Foot Keyboard

Team 42 - ECE445 - Spring 2020 - Final Report
Neva Manalil, Nick Halteman, Aditi Panwar
TA: Johan Mufuta

8 May 2020

Abstract

People suffering from arm or hand loss face greater difficulty when using a computer as user input devices are designed to be used with two hands. At the start of the Spring 2020 semester, Team 1 proposed a foot controlled prosthetic device that could type on a standard keyboard. This design had a few limitations such as the hand only being able to press a couple of keys and requiring the user to have a hand to press the other keys, making it less accessible to those who did not have either hand. To solve these issues, the solution we propose is a foot controlled keyboard that maps all standard keyboard functions and mouse input to a group selection based input system controlled by moving the user's feet. The key improvements of this design is that it is accessible to those who suffer from the loss of both hands, includes all keyboard and mouse functions, and is a single part solution simplifying setup and increasing portability.

Table of Contents

1. Second Project Motivation

In this chapter, we summarize the work done to refine the design process of our new project. The motivation for our project was to design a system to enable those who suffer from limb loss to be able to use a computer easily, efficiently, and on their own. And through this we were able to design an alternative input system to the standard keyboard and mouse which does not require the use of hands.

1.1 Updated Problem Statement

There are nearly two million people in the United States currently living with limb loss and around 185,000 more amputations occur each year [1]. In order to help those facing limb loss, there are prosthetics amputees can purchase. However, even if someone has a prosthetic, it does not guarantee them the ability to type on a keyboard as many prostheses use pre-programmed gestures and are not very dexterous. Typing requires many different precise gestures which realistically cannot be programmed to the arm. The lack of accessibility to amputee friendly computer user input devices makes it difficult for amputees to use a computer. The Universal Declaration of Human Rights states in Article 19 that everyone has the right, “to seek, receive and impart information and ideas through any media and regardless of frontiers” and Section 32 expands that to include the Internet as a human right [2]. Being able to connect to the internet with a computer is crucial to our society today, but current prostheses greatly hinders the ability for amputees to use computers. Alternative solutions exist such as text to speech software. But, there are limitations as it may be prone to inaccuracies and the software does not allow for function keys such as ctrl or alt [3]. Without these function keys or mouse input, the user will not have the same control over the computer as a mouse and keyboard user would. We propose a device that will allow those with hand or arm loss to perform keyboard functions with their feet and have the autonomy to use a computer on their own.

1.2 Updated Solution

Our solution consists of a keyboard that is designed to be operated by only a person’s feet. Because people are typically much less dexterous with their toes, we will not be using a conventional array of buttons, each corresponding to a letter. Instead we have designed a direction based selection tool to take input by moving the user’s entire foot. The top of our keyboard will have two large grip pads, one for each foot, that can be pushed in any direction. The left foot will serve as a group selector, with each direction selecting a group of keys. The right foot will then select a single key from that group of letters to input. We also have a button

to toggle between uppercase and lowercase letters. Allowing for eight directions per foot plus the neutral center position and the case switch button, we can achieve up to 162 unique inputs. This enables us to easily map all 113 unique keystrokes a standard keyboard can perform. Additionally, the keyboard also has mouse functionality to ensure the computer can be completely controlled through only foot inputs. As memorizing the various inputs will take some time, we will also design a cheat sheet that can be placed on a desk so the user can quickly identify what foot positions they need for each key.

At the start of Spring 2020, Project Team 1 designed a prosthetic hand to type on a standard keyboard that was controlled by foot input. The design consisted of a hand prosthetic to be placed on the keyboard and a set of four buttons to be pressed by the user's toes. Both the original design and our new design take input from the user's feet to type keyboard characters; however, the original design maps the inputs to fingers on a prosthetic hand while the new design has a unique mapping for each keyboard character. Since the original design used a prosthetic hand, the user was expected to use their other hand to type the remaining characters. Our device also plugs directly into the computer and is recognized as a keyboard/mouse input device while the original project did not require any connections to the computer.

This solution is able to solve the original problem as it provides a means for those with hand loss to interact with a computer as effectively as using a keyboard and mouse. Since the entire standard keyboard and mouse inputs have a corresponding mapping, the user is able to perform any action they would otherwise have done with the traditional keyboard and mouse. Someone suffering from hand loss would be interested in this particular solution because it provides a complete solution to the problem at hand. Alternative solutions that currently are on the market as well as the original design proposed for this problem require some sort of additional device be used in order to perform all keyboard actions. For example with text to speech software a keyboard and mouse would be required to perform function keys that cannot be performed verbally, and the original solution still requires the other hand to press the remaining keys. Our project solution aims to be an accessible alternative to typing on a traditional, hand-operated keyboard.

1.3 Updated High Level Requirements

1. The keyboard must be able to replicate the 95 unique characters and 18 function keys found on a standard keyboard along with two mouse buttons and directional mouse movement using only the user's feet.

2. The polling rate of the keyboard must be at least 36Hz.

3. The keyboard microcontroller must be able to send the correct keystrokes and mouse inputs to the computer.

1.4 Updated Visual Aid

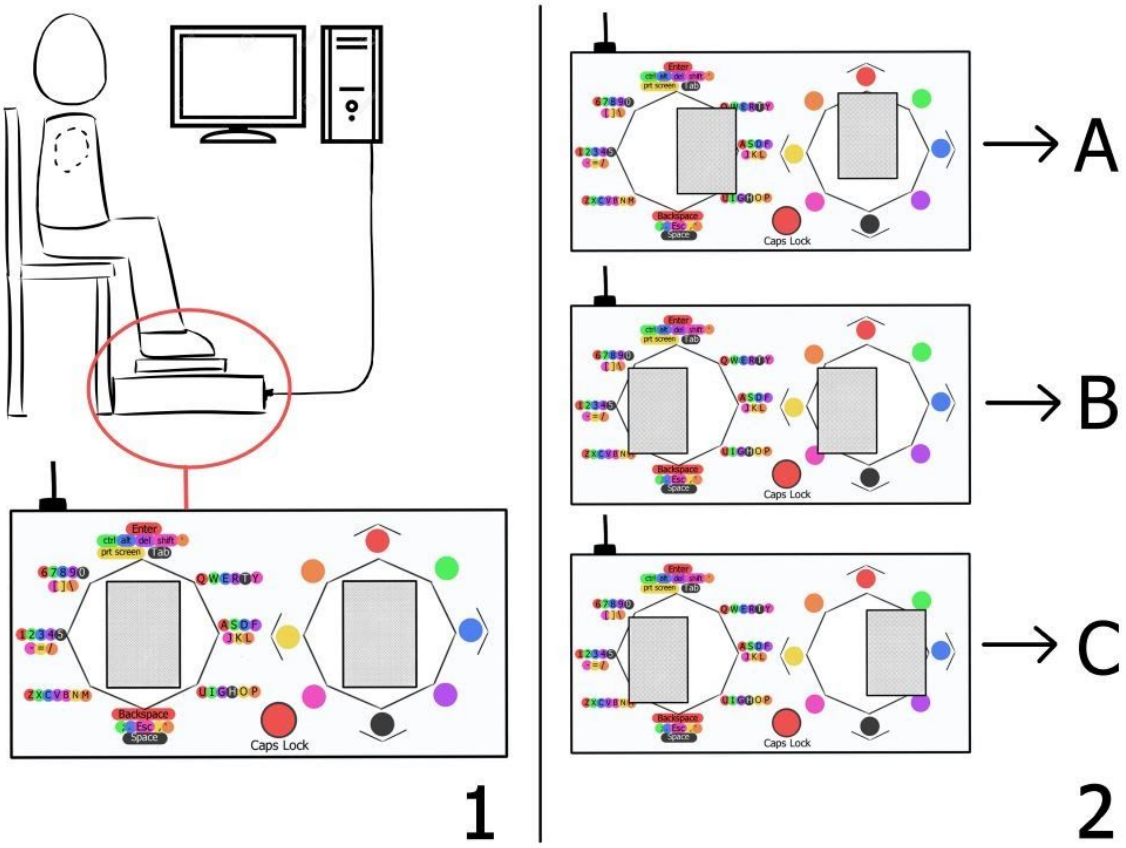


Fig 1 Visual Aid: 1. The foot keyboard connects to the computer through USB. The user must place their feet on the foot pads and slide the foot pad to the corresponding character and color groups to select a character or function key. To do keyboard selection the left foot must select a group first, otherwise the right foot may be used as a mouse input device. 2. Foot orientation to select characters A, B, and C. Only when both feet have made a selection will the character or function be executed.

	↑	↗	→	↘	↓	↙	←	↖	**Stomp**
↑	enter	ctrl/cmd	alt	del	tab	shift	prt screen	` (~)	home
↗	q (Q)	w (W)	e (E)	r (R)	t (T)	y (Y)			
→	a (A)	s (S)	d (D)	f (F)		j (J)	k (K)	l (L)	pg down
↘	u (U)	i (I)		g (G)	h (H)		o (O)	p (P)	
↓	backspace	; (:)	. (>)		space	esc	, (<)	‘ (“)	end
↙	z (Z)	x (X)	c (C)	v (V)		b (B)	n (N)	m (M)	
←	1 (!)	2 (@)	3 (#)	4 ($)	5 (%)	- (_)	= (+)	/ (?)	pg up
↖	6 (^)	7 (&)	8 (*)	9 (( )	0 ( ))	[({)	] (})	\ (|)	
Stomp	↑		→		↓		←		win
Neutral	Mouse Movement								Left/Right Click

Fig 2 Input Mapping: “Cheat sheet” that has the complete character mapping. The left column represents the left foot character group selection and the top row represents the right foot color selection. Characters in parentheses are the alternate characters when the “caps lock” button is enabled or when the previous character input was “shift.” If there is no character in parentheses, then the default character or function associated with that input will be selected. An input without anything mapped will not do anything. When there is no left foot input, the right pad controls the mouse. The right stomp button alone will function as a left click on a short press and a right click on a long press.

1.5 Updated Block Diagram

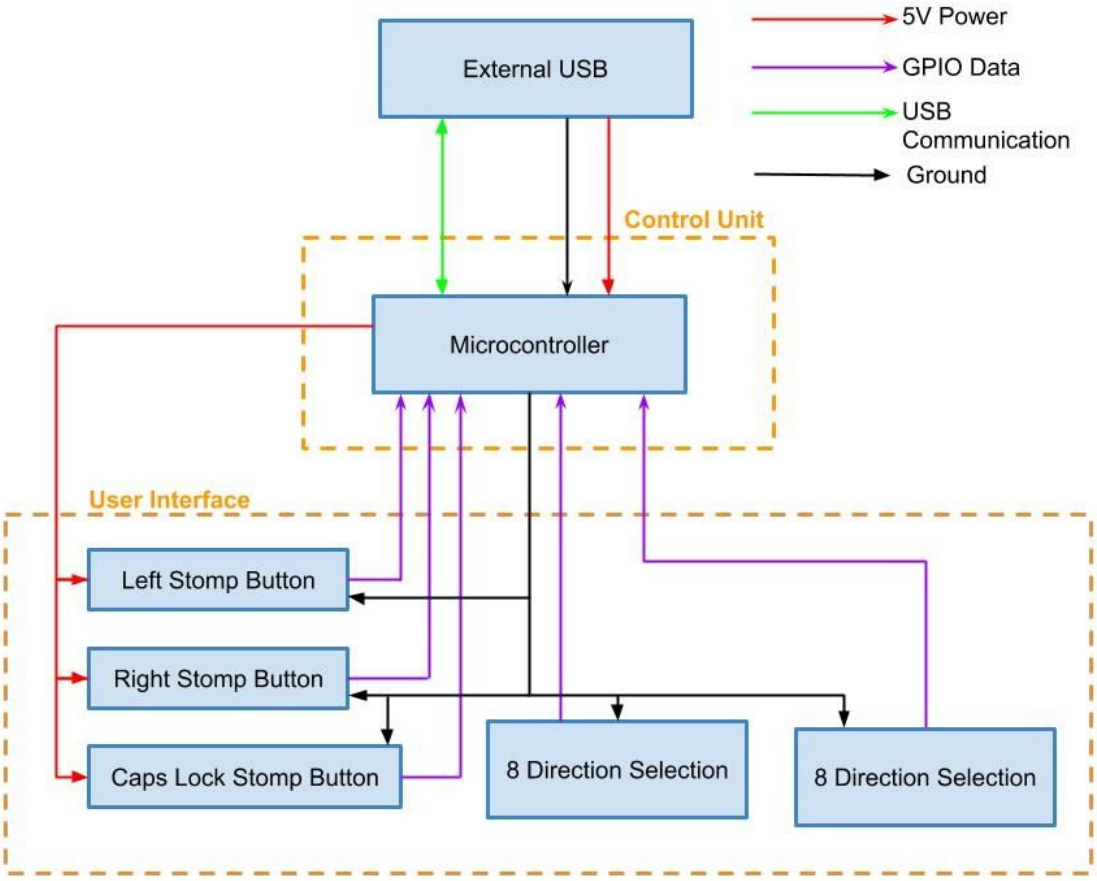


Fig 3 Block Diagram: The user interface gets the user input from the analog stick like device the user controls with their feet. The control unit will convert the user input to a keystroke or mouse movement and then execute the command on a computer.

Our design has two main subsystems, the user interface and the control unit. Through the user interface the user can provide their keyboard and mouse inputs by using the two 8 direction selection modules. Each of these consist of a set of switches which when active will determine the user's selection for a group of characters as well as a center stomp button for additional inputs. The external stomp button allows the user to alternate between capital and lowercase letters and alternate key inputs as seen in Figure 2. The control unit is responsible for converting the user input to the respective character or function as well as sending a signal to the computer via USB to execute the keypress or mouse input.

2. Second Project Implementation

In this chapter, we include implementations we made to our keyboard firmware, the PCB design, and calculations of the tolerance analysis for the physical component from the Design Review.

2.1 Physical Design

Our design calls for a mechanical design that allows the user to easily select between 8 directional switches and a downwards stop switch. To accomplish this our design has 3 primary moving parts. Shown below in Figure 3 in black is a rotating foot grip. Through physical experimentation, it was determined that allowing the foot grip to rotate as the user selects directions allowed for faster and easier input. Holding the black foot grip is a top plate connected to a stem. The top plate rides on the gray piston and blue fram to ensure angular stability during use. The bottom of the stem will actuate a corresponding directional switch, shown in white, when the foot grip and top plate have been moved 2cm in any of the eight directions. Eight springs are used to recenter the grip after each actuation. To allow for angular stability when pressing the stomp button we've implemented a grey piston riding in the blue cylinder. Pressing down on the foot grip will move the foot grip, the top plate, and the piston downwards and press the stomp switch shown at the bottom in white.

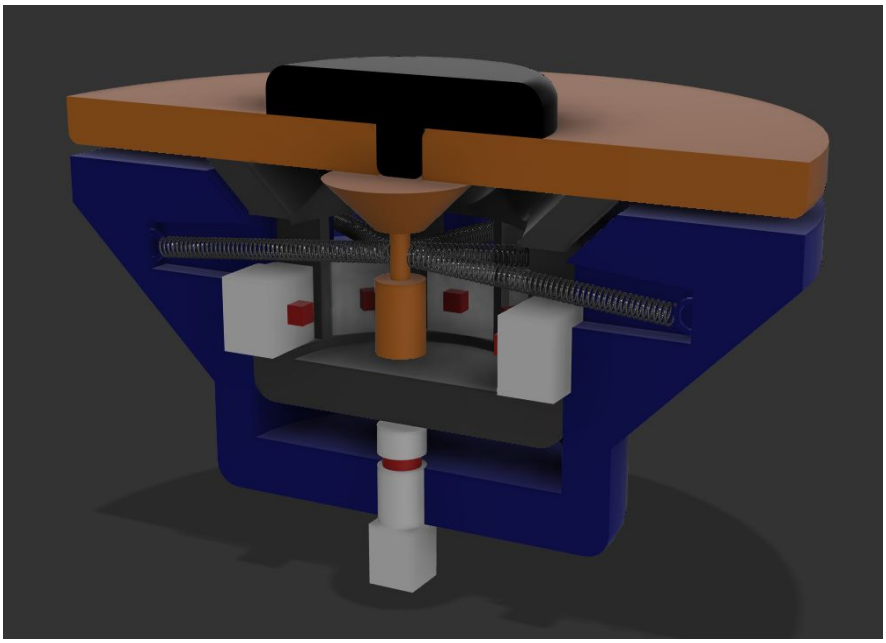


Fig 3 CAD of Physical Design: Cross section of the design showing the interaction between the foot grip, top plate, piston, cylinder, and switches.

A requirement of our physical design is protecting the switches from being depressed too far, potentially damaging them. To prevent this, the moving parts will always be stopped by a hard surface before over actuating any switch. For the directional switches, this is accomplished by the baffles on the top rim of the cylinder. The baffles also will center the stem on the switch to prevent the user from pushing it in between the eight directions. This interaction is shown below in Figure 4.

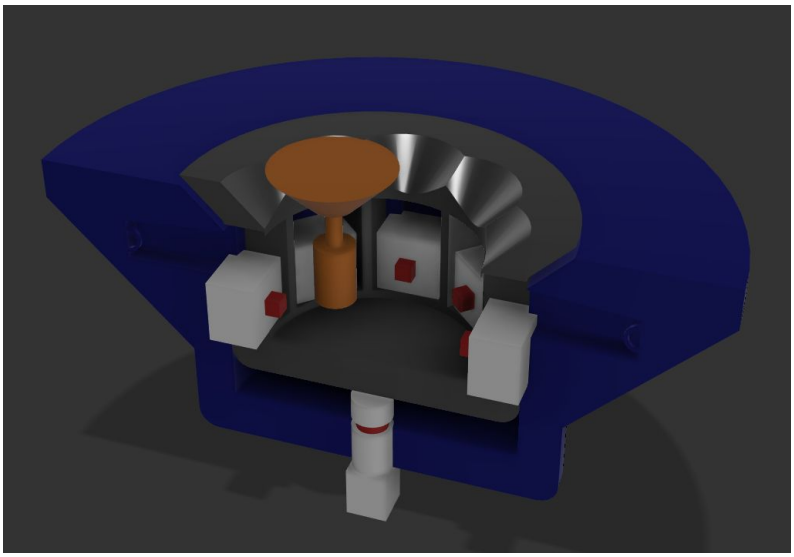


Fig 4 Stem and Piston Interaction: The baffles on the top of the piston will guide the stem into the 8 directions and stop it before it overactuates a switch.

To protect the stomp switch, when the piston is pushed 2.7mm downwards (the actuation distance of the switch [4]) the top of the piston and the blue cylinder are level, which prevents it from being further depressed.

2.2 Physical Design Tolerance

The most important physical parameters in our design are the switch actuation forces and distances. For distances, through experimentation it was determined that at least 1.5cm of travel is required to easily identify between the 8 different directions, so to allow a small buffer our design uses 2cm. With the 2cm travel distance we can then calculate the distance each spring is extended/compressed and the total force on the stem itself. Using the CAD sketch shown in Figure 5 the lengths of the springs at rest was found to be 55mm.

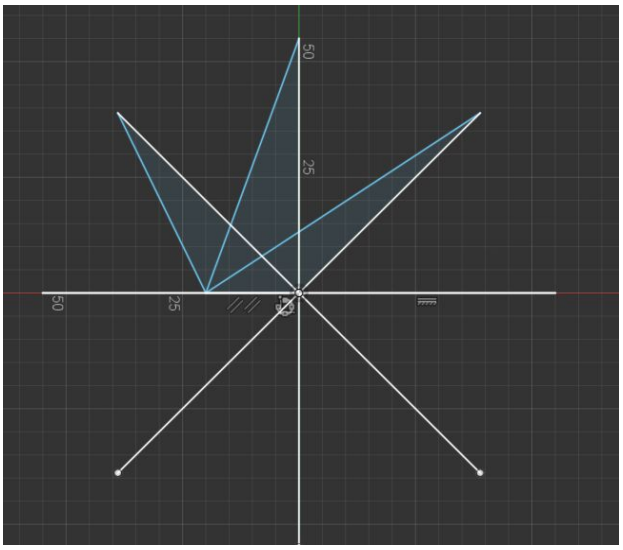


Fig 5 Spring Geometry: The springs at rest shown in white and the 3 extended springs shown in blue. Units are mm.

With 20mm of travel, our springs must be less than or equal to 35mm at rest and be able to extend to 75mm. Because of the scale of our design, the max spring diameter our design can accommodate is about 3mm. Using measurements from Figure 5, the force on the stem when it has been pushed 2cm is:

$$F(k,l) = 2(70-l)kcos(33.4°) + 2(58.5-l)kcos(70°) + 2(43.1-l)kcos(64.1°) + (35-l)k + (75-l)k$$

where k is the spring constant of every spring and l is the rest length of every spring. Commercially available springs around 35mm in length and 3mm in diameter have spring constants in the range of 0.08N/mm to 0.6N/mm[5]. Using a 35mm spring with a spring constant of 0.08N/mm yields a force of 8.632N. This is less than half of the 16.67N actuation force required to press the stomp button [4] which will prevent the user from accidentally pressing stomp when selecting directions.

2.3 Microcontroller Firmware

2.3.1 Overall Firmware Design

A microcontroller is required to communicate between the keyboard and the computer. The microcontroller will take and compile the inputs from the user interface to determine which character has been selected. Once selected, the microcontroller must send a signal to the computer via USB behaving as a standard USB keyboard/mouse would. The microcontroller would also receive power from the computer to power the keyboard.

We need to build firmware for our keyboard but we can not use the standard keymaps available since our keyboard doesn't use the same design.

We have to make sure we're reading three inputs at each time for the keyboard keypresses. One would be from the 8 selection group module or the left stomp button, the other would be the 8 selection character module or the right stomp button and last input is from the Caps Lock stomp button. As the switch input is mechanical, the switch data will be compared with cached switch data from the previous iteration to debounce the input. This adds a strict 5ms input delay, but it is necessary to prevent accidentally sending double or triple keystrokes.

For our mouse controls we will only account for the 8 selection direction module (same as the 8 selection character module) and the right stomp button press. We will only read for mouse controls when the right foot grip provides the input first.

Apart from building the keymap our process will be similar to other default keyboard firmware.

We're using the Keyboard firmwares for Atmel AVR and Cortex-M github repository [6]. To build the firmware we follow the guide to create the makefiles.

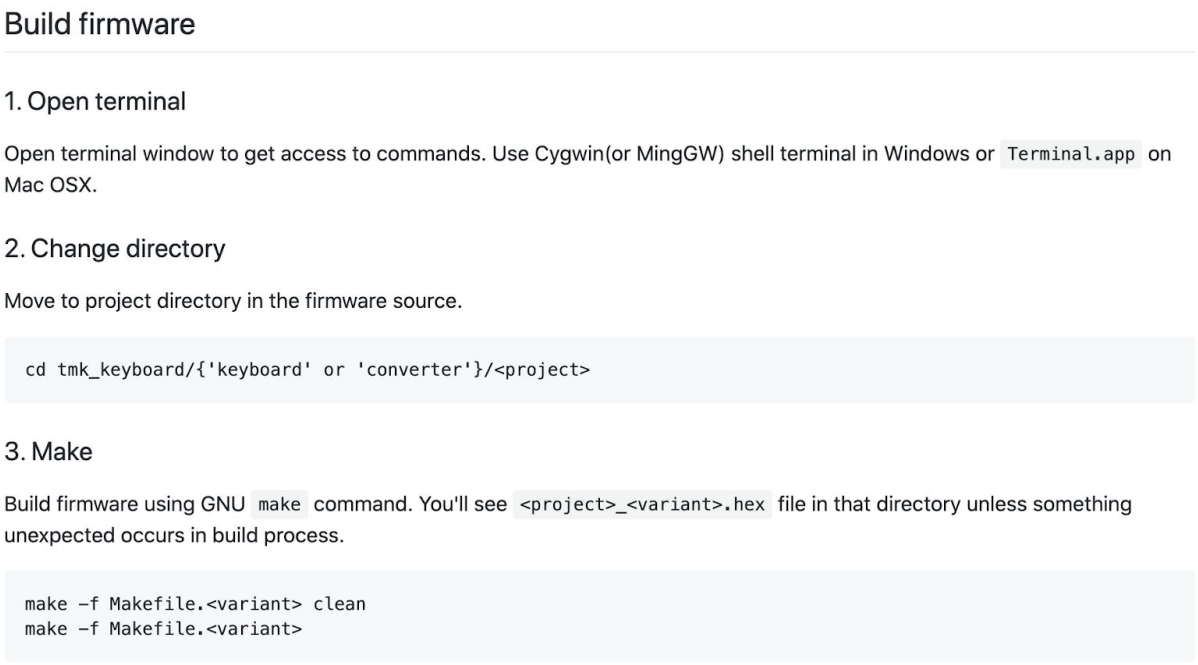


Fig 6 Firmware Build: Steps to follow to build the firmware [6].

In the Makefile for our firmware build we will follow the makefile provided in the repository as shown below, but we will be using our own keymap where the code calls for the keymap_$(KEYMAP).c file.

```
# Target file name (without extension).
TARGET = gh60_lufa

# Directory common source filess exist
TMK_DIR = ../../tmk_core

# Directory keyboard dependent files exist
TARGET_DIR = .

# project specific files
SRC = matrix.c \
  led.c

ifdef KEYMAP
    SRC := keymap_$(KEYMAP).c $(SRC)
else
    SRC := keymap_poker.c $(SRC)
endif

CONFIG_H = config.h


# MCU name
#MCU = at90usb1287
MCU = atmega32u4
```

Fig 7 Makefile: Specifying the microcontroller to be used for building our firmware

```
# Boot Section Size in *bytes*
#   Teensy halfKay   512
#   Teensy++ halfKay 1024
#   Atmel DFU loader 4096
#   LUFA bootloader  4096
#   USBaspLoader     2048
OPT_DEFS += -DBOOTLOADER_SIZE=4096


# Build Options
#   comment out to disable the options.
#
BOOTMAGIC_ENABLE = yes  # Virtual DIP switch configuration(+1000)
MOUSEKEY_ENABLE = yes # Mouse keys(+4700)
EXTRAKEY_ENABLE = yes # Audio control and System control(+450)
CONSOLE_ENABLE = yes  # Console for debug(+400)
COMMAND_ENABLE = yes    # Commands for debug and configuration
#SLEEP_LED_ENABLE = yes  # Breathing sleep LED during USB suspend
NKRO_ENABLE = yes # USB Nkey Rollover


# Optimize size but this may cause error "relocation truncated to fit"
#EXTRALDFLAGS = -Wl,--relax

# Search Path
VPATH += $(TARGET_DIR)
VPATH += $(TMK_DIR)

include $(TMK_DIR)/protocol/lufa.mk
include $(TMK_DIR)/common.mk
include $(TMK_DIR)/rules.mk
```

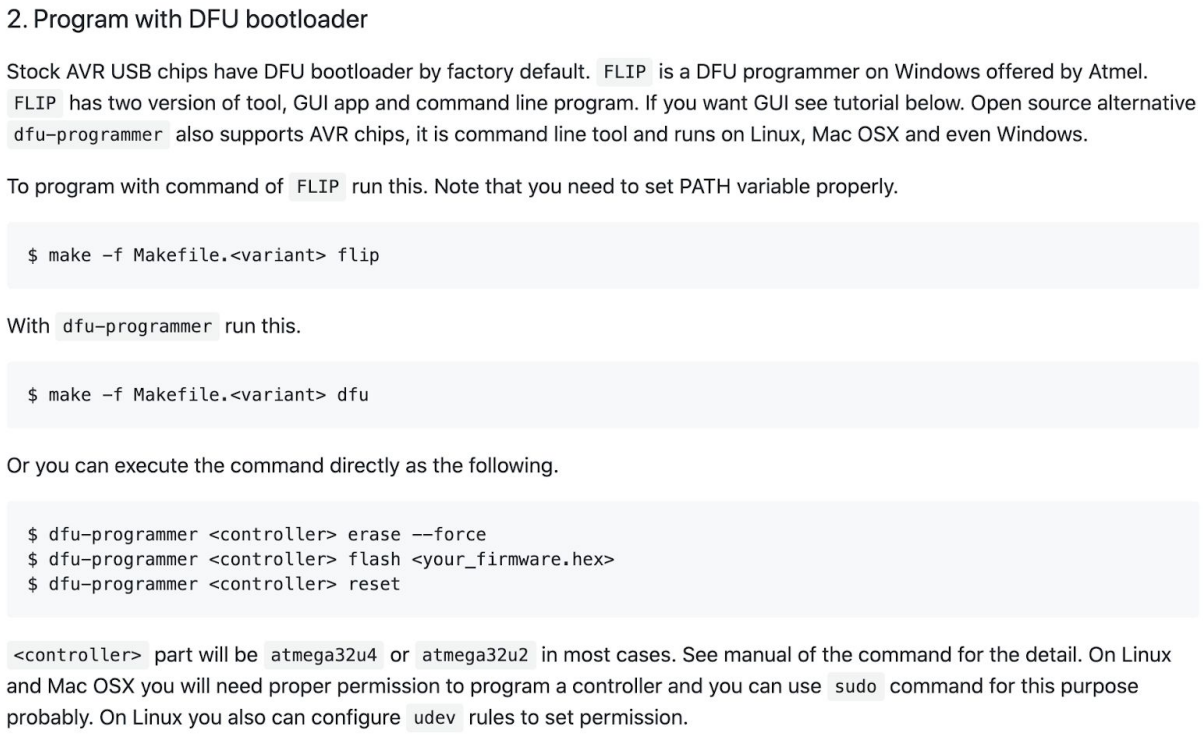
Fig 8 Makefile: To enable mouse key controls through our keyboard.

In the config.h file we set the rows and columns of the keyboard matrix to correspond to the keyboard we designed. We have eight directional inputs and a center stomp button for each foot, resulting in a 9x9 matrix.

```
#ifndef CONFIG_H
#define CONFIG_H


/* USB Device descriptor parameter */
#define VENDOR_ID       0xFEED
#define PRODUCT_ID      0x6060
#define DEVICE_VER      0x0001
#define MANUFACTURER    geekhack
#define PRODUCT         GH60
#define DESCRIPTION     t.m.k. keyboard firmware for GH60

/* key matrix size */
#define MATRIX_ROWS 9
#define MATRIX_COLS 9

/* define if matrix has ghost */
//#define MATRIX_HAS_GHOST

/* Set 0 if debouncing isn't needed */
#define DEBOUNCE    5

/* Mechanical locking support. Use KC_LCAP, KC_LNUM or KC_LSCR instead in keymap
#define LOCKING_SUPPORT_ENABLE
/* Locking resynchronize hack */
#define LOCKING_RESYNC_ENABLE

/* key combination for command */
#define IS_COMMAND() ( \
    keyboard_report->mods == (MOD_BIT(KC_LSHIFT) | MOD_BIT(KC_RSHIFT)) \
)
```

Fig 9 Config.h: Setting the rows and columns of the keyboard matrix according to what we need for our keyboard which is 9 x 9. 9 keys for left foot grip and 9 keys for the right foot grip.

To burn the firmware onto the microcontroller we will use Atmel's dfu-programmer.



2. Program with DFU bootloader

Stock AVR USB chips have DFU bootloader by factory default. `FLIP` is a DFU programmer on Windows offered by Atmel. `FLIP` has two version of tool, GUI app and command line program. If you want GUI see tutorial below. Open source alternative `dfu-programmer` also supports AVR chips, it is command line tool and runs on Linux, Mac OSX and even Windows.

To program with command of `FLIP` run this. Note that you need to set PATH variable properly.

```
$ make -f Makefile.<variant> flip
```

With `dfu-programmer` run this.

```
$ make -f Makefile.<variant> dfu
```

Or you can execute the command directly as the following.

```
$ dfu-programmer <controller> erase --force
$ dfu-programmer <controller> flash <your_firmware.hex>
$ dfu-programmer <controller> reset
```

`<controller>` part will be `atmega32u4` or `atmega32u2` in most cases. See manual of the command for the detail. On Linux and Mac OSX you will need proper permission to program a controller and you can use `sudo` command for this purpose probably. On Linux you also can configure `udev` rules to set permission.

Fig 10 DFU-Programmer: Steps to program the DFU bootloader which is basically being used to burn the firmware in the microcontroller.

2.3.2 Keymap

Due to our keyboard design being significantly different from a standard PC keyboard, we designed our own keymap to define our key layout. To use a custom keymap with the firmware we are building, we follow the instructions provided in the Atmel AVR repository which will build the firmware binary hex file.

Keymap

Several version of keymap are available in advance but you are recommended to define your favorite layout yourself. To define your own keymap create file named `keymap_<name>.c` and see keymap document(you can find in top README.md) and existent keymap files.

To build firmware binary hex file with a certain keymap just do `make` with `KEYMAP` option like:

```
$ make KEYMAP=[poker|poker_set|poker_bit|plain|hasu|spacefn|hhkb|<name>]
```

Fig 11 Build Firmware with a Different Keymap: Creates the firmware binary hex file using the keymap we created.

We set the keymap definition to correspond to the keyboard inputs as shown in Figure 2.

```
/* keymap definition */
#define KEYMAP( \
    K00, K01, K02, K03, K04, K05, K06, K07, K08, \
    K10, K11, K12, K13, K14, K15, K16, K17, K18, \
    K20, K21, K22, K23, K24, K25, K26, K27, K28, \
    K30, K31, K32, K33, K34, K35, K36, K37, K38, \
    K40, K41, K42, K43, K44, K45, K46, K47, K48, \
    K50, K51, K52, K53, K54, K55, K56, K57, K58, \
    K60, K61, K62, K63, K64, K65, K66, K67, K68, \
    K70, K71, K72, K73, K74, K75, K76, K77, K78, \
    K80, K81, K82, K83, K84, K85, K86, K87, K88 \
) { \
    { KC_##K00, KC_##K01, KC_##K02, KC_##K03, KC_##K04, KC_##K05, KC_##K06, KC_##K07, KC_##K08}, \
    { KC_##K10, KC_##K11, KC_##K12, KC_##K13, KC_##K14, KC_##K15, KC_##K16, KC_##K17, KC_##K18}, \
    { KC_##K20, KC_##K21, KC_##K22, KC_##K23, KC_##K24, KC_##K25, KC_##K26, KC_##K27, KC_##K28}, \
    { KC_##K30, KC_##K31, KC_##K32, KC_##K33, KC_##K34, KC_##K35, KC_##K36, KC_##K37, KC_##K38}, \
    { KC_##K40, KC_##K41, KC_##K42, KC_##K43, KC_##K44, KC_##K45, KC_##K46, KC_##K47, KC_##K48}, \
    { KC_##K50, KC_##K51, KC_##K52, KC_##K53, KC_##K54, KC_##K55, KC_##K56, KC_##K57, KC_##K58}, \
    { KC_##K60, KC_##K61, KC_##K62, KC_##K63, KC_##K64, KC_##K65, KC_##K66, KC_##K67, KC_##K68}, \
    { KC_##K70, KC_##K71, KC_##K72, KC_##K73, KC_##K74, KC_##K75, KC_##K76, KC_##K77, KC_##K78}, \
    { KC_##K80, KC_##K81, KC_##K82, KC_##K83, KC_##K84, KC_##K85, KC_##K86, KC_##K87, KC_##K88} \
}
```

Fig 12 Keymap definition: 9 x 9 matrix keymap definition.

Despite having a very different input method than a traditional keyboard, our keys map to the microcontroller pins in a similar way to how standard keyboards are mapped. We use a keyboard matrix where the rows are the left foot inputs and the columns are the right foot inputs. The blank spaces are not mapped to any key and will not have any function when activated.

```
#include "keymap_common.h"

const uint8_t PROGMEM keymaps[][MATRIX_ROWS][MATRIX_COLS] = {
    /* 0: qwerty */
    KEYMAP(enter,  ctrl,  alt,  del,   tab,  shift,  prt_scrn,  `,     home, \
               q,     w,    e,    r,     t,      y,                        \
               a,     s,    d,    f,             j,         k,   l, pg_down, \
               u,     i,          g,     h,                 o,   p,          \
       backspace,     ;,     .,        space,   esc,         ,,   ',     end, \
               z,     x,     c,   v,             b,         n,   m,          \
               1,     2,     3,   4,      5,     -,         =,   /,   pg_up, \
               6,     7,     8,   9,      0,     [,         ],   \,          \
              up,        right,        down,              up                \
};
const action_t PROGMEM fn_actions[] = {};
```

Fig 13 Keymap assignment: Assigns the keys to the keymap according to our current cheatsheet.

2.4 Bill of Materials (BOM)

In Table 1, we have the bill of materials of all the materials we expect to use to build this project. The total cost of all the components is $75.28.

Item	Part # or Manufacturer	Count	Cost
ATMEGA32u4 Microcontroller	ATMEGA32U4-AU	1	$4.00
FS5700 Series Pushbutton Switch (“Stomp button”)	FS5700SPMT2B2M2QE	3	$15.45
Cherry MX Switch	MX1A-C1NW	16	$16.96
Micro USB Female Port	Molex	1	$0.78
USB-A to Micro USB Cable	Anker	1	$7.99
PCB	PCBWay	1	N/A
Keyboard Casing	N/A	1	$30.00
1uF Capacitor	CL10A105KP8NNNC	1	$0.10
Total Cost	**$75.28**		

Table 1 Bill of Materials: Itemized list of all components necessary to build the keyboard.

3. Second Project Conclusions

In this chapter, we summarize the implementations that we made, discuss which parts we were unable to implement, discuss ethics and safety concerns that may arise with this design, and describe the improvements we could make to our design if we were given a full semester to work on this project.

3.1 Implementation Summary

In Chapter 2 we determined all of the parameters for the physical component of our design. With this we are able to prove that our design can be implemented in this way with switches used to get the keyboard inputs, and that there would be no issues with multiple switches being activated. We also have a plan for how the firmware would be burned onto the microcontroller and determine how the computer would recognize each key event. This is also very important as our keyboard layout is very different from a standard keyboard and with this firmware we can ensure the computer will be able to recognize the inputs correctly from the microcontroller. Nick worked on determining the parameters for the physical components and creating the CAD model of the physical design. Aditi worked on determining how to build the firmware to the microcontroller. And, Neva worked on creating the keymap for the firmware that is to be burned onto the microcontroller.

3.2 Unknowns, uncertainties, testing needed

Due to the current circumstances, there are a few components which we are unable to complete, greatly impacting our project's completion. We are not able to build a working prototype of our keyboard as we cannot access the machine shop to discuss the best way to construct our keyboard. We do have the models of our design, but we also do not own the necessary components to accurately test the optimal elasticity of the springs or the distance the foot needs to travel to make contact with a switch. If we were able to test the components, we would test the spring elasticity by determining the average force an adult has when sliding their foot forward on the foot grip and then comparing that to the force exerted by the spring when stretched to reach the switches. And then from there determine the most comfortable spring elasticity for the user.

We were unable to order the microcontroller on time to build the firmware for our keyboard. Although there are plenty of resources on how to build the firmware for a keyboard there is not a lot about adding the mouse controls to the firmware. Typically the way mouse controls are added to a keyboard is that some keys on the keyboard would act as the up, down, right, left keys for

mouse controls. However for our project, we want the microcontroller to send mouse controls without adding it to the keyboard. We would either have to make a different layer for our mouse controls in our keymap or not add it to the firmware. If we have the microcontroller we could have tested our different approaches and would have gone with the one that worked well for us. Another aspect of our project we are unable to complete is testing and assembling the hardware component. We would not be able to order a PCB or any of our other hardware components and expect to receive it in time to complete the project before the semester ended. With the PCB we could have tested the functionality of the keyboard on an online keyboard tester.

3.3 Ethics and Safety

There are a few safety hazards that must be taken into consideration with our product. As the device will be out of sight of the user, it must not have any sharp edges or otherwise dangerous surfaces that the user could unknowingly injure themselves on. The user should be able to apply significant force to any part of the frame without it harming them. As an additional precaution, the frame should not be made of a material that could break or shatter under significant weight. Because we have moving parts, we must ensure the foot pads and stomp buttons cannot pinch the user's feet at any point in their range of motion. Thus for the safety of the users our project must comply with IEEE Code of Ethics #9 [7]:

"*to avoid injuring others, their property, reputation, or employment by false or malicious action;*"

Since our product caters to the people who are unable to use the keyboard with their hands, we must be realistic in stating claims about the features and success of the product, in accordance with IEEE Code of Ethics #3 [7]:

"to be honest and realistic in stating claims or estimates based on available data".

We will make sure we vigorously test our product to get a better accuracy of the success of our product.

Our product does require firmware to interface with the computer and thus we comply by the ACM Code of Conduct #2.8 [8]:

"computing professionals should not access another's computer system, software, or data without a reasonable belief that such an action would be authorized or a compelling belief that it is consistent with the public good."

We will ensure our firmware does not gather any data from the computer and only operates as we advertise, to be an alternate keyboard input device.

For the success of this product we will consider all the constructive criticism and suggestion on improving the performance which adheres to the IEEE Code of Ethics #7 [9]:

"to seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others".

3.4 Project Improvements

Due to the shortened time provided for implementing our project we designed our keyboard to be very simple such as using only switches to receive input and connecting to the PC via a direct USB connection. Given a year to complete this project we could have made additional improvements to increase the ease of use of our keyboard. The following are the improvements we would accomplish:

1. **Intuitive Mouse Movement**
   The way mouse movement is currently implemented is that when the left foot is in the neutral position the right foot position controls the mouse position at a constant velocity. While this is a sufficient method of moving the mouse, for certain applications the user may want to have more control over the mouse speed as they would with a standard computer mouse. Given more time, we would have added additional hardware to make the mouse movement more intuitive by having the velocity of the mouse depend on the displacement of the user's foot in a given direction such as pressure sensors which can determine how far away the foot grip moved based on the force exerted by the user's foot.

2. **Bluetooth Connection Capabilities**
   We would have also liked to add a Bluetooth module to the keyboard. Our device is designed for a user who cannot use their hands and the current implementation of just a USB cable may be difficult for the user to connect to their computer. By adding in a Bluetooth module, we could minimize inconveniences since the user would be able to connect their keyboard without having to physically plug anything in.

3. **Electronic Cheat Sheet**
   Finally, we would have liked to add an electronic version of the input mapping cheat sheet shown in Figure 2. The cheat sheet was designed to be as intuitive as possible, but some users may find it difficult to associate their foot direction with the direction they see on the sheet. The electronic cheat sheet would be a wireless array of LEDs that would communicate over Bluetooth with the foot keyboard. Visually, it would appear similar to the one shown in Figure 2, but the rows and columns would light up based on the

selection the user makes. This would help bridge the association between what is seen on the sheet and the direction the user is moving their foot.

4. Progress Made on First Project

After the design review we had to reconsider a lot of our design choices. Since we were advised to go for a cheaper sensor we settled on IPM-165 radar sensor which operates at 24GHz frequency because it would help improve the range of the device as compared to other lower frequency options. One issue we saw with the cheap sensors was that we would have to amplify the output signal for it to be any use for us. To amplify the signal we decided to go with the LM384 audio amplifier which is powered by a 12V regulator in the power supply and 10k potentiometer to adjust the output level to the DAC. The potentiometer was mainly used to debug or repair. [Figure 14 in Appendix A : Amplifier Circuit]

For the microcontroller we had to change from our initial choice ATmega to ARM Cortex M4 Microcontroller since it contains an easy-to-use blend of control and signal processing capabilities which was vital for our project's success. It also had enough PWM and I2C outputs to control the various devices in the user interface subsystem. This was powered by the 3.3V linear regulator. The main idea behind programming of the microcontroller was that when the push button input drops low, the microcontroller will transition into a scanning loop, which reads data from DAC into a large buffer. Once the buffer is full, new data will go in from one end and the old data will be shifter back and data at the end will be deleted. By analysing the frequency content of the buffer, the microcontroller will determine the velocities of objects in front of the sensor and perform filtering to ignore objects that aren't vehicles. The microcontroller will trigger the vibration motor over 12C with the intensity indicating the urgency of the detection. [Figure 15 in Appendix A : ARM Cortex M4 Microcontroller Circuit]

For the user interface we designed our piezoelectric speaker circuits diagram which notify the user when the cane attachment has been turned on, off or has reached low battery state. We also implemented our vibration motor circuit and which provides feedback to the user when a vehicle is detected by the sensor. [Figure 16 in Appendix A : User Interface Subsystem Circuits]

Lastly we designed our power subsystem which outputs a raw 14.4V level from the 18650 batteries and two regulated levels at 3.3V and 5V. An additional ⅛ battery level is used to measure the voltage of the battery produced with a simple resistor voltage divider. [Figure 17 in Appendix A : Power Subsystem Circuits]

We also talked with the machine shop to make a case for our cane attachment. We decided on making 1 inch diameter clamps for our cane.

5. References

[1] Amputee Coalition, “Limb Loss Statistics”, *The Amputee Coalition*, 2015. [Online] Available: https://www.amputee-coalition.org/resources/limb-loss-statistics/. [Accessed: 03-Apr-2020]

[2] United Nations, “Universal Declaration of Human Rights”, *United Nations*, 2020. [Online] Available: https://www.un.org/en/universal-declaration-human-rights/index.html. [Accessed: 03-Apr-2020]

[3] K. Kuligowska, P. Kisielewicz, A. Włodarz, “Speech Synthesis Systems: Disadvantages and Limitations”, *International Journal of Engineering and Technology,* 2018. [Online] Available: https://www.researchgate.net/publication/325554736_Speech_synthesis_systems_Disadvantages_and_limitations. [Accessed: 03-Apr-2020]

[4] E-Switch, “FS5700 Series Pushbutton Switch”, *E-Switch* [Online] Available: https://sten-eswitch-13110800-production.s3.amazonaws.com/system/asset/product_line/data_sheet/226/FS5700.pdf [Accessed: 8-May-2020]

[5] The Spring Store, “Stock Extension Springs Catalog”, *The Spring Store* [Online] Available: https://www.thespringstore.com/media/download-pdf-entire/Stock-Extension-Spring-Catalog.pdf [Accessed: 8-May-2020]

[6] TMK Firmware “Keyboard firmwares for Atmel AVR and Cortex-M”, *TMK Firmware*, 2017. [Online] Available: https://github.com/tmk/tmk_keyboard. [Accessed: 08-May-2020]

[7] IEEE, "IEEE Code of Ethics", 2020. [Online]. Available: http://www.ieee.org/about/corporate/governance/p7-8.html/. [Accessed: 02-Apr-2020].

[8] ACM, “ACM Code of Ethics and Professional Conduct”, 2020. [Online]. Available: https://www.acm.org/code-of-ethics [Accessed: 16-Apr-2020].

Appendix A 1st Project Materials

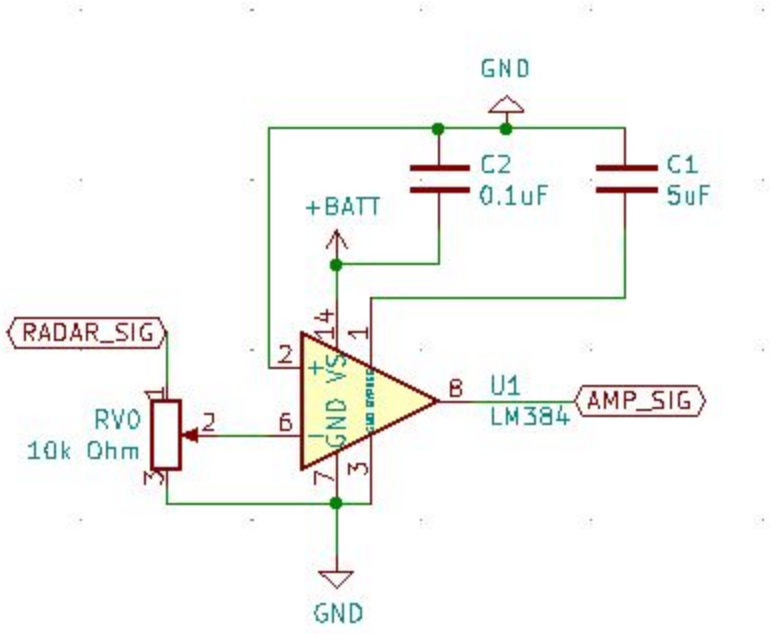


Fig 14 Amplifier Circuit: The IPM-165's output signal needs to be amplified before it can be of any use, so to amplify the signal before it goes to a DAC, we will use a LM384 audio amplifier. This amplifier will be powered by the 12V regulator in the power supply and have a 10k potentiometer to adjust the output level to the DAC.

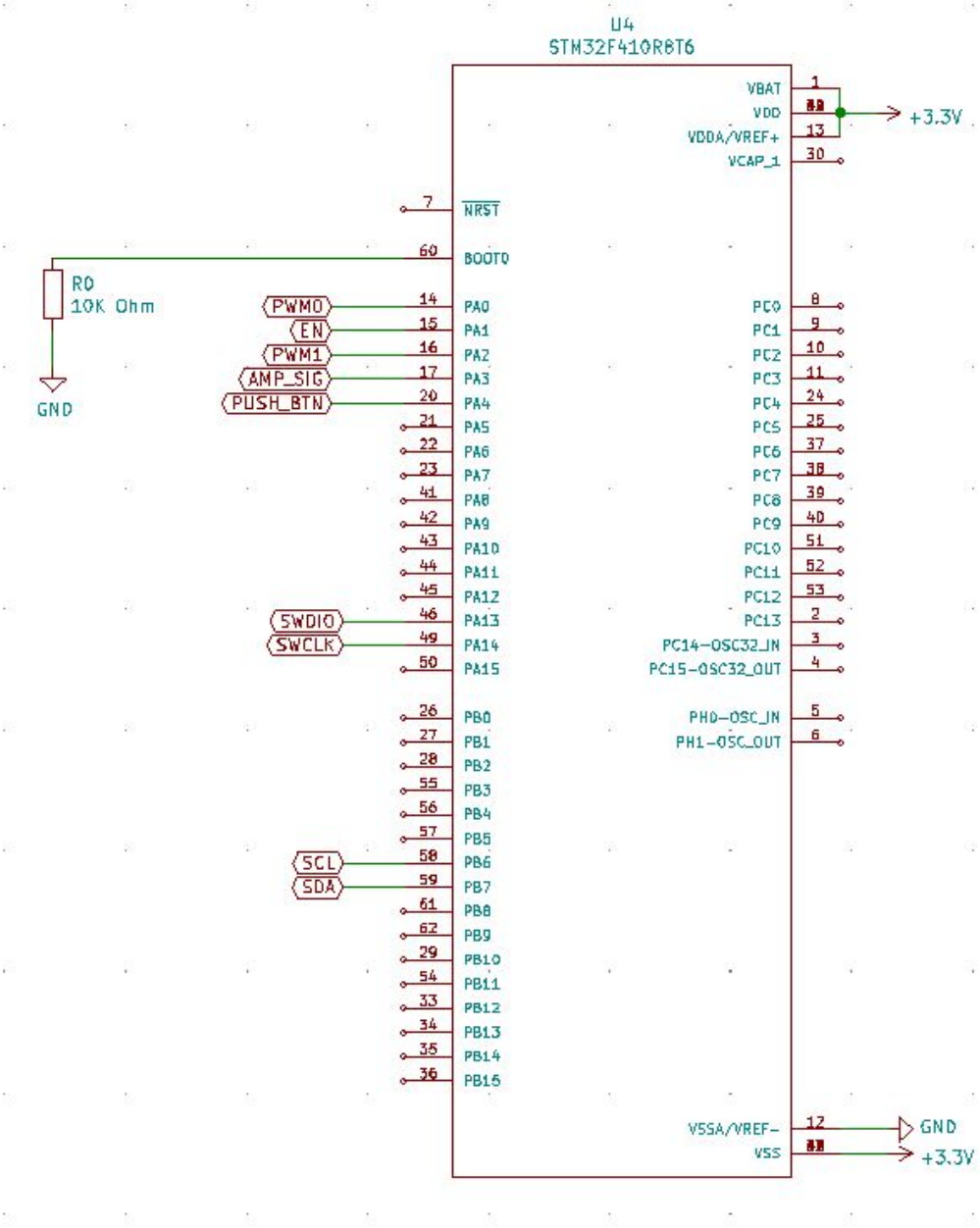


Fig 15 ARM Cortex M4 Microcontroller Circuit: When the push button input drops low, the microcontroller will transition into a scanning loop, which reads data from the DAC into a large buffer. The size of the buffer will be adjusted as necessary to maximize detection accuracy. Once the buffer is full, new data will go in one end and the old data will be shifted back. Data at the end of the buffer is deleted. By analyzing the frequency content of the buffer, the microcontroller will determine the velocities of objects in front of the sensor and perform filtering to ignore objects that aren't vehicles. If there is an object moving faster than 5mph approaching the sensor, the microcontroller will trigger the vibration motor over I2C with the intensity indicating the urgency of the detection, faster objects being more urgent.

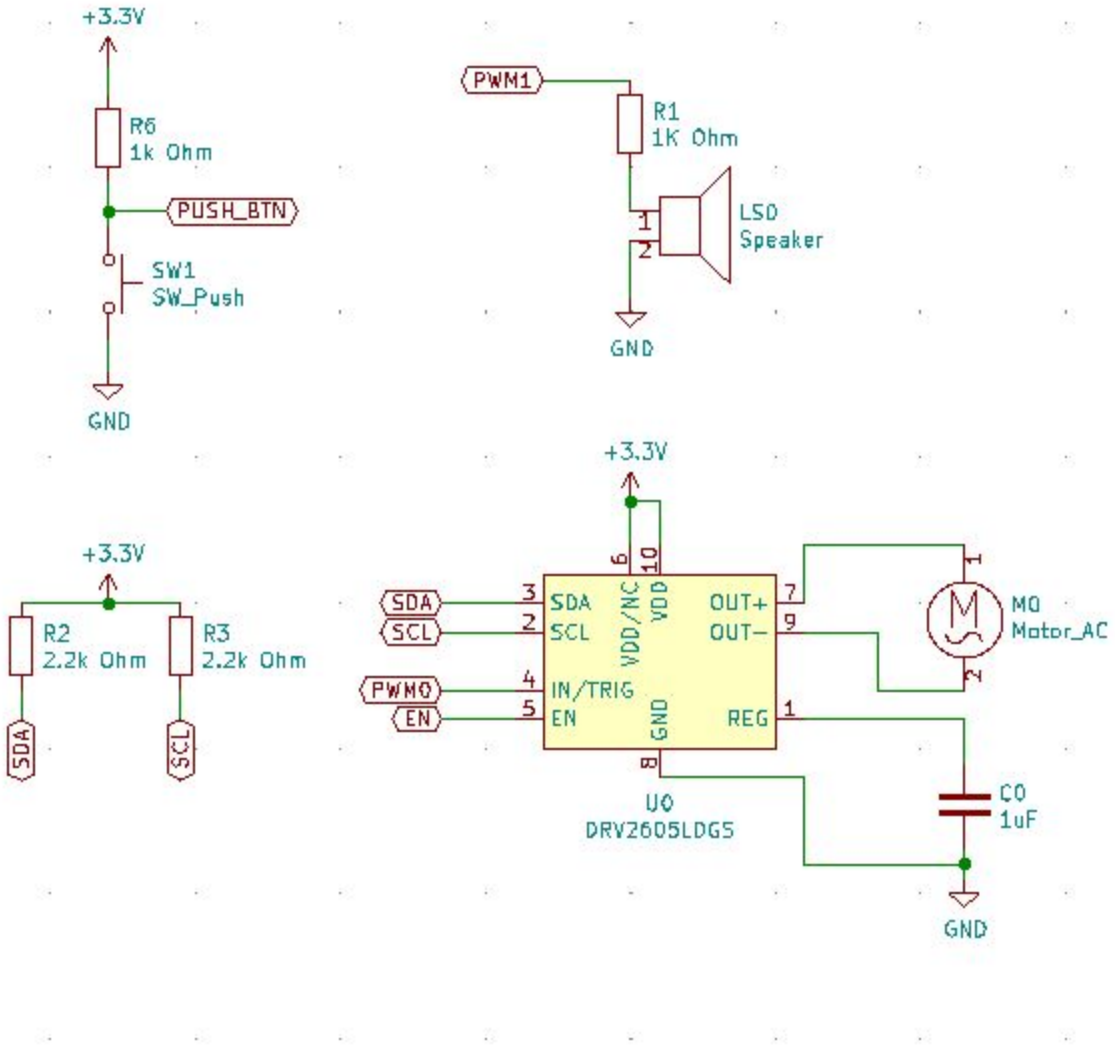


Fig 16 User Interface Subsystem Circuits: The user interface subsystem is responsible for providing user input and feedback to the user when a vehicle is detected.

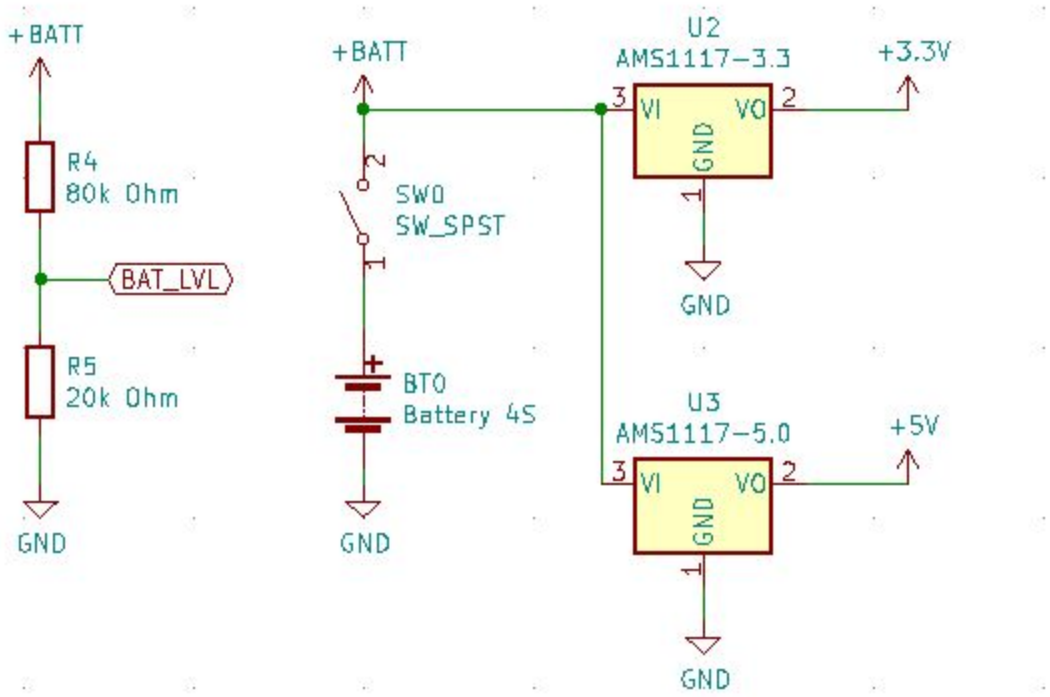


Fig 17 Power Subsystem Circuits: The power system is responsible for providing set voltages to the other subsystems. It outputs a raw approximately 14.4v level straight from the battery, and two regulated levels at 3.3v and 5v.

Appendix B 2nd Project Schematics

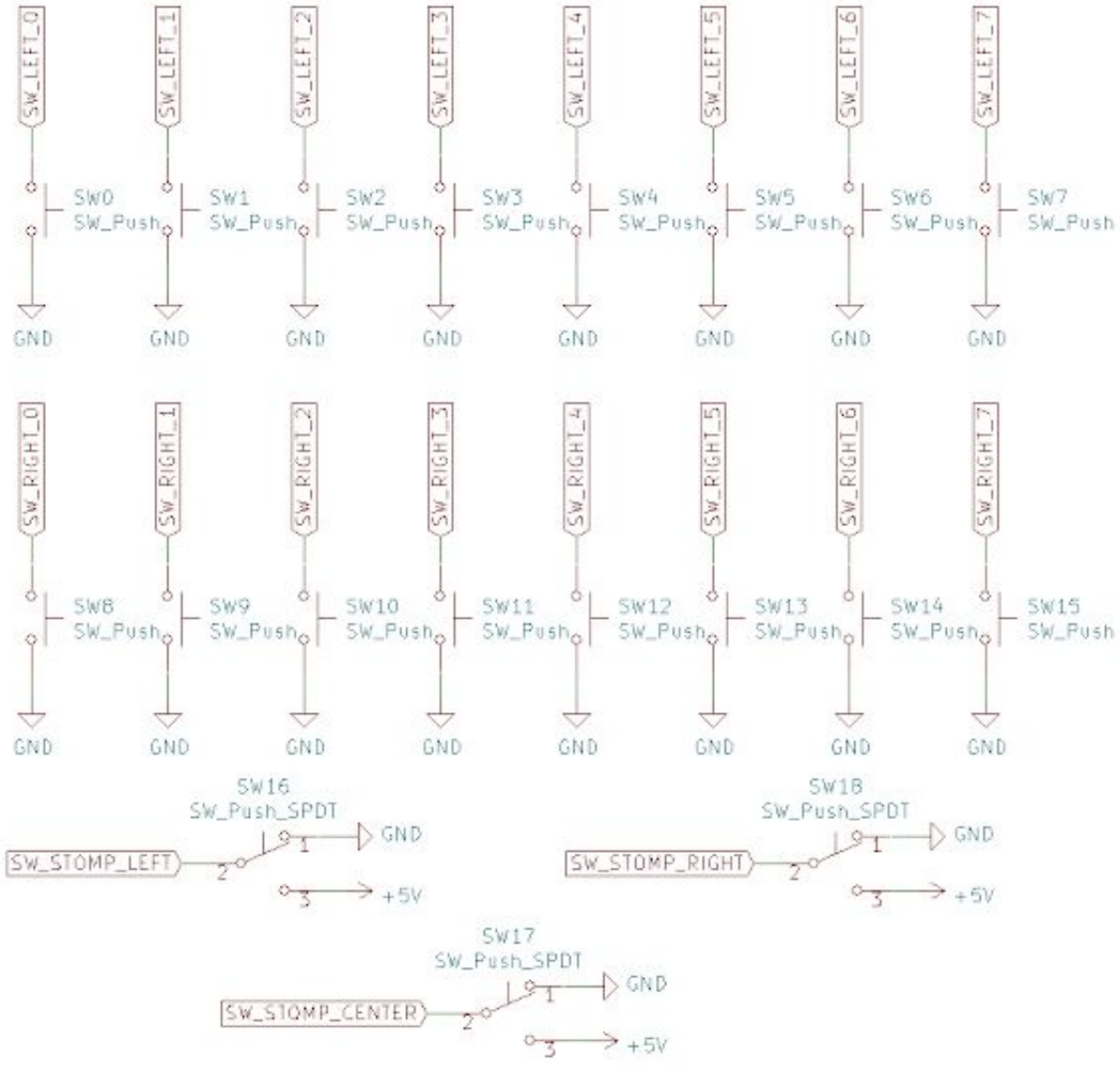


Fig 18 User Interface Schematic: Directional switches will be connected in a pullup configuration, while the SPDT stomp buttons will not.

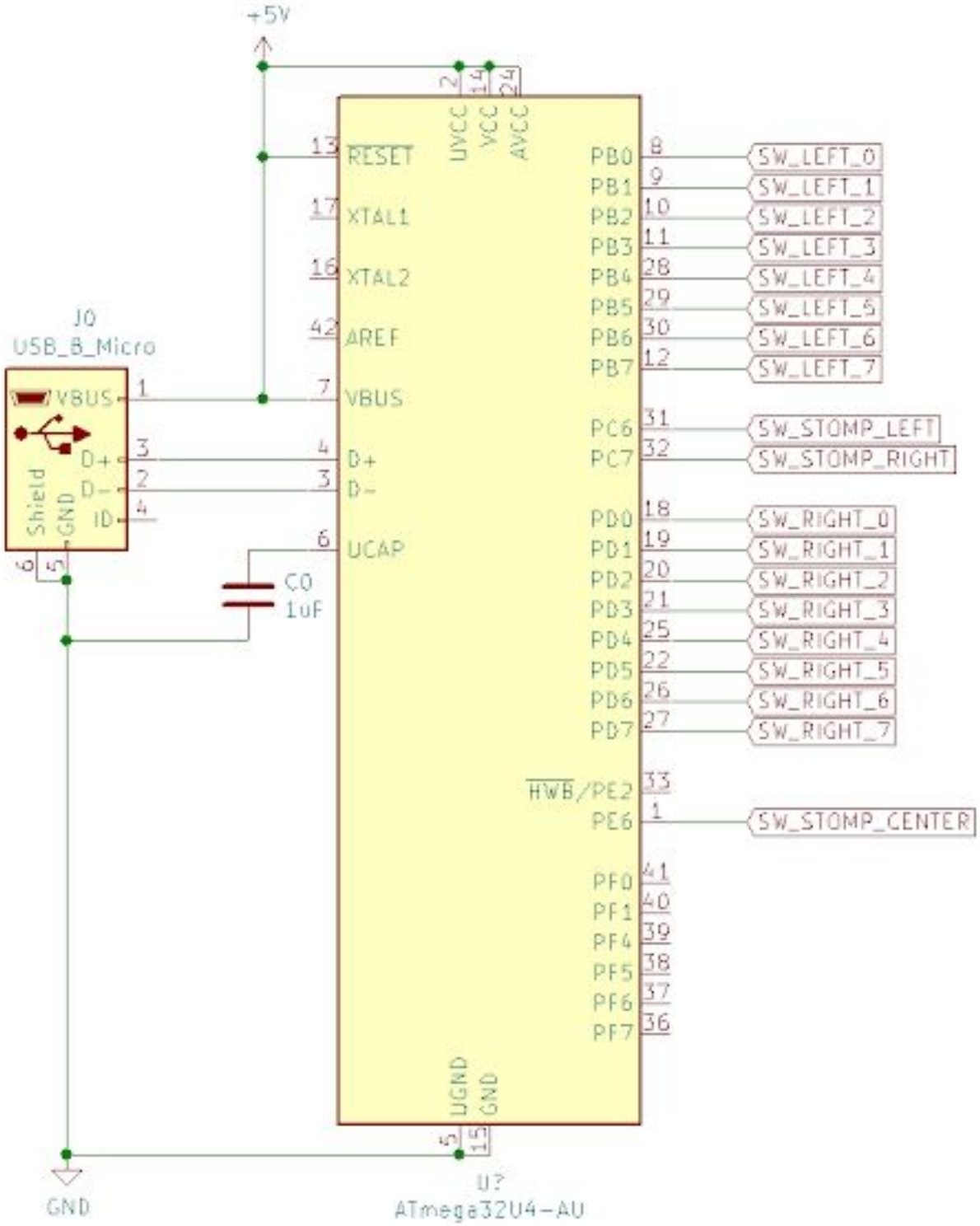


Fig 19 Control Unit Schematic: Power for the system will be supplied over USB.