The Pennsylvania State University

EE300W Section 3 -- Spring 2018

Labortatory 3 Final Report: Embedded Systems
Keypad Subsystem

The WEE Team

Junghsien Wei, Eric Pauley, Eric Mosier

Table of Contents

Abstract

Team WEE designed and built a keypad subsystem, which consists of a CAN bus (MCP2551), Adafruit membrane keyboard, and mbed LPC1768 development board. The keypad subsystem sends commands to the DAC (Digital to Analog), DigiPot (Digital Potentiometer), and ADC (Analog to Digital) subsystems. It is the main control in this embedded system, allowing the user to select a waveform, choose waveform amplitude and start/stop sampling. These actions allow for characterization of the black box, with data being sent via USB to a computer for further analysis.

Introduction

An embedded system has a dedicated function within a larger system. Tasks are made more manageable by being broken down into these embedded subsystems. We integrated multiple subsystems together in order to determine the function of a “black box”. There are four subsystems integrated together, which are the keypad subsystem used to send command, the DAC subsystem used to convert the digital waveforms, the DigiPot subsystem used to output the voltage into the “black box”, and the ADC subsystem used to sample the output of the “black box”. The flow from the DAC to the ADC subsystem is shown in Figure 1.Team WEE is responsible for controlling the system. Our keypad subsystem sends commands to the DAC subsystem to start/stop and select the waveform, the DigiPot subsystem to select waveform amplitude, and the ADC subsystem to start/stop sampling. Moreover, our subsystem receives user input from the keypad to send these commands.

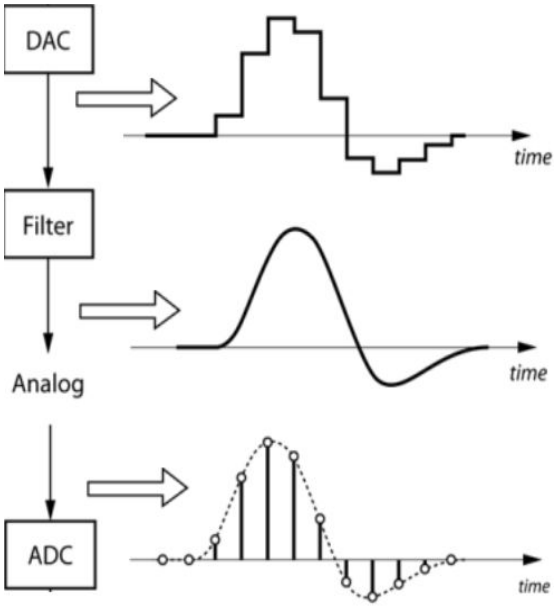


Figure 1: Progress between DAC to ADC

Rationale

Theory of Operation

The keypad subsystem serves as the control interface for the embedded system. It allows the user to set DAC waveform functions and amplitude, DigiPot gain, and enable or disable ADC data sampling. The DigiPot is controlled with buttons 1 – 5; each button corresponds to the desired amplitude, e.g. pressing button 3 will set the DigiPot voltage to 3V. The DAC is controlled with buttons 7, 8, and 0; button 7 toggles between the sine and square waveform, button 8 toggles between 1 Hz and 10 Hz frequencies, and button 0 toggles the output on and off. The ADC data sampling is toggled with the # button.

Theory of Interfaces

The keypad interfaces with each of the subsystems through the CAN bus. Connecting all four subsystems to the same terminal running PuTTY allows a single operator to control the entire system with the keypad while monitoring the other subsystems' responses to keypad input. Since the system uses the CAN bus, the PuTTY interface is only necessary for debugging; the system can operate only with power to the microcontrollers, with the output of the ADC being stored on a text file in the subsystem's memory.

Implementation

Explanation of code and circuitry.

Our goal is to generate and input waveforms into a black box by designing, constructing, testing, and applying a diagnostics suite to characterize a "black box". The diagnostics system block diagram is shown in Figure 2. The keypad subsystems should allow users to control other subsystems. Our keypad subsystem used an mbed, CAN bus, and keypad for hardware, and read-button, message-read, and a while loop with if statements for structure of the software. The functional decomposition of the subsystem is shown in Figure 3. Figure 4 shows the code flow, which consists of polling button states and sending corresponding commands over the CAN bus..

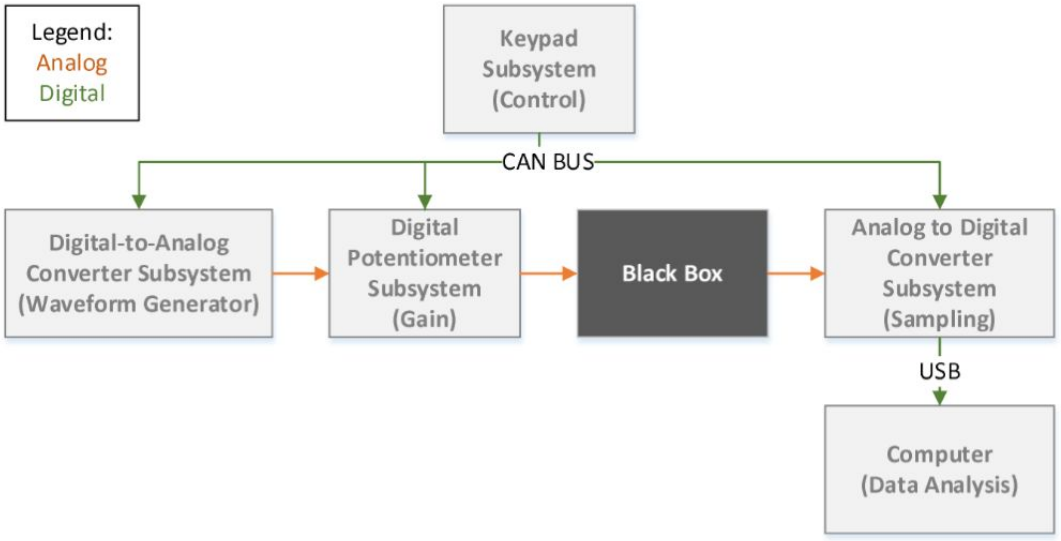


Figure 2: Diagnostics System Block Diagram

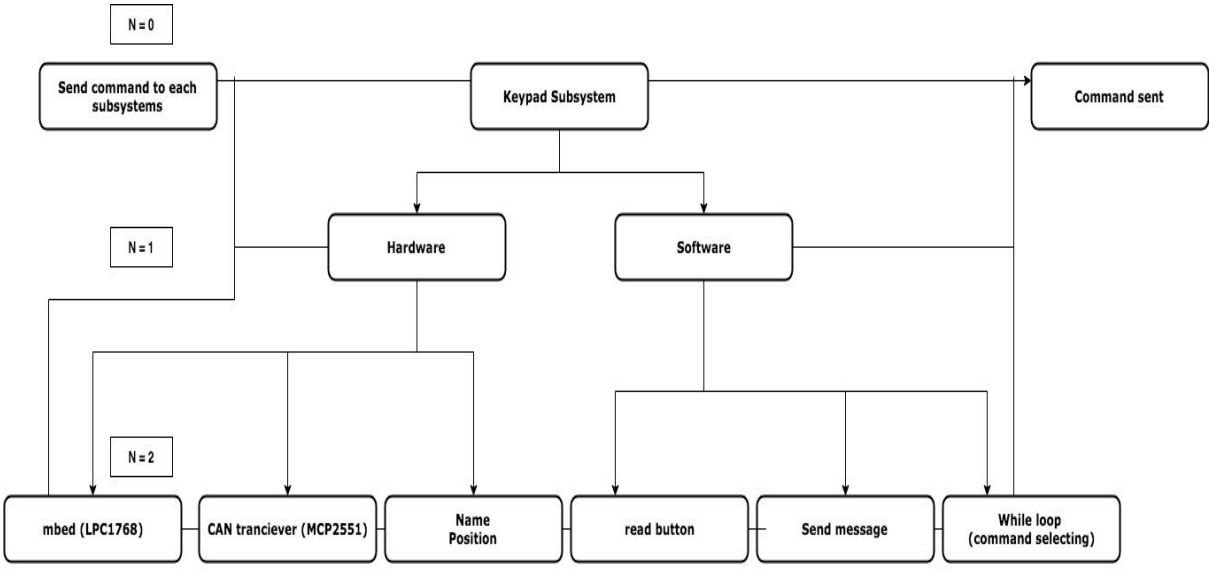


Figure 3: Keypad Block Diagram

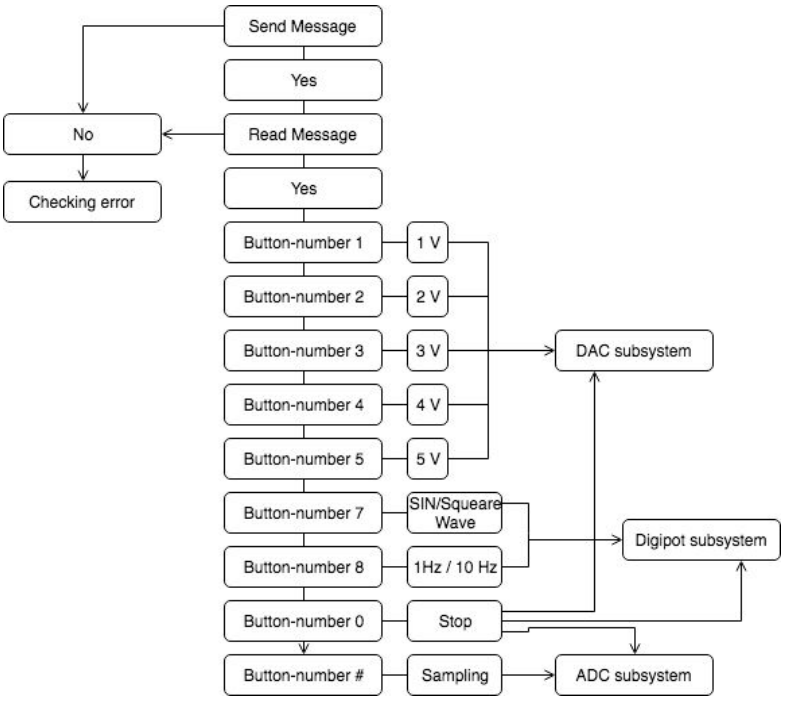


Figure 4: Keypad flow chart

Our program defines device pins for the keyboard and CAN bus. (See Appendix C) Pins 21-27 are used by the keypad. Pins 9, 10, 29, and 30 map to the two CAN bus trancievers.

Before we decided which number was going to control the corresponding instruction, we needed to understand how did the keypad works. (See Figure 6) Each number button is maps to a row and column pin. See Figure 7 for button acquisition code.

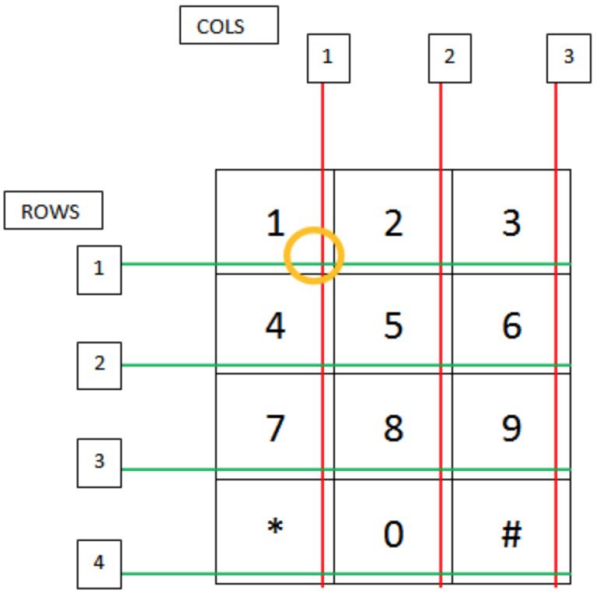


Figure 6: Keypad schematic [1]

The main code checks button state, and sends messages based on presses. (See Appendix C) Each button maps to 0-2 subsystems, sending appropriate commands to implement each button's functionality.

The keyboard subsystem must be verifiable on its own, so a second CAN bus is added to allow receipt of sent subsystem commands. See Figures 10 and 11 for verification CAN bus and final schematic for integration. After finishing our code, we had to build our subsystem circuit to verify our code. Figure 12 shows our subsystem after integration.

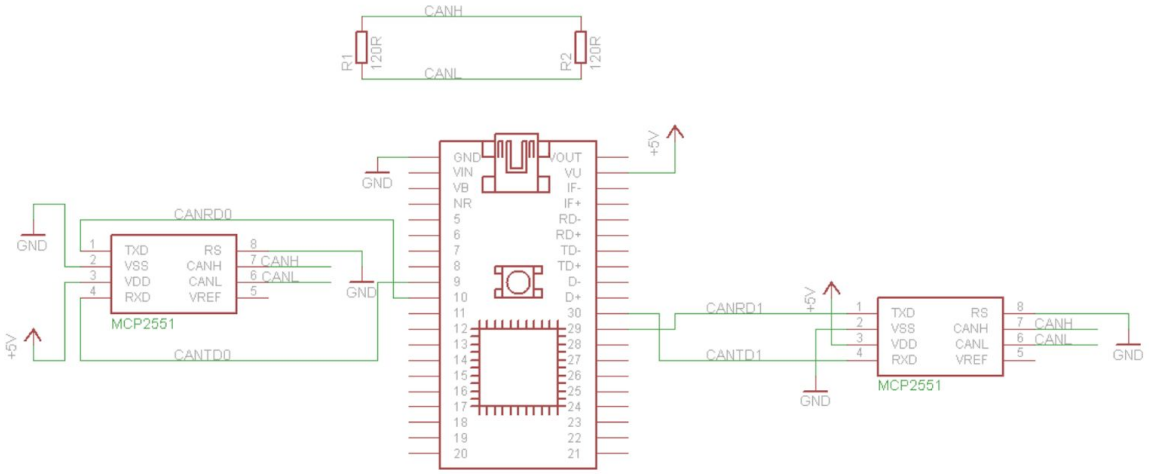


Figure 10: CAN bus schematic. [2]

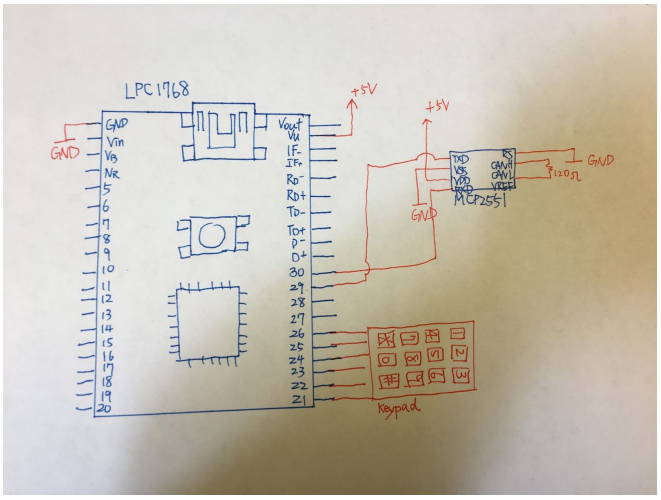


Figure 11: Keypad subsystem schematic

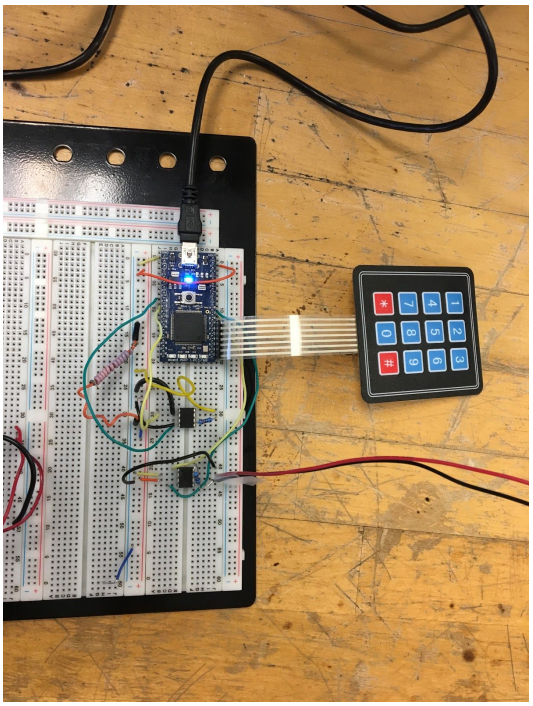


Figure 11: Keypad final integrated circuit.

Verification Testing

We tested functionality of our keyboard subsystem by simulating a CAN receiver. The keyboard waits for key presses, logs them, and sends/receives a message on the shared CAN bus to verify functionality. This test confirmed that our subsystem met all requirements, including receipt of input from user, transmission over a shared serial medium, and console output.

Figure 12 shows the completed and integrated system. Verification of the integrated system was performed by sending commands using the keypad subsystem and reading waveform data using an oscilloscope, as well as verifying that the ADC subsystem correctly saved data to internal storage. Figure 13 shows sample outputs from each subsystem.

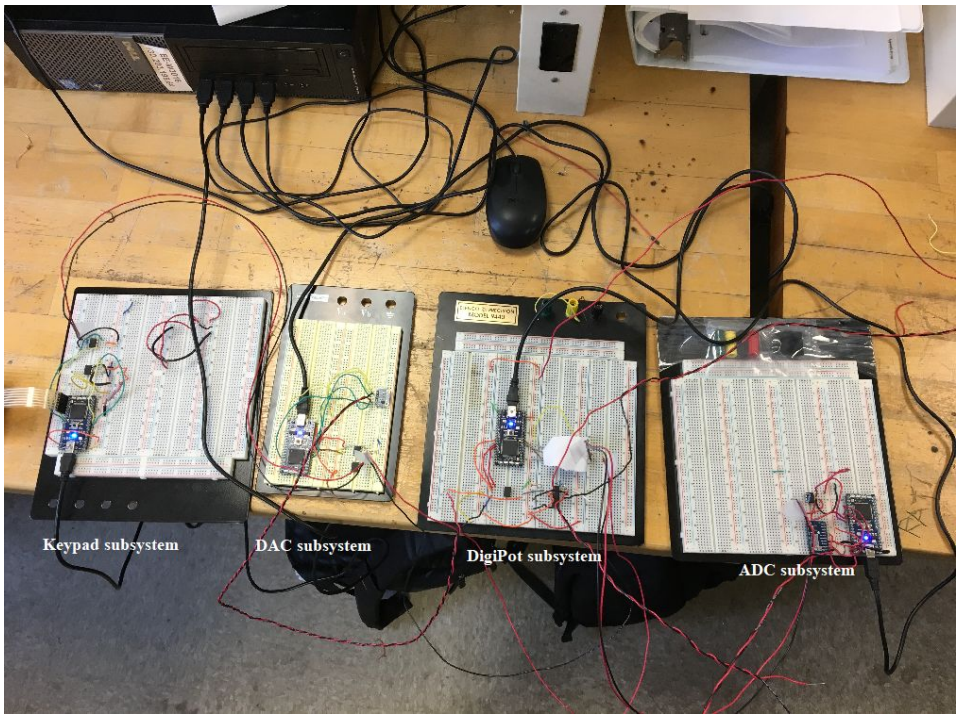


Figure 12: Completed Integrated Circuit

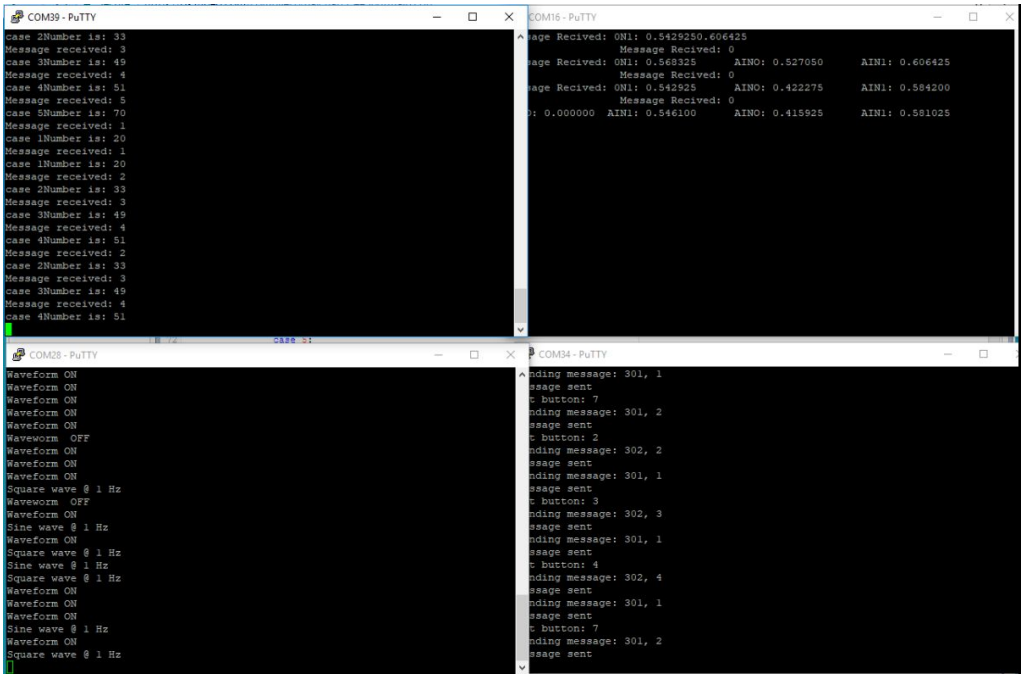


Figure 13: PuTTY output results
(Top Left: DigiPot, Top Right: ADC, Botom Left: DAC, Bottom Right: Keypad)

Validation Testing

The requirements fulfilled by the keyboard subsystem are critical to the analysis of the Black Box. Multiple signals must be passed through the box in order to confirm its

functionality, and the keyboard allows these multiple signals to be selected. Its functionality is essential to the overall mission of determining the functionality of the Black Box.

Integrated System results

The integrated system functioned properly, except for the DigiPot subsystem. No functional DigiPot chips were available in the lab at the time of testing, so we were only able to use a single amplitude signal for testing. See Figure 14 for signals sent to the Black Box.

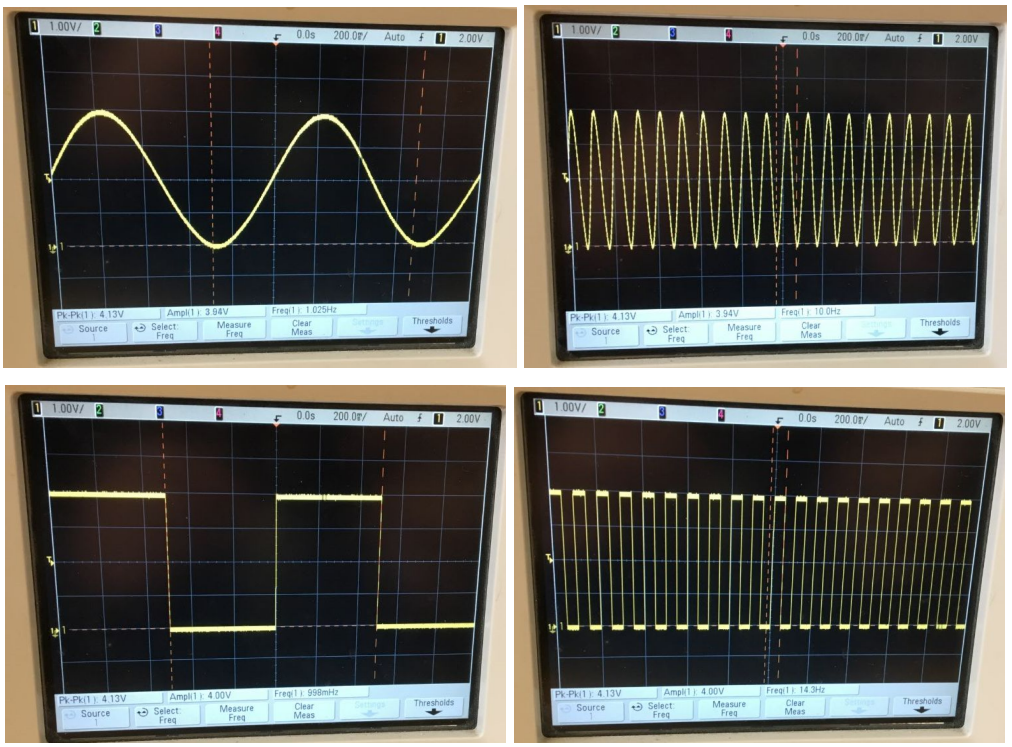


Figure 14. 1/10Hz Sine/Square wave outputs to Black Box

Determination of the black box transfer function was accomplished using only a single 0-4V square wave. We were told that the transfer function was linear, and so can be modeled by $H(j\omega)$. Using a square wave, which is a sequence of unit step functions, we determined that the function impulse response was $H(\delta(t)) = c\delta(t)$, that is to say the function is memoryless. Its transfer function can therefore be modeled by $F(x) = ax + b$, where x is the input voltage. We took our two data points from the square wave and solved for a and b:

$$F(0V) = 1V = a * 0 + b,\ F(4V) = 0.6V = a * 4 + b$$

$$a = -0.1, b = 1V$$

By this method we determined the transfer function of the black box to be $F(x) = 1V - 0.1x$.

Discussion

Approach to Module, Network Interface, and Debugging

We created the block diagram and flowchart first in order to define the requirements of each module. We mapped subsystem functions to specific number buttons. Keypad specifications were conflicting and testing was required to determine device pinouts.

The keypad subsystem was based on “case code”, which certain number had corresponding output. After programing our codes inside the mbed, we integrated LPC1768 mbed with two CAN transceiver MCP2551. One of the CAN transceivers was for sending message, and the other one was for reading message. We used PuTTY to merge our subsystem to debug whether the message was sent.

We learned that using CAN bus was effective to debug the system. The CAN bus allowed microcontrollers and keypad to communicate with each other without a host computer. It was very efficient to test the system for small network, and it was flexible to integrate, which we only needed two wires for CAN bus communication. There are other alternatives, such as SPI (Serial Peripheral Interface), but it handled short distance communication. In addition, SPI didn’t have error checking protocol and only supported one master device, so that was why CAN bus was our first choice for testing.

Modifications to Design During Integration

Our subsystem required no modifications during integration, since its outputs were meticulously verified beforehand. However, other subsystems did not properly receive CAN bus messages and code needed to be modified to correctly support this.

Improvements to Lab

Each subsystem shares significant functionality, such as the CAN bus. Because each subsystem worked independently before integration, each CAN bus implementation was different and so suffered different bugs. In the future these shared systems should be developed collaboratively, so that each subsystem has a standardized implementation. Much of our debugging and modification during integration would have been prevented by this.

Value Statement

Embedded systems allow specific functionality to be efficiently implemented. Our subsystem serves as a working example of this, while also accomplishing a task of analyzing an obscured circuit. By utilizing modern Systems Engineering principles, we’ve met requirements which would cost orders of magnitude more to achieve using conventional lab

equipment. This device, as well as the methodologies behind its implementation, can be applied to more powerful and complex systems.

Conclusion

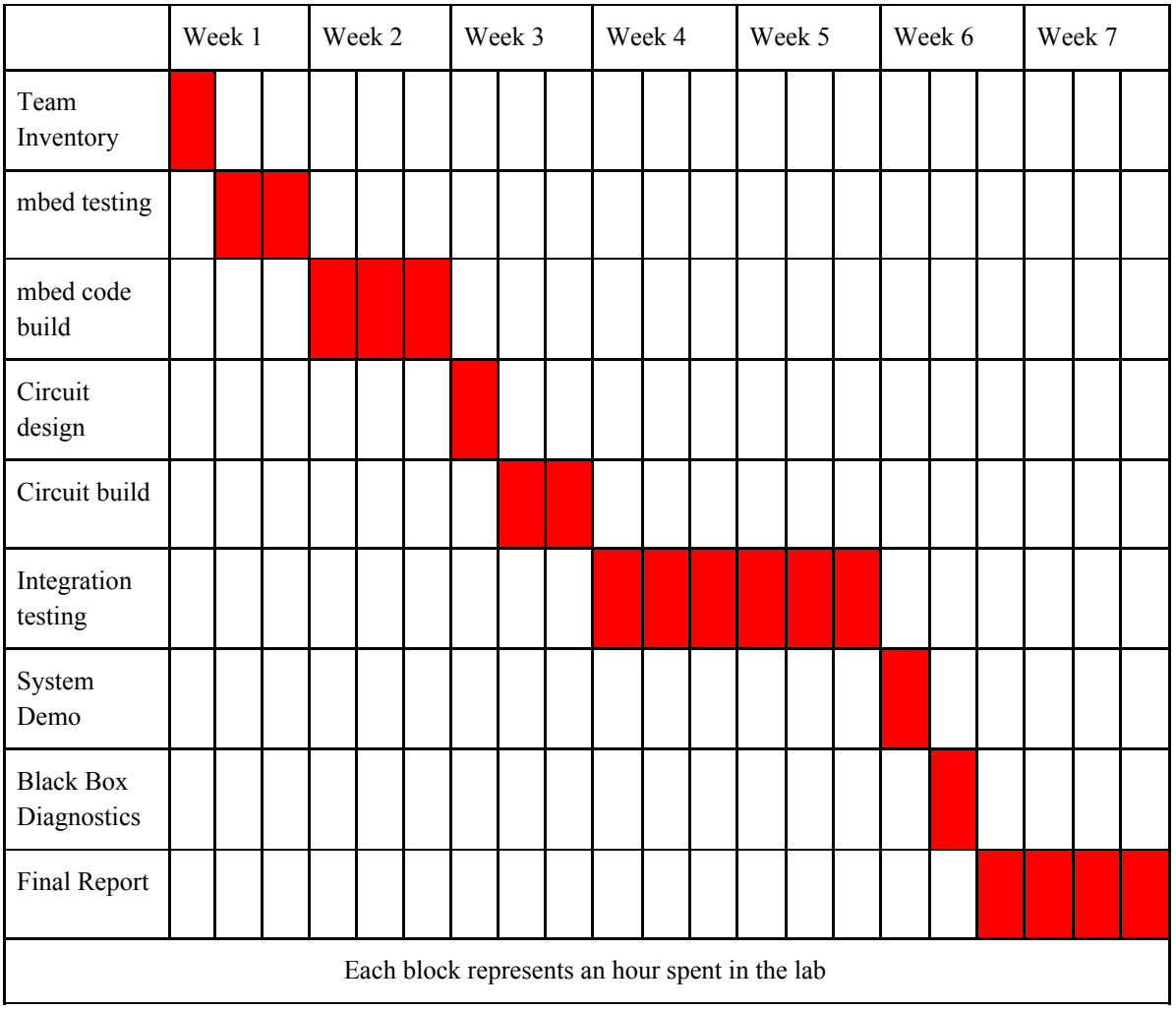
We produced a functioning keypad subsystem and successfully integrated it into a complete circuit analysis suite capable of analyzing the block box. We performed verification and validation of our final solution and requirements. We used our completed system to analyze the block box and determine its transfer function, showing that our integrated system achieves our overall mission.

Appendices

Appendix A: Financial Statement (Bill of Materials)

Item	Part Number	Quantity	Price/Unit	Price	Source
mbed	LPC1768	1	$64.95	$64.95	https://www.adafruit.com/product/834
CAN tranciever	MCP2551	1	$1.23	$1.23	https://www.mouser.com/productdetail/?qs=8g3RLJaa2Uis1lC9gEMPXA%3D%3D
Keypad	Membrane 3x4 Matrix Keypad	1	$3.95	$3.95	https://www.adafruit.com/product/419
120 Ohm resistror	690646	1	$0.01	$0.01	EE Stockroom
Breadboard with wires	FBA_RRBB JW75PCSP	1	$5.95	$5.95	https://www.adafruit.com/product/239
Labor	Junghsien Wei, Eric Pauley, Eric Mosier	3	$16/hour	$1008	7 weeks, 3hr per week https://www.glassdoor.com/Salaries/research-assistant-electrical-engineering-salary-SRCH_KO0,41.htm
			Total$	1089.51	

Appendix B: Gantt Chart



Appendix C: Keyboard Subsystem Code

```
#include "mbed.h"
Ticker flipper;
DigitalIn row1(p27, PullUp);
DigitalIn row2(p26, PullUp);
DigitalIn row3(p25, PullUp);
DigitalIn row4(p24, PullUp);
DigitalOut col1(p23);
DigitalOut col2(p22);
DigitalOut col3(p21);
CAN can1(p30, p29);
CAN can2(p9, p10);

DigitalIn rows[4] = {row1, row2, row3, row4};
DigitalOut cols[3] = {col1, col2, col3};
char buttons[4][3] =
{{1,2,3},{4,5,6},{7,8,9},{10,0,11}};
char read_button(){
   char button = 255;
   for(int col = 0;col < 3; col++){
       cols[col].write(0);
       for(int row = 0;row < 4;row++){
           if(!rows[row].read())
               button = buttons[row][col];
       }
       cols[col].write(1);
   }
   return button;
}
void message(int id, char msg){
   printf("Sending message: %d, %d\n", id, msg);
   if(can1.write(CANMessage(id, &msg, 1))) {
       printf("Message sent\n");
   }else{
       printf("Failed message send\n");
   }
}
int main() {
   while(1) {
       char button = 255;
       while(button == 255){
           wait(0.01);
           button = read_button();
       }
       printf("Got button: %d\n", button);
       switch(button){
       case 1:
       case 2:
       case 3:
       case 4:
       case 5:
           message(302, button);
           message(301, 1);
           break;
       case 0:
           message(301, 0);
           break;
       case 7:
           message(301, 2);
           break;
       case 8:
           message(301, 3);
           break;
       case 11:
           message(303, 0);
           break;
       default:
           printf("Invalid button\n");
       }
       while(read_button() != 255);
       wait(0.1);
       CANMessage msg;
       while(can2.read(msg)) {
           printf("Message received: %d\n",
msg.data[0]);
       }
   }
```

References

1.Youngblood, T. (2015, June 15). Use a Keypad with Your Arduino. Retrieved from https://www.allaboutcircuits.com/projects/use-a-keypad-with-your-arduino/

2. ArmMBED. (n.d.). CAN-Handbook. Retrieved from https://os.mbed.com/handbook/CAN