EEE174 –CPE185 INTRODUCTION TO MICROPROCESSORS

LAB 0 - ARDUINO

Lab Session: Wednesday 6:30PM - 9:10PM

Section 32385

Lab Instructor: Sean Kennedy

Student Name: Andrew Robertson

TABLE OF CONTENTS

PART 1

OVERVIEW

Part 1 of this Arduino lab is to use provided code to get more familiar with the board and coding platform. From here, we then expand the provided code a bit to add functionality to understand serial output and to make use of the analog discovery kit. (For me, this will be my first time using the analog discovery kit)

LAB DISCUSSION

W ORK P ERFORMED / S OLUTION :

1)

The first step instructed me to install the IDE on my computer. Since I have dabbled a very small amount with the Arduino UNO R3 before, I already had this installed. I instead opted to make sure the software was completely up to date.

Next, since the Arduino software is set up to be use with Arduino hardware I was instructed to make some changes so the chipKIT hardware could be recognized and used instead. The process to do this was straight forward and only required adding an additional library repository in the IDE's preferences by copying a provided URL into the "Additional Boards Manager URLs" box. Once this was done I could select the chipKIT as the board to be used under Tools >> Board:.
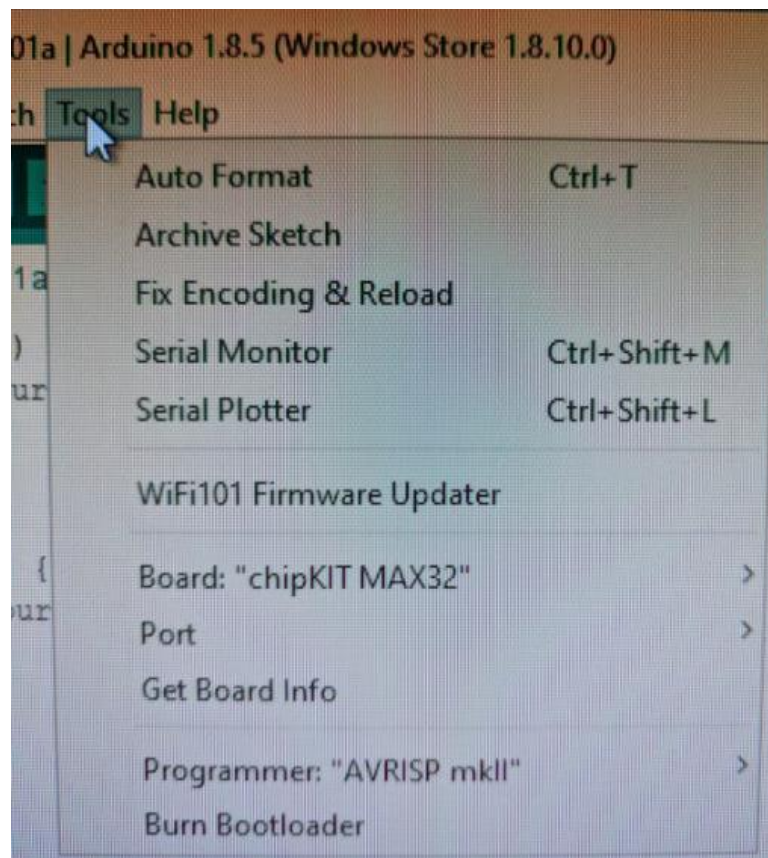
2)

After selecting the board I was using, it was time to upload some simple provided software to make sure everything is in good working order.

Provided with the Arduino IDE is a library of ready to use code that is known to work. From this code I was instructed to select a file named blink, whose purpose is to make the build in LED blink at regular defined intervals.

Before proceeding to upload this code, some time was taken to explain how it worked. From this explanation I gathered a few different topics. First, there are two main sections to an Arduino program – the setup and loop blocks. Setup only runs once and is generally for declaration and assignment. Loop is an infinitely running loop, so any actions here will be taken to no end.

3)

Next was time to compile the code before uploading it to the hardware. Of course, since this code is provided and tested ahead of time it should have no problem compiling and that was the case here. The orange text in the black window below the code is where this IDE outputs status information about the upload process. This gives visual confirmation that will let you know if something isn't working as expected that the problem did not occur from upload.

```
// the setup function runs once when you press reset or power the board
void setup() {
    // initialize digital pin LED_BUILTIN as an output.
    pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
    digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on (HIGH is the voltage level)
    delay(1000);                         // wait for a second
    digitalWrite(LED_BUILTIN, LOW);     // turn the LED off by making the voltage LOW
    delay(1000);                         // wait for a second
}
```

```
Done uploading

Sketch uses 7136 bytes (1%) of program storage space. Maximum is 520192 bytes.
Global variables use 2928 bytes (2%) of dynamic memory, leaving 128144 bytes for local v
Programmer for Microchip PIC32 microcontrollers, Version 2.1.24
        Copyright: (C) 2011-2015 Serge Vakulenko
         Adapter: STK500v2 Bootloader
    Program area: 1d000000-1d1fffff
       Processor: Bootloader
    Flash memory: 2048 kbytes
     Boot memory: 80 kbytes
            Data: 7136 bytes
           Erase: done
   Program flash: .........OOOOOOOOOD######### done
    Verify flash: .........OOOOOOOOOD######### done
    Program rate: 1603 bytes per second
```

4)

Now It is time to modify the blink code to gain some familiarity with serial output and begin moving off the board. We edit the code to output to PIN 11 of the Arduino instead of the built in LED, and due to this an LED must be prepared outside the board. This is accomplished with an LED, a current limiting resistor, a breadboard, and 2 jumper wires to route PIN 11 to the resistor and ground to the other side of the LED. I only had an LED array and resistor array available at the time so I made use of them instead.

The other two additions to the code are serial commands begin and println. The begin command is done in the setup block to initialize the output to a certain rate while the println command in this case was used in the loop block to output a string "Hello World" again and again. To see this output, I opened the serial window seen to the right in the image below:

5)

Now we connect the analog discovery kit to our tested working circuit to get our feet wet with a new platform. Initially I am instructed to take readings of the voltage drop across the current limiting resistor using channel one of the analog discovery kit. This required connecting the orange channel one lead to the positive or LED side of the resistor and the striped orange lead to the grounded side of the resistor. After connecting the leads, I then opened the logger tool of the Waveforms software and configured it as shown in the instructions. Clicking run to view the logged data provided this form:
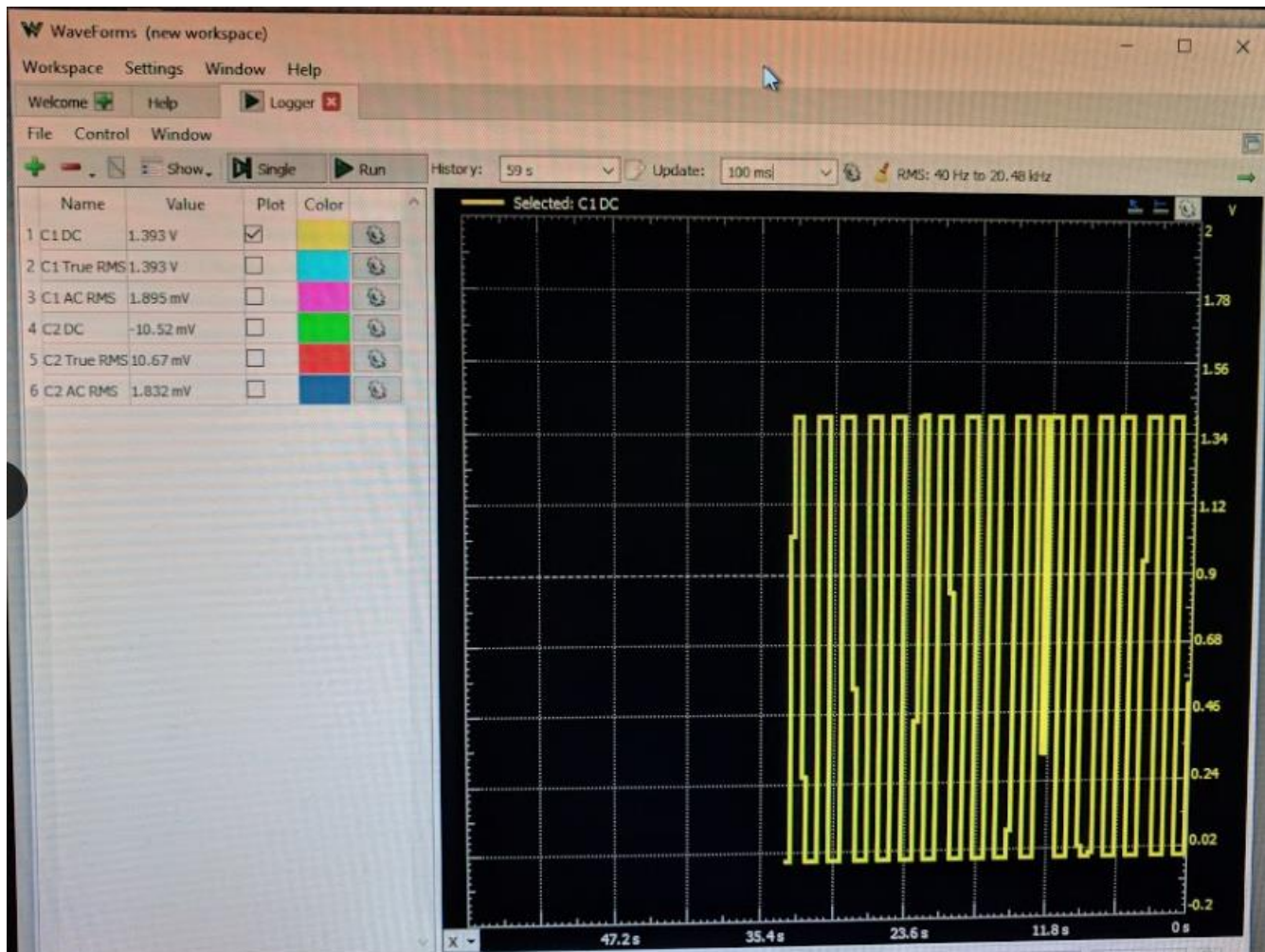
A square wave that oscillates from about 15mV to 1.4V. The jaggedness of the wave form is from the sample rate currently being used. The higher the sample rate, the cleaner this wave would appear in the fast transitioning positive and negative edges.

Next, we are instructed to repeat this procedure for measuring the voltage drop of the resistor and LED combination. To avoid redundancy, the values for the new measurement can be seen in this section's listing files below.

Now I use the scope tool in the Waveform library to analyze the same two scenarios as a moment ago, voltage drop across the resistor alone and then the drop across the resistor LED combination. The scope tool provides some of the same data as the logger but adds a lot of measuring functionality. Now instead of eyeballing values, I can set measurements for maximum and minimum values that can even average over time. These values can also be seen in the listing files below.

LISTING FILES(S):

Waveforms data

Lab 0

Waveforms   min : -4.595 mV (DC) } across resistor R_1 $R_1 =$
             max : 1.394 V   ⎤ logger
             _____⎥
             min : 4.155 mV (DC) } across R, + LED
             max : 3.182 V

       avg min : 3.84 mV
       avg max : 1.39 V   } R,   O-scope
       avg freq : 574.20 mHz

       avg min : 6.70 mV
       avg max : 1.70 V   } R, + LED
       avg freq : 574.27 mHz

          min : 4.09 mV (True RMS)
          max : 1.393 V   } R,
          min : 1.774 mV (AC RMS)
          max : 698 mV

          min : 4.325 mV (True RMS)
          max : 3.18 V   } R, + LED   more
          min : 1.712 mV (AC RMS)   logger
          max : 1.576 V

Arduino Blink code with added serial commands

```
void setup() {
  // put your setup code here, to run once:
  pinMode(11, OUTPUT);

  Serial.begin(9600); //Initialize the serial ouptut to a baud rate of 9600
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(11, HIGH);
  delay(1000);
  digitalWrite(11, LOW);
  delay(1000);

  //Prints Hello World! repeatedly
  Serial.println("Hello World!");
}
```

PART 2

OVERVIEW

Part 2 is to explore another feature of the Arduino / ChipKit MAX32 board, the Analog to Digital Converter. What I know about ADC's going in to this is very basic. I currently only know that a sample rate Is used to check an analog signals level at multiple given times to represent the infinitely fine analog signal as a finite range of digital signals. The greater the sampling rate, the greater the definition of digital data.

LAB DISCUSSION

WORK PERFORMED / SOLUTION:

Using the given diagram, I wired up my Max32 and Analog Discovery kit to both input a waveform via the wave pin on the Analog Discovery, and monitor the waveform on pin A0 using the channel 1 pins in to signal (A0) and ground for reference. Next, if not open, is to open the waveforms program and view channel 1 using the O-scope tool. At this point, a clean sine wave should be visible and we've now confirmed the analog pin A0 and the wave generator function of the Analog Discovery kit are both working. Moving on to see the ADC conversion of this wave, I then uploaded an example Arduino sketch under File >> Examples >> Basics >> AnalogReadSerial. This program takes an analog input of the pin specified and displays the value read to both a serial monitor that can then be viewed via serial plotter. The process of reading a byte of analog data will be done only a certain number of times per second, as specified by Serial.Begin() in the setup block. This is the result:
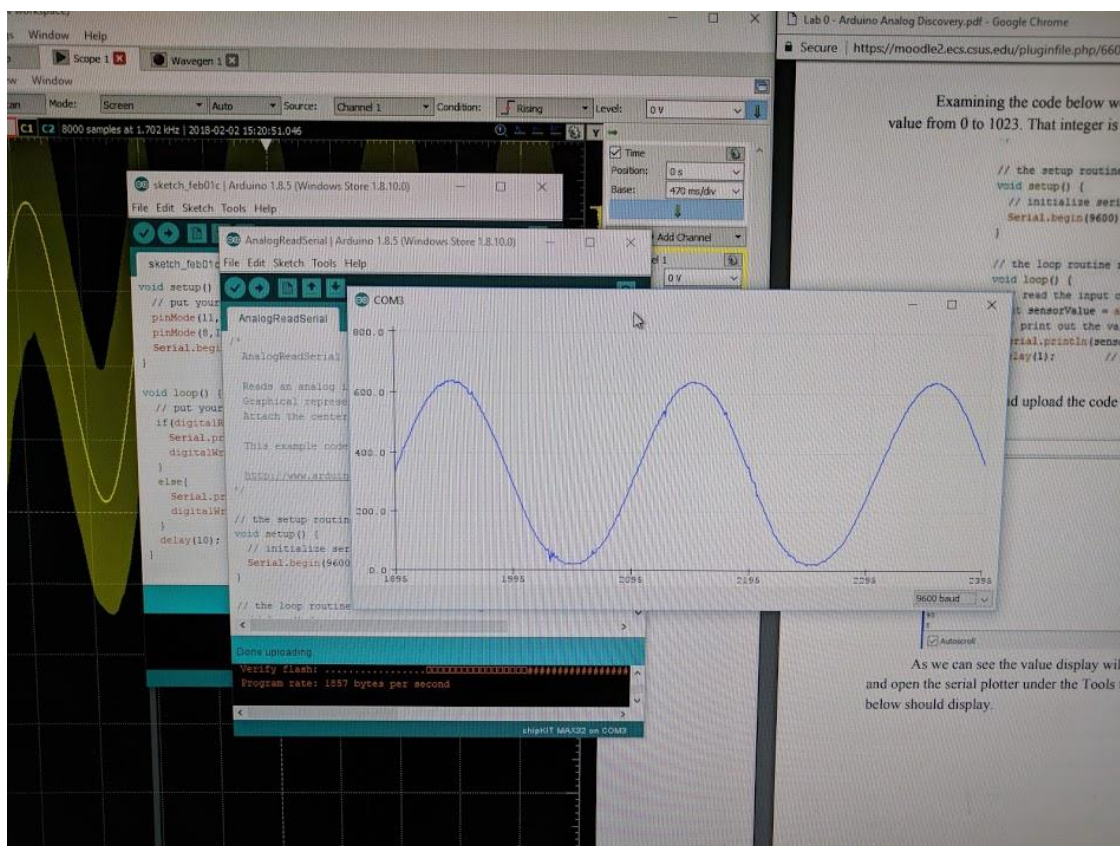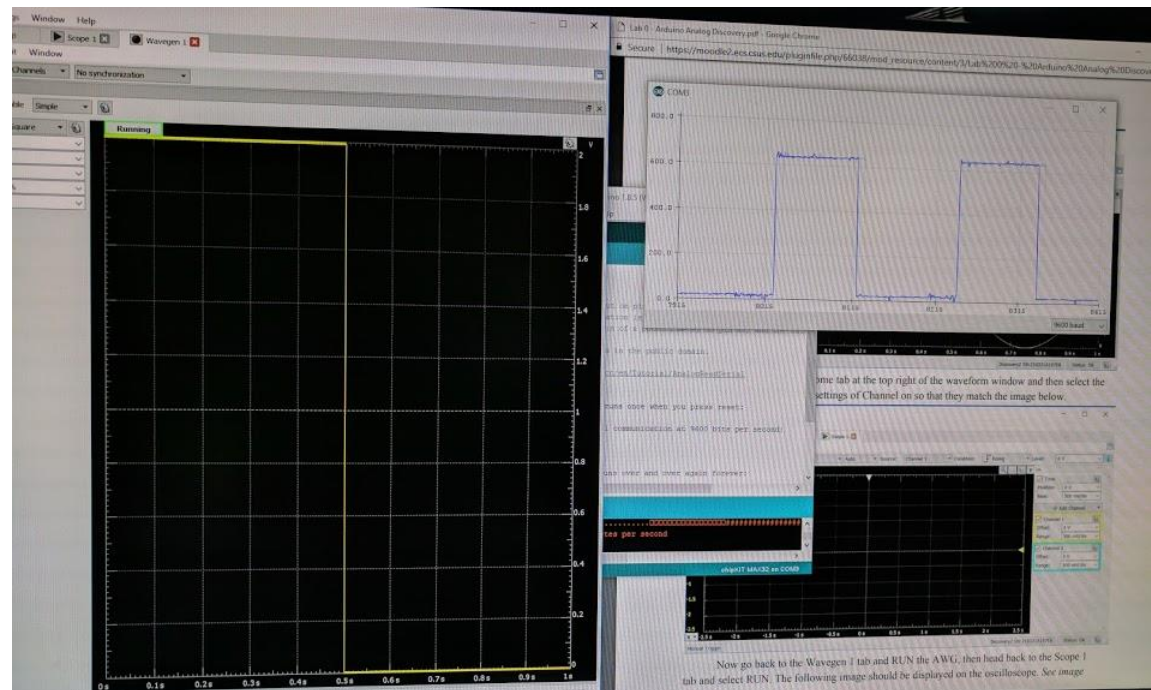

In the listing files below I will show the results for the serial representation of different waveforms created by the Analog Discovery kit by showing the analog waveform generated on the left and the serial representation on the right.
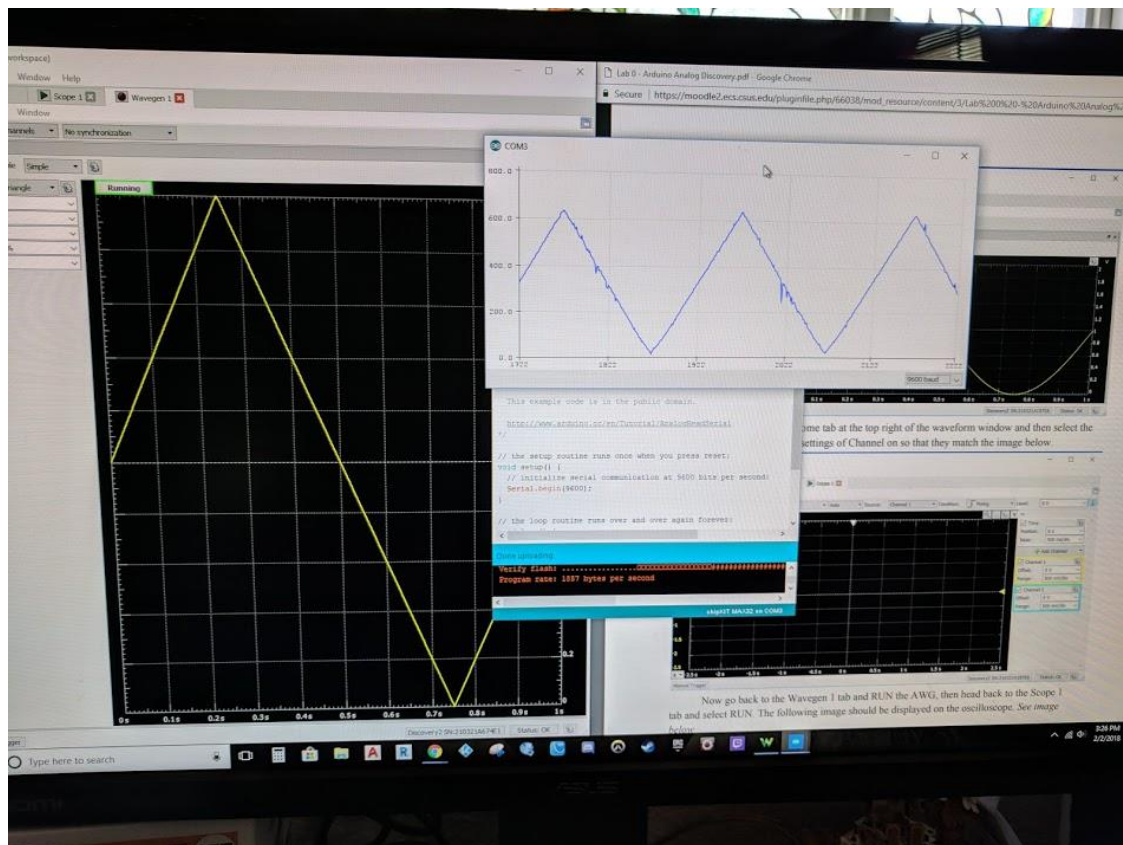
LISTING FILES(S):
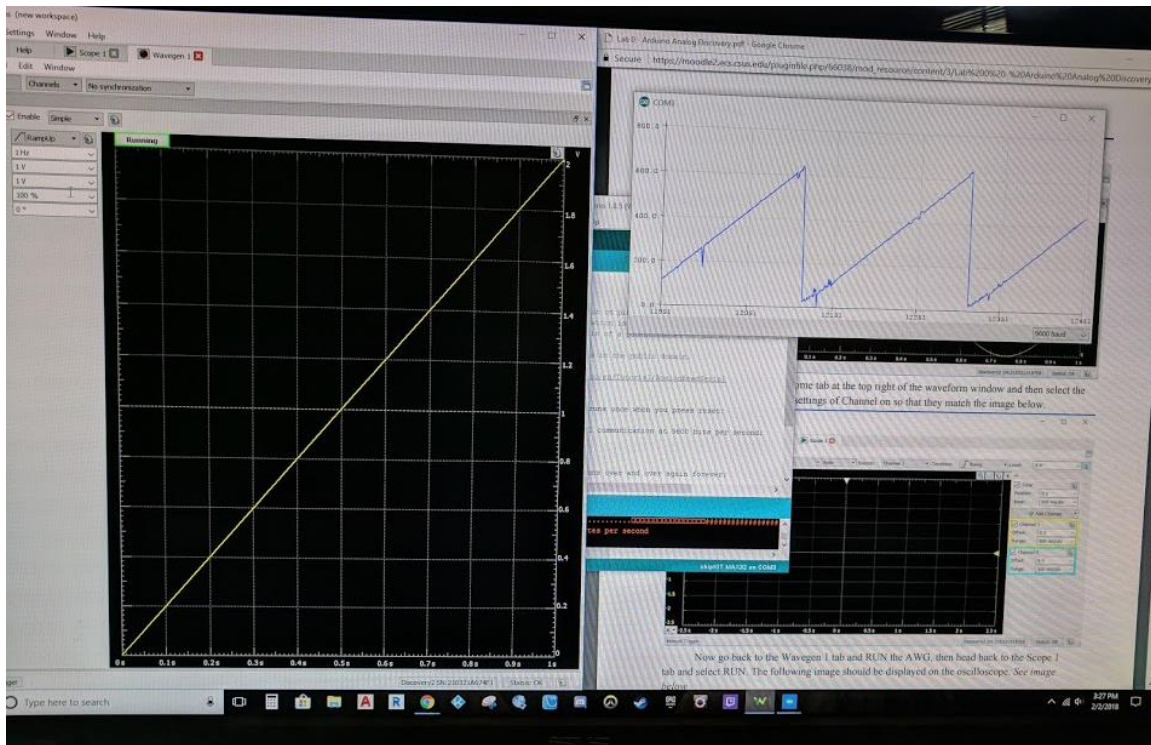
Ramp up wave

Square wave

Triangle wave

PART 3

OVERVIEW

In this part of the lab we will create an analog signal using a digital system via pulse width modulation. Going in to this section I have no previous knowledge of pulse width modulation. We are going to build a dependent simple led circuit to see how this works.

LAB DISCUSSION

WORK PERFORMED / SOLUTION:

The first step of this section is to build on the circuit we were testing a moment ago. I connected a LED in series with a resistor to pin 9 on the Max 32, then connected the other side of the current limiting resistor to ground. We want to monitor this new section with the Analog Discovery kit channel 2, connect the blue positive channel 2 lead to the anode side of the LED and the blue and white stripe channel 2 lead to ground. We then want to upload another piece of example code under File >> Examples >> Analog >> AnalogInOutSerial. Pin 9 is used here for two reasons, reason 1 is because that is what the program we are going to upload calls for. Reason two is because there are designated pins that are capable of outputting a PWM signal. These pins are underlined on the board as 3, 5, 6, and 10. If we tried to do this project with a pin not designated for PWM the LED would still light but would flicker on and off or turn on and off but would not softly transition from bright to dim. After uploading to the board I do indeed get the expected output.
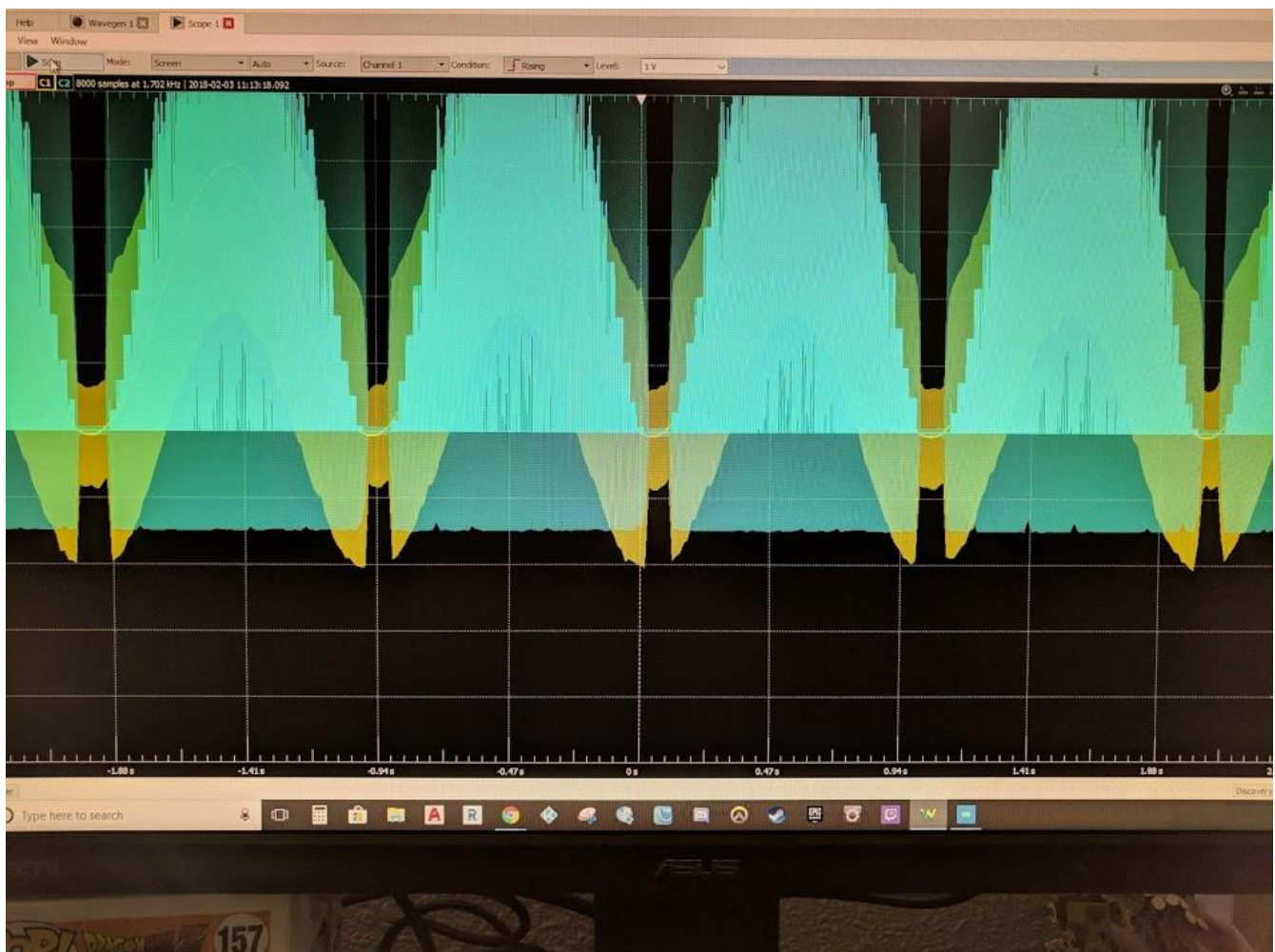
PART 4

OVERVIEW

Choose a project from the list provided and demo your results. Options include Motor Controls, Temperature Sensor, LCD, and Relays. I did not have a temperature sensor, and of the remaining three options I chose motor controls,
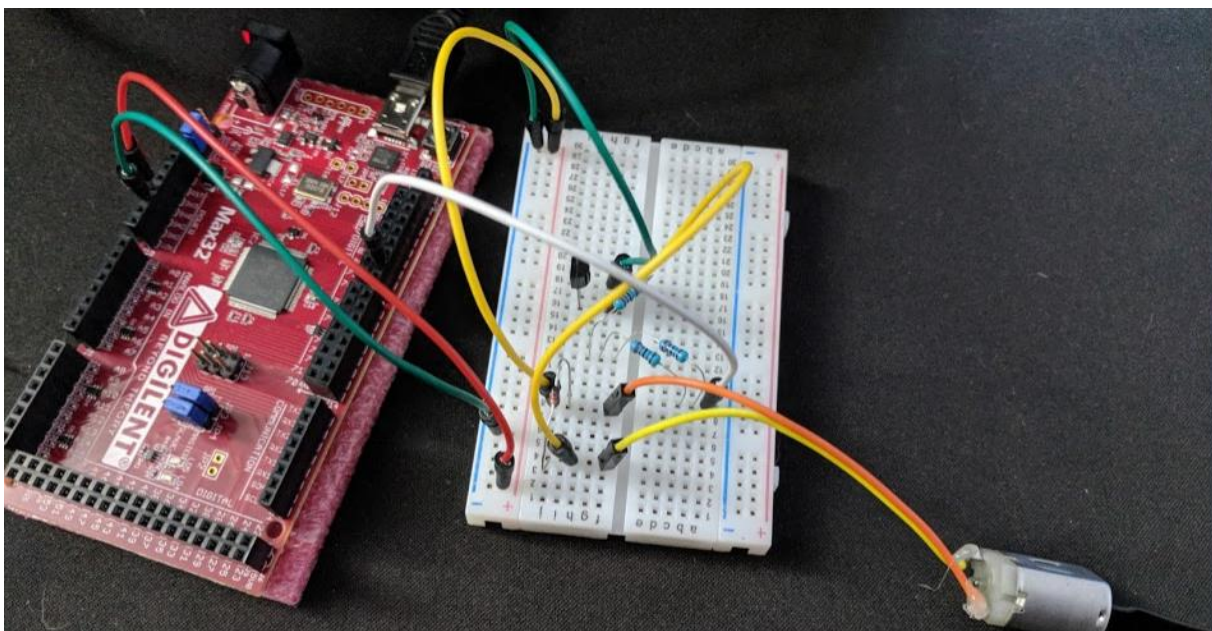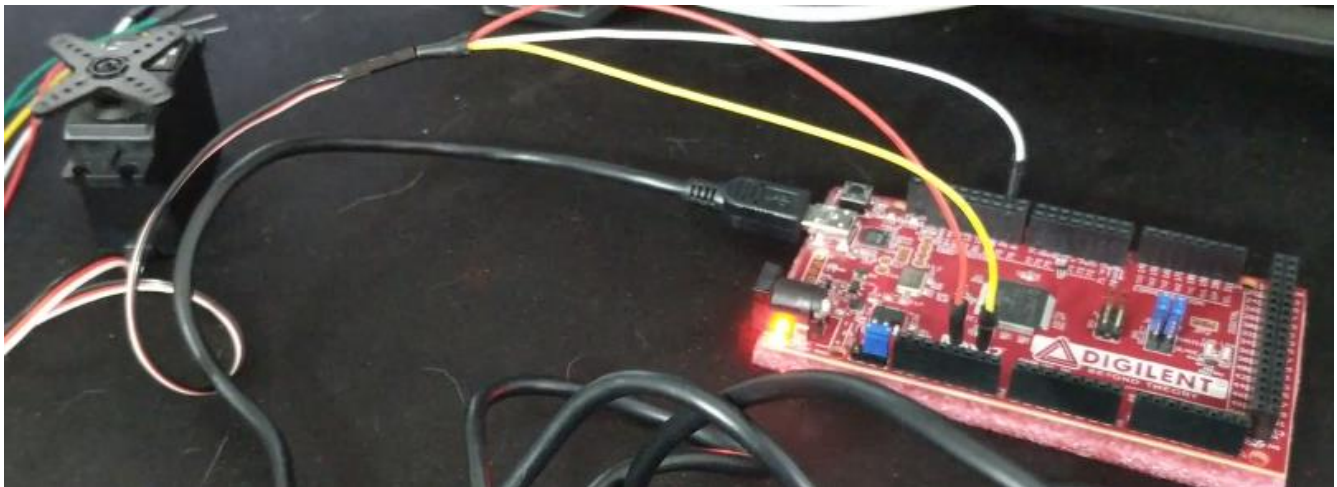
LAB DISCUSSION

WORK PERFORMED / SOLUTION:

Using the provided project links I built both the circuit for the servo motor and DC motor. (https://learn.sparkfun.com/tutorials/sik-experiment-guide-for-arduino---v32/experiment-8-driving-a-servo-motor)

(https://learn.sparkfun.com/tutorials/sik-experiment-guide-for-arduino---v32/experiment-12-driving-a-motor)

When I built these circuits and uploaded their respective programs on campus during lab they both did not work. With the servo circuit I tried using an external power source to no avail then tried driving it with my Arduino Uno using both internal and external power sources and still received no motor action. I then tried all previous permutations with a different servo and still no action. At this point I needed some form of success

and decided to move on to the DC motor temporarily. After building its circuit and uploading the appropriate program I instead ended up doubling down with two failures. Luckily, I got both circuits to work after I got home. As for the servo motor circuit I am still unsure when the original issue was. I hooked everything back up the way it was the first time (power and ground from the MAX32), ran the USB cable to my computer and before I could even try to upload the program again it began working since the program was already stored in memory. I could only come up with one reason for the previous issue, I may have a faulty jumper wire in my kit. The DC motor required a bit more investigating but I eventually found the program. The NPN transistor I was using had a spec sheet that gave reversed pin designations in comparison to the one that was used in the tutorial. After flipping the transistor around, everything worked fine.

LISTING FILES(S):

·M·C·C·
Micro Commercial Components

Micro Commercial Components
20736 Marilla Street Chatsworth
CA 91311
Phone: (818) 701-4933
Fax: (818) 701-4939

S8050-B
S8050-C
S8050-D

NPN Silicon Transistors

Features

- TO-92 Plastic-Encapsulate Transistors
- Capable of 0.625Watts(Tamb=25°C) of Power Dissipation.
- Collector-current 0.5A
- Collector-base Voltage 40V
- Operating and storage junction temperature range: -55°C to +150°C
- Marking : S8050
- Lead Free Finish/RoHS Compliant ("P" Suffix designates RoHS Compliant. See ordering information)
- Case Material: Molded Plastic. UL Flammability Classification Rating 94V-0 and MSL Rating 1

TO-92

Electrical Characteristics @ 25°C Unless Otherwise Specified

Symbol	Parameter	Min	Max	Units
OFF CHARACTERISTICS				
$V_{(BR)CBO}$	Collector-Base Breakdown Voltage (I_C=100uAdc, I_E=0)	40	----	Vdc
$V_{(BR)CEO}$	Collector-Emitter Breakdown Voltage (I_C=0.1mAdc, I_B=0)	25	----	Vdc
$V_{(BR)EBO}$	Emitter-Base Breakdown Voltage (I_E=100uAdc, I_C=0)	5.0	----	Vdc
I_{CBO}	Collector Cutoff Current (V_{CB}=40Vdc, I_E=0)	----	0.1	uAdc
I_{CEO}	Collector Cutoff Current (V_{CE}=20Vdc, I_B=0)	----	0.1	uAdc
I_{EBO}	Emitter Cutoff Current (V_{EB}=3.0Vdc, I_C=0)	----	0.1	uAdc
ON CHARACTERISTICS				
$h_{FE(1)}$	DC Current Gain (I_C=50mAdc, V_{CE}=1.0Vdc)	85	300	----
$h_{FE(2)}$	DC Current Gain (I_C=500mAdc, V_{CE}=1.0Vdc)	50	----	----
$V_{CE(sat)}$	Collector-Emitter Saturation Voltage (I_C=500mAdc, I_B=50mAdc)	----	0.6	Vdc
$V_{BE(sat)}$	Base-Emitter Saturation Voltage (I_C=500mAdc, I_B=50mAdc)	----	1.2	Vdc
V_{BE}	Base- Emitter Voltage (I_C=100mAdc)	----	1.4	Vdc
SMALL-SIGNAL CHARACTERISTICS				
f_T	Transistor Frequency (I_C=20mAdc, V_{CE}=6.0Vdc, f=30MHz)	150	----	MHz

CLASSIFICATION OF h_{FE}

Rank	B	C	D
Range	85-150	120-200	160-300

DIM	INCHES		MM		NOTE
	MIN	MAX	MIN	MAX	
A	.170	.190	4.33	4.83	
B	.170	.190	4.30	4.83	
C	.550	.590	13.97	14.97	
D	.010	.020	0.36	0.56	
E	.130	.160	3.30	3.96	
G	.096	.104	2.44	2.64	

PART 5

OVERVIEW

Using the provided references and I^{2}C communication code, enable one MAX32 to speak with a partners MAX32. After successfully communicating between the two devices, use the bus analyzer on the Analog Discovery kit to observe the signal.
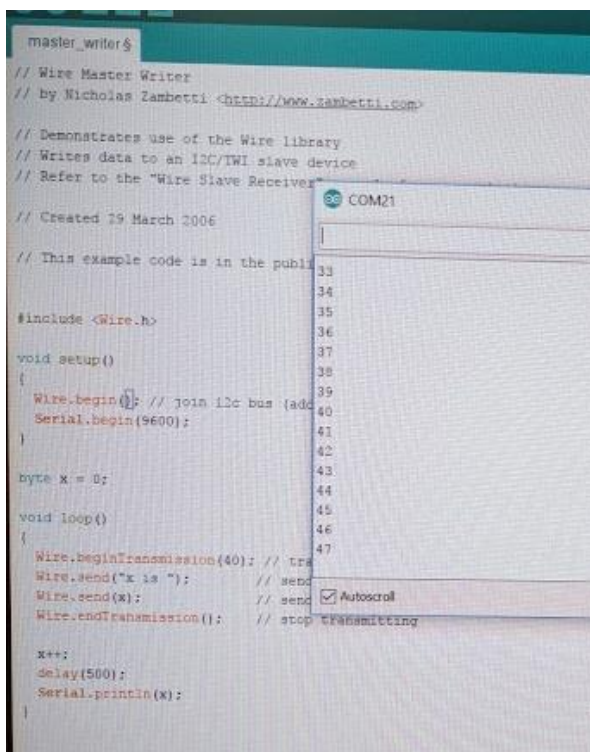
https://learn.sparkfun.com/tutorials/i2c

https://www.arduino.cc/en/Reference/Wire

http://www.allaboutcircuits.com/technical-articles/introduction-to-the-i2c-bus/

LAB DISCUSSION

WORK PERFORMED / SOLUTION:

My partner and I took the incorrect approach here and tried to race to the finish only to end up wasting a bunch more time than if we read the references from the start. After learning there were two dedicated pins for clock and data we first tried connecting just those pins together and got nowhere. We then learned we needed to share a common ground. After connecting our two grounds together we still had no transmission. It was at this point we were stumped for a moment. The information we found online pertained to Arduino's only in a way that was unexpected. With the Arduino boards, the pull up values required for the pins are built in but this is not true for the MAX32 boards. The MAX32 boards require an pull up resistor to be used for both the shared clock and data signals. Once incorporating this into our design, we saw transmission. The twist is that these pull up resistors are part of the schematic shown in the first reference site given. Lesson learned, do not try and skip ahead.

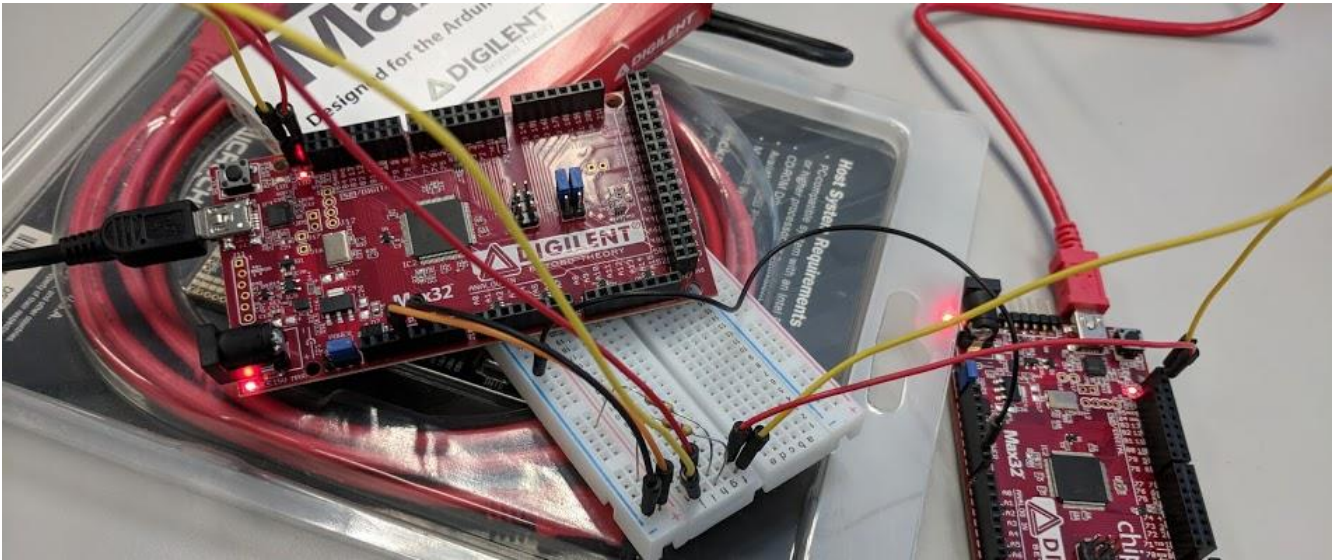This is what we saw on our displays after I modified the Master_Writer code to show the value of x:

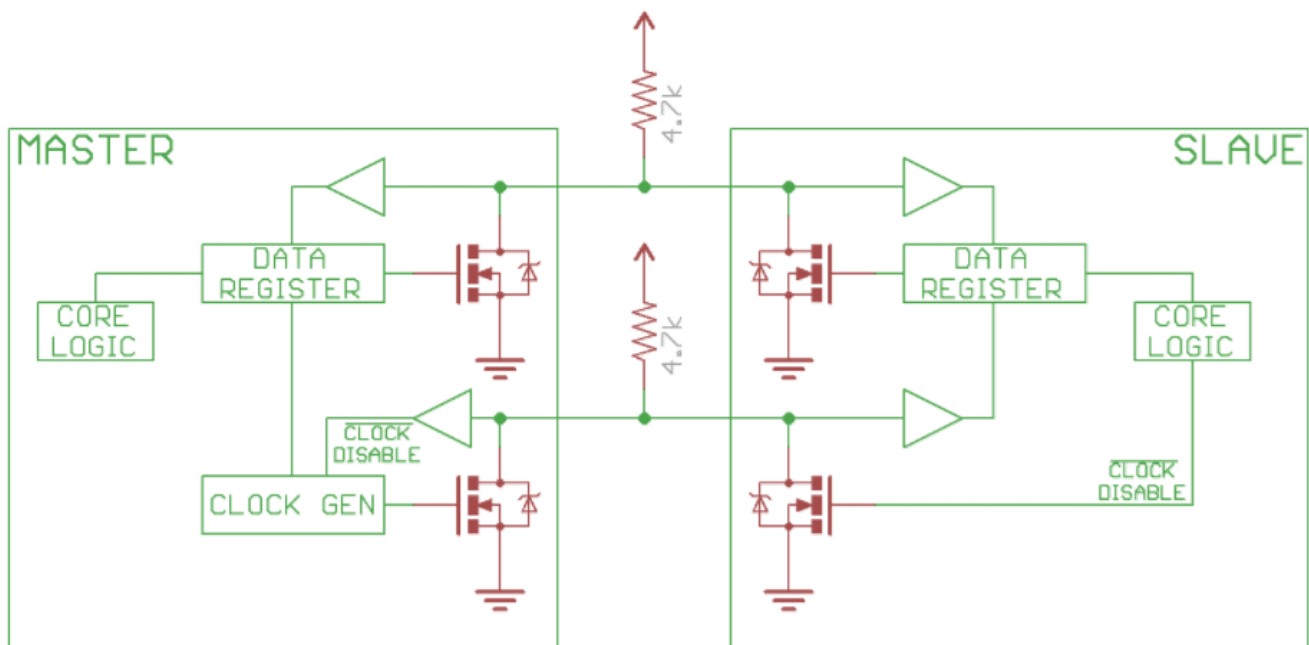On the left is the sending side, and on the right is the receiving end.
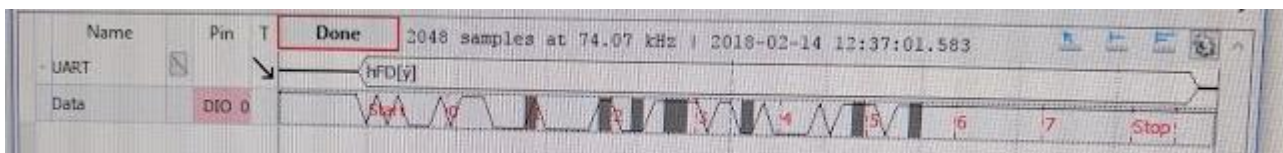
This is the data we observed through the bus analyzer. The transmission format is in the listing files below.
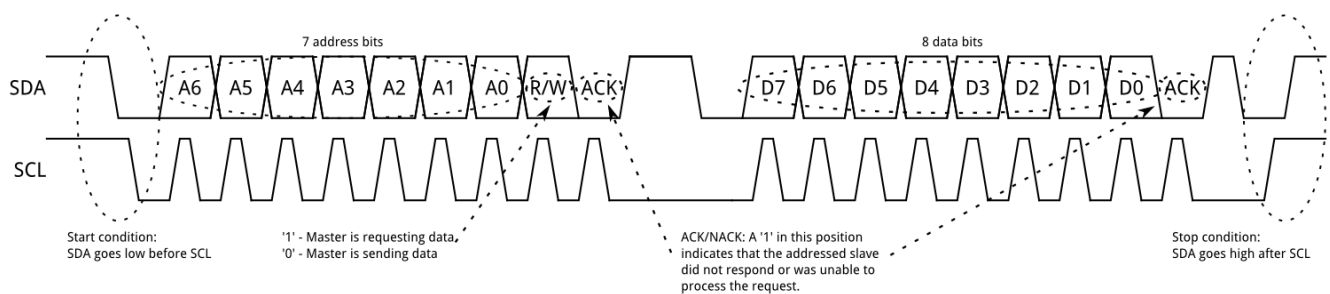
LISTING FILES(S):
(The connection with pull up resistors)

(Very helpful, yet initially overlooked schematic)

(Transmission sequence)

PART 6

OVERVIEW

Now we learn the final aspect of this lab before designing our own piece, interrupts. I am currently unsure if this is the same idea of interrupts as it is with assembly. To tackle part 6 we are instructed to follow the tutorial at https://www.sparkfun.com/tutorials/326 followed by a specific timer interrupt lab at http://chipkit.net/interrupts-made-easy-with-chipkit/ .

(References)

https://www.sparkfun.com/tutorials/326

https://www.arduino.cc/en/Reference/AttachInterrupt

http://www.instructables.com/id/Timer-Interrupts-on-the-DP32/

https://reference.digilentinc.com/chipkit_max32/refmanual

LAB DISCUSSION

WORK PERFORMED / SOLUTION:

After reading the first tutorial I have a good idea of how this works now. Interrupts seem to be best saved for code that needs to be immediately executed at just about any time. The code needs to be relatively short because this interrupt halts everything else that was previously being done. An important note near the end that reinforces this point is the inability to use the delay function. I then proceeded to wire up my breadboard to the requirements of the first program under listing files. (LED pin 13 is shared with the built in LED).
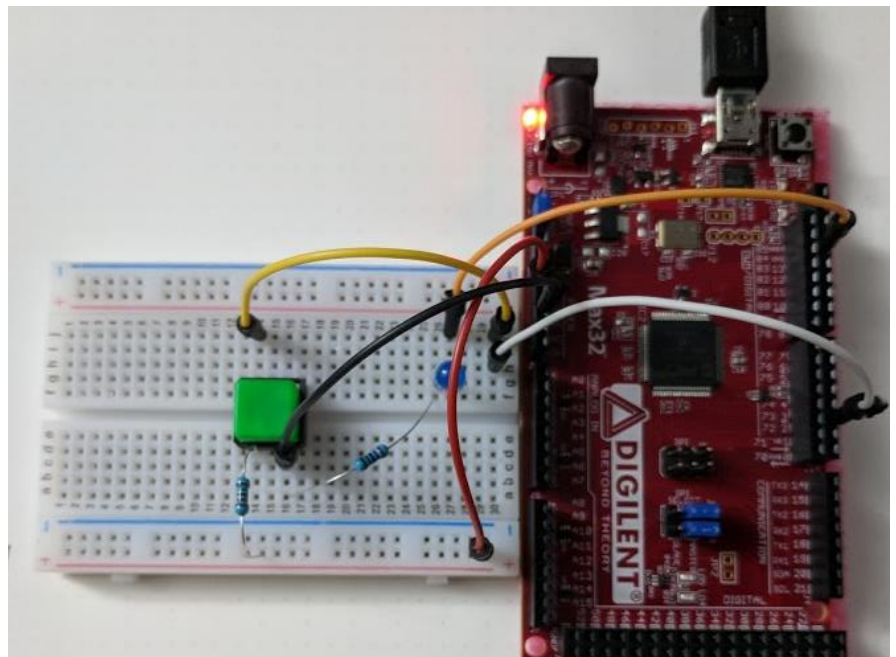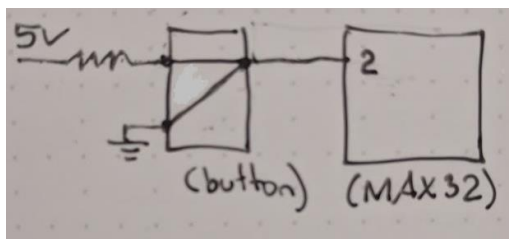
I learned 2 things when testing this code and circuit. The dead jumper wire I suspected from my attempt to drive DC and servo motors is real, and MAX32 assigns pins differently than Arduino for their interrupts. Following my circuit from ground all the way through to the signal wire going to the MAX32 revealed almost a complete loss of signal in the wire used to bridge my board and MAX32 together. Swapping the bad wire fixed the issue. After the circuit still did not work as expected I then checked the corresponding pinouts and found INT0 on MAX32 is pin 3 and not pin 2 as expected with the Arduino boards. INT1 is pin 2 for the MAX32 instead. Changing that small bit of code allowed everything to work as expected.

LISTING FILES(S):

```
/*
Simple Interrupt example 2
by: Jordan McConnell
SparkFun Electronics
created on 10/29/11
license: Beerware- feel free to use this code and maintain
attribution. If we ever meet and you are overcome with gratitude,
feel free to express your feelings via beverage.
*/

int ledPin = 13; // LED is attached to digital pin 13
int x = 0; // variable to be updated by the interrupt

//variables to keep track of the timing of recent interrupts
unsigned long button_time = 0;
unsigned long last_button_time = 0;

void setup() {
  //enable interrupt 0 which uses pin 2
  //jump to the increment function on falling edge
  attachInterrupt(0, increment, FALLING);
  Serial.begin(9600);  //turn on serial communication
}

void loop() {
  digitalWrite(ledPin, LOW);
  delay(3000); //pretend to be doing something useful
  Serial.println(x, DEC); //print x to serial monitor
}

// Interrupt service routine for interrupt 0
void increment() {
  button_time = millis();
  //check to see if increment() was called in the last 250 milliseconds
  if (button_time - last_button_time > 250)
  {
    x++;
    digitalWrite(ledPin, HIGH);
    last_button_time = button_time;
  }
}
```

```
/* This value works for DP32, at 40 MHz */
#define TICKS_PER_SECOND 40000000

#define T3_ON 0x8000
#define T3_PS_1_1 0
#define T3_SOURCE_INT 0
volatile uint32_t counter = 0;

/* Define the Interrupt Service Routine (ISR) */
void __attribute__((interrupt)) myISR() {
  counter++;
  clearIntFlag(_TIMER_3_IRQ);
}
```

```
/* start_timer_3 */
void start_timer_3(uint32_t frequency) {
  uint32_t period;
  period = TICKS_PER_SECOND / frequency;
  T3CONCLR = T3_ON;            /* Turn the timer off */
  T3CON = T3_PS_1_1;           /* Set the prescaler  */
  TMR3 = 0;                    /* Clear the counter  */
  PR3 = period;                /* Set the period     */
  T3CONSET = T3_ON;            /* Turn the timer on  */
}

void setup() {
  start_timer_3(8000);   /* 8 kHz */
  setIntVector(_TIMER_3_VECTOR, myISR);
  setIntPriority(_TIMER_3_VECTOR, 4, 0);
  clearIntFlag(_TIMER_3_IRQ);
  setIntEnable(_TIMER_3_IRQ);
  Serial.begin(9600);
}

void loop() {
  while (counter < 160000) {
    delay(1000);
    Serial.print("Count is now: ");
    Serial.println(counter);
  }
  counter = 0;
}
```

PART 7

OVERVIEW

We are now let loose into the wild to create our own project with the following requirements: Collect some form of analog signal (temperature, sound etc) and represent it in some physical way (LED, LCD, buzzer). After looking inside my kit for a moment I really wanted to know how the analog joystick worked so I chose to receive data from the joystick and represent it with 4 PWM LEDs, one for each cardinal direction. A fifth LED was added after realizing the joystick had a center button

LAB DISCUSSION

Work Performed / Solution:

I often catch myself trying to think through a process one hundred percent before taking even a single step. This usually works well and gives me time to try and catch an element or two that was not immediately apparent but, in this case, it caused me to stare without taking a single step for quite a while, so I decided I would accomplish the easy task first just to get my hands moving.

I began by arranging the 5 LED's on my breadboard the way I wanted them to be, a center red LED for the button surrounded by 4 blue LED's for North, South, East, and West. I then ran 220 ohm pull down resistors from the cathode side of each LED to ground. After doing that I oriented the joystick in a way that felt natural, so I wouldn't mis up the directions while wiring everything together. I then ran wires from all 5 pins of the joystick to corresponding locations on the breadboard. 5V to my designated VCC rail, ground to my ground rail, and then plugged the x, y, and button signals into empty rows to be dealt with later.

I decided I would then try and tackle making the center button correspond to turning on the red center LED first. Unsure if the button signal provided HIGH when pressed or LOW when pressed I hooked up my DMM to find out, black prong to ground and red prong to the button signal. Pressing the button under this condition gave me a LOW signal so I wired a pull up resistor from the button lead to VCC and tested again. The second test gave me a more reliable signal, 4.8V when not pressed and 0V when pressed. I then proceeded to write an if statement for turning the LED on under a LOW condition and to output the buttons status in the serial monitor. I ran into my first problem here.

I was able to see in the serial monitor that my button and the variable attached to its status were working properly but the LED would not turn off. I then looked at my code and realized I didn't write an else statement to turn it off when not pressed. After fixing this issue the center button and LED worked as expected.

The more exciting task was now ahead, figuring out how to represent the signal from the joystick, as this point I still didn't even know what the signal from the joystick looked like so I modified the Arduino example AnalogInOut to display both the x data and the y data. Without moving the stick at all I then saw this:

I proceeded to move the joystick around and found the range of output for the x and y axis to be 0-1023. Without recognizing what is so obvious now, I proceeded to write code to handle the case for x being in the bottom half of the range vs the top half and then copied and modified this code for the y axis. The initial test was underwhelming. I wired the y axis LED's North to South and vice versa, but also 2 of the LED's were lit very well without me touching a thing.

After fixing the simple pinout issue I looked at my code again and still couldn't find the problem, so I resorted to viewing the serial monitor again, this time noticing that despite my assumption the values of x and y for the middle, untouched position was not dead center of the 0-1023 range. They were instead in the mid 700's. Taking note of this I then drew a representation of the joystick, giving ranges for all 4 directions. I watched the serial monitor once more and took note of the data range for x and y when the stick was untouched and saw they varied slightly on their own. To avoid flickering LED's I decided to make this range a dead zone, where PWM output for the corresponding axis LEDs are both 0.

After changing my previous ranges to the newly defined ones, the result was closer to what I expected but still not quite right. The upper ranges of both axis worked fine but the lower range gave a sort of opposite effect than desired. As the stick was pushed further into the lower range the corresponding LED would go from immediately bright to dim instead of the other way around. A moment of thought made me realize that growing numbers in these ranges should not mean growing numbers for PWM output. For example, pushing the joystick west would meant the analog signal would fade from around 700 to 0 but my code used map in a positive relationship instead of the negative one desired. One last update to fix the relationship issue finally completed this project. This was the result:
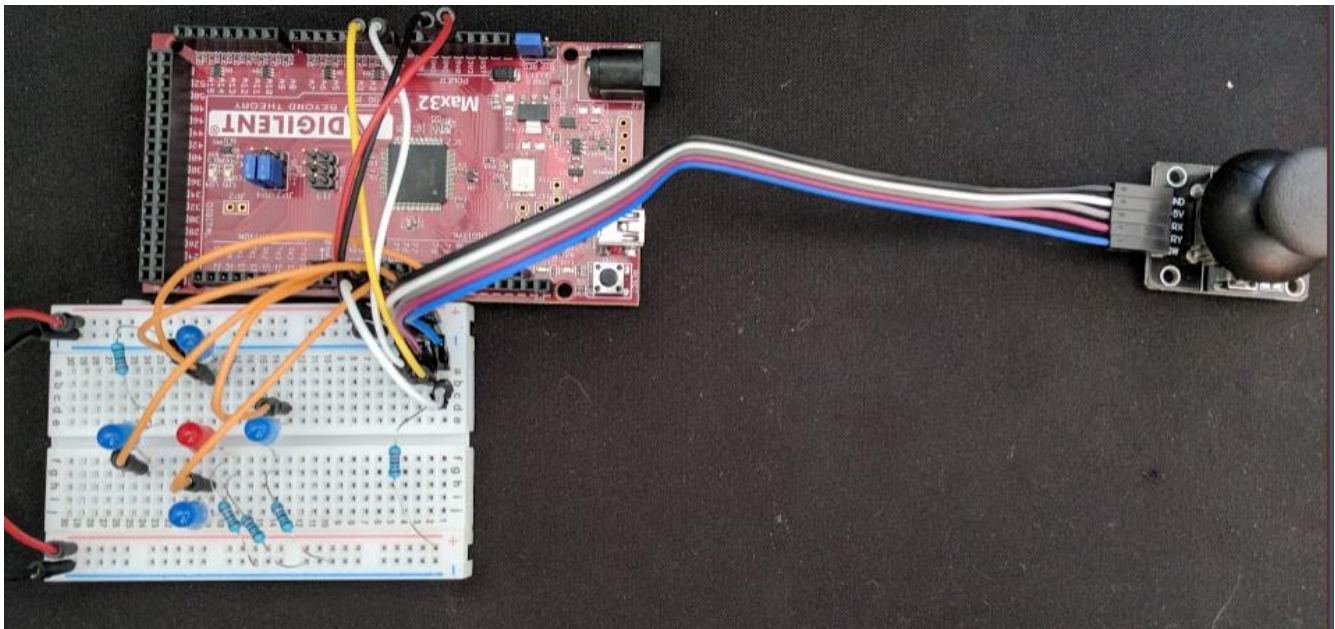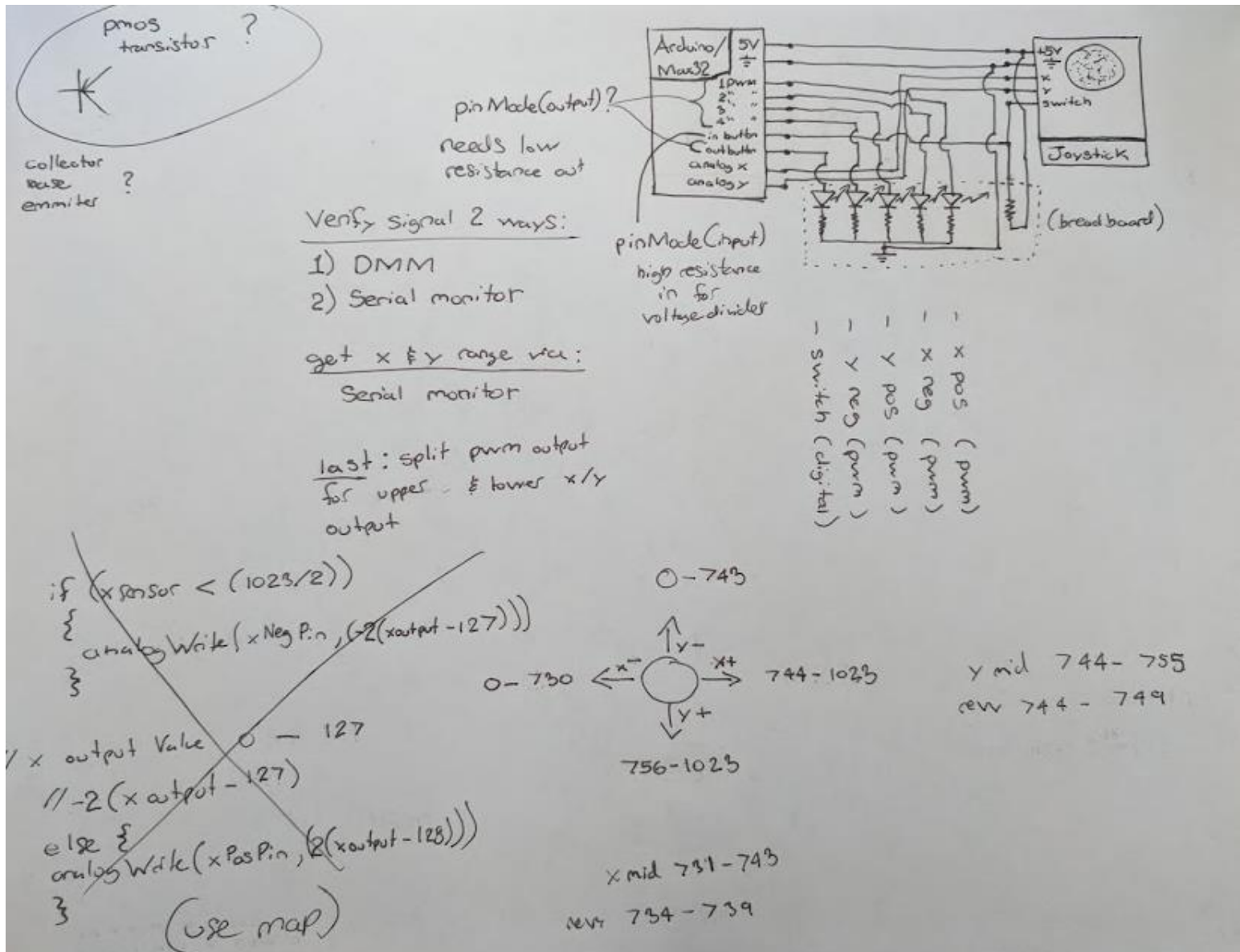

The final versions of the schematic and Arduino sketch are below in the listing files section.

LISTING FILES(S):

Digilent chipKIT MAX32 detailed pinout reference: https://reference.digilentinc.com/chipkit_max32/refmanual

Schematic with joystick analog ranges and revised dead zones.

Final version of code.

```
/*
  Modified by Andrew Robertson for an analog joystick
  to 5 leds for x position (left right), y position (left right)
  and center button. All but center button are PWM driven

  Analog input, analog output, serial output

  Reads an analog input pin, maps the result to a range from 0 to 255 and uses
  the result to set the pulse width modulation (PWM) of an output pin.
  Also prints the results to the Serial Monitor.

  created 29 Dec. 2008
  modified 9 Apr 2012
  by Tom Igoe

  This example code is in the public domain.

  http://www.arduino.cc/en/Tutorial/AnalogInOutSerial
*/

// These constants won't change. They're used to give names to the pins used:
const int xanalogInPin = 54;
const int yanalogInPin = 55;
const int buttonInPin = 4;
const int buttonLEDPin = 7;
const int xPosAnalogOutPin = 3;
const int xNegAnalogOutPin = 5;
const int yPosAnalogOutPin = 6;
const int yNegAnalogOutPin = 9;

int xsensorValue = 0;
int ysensorValue = 0;
int buttonStatus = 1;
int xoutputValue = 0;
int youtputValue = 0;

void setup() {

  pinMode(buttonInPin, INPUT);
  pinMode(buttonLEDPin, OUTPUT);
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);
}
```

```
void loop() {
 // read the analog in value:
 xsensorValue = analogRead(xanalogInPin);
 // change the analog out value:
 if(xsensorValue < 734){
  // map it to the range of the analog out:
   xoutputValue = map(xsensorValue, 0, 733, 0, 255);
   xoutputValue = -1 * (xoutputValue - 255);
   analogWrite(xNegAnalogOutPin, xoutputValue);
 }
 else if(xsensorValue > 739){
  // map it to the range of the analog out:
   xoutputValue = map(xsensorValue, 740, 1023, 0, 255);
   analogWrite(xPosAnalogOutPin, xoutputValue);
 }
 else{
   analogWrite(xNegAnalogOutPin, 0);
   analogWrite(xPosAnalogOutPin, 0);
 }

 // read the analog in value:
 ysensorValue = analogRead(yanalogInPin);
 // change the analog out value:
 if(ysensorValue < 744){
  // map it to the range of the analog out:
   youtputValue = map(ysensorValue, 0, 743, 0, 255);
   youtputValue = -1 * (youtputValue - 255);
   analogWrite(yNegAnalogOutPin, youtputValue);
 }
 else if(ysensorValue > 749){
  // map it to the range of the analog out:
   youtputValue = map(ysensorValue, 750, 1023, 0, 255);
   analogWrite(yPosAnalogOutPin, youtputValue);
 }
 else{
   analogWrite(yNegAnalogOutPin, 0);
   analogWrite(yPosAnalogOutPin, 0);
 }

 // print the results to the Serial Monitor:
 Serial.print("X sensor = ");
 Serial.print(xsensorValue);
 Serial.print("\t X output = ");
 Serial.print(xoutputValue);
 Serial.print("\t");
```

```
// print the results to the Serial Monitor:
Serial.print("Y sensor = ");
Serial.print(ysensorValue);
Serial.print("\t Y output = ");
Serial.print(youtputValue);
Serial.print("\t");

buttonStatus = digitalRead(buttonInPin);
Serial.print("\tButton Status = ");
Serial.println(buttonStatus);
if ((buttonStatus == 0)){
  digitalWrite(buttonLEDPin, HIGH);
}
else{
  digitalWrite(buttonLEDPin, LOW);
}
// wait 2 milliseconds before the next loop for the analog-to-digital
// converter to settle after the last reading:
delay(2);
}
```