Lab 3: Interrupt Handling and ADC

Micah Galos - SID: 862082339
Joseph Ayala - SID: 862080244

Due: November 2nd, 2020 - 11:59PM

Abstract [JDA]

Within this lab the aim was to develop and observe the Analog Digital Control and implementation of interrupts in the K64F Freedom board. This will allow us to develop further insight and experience in the use of the microcontroller for this function of polling with a reference voltage. These tasks of demonstrating the ADC and interrupt will be demonstrated through several components/circuit configurations. What will be required for this demonstration is a voltage divider connected to a potentiometer which will act as an input to the microcontroller. Where on the output side of the microcontroller is going to be dedicated to running and manipulating two seven segment displays that will be connected to D Ports and the C Ports of the Freedom board. While the Voltage divider that will be giving us the values is connected to Pin ADC0_DP0. The settings for the analog digital converter will be determined through the 4-bit DIP Switch that and signal generator coming in via an arduino PWM pin. This system was successfully able to count up from zero to ninety nine and then reset successfully as well counting down from ninety nine to then resetting. As well the implementation of the ADC has been confirmed to work(pre-demo).

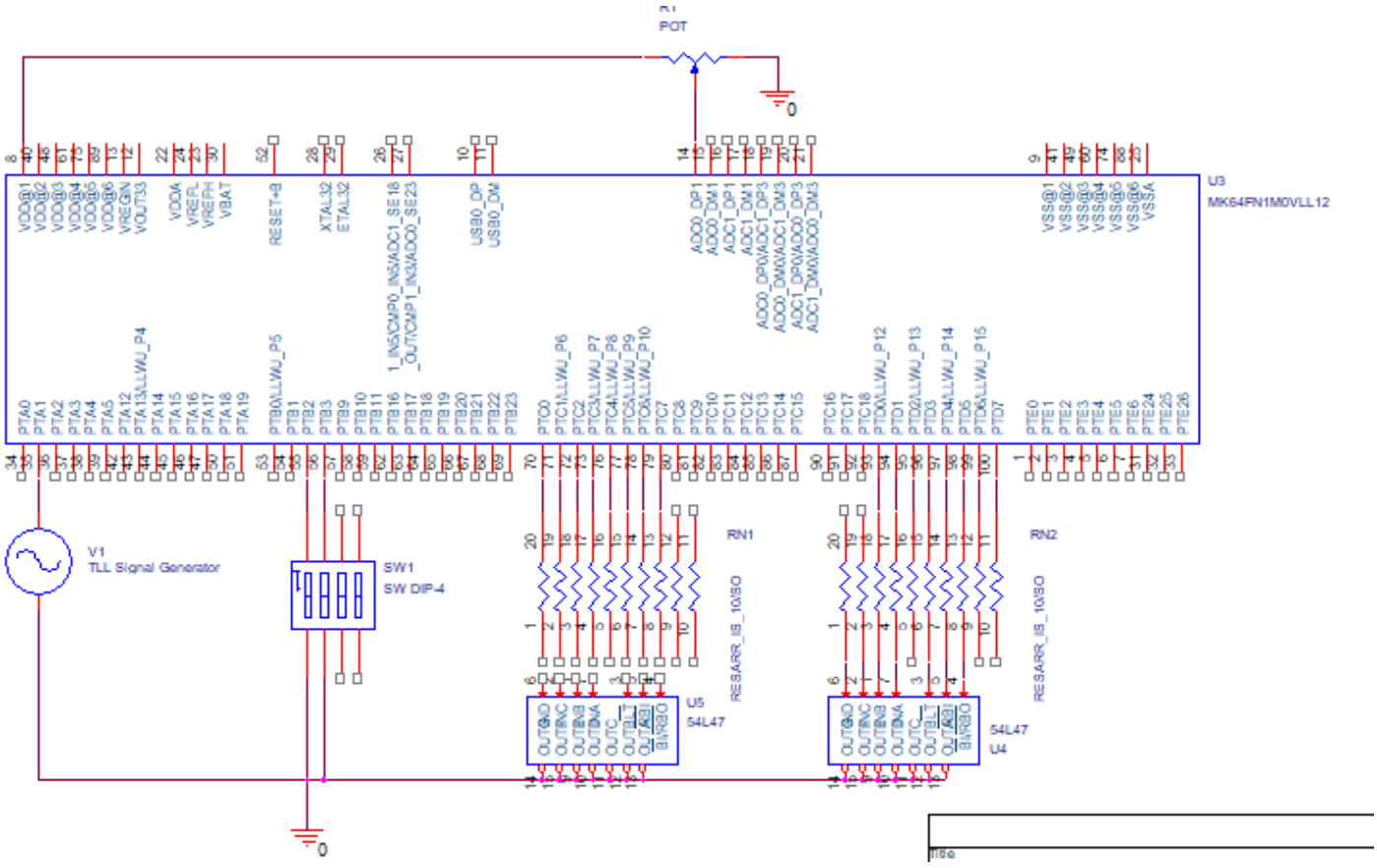
Experiment System Specification [JDA]

In the course of this lab we will be implementing tools from the kinetis library associated with the ADC and interrupts to get the appropriate values from the inputs for the input from the voltage divider and the PWM signal coming from the arduino digital output pin. In terms of designing the software for the course of this lab it took quite a bit of time to get a working solution going after going back and forth with the lab manual, kinetis reference guide. After wiring the hardware accordingly with the resistor arrays on the output of the boards seen in image below to the seven segment led displays, and ground accordingly.

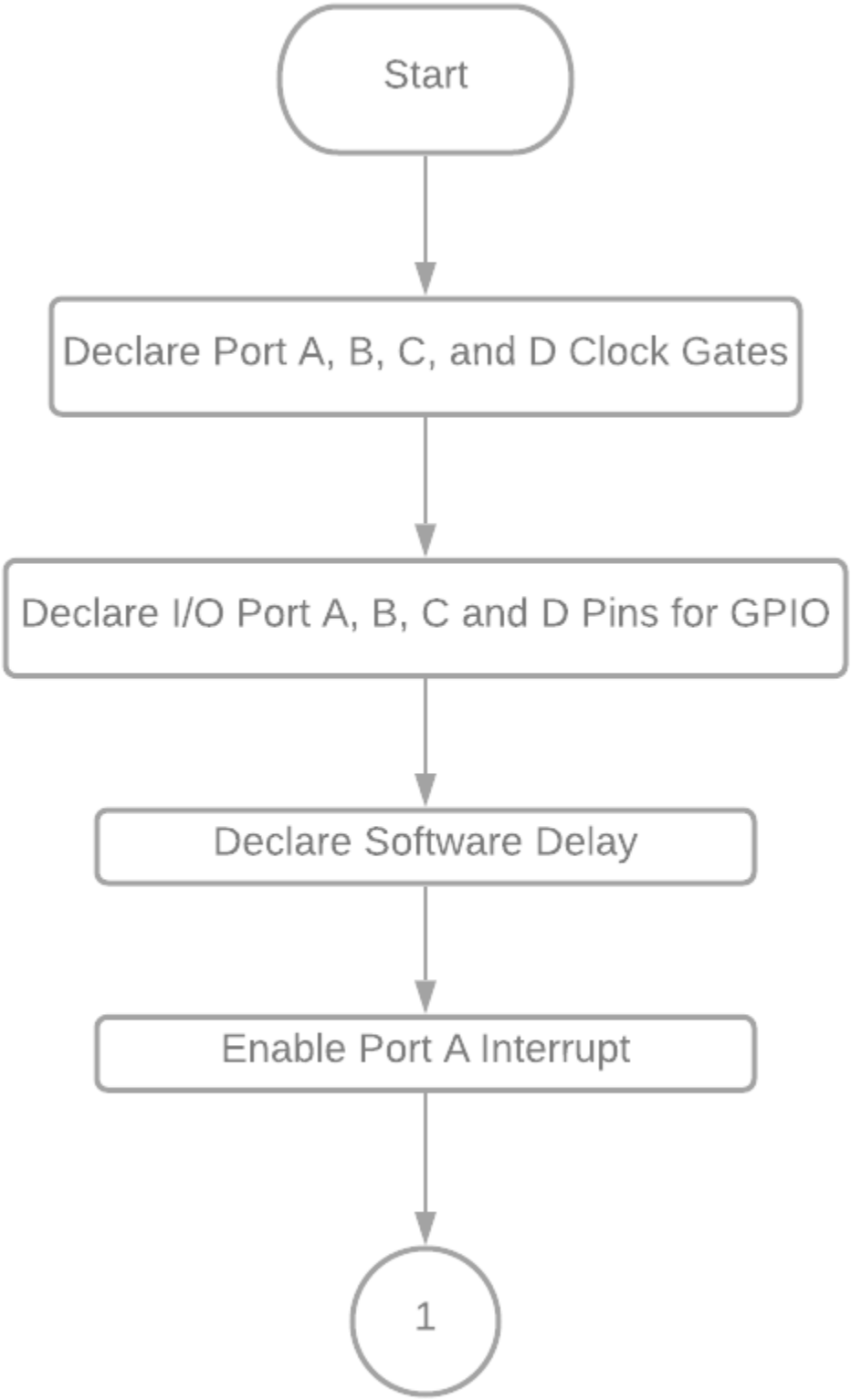
In terms of inputs that are setup for the MK64F are the voltage divider coming from the potentiometer, PWM signal from the Arduino Uno and that 4-Bit Dip switch. These inputs will be determining the effects onto the output including the type of modes that will determine the modes on the MK64. The signal generator used for the purposes of this lab is the arduino uno connected to PTA1 and is indicated as a signal generator in the schematic. Lastly we have the 4-Bit Dip switch used to determine the modes that the MK64 will be in which has been implemented in prior labs.

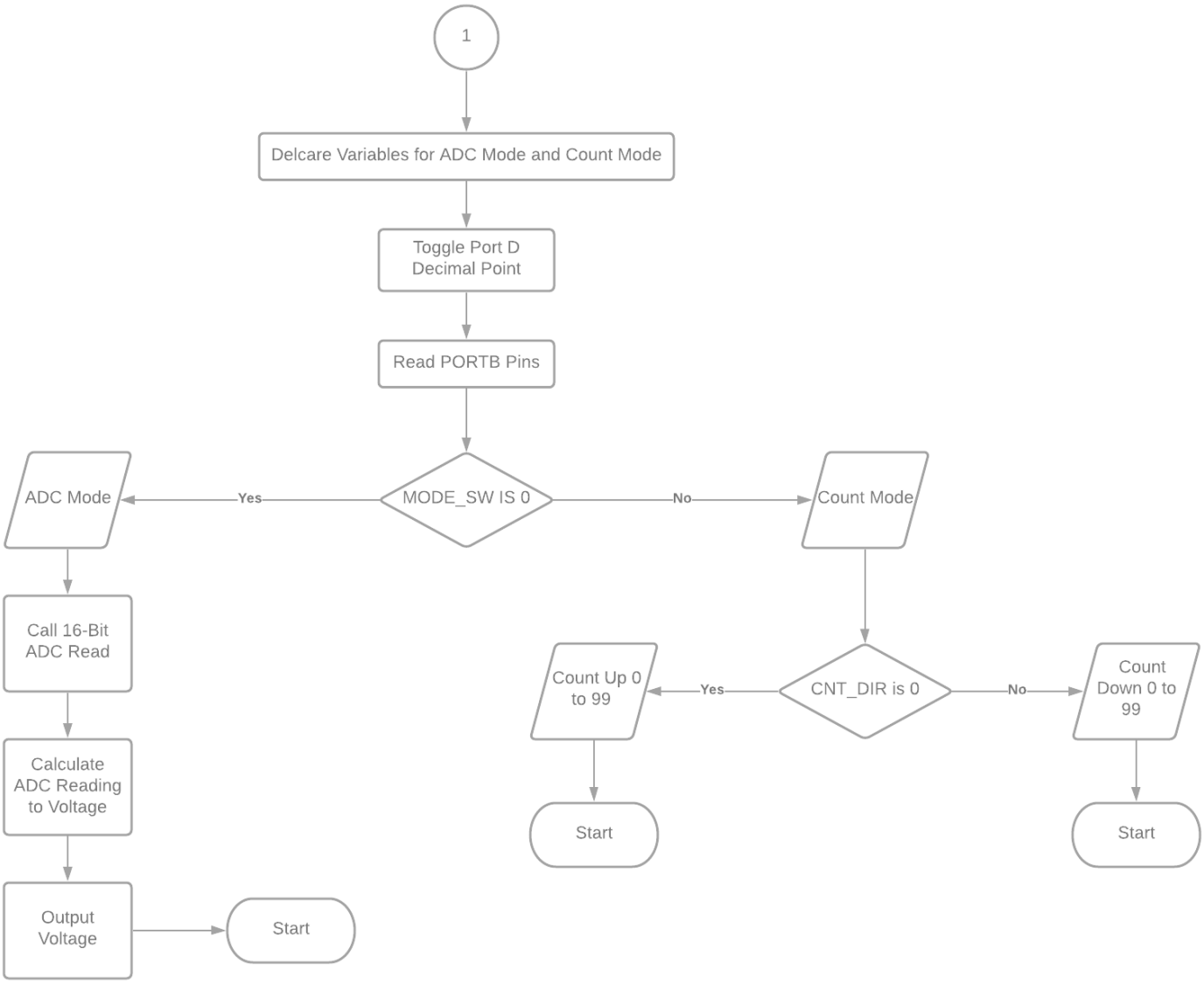
All these components were able to be successfully wired on the same breadboard and as of the writing and submission of this report are still waiting to be demoed.

Hardware Schematic [JDA]

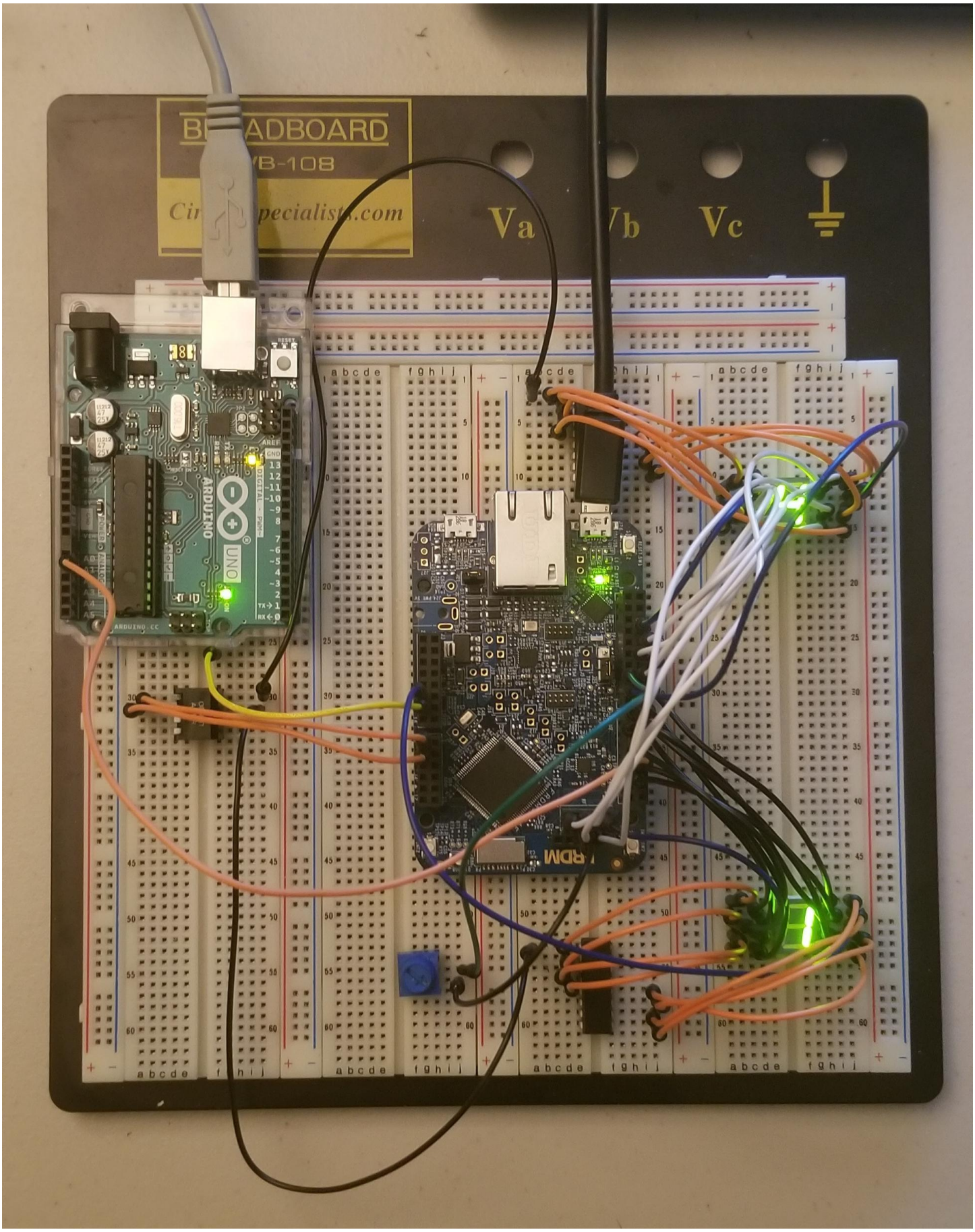


Flowchart [MG]




1
Delcare Variables for ADC Mode and Count Mode
Toggle Port D Decimal Point
Read PORTB Pins
MODE_SW IS 0
Yes
ADC Mode
No
Count Mode
Call 16-Bit ADC Read
Calculate ADC Reading to Voltage
Output Voltage
Start
CNT_DIR is 0
Yes
Count Up 0 to 99
No
Count Down 0 to 99
Start
Start

Board & Circuit [MG]



Software Design [MG]

```
#include "fsl_device_registers.h"

// PORTD 7-SEG
unsigned char seg1[10] = {0x7E, 0x60, 0x3D, 0x79, 0x63, 0x5B, 0x5F, 0x70,
0x7F, 0x7B};

// PORTC 7-SEG
unsigned char seg2[10] = {0xFE, 0xA0, 0x3D, 0xF9, 0xA3, 0xDB, 0xDF, 0xF0,
0xFF, 0xFB};

void software_delay(unsigned long delay)
{
     while (delay > 0) delay--;
}

unsigned short ADC_read16b(void)
{
     ADC0_SC1A = 0x00; //Write to SC1A to start conversion from ADC_0
     while(ADC0_SC2 & ADC_SC2_ADACT_MASK); // Conversion in progress
     while(!(ADC0_SC1A & ADC_SC1_COCO_MASK)); // Until conversion complete
     return ADC0_RA; // ADC conversion result for ADC0
}

void PORTA_IRQHandler(void)
{
     /* Variables */
     unsigned int i = 0, j = 0;
     unsigned short data = 0x00;
     unsigned short AVC = 0x00;
     unsigned long D_out = 0x00, C_out = 0x00;
     unsigned long MODE_SW = 0x00, CNT_DIR = 0x00;
     unsigned long Delay = 0x100000; // Slow

     NVIC_ClearPendingIRQ(PORTA_IRQn);

     //Toggle ones place decimal point;
     GPIOD_PTOR |= 0x80;

     /* Read Port B*/
     MODE_SW = ~(GPIOB_PDIR) & 0x04; // Port B Pin 2
     CNT_DIR = ~(GPIOB_PDIR) & 0x08; // Port B Pin 3

     /* ADC Mode */
     //Read from ADC and convert to decimal value; (e.g., ADC reads 0xFF,
voltage is 1.6)
     if(MODE_SW == 0x00){
```

```
        // V(X) = ( ( X * 3.3 ) / 2^16 - 1 )
        data = ADC_read16b();
        software_delay(Delay);
        AVC = ( (data * 33) / 65535 ); // Calculate as Voltage using ADC
Reading
        D_out = ( (AVC / 10) );
        C_out = ( (AVC) % 10 );

        if(D_out == 0) GPIOD_PDOR = seg1[0];
        else if(D_out == 1) GPIOD_PDOR = seg1[1];
        else if(D_out == 2) GPIOD_PDOR = seg1[2];
        else if(D_out == 3) GPIOD_PDOR = seg1[3];
        else if(D_out == 4) GPIOD_PDOR = seg1[4];
        else if(D_out == 5) GPIOD_PDOR = seg1[5];
        else if(D_out == 6) GPIOD_PDOR = seg1[6];
        else if(D_out == 7) GPIOD_PDOR = seg1[7];
        else if(D_out == 8) GPIOD_PDOR = seg1[8];
        else if(D_out == 9) GPIOD_PDOR = seg1[9];
        software_delay(Delay);

        if(C_out == 0) GPIOC_PDOR = seg2[0];
        else if(C_out == 1) GPIOC_PDOR = seg2[1];
        else if(C_out == 2) GPIOC_PDOR = seg2[2];
        else if(C_out == 3) GPIOC_PDOR = seg2[3];
        else if(C_out == 4) GPIOC_PDOR = seg2[4];
        else if(C_out == 5) GPIOC_PDOR = seg2[5];
        else if(C_out == 6) GPIOC_PDOR = seg2[6];
        else if(C_out == 7) GPIOC_PDOR = seg2[7];
        else if(C_out == 8) GPIOC_PDOR = seg2[8];
        else if(C_out == 9) GPIOC_PDOR = seg2[9];
        software_delay(Delay);
    }

    /* Count Mode */
    else{
        /* Count Direction */
        //count up to 99 and roll over to 0
        if(CNT_DIR == 0x00){
            //for(;;){
                for(i = 0; i < 10; i++){
                    GPIOD_PDOR = seg1[i];
                    if(j > 9) GPIOC_PDOR = seg2[0];
                    for(j = 0; j < 10; j++){
                        GPIOC_PDOR = seg2[j];
                        software_delay(Delay);
                        if(i > 9) GPIOD_PDOR = seg1[0];
```

```
                    }
                }
                GPIOD_PDOR = seg1[0];
                GPIOC_PDOR = seg2[0];
                i = 0; j = 0;
                }
            //}
        //count down to 0 and roll over to 99
        else{
            //for(;;){
                for(i = 9; i > 0; i--){
                    GPIOD_PDOR = seg1[i];
                    if(GPIOD_PDOR == 0x60){
                        GPIOD_PDOR = seg1[1];
                        software_delay(Delay);
                        GPIOD_PDOR = seg1[0];
                    }
                    for(j = 9; j > 0; j--){
                        GPIOC_PDOR = seg2[j];
                        if(GPIOC_PDOR == 0xA0){
                            GPIOC_PDOR = seg2[1];
                            software_delay(Delay);
                            GPIOC_PDOR = seg2[0];
                        }
                        software_delay(Delay);
                    }
                }
                GPIOD_PDOR = seg1[9];
                GPIOC_PDOR = seg2[9];
                i = 9; j = 9;
            //}
        }
    }

    /* Clear ISFR for PORTA, Pin 1*/
    PORTA_ISFR = (1 << 1);
}

int main(void)
{
    /* Configure Clock Gating for Ports B,C, and D; */
    SIM_SCGC5 |= SIM_SCGC5_PORTA_MASK;
    SIM_SCGC5 |= SIM_SCGC5_PORTB_MASK;
    SIM_SCGC5 |= SIM_SCGC5_PORTC_MASK;
    SIM_SCGC5 |= SIM_SCGC5_PORTD_MASK;
```

```
    /* Configure Clock Gating for ADC0 */
    SIM_SCGC6 |= SIM_SCGC6_ADC0_MASK;

    /* NOTE: These are for opening pins of ports */
    /* Configure Port D Pins 0-7 for GPIO; */
    PORTD_GPCLR = 0x00FF0100;

    /* Configure Port C Pins 0-5 and 7-8 for GPIO; */
    PORTC_GPCLR = 0x01BF0100;

    /* Configure Port A Pin 1 for GPIO & Interrupt on Falling-Edge */
    PORTA_PCR1 = 0xA0100;

    /* Configure Port B Pin 2 & 3 for GPIO; */
    PORTB_GPCLR = 0x000C0100;

    /* Clear PORTA ISFR */
    PORTA_ISFR = (1 << 1);

    /* ADC Initialization */
    ADC0_CFG1 = 0x0C; // 16bits ADC; Bus Clock
    ADC0_SC1A = 0x1F; // Disable the module, ADCH = 11111

    /* NOTE: GPIOx_PDDR for I/O Pin Configuration */
    /* Configure Port B Pin 2, 3 for Input; */
    GPIOB_PDDR |= (0 << 2);
    GPIOB_PDDR |= (0 << 3);

    /* Configure Port D Pins 0-7 for Output; */
    GPIOD_PDDR |= 0x000000FF;

    /* Configure Port C Pins 0-5 and 7-8 for Output; */
    GPIOC_PDDR |= 0x000001BF;

    /* Initialize Port C and D to 0; */
    GPIOC_PDOR |= 0x00;
    GPIOD_PDOR |= 0x00;

    /* Enable Port A Interrupt */
    NVIC_EnableIRQ(PORTA_IRQn);

   while(1){
    PORTA_IRQHandler();
   }
   return 0;
}
```

Arduino Uno Function Generator

```
// the setup function runs once when you press reset or power the board
void setup() {
  pinMode(11, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  DDRB = 0xFF;
  PORTB = 0x00;
  pinMode(11, OUTPUT);
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}


    digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage
level)
    delay(1000);                       // delay
  }
}
```

Technical Problems

The first problem in designing the lab was figuring out how to utilize Port D and Port C as count up and count down outputs. At first, I designed by brute force incrementing and decrementing like the previous lab, but the work was too tedious. The issue was solved by running through two arrays, one for Port D and one for Port C separately by a double for-loop. For count up, the inner loop goes through [0:9] on Port C and as the inner loop finishes, increments the outer loop on Port D until it reaches 99, then resets both LEDs to 0. A similar approach was made for decrementing. The second problem was figuring out how to avoid the floating point decimal during the voltage read calculation. The difference was using the input voltage of 3.3, so instead I was suggested to change the value to 33 to avoid said floating points.

Questions

2. For this lab, we use a falling-edge generated interrupt. If you use level-sensitive interrupts, what extra steps (in hardware and/or software) would you need? (don't write the entire code or schematic; answer in a brief manner)

In the realm of software, we could implement a lock such that no two inputs are going into the system at the same time. In addition, there would be some sort of priority flag to the input when a program is running two separated jobs or more at the same time.

3. What are the factors contributing to the interrupt latency? Answer briefly.

One idea I can think of is the MCU architecture we are using to build these projects. There can be variance in latency depending on the board. More importantly, the connection between hardware and software and translating that data to produce such interrupt latency.

4. What is the resolution (minimum distinguishable input voltage) of the ADC implemented in this lab?

From 0 - 3.3 V
16-bit ADC Reading = 2^16 - 1 = 65535
Resolution = 3.3 V / 65535 = 0.5035 µV

Conclusion

In hindsight, the lab could have been finished much sooner than last time. Joe and I threw around ideas to see what worked during the planning process--did not go as planned as expected. At the very least, the debugger helped us out a lot and observed each line of code with the breakpoints. The challenge, in regards to this lab, would be figuring out Port C once again to skip the sixth bit. I wrote down on paper what the 7-bits should output and replace with a one on the seventh bit since we do not care what the sixth bit could be. Nonetheless, I obtained the knowledge to learn from these mistakes and hopefully work faster this time around.