USC Viterbi School of Engineering

Ming Hsieh Department of Electrical Engineering

EE 109L - Introduction to Embedded Systems

Spring 2015 Lab Assignment #4
7-Segment LED Up/Down Counter

1 Introduction

In this lab exercise you will use the Arduino's digital I/O ports to read buttons and control a 7-segment LED display. The display will count up or down between zero and nine, changing the value every half second. Your breadboard will have two pushbuttons, one to make the display count up and the other to make it count down.

From the instructor get the following:

- Two pushbuttons of different colors, or use two that you already have
- A 7-segment LED display
- A resistor pack

2 The 7-Segment LED Display

A 7-segment display consists of seven LEDs arranged to form the digits 0 to 9 by illuminating selected segments. This could be done with seven separate LEDs resulting in 14 pins on the bottom of the package. However these devices are built with all the diodes connected together in such a way that only eight pins are needed as shown in Figure 1. All diodes have two pins, an anode and a cathode. Anode refers to the positive terminal while cathode to the negative. Current flows from the anode to the cathode. In a 7-segment display, if the anodes of all the diodes are tied together it's called a "common anode" display, or it can be a "common cathode" display if all the cathodes are connected together.

In a common anode display, the anodes are connected to the power supply voltage. Each segment's cathode is connected to the output of the controlling device (gate, microcontroller output line, etc.). When a controlling device goes into the low state (logical 0) the current flows through the diode and into the controlling device causing the LED to light up. In this case the controller is said to be **sinking** current. Conversely, in a common cathode display, the cathodes are connected to ground. Each segment's anode is connected to the output of the controlling device and when that output goes into the high state (logical 1) it **sources** current and the segment lights up.

In a design the choice of which to use may depend on the type of device that will be controlling the display. For example, some types of integrated circuits can sink a substantial amount of current but can only source a small amount. In that case the common anode display would be used. The ATmega328P can source or sink current on its I/O lines about equally well so either type could be used.

The 7-segment displays used in this lab exercise (Fig. 2) are the common anode type and fit in standard IC sockets or breadboards. The segments are identified with the letters 'A' through 'G' in Fig. 3. The connections for the common anodes are indicated by "CA". In addition to the seven segments for the digit, there are two additional LEDs for decimal points, one to the left of the digit and one to the right. We won't be using those in the exercise but they can be used to orient the package properly. To identify pin 1 on the package, hold the display so the two decimal point LEDs are at the bottom. Pin 1 is now in the upper left corner. Note that a few of the pins are missing from the bottom of the package, however the pin numbering scheme still numbers their positions even if there isn't a pin there.

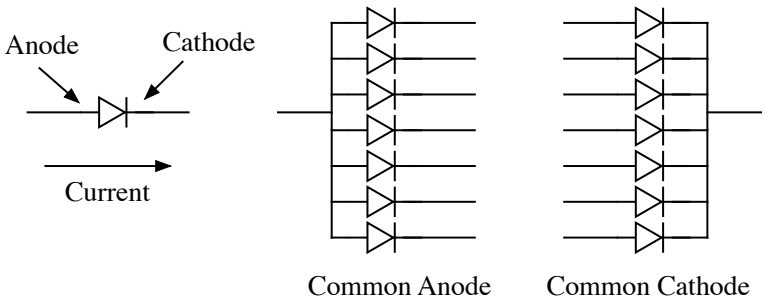


Figure 1: Configuration of diodes in common anode and common cathode displays

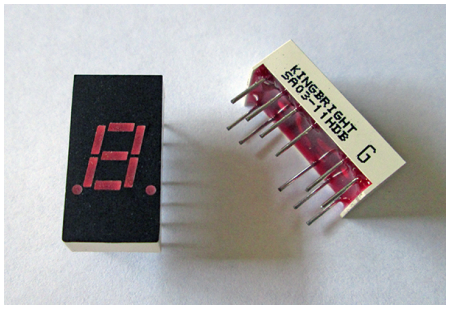
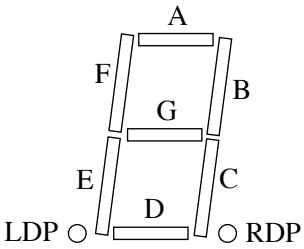


Figure 2: 7-segment LED displays



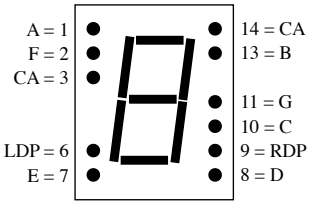


Figure 3: Segment and pin connections

3 Current Limiting Resistors

Each of the seven segments in the display needs its own current limiting resistor as discussed in a previous lab exercise. Rather than use seven separate resistors you will be using a resistor package, also known as a DIP (dual inline package) resistor (Fig. 4). These look like integrated circuits but contain eight identical resistors. The two terminals for each resistor are on the pins on opposite sides of the package as shown in Fig. 5. The value of the internal resistors is marked on the top of the package. Look for a "471" which is the resistor value in the same format as with the color codes: first digit = 4, second digit = 7, and 1 zero to follow to indicate a 470Ω resistor.

The LEDs have a voltage drop of about 1.7V so with a 5V power supply and 470Ω resistors, the current through the LED is given by

$$I = \frac{5.0V - 1.7V}{470\Omega} = 7.0mA$$

which is lower than normally used with LEDs but should be enough in this case.

4 The Circuit

Use your Arduino Uno and your breadboard to build the circuit shown in Fig. 6. The display and the resistor pack should straddle the groove that goes down the center of your breadboard as shown in Fig. 7. All the pins on one side of the package should be in holes on one side of the center of the breadboard, and the pins on the other side should be on the other side of the groove. Try to build the circuit with the display and the resistor pack towards one end of the breadboard so there is space left at the other end to add more components in the future.

The schematic shows both the internal ATmega328P names of the port bits (e.g. PC2) and the corresponding name marked on the Arduino board (A2). The switch inputs are on Arduino bits A2 and A4 which

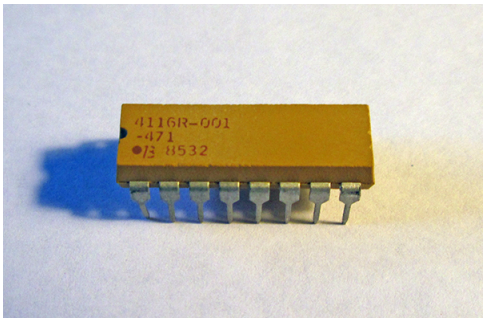


Figure 4: DIP resistor package

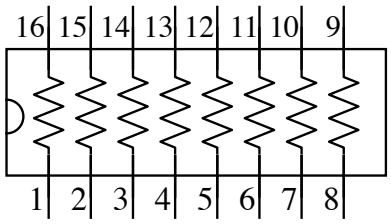


Figure 5: Schematic diagram of a DIP resistor

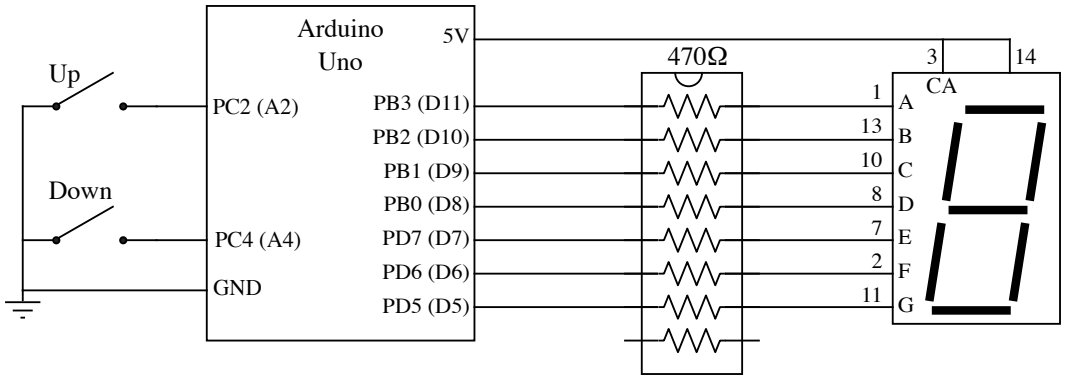


Figure 6: Circuit to control a 7-segment display

are connected to the microcontroller's I/O bits PC2 and PC4. The seven outputs to the display are Arduino bits D5 through D11 which are connected to bits 5 through 7 in Port D and 0 through 3 in Port B of the microcontroller. The signals from each of these output lines will pass through a current limiting resistor and then to one of the segments in the display. Remember to connect the two common anode (CA) pins of the display to the $+5V$ from the Arduino.

5 The Program

Make a new `lab4` folder in your `ee109` folder for your program and copy the `ee109.c` or another lab program in there and rename it to be `lab4.c`. Also as we have done in previous labs, copy a `Makefile` to `lab4` and modify it so it will create the `lab4.o` file.

Your program should use the 7-segment display to show a value that goes up: 0,1, 2, ..., 8, 9, 0, 1, ..., or goes down: 2, 1, 0, 9, 8, ..., depending on the count direction. The count should loop around and start over at 0 after 9 is reached when counting up, or start over at 9 after 0 is reached when counting down. Pressing the "Up" button will switch the counter to counting up, the "Down" button will make it count down. Once a button has been pressed to set the direction, the button does not have to be held down. The counter will continue to count in that direction until the other button is pressed to reverse it.

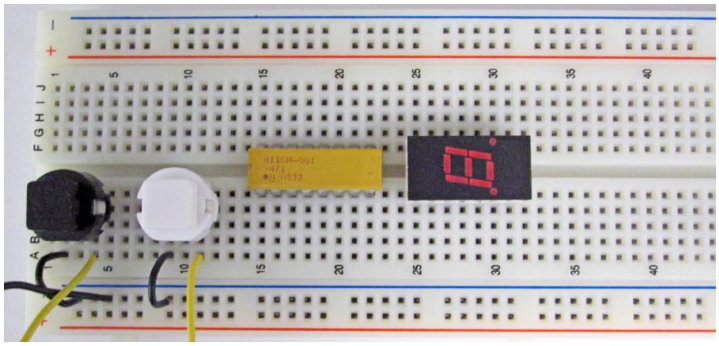


Figure 7: Install DIP packages along center of breadboard. For clarity, this diagram is missing the wires to and from the DIP resistor and 7-segment display. You will need to add at least 15 wires to complete this circuit.

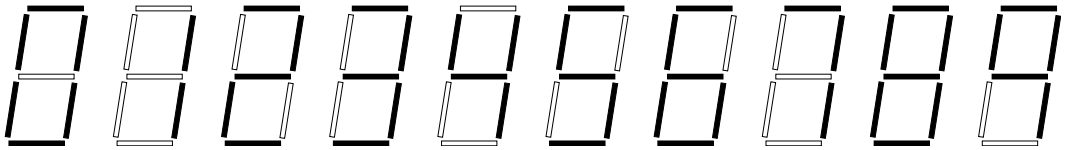


Figure 8: Black segments above are the ones that should be illuminated for each digit

Fig. 8 shows which segments should be lit for each digit being displayed. Using the connections shown in Fig. 6 between the Port bits and the display segments, determine what bits have to be stored in the PORTB bits 0-3 PORTC bits 5-7 for each of the above digits to be displayed. The values for zero have been shown. Remember that with common anode displays, a port bit must be a **zero** to light up the segment. A one in the port bit will cause the segment to be off. **Fill this out before beginning the lab.**

	A B C D E F G									
0:	0 0 0 0 0 0 1	1:	_____	2:	_____	3:	_____	4:	_____
5:	_____	6:	_____	7:	_____	8:	_____	9:	_____

Good programming practice for embedded systems encourages putting code that interfaces to the hardware in as few places as possible. Don't repeat the same lines of code in various parts of the program if it can all be put in one place. This makes it easier to make changes to the hardware interface code if needed. For this lab, the code to change an output bit should only appear once, not repeated in multiple places. For example, the code below is undesirable since it repeats the same command to set bit PD5 and other bits in multiple places.

```
if (digit == 0) {
    PORTD &= ~(1 << PD5);
    PORTD &= ~(1 << PD6);
    // more bit sets and clears
}
else if (digit == 1 {
    PORTD |= (1 << PD5);
    PORTD &= ~(1 << PD5);
    // more bit sets and clears
}
```

All of the changes to the PORT register bits can be placed in a single function that is called with argument(s) that contain the data to set or clear the bits. Each time you need to display a new digit, call this function with the correct arguments. This function should be the only place in the program where the PORT bits for the output lines are modified. Since the output bits are split onto two PORT registers, you can't just copy a whole byte into one PORT registers. Use the "bit fiddling" methods discussed in class (**slides 61-65 in the Unit 4 Notes**) to modify the seven bits (four in PORTB and three in PORTD) **without changing any of the other bits in these registers.**

Your program should do the following operations.

- Configure the Port B and Port D lines being used as outputs.
- Configure the two Port C input lines to have their pull-up resistors enabled.
- Declare "`char`" variables to contain the current count value, and for indicating the direction the counter is counting. Initialize the count direction to be up, and the count value to be zero.
- Start a loop that does the following:
  - Use the current count value variable to display the appropriate digit.
  - Wait one half of a second. As in Lab 3, delays can be implemented with the `_delay_ms` function.
  - Check to see if a button is being pressed and if so, determine which direction to count.
  - Determine what the next value is to be displayed based on the direction of the count, and update your variable storing the current count value accordingly.

6 Testing

In the process of getting your program working it's recommended that you first confirm that you can light up all the display segments. Try just loading the output PORT registers with the value to turn all the segments on and then have the program go into an endless loop so it just sits and does nothing. If this doesn't cause your segments to all light up then you need to resolve this before moving on.

If some of the segment are lighting up but displayed digits look incorrect, write some debugging code to test the segments individually. The debugging code should turn on segment A for half second, then turn off A and turn on B, then turn off B and turn on C, etc., until all the segments have been on by themselves. Put this in a loop and watch the display to see if the segments are going on and off in the correct sequence.

When your program is running you should be able to

- Confirm that each of the ten digits are displayed properly. All the segments are lit that should be lit and no others are on.
- Show that you can change the direction of the counting using the buttons.
- Confirm that in both count directions the sequence wraps around properly when it reaches the maximum or minimum count value.

7 Results

Try to organize your code to use good style and indentation. Examine your solution for repetitive code that can be "factored" and replaced with a function, or other similar enhancements to make the code readable and modular. Points may be deducted for failure to do so. 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.)