Lab 3: Implementing Counters

Lab Group 5
Juliette Mitrovich
Sheila Moroney
Sujani Patel

Lab Summary

The purpose of this lab was to build several digital counter circuits using seven-segment LED displays with an Arduino and integrated circuits. We first learned the basics of a digital counter by programming four LEDs to count in binary. In task 3 we learned how to integrate the two seven-segment displays together. This included learning how to use the Interrupt function in Arduino. We also learned how optical interrupters work and why the resistor connected to the phototransistor is important. In task 4, we learned how to read pinout and wiring diagrams. We also saw firsthand how truth tables work, by wiring different inputs to power and seeing the correct outputs on the display. One thing we would change about this lab is requesting better equipment. We troubleshot our optical interrupter circuit for a long time before we realized the component itself was broken. Then for the 555 timer circuit, we had to borrow someone's personal 10M Ohm resistor because it was not in our kit. Overall, we thought this lab was a good test of our knowledge of digital counters.

Task 1: Implementing a binary counter

For task 1 of lab 3, we were asked to create a binary counter with LEDs, that could go from 0 to 15. The first step was to figure out how many LEDs we would need. Since the highest number we had to represent was 15, we needed four LEDs since 15 represented in base-2 binary has four bits. Next, we wired up all the lights, as shown below in Figure 1, and ensured they could all turn on. We chose to wire them with 220Ω resistors to limit the current through the LEDs and with NPN transistors to allow for the LEDs to turn on and off quicker. We powered the breadboard and the Arduino Uno with the JBtek breadboard power supply, and we had two 0.1 µF capacitors between power and ground to prevent quick changes in the voltage.

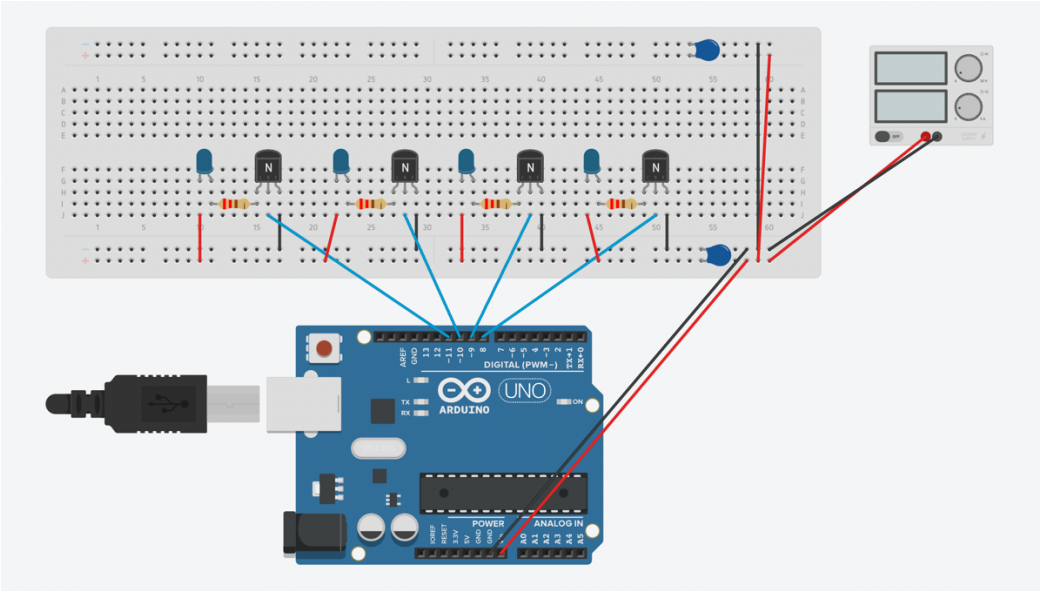

Figure 1. Lab 3 task 2 wiring diagram

Now that the wiring was complete, we began developing the code. After doing some research, we found an Arduino project to reference and get us started [1]. However, we only used the concept since the code to the project was incorrect. We chose to have a FOR loop to count from 0 to 15, and within that loop, have multiple if statements to tell the lights when to turn on and turn off. Each LED had different parameters, but the if statement boiled down to this: if the remainder of the integer divided by x was within a range of numbers, the LED would turn on. If

it was not, the LED would stay off. This idea came from the fact that when counting in base-2, each column has a pattern to it. The 2^0 column transitions from high (1) to low (0) every other number, 2^1 transitions every 2 numbers, 2^2 transitions every 4 numbers, and 2^3 transitions every 8 numbers. Therefore, for each light, we knew the remainder value for which the LEDs should be on and off. Those values became the conditions of the if statements. After all the if statement, but prior to leaving the for loop, we added a delay of 500ms (.5 seconds) because the task instructed us to change states at this rate. For further inspection of the code, please see below in Figure A-1.

Task 2: Creating a counter with an analog input and a seven Segment Display

For task 2, we were asked to count from 0 to 9 on a seven-segment display and be able to vary the speed at which it counts, from 0.5 seconds to 2 seconds. We started by wiring up the display and making sure we could get it to run properly. After some research, we found a great blog at lastminuteengineers.com [2] that helped up wire and code the display. From the specs of our display, we knew it was a common anode and one of the common pins needed to be wired to 5V. Next, we wired each of the 8 pins according to the wiring diagram shown on the website, using 220Ω resistors for each pin to limit the current through the LEDs in the display. We chose to use the same pins as the website, so we did not have to alter the code. After installing the SevSeg Arduino library and copying the code to the Arduino IDE, we successfully counted from 0 to 9. The final step was to use a potentiometer to vary the speed at which it counts. To do so, we decided to take the voltage output of the potentiometer, store it in a variable, and use that variable as the value for the delay command. This was done by converting the 10-bit voltage value to volts, scaling it up so it would be of proper value for the delay command, and using if statements to limit the maximum and minimum values to 2000 and 500, respectively. Once we made sure that the potentiometer could successfully read between 500 and 2000, we then placed this bit of code into the for loop responsible for counting from 0 to 9. At this point, the seven-segment display was able to successfully count from 0 to 9 at varying speeds. Below, you can find the final wiring diagram in Figure 2, and the final code in Figure A-2.

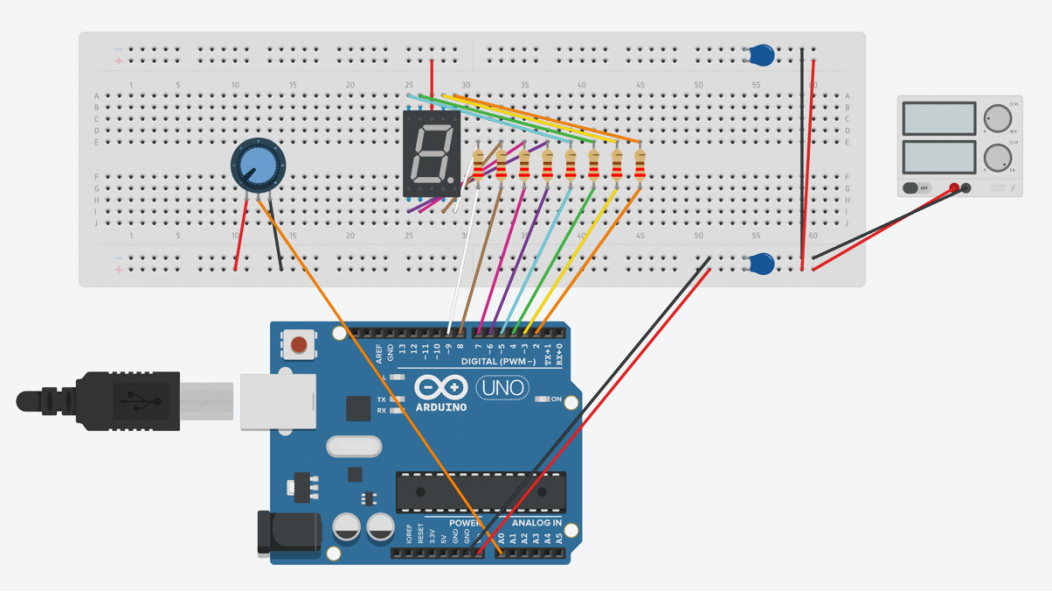

Figure 2**.** Lab 3 task 3 wiring diagram

Task 3

The goal for this task was to use two seven-segment displays to count from 0 to 99 whenever something was passed through the optical interrupter. Also, it needed to have two push buttons, one to enable the count, and one to clear and restart the count. The first step we took was to wire the two displays and get them working. Since we needed a total of 16 digital I/O pins and the Arduino Uno only has 14, we decided to use the Arduino MEGA 2560 one of our teammates had for personal use. By following the same wiring and coding procedures as we did for one seven-segments display in task 2, we got both displays to count from 0 to 9. Next, we moved on to wiring the optical interrupter. The most challenging part of this was to figure out the right resistors. We tried many different resistors, ranging from 220Ω to 100kΩ. Finally, it worked with resistors of value 10kΩ and 560Ω. The last step was to wire the two push buttons. The final wiring diagram can be found below in Figure 3.

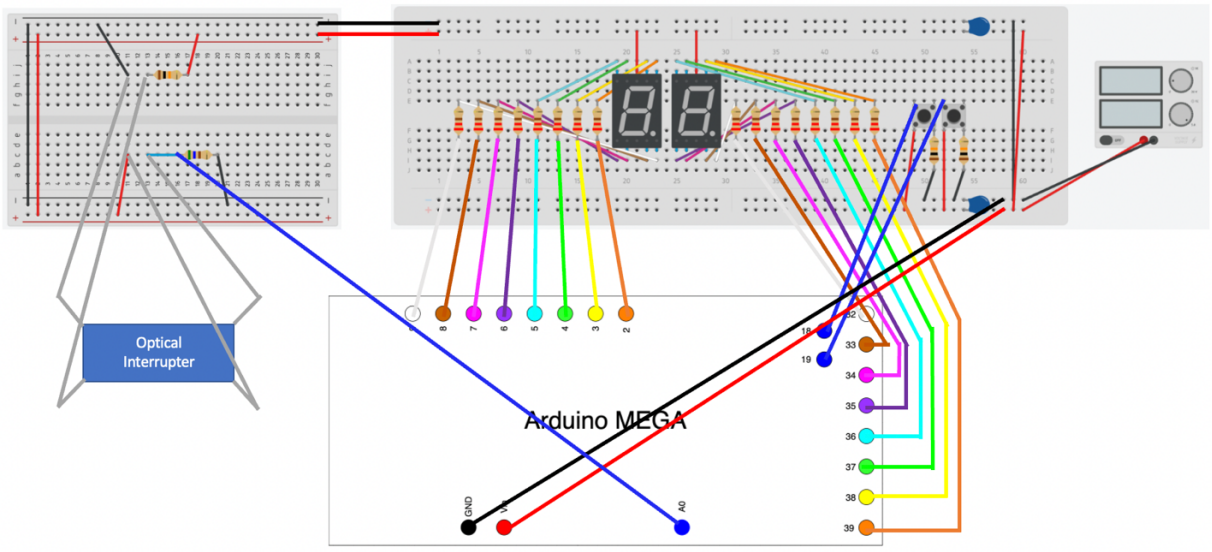

Figure 3. Lab 3 task 3 wiring diagram

To code the displays to count to 99, we started with the for loop used in task 3, with *i* going from 0 to 99. However, instead of simply displaying the number the for loop was on, we had to do some calculations to display the right number on the right display. For the display in the 10s column, we displayed the count divided by 10. Even if the value wasn't perfectly divisible by 10, it would only display the whole number and ignore the remainder. The remainder of said division was used for the display in the 1s column. Next, we coded the displays to only count when the optical interrupter was activated. From testing of the interrupter, we knew that when it was activated, the output was less than 500. Therefore, we just placed the for loop to display the value, inside an if statement with the condition that the interrupter had to read a value of less than 500 to count. We also added a delay of 500ms to make sure it did not count too fast. Finally, we coded the push buttons to enable and clear the displays. We did so by attaching hardware interrupts to those pins. For the enable button, when pushed, the interrupt function told the displays to display 0. For the clear button, when pushed, the interrupt function told the displays to display zero, and it set the count of the for loop back to 1. This way, it would start counting from 1 instead of the number it was on when cleared. In the end, we successfully completed the task. The final code can be found below in Figure A-2.

Task 4

For task 4, the goal was to display the number of pulses in a TTL pulse train on a 7-segment display using integrated circuits. Our wiring diagram is shown in figure 4. The pulse comes from the 7490-decade counter (in count mode), which outputs 4 bits in binary coded decimal (BCD) form. Then the 7447 module decodes from BCD to an output that can be read by the seven-segment display. Following the procedure, we first constructed a 555-timer circuit, using the pinout and circuit diagram. Further on in the procedure, we wired the seven-segment display through the 330 Ohm DIP resistor and to the 7447 module. At this point, we hard wired in the 1, 2, and 7 pins in the 7447 module to power. The result showed a "7" on the display. After consulting the truth table, we saw that this was correct because the inputs 1, 2, and 7 translate to inputs B, C, and A- which output the decimal "7." With the buttons wired up to the right inputs on the 7490, we can control when the 7490 counter resets, or is in count mode. The 555 timer controls the count speed to count up every 1.4 seconds.

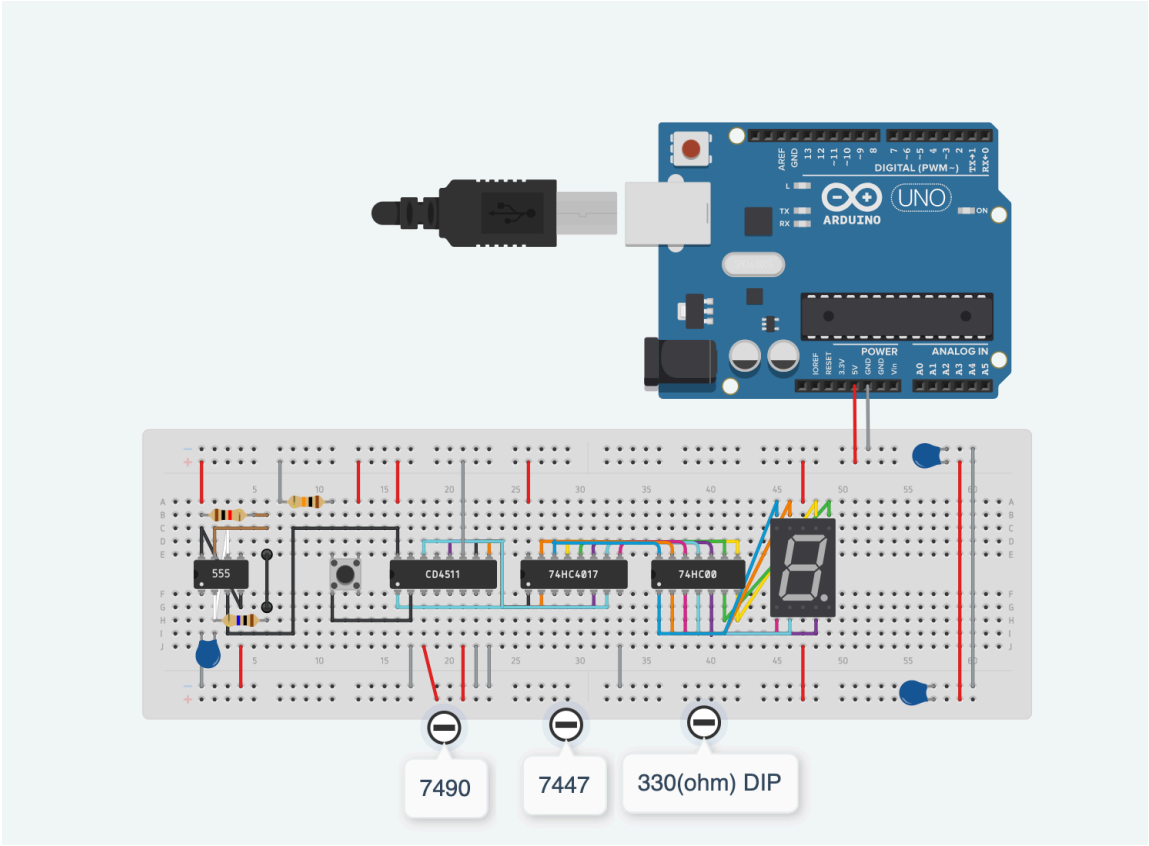

Figure 4. Lab 3 task 4 wiring diagram. Tinker CAD had different model numbers for the decade counter and the decoder

References

1. https://create.arduino.cc/projecthub/Madhur_Bajpai/binary-counter-using-leds-2089d9
2. https://lastminuteengineers.com/seven-segment-arduino-tutorial/\
3. https://www.instructables.com/How-to-use-photo-interrupters-with-your-ARDUINO/

Appendix

Figure A-1. Task 2 code.

```
/*
 * ME 545 - Lab 3 Task 2
 * Group 5: Juliette Mitrovich, Sheila Moroney, and Sujani Patel
 * Display values 0-9 on 7 segment display
 * Use potentiometer to change speed at which is counts (delay)
 */

#include "SevSeg.h"
SevSeg sevseg; // create a name for the seven segment display

//Set to 1 for single digit display
byte numDigits = 1;

//defines common pins while using multi-digit display. Left empty as we have a single digit display
byte digitPins[] = {};

//Defines arduino pin connections in order: A, B, C, D, E, F, G, DP
byte segmentPins[] = {3, 2, 8, 7, 6, 4, 5, 9};
bool resistorsOnSegments = true;

float AREF = 5.0; // reference voltage of Arduino
int pot = A0; // define analog I/O pin for the potentiometer
float pot_bit; // create a variable to store the 10-bit volt output of the potentiometer
float pot_voltage; // create a variable to store the converted voltage value of the potentiometer
float pot_voltage_scaled; // create a variable to store the scaled voltage value of the potentiometer

///// POTENTIOMETER FUNCTION /////
void find_pot_delay_val_range () {
  pot_bit = analogRead(pot); // read the output of the potentiometer
  pot_voltage = AREF * (pot_bit / 1023.0); // convert it to volts
  pot_voltage_scaled = pot_voltage * 1000.0; // scale it up by 1000

// Keep output values within a range of 500ms and 2000ms
  if (2000.0 < pot_voltage_scaled) {
    pot_voltage_scaled = 2000.0;
  }
  if (500 > pot_voltage_scaled) {
    pot_voltage_scaled = 500;
  }
  else {
    pot_voltage_scaled = pot_voltage_scaled;
  }
```

```
  Serial.println(pot_voltage_scaled); // print the scaled value to make sure it's in the right range
}

void setup() {
  Serial.begin(9600); // initalize the serial monitor
  //Initialize sevseg object. Uncomment second line if you use common cathode 7 segment
  sevseg.begin(COMMON_ANODE, numDigits, digitPins, segmentPins, resistorsOnSegments);
  //sevseg.begin(COMMON_CATHODE, numDigits, digitPins, segmentPins,
resistorsOnSegments);

  sevseg.setBrightness(90);
}

void loop() {
  // Display numbers 0 to 9 at varying speeds
  for (int i = 0; i < 10; i++) {
    find_pot_delay_val_range();
    sevseg.setNumber(i);
    sevseg.refreshDisplay();
    delay(pot_voltage_scaled);
  }
}
```

Figure A-2. Task 3 code.

```
/*
  ME 545 - Lab 4 Task 3 Group 5
  Juliette Mitrovich, Sheila Moroney, Sujani Patel
  Goal: Count, on the 7 segment display, every time something passes between the optical
interrupter.
  You should be able to count from 0 to 99. You must also attach interrupts to two push buttons,
one interrupt to
  clear the display, and one interrupt to enable the display

*/

///// SEVEN SEGMENT DISPLAY VARIABLES /////
#include "SevSeg.h"
// define the two 7 segment displays as different names to differentiate and not overwrite data
SevSeg sevseg1;
SevSeg sevseg2;

//Set to 1 for single digit display, 1st display
byte numDigits1 = 1;

//defines common pins while using multi-digit display. Left empty as we have a single digit
display
byte digitPins1[] = {};

//Defines arduino pin connections in order: A, B, C, D, E, F, G, DP
byte segmentPins1[] = {3, 2, 8, 7, 6, 4, 5, 9};
bool resistorsOnSegments1 = true;

//Set to 1 for single digit display, 2nd display
byte numDigits2 = 1;

//defines common pins while using multi-digit display. Left empty as we have a single digit
display
byte digitPins2[] = {};

//Defines arduino pin connections in order: A, B, C, D, E, F, G, DP
byte segmentPins2[] = {33, 32, 38, 37, 36, 34, 35, 39};
bool resistorsOnSegments2 = true;

int i = 1; // Counter to keep track of what number should be displayed

///// PUSH BUTTON VARIABLES /////
```

```
// must be attached to a pin that allows hardware interrupts (MEGA: 2,3,18,19)
int pushButtonEnable = 18;
int pushButtonClear = 19;

// variables to store output of the push buttons
int stateButtonEnable;
int stateButtonClear;

///// OPTICAL INTERRUPT VARIABLES /////
int photoInterrupt = A4;
int photoInterruptVal;

void setup() {
  Serial.begin(9600); // Initialize serial monitor if you want to print anything

  //Initialize sevseg object. Uncomment second line if you use common cathode 7 segment
  sevseg2.begin(COMMON_ANODE, numDigits1, digitPins1, segmentPins1,
resistorsOnSegments1);
  sevseg1.begin(COMMON_ANODE, numDigits2, digitPins2, segmentPins2,
resistorsOnSegments2);
  //sevseg.begin(COMMON_CATHODE, numDigits, digitPins, segmentPins,
resistorsOnSegments);
  sevseg1.setBrightness(90);
  sevseg2.setBrightness(90);

  // Initialize push buttons
  pinMode(pushButtonEnable, INPUT);
  pinMode(pushButtonClear, INPUT);

  // Attach a hardware interrupt to the push buttons, ISR will run whenever there is a change in
pin value
  attachInterrupt(digitalPinToInterrupt(pushButtonEnable), isrEnable, CHANGE);
  attachInterrupt(digitalPinToInterrupt(pushButtonClear), isrClear, CHANGE);

  // Initialize photo interrupt pin
  pinMode(photoInterrupt, INPUT);

}

void loop() {
  // Read data from all necessary sensors
  stateButtonEnable = digitalRead(pushButtonEnable);
  stateButtonClear = digitalRead(pushButtonClear);
```

```
  photoInterruptVal = analogRead(photoInterrupt);

  // If something passes between the optical interrupt and i <= 99, display and increase the value
by 1
  if (500 > photoInterruptVal) {
    if (99 >= i) {
      int disp1 = i % 10;
      int disp2 = i / 10;
      sevseg1.setNumber(disp1);
      sevseg1.refreshDisplay();
      sevseg2.setNumber(disp2);
      sevseg2.refreshDisplay();
      i++;
      delay(500); // add a delay so the display doesn't count too fast
    }
  }
}

///// HARDWARE INTERRUPT FUNCITONS /////
// If the enable button is pushed, turn the display on and set it to 0
void isrEnable () {
  if (stateButtonEnable == 1) {
    sevseg1.setNumber(0);
    sevseg1.refreshDisplay();
    sevseg2.setNumber(0);
    sevseg2.refreshDisplay();
  }
}

// If the clear button is pushed, change the display to 0 and set i back to 1
void isrClear() {
  if (stateButtonClear == 1) {
    sevseg1.setNumber(0);
    sevseg1.refreshDisplay();
    sevseg2.setNumber(0);
    sevseg2.refreshDisplay();
    i = 1; // set i back to 1 so the count will restart
  }
}
```