Lab 3: Embedded Systems

Analog to Digital Converter

Critical Design Review

EE 300W

Section 6

Team: Wheelers

Roshana Molligoda, JongWoo Ha

Submitted: April 29th, 2019

Table of Contents

Abstract

Team Wheelers designed and implemented an Analog to Digital Converter (ADC) as part of a larger system to determine the unknown transfer function of the black box. The physical circuit included a microcontroller, an ADC and a CAN bus. These components were wired in such a way that the microcontroller accurately with the other components. The microcontroller was coded using the mbed interface. The code was organized in such a way that the CAN message from the keypad was received by the microcontroller and triggered the analog to digital conversion. While our team designed the ADC successfully, we were not able to receive commands from the keypad and integrate it with the other subsystems.

Introduction

Electrical Engineers are interested in transfer functions that dictate the input/output relationship of electrical systems. In this lab, our class was given a "black box", a component with an unknown transfer function. Even without knowing the other components, the function of the black box can be determined by applying a wide range of inputs to the system and measuring the corresponding outputs. The goal of the entire embedded system was to determine the transfer function of the black box.

The entire system is made up of four subsystems. The digital to analog converter (DAC) generates an input waveform of both sinusoidal and square wave at 1Hz and 10Hz. The digital potentiometer (DigiPot) modulates the output from the DAC to different voltage ranges. The output from the DigiPot is the input for the black box. The ADC that our team was in charge of, samples the output of the black box at 100Hz. This sampled data is used to determine the transfer function of the black box. The keypad acts as an interface for the user to control the above subsystems.

Analog-to-digital conversion is an essential operation because, while some systems interact with digital values, the other subsystems usually deal with analog signals. This is found in systems such as pressure gauges, a microphone/speaker system, measuring temperature, and so on. All of these systems require the analog data to be converted into digital data before going into the microcontroller.

Rationale

Theory of Operation

The ADC will sample, quantize, and code an analog input signal into a digital output signal. Two major considerations of the ADC are the bit resolution and the sampling rate. The bit resolution determines the accuracy of the ADC. The sampling rate determines the number of samples to be measured within one period of the analog signal. For this lab, designing the ADC involves working with the mbed interface. For the conversion portion, our team used the voltage resolution as a conversion factor to convert the analog input.
The equation is as follows:

$$Voltage\ Resolution = \frac{Input\ Voltage\ Range}{2^{Bit\ Resolution}}\ (V)$$

Theory of Interfaces

As shown in Figure 1 below, the ADC needs to receive commands from the keypad to start and stop, and the sampled data has to be outputted to a PC terminal over USB. Our team had to write two codes in the microcontroller, one for the actual analog to digital conversion and one for receiving messages from the keypad. The communication with the keypad was done using the CAN bus, and our code had to receive the message through the CAN bus onto our microcontroller. In order to do this, the shared CANH and CANL would be connected in between the keypad and the ADC with two resistors. Numerous print statements would need to be included throughout our code to see whether the code works properly or not.

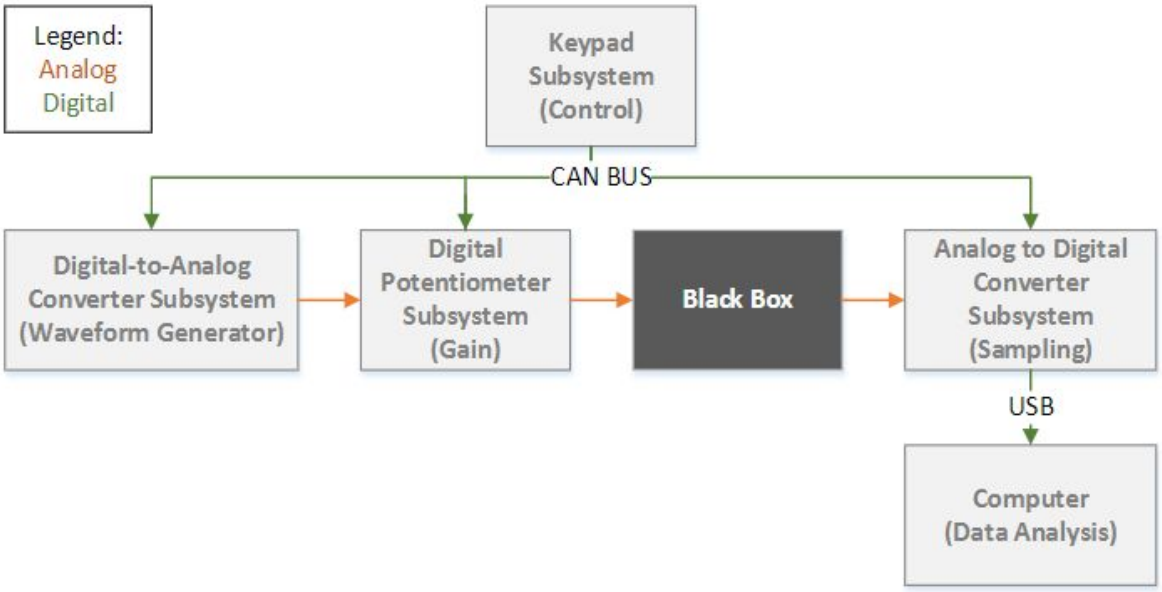

Figure 1: Block diagram of the integrated system

Implementation

Explanation of Circuitry and Code

Physical Circuitry

Figure 2 shows the physical circuitry of the ADC subsystem. The circuitry was designed to connect the microcontroller with the ADC and the CAN bus. Our team used the mbed LPC1768 microcontroller, ADS1015 and the MCP2551 CAN bus.

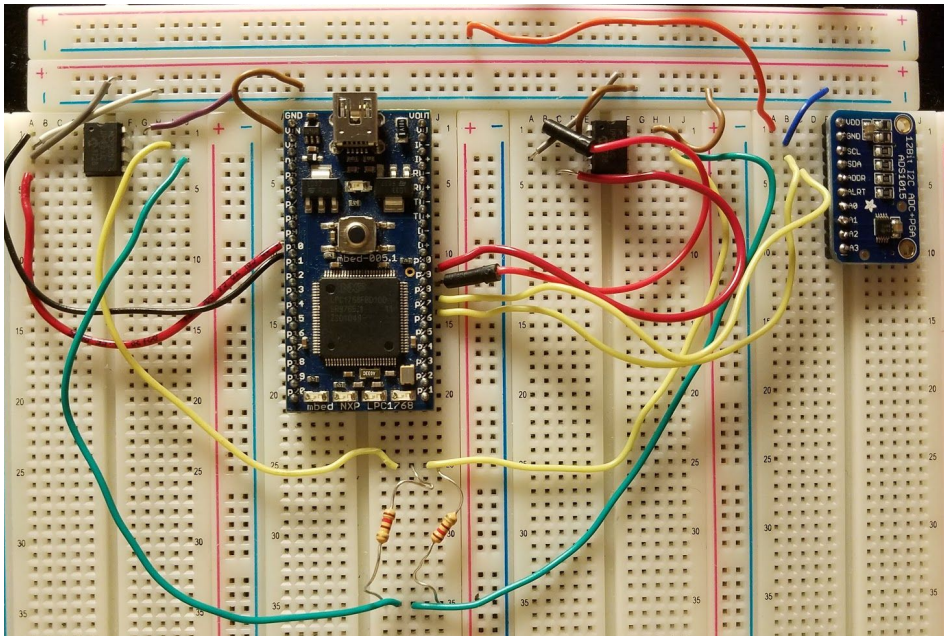

Figure 2: Physical circuitry of the ADC subsystem

The block diagram of ADS1015 is shown in Figure 3. The ADS1015 was powered with a 3.3V DC power source. SCL and SDA, the two pins used by the I2C interface on ADS1015, were connected accordingly onto pins 27 and 28 on the LPC1768 that are dedicated for I2C configuration. The pin configuration of the LPC1768 can be seen from Figure 5. This connection allowed the ADS1015 to properly interact with the microcontroller. AIN0 was used as a single-ended channel input for the output signal from the black box, which was the analog input for ADC.

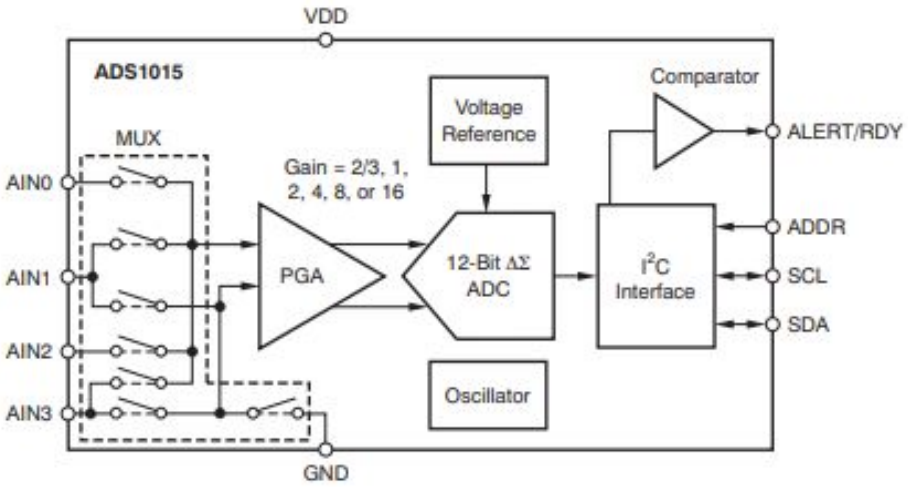

Figure 3: ADS1015 Block Diagram

The mbed LPC1768 microcontroller was powered by connecting it to the computer through a USB. The MCP2551 CAN bus was powered with a 5V DC power source. Our team connected the transmission and read ports of the MCP2551 with pins 30 and 29 on LPC1768 that are dedicated for CAN configuration. As seen from Figure 4 below, two 120Ω resistors were placed in parallel to implement the CANH and CANL nodes. These were the nodes that the CAN BUSs interacted to read and write the messages. For test purpose, our team used two CAN BUSs. As shown in Figure 5, pin 9 and 10 of the LPC1768 were used for CAN write, and pin 29 and 30 were used for CAN read. Once we successfully designed the ADC, one MCP2551 was removed and the other one was kept to read messages from the keypad.

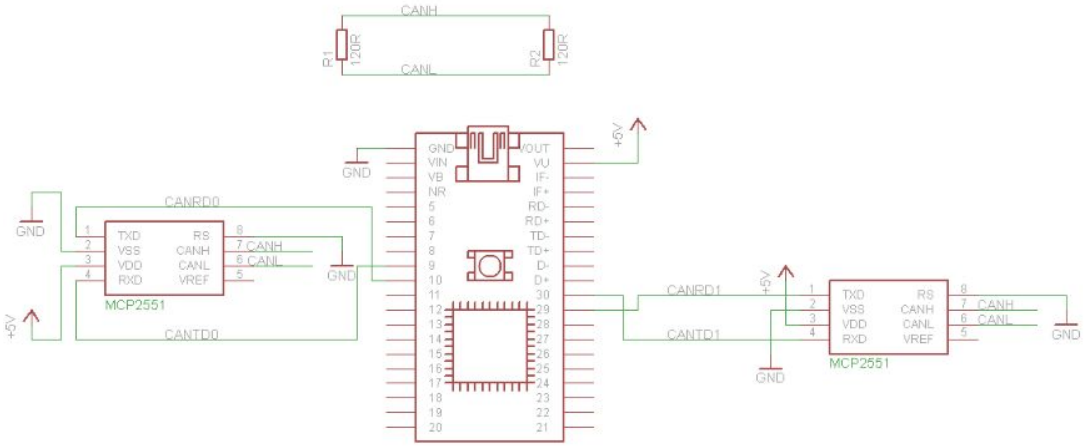

Figure 4: Connection of two CAN buses with one microcontroller

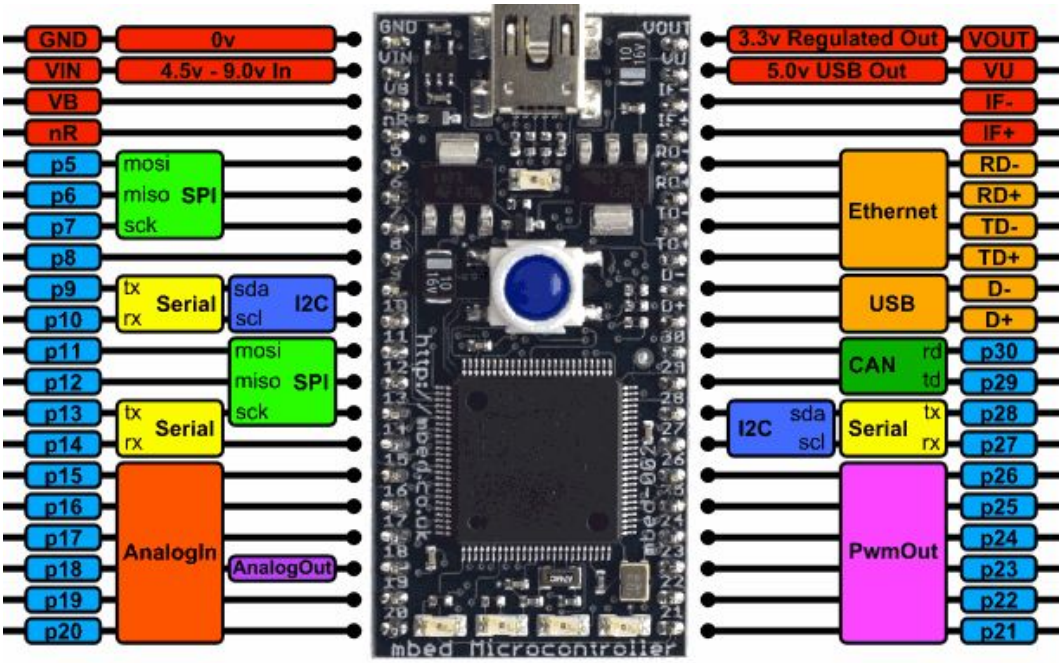

Figure 5: The mbed LPC1768 pinout

Mbed Code

Figure 6 shows the important lines of the code for pin configuration. First, we included two headers files, "mbed.h" and "Adafruit_ADS1015.h", in order to enable the mbed LPC1768 microcontroller and the ADS1015. No changes were made to the two header files because they were given as default by the mbed and are solely used for compatibility purpose. Pin 27 and 28 of the microcontroller are configured for I2C communication, and pins 29 and 30 are configured for CAN bus communication. A local file is declared under the name "local" to store the sampled data as a text file.

```
1  #include "mbed.h"
2  #include "Adafruit_ADS1015.h"
3
4  Serial pc(USBTX, USBRX);
5  I2C i2c(p28, p27);
6  Adafruit_ADS1015 ads(&i2c);
7
8  CAN can1(p30,p29);
9  LocalFileSystem local("local");
10
```

Figure 6: mbed code - Header and Configuration

As seen from Figure 7, The main function is where the input from the black box is read and converted into a digital output. A CAN message variable creates an empty set where the received message can be stored. The keypad team and we agreed to use 8 on the keypad to start and 9 to stop the ADC conversion. Within the infinite while loop, the first if statement detects all messages it receives from the keypad through the CAN bus but does not yet trigger the ADC. The second if statement starts the ADC if the message from the keypad is 8. It prints out a print statement to indicate the start of sampling. Once the sampling starts, the analog input at AIN0 is multiplied with the conversion factor. The inner infinite while loop within this if statement compares the data with 2047 which is the overflow and sets the data (adc0) equal to zero. Every time the code compares, it prints out a statement with the converted output. The wait time is set to 1 ms. A user can stop the conversion by pressing 9 on the keypad to breaking out of the while loop.

```
int main() {
    CANMessage(msg);
    while(1) {
        int message;

        if(can1.read(msg)) {
            pc.printf("CAN Message Received");
            message = msg.data[0];

            if(message == 8) {
                pc.printf("Note: ADC Range: +/- 6.144V (1 bit = 3mV/ADS1015)\n\r");
                pc.printf("Sampling analog waveform...\n\r");

                while(1) {
                    pc.baud(9600);
                    ads.setGain(GAIN_TWOTHIRDS);

                    uint16_t adc0;
                    float conversion_factor = 0.003;

                    FILE * fp = fopen("/local/output.txt","a");
                    adc0 = ads.readADC_SingleEnded(0);

                    if (adc0 > 2047) {
                       adc0 = 0;
                       }

                    pc.printf("AN0: %f\r\n", adc0*conversion_factor);
                    fprintf(fp, "%f\r\n", adc0*conversion_factor);
                    fclose(fp);

                    if(can1.read(msg)) {
                        message = msg.data[0];
                            if (message == 9) {
                                break;
                                }
                    wait_ms(1);
                    }
                }
            }
        }
    }
}

```

Figure 7: Mbed code - Main Function

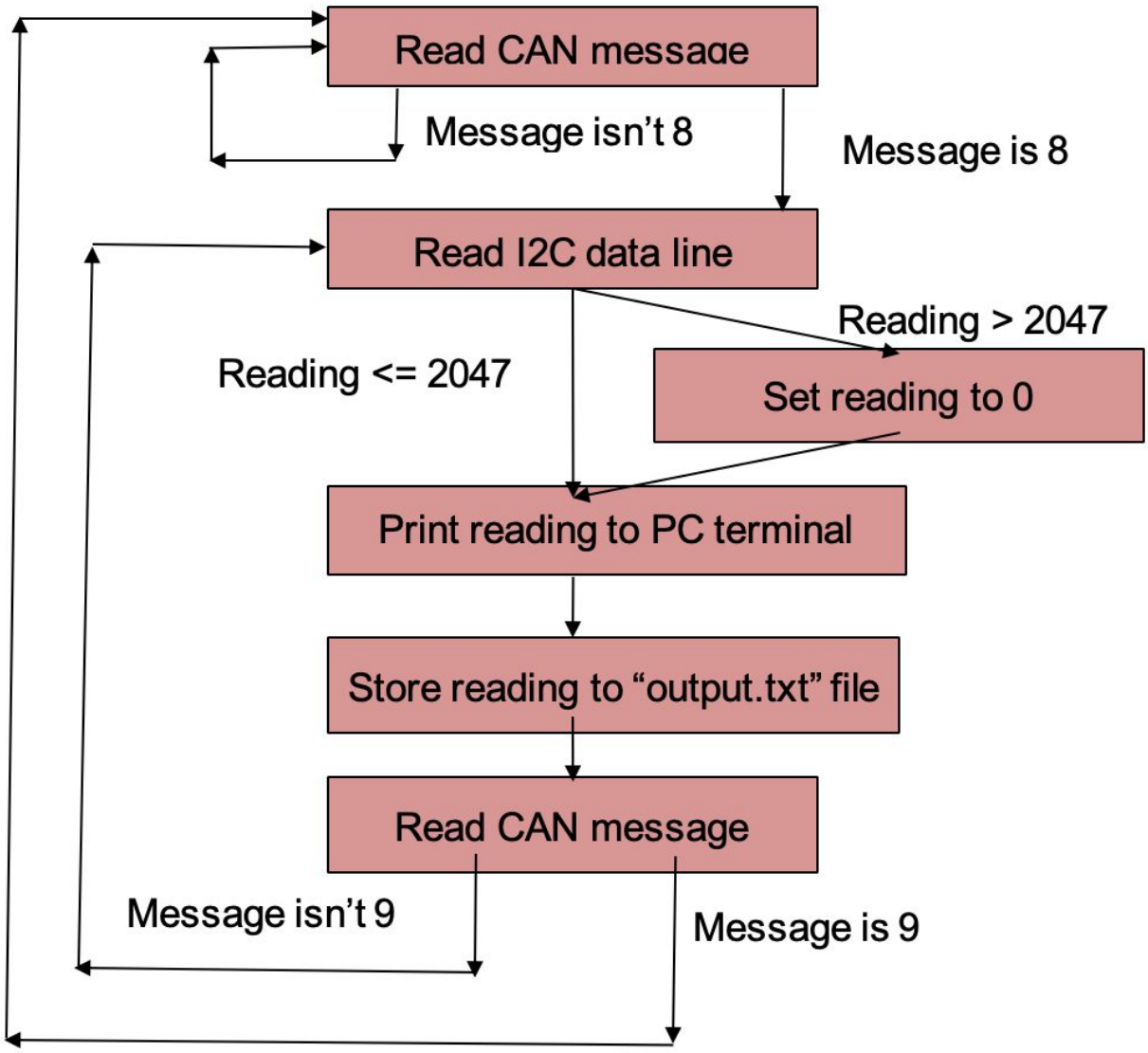

Figure 8: Flow Chart of the Main Function

Verification Testing

As system requirements, the Analog-to-Digital Converter (ADC) subsystem shall:

a. Sample the output of the black box at 100 Hz.
b. Output the samples to a PC terminal over USB or save locally on the mbed.
c. Receive commands from the Keypad subsystem to stop/start sampling.

a. Sample the output of the black box at 100 Hz.

The ADC subsystem took an analog input and converted to a digital output. Several analog inputs were tested before integrating with other subsystems. As an example, Figure 9 shows the output after performing ADC conversion. The input was a 10 Hz sine wave with an amplitude of 4 Vpp. Also, the sampled output of the black box is shown in Figure 11.

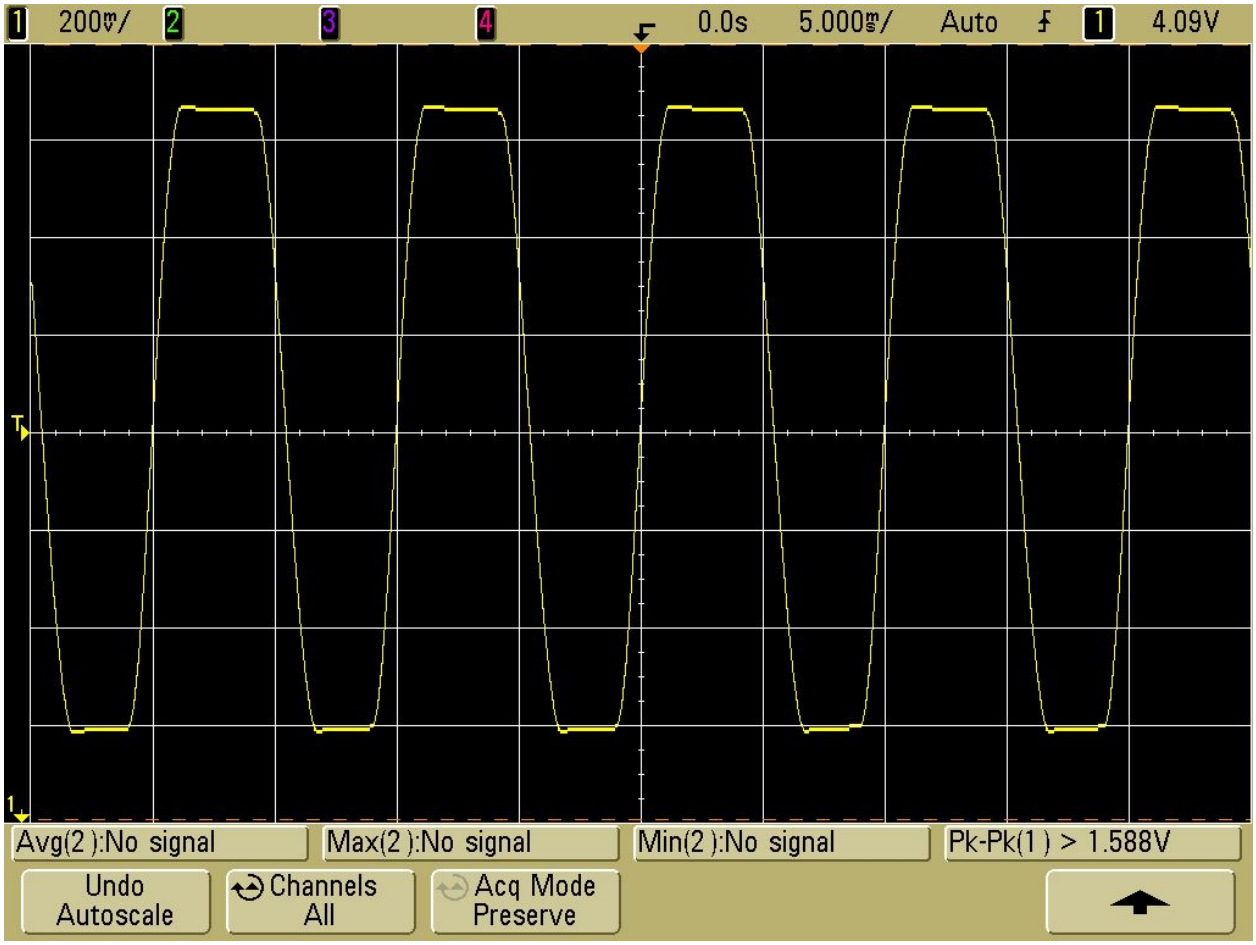

Figure 9: Output from ADC

b. Output the samples to a PC terminal over USB or save locally on the mbed.

The mbed LPC1768 microcontroller was connected to the computer using a USB cable. The output was sampled to a PC terminal over this USB, and those voltage values were displayed on PuTTY every 1 ms, as coded on the mbed. By analyzing the values, we were able to see the steps occurring and verified it using an oscilloscope. Then, we concluded that the ADC subsystem functions as it should be.

c. Receive commands from the Keypad subsystem to stop/start sampling.

We were able to start and stop sampling by using two CAN buses that each writes and reads messages. As shown in Figure 4, pin 9 and 10 of the LPC1768 were used for CAN write, and pin 29 and 30 were used for CAN read. Extra lines of codes were written to have CAN write functions. We were able to successfully stop and start sampling by using the CAN bus system in this way. However, when we put it together with the keypad, we were not able to receive commands from the keypad, which, therefore, did not trigger the ADC conversion. Since other subsystems were not able to communicate as well, we assumed that there a problem with the keypad.

Validation Testing

The goal of this lab was to diagnose the black box which has an unknown characterization. Once all subsystems integrated together, 3 V DC signal was inputted to the DAC and was converted to an analog signal. This analog signal went through the digipot and the black box, and then converted back to a digital signal by the ADC. The output displayed on the oscilloscope and the PuTTY was used to diagnose the black box. We were able to evaluate the usefulness of the ADC subsystem by validating its proper functionality of determining the equation of the black box.

Result from the Integrated System

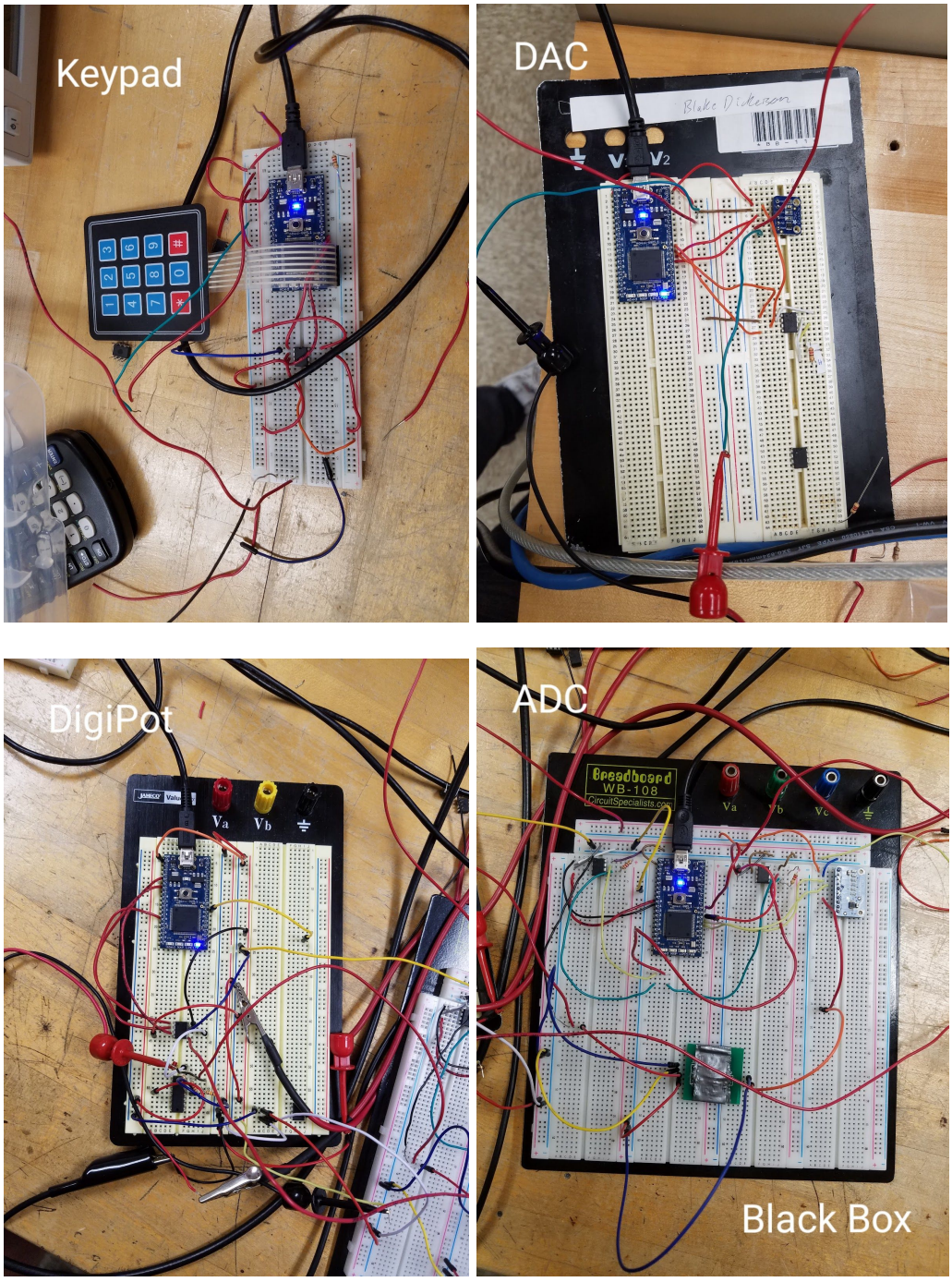

Figure 10: Complete integration of subsystems

Once each subsystem was built, all were gathered to integrate to figure out the black box. The connection of the subsystems is as shown as in Figure 1, and the actual integration is shown in Figure 10. 3 V DC was supplied as an input to DAC. After going through all of the subsystems, we generated the output on the oscilloscope. By analyzing the data on the oscilloscope and the PuTTY, we were able to find out that the equation of the black box was $-0.1x+1$.

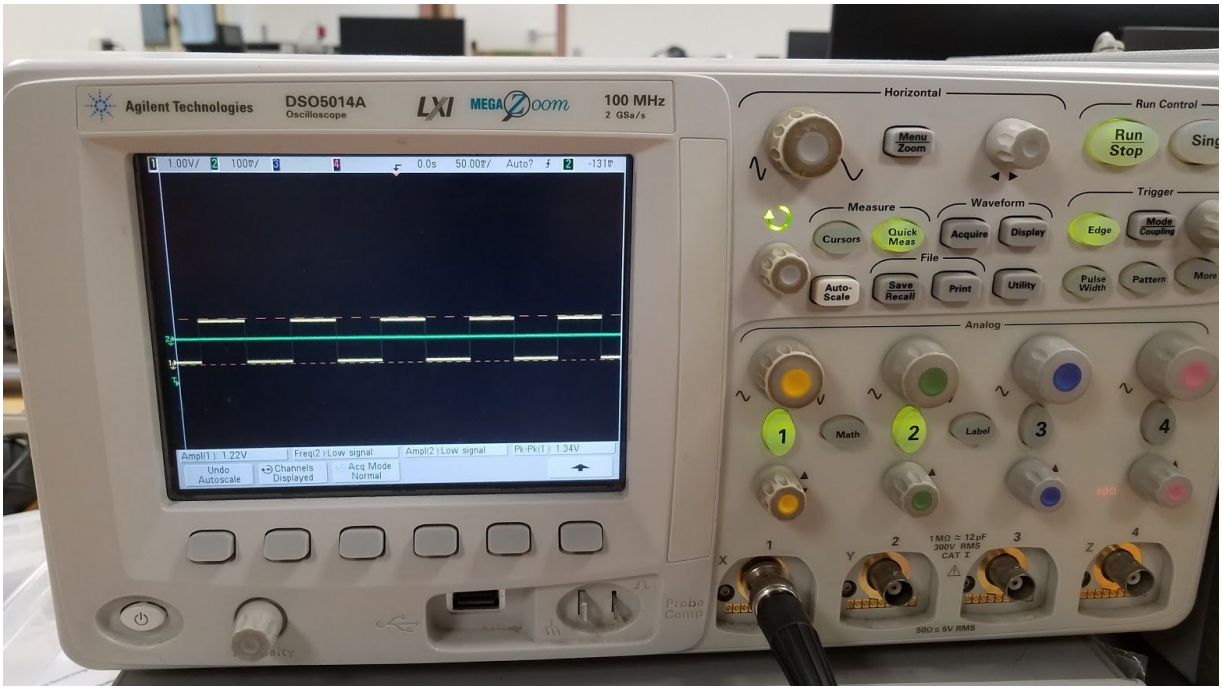

Figure 11: The output on an oscilloscope

Discussion

The CAN bus is an effective method of communication between devices. However, Serial peripheral interface (SPI) may have been an effective alternative of the CAN bus. Nevertheless, the downside of SPI is that it is a four wire, which requires double the number of wires that the CAN bus needs.

There was one simple modification to the design during integration. Since the keypad was not able to communicate with subsystems, we decided to integrate the subsystems without the keypad because of the time constraint. In order to do so, we hardcoded the lines where it was supposed to receive the message from the keypad. Instead of waiting for the message from the keypad to start the ADC, we modified the code in a way that the ADC would keep running. This

modification let the input signal to be fed through each subsystem including the black box. Ultimately, we were able to get the output and diagnose the black box without the keypad.

Our team felt that there could have been more communication between the other teams. Rather than trying to integrate as a whole on the last day, we could have better understood each other's subsystem and have integrated more successfully. In this way, we could have tried debugging the failure of communication using the CAN bus between the keypad and subsystems.

Value Statement

The Wheelers will face challenges that will benefit humanity. Our team worked effectively to figure out the unknown transfer function of the black box. Our team maintained high ethical standards as we engaged in developing the system, and the process has helped each of our members to enhance their knowledge and understanding on electrical systems.

Conclusion

The subsystems were integrated together and found out the characterization of the black box. All of the subsystems worked individually but there were issues that came up when integrating with others. The keypad was not able to communicate with any subsystem. Our team was able to understand the difficulty of combining different subsystems for integration. The I2C protocol was a good device for microcontrollers and DSPs as it was quick to respond and did not have any problem. The mbed module was somewhat volatile, and there were many compiling issues. It seemed to have limited libraries compared to other programs. Unless the errors were the syntax errors, the mbed compiler did not direct us where the errors occurred, and even when it did, the solution could often be vastly different and unrelated to the error it first presented.

Appendix A

Table 1: Bill of Materials

Part No.	Part Name	Price/Unit ($)	Quantity	Price ($)
1	CAN bus	1.12	1	1.12
2	mbed LP1768 Microcontroller	54.95	1	54.95
3	120Ω Resistor	0.05	2	0.10
4	ADS1015	9.95	1	9.95
			Total Price	66.12

Appendix B

Table 2: Gantt Chart

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
Team Inventory							
Subsystem Block Diagram							
Mbed Code Flow Chart							
Testing Plan							
Build							
Integration and Testing							
Black Box Diagnostics and Demo							
CDR							