This is an independently funded project, designed to create a portable vaccine dispenser which runs off of solar power. This product could have applications in delivering preventative healthcare to rural and underserved population in areas with unreliable power.

Solar Powered Vaccine Dispenser

06/13/2020

Michael Hernandez

Contents

Introduction

Vaccines provide acquired immunity to a targeted infectious disease. They are often credited with numerous public health and economic benefits, particularly in developing countries [1]. Despite this, significant challenges still exist in distribution particularly in rural areas of poor countries. Missed opportunities for vaccination correlate significantly with a reduction in income [2].

One of the contributing factors preventing efficient vaccine distribution in rural areas of impoverished countries is gaps in the cold supply chain, particularly in areas without reliable electricity. Currently only 28% of healthcare facilities in sub-Saharan Africa have access to reliable electricity [3]. Cold storage is an important issue because vaccines need to be stored at or around 4.5 degrees Celsius (40 degrees Fahrenheit) or lower [4]. While portable vaccine storage devices are currently on the market, they usually rely on external power devices such as an automobile auxiliary power outlet, or else utilize temporary cooling mechanisms such as dry ice.

Objectives

The main objectives of this project are as follows:

- Design a vaccine storage device that can maintain a constant cold temperature in changing ambient temperature conditions
- Maintain an electricity supply without the need for power outside of the unit
- Include a user interface which it is possible to input a username and password to prevent unauthorized use
- Include security features such as requiring a password to access the unit, as well as to keep a stored log of all usage

System Overview

In order to get this unit functioning properly, nine systems must work together.

Solar Panel

For this project a solar panel capable of delivering 12 volts to a battery manufactured by ECO-WORTHY was selected. For this project a panel capable of generating 25 watts of power was selected. This power for a solar panel was selected because at least 24 watts of power is necessary to properly power the cooling module. This is derived from the power equation.

$$P = \frac{V^2}{R}$$

Since the battery that is used is capable of providing 12 volts, and the selected Peltier module in the cooling system has a resistance of 6Ω, it is calculated that 24 watts of power is required

$$P = \frac{12V^2}{6\Omega}$$

$$P = 24\,Watts$$

Charge Controller

For this project a charge controller manufactured by GHB was selected. A good charge controller will be able to automatically switch power consumption and storage effectively. When the voltage from the solar panel is high (i.e. it is in sunlight and generating power) then the charge controller will direct power from the solar panel directly to the connected device without using battery power. When the solar panel is not generating electricity then the charge controller will begin siphoning power off of the connected battery. Any extra power not used by the connected device will be used to recharge the battery.

Microcontroller

This project utilized an ATMEGA2560 microcontroller using C++ as the programming language. All programming was done in the Arduino IDE. 1,234 lines of code were written. The microcontroller directly controls the electronic lock, data storage, user interface, and dispensing mechanism. It indirectly controls the cooling unit by setting the resistance value in a digital potentiometer which is used as a reference voltage for a comparator in conjunction with a TMP36 module.

Electronic Lock

An electronic solenoid lock by DFRobot was used as the locking mechanism. This lock unlatches when an electric current is running across a solenoid, causing it to contract and unlatching the metal plate holding the lock in place. This lock requires 12 volts and an input current of 1.5 amps to operate properly. The lock should not be turned on for more than 5 seconds at one time otherwise the solenoid controlling the lock could be damaged.

Cooling Unit

The cooling unit utilized a CP4055485 Peltier module by CUI devices. This device has a series load resistance of 6Ω. In order to determine if the cooling module needs to be turned on or off, the current across the module is controlled by a relay. The logic signal for the relay is determined by a LM311 digital comparator which accepts two input voltages for comparison. The first input voltage comes from a TMP36 module. The prototype built on a breadboard uses one TMP36 module, but the final design takes an average from three different modules. The output voltage from the TMP36 is compared against a reference voltage set by a digital potentiometer. This potentiometer's value is set by the microcontroller at startup and is fixed in order to conserve computing power.

Data Storage

In order to store data on device usage, as well as to store information about usernames and passwords, an SD card is used. Every time the dispensing unit or the electronic lock is used, a record of it is stored on the SD card. In order to reference the time and date that the device is used, a real time clock and calendar (RTCC) is referenced to use as a timestamp.

User Interface

The prototype will use three prefrail devices for the user interface. The first is a basic 16x2 character LCD by Sparkfun for displaying information. The second piece is a phone style matrix keypad by Addafruit for data entry. The final piece is a joystick by WGCD for navigating through menus. The final design will use a TFT touchscreen for user input and data display.

Dispensing Mechanism

The dispensing mechanism is what will be used for moving one vaccine vial from the interior of the device to the exterior for removal and subsequent use. A 100mm T16-S Mini Track Actuator by Actuonix was selected for use in this project. A track actuator was chosen over a linear actuator because horizontal motion places extra stress on linear actuators thus reducing the lifespan of the device. Track actuators are similar to linear actuators however they are designed for horizonal motion.

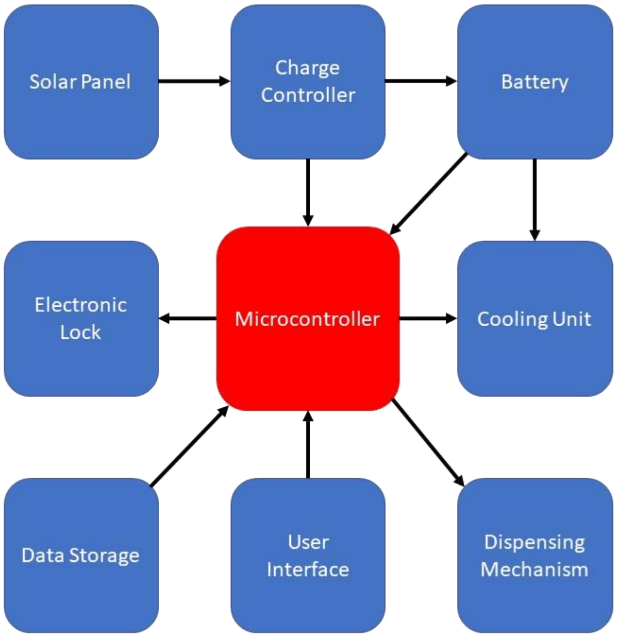


Figure 1 - Block diagram of the system. In this system there is one microcontroller and 8 prefrail systems that are necessary for this unit to function properly.

Detailed View of the Cooling Unit

Since vaccines need to be regulated at a cool temperature, at or below 4°C, particular care was put into designing the cooling unit. In order to cool the device, a CP40555485 Peltier module by CUI Devices was selected.

A Peltier module is an NPN type device which can be used to transfer heat from one side of the device to another. When a current is running across the device one side of the device becomes cold while the other side becomes hot. The Peltier effect is the opposite of the Seebeck effect which states that if different metals are connected in two separate places, and the intersections are kept at different temperatures, then a potential difference between the metals at the junctions will result. The reverse is also true; when a potential difference is created across different metals, a temperature gradient is created. Peltier modules are generally constructed of Bismuth and Telluride to achieve this effect [5].

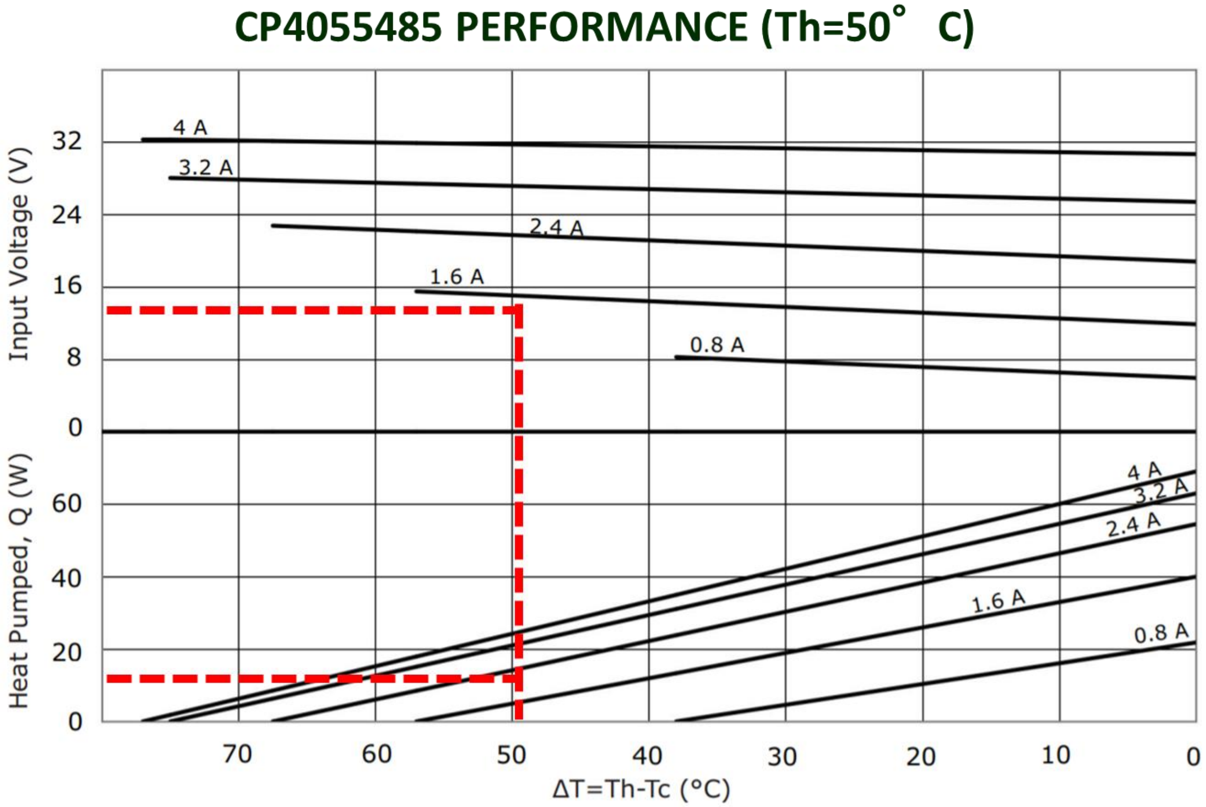


Figure 2 - The performance of the CP4055485 at 50°C can be derived from this graph. By making the assumption that removing 10 watts of heat is significantly higher than what will be required in extreme conditions. Operating at an environmental temperature of 50°C (122°F), this unit will continue to operate properly and be able to maintain a constant internal temperature of 4°C or less.

In order to conserve power as well as to prevent the internal contents from accidently freezing, the Peltier module needed to be turned on and off as internal conditions necessitate. In order to accomplish this a digital comparator was utilized to compare the output voltage from a TMP36 temperature sensor module. In order to adjust the holding temperature, an MCP4131 digital potentiometer by Microchip was used to set a reference voltage. The reference voltage is the same as the output voltage from the TMP36 when the it is at a known temperature. When the voltage from the TMP36 falls below the set voltage, this would indicate that the current temperature is lower than the predetermined temperature. Therefore, the voltage from the TMP36 is lower than the reference voltage from the potentiometer. When this occurs the output from the digital comparator turns off. The output on the digital comparator serves as the logic for a relay which connects the Peltier module to the 12-volt input.

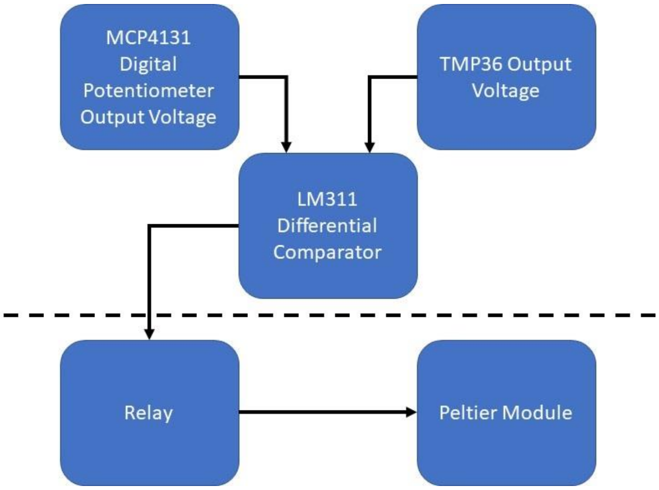

MCP4131 Digital Potentiometer Output Voltage
TMP36 Output Voltage
LM311 Differential Comparator
Relay
Peltier Module

Figure 3 - Block diagram demonstrating how the five different components of the cooling mechanism work together to maintain a constant temperature.

Changing the reference voltage from the potentiometer will change the holding temperature. The digital potentiometer uses transistors to adjust the voltage rather than a wiper mechanism like an analog potentiometer does, it can only change values incrementally in 129 steps. Furthermore, the output voltage from the TMP36 module is quite low, on the order of millivolts. Since the digital potentiometer requires a 5-volt reference voltage, the resolution of the output voltage from the device does not allow for setting the correct temperature. In order to solve this problem, a resistor was placed in between the potentiometer and the comparator. The value of the resistor needed was experimentally determined to be 33.9MΩ. The TMP36 has a scaling factor of 10 mV/°C. Therefore, it was determined that the reference voltage must be 540 mV to correspond to a temperature of 4°C [6].

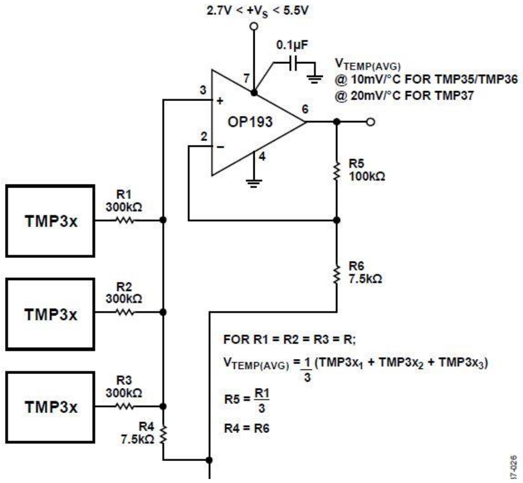


Figure 4 - Recommended TMP36 circuit to average the output voltages from three units [6].

In order to attain a more accurate reading, the final design will utilize the average output from three TMP36 modules. As per the recommended circuit provided in the TMP36's datasheet, the output from all three modules should have a 300KΩ resistor to reduce the output voltage. All three modules will be fed into an OP193 operational amplifier by Analog Devices. The output voltage from the op-amp will be the average from all three TMP36 modules. The finalized design utilized a separate PCB which was connected to the main circuit board using board to wire Molex connectors. This configuration allows for free placement of the TMP36 modules. Therefore, they may be placed anywhere in the cooling column of the finalized device.

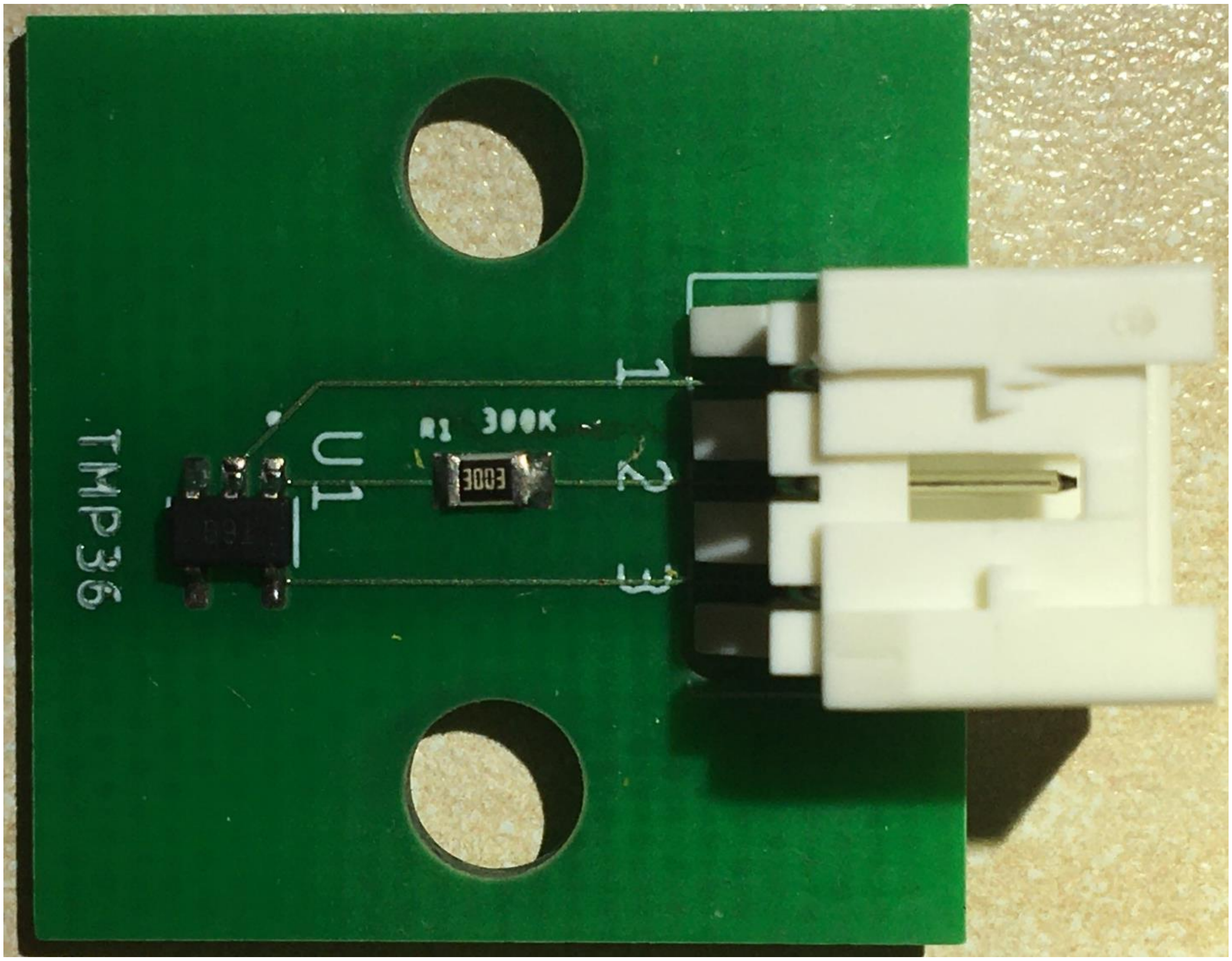


Figure 5 - Custom designed PCB for this project with a TMP36 IC. A 300Ω chip resistor has been placed immediately after the output of the TMP36. A 534260310 board to wire connector by Molex has been affixed to the board to allow for free placement within the final device. The size of this board including the protruding Molex connector is 22.8 mm tall x 27.7 mm wide.

The MCP4131 digital potentiometer is an SPI device. The device is an active low, meaning that it will be selected for if the chip select pin is set to zero volts. This is a convenient feature because it helps to save power – the chip only needs to be selected whenever the output voltage on the potentiometer needs to be adjusted. The device is very easy to address; the device's address is 0x00. In order to set the value of the potentiometer, an integer is stored in the volatile memory of the digital potentiometer. The integer may be anywhere between 0 and 128, 0 would set the output voltage to zero volts, while 128 would set the output voltage to 5 volts. Pins 5 – 7 are the potentiometer pins. Pins 5 and 7 are the terminal pins, while pin 6 is the wiper terminal pin.

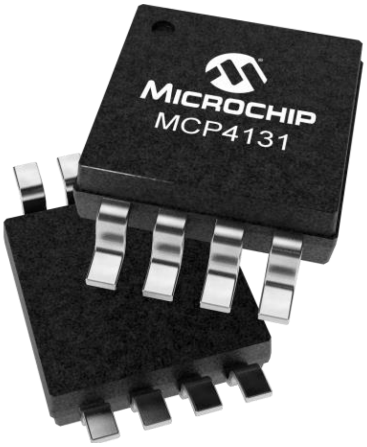


Figure 6 - Surface Mount MCP4131

Timekeeping

In order to maintain accurate usage information a timeclock is required. The Pmod RTCC module by Digilent was selected. This module uses the I^{2}C protocol. In order to set the date and time, first the variable needs to be addressed, then the variable must be written using a hexadecimal value. The module is capable of storing values for the year, month, date, day of the week, hour, minute, and second. The module automatically adjusts for leap years but not for daylight savings time. It is capable of storing two alarms, in which it sends a signal which can be used to activate an interrupt service routine on a microcontroller or prefrail device. The module does have a built-in slot for a battery backup, but this must be enabled by addressing it to the module. This can all be done manually, however Digilent also has made a library publicly available which can be used within the Arduino IDE with only minor changes. The C++ library commands need to be changed from Wire.send to Wire.write, while the Wire.receive should be changed to Wire.read in order to be compatible with Arduino.

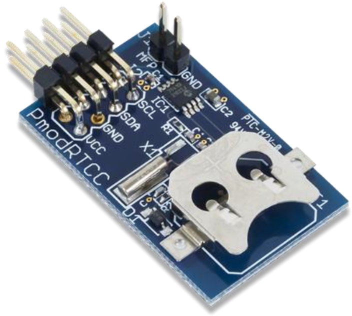


Figure 7 - Pmod RTCC by Digilent

SD Card

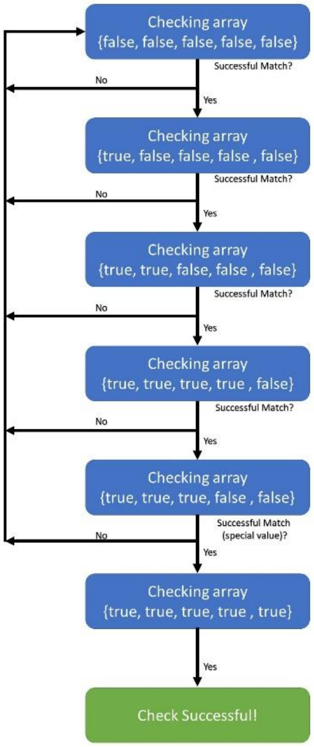

Checking array
{false, false, false, false, false}
Successful Match?
No
Yes
Checking array
{true, false, false, false , false}
Successful Match?
No
Yes
Checking array
{true, true, false, false , false}
Successful Match?
No
Yes
Checking array
{true, true, true, true , false}
Successful Match?
No
Yes
Checking array
{true, true, true, false , false}
Successful Match (special value)?
No
Yes
Checking array
{true, true, true, true , true}
Yes
Check Successful!

Figure 8 - Match checking algorithm

In order to store data about usage, as well as to store information about each user and their password, an SD card needs to be used within this system. Whenever the electronic lock is used, a record of the usage will be written to the SD card. This information includes the username of the individual who accessed it as well as the date and time. The system is to include a method of accessing user information as well, however this has yet to be incorporated into the overall system. This feature has been developed in isolation however. The algorithm works by scanning the entire .txt file on the SD card for a matching value in an array. Each username and password is preceded by and succeeded by a designated character. The designated character for the username is '%' while the character for a password is '$'. A checking array is one character longer than the username or password array. Each value in the array is a Boolean value. If a value is found to match then the Boolean value is changed from false to true. If all four values are true, and the special character is also true, then a flag is sent to indicate a successful match.

The SD card module which was selected for this project was a prefrail PCB which interfaced with the rest of the system by using the SPI protocol. The module was connected to the MISO, MOSI, and SCK pins. The SD card module also needed to be activated by setting the chip select pin voltage to high.

Dispensing Mechanism

A dispensing mechanism is necessary for this project for a few reasons. It ensures that the majority of the vaccines would stay in the cold compartment at any given time. Dispensing only one unit at a time helps to minimize heat transfer whenever the unit is exposed to ambient temperatures. Additionally, it will help to minimize loss of additional vials (i.e. a vial currently in use is broken, lost, or stolen) by only having access to the minimum required number of vials at any given time.

In order to accomplish this, a track actuator by Actuonix was selected. Originally a linear actuator was considered, however a track actuator was selected over a linear actuator due to the horizontal motion involved. When a liner actuator is used in a horizontal motion it places additional stress on the device reducing its lifespan. Track actuators can handle more force than linear actuators can because of their predefined path for the force as they have additional structural support [7].

The selected track actuator is capable of running up to 12 volts, however it is able to still operate with less power, just at a slower speed. The track actuator uses a simple DC motor, which is capable of only running in one direction. Without rewiring the device, it is only possible to move the motile mount in one direction. In order to circumvent this problem and H-bridge was introduced to the system.

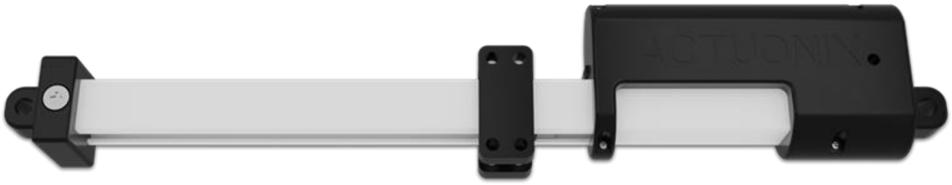


Figure 9 - 100mm T16-P Mini Track Actuator by Actuonix

An H-bridge is a commonly used circuit in robotics which can be used to reverse the polarity of a circuit. It makes use of four or more transistors. Two transistors are in series while two are in parallel. By manipulating the order in which the transistors are turned on or off the H-bridge changes the polarity of the circuit without requiring any rewiring of the circuit. The H-bridge is named because of the H shaped pattern the transistors make while changing polarity [8]. For prototyping on a breadboard an SN754410 by

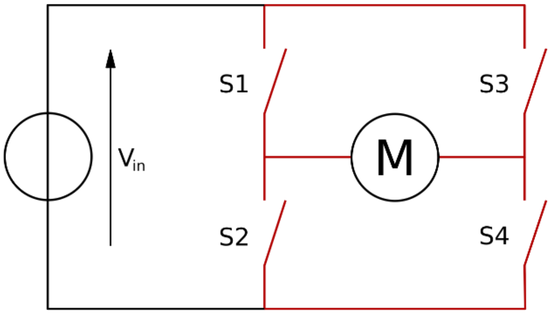


Figure 10 - H-bridge diagram, the motor is placed at M

Texas Instruments was chosen due to its ease of use with a breadboard, however in the final design an LV8548MC H-bridge by ON Semiconductor was selected due to its lower cost. Wile both of these components are H-bridges they operate in slightly different ways. The SN754410 uses pulse width modulation in order to adjust the voltage delivered to the motor, while the LV8548MC is hard wired to a set driving voltage. Additionally, the SN754410 requires a chip select pin in addition to the high voltages for the collector region input on the internal NPN devices to be set to high in order to be active. The LV8548MC does not and will simply function when the appropriate MOSFET gates are turned on.

User Interface

For purposes of prototyping, a basic user interface was designed. This user interface requires three components in order to function properly. The three components are: an LCD screen, a keypad, and a joystick. The LCD screen displays information about the current working menu, while the keypad is used to input information and to confirm desired menu selection. The joystick is used to navigate through menus.

Menu navigation made use of global variables along with multiple switch statements within a loop to select a menu. First the loop will reference the global variable to set the menu, then it will check for a keypress to see if the user has entered a sub menu. If the key was pressed then the menu variable is changed, if the key was not pressed then the menu variable is not changed. The joystick selector works in much the same way; depending on the set of menus that are currently being manipulated a global variable is set, and then referenced to display on the screen. The two systems work in conjunction with each other – The two variable values are added together and the appropriate menu is displayed to the user.

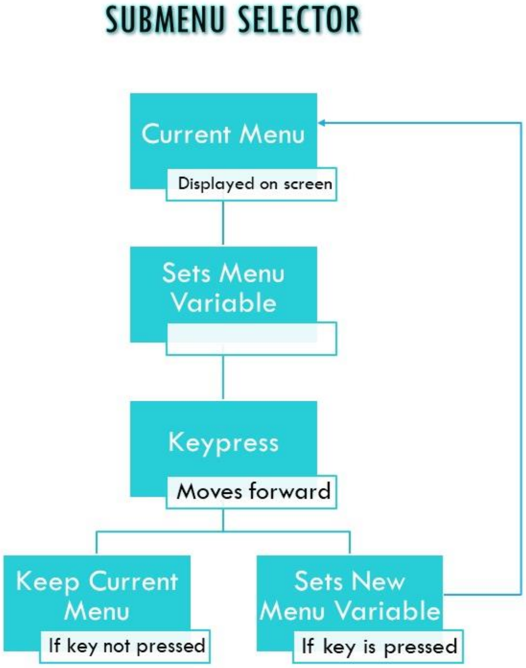


Figure 11 - Menu selection algorithm

PCB Design

A PCB was designed to house most of the electronic components for this project. PCB123 was used as the design software; it is a professional grade software but does not require licenses to run. Instead Sunstone Circuits, the company which makes PCB123 makes a profit when you place an order through their fab house. Gerber files can still be purchased to send to another fab house for $100.

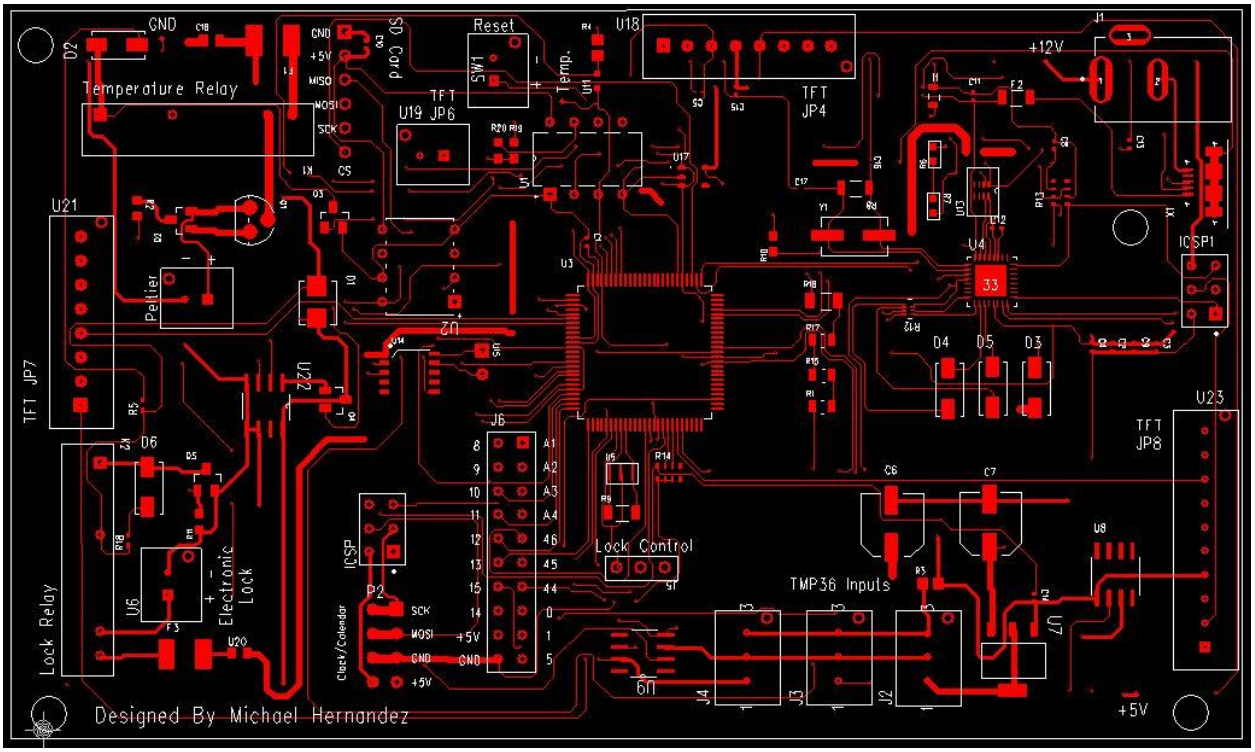


Figure 12 - Top Layer of designed PCB

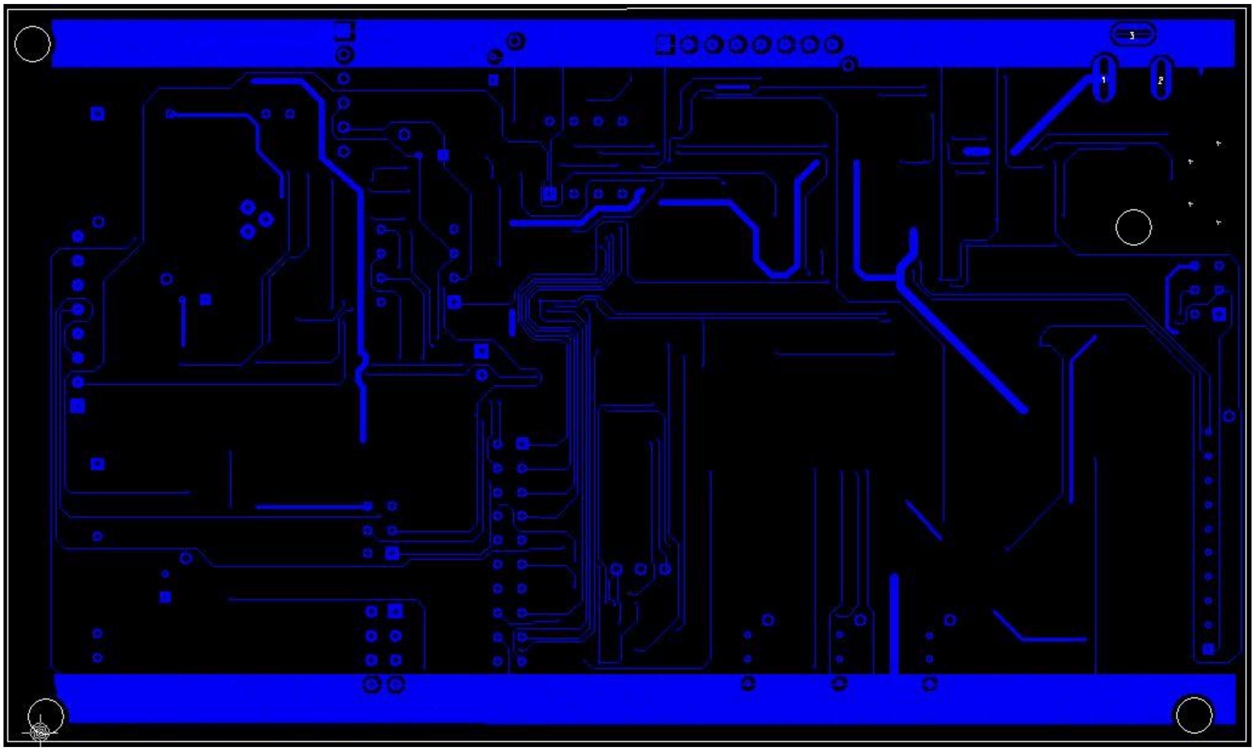


Figure 13 - Bottom layer of designed PCB

The PCB was designed around using a TFT touchscreen for the user interface rather than an LCD and keypad like what the prototype uses. However, header pins were included which allow access for +5V and ground, as well as access to four analog pins and 11 digital pins. A power plane was included on the bottom edge of the PCB as shown in Figure 13, along with a ground plane along the top edge as shown in the same figure. The TFT to board connection was designed to be connected using Molex connectors and wires. Each wire leaving the PCB will be connected to header pins on the TFT.

The reset button needs to be accessible to the end user in the event of system failure so a separate PCB was designed with this need in mind. This PCB will connect to the main board using a Molex connector. The reset button will be electrically connected in the same position as it would be in an Arduino MEGA. The reset button is on one side of the PCB while the connectors are on the opposite side of the PCB so that only the reset button is accessible to the user.

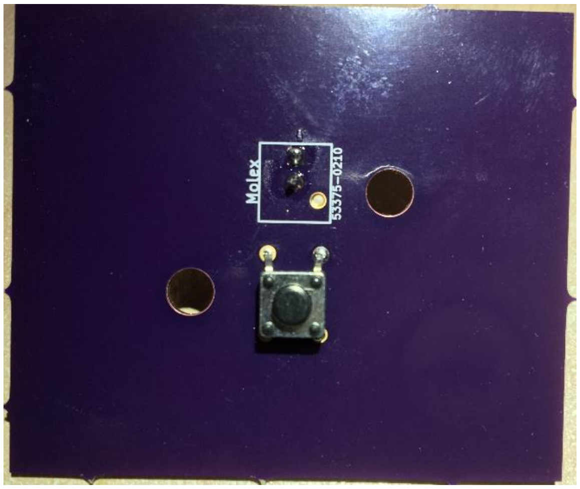


Figure 14 - Reset button PCB (front)

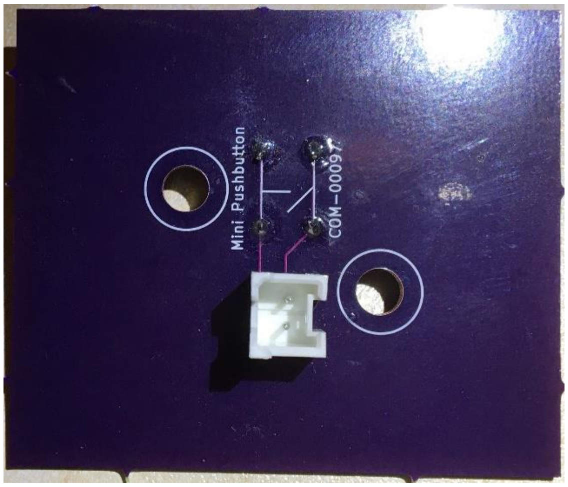


Figure 15 - Reset button PCB (reverse)

The other prefrail devices which need to be wired onto the board are the electronic lock, the Peltier module, and the TMP36 modules. These will also be connected using Molex connectors. The linear actuator, RTCC PCB, and SD card PCB will all be connected directly onto the board using header pins.

The ATMEGA2560 microcontroller along with the necessary components to make it run were included in the PCB as well. Arduino's open source hardware schematics were used as a reference for the design. Most of the header pins that are available on a standard Arduino MEGA were omitted in the redesign, since most pins were hardwired to their relevant components. Data traces were set to the system default of 0.2032mm, this also happens to be the width of the pins leaving the microcontroller. Whenever possible, the width of power traces was set to 0.7 – 1 mm, to decrease the resistance encountered, particularly along the +12-volt traces. Not all power traces were able to be widened appropriately due to the limited space on the board. In such cases, some power traces were set to 0.5mm in order to get them to fit properly alongside other traces. Whenever possible, systems were grouped together on the board. The Arduino hardware was placed on the board and routed first, then the prefrail systems were added to the board and routed.

The board was sent in for fabrication at Sunstone Circuits, and was subsequently sent to Screaming Circuits for assembly.

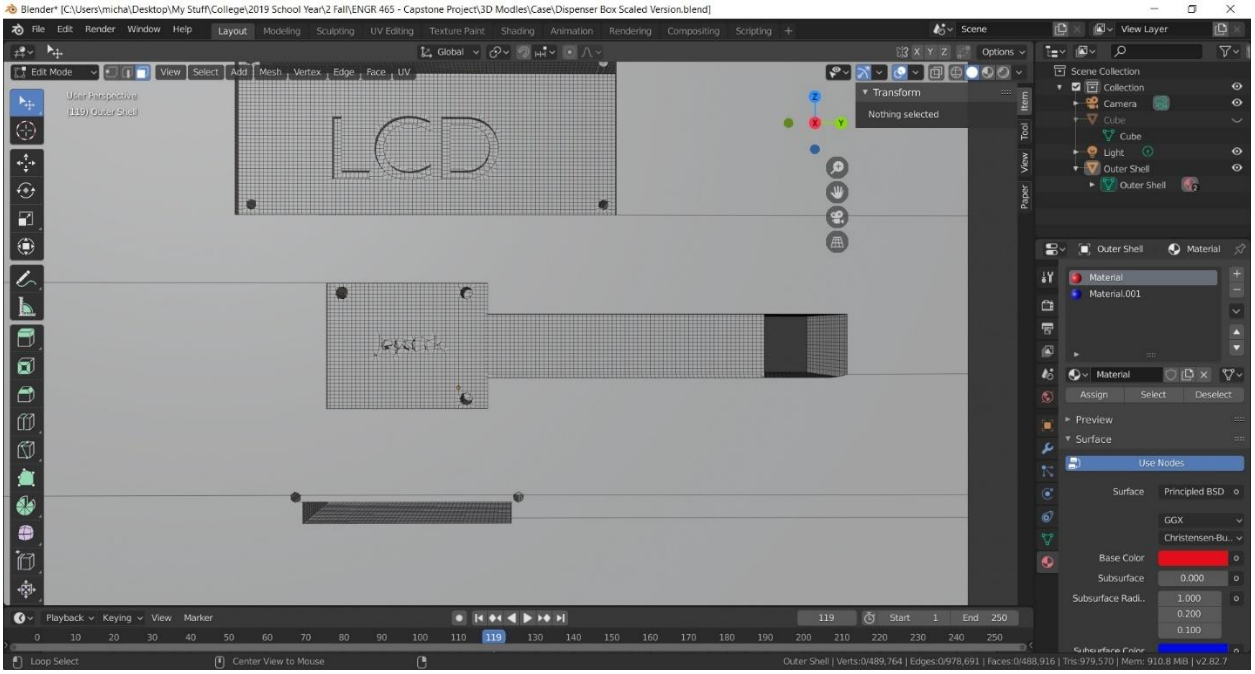
Case Design

The case to house this entire project was designed using Blender, an open source 3D modeling software capable of exporting .stl files suitable for 3D printing.

The column which is to be used for housing the vaccines internally was placed roughly in the middle of the unit to allow for the greatest possible insulation. It is directly connected to the track actuator which is to be affixed on the bottom of the case.

A cutout for the PCB was designed along the back side of the case. This enabled both the Peltier module as well as the user interface peripherals to be wired to the PCB easily.

The user interface devices were all placed on the same side of the device. For all devices, conduit cutouts were designed to connect the device to the PCB. This allows for wires to be ran through the inside of the device. In doing so the device is both less likely to be damaged and is more aesthetically pleasing.


Blender* [C:\Users\micha\Desktop\My Stuff\College\2019 School Year\2 Fall\ENGR 465 - Capstone Project\3D Modles\Case\Dispenser Box Scaled Version.blend]

Surface: Principled BSD
GGX
Christensen-Bu..
Base Color
Subsurface: 0.000
Subsurface Radi..: 1.000, 0.200, 0.100
Subsurface Color

Outer Shell | Verts:0/489,764 | Edges:0/978,691 | Faces:0/488,916 | Tris:979,570 | Mem: 910.8 MiB | v2.82.7

Figure 16 - A cross section of the case showing conduit which allows for wires to be laid connecting the joystick to the PCB

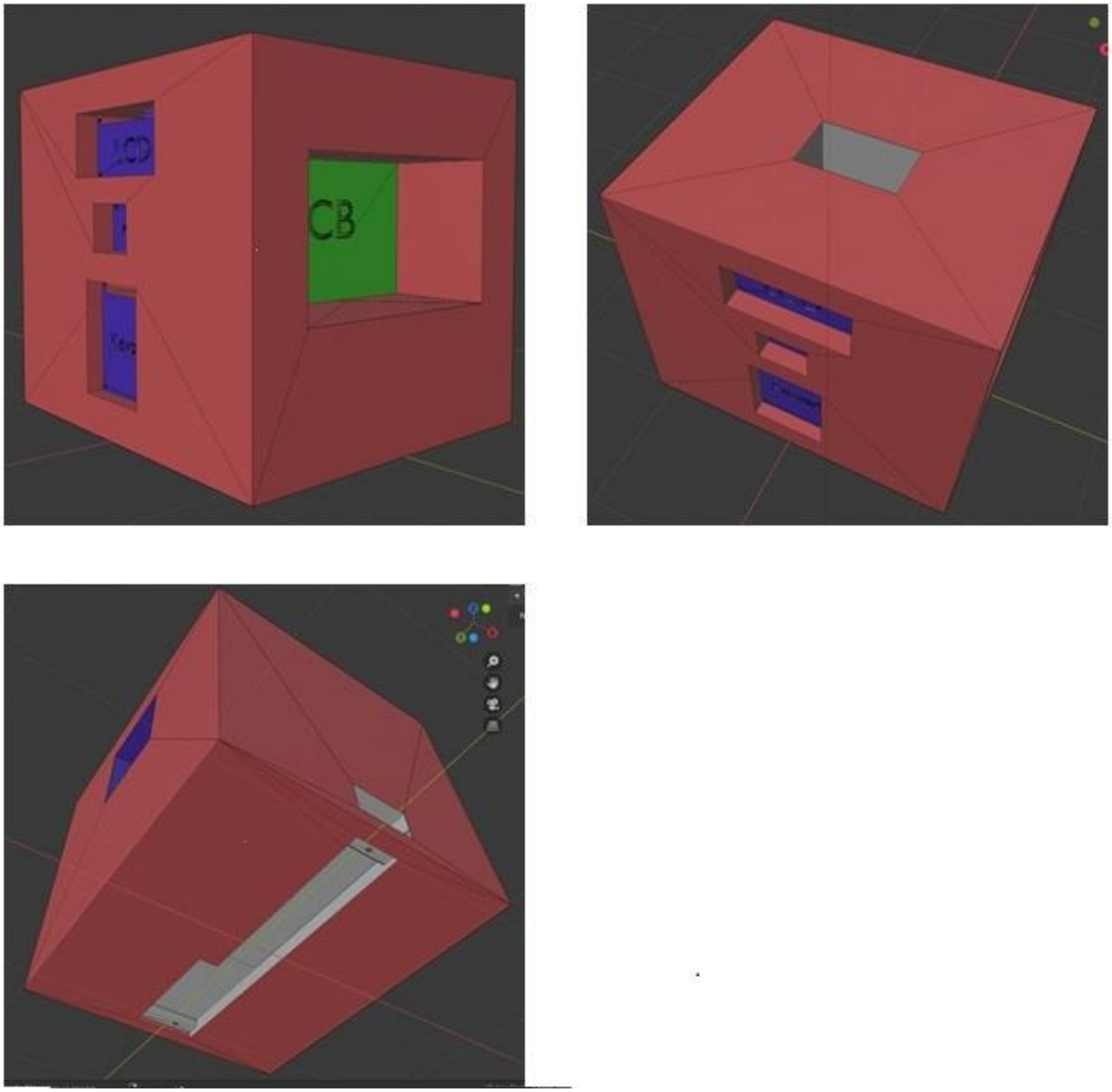
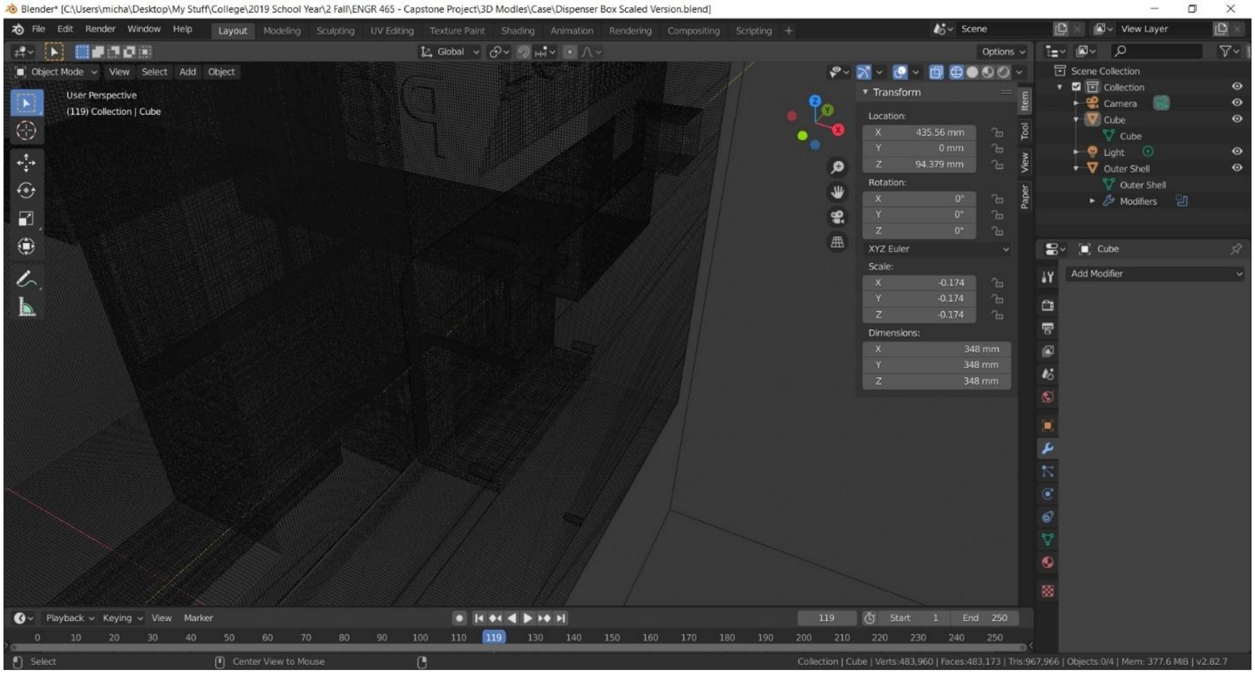


Figure 17 - A case was designed to house the various components. The LCD, keypad, and joystick housing and PCB mount are visible in the upper left image. The upper right image shows the storage column, the bottom image depicts where the track actuator is to be mounted. The UI devices are in blue, the PCB mounting area is in green, and the track actuator and vaccine holding column are in white.

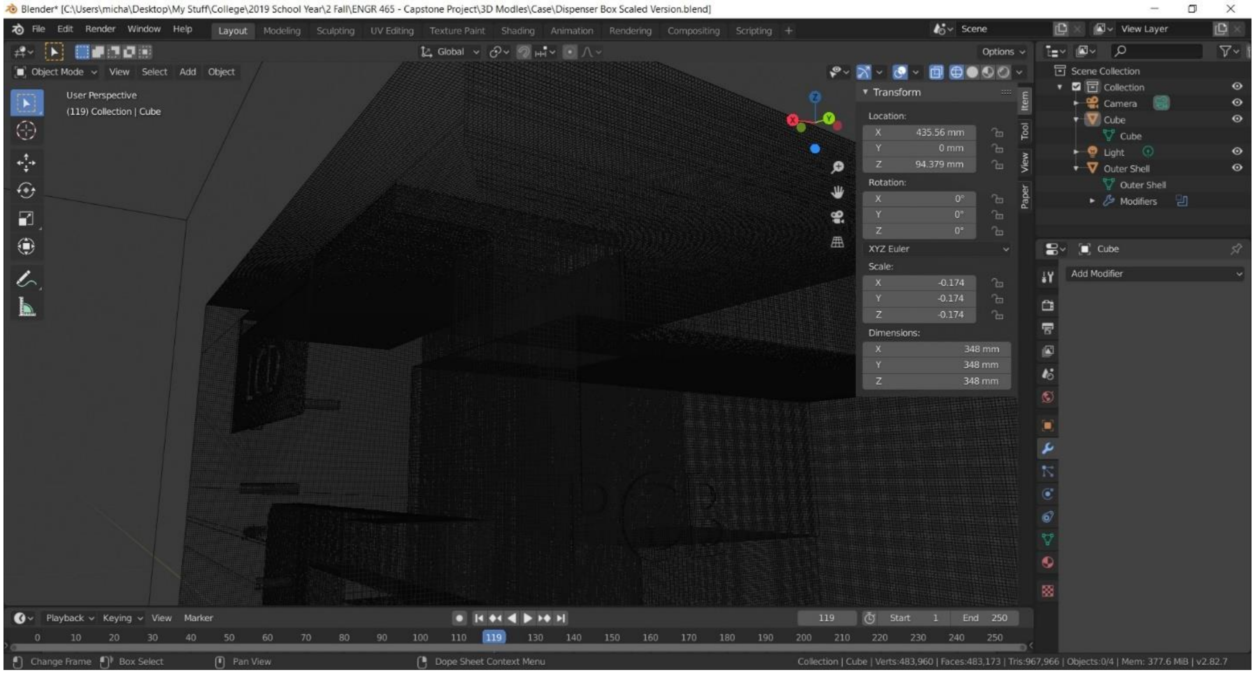

Blender* [C:\Users\micha\Desktop\My Stuff\College\2019 School Year\2 Fall\ENGR 465 - Capstone Project\3D Modles\Case\Dispenser Box Scaled Version.blend]

User Perspective
(119) Collection | Cube

Transform
Location: X 435.56 mm, Y 0 mm, Z 94.379 mm
Rotation: X 0°, Y 0°, Z 0°
XYZ Euler
Scale: X -0.174, Y -0.174, Z -0.174
Dimensions: X 348 mm, Y 348 mm, Z 348 mm

Collection | Cube | Verts:483,960 | Faces:483,173 | Tris:967,966 | Objects:0/4 | Mem: 377.6 MiB | v2.82.7

Figure 18 - Wireframe image of the case showing the conduit connecting the keypad to the PCB

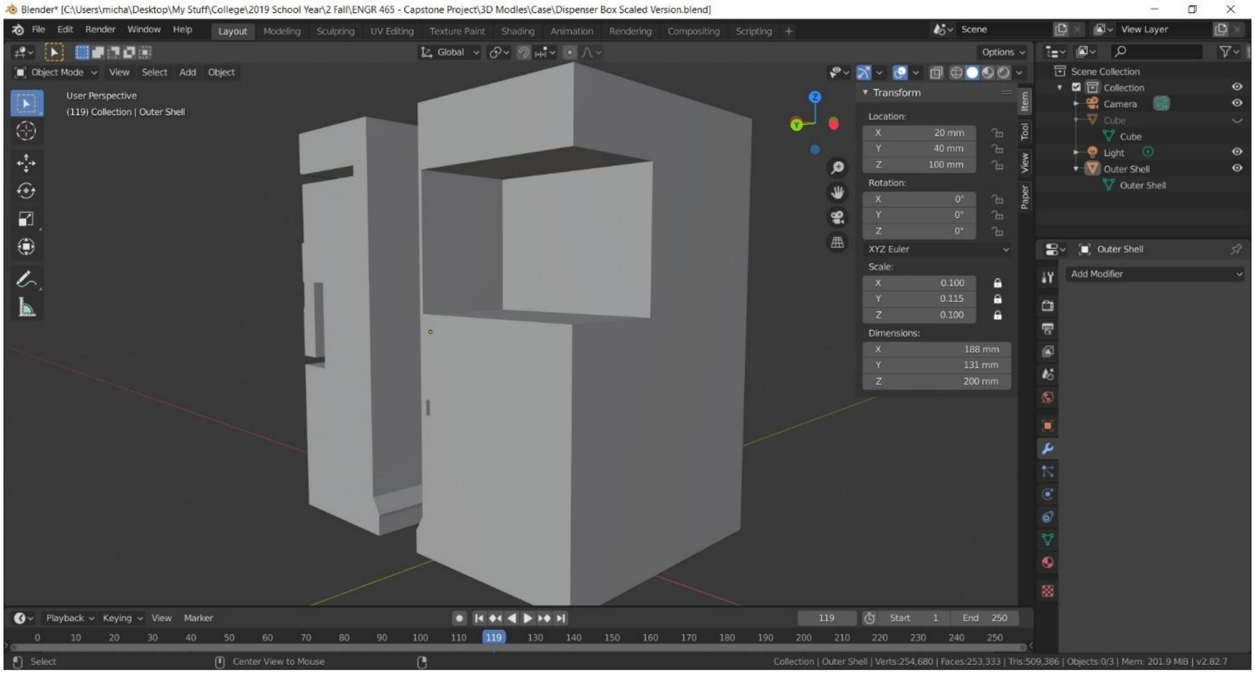

Blender* [C:\Users\micha\Desktop\My Stuff\College\2019 School Year\2 Fall\ENGR 465 - Capstone Project\3D Modles\Case\Dispenser Box Scaled Version.blend]

User Perspective
(119) Collection | Cube

Transform
Location: X 435.56 mm, Y 0 mm, Z 94.379 mm
Rotation: X 0°, Y 0°, Z 0°
XYZ Euler
Scale: X -0.174, Y -0.174, Z -0.174
Dimensions: X 348 mm, Y 348 mm, Z 348 mm

Collection | Cube | Verts:483,960 | Faces:483,173 | Tris:967,966 | Objects:0/4 | Mem: 377.6 MiB | v2.82.7

Figure 19 - Wireframe image of the case showing the conduit connecting the LCD to the PCB



Blender* [C:\Users\micha\Desktop\My Stuff\College\2019 School Year\2 Fall\ENGR 465 - Capstone Project\3D Modles\Case\Dispenser Box Scaled Version.blend]

User Perspective
(119) Collection | Outer Shell

Transform

Location:
X 20 mm
Y 40 mm
Z 100 mm

Rotation:
X 0°
Y 0°
Z 0°

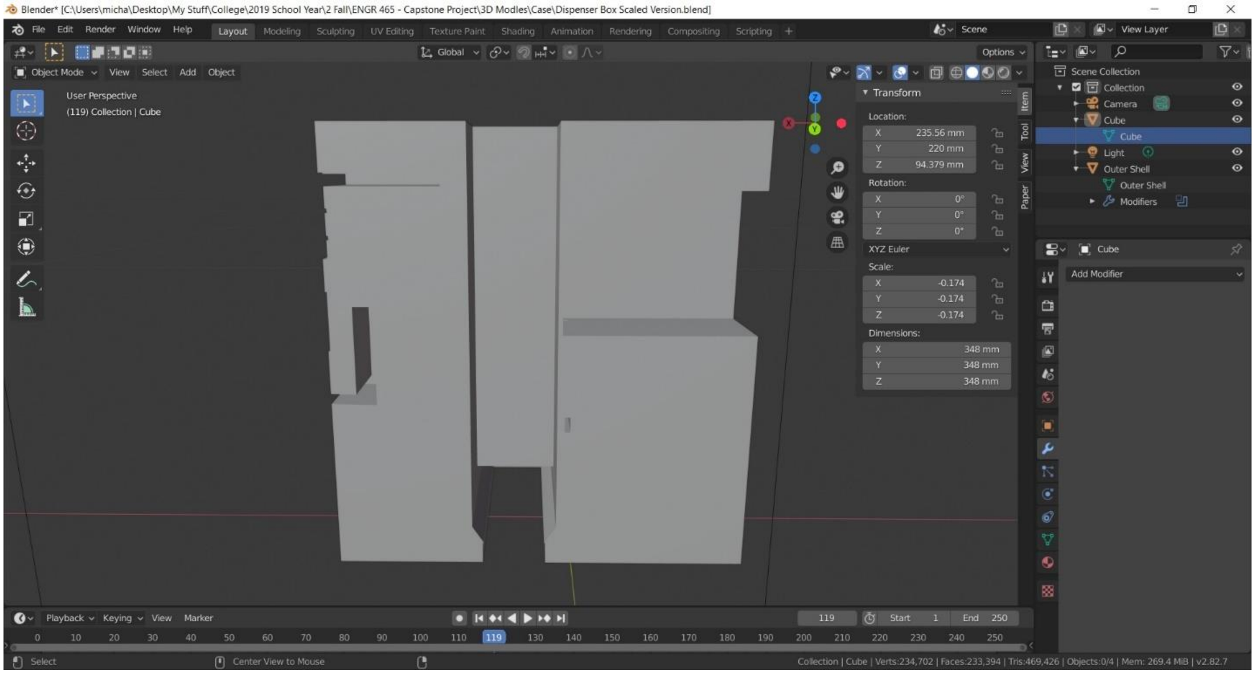
XYZ Euler

Scale:
X 0.100
Y 0.115
Z 0.100

Dimensions:
X 188 mm
Y 131 mm
Z 200 mm

Collection | Outer Shell | Verts:254,680 | Faces:253,333 | Tris:509,386 | Objects:0/3 | Mem: 201.9 MiB | v2.82.7

Figure 20 - Cross section showing how the Peltier module is placed adjacent to the vaccine holding column.



Blender* [C:\Users\micha\Desktop\My Stuff\College\2019 School Year\2 Fall\ENGR 465 - Capstone Project\3D Modles\Case\Dispenser Box Scaled Version.blend]

User Perspective
(119) Collection | Cube

Transform

Location:
X 235.56 mm
Y 220 mm
Z 94.379 mm

Rotation:
X 0°
Y 0°
Z 0°

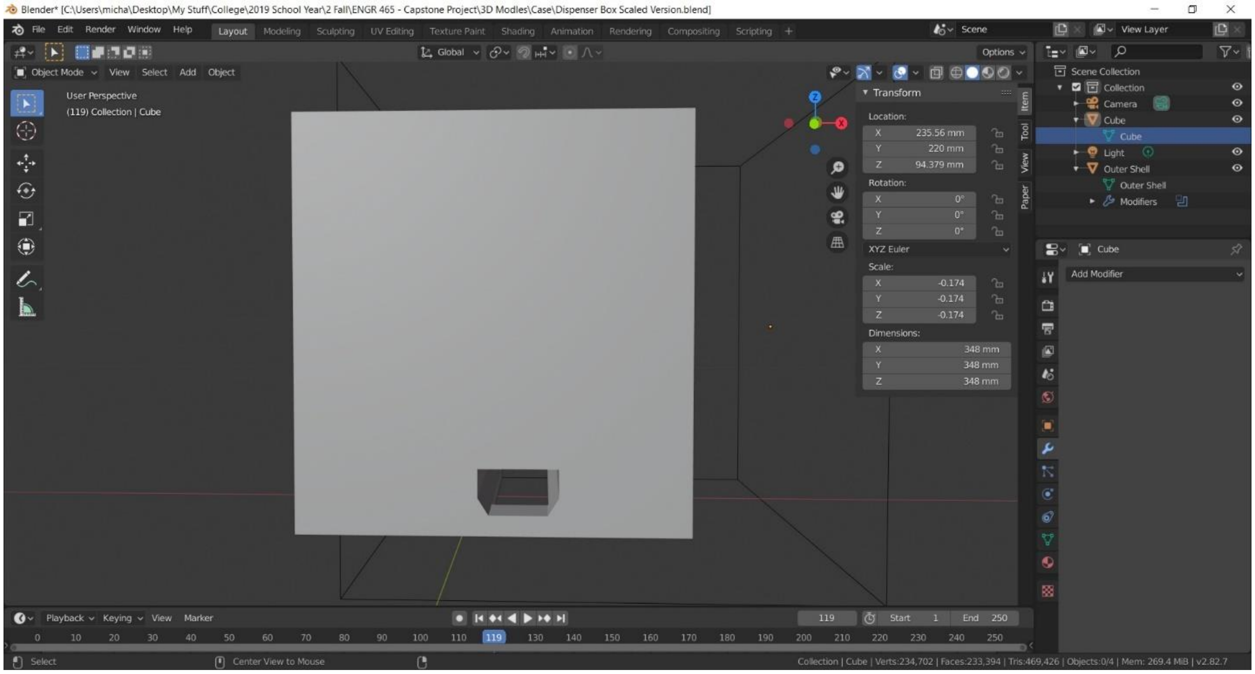
XYZ Euler

Scale:
X -0.174
Y -0.174
Z -0.174

Dimensions:
X 348 mm
Y 348 mm
Z 348 mm

Collection | Cube | Verts:234,702 | Faces:233,394 | Tris:469,426 | Objects:0/4 | Mem: 269.4 MiB | v2.82.7

Figure 21 - Cross section showing the vaccine holding column. Note the cutout on the bottom for the track actuator.



Blender* [C:\Users\micha\Desktop\My Stuff\College\2019 School Year\2 Fall\ENGR 465 - Capstone Project\3D Modles\Case\Dispenser Box Scaled Version.blend]

User Perspective
(119) Collection | Cube

Transform

Location:
X 235.56 mm
Y 220 mm
Z 94.379 mm

Rotation:
X 0°
Y 0°
Z 0°
XYZ Euler

Scale:
X -0.174
Y -0.174
Z -0.174

Dimensions:
X 348 mm
Y 348 mm
Z 348 mm

119 Start 1 End 250

Collection | Cube | Verts:234,702 | Faces:233,394 | Tris:469,426 | Objects:0/4 | Mem: 269.4 MiB | v2.82.7

Figure 22 - Exit hole for vaccines, the user will be able to retrieve a dispensed vaccine from this slot.

Future Steps

While this device is an acceptable prototype, it still must undergo several revisions before it can be a usable product.

The next step before any additional progress can be made is to incorporate the use of a TFT touchscreen rather than the current UI devices. Additional debugging is also necessary, such as the error encountered when writing a log of user information to the SD card; the information is written multiple times while the routine is running rather than only having a single log of the event.

Another improvement that must be made is to change all of the high-level abstraction in the code to bitwise manipulation. This will increase the speed at which the code runs as well as take up less space on the microcontroller.

Finally, while this device has been successfully prototyped in a lab, field testing is required in order to ensure that the device can perform well in extreme environments. The device must be able to be able to withstand physical shocks without breaking the internal contents, and must perform reliably after repeated use. In order to accommodate these changes, the PCB and case will both need to be redesigned to accommodate the final product.

Appendices

Bill of Materials

The per unit cost of this device is **$881.88**.

This cost is expected to decrease significantly if these units were manufactured in bulk.

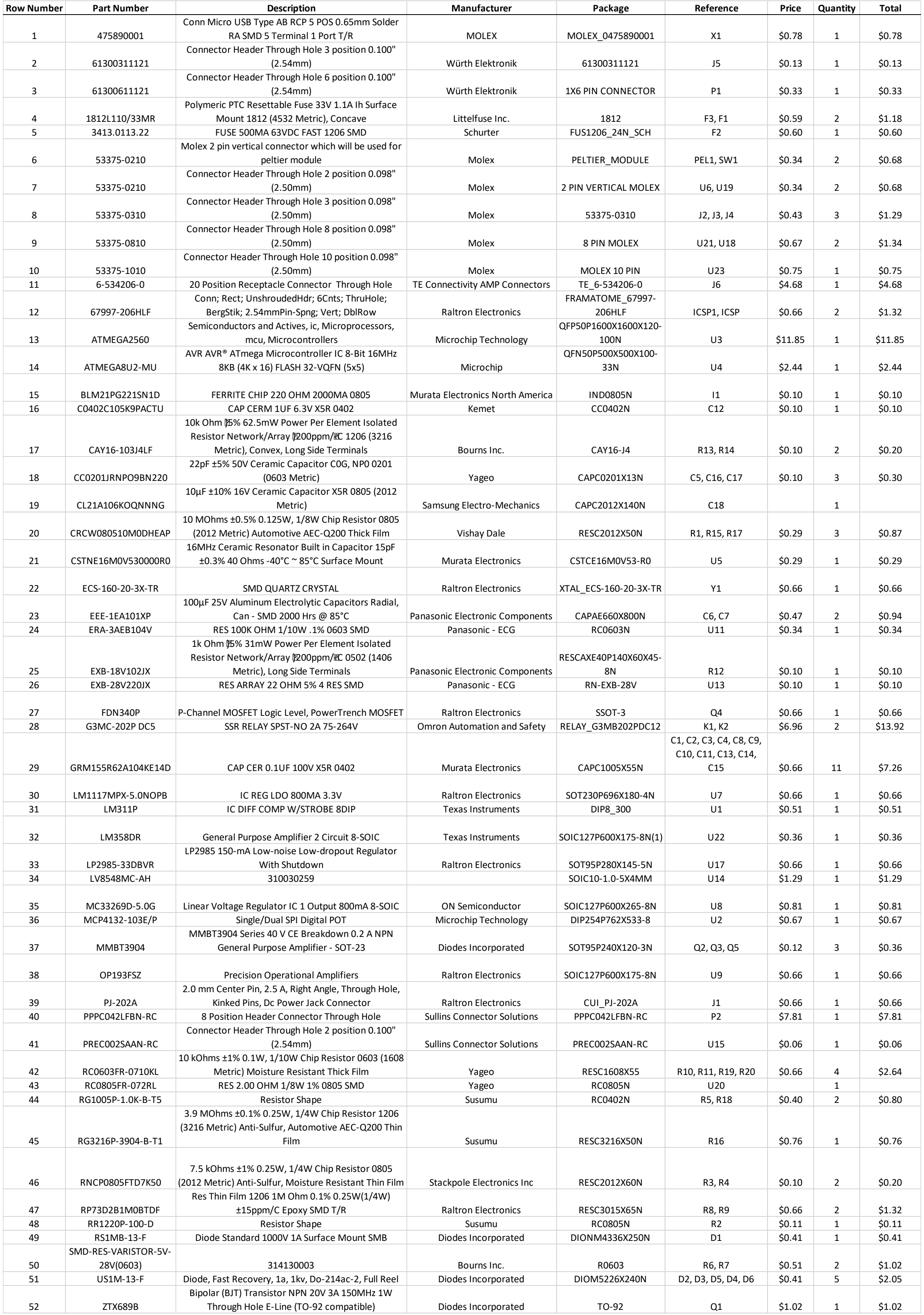
Part	Manufacturer	Description	Cost
PCB Components	Various	See Figure 23	$78.73
CP4055485	CUI Devices	Peltier Module	$71.62
T16-S	Actuonix	Track Actuator	$80.00
1824	Adafruit	3x4 Matrix Keypad	$7.50
Solar Panel	ECO-WORTHY	Solar Panel	$37.93
Charge Controller	GHB	Charge Controller	$29.69
Joystick	WMYCONGCONG	Joystick	$1.30
GDM1602K	GDM1602K	LCD Screen	$16.95
Micro SD Card Module	Wishiot	SD Card PCB	$1.70
1738-FIT0620-ND	DFRobot	Electronic Lock	$7.98
1216	Energizer	Coin Cell Battery	$2.35
COM-00097	Sparkfun	Pushbutton Switch	$0.35
53375-0210	Molex	Board to wire conn.	$0.34
Reset Button PCB	Osh Park	Reset Button PCB	$6.08
TMP36 PCB	SashaPCB*	TMP36 PCB	$13.02
Main PCB	Sunstone Circuits	PCB	$106.34**
Case	Portland 3D	3D Printed Case	$420***

* SashaPCB was the business name of the intermediate vendor who arranged for the manufacturing and transport of the TMP36 PCB. The price listed is the per unit cost for the components, PCB fabrication, and assembly.

** This is the per unit cost for the main PCB, this cost will be significantly reduced as the volume of the order increases.

*** This is the quoted cost; this part was never manufactured.

The total per unit cost for this project. This does not include the cost of various incidentals such as wires, purchased tools, and other incidentals. The cost of prototyping has not been included in the bill of materials. This BOM also does not include the cost of assembly for the main PCB, which was billed by Sunstone Circuits on behalf of Screaming Circuits at $1006.72

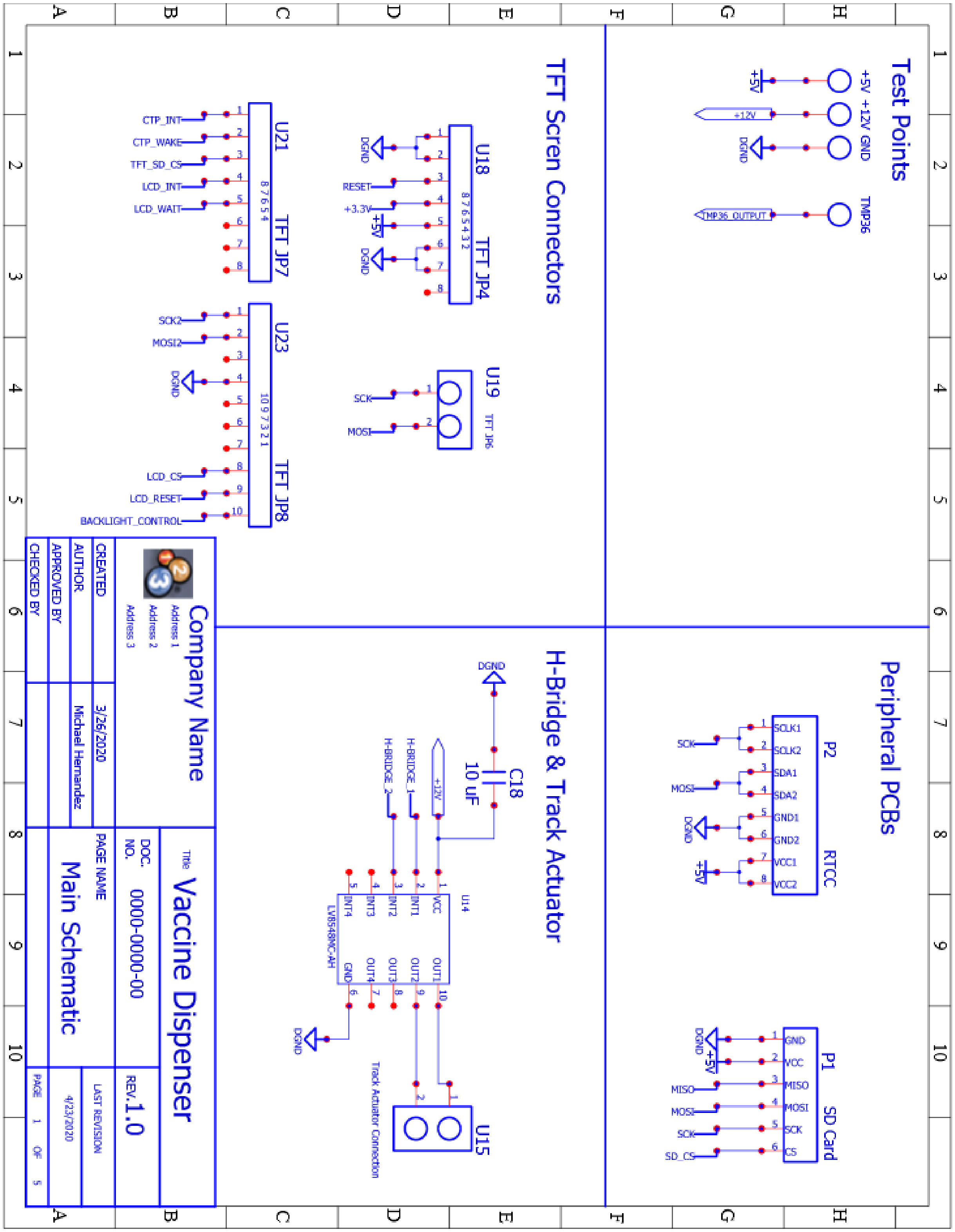


Row Number	Part Number	Description	Manufacturer	Package	Reference	Price	Quantity	Total
1	475890001	Conn Micro USB Type AB RCP 5 POS 0.65mm Solder RA SMD 5 Terminal 1 Port T/R	MOLEX	MOLEX_0475890001	X1	$0.78	1	$0.78
2	61300311121	Connector Header Through Hole 3 position 0.100" (2.54mm)	Würth Elektronik	61300311121	J5	$0.13	1	$0.13
3	61300611121	Connector Header Through Hole 6 position 0.100" (2.54mm)	Würth Elektronik	1X6 PIN CONNECTOR	P1	$0.33	1	$0.33
4	1812L110/33MR	Polymeric PTC Resettable Fuse 33V 1.1A Ih Surface Mount 1812 (4532 Metric), Concave	Littelfuse Inc.	1812	F3, F1	$0.59	2	$1.18
5	3413.0113.22	FUSE 500MA 63VDC FAST 1206 SMD	Schurter	FUS1206_24N_SCH	F2	$0.60	1	$0.60
6	53375-0210	Molex 2 pin vertical connector which will be used for peltier module	Molex	PELTIER_MODULE	PEL1, SW1	$0.34	2	$0.68
7	53375-0210	Connector Header Through Hole 2 position 0.098" (2.50mm)	Molex	2 PIN VERTICAL MOLEX	U6, U19	$0.34	2	$0.68
8	53375-0310	Connector Header Through Hole 3 position 0.098" (2.50mm)	Molex	53375-0310	J2, J3, J4	$0.43	3	$1.29
9	53375-0810	Connector Header Through Hole 8 position 0.098" (2.50mm)	Molex	8 PIN MOLEX	U21, U18	$0.67	2	$1.34
10	53375-1010	Connector Header Through Hole 10 position 0.098" (2.50mm)	Molex	MOLEX 10 PIN	U23	$0.75	1	$0.75
11	6-534206-0	20 Position Receptacle Connector Through Hole	TE Connectivity AMP Connectors	TE_6-534206-0	J6	$4.68	1	$4.68
12	67997-206HLF	Conn; Rect; UnshroudedHdr; 6Cnts; ThruHole; BergStik; 2.54mmPin-Spng; Vert; DblRow	Raltron Electronics	FRAMATOME_67997-206HLF	ICSP1, ICSP	$0.66	2	$1.32
13	ATMEGA2560	Semiconductors and Actives, ic, Microprocessors, mcu, Microcontrollers	Microchip Technology	QFP50P1600X1600X120-100N	U3	$11.85	1	$11.85
14	ATMEGA8U2-MU	AVR AVR® ATmega Microcontroller IC 8-Bit 16MHz 8KB (4K x 16) FLASH 32-VQFN (5x5)	Microchip	QFN50P500X500X100-33N	U4	$2.44	1	$2.44
15	BLM21PG221SN1D	FERRITE CHIP 220 OHM 2000MA 0805	Murata Electronics North America	IND0805N	I1	$0.10	1	$0.10
16	C0402C105K9PACTU	CAP CERM 1UF 6.3V X5R 0402	Kemet	CC0402N	C12	$0.10	1	$0.10
17	CAY16-103J4LF	10k Ohm ▯5% 62.5mW Power Per Element Isolated Resistor Network/Array ▯200ppm/▯C 1206 (3216 Metric), Convex, Long Side Terminals	Bourns Inc.	CAY16-J4	R13, R14	$0.10	2	$0.20
18	CC0201JRNPO9BN220	22pF ±5% 50V Ceramic Capacitor C0G, NP0 0201 (0603 Metric)	Yageo	CAPC0201X13N	C5, C16, C17	$0.10	3	$0.30
19	CL21A106KOQNNNG	10µF ±10% 16V Ceramic Capacitor X5R 0805 (2012 Metric)	Samsung Electro-Mechanics	CAPC2012X140N	C18		1	
20	CRCW080510M0DHEAP	10 MOhms ±0.5% 0.125W, 1/8W Chip Resistor 0805 (2012 Metric) Automotive AEC-Q200 Thick Film	Vishay Dale	RESC2012X50N	R1, R15, R17	$0.29	3	$0.87
21	CSTNE16M0V530000R0	16MHz Ceramic Resonator Built in Capacitor 15pF ±0.3% 40 Ohms -40°C ~ 85°C Surface Mount	Murata Electronics	CSTCE16M0V53-R0	U5	$0.29	1	$0.29
22	ECS-160-20-3X-TR	SMD QUARTZ CRYSTAL	Raltron Electronics	XTAL_ECS-160-20-3X-TR	Y1	$0.66	1	$0.66
23	EEE-1EA101XP	100µF 25V Aluminum Electrolytic Capacitors Radial, Can - SMD 2000 Hrs @ 85°C	Panasonic Electronic Components	CAPAE660X800N	C6, C7	$0.47	2	$0.94
24	ERA-3AEB104V	RES 100K OHM 1/10W .1% 0603 SMD	Panasonic - ECG	RC0603N	U11	$0.34	1	$0.34
25	EXB-18V102JX	1k Ohm ▯5% 31mW Power Per Element Isolated Resistor Network/Array ▯200ppm/▯C 0502 (1406 Metric), Long Side Terminals	Panasonic Electronic Components	RESCAXE40P140X60X45-8N	R12	$0.10	1	$0.10
26	EXB-28V220JX	RES ARRAY 22 OHM 5% 4 RES SMD	Panasonic - ECG	RN-EXB-28V	U13	$0.10	1	$0.10
27	FDN340P	P-Channel MOSFET Logic Level, PowerTrench MOSFET	Raltron Electronics	SSOT-3	Q4	$0.66	1	$0.66
28	G3MC-202P DC5	SSR RELAY SPST-NO 2A 75-264V	Omron Automation and Safety	RELAY_G3MB202PDC12	K1, K2	$6.96	2	$13.92
29	GRM155R62A104KE14D	CAP CER 0.1UF 100V X5R 0402	Murata Electronics	CAPC1005X55N	C1, C2, C3, C4, C8, C9, C10, C11, C13, C14, C15	$0.66	11	$7.26
30	LM1117MPX-5.0NOPB	IC REG LDO 800MA 3.3V	Raltron Electronics	SOT230P696X180-4N	U7	$0.66	1	$0.66
31	LM311P	IC DIFF COMP W/STROBE 8DIP	Texas Instruments	DIP8_300	U1	$0.51	1	$0.51
32	LM358DR	General Purpose Amplifier 2 Circuit 8-SOIC	Texas Instruments	SOIC127P600X175-8N(1)	U22	$0.36	1	$0.36
33	LP2985-33DBVR	LP2985 150-mA Low-noise Low-dropout Regulator With Shutdown	Raltron Electronics	SOT95P280X145-5N	U17	$0.66	1	$0.66
34	LV8548MC-AH	310030259		SOIC10-1.0-5X4MM	U14	$1.29	1	$1.29
35	MC33269D-5.0G	Linear Voltage Regulator IC 1 Output 800mA 8-SOIC	ON Semiconductor	SOIC127P600X265-8N	U8	$0.81	1	$0.81
36	MCP4132-103E/P	Single/Dual SPI Digital POT	Microchip Technology	DIP254P762X533-8	U2	$0.67	1	$0.67
37	MMBT3904	MMBT3904 Series 40 V CE Breakdown 0.2 A NPN General Purpose Amplifier - SOT-23	Diodes Incorporated	SOT95P240X120-3N	Q2, Q3, Q5	$0.12	3	$0.36
38	OP193FSZ	Precision Operational Amplifiers	Raltron Electronics	SOIC127P600X175-8N	U9	$0.66	1	$0.66
39	PJ-202A	2.0 mm Center Pin, 2.5 A, Right Angle, Through Hole, Kinked Pins, Dc Power Jack Connector	Raltron Electronics	CUI_PJ-202A	J1	$0.66	1	$0.66
40	PPPC042LFBN-RC	8 Position Header Connector Through Hole	Sullins Connector Solutions	PPPC042LFBN-RC	P2	$7.81	1	$7.81
41	PREC002SAAN-RC	Connector Header Through Hole 2 position 0.100" (2.54mm)	Sullins Connector Solutions	PREC002SAAN-RC	U15	$0.06	1	$0.06
42	RC0603FR-0710KL	10 kOhms ±1% 0.1W, 1/10W Chip Resistor 0603 (1608 Metric) Moisture Resistant Thick Film	Yageo	RESC1608X55	R10, R11, R19, R20	$0.66	4	$2.64
43	RC0805FR-072RL	RES 2.00 OHM 1/8W 1% 0805 SMD	Yageo	RC0805N	U20		1	
44	RG1005P-1.0K-B-T5	Resistor Shape	Susumu	RC0402N	R5, R18	$0.40	2	$0.80
45	RG3216P-3904-B-T1	3.9 MOhms ±0.1% 0.25W, 1/4W Chip Resistor 1206 (3216 Metric) Anti-Sulfur, Automotive AEC-Q200 Thin Film	Susumu	RESC3216X50N	R16	$0.76	1	$0.76
46	RNCP0805FTD7K50	7.5 kOhms ±1% 0.25W, 1/4W Chip Resistor 0805 (2012 Metric) Anti-Sulfur, Moisture Resistant Thin Film	Stackpole Electronics Inc	RESC2012X60N	R3, R4	$0.10	2	$0.20
47	RP73D2B1M0BTDF	Res Thin Film 1206 1M Ohm 0.1% 0.25W(1/4W) ±15ppm/C Epoxy SMD T/R	Raltron Electronics	RESC3015X65N	R8, R9	$0.66	2	$1.32
48	RR1220P-100-D	Resistor Shape	Susumu	RC0805N	R2	$0.11	1	$0.11
49	RS1MB-13-F	Diode Standard 1000V 1A Surface Mount SMB	Diodes Incorporated	DIONM4336X250N	D1	$0.41	1	$0.41
50	SMD-RES-VARISTOR-5V-28V(0603)	314130003	Bourns Inc.	R0603	R6, R7	$0.51	2	$1.02
51	US1M-13-F	Diode, Fast Recovery, 1a, 1kv, Do-214ac-2, Full Reel	Diodes Incorporated	DIOM5226X240N	D2, D3, D5, D4, D6	$0.41	5	$2.05
52	ZTX689B	Bipolar (BJT) Transistor NPN 20V 3A 150MHz 1W Through Hole E-Line (TO-92 compatible)	Diodes Incorporated	TO-92	Q1	$1.02	1	$1.02

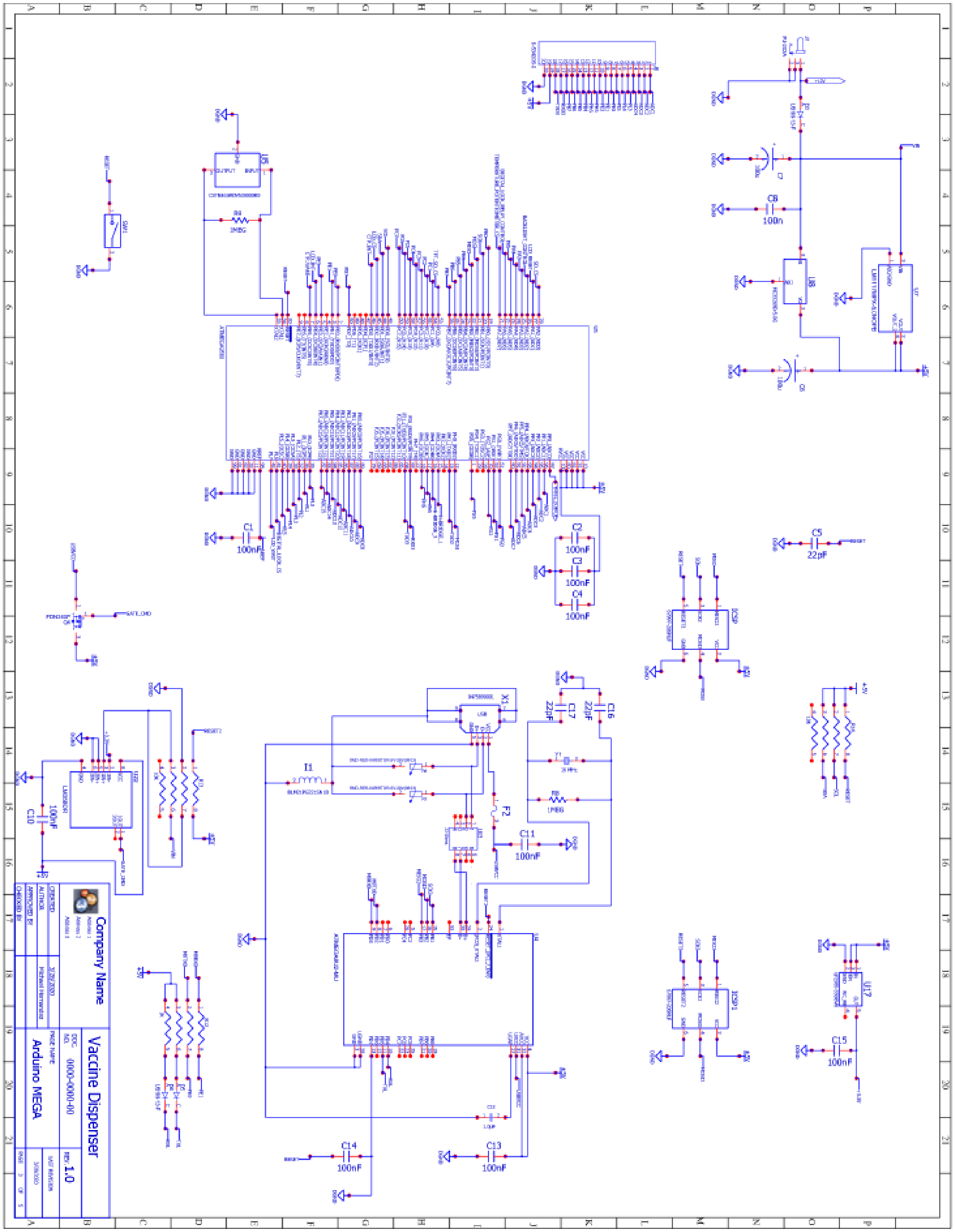
Figure 23 - PCB Components

Schematics

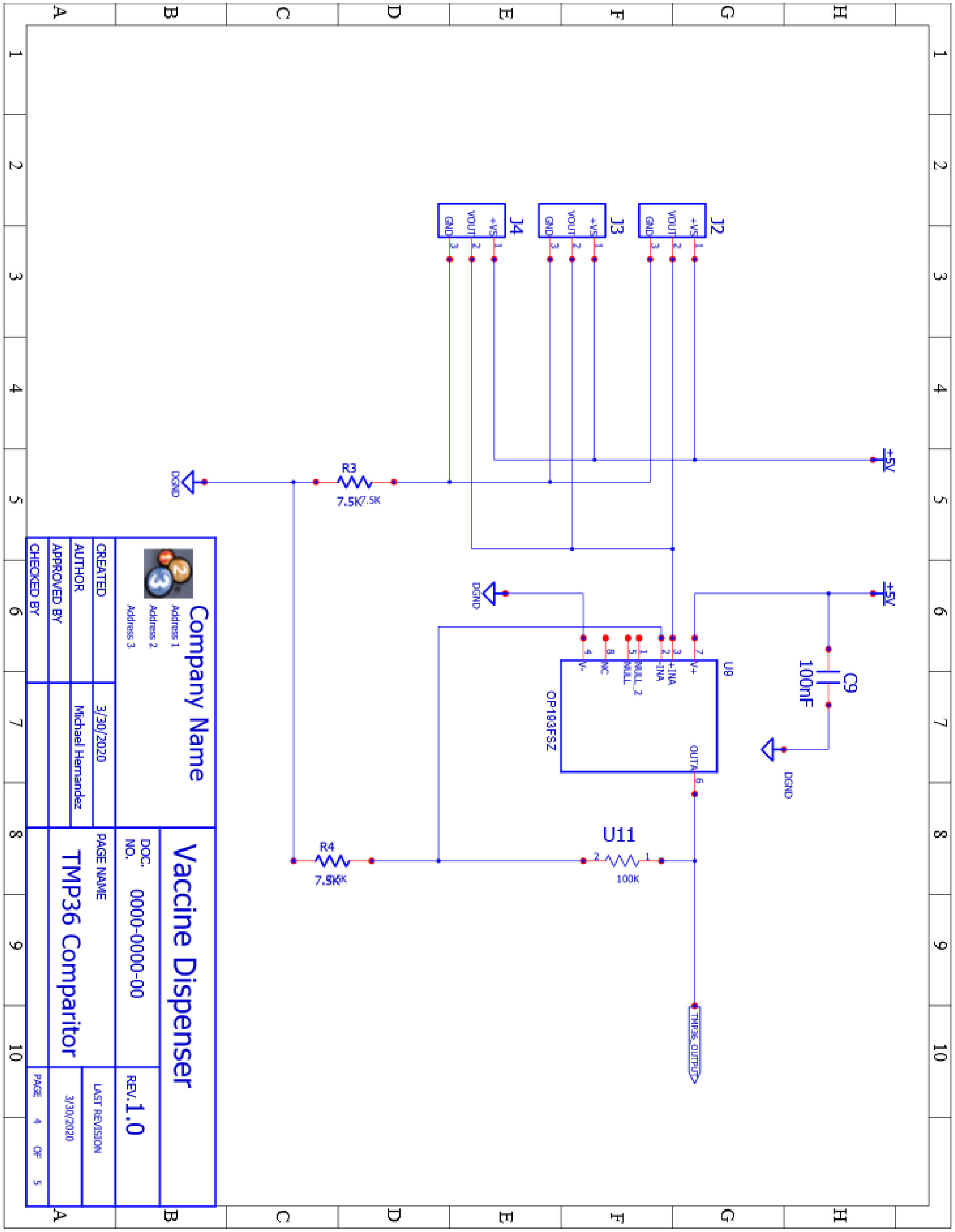
Main Schematic



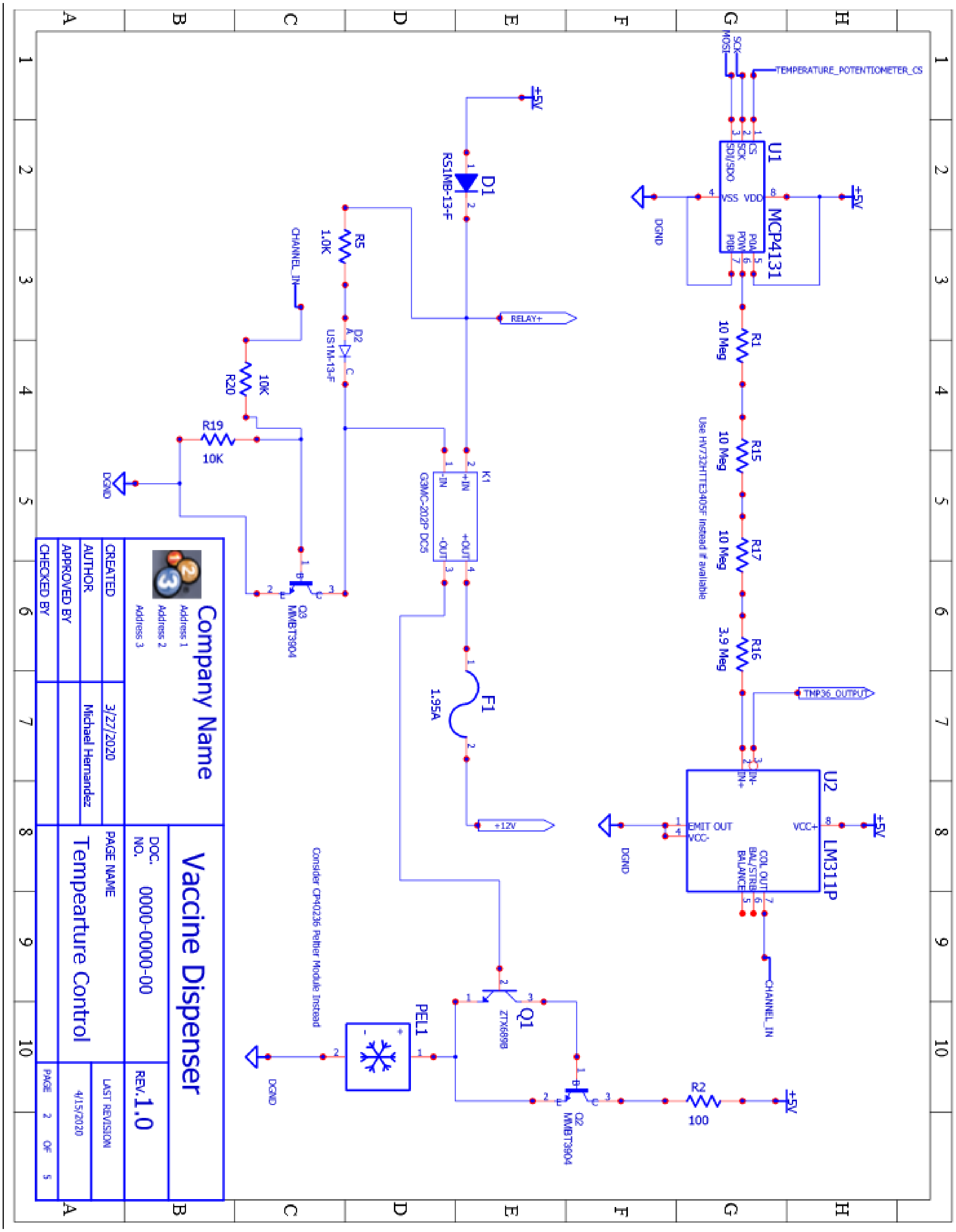
Arduino Mega



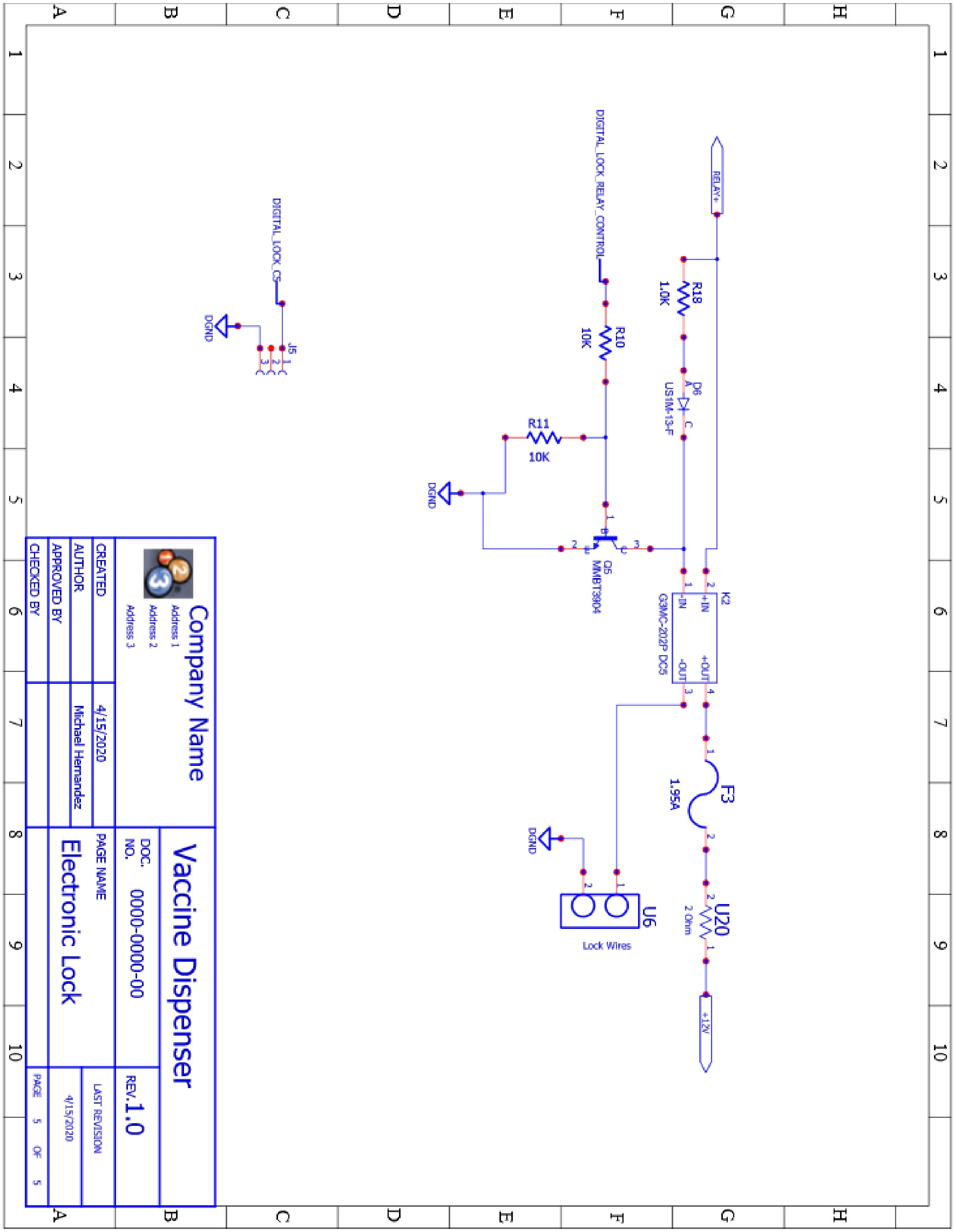
TMP36 Comparator



Temperature Control



Lock Control



Charge Controller

A prototype charge controller was created. The goal was to incorporate the design for a charge controller directly onto the manufactured PCB in order to conserve space. This piece was only partially successful however and was not used in the final design. The design was based off of an open source design on easyeda.com.

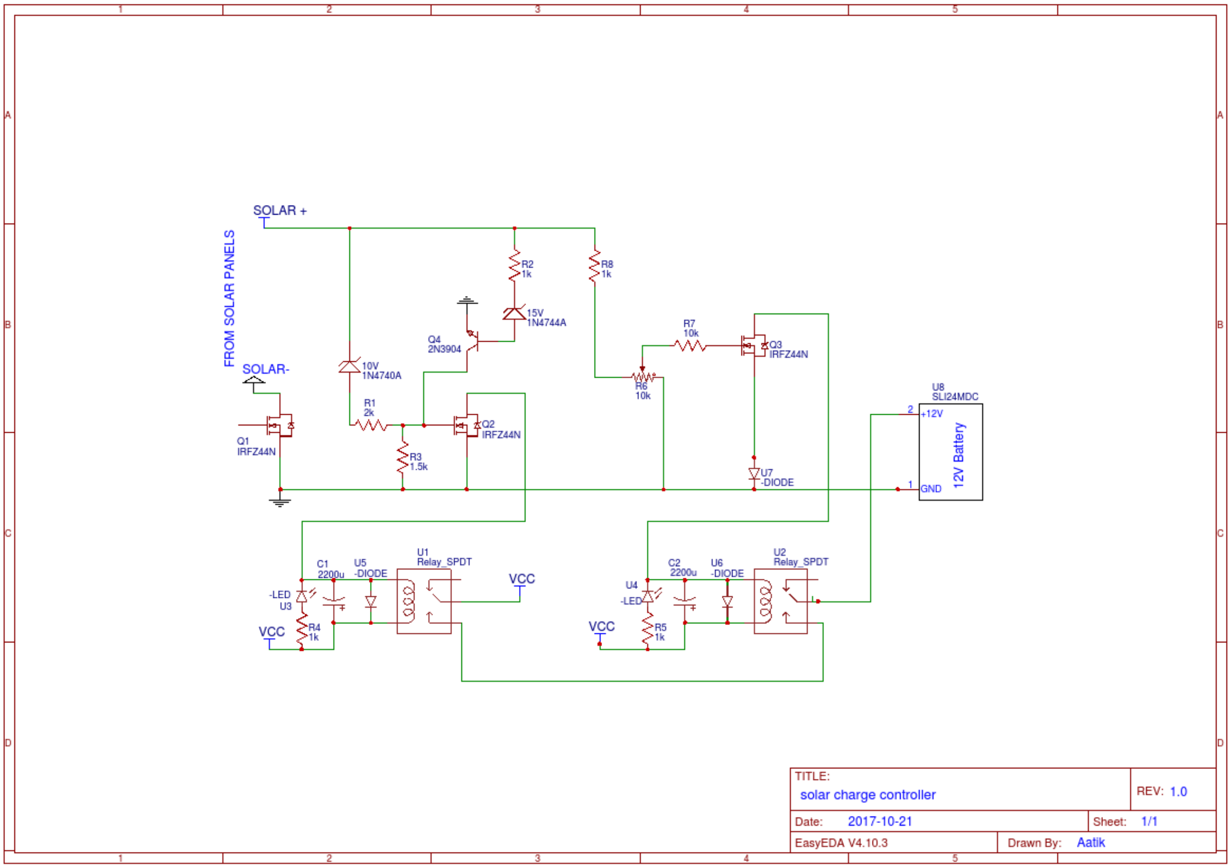


Figure 24 - Solar charge controller schematic

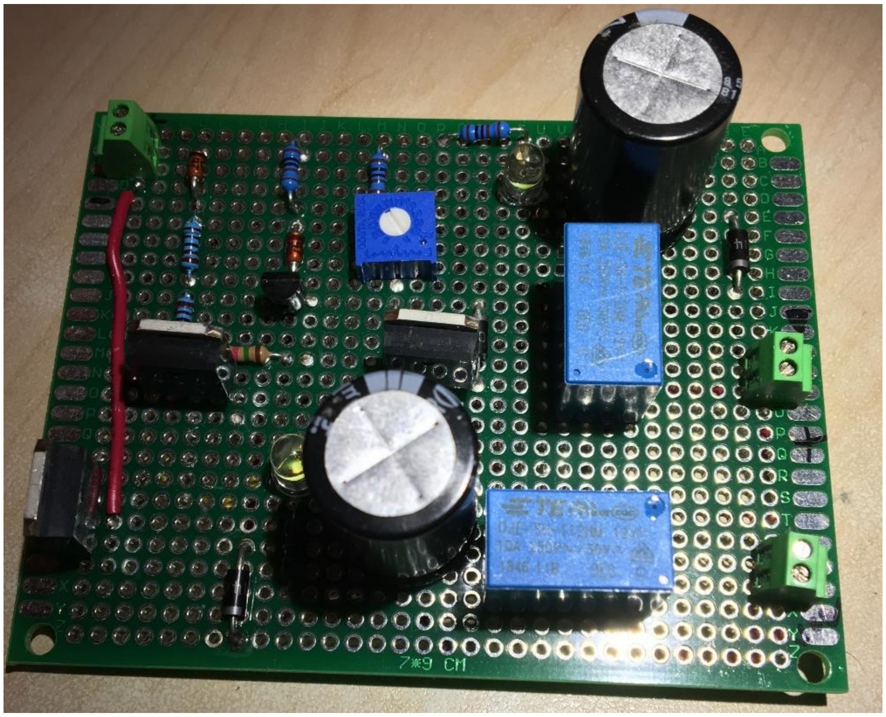


Figure 25 - Charge controller prototype

Code

The code for this project was divided into 11 different files for organization and easy debugging.

Main

```
// Vaccine dispenser
// Designer: Michael Hernandez
// https://michael-hernandez.info/

#include <Key.h>
#include <Keypad.h>
#include <LiquidCrystal.h>
#include <SD.h>
#include <SPI.h>
#include <RTCCI2C.h>
#include <Wire.h>
#include "byteAddress.h"
#include "globalVariables.h"
#include "pinDeclarations.h"

File myFile;  // the file used for SD card manipulation
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );
LiquidCrystal lcd(rs, en, d4, d5, d6, d7); // Sets up which pins go to the LCD display
```

```
RTCCI2C myRTCC;   // real time clock and calendar (RTCC)

//***************************************************************************
************************
//********************************************** Setup
**********************************************
//***************************************************************************
************************
void setup() {
  Serial.begin(9600); // opens serial port, sets data rate to 9600 bps
  while (!Serial) {
    ; // wait for serial port to connect. Needed for Leonardo only
  }

  // set up the real time clock and calendar (RTCC)
  myRTCC.begin();
//  setRTCC();         // calls the function setRTCC to set the day, time, etc.
  myRTCC.startClock();

  // setup the SD card
  if (!SD.begin(SDChipSelect)) {
    Serial.println("SD initialization failed!");
    return;
  }
  Serial.println("SD card initialized.");



  // Set up the cooling module potentiometer
  pinMode (potentiometerCS, OUTPUT);
  SPI.begin();
  digitalWrite(potentiometerCS, LOW);
  SPI.transfer(coolingPotentiometerAddress);
  SPI.transfer(setTemperature);
  digitalWrite(potentiometerCS, HIGH);

  //H bridge setup
  pinMode(DIR_A, OUTPUT);
  pinMode(DIR_B, OUTPUT);
  pinMode(PWM, OUTPUT);

  // Joystick startup routine
  pinMode(SW_pin, INPUT); // Joystick pin
  digitalWrite(SW_pin, HIGH); // joystick pin

  // Digital lock startup routine
  pinMode(lockPullupResistorPin, INPUT_PULLUP);  // sets up the pullup resistor
  pinMode(lockRelayPin, OUTPUT); // sets the pin connected to the lock relay as an output

  // LCD stuff title
  lcd.begin(16, 2); // set up the LCD's number of columns and rows
  lcd.setCursor(0, 0); // sets cursor at column 0, row 0
  lcd.print("Vaccine");
  lcd.setCursor(0, 1); // sets cursor at column 0, row 1
  lcd.print("Dispenser");
  //  delay(5000);
  lcd.clear();

  // LCD stuff copyright info
  lcd.begin(16, 2); // set up the LCD's number of columns and rows
  lcd.setCursor(0, 0); // sets cursor at column 0, row 0
  lcd.print("Copyright 2020");
  lcd.setCursor(0, 1); // sets cursor at column 0, row 1
  lcd.print("Michael H.");
  //  delay(5000);
  lcd.clear();
}

//************************************************************************
************************
//********************************************* Loop
*********************************************
//************************************************************************
************************

void loop() {
  delay(200);  // delays the loop so that the LCD screen can actually be read
  currentTemperature = temperature();
  Serial.println(" ");
  Serial.print("Current temperature: ");
  Serial.println(currentTemperature);
  Serial.print("Current voltage on the temperature pin (mV): ");
  Serial.println(analogRead(temperaturePin));
  Serial.println(" ");
  Serial.print("Current date: ");
  Serial.print(myRTCC.getMonth(RTCC_RTCC), HEX);
  Serial.print("/");
  Serial.print(myRTCC.getDate(RTCC_RTCC), HEX);
  Serial.print("/");
  Serial.print("20");
  Serial.println(myRTCC.getYear());
  Serial.print("Current Time: ");
  Serial.print(myRTCC.getHour(RTCC_RTCC));
  Serial.print(":");
  Serial.print(myRTCC.getMin(RTCC_RTCC));
  Serial.println(" ");

  joystickX = analogRead(X_pin);  // Allows the joystick to work in the X direction

  if (currentMenu < 10) {

    mainMenu = menuSelect(joystickX, 2); // the number of menus is menu + 1, so 2 creates 3
menus


//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Main Menu
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    switch (mainMenu) {

      //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ "to dispense"
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      case 0:
        menuChoice = keypad.getKey();
        dispensed = false;
        lcd.setCursor(0, 0);
        lcd.clear();
        lcd.print("To dispense");
        lcd.setCursor(1, 1);
        lcd.print("press #");
        currentMenu = 1;
        if (menuChoice == '#') { // if the entered character is #
          currentMenu = 10;  // Change the variable 'currentMenu', in order to select a dispense
menu (between 10 & 20)
          key0 = 95; // resets keypadCharacterpad character.
        }
        else {
          currentMenu = currentMenu; // if # isn't entered, do nothing.
        }
        break; // break for case zero, ends 'to dispense' menu

      //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ "settings"
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      case 1:
```

```
      menuChangeFlag = false;
      menuChoice = keypad.getKey();
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("Settings");
      lcd.setCursor(1, 1);
      lcd.print("press #");
      if (menuChoice == '#') { // if the entered character is #
        currentMenu = 30;  // Change the variable 'currentMenu', in order to select a new menu
      }
      else {
        currentMenu = currentMenu; // if # isn't entered, do nothing.
      }
      break;

    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ "unlock"
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    case 2:

      char keyInput; // creates a local variable to read the keypad


      if (menuChangeFlag == true && currentMenu < 10) {
        mainMenu = 1; // change the global variable "mainMenu" to be 1, and force it to go to
the previous screen
        break;
      }

      else if (menuChangeFlag == true && currentMenu >= 10) {
        dispenseMenu = 0;
        joystickX = 0.0;
        break;
      }

      else {
        for (int i = 0; i < 25; i++) {

          joystickX = analogRead(X_pin);  // Allows the joystick to work in the X direction

          if (joystickX < 50) {
            mainMenu = 1; // force the mainMenu variable to be 1
            menuChangeFlag = true; // change the menu flag to be true
            i = 26; // overloads the counter variable in the loop
            mainMenu = menuSelect(joystickX, 1); // changes the number of menus to 2 to force
the program to the next screen
            break; // break the loop
```

```
        }

        lcd.clear();
        lcd.setCursor(-i, 0); // as the counter counts up, changes the position to be the inverse
of that, forcing the display left
        lcd.print("Open refill compartment        "); // spaces added so that the last few letters
don't "get stuck" above "press #"
        lcd.setCursor(1, 1);
        lcd.print("press #");

        if (i == 0) {
          delay(1500); // delay 1.5 seconds
        }
        else {
          delay(300); // delay 3/10 of a second
        }

        keyInput = keypad.getKey();

        if (keyInput == '#') { // if the entered character is #
          menuChangeFlag = true;
          unlockFlag = true;
          currentMenu = 11;  // Change the variable 'currentMenu', in order to select a dispense
menu (between 10 & 20)
          dispenseMenu = 0;
          i = 26; // overloads the counter in the loop
          dispenseMenu = menuSelect(joystickX, 0); // changes the number of menus to 1 to
force the program to the correct screen
          joystickX = 0.0;
          break;
        }

        else {
          currentMenu = currentMenu; // if # isn't entered, do nothing.
        }

      }
    }
  } // closing brace for mainMenu
 } // closing brace for if (currentMenu < 10)


//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Dispense Menu
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

  else if (currentMenu >= 10 && currentMenu < 20) { // Dispense submenu

    dispenseMenu = menuSelect(joystickX, 1);

    switch (dispenseMenu) { // this block starts the dispense submenu

      //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ username input
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      case 0:
        lcd.setCursor(0, 0);
        lcd.clear();
        lcd.print("Enter user ID:");

        keyInput(0); // calls the function keyInput, the 0 indicates that we are not working with a
password array.

        arrayInput(userID, 0);  // calls on the function arrayInput in order to input values into the
username array

        break;  // this is the break for case 0, ends 'username input' menu

      //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ go back
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      case 1:

        flagReset(); // calls the function flagReset to change all values of iDFlag to false
        arrayReset(userID); // calls the function arrayReset to reset the userID array
        keyReset(); // This function resets the keys to be NO_KEY

        lcd.setCursor(0, 0);
        lcd.clear();
        lcd.print("To go back");
        lcd.setCursor(1, 1);
        lcd.print("press *");

        int previousSelect = keypad.getKey();

        if (previousSelect == 42) {
          currentMenu = 1;
        }

        else {
```

```
        currentMenu = currentMenu;
      }

      break; // break for case 1, ends 'go back' menu

    } // switch(dispenseMenu) closing curly brace

  } // closing curly brace for (currentMenu >= 10 && currentMenu < 20) (username input)


//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Password Menu
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

  else if (currentMenu >= 20 && currentMenu < 30) { // this is for the password input

    passwordMenu = menuSelect(joystickX, 1);

    switch (passwordMenu) { // this block starts the password submenu

      //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ password input
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      case 0:
        lcd.setCursor(0, 0);
        lcd.clear();
        lcd.print("Enter password: ");

        keyInput(1); // calls the function keyInput, the 1 denotes that we are working with a
password array.

        arrayInput(passwordInput, 1);  // calls on the function arrayInput in order to input values
into the username array

        usageTime();
        //       myFile = SD.open("history.txt", FILE_WRITE);
        //       if (myFile) {
        //         myFile.println("This is some text");
        //         myFile.close();
        //       }

        break;  // this is the break for case 0, ends 'password input' menu
```

```
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ go back
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    case 1:

      flagReset(); // calls the function flagReset to change all values of iDFlag to false
      arrayReset(passwordInput); // calls the function arrayReset to reset the passwordInput
array
      keyReset(); // This function resets the keys to be NO_KEY

      lcd.setCursor(0, 0);
      lcd.clear();
      lcd.print("To go back");
      lcd.setCursor(1, 1);
      lcd.print("press *");

      int previousSelect = keypad.getKey();

      if (previousSelect == 42) {
        currentMenu = 1;
      }

      else {
        currentMenu = currentMenu;
      }

      break; // break for case 1, ends 'go back' menu

   } // closing brace for the switch statement

  } // closing curly brace for (currentMenu >= 20 && currentMenu < 30) (password input)



//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Settings Menu
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  else if (currentMenu >= 30 && currentMenu < 40) { // this is the settings menu

    settingsMenu = menuSelect(joystickX, 2);

    switch (settingsMenu) {
```

```
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ current Temp
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    case 0:

      currentTemperature = temperature();
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("Current Temp.");
      lcd.setCursor(0, 1);
      lcd.print(currentTemperature);
      lcd.print(" C");
      delay(150);
      break;

    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ go back
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    case 1: // go back

      keyReset(); // This function resets the keys to be NO_KEY

      lcd.setCursor(0, 0);
      lcd.clear();
      lcd.print("To go back");
      lcd.setCursor(1, 1);
      lcd.print("press *");

      int previousSelect = keypad.getKey();

      if (previousSelect == 42) {
        currentMenu = 1;
      }

      else {
        currentMenu = currentMenu;
      }

      break; // break for case 1, ends 'go back' menu

    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ current time
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    case 2:
      lcd.setCursor(0, 0);
      lcd.clear();
      lcd.print("Current time is:");
      lcd.setCursor(1, 1);
      lcd.print(myRTCC.getHour(RTCC_RTCC));
```

```
        lcd.print(":");
        lcd.print(myRTCC.getMin(RTCC_RTCC));

//      int changeTemp = keypad.getKey();
//
//      if (previousSelect == 42) {
//        desiredTempMenu = menuSelect(joystickY, 14);
//        int selectionConfirm;
//
//        switch (desiredTempMenu) {
//          case 0:
//            lcd.clear();
//            lcd.setCursor(0, 0);
//            lcd.print("Scroll to select");
//            lcd.setCursor(0, 1);
//            lcd.print("desired temp.");
//
//          case 1:
//            lcd.clear();
//            lcd.setCursor(0, 0);
//            lcd.print("-4 C");
//            lcd.setCursor(0, 1);
//            lcd.print("press *");
//            selectionConfirm = keypad.getKey();
//
//            if (selectionConfirm == 42) {
//              setTemperature = 43; // we want a voltage from the TMP36 = 0.44 V for -4C;
0.01015625 x 43 ~ 0.437
//              lcd.clear();
//              lcd.setCursor(0, 0);
//              lcd.print("Set temperature");
//              lcd.setCursor(0, 1);
//              lcd.print("changed to -4 C");
//            }
//
//          case 2:
//            lcd.clear();
//            lcd.setCursor(0, 0);
//            lcd.print("-3 C");
//            lcd.setCursor(0, 1);
//            lcd.print("press *");
//            selectionConfirm = keypad.getKey();
//
//            if (selectionConfirm == 42) {
//              setTemperature = 44; // we want a voltage from the TMP36 = 0.44 V for -4C;
0.01015625 x 43 ~ 0.437
//              lcd.clear();
//              lcd.setCursor(0, 0);
//              lcd.print("Set temperature");
//              lcd.setCursor(0, 1);
//              lcd.print("changed to -3 C");
//            }
//
//          case 3:
//            lcd.clear();
//            lcd.setCursor(0, 0);
//            lcd.print("-2 C");
//            lcd.setCursor(0, 1);
//            lcd.print("press *");
//            selectionConfirm = keypad.getKey();
//
//            if (selectionConfirm == 42) {
//              setTemperature = 44; // we want a voltage from the TMP36 = 0.44 V for -4C;
0.01015625 x 43 ~ 0.437
//              lcd.clear();
//              lcd.setCursor(0, 0);
//              lcd.print("Set temperature");
//              lcd.setCursor(0, 1);
//              lcd.print("changed to -2 C");
//            }
//         }
//       }
//
//       else {
//         currentMenu = currentMenu;
//       }
//
//       digitalWrite(potentiometerCS, LOW);
//       SPI.transfer(coolingPotentiometerAddress);
//       SPI.transfer(setTemperature);
//       digitalWrite(potentiometerCS, HIGH);

    }

  } // closing curly brace for (currentMenu >= 30 && currentMenu < 40) (settings)

}
```

File Write Routines

```
void usageTime() {
  // this function writes the date and time of the user to the SD card when used.
  myFile = SD.open("history.txt", FILE_WRITE);
  // if the file opened okay, write to it:
  if (myFile) {
    Serial.print("Writing to test.txt...");
    myFile.print("Used by user: ");
    myFile.print(userIDCheck[0]);
    myFile.print(userIDCheck[1]);
    myFile.print(userIDCheck[2]);
    myFile.println(userIDCheck[3]);
    myFile.print("Used at time: ");
    myFile.print(myRTCC.getHour(RTCC_RTCC));
    myFile.print(":");
    myFile.println(myRTCC.getMin(RTCC_RTCC));
    myFile.print("Used on this date: ");
    myFile.print(myRTCC.getMonth(RTCC_RTCC), HEX);
    myFile.print("/");
    myFile.print(myRTCC.getDate(RTCC_RTCC), HEX);
    myFile.print("/");
    myFile.print("20");
    myFile.println(myRTCC.getYear());
    myFile.println(" ");
    // close the file:
    myFile.close();
    Serial.println("done.");
  } else {
    // if the file didn't open, print an error:
    Serial.println("error opening test.txt");
  }

  // re-open the file for reading:
  myFile = SD.open("history.txt");
  if (myFile) {
    Serial.println("history.txt:");


    myFile.close();
  } else {
    // if the file didn't open, print an error:
    Serial.println("error opening test.txt");
  }
}

void writeLockUsage() {
  myFile = SD.open("history.txt", FILE_WRITE);

  // if the file opened okay, write to it:
```

```
  if (myFile) {
    Serial.print("Writing to history.txt...");
    myFile.println("LOCK USAGE");
    // close the file:
    myFile.close();
    Serial.println("done.");
  } else {
    // if the file didn't open, print an error:
    Serial.println("error opening usagehistory.txt");
  }

  // re-open the file for reading:
  myFile = SD.open("history.txt");
  if (myFile) {
    Serial.println("history.txt");

    // close the file:
    myFile.close();
  } else {
    // if the file didn't open, print an error:
    Serial.println("error opening test.txt");
  }
}

void writeDispenseUsage() {
myFile = SD.open("history.txt", FILE_WRITE);

  // if the file opened okay, write to it:
  if (myFile) {
    Serial.print("Writing to history.txt...");
    myFile.println("DISPENSER USAGE");
    // close the file:
    myFile.close();
    Serial.println("done.");
  } else {
    // if the file didn't open, print an error:
    Serial.println("error opening history.txt");
  }

  // re-open the file for reading:
  myFile = SD.open("history.txt");
  if (myFile) {
    Serial.println("history.txt");

    // close the file:
    myFile.close();
  } else {
    // if the file didn't open, print an error:
    Serial.println("error opening test.txt");
  }
}
```

LCD Management

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ lcdDisplay
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// This function, when called will display any number of digits on the LCD display
void lcdDisplay(char Array[], int digits) {
  switch (digits) {
    case 1:
      lcd.print(Array[0]);
      break;

    case 2:
      lcd.print(Array[0]);
      lcd.print(Array[1]);
      break;

    case 3:
      lcd.print(Array[0]);
      lcd.print(Array[1]);
      lcd.print(Array[2]);
      break;

    case 4:
      lcd.print(Array[0]);
      lcd.print(Array[1]);
      lcd.print(Array[2]);
      lcd.print(Array[3]);
      break;
  }
}
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Load User Info

```

// this function
void readUsername() {
  nameArrayFlag = false;

  myFile = SD.open("userinfo.txt");
```

```
  while (myFile.available()) {
    if (myFile.read() == '<') {
      nameArrayFlag = true;
      testCharacter = '%';
      for (int j = 0; j < 50; j++) {
        if (myFile.read() == '>') {
          nameArrayFlag = false;
        }
        else if (nameArrayFlag == true) {
          nameArray[j] = myFile.read();
        }
        else {
          nameArray[j] = ' ';
        }
      }
    }
  }
  myFile.close();

}

void readPassword() {
  char completeArray [lengthCounter] = {}; // creates an array the size of the text file

  // the following block of code copies the information from the SD card to an array in the
arduino
  myFile = SD.open("userinfo.txt");
  while (myFile.available()) {
    for (int i = 0; i < lengthCounter; i++) {
      completeArray[i] = myFile.read();
    }
  }
  myFile.close();
  Serial.println();
  Serial.println();

  int flagCheck = 0;
  int y = 0;

  for (int i = 0; i < lengthCounter; i++) {
    Serial.print(completeArray[i]);

    if (completeArray[i] == '<') {
      if (completeArray[i - 1] == 'd' && completeArray[i - 2] == 'i') {
        savedPassword[0] = completeArray[i + 1];
        savedPassword[1] = completeArray[i + 2];
        savedPassword[2] = completeArray[i + 3];
        savedPassword[3] = completeArray[i + 4];
      }
    }

  }
}



void fileLengthCounter() {
  myFile = SD.open("userinfo.txt");
  if (myFile) {
    Serial.println("Read:");
    // Reading the whole file
    while (myFile.available()) {
      Serial.write(myFile.read());

      lengthCounter = lengthCounter + 1;
    }
    myFile.close();
  }
  else {
    Serial.println("error opening WRITE.TXT");
  }
}
```

SPI Devices

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ dispenseRoutine
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// This function will move the track acuator to dispense one unit
void dispenseRoutine() {
  if (dispensed == false) {

    writeDispenseUsage();
    usageTime(); // writes the time and user at which the lock was used to the SD card

    //drive forward at full speed by pulling DIR_A High
    //and DIR_B low, while writing a full 255 to PWM to
    //control speed
    digitalWrite(DIR_A, HIGH);
    digitalWrite(DIR_B, LOW);
    analogWrite(PWM, 255);

    //wait 1 second
    delay(25000);

```

```
    //Brake the motor by pulling both direction pins to
    //the same state (in this case LOW). PWM doesn't matter
    //in a brake situation, but set as 0.
    digitalWrite(DIR_A, LOW);
    digitalWrite(DIR_B, LOW);
    analogWrite(PWM, 0);

    //wait 1 second
    delay(500);

    //change direction to reverse by flipping the states
    //of the direction pins from their forward state
    digitalWrite(DIR_A, LOW);
    digitalWrite(DIR_B, HIGH);
    analogWrite(PWM, 255);

    //wait 1 second
    delay(25000);

    //Brake again
    digitalWrite(DIR_A, LOW);
    digitalWrite(DIR_B, LOW);
    analogWrite(PWM, 0);

    //wait 1 second
    delay(500);

    dispensed = true;  // prevents the routine from running a second time

  }
  else {

  }

}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~


//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ setRTCC
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// This function will run once during the setup, and again if the user wants to change the
timezone
// it will set the variables for the real time clock and calendar (RTCC)

void setRTCC() {
```

```
  myRTCC.setSec(RTCC_RTCC, 0x00);
  myRTCC.setMin(RTCC_RTCC, 0x28); // 25 in HEX
  myRTCC.setHour(RTCC_RTCC, 0x10); // 20 in HEX
  myRTCC.setDay(RTCC_RTCC, 0x07);
  myRTCC.setDate(RTCC_RTCC, 0x30);
  myRTCC.setMonth(RTCC_RTCC, 0x05);
  myRTCC.setYear(0x14);
  myRTCC.enableVbat(); // enables the use of the battery backup
}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ temperature
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// this function reads and returns the temperature
// There are 10mV per degree Celcius, therefore 0 C = 0.48V, and 22 C = 0.7 V
int temperature() {

  float reading = analogRead(temperaturePin);
  float voltage = reading * 5.0;
  voltage = voltage / 1024.0;
  //Serial.println(voltage);
  float  temperature = (voltage - 0.5) * 100.0;
  temperature = temperature + 54; // offset correction

  return temperature;

} // closing brace for temperature

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ temperatureControl
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// this function reads and returns the temperature
void temperatureControl() {

  int currentTemp;
  currentTemp = analogRead(temperaturePin) - 2.5; // the -3.5 C is a correction (test showed 75
F, room temp is 71)

  int error = setTemperature - currentTemp; // calculates how far off in degrees the current
temperature is

  if (error > 0) {
    digitalWrite(temperatureOutputPin, HIGH);
```

```
  }


}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ unlock
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// This function will unlock the lock one time and will not work again until the unlock variable
is
// reset
void unlock() {

  if (lockActivated == false) {
    digitalWrite(lockRelayPin, HIGH);
    delay(3000);
    digitalWrite(lockRelayPin, LOW);
  }

  else {
    digitalWrite(lockRelayPin, LOW);
    writeLockUsage(); // writes lock usage to the SD card
    usageTime(); // writes the time and user at which the lock was used to the SD card
  }

}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Array Management

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ arrayCheck
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Compares two arrays against each other, used for user ID and password input

bool arrayCheck(char array1[], char array2[]) { // the two arrays to be compared against each
other are entered as input arguments

  bool checkArrays [4] = {false, false, false, false}; // create a local array of bool values
  int counter = 0;
  bool finalCheck; // this is what the function will return

  for (int i = 0; i < 4; i++) {
```

```
    if (array1[i] == array2[i]) {
      checkArrays[i] = true;
    }

    else {
      checkArrays[i] = false;
    }

  }

  for (int i = 0; i < 4; i++) {
    counter = counter + checkArrays[i]; // counts the number of trues in the ID flag array, if true
then we add a 1
  }

  if (counter == 4) {
    finalCheck = true;
  }

  else {
    finalCheck = false;
  }

  return finalCheck; // returns the final check variable as either true or false

}
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~




//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ arrayInput
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Inputs values into the arary, can be recycled for both username and password.  This function
takes
// an array to be modified as an input value, and does not return a value.

void arrayInput(char inputArray[], bool password) {


  // This block of code deals with the FIRST value of the array of interest
  if (key0 != NO_KEY && iDFlag[0] == false) { // conditions for entering the FIRST value in
the array
    inputArray[0] = key0; // enter the value stored in the key0 variable to the first value of the
input array
    passwordMask[0] = 42;
```

```
    lcd.setCursor(0, 1); // sets cursor at column 0, row 0
    if (password == false) {
      lcdDisplay(inputArray, 1); // display the first value of the input array on the LCD screen
    }
    else { // if the boolean input value "password" is false
      lcdDisplay(passwordMask, 1); // display a * for the first value of the array on the LCD
screen instead.
    }
    iDFlag[0] = true; // set the first value of iDFlag to be true
  }
  else { // if the conditions for entering the FIRST value in the array are not met
    lcd.setCursor(0, 1); // sets cursor at column 0, row 0

    if (password == false) { // if the boolean input value "password" is false
      lcdDisplay(inputArray, 1); // display 1 digit of the entered array
    }
    else { // if the boolean input value "password" is true
      lcdDisplay(passwordMask, 1); // display a * instead to mask the password
    }
  }



  // This block deals with the SECOND value of the array of interest
  if (key1 != NO_KEY && iDFlag[1] == false) { // conditions for entering the SECOND value
in the array
    inputArray[1] = key1; // input the value stored in the variable key1 into the second value in
the input array
    passwordMask[1] = 42;
    lcd.setCursor(0, 1); // sets the LCD cursor to row 2, column 1
    if (password == false) { // if the input variable "password" is set to false (input 0)
      lcdDisplay(inputArray, 2); // display two digits of the input array
    }
    else { // if the input variable "password" is set to true (input 1)
      lcdDisplay(passwordMask, 2); // display two digits of the passwordMask array
    }
    iDFlag[1] = true; // set the second value of the iDFlag array equal to true
  }
  else { // if the conditions for entering the SECOND value in the array are not met
    lcd.setCursor(0, 1); // sets cursor at column 1, row 2
    if (password == false) {  // if the value for the input variable "password" is set to 0
      lcdDisplay(inputArray, 2);  // display two values on the input arrray
    }
  }



  // This block deals with the THIRD value of the array of interest
  if (key2 != NO_KEY && iDFlag[2] == false) { // conditions for entering the THIRD value in
the array
    inputArray[2] = key2; // input the value stored in the variable key2 into the third value in the
input array
    passwordMask[2] = 42;
    lcd.setCursor(0, 1); // sets cursor at column 1, row 2
    if (password == false) {  // if the value for the input variable "password" is set to 0
      lcdDisplay(inputArray, 3); // display three values on the input arrray
    }
    else { // if the conditions for entering the THIRD value in the array are not met
      lcdDisplay(passwordMask, 3); // display three digits of the passwordMask array
    }
    iDFlag[2] = true; // set the third value of the iDFlag array equal to true
  }
  else { // if the conditions for entering the third value of the array are not met
    lcd.setCursor(0, 1); // sets cursor at column 0, row 1
    if (password == false) { // if the value for the input variable "password" is set to 0
      lcdDisplay(inputArray, 3); // display three values on the input arrray
    }
    else {
      lcdDisplay(passwordMask, 3); // display three digits of the passwordMask array
    }
  }



  // This block deals with the FOURTH value of the array of interest
  if (key3 != NO_KEY && iDFlag[3] == false) { // conditions for entering the FOURTH value
in the array
    inputArray[3] = key3; // input the value stored in the variable key3 into the fourth value in
the input array
    passwordMask[3] = 42;
    lcd.setCursor(0, 1); // sets cursor at column 0, row 1
    if (password == false) { // if the value for the input variable "password" is set to 0
      lcdDisplay(inputArray, 4);  // display four values on the input arrray
    }
    else {
      lcdDisplay(passwordMask, 4); // display four values on the passwordMask arrray
    }
    iDFlag[3] = true; // set the final value of the iDFlag array to be true
  }
  else { // if the input conditions are not met
    lcd.setCursor(0, 1); // sets cursor at column 0, row 1
    if (password == false) { // if the input variable "password" is set to 0
      lcdDisplay(inputArray, 4); // display four digits of the input array
    }
    else { // if the input variable "password" is set to 1
```

```
      lcdDisplay(passwordMask, 4);  // dispaly four variables of the array passwordMask
(display four *'s)
    }
  }
} // closing curly brace for the function
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~



//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ arrayReset
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// resets all values in a username or password array back to _
void arrayReset(char inputArray[]) {
  for (int i = 0; i < 4; i++) {
    inputArray[i] = '_';
  }
}
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ flagReset
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// This function does not return any value.
// This function resets the ID flags used to determine which place in an array you are to enter a
// value
void flagReset() {

  for (int i = 0; i < 4; i++) {
    iDFlag[i] = false; // changes the value of the iDFlag to false for cells 0-3 of the array.
  }

}
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ idFlagCount
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Counts the number of true variables in the iDFlag array.  For every true, we add a 1, for
every
// false, we add a 0.  We then reutrn the total value of the count.
int idFlagCount() {

  int counter = 0; // Initialize a local variable of type int to keep a running total of each
successful value.
```

```
  for (int i = 0; i < 4; i++) {
    counter = counter + iDFlag[i]; // counts the number of trues in the ID flag array, if true then
we add a 1
  }

  return counter; // returns the value of the local variable

}
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Byte Addresses

```
// This .h file is for all byte addresses

#ifndef BYTEADDRESS_H
#define BYTEADDRESS_H

byte coolingPotentiometerAddress = 0x00; // address for the potentiometer in the cooling
module

#endif
```

Keypad Management

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ keyInput
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// This function allows the program to accept an input from the keypad for all four entries
void keyInput(bool isPassword) { // enter a 0 for not a password, and 1 for password.

  bool successCheck; // local variable used to move on to next menu if necessary

  if (iDFlag[0] == false && iDFlag[1] == false && iDFlag[2] == false && iDFlag[3] == false)
{
    key0 = keypad.getKey();
  }
  else if (iDFlag[0] == true && iDFlag[1] == false && iDFlag[2] == false && iDFlag[3] ==
false) {
    key1 = keypad.getKey();
  }
  else if (iDFlag[0] == true && iDFlag[1] == true && iDFlag[2] == false && iDFlag[3] ==
false) {
    key2 = keypad.getKey();
  }
  else if (iDFlag[0] == true && iDFlag[1] == true && iDFlag[2] == true && iDFlag[3] ==
false) {
    key3 = keypad.getKey();
```

```
  }


  // check to see if the input userID matches a stored userID
  else if (isPassword == false) { // if we are not working with the password array
    successCheck = arrayCheck(userID, userIDCheck); // compares userID against the
predefined "password"
    if (successCheck == true) {
      currentMenu = 20;
      flagReset(); // calls the function flagReset to change all values of iDFlag to false
      keyReset(); // This function resets the keys to be NO_KEY
      arrayReset(passwordMask);
      arrayReset(userID);
    }
    else {
      arrayReset(userID);
    }
  }

  // If we are inputting the password for the DISPENSING MECHANISM
  else if (isPassword == true && unlockFlag == false) { // if we are working with the password
array
    successCheck = arrayCheck(passwordInput, userIDCheck); // compares userID against the
predefined "password"
    if (successCheck == true) {
      writeDispenseUsage();
      usageTime();
      lcd.clear();
      lcd.setCursor(0, 0); // sets cursor at column 0, row 0
      lcd.print("Success!");
      dispenseRoutine();
      arrayReset(passwordMask);  // clears the password mask values
      arrayReset(passwordInput); // clears out the values for the password
      isPassword = false; // resets the screen to go back to the user id input
    }
    else {
      arrayReset(passwordInput);
    }
  }


  // if we are inputting the password for the ELECTRONIC LOCK
  else if (isPassword == true && unlockFlag == true) { // if we are working with the password
array & unlock
    successCheck = arrayCheck(passwordInput, userIDCheck); // compares userID against the
predefined "password"
```

```
    if (successCheck == true) {
      writeLockUsage();
      usageTime();
      lcd.clear();
      lcd.setCursor(0, 0); // sets cursor at column 0, row 0
      lcd.print("Success!");
      unlock();
      unlockFlag = false;
      dispensed = true; // prevents the dispense motor from running
    }
    else {
      arrayReset(passwordMask);
      arrayReset(passwordInput);
    }
  }

  else { // if all else fails, reset everything
    flagReset(); // calls the function flagReset to change all values of iDFlag to false
    keyReset(); // This function resets the keys to be NO_KEY
    arrayReset(passwordInput);
    arrayReset(passwordMask);
  }

}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ keyReset
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// This void type function will reset the keypad inputs, key0-4 back to NO_KEY
void keyReset() {
  key0 = NO_KEY;
  key1 = NO_KEY;
  key2 = NO_KEY;
  key3 = NO_KEY;
}
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Menu Management

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ menuSelect
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// This function will allow the user to select a menu based on the joystick position, and return a
// variable which will be used to display the correct menu.  The returned variable is called
// menuCount.  This function allows you to control the total number of menus.
```

```
int menuSelect(float joystick, int totalMenus) {  // inputs are joystick position and # of menus
  if (joystick > 1000) {
    menuCount = menuCount + 1;  // upon moving the joystick increase the menu variable by 1
    keypadCharacterpadSelection = 0;  // clear the keypadCharacterpad Selection variable
    if (menuCount > totalMenus) {
      menuCount = totalMenus; // the menuCount can't go over the total number of menus
    }
  }

  else if (joystick < 50) {
    menuCount = menuCount - 1;  // decreases the value of menu count by 1
    keypadCharacterpadSelection = 0;
    if (menuCount < 0) {
      menuCount = 0;  // menu count can't go below 0
    }
  }

  else {
    menuCount = menuCount;  // if the joystick didn't move then keep the menu the same
  }

  return menuCount;  // return which menu you are working in.
}
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

globalVariables.h

```
// This .h file is for my global variables that I will be using in my project

#ifndef GLOBALVARIABLES_H
#define GLOBALVARIABLES_H

//************************************** PASSWORD LOADING STUFF
**************************************
char userStart [5] = {'u', 's', 'e', 'r', '<'};
char idStart [7] = {'u', 's', 'e', 'r', 'i', 'd', '<'};
char passwordStart [9] = {'p', 'a', 's', 's', 'w', 'o', 'r', 'd', '<'};
char terminal = '>';
int lengthCounter = 0;
String input;
char savedPassword [4] = {};
int y = 0;
char nameArray [50] = {};
bool nameArrayFlag = false;
char testCharacter;
int z = 0;
```

```
//****************************************************************************
************************

//*********************************************** ARRAYS
**********************************************
boolean iDFlag [4] = {false, false, false, false}; // Used to check which place we are in the
loop
char passwordInput [4] = {95, 95, 95, 95}; // stores the password values
char passwordMask [4] = {'_', '_', '_', '_'}; // what will be displayed for password values
char userID [4] = {'_', '_', '_', '_'}; // A blank array that will have four slots for a user input
char userIDCheck [4] = {'1', '2', '3', '4'}; // This is the "Correct" user ID that will be compared
to the input
//****************************************************************************
************************

//********************************************** BOOLEAN
**********************************************
bool dispensed = false;        // runs the dispense routine one time
bool englishFlag = true;       // default, used for English language
bool japaneseFlag = false;     //日の丸です！
bool lockActivated = false;    // used in the unlock function to check if the lock has been
unlocked
bool menuChangeFlag = false;   // used for the unlock refill compartment submenu
bool spanishFlag = false;      // used for Spanish language
bool successfulMatch = false;  // used in the SD card password array
bool unlockFlag = false;       // used to unlock the main compartment
//****************************************************************************
************************

//************************************************ CHAR
***********************************************
char checkCharacter = '%';         // used in the password array to separate values
char key0;                         // used in the keypad entry
char key1;                         // used in the keypad entry
char key2;                         // used in the keypad entry
char key3;                         // used in the keypad entry
char keypadCharacterpadInput = 0;  // The last entered value on the keypad
char menuChoice;                   // used for menu selection
//****************************************************************************
************************

//************************************************ FLOAT
**********************************************
float currentTemperature = 0;  // stores the current temperature
float joystickX = 0.0;         // joystick in the X direction
float joystickY = 0.0;         // joystick in the Y direction
```

```
//********************************************************************************
**************************

//*************************************************** INT
*************************************************
int currentMenu = 0;
int desiredTempMenu = 0;
int dispenseMenu = 0;
int iDEntry = 0;
int keypadCharacterpadSelection = 0;
int mainMenu = 0;
int menuCount = 0; // used to select which screen you are working in
int setTemperature = 43;  // set temperature point
int settingsMenu = 0;
int passwordMenu = 0;

int day;          // RTCC day of the week (Sun - Sat)
int hour;         // RTCC hour
int minute;       // RTCC minute
int second;       // RTCC second
//********************************************************************************
**************************

#endif
```

pinDeclarations.h

```
// This .h file is for my pin declarations

#ifndef PINDECLARATIONS_H
#define PINDECLARATIONS_H

//********************************************** H-Bridge Pins
********************************************
const int DIR_A = 5;  // forward direction
const int DIR_B = 6;  // reverse direcion
const int PWM = 7;    // pulse width modulation
//********************************************************************************
**************************

//********************************************* Joystick Stuff
********************************************
const int SW_pin = 23;     // digital pin that the joystick is plugged into
const int X_pin = A1;      // analog pin connected to VRx
const int Y_pin = A2;      // analog pin connected to VRy
//********************************************************************************
**************************
```

```
//************************************ keypadCharacterpad Stuff
*********************************
const byte ROWS = 4; // rows
const byte COLS = 3; // columns

char keys[ROWS][COLS] = {
  {'1', '2', '3'},
  {'4', '5', '6'},
  {'7', '8', '9'},
  {'*', '0', '#'}
};

byte rowPins[ROWS] = {33, 34, 35, 36}; //connect to the row pinouts of the
keypadCharacterpad
byte colPins[COLS] = {30, 31, 32};     //connect to the column pinouts of the
keypadCharacterpad
//**************************************************************************
************************

//******************************************* LCD Stuff
*******************************************
const int rs = 13, en = 12, d4 = 11, d5 = 10, d6 = 9, d7 = 8; // declares which pins are plugged
in where
//**************************************************************************
************************

//******************************************* SDA Pins
********************************************
const int SDChipSelect = 22;          // SD card chip select pin
const int potentiometerCS = 29;       // chip select pin for the digital potentiometer (cooling
module control)
const int lockPullupResistorPin = 4; // This sets the pin that will be used as the lock's pullup
resistor
const int lockRelayPin = 28;          // Sets the pin that will be used to communicate with the
lock
//**************************************************************************
************************

//*************************************** Temperature Stuff
***************************************
int temperaturePin = A0;              // analog pin A0
const int temperatureOutputPin = 44;  // sets the pin which will be used to control the pletier
module
//**************************************************************************
************************
```

```

#endif
```

Parse User Information

This routine was unused in the final project. It needs to be modified somewhat in order to be incorporated into the rest of the code, however this .ino file contains all of the correct logic in order to parse user information from the SD card. It would need to be modified slightly to be usable for loading both the username and the password.

```
#include <SPI.h>
#include <SD.h>

File userInfo;

const int SDChipSelect = 7; // SD card chip select pin
char testPassword [4] = {'1', '2', '3', '4'};
bool arrayCheck [5] = {false, false, false, false, false};
bool successfulMatch = false;
char checkCharacter = '%';

void setup() {
  Serial.begin(9600);
  pinMode(SDChipSelect, OUTPUT);
  SD.begin();

  // Pass 2, check every place in the file for matching info
  userInfo = SD.open("userinfo.txt");
  while (userInfo.available()) {

    // Part 1
    if (arrayCheck[0] == false && arrayCheck[1] == false && arrayCheck[2] == false && arrayCheck[3] == false && arrayCheck[4] == false && successfulMatch == false) {
      if (userInfo.read() == checkCharacter) { // checkCharacter is %
        arrayCheck[0] = true;
        Serial.println("Part 0 A");
      }
      else {
        arrayCheck[0] = false;
        Serial.println("Part 0 B");
      }
    }

    else if (arrayCheck[0] == true && arrayCheck[1] == false && arrayCheck[2] == false && arrayCheck[3] == false && arrayCheck[4] == false && successfulMatch == false) {
      if (userInfo.read() == testPassword[0]) {
```

```
      arrayCheck[1] = true;
      Serial.println("Part 1 A");
    }
    else {
      arrayCheck[0] = false;
      Serial.println("Part 1 B");
    }
  }

  // Part 2
  else if (arrayCheck[0] == true && arrayCheck[1] == true && arrayCheck[2] == false &&
arrayCheck[3] == false && arrayCheck[4] == false && successfulMatch == false) {
    if (userInfo.read() == testPassword[1]) {
      arrayCheck[2] = true;
      Serial.println("Part 2 A");
    }
    else {
      arrayCheck[0] = false;
      arrayCheck[1] = false;
      Serial.println("Part 2 B");
    }
  }

  // Part 3
  else if (arrayCheck[0] == true && arrayCheck[1] == true && arrayCheck[2] == true &&
arrayCheck[3] == false && arrayCheck[4] == false && successfulMatch == false) {
    if (testPassword[2] == userInfo.read()) {
      arrayCheck[3] = true;
      Serial.println("Part 3 A");
    }
    else {
      arrayCheck[0] = false;
      arrayCheck[1] = false;
      arrayCheck[2] = false;
      Serial.println("Part 3 B");
    }
  }

  // Part 4
  else if (arrayCheck[0] == true && arrayCheck[1] == true && arrayCheck[2] == true &&
arrayCheck[3] == true && arrayCheck[4] == false && successfulMatch == false) {
    if (testPassword[3] == userInfo.read()) {
      arrayCheck[4] = true;
      Serial.println("Part 4 A");
    }
    else {
      arrayCheck[0] = false;
      arrayCheck[1] = false;
      arrayCheck[2] = false;
      arrayCheck[3] = false;
      Serial.println("Part 4 B");
    }
  }

  // Part 5
  else if (arrayCheck[0] == true && arrayCheck[1] == true && arrayCheck[2] == true &&
arrayCheck[3] == true && arrayCheck[4] == true && successfulMatch == false) {
    successfulMatch = true;
    Serial.println("Part 5");
    break;
  }

  else {
    break;
  }

}
userInfo.close();


Serial.println();
Serial.println();
Serial.print("The result for the successfulMatch variable is: ");
Serial.println(successfulMatch);

if (successfulMatch == true) {
  Serial.println("Success");
}
else {
  Serial.println("Failure");
}

}

void loop() {

}
```

References

[1] Centers for Disease Control and Prevention, "A CDC Framework for Preventing Infectious Diseases," CDC, Atlanta, 2011.

[2] D. Ndwandwe, O. Uthman and A. Adamu, "Decomposing the gap in missed opportunities for vaccination between poor and non-poor in sub-Saharan Africa: A Multicountry Analyses," *Human Vaccines & Immunotherapeutics,* vol. 14, no. 10, p. 2358–2364, 2018.

[3] World Bank, "What would it take to deploy COVID-19 vaccines through sustainable cold chains?," World Bank, 08 May 2020. [Online]. Available: https://blogs.worldbank.org/energy/what-would-it-take-deploy-covid-19-vaccines-through-sustainable-cold-chains. [Accessed 30 May 2020].

[4] Centers for Disease Control and Prevention, "Vaccine Storage and Handling Resources," *Epidemiology and Prevention of Vaccine-Preventable Diseases,* vol. 13, pp. 63-78, 2015.

[5] S. Price, "Bismuth and Telluride," University of Alaska-Fairbanks, 26 March 2007. [Online]. Available: http://ffden-2.phys.uaf.edu/212_spring2007.web.dir/sedona_price/phys_212_webproj_peltier.html. [Accessed 01 July 2020].

[6] Analog Devices, "TMP35/TMP36/TMP37 Datasheet," Analog Devices, Norwood, 2015.

[7] N. Bong, "Track Linear Actuators and Their Further Applications," Progressive Automations, 10 April 2020. [Online]. Available: https://www.progressiveautomations.com/blogs/products/track-linear-actuators-applications. [Accessed 02 June 2020].

[8] D. Phan, "Selecting and Implementing H-Bridges in DC Motor Control," 2011. [Online]. Available: https://www.egr.msu.edu/classes/ece480/capstone/spring11/group03/App%20Note.pdf. [Accessed 02 June 2020].