in a sensitive region. Without that initial offset weight, you probably will not be able to detect the weight of a penny. When you look at the voltage values on the LCD display after you have placed some weight on it, do you notice anything different compared to the potentiometer?

Note 1: using format strings “%d” and “%X” are useful for printing the raw digital values in decimal or hex. For example, you can use `sprintf(adStr, "%X", adValue);` to print the digital value in hex.

Note 2: Make sure all the characters you send to the LCD display belong to the same voltage. For example, suppose the voltage is 2.345 at the time you start sending the current voltage to the LCD display. Suppose right after the display has received characters '2', '.' and '3', the input voltage changes to 3.198. The LCD display should not show "2.398".

Deliverables for Part 1

1. Your lab TA should verify that your circuit works and shows voltages between 0 and Vdd when the potentiometer is connected. Your lab TA will also observe the behavior of your program when the FSR is connected.
2. Your lab report should include an analysis of your code. The analysis should clearly state how long each of the two tasks outlined in the previous section take to execute.
3. State the maximum sampling rate that a PIC24 chip with Fcy=16MHz can handle, assuming we do not need output anything on the LCD display? In other words, for the maximum value that K can take, what is 1/K?
4. State the maximum display refresh rate assuming we only want to convert 10-bit digital values to the string format "x.xxxx V" and show the 8-character strings on the LCD display? Ignore A/D times here, i.e., assume the data is already in the buffer. In other words, how far can we push the 100ms refresh rate?
5. Assuming the A/D sampling rate described in Item 3, and the fastest LCD display update calculated in Item 4, how many samples do we have to drop and not show on the LCD display?

Part 2: Handling Noise and the Digital Scale

The FSR has a large time-constant, i.e., it takes time for the plastic to deform to a stable shape when you add new weight on top of it, so all your readings must be over many seconds. (HINT: It can help to press on the sensor for a couple seconds when you place a new weight on top.) When you want to report voltage readings, make sure you are consistent, i.e., record numbers after a fixed amount of time, such as ten seconds.

The readings from the FSR are noisy, so you should read multiple values and calculate the average, as opposed to just showing the most recent value as you did in Part 1. You have to modify your code to keep a running average of the last N samples (e.g., N=32) and their sample standard deviation (the square root of the sample variance). It is OK to search the web for C code snippets that calculate the standard deviation, but you are encouraged to optimize it for our application if you can (e.g., does the fact that we are operating on 10-bit integers help speedup the calculations?)

You will output the decimal representation of the *average* voltage on the first line of the LCD display as "x.xxxx V" and the standard deviation on the second line as "xxx.x mV". Note that both formats are 8 characters long. We suggest you use a low-priority interrupt service routine to calculate the running average.

There are three tasks that you have to perform, and you are expected to characterize the timing of each task and analyze its effect on the sampling rate.

- Getting K=16 analog samples per second. You have already analyzed the timing and the maximum value for K in Part 1.

- Calculating the running average and standard deviation using N=32 samples. With K=16, N=32, you are reporting the running average in the most recent 2 seconds. N could be higher (e.g., N=1024).
  As in Part 1, the result you display (here the average and standard deviation) must always represent the latest values that you have calculated. Again you are not allowed to grab 16 samples and then wait around, not sampling, until you can deliver them. In other words, what we want, at any time the the value is printed, is that it represents the "latest and most accurate" value we could get.

- Sending the average and the standard deviation to the LCD display. Because the LCD display communicates over a serial bus, there is going to be a limit on how fast you can update values given you have to send ~16 characters.

Deliverables for Part 2

1. Demo your circuit to your lab TA. Show what happens if you add pennies one by one on top of your 30g initial offset weight. You have to wait for at least 10 seconds so that the TA can monitor how the voltage changes over time. Your LCD display shows voltages and standard deviations using the format specified in Part 2.
2. Generate a table that shows at least ten observed voltages when you have 1 penny, 2 pennies, 3 pennies, 1 quarter, 1 quarter and a penny. In all cases you have the offset weight of ~30g. For each observation wait at least ten seconds to make sure the voltage stabilizes. The table should also list the variance across the ten observations for each weight.
3. Can we use this digital scale to tell how many coins and of what types we have?
4. Suppose we want to sample voltages at the fastest rate possible in PIC24 with Fcy=16MHz (without any considerations for calculating the running average or displaying the values). Also suppose we want to calculate the running average for N=512 points. How many A/D samples have to go without a corresponding running average calculation?

Going Further

A few ideas (all optional) to make the digital scale more interesting:

- Using interrupts and a state machine to output values to the LCD display. The ISR would have to remember what was the last character it sent, and send the next one.

- Convert voltages to grams, and instead of showing voltages on the LCD, show grams.
- Add a button that would zero-offset the current weight. For example, if you have a container with a weight of 20g and the scale is currently showing 20g, you press the button and from this point on, the scale will show 0g with the container. You add a 3g item to the container, and the scale will show 3g. You remove the container, and the scale will show -20g. For those of you who are not used to cooking and baking, this mode of operation is pretty useful when baking a cake. You add X grams of sugar to the container, then you are supposed to add Y grams of flour to the mix, etc.
- Add a button that would report the weight from 10 seconds ago. How would you change the software to enable this feature?