Plug and Play Modular Keyboard

Design Document

1. Introduction

1.1. Contributors: Daniel Chen, Fangqi Han, Christian Held

1.2. Problem and Solution Overview:

The modern job-oriented citizen needs to have technology that lets them meet their goals in a timely manner. In their office, they have the space and comfort to use a full size keyboard effectively. On the go, however, they do not have this luxury. Additionally, left-handed typists find the position of the numpad on keyboards to be awkward. The usual solution to this difficulty is to either buy additional, smaller keyboards for travel or cumbersome keyboard extensions at home. That means one will need multiple keyboards that all provide the same function, but may not have the best configuration for every situation.

With those issues in mind, we think our approach to the keyboard will help those who find themselves travelling or working in tight quarters alleviate the problem of having to sacrifice function for comfort. Our product solves a user's problem by allowing them to maximize their work pace by changing the size and function of their keyboard according to their needs.

Our answer to this situation is to make a keyboard with detachable modules. This will allow users to conform to their working environment while still providing them with maximum utility. Users can simply add or remove modules to the keyboard by plugging or unplugging TRRS(Tip/Ring/Ring/Sleeve) connectors between modules. The keyboard will have adaptable firmware that allows the user to still have the most of the functionality available even with a condensed size keyboard.

1.3. **Visual Aid**

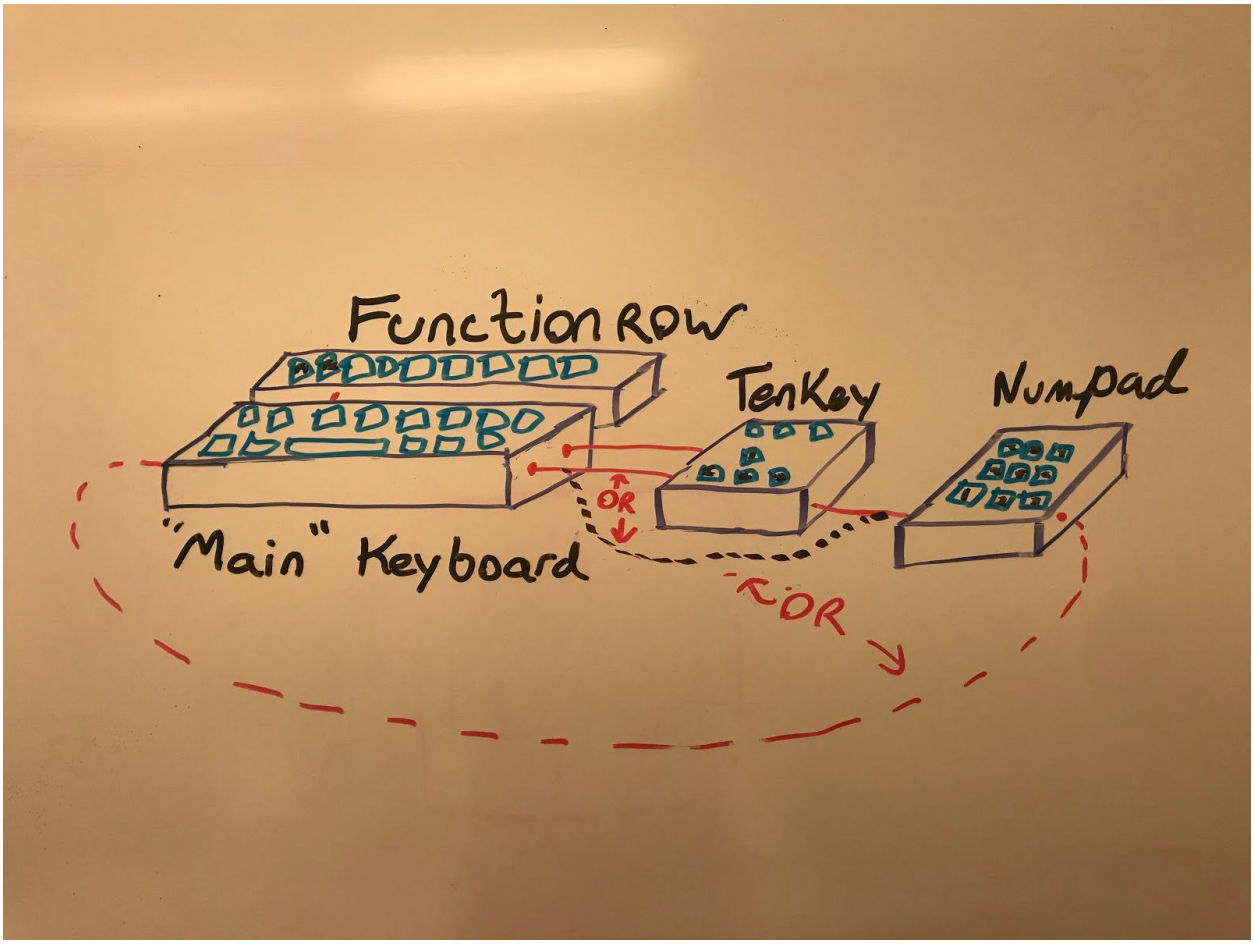

Figure 1. Physical Visual Aid

	Key 1	Key 2	Key 3	Key 4	Key 5	Key 6	Key 7	Key 8	Key 9	Key 10	Key 11	Key 12	Key 13	Key 14
Layer 1	`	1	2	3	4	5	6	7	8	9	0	-	=	⌫
Layer 2	~	!	@	#	$	%	^	&	*	(	)	_	+	

Figure 2. Example Keyboard Layer

Figure 2 provides a keyboard layer example. Layer 1 will be used by default. Holding or pressing a key defined by the firmware (e.g. SHIFT or FN) will cause the keyboard to activate another layer mapped with different symbols (e.g. Layer 2), from which new characters will be typed. When the active layer does not

cover a certain key (e.g. Layer 2 and Key 14 in the example), the keypress function will fall through to the previous layer and send key codes from that layer instead (e.g. BACKSPACE in Layer 1 for Key 14).

1.4. **High-level requirements list:**

- 1.4.i. The Main Keyboard is fully implemented with 61 keys that are recognized correctly as their individual keys in the key switch matrix.
- 1.4.ii. Auxiliary Modules also correctly read each key in the module's key matrix and can send data to the main microcontroller, with at least 10 usable keys on each module.
- 1.4.iii. Firmware can map keypresses to correct keyboard layers, find correct key codes, and send correct signals through USB.
- 1.4.iv. User-determined keypress signals can be assigned to at least 20 programmable keys supported by the firmware.

2. Design

2.1. Block Diagram:

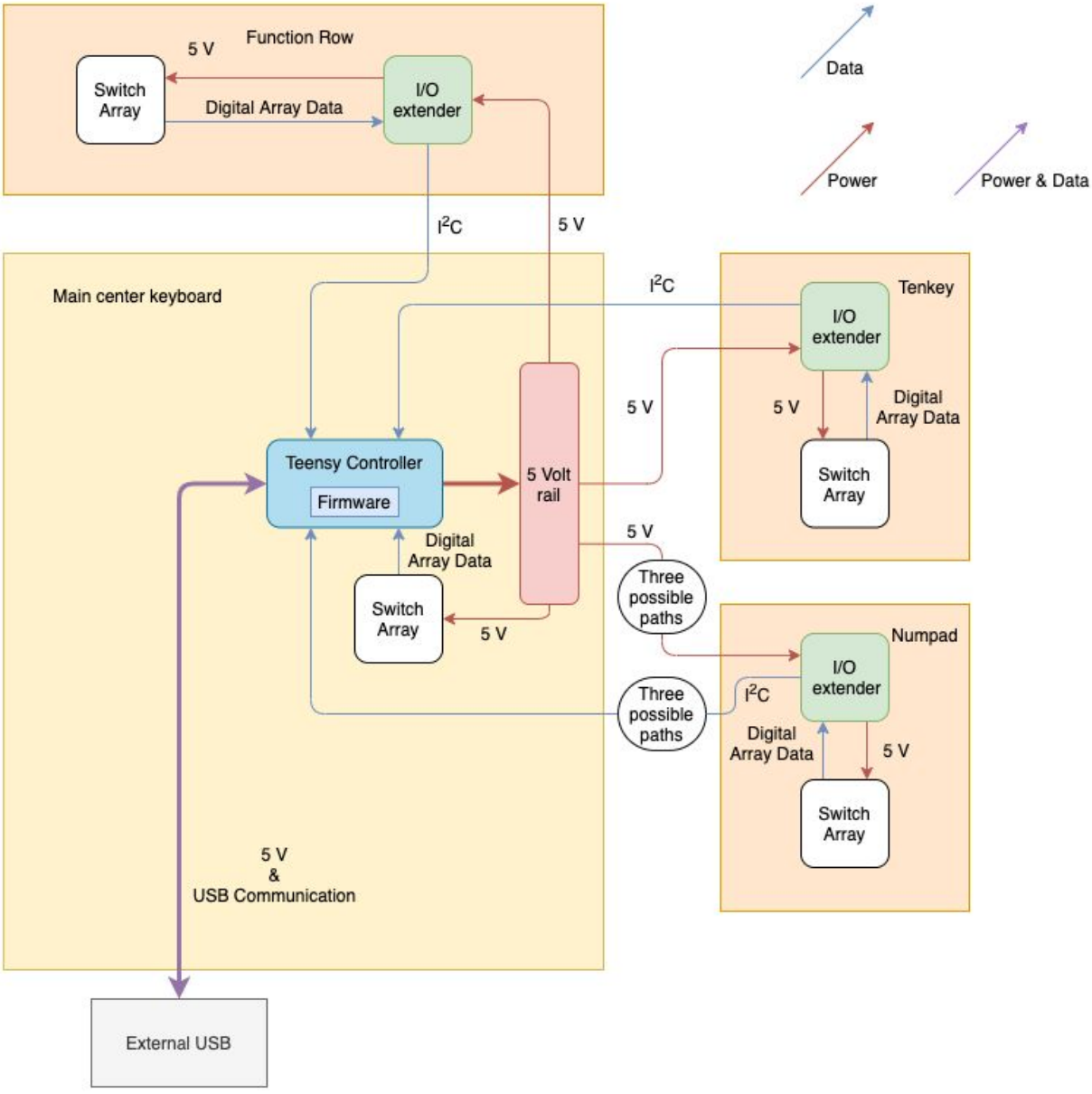

Figure 3. Hardware Block Diagram

An external USB powers the main microcontroller that is the operational key piece. This main component will be the condensed keyboard with most of the functionality without modules. The other modules will draw power from the main component and give I^2C data through a wired connection. When connected the

microcontroller will communicate through the I/O extenders on each module. The firmware on the microcontroller will be able to process key presses from all modules, but the main limitation will be USB data feed back to the computer.

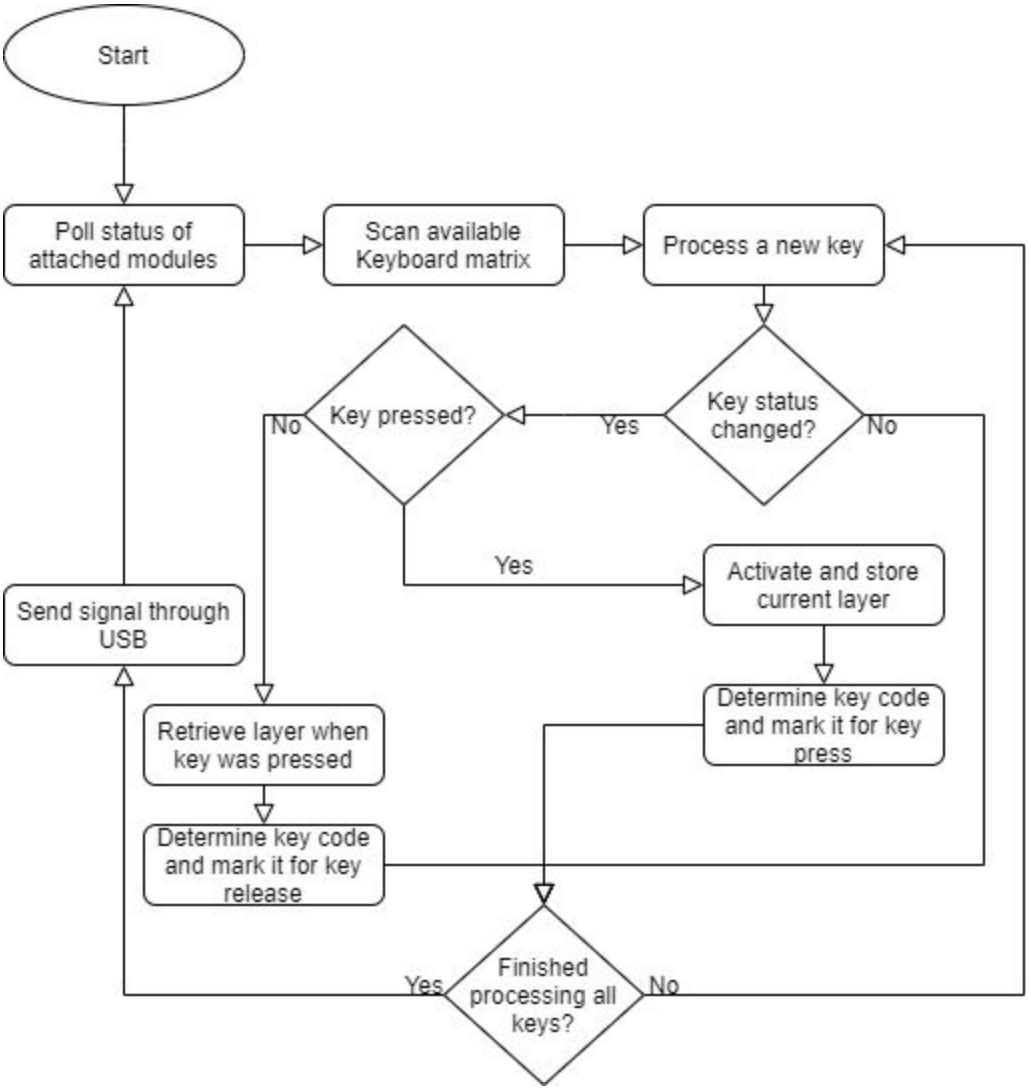

Figure 4. Block Diagram of Firmware's Main Loop

The firmware periodically polls the press/release status of the available keyboard matrix, which is determined by the number of modules attached. For each keypress, the active layer where the key is pressed is stored before it is used to find the correct keycode. For each key release, the layer stored while pressing the same key is retrieved to determine the keycode. Active layers for each key are stored separately. This design solves the problem where wrong keycodes are sent because of activated layers changing between pressing and releasing the same key. Signals will be sent through the USB regardless of whether any keys have been pressed or released.

2.2. **Schematic:**

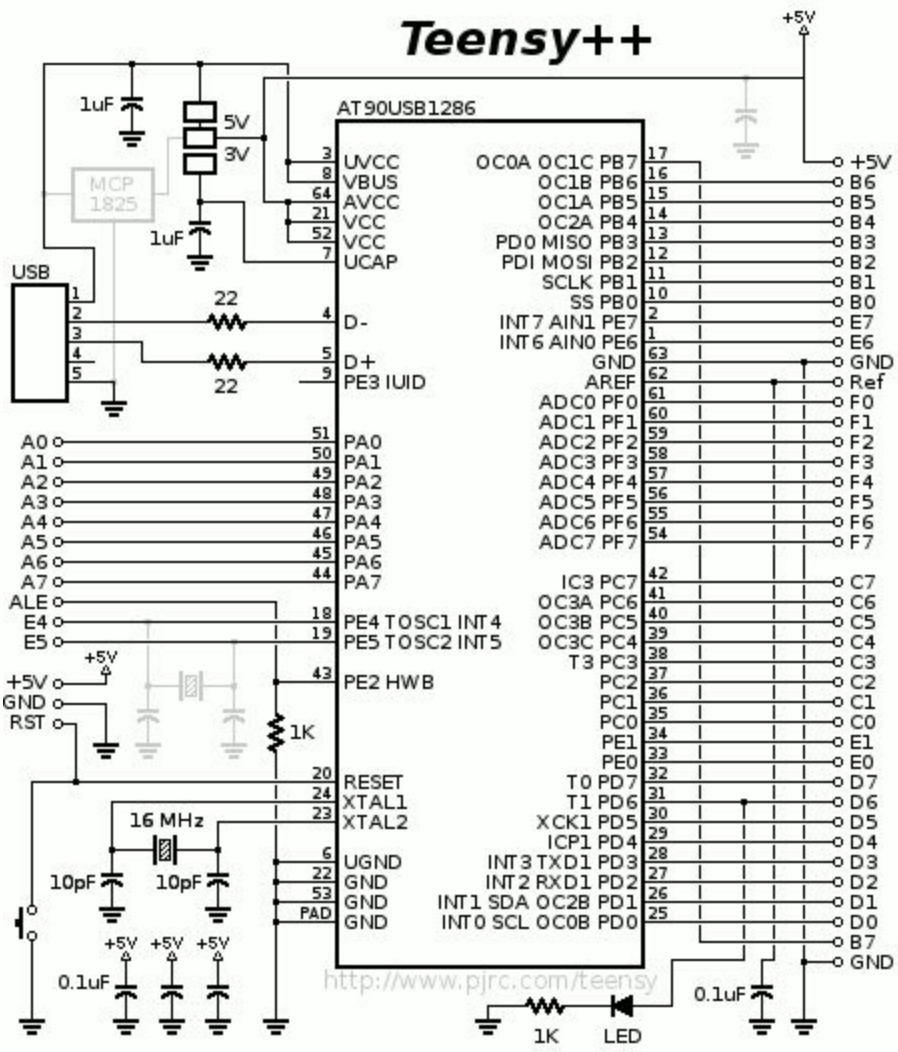

Figure 5. Provided Schematic for Teensy 2.0++[1]

[1] PJRC

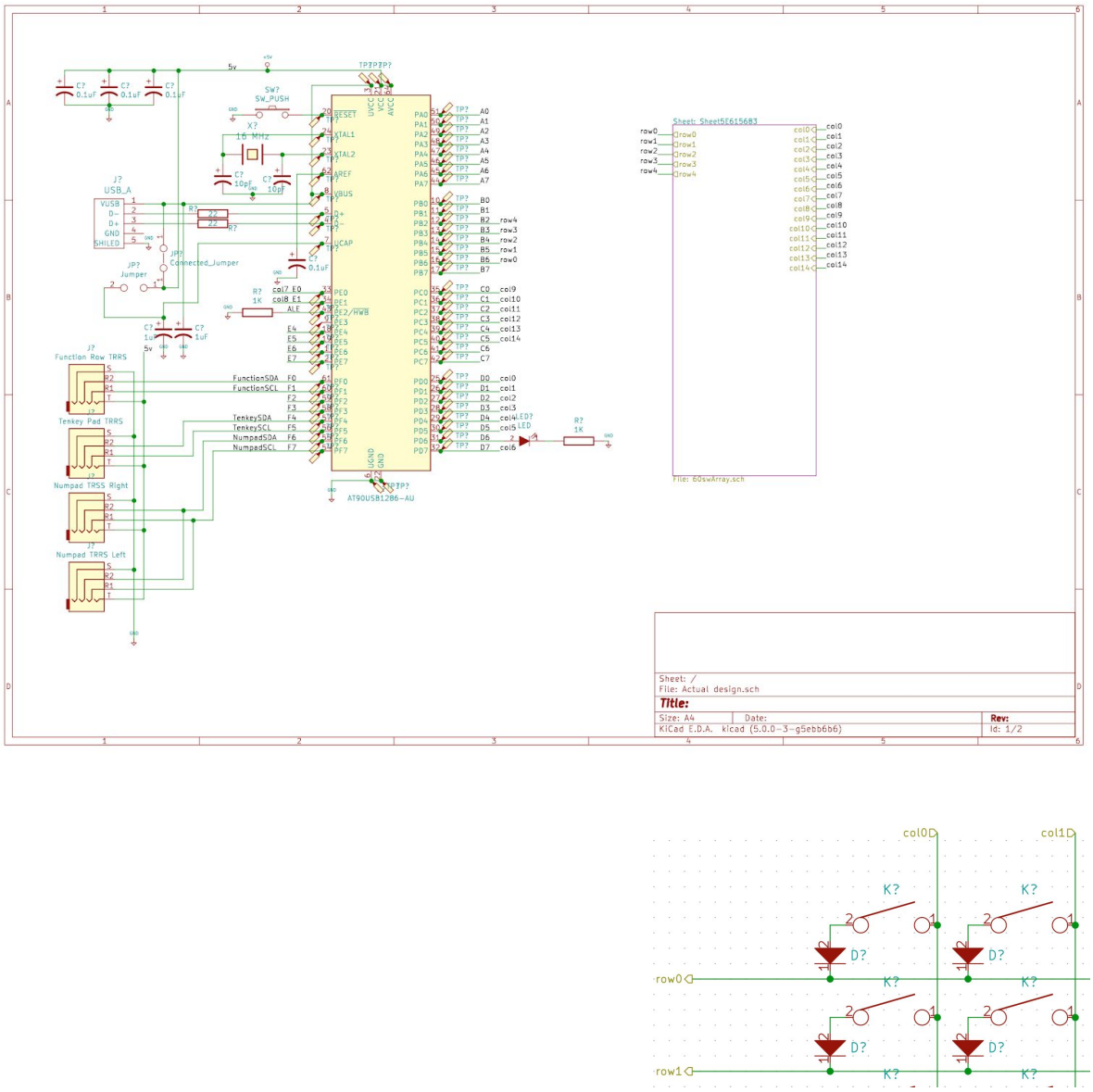

Figure 6. Kicad schematic.

Figure 6 shows the first steps of designing the keyboard PCB. The main challenge tackled in this figure is the routing for the processing chip (AT90USB1286). The chip needs USB for both power and communication, a clock generation system, and capacitors for voltage stabilization (Atmel). Teensy provides a picture of their schematic, included as figure 3), and the difficulty was attaching the right amount of keys and TRRS connections (a 3.5 mm connection with four channels: Tip, two Rings and one Shield). The key matrix (example on the lower right of the figure) is located within the Kicad hierarchical sheet. Most of the pins of the AT90USB1286 are I/O pins and are used to create the key matrix. Six of the I/O pins are dedicated to the I^2C signals from the

modules (it should be noted that the numpad connections are split between two TRRS connectors because it should be able to connect to both sides of the main board).

2.3. **I/O Expander**

2.3.i. These components on the exterior modules are the hub for power and data transfer on the modules. They receive power from the main keyboard and power the key matrix, whose data is sent back to the I/O extender. It interprets the data and returns data back to the central keyboard via I2C connection. Our current chip is the MCP23018 extender.

Requirement	**Verification**
Have a polling speed that can interpret key presses and send an I2C signal to the microcontroller within 10 ms .	Give a unit step signal to both the I/O expander and oscilloscope. Verify that the response signal from the expander is within 10 ms.

2.4. **Microcontroller**

2.4.i. The microcontroller receives power from the computer and disperses it through the connected systems. This power is sent through the matrix that contains all the keyboard switches, as well as to the external components through the I/O extenders. The I/O extenders will send back data they receive, which is organized and sent to the computer as typing inputs. The firmware for interpreting the key matrix inputs is on the microcontroller.

Requirements	**Verification**
1. Have a polling speed that allows a complete scan of the keyboard in around 100 ms. 2. Each key should be able to be read (assuming the module is attached). 3. Be able to run the firmware.	1. Give a unit step signal to both the controller and oscilloscope. Verify that the response signal from the expander is within 100 ms. 2. Run the previous test additionally on all rows and columns in unit tests. 3. Give an I^2C signal (provided from I/O expander) and verify it responds within 100 ms.

2.5. **Key Switch Array**

2.5.i. The input for our keyboard are mechanical key-switches, which when pressed shorts a connection and completes a circuit. When these switches are combined into a matrix and hooked up to a microcontroller, the firmware on the microcontroller can match these completed circuits to which keys were pressed and communicate what keys were pressed to the computer. As the main component that interacts with the user, this is one of the most important components and provides most of the usability and with its specific configuration will do much of the organizing for the microcontroller.

Requirements	**Verification**
1. No signal crossings between the switches. 2. “n” Key Rollover (NKRO). This means that the user can press as many keys as they can and there should be no failure to communicate which keys are pressed (though other bottlenecks might cause issues) (Trybus).	1. Continuity tests will be run for each path the switches are located around. 2. A full verification of every combination of key presses to prove NKRO would be very laborious and unuseful. Due to this constraint, about 15 random iterations of 6 keys will be tested to see if they produce the desired signals at the AT90USB1286 chip. The number 6 was chosen due to the limits of USB data transfering (Bates et al).

2.6. **Connectors**

2.6.i. The wiring between the main hub keyboard and the modules will be a TRRS cable. This is commonly known as a 3.5 mm jack. The TRRS version has four separate wires associated with it. The four things transferred will be 5V power, ground, SDA, and SCL (the latter two are part of the I2C protocol and are the only lines that return from the modules to the main hub) (I2C Info).

Requirements	**Verification**
Maintain a voltage above 4.2 volts	Place the wire in series with a

(lowest allowable voltage for a high signal for the AT90USB1286). The MCP23018 I/O extender indicates that the most it could potentially draw would be 400mA (Microchip Technology). So in addition to assuming exactly 5 volts from the USB connection at most the resistance of the wire could be 1 Ohm.	known resistor value. Apply DC 5 volts to the series and measure the current with an oscilloscope. Then modulate the 5 volts at 100kHz (speed of the I^2C connection) and verify that the impedance does not exceed 1 ohm.

2.7. **Firmware**

2.7.i. The firmware is software coded within the microcontroller that will detect and scan the switch array for pressed and released keys, and send out the corresponding data through the USB connection. Active keyboard layers, which are usually activated by pressing “fn” keys, are determined by the firmware with each keypress and affect the key code outputs based on the keyboard layout. A block diagram of this process is included in the Design section.

The firmware also provides programmability, allowing users to modify their keyboard layout to produce desired usability. Examples include changing the function layer’s layout, having programmable keys, and other features that would help with user productivity and customization.

Requirement	**Verification**
1. The firmware needs to periodically scan pressed and released keys, determine the active layers to find corresponding key codes, and then send out signals accordingly. 2. The firmware needs to periodically poll the status of attached auxiliary modules and adjust scanning range accordingly. 3. The key switch array should be customizable through user input, allowing the user to change corresponding functions of programmable keys.	1. Produce input signals corresponding to pressing and releasing each entry (including those in a different layer) and verify that the firmware has sent out all signals correctly. 2. Produce signals corresponding to a switch array entry on an auxiliary module with and without sending an input indicating that the module is attached. Verify that the firmware only sends out a signal when the module is attached, and that the signal is correct. 3. Assign various characters and commands to the programmable keys. Then send keypress signals for all programmable keys in random order. Verify that the output matches key customizations and is in correct order.

2.8. **Tolerance Analysis:**

2.8.i. According to Wyma et al, the human reaction time is about 210 ms from stimuli to response. Our goal is to have a keystroke event last about 100 ms. This is so the host computer has enough time to process and send the data to wherever it needs to go. The Cherry MX datasheets indicate that the average debounce time for a keystroke is about 5 ms. Assuming the longest pathway is from the exterior module keystroke, the path is as follows: Keystroke, I/O extender, Microcontroller, USB signal.

We made the assumption that our I^2C protocol frequency is 100kHz (I2C INFO), based upon the lowest value that our I/O expander can produce. This means that each 8 bit signal from the I/O expander will take approximately 8*1/100kHz = or 0.08 ms. The microcontroller we have preliminarily selected has a clock speed of 16MHz. This is a high enough frequency that we can assume the time to process a signal from the I/O expander is another round of 0.08 ms of processing time. USB 2.0 protocol indicates that the maximum speed for USB is 480Mbits/s. Again, this is fast enough that we can add another 0.08 ms. This means that a signal can be gathered in about 5 ms + 3*0.08ms = 5.24ms. This can be made faster by using a higher I^2C protocol.

3. Cost and Schedule

3.1. **Cost Analysis:**

3.1.i. Labor: (For each partner in the project)
$50/hour x 2.5 x 50 = $6250 TOTAL per person
Labor For Three Person Team = 3 * TOTAL per person = $18750

3.1.ii. Parts: Include a table listing all parts (description, manufacturer, part #, quantity and cost)

Description	Manufacturer	Part #	Quantity	Cost
Mechanical Key Switches	KBDFans	Aliaz Silent Switches	110	80
Diodes	ON Semiconductor	1N4148TR	101	10.10
Microcontroller Chip	Atmel	AT90USB1286	1	7.92
I/O Expander	Microchip Technology	MCP23018-E/SP	3	4.53
PCB	Bay Area Circuits	N/A	1	35

Key Caps	Tai Hao	N/A	116	50
TRRS Cables	Tensility International Corp	10-03211	3	28.80
TRRS Female Ports	CUI Devices	SJ-43514	4	4.24
USB A to USB mini Cable	AmazonBasics	IFRI	1	5.26
USB Mini Female Port	Hirose Electric Co Ltd	UX60-MB-5S8	1	0.99
Keyboard Stabilizers	KPrepublic	104WKLRS96	11	13.30

3.1.iii. Sum of costs into a grand total
Labor + Parts = $18750 + $240.14 = $18990.14

3.2. **Schedule:**

Week 1	Design Main PCB (C,D)	Design Fn PCB (C)	Order Unit test hardware (D)	Pick Components (C,F)	Create Keycode Library (F)
Week 2	Design NumPad PCB (C,F)	Design Tenkey PCB (C,D)	Order PCBs (C)	Order Components (D)	Create Library for Teensy and USB support (F)
Week 3	Hardware unit tests: switches, Teensy (C,D,F)	Software unit tests: MC is programmable I/O control (C,D,F)	Order PCBs (D)	Finish Keypress Function and Loop for Keyboard Matrix Scan (F)	
Week 4	Solder Main PCB (C)	Solder NumPad PCB (D)	Solder Teney PCB (F)	Design Housing (C,D)	Add Keyboard Layer Support (F)
Week 5	Solder Fn PCB (D)	Test USB Data Returned from Microcontroller (F)	Create Drivers for Programmability (F)	Print Housing (C)	
Week 6	Switch array processing (C,D)	I^2C Key processing (C,D,F)	Create Layout Presets (F)		
Week 7	Multikey processing	Modules operational	Full Assemble (C)		

4. Discussion of Ethics and Safety:

4.1. According to 1.2 of ACM code of conduct, we should design every part that could come in contact with the user to be safe to touch and interact with. Such contact points should not abrade the user in any way. There is an issue with the TRRS connectors: they will have both a 5 volt connection and a ground on the cable itself. This means that the user could potentially burn or shock themselves. The counter-measure to this will be a resettable fuse or breaker that will stop current above a certain threshold.

With using ACM 2.8, any firmware or drivers we utilize will be our own construction or open source material. ACM 2.9 gives protocols for making robust systems. We will in our keyboard choose parts that will not easily degrade. Since there are more connections than the usual amount from a keyboard we should take care that the user cannot easily accidentally maim the product near these hot zones. Our design for this keyboard is intended to be touched and used directly by humans. This means that we should adhere to ACM 3.3 by creating designs that are ergonomic and help the user have a more efficient experience, a crux of our background to this project.

5. Citations:

5.1. PJRC. (n.d.). Teensy Schematics. Retrieved February 22, 2020, from https://www.pjrc.com/teensy/schematic.html

5.2. L., D., Wyma, M., J., William, E., Herron, J., T., … Reed. (2015, February 26). Factors influencing the latency of simple reaction time. Retrieved from https://www.frontiersin.org/articles/10.3389/fnhum.2015.00131/full

5.3. CHERRY MX SPEED SILVER. (n.d.). Retrieved from https://www.cherrymx.de/en/mx-original/mx-speed-silver.html

5.4. I2C Info – I2C Bus, Interface and Protocol. (n.d.). Retrieved from https://i2c.info/

5.5. Bates, B. M., Dezmelyk, R., Ingman, R., & Lieb, R. et al. (n.d.). Universal Serial Bus Hid Usage Tables (Vol. 1.12).

5.6. Atmel. (n.d.). 8-Bit Atmel Microcontroller with 64/128 Kbytes of Isp Flash and Usb Controller.

5.7. Microchip Technology. (n.d.). MCP23018/MCP23S18 16-Bit I/O Expander with Open-Drain Outputs.

5.8. Trybus, M. (2013, September 2). komar's techblog. Retrieved February 27, 2020, from http://blog.komar.be/how-to-make-a-keyboard-the-matrix/