USC Viterbi School of Engineering

Ming Hsieh Department of Electrical Engineering

EE 109L - Introduction to Embedded Systems

Spring 2015 Lab Assignment #5
Analog to Digital Conversion

1 Introduction

In this lab exercise you will use the Arduino's analog-to-digital (ADC) conversion capability to interface a variable resistor to an 7-segment LED display. The goal is to be able to adjust the setting on the variable resistor and have the number on the display change accordingly. For this lab you will write two programs. The first will use the polling method to do the ADC conversions. Then you will modify the program to use interrupts. Writing the programs will involve learning how to configure the ADC module by setting various register bits, how to initiate a conversion and how convert the results to a number that can be displayed.

Reading assignments:
1. **Chapter 7 on Analog-to-Digital Conversion in textbook**
2. **Notes on the ADC from the Assignments page of class web site**

2 The ADC Module

The Atmel ATmega328P microcontroller used on the Arduino Uno has an analog-to-digital conversion (ADC) module capable of converting an analog voltage into a 10-bit number from 0 to 1023. The input to the module can be selected to come from any one of six inputs on the chip. At the maximum 10-bit resolution the ADC can convert signals at a rate of about 15 kSPS (samples per second.) The inputs to the ADC module appear on the Arduino board as connections A0 through A5.

Information on using the ADC module is provided in a separate document "Using the Atmel ATmega328P Analog to Digital Conversion Module" available on the class web site. Refer to that document for information on configuring the various registers in the module and how to do the conversions.

3 The Variable Resistor

The input to the ADC of the Arduino will come from a variable resistor or "potentiometer" (or "pot" for short) hooked up to be a voltage divider. A potentiometer is a three-terminal device (Fig. 1) that has a fixed resistance between two of its wires and a third wire, sometimes called the "slider" that can be adjusted to give anywhere from zero to the full resistance between the slider wire and the other two.

In Figure 2, the resistance between the top and bottom terminals of the potentiometer is given by R. Internally R can be viewed as two resistors $R - R_1$ and R_1 in series so the resistance between the top and bottom terminals is fixed at R. Changing the position of the potentiometer's control changes the resistance of R_1. Since the value of R is fixed, as R_1 increases, $R - R_1$ must decrease by an equal amount, and similarly if R_1 goes down $R - R_1$ must go up. When hooked up as a voltage divider as in Fig. 2 we can see that V is given by

$$V = V_S \frac{R_1}{(R - R_1) + R_1} = V_S \frac{R_1}{R}$$

As the position of the slider is adjusted the value of R_1 can change from R (making $V = V_S$) to zero (making $V = 0$). By adjusting the slider we can get any voltage from 0 to V_S to appear on the slider terminal that will be connected to the ADC input.

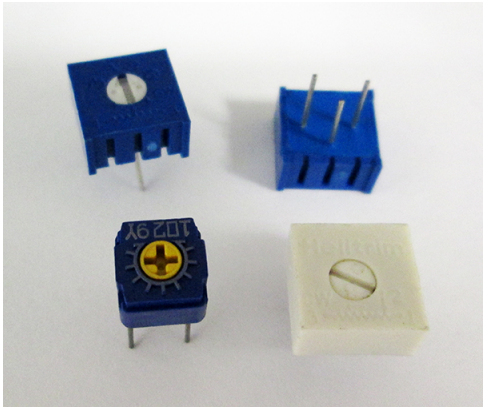

Figure 1: Potentiometers or “pots”

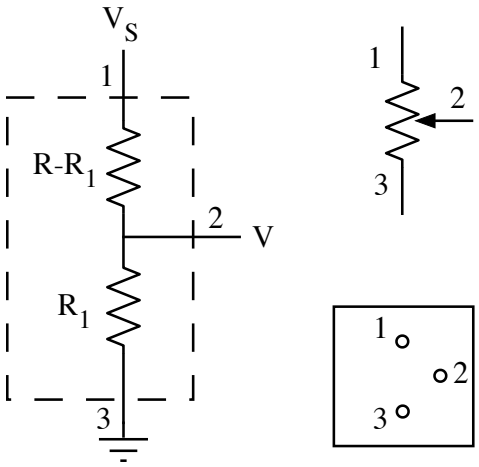

Figure 2: Functional diagram of a potentiometer, schematic symbol, and diagram of pin layout

4 The Circuit

The circuit is shown in Fig. 3. On the potentiometer, the two pins that are on a line parallel to the sides of the device correspond to the top and bottom pins as shown in Fig. 2. The slider pin is the one offset from the other two. Wire the potentiometer up so the slider is connected to the Arduino ADC3 (A3) input and the other two pins are connected to power and ground. To confirm that you have it connected properly, use the multimeter to check that the voltage on the slider pin changes between $+5V$ and ground as you rotate the pot.

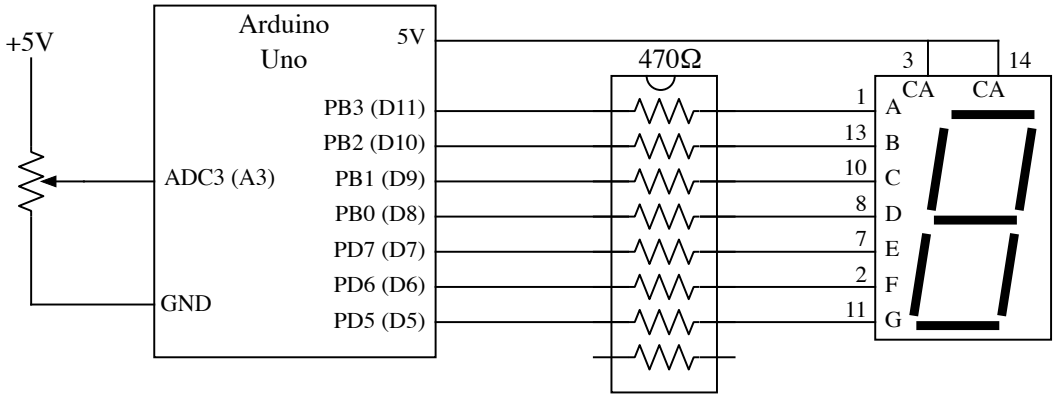

Figure 3: Circuit to control a 7-segment display from a potentiometer

5 Displaying Numbers

Your ADC module will give results that are either 8-bit (0-255) or 10-bit (0-1023) integer numbers. Depending on the results, one of ten digits (0-9) should be displayed. Before writing your program determine what arithmetic operation could you use to convert your ADC value an integer in the range 0-9 **without** using many `if...else if...else if` statements to determine the range. **You are not allowed to use floating point constants (e.g. 13.2) or variables**, but it's okay if your conversion does not yield 10 ranges of exactly the same size.

Here's some space to work it out.

6 The Lab5a Program

The first program to be written (call it `lab5a.c`) will use the polling method to handle the ADC conversions. Like the circuit, your program should borrow a lot of code from the Lab 4 program that made the 7-segment display count up or down. The program should do the operations listed below. Note that you do not have to set or clear a bit in a data direction register or deal with internal pull-up resistors when using a port bit for ADC input. Those registers are only used when using the port bit for digital I/O.

- Configure the PORTB and PORTD lines being used to drive the 7-segment display as outputs.
- Initialize the ADC. Fill in on the register diagrams below the binary values you will use for the ADMUX and ADCSRA registers.

	7	6	5	4	3	2	1	0
ADMUX	REFS1	REFS0	ADLAR		MUX3	MUX2	MUX1	MUX0
				0				

	7	6	5	4	3	2	1	0
ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
			0	0	0			

  - Set/clear the MUX[3:0] bits in ADCMUX to select the input channel you are using
  - Set/clear the REFS[1:0] bits in ADCMUX to select the high voltage reference. Using the AVCC reference is recommended.
  - Set or clear the ADLAR bit in ADCSRA depending on whether you want to use 8 or 10 bit conversion results.
  - Set/clear the ADPS[2:0] bits in ADCSRA to select the prescalar value you have decided on.
  - Set the ADEN bit in ADCSRA to enable the ADC module.

- Start a loop that does the following:
  - Start an ADC conversion by setting the ADSC bit in ADCSRA to a one.
  - Wait for the conversion to complete by starting another loop. Each time through the loop it should read the ADCSRA register and test the state of the ADSC bit. If it's a one, stay in the loop and test again. If it's a zero, break out of the loop.
  - Read the conversion value from the ADC data register.
  - Calculate what number from 0 to 9 you need to display based on the ADC result.
  - Load bits in Ports B and D with the values that will turn on the correct segments for that number.
  - Go back to the start of the loop and do it again.

Unlike in Lab 4 you do **not** need to have a delay in the main loop. The loop (convert analog input, read results, calculate the PORT values, load the PORTs) can run as fast as possible.

When your program is running you should be able to turn the potentiometer control back and forth with the screwdriver and see the number change. At one end of the control's range it should be zero, at the other end it should be 9, and the displayed number should change roughly linearly as the control is rotated. It should appear that each digit is displayed for about an same amount of rotation of the control.

7 The Lab5b Program

Once the Lab5a program is working make a copy of it (`lab5b.c`) and make the following changes.

- Add code to the register initialization steps to set the ADIE bit in the ADCSRA register.
- After the registers have been initialized, enable global interrupts with the function call "`sei();`".
- Start the first ADC conversion by setting the ADSC bit in ADCSRA to a one.
- Write a main loop that does nothing. All of the work to convert the ADC results to 7-segment digits can be done in the interrupt service routine.
- Write the interrupt service routine (ISR). The ISR can do the following tasks.
  - Read the conversion result from the ADC data register.
  - Calculate what number from 0 to 9 you need to display based on the ADC result.
  - Load bits in Ports B and D with the values that will turn on the correct segments for that number.
  - Set the ADSC bit to start the next conversion.

When the program is running it should operate exactly as the Lab5a version did.

8 Results

Once you have the assignment working demonstrate it to one of the instructors. Turn in a copy of your source code (see website for details.)